Master Technologies de l'Hypermédia

Domaine : Sciences, Technologies, Santé

Mention : Informatique

Parcours : Technologies de l'Hypermédia (THYP)

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

Cahier de charges

Boite de Médicaments

Réalisé par : Issam AISSAOUI IDRISSI

Encadré par : Professeur Imad SALEH

Cours : Théories et conception des hypermédias

Table des matières

Chapitre 1 — Présentation du Sujet du Projet

1.1 Introduction

Dans un contexte où le numérique occupe une place centrale dans l'organisation de la vie quotidienne, la gestion des traitements médicamenteux reste encore largement manuelle et fragmentée. Les patients s'appuient souvent sur des boîtes physiques, des ordonnances papier, ou des rappels informels (post-it, alarmes téléphoniques), ce qui laisse une grande place à l'oubli, à la confusion des doses, ou à la mauvaise interprétation des prescriptions.

Le projet **BoîteMed** s'inscrit dans ce constat et propose de transposer le principe de la boîte à médicaments traditionnelle dans un environnement numérique interactif. Il s'agit d'une application web hypermédia permettant à l'utilisateur de centraliser ses informations médicales, de gérer ses maladies et ses médicaments, et de recevoir des rappels automatisés pour la prise de ses traitements.

Ce travail est réalisé dans le cadre du module « Concepts et Théories de l'Hypermédia » du Master Technologies de l'Hypermédia (THYP). Au-delà de l'aspect applicatif, BoîteMed constitue également un support concret pour mettre en œuvre des notions clés de l'hypermédia : navigation non linéaire, structuration de l'information, interactivité et personnalisation.

1.2 Problématique

La prise de médicaments est souvent soumise à des contraintes strictes : horaires précis, durées limitées, conditions particulières (avant ou après le repas, association avec d'autres traitements, etc.). Dans la pratique, de nombreux patients peinent à respecter ces contraintes, ce qui peut entraîner une perte d'efficacité du traitement, voire des risques pour la santé.

Les difficultés rencontrées sont multiples :

- Oubli de certaines prises au cours de la journée ;
- Difficulté à suivre plusieurs traitements en parallèle ;
- Manque de visibilité globale sur le planning médicamenteux ;
- Absence d'outils simples et centralisés pour organiser les informations médicales.

À partir de ces constats, la problématique à laquelle répond le projet peut être formulée de la manière suivante :

Comment concevoir une application hypermédia simple, interactive et accessible qui accompagne l'utilisateur dans la gestion quotidienne de ses médicaments et réduise les oublis de prise, tout en respectant la confidentialité de ses données médicales ?

1.3 Objectifs du projet

L'objectif général du projet **BoîteMed** est de proposer une solution numérique permettant à un utilisateur de gérer de manière structurée ses traitements médicaux et de recevoir des rappels pour la prise de ses médicaments.

Cet objectif général se décline en plusieurs objectifs spécifiques :

- Centraliser les informations médicales de l'utilisateur à travers un profil contenant ses données personnelles et certaines informations de référence (âge, poids, contact d'urgence, médecin traitant, etc.).
- Permettre la gestion des maladies en offrant la possibilité de déclarer et de décrire les pathologies suivies.
- Assurer la gestion des médicaments et des traitements : nom du médicament, dosage, forme, instructions de prise, durée du traitement, et horaires précis.
- Planifier les prises quotidiennes grâce à un planning visuel qui regroupe les médicaments à prendre selon l'heure de la journée.
- Réduire les oublis en mettant en place un système de rappels et de statut de prise (prise, en attente, oubliée).
- Offrir une expérience hypermédia riche**,** reposant sur une navigation non linéaire (par médicament, par maladie, par horaire) et une interface multimodale (texte, icônes, couleurs, alertes).

1.4 Public cible

L'application BoîteMed est conçue pour répondre aux besoins d'un public large, confronté à la gestion de traitements médicamenteux réguliers ou ponctuels :

- **Les patients suivant un traitement chronique**, nécessitant un suivi rigoureux sur le long terme (maladies cardiovasculaires, diabète, etc.).
- **Les personnes âgées**, souvent soumises à une polymédication et particulièrement exposées au risque d'oubli ou de confusion entre les médicaments.
- **Les parents**, qui doivent gérer les traitements de leurs enfants (antibiotiques, sirops, cures ponctuelles).
- **Les utilisateurs occasionnels**, souhaitant simplement organiser une cure ou une prescription temporaire.

Ce public est également caractérisé par des niveaux variés de familiarité avec le numérique. L'interface de BoîteMed devra donc rester simple, lisible et intuitive, afin de s'adresser aussi bien à des utilisateurs habitués aux applications web qu'à des personnes moins à l'aise avec les outils informatiques.

1.5 Périmètre du projet

Dans le cadre de ce projet académique, le périmètre fonctionnel de BoîteMed est clairement délimité afin de garantir une réalisation complète et cohérente.

Sont inclus dans le périmètre :

- Un module d'authentification (inscription, connexion, déconnexion) ;
- La gestion du profil médical (données personnelles et contacts de référence) ;
- La gestion des maladies (création, modification, suppression) ;
- La gestion des médicaments et des traitements (informations détaillées, association à une maladie, durée et horaires) ;
- La visualisation d'un planning quotidien des prises ;
- La gestion des statuts de prise (prise, en attente, oubliée) et l'accès à un historique ;
- La mise en place d'une application web reposant sur un backend Flask et un frontend React, avec SQLite/SQLAlchemy pour la gestion des données.
- Sont en revanche exclus de cette première version :
- L'utilisation de techniques avancées d'intelligence artificielle (recommandation, prédiction des oublis, etc.) ;
- L'intégration avec des objets connectés (montres, piluliers intelligents, capteurs médicaux) ;

- Le développement d’une application mobile native (Android/iOS) ;
- La synchronisation avec des systèmes externes (dossiers médicaux, pharmacies, etc.).

Ces éléments exclus pourront constituer des pistes d’évolution pour de futurs travaux, mais ne font pas partie du cahier des charges fonctionnelles de la version actuelle.

1.6 Présentation synthétique de la solution proposée

Pour répondre à la problématique identifiée, BoîteMed adopte la forme d’une application web hypermédia reposant sur une architecture client–serveur moderne. L’interface utilisateur, développée avec React, permet de naviguer entre différents écrans (profil, maladies, médicaments, planning, historique) de manière fluide et non linéaire. Le backend, basé sur Flask, expose une API REST qui gère la logique métier et la persistance des données dans une base SQLite via SQLAlchemy.

L’application propose ainsi à l’utilisateur :

- Un espace personnel sécurisé, accessible après authentification ;
- Des formulaires simples pour déclarer ses maladies et ses médicaments ;
- Un planning quotidien qui synthétise les prises à effectuer ;
- Des rappels clairs pour l’informer des médicaments à prendre au bon moment ;
- Un historique des prises permettant de suivre le taux d’adhérence au traitement.

En combinant une architecture technique légère et une approche centrée sur l’utilisateur, BoîteMed se positionne comme une « boîte à médicaments numérique » visant à rendre la gestion des traitements plus fiable, plus lisible et plus autonome.

Chapitre 2 — Analyse et Conception

2.1. Méthodologie adoptée

La réalisation de l'application BoîteMed repose sur une démarche agile, adaptée aux projets académiques nécessitant une progression par itérations courtes. La méthode choisie est une variante simplifiée de la méthodologie Agile Scrum, qui permet de structurer efficacement l'avancement tout en restant flexible.

2.1.1. Principes de la méthode Agile utilisés

- Découpage du projet en sprints de courte durée (1 semaine).
- Backlog initial comprenant l'ensemble des fonctionnalités attendues (authentification, profil, maladies, médicaments, planning, historique…).
- Priorisation des fonctionnalités selon leur importance et leur dépendance.
- Revue régulière du code et des interfaces pour ajuster et affiner l'ergonomie.
- Approche incrémentale : chaque sprint ajoute un module fonctionnel à l'application.

Cette méthode permet de maintenir une qualité continue du projet, de valider les choix techniques au fur et à mesure, et de simplifier la gestion des imprévus ou des ajustements.

2.2. Analyse fonctionnelle

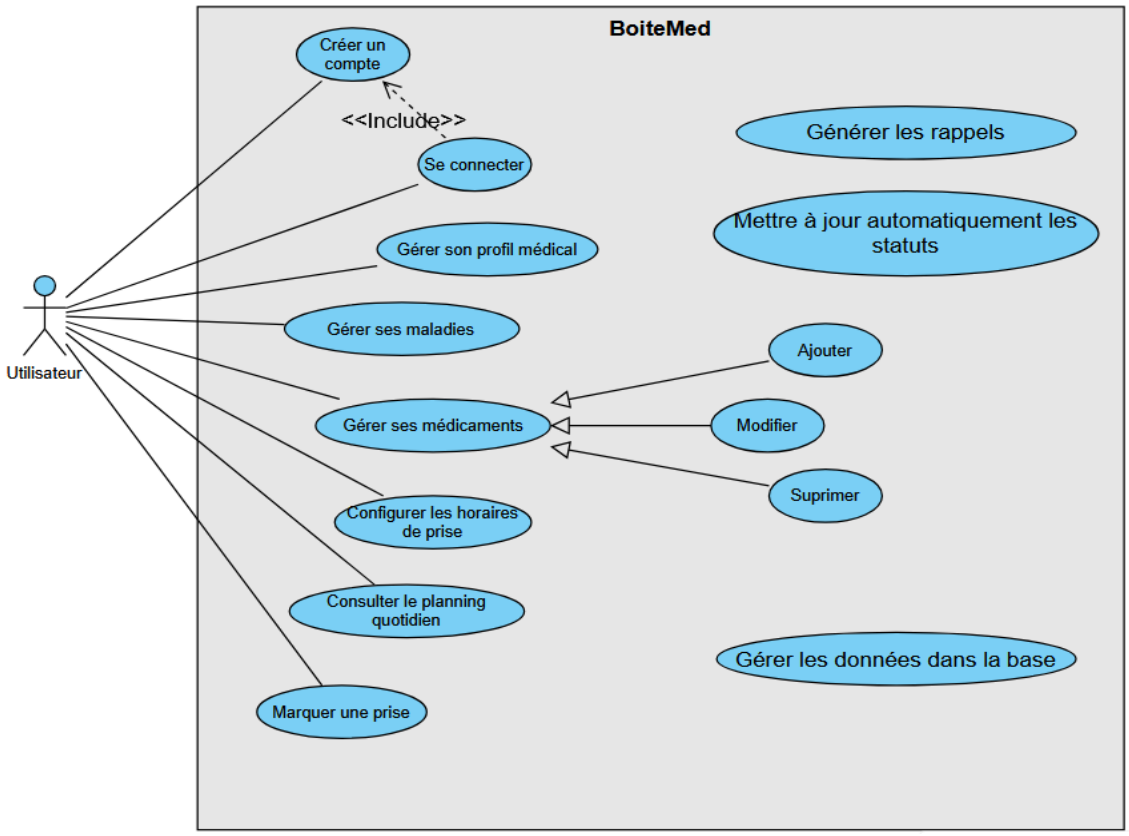
L'analyse fonctionnelle consiste à identifier les acteurs de l'application, les besoins qu'ils expriment, les interactions possibles avec le système et les différents cas d'utilisation. Elle constitue la base de la conception de l'application.

2.2.1. Identification des acteurs du système

Pour l'application BoîteMed, deux acteurs principaux ont été identifiés :

Acteur	Rôle	Responsabilités / Actions
Utilisateur	Acteur principal qui interagit avec l'application	- Créer un compte / se connecter - Gérer son profil médical - Ajouter / consulter ses maladies - Gérer ses médicaments - Configurer les horaires de prise - Consulter le planning quotidien - Marquer une prise (prise / oubliée / en attente) - Consulter l'historique des prises
Système (Back-end + BDD)	Exécute les processus automatiques et assure la cohérence des données	- Générer les rappels automatiquement - Mettre à jour les statuts en fonction du temps - Organiser les prises dans le planning quotidien - Gérer les données dans la base (CRUD via SQLAlchemy) - Assurer la synchronisation avec l'interface front-end

2.2.2. Diagramme global des cas d'utilisation (Use Case Diagram)



2.2.3. Description détaillée des cas d'utilisation

Les cas d'utilisation décrivent la manière dont l'utilisateur interagit avec l'application pour accomplir une tâche fonctionnelle.

2.2.3.1. Cas d'utilisation : S'inscrire

L'Inscription est une étape obligatoire pour tout utilisateur avant d'accéder à l'application.

Élément	Description
Nom du cas d'utilisation	S'inscrire
Acteur	Utilisateur
Objectif	Créer un nouveau compte et accéder à l'application
Préconditions	Aucun compte n'existe avec le même email
Scénario nominal	1. L'utilisateur saisit son email et un mot de passe. 2. Le système vérifie la validité des informations. 3. Le système crée le compte et l'enregistre en base. 4. L'utilisateur peut désormais se connecter.
Scénario alternatif	- Email déjà utilisé → affichage d'un message d'erreur. - Mot de passe non conforme → demande de correction.
Postconditions	Le compte est enregistré en base de données et prêt à être utilisé.

Tableau 2 : Description détaillée du cas d'utilisation « S'inscrire »

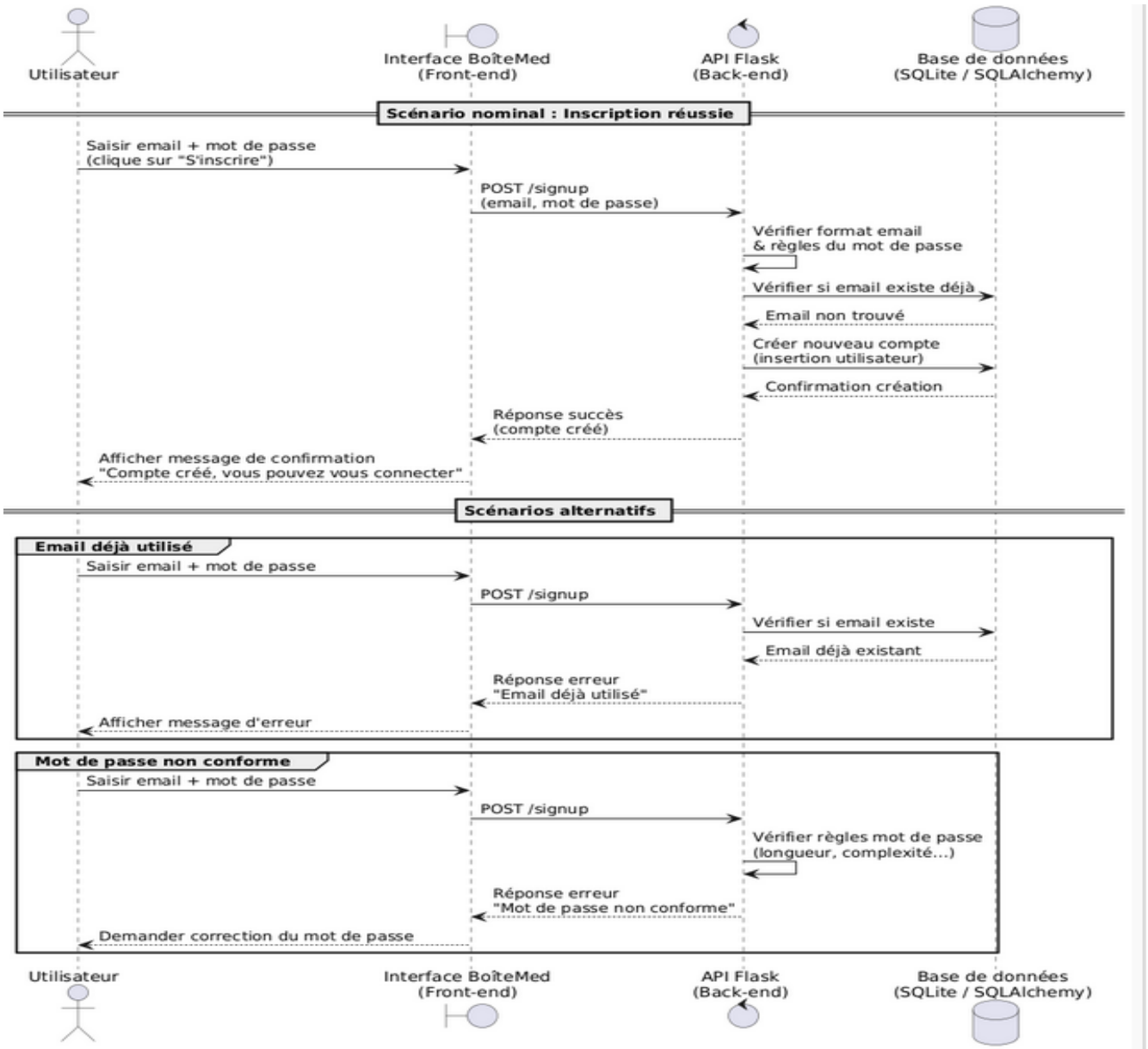


Figure 1 : Diagramme de séquence du cas d'utilisation « Inscription »

2.2.3.2. Cas d'utilisation : Gérer les médicaments

Élément	Description
Nom du cas d'utilisation	Gérer les médicaments
Acteur	Utilisateur
Objectif	Ajouter ou configurer un médicament dans l'application
Scénario nominal	1. L'utilisateur sélectionne la maladie associée (optionnel). 2. Il saisit : nom, forme, dosage, instructions. 3. Il choisit les dates début et fin du traitement. 4. Il définit une ou plusieurs heures de prise. 5. Le système génère automatiquement les prises et les ajoute au planning.
Préconditions	L'utilisateur est authentifié.
Postconditions	Le médicament et ses prises sont correctement enregistrés et visibles dans le planning.
Exceptions / Alternatives	- Champs obligatoires manquants → message d'erreur. - Dates invalides → demande de correction. - Format des heures non conforme → message d'erreur.

Tableau 3 : Description détaillée du cas d'utilisation « Gérer Médicaments »

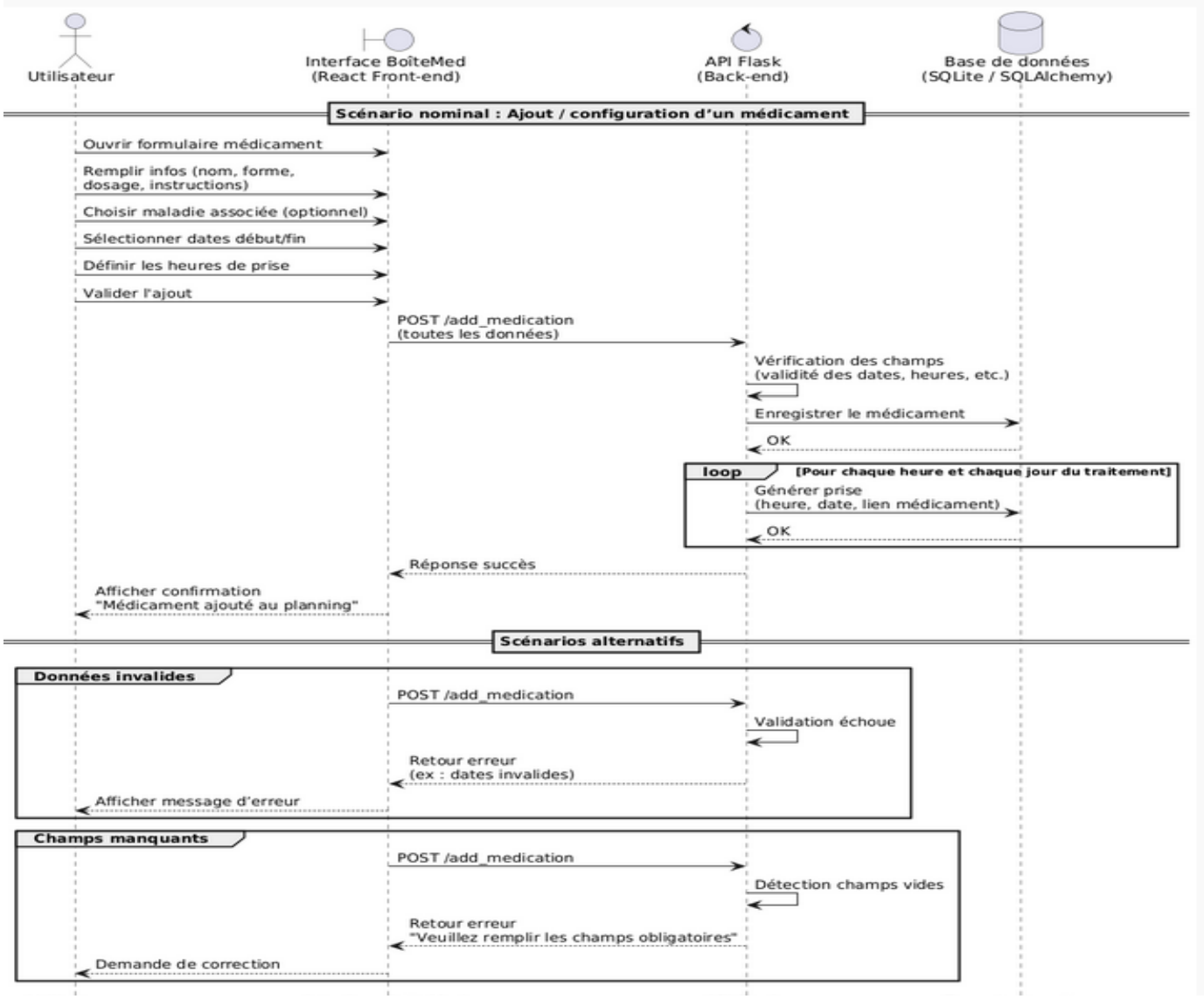


Figure 2 : Diagramme de séquence du cas d'utilisation « Gérer Médicaments »

2.2.3.3. Cas d'utilisation : Consulter le planning du jour

Élément	Description
Nom du cas d'utilisation	Consulter le planning quotidien
Acteur	Utilisateur
Objectif	Visualiser les médicaments à prendre pour la journée en cours.
Préconditions	L'utilisateur est authentifié et au moins un traitement est enregistré avec des prises planifiées.
Scénario nominal	1. L'utilisateur ouvre la page « Planning ». 2. Le système récupère les prises prévues pour la date du jour. 3. Le système affiche une timeline classée par heure. 4. Pour chaque prise, le système affiche : médicament, dose, instructions, statut.
Postconditions	Le planning du jour est affiché, permettant à l'utilisateur de consulter les prises à venir et passées.
Scénarios alternatifs	- Aucun médicament prévu aujourd'hui → afficher un message du type « Aucun médicament prévu pour aujourd'hui ».

Tableau 4 : Description détaillée du cas d'utilisation « Consulter le planning du jour »

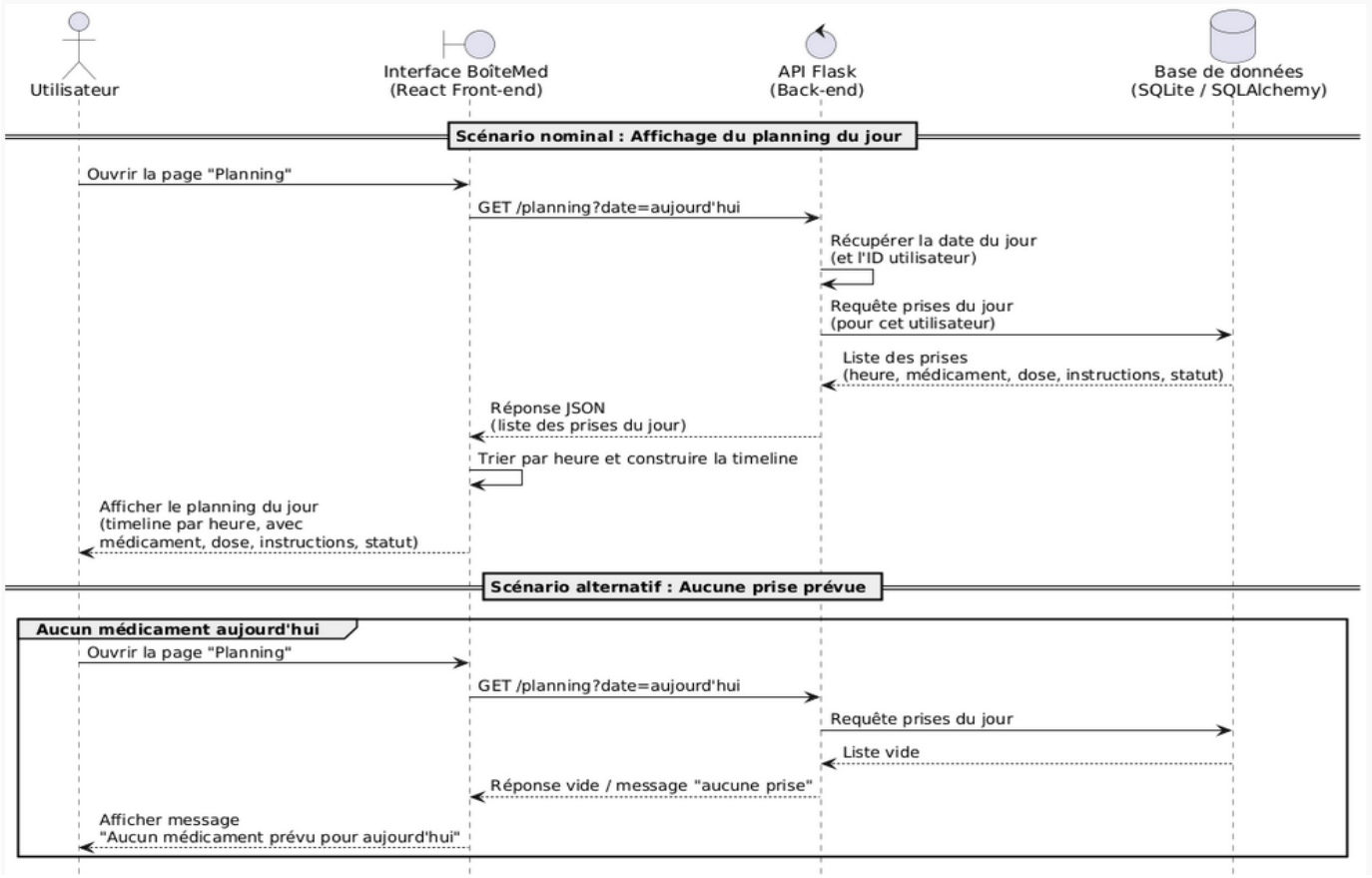


Figure 3 : Diagramme de séquence du cas d'utilisation « Consulter le planning du jour »

2.3. Conception technique

La conception technique permet de définir l'architecture logicielle du projet, les modèles de données, les structures de fichiers et les flux entre le frontend et le backend.

2.3.1. Architecture globale du système

L'architecture globale de l'application BoîteMed repose sur un modèle client–serveur moderne, articulé autour de trois couches principales : le **frontend**, le **backend** et la **base de données**. Le frontend, développé avec **React**, constitue l'interface utilisateur. Il permet d'afficher les différentes pages de l'application (authentification, profil, maladies, médicaments, planning, historique) et d'offrir une navigation fluide grâce à des composants modulaires. Toutes les interactions et actions de l'utilisateur (ajout d'un médicament, modification d'un profil, mise à jour du statut d'une prise, etc.) sont transmises au backend sous forme de requêtes HTTP.

Le backend, développé avec **Flask**, expose une **API REST** qui centralise la logique métier. Il reçoit les demandes émises par le frontend, les traite, applique les règles de gestion (vérification des données, création de traitements, génération des prises, mise à jour des statuts…), puis renvoie une réponse standardisée au client. Le backend utilise également SQLAlchemy en tant qu'ORM pour interagir de manière structurée avec la base de données.

La base de données, implémentée en **SQLite**, stocke l'ensemble des informations persistantes : données des utilisateurs, maladies, médicaments, horaires de prise et historique. SQLAlchemy permet une gestion fiable des entités et de leurs relations tout en assurant une portabilité simple.

Enfin, la communication entre le frontend React et le backend Flask s'effectue via des requêtes HTTP (méthodes GET, POST, PUT, DELETE), ce qui garantit une séparation claire des responsabilités et facilite l'évolution ou la refonte de l'une des couches sans impacter les autres. Cette architecture modulaire et légère assure à BoîteMed une robustesse, une simplicité de maintenance et une évolutivité adaptées à un projet académique structuré.

2.3.2. Architecture Backend (Flask)

Le backend est organisé selon une structure modulaire, basée sur des routes regroupées par domaine :

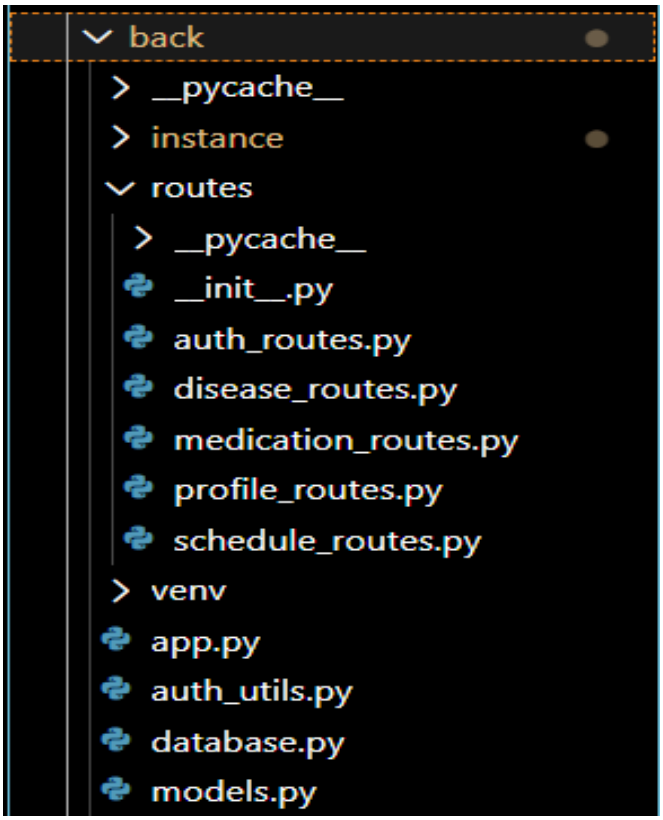


Figure 5 : structure Backend

Rôles des fichiers principaux

- **app.py :** point d'entrée, configuration Flask et CORS
- **database.py :** configuration SQLAlchemy + SQLite
- **models.py :** définition des modèles ORM
- **routes/*** : endpoints REST de chaque module

2.3.3. Architecture Frontend (React)

Le frontend de l'application BoîteMed est développé avec le framework React, choisi pour sa modularité et sa capacité à gérer efficacement les interfaces dynamiques. L'architecture adopte une organisation claire basée sur des pages, des composants réutilisables et une couche dédiée aux appels API. Cette structuration permet d'assurer une séparation logique des responsabilités : les pages gèrent l'affichage global, les composants assurent la présentation et la logique locale, tandis que le module API centralise toutes les communications avec le backend Flask. Cette architecture garantit une application maintenable, évolutive et facile à tester. La figure suivante présente l'organisation générale du frontend.

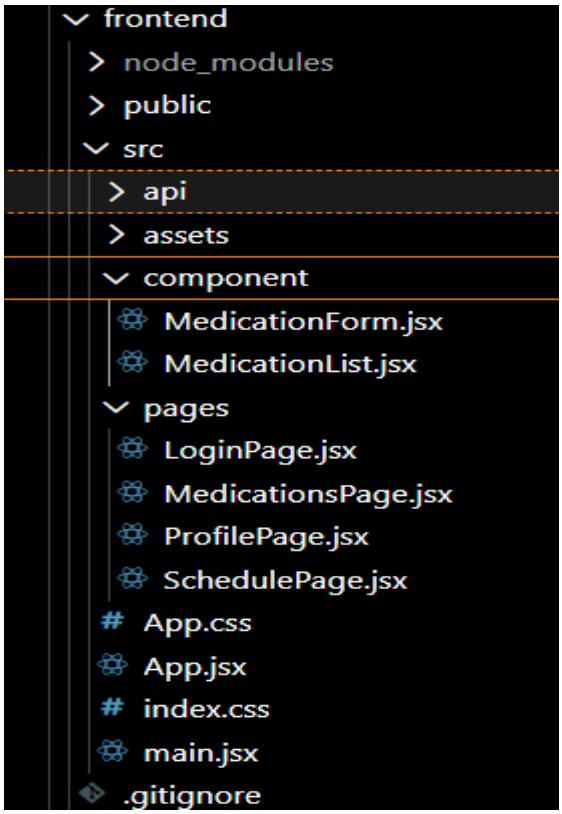


Figure 6 : Structure FrontEnd

Rôles :

- **boiteMedApi.js** : centralise tous les appels API vers Flask
- **Pages/** : écrans principaux
- **Component/** : composants réutilisables
- **App.jsx** : configuration des routes React

2.3.4. Modèle de données (MCD)

Le Modèle Conceptuel de Données (MCD) présente les entités principales du système BoîteMed ainsi que les relations entre elles.
Il constitue la base du futur Modèle Logique de Données (MLD) et décrit la structure informationnelle indépendamment des choix techniques.

Entités principales

Entité	Description
User	Représente un utilisateur de l'application.
Disease	Correspond à une maladie déclarée par un utilisateur.
Medication	Représente un médicament associé à un utilisateur, avec ou sans maladie.
Intake (Prise)	Représente une prise planifiée d'un médicament à une date et une heure précises.

Relations principales

Relation	Cardinalité	Description
User – Disease	1 → N	Un utilisateur peut avoir plusieurs maladies.
User – Medication	1 → N	Un utilisateur peut enregistrer plusieurs médicaments.
Disease – Medication	1 → N	Une maladie peut être associée à plusieurs médicaments.
Medication – Intake	1 → N	Un médicament génère plusieurs prises planifiées.

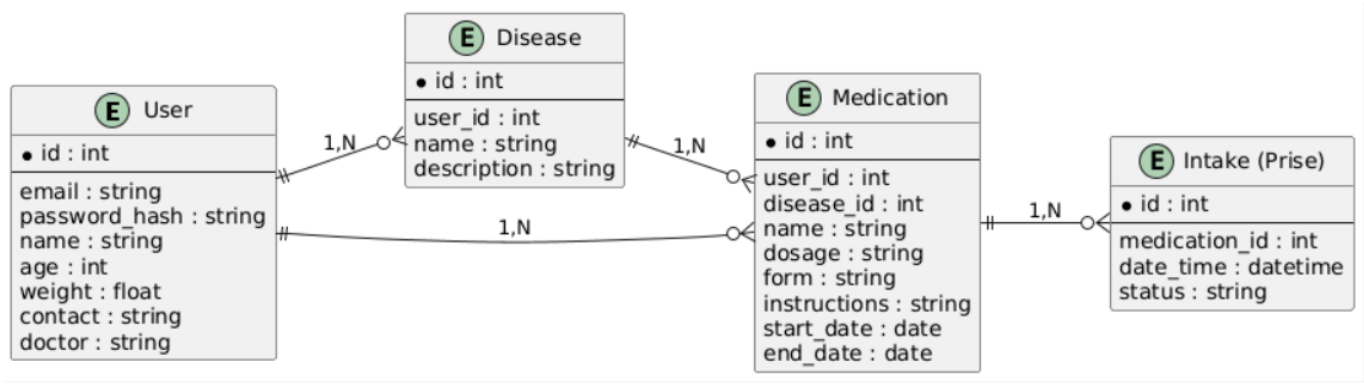


Figure 7 : Modèle MCD

2.3.5. Diagramme de classes

Le diagramme de classes est un schéma fondamental du génie logiciel permettant de représenter les classes, les interfaces ainsi que les relations qui les lient au sein d'un système. Il appartient à la partie statique d'UML, car il ne prend pas en compte les dimensions temporelles ou dynamiques du comportement du système.

Une classe définit les responsabilités, les données (attributs) et le comportement d'un ensemble d'objets similaires. Chaque objet manipulé dans une application est une instance d'une classe. Les classes permettent ainsi de structurer l'information et de modéliser des programmes selon les principes de la programmation orientée objet (POO). Elles offrent un moyen efficace de décomposer un problème complexe en plusieurs entités plus simples et plus cohérentes.

Les classes peuvent être associées entre elles selon différents types de relations, telles que l'héritage, l'association, la composition ou l'agrégation. Chacune de ces relations est représentée par une forme d'arc spécifique dans le diagramme UML, permettant de visualiser clairement les dépendances ou les liens logiques entre les éléments du système.

Graphiquement, une classe est représentée par un rectangle divisé en trois compartiments :

- Nom de la classe
- Attributs (variables internes décrivant son état)
- Méthodes (fonctions représentant son comportement)

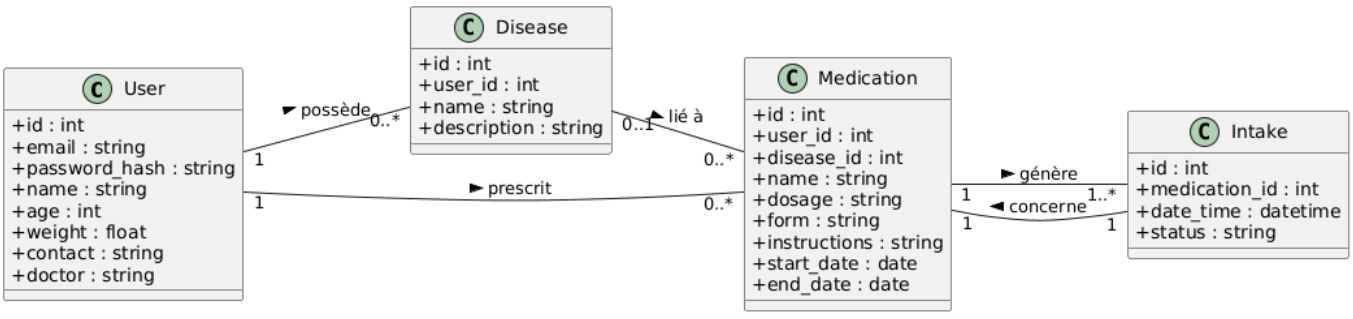


Figure 8 : Diagramme de classe

2.4. Conclusion

Ce chapitre a permis d'établir les fondations conceptuelles et techniques du projet BoîteMed en définissant précisément les besoins fonctionnels de l'application ainsi que les attentes des utilisateurs. L'identification des acteurs et l'analyse des différents cas d'utilisation ont clarifié les interactions essentielles entre l'utilisateur et le système. De plus, la présentation de l'architecture globale, du modèle de données et de l'organisation des couches backend et frontend a permis de poser un cadre technique solide, cohérent et structuré pour la suite du développement. L'ensemble de ces éléments constitue une base méthodologique indispensable pour aborder sereinement la phase de mise en œuvre, qui sera détaillée dans le chapitre suivant.

Chapitre 3 — Mise en œuvre de l'application

3.1. Introduction à l'environnement de développement

La mise en œuvre de l'application BoîteMed repose sur une architecture web moderne et modulaire permettant de séparer clairement la partie interface utilisateur, la logique métier et la gestion des données. Pour garantir flexibilité et maintenabilité, l'application adopte une structure "frontend–backend" distincte : le frontend en React gère l'affichage et l'interaction avec l'utilisateur, tandis que le backend en Flask expose une API REST pour le traitement des requêtes et la gestion des données persistantes via SQLAlchemy et SQLite.

Le développement a été réalisé dans un environnement moderne comprenant :

- Visual Studio Code comme éditeur principal ;
- Git/GitHub pour le versionnement du code ;
- Postman pour tester les endpoints de l'API ;
- Vite pour initialiser rapidement le frontend React ;
- Python venv pour isoler les dépendances backend ;
- Et les bibliothèques essentielles telles que Flask, Flask-CORS, SQLAlchemy, React Router, etc.

Cet environnement permet d'assurer une evolution simple du code, une bonne séparation des responsabilités et une lisibilité optimale du projet.

3.2. Architecture MVC

Bien que l'architecture globale repose sur un découpage frontend/backed, la logique interne du projet suit les principes du modèle MVC (Model–View–Controller)**.**

3.2.1. Composants de l'architecture MVC

- Model (Backend – SQLAlchemy) :
  Il regroupe l'ensemble des entités manipulées par le système (Utilisateur, Maladie, Médicament, Prise). Ces modèles définissent les attributs, les relations et les règles d'intégrité des données stockées en base SQLite.
- View (Frontend – React) :
  Les pages React constituent les vues. Elles présentent à l'utilisateur les informations issues de l'API et permettent l'interaction avec le système via des formulaires et composants réutilisables.
- Controller (Backend – Flask) :
  Les routes déclarées dans Flask agissent comme des contrôleurs : elles reçoivent les requêtes provenant du frontend, exécutent la logique métier, interagissent avec les modèles et renvoient une réponse JSON.

Cette architecture garantit une bonne séparation des responsabilités : chaque couche a un rôle clairement délimité, ce qui facilite la maintenance et l'évolution future de l'application.

3.3. Environnement de travail en modélisation

Pour structurer et documenter la logique interne de l'application, plusieurs outils de modélisation ont été utilisés :

- **Diagrammes UML** (cas d'utilisation, classes, séquence) réalisés via *PlantUML*.
- **Modèle conceptuel des données (MCD)** pour structurer les entités de la base SQLite.
- **Schémas d'architecture** pour illustrer l'organisation globale du système.

Ces représentations visuelles ont permis d'assurer une compréhension globale du fonctionnement de l'application avant le développement.

3.4. Environnement de travail

Le développement du projet a reposé sur un ensemble d'outils cohérents et adaptés :

- **Frontend :**
  - ✓ React
  - ✓ Vite
  - ✓ React Router
  - ✓ Composants personnalisés (MedicationForm, MedicationList…)
- **Backend :**
  - ✓ Python
  - ✓ Flask
  - ✓ Flask-CORS
  - ✓ SQLAlchemy
  - ✓ SQLite
- **Outils complémentaires :**
  - ✓ Postman (tests API)
  - ✓ Git/GitHub (versionnement et collaboration)
  - ✓ VS Code (éditeur principal)

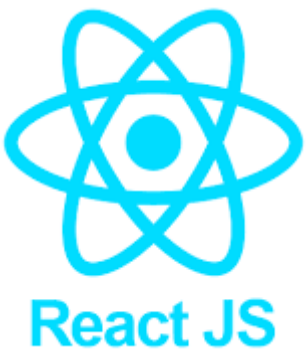
Cet environnement a permis un workflow fluide, avec un aller-retour constant entre les tests backend, la construction des pages frontend et la vérification du fonctionnement global.

3.5. Architectures et langages de programmation utilisés

3.5.1. Technologies utilisées

3.5.1.1. React (Frontend)

React a été choisi pour sa capacité à gérer des interfaces dynamiques et interactives grâce à son système de composants. Chaque fonctionnalité (gestion des médicaments, profil, maladies, planning…) est représentée sous forme de composants indépendants, favorisant la réutilisabilité et la modularité.



3.5.1.2. Flask (Backend)

Flask constitue la base du backend. Grâce à son architecture minimaliste et flexible, il permet de mettre en place des routes REST, de gérer les requêtes HTTP, d'appliquer la logique métier et de communiquer avec la base de données via SQLAlchemy.

3.5.1.3. SQLAlchemy (ORM) & SQLite (Base de données)

SQLAlchemy simplifie la manipulation de la base de données en utilisant des modèles Python au lieu de requêtes SQL brutes.
SQLite est utilisé pour sa légèreté, sa portabilité et sa facilité d'intégration dans un projet académique.

3.5.1.4. Autres outils complémentaires

- Flask-CORS pour autoriser les requêtes venant du frontend React.
- Vite pour optimiser le temps de développement frontend.
- Bibliothèques JavaScript pour la gestion des dates, des formulaires et des appels API.

3.6. Organisation du fonctionnement global

Le fonctionnement général du système suit le flux suivant :

1. **L'utilisateur interagit avec l'interface React.**
   Il consulte son profil, ajoute une maladie, crée un médicament ou modifie une prise.
2. **Le frontend envoie une requête API au backend Flask.**
   Chaque action déclenche un appel HTTP vers l'endpoint correspondant.
3. **Flask traite la requête.**
   Le contrôleur valide les données, applique la logique métier et interagit avec la base via SQLAlchemy.
4. **SQLite stocke ou retourne les données.**
   Les entités sont enregistrées, modifiées ou supprimées.
5. **Flask renvoie une réponse JSON.**
   Le frontend met à jour l'affichage React en temps réel.

Ce cycle assure une application fluide, dynamique et cohérente.

3.7. Conclusion

La mise en œuvre de l'application BoîteMed s'appuie sur une architecture claire et modulaire combinant React pour l'interface utilisateur, Flask pour la logique métier et SQLite pour la gestion des données. La séparation stricte entre frontend et backend rend le projet facilement maintenable et évolutif, tandis que l'utilisation d'outils modernes comme Vite, SQLAlchemy et React Router garantit une bonne qualité de développement.
Ce chapitre montre ainsi comment les choix techniques effectués permettent de répondre efficacement aux besoins fonctionnels définis dans les chapitres précédents.

Conclusion générale

Le développement de l'application BoîteMed s'inscrit dans une démarche visant à répondre à un besoin réel : faciliter et sécuriser la gestion quotidienne des traitements médicamenteux. À travers l'analyse du contexte, la définition des problématiques et l'identification des besoins des utilisateurs, le projet a permis d'élaborer une solution numérique claire, accessible et pensée pour accompagner les patients dans leur autonomie.

Le cahier de charges a structuré ce processus en définissant précisément les objectifs fonctionnels, les attentes du système et l'ensemble des interactions possibles entre l'utilisateur et l'application. La conception détaillée, basée sur une modélisation rigoureuse (UML, MCD, architecture globale), a posé les fondations techniques nécessaires pour aboutir à une implémentation cohérente et évolutive. La mise en œuvre du projet, articulée autour d'une architecture moderne mêlant React**,** Flask et SQLite, démontre la pertinence d'un découpage clair entre présentation, logique métier et gestion des données.

En combinant une interface intuitive, une structure hypermédia flexible et des fonctionnalités essentielles telles que la gestion des maladies, des médicaments, du planning et des rappels, BoîteMed constitue une première version solide d'une boîte à médicaments numérique. Ce travail ouvre également des perspectives d'évolution intéressantes : intégration future de modules de recommandation, notifications avancées, version mobile, synchronisation avec des services externes, ou encore adaptation pour des besoins professionnels.

Ainsi, BoîteMed représente une application à la fois utile, cohérente et techniquement bien structurée, démontrant la capacité à concevoir et déployer une solution numérique complète tout en valorisant les concepts fondamentaux de l'hypermédia.