



ÉCOLE MAROCAINE DES SCIENCES DE L'INGÉNIEUR - RABAT

RAPPORT-PROGRAMMATION PYTHON :

Application de Gestion d'Hôpital

Elèves :

LEZREGUE AYA
BAKIR AYA

Professeur :

Monsieur Khalid NAFIL

Remerciements

Avant toute chose, nous souhaitons exprimer notre plus sincère gratitude à Monsieur Khalid Nafil, professeur de programmation en Python, pour son accompagnement précieux tout au long de ce projet. Son expertise technique, sa pédagogie claire et structurée ainsi que sa passion pour l'enseignement ont été d'un grand apport dans la réussite de ce travail.

Grâce à ses explications détaillées, ses conseils judicieux et sa disponibilité constante, nous avons pu approfondir nos connaissances en développement Python et mieux comprendre les principes de la programmation orientée objet ainsi que l'utilisation du framework Django. Sa patience et son engagement envers ses étudiants nous ont particulièrement marqués et ont grandement facilité notre apprentissage.

Nous tenons également à souligner la qualité de ses interventions et l'attention individuelle qu'il accorde à chacun. Travailler sous sa supervision a été pour nous une expérience enrichissante, tant sur le plan académique que personnel.

En somme, nous remercions sincèrement Monsieur Khalid Nafil pour la confiance qu'il nous a accordée, sa rigueur bienveillante et l'intérêt constant qu'il a porté à notre progression tout au long de cette formation.

Résumé

Dans le cadre de notre formation en ingénierie informatique et réseaux, nous avons développé une application de gestion d'hôpital en utilisant le framework **Django** (Python) et une base de données **MySQL**. Ce projet a pour objectif de faciliter la gestion des patients, des médecins, des rendez-vous, des consultations, des dossiers médicaux et des factures au sein d'un établissement de santé.

L'application propose une interface sécurisée avec un système d'authentification pour différents types d'utilisateurs (Admin, Médecin, Patient). L'administrateur a la possibilité de gérer les comptes, les rendez-vous et les dossiers médicaux. Le médecin peut créer des consultations, accéder aux dossiers médicaux des patients, et visualiser ses rendez-vous. Chaque patient dispose d'un dossier médical unique contenant l'historique des consultations, diagnostics, traitements, allergies, et maladies chroniques.

Les principales fonctionnalités réalisées incluent :

- Création et gestion des utilisateurs (Admin, Médecin, Patient),
- Gestion des rendez-vous et des consultations,
- Création automatique de factures liées aux consultations,
- Système CRUD complet pour toutes les entités (patients, médecins, dossiers...),
- Interface graphique claire et fonctionnelle avec Django Admin et des templates HTML.

Ce projet nous a permis de mettre en pratique nos compétences en **programmation orientée objet, développement web avec Django, modélisation de base de données, et gestion de projet logiciel**. Il illustre concrètement l'importance de l'informatisation des processus médicaux pour une meilleure organisation et un gain de temps significatif au sein des structures hospitalières.

Introduction générale

À l'ère du numérique, les établissements de santé sont appelés à moderniser leurs processus pour répondre aux exigences croissantes en matière de gestion, de sécurité et d'accessibilité des données médicales. La mise en place de systèmes informatiques performants devient ainsi un levier essentiel pour améliorer la qualité des soins et optimiser les opérations internes.

Dans ce contexte, notre projet vise à concevoir et développer une application de gestion hospitalière intégrée. Cette solution permet aux médecins de gérer les dossiers médicaux, les rendez-vous et les consultations de manière centralisée et sécurisée, tout en offrant aux patients une interface simplifiée d'accès aux services de santé.

Le projet a été réalisé avec le framework Django en Python, couplé à une base de données MySQL, selon une architecture modulaire et évolutive. Il intègre également des fonctionnalités d'authentification, de gestion des rôles et de traitement des informations médicales selon les bonnes pratiques de développement web.

Ce rapport retrace l'ensemble des étapes de réalisation du projet, de l'analyse des besoins à la mise en œuvre technique, tout en exposant les choix technologiques, les défis rencontrés et les perspectives d'évolution future.

Objectifs généraux du projet

Les objectifs principaux de ce projet de classe sont :

- Concevoir et développer une application web fonctionnelle pour la gestion hospitalière, en utilisant Python et le framework Django.
- Mettre en pratique les concepts de programmation orientée objet et de développement web vus en cours de Python.
- Permettre une gestion centralisée des utilisateurs (patients, médecins, administrateurs) avec des interfaces distinctes selon les rôles.
- Faciliter la prise de rendez-vous en ligne pour les patients et la gestion des plannings pour les médecins.
- Assurer la gestion sécurisée des consultations et des dossiers médicaux.
- Automatiser la génération et la consultation des factures.
- Mettre en œuvre un système d'authentification sécurisé et une navigation intuitive, en s'appuyant sur les fonctionnalités de Django.

Méthodologie de travail

La réalisation de ce projet de classe a suivi une approche structurée, comprenant les phases suivantes :

1. **Analyse des besoins et étude du cahier des charges :**
Compréhension des fonctionnalités attendues pour l'application de gestion hospitalière, telles que définies dans le sujet du projet.
2. **Conception :** Définition de l'architecture de l'application, modélisation de la base de données (entités et relations) et esquisse des principales interfaces utilisateur.
3. **Développement :** Implémentation progressive des différentes fonctionnalités en Python avec le framework Django. Cela a inclus le développement des modèles, des vues, des templates HTML/CSS, et la gestion des formulaires.
4. **Tests et validation :** Vérification du bon fonctionnement de chaque module et de l'application dans son ensemble, en s'assurant que les fonctionnalités répondent aux exigences initiales.
5. **Rédaction du rapport :** Documentation du travail effectué, des choix techniques réalisés, et des résultats obtenus.

Structure du rapport

- **Chapitre 1 : Contexte général du projet** introduit le projet, décrit son cadre général, la problématique qu'il vise à adresser, et la planification mise en œuvre.
- **Chapitre 2 : Étude du projet** détaille l'analyse fonctionnelle, incluant les besoins des utilisateurs et les cas d'utilisation, ainsi que l'étude technique qui couvre les contraintes et les choix technologiques.
- **Chapitre 3 : Analyse et conception** expose la modélisation UML (diagrammes de classes, de séquence, d'activités), la conception du modèle de données, et l'architecture logicielle retenue pour l'application.
- **Chapitre 4 : Réalisation** décrit l'environnement de développement, le processus de développement de l'application (interfaces, connexion base de données, fonctionnalités implémentées), ainsi que les tests et validations effectués.

Enfin, la **Conclusion Générale** résume les travaux accomplis, discute des apports de ce projet de classe, des limites rencontrées et des améliorations possibles, et ouvre sur des perspectives. Les **Annexes** regroupent les éléments complémentaires tels que les diagrammes complets, des extraits de code pertinents et des captures d'écran de l'application.

Table des matières

Remerciements	1
Résumé	2
Introduction générale	3
Objectifs généraux du projet	4
Méthodologie de travail	5
Structure du rapport	6
1 Contexte général du projet	9
1.1 Présentation générale du projet	9
1.1.1 Description du projet	9
1.1.2 Étude de l'existant	9
1.1.3 Problématique	9
2 Planification du projet	11
2.1 Planification du projet	11
2.1.1 Objectifs à atteindre	11
2.1.2 Planning prévisionnel	11
2.1.3 Répartition des tâches	12
2.2 Analyse fonctionnelle	13
2.2.1 Besoins fonctionnels	13
2.2.2 Cas d'utilisation	13
2.3 Analyse Technique	14
2.3.1 Contraintes techniques	14
2.3.2 Justification des choix	15
3 Analyse et conception	17
3.1 Analyse UML	17
3.1.1 Diagramme de classes	17
3.1.2 Diagrammes de séquence	19
3.1.3 Diagrammes d'activités	23
3.2 Conception de l'architecture	26
3.2.1 Architecture logicielle (MVT)	26
3.2.2 Description des modules (Applications Django)	27
4 Réalisation	29
4.1 Environnement de développement	29

Table des matières

4.1.1	IDE, outils utilisés	29
4.1.2	Structure du projet	30
4.2	Développement de l'application	31
4.3	Développement de l'application	31
4.3.1	Interface utilisateur	31
4.3.2	Connexion base de données	32
4.3.3	Fonctionnalités implémentées	33
4.3.4	Gestion des exceptions et validations	34
4.4	Tests et validation	35
	Conclusion Générale	49

Chapitre 1

Contexte général du projet

1.1 Présentation générale du projet

1.1.1 Description du projet

Le projet présenté dans ce rapport consiste en la création d'une application web de gestion hospitalière, développée en Python à l'aide du framework Django. Cette plate-forme a pour ambition de moderniser et simplifier les interactions courantes au sein d'un établissement de santé fictif. Elle est conçue pour gérer de manière intégrée les informations des différents types d'utilisateurs (patients, médecins, administrateurs), la prise et la gestion des rendez-vous, le suivi des consultations, la tenue des dossiers médicaux numérisés et la génération des factures. L'objectif est de fournir un système où chaque utilisateur dispose d'une interface distincte et adaptée à son rôle, rendue via des templates HTML/CSS et gérée par Django, garantissant une expérience personnalisée et sécurisée, tout en démontrant la maîtrise des concepts de développement web avec Python.

1.1.2 Étude de l'existant

Avant d'entreprendre le développement de notre application, une brève analyse du contexte existant a été considérée. Actuellement, de nombreux établissements hospitaliers s'appuient sur divers systèmes pour leur gestion :

- Certains utilisent encore des méthodes manuelles ou des systèmes de gestion papier, ce qui peut entraîner des lenteurs, des risques d'erreurs et des difficultés d'accès à l'information.
- D'autres ont recours à des logiciels propriétaires, souvent complexes, coûteux à acquérir et à maintenir, et parfois peu flexibles face à l'évolution des besoins.
- Des plateformes de prise de rendez-vous en ligne existent et sont populaires, mais elles ne couvrent pas toujours l'ensemble des fonctionnalités requises par une gestion hospitalière complète (comme la gestion des dossiers médicaux, la facturation intégrée, ou des interfaces spécifiques par rôle au-delà du patient).

Notre projet, bien que réalisé dans un cadre académique, cherche à s'inspirer des besoins non couverts ou partiellement couverts par ces solutions, en proposant une approche centralisée et construite avec des technologies open-source (Python/Django). L'objectif n'est pas de concurrencer les solutions industrielles matures, mais de démontrer la faisabilité d'un tel système et d'en explorer les aspects techniques et fonctionnels clés.

1.1.3 Problématique

- **Dispersion et accessibilité de l'information :** Les données des patients, leurs historiques médicaux et leurs rendez-vous sont souvent stockés dans des systèmes hétérogènes ou sur des supports papier, rendant l'accès rapide et centralisé difficile pour les professionnels de santé.

- **Lourdeur des processus administratifs** : La prise de rendez-vous manuelle, la gestion des dossiers physiques, et la facturation peuvent être chronophages et sources d'erreurs, détournant du temps précieux qui pourrait être alloué aux soins.
- **Manque de fluidité dans la communication** : Une coordination efficace entre les patients, les médecins et le personnel administratif est cruciale. L'absence d'une plateforme commune peut entraîner des malentendus ou des retards.
- **Sécurité et confidentialité des données** : Les informations médicales sont extrêmement sensibles. Assurer leur protection contre les accès non autorisés et garantir leur intégrité est une priorité absolue que les systèmes numériques doivent adresser.
- **Optimisation du temps des praticiens** : Réduire la charge administrative des médecins leur permet de se concentrer davantage sur les aspects cliniques de leur travail.

Chapitre 2

Planification du projet

2.1 Planification du projet

2.1.1 Objectifs à atteindre

Conformément au cahier des charges fourni et aux ambitions de ce projet de classe, les objectifs spécifiques à atteindre étaient les suivants :

- **Gestion des utilisateurs** : Mettre en place un système permettant l'inscription des patients, l'authentification sécurisée des utilisateurs, et l'ajout des médecins. Pour cette dernière tâche, l'interface d'administration intégrée de Django a été privilégiée.
- **Prise de rendez-vous** : Développer un module permettant aux patients de prendre des rendez-vous en ligne, avec la possibilité de filtrer les médecins par spécialité.
- **Gestion des consultations et dossiers médicaux** : Implémenter les fonctionnalités nécessaires pour que les médecins puissent créer, consulter et mettre à jour les dossiers médicaux électroniques de leurs patients suite aux consultations.
- **Gestion de la facturation** : Créer une fonctionnalité pour la génération et l'affichage des factures relatives aux prestations médicales.
- **Interfaces dédiées** : Concevoir et réaliser des interfaces utilisateur distinctes et adaptées pour chaque type d'utilisateur (patient, médecin), assurant une redirection appropriée après connexion, tandis que l'administrateur interagirait principalement via l'interface d'administration de Django.

2.1.2 Planning prévisionnel

Le projet s'est déroulé sur une période de 9 semaines, conformément au calendrier du cours de Python. Le travail a été organisé selon le planning prévisionnel suivant :

- **Phase 1 : Analyse et Spécification (1 semaine) :**
 - Lecture et compréhension approfondie du cahier des charges.
 - Identification des fonctionnalités clés et des entités de données.
 - Premières réflexions sur les technologies à utiliser (confirmation de Django, Python, SQLite).
- **Phase 2 : Conception (1 semaine) :**
 - Modélisation de la base de données (schéma conceptuel des entités Utilisateur, Patient, Médecin, RendezVous, etc.).
 - Conception de l'architecture générale de l'application Django.
 - Esquisse des principales interfaces utilisateur et du flux de navigation.
- **Phase 3 : Développement (3 semaines) :**
 - Mise en place de l'environnement de développement.
 - Développement des modèles Django.
 - Implémentation de la logique d'authentification et de gestion des rôles.

- Création des vues et des templates pour la gestion des rendez-vous.
- Développement des fonctionnalités de gestion des dossiers médicaux et des consultations.
- Mise en place du module de facturation.
- Intégration de l'interface d'administration Django pour la gestion des médecins.
- **Phase 4 : Tests et Validation (3 jours) :**
 - Tests fonctionnels manuels de chaque fonctionnalité (inscription, connexion, prise de RDV, etc.).
 - Vérification des redirections et des permissions selon les rôles.
 - Validation des formulaires.
 - Corrections des bugs identifiés.
- **Phase 5 : Rédaction du rapport et Préparation (5 jours) :**
 - Consolidation de la documentation.
 - Rédaction des différentes sections du rapport.
 - Préparation d'une éventuelle présentation/démonstration.

2.1.3 Répartition des tâches

Ce projet de classe a été réalisé en binôme, lezregue aya et bakir aya. Nous avons adopté une approche de travail hautement collaborative, où la majorité des tâches ont été abordées conjointement et simultanément. Plutôt qu'une répartition stricte des modules, nous avons privilégié des sessions de programmation en binôme ("pair programming") et des discussions constantes pour :

- **Analyse et Conception :** La définition des besoins, la modélisation de la base de données (entités Utilisateur, Patient, Médecin, RendezVous, etc.), et la conception de l'architecture générale de l'application Django ont été réalisées ensemble lors de séances de travail communes.
- **Développement des fonctionnalités :** L'implémentation des différents modules – gestion des utilisateurs (authentification, inscription, profils), prise de rendez-vous, gestion des dossiers médicaux et consultations, facturation – ainsi que la configuration de l'interface d'administration Django, ont été le fruit d'un effort conjoint. Nous avons souvent codé ensemble, l'un écrivant le code pendant que l'autre examinait, suggérait et vérifiait la logique, en alternant les rôles.
- **Création des interfaces utilisateur :** Les templates HTML/CSS ont également été développés de manière collaborative, en discutant de l'agencement et de l'ergonomie au fur et à mesure.
- **Tests et Validation :** Les tests fonctionnels ont été menés par nous deux, chacune vérifiant les fonctionnalités développées par l'autre ou ensemble.
- **Rédaction du rapport :** La rédaction de ce rapport a également été un effort partagé, chaque section étant revue et validée par nous deux.

Cette méthode de travail a permis un partage constant des connaissances, une prise de décision collective et une meilleure cohésion du code produit.

2.2 Analyse fonctionnelle

2.2.1 Besoins fonctionnels

L'étude fonctionnelle a permis d'identifier les besoins essentiels auxquels l'application doit répondre. Ces besoins sont les suivants :

- **BF1 : Gestion des Utilisateurs :**
 - BF1.1 : Permettre aux nouveaux patients de s'inscrire sur la plateforme pour créer un compte.
 - BF1.2 : Permettre à l'administrateur du système d'ajouter de nouveaux médecins (via l'interface d'administration Django).
 - BF1.3 : Assurer l'authentification sécurisée des utilisateurs (patients, médecins, administrateurs) via un formulaire de connexion.
 - BF1.4 : Rediriger chaque utilisateur vers une interface personnalisée et adaptée à son rôle (patient, médecin) après une connexion réussie.
- **BF2 : Gestion des Rendez-vous :**
 - BF2.1 : Permettre aux patients connectés de prendre un rendez-vous avec un médecin.
 - BF2.2 : Offrir la possibilité de filtrer les médecins par spécialité lors du processus de prise de rendez-vous.
 - BF2.3 : Permettre aux patients de consulter la liste de leurs rendez-vous (passés et à venir).
 - BF2.4 : Permettre aux médecins de consulter la liste des rendez-vous qui leur sont assignés.
- **BF3 : Gestion des Consultations et Dossiers Médicaux :**
 - BF3.1 : Permettre aux médecins de créer et de mettre à jour les dossiers médicaux électroniques pour leurs patients.
 - BF3.2 : Permettre aux médecins d'enregistrer les informations relatives à une consultation (diagnostic, prescription, notes).
 - BF3.3 : Permettre aux médecins de consulter les dossiers médicaux des patients qu'ils suivent.
 - BF3.4 : Permettre aux patients de consulter leur propre dossier médical (ou un résumé de celui-ci).
- **BF4 : Gestion des Factures :**
 - BF4.1 : Permettre la génération (automatique ou semi-automatique) de factures suite à une consultation ou à un ensemble de prestations.
 - BF4.2 : Permettre aux patients de consulter les factures qui leur ont été émises.
- **BF5 : Interface d'Administration :**
 - BF5.1 : Fournir une interface pour l'administrateur (via l'admin Django) pour gérer les comptes médecins (ajout, modification, suppression).

2.2.2 Cas d'utilisation

À partir des besoins fonctionnels identifiés, les principaux cas d'utilisation du système ont été définis. Ils décrivent les interactions entre les différents types d'utilisateurs (acteurs) et l'application pour réaliser des tâches spécifiques :

1. **CU1 : S'inscrire en tant que patient**
 - *Acteur : Visiteur (non authentifié)*

- *Description* : Le visiteur accède au formulaire d'inscription, remplit les informations requises (nom, prénom, email, mot de passe, etc.) et soumet le formulaire pour créer un compte patient. Le système valide les informations et crée le nouveau compte.

2. CU2 : Se connecter

- *Acteurs* : Patient, Médecin, Administrateur
- *Description* : Un utilisateur existant saisit ses identifiants (nom d'utilisateur/email et mot de passe) sur la page de connexion. Le système vérifie les identifiants et, en cas de succès, redirige l'utilisateur vers son interface dédiée.

3. CU3 : Prendre un rendez-vous

- *Acteur* : Patient
- *Description* : Le patient connecté accède à la fonction de prise de rendez-vous. Il peut filtrer les médecins par spécialité, choisir un médecin disponible, sélectionner une date et une heure pour le rendez-vous, et confirmer sa demande.

4. CU4 : Consulter ses rendez-vous

- *Acteurs* : Patient, Médecin
- *Description* : Le patient connecté peut visualiser la liste de ses rendez-vous programmés et passés. Le médecin connecté peut visualiser la liste des rendez-vous de ses patients.

5. CU5 : Gérer un dossier médical (par le médecin)

- *Acteur* : Médecin
- *Description* : Le médecin connecté peut sélectionner un patient (par exemple, suite à un rendez-vous), consulter son dossier médical existant, y ajouter des informations de consultation (diagnostic, prescription, notes), ou créer un nouveau dossier si nécessaire.

6. CU6 : Consulter son dossier médical (par le patient)

- *Acteur* : Patient
- *Description* : Le patient connecté peut accéder à une section de son espace personnel pour visualiser les informations de son propre dossier médical (ou un résumé pertinent).

7. CU7 : Consulter ses factures

- *Acteur* : Patient
- *Description* : Le patient connecté peut accéder à la liste de ses factures et visualiser les détails de chacune.

8. CU8 : Gérer les médecins (par l'administrateur)

- *Acteur* : Administrateur
- *Description* : L'administrateur, via l'interface d'administration de Django, peut ajouter de nouveaux médecins au système, modifier leurs informations, ou supprimer leurs comptes.

2.3 Analyse Technique

2.3.1 Contraintes techniques

Le développement du projet a été encadré par un ensemble de contraintes techniques, qu'elles soient imposées par le cadre du projet de classe ou choisies pour des raisons pratiques :

- **Langage et Framework imposés/choisis** : L'utilisation du langage Python (version 3.13) et du framework web Django (version 5.1) était un élément central du projet, conformément aux objectifs pédagogiques du cours.
- **Base de données** : Pour la phase de développement et dans le cadre de ce projet de classe, SQLite a été retenue comme système de gestion de base de données, en raison de sa simplicité d'intégration avec Django et de l'absence de nécessité d'un serveur de base de données dédié.
- **Interface utilisateur** : Le rendu des interfaces utilisateur devait être réalisé en utilisant HTML et CSS, intégrés via le système de templating de Django. L'utilisation de JavaScript était possible mais non obligatoire pour les fonctionnalités de base.
- **Délais du projet** : Le développement devait être complété dans les limites de temps allouées pour le projet de classe (durée de 6 semaines).
- **Hébergement et Déploiement** : Le projet était destiné à fonctionner en environnement de développement local. Aucun déploiement en production sur un serveur distant n'était requis dans le cadre de ce projet.
- **Sécurité (niveau projet de classe)** : Bien qu'une sécurité de niveau production ne soit pas l'objectif principal, les mécanismes de base de Django pour la protection contre les failles courantes (CSRF, XSS via l'échappement des templates) et la gestion sécurisée de l'authentification devaient être utilisés.
- **Ressources** : Le projet a été développé avec les ressources matérielles et logicielles standards disponibles (ordinateurs personnels, VS Code).

2.3.2 Justification des choix

Les choix technologiques décrits ci-dessus ont été guidés principalement par les objectifs pédagogiques du cours de Python et du framework Django, ainsi que par des considérations de pragmatisme pour un projet de cette envergure réalisé en binôme dans un temps limité :

- **Python et Django :**
 - *Contexte pédagogique* : L'utilisation de Python (version 3.13) et du framework Django (version 5.1) était au cœur des exigences de ce projet de classe. L'objectif principal était de mettre en application les concepts et les techniques de développement web enseignés durant le cours, en utilisant spécifiquement cet écosystème.
 - *Adéquation au projet* : Au-delà de l'aspect imposé, Django s'est avéré particulièrement adapté pour développer une application de gestion hospitalière. Ses fonctionnalités "prêtes à l'emploi" comme l'ORM (Object-Relational Mapper) pour l'interaction avec la base de données, le système d'authentification robuste, l'interface d'administration générique (utilisée pour la gestion des médecins), et son système de templating ont considérablement simplifié et accéléré le développement des fonctionnalités requises (gestion des utilisateurs, des rendez-vous, des dossiers médicaux, etc.). La structure MVT (Model-View-Template) de Django a également favorisé une bonne organisation du code.
- **SQLite :**
 - *Simplicité et intégration* : Pour un projet de classe destiné à un développement local et sans contraintes de production, SQLite a été choisi pour sa grande simplicité. Il ne nécessite aucune configuration de serveur de base de données sé-

paré, étant un simple fichier, et s'intègre nativement et sans effort avec Django. Cela nous a permis de nous concentrer sur le développement des fonctionnalités de l'application plutôt que sur l'administration d'une base de données plus complexe.

— **HTML/CSS et Templates Django :**

- *Standard du web* : HTML et CSS sont les technologies standards incontournables pour la création d'interfaces utilisateur web. Le projet n'a pas nécessité l'utilisation de JavaScript pour ses fonctionnalités de base.
- *Dynamisme avec Django* : Le système de templating de Django a permis de générer dynamiquement les pages HTML en y intégrant les données issues de la base de données et de la logique métier des vues, ce qui est essentiel pour une application interactive.

— **Git :**

- *Bonnes pratiques* : L'utilisation de Git, même pour un projet en binôme travaillant en étroite collaboration, a été adoptée comme une bonne pratique de développement. Cela a permis de suivre l'évolution du code, de gérer différentes versions ou idées (via des branches si besoin), et de faciliter la fusion du travail ou le retour à des états antérieurs en cas de problème.

— **Visual Studio Code (VSCode) :**

- *Productivité et flexibilité* : Le choix de VSCode comme environnement de développement intégré s'est fait pour sa légèreté, sa grande flexibilité grâce aux extensions, son auto-complétion efficace pour Python et Django, ses outils de débogage intégrés et sa popularité qui facilite la recherche de support et de configurations.

Chapitre 3

Analyse et conception

3.1 Analyse UML

La phase d'analyse UML (Unified Modeling Language) a été une étape cruciale pour modéliser les aspects statiques et dynamiques de notre application de gestion hospitalière. Elle a permis de définir clairement les entités du système, leurs interactions et les processus métier.

3.1.1 Diagramme de classes

L'analyse a débuté par la création d'un diagramme de classes afin de représenter les entités principales de l'application `HOSPITAL_SYSTEM` et les relations qui les unissent. Ce diagramme constitue le fondement de notre modèle de données et a guidé la création des modèles Django. Les classes principales identifiées sont :

- **Utilisateur** (modèle personnalisé héritant de `AbstractUser` de Django) :
 - Sert de classe de base pour tous les utilisateurs du système, intégrant les fonctionnalités d'authentification robustes fournies par Django.
 - *Attributs hérités et personnalisés* : `id`, `password`, `last_login`, `is_superuser`, `username`, `first_name`, `last_name`, `email`, `is_staff`, `is_active`, `date_joined`.
 - *Attributs spécifiques ajoutés* : `telephone` (`CharField`), `adresse` (`CharField`), `cin` (`CharField`), `ville` (`CharField`), `genre` (`CharField` avec choix prédéfinis '`M`'/'`F`''), `role` (`CharField` avec choix prédéfinis '`patient`'/'`medecin`'/'`admin`').
 - Gère également les `groups` et `user_permissions` de Django pour une gestion fine des droits.
- **Patient** (hérite de **Utilisateur** via l'héritage multi-tables de Django) :
 - Représente un patient enregistré dans le système. Le champ `utilisateur_ptr` constitue la clé primaire et établit une relation `OneToOneField` vers la table **Utilisateur**.
 - *Attributs spécifiques* : `raison_consultation` (`CharField`), `visite` (`BooleanField`, initialisé à `False`).
 - *Relations* : Un **Patient** est un type d'**Utilisateur**. Il peut être associé à plusieurs **RendezVous** et plusieurs **Factures**. Un **Patient** est également lié à un **DossierMedical** (bien que la relation soit définie depuis **DossierMedical** vers **Patient** dans le modèle Django, conceptuellement, le dossier appartient au patient).
- **Medecin** (hérite de **Utilisateur** via l'héritage multi-tables de Django) :
 - Représente un professionnel de santé. Similairement à **Patient**, `utilisateur_ptr` est la clé primaire et une `OneToOneField` vers **Utilisateur**.
 - *Attributs spécifiques* : `specialite` (`CharField`).
 - *Relations* : Un **Medecin** est un type d'**Utilisateur**. Il peut être associé à plusieurs **RendezVous**.
- **Admin** (modèle proxy de **Utilisateur**) :

- Utilisé pour conférer une interface ou une logique métier spécifique aux utilisateurs ayant le rôle administrateur, sans créer de table dédiée en base de données. Ce modèle est basé sur **Utilisateur**.
- **RendezVous** :
 - Représente une prise de rendez-vous entre un patient et un médecin.
 - *Attributs* : **id** (clé primaire auto-générée), **date** (TextField), **heure** (TimeField), **statut** (CharField, ex : 'programmé', 'annulé', 'réalisé').
 - *Relations* : Est associé à un **Patient** (via une ForeignKey) et à un **Medecin** (via une ForeignKey). Un **RendezVous** est lié à une unique **Consultation** (via une OneToOneField définie depuis **Consultation**).
- **Consultation** :
 - Modélise une interaction médicale entre un médecin et un patient.
 - *Attributs* : **id**, **diagnostic** (TextField), **traitement_prescrit** (TextField), **duree** (DurationField), **medicament** (CharField).
 - *Relations* : Est liée à un unique **RendezVous** (via une OneToOneField). Une **Consultation** est également liée à un **DossierMedical** (via une OneToOneField définie depuis **DossierMedical**) et peut générer une **Facture** (relation définie par une ForeignKey depuis **Facture**).
- **DossierMedical** :
 - Contient les informations médicales consolidées d'un patient.
 - *Attributs* : **id**, **groupe_sanguin** (CharField), **alergie** (TextField), **maladie_chronique** (TextField).
 - *Relations* : Dans la modélisation actuelle, est lié à une unique **Consultation** (via une OneToOneField).
 - *Note de conception* : Il est important de noter que, traditionnellement, un **DossierMedical** est lié à un **Patient** et regroupe l'historique de plusieurs consultations. Le modèle actuel, liant un **DossierMedical** à une seule **Consultation**, suggère une sémantique où chaque consultation possède son propre "mini-dossier". Si l'intention est d'avoir un dossier central par patient, une relation (ForeignKey ou OneToOneField) de **DossierMedical** vers **Patient** serait à considérer.
- **Facture** :
 - Représente un document financier généré suite à une prestation (typiquement une consultation).
 - *Attributs* : **id**, **date_fact** (TextField), **montant_tot** (DecimalField).
 - *Relations* : Est liée à une **Consultation** (ForeignKey) et à un **Patient** (ForeignKey) pour faciliter le suivi.

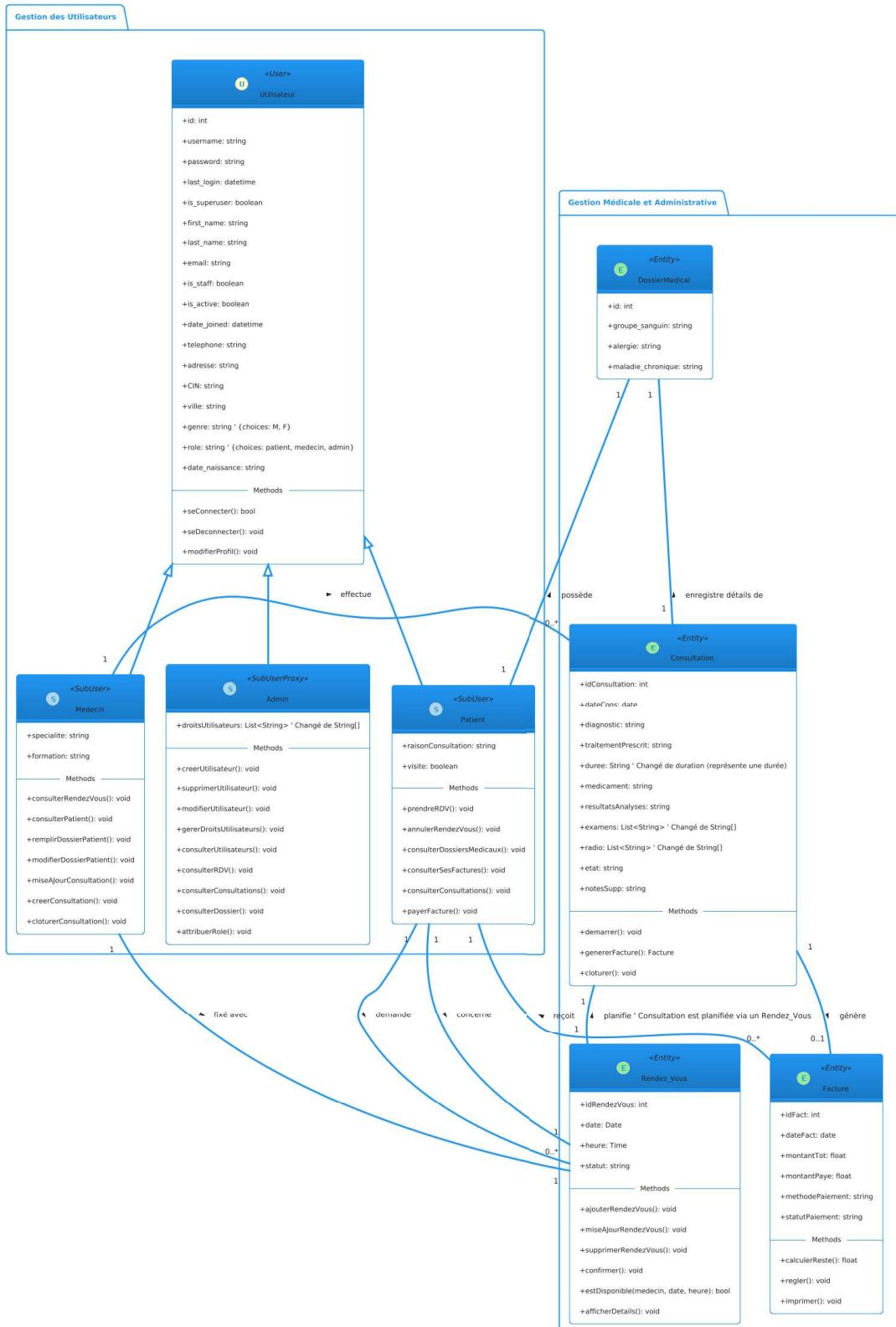


FIGURE 3.1 – Diagramme de classes initial de l'application de gestion hospitalière.

3.1.2 Diagrammes de séquence

Pour illustrer les interactions dynamiques entre les objets (ou acteurs et composants du système) lors de scénarios d'utilisation clés, des diagrammes de séquence ont été concep-

tualisés. Ces diagrammes aident à comprendre l'ordre chronologique des messages échangés pour réaliser une fonctionnalité donnée. Deux exemples significatifs sont présentés ci-dessous.

1. Diagramme de séquence - Authentification Utilisateur :

Ce diagramme décrit le processus de connexion et de déconnexion d'un utilisateur au système.

— Scénario de Connexion :

- L'Utilisateur saisit son email et son mot de passe via l'Interface Utilisateur (IU).
- L'IU transmet ces informations au Système (Vue Django / Logique d'authentification).
- Le Système invoque une opération de vérification des identifiants (ex : `vérifierIdentifiants(email, motDePasse)`), typiquement en interrogeant la Base de Données.
- Si les identifiants sont valides :
 - Le Système établit la session utilisateur.
 - Le Système renvoie une confirmation de succès à l'IU (ex : "Bienvenue [nom_utilisateur]").
 - L'IU affiche le message de bienvenue et redirige l'utilisateur vers son tableau de bord.
- Si les identifiants sont invalides :
 - Le Système renvoie un message d'échec à l'IU (ex : "Identifiants incorrects").
 - L'IU affiche le message d'erreur.

— Scénario de Déconnexion :

- L'Utilisateur initie une demande de déconnexion (ex : clic sur `seDeconnecter()`) via l'IU.
- L'IU transmet la requête au Système.
- Le Système ferme la session de l'utilisateur (ex : `déconnecterSession()`).
- Le Système renvoie une confirmation à l'IU (ex : "Déconnexion réussie").
- L'IU affiche le message et redirige vers la page de connexion ou d'accueil.

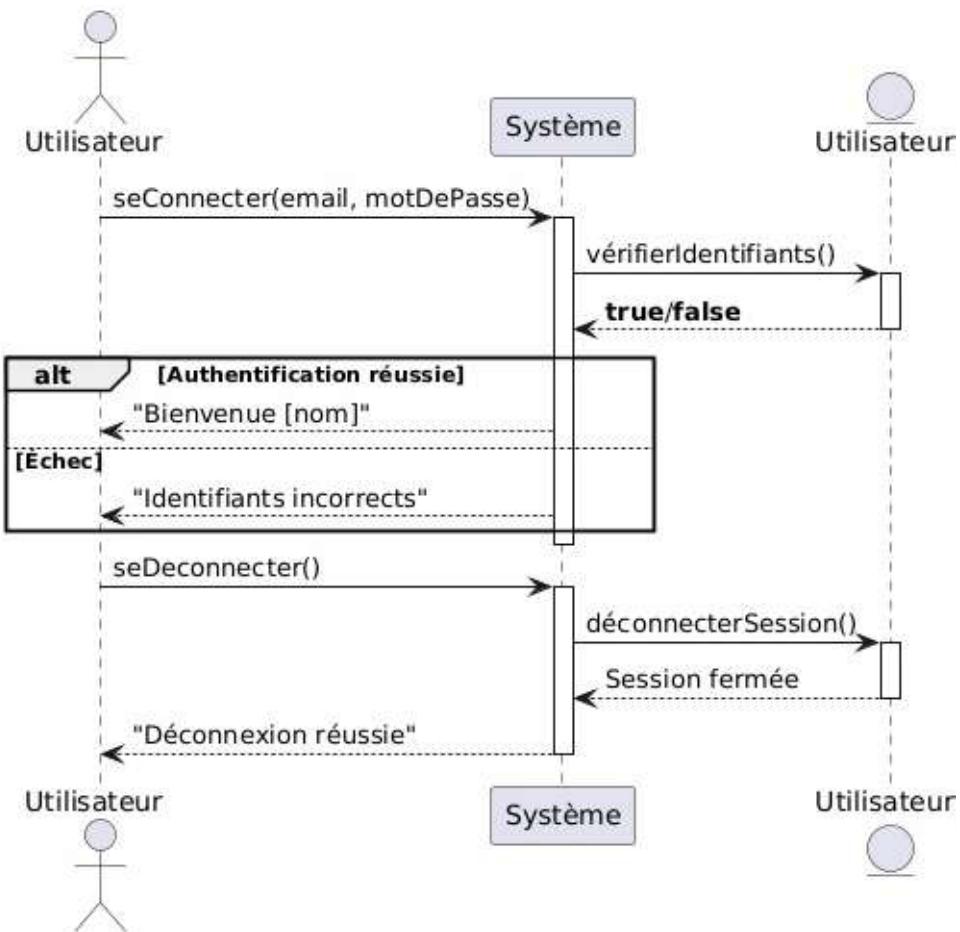


FIGURE 3.2 – Diagramme de séquence illustrant le processus d'authentification utilisateur.

2. Diagramme de séquence - Actions de l'Administrateur :

Ce diagramme illustre certaines opérations clés réalisées par un administrateur, principalement via l'interface d'administration de Django.

— Création d'un Utilisateur (ex : Médecin) :

- L'Admin interagit avec l'interface d'administration pour créer un nouvel utilisateur (ex : `creerUtilisateur(données)`), en fournissant les informations requises (nom d'utilisateur, email, mot de passe, rôle, etc.).
- Le Système (Admin Django / Modèle **Utilisateur**) instancie et sauvegarde un nouvel utilisateur (ex : `Utilisateur.objects.create_user(données)`).
- Un identifiant unique `ID_Utilisateur` est généré.
- Le Système confirme la création à l'Admin (ex : "Utilisateur créé avec succès (ID : X)").

— Attribution d'un Rôle spécifique et création du profil (ex : Médecin) :

- L'Admin (ou le système, si le rôle "Médecin" est choisi lors de la création) assigne le rôle "Médecin" à l'utilisateur (`ID_Utilisateur`) et fournit les informations spécifiques au profil médecin (ex : spécialité).
- Le Système met à jour l'attribut `role` de l'objet **Utilisateur**.
- Le Système crée une instance du modèle **Medecin** liée à l'utilisateur (ex : `Medecin.objects.create(utilisateur_ptr_id=ID_Utilisateur, specialite=...)`). L'`ID_Médecin` sera le même que `ID_Utilisateur` du fait de l'héritage.

- Le Système confirme l'opération (ex : "Profil médecin créé et rôle attribué avec succès").
- **Consultation de Données (exemples) :**
 - *Liste des Patients (par un Médecin ou Admin)* :
 - L'Utilisateur (Médecin/Admin) initie la consultation via son interface.
 - Le Système exécute une requête pour récupérer la liste des patients (ex : `Patient.objects.all()` ou filtrée) depuis la Base de Données.
 - Le Système renvoie la liste (ex : `getListePatients()`) à l'Interface qui l'affiche.
 - *Calendrier des Rendez-vous (par un Médecin ou Admin)* :
 - L'Utilisateur (Médecin/Admin) initie la consultation (ex : `ConsulterRDV()`).
 - Le Système récupère les rendez-vous depuis la Base de Données.
 - Le Système renvoie les données (ex : `getTousRDV()`) à l'Interface qui affiche le calendrier.
 - *État Financier / Factures (par un Admin)* :
 - L'Admin initie la consultation (ex : `ConsulterFactures()`).
 - Le Système récupère les factures (ex : `Facture.objects.all()` ou filtrées) depuis la Base de Données.
 - Le Système renvoie les données (ex : `getToutesFactures()`) à l'Interface qui les affiche.

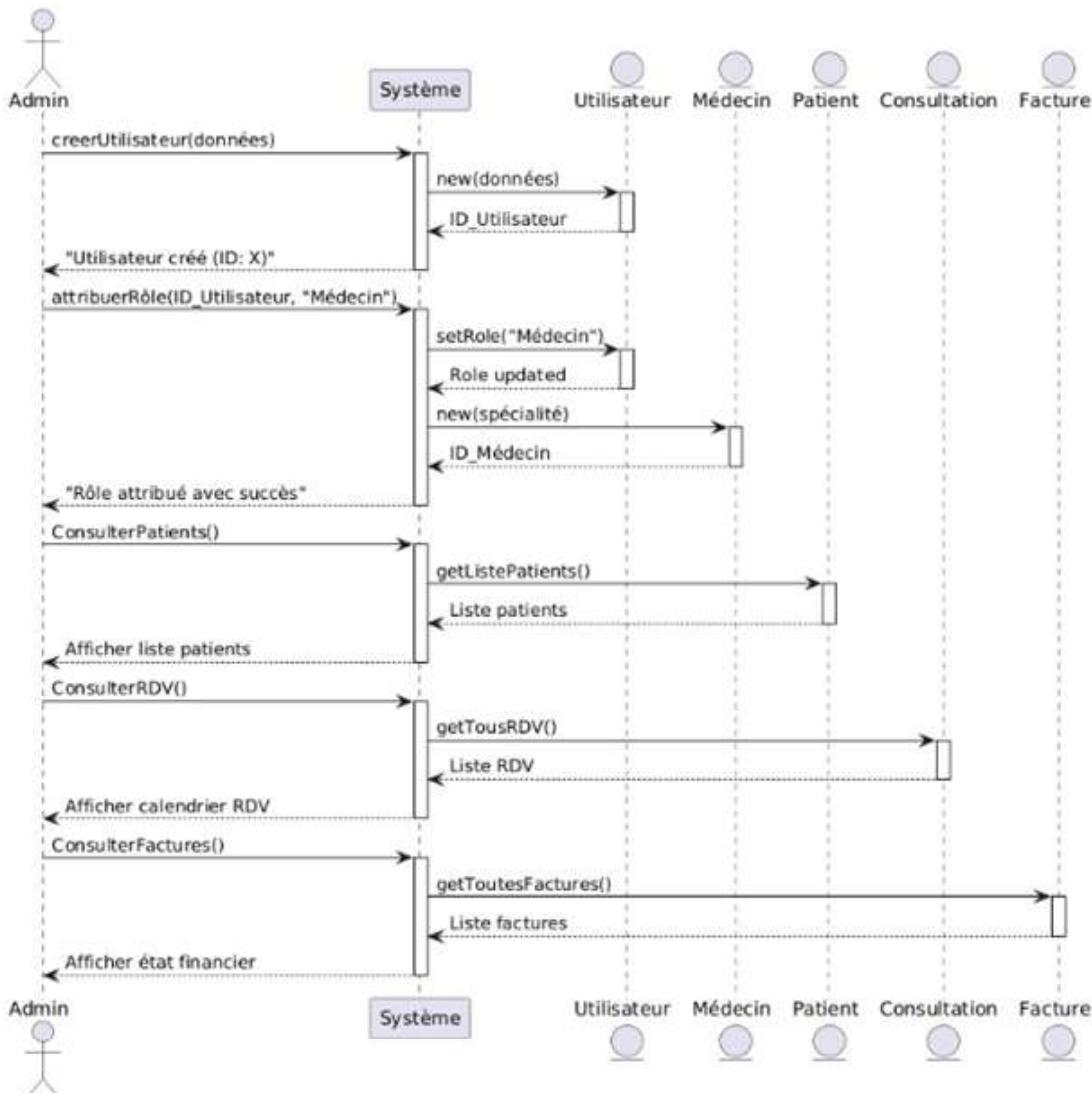


FIGURE 3.3 – Diagramme de séquence illustrant des actions typiques de l'administrateur.

3.1.3 Diagrammes d'activités

Pour compléter l'analyse dynamique, des diagrammes d'activités ont été élaborés afin de modéliser le flux de contrôle de processus métier complexes ou critiques. Ces diagrammes permettent de visualiser les étapes séquentielles, les branchements conditionnels, les actions parallèles et les responsabilités des utilisateurs ou du système. Un exemple significatif est le flux de travail d'un médecin après sa connexion à l'application.

Flux de travail principal du médecin

Ce diagramme d'activité décrit les actions typiques qu'un médecin peut entreprendre après s'être authentifié.

1. Connexion & Authentification :

- *Nœud initial* : Le médecin accède à la page de connexion.
- *Action* : Saisie des identifiants (email/username, mot de passe).
- *Action du système* : Vérification des identifiants et validation du rôle "Médecin".

- *Point de décision (Nœud de garde)* : Authentification réussie ?
 - Si oui : Accès au tableau de bord médecin (flux continue).
 - Si non : Affichage d'un message d'erreur, retour à la page de connexion (fin alternative).

2. Tableau de bord médecin et choix d'action :

- Le médecin est sur son tableau de bord et dispose de plusieurs options principales (représentées par des actions distinctes ou des branches) :
 - Option A : Consulter ses Rendez-vous (RDV) du jour/à venir.
 - Option B : Accéder à la liste de ses Patients et à leurs dossiers.

3. Flux A : Gestion des Rendez-vous

- *Action du système* : Affichage de la liste des rendez-vous programmés pour le médecin (potentiellement filtrés par date, statut).
- *Action du médecin* : Peut sélectionner un rendez-vous pour en voir les détails (patient, motif, heure).
- *Action du médecin (potentielle)* : Marquer un RDV comme "Réalisé". Ceci peut déclencher le passage au flux de "Consultation active" si la consultation n'a pas encore été enregistrée.
- *Fin de ce sous-flux (consultation simple des RDV) ou transition vers une autre activité.*

4. Flux B : Gestion des Patients et Dossiers Médicaux

- *Action du système* : Affichage de la liste des patients (ceux suivis par le médecin ou tous, selon les droits).
- *Action du médecin* : Sélectionne un patient spécifique.
- *Action du système* : Affiche le dossier médical du patient sélectionné ou une interface pour initier une consultation.
- *Point de décision (Type d'interaction avec le dossier)* :
 - *Cas 1 : Consultation du dossier existant (lecture)*
 - *Action du médecin* : Navigue dans les informations du dossier médical (antécédents, allergies, historique des consultations passées, traitements).
 - *Fin de ce sous-flux.*
 - *Cas 2 : Enregistrement d'une nouvelle consultation (active)*
 - *Action du médecin* : Initie une nouvelle entrée de consultation (peut être liée à un RDV existant ou être une consultation spontanée).
 - *Action du médecin* : Saisit les informations de la consultation : diagnostic, traitement prescrit, médicaments, durée, notes cliniques.
 - *Action du médecin (optionnelle)* : Ajoute des résultats d'examens, des ordonnances numérisées, etc.
 - *Action du système* : Enregistre les informations de la consultation dans la base de données, les associant au dossier médical du patient et au rendez-vous si applicable.
 - *Sous-flux : Clôture et Facturation de la consultation*
 - *Point de décision* : La consultation est-elle marquée comme "terminée" par le médecin ?
 - Si oui :
 - *Action du médecin/système* : Clôture officielle de la session de consultation.

- *Action du système (potentielle)* : Génération automatique (ou semi-automatique via une action de l'admin/secrétariat) de la facture correspondante, liée à la consultation et au patient.
- *Action du système* : Mise à jour du statut de la consultation et du RDV (ex : 'Réalisé', 'Facturé').
- *Fin du processus de consultation active et facturation*.

5. *Nœud final* : Fin du flux de travail principal pour cette session.

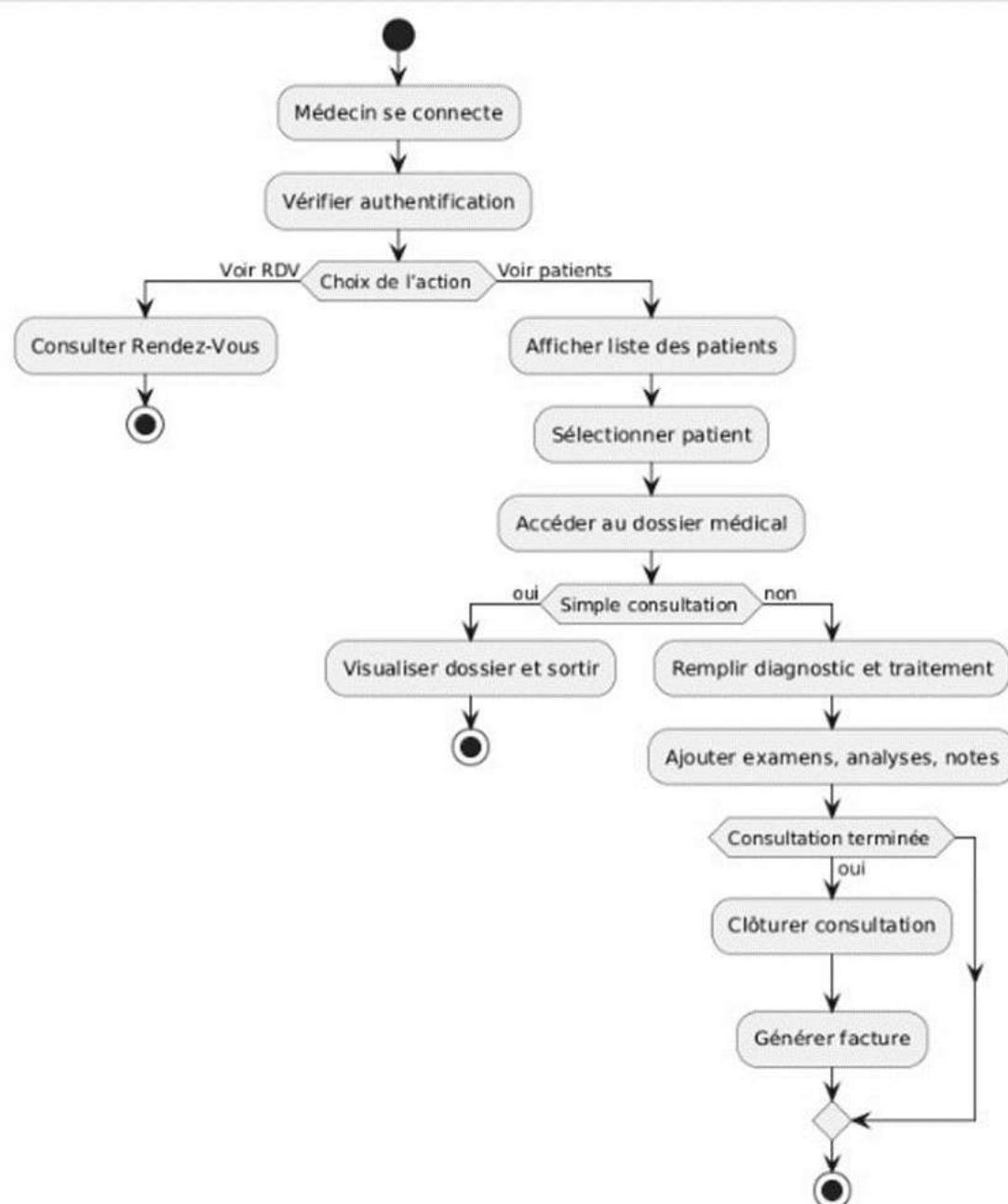


FIGURE 3.4 – Diagramme d'activité illustrant le flux de travail principal du médecin.

3.2 Conception de l'architecture

Après la phase d'analyse, la conception de l'architecture logicielle a été définie pour structurer l'application **HOSPITAL_SYSTEM**, en s'appuyant sur les patrons de conception et les capacités du framework Django.

3.2.1 Architecture logicielle (MVT)

L'application web **HOSPITAL_SYSTEM** a été développée en adoptant l'architecture MVT (Model-View-Template), qui est l'implémentation spécifique par Django du pattern de conception bien connu MVC (Model-View-Controller). Ce choix architectural favorise une séparation claire des préoccupations (données, logique de traitement, présentation), ce qui améliore la modularité, la maintenabilité et la testabilité du code. Voici comment les composants MVT se concrétisent dans notre application principale, nommée **core** :

— Model (Modèle) :

- *Rôle* : Le Modèle constitue la couche d'abstraction des données. Il représente la structure logique de nos informations (patients, médecins, rendez-vous, etc.) et gère toutes les interactions avec la base de données (SQLite dans notre cas). Il encapsule également la logique métier directement liée aux données, comme les règles de validation, les relations entre entités et les comportements spécifiques des objets.
- *Implémentation dans core/models.py* : Toutes nos entités de données (**Utilisateur**, **Patient**, **Medecin**, **Admin**, **RendezVous**, **Consultation**, **DossierMedical**, **Facture**) sont définies comme des classes Python héritant de `django.db.models.Model` (ou de `AbstractUser` pour le modèle **Utilisateur** personnalisé). L'ORM (Object-Relational Mapper) de Django se charge de traduire ces objets Python en tables de base de données et de simplifier les opérations CRUD (Create, Read, Update, Delete) sans nécessiter l'écriture de requêtes SQL directes.

— View (Vue Django) :

- *Rôle* : Dans Django, la Vue est le composant qui traite les requêtes HTTP entrantes et retourne les réponses HTTP appropriées. Elle agit comme un contrôleur : elle reçoit la requête, interagit avec les Modèles pour récupérer ou modifier les données (via l'ORM), applique une logique de traitement ou métier si nécessaire, puis sélectionne et transmet les données au Template pertinent pour la génération de la réponse finale (souvent une page HTML).
- *Implémentation dans core/views.py* : Nous avons employé une combinaison de :
 - *Vues Basées sur des Fonctions (FBV - Function-Based Views)* : Pour des logiques spécifiques et un contrôle plus fin du flux, notamment pour l'inscription (`signup`), la connexion (`login_view` avec redirection conditionnelle basée sur le rôle), l'affichage des spécialités médicales (`specialites_view`), le filtrage des médecins par spécialité (`medecins_par_specialite`), la prise de rendez-vous, et la page d'accueil (`home`).
 - *Vues Basées sur des Classes (CBV - Class-Based Views)* : Pour standardiser et simplifier les opérations CRUD sur nos modèles (**Patient**, **Medecin**, **RendezVous**, **Consultation**, **DossierMedical**, **Facture**). Django offre des vues génériques (telles que `ListView`, `DetailView`, `CreateView`, `UpdateView`, `DeleteView`) qui réduisent considérablement la quantité de code répétitif à écrire.

— **Template (Gabarit) :**

- *Rôle* : Le Template est responsable de la couche de présentation. Il définit la structure et l'apparence des données qui seront affichées à l'utilisateur dans son navigateur. Il s'agit typiquement de fichiers HTML intégrant des éléments dynamiques grâce au langage de template de Django.
- *Implémentation dans core/templates/core/* : Plusieurs fichiers HTML ont été créés (ex : `home.html`, `login.html`, `signup.html`, `specialites.html`, ainsi que des templates dédiés aux opérations CRUD pour chaque modèle). Ces templates utilisent le langage de templating de Django (DTL - Django Template Language) avec ses balises (ex : `{% for item in items %}`, `{% if condition %}`) et filtres pour afficher dynamiquement les données, gérer l'héritage de templates (via un fichier de base comme `base.html`, permettant une structure commune) et inclure des fichiers statiques (CSS, JavaScript).

Le fichier `core/urls.py` joue un rôle essentiel en assurant le mappage (routage) des URLs demandées par le navigateur vers les Vues Django appropriées. De plus, les formulaires, définis dans `core/forms.py` (par exemple, `UtilisateurCreationForm`, `RendezVousForm`, et les divers `ModelForms` utilisés avec les CBV), facilitent la collecte, la validation et le traitement des données soumises par les utilisateurs avant leur manipulation par les Vues et leur persistance par les Modèles.

Cette architecture MVT a fourni un cadre structuré et éprouvé pour le développement de l'application `core`, favorisant une gestion organisée de la logique applicative, des données et de leur présentation.

3.2.2 Description des modules (Applications Django)

Notre projet, `HOSPITAL_SYSTEM`, est architecturé autour d'une application Django principale dénommée `core`. Cette application centralise l'ensemble des modèles, vues, templates, formulaires et URLs nécessaires au fonctionnement de la plateforme de gestion hospitalière.

Application `core`

L'application `core` est le cœur fonctionnel du système.

— *Rôle et fonctionnalités couvertes :*

— *Gestion des utilisateurs et authentification :*

- Modèles : **Utilisateur**, **Patient**, **Medecin**, **Admin**.
- Vues : Fonctions dédiées pour l'inscription (`signup`), la connexion (`login_view`), la déconnexion. La gestion des profils est assurée implicitement via les modèles et l'interface d'administration de Django.
- Formulaires : `UtilisateurCreationForm` (pour l'inscription), `AuthenticationForm` de Django (pour la connexion).

— *Gestion des rendez-vous :*

- Modèle : **RendezVous**.
- Vues : Fonctions pour la sélection de spécialités (`specialites_view`), le filtrage des médecins par spécialité (`medecins_par_specialite`), l'affichage détaillé d'une fiche médecin (`fiche_medecin`), et la prise de rendez-vous effective (`prendre_rendez_vous`). Des vues basées sur les classes (CBV) sont utilisées pour les opérations CRUD standard (liste, détail, création,

- mise à jour, suppression) sur les rendez-vous (ex : `RendezVousListView`, `RendezVousCreateView`).
- Formulaires : `RendezVousForm`.
- *Gestion des consultations et des dossiers médicaux* :
 - Modèles : **Consultation**, **DossierMedical**.
 - Vues : Utilisation de CBV pour les opérations CRUD sur les consultations (ex : `ConsultationListView`, `ConsultationDetailView`) et les dossiers médicaux (ex : `DossierMedicalUpdateView`).
 - Formulaires : `ConsultationForm`, `DossierMedicalForm` (probablement des `ModelForm`).
- *Gestion de la facturation* :
 - Modèle : **Facture**.
 - Vues : Utilisation de CBV pour les opérations CRUD sur les factures (ex : `FactureListView`, `FactureCreateView`).
 - Formulaires : `FactureForm` (probablement un `ModelForm`).
- *Fonctionnalités générales et navigation* :
 - Vues : Page d'accueil (`home`), tableaux de bord spécifiques aux rôles.
- *Structure interne de l'application core* :
 - `models.py` : Contient la définition de tous les modèles de données, comme détaillé précédemment.
 - `views.py` : Héberge la logique de traitement des requêtes, combinant des vues basées sur des fonctions (FBV) pour les flux personnalisés (authentification, processus de prise de RDV) et des vues basées sur des classes (CBV) pour les opérations CRUD standard.
 - `forms.py` : Définit les formulaires Django utilisés pour la saisie et la validation des données utilisateur (ex : `UtilisateurCreationForm`, `RendezVousForm`, et les `ModelForm` pour les autres entités).
 - `urls.py` : Gère le routage des URLs spécifiques à l'application `core` vers les vues correspondantes. Ce fichier est inclus dans le routeur principal du projet.
 - `templates/core/` : Répertoire contenant les fichiers HTML (templates) pour l'affichage des différentes pages et interfaces (ex : `home.html`, `login.html`, `signup.html`, `specialites.html`, `prendre_rendez_vous.html`, ainsi que les templates génériques pour les listes, détails, formulaires de création/modification et pages de confirmation de suppression).
 - `static/core/css/` (et potentiellement `js/`, `img/`) : Contient les fichiers statiques tels que les feuilles de style CSS (ex : `home.css`, `login.css`), les scripts JavaScript et les images propres à l'application `core`.
 - `admin.py` : Configure l'enregistrement des modèles de `core` pour qu'ils soient accessibles et gérables via l'interface d'administration de Django, offrant des fonctionnalités de gestion de données prêtes à l'emploi.
 - `apps.py` : Fichier de configuration de l'application Django.
 - `tests.py` : Destiné à contenir les tests unitaires et d'intégration pour l'application `core`.

Le projet principal `HOSPITAL_SYSTEM` (le répertoire qui contient `settings.py` et le `urls.py` global) orchestre la configuration générale du projet (base de données, applications installées, middleware, etc.) et le routage principal des URLs, lequel délègue les URLs préfixées par `core/` (par exemple) à l'application `core` elle-même.

Chapitre 4

Réalisation

4.1 Environnement de développement

4.1.1 IDE, outils utilisés

La réalisation de l'application HOSPITAL_SYSTEM s'est appuyée sur un environnement de développement configuré avec les outils suivants :

- **Système d'exploitation** : Windows 11
- **Langage de programmation** : Python 3.13
 - Installé globalement sur le système via [Indiquez comment, ex : le site officiel Python.org, le Microsoft Store, Anaconda].
- **Framework web** : Django 5.1
 - Installé globalement via pip (le gestionnaire de paquets Python).
- **Environnement de Développement Intégré (IDE)** : Visual Studio Code (VSCode)
 - Utilisé pour l'édition du code, le débogage, la gestion des fichiers du projet et l'intégration native avec Git et le terminal.
 - Extensions VSCode principales utilisées : L'extension officielle Python de Microsoft.
- **Base de données** : SQLite 3
 - Utilisée comme moteur de base de données par défaut avec Django.
- **Système de gestion de version** : Git
 - Utilisé pour le suivi des modifications du code source et la collaboration, principalement via les fonctionnalités intégrées à VSCode et la ligne de commande.
- **Navigateur web** : Microsoft Edge
 - Utilisé pour visualiser l'application, tester les fonctionnalités et utiliser ses outils de développement intégrés.
- **Terminal (intégré à VSCode, PowerShell, ou Invite de commandes Windows)** :
 - Utilisé pour exécuter les commandes Django (`manage.py runserver`, `manage.py makemigrations`, `manage.py migrate`, etc.), les commandes Git.

4.1.2 Structure du projet

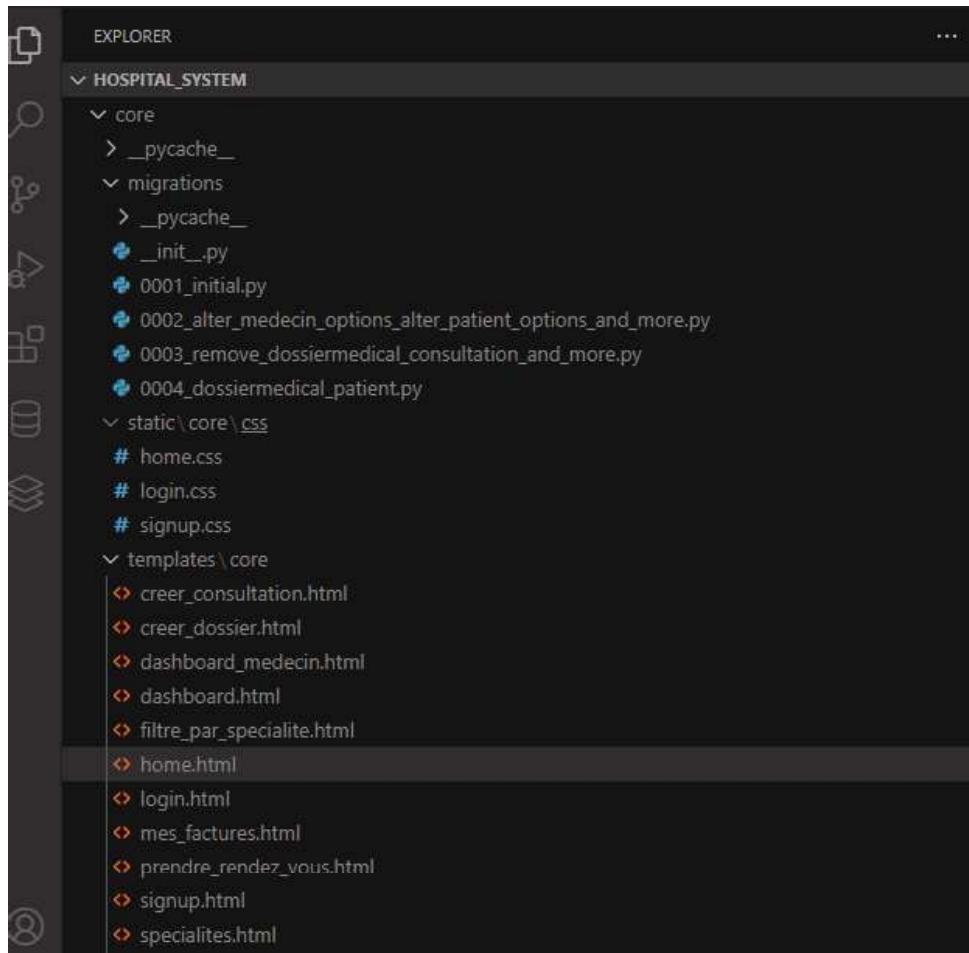


FIGURE 4.1 – Structure des dossiers et fichiers principaux du projet HOSPITAL_SYSTEM.

4.2 Développement de l'application

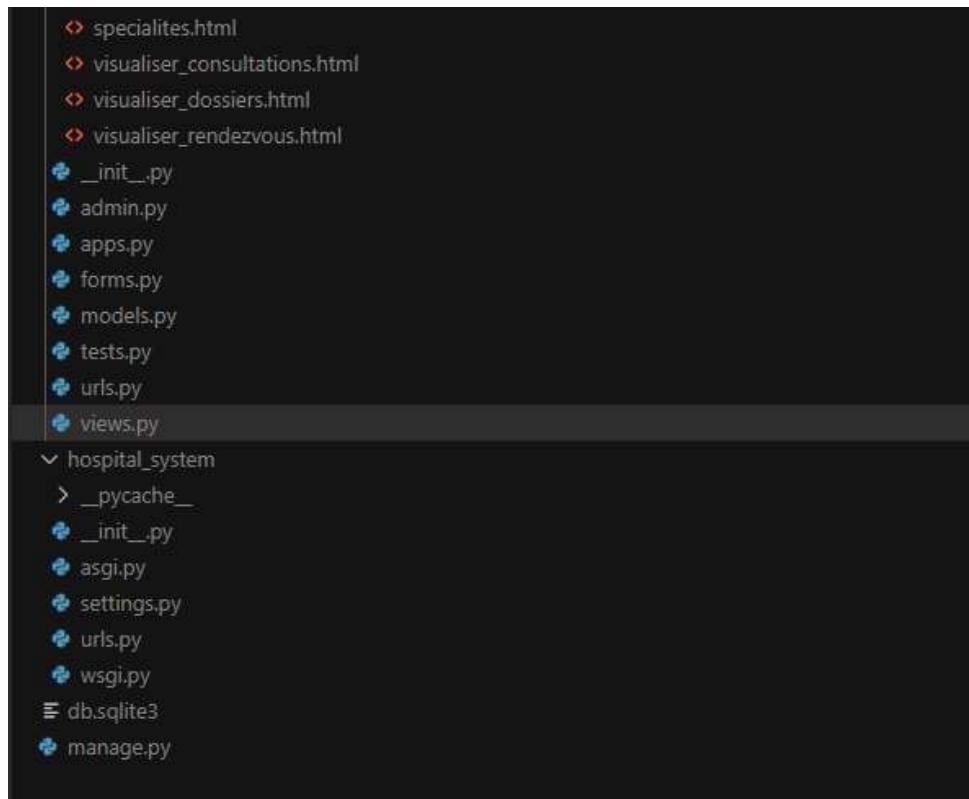


FIGURE 4.2 – Structure des dossiers et fichiers principaux du projet HOSPITAL_SYSTEM.

4.3 Développement de l'application

4.3.1 Interface utilisateur

L'interface utilisateur de l'application HOSPITAL_SYSTEM a été construite en utilisant HTML5 pour la structure, CSS3 pour la mise en forme, et le système de templating de Django pour l'intégration dynamique des données.

— Structure des Templates :

- Un ensemble de templates a été créé dans le répertoire `core/templates/core/` pour les différentes pages et fonctionnalités de l'application. Cela inclut :
 - Des pages générales : `home.html` (page d'accueil), `login.html` (connexion), `signup.html` (inscription).
 - Des pages spécifiques aux patients : `specialites.html` (liste des spécialités), `filtre_par_specialite.html` (liste des médecins par spécialité), `prendre_rendez_vous.html` (formulaire de prise de rendez-vous).
 - Une page de tableau de bord : `dashboard.html` (potentiellement pour les médecins ou un tableau de bord générique post-connexion).
 - Des templates pour les opérations CRUD (Create, Read, Update, Delete) pour chaque modèle principal (**Patient**, **Medecin**, **RendezVous**, **Consultation**, **DossierMedical**, **Facture**). Ces templates sont typiquement nom-

- més `_list.html`, `_detail.html`, `_form.html`, et `_confirm_delete.html` et sont utilisés par les Vues Basées sur les Classes (CBV).
- Héritage de templates : Une approche d'héritage de templates a probablement été utilisée (même si non explicitement visible dans la structure de base), avec un fichier `base.html` (ou équivalent) définissant la structure commune des pages (entête, navigation, pied de page). Les autres templates héritent de ce template de base pour assurer une cohérence visuelle et éviter la duplication de code HTML.
 - **Stylisation CSS :**
 - Des fichiers CSS personnalisés (`home.css`, `login.css`, `signup.css`) ont été créés dans `core/static/core/css/` pour appliquer des styles spécifiques à certaines pages ou sections de l'application.
 - L'objectif était de fournir une interface claire et fonctionnelle, bien que le design avancé ne soit pas le focus principal de ce projet axé sur le backend avec Django.
 - **Intégration dynamique avec Django :**
 - Le langage de templating de Django (DTL) a été intensivement utilisé pour :
 - Afficher dynamiquement les données passées par les vues (ex : listes de médecins, détails d'un rendez-vous, informations du patient).
 - Utiliser des structures de contrôle comme les boucles `{% for %}` pour itérer sur des listes d'objets, et les conditions `{% if %}` pour afficher des contenus spécifiques.
 - Gérer les URLs avec la balise `{% url %}` pour créer des liens dynamiques et maintenables.
 - Afficher les formulaires Django (ex : `{{ form.as_p }}`) et leurs erreurs de validation.

L'ensemble a été conçu pour offrir une expérience utilisateur fonctionnelle, permettant aux différents acteurs (patients, médecins) d'interagir avec le système de manière intuitive.

4.3.2 Connexion base de données

La gestion de la base de données dans l'application `HOSPITAL_SYSTEM` est entièrement prise en charge par l'ORM (Object-Relational Mapper) de Django, qui abstrait les interactions directes avec la base de données SQLite.

- **Configuration :**
 - La connexion à la base de données SQLite est configurée dans le fichier `hospital_system/settings.py`. Par défaut, Django est configuré pour utiliser SQLite, et le fichier de base de données (`db.sqlite3`) est créé à la racine du projet. La configuration typique est la suivante :

```
% Pour afficher du code tel quel
# hospital_system/settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3'
    }
}
```
- **Interaction via les Modèles Django :**
 - Toutes les opérations sur la base de données (création de tables, insertion, sélection,

mise à jour, suppression de données) sont effectuées via les modèles Django définis dans `core/models.py`.

- Par exemple, pour créer un nouveau patient, on instancie le modèle **Patient** et on appelle sa méthode `save()`. Pour récupérer tous les médecins, on utilise `Medecin.objects.all()`.
- L'ORM de Django traduit ces opérations Python en requêtes SQL optimisées pour SQLite, rendant le code plus lisible, plus portable (facilitant un éventuel changement de SGBD) et moins sujet aux erreurs de syntaxe SQL.
- **Migrations :**
 - Le système de migration de Django a été utilisé pour gérer l'évolution du schéma de la base de données.
 - Chaque fois que des modifications étaient apportées aux modèles dans `core/models.py` (ajout d'un champ, modification d'un type, suppression d'un modèle, etc.), les commandes suivantes étaient exécutées :
 1. `python manage.py makemigrations core` : Pour créer un nouveau fichier de migration dans `core/migrations/` décrivant les changements.
 2. `python manage.py migrate` : Pour appliquer ces changements au schéma de la base de données SQLite.
 - Cela garantit que la structure de la base de données est toujours synchronisée avec la définition des modèles. Les fichiers de migration (ex : `0001_initial.py`, `0002.Alter_medeci...py`) conservent un historique de ces évolutions.

L'utilisation de l'ORM de Django et de son système de migration a grandement simplifié la gestion de la persistance des données, nous permettant de nous concentrer sur la logique métier de l'application.

4.3.3 Fonctionnalités implémentées

Le développement de l'application `HOSPITAL_SYSTEM` a permis de mettre en œuvre un ensemble de fonctionnalités clés, conformément aux objectifs du projet. Les principales réalisations incluent :

1. **Gestion des Utilisateurs et Authentification :**
 - *Inscription des Patients* : Une interface (`signup.html` et vue `signup`) permet aux nouveaux utilisateurs de s'inscrire en tant que patients.
 - *Connexion des Utilisateurs* : Une page de connexion (`login.html` et vue `login_view`) permet aux patients, médecins et administrateurs de s'authentifier à l'aide du formulaire `AuthenticationForm`.
 - *Redirection basée sur les Rôles* : Après une connexion réussie, les utilisateurs sont redirigés vers des interfaces appropriées :
 - Les patients sont dirigés vers la page de sélection des spécialités (`specialites.html`).
 - Les médecins sont dirigés vers leur tableau de bord (`medecin_dashboard` - URL à confirmer).
 - Les superutilisateurs (administrateurs) sont redirigés vers l'interface d'administration de Django (`/admin/`).
 - *Gestion des Médecins par l'Admin* : L'interface d'administration de Django est utilisée pour créer, modifier et supprimer les comptes médecins, ainsi que pour leur assigner la spécialité.
2. **Prise et Gestion des Rendez-vous :**

- *Affichage des Spécialités et des Médecins* : Les patients peuvent consulter la liste des spécialités médicales disponibles (`specialites_view`) et voir les médecins correspondants à une spécialité choisie.
- *Prise de Rendez-vous en Ligne* : Les patients connectés peuvent sélectionner un médecin et prendre un rendez-vous via un formulaire (`prendre_rendez_vous` et `RendezVousForm`). Le rendez-vous est associé au patient et au médecin.
- *Consultation des Rendez-vous (CRUD)* : Des vues basées sur les classes (`RendezVousList`, `RendezVousDetail`, etc.) permettent de lister, voir en détail, mettre à jour et supprimer des rendez-vous (principalement pour les médecins ou administrateurs via l'interface de l'application ou l'admin Django).

3. Gestion des Consultations et Dossiers Médicaux :

- *Enregistrement des Consultations* : Les médecins peuvent enregistrer les détails d'une consultation (diagnostic, traitement, etc.) via des formulaires et des vues dédiées.
- *Gestion des Dossiers Médicaux* : Les informations relatives à une consultation (groupe sanguin, allergies, maladies chroniques associées à la consultation) sont enregistrées dans le **DossierMedical** lié à cette consultation. Des vues CRUD (`CBV DossierMedicalList`, `DossierMedicalDetail`, etc.) sont disponibles.
- *Consultation des Dossiers* : Les médecins (et potentiellement les patients pour leurs propres données, selon les droits d'accès implémentés) peuvent consulter les informations des dossiers médicaux.

4. Gestion de la Facturation :

- *Génération et Consultation des Factures* : Des fonctionnalités (`CBV FactureCreate`, `FactureList`, etc.) permettent de créer et de consulter les factures associées aux consultations et aux patients.

5. Interfaces Spécifiques :

- *Interface Patient* : Flux de prise de rendez-vous, consultation de ses informations (dossier médical, factures - si implanté pour le patient).
- *Interface Médecin* : Tableau de bord, gestion de ses rendez-vous, gestion des consultations et des dossiers médicaux de ses patients.
- *Interface Administrateur* : Principalement l'interface d'administration de Django pour la gestion des utilisateurs (médecins) et la supervision des données de l'application.

Ces fonctionnalités ont été développées en s'appuyant sur les modèles, les vues (FBV et CBV), les formulaires et les templates de l'application `core`.

4.3.4 Gestion des exceptions et validations

La robustesse et la fiabilité de l'application `HOSPITAL_SYSTEM` reposent en partie sur une gestion adéquate des exceptions et une validation rigoureuse des données.

— Validation des Données avec les Formulaires Django :

- L'une des principales méthodes de validation des données a été l'utilisation des formulaires Django (définis dans `core/forms.py`), que ce soit des `Form` personnalisés (comme `UtilisateurCreationForm`) ou des `ModelForm` (pour `RendezVousForm`, `ConsultationForm`, `FactureForm`, `PatientForm`).
- Ces formulaires effectuent plusieurs niveaux de validation :
 - Validation des types de champs : S'assurer que les données correspondent au type attendu (ex : un entier pour un âge, une date valide pour `DateField`).

- Contraintes des modèles : Les contraintes définies au niveau des modèles (ex : `max_length` pour un `CharField`, `unique=True` pour `username`) sont automatiquement vérifiées.
- Validations personnalisées : Des méthodes `clean_<nom_du_champ>()` ou `clean()` peuvent être ajoutées aux formulaires pour des logiques de validation plus complexes si nécessaire (bien que non explicitement détaillées ici, c'est une capacité de Django).
- En cas d'échec de validation, les formulaires Django collectent les erreurs et permettent de les réafficher à l'utilisateur à côté des champs concernés dans les templates, facilitant ainsi la correction.
- **Gestion des Exceptions dans les Vues :**
 - *Exceptions HTTP standards* : Django gère nativement certaines exceptions en renvoyant des réponses HTTP appropriées. Par exemple :
 - `Http404` : Le raccourci `get_object_or_404()` a été utilisé dans des vues comme `fiche_medecin` ou `prendre_rendez_vous` (et implicitement dans les `DetailView`, `UpdateView`, `DeleteView` des CBV) pour lever une exception 404 si un objet demandé n'existe pas en base de données.
 - *Blocs try-except* : Pour des opérations potentiellement sources d'erreurs non gérées par défaut par Django (ex : interactions complexes avec des services externes, ce qui n'est pas le cas ici, ou des logiques métier spécifiques), des blocs `try-except` pourraient être utilisés dans les vues pour intercepter des exceptions spécifiques et y réagir de manière appropriée (ex : logger l'erreur, afficher un message convivial à l'utilisateur). Dans le cadre de ce projet, la gestion des erreurs s'est principalement appuyée sur les mécanismes de Django (formulaires, exceptions HTTP).
 - *Exemple dans login_view* : dans la vue `login_view` est une forme de gestion d'exception pour déterminer si l'utilisateur connecté est un patient avant de tenter d'autres vérifications de rôle.
- **Protection par Décorateurs :**
 - Le décorateur `@login_required` est utilisé sur les vues qui ne doivent être accessibles qu'aux utilisateurs authentifiés. Si un utilisateur non authentifié tente d'y accéder, Django le redirige automatiquement vers la page de connexion définie dans `settings.LOGIN_URL`.
- **Messages Utilisateur (Django Messages Framework) :**
 - Bien que non explicitement montré dans les extraits de code fournis, le framework de messages de Django (`django.contrib.messages`) est un outil courant pour afficher des notifications non persistantes à l'utilisateur (ex : "Inscription réussie", "Rendez-vous confirmé", "Erreur lors de la sauvegarde"). Son utilisation aurait pu être envisagée pour améliorer le retour d'information à l'utilisateur.

La combinaison de la validation des formulaires, de la gestion des exceptions HTTP de Django, et de l'utilisation de décorateurs contribue à la stabilité et à la sécurité de base de l'application.

4.4 Tests et validation

Les tests fonctionnels et la validation de l'application ont été effectués en interagissant avec les différentes interfaces pour s'assurer du bon fonctionnement des fonctionnalités implémentées. Les captures d'écran suivantes illustrent quelques-unes des étapes clés et des interfaces de l'application `HOSPITAL_SYSTEM`.

La connection :

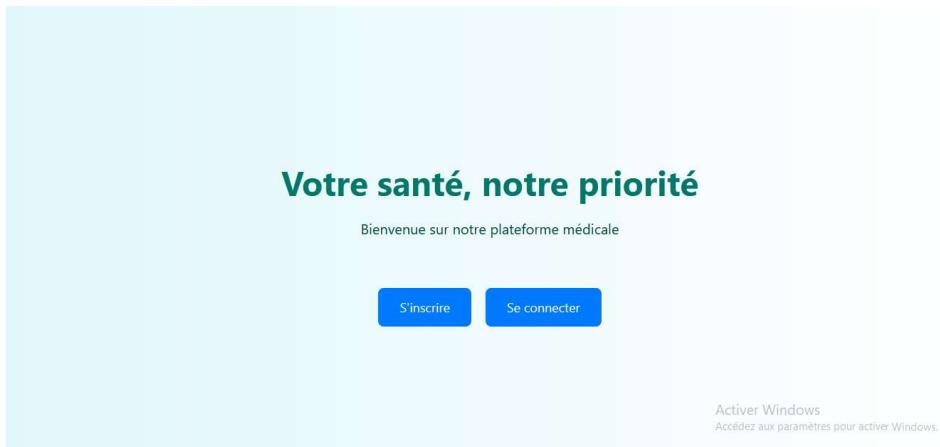


FIGURE 4.3 – Page d'accueil de l'application HOSPITAL_SYSTEM.

Créer un compte patient

Nom d'utilisateur *

Prénom

Nom

Adresse électronique

Téléphone *

Adresse *

FIGURE 4.4 – Début du formulaire d'inscription pour un nouveau patient.

The screenshot shows a vertical list of five input fields for personal information:

- Cin ***: An empty text input field.
- Ville ***: An empty text input field.
- Genre ***: A dropdown menu currently showing "-----".
- Mot de passe ***: An empty text input field.
- Confirmation du mot de passe ***: An empty text input field.

FIGURE 4.5 – Suite du formulaire d’inscription patient (informations personnelles).

The screenshot shows the final section of the registration form:

- Raison de visite ***: An empty text input field.
- A-t-il déjà visité l’hôpital ?**: A question followed by an empty checkbox input field.
- S’inscrire**: A large green rectangular button with white text.
- Déjà un compte ? Se connecter**: Text at the bottom of the button.

FIGURE 4.6 – Fin du formulaire d’inscription patient (informations de connexion et visite).

Interface d’administration :

Connexion

Nom d'utilisateur :

aya123

Mot de passe :

Se connecter

Pas encore inscrit ? [Créer un compte](#)

FIGURE 4.7 – Page de connexion des admins.

Chapitre 4. Réalisation

FIGURE 4.8 – Tableau de bord principal de l’interface d’administration Django pour l’application `core`.

FIGURE 4.9 – Liste des consultations dans l’interface d’administration.

FIGURE 4.10 – Gestion des médecins via l’interface d’administration.

Chapitre 4. Réalisation

The screenshot shows a dark-themed administrative interface. On the left, there is a sidebar with a search bar and a navigation menu under 'AUTHENTIFICATION ET AUTORISATION' and 'CORE'. The 'CORE' section includes categories like 'Groupes', 'Administrateurs', 'Consultations', 'Dossiers médicaux', 'Factures', 'Médecins', 'Patients', and 'Rendez-vous', each with an 'Ajouter' button. The main area is titled 'Ajout de facture' and contains fields for 'Patient' (with a dropdown and edit icon), 'Consultation' (dropdown), 'Date fact.' (calendar icon showing 'Aujourd'hui'), and 'Montant tot.' (text input set to '300,0'). At the bottom are three buttons: 'ENREGISTRER', 'Enregistrer et ajouter un nouveau', and 'Enregistrer et continuer les modifications'.

FIGURE 4.11 – Formulaire d’ajout d’une facture dans l’interface d’administration.

Interface des Médecin :

The screenshot shows a login form titled "Connexion". It has two input fields: "Nom d'utilisateur:" containing "ikram_RB" and "Mot de passe:" containing several dots. Below the inputs is a green button labeled "Se connecter". At the bottom, it says "Pas encore inscrit ? [Créer un compte](#)".

FIGURE 4.12 – Page de connexion utilisée par le médecin.



FIGURE 4.13 – Tableau de bord du médecin après connexion.

The screenshot shows a web-based application titled "Créer un dossier médical". At the top, there is a dropdown menu labeled "Sélectionner un patient:" containing the name "iman-salim". Below this, a red error message reads "Veuillez sélectionner un patient pour créer un dossier.". A large green button labeled "Créer le dossier" is centered. At the bottom left, there is a link "← Retour à l'accueil".

FIGURE 4.14 – Interface médecin : Sélection du patient pour la création d'un dossier médical.

This screenshot shows the continuation of the medical record creation form. It includes a dropdown menu for selecting a patient ("Sélectionner un patient:" showing "iman-salim"), a note that the field is mandatory ("• Ce champ est obligatoire."), and a section for "Consultations:" which lists a single appointment: "Consultation - RDV 2025-05-14 à 12:00:00 - iman-salim avec ikram_RB". Another note indicates this field is mandatory ("• Ce champ est obligatoire."). There is also a section for "Groupe sanguin:" with a dropdown menu showing "Ac" and "Aci", and a note that it is mandatory ("• Ce champ est obligatoire.").

FIGURE 4.15 – Début du formulaire de création d'un dossier médical par le médecin.

Alergie :

• Ce champ est obligatoire.

Maladie chronique :

Ac
Acc

Créer le dossier

FIGURE 4.16 – Suite du formulaire de création d'un dossier médical (allergie, maladie chronique).

Mes Rendez-vous		
Patient	Date	Heure
youssef El	31 mai 2025	08:00
youssef El	30 mai 2025	08:00
youssef El	30 mai 2025	10:00
iman salim	14 mai 2025	12:00

FIGURE 4.17 – Interface médecin : Vue des rendez-vous programmés.

Mon Dossier Médical		
Groupe Sanguin	Allergie	Maladie Chronique
A+	aucune	yfashgvnxz
B+	giukbj	yghjnm
A+	iohefwlnk	ieohlknzd
B+	yjhv	ugkjbm

FIGURE 4.18 – Interface médecin : Consultation d'un dossier médical patient (exemple de données).

Créer une Consultation

Rendezvous :

Diagnostic :

Traitements prescrits :

Activez
Accédez

FIGURE 4.19 – Début du formulaire de création d'une nouvelle consultation par le médecin.

Traitement prescrit :

Duree :
Ex: 1:30:00 pour 1h30

Medicament :

Créer

Activer Windows
Accédez aux paramètres

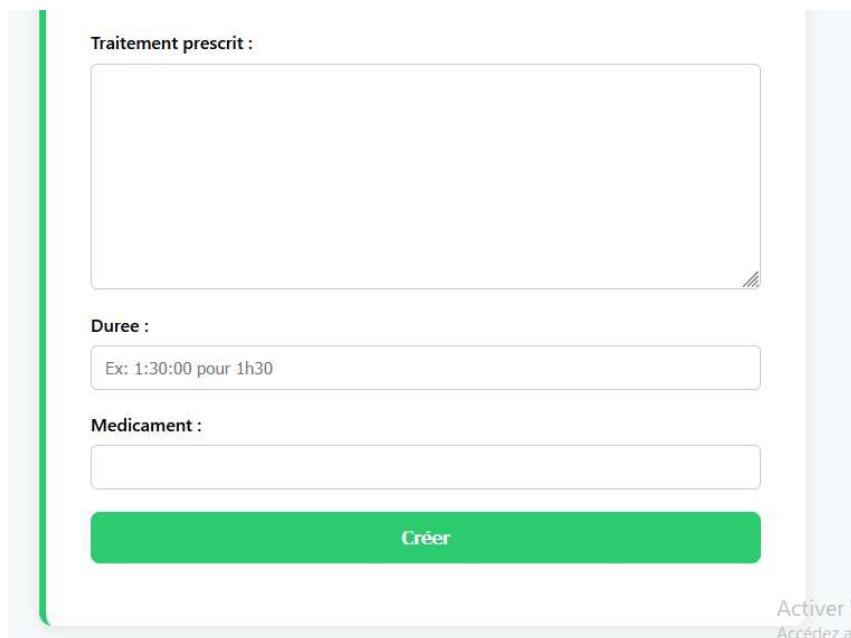


FIGURE 4.20 – Suite du formulaire de création d'une consultation (durée, médicament).

Liste des consultations	
Date : 30 mai 2025	
Patient : youssef El	

Liste des consultations	
Date : 30 mai 2025	
Patient : youssef El	
Diagnostic : ewsd	
Traitemet : wefea	
Médicament : uybjk	
Durée : 0:20:00	

Liste des consultations	
Date : 14 mai 2025	
Patient : iman salim	
Diagnostic : qigudkjab	
Traitemet : g3qjbwDNA	
Médicament : fgvhbj	
Durée : 0:30:00	

Activer Windows
Accédez aux paramètres



FIGURE 4.21 – Interface médecin : Liste des consultations enregistrées.

Interface des patients :

Connexion

Nom d'utilisateur :

YoussefEl

Mot de passe :

••••••••

Se connecter

Pas encore inscrit ? [Créer un compte](#)

FIGURE 4.22 – Page de connexion utilisée par le patient.



FIGURE 4.23 – Tableau de bord du patient après connexion.

Mes Factures		
Date	Consultation	Montant
29 mai 2025	RDV 2025-05-29 à 10:00:00 - YoussefEl avec sara_NB	300,00 DH
30 mai 2025	RDV 2025-05-30 à 08:00:00 - YoussefEl avec ikram_RB	300,00 DH

FIGURE 4.24 – Interface patient : Liste des factures enregistrées.

Mon Dossier Médical		
Groupe Sanguin	Allergie	Maladie Chronique
A+	iohefwkn	ieohlknnsd

FIGURE 4.25 – Interface patient : son dossier medical.



FIGURE 4.26 – Interface patient : choix de specialite pour prendre un rendez vous.



FIGURE 4.27 – Interface patient : prendre un rendez vous avec un medecin choisi.

L'interface patient affiche le titre "Prendre un rendez-vous avec Dr ikram robai". Il y a deux champs à remplir : "Date : jj / mm / aaaa" et "Heure : -- : --". En bas de l'écran, il y a un bouton "Valider le rendez-vous" et un lien "← Retour".

FIGURE 4.28 – Interface patient : remplissage de Date et Heure pour valider son rendez vous.

Conclusion Générale

Ce projet de développement d'une application web de gestion hospitalière a constitué une étape essentielle dans notre parcours académique et professionnel. Il nous a permis de mobiliser et de consolider les compétences acquises en programmation orientée objet, en développement web avec le framework Django, ainsi qu'en conception de bases de données et en modélisation des systèmes d'information.

À travers la mise en œuvre des différentes fonctionnalités telles que la gestion des utilisateurs selon leurs rôles (patients, médecins, administrateurs), la prise de rendez-vous, la création et le suivi des consultations médicales, la tenue des dossiers médicaux et la génération automatisée des factures, nous avons pu concevoir une plateforme complète, sécurisée, et adaptée aux besoins d'un établissement de santé.

Ce projet nous a également permis d'acquérir une meilleure compréhension de l'importance de la structuration du code, de la gestion des vues et des routes, ainsi que de l'authentification et de l'autorisation des utilisateurs. L'aspect collaboratif du travail a favorisé l'échange de connaissances, la gestion de projet en équipe, et le développement de notre autonomie dans la résolution de problèmes concrets.

Enfin, cette expérience a renforcé notre motivation à poursuivre dans le domaine du développement d'applications professionnelles, et nous a sensibilisés à l'impact que peuvent avoir les solutions numériques sur la qualité et l'efficacité des services de santé.