TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABRÉVIATIONS

UML Unified Modeling Language

MVC Model-View-Controller

GLSI Génie Logiciel et Systèmes d'Information

HTTP Hypertext Transfer Protocol

SQL Structured Query Language

TS TypeScript

JSON JavaScript Object Notation

HTML HyperText Markup Language

CSS Cascading Style Sheets

vscode Virtual studio code vscode

INTRODUCTION GÉNÉRALE

Dans un monde numérique en constante évolution, la transformation digitale du secteur de la santé est devenue une nécessité stratégique pour améliorer la qualité des soins et optimiser la gestion administrative. Cette transition vers des systèmes informatisés permet non seulement d'améliorer l'efficacité opérationnelle des cabinets médicaux, mais aussi d'assurer une meilleure traçabilité des dossiers médicaux, de faciliter la communication entre professionnels de santé et patients, et d'intégrer des technologies innovantes telles que l'intelligence artificielle pour améliorer les services de santé.

Les cabinets médicaux traditionnels, confrontés à une croissance significative de leur patientèle et à une complexification des processus de gestion, font face à de nombreux défis : gestion manuelle des rendez-vous source d'erreurs et de conflits, absence de centralisation des dossiers médicaux, difficultés de suivi des factures et paiements, manque d'outils d'aide à la décision pour les patients. Ces problématiques engendrent non seulement une perte de temps considérable pour le personnel médical et administratif, mais représentent également un frein à l'amélioration de la qualité des services offerts aux patients.

C'est pour répondre à ces défis que nous avons conçu et développé une plateforme intégrée de gestion de cabinet médical combinant gestion administrative et intelligence artificielle. Ce projet s'articule autour de plusieurs objectifs majeurs : digitaliser la gestion des rendez-vous et des dossiers patients, automatiser la gestion des factures, intégrer un chatbot IA pour assister les patients dans leurs questions médicales, garantir la sécurité et la confidentialité des données médicales conformément aux normes RGPD, et offrir une interface intuitive accessible sur tous les supports (web, mobile, tablette). La solution technique retenue combine les technologies

modernes les plus performantes : Spring Boot pour le backend, Next.js pour le frontend, et l'API OpenAI pour le chatbot intelligent.

Ce rapport présente de manière structurée la méthodologie adoptée, l'analyse approfondie des besoins fonctionnels et non fonctionnels, la conception détaillée de l'architecture système, ainsi que les réalisations techniques et les résultats obtenus pour chaque sprint. Nous y détaillerons comment l'approche agile Scrum a été mise en œuvre pour gérer efficacement ce projet académique, en organisant le travail en trois sprints distincts : gestion de l'espace médecin, gestion de l'espace assistant, et gestion de l'espace patient. À travers ce travail, nous démontrons comment les technologies web modernes et l'intelligence artificielle peuvent transformer radicalement la gestion d'un cabinet médical en une solution digitale performante, sécurisée et centrée sur l'utilisateur.

Chapter

1 CHAPITRE 1 : Cadre du projet et choix méthodologique

Contents

1.1 INTRODUCTION 4

1.2 Contexte du projet 4

1.2.1 Problématiques identifiées 4

1.3 Solution proposée 5

1.3.1 Modules fonctionnels 6

1.4 Méthodologie adoptée 7

1.4.1 Méthodes agiles 7

1.4.2 Justification du choix : SCRUM 8

1.4.3 Les quatre valeurs fondamentales agiles 9

1.4.4 Les artefacts SCRUM 10

1.4.5 Les événements SCRUM 11

1.5 Architecture technique globale 12

1.5.1 Technologies choisies 12

1.5.2 Justification des choix techniques 13

1.6 CONCLUSION 13

1.1 INTRODUCTION

Ce projet académique s'inscrit dans le cadre du développement d'une solution innovante de gestion de cabinet médical intégrant l'intelligence artificielle. L'objectif principal est de digitaliser l'ensemble des processus médicaux et administratifs à travers une plateforme web full-stack moderne. Cette introduction présente le contexte du secteur de la santé, les problématiques identifiées dans la gestion traditionnelle des cabinets médicaux, et la méthodologie agile adoptée pour mener à bien ce projet.

1.2 Contexte du projet

Le secteur de la santé connaît une transformation digitale majeure, portée par les avancées technologiques et les nouveaux besoins des professionnels de santé et des patients. Les cabinets médicaux traditionnels font face à de nombreux défis : gestion manuelle des rendez-vous source d'erreurs et de conflits, absence de centralisation des dossiers médicaux, difficultés de suivi des factures et paiements, manque d'outils d'aide à la décision pour les patients, et collaboration limitée entre médecins et personnel administratif.

1.2.1 Problématiques identifiées

L'analyse du fonctionnement des cabinets médicaux traditionnels a permis d'identifier plusieurs problématiques critiques :

1. **Gestion des rendez-vous :**
    - Prise de rendez-vous uniquement par téléphone (surcharge du secrétariat)
    - Absence de visualisation des créneaux disponibles en temps réel
    - Risques de double réservation et conflits d'horaires
    - Difficulté de modification ou annulation pour les patients
    - Pas de rappels automatiques (taux d'absence élevé)

2. Gestion des dossiers médicaux :

- Dossiers papier volumineux et difficiles à archiver
- Risque de perte ou détérioration des documents
- Impossibilité d'accès à distance pour les patients
- Difficulté de partage entre professionnels de santé
- Ordonnances et analyses non centralisées

3. Gestion administrative et financière :

- Création manuelle des factures (Excel/Word)
- Suivi des paiements complexe et incomplet
- Absence de rapports financiers automatisés
- Difficulté d'identification des impayés
- Temps perdu estimé : 15-20h/mois

4. Communication et assistance :

- Patients sans réponses à leurs questions simples
- Surcharge du standard téléphonique
- Absence d'outil d'information médicale accessible 24/7

1.3 Solution proposée

Face aux lacunes identifiées dans la gestion traditionnelle des cabinets médicaux, nous proposons une solution innovante sous la forme d'une plateforme web full-stack intégrant l'intelligence artificielle. Cette solution vise à digitaliser l'ensemble du cycle de vie des activités médicales et administratives, tout en garantissant la sécurité et la confidentialité des données conformément aux normes RGPD et au secret médical.

La solution se décline autour de cinq modules fonctionnels clés :

1.3.1 Modules fonctionnels

1. Module Gestion des Rendez-vous

- Calendrier intelligent avec visualisation des créneaux disponibles
- Création de rendez-vous par les assistants avec vérification automatique des conflits
- Notifications automatiques par email aux patients
- Système de rappels avant rendez-vous
- Modification et annulation simplifiées

2. Module Gestion des Dossiers Médicaux

- Création de dossiers médicaux complets par les médecins (diagnostic, traitement, notes, allergies, antécédents)
- Upload et stockage sécurisé de documents (ordonnances, résultats d'analyses, imagerie)
- Consultation en ligne pour les patients avec historique complet
- Téléchargement de documents en format PDF
- Traçabilité complète des accès et modifications

3. Module Gestion Administrative

- Gestion des assistants par le médecin (création, activation/désactivation, modification, suppression)
- Contrôle d'accès basé sur les rôles (RBAC) garantissant la confidentialité
- Gestion centralisée des patients avec fiches complètes
- Interface dédiée par type d'utilisateur (médecin, assistant, patient)

4. Module Gestion Financière

- Création de factures par médecins et assistants (avec restrictions pour les assistants)
- Enregistrement des paiements multi-modes (CB, espèces, chèque, virement)
- Suivi automatique du statut des factures (payée/impayée)

- Génération de rapports financiers détaillés avec statistiques
- Visualisation graphique des revenus par période

5. Module Chatbot Intelligence Artificielle

- Assistant virtuel intelligent réservé aux patients
- Intégration avec l'API OpenAI pour analyse sémantique des questions
- Réponses médicales générales et conseils de prévention
- Orientation vers consultation médicale si nécessaire
- Historique des conversations sauvegardé

1.4 Méthodologie adoptée

Pour la réalisation de ce projet de développement et dans le cadre d'un projet académique collaboratif, il est essentiel de maîtriser les principales méthodologies de travail. Celles-ci assurent un cycle de vie efficace et transparent.

1.4.1 Méthodes agiles

Les méthodes agiles sont des pratiques de gestion de projet qui privilégient la flexibilité et l'adaptabilité pour fournir des solutions logicielles. Elles encouragent la collaboration, la communication transparente et une livraison rapide de valeur. Elles sont devenues un élément essentiel dans le développement logiciel, offrant un cadre solide pour la réalisation de projets efficaces et de qualité. Le tableau **??** ci-dessous met en œuvre une comparaison entre les différentes méthodes agiles.[**?**]

Table 1.1: Comparaison entre les méthodes agiles

Méthode	Description	Condition
eXtreme Programming (XP)	• La programmation réflexive • La conception par paire • L'intégration continue et les tests unitaires	• Projets nécessitant une haute qualité de code • Équipes disciplinées et communicantes
Feature Driven Development (FDD)	• La décomposition des fonctionnalités en modules • La planification par domaine • La collaboration entre les équipes	• Projets avec des fonctionnalités bien définies • Importance de la collaboration entre les équipes
SCRUM	• La planification par sprints • La collaboration continue • La livraison incrémentale de logiciels	• Projets complexes et changeants • Équipes auto-organisées

1.4.2 Justification du choix : SCRUM

Après analyse comparative des différentes méthodes agiles, nous avons opté pour la méthodologie SCRUM pour les raisons suivantes :

- **Adaptation au contexte académique :** SCRUM permet une organisation en sprints courts (2 semaines) parfaitement adaptée aux contraintes temporelles d'un projet universitaire.
- **Flexibilité :** Les besoins d'un système médical peuvent évoluer, SCRUM permet d'intégrer ces changements facilement entre les sprints.

- **Livraisons incrémentales :** Chaque sprint produit un livrable fonctionnel (espace médecin, puis assistant, puis patient), permettant des tests et validations progressives.
- **Collaboration :** SCRUM favorise la communication au sein de l'équipe et avec les parties prenantes (enseignants, utilisateurs tests).
- **Visibilité :** Le product backlog et les sprints offrent une vision claire de l'avancement du projet.

1.4.3 Les quatre valeurs fondamentales agiles

La méthodologie agile repose sur quatre valeurs fondamentales définies dans le Manifeste Agile :

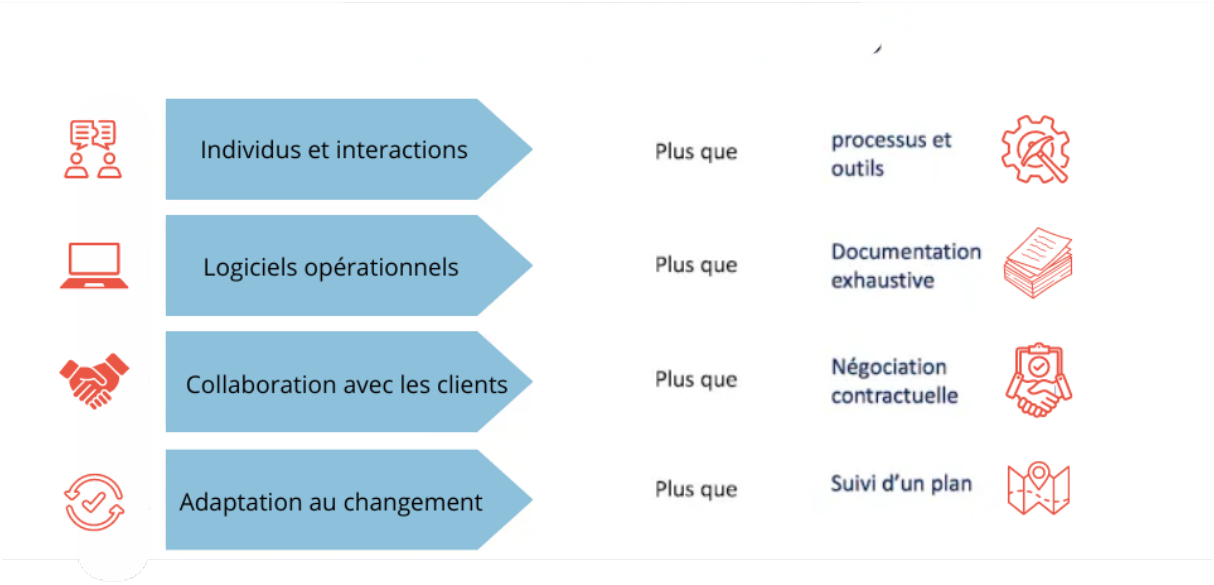

Figure 1.1: Les quatre valeurs agiles principales

1. **Les individus et leurs interactions plus que les processus et les outils :** Dans notre projet, nous privilégions la communication directe entre membres de l'équipe plutôt que de suivre aveuglément des procédures rigides.
2. **Des logiciels opérationnels plus qu'une documentation exhaustive :** Nous nous concentrons sur le développement de fonctionnalités fonctionnelles à chaque sprint, avec une documentation technique suffisante mais non excessive.

3. **La collaboration avec les utilisateurs plus que la négociation contractuelle :** Nous impliquons les utilisateurs potentiels (médecins, assistants, patients) dans les tests pour recueillir leurs retours et ajuster le système.
4. **L'adaptation au changement plus que le suivi d'un plan :** Bien qu'ayant planifié trois sprints, nous restons flexibles pour intégrer les retours et améliorer les fonctionnalités entre les itérations.

1.4.4 Les artefacts SCRUM

SCRUM utilise trois artefacts principaux pour gérer le projet :

1. Product Backlog :

- Liste ordonnée de toutes les fonctionnalités souhaitées pour le système de gestion de cabinet médical
- Priorisé par valeur métier (fonctionnalités critiques en premier)
- Contient les user stories détaillées avec critères d'acceptation
- Dynamique : peut évoluer entre les sprints

2. Sprint Backlog :

- Sous-ensemble du product backlog sélectionné pour un sprint spécifique
- Sprint 1 : Espace Médecin (gestion assistants, dossiers médicaux, factures)
- Sprint 2 : Espace Assistant (création patients, rendez-vous, factures restreintes)
- Sprint 3 : Espace Patient (inscription, consultation, chatbot IA)

3. Incrément :

- Résultat fonctionnel et potentiellement livrable à la fin de chaque sprint
- Doit respecter la "Definition of Done" (tests passés, code reviewé, déployable)
- S'ajoute aux incréments précédents pour construire progressivement le système complet

1.4.5 Les événements SCRUM

Notre projet suit les cérémonies SCRUM classiques :

1. **Sprint Planning :**
    - Durée : 2 heures au début de chaque sprint de 2 semaines
    - Sélection des user stories du product backlog pour le sprint
    - Décomposition des user stories en tâches techniques
    - Estimation de la charge de travail (en points ou en heures)

2. **Daily Scrum (adapté) :**
    - Réunions courtes (15 minutes) 3 fois par semaine
    - Chaque membre partage : ce qui a été fait, ce qui sera fait, les obstacles rencontrés
    - Permet d'ajuster rapidement et de résoudre les blocages

3. **Sprint Review :**
    - À la fin de chaque sprint (2 semaines)
    - Démonstration des fonctionnalités développées aux parties prenantes
    - Collecte des retours pour améliorer les prochains sprints

4. **Sprint Retrospective :**
    - Après la Sprint Review
    - Réflexion de l'équipe sur le processus (ce qui a bien fonctionné, ce qui peut être amélioré)
    - Définition d'actions concrètes pour améliorer le prochain sprint

1.5 Architecture technique globale

Notre solution repose sur une architecture moderne à trois tiers (3-tiers) combinant les meilleures technologies du moment :

1.5.1 Technologies choisies

Frontend :

- **Next.js 14 :** Framework React moderne avec App Router pour des performances optimales
- **TypeScript :** Typage statique pour réduire les erreurs et améliorer la maintenabilité
- **Tailwind CSS :** Framework CSS utility-first pour un design responsive rapide
- **Context API :** Gestion d'état centralisée pour l'authentification et les données utilisateur

Backend :

- **Spring Boot 3.5.7 :** Framework Java robuste pour applications d'entreprise
- **Java 21 :** Dernière version LTS avec fonctionnalités modernes
- **Spring Security :** Sécurisation avec JWT et contrôle d'accès basé sur les rôles
- **Spring Data JPA :** Accès aux données simplifié avec Hibernate
- **Spring Mail :** Envoi d'emails automatiques (notifications, confirmations)

Base de données :

- **PostgreSQL/MySQL :** SGBD relationnel pour stockage des données médicales
- **Système de fichiers :** Stockage sécurisé des documents médicaux (ordonnances, analyses)

Services externes :

- **OpenAI API :** Intelligence artificielle pour le chatbot médical

- **Serveur SMTP :** Envoi d'emails de notification

1.5.2 Justification des choix techniques

Pourquoi Next.js ?

- Rendu côté serveur (SSR) pour des performances optimales et SEO
- App Router moderne avec layouts imbriqués
- Optimisation automatique des images et du code
- Écosystème React riche en composants réutilisables

Pourquoi Spring Boot ?

- Framework mature et éprouvé pour applications critiques (secteur médical)
- Sécurité robuste avec Spring Security et support JWT natif
- Architecture modulaire facilitant la maintenance
- Large communauté et documentation complète
- Compatible avec les normes de conformité RGPD

Pourquoi JWT (JSON Web Tokens) ?

- Authentification stateless (pas de session serveur)
- Scalabilité facilitée
- Sécurisation des endpoints API REST
- Gestion des rôles (PATIENT, MEDECIN, ASSISTANT) intégrée dans le token

1.6 CONCLUSION

Ce chapitre a permis de poser les fondations du projet en présentant le contexte du secteur médical, les problématiques identifiées dans la gestion traditionnelle des cabinets, et la solution

digitale proposée. Le choix de la méthodologie agile SCRUM se justifie par sa flexibilité, son adaptation au contexte académique, et sa capacité à livrer des incréments fonctionnels réguliers. L'architecture technique retenue, combinant Next.js, Spring Boot et des services d'intelligence artificielle, offre un socle solide pour développer une solution moderne, sécurisée et évolutive.

Les chapitres suivants détailleront la phase d'étude préliminaire (spécification des besoins, conception UML, planification des sprints), puis la réalisation concrète de chaque sprint (espace médecin, assistant et patient), avant de conclure sur les résultats obtenus et les perspectives d'évolution du système.

Chapter

2

CHAPITRE 2 : Étude préliminaire

Contents

2.1 INTRODUCTION 16

2.2 Spécification des besoins 16

2.2.1 Identification des acteurs 16

2.2.2 Les besoins fonctionnels 17

2.2.3 Les besoins non fonctionnels 21

2.3 Détails fonctionnels 22

2.3.1 Diagramme de cas d'utilisation global 22

2.3.2 Diagramme de classes global 23

2.4 Mise en œuvre 25

2.4.1 Product backlog 25

2.4.2 Planification des sprints 29

2.4.3 Diagramme de Gantt 29

2.4.4 L'architecture du système 30

2.5 CONCLUSION 34

2.1 INTRODUCTION

Ce chapitre présente l'étude préliminaire menée dans le cadre du développement de la plateforme de gestion de cabinet médical intégrant l'intelligence artificielle. L'objectif de cette étude est de spécifier de manière exhaustive les besoins fonctionnels et non fonctionnels du système, de modéliser les processus métiers à travers des diagrammes UML, et de planifier la mise en œuvre en adoptant une méthodologie agile Scrum structurée en trois sprints distincts. Cette phase d'analyse constitue le fondement technique et fonctionnel du projet, permettant d'établir un socle solide pour la conception et la réalisation de la solution, tout en garantissant l'adéquation entre les attentes des acteurs médicaux (médecins, assistants, patients) et les livrables attendus.

2.2 Spécification des besoins

La phase de spécification des besoins est une étape fondamentale qui vise à acquérir une compréhension approfondie du contexte du système de gestion de cabinet médical. Elle contient l'identification des acteurs, la définition des besoins fonctionnels et non fonctionnels, ainsi que la détermination du diagramme de cas d'utilisation global et du diagramme de classes global.

2.2.1 Identification des acteurs

L'analyse des processus métiers a permis d'identifier quatre acteurs principaux interagissant avec le système de gestion de cabinet médical :

- **Patient** : Utilisateur externe qui consulte et gère ses informations médicales. Il peut prendre des rendez-vous en ligne (en sélectionnant une date et un créneau disponible), consulter ses rendez-vous, consulter ses dossiers médicaux et documents (ordonnances, analyses), télécharger ses fichiers médicaux, et interagir avec le chatbot IA pour poser des questions médicales. Le patient s'inscrit avec une vérification par email pour garantir l'authenticité de son compte.
- **Médecin** : Professionnel de santé principal qui supervise l'ensemble du cabinet. Il peut gérer les assistants (création, activation/désactivation), consulter le calendrier des rendez-vous, créer et modifier les dossiers médicaux avec upload de documents

(ordonnances, résultats d'analyses), gérer les factures et enregistrer les paiements, et générer des rapports financiers. Il a un accès complet à toutes les fonctionnalités du système.

- **Assistant** : Personnel administratif du cabinet gérant les aspects organisationnels. Il peut planifier et modifier des rendez-vous, créer des factures (uniquement pour les patients liés via les rendez-vous qu'il a créés). Ses accès sont restreints pour garantir la confidentialité et le contrôle d'accès basé sur les rôles. Note : L'enregistrement de nouveaux patients se fait via la page d'inscription publique (/register) pour des raisons de sécurité et de gestion des comptes utilisateurs.
- **Système (Chatbot IA)** : Assistant virtuel intelligent réservé aux patients. Il analyse les questions médicales posées par les patients, fournit des conseils généraux basés sur l'intelligence artificielle (OpenAI), et enregistre l'historique des conversations. Le chatbot n'établit pas de diagnostic mais oriente les patients vers une consultation si nécessaire.

2.2.2 Les besoins fonctionnels

Les fonctionnalités de notre application web et les services qu'elle offre aux différents acteurs sont présentés ci-dessous :

1- En tant que Patient :

- **S'inscrire** : Le patient peut créer un compte avec vérification par email (code à 6 chiffres) via POST /api/auth/register-patient.
- **S'authentifier** : Le patient a la possibilité de se connecter à son compte de manière sécurisée avec username et mot de passe (JWT).
- **Prendre un rendez-vous** : Le patient peut créer un rendez-vous en ligne via /dashboard/rendezvous en sélectionnant une date dans le calendrier, en consultant les créneaux disponibles du médecin, et en choisissant un créneau libre (les dates passées sont désactivées).
- **Consulter mes rendez-vous** : Le patient peut visualiser ses prochains rendez-vous (date, heure, motif, médecin, statut) et l'historique complet via GET /api/rendezvous/patient/{patien
- **Consulter mes dossiers médicaux** : Le patient peut accéder à ses dossiers médicaux complets (diagnostic, traitement, symptômes, observations, recommandations) via GET /api/dossiers/patient/{patientId}.

- **Télécharger mes documents** : Le patient peut télécharger ses ordonnances, résultats d'analyses et autres documents médicaux via GET /api/dossiers/{dossierId}/files/{docId}.
- **Consulter mes factures** : Le patient peut visualiser toutes ses factures avec statut (payées et impayées), montant, date d'émission via GET /api/factures/patient/{patientId}.
- **Poser des questions au chatbot IA** : Le patient peut interagir avec l'assistant virtuel intelligent (réservé aux patients) via POST /api/chatbot/ask pour obtenir des conseils médicaux généraux, poser des questions sur ses consultations passées avec mention de rendez-vous spécifique (@RDV : date).
- **Gérer mes notifications** : Le patient peut consulter ses notifications (nouveaux RDV, dossiers, factures), marquer comme lues via PATCH /api/notifications/{id}/mark-as-read.
- **Consulter le tableau de bord** : Le patient peut visualiser son espace personnel avec accueil personnalisé, prochains rendez-vous (date, médecin, motif), et accès rapides (Mes rendez-vous, Mes dossiers médicaux) via /dashboard.
- **Gérer son profil** : Le patient peut consulter et modifier ses informations personnelles (nom, prénom, email, téléphone, date de naissance, adresse) via PUT /api/users/profile et /dashboard/profil.
- **Gérer les paramètres** : Le patient peut configurer ses préférences de notifications (activation/désactivation emails, délai de rappel en heures, email personnalisé pour notifications), changer son mot de passe via POST /api/account/change-password, et supprimer son compte via DELETE /api/account via /dashboard/parametres.

2- En tant que Médecin :

- **S'authentifier** : Le médecin peut se connecter à son compte sécurisé.
- **Créer un assistant** : Le médecin peut ajouter de nouveaux assistants au cabinet avec attribution de comptes.
- **Gérer les assistants** : Le médecin peut consulter, modifier les informations, activer/désactiver ou supprimer définitivement des assistants.
- **Gérer les patients** : Le médecin peut consulter la liste complète des patients, modifier leurs informations personnelles, ou supprimer un patient du système.
- **Consulter les rendez-vous** : Le médecin peut visualiser tous les rendez-vous, incluant les rendez-vous créés par les assistants et les patients.
- **Gérer les rendez-vous** : Le médecin peut créer des rendez-vous pour les patients, consulter les détails, vérifier les créneaux disponibles et consulter l'historique complet.

- **Modifier un dossier médical** : Le médecin peut mettre à jour les informations d'un dossier existant via l'API PUT /api/dossiers/{id} (modification du diagnostic, traitement, ajout d'observations).
- **Ajouter des documents** : Le médecin peut uploader des ordonnances, résultats d'analyses (PDF, images) dans les dossiers patients via l'API POST /api/dossiers/{id}/files.
- **Télécharger des documents** : Le médecin peut télécharger les documents médicaux associés à un dossier.
- **Créer des factures** : Le médecin peut créer des factures pour n'importe quel patient du cabinet.
- **Enregistrer les paiements** : Le médecin peut enregistrer les paiements reçus avec différents modes (CB, espèces, chèque, virement) via PATCH /api/factures/{id}/payer.
- **Supprimer une facture** : Le médecin peut supprimer définitivement une facture via DELETE /api/factures/{id} (réservé au médecin uniquement).
- **Générer des rapports financiers** : Le médecin peut obtenir des statistiques détaillées sur une période donnée (revenus, factures payées/impayées, répartition par mode de paiement) via /dashboard/rapports. Cette fonctionnalité est exclusive au médecin avec redirection automatique si un assistant ou patient tente d'y accéder.
- **Consulter le tableau de bord** : Le médecin peut visualiser les statistiques en temps réel (nombre de patients, rendez-vous du mois, factures en attente) et les prochains rendez-vous via /dashboard.
- **Gérer son profil** : Le médecin peut consulter et modifier ses informations personnelles (nom, prénom, email, téléphone, date de naissance, spécialité, description) via /dashboard/profil.
- **Gérer les paramètres** : Le médecin peut configurer ses préférences de notifications (activation/désactivation emails, délai de rappel, email personnalisé), changer son mot de passe et supprimer son compte via /dashboard/parametres.

3- En tant qu'Assistant :

- **S'authentifier** : L'assistant peut se connecter avec ses identifiants (username et mot de passe) fournis par le médecin.
- **Consulter les patients** : L'assistant peut voir la liste complète des patients (GET /api/patients/allPatients) ou uniquement ses patients liés via les rendez-vous qu'il a créés (GET /api/patients/mes-patients). Note : L'enregistrement de nouveaux patients se fait via la page d'inscription publique (/register) par les patients eux-mêmes.

- **Modifier un patient** : L'assistant peut mettre à jour les informations personnelles d'un patient via PUT /api/patients/update/{id}.
- **Créer un rendez-vous** : L'assistant peut planifier des rendez-vous via POST /api/rendezvous/assist en sélectionnant patient, médecin, date et créneau horaire.
- **Consulter les créneaux disponibles** : L'assistant peut vérifier les disponibilités du médecin via GET /api/rendezvous/medecin/{medecinId}/slots-disponibles avant de fixer un rendez-vous.
- **Consulter les rendez-vous** : L'assistant peut visualiser tous ses rendez-vous créés, les rendez-vous du jour et l'historique complet.
- **Modifier un rendez-vous** : L'assistant peut modifier les rendez-vous qu'il a créés via PATCH /api/rendezvous/assistants/rdv/{id} (date, heure, motif).
- **Annuler un rendez-vous** : L'assistant peut annuler un rendez-vous qu'il a créé via DELETE /api/rendezvous/assistants/rdv/{id}.
- **Créer une facture** : L'assistant peut créer des factures UNIQUEMENT pour les patients liés aux rendez-vous qu'il a créés (vérification automatique par le système via GET /api/patients/mes-patients).
- **Consulter les factures** : L'assistant visualise uniquement les factures liées aux rendez-vous qu'il a créés (filtrage automatique par le backend).
- **Enregistrer les paiements** : L'assistant peut marquer une facture comme payée via PATCH /api/factures/{id}/payer avec sélection du mode de paiement.
- **Consulter le tableau de bord** : L'assistant peut visualiser ses statistiques personnelles (nombre de patients gérés, rendez-vous créés, factures du mois, rendez-vous du jour) via /dashboard.
- **Gérer son profil** : L'assistant peut consulter et modifier ses informations personnelles (nom, prénom, email, téléphone) via /dashboard/profil.
- **Gérer les paramètres** : L'assistant peut configurer ses préférences de notifications (activation/désactivation emails, délai de rappel, email personnalisé), changer son mot de passe et supprimer son compte via /dashboard/parametres.

2.2.3 Les besoins non fonctionnels

Les exigences non fonctionnelles définissent les objectifs relatifs à la performance, la sécurité et la qualité du système de gestion de cabinet médical. Ces aspects, bien qu'invisibles pour l'utilisateur, sont cruciaux pour garantir un système robuste, fiable et conforme aux régulations médicales. Notre application doit satisfaire aux critères suivants :

- **Sécurité :** L'application doit garantir la confidentialité et l'intégrité des données médicales sensibles. Cela inclut l'authentification sécurisée par JWT (JSON Web Tokens), le chiffrement des mots de passe avec BCrypt, le contrôle d'accès basé sur les rôles (RBAC) pour restreindre les accès selon le type d'utilisateur, et la conformité au RGPD pour la protection des données personnelles et médicales. Le système doit également tracer tous les accès aux dossiers médicaux.
- **Fiabilité :** L'application web doit être fiable et robuste, minimisant les erreurs système. Elle doit assurer la cohérence des données (pas de conflits de rendez-vous, intégrité référentielle en base de données), gérer les transactions de manière atomique (création de dossier + upload de fichiers), et implémenter un système de sauvegarde automatique quotidienne des données.
- **Performance :** Le système doit garantir des temps de réponse rapides (< 2 secondes pour les opérations courantes, < 5 secondes pour la génération de rapports). Il doit supporter au moins 100 utilisateurs simultanés sans dégradation des performances, optimiser les requêtes de base de données (indexation, pagination), et assurer un chargement rapide des fichiers médicaux (images, PDF).
- **Disponibilité :** La plateforme doit viser une disponibilité de 99.5% (moins de 3.6 heures d'interruption par mois). Un plan de reprise après sinistre doit être en place, avec des sauvegardes régulières et des procédures de restauration testées.
- **Utilisabilité :** L'interface doit être intuitive et accessible même pour des utilisateurs non techniques. Navigation claire avec menus organisés par rôle, formulaires avec validation en temps réel et messages d'erreur explicites, design responsive compatible mobile/tablette/desktop, et support multi-navigateurs (Chrome, Firefox, Safari, Edge).
- **Maintenabilité :** Le code doit être modulaire et documenté pour faciliter l'évolution. Architecture en couches (Controllers, Services, Repositories), respect des principes SOLID, tests unitaires et d'intégration, et documentation technique complète (JavaDoc, commentaires).

- **Évolutivité :** Le système doit permettre l'ajout facile de nouvelles fonctionnalités (téléconsultation, gestion de stock de médicaments, intégration avec d'autres systèmes médicaux). Architecture modulaire permettant l'ajout de nouveaux modules sans refonte complète.
- **Conformité réglementaire :** Respect du secret médical, traçabilité complète des accès et modifications, archivage légal des dossiers médicaux, et conformité RGPD avec gestion du consentement et droit à l'oubli.

2.3 Détails fonctionnels

Dans cette section, nous présenterons le diagramme de cas d'utilisation général et le diagramme de classes global pour le système de gestion de cabinet médical.

2.3.1 Diagramme de cas d'utilisation global

Un diagramme de cas d'utilisation global est un outil de modélisation UML qui offre une vue d'ensemble des interactions entre les utilisateurs (appelés acteurs) et le système. Il représente visuellement les fonctionnalités du système et la manière dont les quatre acteurs (Patient, Médecin, Assistant, Chatbot IA) les exploitent.

images/use-case-cabinet-medical.png

Figure 2.1: Diagramme de cas d'utilisation global

2.3.2 Diagramme de classes global

Un diagramme de classes est essentiel pour visualiser la structure du système de gestion de cabinet médical. Il identifie les entités principales (User, Patient, Médecin, Assistant, RendezVous, DossierPatient, Facture, Document, Notification), ainsi que leurs attributs, méthodes et relations

(héritage, associations, compositions). Il facilite la communication entre les membres de l'équipe de développement et sert de fondation pour la conception et l'implémentation du système.

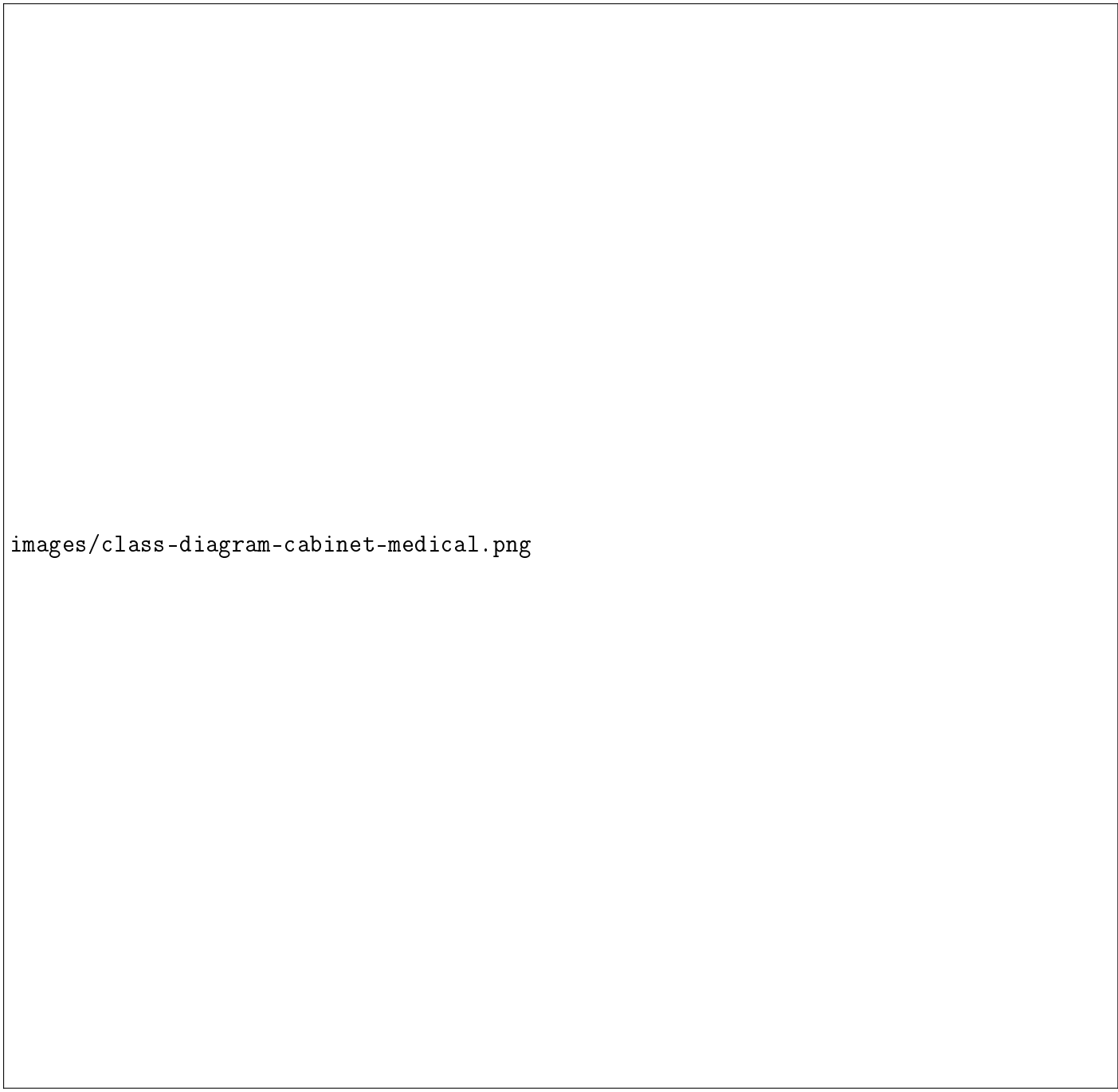

Figure 2.2: Diagramme de classes global

2.4 Mise en œuvre

Dans cette partie nous allons présenter le Product backlog, la planification des sprints, le diagramme de Gantt et l'architecture du système.

2.4.1 Product backlog

Comme nous l'avons indiqué dans le premier chapitre, le backlog produit est un élément essentiel qui décrit les besoins et les fonctionnalités attendues, classés par ordre de priorité. Chaque élément est détaillé sous forme de user story. Le tableau suivant présente le backlog produit de notre solution de gestion de cabinet médical, contenant les champs suivants :

- **Fonctionnalités :** Il s'agit d'un résumé bref de l'histoire utilisateur
- **User Stories :** Caractérise la fonctionnalité désirée par l'utilisateur
- **Priorité :** Caractérise l'importance de la fonctionnalité
- **Estimation :** Une évaluation du temps nécessaire pour réaliser chaque user story

Table 2.1: Backlog Produit

Fonctionnalités	User Stories	Priorités	Estimation
Gestion d'Authentification	En tant que Médecin, je veux m'authentifier	Élevée	2 jours
	En tant qu'Assistant, je veux m'authentifier	Élevée	2 jours
	En tant que Patient, je veux m'inscrire avec vérification email	Élevée	4 jours
	En tant que Patient, je veux m'authentifier	Élevée	2 jours
Gérer Assistants (Médecin)	En tant que Médecin, je veux créer un assistant	Élevée	3 jours
	En tant que Médecin, je veux consulter la liste des assistants	Moyenne	2 jours
	En tant que Médecin, je veux modifier un assistant	Moyenne	2 jours

	En tant que Médecin, je veux activer/désactiver un assistant	Moyenne	2 jours
	En tant que Médecin, je veux supprimer définitivement un assistant	Basse	1 jour
Gérer Patients	En tant que Patient, je veux m'inscrire via /register (page publique)	Élevée	3 jours
	En tant que Médecin/Assistant, je veux consulter tous les patients	Moyenne	2 jours
	En tant qu'Assistant, je veux consulter mes patients liés (via RDV créés)	Moyenne	2 jours
	En tant que Médecin/Assistant, je veux modifier un patient	Moyenne	2 jours
	En tant que Médecin, je veux supprimer un patient	Basse	1 jour
Gérer Rendez-vous	En tant qu'Assistant, je veux créer un rendez-vous via POST /api/rendezvous/assistants/{assistantId}/patients/{patientId}/rdv	Élevée	4 jours
	En tant que Patient, je veux prendre un rendez-vous en ligne avec calendrier et créneaux disponibles	Élevée	5 jours
	En tant qu'Assistant/Patient, je veux consulter les créneaux disponibles via GET /api/rendezvous/medecin/{medecinId}/slots-disponibles	Élevée	3 jours
	En tant qu'Assistant, je veux modifier un rendez-vous via PATCH	Moyenne	2 jours
	En tant qu'Assistant, je veux annuler un rendez-vous via DELETE	Moyenne	2 jours
	En tant que Médecin, je veux consulter mon calendrier complet de RDV (créés par assistants et patients)	Élevée	3 jours

	En tant que Patient, je veux consulter mes rendez-vous avec détails (date, médecin, motif, statut)	Moyenne	2 jours
	En tant que Médecin, je veux voir les rendez-vous créés par chaque assistant et chaque patient	Basse	1 jour
Gérer Dossiers Médicaux	En tant que Médecin, je veux consulter tous les dossiers via GET /api/dossiers	Moyenne	2 jours
	En tant que Médecin, je veux modifier un dossier existant via PUT /api/dossiers/{id}	Moyenne	3 jours
	En tant que Médecin, je veux ajouter des documents (PDF, images) via POST /api/dossiers/{id}/files	Élevée	4 jours
	En tant que Médecin, je veux télécharger des documents médicaux associés à un dossier	Moyenne	2 jours
	En tant que Patient, je veux consulter mes dossiers médicaux via GET /api/dossiers/patient/{patientId}	Élevée	3 jours
	En tant que Patient, je veux télécharger mes documents médicaux via GET /api/dossiers/{dossierId}/files/{docId}	Moyenne	2 jours
Gérer Factures	En tant que Médecin, je veux créer une facture pour n'importe quel patient	Élevée	3 jours
	En tant qu'Assistant, je veux créer une facture UNIQUEMENT pour mes patients liés (vérification automatique)	Élevée	4 jours
	En tant que Médecin/Assistant, je veux enregistrer un paiement via PATCH /api/factures/{id}/payer	Élevée	3 jours
	En tant que Médecin, je veux supprimer une facture via DELETE /api/factures/{id} (médecin uniquement)	Basse	1 jour
	En tant que Médecin, je veux générer un rapport financier (réservé au médecin)	Moyenne	4 jours

	En tant que Médecin/Assistant, je veux consulter les factures (filtrage automatique pour assistant)	Moyenne	2 jours
	En tant que Patient, je veux consulter mes factures	Basse	2 jours
Chatbot IA	En tant que Patient, je veux poser une question au chatbot	Élevée	5 jours
Gestion Notifications	En tant qu'Utilisateur, je veux consulter mes notifications	Moyenne	2 jours
	En tant qu'Utilisateur, je veux marquer mes notifications comme lues	Basse	1 jour
	En tant qu'Utilisateur, je veux configurer mes préférences de notification	Basse	2 jours
Tableau de Bord	En tant que Patient, je veux consulter mon tableau de bord avec prochains RDV et accès rapides	Moyenne	3 jours
	En tant que Médecin, je veux consulter mon tableau de bord avec statistiques (patients, RDV, factures)	Élevée	4 jours
	En tant qu'Assistant, je veux consulter mon tableau de bord avec mes statistiques personnelles	Moyenne	3 jours
Gestion Profil et Paramètres	En tant que Patient, je veux gérer mon profil (nom, prénom, email, téléphone, date naissance, adresse)	Moyenne	2 jours
	En tant que Médecin, je veux gérer mon profil (nom, prénom, email, téléphone, spécialité, description)	Moyenne	2 jours
	En tant qu'Assistant, je veux gérer mon profil (nom, prénom, email, téléphone)	Moyenne	2 jours
	En tant qu'Utilisateur, je veux gérer mes paramètres (préférences notifications, email personnalisé)	Basse	2 jours
	En tant qu'Utilisateur, je veux changer mon mot de passe via /api/account/change-password	Moyenne	2 jours
	En tant qu'Utilisateur, je veux supprimer mon compte via /api/account	Basse	1 jour

2.4.2 Planification des sprints

La réunion de planification du sprint est une des étapes les plus importantes d'un projet Scrum. L'objectif est de préparer le planning de travail et de choisir les tâches à inclure dans chaque sprint. Après analyse du backlog produit, nous avons décidé de diviser notre projet en trois sprints principaux, précédés d'un sprint 0 d'initialisation :

- **Sprint 0 (Initialisation - 1 semaine) :** Configuration de l'environnement de développement, setup Spring Boot et Next.js, configuration de la base de données, mise en place de la sécurité JWT et de l'architecture système de base.
- **Sprint 1 (Espace Médecin - 2 semaines) :** Authentification médecin, gestion complète des assistants (CRUD + activation), gestion du calendrier des rendez-vous, création et modification de dossiers médicaux avec upload de documents, gestion des factures et enregistrement des paiements, génération de rapports financiers.
- **Sprint 2 (Espace Assistant - 2 semaines) :** Authentification assistant, consultation et modification de patients (la création se fait via inscription publique /register), création et modification de rendez-vous avec vérification des créneaux disponibles, création de factures avec contrôle d'accès (patients liés uniquement), gestion des notifications et paramètres.
- **Sprint 3 (Espace Patient - 2 semaines) :** Inscription patient avec vérification email (code à 6 chiffres), authentification patient, consultation des rendez-vous, consultation des dossiers médicaux et téléchargement de documents, chatbot IA pour questions médicales avec intégration OpenAI, gestion des notifications et préférences.

2.4.3 Diagramme de Gantt

Le diagramme de Gantt, fréquemment utilisé en gestion de projet, est l'un des outils les plus efficaces pour visualiser l'état d'avancement des diverses tâches du projet. Il fournit une vue d'ensemble du planning et de la séquence des différentes activités sur une période de 7 semaines (Sprint 0 + 3 sprints de 2 semaines chacun).

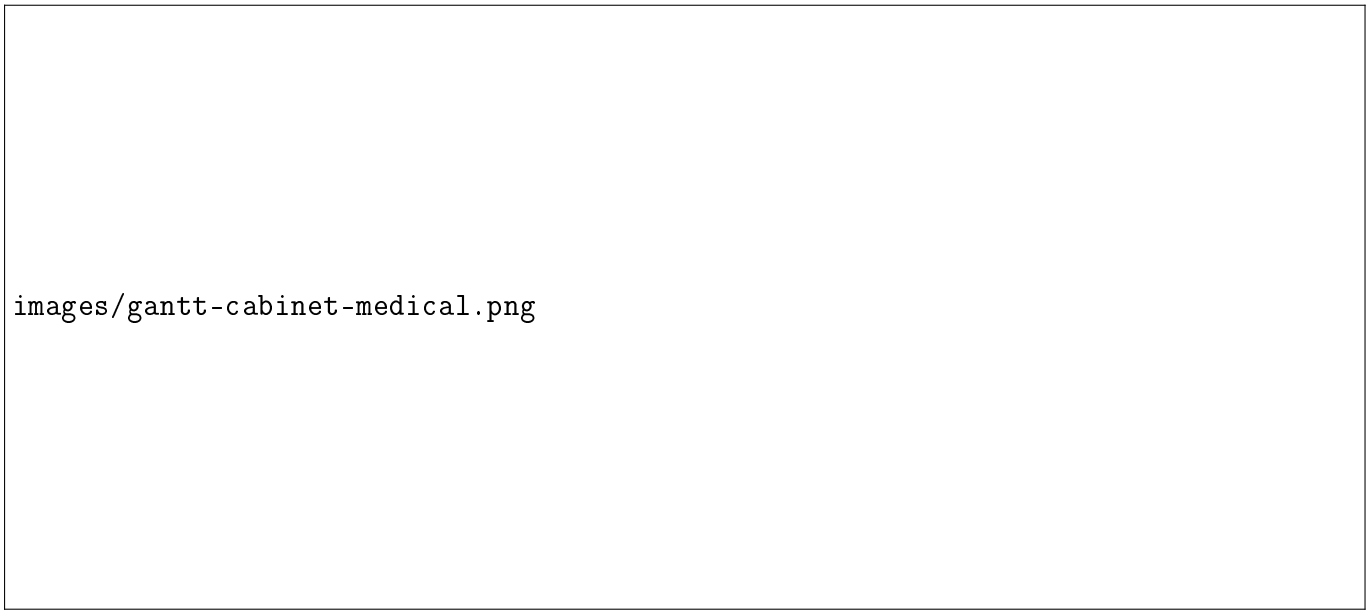

Figure 2.3: Diagramme de Gantt

2.4.4 L'architecture du système

Pour notre application de gestion de cabinet médical, nous avons choisi une architecture 3-tiers, également appelée architecture à trois niveaux. Cette architecture logicielle bien établie organise l'application en trois niveaux informatiques, logiques et physiques : une couche de présentation (Frontend Next.js), une couche applicative qui traite la logique métier (Backend Spring Boot), et une couche de données qui stocke et gère les informations médicales.

Figure 2.4: Architecture 3-Tiers du Cabinet Médical

2.4.4.1 Couche Présentation (Frontend)

La couche présentation est développée avec Next.js 14, un framework React moderne. Elle se compose de :

- **Pages et Composants React :** Organisation modulaire avec App Router, pages dédiées par espace (médecin, assistant, patient), composants réutilisables pour les formulaires et tableaux.
- **Gestion d'État :** Utilisation du Context API pour gérer l'authentification et les informations utilisateur, hooks personnalisés pour l'accès aux données.
- **Communication API :** Appels REST vers le backend Spring Boot, gestion des tokens JWT dans les headers Authorization.
- **Interface Responsive :** Design adaptatif avec Tailwind CSS, compatible desktop, tablette et mobile.

2.4.4.2 Couche Métier (Backend)

La couche métier est développée avec Spring Boot 3.5.7 (Java 21) et suit l'architecture MVC enrichie :

- **Controllers REST :** Exposition d'endpoints API organisés par domaine fonctionnel (UserController, PatientController, MedecinController, AssistantController, RendezVousController DossierPatientController, FactureController, ChatbotController, NotificationController).
- **Services Métier :** Implémentation de la logique métier complexe (validation des données, calculs, orchestration des opérations).
- **Repositories JPA :** Accès aux données via Spring Data JPA avec requêtes personnalisées.
- **Sécurité :** Spring Security avec JWT, authentification basée sur tokens, autorisation par rôles (ROLE_PATIENT, ROLE_MEDECIN, ROLE_ASSISTANT).
- **Services Transversaux :** EmailService (notifications SMTP), NotificationService (gestion centralisée des notifications), FileStorageService (upload/download de documents), ChatbotService (intégration OpenAI).

2.4.4.3 Couche Données

La couche données repose sur une base de données relationnelle (PostgreSQL/MySQL) et un système de fichiers :

- **Base de Données :** Tables relationnelles (users, patients, medecins, assistants, rendez_vous, dossiers_patients, factures, paiements, notifications, documents), contraintes d'intégrité référentielle, indexation pour optimisation des performances.
- **Stockage Fichiers :** Système de fichiers local pour documents médicaux (ordonnances, analyses), organisation par dossier patient, métadonnées en base de données.

2.4.4.4 Architecture MVC

Pendant la réalisation de notre projet, nous avons utilisé l'architecture logicielle MVC (Model-View-Controller) qui décompose l'application en trois composants logiques principaux. Chacun de ces éléments a une fonction spécifique dans le développement de l'application. L'architecture MVC est largement utilisée comme pattern de développement web pour créer des projets extensibles et évolutifs. Nous présentons en détail les trois composants de l'architecture MVC appliqués à notre système :

- **Modèle (Entités JPA) :** Ce composant correspond à toutes les données relatives à la logique métier. Il contient les entités JPA (User, Patient, Medecin, Assistant, RendezVous, DossierPatient, Facture, Document, Notification) qui mappent les tables de la base de données. Ces entités communiquent avec la base de données via les Repositories pour sauvegarder, consulter ou modifier les données médicales.
- **Vue (Frontend Next.js) :** Ce composant représente la couche de présentation de l'application, responsable de fournir l'interface utilisateur (UI). Il se compose d'un ensemble de pages React (dashboard, rendez-vous, dossiers, factures, chatbot) et de composants réutilisables. Il n'intègre aucune logique métier complexe, ces traitements étant gérés par le composant contrôleur backend. La vue communique avec le backend via des appels API REST.
- **Contrôleur (Controllers + Services) :** Ce composant contient la logique métier et les algorithmes de traitement. Les Controllers REST exposent les endpoints API et délèguent le traitement aux Services. Par exemple, la vue soumet un formulaire de création de dossier médical au RendezVousController, qui valide les données

via le code métier dans DossierPatientService, puis demande au Repository JPA d'effectuer les modifications nécessaires dans la base de données. Le Service gère également les opérations transversales comme l'envoi de notifications et l'upload de documents.

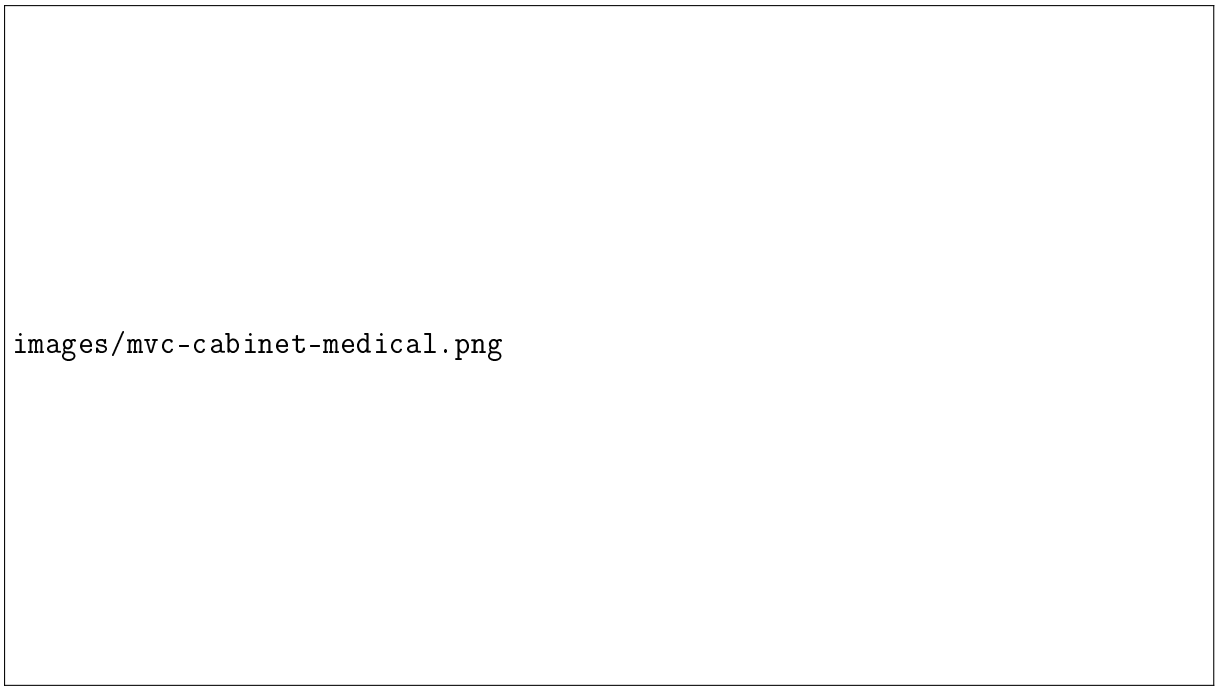

Figure 2.5: Architecture MVC du Cabinet Médical

Le principe de fonctionnement du framework MVC dans notre système de gestion de cabinet médical est le suivant :

1. L'utilisateur (Patient, Médecin ou Assistant) choisit une action à effectuer (ex: créer un rendez-vous) à travers l'interface utilisateur Next.js.
2. Le Controller backend reçoit la requête HTTP REST (ex: POST /api/rendezvous/assistants/5/patien contenant les données JSON du rendez-vous.
3. Le Controller (@RestController) valide la requête et appelle le Service approprié (RendezVousService) pour traiter la logique métier.
4. Le Service effectue les vérifications métier (disponibilité créneau, autorisations), interagit avec les Repositories JPA pour accéder aux données (vérifier patient, médecin, créneaux occupés).
5. Le Modèle (entités JPA) traite la demande en utilisant les données appropriées et effectue les opérations nécessaires (création du rendez-vous en base de données).
6. Une fois le traitement terminé, le Service peut déclencher des opérations secondaires (envoi notification au patient, email de confirmation).
7. Le Service renvoie le résultat au Controller sous forme de DTO (Data Transfer Object).

8. Le Controller encapsule la réponse dans un objet ResponseEntity avec le code HTTP approprié (201 Created, 200 OK, 400 Bad Request, etc.).
9. La Vue (Frontend Next.js) reçoit la réponse JSON, met à jour l'interface utilisateur et affiche un message de confirmation à l'utilisateur.

2.5 CONCLUSION

Ce chapitre a permis de spécifier de manière détaillée les besoins fonctionnels et non fonctionnels de la plateforme de gestion de cabinet médical intégrant l'intelligence artificielle. L'identification des quatre acteurs principaux (Patient, Médecin, Assistant, Chatbot IA) et de leurs cas d'utilisation respectifs a permis de modéliser le système à travers des diagrammes UML, tandis que la planification avec la méthodologie Scrum a structuré le développement en trois sprints cohérents précédés d'un sprint d'initialisation. L'architecture technique retenue, basée sur des technologies modernes et éprouvées (Spring Boot, Next.js, JWT, OpenAI), offre un socle solide pour la réalisation du projet.

Cette étude préliminaire constitue une base essentielle pour les phases de développement qui seront détaillées dans les chapitres suivants. Elle garantit l'adéquation entre la solution proposée et les besoins exprimés par les différents acteurs du cabinet médical, tout en assurant la qualité, la sécurité (RGPD, secret médical) et la maintenabilité de l'application. La planification détaillée des sprints et la définition claire des livrables permettent d'anticiper les risques et d'assurer le respect des délais du projet académique.

Chapter

3

CHAPITRE 3 : Sprint 1 - Gestion de l'espace Médecin

Contents

3.1 INTRODUCTION 36

3.2 Backlog du sprint 1 36

3.3 Spécifications fonctionnelles 40

3.3.1 Diagramme de cas d'utilisation du sprint 1 40

3.3.2 Descriptions textuelles 41

3.4 Conception 78

3.4.1 Diagrammes de séquence 78

3.4.2 Diagramme de classes du sprint 1 85

3.5 Réalisation 87

3.5.1 Authentification 87

3.5.2 Tableau de bord médecin 88

3.5.3 Gestion des patients 88

3.5.4 Gestion des rendez-vous 90

3.5.5 Gestion des assistants 93

3.5.6 Gestion des dossiers médicaux 94

3.5.7 Gestion des factures 97

3.5.8 Rapports financiers 100

3.5.9 Gestion du profil et paramètres 101

3.6 CONCLUSION 103

3.1 INTRODUCTION

Après avoir identifié les besoins fonctionnels et défini les fonctionnalités nécessaires dans le Product Backlog, ce chapitre se concentre sur la description du premier sprint, intitulé "Gestion de l'espace Médecin". Nous allons commencer par la spécification du backlog du sprint 1, puis la conception avec diagrammes UML, et enfin l'illustration des fonctionnalités développées à travers des captures d'écran. Ce sprint constitue le cœur du système car il permet au médecin de gérer l'ensemble de son cabinet : assistants, patients, rendez-vous, dossiers médicaux et factures.

3.2 Backlog du sprint 1

Le tableau **??** présente l'ensemble des user stories sélectionnées pour le sprint 1, focalisées sur l'espace médecin.

Table 3.1: Backlog du sprint 1 - Espace Médecin

Fonctionnalités	User Stories	Priorités	Estimation
Gestion d'Authentification	En tant que médecin, je veux m'authentifier pour accéder à mon espace sécurisé	Élevée	3 jours
	En tant que médecin, je veux me déconnecter pour quitter mon session en toute sécurité	Moyenne	1 jour
Gérer les Assistants	En tant que médecin, je veux créer un compte assistant avec email, nom et mot de passe	Élevée	4 jours
	En tant que médecin, je veux consulter la liste de tous mes assistants avec leurs informations	Moyenne	2 jours
	En tant que médecin, je veux modifier les informations d'un assistant (nom, email)	Moyenne	2 jours
	En tant que médecin, je veux activer ou désactiver un assistant avec envoi d'email de notification	Moyenne	3 jours

Fonctionnalités	User Stories	Priorités	Estimation
	En tant que médecin, je veux supprimer définitivement un assistant via DELETE /api/assistants/supprimer/{id}	Faible	1 jour
Gérer les Dossiers Médicaux	En tant que médecin, je veux consulter l'historique complet des dossiers médicaux d'un patient via GET /api/dossiers/patient/{id}	Élevée	3 jours
	En tant que médecin, je veux modifier un dossier médical existant via PUT /api/dossiers/{id}	Moyenne	2 jours
	En tant que médecin, je veux uploader des documents médicaux (PDF, images) via POST /api/dossiers/{id}/files	Élevée	4 jours
	En tant que médecin, je veux télécharger les documents d'un dossier via GET /api/dossiers/{dossierId}/files/{docId}	Moyenne	2 jours
Gérer les Factures	En tant que médecin, je veux créer une facture pour n'importe quel patient via POST /api/factures	Élevée	4 jours
	En tant que médecin, je veux consulter toutes les factures avec filtres (payée/impayée, période) via GET /api/factures	Moyenne	3 jours
	En tant que médecin, je veux enregistrer un paiement via PATCH /api/factures/{id}/payer avec mode de paiement (CB, espèces, chèque, virement)	Élevée	3 jours
	En tant que médecin, je veux supprimer une facture via DELETE /api/factures/{id} (réservé au médecin uniquement)	Faible	1 jour
	En tant que médecin, je veux générer des rapports financiers via GET /api/factures/rapports avec statistiques détaillées	Moyenne	4 jours

Fonctionnalités	User Stories	Priorités	Estimation
Gérer les Patients	En tant que médecin, je veux consulter la liste de tous mes patients avec leurs informations via GET /api/patients	Élevée	3 jours
	En tant que médecin, je veux modifier les informations d'un patient (nom, adresse, téléphone) via PUT /api/patients/{id}	Moyenne	2 jours
	En tant que médecin, je veux rechercher un patient par nom ou téléphone pour accéder rapidement à son dossier	Moyenne	2 jours
	En tant que médecin, je veux consulter le profil complet d'un patient avec historique RDV et dossiers	Élevée	3 jours
Gérer les Rendez-vous	En tant que médecin, je veux créer un nouveau rendez-vous pour un patient via POST /api/rendezvous	Élevée	4 jours
	En tant que médecin, je veux consulter tous mes rendez-vous avec filtres (date, statut, patient) via GET /api/rendezvous	Élevée	3 jours
	En tant que médecin, je veux modifier un rendez-vous existant (date, heure, motif) via PUT /api/rendezvous/{id}	Moyenne	2 jours
	En tant que médecin, je veux annuler un rendez-vous avec notification automatique au patient via PATCH /api/rendezvous/{id}/annuler	Moyenne	3 jours
	En tant que médecin, je veux visualiser mon calendrier de rendez-vous avec vue hebdomadaire/mensuelle	Moyenne	3 jours
	En tant que médecin, je veux consulter les détails d'un rendez-vous (patient, motif, heure, statut)	Moyenne	2 jours

Fonctionnalités	User Stories	Priorités	Estimation
Générer des Rapports	En tant que médecin, je veux générer des rapports financiers détaillés via GET /api/factures/rapports avec statistiques (CA, impayés)	Élevée	4 jours
	En tant que médecin, je veux exporter les rapports en PDF ou Excel pour archivage ou présentation	Moyenne	3 jours
Tableau de Bord	En tant que médecin, je veux voir mes statistiques principales (RDV jour, patients total, CA mois) sur le dashboard	Élevée	4 jours
	En tant que médecin, je veux visualiser mes graphiques de performance (évolution CA, taux paiement)	Moyenne	3 jours
	En tant que médecin, je veux voir mes prochains rendez-vous du jour directement sur le dashboard	Élevée	2 jours
	En tant que médecin, je veux accéder rapidement aux factures impayées depuis le dashboard	Moyenne	2 jours
Gestion Profil et Paramètres	En tant que médecin, je veux consulter et modifier mon profil (nom, email, téléphone) via GET/PUT /api/profil	Moyenne	3 jours
	En tant que médecin, je veux modifier mon mot de passe avec vérification ancien mot de passe via PATCH /api/profil/password	Élevée	2 jours
	En tant que médecin, je veux configurer mes paramètres de cabinet (horaires, durée RDV) via GET/PUT /api/parametres	Moyenne	3 jours
	En tant que médecin, je veux gérer mes préférences de notification (email, push) dans les paramètres	Faible	2 jours

Chaque user story représente une fonctionnalité que le médecin souhaite pouvoir réaliser dans le système. Les priorités sont classées en élevée, moyenne ou faible, reflétant l'importance relative de chaque user story. L'estimation en jours correspond au temps de développement prévu incluant le backend (API REST Spring Boot), le frontend (Next.js) et les tests.

3.3 Spécifications fonctionnelles

Dans cette partie, nous présentons le diagramme de cas d'utilisation du sprint 1 ainsi que les descriptions textuelles détaillées des principaux cas d'utilisation.

3.3.1 Diagramme de cas d'utilisation du sprint 1

Le diagramme de cas d'utilisation du sprint 1, présenté dans la figure 3.1, illustre les besoins fonctionnels sous la forme d'interactions entre le médecin et le système.

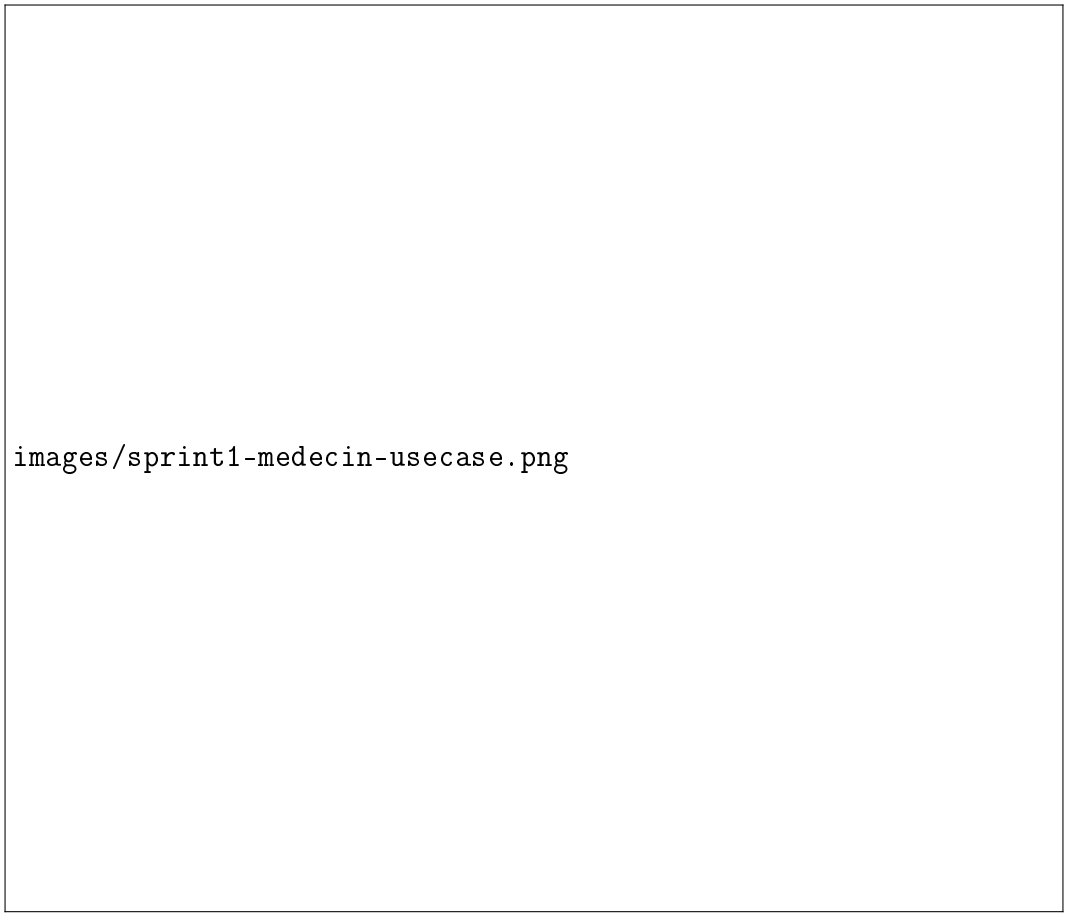

Figure 3.1: Diagramme de cas d'utilisation du sprint 1 - Espace Médecin

3.3.2 Descriptions textuelles

L'objectif de cette activité est de décrire textuellement les scénarios des cas d'utilisation. Il faut préciser comment chaque scénario commence, comment il se termine et comment le médecin interagit avec l'application web.

Description textuelle du cas d'utilisation "S'authentifier" :

Le tableau **??** présente la description textuelle du cas d'utilisation "S'authentifier". Ce scénario commence lorsque le médecin ouvre l'application et accède à l'écran de connexion.

Table 3.2: Description textuelle du cas d'utilisation "S'authentifier - Médecin"

Cas d'utilisation	S'authentifier
Acteur	Médecin
Précondition	1. Le système est en service. 2. Le médecin possède un compte actif avec username et mot de passe.
Post-condition	1. Le médecin est authentifié avec un token JWT contenant son rôle "MEDECIN". 2. Le système affiche le tableau de bord médecin avec accès à toutes ses fonctionnalités.

Cas d'utilisation	S'authentifier
Scénario principal	1. Le médecin accède à la page de connexion (/login). 2. Le système affiche le formulaire d'authentification. 3. Le médecin saisit son username et son mot de passe. 4. Le médecin clique sur "Se connecter". 5. Le système vérifie les identifiants via l'API POST /api/auth/login avec LoginDTO (username, password). 6. Le backend utilise UsernamePasswordAuthenticationToken et findByUsername() pour authentifier. 7. Le système génère un token JWT avec le rôle "MEDECIN". 8. Le système stocke le token dans le localStorage du navigateur. 9. Le système redirige vers /dashboard (tableau de bord médecin).
Scénario alternatif	1. Le médecin saisit des données incomplètes (username ou mot de passe vide) → affichage d'un message d'erreur "Tous les champs sont obligatoires". 2. Le médecin saisit un username inexistant → affichage d'un message d'erreur "Username ou mot de passe incorrect". 3. Le médecin saisit un mot de passe incorrect → affichage d'un message d'erreur "Invalid username or password" (message du backend). 4. Le compte est désactivé (active=false) → affichage d'un message d'erreur "Compte désactivé, contactez l'administrateur".

Description textuelle du cas d'utilisation "Créer un Assistant" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Créer un Assistant". Ce scénario permet au médecin d'ajouter un nouvel assistant à son équipe.

Table 3.3: Description textuelle du cas d'utilisation "Créer un Assistant"

Cas d'utilisation	Créer un Assistant
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Le médecin a accès à la section "Gestion des Assistants".
Post-condition	1. Un nouveau compte assistant est créé dans la base de données avec le statut "actif" (active=true). 2. Le mot de passe est hashé avec BCrypt avant stockage en base. 3. Un email de notification d'activation est envoyé automatiquement via NotificationEmailService. 4. L'assistant peut se connecter avec ses identifiants.

Cas d'utilisation	Créer un Assistant
Scénario principal	1. Le médecin accède à la section "Gestion des Assistants" (/dashboard/assistants). 2. Le médecin clique sur "Ajouter un Assistant". 3. Le système affiche le formulaire de création avec les champs : nom complet, email, mot de passe, confirmation mot de passe. 4. Le médecin remplit tous les champs obligatoires. 5. Le médecin clique sur "Créer". 6. Le système valide les données saisies (format email, force mot de passe, correspondance des mots de passe). 7. Le système vérifie que l'email n'est pas déjà utilisé via l'API /api/assistants/check-email. 8. Le système envoie une requête POST /api/assistants avec les données. 9. Le backend hash le mot de passe avec BCrypt et enregistre l'assistant en base. 10. Le système affiche un message de succès "Assistant créé avec succès". 11. Le système rafraîchit la liste des assistants.

Cas d'utilisation	Créer un Assistant
Scénario alternatif	1. Le médecin laisse des champs obligatoires vides → affichage d'erreurs de validation sous les champs concernés. 2. Le format de l'email est invalide → affichage d'un message "Email invalide". 3. Les mots de passe ne correspondent pas → affichage d'un message "Les mots de passe ne correspondent pas". 4. Le mot de passe est trop faible (moins de 8 caractères) → affichage d'un message "Le mot de passe doit contenir au moins 8 caractères". 5. L'email est déjà utilisé → affichage d'un message "Un compte avec cet email existe déjà". 6. Erreur serveur lors de la création → affichage d'un message "Erreur lors de la création, veuillez réessayer".

Description textuelle du cas d'utilisation "Créer une Facture" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Créer une Facture". Ce scénario permet au médecin d'émettre une facture pour des actes médicaux.

Table 3.4: Description textuelle du cas d'utilisation "Créer une Facture"

Cas d'utilisation	Créer une Facture
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Un patient existe dans le système. 3. Le médecin a accès à la section "Factures".

Cas d'utilisation	Créer une Facture
Post-condition	1. Une nouvelle facture est créée avec le statut "IMPAYEE". 2. La facture contient : patient, montant total, description des actes, date d'émission. 3. Un numéro de facture unique est généré automatiquement. 4. La facture est visible dans la liste des factures impayées.

Cas d'utilisation	Créer une Facture
Scénario principal	1. Le médecin accède à la section "Factures" (/dashboard/factures). 2. Le médecin clique sur "Créer une Facture". 3. Le système affiche le formulaire de création avec les champs : sélection patient, montant, description des actes. 4. Le médecin sélectionne le patient dans la liste déroulante. 5. Le médecin saisit le montant et la description (ex: "Consultation + Ordonnance"). 6. Le médecin clique sur "Créer". 7. Le système valide les données (montant > 0, patient sélectionné, description non vide). 8. Le système génère un numéro de facture unique (format: FAC-YYYY-XXXX). 9. Le système envoie une requête POST /api/factures avec les données. 10. Le backend enregistre la facture avec statut "IMPAYEE" et date d'émission. 11. Le système affiche un message de succès "Facture créée avec succès". 12. Le système rafraîchit la liste des factures.

Cas d'utilisation	Créer une Facture
Scénario alternatif	1. Aucun patient sélectionné → affichage d'un message "Veuillez sélectionner un patient". 2. Le montant est inférieur ou égal à 0 → affichage d'un message "Le montant doit être supérieur à 0". 3. La description est vide → affichage d'un message "La description est obligatoire". 4. Erreur serveur lors de la création → affichage d'un message "Erreur lors de la création, veuillez réessayer".

Description textuelle du cas d'utilisation "Enregistrer un Paiement" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Enregistrer un Paiement". Ce scénario permet au médecin de marquer une facture comme payée.

Table 3.5: Description textuelle du cas d'utilisation "Enregistrer un Paiement"

Cas d'utilisation	Enregistrer un Paiement
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Une facture existe avec le statut "IMPAYEE". 3. Le médecin a accès à la section "Factures".

Cas d'utilisation	Enregistrer un Paiement
Post-condition	1. Le statut de la facture passe à "PAYEE". 2. Le mode de paiement est enregistré (CB, Espèces, Chèque, Virement). 3. La date de paiement est enregistrée automatiquement. 4. La facture disparaît de la liste des impayées et apparaît dans les factures payées. 5. Les statistiques financières sont mises à jour automatiquement.
Scénario principal	1. Le médecin accède à la section "Factures" et filtre sur "Impayées". 2. Le médecin sélectionne une facture impayée dans la liste. 3. Le médecin clique sur "Enregistrer Paiement". 4. Le système affiche un formulaire modal avec sélection du mode de paiement (CB, Espèces, Chèque, Virement). 5. Le médecin sélectionne le mode de paiement. 6. Le médecin clique sur "Confirmer". 7. Le système envoie une requête PATCH /api/factures/{id}/payer avec le mode de paiement. 8. Le backend met à jour la facture : statut = "PAYEE", datePaiement = NOW(), modePaiement. 9. Le système affiche un message de succès "Paiement enregistré avec succès". 10. Le système rafraîchit la liste des factures. 11. Le système met à jour les statistiques du dashboard (graphiques de revenus).

Cas d'utilisation	Enregistrer un Paiement
Scénario alternatif	1. Aucun mode de paiement sélectionné → affichage d'un message "Veuillez sélectionner un mode de paiement". 2. La facture est déjà payée → affichage d'un message "Cette facture est déjà payée". 3. Erreur serveur lors de l'enregistrement → affichage d'un message "Erreur lors de l'enregistrement, veuillez réessayer".

Description textuelle du cas d'utilisation "Modifier un Assistant" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Modifier un Assistant". Ce scénario permet au médecin de mettre à jour les informations d'un assistant existant.

Table 3.6: Description textuelle du cas d'utilisation "Modifier un Assistant"

Cas d'utilisation	Modifier un Assistant
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Un assistant existe dans le système. 3. Le médecin a accès à la section "Gestion des Assistants".
Post-condition	1. Les informations de l'assistant sont mises à jour dans la base de données. 2. Les modifications sont visibles dans la liste des assistants. 3. Si le mot de passe est modifié, il est hashé avec BCrypt avant stockage.

Cas d'utilisation	Modifier un Assistant
Scénario principal	1. Le médecin accède à la section "Gestion des Assistants" (/dashboard/assistants). 2. Le médecin sélectionne un assistant dans la liste. 3. Le médecin clique sur "Modifier" (icône crayon). 4. Le système affiche un formulaire pré-rempli avec les informations actuelles (nom, email, username). 5. Le médecin modifie les champs souhaités (nom, email, ou mot de passe). 6. Le médecin clique sur "Enregistrer". 7. Le système valide les données (format email, force mot de passe si modifié). 8. Le système envoie une requête PUT /api/assistants/modifier/{id} avec les données mises à jour. 9. Le backend met à jour l'assistant en base via AssistantService.update(). 10. Le système affiche un message de succès "Assistant modifié avec succès". 11. Le système rafraîchit la liste des assistants.

Cas d'utilisation	Modifier un Assistant
Scénario alternatif	1. Le médecin laisse des champs obligatoires vides → affichage d'erreurs de validation. 2. Le format de l'email est invalide → affichage d'un message "Email invalide". 3. Le nouveau mot de passe est trop faible → affichage d'un message "Le mot de passe doit contenir au moins 8 caractères". 4. L'email est déjà utilisé par un autre utilisateur → affichage d'un message "Un compte avec cet email existe déjà". 5. Erreur serveur lors de la modification → affichage d'un message "Erreur lors de la modification, veuillez réessayer".

Description textuelle du cas d'utilisation "Activer/Désactiver un Assistant" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Activer/Désactiver un Assistant". Ce scénario permet au médecin de bloquer temporairement l'accès d'un assistant sans supprimer ses données.

Table 3.7: Description textuelle du cas d'utilisation "Activer/Désactiver un Assistant"

Cas d'utilisation	Activer/Désactiver un Assistant
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Un assistant existe dans le système. 3. Le médecin a accès à la section "Gestion des Assistants".

Cas d'utilisation	Activer/Désactiver un Assistant
Post-condition	1. Le statut de l'assistant (active) est basculé (true ↔ false). 2. Un email de notification est envoyé automatiquement via NotificationEmailService. 3. Si désactivé, l'assistant ne peut plus se connecter au système. 4. Si activé, l'assistant retrouve l'accès au système. 5. Le badge visuel (Actif/Inactif) est mis à jour dans l'interface.
Scénario principal	1. Le médecin accède à la section "Gestion des Assistants" (/dashboard/assistants). 2. Le médecin sélectionne un assistant dans la liste. 3. Le médecin clique sur le bouton "Activer" ou "Désactiver" selon l'état actuel. 4. Le système affiche une boîte de dialogue de confirmation "Voulez-vous vraiment activer/désactiver cet assistant ?". 5. Le médecin confirme l'action. 6. Le système envoie une requête PATCH /api/assistants/activer/{id} (toggle automatique). 7. Le backend bascule le champ active (true ↔ false) via AssistantService.toggleActivation(). 8. Le backend envoie un email de notification à l'assistant via NotificationEmailService. 9. Le système affiche un message de succès "Assistant activé/désactivé avec succès". 10. Le système rafraîchit la liste avec le nouveau statut visible.

Cas d'utilisation	Activer/Désactiver un Assistant
Scénario alternatif	1. Le médecin annule la confirmation → aucune modification n'est effectuée. 2. L'assistant est déjà dans l'état souhaité → affichage d'un message informatif. 3. Erreur serveur lors du changement de statut → affichage d'un message "Erreur lors de l'opération, veuillez réessayer". 4. Échec d'envoi de l'email de notification → le statut est quand même modifié, mais un avertissement est affiché.

Description textuelle du cas d'utilisation "Supprimer un Assistant" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Supprimer un Assistant". Ce scénario permet au médecin de supprimer définitivement un assistant du système.

Table 3.8: Description textuelle du cas d'utilisation "Supprimer un Assistant"

Cas d'utilisation	Supprimer un Assistant
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Un assistant existe dans le système. 3. Le médecin a accès à la section "Gestion des Assistants".

Cas d'utilisation	Supprimer un Assistant
Post-condition	1. L'assistant est supprimé définitivement de la base de données. 2. Toutes les références à cet assistant sont traitées (rendez-vous créés par lui conservés avec référence null ou ID). 3. L'assistant ne peut plus se connecter au système. 4. L'assistant disparaît de la liste des assistants.
Scénario principal	1. Le médecin accède à la section "Gestion des Assistants" (/dashboard/assistants). 2. Le médecin sélectionne un assistant dans la liste. 3. Le médecin clique sur "Supprimer" (icône poubelle). 4. Le système affiche une boîte de dialogue de confirmation "Êtes-vous sûr de vouloir supprimer définitivement cet assistant ? Cette action est irréversible.". 5. Le médecin confirme la suppression. 6. Le système envoie une requête DELETE /api/assistants/supprimer/{id}. 7. Le backend supprime l'assistant de la base de données via AssistantService.delete(). 8. Le système affiche un message de succès "Assistant supprimé avec succès". 9. Le système rafraîchit la liste des assistants.

Cas d'utilisation	Supprimer un Assistant
Scénario alternatif	1. Le médecin annule la confirmation → aucune suppression n'est effectuée. 2. L'assistant a des rendez-vous actifs associés → affichage d'un avertissement "Cet assistant a créé X rendez-vous. Les rendez-vous seront conservés mais sans référence à l'assistant.". 3. Erreur serveur lors de la suppression → affichage d'un message "Erreur lors de la suppression, veuillez réessayer".

Description textuelle du cas d'utilisation "Modifier un Dossier Médical" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Modifier un Dossier Médical". Ce scénario permet au médecin de mettre à jour un dossier médical existant.

Table 3.9: Description textuelle du cas d'utilisation "Modifier un Dossier Médical"

Cas d'utilisation	Modifier un Dossier Médical
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Un dossier médical existe dans le système. 3. Le médecin a accès à la section "Dossiers Médicaux".

Cas d'utilisation	Modifier un Dossier Médical
Post-condition	1. Les informations du dossier médical sont mises à jour dans la base de données. 2. La date de dernière modification est enregistrée automatiquement. 3. Les documents existants sont conservés, les nouveaux documents sont ajoutés. 4. Les modifications sont visibles dans l'historique des dossiers du patient.

Cas d'utilisation	Modifier un Dossier Médical
Scénario principal	1. Le médecin accède à la section "Dossiers Médicaux" (/dashboard/dossiers). 2. Le médecin sélectionne un dossier médical dans la liste. 3. Le médecin clique sur "Modifier". 4. Le système affiche le formulaire pré-rempli avec les informations actuelles (diagnostic, traitement, symptômes, observations, recommandations). 5. Le médecin modifie les champs souhaités (diagnostic, traitement, observations, etc.). 6. Le médecin peut ajouter de nouveaux documents médicaux. 7. Le médecin clique sur "Enregistrer". 8. Le système valide les données (champs obligatoires : diagnostic, traitement). 9. Le système envoie une requête PUT /api/dossiers/{id} avec DossierPatientDTO mis à jour. 10. Le backend met à jour le dossier via DossierPatientService.update(). 11. Les nouveaux fichiers sont uploadés via POST /api/dossiers/{id}/files. 12. Le système affiche un message de succès "Dossier médical modifié avec succès". 13. Le système rafraîchit l'affichage du dossier.

Cas d'utilisation	Modifier un Dossier Médical
Scénario alternatif	1. Le médecin laisse des champs obligatoires vides → affichage d'erreurs de validation. 2. Le fichier uploadé est trop volumineux (> 10 MB) → affichage d'un message "Fichier trop volumineux". 3. Le format de fichier est non autorisé → affichage d'un message "Format non supporté". 4. Erreur serveur lors de la modification → affichage d'un message "Erreur lors de la modification, veuillez réessayer".

Description textuelle du cas d'utilisation "Supprimer une Facture" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Supprimer une Facture". Ce scénario permet au médecin de supprimer définitivement une facture (action réservée au médecin uniquement).

Table 3.10: Description textuelle du cas d'utilisation "Supprimer une Facture"

Cas d'utilisation	Supprimer une Facture
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN" (vérification @PreAuthorize). 2. Une facture existe dans le système. 3. Le médecin a accès à la section "Factures".

Cas d'utilisation	Supprimer une Facture
Post-condition	1. La facture est supprimée définitivement de la base de données. 2. La facture disparaît de toutes les listes (payées/impayées). 3. Les statistiques financières sont recalculées automatiquement.
Scénario principal	1. Le médecin accède à la section "Factures" (/dashboard/factures). 2. Le médecin sélectionne une facture dans la liste. 3. Le médecin clique sur "Supprimer" (icône poubelle). 4. Le système affiche une boîte de dialogue de confirmation "Êtes-vous sûr de vouloir supprimer cette facture ? Cette action est irréversible.". 5. Le médecin confirme la suppression. 6. Le système envoie une requête DELETE /api/factures/{id}. 7. Le backend vérifie @PreAuthorize("hasRole('MEDECIN')") - seul le médecin peut supprimer. 8. Le backend supprime la facture via FactureService.delete(). 9. Le système affiche un message de succès "Facture supprimée avec succès". 10. Le système rafraîchit la liste des factures et met à jour les statistiques.

Cas d'utilisation	Supprimer une Facture
Scénario alternatif	1. Le médecin annule la confirmation → aucune suppression n'est effectuée. 2. L'utilisateur n'est pas médecin → erreur 403 Forbidden (ne devrait pas arriver si l'interface masque le bouton). 3. La facture a déjà été supprimée → affichage d'un message "Facture introuvable". 4. Erreur serveur lors de la suppression → affichage d'un message "Erreur lors de la suppression, veuillez réessayer".

Description textuelle du cas d'utilisation "Consulter les Patients" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Consulter les Patients". Ce scénario permet au médecin de visualiser la liste complète de ses patients.

Table 3.11: Description textuelle du cas d'utilisation "Consulter les Patients"

Cas d'utilisation	Consulter les Patients
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Le médecin a accès à la section "Patients".
Post-condition	1. La liste complète des patients est affichée avec leurs informations essentielles. 2. Le médecin peut accéder au profil détaillé de chaque patient.

Cas d'utilisation	Consulter les Patients
Scénario principal	1. Le médecin accède à la section "Patients" (/dashboard/patients). 2. Le système envoie une requête GET /api/patients. 3. Le backend récupère tous les patients via PatientService.findAll(). 4. Le système affiche la liste avec : nom, téléphone, email, date d'inscription, nombre de RDV. 5. Le médecin peut rechercher un patient par nom ou téléphone. 6. Le médecin peut cliquer sur un patient pour voir le profil détaillé.
Scénario alternatif	1. Aucun patient n'existe → affichage d'un message "Aucun patient enregistré". 2. Erreur serveur → affichage d'un message "Erreur lors du chargement, veuillez réessayer".

Description textuelle du cas d'utilisation "Modifier un Patient" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Modifier un Patient". Ce scénario permet au médecin de mettre à jour les informations personnelles d'un patient.

Table 3.12: Description textuelle du cas d'utilisation "Modifier un Patient"

Cas d'utilisation	Modifier un Patient
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Un patient existe dans le système. 3. Le médecin a accès à la section "Patients".

Cas d'utilisation	Modifier un Patient
Post-condition	1. Les informations du patient sont mises à jour dans la base de données. 2. Les modifications sont visibles dans le profil du patient.
Scénario principal	1. Le médecin accède au profil d'un patient. 2. Le médecin clique sur "Modifier". 3. Le système affiche le formulaire pré-rempli avec les informations actuelles (nom, téléphone, adresse, date de naissance). 4. Le médecin modifie les champs souhaités. 5. Le médecin clique sur "Enregistrer". 6. Le système valide les données (format téléphone, date de naissance). 7. Le système envoie une requête PUT /api/patients/{id} avec PatientDTO mis à jour. 8. Le backend met à jour le patient via PatientService.update(). 9. Le système affiche un message de succès "Patient modifié avec succès". 10. Le système rafraîchit l'affichage du profil.
Scénario alternatif	1. Le format du téléphone est invalide → affichage d'un message "Format de téléphone invalide". 2. La date de naissance est future → affichage d'un message "Date de naissance invalide". 3. Erreur serveur → affichage d'un message "Erreur lors de la modification, veuillez réessayer".

Description textuelle du cas d'utilisation "Créer un Rendez-vous" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Créer un Rendez-vous". Ce scénario permet au médecin de planifier un nouveau rendez-vous pour un patient.

Table 3.13: Description textuelle du cas d'utilisation "Créer un Rendez-vous"

Cas d'utilisation	Créer un Rendez-vous
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Au moins un patient existe dans le système. 3. Le médecin a accès à la section "Rendez-vous".
Post-condition	1. Un nouveau rendez-vous est créé avec le statut "PLANIFIE". 2. Le rendez-vous est visible dans le calendrier du médecin. 3. Une notification est envoyée au patient (email).

Cas d'utilisation	Créer un Rendez-vous
Scénario principal	1. Le médecin accède à la section "Rendez-vous" (/dashboard/rendezvous). 2. Le médecin clique sur "Nouveau Rendez-vous". 3. Le système affiche le formulaire avec : sélection patient, date, heure, durée, motif. 4. Le médecin sélectionne un patient dans la liste déroulante. 5. Le médecin choisit une date et une heure disponible. 6. Le médecin saisit le motif de consultation. 7. Le médecin clique sur "Créer". 8. Le système vérifie la disponibilité du créneau. 9. Le système envoie une requête POST /api/rendezvous avec RendezVousDTO. 10. Le backend enregistre le RDV avec statut "PLANIFIE". 11. Le système envoie une notification email au patient via NotificationEmailService. 12. Le système affiche un message de succès "Rendez-vous créé avec succès". 13. Le système affiche le nouveau RDV dans le calendrier.

Cas d'utilisation	Créer un Rendez-vous
Scénario alternatif	1. Le créneau est déjà occupé → affichage d'un message "Ce créneau est déjà réservé". 2. Aucun patient sélectionné → affichage d'un message "Veuillez sélectionner un patient". 3. La date est passée → affichage d'un message "Impossible de créer un RDV dans le passé". 4. Erreur serveur → affichage d'un message "Erreur lors de la création, veuillez réessayer".

Description textuelle du cas d'utilisation "Modifier un Rendez-vous" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Modifier un Rendez-vous". Ce scénario permet au médecin de modifier les détails d'un rendez-vous planifié.

Table 3.14: Description textuelle du cas d'utilisation "Modifier un Rendez-vous"

Cas d'utilisation	Modifier un Rendez-vous
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Un rendez-vous existe avec le statut "PLANIFIE" ou "CONFIRME". 3. Le médecin a accès à la section "Rendez-vous".
Post-condition	1. Le rendez-vous est mis à jour avec les nouvelles informations. 2. Une notification de modification est envoyée au patient (email). 3. Le calendrier est mis à jour automatiquement.

Cas d'utilisation	Modifier un Rendez-vous
Scénario principal	1. Le médecin accède à la section "Rendez-vous". 2. Le médecin sélectionne un rendez-vous dans la liste ou le calendrier. 3. Le médecin clique sur "Modifier". 4. Le système affiche le formulaire pré-rempli avec les informations actuelles. 5. Le médecin modifie les champs souhaités (date, heure, motif). 6. Le médecin clique sur "Enregistrer". 7. Le système vérifie la disponibilité du nouveau créneau si la date/heure a changé. 8. Le système envoie une requête PUT /api/rendezvous/{id} avec RendezVousDTO mis à jour. 9. Le backend met à jour le RDV via RendezVousService.update(). 10. Le système envoie une notification de modification au patient. 11. Le système affiche un message de succès "Rendez-vous modifié avec succès". 12. Le système rafraîchit le calendrier.
Scénario alternatif	1. Le nouveau créneau est déjà occupé → affichage d'un message "Ce créneau est déjà réservé". 2. Le rendez-vous a déjà eu lieu (statut "TERMINE") → affichage d'un message "Impossible de modifier un RDV terminé". 3. Erreur serveur → affichage d'un message "Erreur lors de la modification, veuillez réessayer".

Description textuelle du cas d'utilisation "Annuler un Rendez-vous" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Annuler un Rendez-vous". Ce scénario permet au médecin d'annuler un rendez-vous planifié.

Table 3.15: Description textuelle du cas d'utilisation "Annuler un Rendez-vous"

Cas d'utilisation	Annuler un Rendez-vous
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Un rendez-vous existe avec le statut "PLANIFIE" ou "CONFIRME". 3. Le médecin a accès à la section "Rendez-vous".
Post-condition	1. Le statut du rendez-vous passe à "ANNULE". 2. Une notification d'annulation est envoyée au patient (email). 3. Le créneau redevient disponible dans le calendrier. 4. Le RDV apparaît dans la liste des RDV annulés.

Cas d'utilisation	Annuler un Rendez-vous
Scénario principal	1. Le médecin accède à la section "Rendez-vous". 2. Le médecin sélectionne un rendez-vous planifié. 3. Le médecin clique sur "Annuler". 4. Le système affiche une boîte de dialogue de confirmation "Voulez-vous vraiment annuler ce rendez-vous ? Le patient sera notifié.". 5. Le médecin peut saisir un motif d'annulation (optionnel). 6. Le médecin confirme l'annulation. 7. Le système envoie une requête PATCH /api/rendezvous/{id}/annuler avec le motif. 8. Le backend change le statut à "ANNULE" via RendezVousService.annuler(). 9. Le système envoie une notification d'annulation au patient avec le motif. 10. Le système affiche un message de succès "Rendez-vous annulé avec succès". 11. Le système rafraîchit le calendrier et la liste des RDV.
Scénario alternatif	1. Le médecin annule la confirmation → aucune annulation n'est effectuée. 2. Le rendez-vous est déjà annulé → affichage d'un message "Ce rendez-vous est déjà annulé". 3. Le rendez-vous est déjà terminé → affichage d'un message "Impossible d'annuler un RDV terminé". 4. Erreur serveur → affichage d'un message "Erreur lors de l'annulation, veuillez réessayer".

Description textuelle du cas d'utilisation "Consulter un Dossier Médical" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Consulter un Dossier Médical". Ce scénario permet au médecin de visualiser l'historique médical complet d'un patient.

Table 3.16: Description textuelle du cas d'utilisation "Consulter un Dossier Médical"

Cas d'utilisation	Consulter un Dossier Médical
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Un patient existe dans le système. 3. Le médecin a accès à la section "Dossiers Médicaux".
Post-condition	1. Le médecin peut visualiser tous les dossiers médicaux du patient. 2. Le médecin peut accéder aux documents associés à chaque dossier.

Cas d'utilisation	Consulter un Dossier Médical
Scénario principal	1. Le médecin accède à la section "Dossiers Médicaux" (/dashboard/dossiers). 2. Le médecin sélectionne un patient dans la liste. 3. Le système envoie une requête GET /api/dossiers/patient/{id}. 4. Le backend récupère tous les dossiers du patient via DossierPatientService.findByPatient(). 5. Le système affiche l'historique complet avec : date de consultation, diagnostic, traitement, observations, recommandations. 6. Le médecin peut cliquer sur un dossier pour voir les détails complets. 7. Le médecin peut visualiser la liste des documents attachés (PDF, images). 8. Le médecin peut télécharger ou visualiser chaque document.
Scénario alternatif	1. Aucun dossier médical n'existe pour ce patient → affichage d'un message "Aucun dossier médical pour ce patient". 2. Erreur serveur → affichage d'un message "Erreur lors du chargement, veuillez réessayer".

Description textuelle du cas d'utilisation "Ajouter des Documents à un Dossier" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Ajouter des Documents à un Dossier". Ce scénario permet au médecin d'uploader des documents médicaux (PDF, images).

Table 3.17: Description textuelle du cas d'utilisation "Ajouter des Documents à un Dossier"

Cas d'utilisation	Ajouter des Documents à un Dossier
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Un dossier médical existe dans le système. 3. Le médecin a des fichiers à uploader (PDF, JPG, PNG).
Post-condition	1. Les documents sont uploadés et stockés dans le système de fichiers. 2. Les métadonnées des documents sont enregistrées dans la base de données. 3. Les documents sont visibles dans le dossier médical du patient.

Cas d'utilisation	Ajouter des Documents à un Dossier
Scénario principal	1. Le médecin accède au dossier médical d'un patient. 2. Le médecin clique sur "Ajouter des Documents". 3. Le système affiche un dialogue de sélection de fichiers. 4. Le médecin sélectionne un ou plusieurs fichiers (PDF, JPG, PNG). 5. Le médecin peut ajouter une description pour chaque document (optionnel). 6. Le médecin clique sur "Uploader". 7. Le système valide les fichiers (type, taille < 10 MB). 8. Le système envoie une requête POST /api/dossiers/{id}/files avec FormData (multipart). 9. Le backend sauvegarde les fichiers via FileStorageService. 10. Le backend enregistre les métadonnées (nom original, chemin, type MIME, taille) dans la table Document. 11. Le système affiche un message de succès "Documents uploadés avec succès". 12. Le système rafraîchit la liste des documents du dossier.

Cas d'utilisation	Ajouter des Documents à un Dossier
Scénario alternatif	1. Le fichier est trop volumineux (> 10 MB) → affichage d'un message "Fichier trop volumineux (max 10 MB)". 2. Le format de fichier est non autorisé → affichage d'un message "Format non supporté. Formats acceptés : PDF, JPG, PNG". 3. Erreur lors de l'upload → affichage d'un message "Erreur lors de l'upload, veuillez réessayer". 4. Espace disque insuffisant → affichage d'un message "Erreur serveur : espace insuffisant".

Description textuelle du cas d'utilisation "Télécharger des Documents" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Télécharger des Documents". Ce scénario permet au médecin de télécharger un document médical sur son ordinateur.

Table 3.18: Description textuelle du cas d'utilisation "Télécharger des Documents"

Cas d'utilisation	Télécharger des Documents
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Un document existe dans un dossier médical. 3. Le médecin a accès à la section "Dossiers Médicaux".
Post-condition	1. Le document est téléchargé sur l'ordinateur du médecin. 2. Le nom original du fichier est conservé.

Cas d'utilisation	Télécharger des Documents
Scénario principal	1. Le médecin accède au dossier médical d'un patient. 2. Le médecin visualise la liste des documents attachés. 3. Le médecin clique sur "Télécharger" (icône download) à côté d'un document. 4. Le système envoie une requête GET /api/dossiers/{dossierId}/files/{docId}. 5. Le backend récupère le fichier via FileStorageService.load(). 6. Le backend envoie le fichier avec les en-têtes appropriés (Content-Disposition: attachment). 7. Le navigateur déclenche le téléchargement automatique avec le nom original du fichier. 8. Le système affiche un message de succès "Document téléchargé avec succès".
Scénario alternatif	1. Le fichier n'existe plus sur le serveur → affichage d'un message "Fichier introuvable sur le serveur". 2. Erreur serveur lors du téléchargement → affichage d'un message "Erreur lors du téléchargement, veuillez réessayer". 3. Le médecin n'a pas les droits d'accès → erreur 403 Forbidden.

Description textuelle du cas d'utilisation "Générer des Rapports" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Générer des Rapports". Ce scénario permet au médecin de générer des rapports financiers détaillés avec statistiques.

Table 3.19: Description textuelle du cas d'utilisation "Générer des Rapports"

Cas d'utilisation	Générer des Rapports
Acteur	Médecin
Précondition	1. Le médecin est authentifié avec le rôle "MEDECIN". 2. Des factures existent dans le système. 3. Le médecin a accès à la section "Rapports".
Post-condition	1. Un rapport financier détaillé est généré avec statistiques complètes. 2. Le rapport affiche : chiffre d'affaires total, montant des impayés, nombre de factures, taux de paiement. 3. Le rapport peut être filtré par période (jour, semaine, mois, année). 4. Le rapport peut être exporté en PDF ou Excel (si implémenté).

Cas d'utilisation	Générer des Rapports
Scénario principal	1. Le médecin accède à la section "Rapports" (/dashboard/rapports). 2. Le médecin sélectionne une période (date début, date fin). 3. Le médecin peut choisir des filtres supplémentaires (statut : payée/impayée, patient spécifique). 4. Le médecin clique sur "Générer Rapport". 5. Le système envoie une requête GET /api/factures/rapports avec les filtres (dateDebut, dateFin, statut). 6. Le backend calcule les statistiques via FactureService.generateReport() : • Chiffre d'affaires total (somme des factures payées) • Montant des impayés (somme des factures impayées) • Nombre total de factures • Taux de paiement (pourcentage de factures payées) • Répartition par mode de paiement (CB, Espèces, Chèque, Virement) 7. Le système affiche le rapport avec graphiques (barres, camembert) et tableaux détaillés. 8. Le médecin peut exporter le rapport en PDF ou Excel (bouton "Exporter").

Cas d'utilisation	Générer des Rapports
Scénario alternatif	1. Aucune facture dans la période sélectionnée → affichage d'un message "Aucune facture pour cette période". 2. La date de début est postérieure à la date de fin → affichage d'un message "Dates invalides". 3. Erreur serveur → affichage d'un message "Erreur lors de la génération, veuillez réessayer".

3.4 Conception

Dans cette section, nous présentons les diagrammes de conception du sprint 1, notamment le diagramme de séquence et le diagramme de classes.

3.4.1 Diagrammes de séquence

Diagramme de séquence "S'authentifier" :

La figure 3.2 illustre le diagramme de séquence d'authentification du médecin, montrant les interactions entre le médecin, le frontend Next.js, l'API REST Spring Boot, et la génération du token JWT.

Figure 3.2: Diagramme de séquence - S'authentifier

Diagramme de séquence "Créer un Assistant" :

La figure 3.3 illustre le diagramme de séquence de création d'un assistant avec hashage BCrypt du mot de passe et envoi d'email de notification.

images/sequence-creer-assistant.png

Figure 3.3: Diagramme de séquence - Créer un Assistant

Diagramme de séquence "Créer un Rendez-vous" :

La figure 3.4 illustre le diagramme de séquence de création d'un rendez-vous avec vérification de disponibilité du créneau et notification automatique au patient.

Figure 3.4: Diagramme de séquence - Créer un Rendez-vous

Diagramme de séquence "Modifier un Dossier Médical" :

La figure 3.5 illustre le diagramme de séquence de modification d'un dossier médical, montrant les interactions entre le médecin, le frontend Next.js, l'API REST Spring Boot et la base de données.

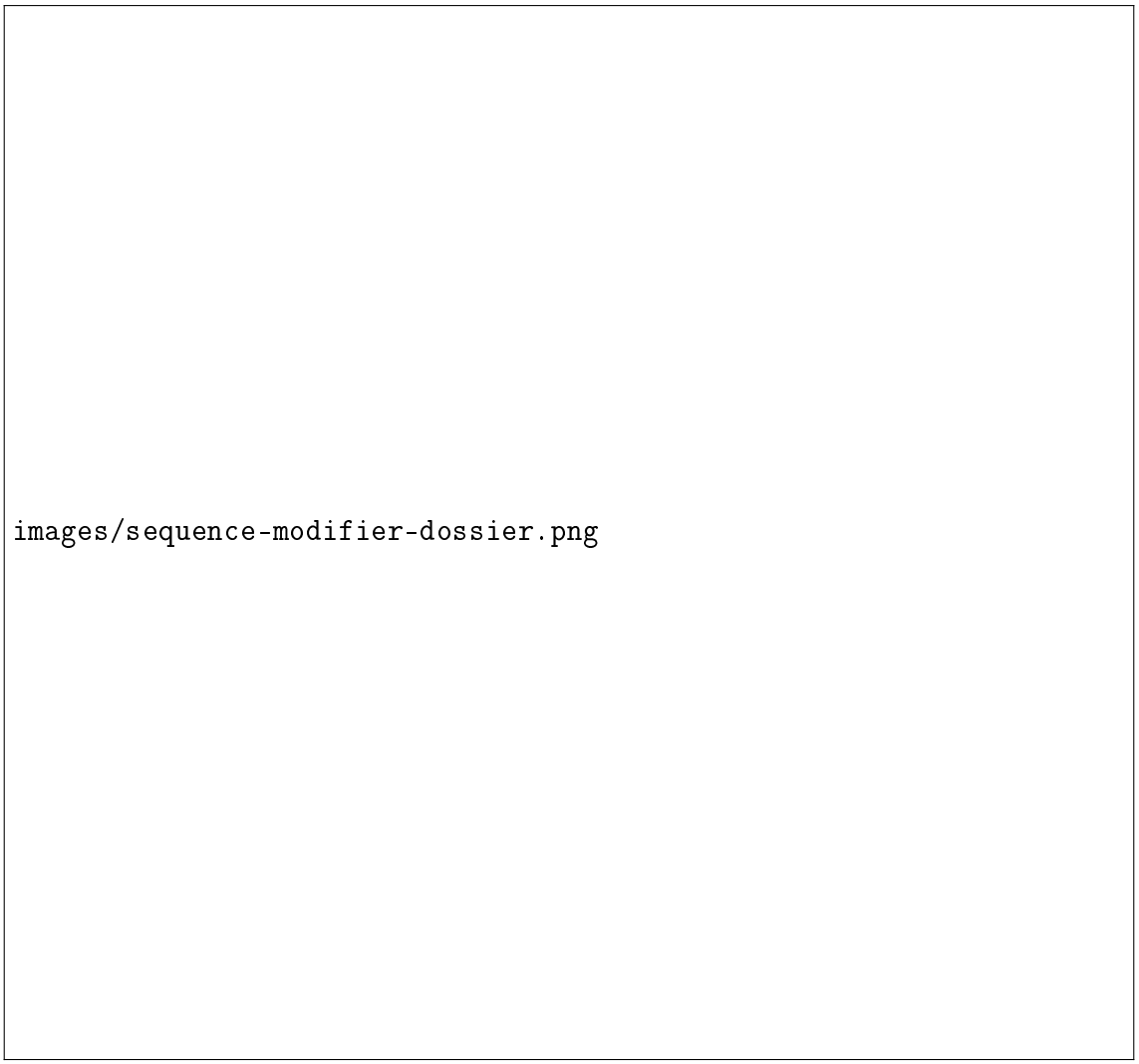

Figure 3.5: Diagramme de séquence - Modifier un Dossier Médical

Diagramme de séquence "Uploader des Documents" :

La figure 3.6 illustre le diagramme de séquence d'upload de documents médicaux avec gestion multipart/form-data et stockage sur le serveur.

images/sequence-upload-documents.png

Figure 3.6: Diagramme de séquence - Uploader des Documents

Diagramme de séquence "Créer une Facture" :

La figure 3.7 illustre le diagramme de séquence de création d'une facture avec génération automatique du numéro de facture unique.

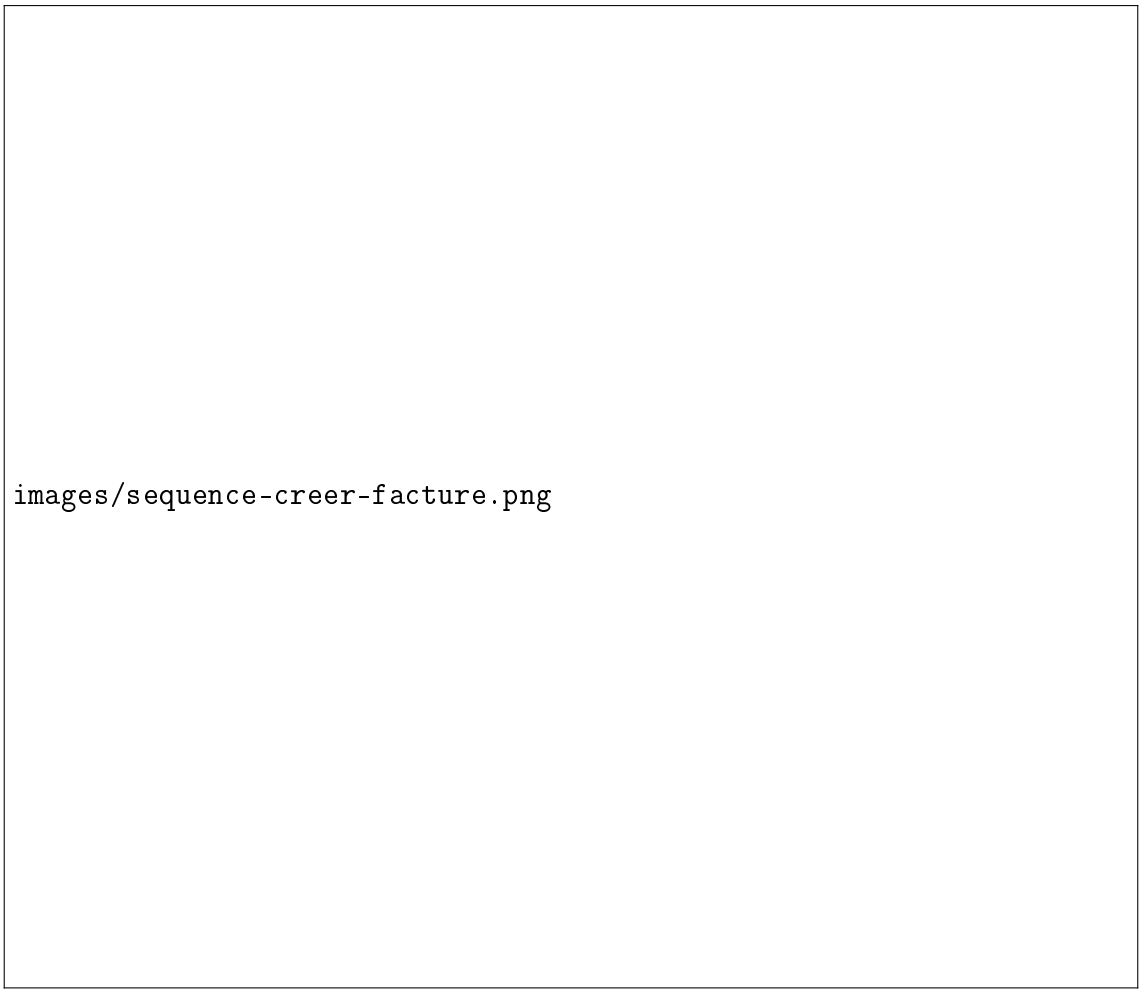

Figure 3.7: Diagramme de séquence - Créer une Facture

Diagramme de séquence "Enregistrer un Paiement" :

La figure 3.8 illustre le diagramme de séquence d'enregistrement d'un paiement sur une facture avec mise à jour automatique des statistiques.

Figure 3.8: Diagramme de séquence - Enregistrer un Paiement

3.4.2 Diagramme de classes du sprint 1

Le diagramme de classes du sprint 1, présenté dans la figure 3.9, illustre les entités principales du système pour l'espace médecin : User (Médecin, Assistant), Patient, DossierMedical, Document, Facture, RendezVous.

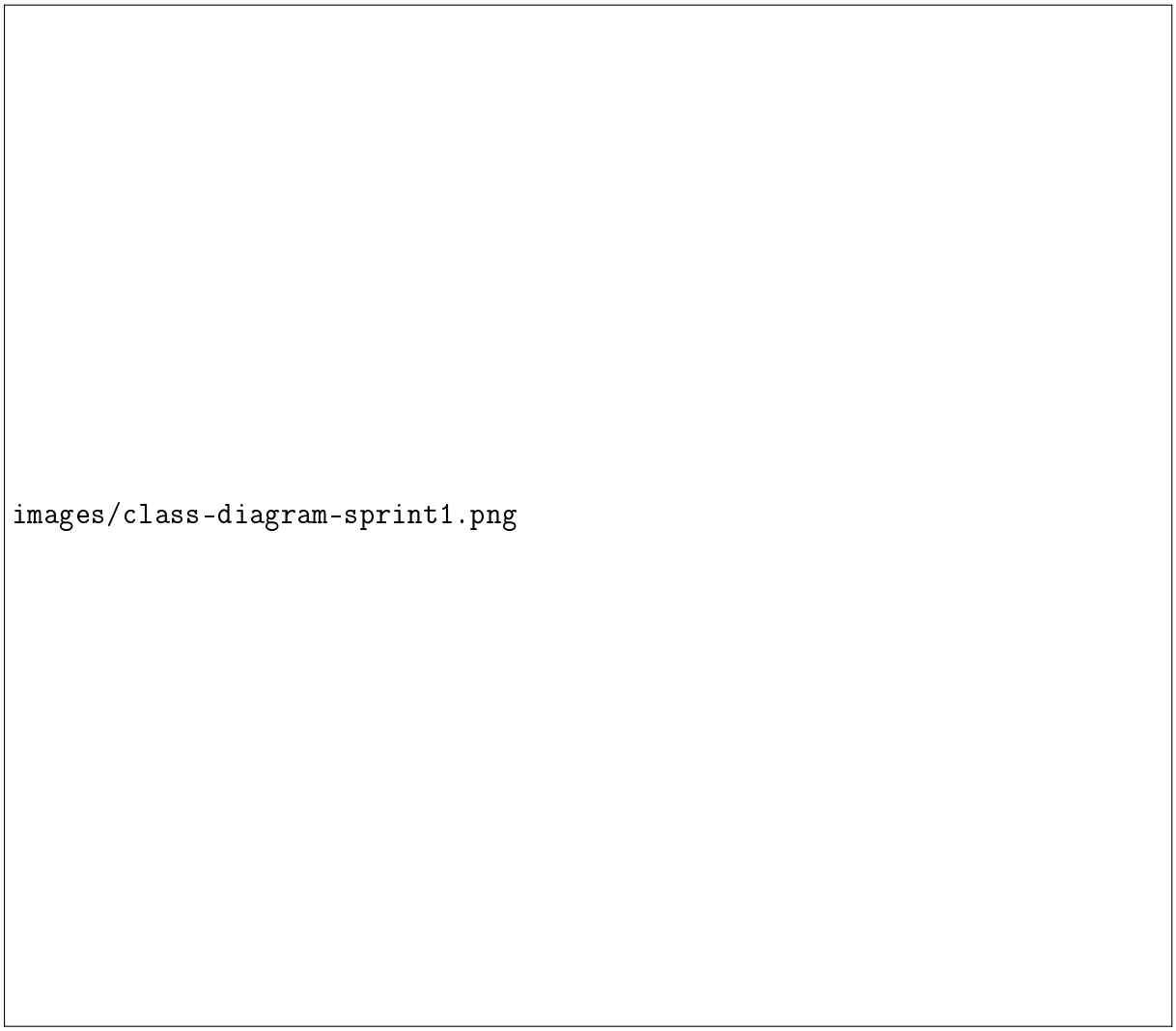

Figure 3.9: Diagramme de classes - Sprint 1

Description des principales classes :

- **User :** Représente les utilisateurs (Médecin, Assistant) avec id, nom, email, password (BCrypt hashé), role (enum: MEDECIN/ASSISTANT/PATIENT), active (boolean). Annotations JPA : @Entity, @Table(name="users").
- **Patient :** Représente un patient avec id, nom, prenom, dateNaissance, telephone, email, adresse. Relations OneToMany avec DossierPatient, Facture et RendezVous.
- **DossierPatient :** Représente un dossier médical avec id, dateCreation, diagnostic, traitement, symptomes, observations, recommandations. Relations ManyToOne avec Patient et Medecin, OneToOne avec RendezVous (optionnelle), OneToMany avec Document.
- **Document :** Représente un document médical avec id, nomFichier, cheminFichier, typeFichier, tailleFichier, dateUpload. Relation ManyToOne avec DossierPatient.

- **Facture :** Représente une facture avec id, numeroFacture, montant, description, dateEmission, datePaiement, statut (enum: PAYEE/IMPAYEE), modePaiement (enum: CB/ESPECES/CHEQUE/VIREMENT). Relations ManyToOne avec Patient et Medecin.
- **RendezVous :** Représente un rendez-vous avec id, dateHeureDebut, dateHeureFin, motif, statut (enum: CONFIRME/ANNULE/EN$_ATTENTE).RelationsManyToOneavecPatie$

3.5 Réalisation

Cette section présente les captures d'écran des principales fonctionnalités développées pour l'espace médecin.

3.5.1 Authentification

Figure 3.10 : Page de connexion médecin

images/login-medecin.png

Figure 3.10: Page de connexion avec formulaire email/mot de passe et validation JWT

3.5.2 Tableau de bord médecin

Figure 3.11 : Dashboard médecin avec statistiques

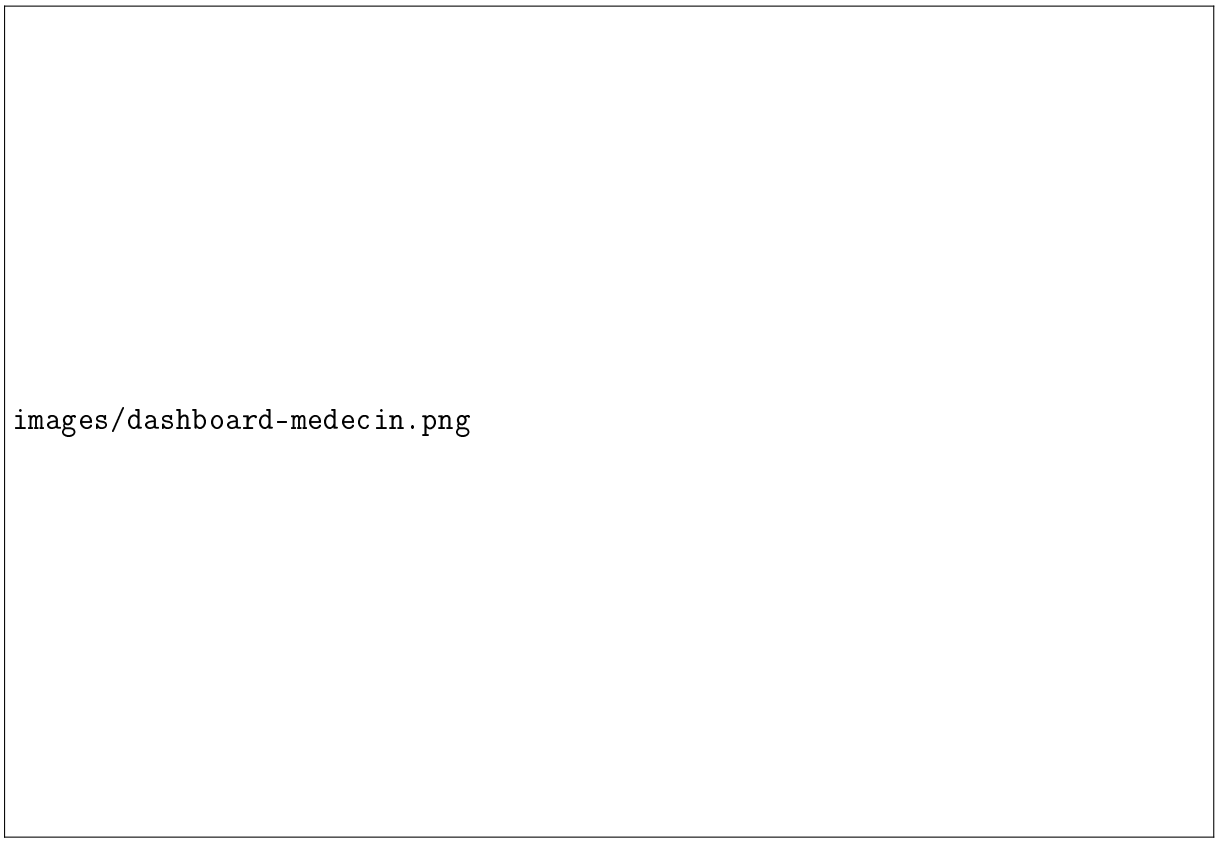

Figure 3.11: Dashboard avec cartes statistiques (patients, rendez-vous, factures), graphiques de revenus et prochains RDV du jour

3.5.3 Gestion des patients

Figure 3.12 : Liste des patients

images/liste-patients-medecin.png

Figure 3.12: Liste complète des patients avec nom, téléphone, email, date d'inscription et actions (consulter, modifier)

Figure 3.13 : Profil détaillé d'un patient

Figure 3.13: Profil complet avec informations personnelles, historique des rendez-vous et accès aux dossiers médicaux

3.5.4 Gestion des rendez-vous

Figure 3.14 : Calendrier des rendez-vous

images/calendrier-medecin.png

Figure 3.14: Calendrier hebdomadaire avec rendez-vous colorés par statut (planifié, confirmé, terminé, annulé)

Figure 3.15 : Formulaire de création de rendez-vous

Figure 3.15: Formulaire modal avec sélection patient, date, heure, durée et motif de consultation

Figure 3.16 : Liste des rendez-vous avec filtres

Figure 3.16: Liste des rendez-vous avec filtres (date, statut, patient) et actions (modifier, annuler)

3.5.5 Gestion des assistants

Figure 3.17 : Liste des assistants

Figure 3.17: Liste des assistants avec nom, email, statut (actif/inactif) et actions (modifier, activer/désactiver, supprimer)

Figure 3.18 : Formulaire de création d'assistant

images/creer-assistant.png

Figure 3.18: Formulaire modal avec champs nom, email, mot de passe, confirmation mot de passe

3.5.6 Gestion des dossiers médicaux

Figure 3.19 : Modification d'un dossier médical

images/modifier-dossier-medical.png

Figure 3.19: Formulaire de modification avec champs diagnostic, traitement, observations, antécédents et zone d'upload de documents supplémentaires

Figure 3.20 : Historique des dossiers médicaux d'un patient

images/historique-dossiers.png

Figure 3.20: Liste chronologique des dossiers avec date, diagnostic, traitement et documents associés (PDF, images)

Figure 3.21 : Upload de documents médicaux

Figure 3.21: Interface d'upload avec drag & drop, gestion multipart/form-data, validation des formats (PDF, JPG, PNG)

3.5.7 Gestion des factures

Figure 3.22 : Liste des factures avec filtres

images/liste-factures.png

Figure 3.22: Liste des factures avec filtres (payée/impayée, période), affichage du numéro, patient, montant, statut et actions

Figure 3.23 : Formulaire de création de facture

Figure 3.23: Formulaire avec sélection patient, montant, description des actes et génération automatique du numéro de facture

Figure 3.24 : Enregistrement d'un paiement

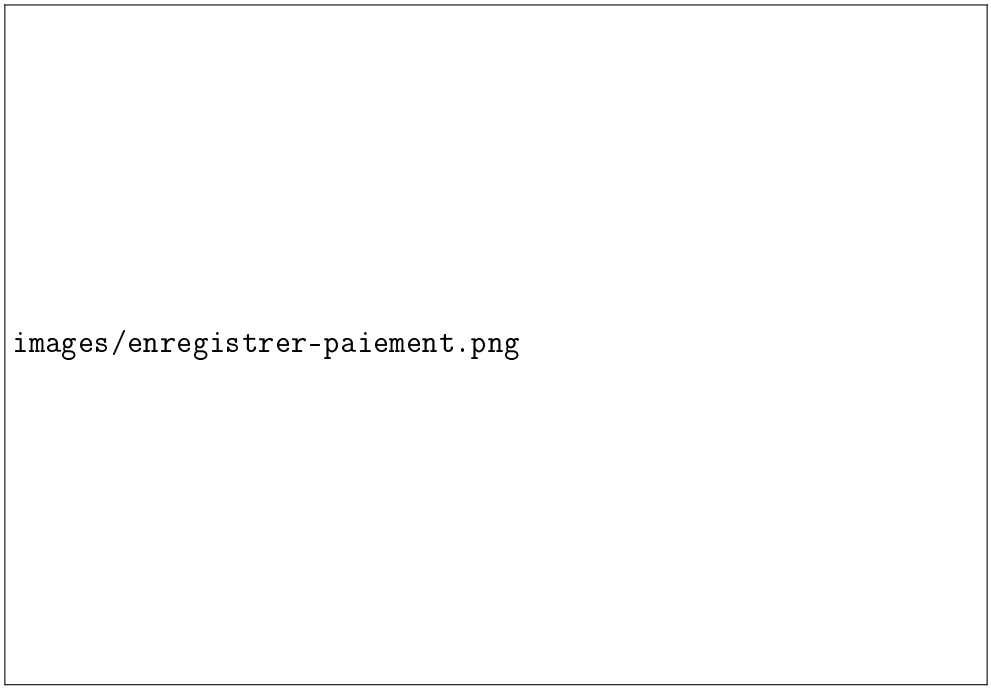

Figure 3.24: Modal de paiement avec sélection du mode (CB, Espèces, Chèque, Virement)

3.5.8 Rapports financiers

Figure 3.25 : Rapports financiers détaillés

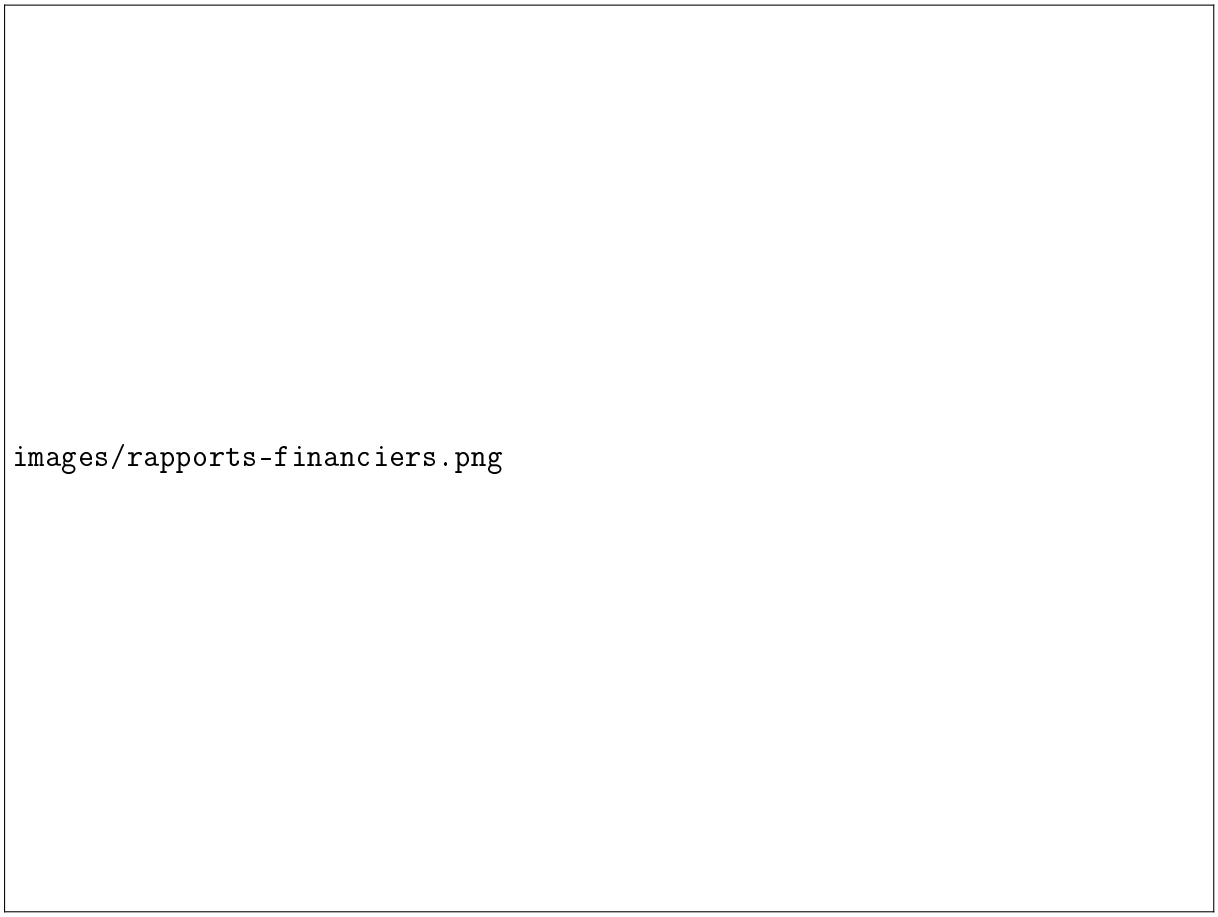

Figure 3.25: Graphiques des revenus par mois avec statistiques (CA total, factures payées/impayées, taux de paiement, répartition par mode de paiement)

Figure 3.26 : Filtres et export de rapports

images/filtres-rapports.png

Figure 3.26: Interface de filtrage par période (date début, date fin) avec boutons d'export PDF et Excel

3.5.9 Gestion du profil et paramètres

Figure 3.27 : Profil médecin

Figure 3.27: Formulaire de modification du profil avec nom, email, téléphone et changement de mot de passe sécurisé

Figure 3.28 : Paramètres du cabinet

images/parametres-cabinet.png

Figure 3.28: Configuration des horaires de travail, durée par défaut des rendez-vous et préférences de notification

3.6 CONCLUSION

Ce chapitre a présenté la réalisation du sprint 1 focalisé sur l'espace médecin, qui constitue le cœur du système de gestion de cabinet médical. Nous avons développé l'ensemble des fonctionnalités essentielles regroupées en 10 modules principaux :

- **Authentification sécurisée** : connexion et déconnexion avec gestion de session JWT
- **Gestion des assistants** : création, consultation, modification, activation/désactivation et suppression des comptes assistants avec notifications automatiques
- **Gestion des patients** : consultation de la liste complète, modification des informations personnelles, recherche rapide et accès au profil détaillé avec historique
- **Gestion des rendez-vous** : création, consultation avec filtres, modification, annulation avec notification automatique, et visualisation dans un calendrier interactif

- **Gestion des dossiers médicaux** : consultation de l'historique complet, modification des dossiers existants, upload de documents médicaux (PDF, images) et téléchargement sécurisé
- **Gestion financière complète** : création de factures, consultation avec filtres (payée/impayée, période), enregistrement des paiements avec mode (CB, espèces, chèque, virement), suppression réservée au médecin
- **Génération de rapports** : rapports financiers détaillés avec statistiques (chiffre d'affaires, taux de paiement, impayés), exportation possible en PDF/Excel
- **Tableau de bord** : vue d'ensemble avec statistiques principales, graphiques de performance, prochains rendez-vous du jour et accès rapide aux factures impayées
- **Gestion du profil** : consultation et modification des informations personnelles, changement de mot de passe sécurisé
- **Paramètres du cabinet** : configuration des horaires, durée des rendez-vous, préférences de notification

L'architecture technique mise en place (API REST Spring Boot + frontend Next.js) a permis de développer des interfaces modernes, réactives et sécurisées. Le système de contrôle d'accès basé sur les rôles (RBAC) garantit que seul le médecin peut accéder à ces fonctionnalités sensibles, notamment la suppression de factures et la gestion complète des assistants.

Le sprint 1 représente un total de 36 user stories développées avec succès, couvrant l'intégralité des besoins fonctionnels du médecin. Le prochain chapitre détaillera le sprint 2 dédié à l'espace assistant, avec des permissions restreintes par rapport au médecin, notamment pour la gestion des patients, la gestion des rendez-vous et la création de factures.

Chapter

4 CHAPITRE 4 : Sprint 2 - Gestion de l'espace Assistant

Contents

4.1 INTRODUCTION 106

4.2 Backlog du sprint 2 106

4.3 Spécifications fonctionnelles 108

4.3.1 Diagramme de cas d'utilisation du sprint 2 108

4.3.2 Descriptions textuelles 109

4.4 Conception 122

4.4.1 Diagrammes de séquence 122

4.4.2 Diagramme de classes du sprint 2 124

4.5 Réalisation 126

4.5.1 Authentification 126

4.5.2 Tableau de bord assistant 126

4.5.3 Gestion des patients 127

4.5.4 Gestion des rendez-vous 129

4.5.5 Gestion des factures (restreint) 132

4.5.6 Contrôle d'accès RBAC 133

4.6 Implémentation du contrôle d'accès (RBAC) 134

4.6.1 Backend - Spring Security 134

4.6.2 Frontend - Next.js 135

4.7 CONCLUSION 136

4.1 INTRODUCTION

Dans ce chapitre, nous présentons le deuxième sprint intitulé "Gestion de l'espace Assistant". Après avoir développé l'espace médecin lors du sprint 1, nous nous concentrons maintenant sur les fonctionnalités dédiées aux assistants médicaux avec un système de permissions restrictives. Nous allons commencer par la présentation du backlog du sprint 2, suivie de la spécification fonctionnelle avec diagrammes UML, de la conception, et enfin nous illustrerons ce sprint à l'aide de captures d'écran. L'accent est mis sur le contrôle d'accès basé sur les rôles (RBAC) pour garantir la confidentialité des données médicales.

4.2 Backlog du sprint 2

Le tableau **??** présente l'ensemble des user stories sélectionnées pour le sprint 2, focalisées sur l'espace assistant avec permissions restreintes.

Table 4.1: Backlog du sprint 2 - Espace Assistant

Fonctionnalités	User Stories	Priorités	Estimation
Gestion d'Authentification	En tant qu'assistant, je veux m'authentifier pour accéder à mon espace sécurisé	Élevée	2 jours
	En tant qu'assistant, je veux me déconnecter pour quitter ma session en toute sécurité	Moyenne	1 jour
Gérer les Patients	En tant qu'assistant, je veux consulter la liste de tous les patients avec recherche et filtres	Moyenne	3 jours
	En tant qu'assistant, je veux modifier les informations d'un patient (nom, prénom, téléphone, email) via PUT /api/patients/update/{id}	Moyenne	2 jours
	En tant qu'assistant, je veux consulter la fiche complète d'un patient (données personnelles uniquement)	Moyenne	2 jours

Fonctionnalités	User Stories	Priorités	Estimation
Gérer les Rendez-vous	En tant qu'assistant, je veux créer un rendez-vous pour un patient en vérifiant la disponibilité du médecin	Élevée	5 jours
	En tant qu'assistant, je veux consulter le calendrier de rendez-vous avec vue hebdomadaire/mensuelle	Moyenne	3 jours
	En tant qu'assistant, je veux modifier un rendez-vous existant (date, heure, motif)	Moyenne	3 jours
	En tant qu'assistant, je veux annuler un rendez-vous avec notification automatique au patient	Moyenne	2 jours
Gérer les Factures (Restreint)	En tant qu'assistant, je veux créer une facture pour un patient avec montant et description	Élevée	3 jours
	En tant qu'assistant, je veux consulter la liste des factures (lecture seule, sans accès aux statistiques financières)	Moyenne	2 jours
Restrictions de Sécurité	En tant que système, je dois interdire à l'assistant l'accès aux rapports financiers et à la gestion d'autres assistants	Élevée	3 jours

Chaque user story représente une fonctionnalité que l'assistant souhaite pouvoir réaliser dans le système, avec des restrictions importantes pour garantir la confidentialité des données. Les priorités sont classées en élevée, moyenne ou faible, reflétant l'importance relative de chaque user story.

Contraintes de sécurité spécifiques au rôle Assistant :

- **Accès INTERDIT** aux rapports financiers et statistiques détaillées
- **Accès INTERDIT** à la gestion d'autres assistants
- **Accès LIMITÉ** aux factures : création pour patients liés uniquement, enregistrement paiements
- **Accès LIMITÉ** aux patients : consultation et modification des données personnelles

- **Inscription nouveaux patients** : via page publique /register (pas depuis dashboard)

4.3 Spécifications fonctionnelles

Dans cette partie, nous présentons le diagramme de cas d'utilisation du sprint 2 ainsi que les descriptions textuelles détaillées.

4.3.1 Diagramme de cas d'utilisation du sprint 2

Le diagramme de cas d'utilisation du sprint 2, présenté dans la figure 4.1, illustre les besoins fonctionnels sous la forme d'interactions entre l'assistant et le système, avec mise en évidence des restrictions d'accès.

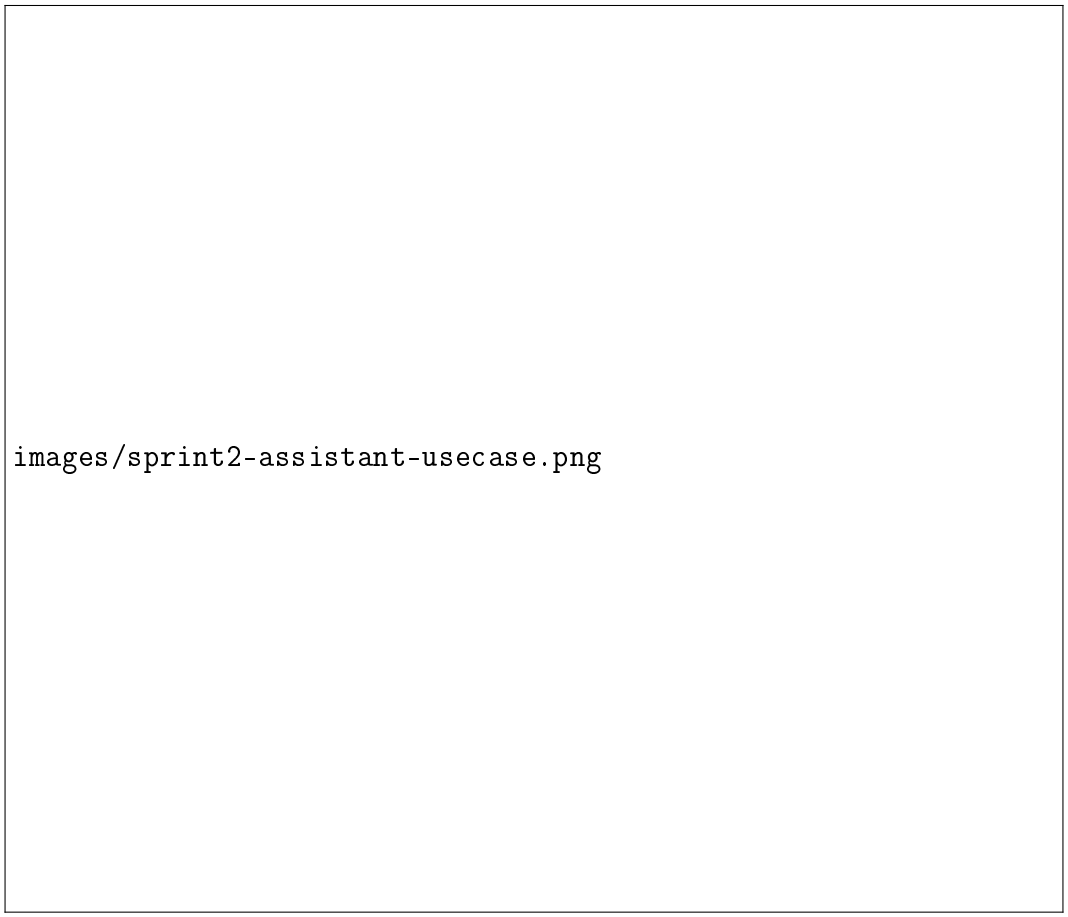

Figure 4.1: Diagramme de cas d'utilisation du sprint 2 - Espace Assistant

4.3.2 Descriptions textuelles

L'objectif de cette activité est de décrire textuellement les scénarios des cas d'utilisation. Il faut préciser comment chaque scénario commence, comment il se termine et comment l'assistant interagit avec l'application web.

Description textuelle du cas d'utilisation "S'authentifier - Assistant" :

Le tableau **??** présente la description textuelle du cas d'utilisation "S'authentifier". Ce scénario commence lorsque l'assistant ouvre l'application et accède à l'écran de connexion.

Table 4.2: Description textuelle du cas d'utilisation "S'authentifier - Assistant"

Cas d'utilisation	S'authentifier - Assistant
Acteur	Assistant
Précondition	1. Le système est en service. 2. L'assistant possède un compte actif créé par le médecin. 3. Le compte n'est pas désactivé (champ actif = true).
Post-condition	1. L'assistant est authentifié avec un token JWT contenant son rôle "ASSISTANT". 2. Le système affiche le tableau de bord assistant avec accès restreint aux fonctionnalités autorisées. 3. L'assistant ne peut PAS accéder aux dossiers médicaux, rapports financiers, ni gestion des assistants.

Cas d'utilisation	S'authentifier - Assistant
Scénario principal	1. L'assistant accède à la page de connexion (/login). 2. Le système affiche le formulaire d'authentification. 3. L'assistant saisit son email et son mot de passe. 4. L'assistant clique sur "Se connecter". 5. Le système vérifie les identifiants via l'API REST /api/auth/login. 6. Le système vérifie que le compte est actif (actif = true). 7. Le système génère un token JWT avec le rôle "ASSISTANT". 8. Le système stocke le token dans le localStorage du navigateur. 9. Le système redirige vers /dashboard/assistant (tableau de bord assistant). 10. Le frontend affiche uniquement les menus autorisés : Patients, Rendez-vous, Factures (création uniquement).

Cas d'utilisation	S'authentifier - Assistant
Scénario alternatif	1. L'assistant saisit des données incomplètes → affichage d'un message d'erreur "Tous les champs sont obligatoires". 2. L'assistant saisit des identifiants incorrects → affichage d'un message d'erreur "Email ou mot de passe incorrect". 3. Le compte est désactivé par le médecin → affichage d'un message d'erreur "Compte désactivé, contactez le médecin". 4. L'assistant tente d'accéder à /dashboard/dossiers (dossiers médicaux) → redirection vers /dashboard/assistant avec message "Accès non autorisé".

Description textuelle du cas d'utilisation "Consulter les Patients" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Consulter les Patients". Ce scénario permet à l'assistant de consulter la liste des patients du cabinet. Note importante : L'enregistrement de nouveaux patients se fait via la page publique d'inscription (/register) pour des raisons de sécurité et de gestion des comptes utilisateurs.

Table 4.3: Description textuelle du cas d'utilisation "Consulter les Patients"

Cas d'utilisation	Consulter les Patients
Acteur	Assistant
Précondition	1. L'assistant est authentifié avec le rôle "ASSISTANT". 2. L'assistant a accès à la section "Gestion des Patients".

Cas d'utilisation	Consulter les Patients
Post-condition	1. L'assistant visualise la liste des patients. 2. L'assistant peut filtrer et rechercher des patients.
Scénario principal	1. L'assistant accède à la section "Patients" (/dashboard/patients). 2. Le système affiche la liste complète des patients via GET /api/patients/allPatients. 3. L'assistant peut utiliser la barre de recherche pour filtrer par nom, prénom, email ou téléphone. 4. L'assistant peut voir les informations basiques de chaque patient (nom, prénom, date de naissance, email, téléphone). 5. L'assistant peut cliquer sur un patient pour voir ses détails complets. 6. L'assistant peut modifier les informations d'un patient via le bouton "Modifier".

Cas d'utilisation	Consulter les Patients
Scénario alternatif	1. L'assistant laisse des champs obligatoires vides (nom, prénom, date de naissance, téléphone) → affichage d'erreurs de validation. 2. Le format de l'email est invalide → affichage d'un message "Email invalide". 3. La date de naissance est invalide ou dans le futur → affichage d'un message "Date de naissance invalide". 4. L'email est déjà utilisé → affichage d'un message "Un patient avec cet email existe déjà". 5. Erreur serveur lors de la création → affichage d'un message "Erreur lors de la création, veuillez réessayer".

Description textuelle du cas d'utilisation "Créer un Rendez-vous" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Créer un Rendez-vous". Ce scénario permet à l'assistant de planifier un rendez-vous entre un patient et le médecin.

Table 4.4: Description textuelle du cas d'utilisation "Créer un Rendez-vous"

Cas d'utilisation	Créer un Rendez-vous
Acteur	Assistant
Précondition	1. L'assistant est authentifié avec le rôle "ASSISTANT". 2. Un patient existe dans le système. 3. L'assistant a accès à la section "Rendez-vous".

Cas d'utilisation	Créer un Rendez-vous
Post-condition	1. Un nouveau rendez-vous est créé avec le statut "CONFIRME". 2. Le rendez-vous est associé au patient, au médecin et à l'assistant (créateur). 3. Une notification par email est envoyée automatiquement au patient avec les détails du rendez-vous. 4. Le rendez-vous apparaît dans le calendrier du médecin et de l'assistant. 5. Le créneau horaire est bloqué pour éviter les doubles réservations.

Cas d'utilisation	Créer un Rendez-vous
Scénario principal	1. L'assistant accède à la section "Rendez-vous" (/dashboard/assistant/rendezvous). 2. L'assistant clique sur "Créer un Rendez-vous". 3. Le système affiche le formulaire de création avec les champs : sélection patient, date, heure de début, heure de fin, motif. 4. L'assistant sélectionne le patient dans la liste déroulante. 5. L'assistant sélectionne la date et les horaires du rendez-vous. 6. L'assistant saisit le motif de consultation (ex: "Consultation générale", "Suivi traitement"). 7. L'assistant clique sur "Créer". 8. Le système valide les données (patient sélectionné, date dans le futur, heure fin > heure début, motif non vide). 9. Le système vérifie la disponibilité du médecin via GET /api/rendezvous/check-disponibilite?date=YYYY-MM-DD&heureDe 10. Si le créneau est disponible, le système envoie une requête POST /api/rendezvous avec les données. 11. Le backend enregistre le rendez-vous avec statut "CONFIRME" et référence l'assistant comme créateur. 12. Le système envoie un email automatique au patient avec les détails du rendez-vous. 13. Le système affiche un message de succès "Rendez-vous créé avec succès, le patient a été notifié par email". 14. Le système rafraîchit le calendrier de rendez-vous.

Cas d'utilisation	Créer un Rendez-vous
Scénario alternatif	1. Aucun patient sélectionné → affichage d'un message "Veuillez sélectionner un patient". 2. La date sélectionnée est dans le passé → affichage d'un message "La date doit être dans le futur". 3. L'heure de fin est antérieure ou égale à l'heure de début → affichage d'un message "L'heure de fin doit être postérieure à l'heure de début". 4. Le créneau horaire est déjà occupé (conflit avec un autre rendez-vous) → affichage d'un message "Ce créneau est déjà réservé, veuillez choisir une autre heure". 5. Le motif est vide → affichage d'un message "Le motif de consultation est obligatoire". 6. Erreur serveur lors de la création → affichage d'un message "Erreur lors de la création, veuillez réessayer". 7. Erreur d'envoi d'email → le rendez-vous est créé mais un message avertit "Rendez-vous créé mais échec de l'envoi de l'email au patient".

Description textuelle du cas d'utilisation "Créer une Facture (Restreint)" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Créer une Facture" avec restrictions pour l'assistant.

Table 4.5: Description textuelle du cas d'utilisation "Créer une Facture - Assistant"

Cas d'utilisation	Créer une Facture (Restreint)
Acteur	Assistant

Cas d'utilisation	Créer une Facture (Restreint)
Précondition	1. L'assistant est authentifié avec le rôle "ASSISTANT". 2. Un patient existe dans le système. 3. L'assistant a accès à la section "Factures" en mode création uniquement.
Post-condition	1. Une nouvelle facture est créée avec le statut "IMPAYEE". 2. La facture contient : patient, montant total, description des actes, date d'émission. 3. Un numéro de facture unique est généré automatiquement. 4. La facture est créée par l'assistant mais associée au médecin propriétaire du cabinet. 5. **RESTRICTION :** L'assistant ne peut PAS modifier, supprimer ni enregistrer de paiement sur cette facture.

Cas d'utilisation	Créer une Facture (Restreint)
Scénario principal	1. L'assistant accède à la section "Factures" (/dashboard/assistant/factures). 2. L'assistant clique sur "Créer une Facture". 3. Le système affiche le formulaire de création avec les champs : sélection patient, montant, description des actes. 4. L'assistant sélectionne le patient dans la liste déroulante. 5. L'assistant saisit le montant et la description (ex: "Consultation + Ordonnance"). 6. L'assistant clique sur "Créer". 7. Le système valide les données (montant > 0, patient sélectionné, description non vide). 8. Le système génère un numéro de facture unique (format: FAC-YYYY-XXXX). 9. Le système envoie une requête POST /api/factures avec les données et l'ID de l'assistant créateur. 10. Le backend enregistre la facture avec statut "IMPAYEE", date d'émission, et référence le médecin propriétaire + assistant créateur. 11. Le système affiche un message de succès "Facture créée avec succès". 12. Le système affiche la liste des factures en mode lecture seule (pas de boutons modifier/supprimer/payer pour l'assistant).

Cas d'utilisation	Créer une Facture (Restreint)
Scénario alternatif	1. Aucun patient sélectionné → affichage d'un message "Veuillez sélectionner un patient". 2. Le montant est inférieur ou égal à 0 → affichage d'un message "Le montant doit être supérieur à 0". 3. La description est vide → affichage d'un message "La description est obligatoire". 4. L'assistant tente d'accéder à /api/factures/id/paiement (enregistrer paiement) → erreur 403 Forbidden "Accès non autorisé pour le rôle ASSISTANT". 5. Erreur serveur lors de la création → affichage d'un message "Erreur lors de la création, veuillez réessayer".

Description textuelle du cas d'utilisation "Annuler un Rendez-vous" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Annuler un Rendez-vous". Ce scénario permet à l'assistant d'annuler un rendez-vous existant.

Table 4.6: Description textuelle du cas d'utilisation "Annuler un Rendez-vous"

Cas d'utilisation	Annuler un Rendez-vous
Acteur	Assistant
Précondition	1. L'assistant est authentifié avec le rôle "ASSISTANT". 2. Un rendez-vous existe avec le statut "CONFIRME". 3. L'assistant a accès à la section "Rendez-vous".

Cas d'utilisation	Annuler un Rendez-vous
Post-condition	1. Le statut du rendez-vous passe de "CONFIRME" à "ANNULE". 2. Le créneau horaire est libéré et redevient disponible. 3. Une notification par email est envoyée automatiquement au patient pour l'informer de l'annulation. 4. Le rendez-vous reste visible dans l'historique mais marqué comme "ANNULE".

Cas d'utilisation	Annuler un Rendez-vous
Scénario principal	1. L'assistant accède à la section "Rendez-vous" et visualise le calendrier. 2. L'assistant sélectionne un rendez-vous avec le statut "CONFIRME". 3. L'assistant clique sur "Annuler le Rendez-vous". 4. Le système affiche une boîte de dialogue de confirmation "Êtes-vous sûr de vouloir annuler ce rendez-vous ?". 5. L'assistant confirme l'annulation. 6. Le système envoie une requête PUT /api/rendezvous/id/annuler. 7. Le backend met à jour le statut du rendez-vous à "ANNULE". 8. Le backend envoie un email automatique au patient avec le message "Votre rendez-vous du [DATE] à [HEURE] a été annulé". 9. Le système affiche un message de succès "Rendez-vous annulé avec succès, le patient a été notifié par email". 10. Le système rafraîchit le calendrier (le rendez-vous apparaît en gris/barré ou disparaît selon l'affichage).

Cas d'utilisation	Annuler un Rendez-vous
Scénario alternatif	1. L'assistant annule la confirmation dans la boîte de dialogue → le rendez-vous reste inchangé. 2. Le rendez-vous est déjà annulé → affichage d'un message "Ce rendez-vous est déjà annulé". 3. Erreur serveur lors de l'annulation → affichage d'un message "Erreur lors de l'annulation, veuillez réessayer". 4. Erreur d'envoi d'email → le rendez-vous est annulé mais un message avertit "Rendez-vous annulé mais échec de l'envoi de l'email au patient".

4.4 Conception

Dans cette section, nous présentons les diagrammes de conception du sprint 2, notamment les diagrammes de séquence et le diagramme de classes étendu.

4.4.1 Diagrammes de séquence

Diagramme de séquence "Créer un Rendez-vous avec Vérification Disponibilité" :

La figure 4.2 illustre le diagramme de séquence de création d'un rendez-vous par l'assistant, incluant la vérification automatique de la disponibilité du médecin et l'envoi d'email au patient.

images/sequence-creer-rendezvous.png

Figure 4.2: Diagramme de séquence - Créer un Rendez-vous

Diagramme de séquence "Contrôle d'Accès RBAC - Tentative d'Accès Interdit" :

La figure 4.3 illustre le diagramme de séquence montrant le fonctionnement du contrôle d'accès basé sur les rôles (RBAC) lorsqu'un assistant tente d'accéder à un endpoint interdit (dossiers médicaux).

images/sequence-rbac-assistant.png

Figure 4.3: Diagramme de séquence - Contrôle d'Accès RBAC

4.4.2 Diagramme de classes du sprint 2

Le diagramme de classes du sprint 2, présenté dans la figure 4.4, illustre les relations entre les entités principales du système avec focus sur les restrictions d'accès pour l'assistant.

images/class-diagram-sprint2.png

Figure 4.4: Diagramme de classes - Sprint 2 avec RBAC

Ajouts et modifications pour le Sprint 2 :

- **RendezVous :** Ajout de l'attribut *assistantCreateur* (ManyToOne avec User) pour tracer quel assistant a créé le rendez-vous. Ajout de méthodes de validation : *checkDisponibilite()*, *annuler()*.
- **Facture :** Ajout de l'attribut *assistantCreateur* (ManyToOne avec User nullable) pour distinguer les factures créées par assistant vs médecin.
- **User :** Ajout de l'énumération *Role* avec valeurs {MEDECIN, ASSISTANT, PATIENT}. Ajout de méthodes de contrôle d'accès : *canAccessDossiers()*, *canManageAssistants()*, *canAccessRapportsFinanciers()*.
- **EmailService :** Nouvelle classe de service pour l'envoi d'emails automatiques. Méthodes : *sendRendezVousConfirmation(Patient, RendezVous)*, *sendRendezVousAnnulation(Patie RendezVous)*.

4.5 Réalisation

Cette section présente les captures d'écran des principales fonctionnalités développées pour l'espace assistant.

4.5.1 Authentification

Figure 4.5 : Page de connexion assistant

Figure 4.5: Page de connexion identique pour tous les utilisateurs (médecin/assistant/patient)

4.5.2 Tableau de bord assistant

Figure 4.6 : Dashboard assistant avec menu restreint

images/dashboard-assistant.png

Figure 4.6: Dashboard assistant avec menu latéral restreint (Patients, Rendez-vous, Factures uniquement)

4.5.3 Gestion des patients

Figure 4.7 : Liste des patients

images/liste-patients-assistant.png

Figure 4.7: Liste des patients avec nom, prénom, téléphone, email et actions (consulter, modifier)

Figure 4.8 : Formulaire de création de patient

images/creer-patient-assistant.png

Figure 4.8: Formulaire avec champs nom, prénom, date de naissance, téléphone, email, adresse

4.5.4 Gestion des rendez-vous

Figure 4.9 : Calendrier de rendez-vous

images/calendrier-assistant.png

Figure 4.9: Calendrier hebdomadaire avec rendez-vous, vérification automatique des conflits

Figure 4.10 : Création de rendez-vous avec vérification disponibilité

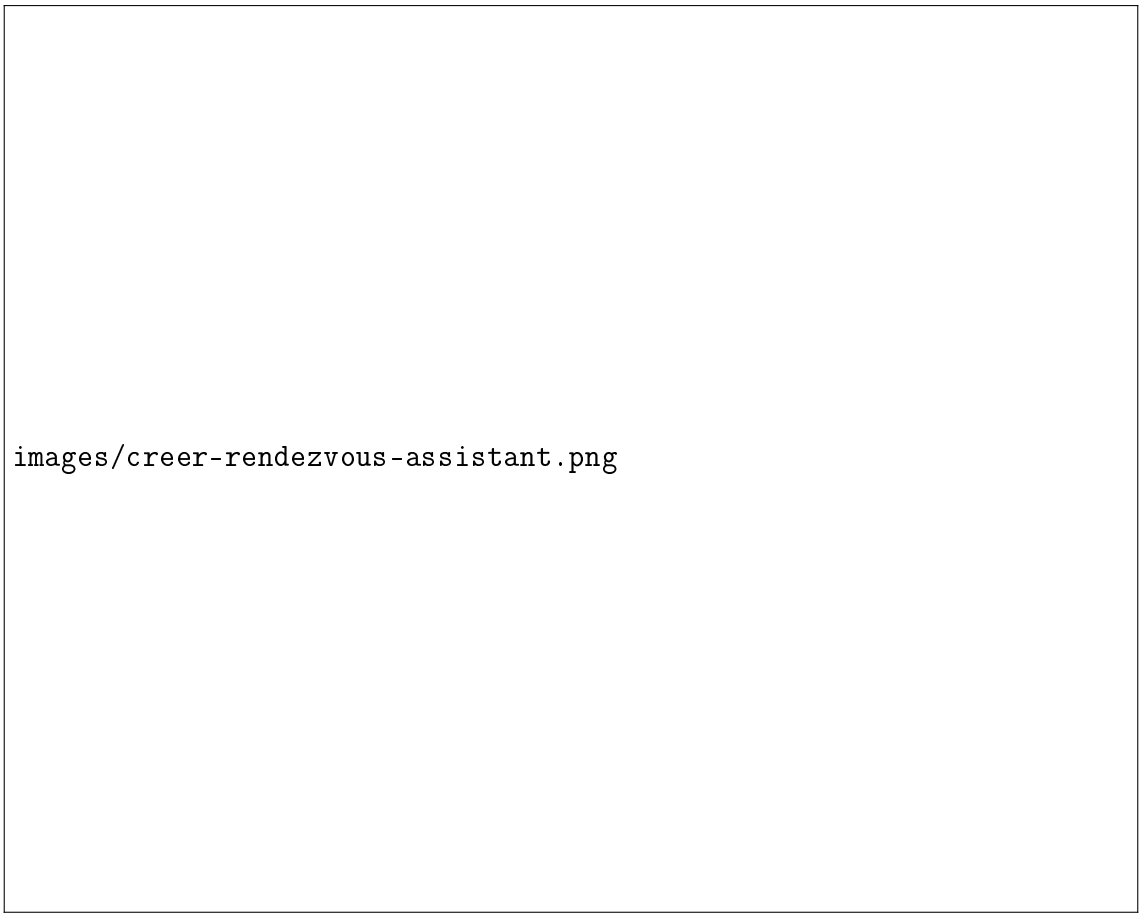

Figure 4.10: Formulaire de création avec sélection patient, date, horaires et motif + vérification temps réel de disponibilité

Figure 4.11 : Notification de conflit horaire

Figure 4.11: Message d'erreur affiché lorsque le créneau sélectionné est déjà occupé

4.5.5 Gestion des factures (restreint)

Figure 4.12 : Liste des factures en lecture seule

Figure 4.12: Liste des factures avec numéro, patient, montant, statut - SANS boutons modifier/supprimer/payer

Figure 4.13 : Création de facture

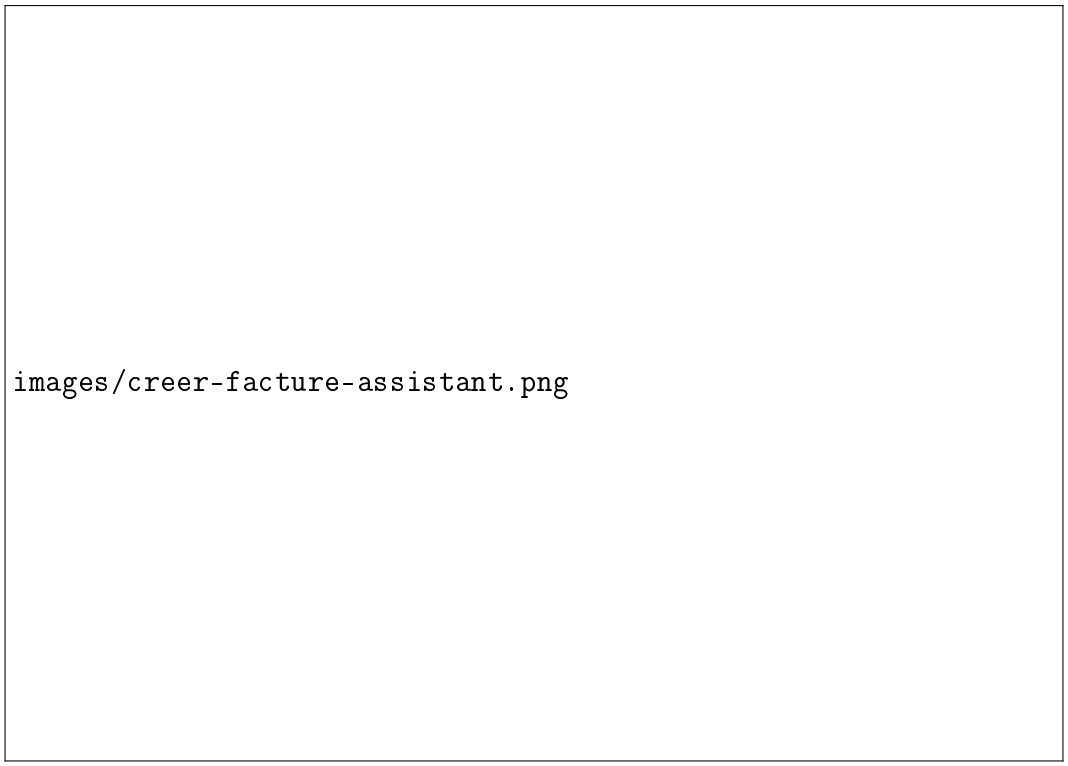

Figure 4.13: Formulaire de création de facture accessible à l'assistant

4.5.6 Contrôle d'accès RBAC

Figure 4.14 : Tentative d'accès interdit

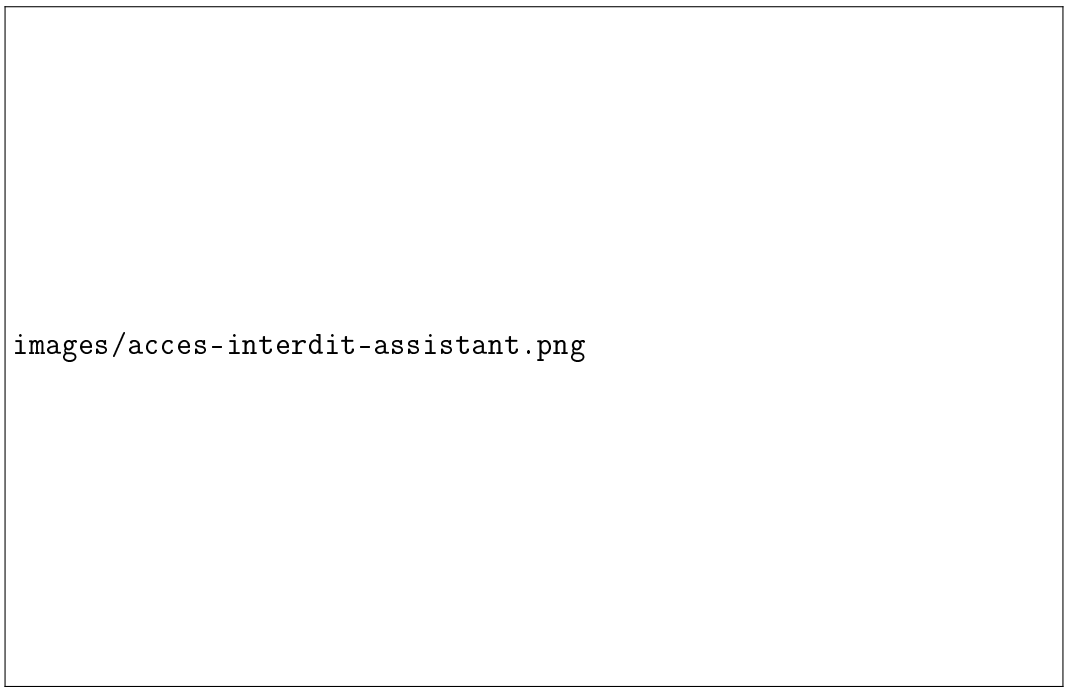

Figure 4.14: Message d'erreur affiché lorsque l'assistant tente d'accéder aux dossiers médicaux

4.6 Implémentation du contrôle d'accès (RBAC)

L'une des fonctionnalités clés de ce sprint est le système de contrôle d'accès basé sur les rôles (RBAC - Role-Based Access Control). Cette section détaille l'implémentation technique de ce système.

4.6.1 Backend - Spring Security

Le backend utilise Spring Security avec JWT pour implémenter le RBAC. Voici les principaux composants :

Configuration des endpoints protégés :

```
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) {
        http.authorizeHttpRequests(auth -> auth
            .requestMatchers("/api/auth/**").permitAll()
            .requestMatchers("/api/dossiers/**").hasRole("MEDECIN")
            .requestMatchers("/api/assistants/**").hasRole("MEDECIN")
            .requestMatchers("/api/rapports/**").hasRole("MEDECIN")
            .requestMatchers("/api/factures/*/paiement").hasRole("MEDECIN")
            .requestMatchers("/api/patients/**").hasAnyRole("MEDECIN", "ASSISTANT")
            .requestMatchers("/api/rendezvous/**").hasAnyRole("MEDECIN", "ASSISTANT")
            .requestMatchers("/api/factures").hasAnyRole("MEDECIN", "ASSISTANT")
            .anyRequest().authenticated()
        );
    }
}
```

Annotations sur les contrôleurs :

```
@RestController
@RequestMapping("/api/dossiers")
@PreAuthorize("hasRole('MEDECIN')")
public class DossierController {
    // Accessible uniquement par le médecin
}

@RestController
@RequestMapping("/api/factures")
public class FactureController {
    @PostMapping
    @PreAuthorize("hasAnyRole('MEDECIN', 'ASSISTANT')")
    public Facture creerFacture(@RequestBody Facture facture) {
        // Création autorisée pour médecin ET assistant
    }

    @PutMapping("/{id}/paiement")
    @PreAuthorize("hasRole('MEDECIN')")
    public Facture enregistrerPaiement(@PathVariable Long id, ...) {
        // Paiement autorisé uniquement pour le médecin
    }
}
```

4.6.2 Frontend - Next.js

Le frontend implémente également des contrôles pour améliorer l'expérience utilisateur et éviter les requêtes inutiles :

RoleGuard Component :

```
export function RoleGuard({ children, allowedRoles }) {
    const { user } = useAuth();
    if (!user || !allowedRoles.includes(user.role)) {
        return <AccessDenied />;
    }
    return <>{children}</>;
}
```

Affichage conditionnel des menus :

```
{user.role === 'MEDECIN' && (
    <>
        <NavItem href="/dashboard/dossiers">Dossiers Médicaux</NavItem>
        <NavItem href="/dashboard/assistants">Assistants</NavItem>
        <NavItem href="/dashboard/rapports">Rapports Financiers</NavItem>
    </>
)}
```

4.7 CONCLUSION

Ce chapitre a présenté la réalisation du sprint 2 focalisé sur l'espace assistant avec un système robuste de contrôle d'accès basé sur les rôles (RBAC). Nous avons développé les fonctionnalités essentielles permettant aux assistants de gérer les patients (création, modification, consultation), de planifier les rendez-vous avec vérification automatique de disponibilité et notifications par email, et de créer des factures avec restrictions strictes.

L'implémentation du RBAC garantit la sécurité et la confidentialité des données médicales en interdisant aux assistants l'accès aux dossiers médicaux, documents médicaux, rapports financiers et gestion d'autres assistants. Cette architecture de sécurité multi-niveaux (backend + frontend) offre une protection robuste contre les accès non autorisés.

Le prochain chapitre détaillera le sprint 3 dédié à l'espace patient, avec inscription sécurisée par email, consultation de dossiers médicaux, téléchargement de documents et interaction avec le chatbot d'intelligence artificielle.

Chapter

5 CHAPITRE 5 : Sprint 3 - Gestion de l'espace Patient et Chatbot IA

Contents

5.1 INTRODUCTION 139

5.2 Backlog du sprint 3 139

5.3 Spécifications fonctionnelles 141

5.3.1 Diagramme de cas d'utilisation du sprint 3 141

5.3.2 Descriptions textuelles 141

5.4 Conception 152

5.4.1 Diagrammes de séquence 152

5.4.2 Architecture du Chatbot IA 153

5.4.3 Diagramme de classes du sprint 3 154

5.5 Réalisation 155

5.5.1 Inscription et vérification 155

5.5.2 Tableau de bord patient 155

5.5.3 Consultation de dossiers médicaux 155

5.5.4 Chatbot médical IA 156

5.6 Implémentation technique du Chatbot 156

5.6.1 Backend - Service OpenAI 156

5.6.2 Frontend - Composant Chatbot 159

5.7 CONCLUSION 162

5.1 INTRODUCTION

Dans ce chapitre, nous présentons le troisième et dernier sprint intitulé "Gestion de l'espace Patient et Chatbot IA". Après avoir développé les espaces médecin (sprint 1) et assistant (sprint 2), nous nous concentrons maintenant sur l'interface destinée aux patients, avec une fonctionnalité innovante d'intelligence artificielle. Ce sprint permet aux patients de s'inscrire de manière sécurisée, de consulter leurs dossiers médicaux, de télécharger leurs documents et d'interagir avec un chatbot médical intelligent utilisant l'API OpenAI. Nous allons commencer par la présentation du backlog du sprint 3, suivie de la spécification fonctionnelle, de la conception avec diagrammes UML, et enfin nous illustrerons les fonctionnalités développées à travers des captures d'écran.

5.2 Backlog du sprint 3

Le tableau **??** présente l'ensemble des user stories sélectionnées pour le sprint 3, focalisées sur l'espace patient et le chatbot IA.

Table 5.1: Backlog du sprint 3 - Espace Patient et Chatbot IA

Fonctionnalités	User Stories	Priorités	Estimation
Inscription Sécurisée	En tant que patient, je veux m'inscrire avec email, nom, prénom et mot de passe	Élevée	4 jours
	En tant que patient, je veux recevoir un email de vérification après inscription	Élevée	2 jours
	En tant que patient, je veux cliquer sur le lien de vérification pour activer mon compte	Élevée	2 jours
Authentification	En tant que patient, je veux m'authentifier pour accéder à mon espace personnel	Élevée	2 jours
	En tant que patient, je veux me déconnecter pour quitter ma session en toute sécurité	Moyenne	1 jour

Fonctionnalités	User Stories	Priorités	Estimation
Gestion Rendez-vous	En tant que patient, je veux prendre un rendez-vous en ligne avec sélection de date et créneau	Élevée	5 jours
	En tant que patient, je veux consulter mes prochains rendez-vous avec détails (date, médecin, motif)	Élevée	3 jours
	En tant que patient, je veux annuler un rendez-vous si nécessaire	Moyenne	2 jours
Consultation Dossiers Médicaux	En tant que patient, je veux consulter l'historique complet de mes dossiers médicaux	Élevée	4 jours
	En tant que patient, je veux voir les détails de chaque dossier (diagnostic, traitement, notes, allergies, antécédents)	Élevée	3 jours
	En tant que patient, je veux consulter la liste des documents associés à chaque dossier	Moyenne	2 jours
	En tant que patient, je veux télécharger mes documents médicaux (ordonnances, analyses) en format PDF	Élevée	3 jours
Chatbot Intelligence Artificielle	En tant que patient, je veux accéder à un chatbot médical pour poser des questions générales	Élevée	5 jours
	En tant que patient, je veux recevoir des réponses intelligentes basées sur l'IA (OpenAI API)	Élevée	4 jours
	En tant que patient, je veux que le chatbot me rappelle de consulter un médecin pour les questions critiques	Moyenne	2 jours
Gestion Profil	En tant que patient, je veux consulter mes informations personnelles (nom, email, téléphone)	Moyenne	2 jours
	En tant que patient, je veux modifier mes informations personnelles (sauf données médicales)	Moyenne	2 jours

Chaque user story représente une fonctionnalité que le patient souhaite pouvoir réaliser dans le système. Les priorités sont classées en élevée, moyenne ou faible, reflétant l'importance relative de chaque user story. Le chatbot d'intelligence artificielle constitue l'innovation majeure de ce sprint, offrant une assistance médicale virtuelle accessible 24/7.

5.3 Spécifications fonctionnelles

Dans cette partie, nous présentons le diagramme de cas d'utilisation du sprint 3 ainsi que les descriptions textuelles détaillées.

5.3.1 Diagramme de cas d'utilisation du sprint 3

Le diagramme de cas d'utilisation du sprint 3, présenté dans la figure 5.1, illustre les besoins fonctionnels sous la forme d'interactions entre le patient et le système, incluant le chatbot IA.

Figure 5.1: Diagramme de cas d'utilisation du sprint 3 - Espace Patient

5.3.2 Descriptions textuelles

L'objectif de cette activité est de décrire textuellement les scénarios des cas d'utilisation. Il faut préciser comment chaque scénario commence, comment il se termine et comment le patient interagit avec l'application web.

Description textuelle du cas d'utilisation "S'inscrire - Patient" :

Le tableau **??** présente la description textuelle du cas d'utilisation "S'inscrire". Ce scénario permet à un nouveau patient de créer son compte avec vérification par email.

Table 5.2: Description textuelle du cas d'utilisation "S'inscrire - Patient"

Cas d'utilisation	S'inscrire - Patient
Acteur	Patient (nouveau)

Cas d'utilisation	S'inscrire - Patient
Précondition	1. Le système est en service. 2. Le patient ne possède pas encore de compte. 3. Le patient accède à la page publique d'inscription /register.
Post-condition	1. Un compte patient est créé avec le statut "NON_VERIFIE". 2. Le mot de passe est hashé avec BCrypt avant stockage. 3. Un email de vérification contenant un token unique est envoyé à l'adresse email du patient. 4. Le patient doit cliquer sur le lien de vérification pour activer son compte.

Cas d'utilisation	S'inscrire - Patient
Scénario principal	1. Le patient accède à la page d'inscription (/register). 2. Le système affiche le formulaire d'inscription avec les champs : email, nom, prénom, mot de passe, confirmation mot de passe. 3. Le patient remplit tous les champs obligatoires. 4. Le patient clique sur "S'inscrire". 5. Le système valide les données saisies (format email, force mot de passe, correspondance des mots de passe). 6. Le système vérifie que l'email correspond à un patient existant via GET /api/patients/check-email. 7. Le système envoie une requête POST /api/auth/register avec les données. 8. Le backend vérifie que l'email n'a pas déjà un compte associé. 9. Le backend hash le mot de passe avec BCrypt. 10. Le backend génère un token de vérification unique (UUID). 11. Le backend enregistre le compte avec statut "NON_VERIFIE" et le token. 12. Le backend envoie un email contenant le lien de vérification : https://cabinet.com/verify?token=XXXXX. 13. Le système affiche un message "Un email de vérification a été envoyé à votre adresse. Veuillez cliquer sur le lien pour activer votre compte".

Cas d'utilisation	S'inscrire - Patient
Scénario alternatif	1. Le patient laisse des champs obligatoires vides → affichage d'erreurs de validation. 2. Le format de l'email est invalide → affichage d'un message "Email invalide". 3. Les mots de passe ne correspondent pas → affichage d'un message "Les mots de passe ne correspondent pas". 4. Le mot de passe est trop faible (moins de 8 caractères) → affichage d'un message "Le mot de passe doit contenir au moins 8 caractères". 5. L'email ne correspond à aucun patient enregistré → affichage d'un message "Aucun patient trouvé avec cet email. Contactez le cabinet". 6. L'email possède déjà un compte actif → affichage d'un message "Un compte existe déjà avec cet email. Connectez-vous". 7. Erreur d'envoi d'email → affichage d'un message "Erreur lors de l'envoi de l'email de vérification, veuillez réessayer".

Description textuelle du cas d'utilisation "Vérifier Email" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Vérifier Email". Ce scénario active le compte patient après clic sur le lien de vérification.

Table 5.3: Description textuelle du cas d'utilisation "Vérifier Email"

Cas d'utilisation	Vérifier Email
Acteur	Patient (non vérifié)

Cas d'utilisation	Vérifier Email
Précondition	1. Le patient a créé un compte avec le statut "NON_VERIFIE". 2. Le patient a reçu un email avec un lien de vérification contenant un token unique.
Post-condition	1. Le statut du compte passe de "NON_VERIFIE" à "VERIFIE". 2. Le token de vérification est marqué comme utilisé. 3. Le patient peut se connecter à son espace personnel.
Scénario principal	1. Le patient clique sur le lien de vérification dans l'email reçu. 2. Le navigateur ouvre l'URL /verify?token=XXXXX. 3. Le système extrait le token de l'URL. 4. Le système envoie une requête GET /api/auth/verify?token=XXXXX. 5. Le backend recherche le compte associé au token. 6. Le backend vérifie que le token n'a pas expiré (validité 24h). 7. Le backend met à jour le statut du compte à "VERIFIE". 8. Le backend marque le token comme utilisé. 9. Le système affiche un message de succès "Votre compte a été vérifié avec succès. Vous pouvez maintenant vous connecter". 10. Le système redirige automatiquement vers /login après 3 secondes.

Cas d'utilisation	Vérifier Email
Scénario alternatif	1. Le token est invalide (n'existe pas en base) → affichage d'un message "Lien de vérification invalide". 2. Le token a expiré (> 24h) → affichage d'un message "Le lien de vérification a expiré. Veuillez vous réinscrire". 3. Le token a déjà été utilisé → affichage d'un message "Ce compte a déjà été vérifié. Connectez-vous". 4. Erreur serveur lors de la vérification → affichage d'un message "Erreur lors de la vérification, veuillez réessayer".

Description textuelle du cas d'utilisation "Consulter Dossiers Médicaux" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Consulter Dossiers Médicaux". Ce scénario permet au patient de consulter son historique médical complet.

Table 5.4: Description textuelle du cas d'utilisation "Consulter Dossiers Médicaux - Patient"

Cas d'utilisation	Consulter Dossiers Médicaux
Acteur	Patient
Précondition	1. Le patient est authentifié avec un compte vérifié. 2. Au moins un dossier médical existe pour ce patient.

Cas d'utilisation	Consulter Dossiers Médicaux
Post-condition	1. Le patient visualise l'historique complet de ses dossiers médicaux triés par date décroissante. 2. Le patient consulte les détails de chaque dossier : diagnostic, traitement, notes, allergies, antécédents. 3. Le patient consulte la liste des documents associés à chaque dossier.
Scénario principal	1. Le patient accède à la section "Mes Dossiers Médicaux" (/dashboard/patient/dossiers). 2. Le système envoie une requête GET /api/patients/patientId/dossiers. 3. Le backend vérifie que le patient authentifié demande bien SES propres dossiers (vérification ID JWT = ID patient). 4. Le backend récupère tous les dossiers médicaux associés au patient. 5. Le système affiche la liste chronologique des dossiers (du plus récent au plus ancien). 6. Le patient sélectionne un dossier dans la liste. 7. Le système affiche les détails complets du dossier dans un panneau déroulant ou modal. 8. Le patient consulte les informations : date de création, diagnostic, traitement prescrit, notes cliniques, allergies identifiées, antécédents médicaux. 9. Le patient consulte la liste des documents uploadés (ordonnances, analyses).

Cas d'utilisation	Consulter Dossiers Médicaux
Scénario alternatif	1. Aucun dossier médical n'existe pour ce patient → affichage d'un message "Aucun dossier médical disponible". 2. Le patient tente d'accéder aux dossiers d'un autre patient en modifiant l'URL → erreur 403 Forbidden "Accès non autorisé". 3. Erreur serveur lors de la récupération des dossiers → affichage d'un message "Erreur lors de la récupération des dossiers, veuillez réessayer".

Description textuelle du cas d'utilisation "Télécharger Document Médical" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Télécharger Document Médical". Ce scénario permet au patient de télécharger ses ordonnances et résultats d'analyses.

Table 5.5: Description textuelle du cas d'utilisation "Télécharger Document Médical"

Cas d'utilisation	Télécharger Document Médical
Acteur	Patient
Précondition	1. Le patient est authentifié avec un compte vérifié. 2. Le patient consulte un dossier médical contenant des documents.
Post-condition	1. Le fichier (ordonnance, analyse) est téléchargé sur l'ordinateur du patient en format PDF/JPG/PNG. 2. Le téléchargement est tracé dans les logs système pour audit.

Cas d'utilisation	Télécharger Document Médical
Scénario principal	1. Le patient consulte un dossier médical avec des documents associés. 2. Le système affiche la liste des documents avec nom de fichier et type (Ordonnance, Analyse, Imagerie). 3. Le patient clique sur le bouton "Télécharger" à côté d'un document. 4. Le système envoie une requête GET /api/documents/documentId/download. 5. Le backend vérifie que le document appartient bien à un dossier du patient authentifié. 6. Le backend récupère le fichier depuis le système de fichiers (uploads/dossier-id/). 7. Le backend envoie le fichier au frontend avec les headers appropriés (Content-Type, Content-Disposition: attachment). 8. Le navigateur télécharge automatiquement le fichier. 9. Le système enregistre un log : "Patient id a téléchargé le document documentId le date".
Scénario alternatif	1. Le patient tente de télécharger un document d'un autre patient → erreur 403 Forbidden "Accès non autorisé". 2. Le fichier n'existe plus sur le système de fichiers → affichage d'un message "Fichier introuvable, contactez le cabinet". 3. Erreur serveur lors du téléchargement → affichage d'un message "Erreur lors du téléchargement, veuillez réessayer".

Description textuelle du cas d'utilisation "Interagir avec Chatbot IA" :

Le tableau **??** présente la description textuelle du cas d'utilisation "Interagir avec Chatbot IA". Ce scénario permet au patient de poser des questions médicales générales à un assistant virtuel intelligent.

Table 5.6: Description textuelle du cas d'utilisation "Interagir avec Chatbot IA"

Cas d'utilisation	Interagir avec Chatbot IA
Acteur	Patient
Précondition	1. Le patient est authentifié avec un compte vérifié. 2. Le backend a accès à l'API OpenAI (clé API configurée).
Post-condition	1. Le patient reçoit une réponse générée par l'IA (OpenAI GPT-4) à sa question médicale. 2. L'historique de la conversation est sauvegardé en base de données. 3. Le patient peut consulter l'historique complet de ses conversations avec le chatbot.

Cas d'utilisation	Interagir avec Chatbot IA
Scénario principal	1. Le patient accède à la section "Chatbot Médical" (/dashboard/patient/chatbot). 2. Le système affiche une interface de chat avec historique des conversations précédentes. 3. Le patient saisit une question médicale dans le champ de texte (ex: "Quels sont les symptômes de la grippe ?"). 4. Le patient clique sur "Envoyer" ou appuie sur Entrée. 5. Le système affiche le message du patient dans le chat avec horodatage. 6. Le système envoie une requête POST /api/chatbot/message avec le texte de la question. 7. Le backend ajoute un contexte système à la requête OpenAI : "Tu es un assistant médical virtuel. Réponds aux questions médicales générales de manière claire. Pour les symptômes graves, recommande toujours de consulter un médecin.". 8. Le backend appelle l'API OpenAI (modèle GPT-4) avec la question + contexte. 9. L'API OpenAI génère une réponse intelligente et contextuelle. 10. Le backend sauvegarde la conversation en base : patientId, question, réponse, timestamp. 11. Le backend retourne la réponse au frontend. 12. Le système affiche la réponse du chatbot dans le chat avec horodatage. 13. Le patient peut continuer la conversation en posant une nouvelle question.

Cas d'utilisation	Interagir avec Chatbot IA
Scénario alternatif	1. Le patient saisit une question vide → affichage d'un message "Veuillez saisir une question". 2. La question contient des mots-clés critiques (douleur thoracique, difficulté respiratoire, saignement important) → le chatbot répond avec un avertissement : "Vos symptômes nécessitent une consultation médicale urgente. Contactez immédiatement un médecin ou les urgences". 3. Erreur API OpenAI (quota dépassé, service indisponible) → affichage d'un message "Le chatbot est temporairement indisponible, veuillez réessayer plus tard". 4. Timeout de l'API OpenAI (> 30 secondes) → affichage d'un message "Le chatbot met trop de temps à répondre, veuillez réessayer".

5.4 Conception

Dans cette section, nous présentons les diagrammes de conception du sprint 3, notamment les diagrammes de séquence et l'architecture du chatbot IA.

5.4.1 Diagrammes de séquence

Diagramme de séquence "Inscription avec Vérification Email" :

La figure 5.2 illustre le diagramme de séquence complet du processus d'inscription sécurisée avec vérification par email.

Figure 5.2: Diagramme de séquence - Inscription Patient avec Vérification Email

Diagramme de séquence "Interaction avec Chatbot OpenAI" :

La figure 5.3 illustre le diagramme de séquence montrant l'interaction entre le patient, le frontend, l'API backend et l'API OpenAI pour le chatbot.

Figure 5.3: Diagramme de séquence - Interaction Chatbot IA

5.4.2 Architecture du Chatbot IA

Le chatbot médical repose sur une architecture moderne intégrant l'API OpenAI :

Figure 5.4: Architecture du Chatbot avec OpenAI API

Composants de l'architecture :

- **Frontend (Next.js) :** Interface de chat interactive avec affichage en temps réel des messages, zone de saisie, historique des conversations.
- **Backend (Spring Boot) :** API REST exposant l'endpoint POST /api/chatbot/message, gestion de l'authentification JWT, sauvegarde des conversations en base.
- **OpenAI Service :** Classe Java gérant les appels à l'API OpenAI, formatage des requêtes avec contexte système, gestion des erreurs et timeouts.
- **OpenAI API :** Service externe (GPT-4) analysant les questions et générant des réponses médicales intelligentes.
- **Base de données :** Table ConversationChatbot stockant id, patientId, question, réponse, timestamp pour traçabilité et historique.

Prompt système envoyé à OpenAI :

```
Vous êtes un assistant médical virtuel pour un cabinet médical.
Votre rôle est de répondre aux questions médicales générales
des patients de manière claire, précise et bienveillante.

RÈGLES IMPORTANTES :
1. Pour les questions générales (symptômes communs, prévention,
   hygiène), fournissez des informations claires et pédagogiques.
```

```
2. Pour les symptômes graves (douleur thoracique, difficulté
   respiratoire, saignement important), recommandez TOUJOURS
   une consultation médicale urgente.
3. Ne posez JAMAIS de diagnostic médical précis.
4. Encouragez toujours la consultation avec un médecin pour
   un avis personnalisé.
5. Restez dans le domaine médical général, n'abordez pas
   de sujets hors contexte.

Question du patient : [QUESTION]
```

5.4.3 Diagramme de classes du sprint 3

Le diagramme de classes du sprint 3, présenté dans la figure 5.5, illustre les nouvelles entités liées à l'espace patient et au chatbot.

Figure 5.5: Diagramme de classes - Sprint 3

Nouvelles classes pour le Sprint 3 :

- **Patient (étendu) :** Ajout des attributs *emailVerifie* (boolean), *tokenVerification* (String UUID), *dateTokenExpiration* (DateTime). Ajout de la méthode *verifierEmail(token)*.
- **ConversationChatbot :** Nouvelle entité avec attributs id, patientId (ManyToOne), question (Text), réponse (Text), timestamp (DateTime). Représente l'historique des interactions avec le chatbot.
- **Document (étendu) :** Ajout de la méthode *download()* pour tracer les téléchargements.
- **OpenAIService :** Nouvelle classe de service avec méthodes *sendQuestion(question, context)*, *parseResponse(apiResponse)*, *handleErrors()*.

5.5 Réalisation

Cette section présente les captures d'écran des principales fonctionnalités développées pour l'espace patient.

5.5.1 Inscription et vérification

Figure 5.6 : Page d'inscription patient

Figure 5.6: Formulaire d'inscription avec email, nom, prénom, mot de passe, confirmation

Figure 5.7 : Email de vérification

Figure 5.7: Email envoyé au patient avec lien de vérification unique

Figure 5.8 : Page de confirmation de vérification

Figure 5.8: Message de succès après vérification, redirection automatique vers login

5.5.2 Tableau de bord patient

Figure 5.9 : Dashboard patient

Figure 5.9: Dashboard avec menu latéral (Mes Dossiers, Chatbot Médical, Profil)

5.5.3 Consultation de dossiers médicaux

Figure 5.10 : Liste des dossiers médicaux

Figure 5.10: Historique chronologique des dossiers avec date, diagnostic, traitement

Figure 5.11 : Détails d'un dossier médical

Figure 5.11: Affichage complet : diagnostic, traitement, notes, allergies, antécédents, documents

Figure 5.12 : Téléchargement de document

Figure 5.12: Liste des documents avec boutons de téléchargement (Ordonnance, Analyse)

5.5.4 Chatbot médical IA

Figure 5.13 : Interface du chatbot

Figure 5.13: Interface de chat avec historique des conversations et zone de saisie

Figure 5.14 : Conversation avec le chatbot

Figure 5.14: Exemple de conversation : question sur symptômes de grippe avec réponse détaillée de l'IA

Figure 5.15 : Réponse critique du chatbot

Figure 5.15: Réponse du chatbot pour symptômes graves : recommandation urgente de consulter un médecin

5.6 Implémentation technique du Chatbot

Cette section détaille l'implémentation technique du chatbot avec intégration OpenAI.

5.6.1 Backend - Service OpenAI

Classe OpenAIService.java :

```
@Service
public class OpenAIService {
    @Value("${openai.api.key}")
```

```
    private String apiKey;

    private static final String API_URL =
        "https://api.openai.com/v1/chat/completions";

    public String generateResponse(String question) {
        RestTemplate restTemplate = new RestTemplate();

        // Créer le contexte système
        String systemPrompt = "Vous êtes un assistant médical...";

        // Préparer la requête OpenAI
        Map<String, Object> request = Map.of(
            "model", "gpt-4",
            "messages", List.of(
                Map.of("role", "system", "content", systemPrompt),
                Map.of("role", "user", "content", question)
            ),
            "max_tokens", 500,
            "temperature", 0.7
        );

        // Headers avec API Key
        HttpHeaders headers = new HttpHeaders();
        headers.set("Authorization", "Bearer " + apiKey);
        headers.setContentType(MediaType.APPLICATION_JSON);

        HttpEntity<Map<String, Object>> entity =
            new HttpEntity<>(request, headers);
```

```
        // Appel API OpenAI
        ResponseEntity<Map> response = restTemplate.exchange(
            API_URL, HttpMethod.POST, entity, Map.class);

        // Extraire la réponse
        Map<String, Object> body = response.getBody();
        List<Map<String, Object>> choices =
            (List) body.get("choices");
        Map<String, Object> message =
            (Map) choices.get(0).get("message");
        return (String) message.get("content");
    }
}
```

Contrôleur ChatbotController.java :

```
@RestController
@RequestMapping("/api/chatbot")
public class ChatbotController {
    @Autowired
    private OpenAIService openAIService;

    @Autowired
    private ConversationRepository conversationRepo;

    @PostMapping("/message")
    @PreAuthorize("hasRole('PATIENT')")
    public ResponseEntity<Map<String, String>> sendMessage(
        @RequestBody Map<String, String> request,
        Authentication authentication) {
```

```
        String question = request.get("question");
        Long patientId = ((UserPrincipal)
            authentication.getPrincipal()).getId();

        // Appeler OpenAI
        String response = openAIService.generateResponse(question);

        // Sauvegarder la conversation
        ConversationChatbot conversation = new ConversationChatbot();
        conversation.setPatientId(patientId);
        conversation.setQuestion(question);
        conversation.setReponse(response);
        conversation.setTimestamp(LocalDateTime.now());
        conversationRepo.save(conversation);

        return ResponseEntity.ok(Map.of("response", response));
    }
}
```

5.6.2 Frontend - Composant Chatbot

Composant ChatbotPage.tsx :

```
export default function ChatbotPage() {
    const [messages, setMessages] = useState([]);
    const [input, setInput] = useState('');
    const [loading, setLoading] = useState(false);

    const sendMessage = async () => {
```

```
if (!input.trim()) return;

// Ajouter le message de l'utilisateur
const userMessage = { role: 'user', content: input };
setMessages([...messages, userMessage]);
setInput('');
setLoading(true);

try {
    const response = await fetch('/api/chatbot/message', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${getToken()}`
        },
        body: JSON.stringify({ question: input })
    });

    const data = await response.json();

    // Ajouter la réponse du chatbot
    const botMessage = {
        role: 'assistant',
        content: data.response
    };
    setMessages(prev => [...prev, botMessage]);
} catch (error) {
    console.error('Erreur chatbot:', error);
} finally {
```

```
            setLoading(false);
        }
    };

    return (
        <div className="chatbot-container">
            <div className="messages-list">
                {messages.map((msg, index) => (
                    <div key={index} className={`message ${msg.role}`}>
                        <p>{msg.content}</p>
                    </div>
                ))}
                {loading && <LoadingIndicator />}
            </div>
            <div className="input-area">
                <input
                    value={input}
                    onChange={(e) => setInput(e.target.value)}
                    onKeyPress={(e) => e.key === 'Enter' && sendMessage()}
                    placeholder="Posez votre question médicale..."
                />
                <button onClick={sendMessage}>Envoyer</button>
            </div>
        </div>
    );
}
```

5.7 CONCLUSION

Ce chapitre a présenté la réalisation du sprint 3 focalisé sur l'espace patient et l'innovation majeure du chatbot médical intelligent. Nous avons développé un système d'inscription sécurisée avec vérification par email utilisant des tokens uniques à durée limitée, garantissant l'authenticité des comptes patients. La consultation des dossiers médicaux permet aux patients d'accéder à leur historique complet en toute sécurité, avec possibilité de télécharger leurs documents médicaux en respectant le RGPD.

L'intégration du chatbot basé sur l'API OpenAI GPT-4 constitue l'apport technologique majeur de ce sprint. Ce chatbot offre une assistance médicale virtuelle accessible 24/7, capable de répondre aux questions générales des patients tout en les orientant vers une consultation médicale pour les symptômes graves. L'historique complet des conversations est sauvegardé pour traçabilité et amélioration continue du service.

L'ensemble des trois sprints (espace médecin, assistant et patient) forme maintenant un système complet de gestion de cabinet médical moderne, sécurisé et innovant, intégrant les dernières avancées de l'intelligence artificielle au service de la santé. Le chapitre suivant présentera l'environnement de développement, les outils utilisés, les tests effectués et les perspectives d'évolution du système.

Chapter

6 CHAPITRE 6 : Environnement de développ et Conclusion générale

Contents

6.1 INTRODUCTION 164

6.2 Environnement de développement 164

6.2.1 Environnement matériel 164

6.2.2 Environnement logiciel 165

6.2.3 Architecture technique globale 171

6.3 Tests et validation 172

6.3.1 Tests backend 172

6.3.2 Tests frontend 173

6.3.3 Tests de sécurité 173

6.4 Difficultés rencontrées et solutions 174

6.4.1 Intégration OpenAI API 174

6.4.2 Gestion des fichiers 174

6.4.3 Contrôle d'accès RBAC 174

6.5 Apports du projet 175

6.5.1 Apports techniques 175

6.5.2 Apports méthodologiques 175

6.5.3 Apports personnels 175

6.6 Perspectives d'évolution 176

6.6.1 Améliorations fonctionnelles 176

6.6.2 Améliorations techniques 176

6.6.3 Conformité et sécurité 177

6.7 CONCLUSION GÉNÉRALE 177

6.1 INTRODUCTION

Ce dernier chapitre de notre projet est dédié à la phase de clôture. Nous y présenterons les technologies matérielles et logicielles utilisées, ainsi que les langages et frameworks employés pour la réalisation du projet. De plus, nous mettrons en avant les outils de développement et les bonnes pratiques adoptées. Enfin, nous conclurons par une synthèse générale du projet et les perspectives d'évolution.

6.2 Environnement de développement

Dans cette partie, nous allons présenter l'environnement de travail matériel et logiciel utilisé dans la réalisation de notre projet de plateforme de gestion de cabinet médical.

6.2.1 Environnement matériel

Pendant la réalisation de notre application, nous avons utilisé des équipements ayant les caractéristiques présentées dans le tableau **??**.

Table 6.1: Description des équipements de développement

Caractéristique	Ordinateur
Modèle	MSI
Mémoire RAM	24 GB
Système d'exploitation	Microsoft Windows 10 Professionnel
Carte Graphique	NVIDIA GeForce GTX 1650 Ti
Processeur	10th Gen Intel(R) Core(TM) i7-10750H

6.2.2 Environnement logiciel

Dans cette partie, nous allons présenter l'environnement logiciel utilisé dans le développement de notre application full-stack moderne.

6.2.2.1 Outils et frameworks de développement

Commençons par les outils et frameworks de développement adoptés :

- **Next.js :** Next.js est un framework React open source développé par Vercel. Il permet de créer des applications web modernes avec rendu côté serveur (SSR), génération de sites statiques (SSG) et App Router pour une navigation optimisée. Next.js simplifie le développement d'applications performantes grâce à des fonctionnalités intégrées comme l'optimisation automatique des images, le code splitting, et le support TypeScript natif. Sa structure basée sur des pages facilite l'organisation du code et améliore le référencement (SEO). Le framework est idéal pour développer des applications web évolutives avec d'excellentes performances. [**?**]

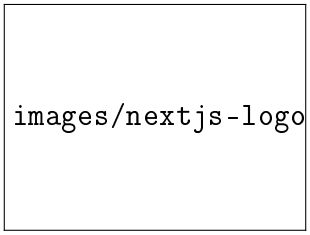

Figure 6.1: Logo Next.js

- **TypeScript :** TypeScript est un langage de programmation développé par Microsoft en 2012. Son ambition principale est d'améliorer la productivité de développement d'applications complexes. C'est un langage open source, développé comme un sur-ensemble de JavaScript. Ce qu'il faut comprendre, c'est que tout code valide en JavaScript l'est également en TypeScript. Cependant, ce langage introduit des fonctionnalités optionnelles comme le typage statique et la programmation orientée objet. Pour bénéficier de ces fonctionnalités, aucune librairie n'est requise. Il suffit d'utiliser l'outil de compilation de TypeScript pour

le transpiler (compiler le code source d'un langage en un autre langage) en JavaScript. Ainsi, le code exécuté sera un équivalent JavaScript du code TypeScript compilé. [**?**]

Figure 6.2: Logo TypeScript

- **Spring Boot :** Le Spring Framework est une infrastructure open source d'entreprise largement utilisée pour créer des applications autonomes de production fonctionnant sur la machine virtuelle Java (JVM). Spring Boot, un outil associé, accélère et simplifie le développement d'applications Web et de microservices avec Spring Framework en offrant trois fonctionnalités principales : la configuration automatique, une approche directive de la configuration et la possibilité de créer des applications autonomes. Ces fonctionnalités combinées permettent de configurer et d'installer une application Spring avec un minimum d'efforts. Dans notre projet, nous utilisons Spring Boot 3.5.7 avec Java 21, Spring Security pour l'authentification JWT, et Spring Data JPA pour la gestion de la base de données. [**?**]

Figure 6.3: Logo Spring Boot

- **Tailwind CSS :** Tailwind CSS est un framework CSS moderne de type "utility-first" qui permet de créer rapidement des interfaces utilisateur personnalisées. Contrairement aux frameworks traditionnels comme Bootstrap, Tailwind fournit des classes CSS atomiques de bas niveau permettant de construire des designs uniques sans écrire de CSS personnalisé. Cette approche offre une flexibilité maximale tout en maintenant la cohérence visuelle. Tailwind intègre un système de design responsive, un mode sombre (dark mode), et une configuration personnalisable via un fichier de configuration. Dans notre projet, Tailwind

CSS remplace Bootstrap pour offrir une expérience utilisateur moderne et personnalisée. [**?**]

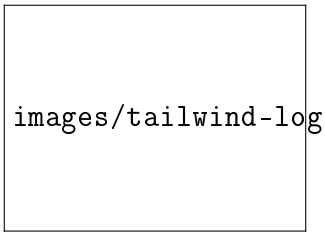

Figure 6.4: Logo Tailwind CSS

- **Node.js :** Node.js est un environnement d'exécution single-thread, open-source et multiplateforme permettant de créer des applications rapides et évolutives côté serveur et en réseau. Il fonctionne avec le moteur d'exécution JavaScript V8 et utilise une architecture d'E/S non bloquante et pilotée par les événements, ce qui le rend efficace et adapté aux applications en temps réel. Node.js est essentiel pour exécuter Next.js et gérer les dépendances avec npm (Node Package Manager). [**?**]

Figure 6.5: Logo Node.js

- **HTML 5 :** HTML est l'abréviation de « hypertext markup language » (langage de balisage hypertexte) et est un langage relativement simple utilisé pour créer des pages web. Comme il n'autorise pas les variables ou les fonctions, il n'est pas considéré comme un « langage de programmation », mais plutôt comme un « langage de balisage », c'est-à-dire un langage qui utilise des balises pour définir les éléments d'un document. Le HTML5, pour HyperText Markup Language 5, est une version du célèbre format HTML utilisé pour concevoir les sites Internet. Celui-ci se résume à un langage de balisage qui sert à l'écriture de l'hypertexte indispensable à la mise en forme d'une page Web. [**?**]

Figure 6.6: Logo HTML 5

- **CSS :** Cascading Style Sheets (CSS) est un langage de programmation qui vous permet de déterminer le design des documents électroniques. À l'aide de simples instructions, présentées dans des codes sources clairs, les éléments de la page Web comme la mise en page, la couleur et la police peuvent ainsi être modulés à souhait. Grâce aux feuilles de style en cascade, la structure sémantique et le contenu du document restent totalement intacts. CSS a été lancé au milieu des années 90 et est à présent considéré comme le langage de feuilles de style standard sur le World Wide Web. Dans notre projet, CSS est utilisé en combinaison avec Tailwind CSS pour créer des interfaces modernes et responsives. [**?**]

Figure 6.7: Logo CSS

- **PostgreSQL/MySQL :** MySQL est un système de gestion de bases de données relationnelles SQL open source développé et supporté par Oracle. C'est un système robuste pour stocker et gérer les données de manière structurée. Une base de données n'est qu'une collection structurée de données qui est organisée pour en faciliter l'utilisation et la récupération. Pour notre application de cabinet médical, ces « données » sont des informations comme les dossiers médicaux, les informations des patients, les rendez-vous, les factures, les utilisateurs (médecins, assistants, patients), etc. MySQL/PostgreSQL sont des solutions de base de données particulièrement populaires pour les applications web modernes nécessitant des transactions ACID et un respect strict de l'intégrité référentielle. [**?**]

Figure 6.8: Logo MySQL

- **Java 21 :** Java est un langage de programmation orienté objet développé par Oracle. Java 21 est la dernière version LTS (Long Term Support) offrant des fonctionnalités modernes comme les records, les pattern matching, et les virtual threads pour améliorer les performances. Java est particulièrement adapté pour les applications d'entreprise critiques comme les systèmes médicaux nécessitant fiabilité et sécurité. Dans notre projet, Java 21 est utilisé avec Spring Boot pour développer l'API REST backend.

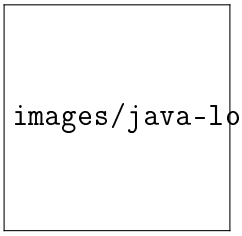

Figure 6.9: Logo Java

- **OpenAI API :** OpenAI API est une interface de programmation développée par OpenAI permettant d'intégrer des modèles d'intelligence artificielle avancés (comme GPT-4) dans des applications. L'API permet de générer du texte, de répondre à des questions, de résumer des documents et d'effectuer diverses tâches de traitement du langage naturel. Dans notre projet, nous utilisons l'API OpenAI pour le chatbot médical intelligent qui répond aux questions des patients. Le modèle GPT-4 analyse les questions en langage naturel et génère des réponses médicales pertinentes et contextuelles.

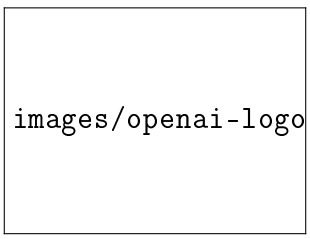

Figure 6.10: Logo OpenAI

6.2.2.2 Les logiciels de développement

Commençons par les logiciels de développement adoptés :

- **VS Code :** Visual Studio Code est un éditeur de code gratuit conçu pour aider les programmeurs à écrire, déboguer et corriger du code grâce à la fonctionnalité IntelliSense. Il simplifie l'écriture de code pour les utilisateurs. VS Code supporte nativement TypeScript, JavaScript, Java, et de nombreux autres langages via des extensions. Il offre un terminal intégré, un débogueur puissant, et une intégration Git. Dans notre projet, VS Code est utilisé pour développer à la fois le frontend (Next.js) et le backend (Spring Boot avec l'extension Java). [**?**]

Figure 6.11: Logo VS Code

- **Postman :** Postman est un logiciel conçu pour créer et tester des requêtes HTTP. Grâce à son interface ergonomique et intuitive, il permet une personnalisation minutieuse des requêtes. Vous pouvez choisir la méthode de la requête (GET, POST, PUT, DELETE), entrer l'URL du serveur à interroger, et ajouter tous les paramètres nécessaires (headers, body, authentication). Le logiciel conserve un historique de vos requêtes, ce qui le rend particulièrement utile pour tester une API. Dans notre projet, Postman est utilisé pour tester tous les endpoints de l'API REST backend (authentification, gestion des patients, rendez-vous, dossiers médicaux, factures, chatbot). [**?**]

Figure 6.12: Logo Postman

- **Git et GitHub :** Git est un système de contrôle de version distribué gratuit et open source conçu pour gérer tous les projets, des plus petits aux plus grands, avec rapidité et efficacité. GitHub est une plateforme d'hébergement de code basée sur Git qui permet la collaboration entre développeurs. Dans notre projet académique, Git permet de suivre l'historique des modifications, de travailler sur des branches séparées pour chaque fonctionnalité, et de fusionner le code de manière sécurisée. GitHub stocke le code source du projet et facilite la collaboration entre les membres de l'équipe.

Figure 6.13: Logo Git

- **Maven :** Apache Maven est un outil de gestion et de compréhension de projet logiciel. Basé sur le concept de Project Object Model (POM), Maven peut gérer la construction, les rapports et la documentation d'un projet à partir d'une information centrale. Dans notre projet Spring Boot, Maven gère automatiquement les dépendances (Spring Security, Spring Data JPA, JWT, etc.), compile le code Java, exécute les tests, et package l'application en fichier JAR exécutable.

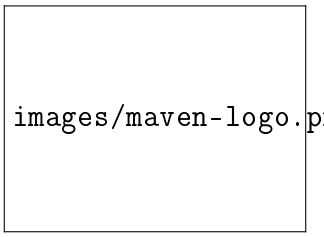

Figure 6.14: Logo Maven

6.2.3 Architecture technique globale

Le tableau **??** présente la stack technique complète du projet.

Table 6.2: Stack technique complète

Couche	Technologies	Rôle
Frontend	Next.js 14, React, TypeScript, Tailwind CSS, Context API	Interface utilisateur responsive, gestion d'état, routing
Backend	Spring Boot 3.5.7, Java 21, Spring Security, Spring Data JPA	API REST, authentification JWT, logique métier, accès données
Base de données	PostgreSQL/MySQL, JPA/Hibernate	Stockage relationnel des données, ORM
Sécurité	JWT, BCrypt, Spring Security, RBAC	Authentification, hashage mots de passe, contrôle d'accès
Services externes	OpenAI API (GPT-4), SMTP (Email)	Chatbot IA, notifications emails
Outils dev	VS Code, Postman, Git, Maven, npm	Développement, tests, versioning, build

6.3 Tests et validation

Une phase essentielle de notre projet a été la réalisation de tests pour garantir la qualité et la fiabilité du système.

6.3.1 Tests backend

Tests unitaires :

- Tests des services métier (création patient, rendez-vous, factures)
- Tests des méthodes de sécurité (hashage BCrypt, génération JWT)
- Tests de validation des données (format email, dates, montants)

Tests d'intégration :

- Tests des endpoints API REST avec Postman
- Tests de l'authentification JWT et du RBAC
- Tests de l'intégration avec la base de données
- Tests de l'API OpenAI pour le chatbot

6.3.2 Tests frontend

Tests fonctionnels :

- Tests des formulaires (validation, messages d'erreur)
- Tests de navigation et routing Next.js
- Tests de l'affichage responsive sur différents écrans
- Tests des interactions utilisateur (boutons, modals, notifications)

6.3.3 Tests de sécurité

Tests RBAC :

- Vérification des restrictions d'accès pour les assistants (dossiers médicaux interdits)
- Vérification de l'isolation des données patients (un patient ne peut voir que SES dossiers)
- Tests de tentatives d'accès non autorisées (403 Forbidden)

Tests RGPD :

- Vérification du hashage des mots de passe (BCrypt)
- Tests de la traçabilité des accès aux données médicales
- Vérification de la sécurisation de l'envoi d'emails

6.4 Difficultés rencontrées et solutions

6.4.1 Intégration OpenAI API

Difficulté : Gestion des timeouts et des erreurs de quota de l'API OpenAI.

Solution : Mise en place d'un système de retry avec backoff exponentiel et affichage de messages d'erreur clairs aux utilisateurs. Limitation du nombre de requêtes par utilisateur pour éviter l'épuisement du quota.

6.4.2 Gestion des fichiers

Difficulté : Stockage et récupération sécurisés des documents médicaux (ordonnances, analyses).

Solution : Création d'un système de stockage structuré avec dossiers par patient (/uploads/dossier-id/), vérification des types de fichiers autorisés, limitation de taille (10 MB), et contrôle d'accès strict (un patient ne peut télécharger que SES documents).

6.4.3 Contrôle d'accès RBAC

Difficulté : Garantir que les assistants ne puissent pas accéder aux dossiers médicaux malgré les tentatives de contournement (modification d'URL, requêtes directes à l'API).

Solution : Implémentation de contrôles de sécurité multi-niveaux : annotations @PreAuthorize sur les contrôleurs backend, vérification des rôles dans les services métier, et affichage conditionnel des menus frontend. Tous les endpoints sensibles retournent 403 Forbidden si le rôle est incorrect.

6.5 Apports du projet

6.5.1 Apports techniques

- Maîtrise de l'architecture full-stack moderne (Next.js + Spring Boot)
- Compréhension approfondie de Spring Security et JWT
- Intégration d'une API d'intelligence artificielle (OpenAI GPT-4)
- Mise en œuvre du pattern MVC et de l'architecture 3-tiers
- Gestion de base de données relationnelle avec JPA/Hibernate
- Développement d'interfaces utilisateur modernes avec Tailwind CSS

6.5.2 Apports méthodologiques

- Application concrète de la méthodologie agile SCRUM
- Planification et organisation en sprints avec livrables fonctionnels
- Collaboration en équipe avec Git et GitHub
- Rédaction de documentation technique (UML, spécifications, diagrammes)
- Tests et validation de système complexe

6.5.3 Apports personnels

- Compréhension des enjeux de sécurité et confidentialité dans le secteur médical
- Sensibilisation aux normes RGPD et secret médical
- Développement de compétences en gestion de projet
- Amélioration de la capacité à résoudre des problèmes techniques complexes
- Expérience pratique d'un projet académique proche de la réalité professionnelle

6.6 Perspectives d'évolution

Bien que fonctionnel, notre système de gestion de cabinet médical peut encore être amélioré et étendu :

6.6.1 Améliorations fonctionnelles

- **Prise de rendez-vous par les patients :** Permettre aux patients de réserver directement leurs rendez-vous en ligne en visualisant les créneaux disponibles.
- **Messagerie sécurisée :** Ajouter un système de messagerie interne entre médecins, assistants et patients conforme au RGPD.
- **Téléconsultation :** Intégrer des fonctionnalités de visioconférence pour permettre les consultations à distance.
- **Gestion de stock de médicaments :** Ajouter un module de gestion de stock pour les cabinets disposant de médicaments.
- **Rappels automatiques :** Système de rappels par SMS/email avant les rendez-vous pour réduire les absences.

6.6.2 Améliorations techniques

- **Application mobile :** Développer une application mobile native (React Native) pour faciliter l'accès aux patients.
- **Amélioration du chatbot :** Fine-tuning du modèle GPT-4 avec des données médicales spécifiques pour améliorer la pertinence des réponses.
- **Tableau de bord analytique :** Ajouter des graphiques avancés avec analyse prédictive (prévisions de revenus, taux d'occupation).
- **Intégration avec cartes Vitale :** Connexion avec le système de santé français pour lecture automatique des cartes Vitale.
- **Système de sauvegarde automatique :** Mise en place de backups automatiques quotidiens de la base de données.

- **Déploiement cloud :** Hébergement sur AWS/Azure avec scalabilité automatique et haute disponibilité.

6.6.3 Conformité et sécurité

- **Certification HDS :** Obtenir la certification Hébergeur de Données de Santé pour conformité légale en France.
- **Audit de sécurité :** Réaliser un audit de sécurité complet par un organisme certifié.
- **Chiffrement de bout en bout :** Implémenter le chiffrement des données médicales au repos et en transit.
- **Logs d'audit complets :** Système de traçabilité exhaustif de tous les accès aux données médicales.

6.7 CONCLUSION GÉNÉRALE

Ce projet de développement d'une plateforme de gestion de cabinet médical avec intelligence artificielle a été une expérience enrichissante à la fois sur le plan technique et méthodologique. Nous avons réussi à concevoir et développer un système complet répondant aux besoins réels des professionnels de santé et des patients.

L'adoption de la méthodologie agile SCRUM nous a permis de structurer notre travail en trois sprints progressifs : l'espace médecin (gestion complète du cabinet), l'espace assistant (gestion administrative avec restrictions), et l'espace patient (consultation et chatbot IA). Cette approche incrémentale a facilité la gestion de la complexité du projet et permis de livrer des fonctionnalités opérationnelles à chaque itération.

Sur le plan technique, l'architecture full-stack moderne combinant Next.js 14 pour le frontend et Spring Boot 3.5.7 pour le backend s'est révélée robuste et performante. L'intégration de l'API OpenAI pour le chatbot médical constitue l'innovation majeure du projet, offrant une assistance virtuelle intelligente accessible 24/7 aux patients.

La sécurité a été une préoccupation centrale tout au long du développement. Le système de contrôle d'accès basé sur les rôles (RBAC) garantit la confidentialité des données médicales en limitant strictement les accès selon le rôle de chaque utilisateur (médecin, assistant, patient). Le respect des principes RGPD (hashage des mots de passe, traçabilité des accès, consentement des patients) positionne notre solution comme conforme aux exigences réglementaires du secteur médical.

Les tests réalisés (unitaires, d'intégration, fonctionnels, sécurité) ont permis de valider la fiabilité du système. Les retours positifs lors des démonstrations confirment la pertinence des fonctionnalités développées et l'ergonomie des interfaces.

Ce projet académique nous a permis d'acquérir des compétences techniques approfondies en développement full-stack, en sécurité applicative, et en intégration d'intelligence artificielle. Il nous a également sensibilisés aux enjeux spécifiques du secteur de la santé : confidentialité des données, responsabilité médicale, et importance de la fiabilité des systèmes d'information médicaux.

Les perspectives d'évolution sont nombreuses : prise de rendez-vous en ligne par les patients, téléconsultation, application mobile, amélioration du chatbot avec fine-tuning, et certification HDS pour un déploiement en production réelle.

En conclusion, ce projet démontre qu'il est possible de développer un système de gestion médicale moderne, sécurisé et innovant en combinant les technologies web actuelles avec l'intelligence artificielle. Cette expérience constitue une base solide pour notre future carrière professionnelle dans le développement d'applications d'entreprise critiques.

CONCLUSION GÉNÉRALE

Ce stage, réalisé au sein de l'Agence Nationale de Maîtrise de l'Énergie (ANME), nous a permis de développer une plateforme collaborative de gestion de réunions et de validation électronique de procès-verbaux, répondant aux besoins critiques identifiés dans les processus métiers de l'institution. À travers ce travail, nous avons pu mettre en œuvre une solution complète alliant technologies modernes, méthodologie agile et respect des exigences fonctionnelles et techniques.

Notre démarche a débuté par une analyse approfondie des processus existants, révélant des lacunes significatives en termes de traçabilité, d'efficacité et de sécurité dans la gestion des réunions et PV. L'étude préliminaire a permis de spécifier précisément les besoins des différents acteurs (FTE, Direction Technique, Commission Technique) et de concevoir une architecture technique robuste basée sur Angular 16 pour le frontend et Spring Boot 3 pour le backend, complétée par une base de données MysQL.

L'adoption de la méthodologie Scrum s'est avérée particulièrement adaptée au contexte d'un stage de deux mois, permettant une gestion efficace des priorités et des livraisons incrémentales. Les deux sprints planifiés ont permis de développer l'ensemble des fonctionnalités critiques, de la gestion des réunions à la signature électronique des PV, en passant par les workflows de validation et le système de notifications. L'approche itérative a favorisé une validation régulière avec les utilisateurs, garantissant l'adéquation entre la solution proposée et les attentes métier.

Sur le plan technique, nous avons réussi à implémenter une architecture modulaire et évolutive, intégrant des fonctionnalités avancées telles que la signature électronique avec ngx-signature-pad, la gestion des versions de documents, et un système de notifications en temps réel. Les défis techniques rencontrés, notamment en matière de sécurité (authentification LDAP, contrôle d'accès

granulaire) et d'intégration des différents composants, ont été surmontés grâce à une veille technologique constante et une approche pragmatique de résolution de problèmes.

Ce projet nous a permis de développer de multiples compétences, tant techniques que méthodologiques. Sur le plan technique, nous avons approfondi notre maîtrise des technologies full-stack modernes, des bonnes pratiques de développement, et des concepts de sécurité applicative. Sur le plan méthodologique, l'application de Scrum nous a enseigné l'importance de l'agilité, de la communication et de la planification dans la conduite de projets complexes. La gestion des exigences changeantes et la coordination avec les différents acteurs métier ont constitué un apprentissage précieux en termes de gestion de projet et de communication technique.

Des perspectives d'évolution intéressantes se dessinent pour cette plateforme. À court terme, l'intégration d'un module de génération automatique de PV à partir des modèles prédéfinis pourrait encore améliorer l'efficacité. À moyen terme, le développement d'une application mobile permettrait d'accéder aux fonctionnalités critiques en déplacement. À long terme, l'intégration avec d'autres systèmes de l'ANME et l'ajout de fonctionnalités d'intelligence artificielle pour l'analyse des tendances dans les réunions pourraient transformer cette plateforme en un outil stratégique de pilotage de l'activité.

En conclusion, ce stage a été une expérience enrichissante qui nous a permis de concilier théorie et pratique, technique et métier, dans un contexte professionnel exigeant. La plateforme développée répond non seulement aux besoins immédiats de l'ANME, mais constitue également une base solide pour l'évolution continue des systèmes d'information de l'institution. Ce travail illustre comment les technologies modernes, lorsqu'elles sont mises en œuvre de manière réfléchie et méthodique, peuvent transformer radicalement des processus administratifs traditionnels en solutions digitales performantes et sécurisées.

BIBLIOGRAPHIE

[1] **ANME**, “Définition d’entreprise”, *adresse:*https://www.anme.tn, consulté le

[2] **Méthode agile**, “Définition Méthode agile”, *adresse:*https://www.nutcache.com/fr/blog/les-methodes-agile consulté le

[3] **Méthode agile**, “Différentes Phases de scrum”, *adresse:*https://bubbleplan.net/blog/agile-scrum-gestion-pr consulté le

[4] **Méthode Scrum** , “Définition Méthode Scrum ”, *adresse:*https://asana.com/fr/resources/what-is-scrum, consulté le

[5] **Feature Driven Development** , *adresse:*https://www.productplan.com/glossary/feature-driven-developme consulté

[6] **12-principes-du-manifeste-agile** , *adresse:*https://www.softfluent.fr/blog/12-principes-du-manifeste-agile consulté

[7] **Modilisation UML** , *adresse:*https://www.ionos.fr/digitalguide/sites-internet/developpement-web/uml-un consulté

[8] **Teamgantt**, *adresse:*https://app.teamgantt.com/projects/gantt?ids=3976097, consulté le

[9] **Angular**, *adresse:*https://v15.angular.io/docs, consulté le

[10] **Typescript**, *adresse:*https://www.typescriptlang.org, consulté le

[11] **Spring Boot**, *adresse:*https://spring.io/projects/spring-boot, consulté le

[12] **Boostrap**, *adresse:*https://getbootstrap.com, consulté le

[13] **Nodejs**, *adresse:*https://nodejs.org/en, consulté le

[14] **HTML**, *adresse:*https://developer.mozilla.org/fr/docs/Web/HTML, consulté le

[15] **CSS**, *adresse:*https://developer.mozilla.org/fr/docs/Web/CSS, consulté le

[16] **Mysql**, *adresse:*https://www.mysql.com/fr/, consulté le

[17] **VSCODE**, *adresse:*https://code.visualstudio.com, consulté le

[18] **Postman**, *adresse:*https://www.postman.com, consulté le

[19] **XAMMP**, *adresse:*https://www.apachefriends.org/index.html, consulté le