
âarchitecture du projet logo 5366 l'architecture du projet figure.0.53



Ministère de l'Enseignement Supérieur et de
la Recherche Scientifique Direction Générale
des Etudes Technologiques

Institut Supérieur des Etudes
Technologiques de Rades

République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
المعهد العالي للدراسات التكنولوجية برايس
Institut Supérieur Des Etudes Technologiques De Rades

RAPPORT DE PROJET D'INTÉGRATION

LICENCE EN TECHNOLOGIES DE L'INFORMATIQUE

Parcours : Développement des systèmes d'information

Développement d'une application "Medical-Solution"

Travail présenté à :

Mme Mariem Baccouche
Mr Faouzi Maddouri

Réalisé par :

Eya Mellehi
Yessmine Snoussi
Imen Nouri

Année universitaire : 2024/2025

Remerciements

Au terme de ce projet, nous tenons à exprimer toute notre gratitude aux responsables et aux enseignants de Institut Supérieur des Etudes Technologiques de Rades, qui nous ont soutenus dans la réalisation de ce travail.

Nous adressons nos plus sincères remerciements à Mme Mariem Baccouche et M. Faouzi Maddouri, nos enseignants, pour leur encadrement précieux dans le cadre de la matière *Projet d'Intégration*. Grâce à leurs conseils avisés, leur soutien constant et leurs encouragements, nous avons pu mener ce projet à bien. Leur accompagnement a été une source d'inspiration et a grandement contribué à l'aboutissement de ce travail.

Enfin, nous remercions toutes les personnes qui ont, de près ou de loin, apporté leur aide et leurs conseils pour mener ce projet à terme, et nous espérons que notre travail saura répondre aux attentes des membres du jury.

Table des matières

Remerciements	i
Introduction Générale	1
Introduction à l'application	2
1 Introduction	2
2 Fonctionnalités principales	2
2.1 Règles de gestion et flux d'information	2
3 Méthodologie de travail	3
3.1 Scrum	3
3.2 L'Équipe Scrum	4
3.3 Les événements Scrum	4
3.4 Les artefacts Scrum	5
4 Environnement de travail	5
4.1 StarUML	5
4.2 Postman	5
5 Environnement logiciel	6
5.1 C# :	6
5.2 Angular	6
5.3 Spring Boot	7
5.4 Visual Studio Code	7
5.5 Visual Studio	8
5.6 Neon PostgreSQL	8
5.7 GitHub	9
6 Conclusion	9
Spécification des besoins et analyse	10
1 Introduction	10
2 Spécification des besoins	10
2.1 Identification des besoins fonctionnels	10
2.2 Identification des besoins non fonctionnels	11
2.3 Identification des acteurs	11
3 Modélisation	12
3.1 Diagramme de packages	12
3.2 Diagramme de cas d'utilisation	13
4 Product backlog	14
5 Conclusion	14

TABLE DES MATIÈRES

Réalisation et tests	15
1 Sprint 1 :Authentification et consultation des Rendez-vous pour les Patients et Agents d'Accueil	15
1.1 Objectif du Sprint	15
1.2 Backlog du sprint 1	15
1.3 Conception	17
1.4 Les interfaces	21
1.5 Plan de test	29
1.6 Bilan du Sprint	33
2 Sprint 2 :Authentification et consultation des Rendez-vous par les Secrétaires Médicales	34
2.1 Objectif du Sprint	34
2.2 Backlog du sprint 2	35
2.3 Coneption	37
2.4 Les interfaces	41
2.5 Plan de test	45
2.6 Bilan du Sprint 2	47
3 Sprint 3 : Authentification et consultation des Rendez-vous par les Médecins	49
3.1 Objectif du Sprint	49
3.2 Identidication des backlog du sprint 3	49
3.3 Conception	50
3.4 Test	58
3.5 Bilan du Sprint 3	59
4 Sprint 4 :Ajustement des disponibilités des médecins	60
4.1 Objectif du Sprint	60
4.2 Identidication des backlog du sprint 4	60
4.3 Conception	60
4.4 Bilan de sprint	65
5 Présentation du Code Source	66
5.1 Organisation du Code	66
6 Exemples de Code	67
6.1 Sélection et mise à jour du rendez-vous	82
7 Plan de test du partie mobile	84
7.1 Plan de test	84
7.2 Scénario de test	86
7.3 Rapport de test	87
8 Conclusion	88
Conclusion Générale	89
Bibliographie	90

Table des figures

1	Scrum	4
2	Logo du StarUML	5
3	Logo du Postman	6
4	Logo du c#	6
5	Logo de Angular	7
6	Logo de SpringBoot	7
7	Logo de Visual Studio Code	8
8	Logo de Visual Studio	8
9	Logo de Neon PostgreSQL	8
10	Logo de GitHub	9
11	Diagramme de Package	12
12	Diagramme de cas d'utilisation global	13
13	Diagramme de cas d'utilisation pour la gestion des rendez-vous patient et leur consultation par l'agent	17
14	Diagramme de classe pour les demandes des rendez-vous et leur confirmation	18
15	Diagramme de séquence du prise du rendez-vous du patient	19
16	Diagramme d'activité du Consultation des rendez-vous de l'agent	20
17	Interface de connexion pour les agents	21
18	Affichage des médecins et gestion de leur calendrier	22
19	Liste des rendez-vous en attente de validation	23
20	Génération d'un fichier PDF pour les rendez-vous confirmés	23
21	Génération d'un fichier PDF pour les rendez-vous confirmés	24
22	Interface d'inscription	25
23	Interface d'inscription	26
24	Liste des médecins disponibles	27
25	Formulaire de demande de rendez-vous	28
26	Message de confirmation après une demande de rendez-vous	29
27	Diagramme de cas d'utilisation d'Authentification et consultation des Rendez-vous par les Secrétaires Médicales	37
28	Diagramme de classes	38
29	Diagramme de séquence du consultation et décalage des rendez-vous	39
30	Diagramme d'activité du décalage du rendez-vous par la secrétaire	40
31	login du secretaire	41
32	affichage des rendez-vous en attente	42
33	affichage des rendez-vous confirmés	43
34	décalage de rendez-vous	43
35	notification de decalage par mail	44
36	notification de confirmation par mail	44
37	Diagramme de cas d'utilisation du sprint 3	50

TABLE DES FIGURES

38	Diagramme des classes du sprint 3	51
39	Diagramme d'activité de modification de date des rendez-vous par le patient	52
40	Inscription du médecin	53
41	Authentification du médecin	54
42	Consultation de calendrier par le médecin	54
43	Consultation des rendez-vous par le patient	55
44	Annulation d'un rendez-vous par le patient	56
45	Décalage de rendez-vous par le patient	57
46	Diagramme de cas d'utilisation	60
47	Diagramme des classes	61
48	Diagramme d'activité de modification de disponibilité par le médecin	62
49	Consultation du détails du rendez-vous par le médecin	63
50	Ajout de disponibilité par le médecin	63
51	Annulation de disponibilité par le médecin	63
52	Notification du décalage	64
53	l'architecture du projet	66
54	Plan de test de la partie mobile	85
55	Scénario dde test de la partie mobiles	86
56	Rapport de test de la partie mobile	87

Liste des tableaux

1	Product Backlog	14
3	backlog de sprint 1	16
4	Bilan du Sprint 1	34
5	Backlog de sprint 2	36
15	Bilan du Sprint 2	48
16	Sprint Backlog	49
17	Données de test	58
18	Cas de test	58
19	Données de test	58
20	Cas de test	59
21	Bilan du Sprint 3	59
22	Bilan du sprint - Ajustement des disponibilités des médecins	65

Introduction Générale

Choisir ce projet comme sujet de notre intégration était une décision réfléchie, motivée par notre intérêt commun pour la création d'applications pratiques et fonctionnelles dans le domaine de la santé. En tant qu'étudiants en Technologies de l'Informatique, nous avons été particulièrement attirés par les défis liés à la manipulation des rendez-vous, à la coordination entre différents types d'utilisateurs (patients, secrétaires, médecins, agents d'accueil), et à l'optimisation des processus administratifs dans un centre médical.

Au sein de l'Institut Supérieur des Études Technologiques de Rades, au Département des Technologies de l'Informatique, nous avons eu l'opportunité, en tant qu'équipe, de réaliser un projet d'intégration dans le cadre de notre formation. Ce projet a consisté en la conception et le développement d'une application destinée à la gestion d'un centre médical.

Les problématiques liées aux prises de rendez-vous médicaux sont multiples et complexes. Dans les centres médicaux traditionnels, les patients rencontrent souvent des difficultés pour planifier leurs rendez-vous, que ce soit en raison de l'indisponibilité des créneaux horaires adaptés, d'un manque de visibilité sur les disponibilités des médecins, ou encore d'une organisation inefficace. Ces obstacles peuvent entraîner des frustrations pour les patients, mais également des désagréments pour les professionnels de santé, qui doivent jongler avec des emplois du temps surchargés et parfois mal coordonnés. À cela s'ajoutent des annulations de dernière minute, des erreurs administratives et une communication limitée entre les différentes parties prenantes.

Ces défis illustrent l'importance d'une solution numérique moderne et centralisée qui permettrait de rationaliser le processus de gestion des rendez-vous. Notre objectif principal était donc de concevoir une solution complète et intégrée pour la gestion des activités d'un centre médical, comprenant une interface mobile, desktop et web, afin de répondre aux besoins des différents utilisateurs. Ce projet nous a offert l'opportunité de mettre en pratique nos compétences en développement d'applications, en gestion de bases de données et en design d'interfaces utilisateur.

Introduction à l'application

1 Introduction

L'application est destinée à un centre médical avec un accueil centralisé et plusieurs cabinets de différentes spécialités médicales. Son objectif est de faciliter la consultation des rendez-vous et d'assurer une communication fluide entre les différents acteurs du système : agents d'accueil, secrétaires médicales, patients et médecins. Elle permet une coordination efficace des rendez-vous tout en tenant compte des spécificités de chaque cabinet.

2 Fonctionnalités principales

L'application de gestion des rendez-vous offre une interface intuitive aux patients pour demander, confirmer ou annuler leurs rendez-vous. Les agents d'accueil et les secrétaires médicales l'utilisent pour gérer les emplois du temps des médecins et informer les patients des changements éventuels. Les médecins peuvent consulter leur agenda et ajuster leurs disponibilités en fonction de leurs besoins professionnels.

2.1 Règles de gestion et flux d'information

- **Prise de rendez-vous :** Les agents d'accueil ont accès à l'agenda global pour organiser les rendez-vous des patients en fonction des disponibilités des cabinets.
- **Circulation des informations :** Les secrétaires médicales et les médecins reçoivent de manière immédiate les mises à jour sur les rendez-vous.
- **Changement de rendez-vous :** Les secrétaires médicales peuvent ajuster les rendez-vous en cas d'empêchement et en informer les patients. Les médecins peuvent modifier leurs créneaux horaires et leurs disponibilités, et ces changements se répercutent sur les agendas.
- **Flux d'infomation :** Le patient demande un rendez-vous pour une consultation. Il peut indiquer le médecin, ou bien la spécialité sans préférence pour un médecin particulier.
l'agent valide ou non le rendez-vous , il peut suggérer une autre date ou un autre médecin dans la même spécialité .
le patient reçoit une notification de validation de rendez-vous ou une proposition pour une autre date ou un autre médecin .
il peut demander le nouveau rendez- vous proposé qui sera validé par l'agent si il est encore disponible .
la secrétaire peut consulter les rendez-vous des patients de son cabinet , elle peut confirmer le rendez-vous comme l'annuler , une notification est envoyé au patient .
un patient peut annuler ou confirmer un rendez-vous , le médecin peut changer ses

disponibilités futures : jour et tranche horraire.

l'agent peut consulter les disponibilités des médecins par jour , par spécialité , par médecin.

3 Méthodologie de travail

Pour ce projet, nous avons choisi de travailler avec une méthode Agile : Scrum, car elle nous permet de gérer efficacement les différentes étapes de conception et de développement de notre application de gestion pour un centre médical tout en restant flexibles face aux ajustements nécessaires.

3.1 Scrum

Scrum est un framework de gestion de projets Agile qui structure le travail en cycles itératifs et incrémentaux, appelés Sprints. Cela a permis à notre équipe d'intégrer rapidement des fonctionnalités, de répondre aux besoins spécifiques des utilisateurs finaux et de livrer une solution fonctionnelle étape par étape.[8]

Dans le cadre de notre projet :

Division du travail :

Le Product Backlog a été constitué à partir des besoins identifiés, tels que :

- Gestion des rendez-vous pour les patients.
- Affichage des disponibilités des médecins.
- Interface pour les secrétaires médicales et les agents d'accueil.
- Création d'un tableau de bord pour les médecins.

Exécution des Sprints :

- Chaque Sprint, d'une durée de deux semaines, avait un objectif précis, comme le développement de l'interface mobile pour les patients ou l'intégration des fonctionnalités de gestion des disponibilités.
- À la fin de chaque Sprint, une version testable de l'application était évaluée pour validation.

Adaptations : À chaque Sprint Review, nous avons recueilli les retours des parties prenantes (enseignants et utilisateurs fictifs) et ajusté les priorités dans le Product Backlog.

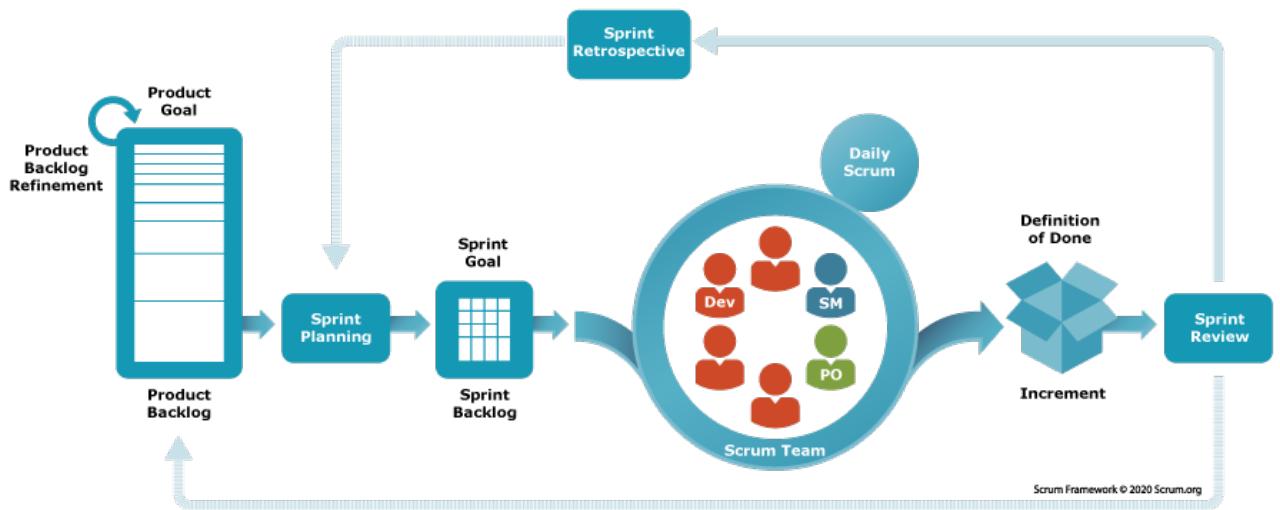


FIGURE 1 – Scrum

3.2 L'Équipe Scrum

Notre équipe Scrum était composée de trois membres, chacun ayant un rôle spécifique pour assurer l'avancement du projet.

Scrum Master : Le rôle de Scrum Master a été assuré à tour de rôle. Eya Mellehi a occupé ce rôle pendant le Sprint 1, Yessmine Snoussi pendant le Sprint 2, et Imen Nouri pendant le Sprint 3. Chaque Scrum Master a supervisé la gestion du Sprint et éliminé les obstacles pour garantir son bon déroulement.

Product Owner : notre professeur, a joué le rôle de Product Owner. Elle a géré le Product Backlog, priorisé les tâches en fonction des besoins des utilisateurs (patients, secrétaires, médecins, agent d'accueil), et s'est assurée que chaque Sprint réponde aux objectifs définis pour le projet.

Développeurs : Les trois membres ont assuré les tâches de développement. Eya Mellehi a travaillé sur l'interface mobile, Yessmine Snoussi sur l'interface web, et Imen Nouri sur l'interface desktop et l'intégration des fonctionnalités.

[8]

3.3 Les événements Scrum

Dans le cadre de notre projet, voici comment les événements Scrum ont été adaptés :

Sprint Planning : Lors de ces réunions, nous avons décidé des tâches à accomplir pour chaque Sprint, comme la création de la page de connexion ou la mise en place du système de prise de rendez-vous.

Daily Scrum : Nous avons tenu des réunions quotidiennes de 15 minutes pour discuter des progrès réalisés, par exemple :

- Défis techniques liés à la connexion avec la base de données.
- Ajustements nécessaires dans l'interface utilisateur.

Sprint Review : À la fin de chaque Sprint, une démonstration de la fonctionnalité développée a été présentée, comme l'affichage des rendez-vous confirmés dans l'application mobile.

Sprint Retrospective : Ces réunions nous ont permis d'identifier des axes d'amélioration, notamment sur la répartition des tâches ou la gestion des délais.

3.4 Les artefacts Scrum

Voici comment les artefacts Scrum ont été appliqués :

Product Backlog : Contenait toutes les fonctionnalités identifiées pour l'application, comme :

- Gestion des annulations de rendez-vous par les patients.
- Consultation et modification des disponibilités des médecins.
- Notifications pour les patients .

Sprint Backlog : Comprenait les tâches sélectionnées pour chaque Sprint, comme la création de l'interface utilisateur des secrétaires ou l'implémentation du système de notifications.

Increment : Chaque Sprint a abouti à une version fonctionnelle, par exemple, un prototype permettant aux patients de consulter les horaires des médecins et de prendre des rendez-vous.

4 Environnement de travail

4.1 StarUML

Nous avons utilisé StarUML pour modéliser les interactions clés et structurer les composants de l'application. Cet outil nous a permis de créer des diagrammes **UML** tels que les cas d'utilisation, les diagrammes de séquence et de classes, facilitant ainsi la compréhension des exigences et des relations entre les entités.

Grâce à StarUML, nous avons pu organiser les tâches de développement, anticiper les problèmes potentiels et réduire les risques d'erreurs lors de la phase de test. Cette approche a contribué à garantir un développement clair et bien planifié.

[1]



FIGURE 2 – Logo du StarUML

4.2 Postman

Nous avons utilisé Postman pour tester et valider les **API** développées dans le projet. Cet outil nous a permis de simuler des requêtes HTTP telles que GET, POST et PUT afin

de vérifier le bon fonctionnement des endpoints. Par exemple, nous avons utilisé Postman pour tester l'ajout de patients et la gestion des rendez-vous en envoyant des données JSON au serveur et en analysant les réponses retournées.

De plus, nous avons sauvegardé nos requêtes dans des collections, ce qui nous a aidés à organiser nos tests et à reproduire facilement des scénarios lors de modifications ou de débogages. Cela nous a assuré une interaction fiable entre le client et le serveur.

[7]



FIGURE 3 – Logo du Postman

5 Environnement logiciel

5.1 C# :

Nous avons utilisé le langage *C#* avec *Xamarin.Forms* pour développer la partie mobile et la partie desktop de l'interface utilisateur de l'application. *C#* a permis de bénéficier de la compatibilité multiplateforme offerte par la plateforme .NET, facilitant ainsi le développement d'applications pouvant fonctionner sur différents appareils. Grâce à *Xamarin.Forms*, nous avons créé une expérience utilisateur cohérente pour les utilisateurs mobiles, tandis que l'interface desktop a été conçue pour répondre aux besoins spécifiques des secrétaires et des agents d'accueil. Ce choix technologique a simplifié l'intégration des fonctionnalités avec le serveur et assuré une bonne interopérabilité entre les différentes plateformes.

[3]

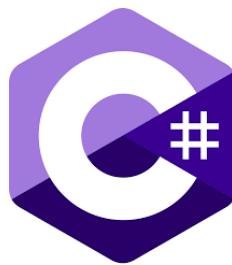


FIGURE 4 – Logo du c#

5.2 Angular

Nous avons utilisé *Angular* pour développer la partie web destinée aux médecins dans le projet. Ce framework open source, géré par Google, nous a permis de créer une

application web dynamique et réactive en utilisant HTML, CSS et JavaScript. Grâce à son architecture basée sur des composants, nous avons divisé l'interface en éléments indépendants réutilisables, ce qui a facilité la gestion de la logique, la maintenance et l'évolution de l'application. *Angular* a ainsi contribué à offrir une expérience utilisateur fluide et intuitive dans l'interface web permettant aux médecins de gérer leurs rendez-vous et consulter leur emploi du temps.[2]



FIGURE 5 – Logo de Angular

5.3 Spring Boot

Nous avons utilisé *Java Spring Boot* pour développer la partie back-end de l'interface destinée aux médecins. Il nous a permis de gérer la logique métier, de créer des services pour le traitement des requêtes HTTP et d'assurer la communication avec la base de données. Grâce à *Spring Boot*, nous avons construit une architecture solide et efficace pour permettre un fonctionnement optimal de l'interface web pour les médecins.

[9]



FIGURE 6 – Logo de SpringBoot

5.4 Visual Studio Code

Nous avons utilisé *Visual Studio Code* comme éditeur de code pour le développement avec *Java Spring Boot*. Cet éditeur léger et puissant nous a permis d'écrire et de gérer efficacement le code, tout en bénéficiant de ses fonctionnalités avancées telles que le débogage, l'intégration Git et la prise en charge de multiples langages via des extensions. Grâce à *Visual Studio Code*, nous avons pu travailler de manière collaborative sur la partie back-end de l'interface médecin avec *Java Spring Boot*. [4]

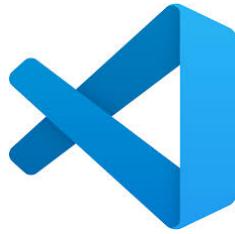


FIGURE 7 – Logo de Visual Studio Code

5.5 Visual Studio

Nous avons utilisé Visual Studio comme environnement de développement pour créer l'application avec Xamarin.Forms. Cet **IDE** complet nous a permis d'écrire, de modifier, de déboguer et de gérer le code efficacement tout au long du cycle de développement. Grâce à ses outils intégrés, comme le débogage, le contrôle de code source, les compilateurs et les fonctionnalités avancées, Visual Studio a facilité la création, les tests et le déploiement de l'application sur différentes plateformes (mobile et desktop) avec Xamarin.Forms.

[10]



FIGURE 8 – Logo de Visual Studio

5.6 Neon PostgreSQL

Nous avons utilisé Neon comme service de base de données pour notre projet. Ce service innovant, basé sur une architecture serverless et compatible avec PostgreSQL, a simplifié la gestion et la mise à l'échelle de la base de données dans un environnement cloud.

Grâce à Neon, nous avons bénéficié d'une solution flexible et évolutive, sans la complexité généralement associée à l'administration des bases de données. Il a permis de répondre efficacement aux besoins de l'application en matière de performance et de scalabilité, tout en s'intégrant parfaitement avec l'architecture cloud native de notre projet.

[6]



FIGURE 9 – Logo de Neon PostgreSQL

5.7 GitHub

Nous avons utilisé GitHub pour héberger et gérer notre travail collaboratif sur le projet. Cette plateforme cloud nous a permis de stocker notre code, de suivre l'évolution de notre travail au fil du temps, et de faciliter les échanges entre membres de l'équipe.

Grâce à ses fonctionnalités telles que le suivi des modifications, la révision du code, et les demandes de fusion (pull requests), GitHub a amélioré la coordination et la qualité du développement. Cela nous a également permis de collaborer de manière efficace tout en protégeant l'intégrité du travail de chaque membre de l'équipe. [5]



FIGURE 10 – Logo de GitHub

6 Conclusion

Ce chapitre présente l'application de gestion des rendez-vous médicaux, ses objectifs, et la méthodologie Agile Scrum adoptée. Il décrit également les outils techniques utilisés, posant ainsi les bases pour les aspects techniques à venir. Dans le chapitre suivant, nous détaillerons la spécification des besoins et la conception.

Spécification des besoins et analyse

1 Introduction

Dans ce chapitre, nous nous intéressons aux besoins des utilisateurs considérés dans notre projet à travers les spécifications fonctionnelles et non fonctionnelles, puis les entités externes qui vont interagir avec le système. Enfin, le diagramme de cas d'utilisation globale et le backlog produit sont présentés afin d'aboutir à une application de qualité qui répond aux besoins des clients.

2 Spécification des besoins

2.1 Identification des besoins fonctionnels

Un besoin fonctionnel est un besoin spécifiant une action qu'un système doit être capable d'effectuer, sans considérer aucune contrainte physique. C'est un besoin du point de vue de l'utilisateur.

Les besoins fonctionnels sont décrits en termes de fonctionnalités pour chaque acteur du système :

Agent d'accueil

- **Prise de rendez-vous** : Consultation des disponibilités de chaque cabinet et prise de rendez-vous pour les patients.
- **Confirmation et suivi** : Envoi des notifications de confirmation aux patients et aux secrétaires.

Secrétaire médicale

- **Consultation des rendez-vous.**
- **Décalage des rendez-vous.**
- **Annulation des rendez-vous.**
- **Notification des patients** : Suivi des notifications en cas de changement de rendez-vous.

Patient

- **Prise de rendez-vous** : Demande de rendez-vous en fonction des disponibilités des cabinets.
- **Consultation de ses rendez-vous.**
- **Confirmation de ses rendez-vous.**
- **Annulation de ses rendez-vous.**

Médecin

- **Consultation de l'agenda** : Visualisation de son emploi du temps quotidien.
- **Changement des disponibilités** : Suivi et ajustement de ses créneaux horaires disponibles.

2.2 Identification des besoins non fonctionnels

Les besoins non fonctionnels définissent les critères relatifs au fonctionnement global du système, sans se concentrer sur les spécificités des actions à réaliser. Ils englobent :

- **Facilité d'utilisation** : L'application doit être intuitive et simple à utiliser pour tous les acteurs. Des interfaces claires et réactives . Des tests utilisateurs assureront une bonne expérience.
- **Performance** : L'application doit être rapide et réactive, avec des temps de réponse optimisés grâce à **Spring Boot** et **MySQL**.Des tests de montée en charge garantiront la stabilité même avec un grand nombre d'utilisateurs.
- **Capacité fonctionnelle** : conformités aux besoins et application des règles de gestion, la précision des résultats obtenus, interopérabilité de l'application : minimiser l'intervention humaine.

2.3 Identification des acteurs

— Agent d'accueil :

L'agent d'accueil est le premier point de contact pour les patients. Il est responsable de l'organisation initiale des rendez-vous, coordonnant les disponibilités entre les patients et les cabinets médicaux. Il accueille également les patients lors de leur arrivée au centre médical, assurant une expérience fluide.

— Secrétaire médicale :

La secrétaire médicale supervise les rendez-vous au sein de son cabinet. Elle ajuste les rendez-vous en fonction des disponibilités du médecin et informe les patients de tout changement. Elle organise l'agenda du médecin pour assurer une prestation médicale optimale.

— Patient :

Le patient utilise l'application pour demander et consulter ses rendez-vous. Il interagit avec les agents d'accueil et les secrétaires pour confirmer ses rendez-vous et est informé de tout changement.

— Médecin :

Le médecin fournit des services médicaux aux patients et organise ses consultations via l'application. Il consulte son agenda pour suivre ses rendez-vous et ajuste ses disponibilités en fonction de ses impératifs professionnels. Il reste en communication avec les secrétaires pour toute modification d'emploi du temps.

3 Modélisation

3.1 Diagramme de packages

Le diagramme de packages est un diagramme UML qui permet de structurer un système en regroupant les éléments liés dans des packages. Il est utilisé pour organiser les composants et modules du système, facilitant ainsi la compréhension et la gestion des relations entre eux.

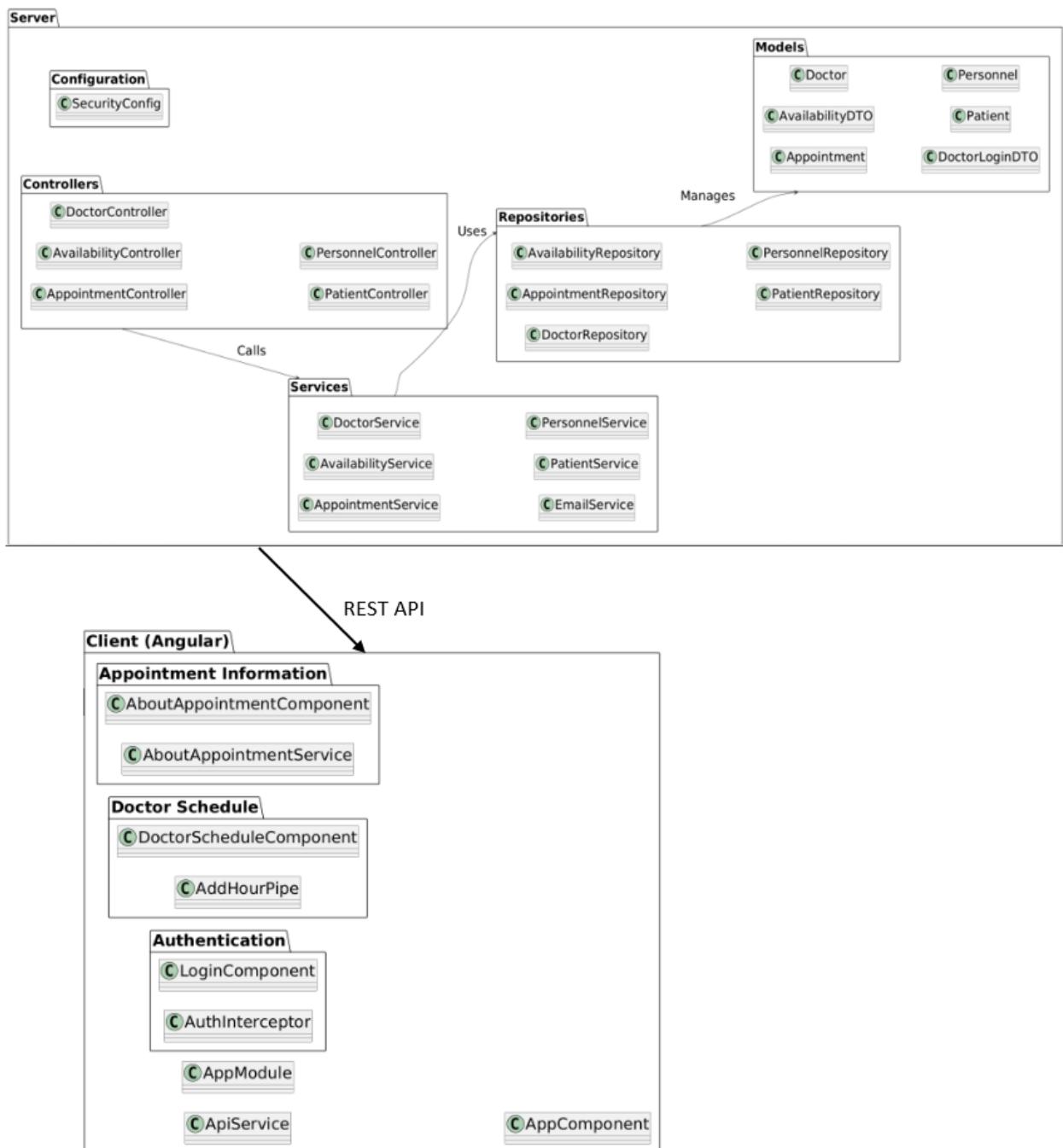


FIGURE 11 – Diagramme de Package

3.2 Diagramme de cas d'utilisation

Ce diagramme de cas d'utilisation représente l'interaction entre les différents acteurs impliqués dans le processus de gestion des rendez-vous médicaux, notamment les patients, les secrétaires, les agents d'accueil, et les médecins.

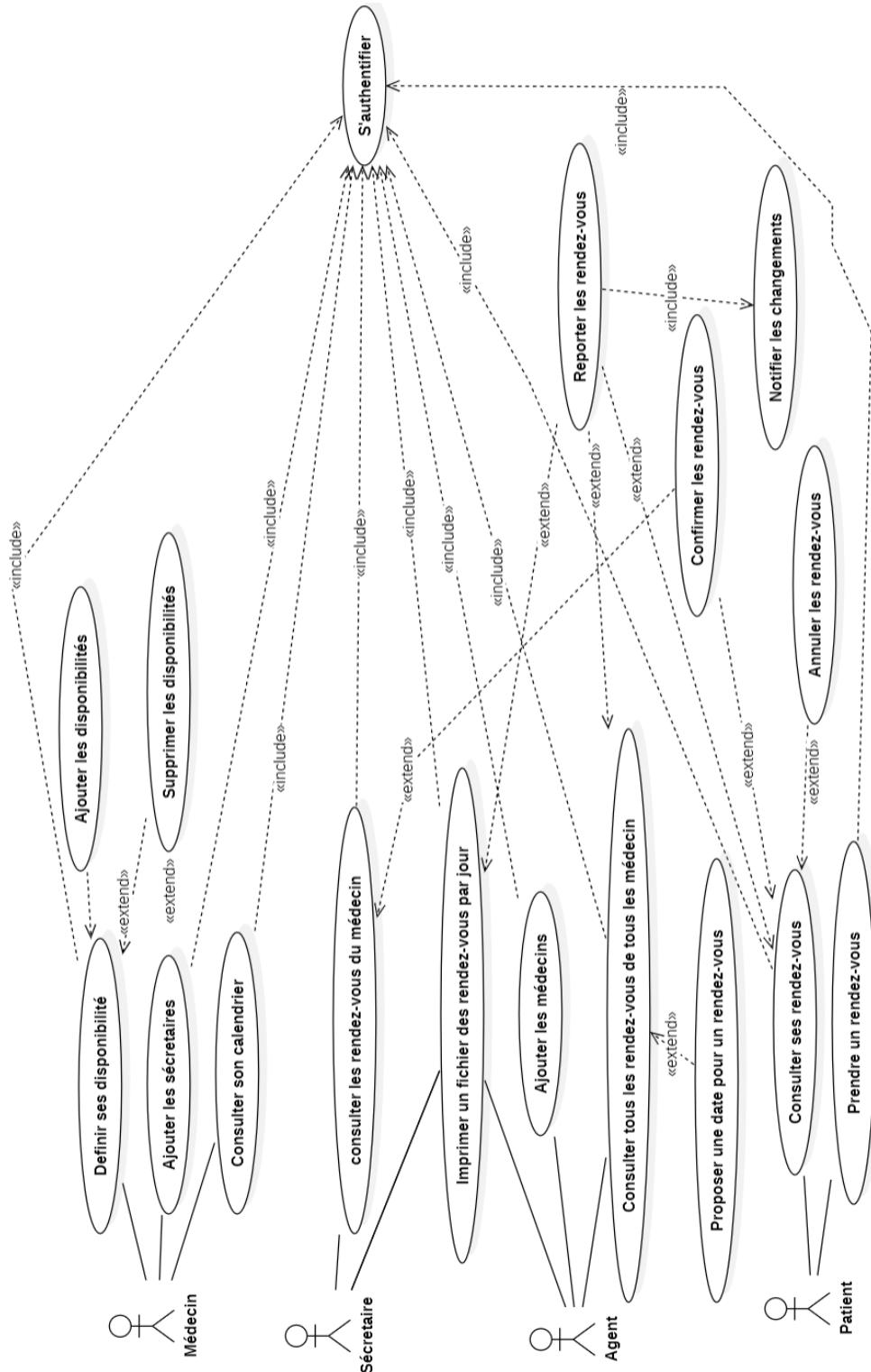


FIGURE 12 – Diagramme de cas d'utilisation global

4 Product backlog

User Story	Temps estimé	Priorité
En tant que patient, je veux pouvoir demander un rendez-vous via l'application après m'être authentifié.	3 jours	Haute
En tant qu'agent d'accueil, je veux consulter les rendez-vous des patients après m'être authentifié.	4 jours	Haute
En tant qu'agent d'accueil, je veux confirmer les demandes de rendez-vous des patients.	3 jours	Normal
En tant qu'agent d'accueil, je veux pouvoir imprimer les rendez-vous par jour et par cabinet.	4 jours	Normal
En tant que secrétaire médicale, je veux consulter les rendez-vous après m'être authentifiée.	3 jours	Haute
En tant que secrétaire médicale, je veux ajuster les rendez-vous en fonction des changements des médecins ou des imprévus.	2 jours	Haute
En tant que patient, je veux consulter les notifications si mon rendez-vous est modifié par la secrétaire médicale.	3 jours	Haute
En tant qu'agent d'accueil, dès que je confirme un rendez-vous, une notification sera envoyée aux patients.	3 jours	Normal
Correction de la fonctionnalité permettant aux agents d'accueil d'imprimer les rendez-vous par jour et par cabinet.	1 jour	Haute
En tant que médecin, je veux accéder à mon agenda après m'être authentifié.	4 jours	Haute
En tant que patient, je veux annuler mon rendez-vous après authentification.	3 jours	Haute
En tant que patient, je veux décaler mon rendez-vous après authentification.	5 jours	Haute
En tant que médecin, je veux ajuster mes disponibilités en fonction de mes impératifs professionnels.	4 jours	Haute

TABLE 1 – Product Backlog

5 Conclusion

Nous avons souligné l'importance des spécifications et des acteurs dans le projet. La modélisation permet d'organiser les composants du système, préparant ainsi le terrain pour la gestion du projet dans les chapitres suivants.

Réalisation et tests

Introduction

La phase de réalisation et de test est essentielle dans le développement d'un système, permettant de concrétiser les spécifications et de valider le bon fonctionnement de l'application. Ce chapitre couvre le développement des fonctionnalités, les tests effectués, et l'analyse des résultats pour garantir la robustesse et l'efficacité du système.

1 Sprint 1 :Authentification et consultation des Rendez-vous pour les Patients et Agents d'Accueil

1.1 Objectif du Sprint

L'objectif de ce sprint était de mettre en place les fonctionnalités de base liées à l'authentification, l'inscription et la gestion des rendez-vous sur la plateforme.

1.2 Backlog du sprint 1

User Story	Tâche	Temps estimé	Responsable
User Story 1 : En tant que patient, je veux pouvoir demander un rendez-vous via l'application après m'être authentifié.	Concevoir l'interface de connexion et d'inscription pour le patient.	0.5 jour	Yessmine Snoussi
	Implémenter la logique d'authentification (API et base de données).	1 jour	Yessmine Snoussi
	Concevoir l'interface pour demander un rendez-vous (formulaire).	0.5 jour	Yessmine Snoussi
	Implémenter la logique de création et de validation des demandes de rendez-vous.	0.5 jour	Yessmine Snoussi
	Tester le parcours complet (authentification + demande).	0.5 jour	Yessmine Snoussi

LISTE DES TABLEAUX

User Story 2 : En tant qu'agent d'accueil, je veux pouvoir m'authentifier pour accéder à la consultation des rendez-vous des patients.	Concevoir l'interface d'authentification pour l'agent d'accueil.	0.5 jour	Yessmine Snoussi
	Implémenter la logique d'authentification (API et base de données).	1 jour	Yessmine Snoussi
	Créer l'interface pour afficher les rendez-vous des patients.	1 jour	Yessmine Snoussi
	Implémenter la logique d'affichage des rendez-vous (filtrage, tri, données en temps réel).	1 jour	Yessmine Snoussi
	Tester la fonctionnalité complète (authentification + consultation des rendez-vous).	0.5 jour	Yessmine Snoussi
User Story 3 : En tant qu'agent d'accueil, je veux confirmer les demandes de rendez-vous des patients.	Ajouter un bouton de confirmation pour chaque demande de rendez-vous dans l'interface.	0.5 jour	Imen Nouri
	Implémenter la logique de confirmation dans la base de données.	1 jour	Imen Nouri
	Tester le flux de confirmation des demandes.	1 jour	Imen Nouri
User Story 4 : En tant qu'agent d'accueil, je veux pouvoir imprimer les rendez-vous par jour et par cabinet.	Ajouter un bouton d'impression dans l'interface des rendez-vous.	0.5 jour	Imen Nouri
	Implémenter une fonctionnalité de génération de rapports PDF contenant les rendez-vous.	1.5 jour	Imen Nouri
	Ajouter des filtres (par jour et par cabinet) pour les données à imprimer.	1 jour	Imen Nouri
	Tester l'impression avec différents scénarios.	1 jour	Imen Nouri

TABLE 3: backlog de sprint 1

1.3 Conception

Diagramme de Cas d'Utilisation

Le système permet à deux types d'utilisateurs d'interagir : le patient et l'agent. Le patient peut soumettre une demande de rendez-vous via l'application, mais cette action nécessite une authentification préalable pour garantir que seuls les utilisateurs validés puissent faire des demandes. Quant à l'agent, il doit également s'authentifier pour accéder aux fonctionnalités de gestion des rendez-vous. Une fois authentifié, l'agent peut consulter la liste complète des rendez-vous des patients. Par la suite, il a la possibilité d'effectuer des actions supplémentaires sur ces rendez-vous, comme les imprimer pour obtenir une version papier ou les confirmer pour valider les demandes de rendez-vous soumises par les patients. Ces actions sont liées de manière à ce que l'agent doive d'abord consulter les rendez-vous avant de pouvoir les imprimer ou les confirmer, et chacune de ces actions vient compléter et étendre la fonctionnalité de consultation des rendez-vous.

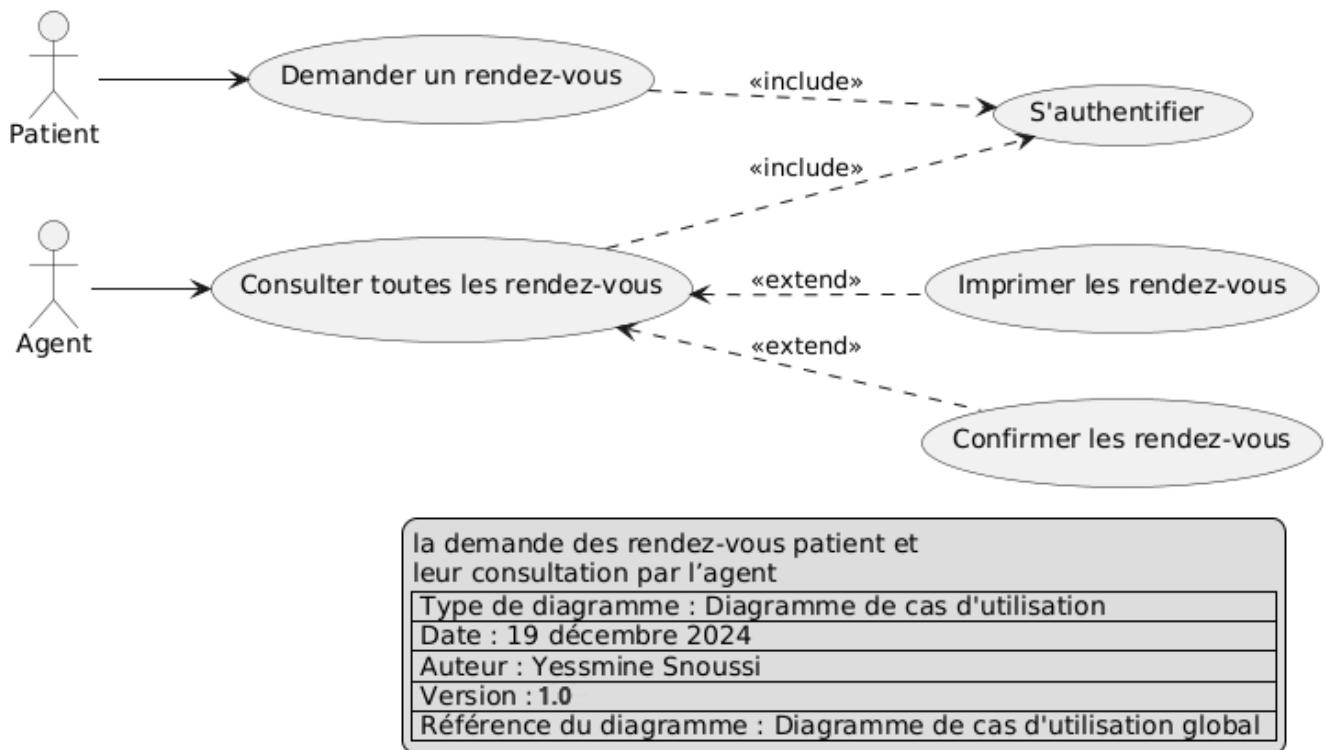


FIGURE 13 – Diagramme de cas d'utilisation pour la gestion des rendez-vous patient et leur consultation par l'agent

Diagramme de Classe

Le système est conçu pour faciliter la gestion des rendez-vous entre les patients et les médecins. Chaque patient, structuré par des informations personnelles telles que son nom, prénom, téléphone et email, peut soumettre une demande de rendez-vous via l'application après avoir fourni ces informations et s'être authentifié pour garantir la sécurité et la validité de la demande. Une fois la demande soumise, elle est enregistrée sous forme de rendez-vous, qui contient des informations essentielles telles que la date et l'état, qui peut être "en attente" par défaut. Un agent a la possibilité de consulter la liste des rendez-vous, de vérifier les demandes, de confirmer un rendez-vous et d'imprimer une version papier des rendez-vous. Le médecin, de son côté, dispose de ses propres informations personnelles (nom, spécialité, téléphone, email) pour que les patients puissent facilement le contacter et prendre un rendez-vous. Chaque patient peut soumettre plusieurs demandes de rendez-vous, chaque rendez-vous est associé à un seul patient et un seul médecin, et peut être confirmé ou rester en attente. Un agent peut consulter et imprimer les rendez-vous après leur confirmation, tandis qu'un médecin peut être lié à plusieurs rendez-vous.

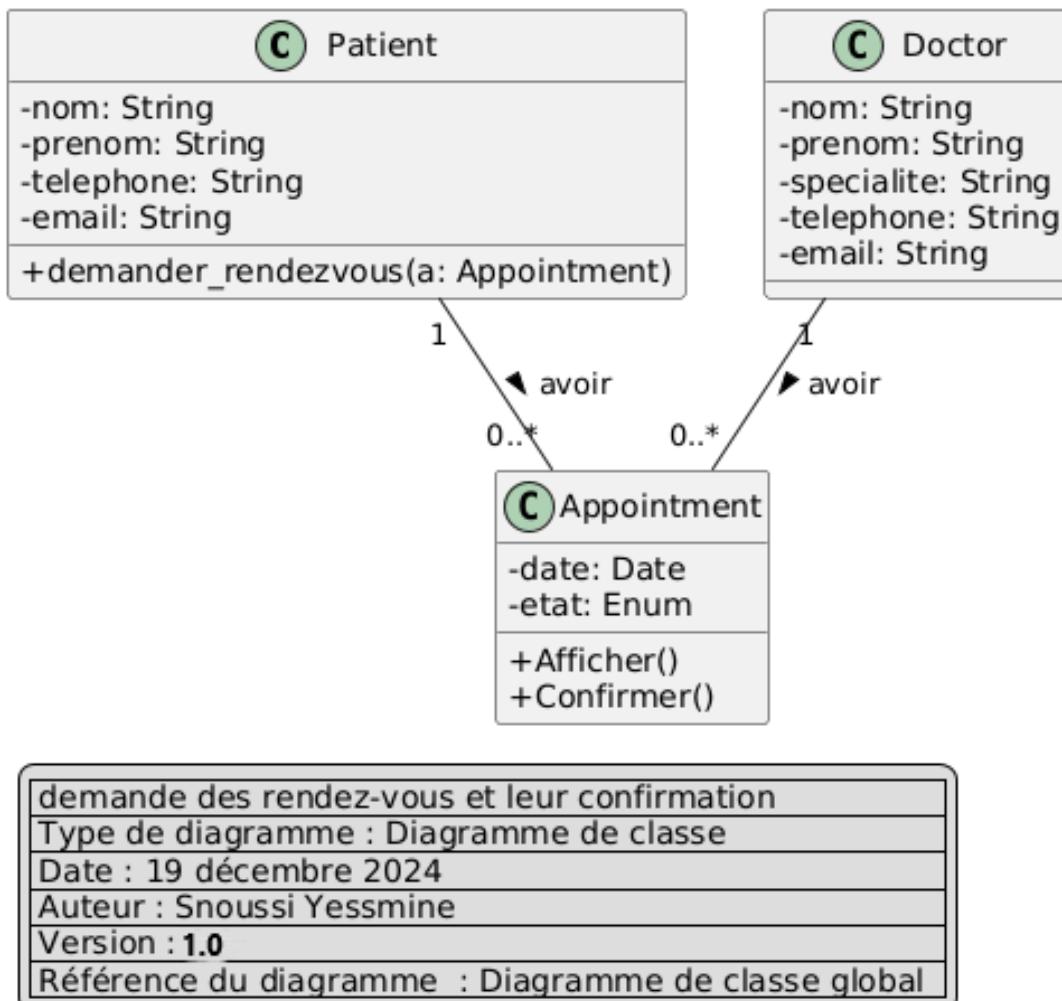
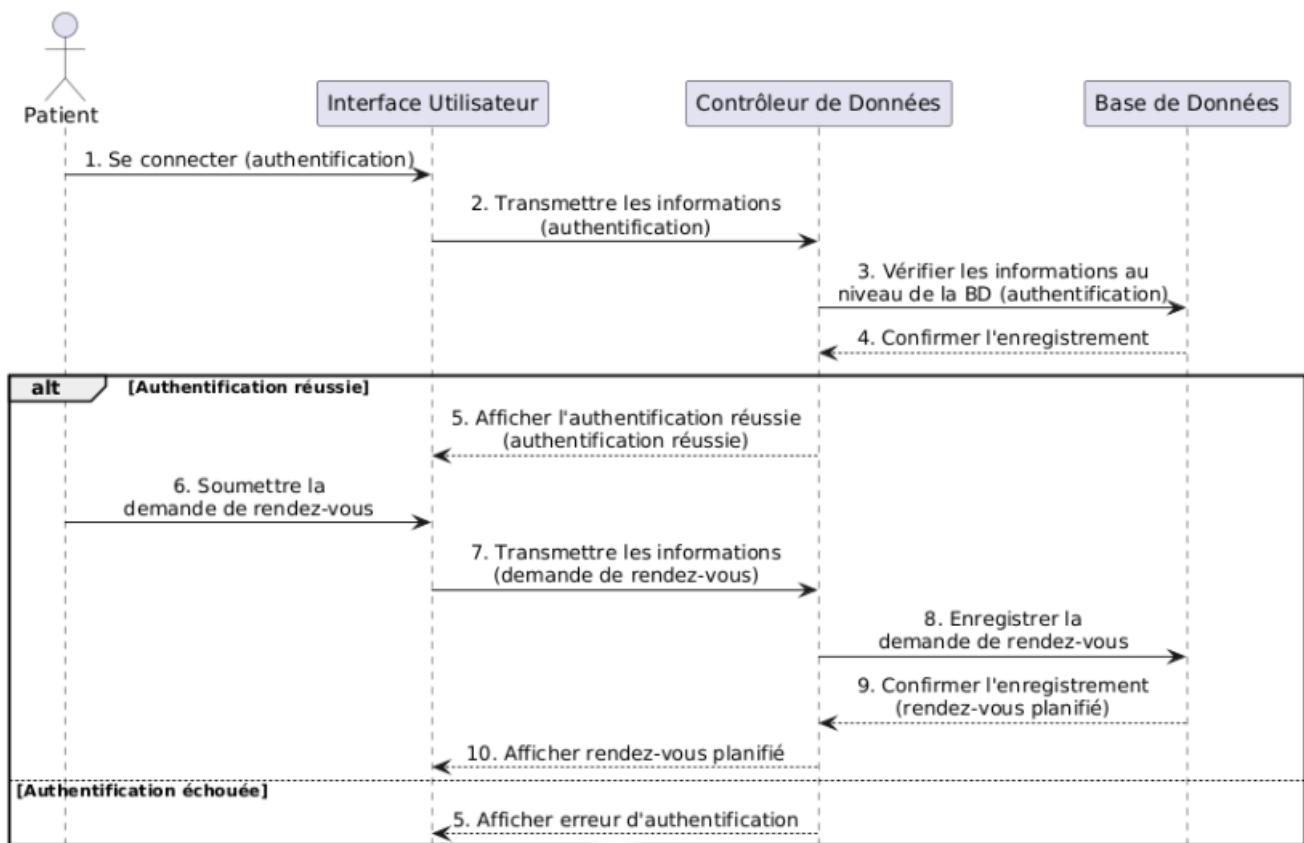


FIGURE 14 – Diagramme de classe pour les demandes des rendez-vous et leur confirmation

Diagramme de séquence

Le diagramme de séquence illustre le processus d'authentification d'un patient et de demande de rendez-vous dans un système de gestion. Le patient commence par se connecter via l'interface utilisateur, qui transmet les informations d'authentification au contrôleur de données. Ce dernier vérifie les informations du patient en consultant la base de données. Si la vérification est réussie, l'interface affiche un message confirmant l'authentification. Une fois authentifié, le patient peut soumettre une demande de rendez-vous, en choisissant un médecin disponible selon ses préférences et les horaires proposés. Cette demande est envoyée à l'interface, puis transmise au contrôleur, qui l'enregistre dans la base de données. Après l'enregistrement, la base de données confirme le rendez-vous au contrôleur, et ce dernier informe l'interface que le rendez-vous a été ajouté avec succès. Ce flux d'actions assure une gestion fluide et sécurisée des demandes de rendez-vous des patients dans le système.



Authentification du patient et Demande de rendez-vous	
Type de diagramme :	Diagramme de séquence
Date :	19 décembre 2024
Auteur :	Yessmine Snoussi
Version :	1.0

FIGURE 15 – Diagramme de séquence du prise du rendez-vous du patient

Diagramme de d'activité

Le diagramme d'activité décrit le processus du consultation des rendez-vous par l'agent d'accueil dans un système de gestion. Après l'authentification, l'agent sélectionne la date pour afficher les rendez-vous programmés. Une fois la date choisie, l'agent consulte les rendez-vous disponibles. À ce moment, deux actions sont possibles : il peut soit confirmer un rendez-vous, notifier le patient et revenir ensuite à la consultation des rendez-vous, soit générer et imprimer la liste des rendez-vous, puis revenir à la consultation. Si l'agent n'est pas authentifié, un message d'erreur s'affiche pour l'en informer. Ce diagramme montre ainsi un flux d'activités simple et fluide, où l'agent d'accueil peut gérer les rendez-vous en toute efficacité, en revenant à l'étape de consultation après chaque action effectuée, que ce soit pour la confirmation des rendez-vous ou pour l'impression de la liste.

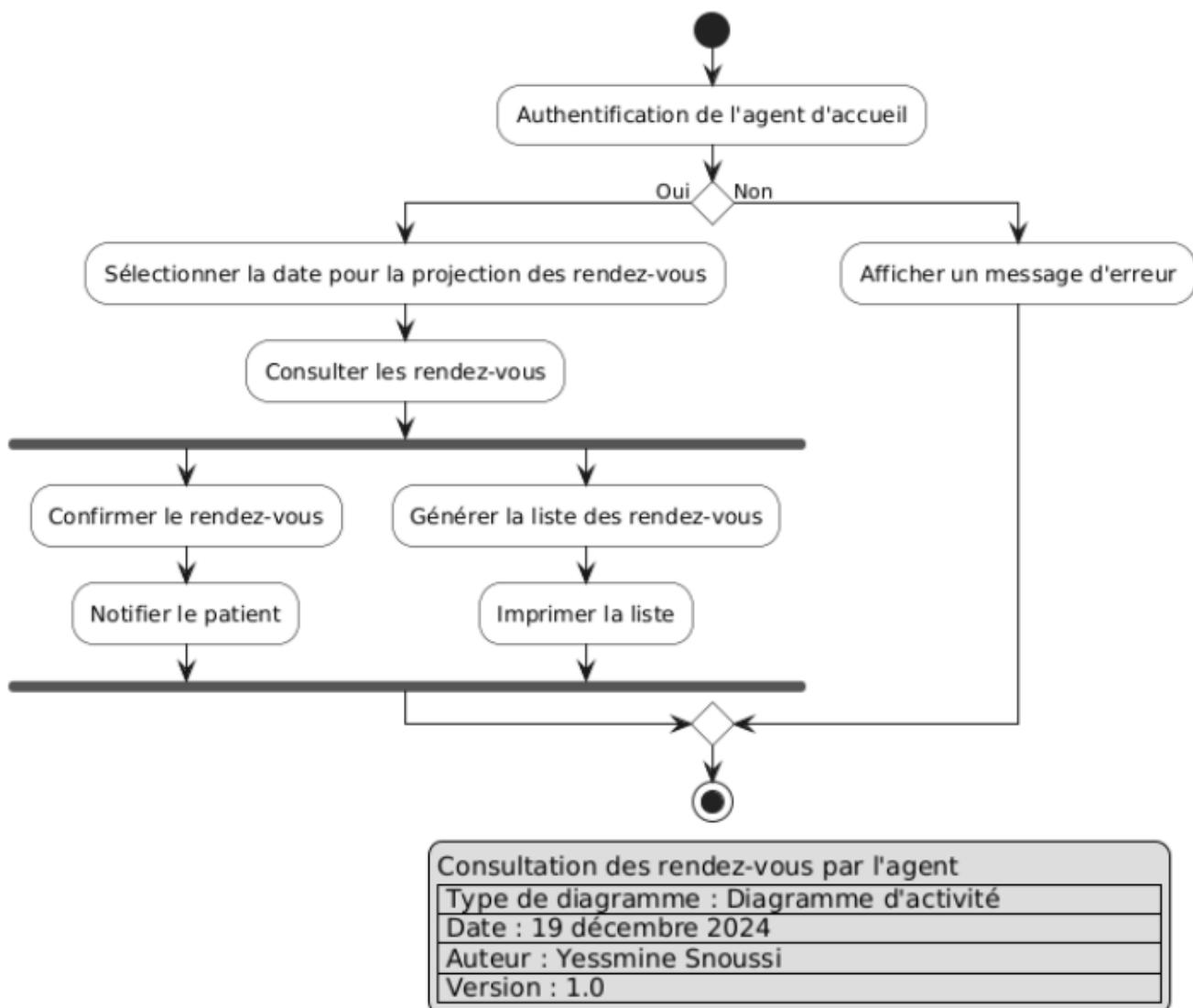


FIGURE 16 – Diagramme d'activité du Consultation des rendez-vous de l'agent

1.4 Les interfaces

— Partie Agent :

L'interface de connexion de l'agent d'accueil comprend un champ pour l'email, un autre pour le mot de passe, et un bouton **show password** pour afficher le mot de passe saisi. Un bouton **Login** permet de valider les informations et de procéder à l'authentification. Si les données sont correctes, l'agent accède à l'interface principale du système de gestion des rendez-vous. En cas d'erreur, un message s'affiche. Cette interface est conçue pour être simple, sécurisée et facile à utiliser, offrant une expérience fluide.

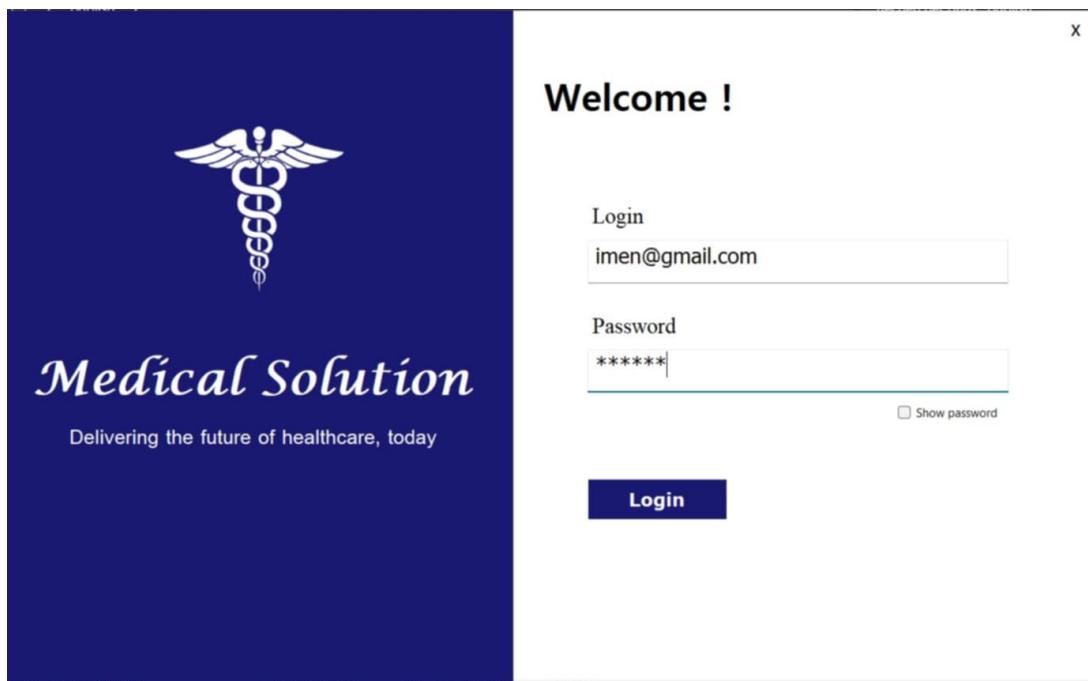


FIGURE 17 – Interface de connexion pour les agents

LISTE DES TABLEAUX

L'interface qui suit la connexion de l'agent d'accueil contient un menu principal avec plusieurs éléments. Ce menu inclut les options suivantes : “**Home**”, “**List of Doctors**”, “**Appointment Request**”, et “**Logout**”. Lorsque l'agent clique sur “**List of Doctors**”, une nouvelle page s'affiche, contenant la liste de tous les docteurs disponibles dans le système. Pour chaque docteur, les informations suivantes sont présentes : son nom, sa spécialité, son email, et son numéro de téléphone. En plus de ces informations, un bouton “**Check Calendar**” est présent pour chaque docteur. En cliquant sur ce bouton, l'agent peut consulter le calendrier de ce docteur et voir tous les rendez-vous programmés pour lui. Cette fonctionnalité permet à l'agent d'accéder facilement aux informations des docteurs et de gérer les rendez-vous. Le menu permet aussi de revenir à la page d'accueil en cliquant sur “**Home**” ou de se déconnecter en utilisant l'option “**Logout**”. L'interface est intuitive et organisée pour faciliter la navigation et la gestion des docteurs et des rendez-vous.

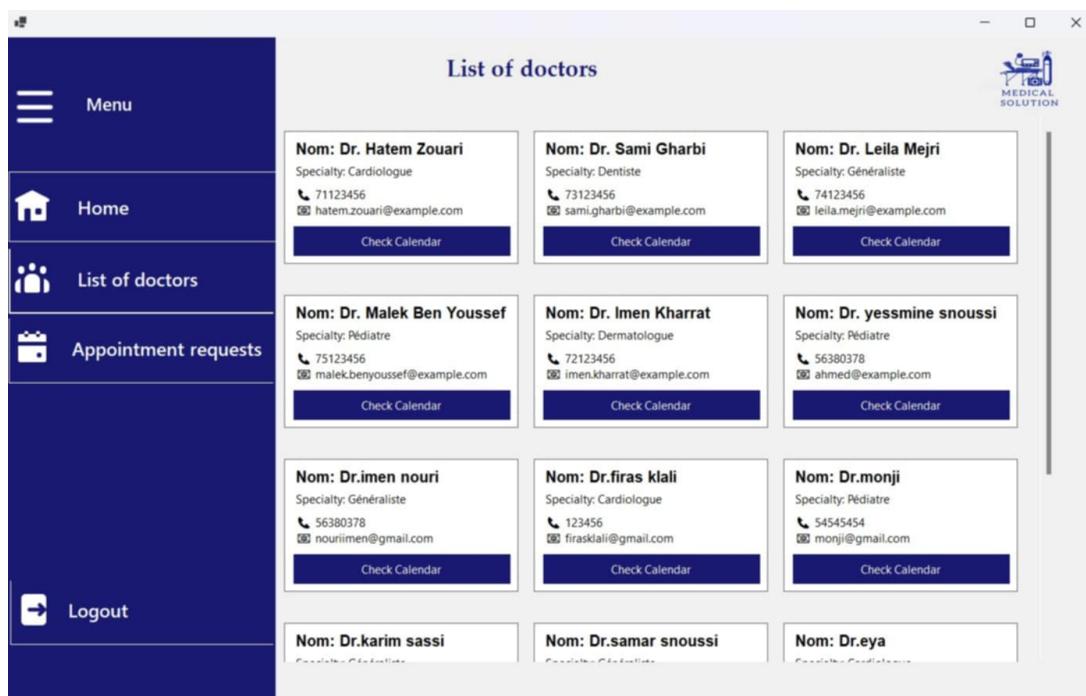


FIGURE 18 – Affichage des médecins et gestion de leur calendrier

LISTE DES TABLEAUX

La section **Appointment Request** de l'interface est dédiée à la gestion des rendez-vous en attente de confirmation. Elle affiche une liste des rendez-vous avec les informations essentielles : le nom des patients, la date, et l'heure de chaque rendez-vous. Pour chaque rendez-vous, un bouton **Confirmer** est disponible. Lorsque l'utilisateur clique sur ce bouton, le rendez-vous est validé et automatiquement retiré de la liste d'attente. Cette fonctionnalité permet de traiter efficacement les demandes en attente et de maintenir une liste des rendez-vous organisée et à jour.



FIGURE 19 – Liste des rendez-vous en attente de validation

Le fichier PDF généré après avoir cliqué sur “Imprimer” contient les informations des rendez-vous confirmés pour la date sélectionnée. On y trouve le nom du docteur ainsi que les détails des rendez-vous confirmés, notamment les noms des patients, les dates, et les heures correspondantes. Ce document permet d'avoir un aperçu clair et précis des rendez-vous planifiés pour le jour choisi.



FIGURE 20 – Génération d'un fichier PDF pour les rendez-vous confirmés

LISTE DES TABLEAUX

L'interface de l'agent permet de consulter les rendez-vous confirmés d'un médecin sélectionné via un bouton "*Check Calendar*". Après avoir choisi un médecin, l'agent peut visualiser les rendez-vous confirmés, incluant le nom du patient, la date et l'heure. Une zone de sélection de date permet de filtrer les rendez-vous selon le jour choisi, et un bouton "*Imprimer*" génère un rapport imprimable de la liste des rendez-vous pour cette date. Cette interface offre une navigation fluide, permettant à l'agent de gérer efficacement les rendez-vous et d'obtenir une version papier du planning.

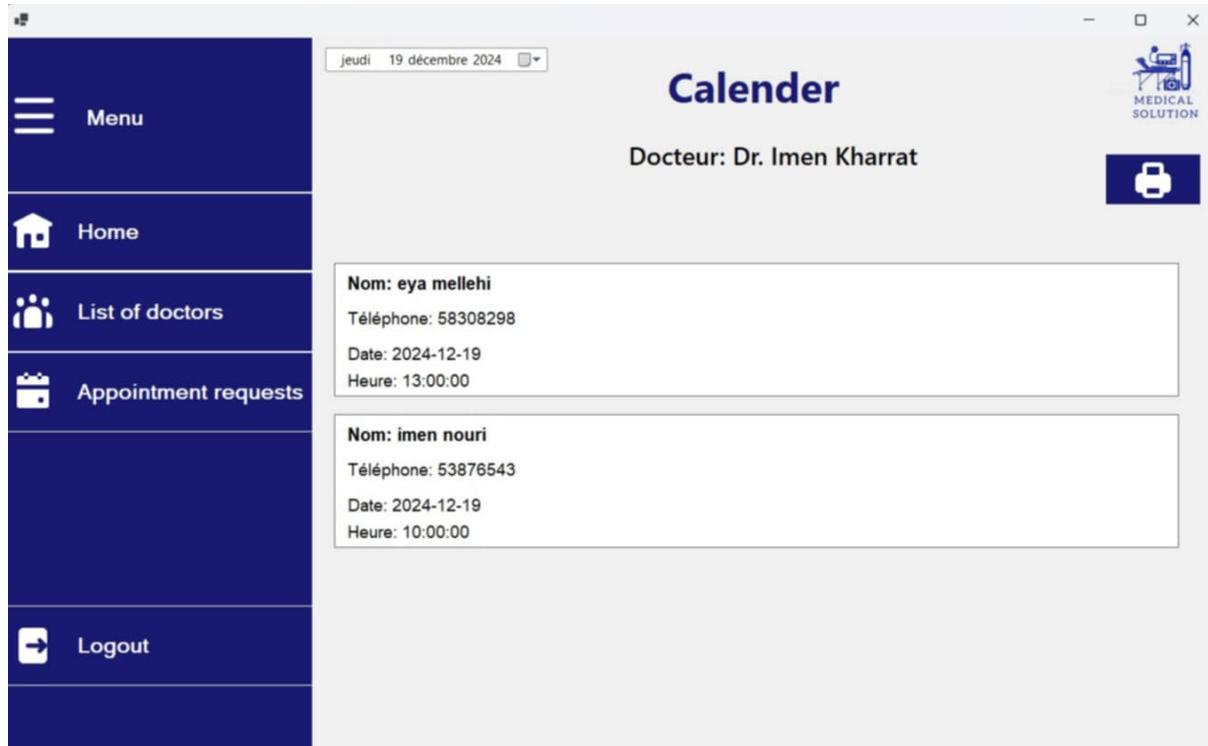


FIGURE 21 – Génération d'un fichier PDF pour les rendez-vous confirmés

LISTE DES TABLEAUX

— Partie Patient :

L'interface d'inscription du patient permet de créer un compte en remplissant les champs suivants : nom complet, adresse email, numéro de téléphone et mot de passe. Un bouton **Signup** valide les informations saisies. Si elles sont correctes, le compte est créé, sinon un message d'erreur est affiché. L'interface est simple et sécurisée, assurant une inscription fluide.

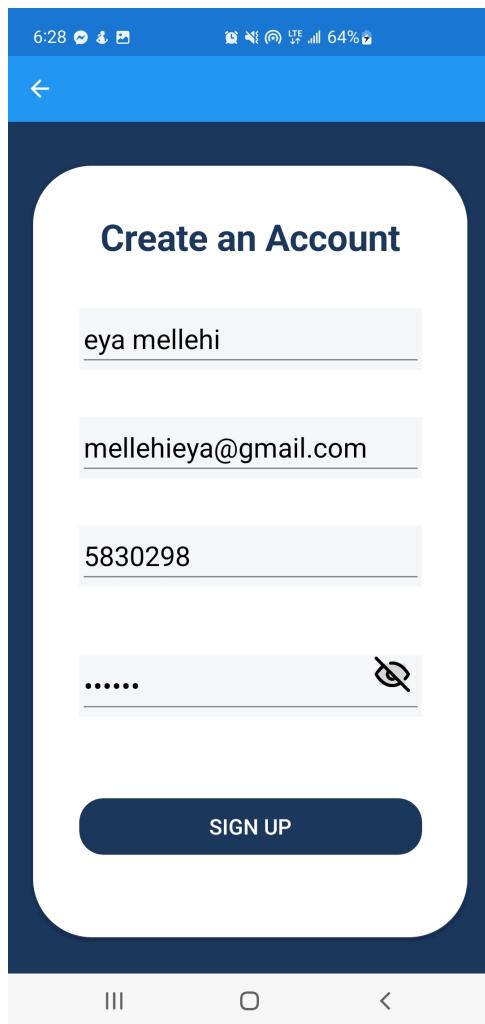


FIGURE 22 – Interface d'inscription

LISTE DES TABLEAUX

L'interface de connexion permet aux utilisateurs de se connecter en entrant leur adresse email et leur mot de passe dans des champs dédiés. Un bouton **Login** est présent pour soumettre les informations. Si l'utilisateur n'a pas encore de compte, un lien **Don't have an account ? Sign up here** est disponible pour le rediriger vers la page d'inscription. Cette interface est simple et sécurisée, assurant une connexion rapide et facile pour les patients.

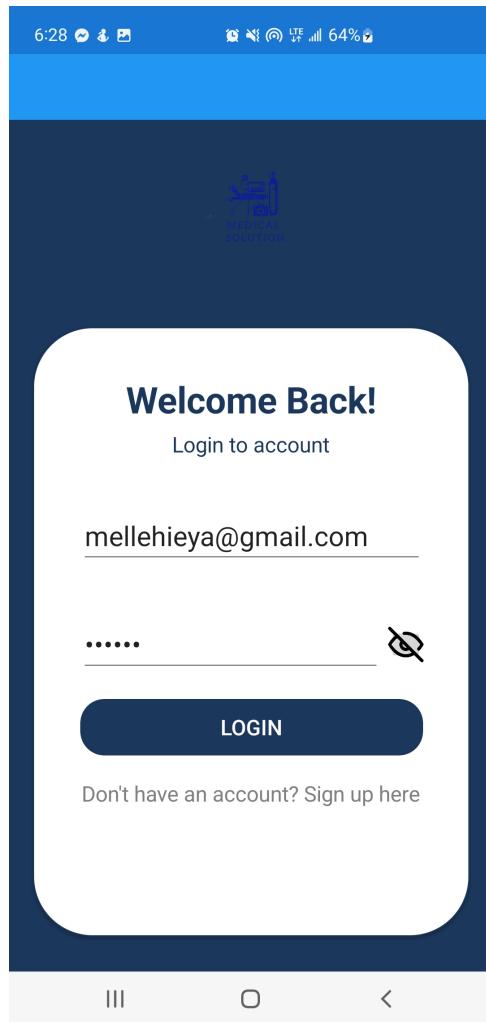


FIGURE 23 – Interface d'inscription

LISTE DES TABLEAUX

La première interface du patient affiche une liste de docteurs triée par spécialité. Chaque docteur est présenté avec son nom et sa spécialité, accompagné d'un bouton *Make Appointment* permettant au patient de prendre un rendez-vous avec ce docteur. En haut de l'interface, un bouton *Check My Appointment* est également présent. Ce bouton permet au patient de consulter ses rendez-vous programmés et de vérifier les informations relatives à ses rendez-vous. L'interface est conçue pour offrir une navigation intuitive et facile, permettant au patient de prendre rendez-vous ou de vérifier ses rendez-vous passés rapidement.

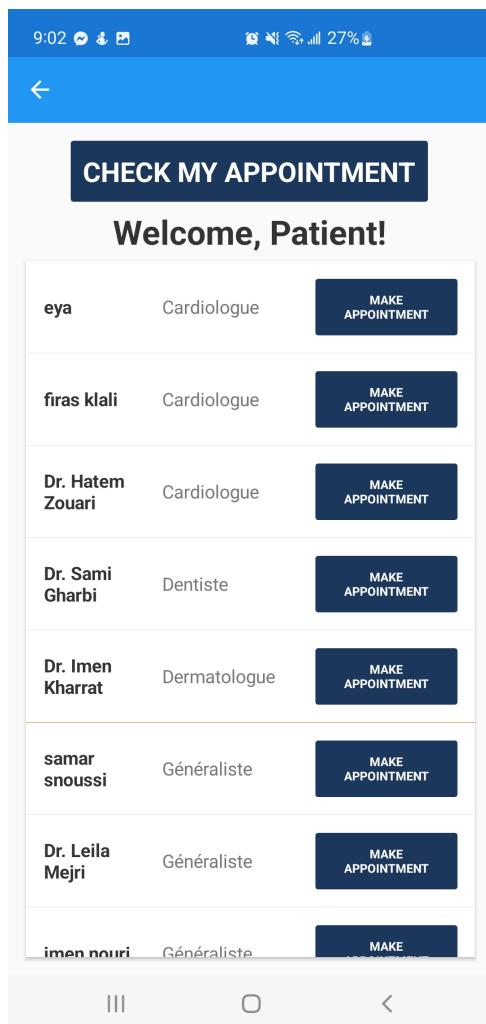


FIGURE 24 – Liste des médecins disponibles

LISTE DES TABLEAUX

Lorsque le patient clique sur le bouton *Make Appointment*, l'interface affiche les disponibilités du docteur sélectionné. Les plages horaires disponibles sont présentées, permettant au patient de choisir la date et l'heure qui lui conviennent. Une fois la date sélectionnée, le patient peut soumettre la demande de rendez-vous en cliquant sur le bouton *Submit Appointment*. Ce bouton permet d'ajouter le rendez-vous choisi dans le système, en associant la date et l'heure avec le docteur sélectionné. L'interface est simple et fluide, garantissant une gestion facile des rendez-vous pour le patient.

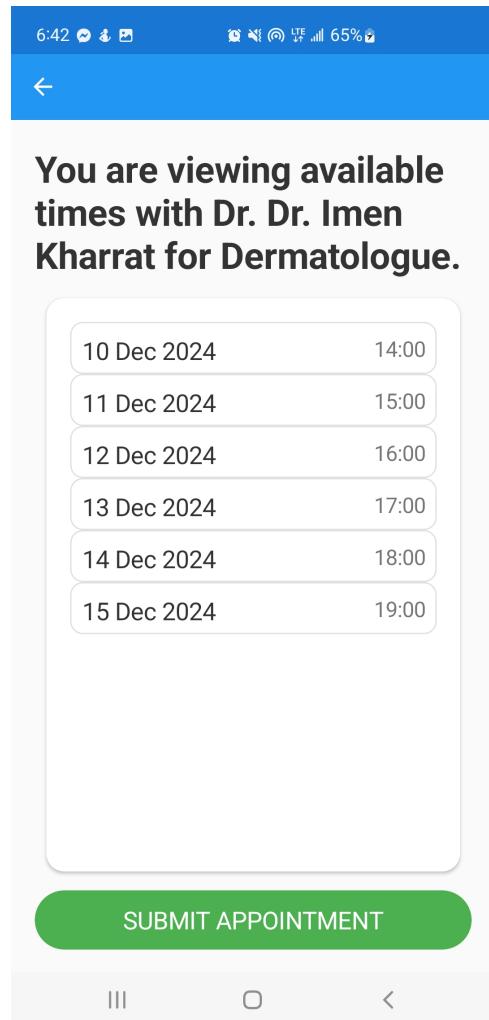


FIGURE 25 – Formulaire de demande de rendez-vous

LISTE DES TABLEAUX

Lorsque le patient clique sur le bouton *Submit Appointment*, un message de réussite s'affiche pour confirmer que le rendez-vous a été ajouté avec succès. Ce message informe le patient que la demande de rendez-vous a été enregistrée et que le rendez-vous est maintenant programmé pour la date et l'heure choisies.

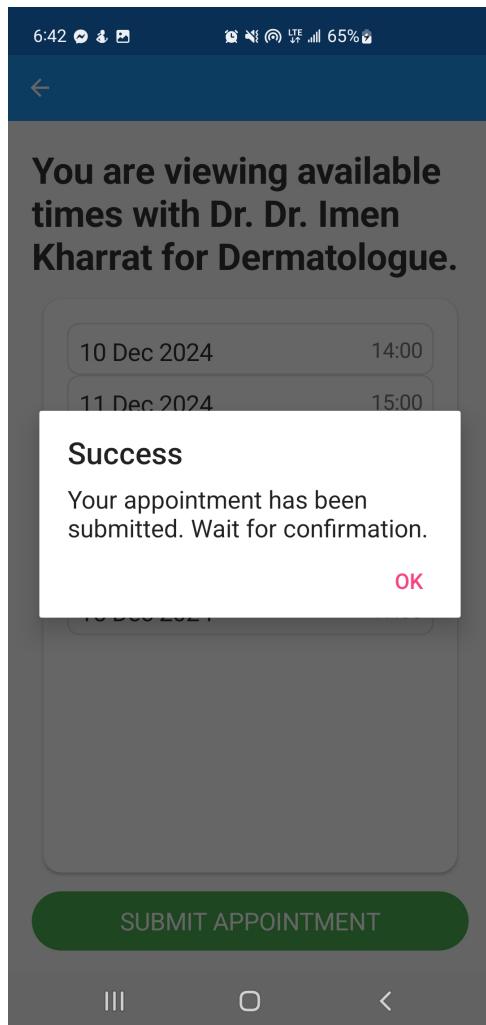


FIGURE 26 – Message de confirmation après une demande de rendez-vous

1.5 Plan de test

Authentification et Gestion des Rendez-vous par les agents

Scénarios de Tests

Id scénario	Description	Type de test
SC001	Authentification des agents	Test unitaire
SC002	Demande de confirmation des rendez-vous	Test d'intégration
SC004	Impression des rendez-vous confirmés	Test d'intégration

LISTE DES TABLEAUX

Suite 1 : Authentification des Agents

Objet de Test

- * Fonctionnalité d'authentification des agents.

Propriété à Tester

- * Validation des informations de connexion (email et mot de passe).
- * Accès à l'interface principale après authentification.

Données de Test

Nom d'utilisateur	Mot de passe
agentvalid	correct123
agentinvalid	wrongpass

Cas de Tests

ID Cas	Étapes	Résultat Attendu
TC S1001	1. Entrer un nom d'utilisateur valide. 2. Entrer un mot de passe valide. 3. Cliquer sur "Login".	Authentification réussie, redirection vers l'interface principale.
TC S1002	1. Entrer un nom d'utilisateur valide. 2. Entrer un mot de passe invalide. 3. Cliquer sur "Login".	Message d'erreur : "Identifiants incorrects".
TC S1003	1. Entrer un nom d'utilisateur invalide. 2. Entrer un mot de passe valide. 3. Cliquer sur "Login".	Message d'erreur : "Identifiants incorrects".
TC S1004	1. L'agent s'authentifie avec un nom d'utilisateur et un mot de passe valides. 2. Cliquer sur "Consulter les rendez-vous".	L'agent accède à la liste des rendez-vous.

LISTE DES TABLEAUX

Suite 2 : Gestion des Rendez-vous

Objet de Test

* Gestion des rendez-vous (confirmation, impression).

Propriété à Tester

* Confirmation d'un rendez-vous.

* Impression des rendez-vous confirmés.

Données de Test

Rendez-vous à confirmer	Rendez-vous confirmés
Patient1, 12/12/2024, 10 :00	Patient2, 13/12/2024, 11 :00

Cas de Tests

ID Cas	Étapes	Résultat Attendu
TC S2001	1. L'agent consulte la liste des rendez-vous. 2. Confirmer un rendez-vous en attente. 3. Cliquer sur "Confirmer".	Le rendez-vous est marqué comme confirmé et disparaît de la liste.
TC S2002	1. L'agent consulte la liste des rendez-vous. 2. Cliquer sur "Imprimer".	Une version imprimée des rendez-vous confirmés est générée.

Authentification et Demande de Rendez-vous du Patient

Scénarios de Tests

Id scénario	Description	Type de test
SC001	Authentification des patients	Test unitaire
SC002	Demande de rendez-vous	Test d'intégration

Suite 1 : Authentification des Patients

Objet de Test

* Fonctionnalité de connexion des patients.

Propriétés à Tester

* Validation des informations d'identification.

Données de Test

LISTE DES TABLEAUX

Nom d'utilisateur	Mot de passe
patient_valide	correct123
patient_invalide	erreur456

Cas de Tests

ID Cas	Étapes	Résultat Attendu
TC S1001	1. Entrer un nom d'utilisateur valide. 2. Entrer un mot de passe valide. 3. Cliquer sur "Se connecter".	Connexion réussie, redirection au tableau de bord.
TC S1002	1. Entrer un nom d'utilisateur valide. 2. Entrer un mot de passe invalide. 3. Cliquer sur "Se connecter".	Message d'erreur : "Identifiants incorrects".

Suite 2 : Demande de Rendez-vous

Objet de Test

* Soumission de demandes de rendez-vous par les patients.

Propriétés à Tester

* Vérification de la soumission correcte d'une demande de rendez-vous.

* Validation des informations saisies dans la demande.

Données de Test

Nom du Patient	Date du Rendez-vous
Patient1	12/12/2024 10 :00
Patient2	13/12/2024 11 :00

Cas de Tests

LISTE DES TABLEAUX

ID Cas	Étapes	Résultat Attendu
TC S2001	1. Remplir le formulaire de demande de rendez-vous. 2. Cliquer sur "Soumettre".	La demande de rendez-vous est enregistrée et affichée dans la liste des rendez-vous en attente.
TC S2002	1. Entrer des informations incorrectes dans le formulaire. 2. Cliquer sur "Soumettre".	Message d'erreur : "Les informations sont invalides".

1.6 Bilan du Sprint

LISTE DES TABLEAUX

Section	Détails
1. Introduction	<ul style="list-style-type: none"> - Date du sprint : (du 04/10/2024 au 18/10/2024). - Objectifs du sprint : Développer la fonctionnalité de gestion des rendez-vous et implémenter l'authentification des patients et agents.
2. Travail accompli	<ul style="list-style-type: none"> - Tâches terminées : <ol style="list-style-type: none"> 1. Création de l'interface de connexion. 2. Implémentation de la base de données pour stocker les utilisateurs. 3. Développement des interfaces de l'agent et patient de gestion des rendez-vous. - Tâches en cours : <ol style="list-style-type: none"> 1. l'impression de la liste des rendez-vous
3. Difficultés rencontrées	<ul style="list-style-type: none"> - Problèmes techniques : Difficulté d'intégration de la base de données avec l'application mobile.
4. Solutions apportées	<ul style="list-style-type: none"> - Mesures prises : <ol style="list-style-type: none"> 1. Correction des erreurs d'intégration de la base de données avec la mise en place de scripts SQL supplémentaires. 2. Clarification des spécifications de la fonctionnalité lors des réunions.
5. Améliorations à apporter	<ul style="list-style-type: none"> - Suggestions d'amélioration : <ol style="list-style-type: none"> 1. Améliorer la gestion des erreurs lors de l'authentification. 2. Optimiser l'interface mobile pour une meilleure expérience utilisateur. 3. Ajouter des tests unitaires pour les fonctionnalités de gestion des rendez-vous.
6. Prochaines étapes	<ul style="list-style-type: none"> - Tâches à venir : <ol style="list-style-type: none"> 1. Finaliser la fonctionnalité de notification des rendez-vous. 2. Terminer les tests d'intégration. 3. Implémenter la gestion des rôles pour l'accès aux rendez-vous. - Points à finaliser : <ol style="list-style-type: none"> 1. Correction d'impression de la liste. 2. Finalisation de l'interface utilisateur.
7. Conclusion	<ul style="list-style-type: none"> - Bilan global du sprint : Le sprint a permis d'implémenter les fonctionnalités clés du projet, à savoir l'authentification et la gestion des rendez-vous par l'agent. Cependant il reste l'amélioration de l'interface patient.

TABLE 4 – Bilan du Sprint 1

2 Sprint 2 :Authentification et consultation des Rendez-vous par les Secrétaires Médicales

2.1 Objectif du Sprint

Le Sprint 2 a pour objectif principal de mettre en place les fonctionnalités suivantes :

- Authentification des secrétaires médicales pour leur permettre de consulter les rendez-vous.
- Consultation et demande de décalage des rendez-vous par les patients après authentification.

LISTE DES TABLEAUX

- Confirmation des rendez-vous par les agents d'accueil, avec envoi de notifications aux patients.
- Correction de la fonctionnalité d'impression des rendez-vous par jour et par cabinet pour les agents d'accueil.

2.2 Backlog du sprint 2

User Story	Tâche	Temps estimé	Responsable
User Story 1 : En tant que secrétaire médicale, je veux m'authentifier pour accéder à la consultation des rendez-vous.	Concevoir l'interface d'authentification.	1 jour	Imen Nouri
	Implémenter la logique d'authentification dans l'application.	1 jour	Imen Nouri
	Tester le parcours complet de l'authentification.	1 jour	Imen Nouri
User Story 2 : En tant que secrétaire médicale, je veux ajuster les rendez-vous en fonction des changements des médecins ou des imprévus.	Implémenter les modifications dans la base de données et l'API.	0.5 jour	Imen Nouri
	Concevoir l'interface permettant de modifier les rendez-vous.	0.5 jour	Imen Nouri
	Valider l'intégration avec les autres fonctionnalités liées.	1 jour	Imen Nouri
User Story 3 : En tant que patient, je veux consulter des notifications si mon rendez-vous est modifié par la secrétaire médicale.	Ajouter une fonctionnalité de notification en temps réel pour les rendez-vous.	1 jour	Eya Mellehi
	Implémenter la logique de notification dans la base de données et l'API.	1 jour	Eya Mellehi
	Tester le système de notification pour différents types de modifications.	1 jour	Eya Mellehi

LISTE DES TABLEAUX

User Story	Tâche	Temps estimé	Responsable
User Story 4 : En tant qu'agent d'accueil, dès que je confirme un rendez-vous, une notification sera envoyée aux patients.	Implémenter la logique d'envoi de notifications après confirmation.	1 jour	Eya Mellehi
	Tester le flux complet de confirmation et d'envoi de notification.	1 jour	Eya Mellehi
User Story 5 : Correction de l'impression des rendez-vous par jour et cabinet.	Corriger les erreurs dans le fichier PDF généré.	0.5 jour	Imen Nouri
	Tester la fonctionnalité d'impression avec des filtres (par jour et par cabinet).	0.5 jour	Imen Nouri

TABLE 5: Backlog de sprint 2

2.3 Conception

Diagramme de cas d'utilisation

Le système de gestion des rendez-vous repose sur trois acteurs principaux : le secrétaire, l'agent et le patient. Le secrétaire est responsable de la gestion des rendez-vous, pouvant les consulter et les décaler en cas de besoin. Il est également en charge d'envoyer des notifications aux patients concernant les modifications de leurs rendez-vous. L'agent, quant à lui, confirme les rendez-vous et imprime les fiches des rendez-vous, qui fournissent des informations détaillées sur les consultations planifiées. Enfin, le patient consulte ses notifications, qui l'informent des changements relatifs à ses rendez-vous.

Pour garantir l'accès à ces fonctionnalités, tous les utilisateurs (secrétaire, agent, patient) doivent d'abord s'authentifier dans le système. L'authentification est donc un prérequis essentiel pour accéder à toutes les fonctionnalités disponibles. Certaines actions, telles que la consultation des rendez-vous, l'impression des fiches ou la consultation des notifications, incluent systématiquement l'étape d'authentification. De plus, la modification des rendez-vous et leur confirmation impliquent l'envoi automatique de notifications aux patients pour les tenir informés des changements. Ce système assure ainsi une gestion fluide des rendez-vous tout en garantissant la sécurité d'accès et la communication en temps réel avec les patients.

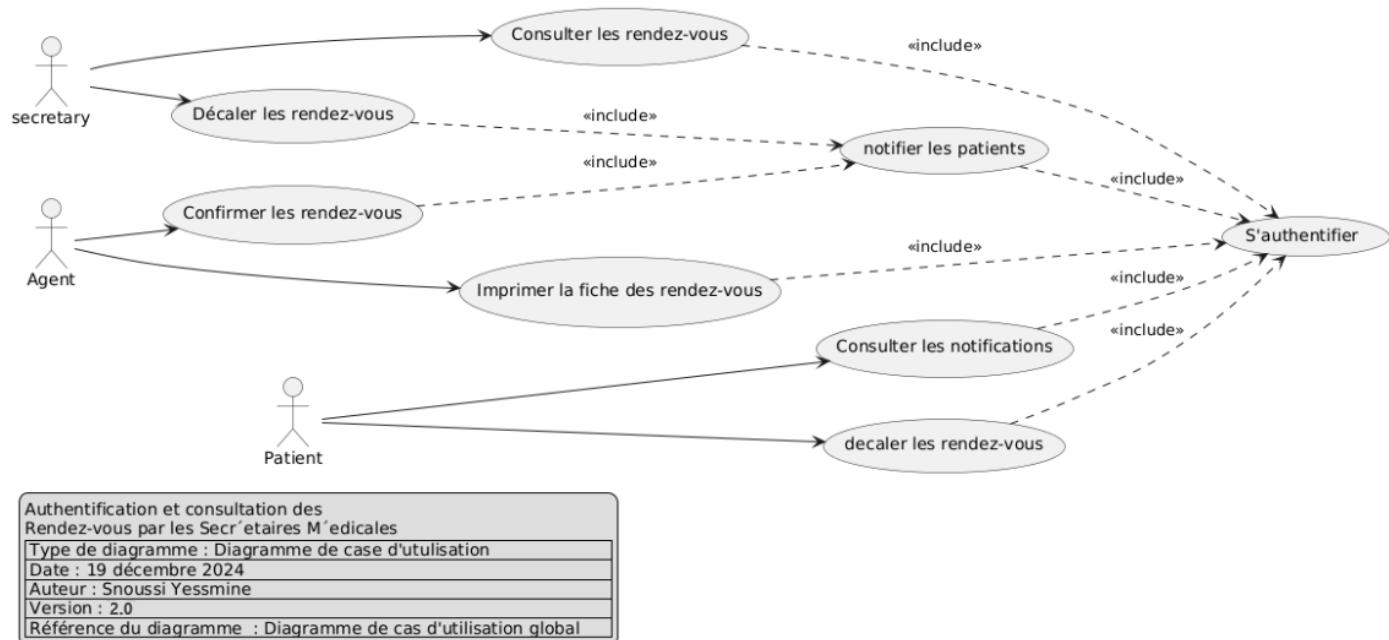


FIGURE 27 – Diagramme de cas d'utilisation d'Authentification et consultation des Rendez-vous par les Secrétaires Médicales

Diagramme de classes

Le système permet aux patients de prendre des rendez-vous avec des médecins en fonction de leurs disponibilités. Chaque patient dispose d'un profil contenant ses informations personnelles telles que son nom, prénom, téléphone et email. Lorsqu'un patient souhaite prendre un rendez-vous, il peut consulter les créneaux disponibles des médecins. Les informations des médecins sont également stockées, incluant leur nom, prénom, spécialité et leurs coordonnées. Chaque médecin définit ses disponibilités, comprenant la date et l'heure auxquelles il peut recevoir des patients. Le patient peut alors demander un rendez-vous, qui sera soumis à la confirmation de l'agent, en fonction de la disponibilité du médecin. Une fois la demande validée, le rendez-vous est confirmé et peut être affiché dans le système. Ce processus assure une gestion fluide et organisée des prises de rendez-vous en fonction des créneaux horaires des médecins.

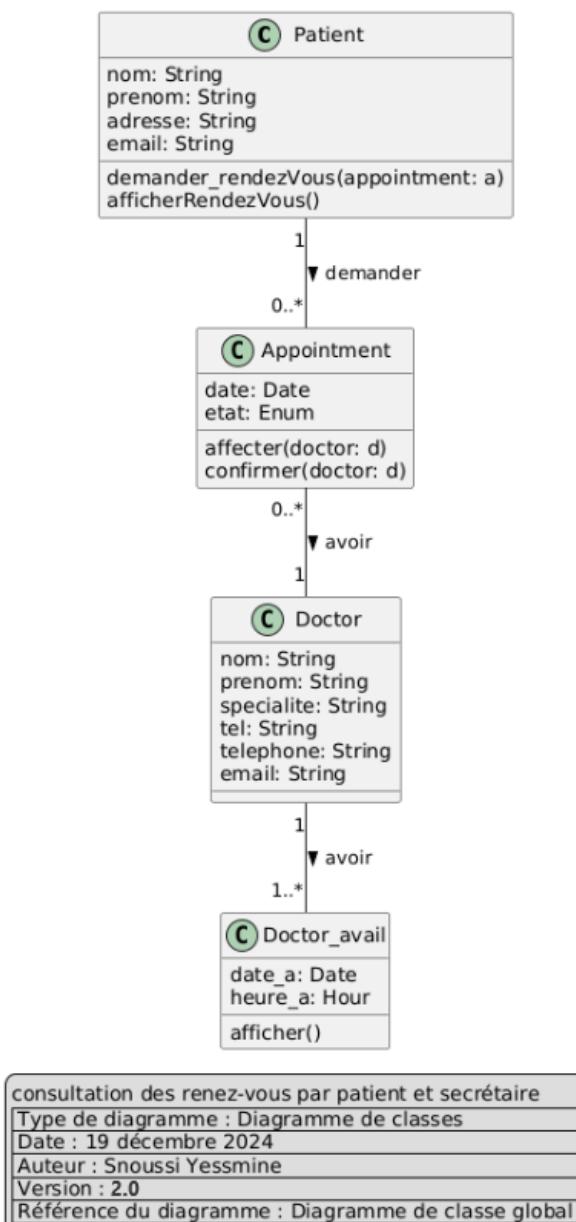


FIGURE 28 – Diagramme de classes

Diagramme de séquence

Le processus commence lorsque la secrétaire se connecte à l'interface utilisateur du système en saisissant ses identifiants. Une fois ces identifiants vérifiés, le système authentifie la secrétaire. Après l'authentification, la secrétaire a accès à une interface qui lui permet de consulter les rendez-vous existants. Si la secrétaire souhaite modifier un rendez-vous, par exemple en raison d'un changement de disponibilité du médecin ou d'une autre contrainte, elle peut demander un décalage de ce rendez-vous. Pour cela, elle sélectionne le rendez-vous à décaler et l'indique via l'interface. Une fois la demande de décalage effectuée, le système vérifie cette demande et met à jour la base de données en fonction des nouvelles informations de rendez-vous. La base de données confirme la mise à jour et le système génère ensuite une notification pour informer le patient de la modification du rendez-vous. Enfin, la confirmation du décalage est renvoyée à la secrétaire via l'interface utilisateur, ce qui lui permet de savoir que l'action a été correctement effectuée et que la notification a été envoyée. Ainsi, le système assure une gestion fluide des rendez-vous, en permettant à la secrétaire d'effectuer des actions telles que la consultation, la modification (décalage) des rendez-vous et la communication avec les patients via des notifications.

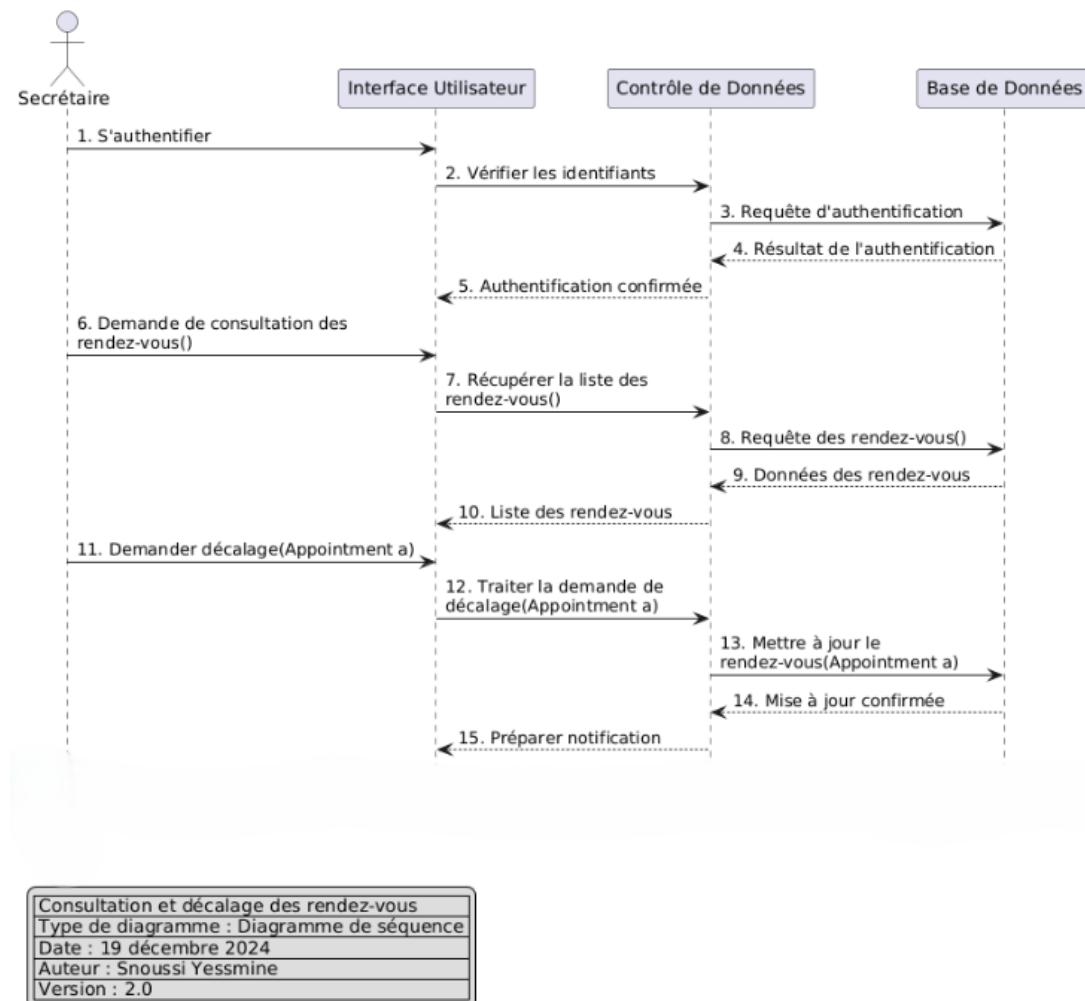


FIGURE 29 – Diagramme de séquence du consultation et décalage des rendez-vous

Diagramme d'activité

Le processus commence par l'authentification de la secrétaire dans le système. Si l'authentification réussit, elle accède à une interface lui permettant de consulter la liste des rendez-vous existants. Elle peut alors sélectionner un rendez-vous spécifique qu'elle souhaite modifier. En cas de besoin de reprogrammation, une nouvelle date est choisie. Si cette nouvelle date est disponible, le système envoie automatiquement un email au patient pour l'informer du changement. En revanche, si la date n'est pas disponible, la secrétaire doit sélectionner une autre date. Si l'authentification échoue, le système affiche un message d'erreur, informant la secrétaire qu'elle doit réessayer ou vérifier ses identifiants. Ainsi, ce processus assure une gestion efficace des décalages de rendez-vous tout en maintenant une communication fluide avec les patients.

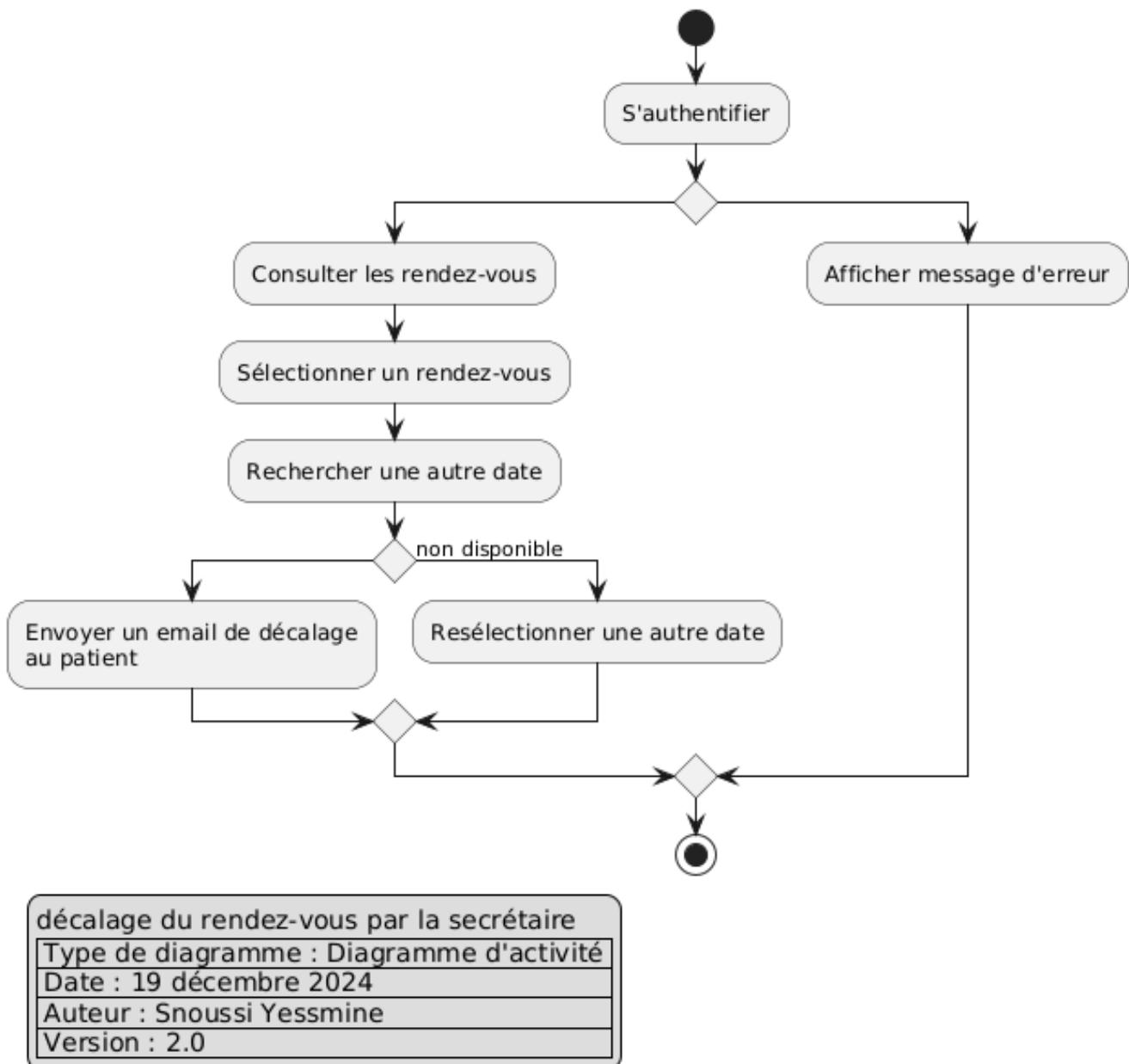


FIGURE 30 – Diagramme d'activité du décalage du rendez-vous par la secrétaire

2.4 Les interfaces

L'interface de connexion de secrétaire comprend un champ pour l'email, un autre pour le mot de passe, et un bouton **show password** pour afficher le mot de passe saisi. Un bouton **Login** permet de valider les informations et de procéder à l'authentification. Si les données sont correctes, l'agent accède à l'interface principale du système de gestion des rendez-vous. En cas d'erreur, un message s'affiche. Cette interface est conçue pour être simple, sécurisée et facile à utiliser, offrant une expérience fluide.

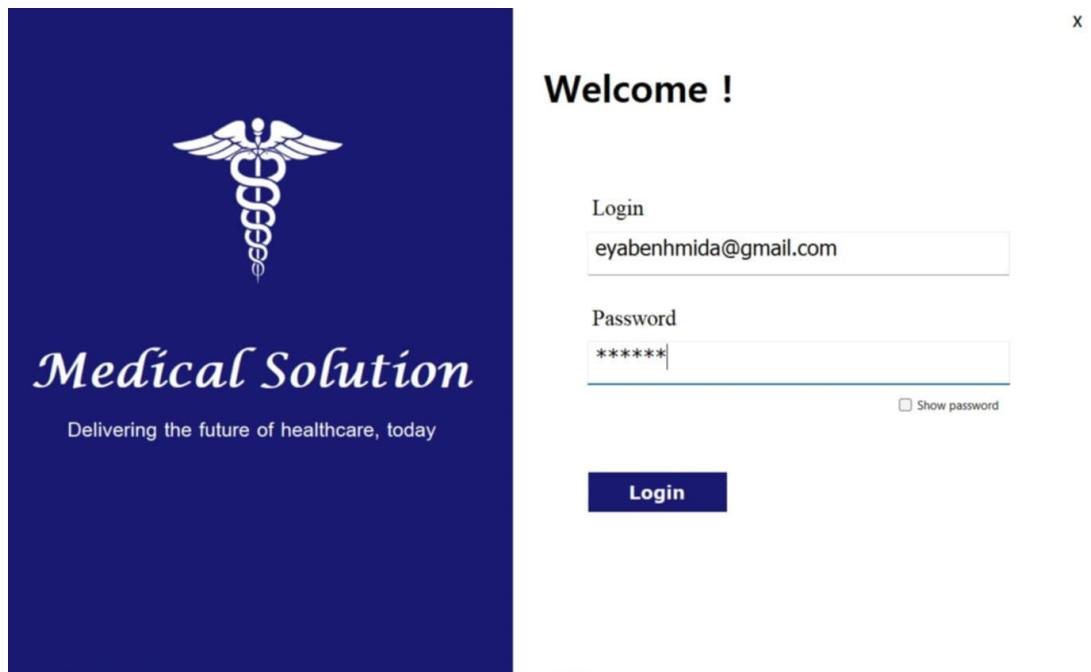


FIGURE 31 – login du secrétaire

LISTE DES TABLEAUX

L'interface de la secrétaire est organisée autour de trois sections principales accessibles via une barre de navigation : **Home**, **Calendar**, et **Appointment Requests**. La section **Home** sert de tableau de bord, offrant un aperçu rapide des activités récentes et des notifications importantes. La section **Calendar** permet à la secrétaire de consulter les disponibilités du médecin, de visualiser les rendez-vous confirmés et de naviguer facilement entre les différentes dates. Enfin, la section **Appointment Requests** est spécifiquement dédiée à la gestion des demandes de rendez-vous en attente de confirmation. Elle affiche une liste des demandes reçues, avec des informations essentielles telles que le nom du patient, la date et l'heure du rendez-vous. Chaque demande inclut un bouton **Confirmer** qui, une fois cliqué, valide le rendez-vous et le transfère automatiquement dans la liste des rendez-vous confirmés, tout en le retirant de la liste d'attente. Cette interface intuitive permet à la secrétaire de gérer les rendez-vous de manière organisée et efficace, tout en maintenant à jour les plannings des médecins.

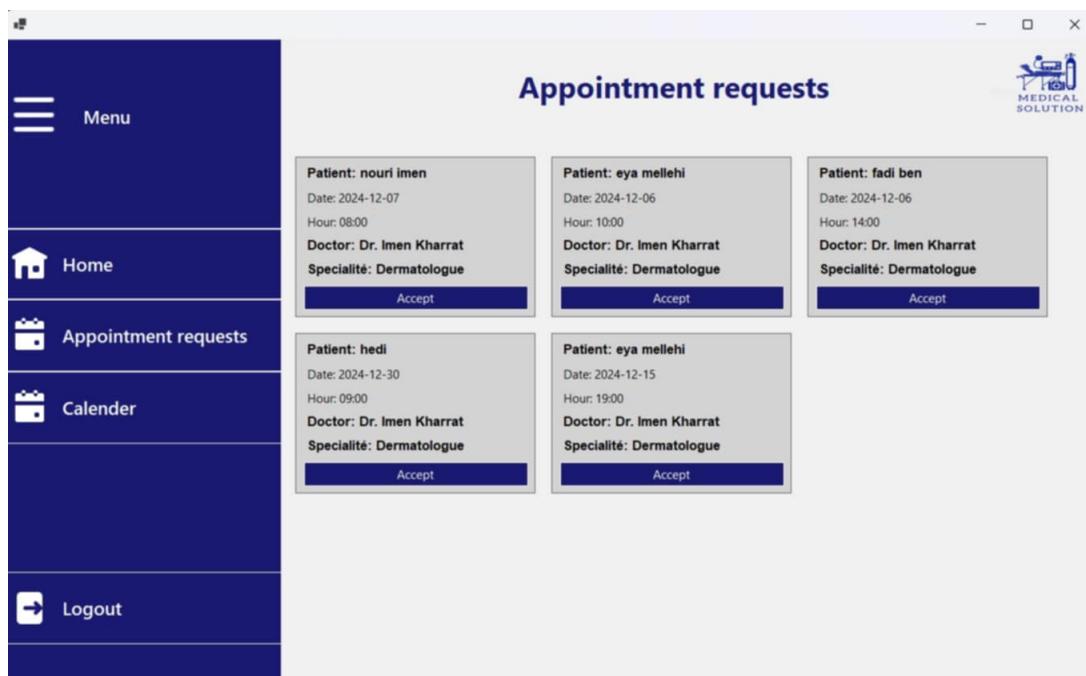


FIGURE 32 – affichage des rendez-vous en attente

LISTE DES TABLEAUX

L'interface de la secrétaire permet de consulter les rendez-vous confirmés du médecin sélectionné en cliquant sur le bouton "*Calendar*". Une fois le médecin choisi, l'agent peut visualiser les rendez-vous confirmés, incluant le nom du patient, la date et l'heure. L'agent peut à tout moment décaler un rendez-vous en modifiant la date et l'heure. Une zone de sélection de date permet de filtrer les rendez-vous selon le jour choisi, et un bouton "*Imprimer*" génère un rapport imprimable des rendez-vous filtrés pour cette date. Cette interface facilite la gestion des rendez-vous, offrant une navigation fluide et des fonctionnalités pratiques pour ajuster le planning et obtenir une version papier du calendrier filtré.

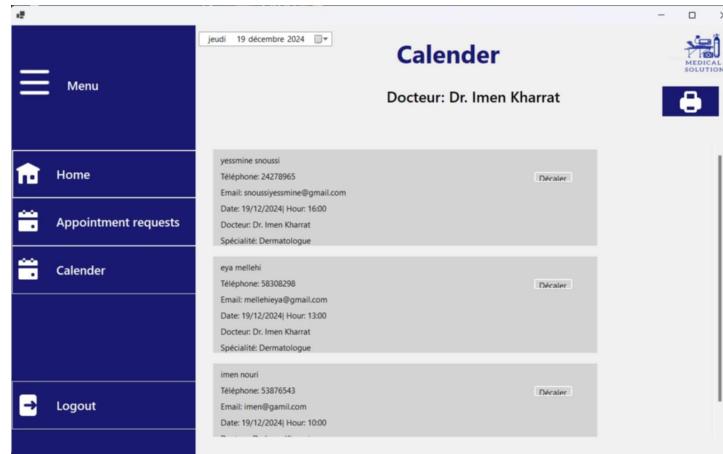


FIGURE 33 – affichage des rendez-vous confirmés

La photo illustre l'interface de l'agent, montrant le processus de décalage d'un rendez-vous. Lorsqu'un rendez-vous doit être modifié, l'agent peut sélectionner une nouvelle date et heure, et le système met automatiquement à jour l'affichage. Un message de confirmation est affiché pour informer l'agent que le rendez-vous a été décalé avec succès. Ce message inclut les nouveaux détails du rendez-vous et confirme que la modification a été prise en compte. Cette fonctionnalité garantit une gestion fluide des rendez-vous et assure une communication claire entre l'agent et le système.

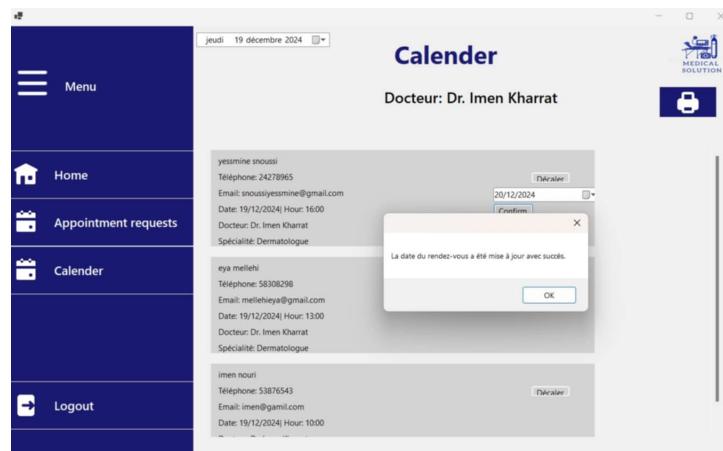


FIGURE 34 – décalage de rendez-vous

LISTE DES TABLEAUX

La capture d'écran illustre un email de confirmation envoyé au patient après le décalage de son rendez-vous. L'email contient l'objet "*votre rendez-vous a été décalé*", un message personnalisé, les détails du rendez-vous (date, heure)

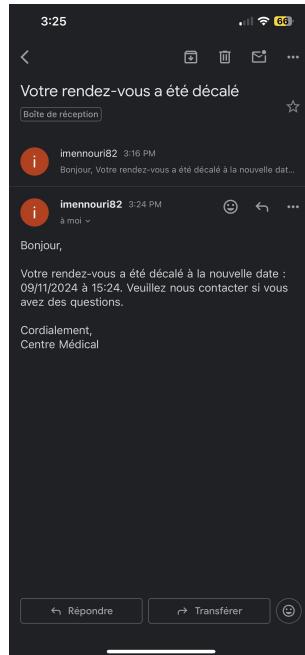


FIGURE 35 – notification de decalage par mail

La capture d'écran illustre un email de confirmation envoyé au patient après validation de son rendez-vous. L'email contient l'objet "*Confirmation de votre rendez-vous médical*", un message personnalisé, les détails du rendez-vous (date, heure)

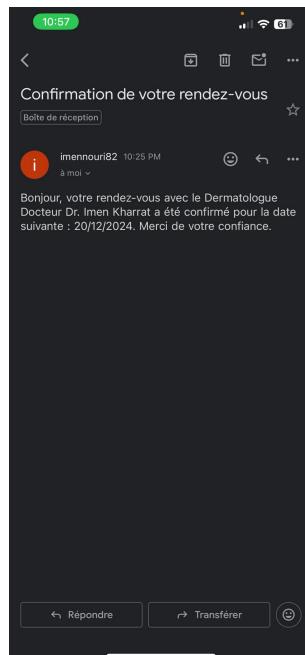


FIGURE 36 – notification de confirmation par mail

2.5 Plan de test

Authentification de la secrétaire médicale

Scénarios de Tests

ID Sécnario	Description	Type de Test
SC101	Authentification réussie avec des identifiants valides	Test unitaire
SC102	échec d'authentification avec des identifiants invalides	Test unitaire
SC103	Redirection après authentification réussie	Test d'intégration

Suite de Tests : Authentification

Propriétés à Tester :

- Validation des identifiants (email et mot de passe).
- Gestion des erreurs lors d'une authentification échoué.
- Redirection vers l'interface de consultation après authentification réussie.

Données de Test :

Nom d'utilisateur	Mot de passe
secretairevalid	password123
secretaireinvalid	wrongpass

Cas de Tests

ID Cas	Étapes	Résultat Attendu
TC101	1. Entrer un nom d'utilisateur valide. 2. Entrer un mot de passe valide. 3. Cliquer sur "Se connecter".	Authentification réussie, redirection vers l'interface de consultation des rendez-vous.
TC102	1. Entrer un nom d'utilisateur valide. 2. Entrer un mot de passe invalide. 3. Cliquer sur "Se connecter".	Message d'erreur affiché : "Identifiants incorrects".
TC103	1. Entrer un nom d'utilisateur invalide. 2. Entrer un mot de passe valide. 3. Cliquer sur "Se connecter".	Message d'erreur affiché : "Identifiants incorrects".

LISTE DES TABLEAUX

Ajustement des rendez-vous

Scénarios de Tests

ID Scénario	Description	Type de Test
SC201	Modification des rendez-vous dans la base de données	Test unitaire
SC202	Mise à jour de l'interface après modification	Test d'intégration
SC203	Vérification de l'intégration avec les autres fonctionnalités	Test d'intégration

Cas de Tests

ID Cas	Étapes	Résultat Attendu
TC201	1. Modifier un rendez-vous via l'interface. 2. Enregistrer les modifications.	La base de données est mise à jour avec les nouvelles informations du rendez-vous.
TC202	1. Modifier un rendez-vous via l'interface. 2. Vérifier l'interface après sauvegarde.	Les modifications sont visibles sur l'interface et cohérentes avec les données enregistrées.

Notification en cas de modification de rendez-vous

Scénarios de Tests

ID Scénario	Description	Type de Test
SC301	Envoi de notification après modification	Test unitaire
SC302	Affichage de la notification dans l'application	Test d'intégration

Cas de Tests

ID Cas	Étapes	Résultat Attendu
TC301	1. Modifier un rendez-vous via l'interface. 2. Vérifier que le patient reçoit une notification.	Le patient reçoit une notification précisant les détails du changement.

LISTE DES TABLEAUX

TC302	1. Ouvrir l'application du patient. 2. Consulter la section des notifications.	La notification apparaît dans l'application du patient avec les informations pertinentes.
-------	-----------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------

Confirmation des rendez-vous avec notification

Scénarios de Tests

ID Scénario	Description	Type de Test
SC401	Confirmation de rendez-vous avec notification	Test unitaire

Cas de Tests

ID Cas	Étapes	Résultat Attendu
TC401	1. Confirmer un rendez-vous via l'interface. 2. Vérifier que le patient reçoit une notification.	Une notification est envoyée immédiatement après confirmation du rendez-vous.

2.6 Bilan du Sprint 2

LISTE DES TABLEAUX

Section	Détails
1. Introduction	<ul style="list-style-type: none"> - Date du sprint : (du 28/10/2024 au 08/11/20). - Objectifs du sprint : créer l'interface secrétaire et optimiser la gestion des rendez-vous, incluant la fonctionnalité de notification.
2. Travail accompli	<ul style="list-style-type: none"> - Tâches terminées : <ol style="list-style-type: none"> 1. Implémentation de l'interface de connexion pour la secrétaire. 2. Développement de l'interface principale avec les sections Home, Calendar et Appointment Requests. 3. Intégration de la fonctionnalité de gestion des demandes de rendez-vous. 4. Développement du système de confirmation des rendez-vous avec notifications par email. - Tâches en cours : <ol style="list-style-type: none"> 1. Test d'intégration pour la fonctionnalité de décalage des rendez-vous. 2. Finalisation des tests pour la notification en cas de modification.
3. Difficultés rencontrées	<ul style="list-style-type: none"> - Problèmes techniques : Difficulté d'envoi des notifications en temps réel pour les patients.
4. Solutions apportées	<ul style="list-style-type: none"> - Mesures prises : <ol style="list-style-type: none"> 1. Optimisation des requêtes vers la base de données pour améliorer la rapidité d'affichage des rendez-vous. 2. Mise en place d'une API centralisée pour l'envoi des notifications. 3. Définition claire des spécifications fonctionnelles lors des réunions.
5. Améliorations à apporter	<ul style="list-style-type: none"> - Suggestions d'amélioration : <ol style="list-style-type: none"> 1. Améliorer la fluidité de navigation dans la section Calendar. 2. Ajouter des options de personnalisation pour les emails de notification. 3. amélioration du design.
6. Prochaines étapes	<ul style="list-style-type: none"> - Tâches à venir : <ol style="list-style-type: none"> 1. Finaliser les tests unitaires et d'intégration pour toutes les nouvelles fonctionnalités. 2. Améliorer l'ergonomie de l'interface pour les utilisateurs. 3. Implémenter une gestion des conflits de rendez-vous pour les patients. - Points à finaliser : <ol style="list-style-type: none"> 1. Correction des bugs liés aux notifications. 2. Validation des rapports de tests.
7. Conclusion	<ul style="list-style-type: none"> - Bilan global du sprint : Ce sprint a permis de poser les bases pour une gestion efficace des rendez-vous, avec une interface intuitive pour les secrétaires. Les notifications ajoutent un niveau supplémentaire de communication avec les patients, mais des ajustements sont encore nécessaires pour garantir leur fiabilité et rapidité.

TABLE 15 – Bilan du Sprint 2

3 Sprint 3 : Authentification et consultation des Rendez-vous par les Médecins

3.1 Objectif du Sprint

Le Sprint 3 avait pour objectif principal de mettre en place les fonctionnalités suivantes :

- - Authentification des médecins pour leur permettre de consulter leurs agenda rendez-vous.
- - Demande de modification des rendez-vous par les patients après authentification.

3.2 Identidication des backlog du sprint 3

User Story	Tâche	Temps estimé	Responsable
En tant que médecin, je veux m'authentifier pour accéder à mon agenda.	Concevoir l'interface de connexion pour les médecins.	1 jour	Yessmine Snoussi
	Implémenter la logique d'authentification (API et base de données).	2 jours	Yessmine Snoussi
	Tester le parcours complet d'authentification.	1 jour	Yessmine Snoussi
En tant que médecin, je veux consulter mon agenda pour suivre mes rendez-vous.	Créer l'interface d'affichage de l'agenda.	1 jour	Yessmine Snoussi
	Implémenter la logique pour récupérer et afficher les rendez-vous.	1.5 jours	Yessmine Snoussi
	Tester la fonctionnalité de consultation de l'agenda.	0.5 jour	Yessmine Snoussi
En tant que patient, je veux demander une modification (dates ou horaires) de mes rendez-vous après authentification.	Concevoir l'interface pour demander une modification de rendez-vous.	1 jour	Eya Mellehi
	Implémenter la logique pour modifier les rendez-vous (API et base de données).	2 jours	Eya Mellehi
	Ajouter une validation pour vérifier les disponibilités.	1 jour	Eya Mellehi
	Tester le processus de modification des rendez-vous.	1 jour	Eya Mellehi

TABLE 16: Sprint Backlog

3.3 Conception

Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation pour le Sprint 3 illustre les interactions spécifiques des médecins avec le système. Les principaux cas incluent l'authentification des médecins, la consultation de leur agenda (liste des rendez-vous), et la modification des rendez-vous des patients. Ces fonctionnalités permettent aux médecins de gérer efficacement leur emploi du temps tout en assurant une meilleure coordination avec les patients et le personnel médical.

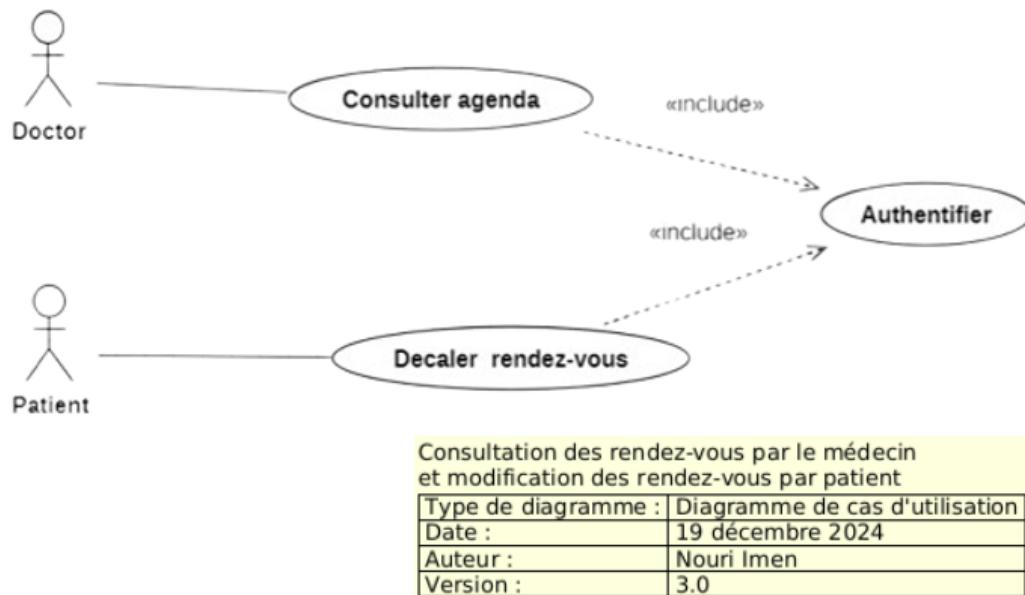


FIGURE 37 – Diagramme de cas d'utilisation du sprint 3

Diagramme des classes

Le diagramme de classes pour le Sprint 3 met en évidence les principales classes du système, à savoir Médecin, RendezVous, et Agenda. La classe Médecin est associée à un Agenda, lequel contient plusieurs RendezVous. Chaque RendezVous inclut des informations essentielles telles que la date, l'heure, ainsi que le patient associé. Cette organisation des classes reflète la logique métier nécessaire pour permettre aux médecins de gérer et consulter leurs rendez-vous directement depuis le système. Grâce à cette structure, chaque médecin peut accéder facilement à son planning et suivre les détails de chaque rendez-vous avec ses patients.

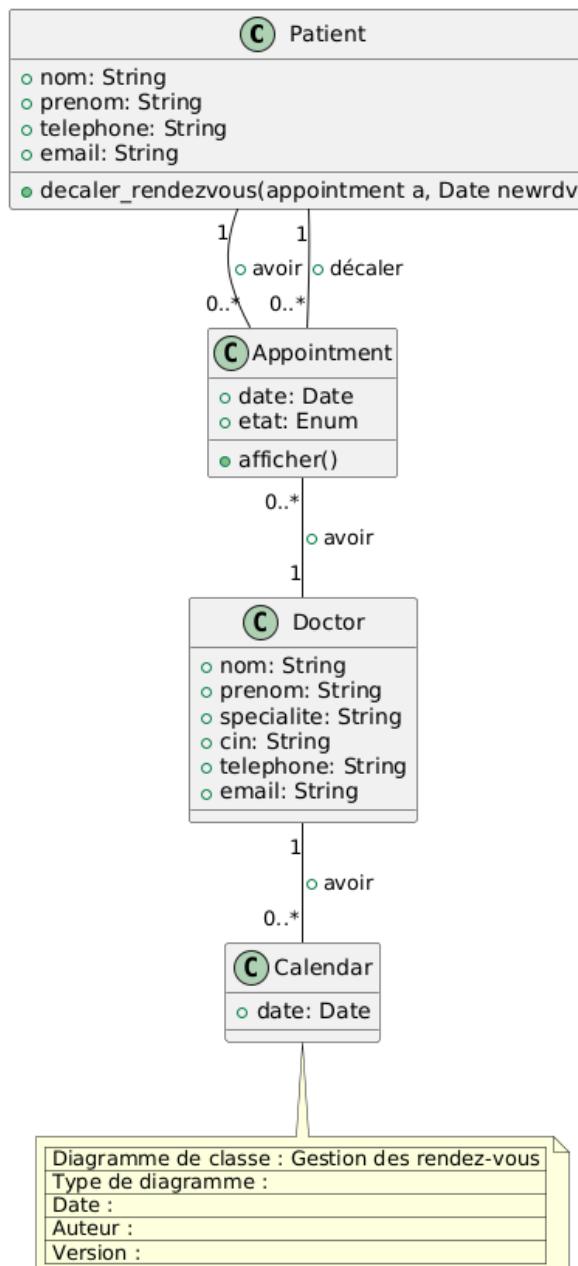
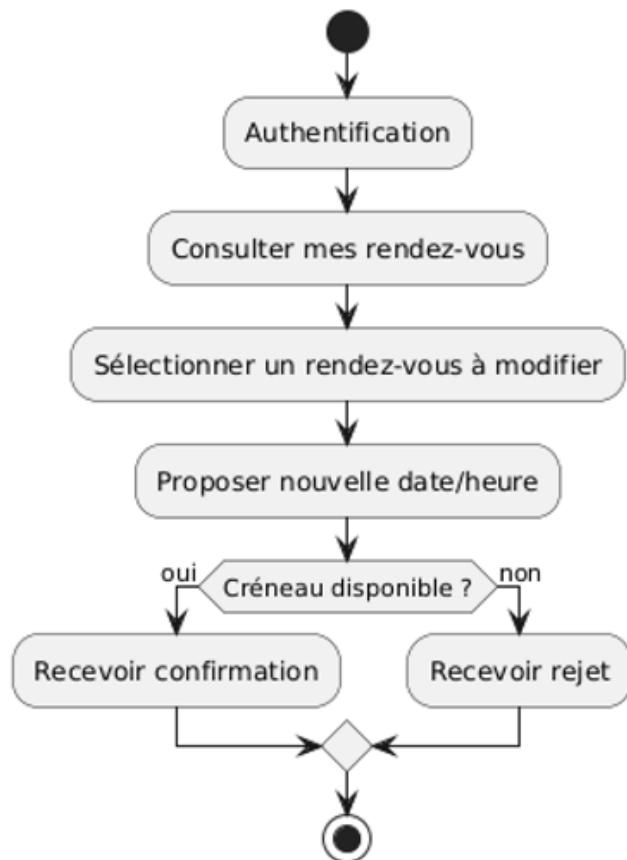


FIGURE 38 – Diagramme des classes du sprint 3

Diagramme d'activité

Le patient s'authentifie dans le système, puis accéde à la liste de ses rendez-vous. Il sélectionne ensuite le rendez-vous qu'il souhaite modifier, puis propose une nouvelle date et heure. Une fois la modification soumise, le système vérifie la disponibilité du créneau proposé. Le patient reçoit ensuite une confirmation si la nouvelle date/heure est valide, ou un rejet si le créneau est déjà occupé ou non disponible. Cette séquence d'activités permet de gérer de manière fluide les modifications de rendez-vous tout en assurant une gestion optimale des disponibilités.



modification de la date de rendez-vous par le patient	
Type de diagramme	Diagramme d'activité
Date	19 décembre 2024
Auteur	Nouri Imen
Version	3.0

FIGURE 39 – Diagramme d'activité de modification de date des rendez-vous par le patient

Interfaces médecin

L'interface d'inscription du médecin permet de créer un compte en remplissant les champs suivants : nom complet, adresse email, spécialité, numéro de téléphone et mot de passe. En cliquant sur le bouton S'inscrire, il valide les informations saisies. Si les données sont correctes, le compte est créé avec succès, sinon un message d'erreur est affiché pour signaler les erreurs. L'interface est simple et sécurisée, offrant ainsi une expérience d'inscription fluide et protégée pour le médecin.

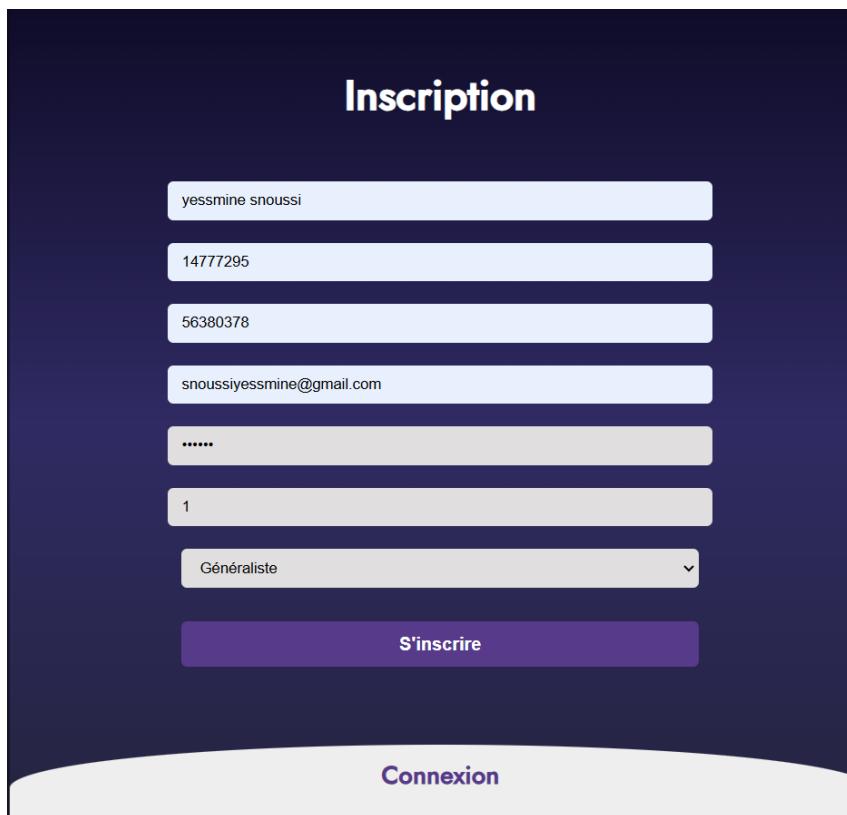


FIGURE 40 – Inscription du médecin

L'interface de connexion permet aux médecins d'accéder au système en remplaçant deux champs : email et mot de passe. Le bouton se connecter valide les informations saisies. Si les identifiants sont corrects, l'accès est accordé, sinon un message d'erreur est affiché pour informer l'utilisateur d'une erreur de connexion. L'interface est intuitive, rapide et sécurisée, offrant ainsi une expérience de connexion aisée et protégée.

LISTE DES TABLEAUX

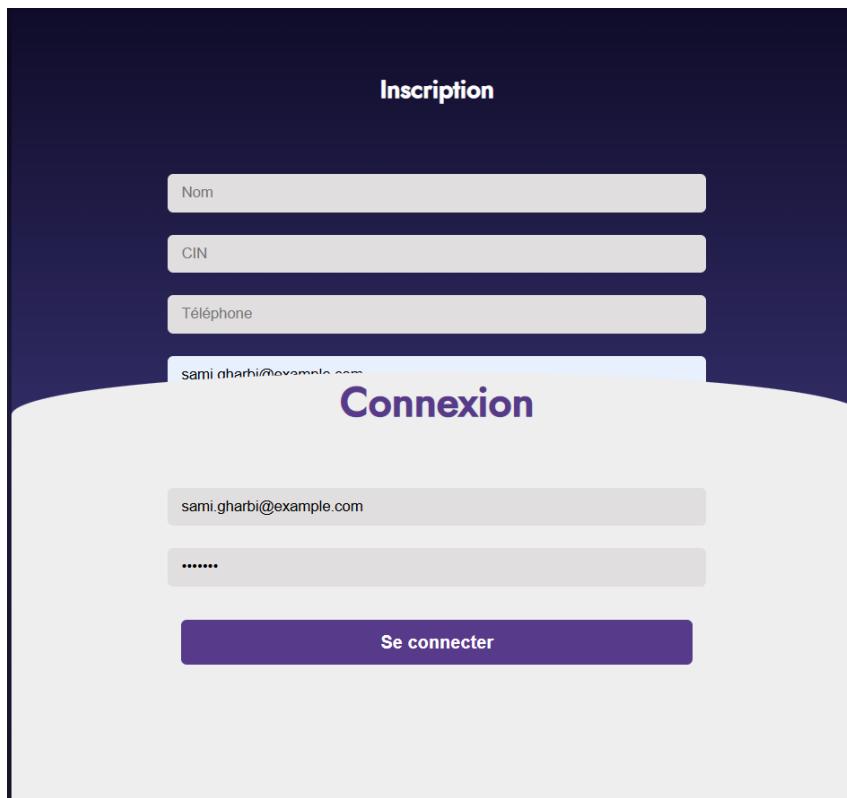


FIGURE 41 – Authentification du médecin

Cette interface permet aux médecins de consulter leur calendrier et de visualiser les rendez-vous programmés. Le médecin peut naviguer entre les différentes dates et afficher les détails des rendez-vous associés pour une gestion efficace de son emploi du temps.

CONSULTATION DES RENDEZ-VOUS							About Me	Logout
HEURE	Emploi du temps du Dr. Imen Kharrat							
	Semaine du 2024-12-02 au 2024-12-07							
09:00								
10:00	imén nourri							
11:00							hedi	
12:00		Disponible						
13:00							eya mellehi	
14:00							Disponible	
15:00								
16:00						yessmine snoussi		

FIGURE 42 – Consultation de calendrier par le médecin

LISTE DES TABLEAUX

Cette interface permet aux médecins la consultation du calendrier pour visualiser les rendez-vous. Le médecin peut naviguer entre les dates et afficher les détails des rendez-vous associés.

Interfaces patient

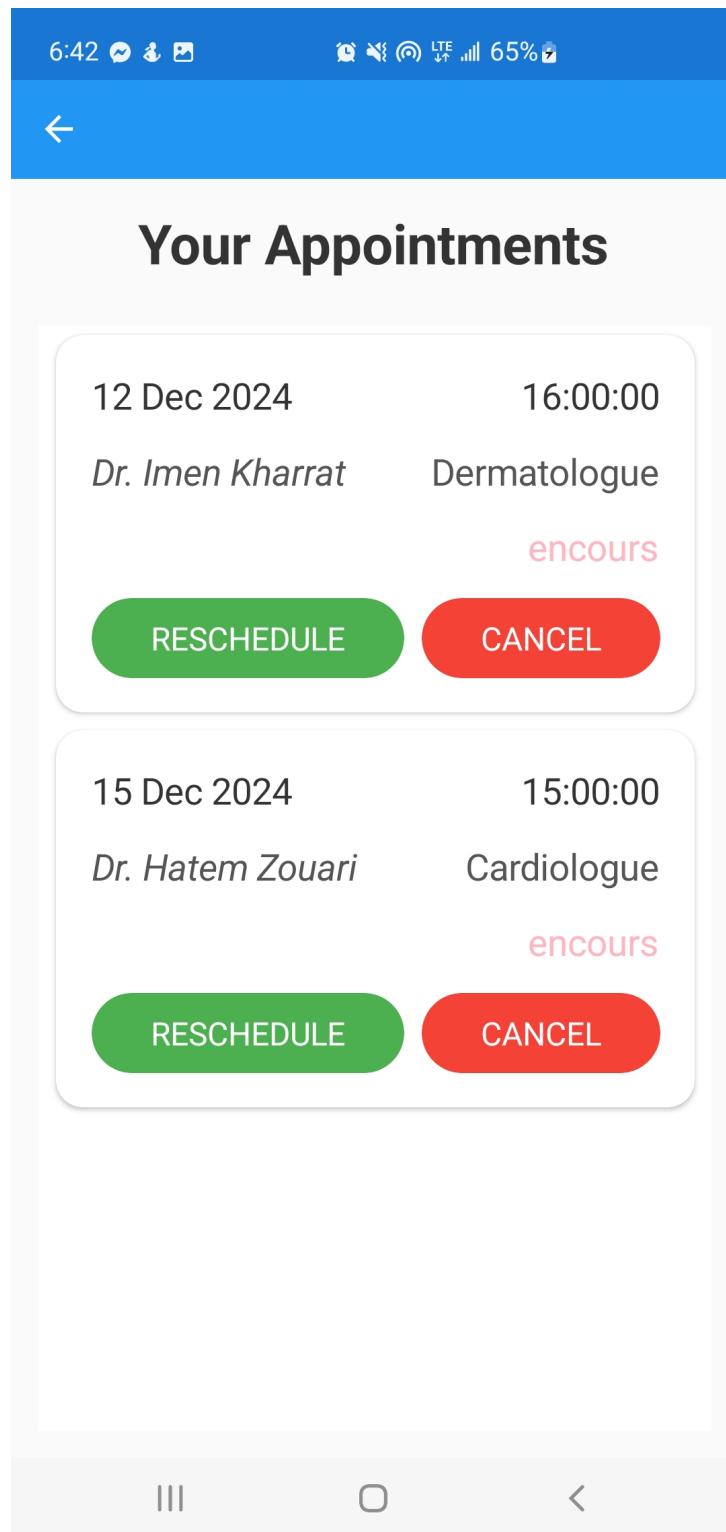


FIGURE 43 – Consultation des rendez-vous par le patient

LISTE DES TABLEAUX

Cette interface permet au patient de gérer ses demandes liées à ses rendez-vous. Elle affiche les informations concernant la demande de décalage de la date et de l'heure du rendez-vous, ou bien son annulation. Le patient peut examiner les détails de sa demande et, si nécessaire, utiliser le bouton **CANCEL** pour annuler la demande en cours. Cette fonctionnalité offre une flexibilité supplémentaire tout en facilitant la gestion autonome des rendez-vous.

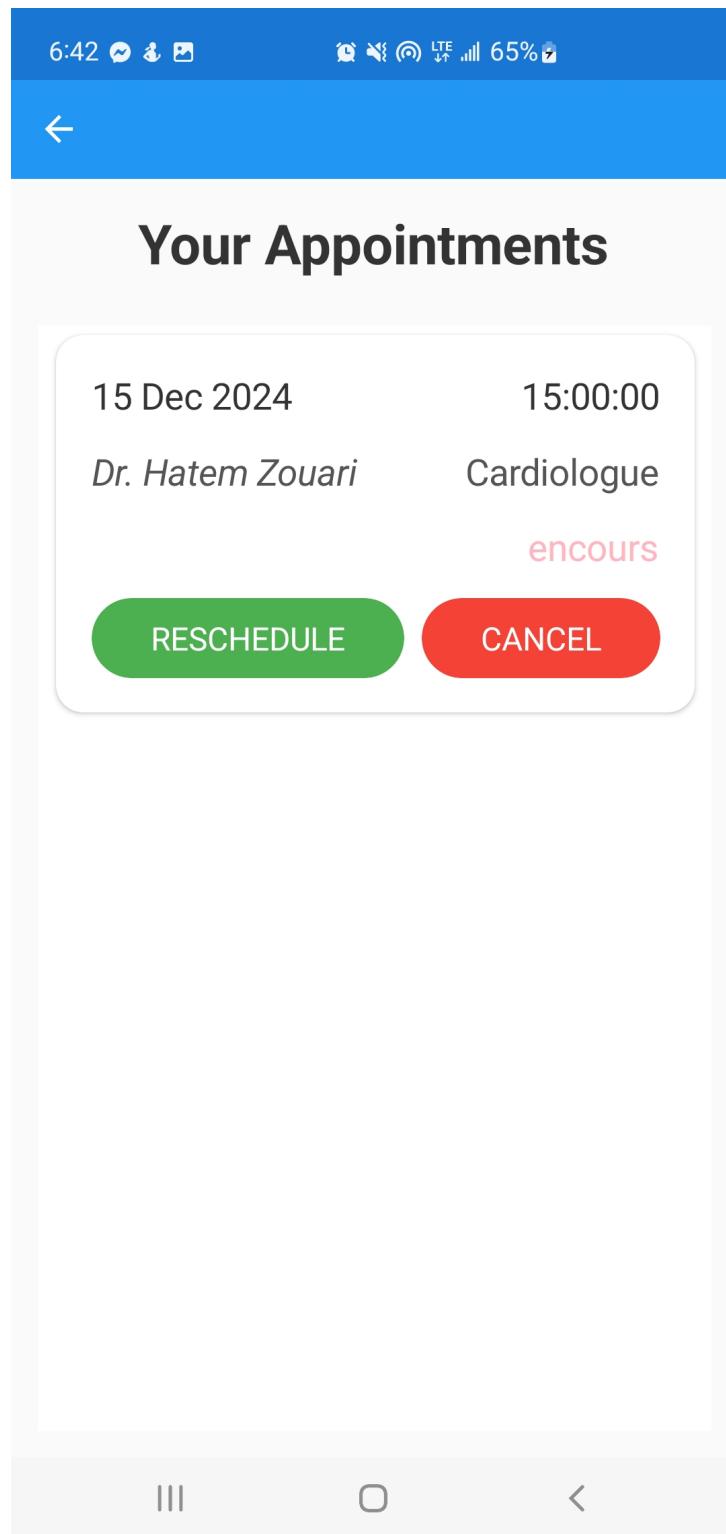


FIGURE 44 – Annulation d'un rendez-vous par le patient

LISTE DES TABLEAUX

Cette interface permet au patient de demander le décalage de son rendez-vous. En cliquant sur le bouton Reschedule, une page s'ouvre affichant les dates disponibles pour le médecin. Le patient peut alors sélectionner sa date préférée en cliquant sur le bouton Reschedule correspondant à cette date. Une fois la sélection effectuée, la demande de décalage est envoyée pour traitement.

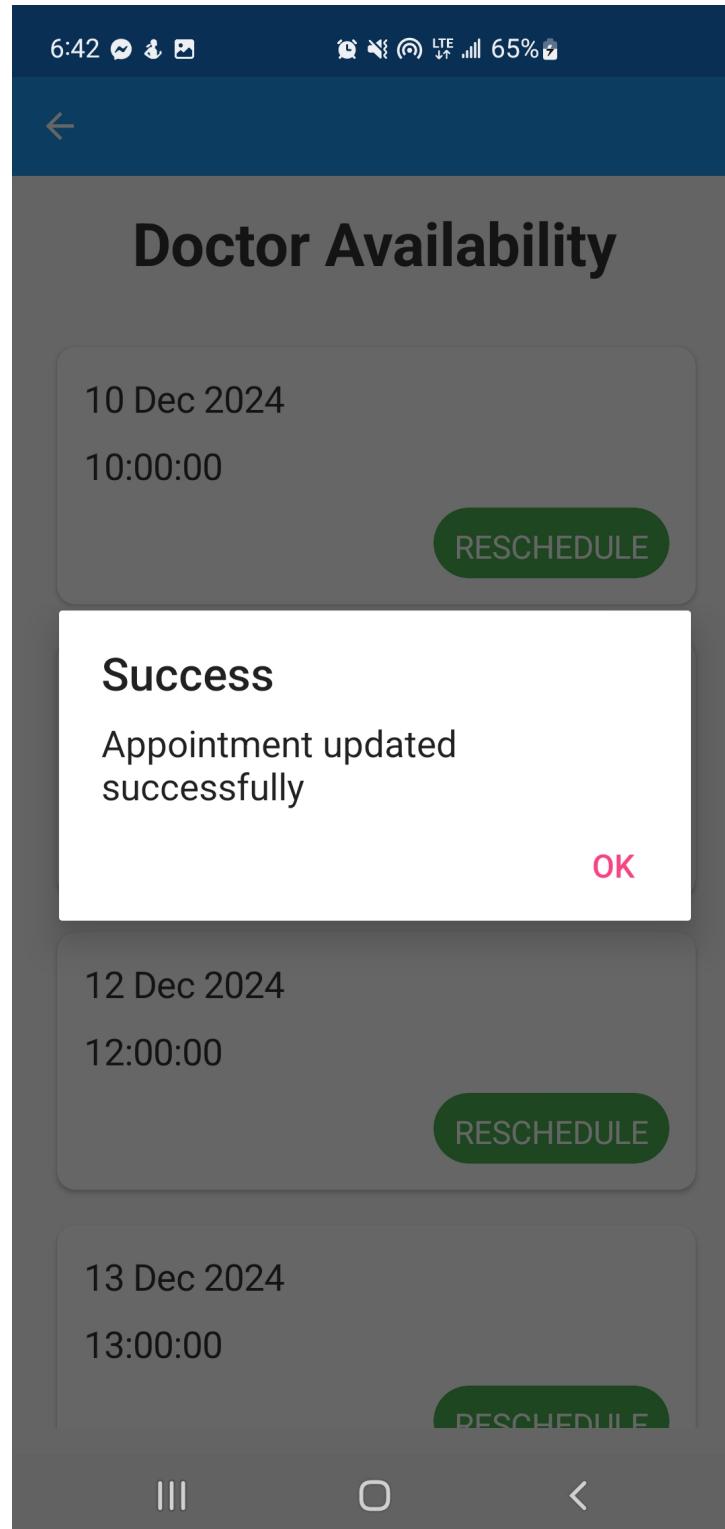


FIGURE 45 – Décalage de rendez-vous par le patient

LISTE DES TABLEAUX

3.4 Test

Suite 1 : Authentification des Médecin

Objet de Test

* Fonctionnalité de connexion des patients.

Propriété à Tester

* Validation des informations d'identification.

Données de Test

Nom d'utilisateur	Mot de passe
médecin _{valide}	correct123
médecin _{nvalide}	erreur456

TABLE 17 – Données de test

Cas de Tests

ID Cas	Étapes	Résultat Attendu	Résultat
TC S1001	<ol style="list-style-type: none"> Entrer un nom d'utilisateur valide. Entrer un mot de passe valide. Cliquer sur "Se connecter". 	Connexion réussie, redirection au tableau de bord.	Succès
TC S1002	<ol style="list-style-type: none"> Entrer un nom d'utilisateur valide. Entrer un mot de passe invalide. Cliquer sur "Se connecter". 	Message d'erreur : "Identifiants incorrects".	Succès

TABLE 18 – Cas de test

Suite 2 : Modification par les Patients

Objet de Test : Modification des rendez-vous par les patients après connexion.

Propriété à Tester : Transmission des modifications à la secrétaire pour validation.

Données de Test :

Date Actuelle	Nouvelle Date
12/12/24	15/12/24

TABLE 19 – Données de test

Cas de Tests :

LISTE DES TABLEAUX

ID Cas	Étapes	Résultat Attendu	Résultat
TC S3001	<ol style="list-style-type: none"> 1. Accéder à "Modifier un rendez-vous". 2. Sélectionner une nouvelle date et heure. 3. Confirmer. 	La demande est envoyée pour validation.	Succès

TABLE 20 – Cas de test

3.5 Bilan du Sprint 3

Section	Détails
1. Introduction	<ul style="list-style-type: none"> - Date : 04/10/2024 au 18/10/2024 - Objectifs : Développer la gestion des rendez-vous et l'authentification des médecins/patients.
2. Travail accompli	<ul style="list-style-type: none"> - Tâches terminées : 1. Authentification des médecins. 2. Consultation et modification des rendez-vous. 3. Validation des disponibilités et tests de fonctionnement.
3. Difficultés rencontrées	<ul style="list-style-type: none"> - Intégration difficile de la base de données avec l'application mobile.
4. Solutions apportées	<ul style="list-style-type: none"> - Correction des erreurs d'intégration avec des scripts SQL. - Clarification des spécifications fonctionnelles.
5. Améliorations à apporter	<ul style="list-style-type: none"> - Améliorer la gestion des erreurs et l'interface mobile. - Ajouter des tests unitaires.
6. Prochaines étapes	<ul style="list-style-type: none"> - Finaliser la fonctionnalité de notification des rendez-vous. - Implémenter la gestion des rôles et terminer les tests.
7. Conclusion	<ul style="list-style-type: none"> - Le sprint a permis de poser les bases pour l'authentification et la gestion des rendez-vous, avec des améliorations nécessaires sur l'interface et les tests.

TABLE 21 – Bilan du Sprint 3

4 Sprint 4 :Ajustement des disponibilités des médecins

4.1 Objectif du Sprint

Le Sprint 4 avait pour objectif principal de mettre en place la fonctionnalités suivante :

- En tant que médecin, je veux ajuster mes disponibilités en fonction de mes impératifs professionnels.

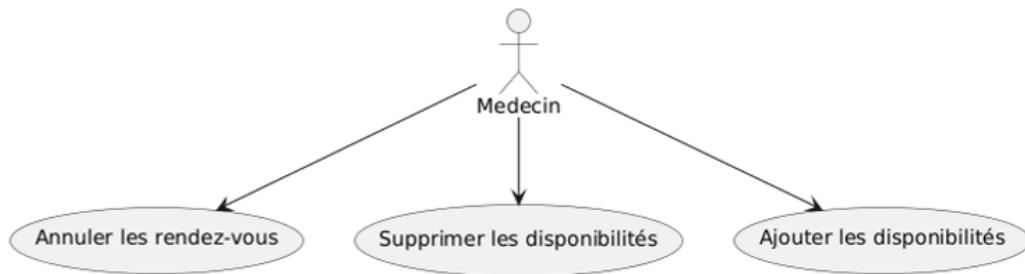
4.2 Identidication des backlog du sprint 4

User story	Temps estimé	Responsable	Priorité
En tant que médecin, je veux ajuster mes disponibilités en fonction de mes impératifs professionnels.	4 jours	Yessmine Snoussi	Haute

4.3 Conception

Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation pour le Sprint 3 illustre les interactions spécifiques des médecins avec le système. Les principaux cas incluent l'authentification des médecins, la consultation de leur agenda (liste des rendez-vous), et la modification des rendez-vous des patients. Ces fonctionnalités permettent aux médecins de gérer efficacement leur emploi du temps tout en assurant une meilleure coordination avec les patients et le personnel médical.



Type de diagramme	Diagramme de cas d'utilisation
Date	19 décembre 2024
Auteur	Nouri Imen
Version	4.0

FIGURE 46 – Diagramme de cas d'utilisation

Diagramme des classes

Le diagramme de classes pour le Sprint 3 met en évidence les classes clés impliquées, comme Medecin, RendezVous, et Agenda. La classe Medecin est associée à un Agenda, qui contient plusieurs RendezVous. Chaque rendez-vous inclut des détails tels que la date, l'heure, et le patient associé. Ces relations reflètent la logique nécessaire pour permettre aux médecins de consulter leurs rendez-vous directement depuis le système.

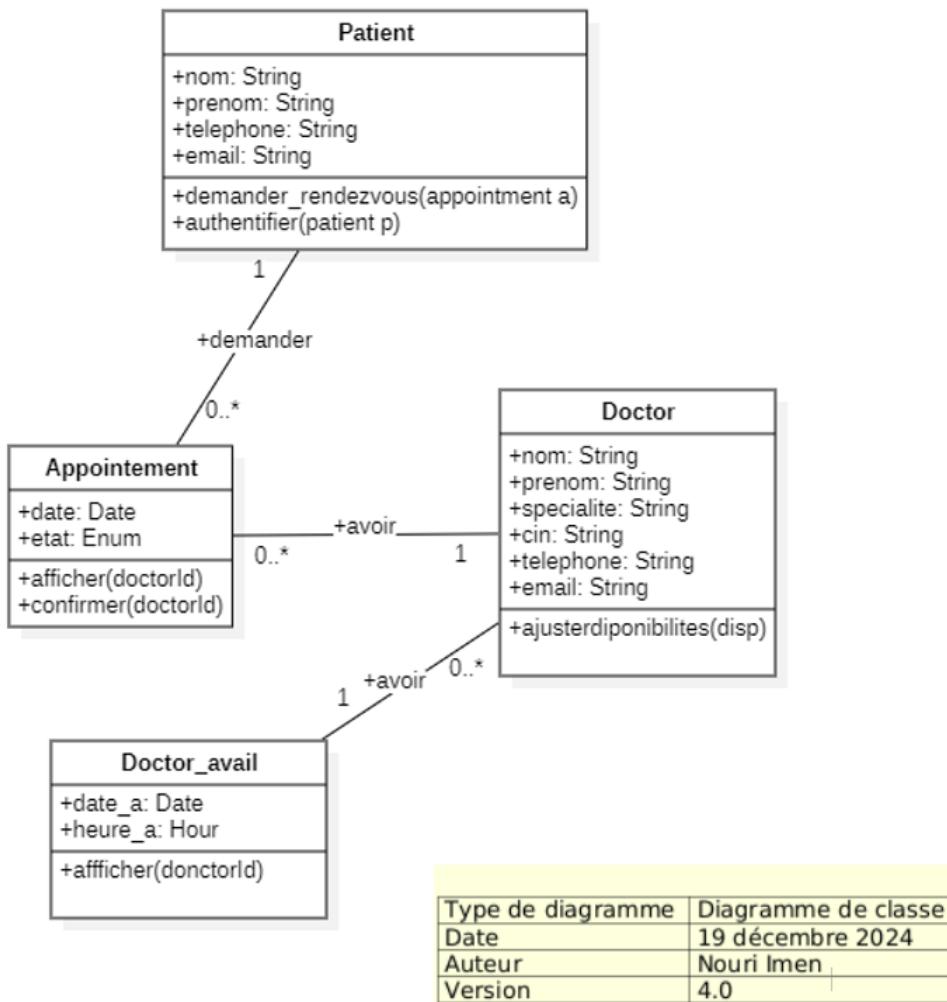
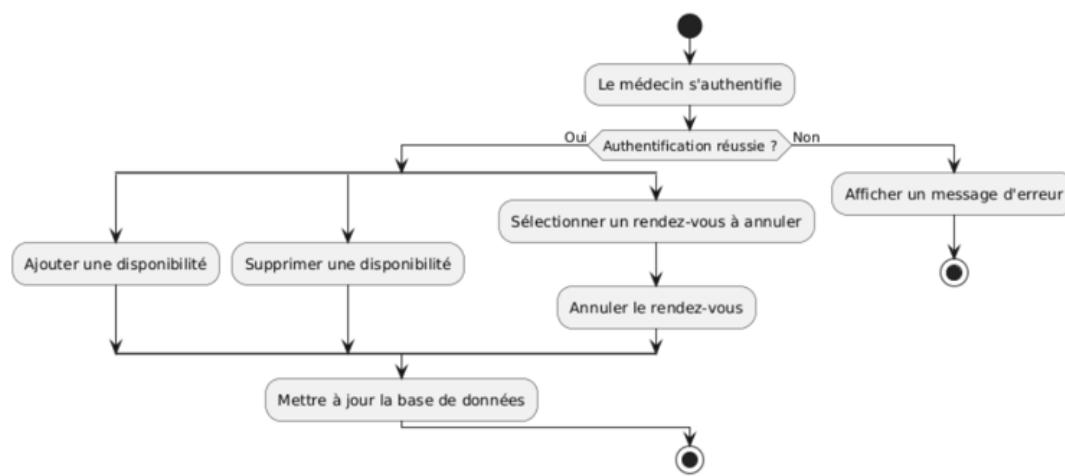


FIGURE 47 – Diagramme des classes

Diagramme d'activité

Le diagramme d'activité décrit les actions qu'un médecin peut entreprendre après s'être authentifié. Tout commence par une tentative d'authentification. Si celle-ci est réussie, le médecin accède à plusieurs options : il peut ajouter une nouvelle disponibilité en définissant une plage horaire, supprimer une disponibilité existante, ou encore sélectionner un rendez-vous pour l'annuler. Dans tous les cas, chaque action est suivie d'une mise à jour de la base de données pour garantir que les modifications sont enregistrées. En revanche, si l'authentification échoue, un message d'erreur est affiché, indiquant que l'accès est refusé. Ce diagramme met ainsi en évidence les différents scénarios possibles après l'authentification, avec un flux clair pour chaque type d'action.



modification de disponibilité par le médecin	
Type de diagramme	Diagramme d'activité
Date	19 décembre 2024
Auteur	Nouri Imen
Version	4.0

FIGURE 48 – Diagramme d'activité de modification de disponibilité par le médecin

LISTE DES TABLEAUX

Interfaces médecin

Cette interface permet au médecin de visualiser les détails d'un rendez-vous en cliquant sur la case de rendez-vous dans le calendrier et propose une option pour annuler le rendez-vous si nécessaire.

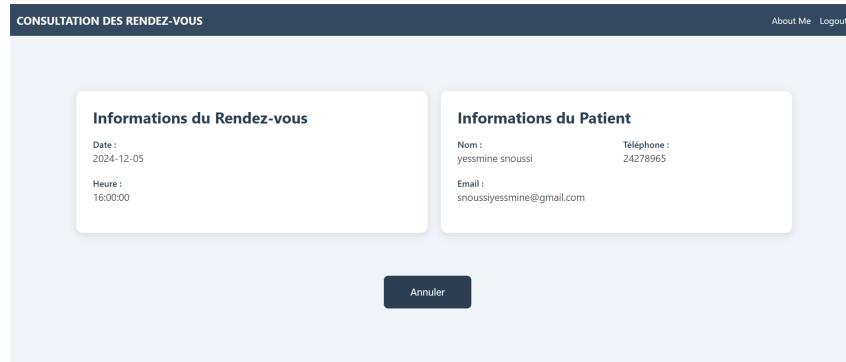


FIGURE 49 – Consultation du détails du rendez-vous par le médecin

Cette interface s'affiche lorsqu'un utilisateur appuie sur une case du calendrier contenant un symbole +, rendant la case disponible pour ajouter un rendez-vous.



FIGURE 50 – Ajout de disponibilité par le médecin

Cette interface permet à le médecin d'annuler une disponibilité en appuyant sur une case marquée comme disponible et en confirmant l'action en cliquant sur le bouton OK.



FIGURE 51 – Annulation de disponibilité par le médecin

LISTE DES TABLEAUX

Cette interface montre la notification reçue par le patient lors du décalage du médecin avec le donner la liberté de choisir un nouveau date pour le rendez-vous

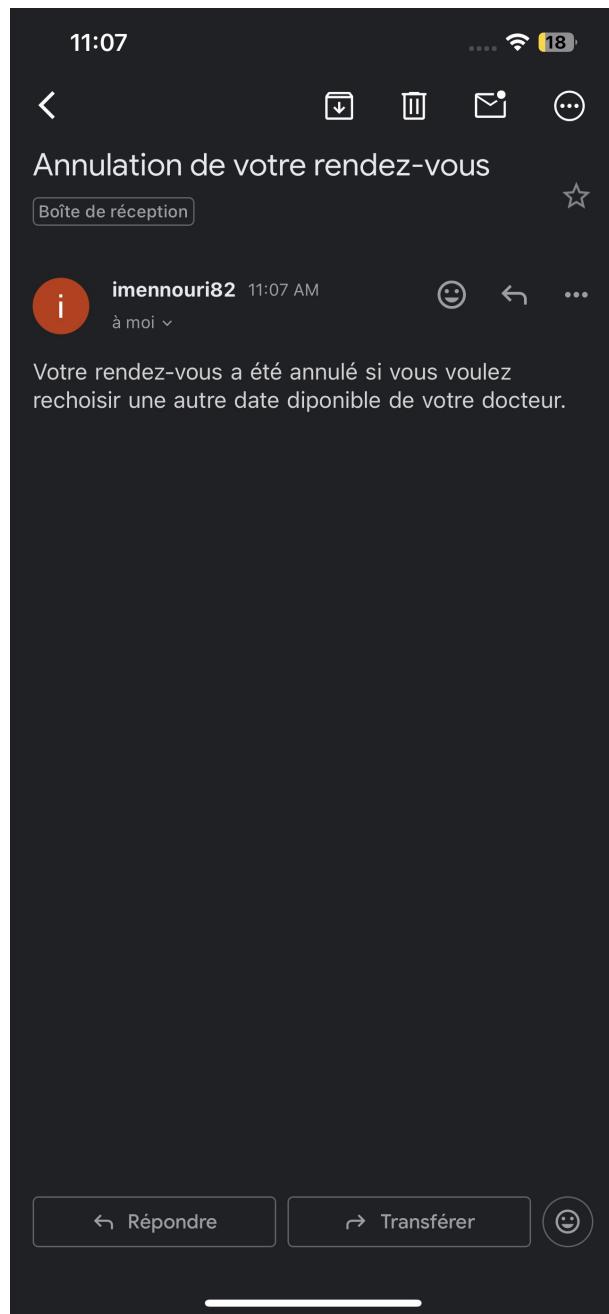


FIGURE 52 – Notification du décalage

4.4 Bilan de sprint

Critére	Valeur planifiée	Valeur réalisée	Observations
User Story : Ajustement des disponibilités des médecins	4 jours	4 jours	Objectif atteint avec succès
Nombre de tâches associées	3	3	Toutes les tâches sont terminées
Priorité	Haute	Haute	Aucun changement de priorité
Responsable	Yessmine Snoussi	Yessmine Snoussi	Travail réalisé comme prévu

TABLE 22 – Bilan du sprint - Ajustement des disponibilités des médecins

Conclusion

Le sprint a permis de finaliser avec succès la fonctionnalité d'ajustement des disponibilités des médecins, priorisée comme haute et réalisée en 4 jours par Yessmine Snoussi. Toutes les tâches ont été accomplies dans les délais, offrant aux médecins une flexibilité dans la gestion de leur emploi du temps. Aucun obstacle majeur n'a été rencontré, reflétant une bonne planification et exécution.

5 Présentation du Code Source

5.1 Organisation du Code

L'application mobile est développée en utilisant **Xamarin.Forms**, ce qui permet de créer une interface utilisateur partagée et de gérer les fonctionnalités pour plusieurs plateformes (*iOS* et *Android*). L'architecture du projet suit une séparation claire en différents modules pour assurer une meilleure organisation et maintenabilité. Voici une description des principaux composants :

Interface Utilisateur (UI)

Le dossier **Views** contient les pages de l'application (**XAML**), où l'interface graphique est définie. Les styles communs (comme les couleurs et polices) sont centralisés dans un fichier **App.xaml**.

Communication avec la Base de Données

Le dossier **Service** contient la classe responsables de la communication avec le serveur **Node.js** via des appels HTTP (**HttpClient**). Ce classe gère les requêtes pour envoyer ou récupérer des données depuis la base de données MySQL.

Modèles de Données (Models)

Les classes définissant les structures des données (par exemple, `Patient`, `Appointment`, `Doctor`) sont regroupées dans le dossier `Models`. Chaque modèle correspond aux entités utilisées dans l'application et facilite la sérialisation/déserialisation des données.

La capture d'écran ci-dessous illustre l'architecture du projet Xamarin.Forms, qui est organisée

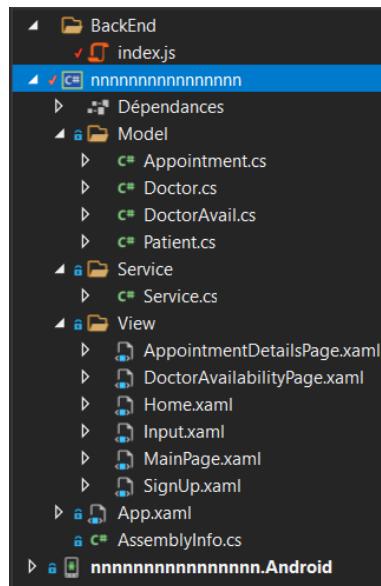


FIGURE 53 – l'architecture du projet

6 Exemples de Code

Fonctionnalité : Incription d'un Patient (SignUp)

La fonctionnalité SignUp permet à un utilisateur de s'inscrire comme patient dans l'application en fournissant les informations nécessaires telles que le nom, le téléphone, l'adresse e-mail et le mot de passe. Les données saisies sont validées côté client avant d'être envoyées au serveur.

Processus de Validation (Code côté client)

Code Source de la Méthode OnSignUpClicked

```

1 private async void OnSignUpClicked(object sender, EventArgs e)
2 {
3     String email = EmailEntry.Text?.Trim();
4     if (string.IsNullOrWhiteSpace(NomEntry.Text))
5     {
6         await DisplayAlert("Erreur", "Le champ 'Nom' est
7         obligatoire.", "OK");
8         return;
9     }
10
11    if (string.IsNullOrWhiteSpace(TelephoneEntry.Text) ||
12        !Regex.IsMatch(TelephoneEntry.Text, @"^\d{8}$"))
13    {
14        await DisplayAlert("Erreur", "Veuillez entrer un numero
15         de telephone valide (8 chiffres).", "OK");
16        return;
17    }
18
19    if (string.IsNullOrWhiteSpace(email) || !IsValidEmail(email))
20    {
21        await DisplayAlert("Erreur", "Veuillez entrer une
22         adresse email valide.", "OK");
23        return;
24    }
25
26    if (string.IsNullOrWhiteSpace(PasswordEntry.Text) ||
27        PasswordEntry.Text.Length < 6)
28    {
29        await DisplayAlert("Erreur", "Le mot de passe doit
30         contenir au moins 6 caracteres.", "OK");
31        return;
32    }
33
34    var patient = new Patient
35    {
36        Nom = NomEntry.Text,

```

LISTE DES TABLEAUX

```
37     Telephone = TelephoneEntry.Text ,
38     Email = email ,
39     pwd = PasswordEntry.Text
40 };
41     Console.WriteLine($"Nom:{patient.Nom}, Telephone:
42 {patient.Telephone}, Email:{patient.Email}, pwd:
43 {patient.pwd}");
44
45     try
46     {
47         await _service.AddPatientAsync(patient);
48         await Navigation.PushAsync(new MainPage());
49     }
50     catch (Exception ex)
51     {
52         await DisplayAlert("Erreur", ex.Message, "OK");
53     }
54 }
55
56 private bool IsValidEmail(string email)
57 {
58     var emailRegex = new Regex(@"^[\w\.-]+@[^\w\.-]+\.\w{2,}$");
59     return emailRegex.IsMatch(email);
60 }
```

Listing 1 – Code de la méthode `OnSignUpClicked`

Explication

- **Validation des Champs :**
 - Chaque champ est vérifié pour s'assurer qu'il n'est pas vide.
 - Le champ téléphone doit contenir exactement 8 chiffres.
 - L'adresse e-mail est validée avec une expression régulière.
 - Le mot de passe doit contenir au moins 6 caractères.
- **Création d'un Objet Patient :** Les données valides sont utilisées pour instancier un objet Patient.
- **Appel au Service :** L'objet Patient est transmis au service via une méthode asynchrone pour l'enregistrement dans la base de données.
- **Gestion des Exceptions :** En cas d'erreur côté serveur, un message d'erreur est affiché.

Communication avec le Serveur (Service)

Code Source de la Méthode `AddPatientAsync`

```
1 public async Task AddPatientAsync(Patient patient)
2 {
3     var json = JsonConvert.SerializeObject(patient);
4     var content = new StringContent(json, Encoding.UTF8,
```

```

5     "application/json");
6     var response = await _httpClient.PostAsync
7                     ("patient", content);
8
9     var responseContent = await response.Content.ReadAsStringAsync();
10    Console.WriteLine($"Server Response: {responseContent}");
11
12    if (!response.IsSuccessStatusCode)
13    {
14        throw new Exception($"Failed to add patient:
15 {responseContent}");
16    }
17 }
```

Listing 2 – Code de la méthode AddPatientAsync

Explication

- **Sérialisation des Données :** L'objet Patient est converti en format JSON à l'aide de la bibliothèque JsonConvert.
- **Envoi d'une Requête POST :** La requête POST envoie les données sérialisées au serveur à l'endpoint "patient".
- **Gestion de la Réponse :**
 - Si la réponse est un succès, aucune action supplémentaire n'est nécessaire.
 - Si la réponse échoue, une exception est levée avec le message d'erreur retourné par le serveur.

Fonctionnalité :l’Affichage des Médecins

Cette section explique le processus d’interaction entre une application Xamarin mobile et un serveur Node.js utilisant PostgreSQL pour récupérer et afficher une liste de médecins.

Client Code (Xamarin)

```

1 private async Task LoadDoctorsAsync()
2 {
3     try
4     {
5         string baseUrl = $"{BaseUrl}/doctors";
6
7         using (var client = new HttpClient())
8         {
9             var response = await client.GetStringAsync(baseUrl);
10            // Get the response from the API
11            Console.WriteLine("API Response: " + response);
12
13            // Directly deserialize the JSON array into a list
14            // of Doctor objects
15            var doctors = JsonConvert.DeserializeObject<
```

LISTE DES TABLEAUX

```
16         <List<Doctor>>(response);
17
18         foreach (var doctor in doctors)
19         {
20             Console.WriteLine($"Doctor\u00b7Name:\u00b7{doctor.Nom},
21 \u00b7Specialty:\u00b7{doctor.Specialite}");
22         }
23
24         // Bind the doctors to the ListView
25         DoctorsListView.ItemsSource = doctors;
26     }
27 }
28 catch (Exception ex)
29 {
30     Console.WriteLine($"Error\u00b7loading\u00b7doctors:\u00b7{ex.Message}"
31 );
32 }
```

subsectionExplication

- **URL de l'API** : La variable baseUrl contient l'URL de l'API pour récupérer les médecins (/doctors).
- **Requête GET** : Le HttpClient envoie une requête GET pour obtenir la liste des médecins au format JSON.
- **Déserialisation** : La réponse JSON est déserialisée en une liste d'objets Doctor.
- **Affichage des Médecins** : Chaque médecin est loggé dans la console, et la liste des médecins est ensuite liée à un ListView pour l'affichage.
- **Gestion des Erreurs** : En cas d'erreur, un message d'erreur est affiché à l'utilisateur.

Code Côté Serveur (Node.js + PostgreSQL)

Le code Côté serveur est écrit en Node.js. Il utilise la bibliothèque pg pour interagir avec la base de données PostgreSQL afin de récupérer les médecins.

```
1 app.get('/api/doctors', (req, res) => {
2     const query = "SELECT*\u00b7FROM\u00b7doctor\u00b7ORDER\u00b7BY\u00b7specialite";
3     client.query(query, (err, results) => {
4         if (err) {
5             return res.status(500)
6             .send("Erreur\u00b7lors\u00b7de\u00b7la\u00b7recuperation\u00b7des\u00b7medecins");
7         }
8         res.json(results.rows); // Renvoie uniquement les
9         donnees});});
```

Explanation

- **Requête SQL** : Le serveur envoie une requête SQL pour récupérer tous les médecins de la base de données, triés par spécialité.
- **Exécution de la Requête** : La requête est exécutée à l'aide de la méthode client.query de PostgreSQL. Les résultats sont récupérés dans results.rows.

- **Gestion des Erreurs** : Si une erreur se produit lors de l'exécution de la requête, une erreur 500 est renvoyée au client.
- **Réponse JSON** : Si la requête réussit, le serveur renvoie une réponse JSON contenant la liste des médecins.

Interaction Client-Serveur

La communication entre le client et le serveur se fait comme suit :

- **Requête GET** : Le client Xamarin envoie une requête GET à l'endpoint /api/doctors pour récupérer la liste des médecins.
- **Requête SQL sur le Serveur** : Après avoir reçu la requête, le serveur interroge la base de données PostgreSQL pour récupérer les données des médecins.
- **Réponse au Client** : Une fois que la base de données renvoie les résultats, le serveur envoie une réponse JSON contenant la liste des médecins au client.
- **Traitements par le Client** : Le client déserialise cette réponse JSON en objets Doctor et les lie au ListView pour l'affichage.

Page d'Accueil avec Liste des Médecins et Gestion des Rendez-vous

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     x:Class="nnnnnnnnnnnnnnn.Home">
5
6     <StackLayout VerticalOptions="FillAndExpand"
7         HorizontalOptions="FillAndExpand" Padding="15">
8         <Button Text="Check My Appointment"
9             FontSize="18"
10            FontAttributes="Bold"
11            TextColor="White"
12            BackgroundColor="#1C375C"
13            Clicked="OnCheckAppointmentClicked"
14            HorizontalOptions="Center"
15            Padding="10" />
16         <!-- Welcome Label -->
17         <Label x:Name="Appusername"
18             FontSize="22"
19             FontAttributes="Bold"
20             TextColor="#333333"
21             HorizontalOptions="Center"
22             Text="Welcome ,Patient !" />
23
24         <!-- Doctors ListView -->
25         <ListView x:Name="DoctorsListView"
26             BackgroundColor="White"
27             SeparatorVisibility="Default"
28             HasUnevenRows="True">
29             <ListView.ItemTemplate>
30                 <DataTemplate>
```

LISTE DES TABLEAUX

```
31             <ViewCell>
32                 <Grid Padding="15"
33                     BackgroundColor="White" RowSpacing="5"
34                     ColumnSpacing="10">
35                         <Grid.ColumnDefinitions>
36                             <ColumnDefinition Width="3*" />
37                             <ColumnDefinition Width="4*" />
38                             <ColumnDefinition Width="4*" />
39                     </Grid.ColumnDefinitions>
40
41             <Label Text="{Binding\u00b7Nom}"
42                     FontSize="12"
43                     FontAttributes="Bold"
44                     TextColor="#333333"
45                     VerticalOptions="Center"
46                     Grid.Column="0" />
47
48             <Label Text="{Binding\u00b7Specialite}"
49                     FontSize="12"
50                     TextColor="#777777"
51                     VerticalOptions="Center"
52                     Grid.Column="1" />
53
54             <Button Text="Make\u00b7Appointment"
55                     FontSize="8"
56                     VerticalOptions="Center"
57                     TextColor="White"
58                     BackgroundColor="#1C375C"
59                     FontAttributes="Bold"
60                     WidthRequest="100"
61                     Clicked="OnMakeAppointmentClicked"
62                     Grid.Column="2" />
63         </Grid>
64     </ViewCell>
65 </DataTemplate>
66 </ListView.ItemTemplate>
67 </ListView>
68 </StackLayout>
69
70 </ContentPage>
```

Gestion des Rendez-vous Médicaux

Vérification des Rendez-vous Existants

Lorsqu'un patient souhaite consulter ses rendez-vous existants, il clique sur le bouton "Check My Appointment" dans l'application. Le système envoie une requête HTTP pour récupérer les rendez-vous du patient à partir du serveur. Si des rendez-vous sont trouvés, ils sont affichés à l'utilisateur dans une nouvelle page. Sinon, un message indiquant qu'aucun

LISTE DES TABLEAUX

rendez-vous n'a été trouvé est montré.

Code pour Vérifier les Rendez-vous

```
1 private async void OnCheckAppointmentClicked(object sender,
2 EventArgs e)
3 {
4     try
5     {
6         string baseUrl = $"{BaseUrl}/appointments/patient/
7 {_PatientId}";
8         using (var client = new HttpClient())
9         {
10             var response = await client.GetStringAsync(baseUrl);
11             var appointments = JsonConvert.DeserializeObject<
12                 <List<Appointment>>(response);
13
14             if (appointments != null && appointments.Count > 0)
15             {
16                 await Navigation.PushAsync
17                     (new AppointmentDetailsPage(appointments,
18                         _PatientId));
19             }
20             else
21             {
22                 await DisplayAlert("No Appointment",
23                     "You don't have any appointments.", "OK");
24             }
25         }
26     }
27     catch (Exception ex)
28     {
29         await DisplayAlert("Error",
30             "There was an issue fetching the appointment.", "OK");
31         Console.WriteLine($"Error: {ex.Message}");
32     }
33 }
```

Explication :

- **Requête HTTP** : Une requête HTTP GET est envoyée pour récupérer les rendez-vous du patient en utilisant son ID.
- **Déserialisation JSON** : La réponse de l'API est convertie en une liste d'objets Appointment.
- **Affichage des rendez-vous** : Si des rendez-vous sont trouvés, l'application affiche une page avec les détails de ces rendez-vous. Sinon, un message d'alerte est affiché à l'utilisateur.

Prise de Rendez-vous

Lorsqu'un patient souhaite prendre un rendez-vous avec un médecin, il clique sur le bouton "Make Appointment" dans la liste des médecins. Cette action redirige l'utilisateur vers une nouvelle page où il pourra entrer les détails nécessaires pour finaliser la prise de rendez-vous.

Code pour Prendre un Rendez-vous

```
1 private async void OnMakeAppointmentClicked(object sender,
2                                         EventArgs e)
3 {
4     var button = (Button)sender;
5     var doctor = (Doctor)button.BindingContext;
6
7     if (doctor == null)
8     {
9         await DisplayAlert("Error", "Doctor data not found.", "OK");
10    return;
11 }
12
13    await Navigation.PushAsync(new Input(doctor.Specialite,
14 doctor.Nom, doctor.Codem, _PatientId));
15 }
```

Explication :

- **Récupération des données du médecin** : Lorsqu'un bouton est cliqué, les données du médecin sont extraites du BindingContext du bouton.
- **Navigation vers la page de prise de rendez-vous** : Si les données du médecin sont valides, l'utilisateur est redirigé vers une page où il pourra entrer les informations nécessaires pour prendre un rendez-vous.
- **Gestion des erreurs** : Si les données du médecin ne sont pas disponibles, un message d'erreur est affiché.

Affichage des Détails des Rendez-vous

Lorsque le patient consulte ses rendez-vous, une page est affichée avec une liste de tous ses rendez-vous. Chaque rendez-vous peut être sélectionné pour consulter plus de détails ou effectuer des actions telles que l'annulation.

Code pour Afficher les Rendez-vous

```
1 [XamlCompilation(XamlCompilationOptions.Compile)]
2 public partial class AppointmentDetailsPage : ContentPage
3 {
4     private int _patientId;
5     private const string BaseUrl = "http://192.168.1.6:4003/api";
6
7     public AppointmentDetailsPage(List<Appointment> appointments,
```

LISTE DES TABLEAUX

```
8                                         int PatientId)
9 {
10    InitializeComponent();
11    _patientId = PatientId;
12    Console.WriteLine("Patient ID: " + _patientId);
13    AppointmentsListView.ItemsSource = appointments;
14
15    MessagingCenter.Subscribe<DoctorAvailabilityPage>(this,
16    "AppointmentUpdated", (sender) =>
17    {
18        LoadAppointments();
19    });
20}
21
22 private async void LoadAppointments()
23 {
24     try
25     {
26         string apiUrl =
27         $"{BaseUrl}/appointments/patient/{_patientId}";
28         using (HttpClient client = new HttpClient())
29         {
30             var response = await client.GetAsync(apiUrl);
31
32             if (response.IsSuccessStatusCode)
33             {
34                 string responseBody = await response.Content
35                     .ReadAsStringAsync();
36
37                 if (string.IsNullOrEmpty(responseBody))
38                 {
39                     AppointmentsListView.ItemsSource =
40                     new List<Appointment>();
41                     Console.WriteLine
42                     ("No active appointments found.");
43                     await DisplayAlert("No Appointments",
44                     "You don't have any appointments.", "OK");
45                 }
46                 else
47                 {
48                     var appointments = JsonConvert
49                     .DeserializeObject<List<Appointment>>
50                     (responseBody);
51                     if (appointments != null &&
52                     appointments.Count > 0)
53                     {
54                         AppointmentsListView.ItemsSource
55                         = appointments;
```

```

56             }
57         else
58         {
59             AppointmentsListView.ItemsSource =
60             new List<Appointment>();
61             Console.WriteLine
62             ("No\u00e2active\u00e2appointments\u00e2found.");
63             await DisplayAlert
64             ("No\u00e2Appointments",
65             "You\u00e2don't\u00e2have\u00e2any\u00e2appointments."
66             , "OK");
67         }
68     }
69 }
70 else if (response.StatusCode == System.Net
71         .HttpStatusCode.NoContent)
72 {
73     Console.WriteLine
74     ("No\u00e2appointments\u00e2found\u00e2or\u00e2all\u00e2cancelled.");
75     await DisplayAlert("No\u00e2Appointments",
76     "You\u00e2don\u00e2t\u00e2have\u00e2any\u00e2appointments.", "OK");
77 }
78 else
79 {
80     Console.WriteLine($"Error\u00e2fetching\u00e2appointments:
81 {response.ReasonPhrase}");
82     await DisplayAlert("Error",
83     "An\u00e2error\u00e2occurred\u00e2while\u00e2fetching\u00e2appointments",
84     "OK");
85 }
86 }
87 }
88 catch (Exception ex)
89 {
90     Console.WriteLine
91     ($"Error\u00e2fetching\u00e2appointments:\u00e2{ex.Message}");
92     await DisplayAlert("Error",
93     "An\u00e2error\u00e2occurred\u00e2while\u00e2fetching\u00e2appointments",
94     "OK");
95 }
}

```

Explication

- **Requête HTTP pour récupérer les rendez-vous** : Une requête GET est envoyée pour obtenir la liste des rendez-vous du patient en utilisant son ID.
- **Gestion des erreurs** : Si le serveur retourne un code de statut indiquant qu'il n'y a pas de contenu (204 No Content) ou en cas d'erreur réseau, une alerte est affichée à l'utilisateur.
- **Affichage des rendez-vous** : Si des rendez-vous sont récupérés avec succès, la liste des rendez-vous est affichée dans une ListView.

Annulation de Rendez-vous

Lorsqu'un patient souhaite annuler un rendez-vous, il peut cliquer sur un bouton d'annulation à côté du rendez-vous concerné. Cette action envoie une requête PUT pour annuler le rendez-vous sur le serveur.

Code pour Annuler un Rendez-vous

```
1 private async void OnCancelAppointmentClicked(object sender,
2                                         EventArgs e)
3 {
4     try
5     {
6         var button = (Button)sender;
7         var appointment = (Appointment)button.BindingContext;
8
9         Console.WriteLine
10        ($" Cancelling appointment with ID: {appointment.Id}");
11
12        string apiUrl =
13        $"{BaseUrl}/appointments/cancel/{appointment.Id}";
14
15        using (HttpClient client = new HttpClient())
16        {
17            var response = await client.PutAsync(apiUrl, null);
18
19            if (response.IsSuccessStatusCode)
20            {
21                Console.WriteLine
22                (" Appointment canceled successfully");
23                await DisplayAlert
24                (" Success", " Appointment canceled successfully",
25                , "OK");
26                await Navigation.PopAsync();
27            }
28            else
29            {
30                Console.WriteLine($" Failed to cancel appointment:
31                {response.StatusCode}");
32                await DisplayAlert("Error",
33                " Failed to cancel appointment",
34                "OK");
35            }
36        }
37    }
38    catch (Exception ex)
39    {
40        Console.WriteLine
41        ($" Error cancelling appointment: {ex.Message}");
42        await DisplayAlert("Error",
43        " An error occurred while canceling the appointment",
```

LISTE DES TABLEAUX

```
44     "OK");
45 }
46 }
```

Explication

- **Annulation via requête PUT** : Un appel API est effectué pour annuler un rendez-vous spécifique. Le bouton cliqué contient l'ID du rendez-vous à annuler.
- **Gestion des erreurs** : Si l'annulation échoue, un message d'erreur est affiché, sinon un message de succès est montré.

Gestion des Rendez-vous et des Créneaux Horaires

Cette section décrit la gestion des créneaux horaires disponibles pour un médecin et la réservation d'un rendez-vous par un patient. Nous avons plusieurs méthodes permettant de charger, sélectionner et soumettre les créneaux horaires.

Attributs et Initialisation du class Input

Dans cette classe, plusieurs attributs privés sont utilisés pour stocker les informations du médecin, du patient, ainsi que la liste des créneaux horaires disponibles. Le constructeur initialise ces variables à partir des paramètres passés lors de la création de la page.

```
1 public partial class Input : ContentPage
2 {
3     private string _doctorName;
4     private int _doctorId;
5     private const string BaseUrl="http://192.168.1.6:4003/api";
6     private int _patientId;
7     private List<DoctorAvail> _availableSlots;
8
9     public Input(string specialty, string doctorName,
10                 int doctorId, int patientId)
11    {
12        InitializeComponent();
13        _doctorName = doctorName;
14        _doctorId = doctorId;
15        _patientId = patientId;
16
17       InputLabel.Text =$"You are viewing available times with
18 Dr.{doctorName} for {specialty}.";
19        _availableSlots = new List<DoctorAvail>();
20    }
21 }
```

Chargement des Créneaux Horaires Disponibles

La méthode `LoadAvailableSlots` récupère les créneaux horaires disponibles pour un médecin en envoyant une requête HTTP à l'API backend. Les créneaux sont ensuite affichés dans un `ListView`.

LISTE DES TABLEAUX

```
1 private async Task LoadAvailableSlots()
2 {
3     try
4     {
5         string apiUrl=
6             $"{BaseUrl}/doctors/availability/{_doctorId}";
7         using (HttpClient client = new HttpClient())
8         {
9             var response = await client.GetStringAsync(apiUrl);
10            _availableSlots = JsonConvert.DeserializeObject
11                <List<DoctorAvail>>(response);
12            AvailableTimesListView.ItemsSource=_availableSlots;
13        }
14    }
15    catch (Exception ex)
16    {
17        Console.WriteLine($"Error: {ex.Message}");
18        await DisplayAlert("Error",
19            "An error occurred while fetching available slots.
20            Please try again.", "OK");
21    }
22 }
```

Sélection et Soumission du Créneau Horaire

Lorsque l'utilisateur sélectionne un créneau dans la liste, la méthode `OnSubmitButtonClicked` est appelée pour valider la réservation. Si un créneau est sélectionné, un objet `Appointment` est créé et envoyé à l'API pour être sauvegardé. Le créneau est ensuite supprimé de la liste pour marquer qu'il a été réservé.

```
1 private async void OnSubmitButtonClicked(object sender,
2                                         EventArgs e)
3 {
4     if (AvailableTimesListView.SelectedItem == null)
5     {
6         await DisplayAlert("Error",
7             "Please select an available time slot.", "OK");
8         return;
9     }
10
11    var selectedSlot = (DoctorAvail)AvailableTimesListView
12        .SelectedItem;
13    Appointment appointment = new Appointment
14    {
15        Codem = _doctorId,
16        Patient = _patientId,
17        Date = selectedSlot.Date,
18        Etat = "encours",
```

LISTE DES TABLEAUX

```
19         Hour = selectedSlot.Hour
20     };
21     await SaveAppointmentAsync(appointment);
22     await DeleteTimeSlotAsync(selectedSlot);
23     await DisplayAlert("Success",
24         "Your appointment has been submitted. Wait for confirmation.",
25         "OK");
26     await Navigation.PopAsync();
27 }
```

Sauvegarde du Rendez-vous

La méthode `SaveAppointmentAsync` envoie les informations du rendez-vous à l'API backend pour les enregistrer dans la base de données. Si la demande est réussie, un message de confirmation est affiché à l'utilisateur.

```
1 private async Task SaveAppointmentAsync(Appointment appointment)
2 {
3     try
4     {
5         string apiUrl = $"{BaseUrl}/appointments";
6         using (HttpClient client = new HttpClient())
7         {
8             var json = JsonConvert.SerializeObject(appointment);
9             var content = new StringContent(json, Encoding.UTF8,
10                                         "application/json");
11             var response=await client.PostAsync(apiUrl, content);
12         }
13     }
14     catch (Exception ex)
15     {
16         Console.WriteLine($"Error: {ex.Message}");
17         await DisplayAlert("Error",
18             "An error occurred while saving your appointment.
19             Please try again.", "OK");
20     }
21 }
```

Suppression du Créneau Horaire Réservé

Une fois qu'un rendez-vous est enregistré, le créneau horaire sélectionné est supprimé de la liste de disponibilité du médecin en appelant l'API avec la méthode `DeleteTimeSlotAsync`.

```
1 private async Task DeleteTimeSlotAsync(DoctorAvail selectedSlot)
2 {
3     try
4     {
5         string apiUrl=${BaseUrl}/doctor_avail/{selectedSlot.Id};
6         using (HttpClient client = new HttpClient())
```

LISTE DES TABLEAUX

```
7     {
8         var response = await client.DeleteAsync(apiUrl);
9     }
10    }
11    catch (Exception ex)
12    {
13        Console.WriteLine($"Error:{ex.Message}");
14        await DisplayAlert("Error", "An error occurred while
15 deleting the time slot. Please try again.", "OK");
16    }
17 }
```

Description de la classe DoctorAvailabilityPage

La classe DoctorAvailabilityPage est une page dans l'application qui permet de visualiser les créneaux horaires disponibles pour un médecin et de mettre à jour un rendez-vous en sélectionnant un créneau. Elle contient des fonctionnalités pour récupérer les créneaux horaires du médecin via une API, afficher ces créneaux dans une liste, et permettre à l'utilisateur de sélectionner un créneau pour l'ajouter à son rendez-vous.

```
1 public partial class DoctorAvailabilityPage : ContentPage
2 {
3     private int doctorId;
4     private int appointmentId;
5     private const string BaseUrl = "http://192.168.1.6:4003/api";
6
7     public DoctorAvailabilityPage(int doctorId, int appointmentId)
8     {
9         InitializeComponent();
10        this.doctorId = doctorId;
11        this.appointmentId = appointmentId;
12        Console.WriteLine($"Doctor ID: {doctorId},
13 Appointment ID: {appointmentId}");
14        LoadDoctorAvailability();}}
```

Chargement de la disponibilité du médecin

La méthode LoadDoctorAvailability envoie une requête HTTP pour récupérer la liste des créneaux horaires disponibles pour un médecin spécifique. Les créneaux sont ensuite affichés dans un ListView.

```
1 private async void LoadDoctorAvailability()
2 {
3     try
4     {
5         string apiUrl = $"{BaseUrl}/doctors/availability/{doctorId}";
6         using (HttpClient client = new HttpClient())
7         {
8             Console.WriteLine($"Doctor ID: {doctorId}");
```

LISTE DES TABLEAUX

```
9         var response = await client.GetStringAsync(apiUrl);
10        Console.WriteLine("API Response: " + response);
11
12        var availabilities = JsonConvert.DeserializeObject<List<DoctorAvail>>(response);
13
14        if (availabilities != null && availabilities.Any())
15        {
16            Device.BeginInvokeOnMainThread(() =>
17            {
18                AvailabilityListView.ItemsSource =
19                    availabilities;
20            });
21        }
22    }
23
24    else
25    {
26        Console.WriteLine
27            ("No availability data returned");
28    }
29}
30
31    catch (Exception ex)
32    {
33        Console.WriteLine($"Error fetching availability:
34 {ex.Message}");
35    }
36}
```

6.1 Sélection et mise à jour du rendez-vous

Lorsque l'utilisateur sélectionne un créneau horaire, la méthode `OnSelectAvailabilityClicked` est déclenchée. Elle effectue deux actions :

1. Supprime le créneau horaire de la liste des disponibilités.
2. Met à jour le rendez-vous avec les informations du créneau sélectionné.

```
1 private async void OnSelectAvailabilityClicked(object sender,
2                                                 EventArgs e)
3 {
4     var button = (Button)sender;
5     var selectedAvailability=(DoctorAvail)button.BindingContext;
6
7     try
8     {
9         string deleteApiUrl=
10             $"{BaseUrl}/doctor_avail/{selectedAvailability.Id}";
11
12         using (HttpClient client = new HttpClient())
```

LISTE DES TABLEAUX

```
13     {
14         var deleteResponse=await client
15                         .DeleteAsync(deleteApiUrl);
16
17         if (deleteResponse.IsSuccessStatusCode)
18         {
19             Console.WriteLine
20                 ("Availability\u00a0deleted\u00a0successfully.");
21         }
22         else
23         {
24             await DisplayAlert("Error",
25                 "Failed\u00a0to\u00a0delete\u00a0the\u00a0availability", "OK");
26             return;
27         }
28     }
29
30     string updateApiUrl =
31         $"{BaseUrl}/appointments/{appointmentId}";
32
33     var updatePayload = new
34     {
35         date = selectedAvailability.Date,
36         hour = selectedAvailability.Hour,
37         etat = "en cours",
38     };
39
40     using (HttpClient client = new HttpClient())
41     {
42         var content = new StringContent
43             (JsonConvert.SerializeObject(updatePayload),
44             System.Text.Encoding.UTF8, "application/json");
45         var updateResponse = await client.
46                         PutAsync(updateApiUrl, content);
47
48         if (updateResponse.IsSuccessStatusCode)
49         {
50             await DisplayAlert("Success",
51                 "Appointment\u00a0updated\u00a0successfully", "OK");
52             // Optionally, perform other actions or navigate
53             await Navigation.PopAsync();
54             MessagingCenter.Send(this, "AppointmentUpdated");
55         }
56         else
57         {
58             await DisplayAlert("Error",
59                 "Failed\u00a0to\u00a0update\u00a0appointment", "OK");
60         }
61     }
62 }
```

```
61         }
62     }
63     catch (Exception ex)
64     {
65         Console.WriteLine
66         ($"Error updating appointment: {ex.Message}");
67         await DisplayAlert("Error",
68         "An error occurred while updating the appointment", "OK");
69     }
70 }
```

7 Plan de test du partie mobile

7.1 Plan de test

LISTE DES TABLEAUX

ID du Cas de Test	User Story	Enigence	Condition de Test	Objet de Test	Base de Test	Priorité	Résultat Attendu	Résultat Observé	Résultat	Commentaire
TC_Patient_01	Authentification du Patient	Sécurité	Login avec identifiants valides	Interface d'authentification	Spécification API	Haute	Le patient se connecte avec succès avec des identifiants valides.	Le patient s'est connecté avec succès, tableau de bord affiché.	Succès	Fonctionnalité entièrement opérationnelle.
TC_Patient_02	Demande de Rendez-vous	Gestion RDV	Formulaire de RDV	Interface de demande	Spécification fonctionnelle	Haute	Le patient peut demander un rendez-vous après s'être connecté.	Rendez-vous ajouté à la liste des demandes en attente.	Succès	Interface et validation fonctionnent bien.
TC_Patient_03	Notification pour confirmation de RDV	Notification	Confirmation par secrétaire/agent	Notification utilisateur	Scénarios de notification	Normale	Le patient reçoit une notification si la secrétaire ou l'agent confirme la demande de rendez-vous.	Notification reçue par email avec informations correctes.	Succès	Le temps de livraison de notification est bon.
TC_Patient_04	Notification pour Modification de RDV	Notification	Modification par secrétaire	Notification utilisateur	Scénarios de notification	Normale	Le patient reçoit une notification si la secrétaire modifie le rendez-vous.	Notification reçue par email avec informations correctes.	Succès	Le temps de livraison de notification est bon.
TC_Patient_05	Modification de Rendez-vous	Gestion RDV	Modification par patient	Interface de modification	Spécification fonctionnelle	Haute	Le patient demande avec succès une modification d'un rendez-vous existant.	Modification envoyée avec succès, mise à jour de l'état à "En cours".	Succès	La logique d'approbation reste à tester.
TC_Patient_06	Notification d'annulation de Rendez-vous	Notification	Modification par docteur	Interface de modification	Spécification fonctionnelle	Haute	Le patient reçoit une notification si le docteur annule le rendez-vous.	Notification reçue par email avec informations correctes.	Succès	Le temps de livraison de notification est bon.

FIGURE 54 – Plan de test de la partie mobile

7.2 Scénario de test

ID du Scénario de Test	Nom du Scénario de Test	User Story	Pré-condition	Étapes de Test	Données de Test	Post-condition	Priorité
SC_Patient_01	Authentification du Patient	En tant que patient, je veux m'authentifier	- Compte patient valide; - Application fonctionnelle; - Connexion stable	1. Ouvrir l'application; 2. Entrer le nom d'utilisateur et le mot de passe; 3. Cliquer sur "Connexion"; 4. Vérifier le message de succès.	login: patient01; Mot de passe : mdpValide	Le patient est connecté et redirigé vers le tableau de bord.	Haute
SC_Patient_02	Demande de Rendez-vous	En tant que patient, je veux demander un rendez-vous	- Patient connecté; - Créneaux disponibles	1. Accéder à "Demander un Rendez-vous"; 2. Choisir la date et l'heure; 3. Soumettre la demande; 4. Vérifier que la demande est listée comme "En attente".	- Date du rendez-vous : 2024-12-20; - Heure : 10:30	La demande de rendez-vous est affichée comme "En attente" dans le profil du patient.	Haute
SC_Patient_03	Notification pour Modification de RDV	En tant que patient, je veux être notifié pour les modifications	- Rendez-vous confirmé; - Modification effectuée par la secrétaire	1. La secrétaire modifie la date ou l'heure; 2. Le système envoie une notification; 3. Le patient reçoit et consulte la notification.	Type de notification : Email	La notification contient les détails de la modification et confirme les changements.	Normale
SC_Patient_04	Modification de Rendez-vous	En tant que patient, je veux modifier un rendez-vous	- Patient connecté; - Rendez-vous existant	1. Accéder à la section des rendez-vous; 2. Sélectionner un rendez-vous à modifier; 3. Proposer des changements (date ou heure); 4. Soumettre la demande.	- Nouvelle date : 2024-12-21; - Nouvelle heure : 15:00	La demande de modification est envoyée à la secrétaire pour approbation.	Haute

FIGURE 55 – Scénario de test de la partie mobiles

7.3 Rapport de test

ID Cas de Test	Testeur	Plateforme	Priorité	Résultat	Commentaires	Cout du Test
TC_Patient_01	QA Team	Android	Haute	Succès	Aucun problème rencontré.	2 heures
TC_Patient_02	QA Team	Web	Haute	Succès	Validation correcte des données d'entrée.	1.5 heures
TC_Patient_03	QA Team	iOS	Normale	Succès	Notifications reçues en temps réel.	2 heures
TC_Patient_04	QA Team	Android	Haute	Succès	Interface intuitive, aucun bug trouvé.	3 heures
TC_Patient_05	QA Team	Web	Haute	Succès	Modification envoyée et mise à jour de l'état "En attente".	2 heures
TC_Patient_06	QA Team	iOS	Haute	Succès	Notification d'annulation reçue correctement.	2 heures

FIGURE 56 – Rapport de test de la partie mobile

8 Conclusion

Ce chapitre a présenté l'organisation du projet à travers les sprints, la conception des interfaces et l'explication du code. Chaque sprint a permis de structurer le développement en étapes claires, tandis que l'interface a été pensée pour optimiser l'expérience utilisateur. Le code a été détaillé pour expliquer son fonctionnement, en particulier les interactions avec le backend.

Conclusion Générale

Ce rapport a mis en œuvre l'ensemble du processus de développement d'une application mobile et bureau, avec une attention particulière portée à la gestion des rendez-vous dans un centre médical. Nous avons commencé par présenter le contexte du projet, en exposant les objectifs et les rôles de chaque acteur impliqué, ainsi que la méthodologie utilisée pour structurer le développement. Ensuite, nous avons détaillé les spécifications fonctionnelles et non fonctionnelles, ce qui a permis de poser les bases nécessaires à la mise en œuvre de l'application.

La phase de réalisation a permis d'explorer l'environnement de développement et les outils utilisés, notamment Xamarin.Forms et MySQL pour la gestion de la base de données. Nous avons également détaillé les différentes interfaces de l'application, en précisant les fonctionnalités de chaque module, telles que la gestion des rendez-vous, la consultation des horaires des médecins, ainsi que les mécanismes de réservation et de confirmation. Les tests effectués ont assuré le bon fonctionnement des fonctionnalités critiques, garantissant une expérience utilisateur fluide et fiable.

Ce projet nous a offert une excellente opportunité d'élargir nos compétences en développement mobile et d'approfondir nos connaissances en gestion des bases de données et en développement d'applications multiplateformes. Il nous a également permis de mieux comprendre les aspects pratiques du travail en équipe et de la collaboration avec d'autres développeurs, dans le cadre de la gestion de projets complexes.

En conclusion, ce projet n'a pas seulement permis de concevoir et réaliser une application fonctionnelle et robuste, mais a également constitué une expérience d'apprentissage précieuse pour notre future carrière professionnelle, nous préparant à relever les défis du développement logiciel dans un environnement en constante évolution.

En dépit de toutes les fonctionnalités fournies, notre projet reste extensible et ouvert à de nombreuses perspectives qui lui donneront davantage d'ampleur. Nous espérons poursuivre le développement de l'amélioration de l'expérience utilisateur et de la sécurisation accrue. Bien que la connexion via Google et le reCAPTCHA offrent une protection de base.

Bibliographie

- [1] Staruml, n.d. Consulté le 16 octobre 2024. URL : <https://staruml.io/>.
- [2] Angular. Définition angular, 2024. URL : <https://bility.fr/definition-angular#:~:text=Angular%20est%20un%20framework%20qui,utilisant%20HTML%2C%20CSS%20et%20JavaScript>.
- [3] c#. Définition c#, 2024. URL : <https://learn.microsoft.com/fr-fr/dotnet/csharp/tour-of-csharp/overview>.
- [4] Visual Studio code. Définition visual studio code, 2024. URL : <https://visualstudio.microsoft.com/fr/>.
- [5] GitHub. Qu'est-ce que github ?, 2024. URL : <https://docs.github.com/fr/get-started/start-your-journey/about-github-and-git>.
- [6] Neon. Qu'est-ce que neon, 2024. URL : <https://www.datasentinel.io/fr/integration/neon>.
- [7] Postman. Définition postman, 2024. URL : <https://bility.fr/definition-postman/>.
- [8] scrum. Qu'est-ce que scrum ?, 2024. URL : <https://www.scrum.org/>.
- [9] SpringBoot. Définition springboot, 2024. URL : <https://azure.microsoft.com/fr-fr/resources/cloud-computing-dictionary/what-is-java-spring-boot#:~:text=Qu'est%20ce%20que%20Spring,microservices%20et%20des%20applications%20web>.
- [10] Visual Studio. Définition visual studio, 2024. URL : <https://learn.microsoft.com/fr-fr/visualstudio/get-started/visual-studio-ide?view=vs-2022>.