



République Tunisienne  
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique  
Université Tunis El Manar  
École Nationale d'Ingénieurs de Tunis



## DÉPARTEMENT TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

### Projet de Fin d'Études

Présenté par  
**Ayoub DAMMAK**

Pour l'obtention du  
**Diplôme National d'Ingénieur en Informatique**

---

## Plateforme SaaS de gestion des cliniques avec architecture microservices et multitenant

---

Réalisé au sein de  
**Corilus TN**



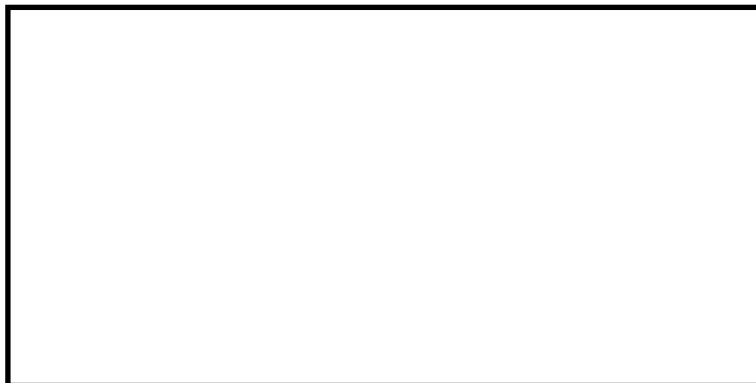
Soutenu le ..09/2025  
Devant le Jury :

Président	: Mr/Mme ...
Rapporteur	: Mr/Mme ...
Encadrant Organisme d'accueil	: M. Alaeddine MOUSSA
Encadrant ENIT	: Mme Imen Boudali

Année universitaire 2025/2026

## Signatures

**Encadrant de l'organisme d'accueil : M. Alaeddine MOUSSA**

A large, empty rectangular box with a black border, intended for the signature of M. Alaeddine MOUSSA.

**Encadrant de l'ENTT : Imene BOUDALI**

A large, empty rectangular box with a black border, intended for the signature of Imene BOUDALI.

## ***Dédicaces***

### ***A mon cher père***

*En toute admiration et gratitude, dont le dévouement et la bienveillance n'ont pas de limites.  
Tes enseignements et tes sacrifices ont tracé le chemin de ma vie.*

### ***A ma chère mère***

*En signe d'affection et d'amour que je vous dédie ce travail, vous qui ravivez dans mon esprit  
un sentiment profond d'une vie pleine de réussite et de chaleureuses bénédictions.*

### ***A mes chers frères***

*Qu'ils trouvent ici l'expression de mes sentiments d'amour et de respect pour leurs  
encouragements et leurs soutiens moraux.*

### ***A mes chers amis***

*Pour les beaux moments qu'on a passés ensemble, pour les merveilleux souvenirs que je garde  
précieusement et pour leurs soutiens tout au long de mon parcours éducatif.*

### ***A tous ceux qui m'ont soutenu***

*Que cette œuvre soit l'expression de ma grande affection et un preuve de mon grand  
attachement et de mon profond amour.*

Ayoub DAMMAK

## **Remerciements**

*C'est avec un grand plaisir que je réserve ces lignes en signe de gratitude et de profonde reconnaissance à tous ceux qui m'ont aidé à réaliser ce travail.*

*Je tiens tout d'abord à remercier et exprimer toute ma reconnaissance envers mon encadrantes académique madame **Imene Boudali** pour sa aide précieuse, ses encouragements et ses conseils judicieux.*

*Ma gratitude va également à monsieur **Alaeddine Moussa**, mon encadrant à Corilus TN pour sa disponibilité, son soutien et pour ses encouragements continus qui m'ont guidé durant les différentes phases de ce projet.*

*Mes vifs remerciements s'adressent également à toute l'équipe de Corilus TN pour l'accueil chaleureux, l'aide, le temps alloué et les efforts fournis à nous guider et à nous intégrer dans l'environnement de travail.*

*Enfin, je saisis cette occasion pour remercier tous les membres du jury qui ont accepté de juger ce rapport de projet de fin de mes études tout en espérant qu'ils y trouvent la qualité, la clarté et motivation qu'ils attendent.*

## ***Résumé***

Ce rapport s'inscrit dans le cadre du projet de fin d'études, élaboré pour l'obtention du Diplôme National d'Ingénieur en Informatique et réalisé au sein de l'entreprise **Corilus TN**. Le projet consiste à concevoir et à développer une application permettant la gestion des cliniques en architecture microservices en utilisant principalement les technologies .NET et ReactJs.

**Mots Clés :** .NET 9, C#, ReactJs, SQL Server, microservices

# Table de Matières

<b>Introduction Générale</b>	<b>1</b>
<b>I Présentation Générale du Projet</b>	<b>2</b>
1 Présentation de l'organisme d'accueil . . . . .	2
1.1 Contexte de l'organisme d'accueil . . . . .	2
1.2 Missions et Activités Principales . . . . .	2
2 Étude Générale du Projet . . . . .	3
2.1 Problématique du Projet . . . . .	3
2.2 Objectifs du Projet . . . . .	3
2.3 Étude et Synthèse des Solutions Similaires . . . . .	3
2.4 Solution Proposée . . . . .	4
3 Méthodologie adoptée et Planification du travail . . . . .	5
3.1 Méthodologie adoptée : SCRUM . . . . .	5
3.2 Les rôles Scrum . . . . .	6
3.3 Planification du Projet selon Scrum . . . . .	7
3.4 Planification des Tâches . . . . .	8
<b>II Analyse et Spécification des Besoins</b>	<b>9</b>
1 Spécification des Besoins . . . . .	9
1.1 Identification des Acteurs . . . . .	9
1.2 Identification des Besoins Fonctionnels . . . . .	9
1.3 Identification des Besoins Non Fonctionnels . . . . .	10
2 Diagramme de Cas d'Utilisation Général . . . . .	10
3 Backlog du Produit . . . . .	11
4 Raffinement des cas d'utilisation . . . . .	14
4.1 Diagramme de cas d'utilisation "Authentification" . . . . .	14
4.2 Diagramme de cas d'utilisation "Gérer les comptes utilisateurs" . . . . .	15
4.3 Diagramme de cas d'utilisation "Gérer les cliniques" . . . . .	16
4.4 Diagramme de cas d'utilisation "Gérer les médecins" . . . . .	17
4.5 Diagramme de cas d'utilisation "Gérer les patients" . . . . .	18
4.6 Diagramme de cas d'utilisation "Gérer les dossiers médicaux" . . . . .	19
4.7 Diagramme de cas d'utilisation "Gérer les rendez-vous" . . . . .	20
4.8 Diagramme de cas d'utilisation "Gérer les consultations" . . . . .	21
4.9 Diagramme de cas d'utilisation "Gérer les factures et paiements" . . . . .	22
4.10 Diagramme de cas d'utilisation "Gérer les notifications" . . . . .	23
4.11 Diagramme de cas d'utilisation "Gérer les rapports et les statistiques" . . . . .	24
<b>III Conception de l'application</b>	<b>26</b>
1 Architecture Globale de l'Application . . . . .	26

1.1	Choix de l'Architecture en Microservices . . . . .	26
1.2	La Conception Pilotée par le Domaine . . . . .	27
1.3	Architecture Globale de l'Application . . . . .	27
1.4	Routage des Requêtes vers les Microservices . . . . .	29
1.5	Communication entre les composants de l'application . . . . .	30
1.6	Gestion des bases de données par microservice . . . . .	30
2	Architecture des Microservices Réalisés . . . . .	31
3	Diagrammes UML . . . . .	32
3.1	Diagramme de classes global . . . . .	32
3.2	Diagrammes de séquences . . . . .	33
<b>IV</b>	<b>Réalisation</b>	<b>37</b>
1	Environnement Matériel . . . . .	37
2	Environnements Logiciels . . . . .	37
3	Technologies Utilisées . . . . .	37
3.1	Choix Technologique pour le Back-end . . . . .	38
3.2	Choix Technologique pour le Front-end . . . . .	38
4	Travail réalisé et scénarios d'exécution . . . . .	38
5	Tests Unitaires . . . . .	38
<b>V</b>	<b>Intégration &amp; DevOps</b>	<b>39</b>
1	Conteneurisation & Orchestration . . . . .	39
2	CI/CD . . . . .	39
3	Observabilité & Monitoring . . . . .	39
4	Sécurité & Gouvernance . . . . .	39
5	Tests intégration & performance . . . . .	39
	<b>Références</b>	<b>42</b>

# Liste des Figures

I.1	Processus de Scrum [6]	6
I.2	Interface de service Azure Boards	7
I.3	Calendrier de planification des tâches	8
II.1	Diagramme de Cas d'Utilisation Général	11
II.2	<i>Diagramme de cas d'utilisation "Authentification"</i>	14
II.3	<i>Diagramme de cas d'utilisation "Gérer les comptes utilisateurs"</i>	15
II.4	<i>Diagramme de cas d'utilisation "Gérer les cliniques"</i>	16
II.5	<i>Diagramme de cas d'utilisation "Gérer les médecins"</i>	17
II.6	<i>Diagramme de cas d'utilisation "Gérer les patients"</i>	18
II.7	<i>Diagramme de cas d'utilisation "Gérer les dossiers médicaux"</i>	19
II.8	<i>Diagramme de cas d'utilisation "Gérer les rendez-vous"</i>	20
II.9	<i>Diagramme de cas d'utilisation "Gérer les consultations"</i>	21
II.10	<i>Diagramme de cas d'utilisation "Gérer les factures et paiements"</i>	22
II.11	<i>Diagramme de cas d'utilisation "Gérer les notifications"</i>	23
II.12	<i>Diagramme de cas d'utilisation "Gérer les rapports et les statistiques"</i>	24
III.1	Architecture globale de l'application	28
III.2	Diagramme de séquence de "API-Gateway"	29
III.3	Architecture interne des microservices	31
III.4	Diagramme de classes global de la plateforme	33
III.5	Diagramme de séquence — Authentification	34
III.6	Diagramme de séquence — Ajouter une clinique	34
III.7	Diagramme de séquence — Prise de rendez-vous	35
III.8	Diagramme de séquence — Paiement en ligne	35
III.9	Diagramme de séquence — Notification utilisateur	36



# Liste des Tableaux

I.1	Comparatif des solutions similaires . . . . .	4
II.2	<i>Raffinement du cas d'utilisation : S'authentifier</i> . . . . .	14
II.3	<i>Raffinement du cas d'utilisation : Consulter le profil d'un utilisateur</i> . . . . .	15
II.4	<i>Raffinement du cas d'utilisation : Créer une clinique</i> . . . . .	16
II.5	<i>Raffinement du cas d'utilisation : Ajouter un médecin</i> . . . . .	17
II.6	<i>Raffinement du cas d'utilisation : Ajouter un patient</i> . . . . .	18
II.7	<i>Raffinement du cas d'utilisation : Consulter le dossier médical du patient</i> . . . . .	19
II.8	<i>Raffinement du cas d'utilisation : Prendre un rendez-vous</i> . . . . .	20
II.9	<i>Raffinement du cas d'utilisation : Ajouter une consultation</i> . . . . .	21
II.10	<i>Raffinement du cas d'utilisation : Payer une facture en ligne</i> . . . . .	22
II.11	<i>Raffinement du cas d'utilisation : Consulter les notifications</i> . . . . .	23
II.12	<i>Raffinement du cas d'utilisation : Gérer les rapports et statistiques</i> . . . . .	24
III.1	Tableau comparatif des architectures . . . . .	26
IV.1	Caractéristiques du Matériel . . . . .	37

## *Introduction Générale*

Dans un monde de plus en plus numérique, le secteur de la santé est en pleine transformation. Les établissements médicaux, tels que les cliniques, sont confrontés à des défis croissants en matière de gestion des données, d'accessibilité des informations et d'efficacité opérationnelle. Pour répondre à ces enjeux, les solutions logicielles jouent un rôle crucial en automatisant les processus, en centralisant les données et en améliorant l'expérience des patients et des professionnels de santé.

Ce projet de fin d'études (PFE) s'inscrit dans ce contexte en proposant une plateforme de gestion des cliniques basée sur une architecture en microservices. Cette solution vise à simplifier la gestion des patients, des rendez-vous, des dossiers médicaux et des factures, tout en garantissant la sécurité et la confidentialité des données.

Ce rapport présente les différentes étapes du projet et se compose de cinq chapitres :

- Le premier chapitre, intitulé « **Présentation Générale du Projet** », présente l'organisme d'accueil, décrit la problématique, expose les objectifs du projet et détaille la méthodologie Scrum adoptée.
- Le deuxième chapitre, intitulé « **Analyse et Spécification des Besoins** », expose les besoins fonctionnels et non fonctionnels, les cas d'utilisation, ainsi que le raffinement des principales fonctionnalités du système.
- Le troisième chapitre, intitulé « **Conception de l'Application** », présente la conception générale et détaillée de la solution, à travers les diagrammes UML et les choix architecturaux retenus.
- Le quatrième chapitre, intitulé « **Réalisation** », décrit l'implémentation des fonctionnalités développées, en mettant en évidence les technologies et les environnements utilisés.
- Le cinquième chapitre, intitulé « **Intégration & DevOps** », traite des aspects liés à la conteneurisation, à l'intégration continue, à la surveillance du système ainsi qu'aux tests d'intégration et de performance.

Finalement, le rapport se termine par une conclusion générale résumant le travail réalisé et ouvrant quelques perspectives.

## *Chapitre I Présentation Générale du Projet*

### Introduction

Ce premier chapitre vise à poser le cadre général de notre projet. Il débute par la présentation de l'organisme d'accueil, **Corilus TN**. Ensuite, une analyse des solutions similaires existantes est menée afin d'identifier les axes d'amélioration à cibler. Enfin, nous présentons la méthodologie de gestion de projet adoptée, fondée sur le cadre agile **Scrum**, ainsi que la planification des différents sprints.

## 1 Présentation de l'organisme d'accueil

### 1.1 Contexte de l'organisme d'accueil

Pour réaliser un projet de qualité, il est essentiel de comprendre l'organisme d'accueil. Cette section présente **Corilus TN**, un acteur majeur dans le domaine des logiciels de santé en Belgique [1].

En tant que principal fournisseur de logiciels pour le marché de la santé belge, Corilus TN a pour mission de façonner un système de santé intelligent et efficace en connectant les prestataires de soins, les patients et les parties prenantes.

Dans un monde en constante évolution, les changements démographiques et médicaux nécessitent de nouveaux modèles de soins. Corilus répond à ces exigences en mettant en œuvre, de manière intuitive et efficace, la nouvelle législation sur les soins de santé dans ses logiciels médicaux. Plus de **40 000 clients**, parmi lesquels des pharmaciens, des dentistes, des médecins généralistes, des physiothérapeutes, des centres de soins résidentiels et des infirmières, utilisent quotidiennement ses solutions, contribuant ainsi à l'amélioration des services de santé.

### 1.2 Missions et Activités Principales

Les services sont au cœur de l'approche de Corilus. L'entreprise fournit des solutions à haute valeur ajoutée dans le domaine de l'eHealth en Belgique.

Parmi les services offerts, Corilus se distingue par des solutions innovantes telles que [1] :

- **Care Connect** : Une plateforme facilitant l'interaction entre les patients et les professionnels de santé. Elle est divisée en plus de dix catégories selon la spécialité du prestataire de soins, telles que CareConnect Médecin généraliste, CareConnect Pharmacien, CareConnect Dentiste, etc. Cette solution permet la gestion des données administratives, le suivi des statuts et la prise de rendez-vous.
- **Health Connect** : Une plateforme d'échange sécurisé de courriels via des services tels que eHealthBox, Medimail et Mexi, garantissant la confidentialité des informations médicales.
- **Helena** : Un environnement sécurisé pour la gestion et le partage des documents médicaux. Cette solution permet aux patients et aux prestataires de soins de communiquer en toute sécurité, garantissant ainsi un suivi efficace et confidentiel.

## 2 Étude Générale du Projet

Notre projet consiste en une solution informatique dédiée à la gestion efficace des cliniques. Afin de concevoir un livrable répondant pleinement aux besoins identifiés, il est essentiel de comprendre en profondeur le domaine d'étude. Cela passe notamment par l'analyse de la problématique actuelle et la présentation de la solution proposée.

### 2.1 Problématique du Projet

Dans le secteur de la santé, la gestion efficace des informations médicales est essentielle pour assurer la qualité des soins, la sécurité des patients et l'efficacité administrative. Cependant, de nombreuses cliniques rencontrent des difficultés liées à l'organisation et à la centralisation des données.

Les principales problématiques identifiées sont :

- **Fragmentation des données** : Les informations patients (dossiers médicaux, rendez-vous, facturation) sont souvent dispersées entre différents systèmes non interconnectés.
- **Manque d'accessibilité** : Les professionnels de santé n'ont pas toujours un accès rapide et sécurisé aux informations critiques.
- **Complexité de la gestion** : Les processus administratifs sont souvent manuels et sources d'erreurs, entraînant des retards et des incohérences.
- **Interopérabilité limitée** : Il est difficile d'intégrer les systèmes de gestion existants avec des solutions tierces.

Ces difficultés entraînent une perte de temps, une augmentation des coûts et un risque accru d'erreurs médicales.

### 2.2 Objectifs du Projet

L'objectif principal de ce projet est de concevoir et de développer un système de gestion des cliniques basé sur une architecture en microservices, afin d'optimiser la gestion des informations médicales et d'améliorer l'efficacité opérationnelle.

Les objectifs spécifiques sont :

- Automatiser la gestion des patients, des médecins et des rendez-vous.
- Fournir une interface utilisateur intuitive pour le personnel médical et administratif.
- Garantir la confidentialité et la sécurité des informations sensibles.
- Mettre en place un système de notifications automatisé pour les patients et les professionnels de santé.
- Permettre l'interopérabilité avec d'autres systèmes de gestion médicale.
- Assurer la scalabilité pour l'ajout futur de nouvelles fonctionnalités.

### 2.3 Étude et Synthèse des Solutions Similaires

Afin de mieux comprendre les besoins et d'apporter des innovations pertinentes, nous avons étudié plusieurs solutions existantes sur le marché :

TABLE I.1 – Comparatif des solutions similaires

Solution	Caractéristiques	Avantages	Inconvénients
<b>Doctolib</b> [2]	Plateforme web et mobile. Gestion des rendez-vous (présentiel / téléconsultation), rappels. Messagerie sécurisée, partage de documents. Fonctionnalités de gestion de cabinet (agenda, facturation, télétransmission).	Interface intuitive et populaire en Europe. Solution tout-en-un pour les cabinets. Bonne réputation.	Coût mensuel non communiqué. Risque de dépendance à un tiers. Enjeux de confidentialité (RGPD).
<b>Cliniko</b> [3]	Plateforme cloud de gestion de cabinet. Fonctionnalités : agenda, dossiers patients, facturation, paiements, rapports. Tarification : 45 \$/mois/praticien + 10 ¢/SMS. Conforme HIPAA et RGPD.	Interface moderne et ergonomique. Tarification claire. Support client efficace.	SMS payants. Pas d'application mobile native. Moins adaptée aux grandes structures.
<b>Zocdoc</b> [4]	Réservation de rendez-vous en ligne (présentiel ou téléconsultation). Recherche de médecins par spécialité, assurance et localisation. Avis patients, intégration avec Google et sites web.	Gratuit pour les patients. Outil de visibilité pour les praticiens. Expérience utilisateur simplifiée.	Centrée sur la prise de rendez-vous (pas de gestion de cabinet). Coût variable selon le référencement. Moins adaptée en dehors des États-Unis.

**Limites des solutions existantes :**

- Absence de personnalisation avancée dans les solutions propriétaires.
- Intégration difficile avec des systèmes tiers.
- Coûts élevés des solutions commerciales.
- Certaines solutions sont moins adaptées à la gestion complète sur mobile, notamment pour le personnel administratif.

**2.4 Solution Proposée**

Afin de répondre aux besoins spécifiques des professionnels de santé tout en surmontant les limites des solutions existantes, nous proposons une plateforme SaaS moderne, évolutive et sécurisée, conçue pour le personnel médical de Corilus.

Cette solution repose sur une architecture à microservices distribuée, offrant une forte modularité, une personnalisation avancée selon les rôles des utilisateurs (administrateurs, médecins, patients, etc.) et une intégration facilitée avec d'autres systèmes externes via des API.

La solution consiste à mettre en place une plateforme SaaS permettant :

- **La gestion centralisée de la clinique** par l'administrateur (médecins, patients, disponibilités, etc.).
- **La planification des rendez-vous en ligne** par les patients, avec notifications et rappels automatiques.
- **Un accès sécurisé et confidentiel** pour chaque utilisateur (authentification JWT, gestion des rôles, contrôle d'accès).
- **Une facturation intelligente et des paiements en ligne**, avec suivi en temps réel des factures et des transactions.
- **La génération de rapports dynamiques**, de statistiques de performance et d'analyses avancées pour la prise de décision.
- **Une compatibilité multi-plateforme (desktop / mobile)** assurant une accessibilité fluide, y compris pour les cliniciens nomades.

En résumé, cette solution se distingue par sa souplesse, sa scalabilité, sa sécurité et son adaptabilité. Elle a été conçue pour offrir aux cliniques un outil de gestion intégré, tout en garantissant une expérience utilisateur fluide, moderne et conforme aux exigences métiers.

### 3 Méthodologie adoptée et Planification du travail

Dans cette partie nous allons présenter tout d'abord la méthodologie adoptée durant notre projet.

#### 3.1 Méthodologie adoptée : SCRUM

Les processus et techniques de gestion de projet jouent un rôle crucial dans l'exécution, l'atteinte des objectifs, et permettent d'organiser le projet de manière structurée et rationalisée.

Pour ce faire, nous avons adopté la méthode **SCRUM**, qui fait partie des méthodes agiles, comme processus de développement de notre projet.

Scrum vise à subdiviser l'organisation en petites équipes auto-organisées et pluridisciplinaires, afin de réduire les difficultés telles que le manque de planification, l'évolution constante des besoins ou encore le manque d'implication des clients.

Les principes de base de Scrum sont les suivants [5] :

- Dégager dans un premier temps le maximum de fonctionnalités à réaliser pour constituer le backlog du produit.
- Définir les priorités des fonctionnalités et choisir celles qui seront réalisées dans chaque itération.
- Par la suite, focaliser l'équipe de manière itérative sur l'ensemble des fonctionnalités à développer, dans des itérations appelées **Sprints**.

Ces sprints peuvent durer de deux à quatre semaines. Une réunion entre le client et l'équipe projet est programmée à la fin de chaque sprint pour présenter l'état d'avancement, recueillir les commentaires et planifier les prochaines étapes [5].

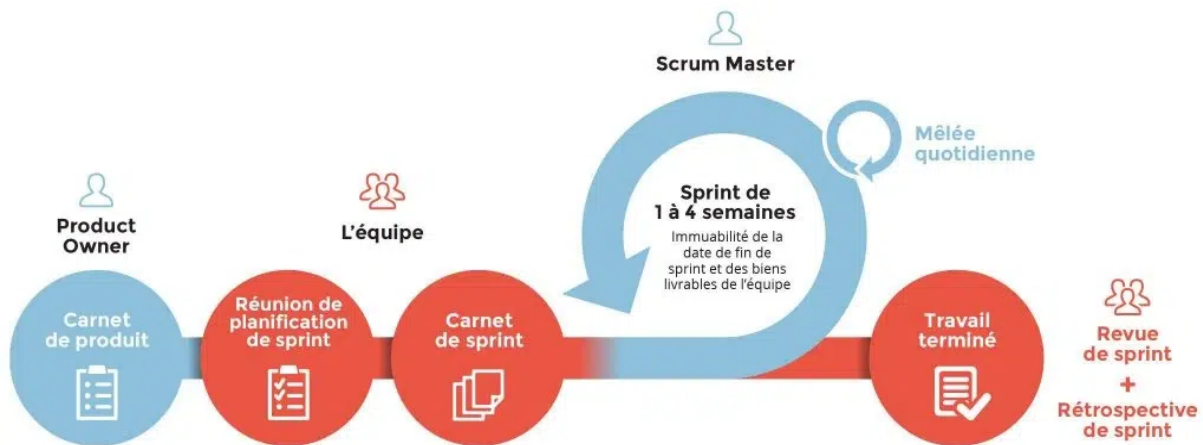


FIGURE I.1 – Processus de Scrum [6]

La figure I.1 illustre à la fois les différents rôles dans Scrum ainsi que le déroulement du processus.

Le choix de Scrum comme méthodologie de pilotage pour notre projet repose sur les atouts suivants [5] :

- Une souplesse et une flexibilité importantes, bénéfiques pour le chef de projet ainsi que pour toute l'équipe de développement.
- La tenue de réunions quotidiennes permettant un suivi continu du développement et une intervention rapide en cas de besoin.
- La promotion d'une collaboration constante entre les membres de l'équipe et le partage de connaissances.
- L'adoption d'une approche itérative et incrémentale, optimisant la prédictibilité et le contrôle des risques.
- Une forte capacité d'adaptation aux changements, rendue possible par des itérations courtes, permettant une progression rapide dans la réalisation du projet.

### 3.2 Les rôles Scrum

Les acteurs d'un projet Scrum sont les suivants [5] :

- **Le « Product Owner (PO) »** : responsable du produit au sein de l'équipe projet côté client. Son rôle consiste à :
  - Porter la vision du produit à réaliser ; il s'agit généralement d'un expert métier.
  - Travailler en collaboration directe avec l'équipe de développement. Il est chargé d'alimenter le « backlog produit » et de prioriser les *user stories* à réaliser.
  - Être interne ou externe à l'organisation, bien qu'il s'agisse souvent du client.
- **Le « Scrum Master (SM) »** : chargé de superviser les membres de l'équipe en les aidant, les encourageant et en facilitant la communication et le partage des informations essentielles.
- **L'équipe de développement** :
  - Transforme les besoins exprimés par le Product Owner, sous forme de *user stories*, en fonctionnalités concrètes, opérationnelles et utilisables.
  - Livre régulièrement une version fonctionnelle du produit.

### 3.3 Planification du Projet selon Scrum

Afin d'appliquer correctement la méthode **Scrum**, nous devons suivre le cycle de vie d'un sprint tel qu'illustré dans la figure I.1.

- Le *Product Owner* crée le *Product Backlog* en définissant et en priorisant les *user stories*.
- Lors de la planification du sprint, l'équipe sélectionne les *user stories* les plus prioritaires du *Product Backlog* pour constituer le *Sprint Backlog*.
- Ensuite, l'équipe de développement implémente les *user stories* choisies sur une période de deux à quatre semaines.
- Des réunions courtes quotidiennes, appelées *Daily Scrum*, sont tenues pour synchroniser les membres de l'équipe.
- Le travail doit être achevé à la fin du sprint et une démonstration des fonctionnalités développées est réalisée pour le client.
- Le sprint se clôt par une *Sprint Review*, qui permet d'évaluer les fonctionnalités livrées, suivie d'une *Sprint Retrospective* visant à identifier les axes d'amélioration pour les prochaines itérations.

Pour gérer efficacement ce projet et faciliter le suivi des tâches, nous avons utilisé **Microsoft Azure DevOps**, un outil de gestion de projet qui permet aux équipes de développement de planifier, suivre et livrer des logiciels de manière plus efficace. Il offre un ensemble de services intégrés pour la gestion de projets agiles, notamment [7] :

- **Azure Repos** : Fournit un système de contrôle de version distribué (Git) et centralisé (TFVC), permettant de gérer le code source de manière collaborative.
- **Azure Boards** : Permet de planifier, suivre et discuter des tâches. Il offre une vue d'ensemble de l'état d'avancement du projet. La figure I.2 illustre l'interface de Azure Boards.

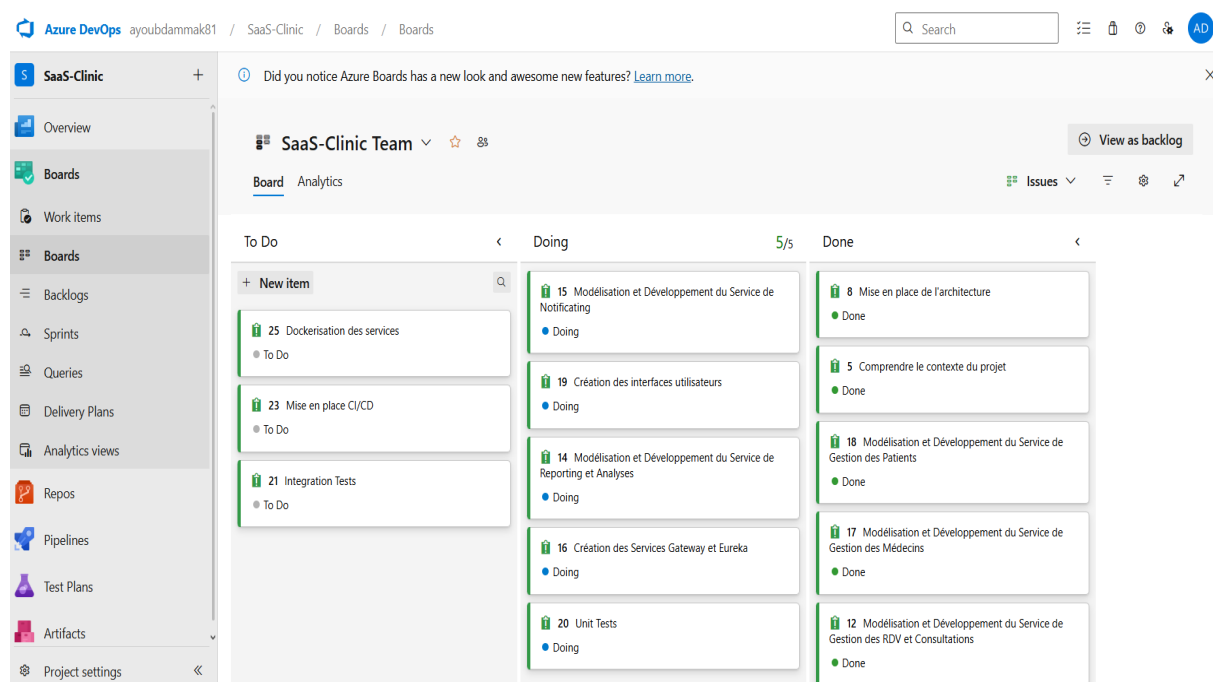


FIGURE I.2 – Interface de service Azure Boards



### 3.4 Planification des Tâches

La planification constitue l'une des phases les plus essentielles en amont du projet. Elle permet d'organiser les tâches à accomplir et d'estimer la charge de travail associée à chacune d'elles.

Dans le cadre de notre démarche agile, le travail a été structuré en **six sprints**, chacun correspondant à une phase clé du projet. Cette structuration en sprints permet une exécution incrémentale et itérative, favorisant les retours d'expérience rapides ainsi que l'amélioration continue.

La figure I.3 présente les différents sprints accompagnés de leurs intitulés respectifs :

- **Sprint 0** : Analyse et spécification des besoins
- **Sprint 1** : Conception technique
- **Sprint 2** : Développement du back-end
- **Sprint 3** : Développement du front-end
- **Sprint 4** : Intégration et tests globaux
- **Sprint 5** : Déploiement et phase DevOps

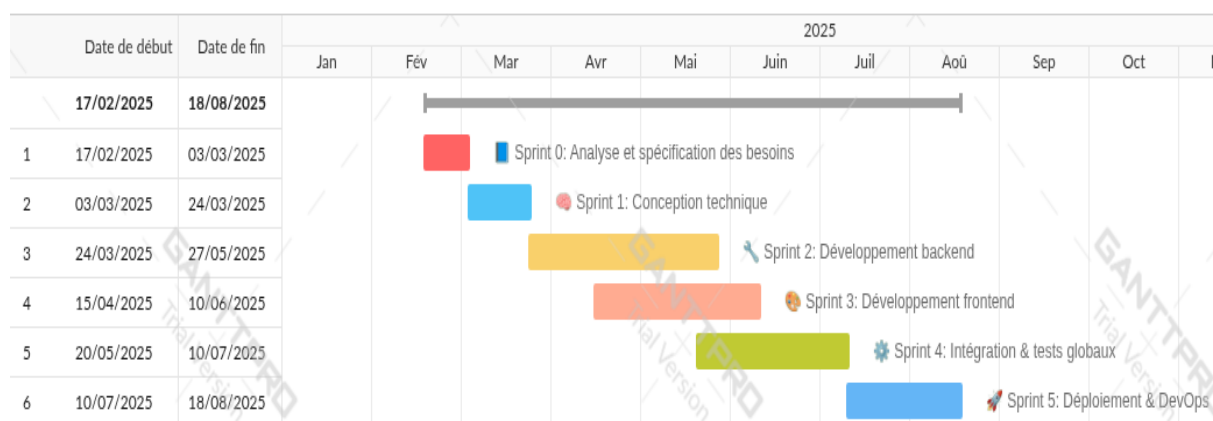


FIGURE I.3 – Calendrier de planification des tâches

## Conclusion

Ce premier chapitre a permis de poser les fondations du projet. Nous avons tout d'abord présenté l'organisme d'accueil, **Corilus TN**. Ensuite, nous avons analysé les solutions existantes afin d'en identifier les limites, ce qui nous a conduits à proposer une plateforme innovante, adaptée aux besoins des cliniques. Enfin, nous avons décrit la méthodologie agile **Scrum**, choisie pour piloter le développement du projet, ainsi que la planification des tâches à travers des sprints structurés et gérés via Azure DevOps.

Le chapitre suivant sera consacré à l'analyse détaillée des besoins fonctionnels et techniques de la solution, première étape vers la conception de l'application.

## *Chapitre II Analyse et Spécification des Besoins*

### Introduction

Ce chapitre est consacré à la définition des besoins fonctionnels et non fonctionnels du projet, ainsi qu'à la présentation des acteurs impliqués dans la plateforme. Il aborde également le raffinement des diagrammes de cas d'utilisation par fonctionnalité. Ce chapitre constitue une base essentielle pour la conception et la mise en œuvre du projet.

## 1 Spécification des Besoins

La spécification des besoins constitue la phase de départ de tout projet de développement de plateforme. Dans cette section, nous allons identifier et formaliser les besoins liés à notre solution, en mettant l'accent sur les exigences fonctionnelles et non fonctionnelles, dans le but de concevoir un système performant, sécurisé et répondant aux attentes des utilisateurs.

### 1.1 Identification des Acteurs

Avant d'énumérer les acteurs de notre plateforme, il convient de définir ce qu'est un acteur. Un acteur est une entité externe au système qui interagit avec ce dernier. Il peut s'agir d'une personne ou d'un autre système informatique. Nous avons identifié quatre acteurs principaux qui interagiront avec la plateforme :

- **Super Administrateur** : Gère la plateforme dans sa globalité et attribue les accès aux administrateurs des cliniques.
- **Administrateur de la clinique** : Gère les utilisateurs internes, les factures, les ressources et les rapports.
- **Médecin** : Consulte et met à jour les dossiers médicaux des patients.
- **Patient** : Prend des rendez-vous, consulte son dossier médical et gère ses factures.

### 1.2 Identification des Besoins Fonctionnels

Les besoins fonctionnels décrivent les services que le système doit offrir à ses utilisateurs. Ils traduisent les fonctionnalités principales attendues de la plateforme, à savoir :

- **Gérer les cliniques** : Créer, modifier et supprimer des cliniques, ainsi que filtrer celles-ci par nom ou adresse.
- **Gérer les comptes utilisateurs** : Permettre la création, la modification, la suppression et l'authentification des utilisateurs (patients, médecins, administrateurs, super administrateur). Gérer les rôles et les permissions associés.
- **Gérer les patients** : Enregistrer de nouveaux patients, mettre à jour leurs informations personnelles, consulter leur historique médical et gérer leurs dossiers médicaux.

- **Gérer les médecins** : Gérer les profils des médecins, leurs spécialités, disponibilités et horaires de consultation.
- **Gérer les rendez-vous** : Permettre aux patients de prendre des rendez-vous en ligne, ainsi que de modifier, annuler ou confirmer un rendez-vous.
- **Gérer les consultations** : Enregistrer les consultations effectuées par les médecins, y ajouter des observations et des prescriptions médicales.
- **Gérer les factures et paiements** : Générer les factures des consultations et traitements, gérer les paiements en ligne, et suivre l'historique des transactions.
- **Gérer les rapports et analyses** : Générer des rapports statistiques sur les consultations, les revenus, l'utilisation des ressources et la satisfaction des patients.
- **Gérer les notifications** : Envoyer des notifications aux utilisateurs (par email ou SMS) pour les rappels de rendez-vous, alertes système, changements de statut, etc.

### 1.3 Identification des Besoins Non Fonctionnels

Les besoins non fonctionnels définissent les caractéristiques générales de la plateforme, en termes de qualité de service, de performance, de sécurité, etc. Ils sont essentiels pour garantir la fiabilité et la robustesse du système.

- **Sécurité** : La protection des données personnelles des utilisateurs ainsi que des bases de données est primordiale. Toutes les communications doivent être sécurisées via des protocoles de chiffrement (ex : HTTPS, TLS). L'authentification et l'autorisation seront assurées par un système de gestion des accès basé sur les rôles (RBAC).
- **Fiabilité et disponibilité** : La plateforme doit être disponible 24h/24 et 7j/7, avec un taux de disponibilité supérieur à 99,9 %. Des mécanismes de tolérance aux pannes doivent être mis en place (réplication des bases de données, sauvegardes régulières, etc.).
- **Performance** : Le système doit offrir un temps de réponse inférieur à 2 secondes pour la majorité des opérations. L'optimisation des requêtes, la mise en cache et la gestion efficace des ressources doivent être assurées.
- **Scalabilité** : La plateforme doit pouvoir gérer l'augmentation du nombre d'utilisateurs et de données sans dégradation des performances. Elle devra supporter l'ajout de nouveaux serveurs et permettre la répartition de la charge (*load balancing*).
- **Isolation des données** : Chaque clinique doit disposer d'un accès exclusif à ses propres données, sans interférence avec celles des autres. Une architecture multi-tenant devra être mise en œuvre afin de garantir une séparation stricte des informations et des permissions.

## 2 Diagramme de Cas d'Utilisation Général

Nous présentons à travers la figure II.2 une vue générale du comportement fonctionnel de notre plateforme. Cette figure II.2 montre les fonctionnalités générales de notre plateforme comme la gestion des patients, des comptes et des factures.

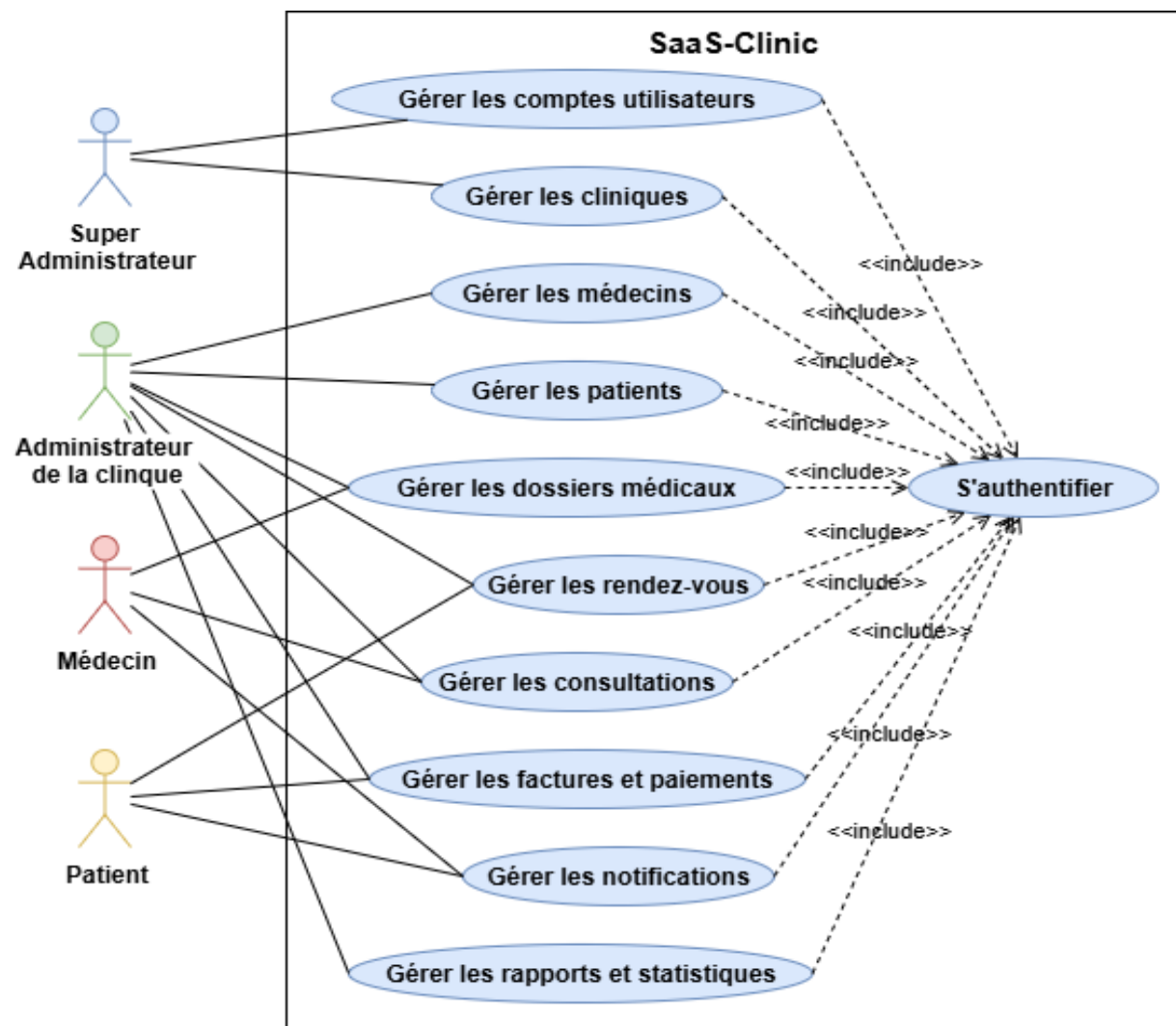


FIGURE II.1 – Diagramme de Cas d'Utilisation Général

### 3 Backlog du Produit

La réalisation du Backlog du produit représente une étape importante dans le cadre Scrum. Le Backlog du produit correspond à une liste hiérarchisée des besoins et des exigences des clients, souvent appelés user stories. Le Backlog Produit est détaillé dans le tableau 2.1 qui comprend les champs les plus importants suivants :

- **ID-US :** C'est l'identifiant de chaque User Story.
- **User Story :** Une description de la forme "En tant que [rôle], je veux [fonctionnalité] pour [objectif]"
- **Priorité :** C'est la valeur métier qui détermine la priorisation du développement des User Stories en fonction des attentes et les besoins du client.

ID	Fonctionnalité	ID-US	User Story	Priorité
F01	Gestion des utilisateurs	US01	En tant qu'administrateur, je veux pouvoir créer, modifier et supprimer des comptes utilisateurs pour gérer l'accès à la plateforme.	Haute
		US02	En tant qu'utilisateur, je veux pouvoir m'authentifier pour accéder à mon espace personnel.	Haute
		US03	En tant qu'administrateur, je veux pouvoir assigner des rôles aux utilisateurs pour contrôler leurs permissions.	Moyenne
F02	Gestion des cliniques	US04	En tant que super administrateur, je veux pouvoir ajouter une nouvelle clinique pour qu'elle puisse utiliser la plateforme.	Haute
		US05	En tant qu'administrateur de clinique, je veux modifier les informations de ma clinique pour les maintenir à jour.	Moyenne
		US06	En tant que patient, je veux voir la liste des cliniques pour choisir où prendre un rendez-vous.	Basse
F03	Gestion des médecins	US07	En tant qu'administrateur de clinique, je veux ajouter, modifier et supprimer des médecins pour gérer le personnel médical.	Haute
		US08	En tant que médecin, je veux pouvoir voir mon emploi du temps pour organiser mes consultations.	Moyenne
F04	Gestion des patients	US09	En tant qu'administrateur de clinique, je veux enregistrer les informations des patients pour assurer un suivi médical.	Haute
		US10	En tant que patient, je veux pouvoir voir mon historique médical pour suivre mes traitements.	Moyenne
F05	Gestion des rendez-vous	US11	En tant que patient, je veux prendre un rendez-vous avec un médecin pour une consultation.	Haute
Suite à la page suivante...				

*Suite de la table précédente*

ID	Fonctionnalité	ID-US	User Story	Priorité
		US12	En tant que médecin, je veux voir mes rendez-vous pour organiser ma journée.	Haute
		US13	En tant qu'administrateur, je veux gérer les disponibilités des médecins pour assurer une bonne planification.	Moyenne
F06	Gestion des consultations	US14	En tant que médecin, je veux créer une consultation et ajouter des observations médicales pour suivre mes patients.	Haute
		US15	En tant que patient, je veux voir mes consultations passées pour consulter les diagnostics.	Moyenne
F07	Gestion des dossiers médicaux	US16	En tant que médecin, je veux créer un dossier médical pour enregistrer l'historique médical du patient.	Haute
		US17	En tant qu'administrateur, je veux associer un dossier médical à un patient pour centraliser ses données.	Moyenne
F08	Gestion des factures	US18	En tant que patient, je veux voir mes factures et leur statut pour gérer mes paiements.	Haute
		US19	En tant qu'administrateur, je veux générer des factures pour chaque consultation effectuée.	Moyenne
		US20	En tant que patient, je veux payer ma facture en ligne pour éviter les déplacements.	Haute
F09	Notifications et rappels	US23	En tant que patient, je veux recevoir des rappels pour mes prochains rendez-vous afin de ne pas les oublier.	Haute
		US24	En tant que médecin, je veux être notifié en cas d'annulation ou de modification d'un rendez-vous.	Moyenne
F10	Génération de rapports et statistiques	US25	En tant qu'administrateur, je veux générer des rapports d'activité pour suivre les performances de la clinique.	Basse

*Suite à la page suivante...*

Suite de la table précédente

ID	Fonctionnalité	ID-US	User Story	Priorité
		US26	En tant que médecin, je veux générer un rapport médical après chaque consultation.	Moyenne

## 4 Raffinement des cas d'utilisation

Dans cette partie, nous présentons les diagrammes de cas d'utilisation raffinés par fonctionnalité.

### 4.1 Diagramme de cas d'utilisation "Authentification"

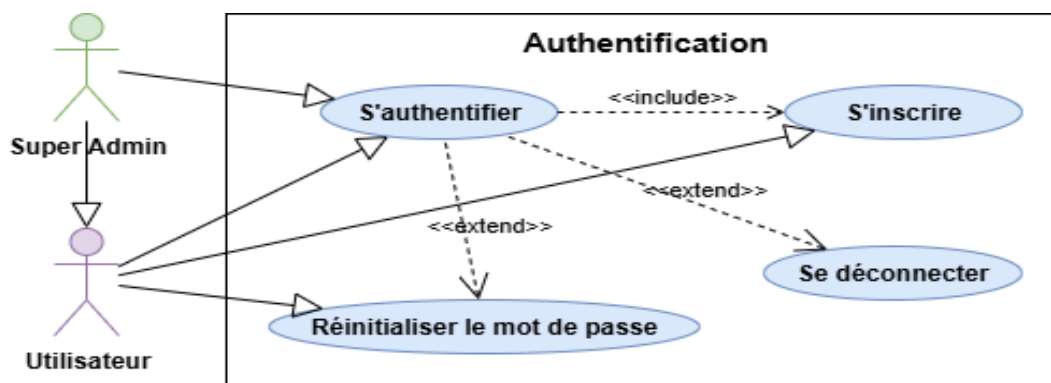


FIGURE II.2 – Diagramme de cas d'utilisation "Authentification"

TABLE II.2 – Raffinement du cas d'utilisation : S'authentifier

<b>Nom du cas</b>	S'authentifier
<b>Acteurs</b>	Utilisateur, SuperAdmin
<b>Objectif</b>	Permettre à l'utilisateur de se connecter au système de manière sécurisée.
<b>Préconditions</b>	L'utilisateur possède déjà un compte valide.
<b>Postconditions</b>	L'utilisateur est connecté à son espace personnel.
<b>Scénario principal</b>	<ol style="list-style-type: none"> <li>1. L'utilisateur accède à la page de connexion.</li> <li>2. Il saisit son email et son mot de passe.</li> <li>3. Il clique sur le bouton « Se connecter ».</li> <li>4. Le système vérifie les informations.</li> <li>5. Si elles sont valides, l'utilisateur est redirigé vers son tableau de bord.</li> </ol>
<b>Scénarios alternatifs</b>	<p>A1. Les identifiants sont incorrects : un message d'erreur s'affiche.</p> <p>A2. L'utilisateur a oublié son mot de passe : il clique sur « Mot de passe oublié » et suit le processus de réinitialisation.</p>

## 4.2 Diagramme de cas d'utilisation "Gérer les comptes utilisateurs"

Le diagramme de cas d'utilisation "Gérer les comptes utilisateurs" est illustré dans la figure II.3.

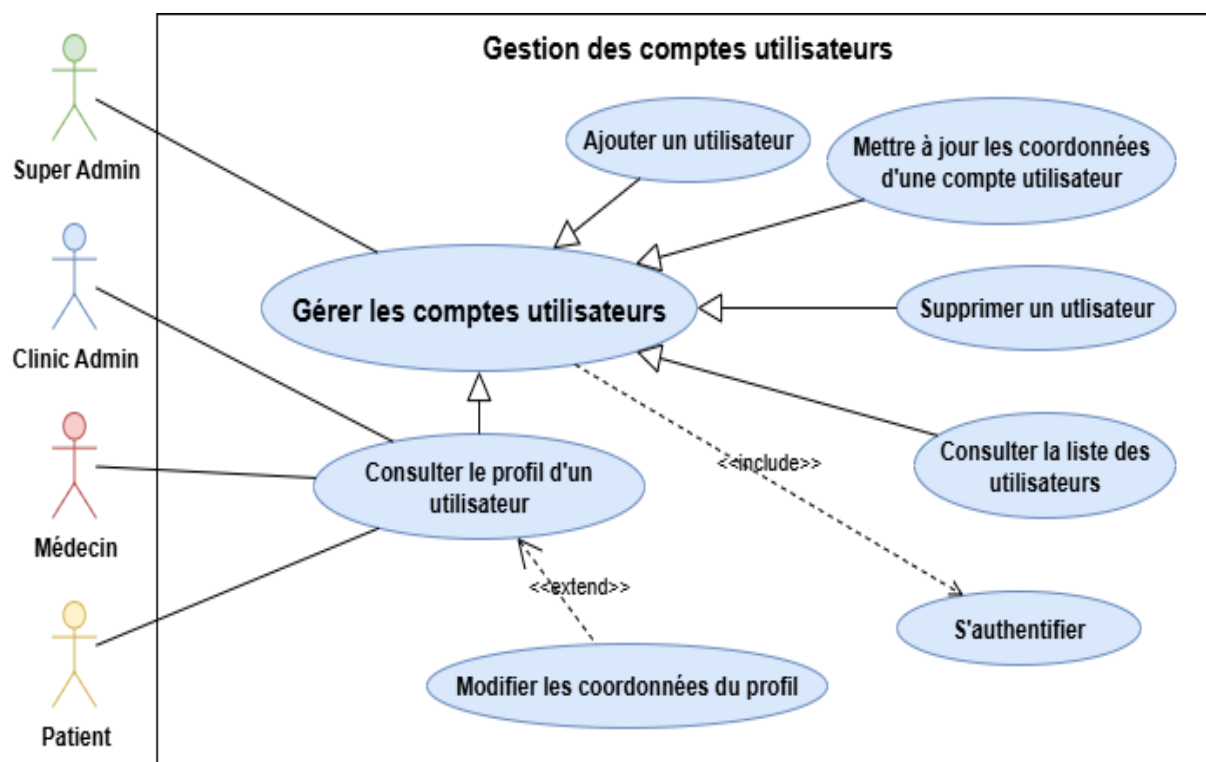


FIGURE II.3 – Diagramme de cas d'utilisation "Gérer les comptes utilisateurs"

TABLE II.3 – Raffinement du cas d'utilisation : Consulter le profil d'un utilisateur

<b>Nom du cas</b>	Consulter le profil d'un utilisateur
<b>Acteurs</b>	Super Admin, Clinic Admin, Médecin, Patient
<b>Objectif</b>	Permettre à l'utilisateur de visualiser son propre profil ou celui d'un autre utilisateur (selon ses droits).
<b>Préconditions</b>	L'utilisateur est authentifié.
<b>Postconditions</b>	Les informations du profil sont affichées à l'écran.
<b>Scénario principal</b>	<ol style="list-style-type: none"> <li>1. L'utilisateur clique sur « Mon profil » ou recherche un autre utilisateur.</li> <li>2. Le système récupère les informations du profil demandé.</li> <li>3. Le système affiche les données du profil (nom, email, rôle, coordonnées, etc.).</li> </ol>
<b>Scénarios alternatifs</b>	<p>A1. L'utilisateur n'a pas les droits pour consulter certains profils : un message d'accès refusé s'affiche.</p> <p>A2. L'utilisateur n'est pas connecté : il est redirigé vers la page de connexion (voir le cas d'utilisation "S'authentifier").</p>



### 4.3 Diagramme de cas d'utilisation "Gérer les cliniques"

Le diagramme de cas d'utilisation "Gérer les cliniques" est illustré dans la figure II.4.

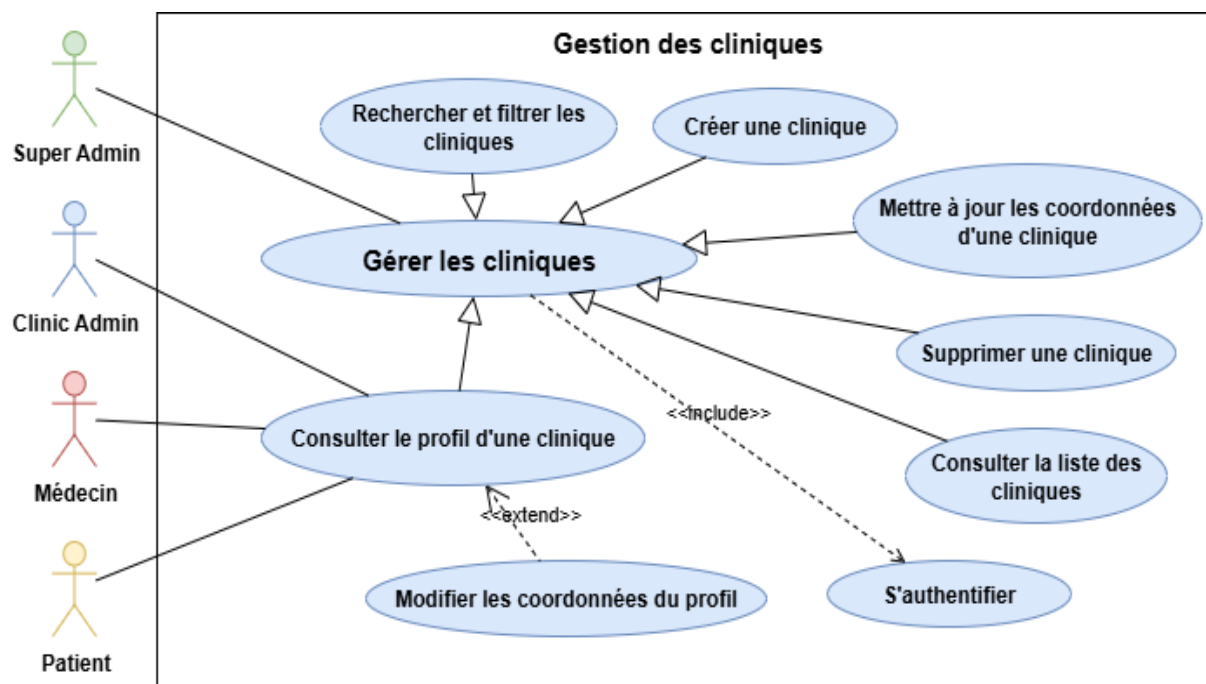


FIGURE II.4 – Diagramme de cas d'utilisation "Gérer les cliniques"

TABLE II.4 – Raffinement du cas d'utilisation : Créer une clinique

<b>Nom du cas</b>	Créer une clinique
<b>Acteurs</b>	Super Admin, Clinic Admin
<b>Objectif</b>	Permettre à un administrateur de créer une nouvelle fiche de clinique dans le système.
<b>Préconditions</b>	L'utilisateur est authentifié et autorisé à gérer les cliniques.
<b>Postconditions</b>	Une nouvelle clinique est ajoutée à la base de données et visible dans la liste des cliniques.
<b>Scénario principal</b>	<ol style="list-style-type: none"> <li>1. L'utilisateur accède à l'interface de création de clinique.</li> <li>2. Il saisit les informations requises : nom, adresse, téléphone, email, etc.</li> <li>3. Il clique sur le bouton de validation.</li> <li>4. Le système vérifie les données saisies.</li> <li>5. La nouvelle clinique est enregistrée et un message de confirmation s'affiche.</li> </ol>
<b>Scénarios alternatifs</b>	<p>Des champs obligatoires sont manquants : le système affiche un message d'erreur.</p> <p>A2. La clinique existe déjà (doublon détecté) : un message d'alerte est affiché pour éviter la duplication.</p>

#### 4.4 Diagramme de cas d'utilisation "Gérer les médecins"

Le diagramme de cas d'utilisation "Gérer les médecins" est illustré dans la figure II.5.

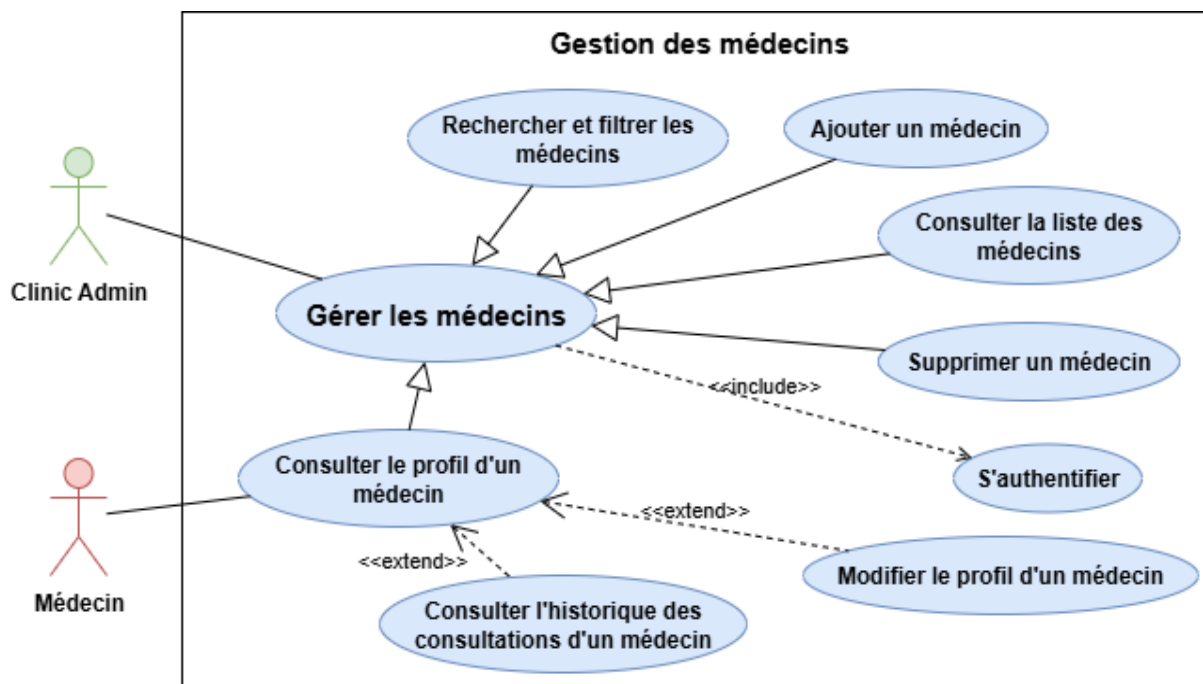


FIGURE II.5 – Diagramme de cas d'utilisation "Gérer les médecins"

TABLE II.5 – Raffinement du cas d'utilisation : Ajouter un médecin

<b>Nom du cas</b>	Ajouter un médecin
<b>Acteurs</b>	ClinicAdmin
<b>Objectif</b>	Permettre à l'administrateur de créer un nouveau compte médecin en renseignant ses informations personnelles et professionnelles.
<b>Préconditions</b>	L'administrateur est authentifié et autorisé à gérer les médecins.
<b>Postconditions</b>	Un nouveau médecin est ajouté à la base de données, et il peut accéder au système via ses identifiants.
<b>Scénario principal</b>	<ol style="list-style-type: none"> <li>1. Le ClinicAdmin accède à l'interface de gestion des médecins.</li> <li>2. Il clique sur « Ajouter un médecin ».</li> <li>3. Il saisit les informations requises : nom, prénom, spécialité, email, mot de passe, etc.</li> <li>4. Il valide la création.</li> <li>5. Le système enregistre les données et affiche un message de confirmation.</li> </ol>
<b>Scénarios alternatifs</b>	<ol style="list-style-type: none"> <li>A1. Des champs obligatoires sont manquants ou invalides : le système affiche un message d'erreur.</li> <li>A2. Le médecin existe déjà (même email) : le système refuse la création et affiche un message d'alerte.</li> <li>A3. Une erreur serveur survient : un message d'erreur s'affiche et aucune donnée n'est enregistrée.</li> </ol>

#### 4.5 Diagramme de cas d'utilisation "Gérer les patients"

Le diagramme de cas d'utilisation "Gérer les patients" est illustré dans la figure II.6.

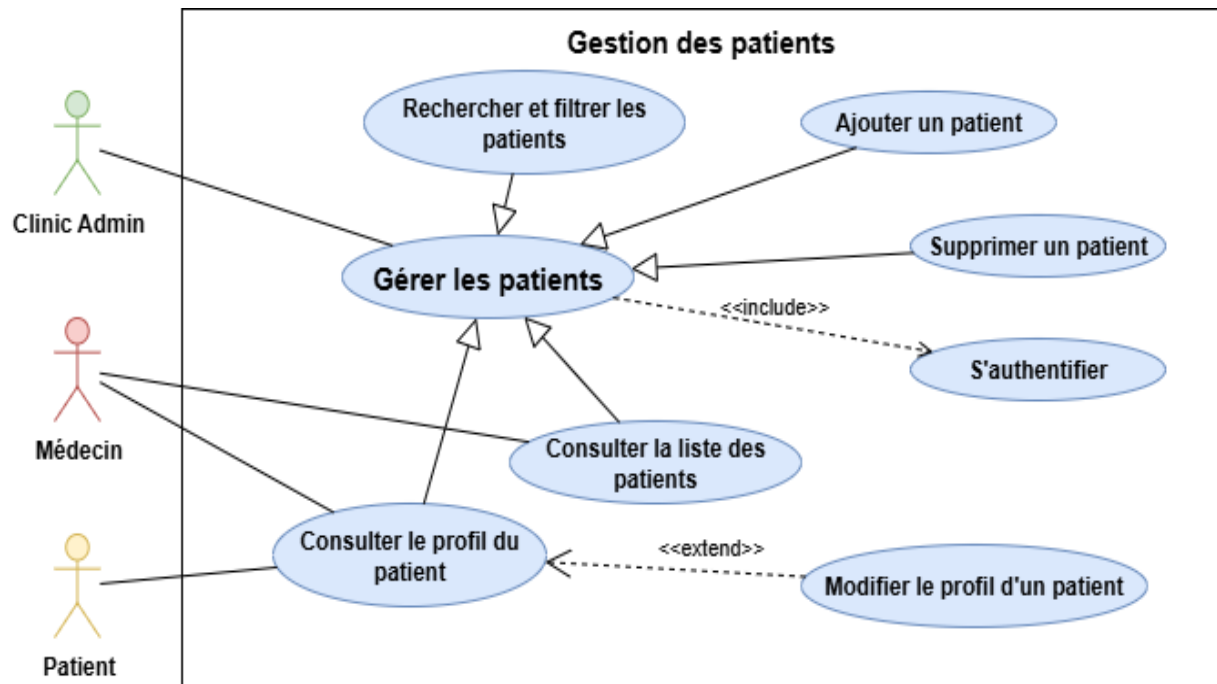


FIGURE II.6 – Diagramme de cas d'utilisation "Gérer les patients"

TABLE II.6 – Raffinement du cas d'utilisation : Ajouter un patient

<b>Nom du cas</b>	Ajouter un patient
<b>Acteurs</b>	ClinicAdmin
<b>Objectif</b>	Permettre à l'administrateur de clinique d'enregistrer un nouveau patient dans le système avec ses informations personnelles et administratives.
<b>Préconditions</b>	L'acteur est authentifié et dispose des autorisations nécessaires pour créer un nouveau patient.
<b>Postconditions</b>	Le nouveau patient est ajouté à la base de données et devient visible dans la liste des patients.
<b>Scénario principal</b>	<ol style="list-style-type: none"> <li>1. Le Clinic Admin accède au module de gestion des patients.</li> <li>2. Il sélectionne l'option « Ajouter un patient ».</li> <li>3. Le système affiche un formulaire de création.</li> <li>4. Le Clinic Admin saisit les informations du patient (nom, prénom, CIN, date de naissance, sexe, contact, etc.).</li> <li>5. Il valide l'ajout.</li> <li>6. Le système vérifie les données, enregistre le patient et affiche un message de confirmation.</li> </ol>

<b>Scénarios alternatifs</b>	A1. Données obligatoires manquantes ou incorrectes : le système affiche un message d'erreur et invite à corriger. A2. Un patient avec les mêmes informations d'identification existe déjà : le système signale un doublon potentiel. A3. Problème technique lors de l'enregistrement : le système affiche une erreur et suggère de réessayer ultérieurement.
------------------------------	--

#### 4.6 Diagramme de cas d'utilisation "Gérer les dossiers médicaux"

Le diagramme de cas d'utilisation "Gérer les dossiers médicaux" est illustré dans la figure II.7.

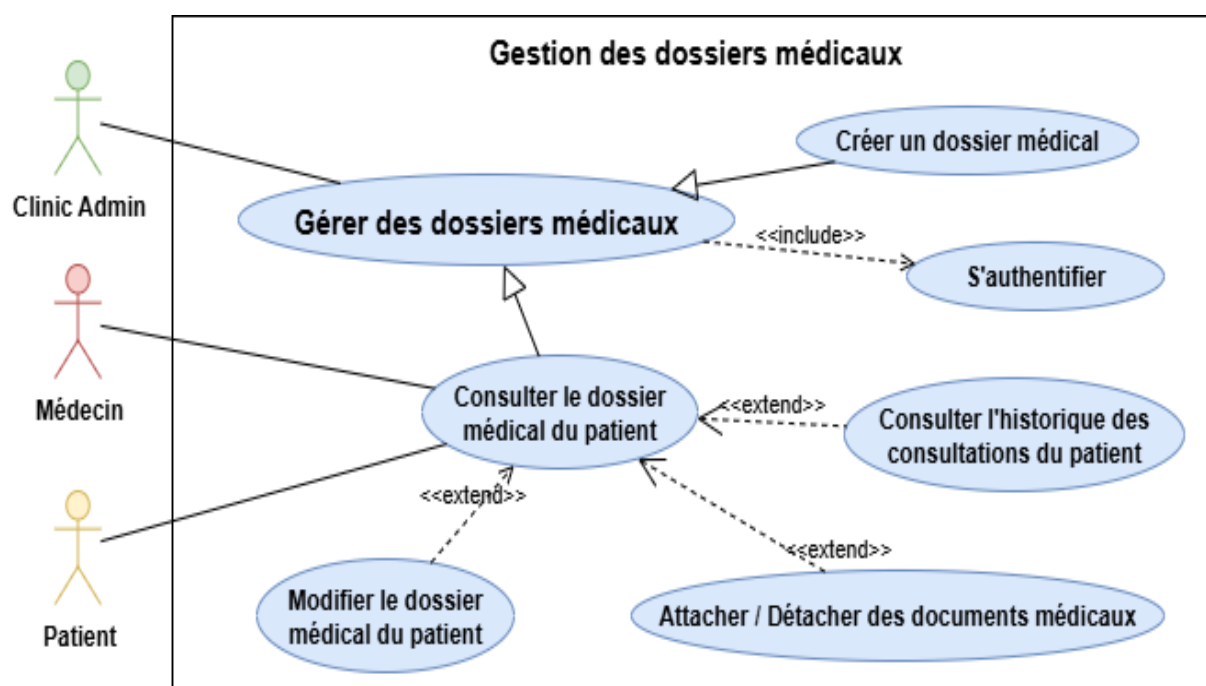


FIGURE II.7 – Diagramme de cas d'utilisation "Gérer les dossiers médicaux"

TABLE II.7 – Raffinement du cas d'utilisation : Consulter le dossier médical du patient

<b>Nom du cas</b>	Consulter le dossier médical du patient
<b>Acteurs</b>	Clinic Admin, Médecin, Patient
<b>Objectif</b>	Permettre à un médecin, un administrateur de clinique ou à un patient d'accéder aux informations médicales du patient.
<b>Préconditions</b>	L'utilisateur est authentifié et dispose des autorisations nécessaires.
<b>Postconditions</b>	Le dossier médical du patient est affiché à l'écran.
<b>Scénario principal</b>	1. L'utilisateur accède à la fonctionnalité de consultation. 2. Il sélectionne un patient depuis la liste. 3. Le système récupère les données médicales associées au patient. 4. Le dossier médical s'affiche à l'écran.

<b>Scénarios alternatifs</b>	A1. Aucun patient n'est sélectionné : le système affiche un message invitant à sélectionner un patient. A2. L'utilisateur n'a pas les droits d'accès : un message d'erreur s'affiche.
------------------------------	--

#### 4.7 Diagramme de cas d'utilisation "Gérer les rendez-vous"

Le diagramme de cas d'utilisation "Gérer les rendez-vous" est illustré dans la figure II.8.

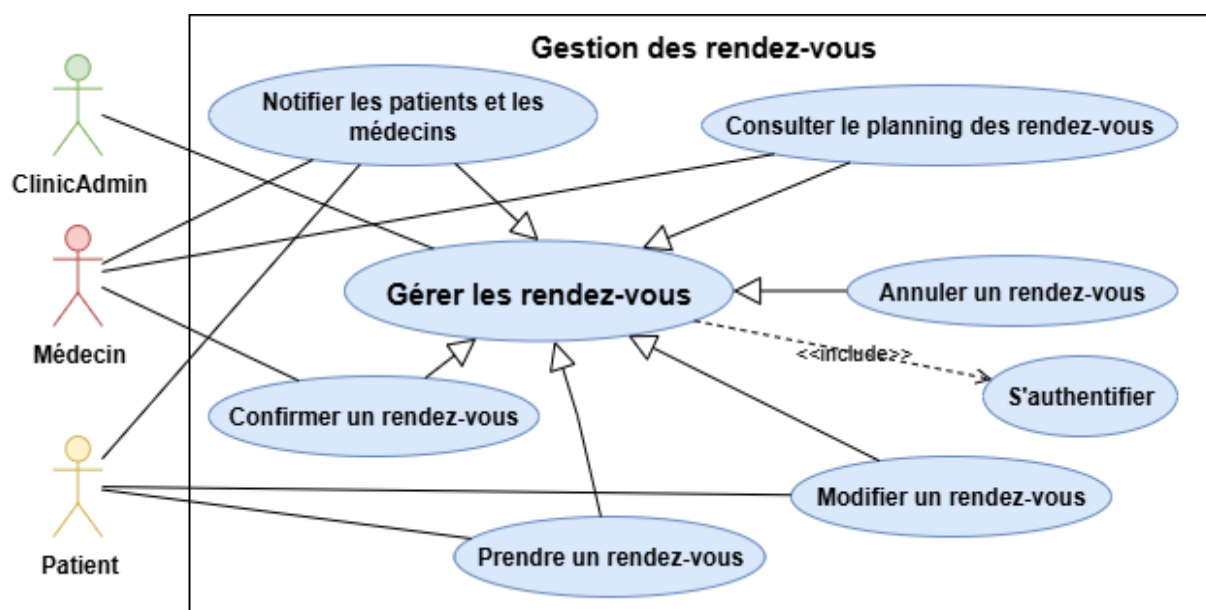


FIGURE II.8 – Diagramme de cas d'utilisation "Gérer les rendez-vous"

TABLE II.8 – Raffinement du cas d'utilisation : Prendre un rendez-vous

<b>Nom du cas</b>	Prendre un rendez-vous
<b>Acteurs</b>	Patient
<b>Objectif</b>	Permettre à un patient de réserver un rendez-vous auprès d'un médecin via le système.
<b>Préconditions</b>	Le patient est authentifié et a accès au système.
<b>Postconditions</b>	Le rendez-vous est enregistré dans le planning du médecin et notifié si nécessaire.
<b>Scénario principal</b>	1. Le patient accède à l'espace de prise de rendez-vous. 2. Il sélectionne une spécialité ou un médecin. 3. Il choisit une date et une heure disponibles dans le planning. 4. Il valide sa demande de rendez-vous. 5. Le système enregistre le rendez-vous et affiche une confirmation.

<b>Scénarios alternatifs</b>	<p>A1. Aucun créneau n'est disponible : le système invite à choisir une autre date.</p> <p>A2. Le patient annule le processus avant la validation : aucun rendez-vous n'est enregistré.</p>
------------------------------	---

#### 4.8 Diagramme de cas d'utilisation "Gérer les consultations"

Le diagramme de cas d'utilisation "Gérer les consultations" est illustré dans la figure II.9.

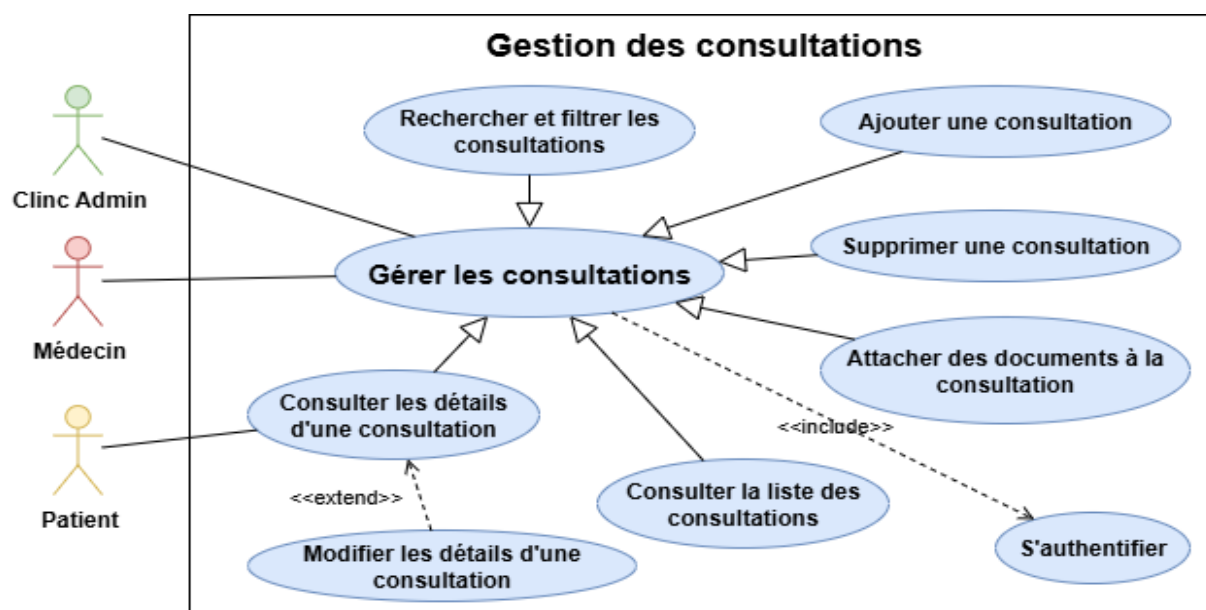


FIGURE II.9 – Diagramme de cas d'utilisation "Gérer les consultations"

TABLE II.9 – Raffinement du cas d'utilisation : Ajouter une consultation

<b>Nom du cas</b>	Ajouter une consultation
<b>Acteurs</b>	Médecin, Clinic Admin
<b>Objectif</b>	Permettre à un utilisateur autorisé d'enregistrer une nouvelle consultation dans le système.
<b>Préconditions</b>	L'utilisateur est authentifié et a accès au module de gestion des consultations.
<b>Postconditions</b>	La consultation est enregistrée et disponible dans la liste des consultations.
<b>Scénario principal</b>	<ol style="list-style-type: none"> <li>1. L'utilisateur accède à l'interface d'ajout de consultation.</li> <li>2. Il sélectionne le patient concerné.</li> <li>3. Il saisit les informations liées à la consultation (date, symptômes, diagnostic, traitement, etc.).</li> <li>4. Il valide la création.</li> <li>5. Le système enregistre les données et confirme l'ajout.</li> </ol>

<b>Scénarios alternatifs</b>	<p>A1. Des informations obligatoires sont manquantes : le système affiche un message d'erreur.</p> <p>A2. Le patient sélectionné n'existe pas dans la base : l'utilisateur est invité à vérifier ou créer le patient.</p>
------------------------------	---

#### 4.9 Diagramme de cas d'utilisation "Gérer les factures et paiements"

Le diagramme de cas d'utilisation "Gérer les factures et paiements" est illustré dans la figure II.10.

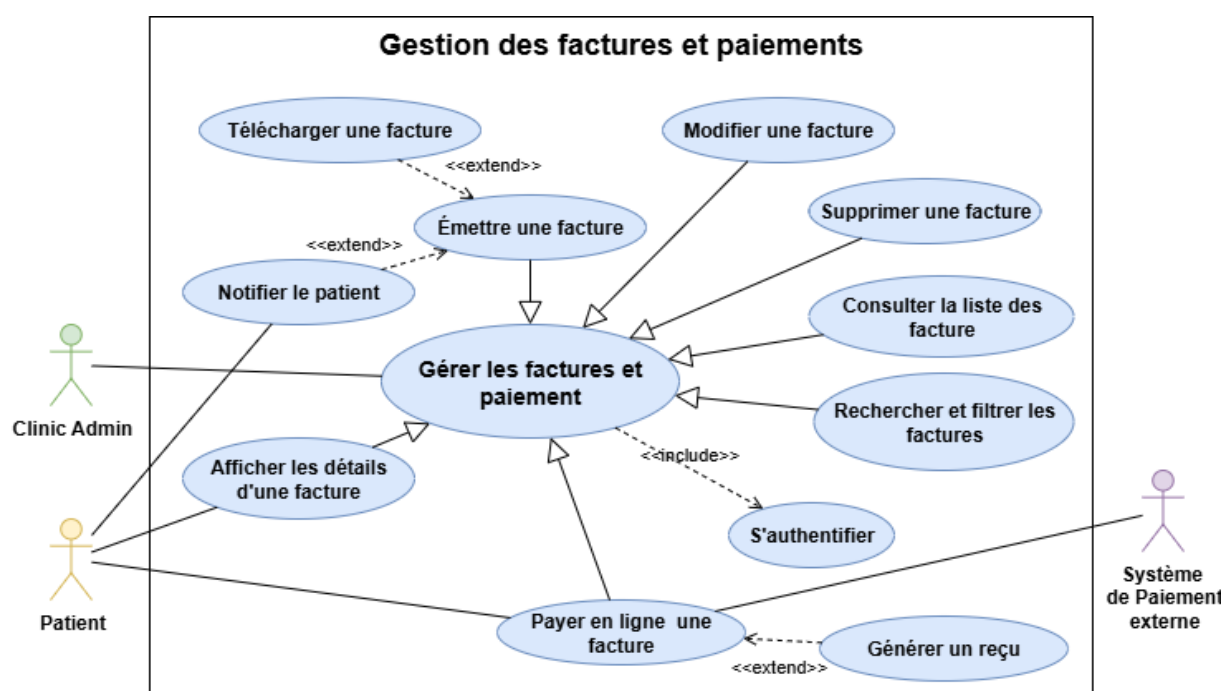


FIGURE II.10 – Diagramme de cas d'utilisation "Gérer les factures et paiements"

TABLE II.10 – Raffinement du cas d'utilisation : Payer une facture en ligne

<b>Nom du cas</b>	Payer une facture en ligne
<b>Acteurs</b>	Patient, Système de paiement externe
<b>Objectif</b>	Permettre à un patient de régler une facture en ligne de manière sécurisée via un système de paiement externe.
<b>Préconditions</b>	Le patient est authentifié, dispose d'au moins une facture impayée et d'un moyen de paiement valide.
<b>Postconditions</b>	Le paiement est validé, la facture est marquée comme payée et un reçu est généré et envoyé au patient.



<b>Scénario principal</b>	<ol style="list-style-type: none"> <li>1. Le patient accède à la liste de ses factures.</li> <li>2. Il sélectionne une facture impayée.</li> <li>3. Le système affiche les détails de la facture.</li> <li>4. Le patient clique sur le bouton « Payer ».</li> <li>5. Le système redirige vers l'interface du système de paiement externe.</li> <li>6. Le patient saisit ses informations de paiement.</li> <li>7. Le système externe valide ou rejette le paiement.</li> <li>8. En cas de succès, le système met à jour l'état de la facture.</li> <li>9. Un reçu est généré.</li> <li>10. Une notification est envoyée au patient.</li> </ol>
<b>Scénarios alternatifs</b>	<p>A1. Paiement refusé : un message d'erreur est affiché et le patient peut réessayer.</p> <p>A2. Le patient annule l'opération : retour à la liste des factures sans modification.</p> <p>A3. Facture déjà réglée : le système empêche le paiement et affiche un message informatif.</p>

#### 4.10 Diagramme de cas d'utilisation "Gérer les notifications"

Le diagramme de cas d'utilisation "Gérer les notifications" est illustré dans la figure II.11.

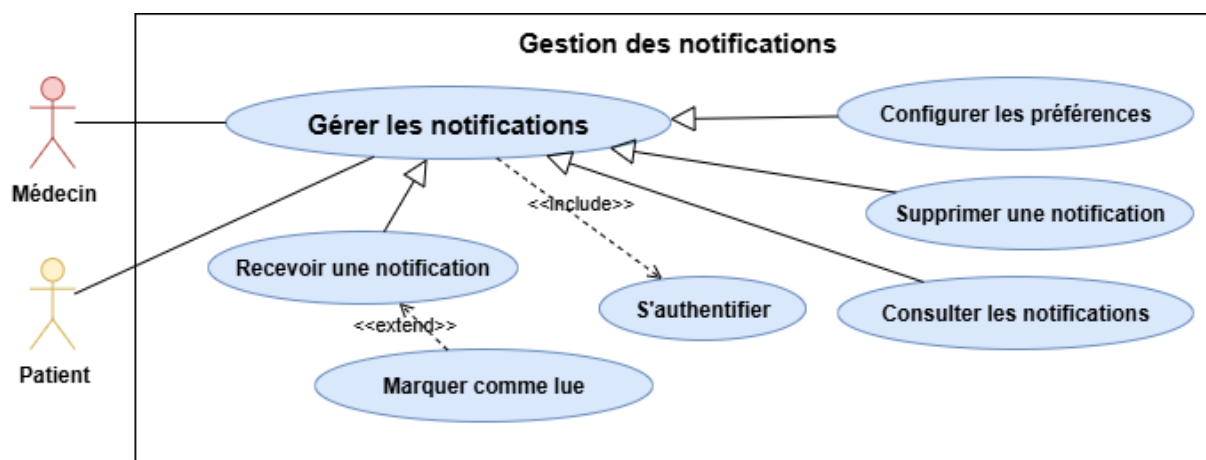


FIGURE II.11 – Diagramme de cas d'utilisation "Gérer les notifications"

TABLE II.11 – Raffinement du cas d'utilisation : Consulter les notifications

<b>Nom du cas</b>	Consulter les notifications
<b>Acteurs</b>	Patient, Médecin
<b>Objectif</b>	Permettre à l'utilisateur de visualiser l'ensemble de ses notifications (nouvelles ou archivées).
<b>Préconditions</b>	L'utilisateur est authentifié et possède au moins une notification disponible dans le système.



<b>Postconditions</b>	Les notifications sont affichées à l'écran ; l'utilisateur peut les lire ou effectuer d'autres actions (marquer comme lue, supprimer, etc.).
<b>Scénario principal</b>	<ol style="list-style-type: none"> <li>1. L'utilisateur accède à l'interface de gestion des notifications.</li> <li>2. Le système récupère les notifications liées à l'utilisateur.</li> <li>3. Le système affiche les notifications triées par date (les plus récentes en premier).</li> <li>4. L'utilisateur parcourt la liste et lit les notifications.</li> </ol>
<b>Scénarios alternatifs</b>	<p>A1. Aucune notification n'est trouvée : le système affiche un message du type « Aucune notification disponible ».</p> <p>A2. Erreur de récupération des données : le système affiche un message d'erreur technique.</p>

#### 4.11 Diagramme de cas d'utilisation "Gérer les rapports et les statistiques"

Le diagramme de cas d'utilisation "Gérer les rapports et les statistiques" est illustré dans la figure II.12.

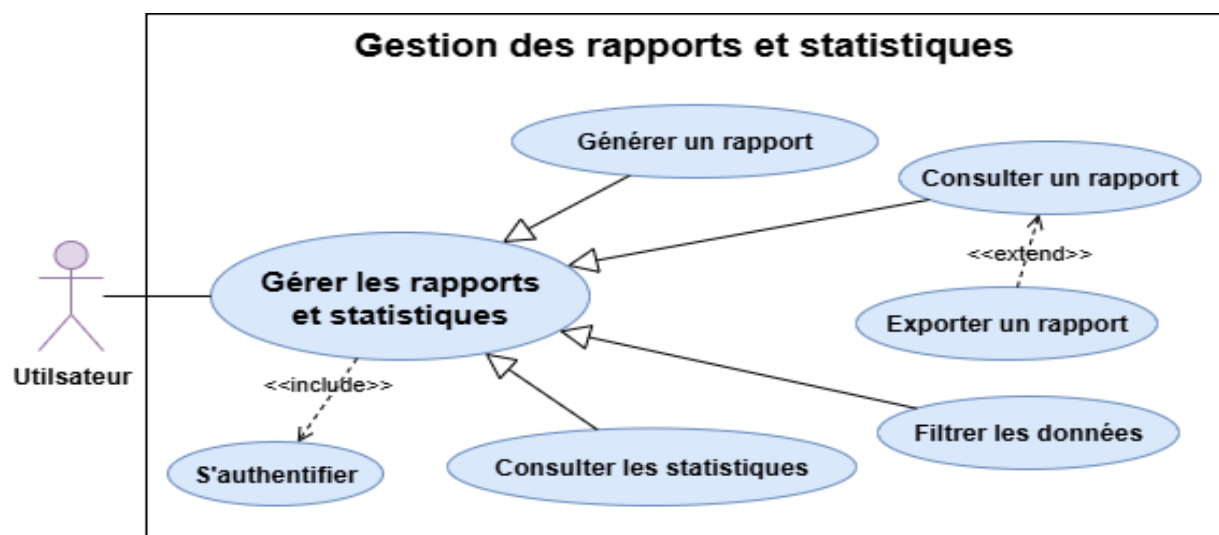


FIGURE II.12 – Diagramme de cas d'utilisation "Gérer les rapports et les statistiques"

TABLE II.12 – Raffinement du cas d'utilisation : Gérer les rapports et statistiques

<b>Nom du cas</b>	Gérer les rapports et statistiques
<b>Acteur</b>	Utilisateur (SuperAdmin, ClinicAdmin, Médecin, Patient)
<b>Objectif</b>	Permettre à un utilisateur authentifié de générer, consulter, filtrer, exporter des rapports, et visualiser des statistiques personnalisées selon son rôle (Patient, Médecin, Admin, ClinicAdmin ou Super-Admin).

<b>Préconditions</b>	L'utilisateur est authentifié et dispose des autorisations nécessaires pour accéder au service de reporting.
<b>Postconditions</b>	Les rapports ou statistiques personnalisés sont affichés ou exportés conformément aux critères définis par l'utilisateur et à ses droits d'accès.
<b>Scénario principal</b>	<ol style="list-style-type: none"><li>1. L'utilisateur se connecte au système.</li><li>2. Il accède au module de gestion des rapports et statistiques.</li><li>3. Le système identifie son rôle (ex. Médecin, Admin).</li><li>4. L'utilisateur sélectionne une action (générer, consulter, exporter un rapport, consulter des statistiques).</li><li>5. Il définit des critères (plage de dates, services, utilisateurs, etc.).</li><li>6. Le système affiche un rapport ou des statistiques personnalisées selon son profil.</li><li>7. L'utilisateur peut exporter les résultats si souhaité.</li></ol>
<b>Scénarios alternatifs</b>	<p>A1. L'utilisateur n'a pas les permissions nécessaires : le système affiche un message d'erreur d'accès.</p> <p>A2. Les critères de filtrage sont invalides : le système invite à les corriger.</p> <p>A3. Une erreur technique empêche la génération : le système affiche une alerte et suggère une reprise.</p>

## Conclusion

Au cours de ce chapitre, nous avons identifié les différents acteurs du système, ainsi que défini les exigences fonctionnelles et non fonctionnelles de la plateforme. Nous avons également modélisé les cas d'utilisation afin de structurer les besoins sous forme de scénarios concrets. Cette modélisation constitue une base solide pour la conception technique du système et facilitera par la suite la validation fonctionnelle à travers des tests.

## *Chapitre III Conception de l'application*

### Introduction

Dans ce chapitre, nous abordons l'aspect architectural de mon projet en mettant en évidence les raisons de notre choix et en présentant les différents services qui composent la plateforme.

## 1 Architecture Globale de l'Application

Dans notre projet, nous avons adopté une architecture microservices. Ce choix a été justifié par une étude comparative entre l'architecture monolithique et l'architecture microservices.

### 1.1 Choix de l'Architecture en Microservices

Pour bien illustrer les caractéristiques des différentes architectures, nous avons choisi de dresser un tableau comparatif III.1 entre deux architectures différentes [8].

TABLE III.1 – Tableau comparatif des architectures

Critère	Architecture Monolithique	Architecture Microservices
Développement	Le développement est assez simple.	Une modification apportée à un élément a moins de risque d'entraîner des changements non prévus au sein d'autres éléments, puisque les modules sont relativement indépendants.
Testabilité	Le test est très simple. Il suffit de lancer l'application et de lancer des tests de bout en bout.	Les services sont indépendants et faiblement couplés.
Couplage	Tous les composants sont interconnectés et interdépendants.	Les services sont indépendants et faiblement couplés.
Flexibilité technologique	Architecture monolithique n'est pas flexible. Nous ne pouvons pas utiliser différentes technologies. La pile technologique est décidée au début et suivie tout au long.	Les services n'ont pas besoin de partager la même pile technologique, les mêmes bibliothèques ou les mêmes frameworks. En effet, les choix technologiques peuvent être adaptés aux besoins spécifiques de chaque micro-service.
<i>Suite à la page suivante...</i>		

*Suite du tableau comparatif*

Critère	Architecture Monolithique	Architecture Micro-services
Fiabilité	Ce n'est pas fiable. Si une fonctionnalité tombe en panne, toute l'application peut tomber en panne.	Les services peuvent être déployés indépendamment. Une équipe peut mettre à jour un service existant sans reconstruire ni redéployer l'intégralité du système.
Évolutivité	Les applications monolithiques sont difficiles à faire évoluer une fois qu'elles sont plus grandes.	Les services sont responsables de la persistance de leurs propres données ou de leur état externe.

**✓ Synthèse :**

Suite à l'étude des critères des deux architectures, nous avons choisi d'adopter l'architecture micro-services pour la mise en place de notre application. Notre choix est justifié par plusieurs raisons :

- L'extensibilité : Tolérer l'injection d'autres composants selon la demande de nos clients.
- L'utilisation des microservices par des grandes entreprises comme Amazon, Netflix, eBay donne assez de confiance pour que ce style d'architecture soit prêt à être évalué et utilisé par les développeurs d'applications professionnelles.
- La réduction du délai de mise en marché ou « time to market »
- La factorisation et la réutilisation des microservices.

## 1.2 La Conception Pilotée par le Domaine

La conception pilotée par le domaine ou Domain Driven Design (DDD) est une approche de conception logicielle qui propose une modélisation basée sur la réalité de l'entreprise comme référence pour les cas d'utilisation. Dans le contexte de la création d'applications, DDD parle de problèmes en tant que domaines. Il décrit les parties de problématiques indépendantes en termes des contextes limités (chaque contexte limité est corrélé à un microservice) et met l'accent sur un langage commun pour aborder ces problèmes [9].

Alors, Nous avons opté pour le concept de DDD lors de la répartition des microservices dont chacun prend en charge une ou plusieurs tâches associées.

## 1.3 Architecture Globale de l'Application

Dans cette section, nous présentons l'architecture globale de l'application, basée sur une approche microservices. L'application se compose de 12 microservices backend et d'un microservice frontend. Chaque microservice fonctionne de manière indépendante et communique avec les autres via des APIs REST sécurisées, orchestrées par l'API Gateway.

La figure III.1 illustre la structure modulaire de l'application, mettant en évidence les interactions entre les services principaux.

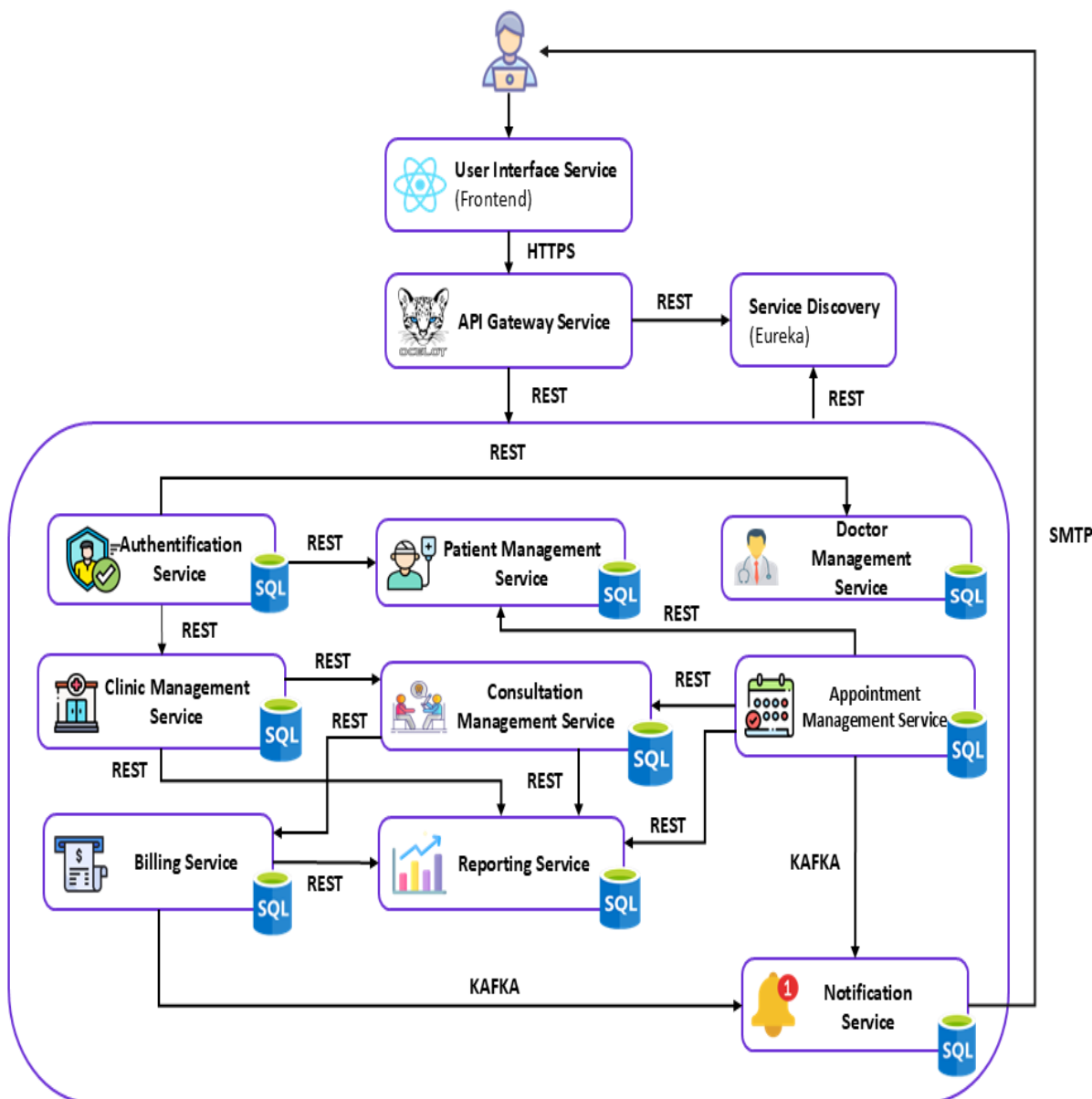


FIGURE III.1 – Architecture globale de l'application

- **Authentication Service** : Gère l'authentification, l'inscription des utilisateurs, la gestion des rôles et des permissions. Il est également responsable de la gestion des informations des utilisateurs.
- **Clinic Management Service** : Responsable de la gestion des cliniques (coordonnées, spécialités, horaires, etc.). Il fournit des endpoints pour la création, la mise à jour et la suppression des cliniques.
- **Patient Management Service** : Gère les opérations liées aux patients : enregistrement, mise à jour des informations personnelles, consultation du dossier médical et de l'historique de soins.
- **Doctor Management Service** : Permet la gestion des profils des médecins : création, modification, spécialités, disponibilité et emploi du temps.
- **Appointment Management Service** : Gère les rendez-vous : prise de rendez-vous, gestion des créneaux, modification, annulation et confirmation. Il interagit avec les services

de gestion des médecins et des patients pour assurer la cohérence des données.

- **Consultation Service** : Prend en charge l'enregistrement des consultations : diagnostics, prescriptions, comptes rendus, etc.
- **Billing Service** : Assure la gestion des factures, des devis et des paiements. Il génère automatiquement les factures à la suite d'une consultation ou d'un acte médical.
- **Notification Service** : Envoie des notifications (email, SMS, push) aux patients et professionnels pour les rappels de rendez-vous ou les événements importants.
- **Reporting Service** : Génère des rapports analytiques (nombre de consultations, revenus, statistiques d'utilisation) exportables au format PDF ou Excel.
- **API Gateway Service** : Sert de point d'entrée unique pour toutes les requêtes. Il centralise la sécurité, le routage, l'authentification, et assure l'équilibrage de charge.
- **Eureka Service** : Implémente un registre de services (Service Discovery) via Netflix Eureka, permettant aux microservices de se découvrir et de communiquer dynamiquement.
- **User Interface Service** : Seul microservice frontend. Il offre une interface utilisateur réactive via une application web développée avec React, destinée aux patients, médecins et administrateurs.

## 1.4 Routage des Requêtes vers les Microservices

Le microservice « API-Gateway » assure le routage des requêtes des utilisateurs vers les microservices concernés. Autrement dit, l'utilisateur envoie sa requête au point d'entrée unique, sans se soucier du service qui la traitera en arrière-plan. Prenons l'exemple d'un super administrateur souhaitant consulter la liste des utilisateurs. Plutôt que d'envoyer directement sa requête au microservice « Authentication-service », il la transmet au microservice « API-Gateway ». Ce dernier vérifie d'abord sa validité, puis la redirige automatiquement vers le service concerné si elle est autorisée. Par conséquent, la gateway doit connaître l'ensemble des microservices disponibles ainsi que les types de requêtes qu'ils sont capables de traiter.

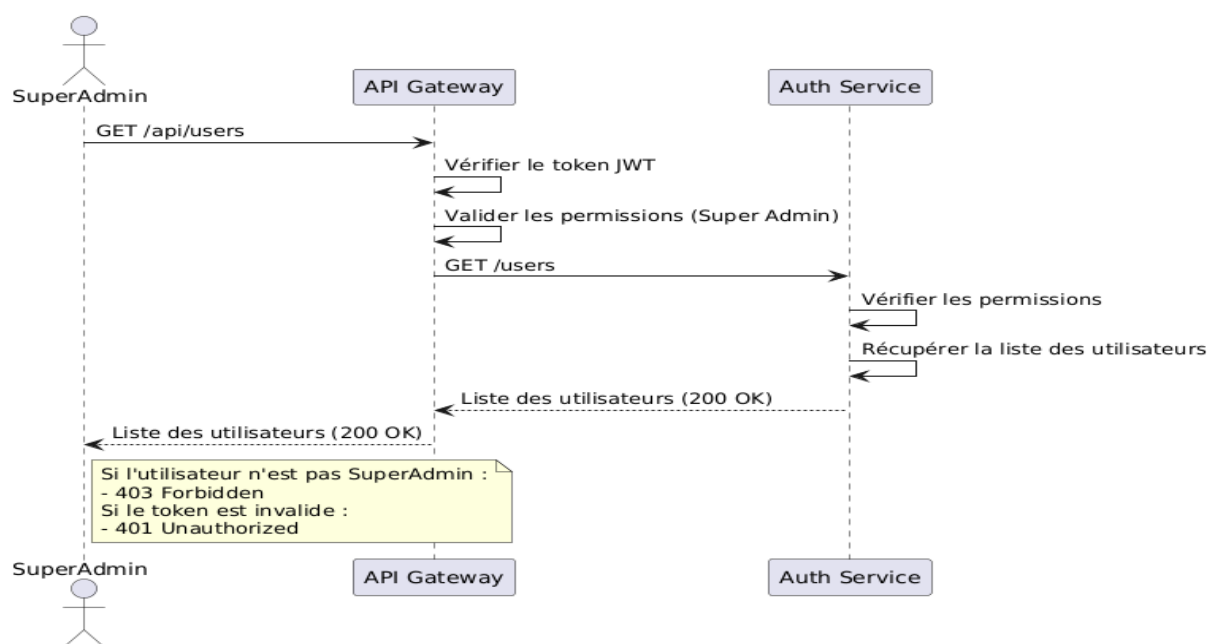


FIGURE III.2 – Diagramme de séquence de "API-Gateway"

La figure III.2 illustre le diagramme de séquence du microservice "API-Gateway". Le super administrateur envoie une requête pour obtenir la liste des utilisateurs. La gateway la reçoit, en vérifie la validité et l'authentification. En cas d'erreur (accès non autorisé, requête malformée...), un message d'erreur est immédiatement retourné au client. Sinon, la requête est redirigée vers le microservice responsable — ici, le « authentication-service ». Ce dernier traite la demande, génère la réponse (liste des utilisateurs), la renvoie à la gateway, qui la transmet à son tour au super administrateur.

## 1.5 Communication entre les composants de l'application

Les microservices de l'application interagissent à travers plusieurs modes de communication, adaptés selon le besoin (temps réel, décorrélation, notification, etc.) :

- **Communication via REST (HTTPS)** [10] : utilisée pour les échanges synchrones entre services. Ce mode est privilégié pour les requêtes nécessitant une réponse immédiate (ex. : consultation d'informations via API). Il est également utilisé entre le frontend et l'API Gateway, ainsi qu'entre certains microservices.
- **Communication via Kafka** [11] : adoptée pour les communications asynchrones. Kafka permet la transmission d'événements inter-services (Event-Driven Architecture), comme l'envoi automatique d'une notification après la prise d'un rendez-vous ou le déclenchement d'un rapport.
- **Communication via SMTP** [12] : utilisée pour l'envoi de courriers électroniques à destination des utilisateurs (patients, médecins, administrateurs). Elle est exploitée pour les confirmations de rendez-vous, rappels automatiques ou alertes système.

## 1.6 Gestion des bases de données par microservice

Conformément aux principes de l'architecture microservices, chaque service dispose de sa propre base de données indépendante. Cette séparation garantit [13] :

- L'**autonomie** de chaque service, qui peut évoluer indépendamment,
- L'**isolation des pannes**, limitant la propagation d'erreurs,
- La **cohérence du domaine fonctionnel**, chaque base étant strictement liée au périmètre métier du service.

Ce découplage respecte également les fondements du *Domain-Driven Design (DDD)*, où chaque *Bounded Context* possède son propre modèle de données.

Voici un aperçu de la répartition des bases de données :

- **Authentication Service** : stocke les données des utilisateurs, rôles, permissions et jetons d'authentification.
- **Clinic Management Service** : contient les informations des cliniques (adresse, spécialités, disponibilités...).
- **Patient Management Service** : gère les données personnelles des patients ainsi que leur dossier médical.
- **Doctor Management Service** : enregistre les informations professionnelles des médecins (profil, spécialités, emploi du temps...).

- **Appointment Management Service** : conserve les données relatives aux rendez-vous (créneaux, statuts, historique...).
- **Consultation Service** : stocke les comptes rendus de consultation, diagnostics et prescriptions.
- **Billing Service** : gère les informations de facturation, paiements, devis et leurs statuts.
- **Notification Service** : enregistre les notifications envoyées (type, contenu, destinataires, statut).
- **Reporting Service** : conserve les rapports générés, les indicateurs de performance et les données statistiques.

Chaque microservice accède uniquement à sa propre base via ses API, sans accès direct inter-base. Cela favorise la scalabilité horizontale, la sécurité et la maintenabilité du système dans son ensemble.

## 2 Architecture des Microservices Réalisés

La figure III.3 illustre l'architecture en trois couches adoptée pour la majorité des microservices.

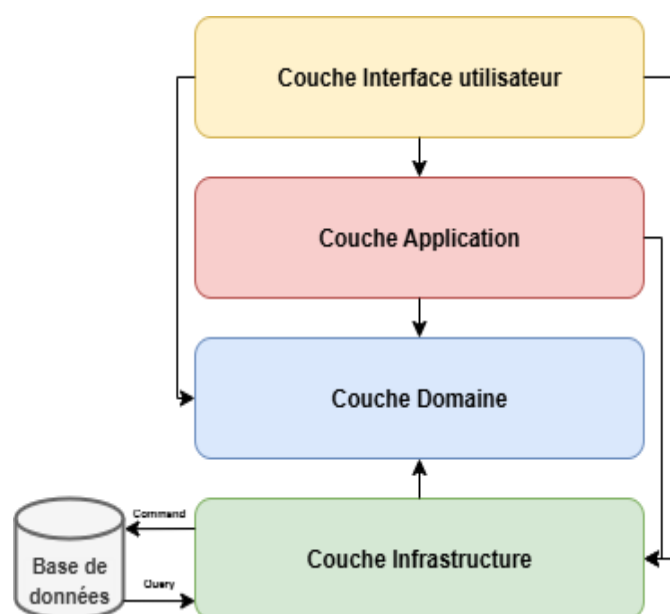


FIGURE III.3 – Architecture interne des microservices

Cette architecture repose sur une séparation claire des responsabilités, selon l'approche recommandée par [14] :

- **Couche Domaine (Domain Layer)** : Il s'agit du cœur de l'application ; cette couche contient toute la logique métier essentielle. Elle est indépendante de l'infrastructure, de la base de données, des frameworks ou de toute couche de présentation.
- **Couche Application (Application Layer)** : Elle orchestre la logique métier fournie par la couche Domaine pour répondre à des cas d'utilisation spécifiques. Elle ne contient pas de logique métier en soi, mais coordonne les appels aux agrégats, services de domaine et ressources d'infrastructure nécessaires.



- **Couche Infrastructure (Infrastructure Layer) :** Cette couche fournit des implémentations concrètes des interfaces définies dans les couches supérieures. Elle permet l'interaction avec le monde extérieur : bases de données, fichiers, services tiers, etc.
- **Couche Interface Utilisateur / Accès (Interface Layer) :** Elle sert de point d'entrée à l'application. Elle reçoit les requêtes externes, les traduit en appels vers la couche Application, puis renvoie les réponses appropriées.

### 3 Diagrammes UML

Dans cette section, nous présentons les principaux diagrammes UML permettant de décrire la structure statique et les interactions dynamiques du système. Ces diagrammes sont essentiels pour visualiser les entités principales, leurs relations ainsi que les échanges entre les composants du système à travers différents scénarios d'usage.

#### 3.1 Diagramme de classes global

La figure ci-dessous représente le diagramme de classes global de notre application de gestion de clinique. Il illustre les entités principales du système, leurs attributs, ainsi que les relations existantes entre elles.

- **Utilisateur (User)** est une entité centrale représentant toutes les personnes utilisant le système (patients, médecins, administrateurs de clinique, super administrateurs). Chaque utilisateur possède un identifiant, un nom complet, un email, un mot de passe chiffré et un rôle. Le rôle est défini par l'énumération `Role` (Super Administrateur, Administrateur de la clinique, Médecin, Patient).
- **Clinique** contient les informations de chaque clinique (nom, adresse, type, statut, etc.). Elle est reliée à un administrateur via `ClinicAdmin`.
- **ClinicAdmin, Medecin, Patient** et **SuperAdmin** héritent tous logiquement de la base `Utilisateur` (association 1-1), et portent les informations spécifiques à leur rôle.
- **Medecin** est relié à un ou plusieurs objets `Disponibilite` indiquant ses horaires. Il est également lié à des rendez-vous (`RendezVous`) et à des consultations (`Consultation`).
- **Patient** est lié à son `DossierMedical`, qui contient des documents médicaux (`Document`) et est associé à plusieurs consultations et rendez-vous.
- **Consultation** regroupe les diagnostics et notes du médecin pour une séance médicale. Elle peut contenir plusieurs `DocumentMedical`.
- **RendezVous** modélise les prises de rendez-vous entre médecins et patients, et possède un statut défini par l'énumération `RDVStatus` (Confirmé, Annulé, En attente).
- **Facture** est générée suite à une consultation et appartient à un patient. Elle contient le montant, le statut (Impayée, Payée, etc.) défini par `FactureStatus`, ainsi que le montant payé.

Ce diagramme structure l'ensemble des relations métiers de l'application et facilite l'implémentation DDD (Domain Driven Design) en séparant clairement les entités selon leurs responsabilités fonctionnelles.

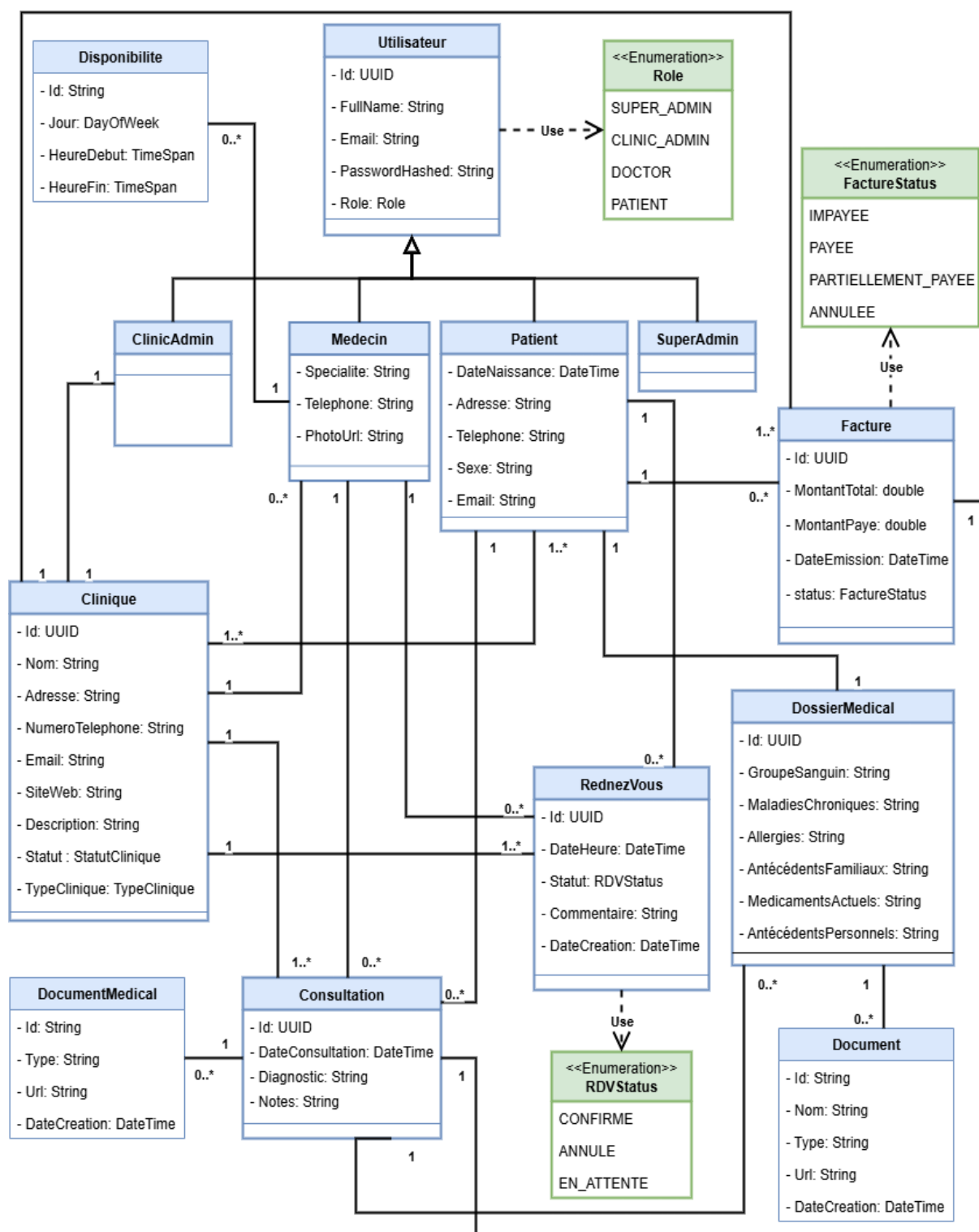


FIGURE III.4 – Diagramme de classes global de la plateforme

## 3.2 Diagrammes de séquences

Les diagrammes de séquences permettent de décrire le déroulement temporel des interactions entre les objets du système lors de cas d'utilisation spécifiques. Chaque scénario met en évidence les messages échangés entre les acteurs, la gateway, les services métiers et la base de données.

## Diagramme de séquences relatif à l'authentification

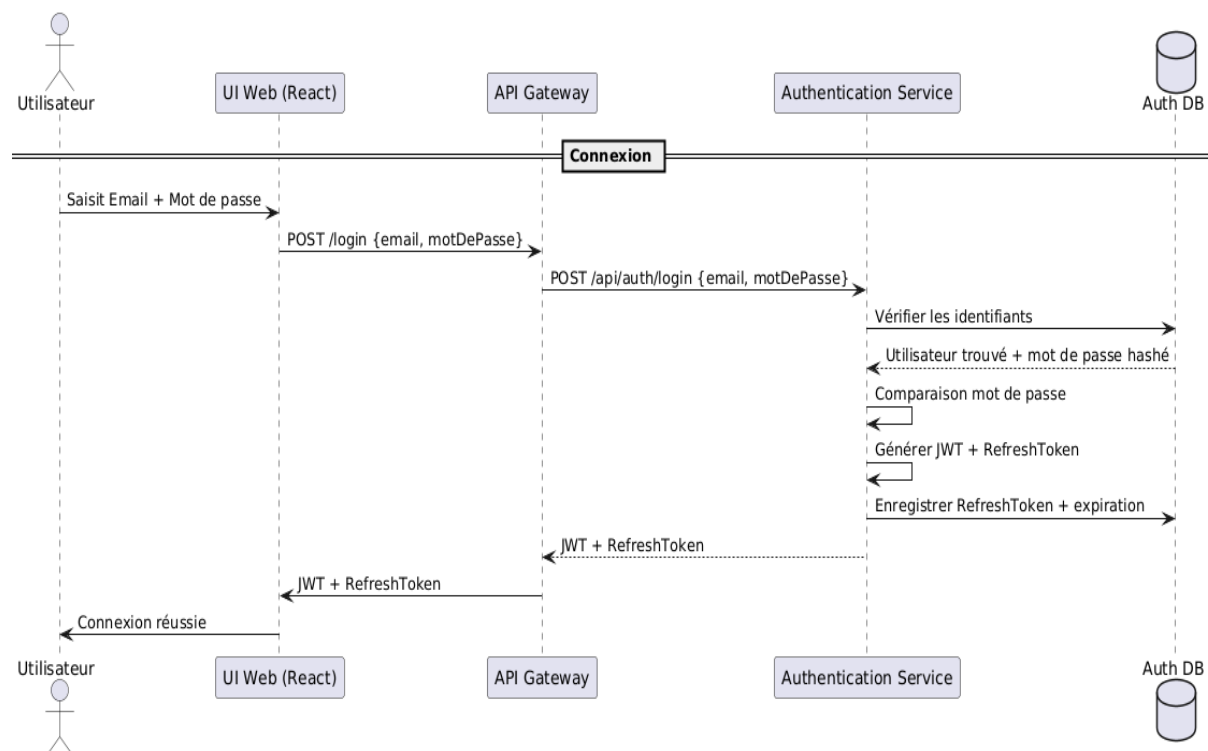


FIGURE III.5 – Diagramme de séquence — Authentification

## Diagramme de séquences relatif l'ajout d'une clinique

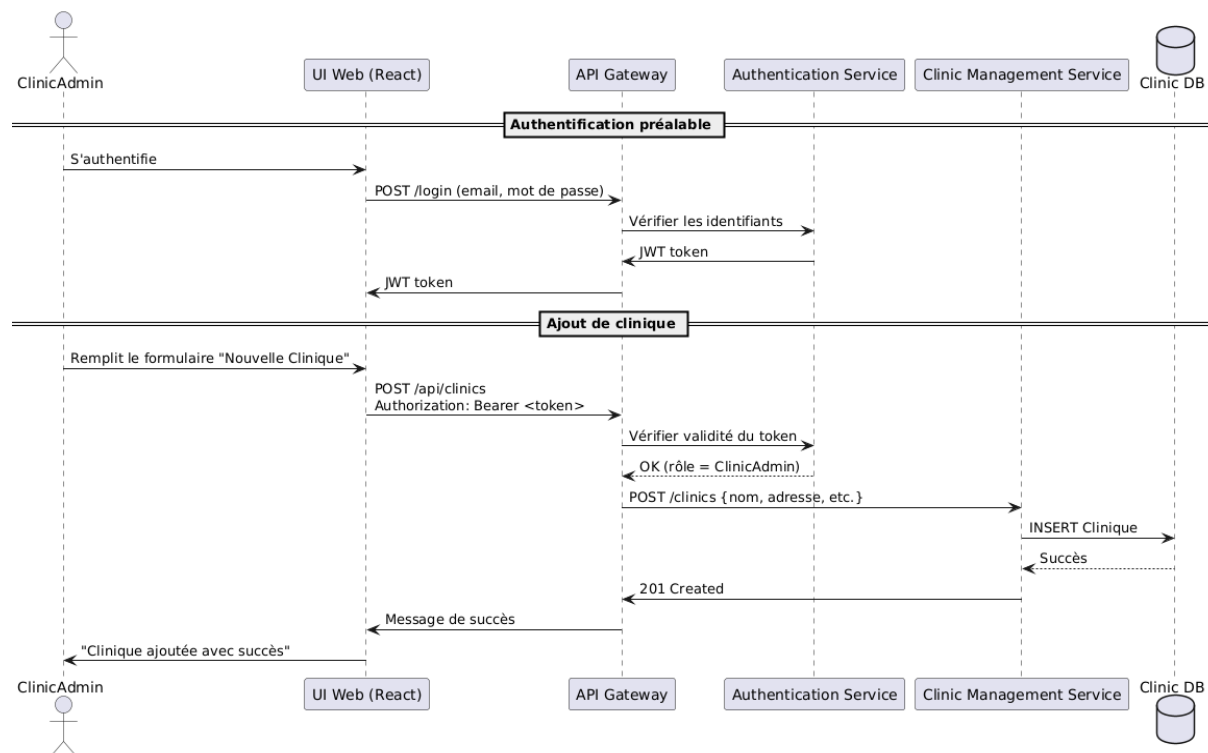


FIGURE III.6 – Diagramme de séquence — Ajouter une clinique

## Diagramme de séquences relatif à la prise d'un rendez-vous

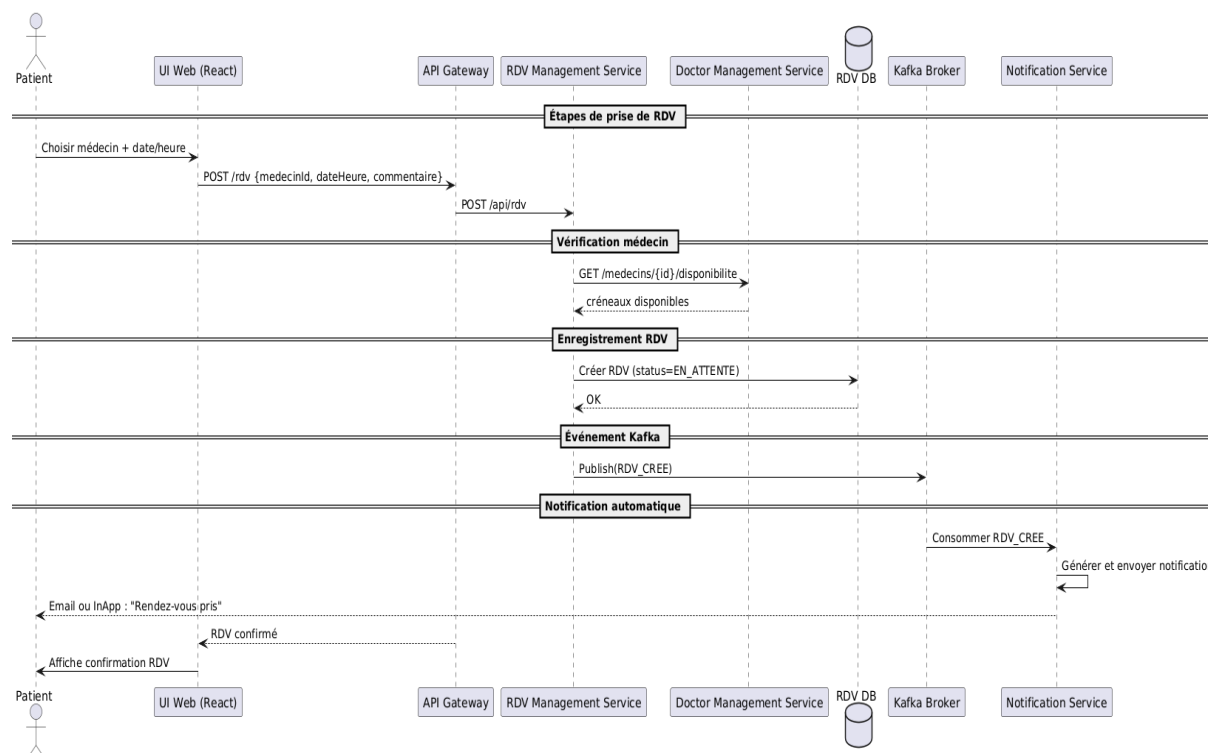


FIGURE III.7 – Diagramme de séquence — Prise de rendez-vous

## Diagramme de séquences relatif au paiement d'une facture en ligne

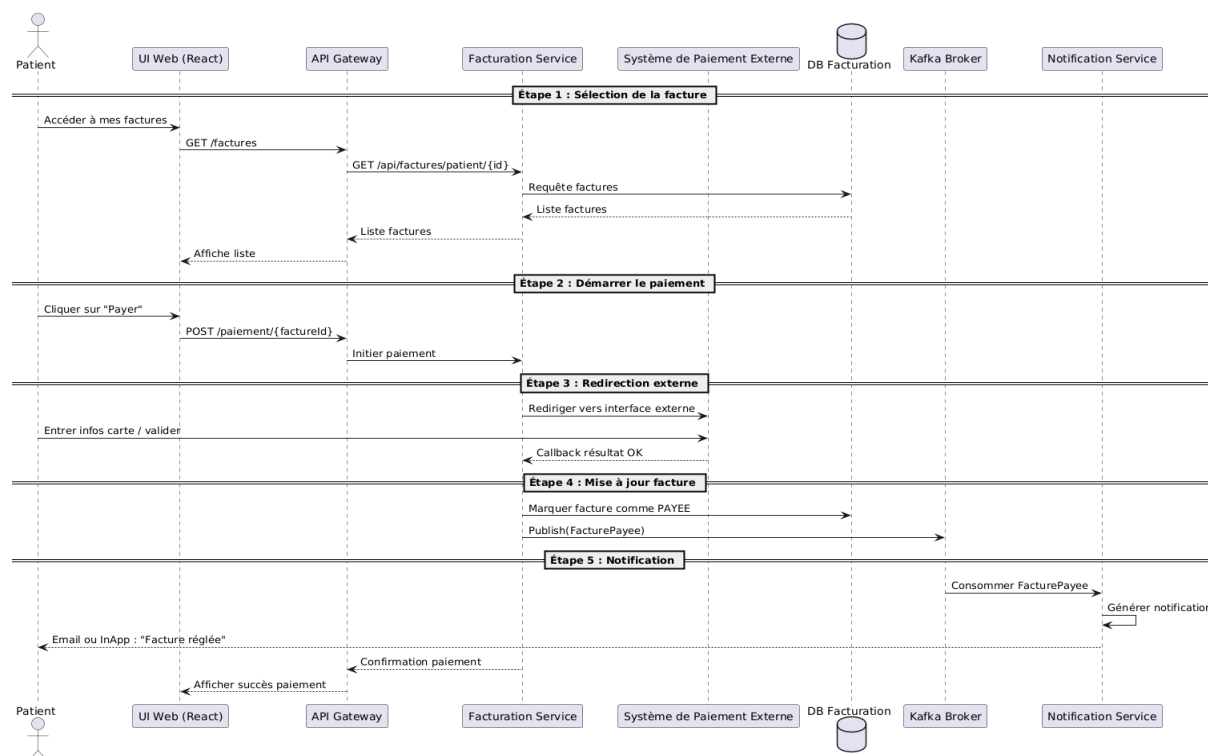


FIGURE III.8 – Diagramme de séquence — Paiement en ligne

## Diagramme de séquences relatif à la réception d'une notification

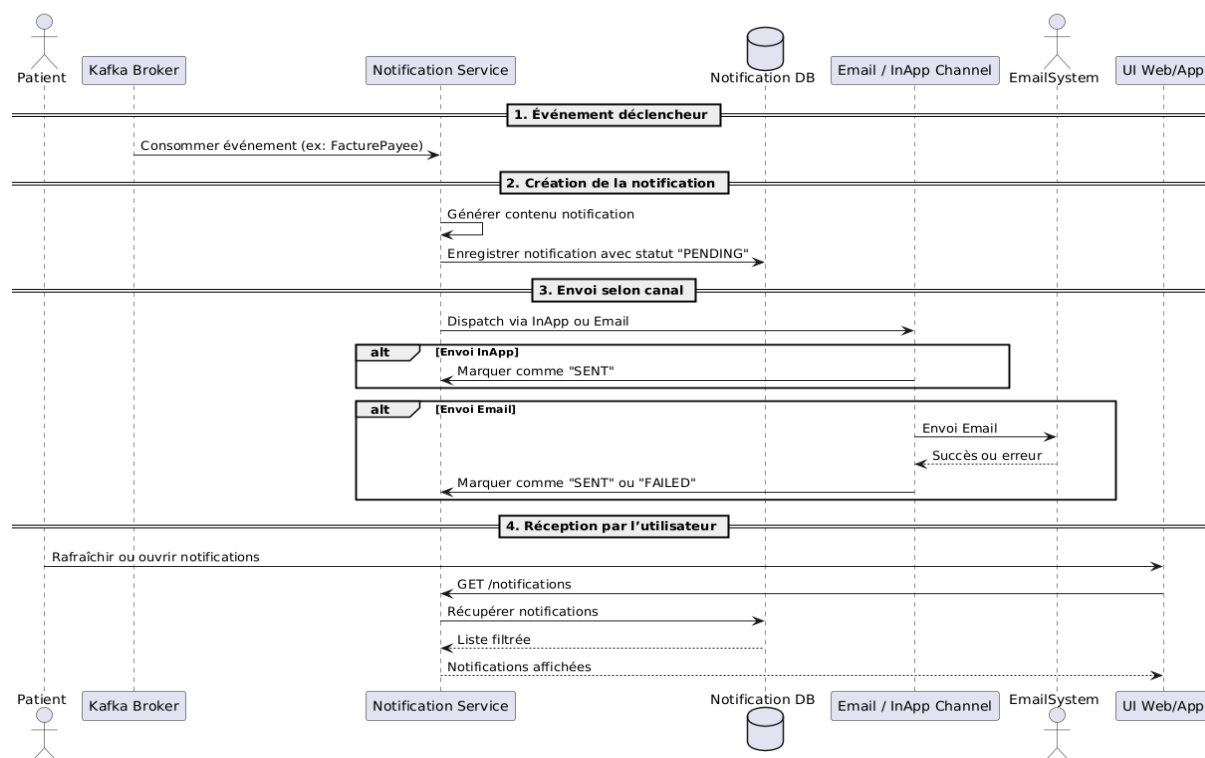


FIGURE III.9 – Diagramme de séquence — Notification utilisateur

## *Chapitre IV Réalisation*

### Introduction

#### 1 Environnement Matériel

Au cours de la phase de la réalisation de notre projet, nous avons utilisé un ordinateur DELL qui possède les caractéristiques suivantes IV.1 :

<b>Processeur</b>	Intel Core i7-11800H @ 2.30 GHz
<b>Mémoire RAM</b>	16 Go
<b>Disque Dur</b>	512 Go
<b>Système d'exploitation</b>	Windows 11 Professionnel 64 bits

TABLE IV.1 – Caractéristiques du Matériel

#### 2 Environnements Logiciels

Dans cette partie, nous allons citer les outils utilisés lors du réalisation de l'application afin de satisfaire les besoins.

- **Visual Studio [15]** : nous avons utilisé Visual Studio comme un éditeur de code pour développer la partie backend.
- **Visual Studio Code [16]** : C'est un IDE open source que nous avons l'utilisé afin de développer la partie front-end.
- **SQL Server Management System (SSMS) [17]** : Comme montre notre architecture globale 2.2 chaque microservice admet sa propre base de données SQL.
- **Postman [18]** : C'est un outil de développement d'API permettant aux développeurs de documenter, de tester et de collaborer sur des API. Nous l'avons utilisé pour de tester les fonctionnalités de l'application.
- **Draw.io [19]** : C'est une plateforme qui possède divers modules et une collection de modélisation en UML. Nous avons utilisé cet outil afin de créer l'architecture globale, les diagrammes de cas d'utilisation, de classes et de séquence.

#### 3 Technologies Utilisées

Dans cette section, nous présentons les technologies avec lesquelles nous effectuons notre projet.

### 3.1 Choix Technologique pour le Back-end

La partie back-end de l'application sera développée en utilisant C# comme langage de programmation.

- **.NET 9.0** : .NET 9.0 offre une performance élevée, une prise en charge native du multithreading et une compatibilité multiplateforme (Windows, Linux, macOS). Il est idéal pour construire des microservices robustes et évolutifs.
- **WebAPI** : WebAPI est utilisé pour exposer des endpoints RESTful, permettant une communication fluide entre les microservices et le front-end.
- **Entity Framework Core (EF Core)** : Entity Framework Core est un ORM (Object-Relational Mapping) puissant qui simplifie l'accès aux données et la gestion des bases de données.
- **xUnit** : xUnit est un framework de test unitaire pour .NET, permettant de garantir la qualité et la fiabilité du code.

### 3.2 Choix Technologique pour le Front-end

Après avoir choisi le framework .NET pour le back-end, nous allons choisir maintenant la technologie pour le front-end de l'application. Pour cela, on a choisi ReactJs.

- **React (TypeScript)** : React est une bibliothèque JavaScript largement adoptée pour la création d'interfaces utilisateur dynamiques et réactives.
- **Redux** : Redux est utilisé pour gérer l'état global de l'application, notamment les données des patients, des médecins et des rendez-vous.
- **Redux-Saga** : Redux-Saga est utilisé pour gérer les effets secondaires asynchrones, comme les appels API ou les notifications.
- **React Router** : React Router permet de gérer la navigation entre les différentes pages de l'application.
- **React Bootstrap** : React Bootstrap offre des composants UI prêts à l'emploi, permettant de créer des interfaces utilisateur modernes et responsives.
- **TDD (Test-Driven Development)** : Le TDD est utilisé pour garantir la qualité du code front-end en écrivant des tests avant le développement des fonctionnalités.

## 4 Travail réalisé et scénarios d'exécution

## 5 Tests Unitaires

## Conclusion

## *Chapitre V Intégration & DevOps*

### **Introduction**

#### **1 Conteneurisation & Orchestration**

#### **2 CI/CD**

#### **3 Observabilité & Monitoring**

#### **4 Sécurité & Gouvernance**

#### **5 Tests intégration & performance**

### **Conclusion**



## *Discussion et Perspectives*

## *Conclusion Générale*

## Références

- [1] *À propos de nous.* fr. <https://www.corilus.be/fr/a-propos-de-nous> (visité le 18 mars 2025).
- [2] *Doctolib.* <https://www.doctolib.fr> (visité le 11 juill. 2025).
- [3] *Cliniko.* <https://www.cliniko.com> (visité le 11 juill. 2025).
- [4] *Zocdoc.* <https://www.zocdoc.com> (visité le 11 juill. 2025).
- [5] *Qu'est-ce que Scrum et comment se lancer.* <https://www.atlassian.com/fr/agile/scrum> (visité le 18 mars 2025).
- [6] Hello POMELO. *9 points clés de la méthode agile Scrum.* fr-FR. 2023. <https://hello-pomelo.com/articles/9-points-cles-de-la-methode-agile-scrum/> (visité le 18 mars 2025).
- [7] *Azure DevOps Services | Microsoft Azure.* en-US. <https://azure.microsoft.com/en-us/products/devops> (visité le 19 mars 2025).
- [8] *What are microservices?* <http://microservices.io/index.html> (visité le 19 mars 2025).
- [9] JAMESMONTMAGNO. *Designing a DDD-oriented microservice - .NET.* en-us. 2022. <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice> (visité le 19 mars 2025).
- [10] *1. Microservices - Building Microservices [Book].* en. <https://www.oreilly.com/library/view/building-microservices/9781491950340/ch01.html> (visité le 14 juill. 2025).
- [11] *Apache Kafka.* en. <https://kafka.apache.org/documentation/> (visité le 14 juill. 2025).
- [12] *Simple Mail Transfer Protocol.* Request for Comments RFC 821. Internet Engineering Task Force, 1982. <https://datatracker.ietf.org/doc/rfc821/> (visité le 14 juill. 2025).
- [13] *Microservices Patterns - Chris Richardson.* en. <https://www.manning.com/books/microservices-patterns> (visité le 14 juill. 2025).
- [14] Vaughn VERNON. *Implementing domain-driven design.* Addison-Wesley, 2013.
- [15] ANANDMEG. *What is the Visual Studio IDE?* en-us. 2024. <https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022> (visité le 19 mars 2025).
- [16] *Visual Studio Code - Code Editing. Redefined.* en. <https://code.visualstudio.com/> (visité le 19 mars 2025).
- [17] RWESTMSFT. *SQL Server Management Studio (SSMS).* en-us. 2025. <https://learn.microsoft.com/en-us/ssms/sql-server-management-studio-ssms> (visité le 19 mars 2025).

- [18] *Postman : The World's Leading API Platform | Sign Up for Free.* en. <https://www.postman.com/> (visité le 19 mars 2025).
- [19] *diagrams.net (draw.io) : une plateforme pour créer des diagrammes et des organigrammes.* fr-FR. <https://www.blogdumoderateur.com/tools/diagrams-net-draw-io/> (visité le 19 mars 2025).
- [20] *GNU Health.* en. Page Version ID : 1264421796. 2024. [https://en.wikipedia.org/w/index.php?title=GNU\\_Health&oldid=1264421796](https://en.wikipedia.org/w/index.php?title=GNU_Health&oldid=1264421796) (visité le 18 mars 2025).
- [21] *OpenMRS.* en. Page Version ID : 1268124775. 2025. <https://en.wikipedia.org/w/index.php?title=OpenMRS&oldid=1268124775> (visité le 18 mars 2025).
- [22] JAMESMONTMAGNO. *.NET Microservices. Architecture for Containerized .NET Applications - .NET.* en-us. <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/> (visité le 14 juill. 2025).