Documentation technique

Réflexions Initiales Technologiques

1. Introduction

Le projet SoigneMoi nécessite le développement d'une suite d'applications pour différentes plateformes (Web, Mobile, et Desktop). Chaque application est développée en utilisant des technologies adaptées à son environnement cible. Le choix des technologies a été fait en tenant compte des besoins et des contraintes de chaque application.

2. Technologies Utilisées

2.1. Site Web

- **TypeScript** : J'utilise Typescript en raison de son typage qui facilite le développement.
- **Next.js** : Next.js a été choisi en tant que framework web car il s'integre parfaitement avec React, en plus d'offrir beaucoup de fonctionalités qui facilite le développement tels que les server actions permettant d'executer des fonctions coté serveur, le SSR permetant d'accelerer le chargement des pages en generant ce qui est statique en amont, ainsi que les API routes qui seront utilisé dans les applications mobile et bureautique.
- **Tailwind CSS** : Tailwind CSS est utilisé pour son approche utilitaire, qui permet de donner le style voulu au html en reduisant la taille du code ainsi que garder un design system unifié.
- **Shadcn** : Shadcn est utilisé pour la gestion des composants réutilisables, ce qui permet une grande cohérence dans le design de l'application ainsi qu'acceler le developpement en offrant des composant préfait, épurée et accessibles.
- **Prisma** : Prisma est un ORM qui simplifie les interactions avec la base de données PostgreSQL. Il permet une construction rapide des modèles de données, et une compatibilité avec TypeScript pour une meilleure productivité.

- **PostgreSQL** : PostgreSQL a été choisi comme base de données relationnelle pour sa robustesse, sa fiabilité, et ses fonctionnalités avancées comme la gestion des transactions et les performances en lecture/écriture.

2.2. Application Mobile

- **TypeScript** : TypeScript est également utilisé pour l'application mobile, assurant ainsi la cohérence entre le code du site web et de l'application mobile, tout en bénéficiant du typage statique.
- **React Native** : React Native permet de développer une application mobile multiplateforme (iOS et Android) avec une base de code unique, ce qui réduit le temps de développement.
- **Expo** : Expo simplifie la gestion des builds et des déploiements sur les appareils réels, accélérant ainsi le développement de l'application mobile. Il offre aussi un large éventail de composants et d'APIs pour développer rapidement des fonctionnalités mobiles.

2.3. Application Bureautique

- **Python** : Python a été choisi pour sa simplicité et sa puissance, particulièrement adapté pour le développement rapide d'une application bureautique.
- **Custom Tkinter** : Tkinter, avec des personnalisations, est utilisé pour créer des interfaces graphiques natives. Il est léger et directement intégré avec Python.

3. Architecture API

Les applications mobile et bureautique utilisent l'API du site web pour effectuer la plupart de leurs opérations. L'API est construite en suivant les principes REST, ce qui facilite la communication entre les différentes applications.

- **API REST** : Développement d'une API RESTful avec Next.js pour gérer les opérations CRUD (Create, Read, Update, Delete) sur les ressources nécessaires aux différentes applications.
- **Authentification** : Implémentation d'un système d'authentification sécurisé (JWT) pour garantir que seules les applications et les utilisateurs autorisés accèdent aux ressources de l'API.

4. Justification des Choix Technologiques

- **Consistance et Cohérence** : L'utilisation de TypeScript sur les différentes plateformes (Web et Mobile) permet une meilleure consistance du code, facilitant ainsi le partage de logique entre les applications.
- **Performance** : Next.js, associé à Prisma et PostgreSQL, offre une excellente performance en termes de vitesse de réponse et de gestion de grandes quantités de données.
- **Scalabilité** : Les technologies choisies permettent une évolution future du projet, que ce soit en termes de fonctionnalités additionnelles ou d'augmentation du nombre d'utilisateurs.
- **Maintenabilité** : Grâce à TypeScript et aux pratiques de développement modernes (comme l'utilisation d'ORM et de composants réutilisables), le code est plus facile à maintenir et à faire évoluer.

5. Conclusion

Le choix des technologies pour ce projet a été guidé par le besoin de créer un ensemble d'applications performantes et sécurisées. Chaque technologie sélectionnée apporte des avantages spécifiques, répondant aux exigences fonctionnelles et techniques du projet SoigneMoi.

Configuration de l’environnement de travail

1. Choix des outils et logiciels :

- **IDE** : J'ai choisi d'utiliser VS Code comme IDE pour ce projet pour sa légèreté et ses extensions qui facilitent le développement.
- **Gestion des versions** : J'ai utilisé Git pour la gestion des versions, avec GitHub comme plateforme de dépôt. Cela permet de suivre les modifications du code et de maintenir un historique clair des versions.
- **Base de données** : J’ai commencé par utiliser SQLite, une base de données en local, pour sa facilité a mettre en place ensuite j'ai changé pour PostgreSQL, une base de données relationnelle robuste et extensible. PostgreSQL est particulièrement adapté aux projets qui nécessitent des opérations complexes et une gestion des transactions fiable.
- **Framework et librairies** :

    - **Web** : Next.js pour le développement de l'application Web, avec Prisma en tant qu'ORM, associé à Tailwind CSS, et shadcn pour le design.
    - **Mobile** : React Native avec Expo pour le développement de l'application mobile.
    - **Bureautique** : Python avec Custom Tkinter pour l'application bureautique.
- **Environnement de test** : Jest pour les tests unitaires et Cypress pour les tests fonctionnels de l'application Web. Pour l'application mobile, j'ai utilisé Detox.

2. Installation et configuration des outils :

- **Node.js et pnpm** : Installation de Node.js pour la gestion des dépendances via pnpm, utilisé pour les applications Web et mobiles.
- **PostgreSQL** : Configuration d'une instance locale de PostgreSQL pour le développement.
- **Extensions VS Code** :
    - ESLint : pour l'analyse statique du code.
    - Prettier : pour le formatage automatique du code.
    - GitLens : pour une meilleure visualisation de l'historique Git.
    - Pylance : Support de language performant
    - Python : Support de language pour python
    - Prisma: Surligneur de syntaxe, auto-completion et linting pour le fichier prisma
    - Autopep8 : formatteur de code python

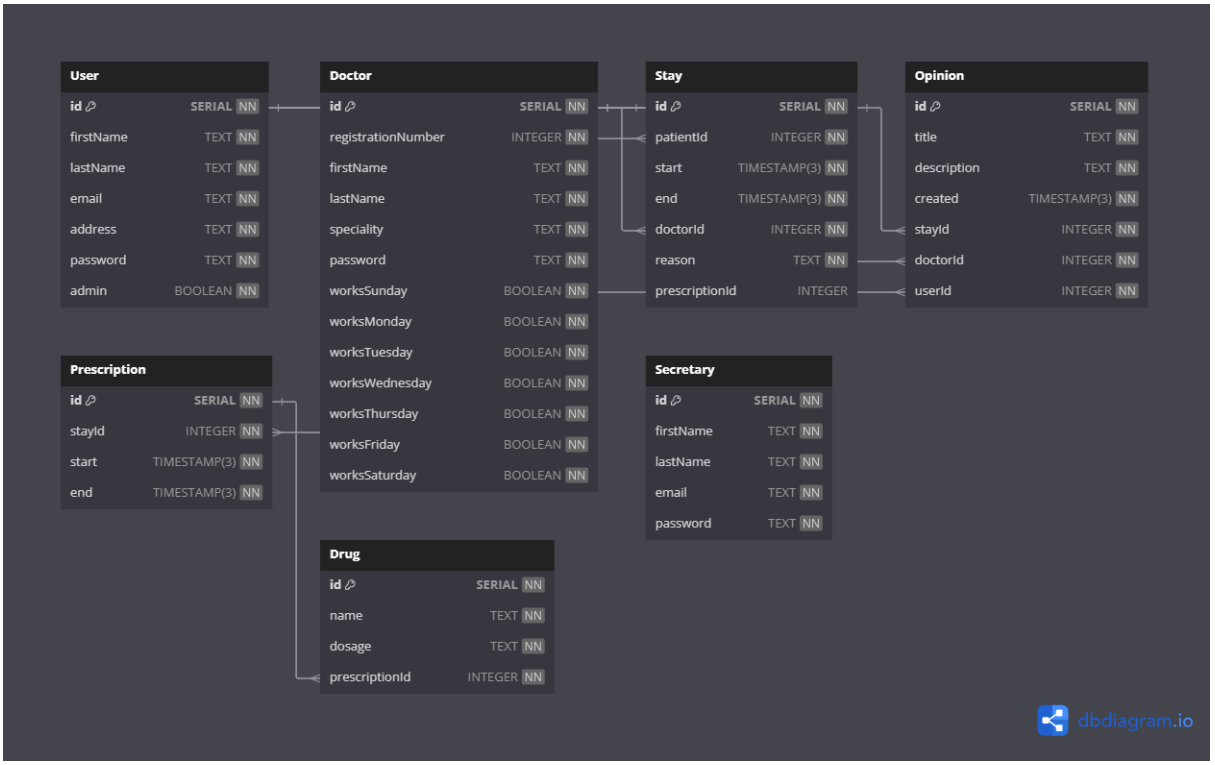
3. Gestion des environnements de développement :

- **Variables d'environnement** : Mise en place des fichiers `.env` ou j'ai stocké les informations sensibles comme les url d'API, le lien vers la base de données, etc. Ces fichiers sont mis dans le `.gitignore`.
- **Scripts d'initialisation** : Mise en place de scripts pour l'utilisation rapide des dépendances et la configuration des environnements (ex. : `pnpm run dev`, `pnpm run build`, `pnpm start`).

4. Sécurisation de l'environnement :

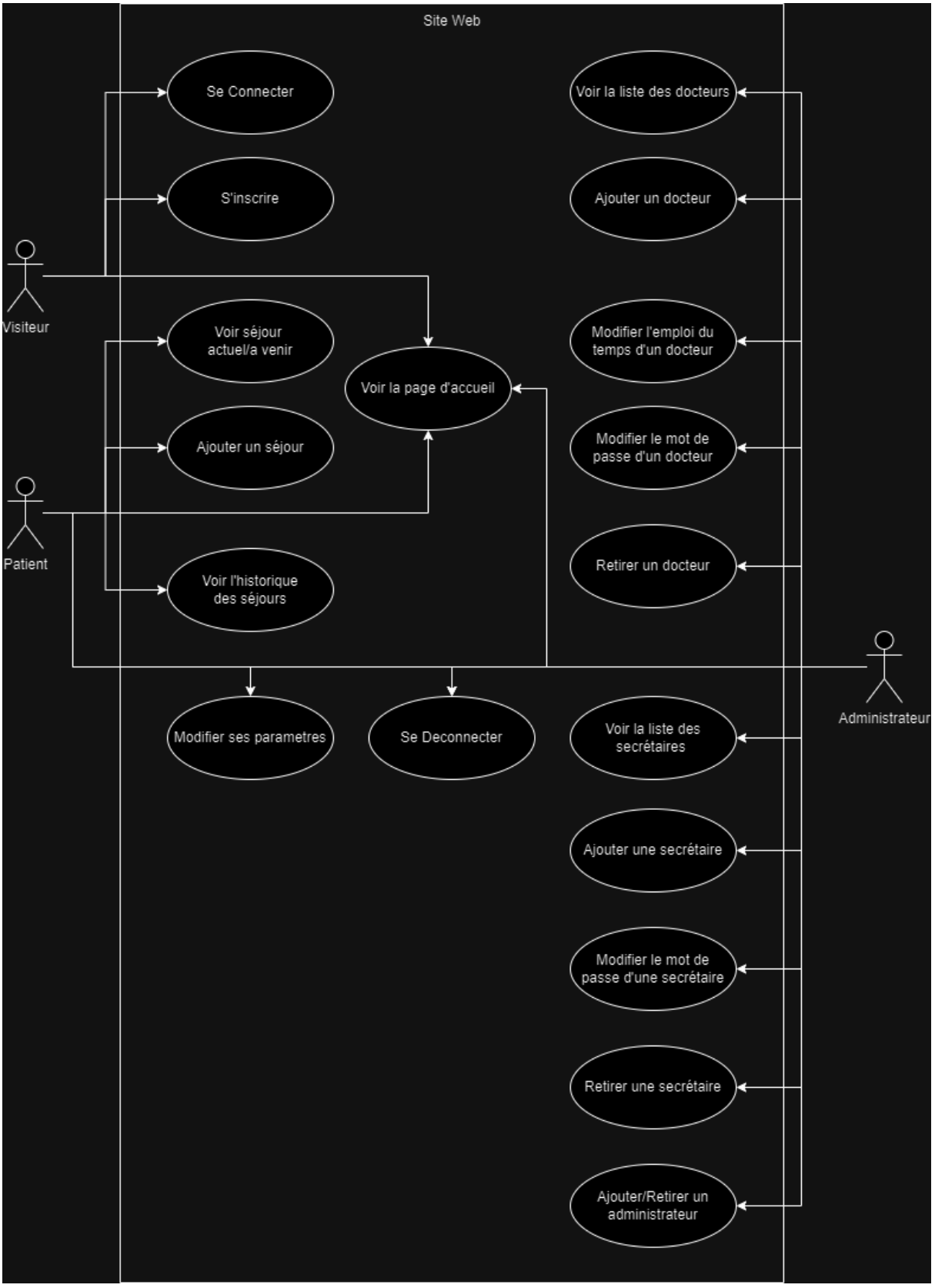
- **Linting et formatage** : j'ai utilisé ESLint et Prettier pour garder un code propre et organisé.

Modèle conceptuel de données



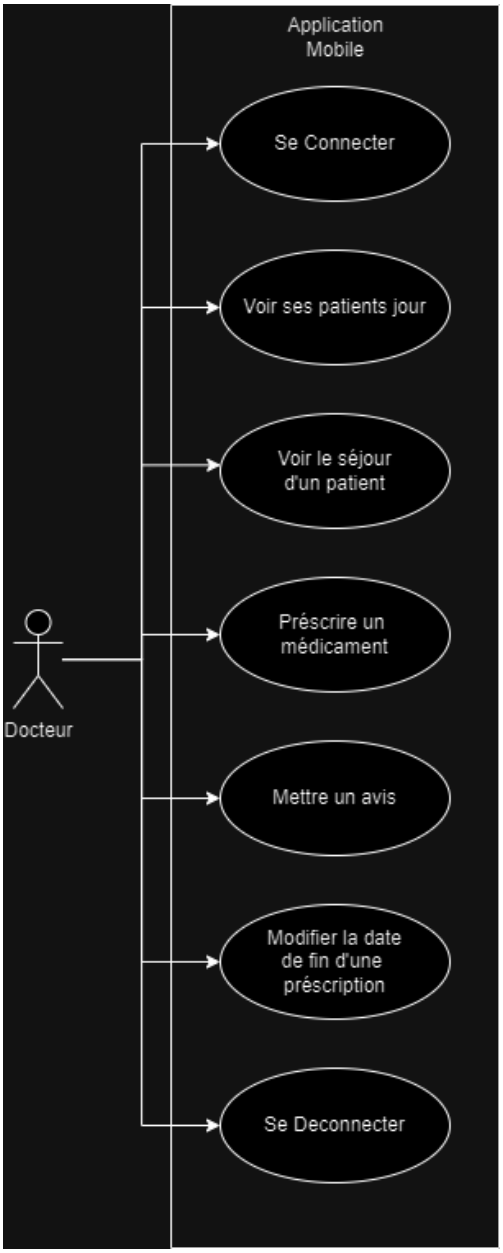
Diagrammes de cas d'utilisation

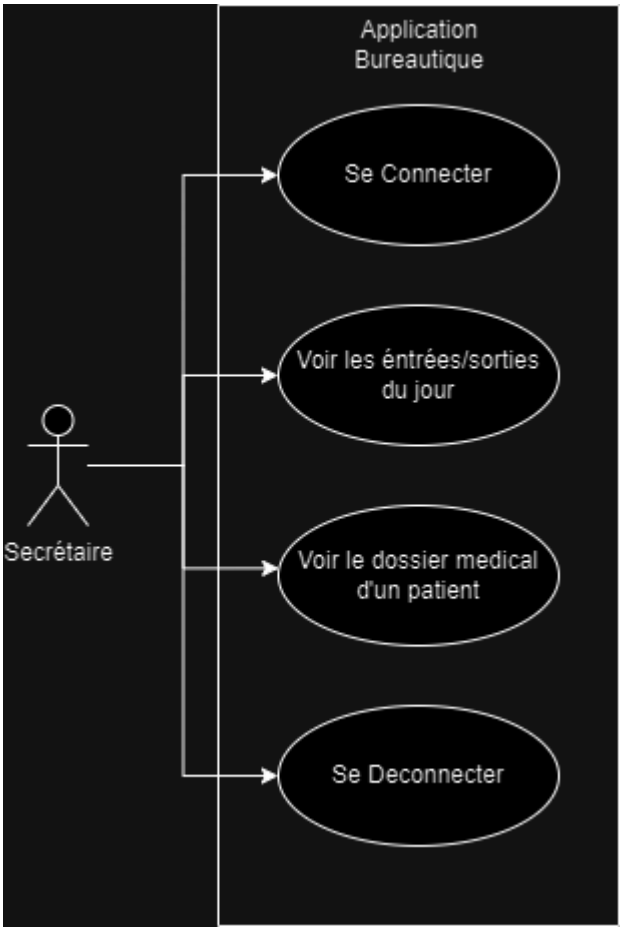
1. Site Web



2. Application Mobile

3. Application Bureautique

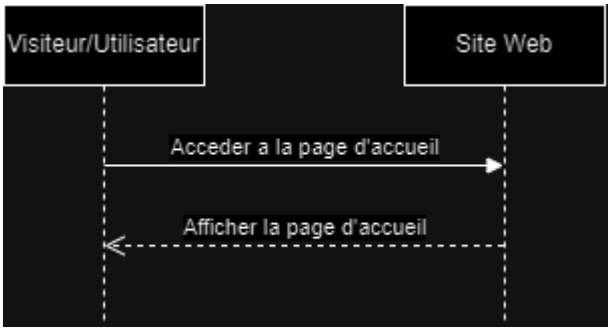




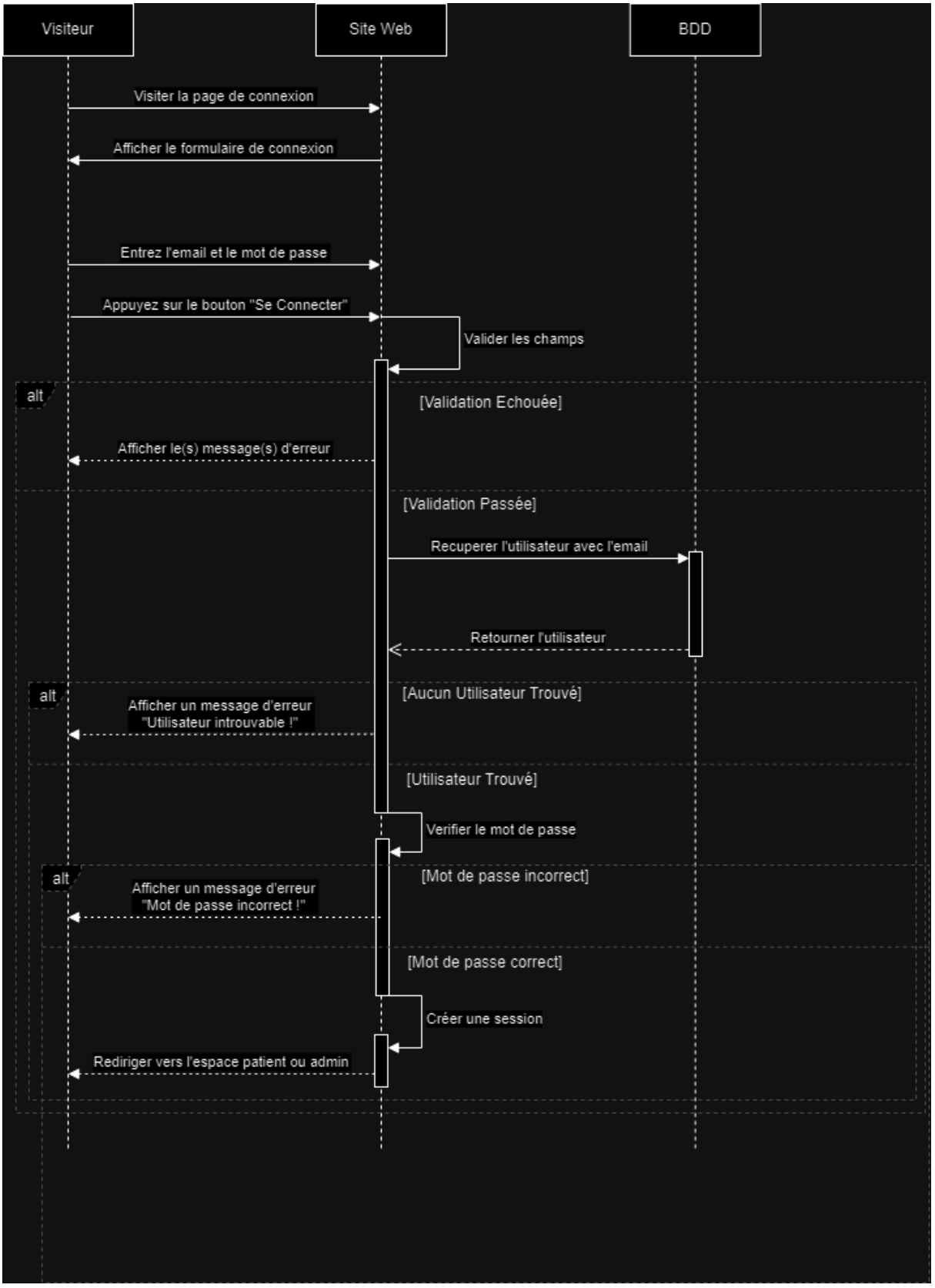
Diagrammes de séquence

1. Site Web

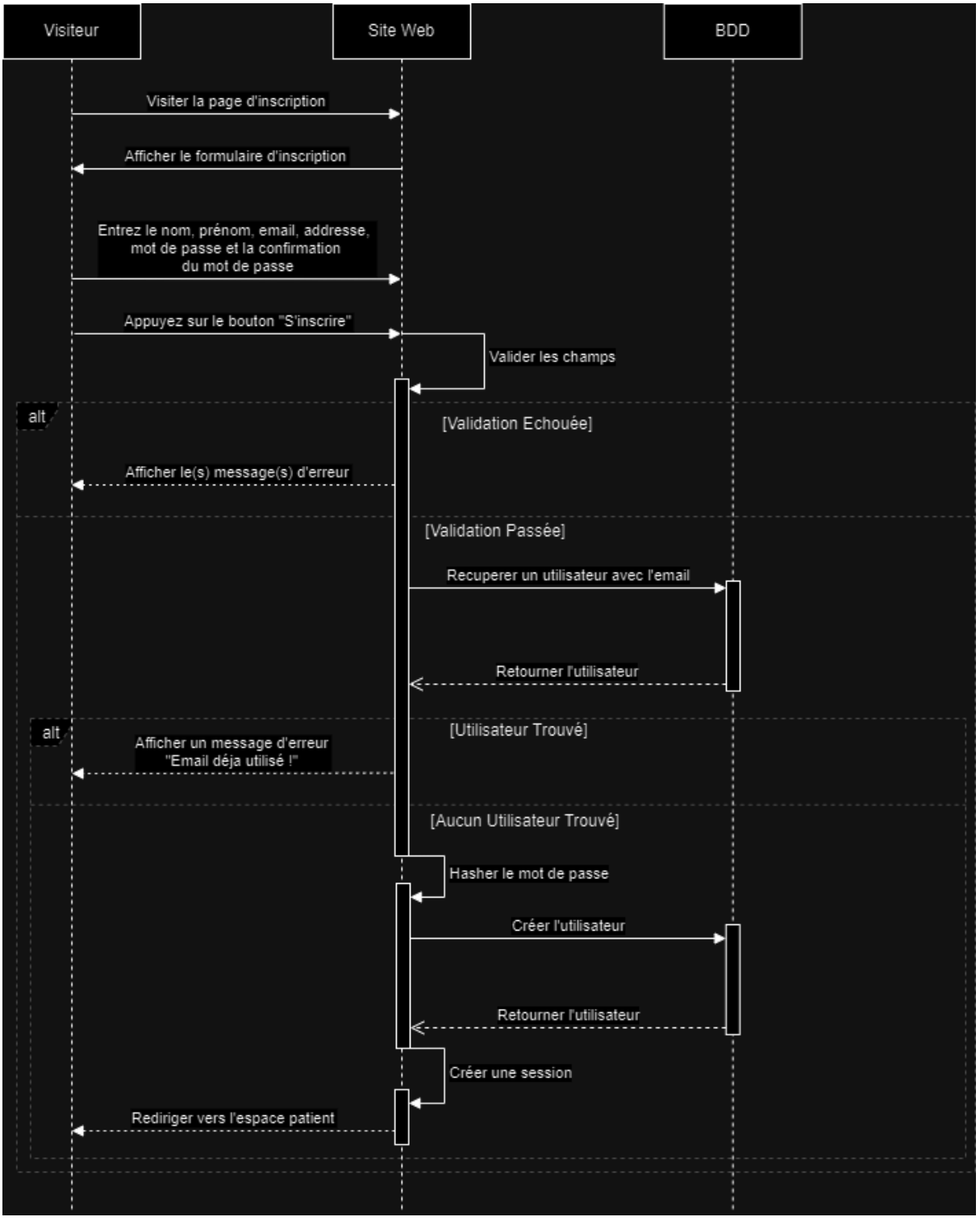
1.1. Voir la page d'accueil



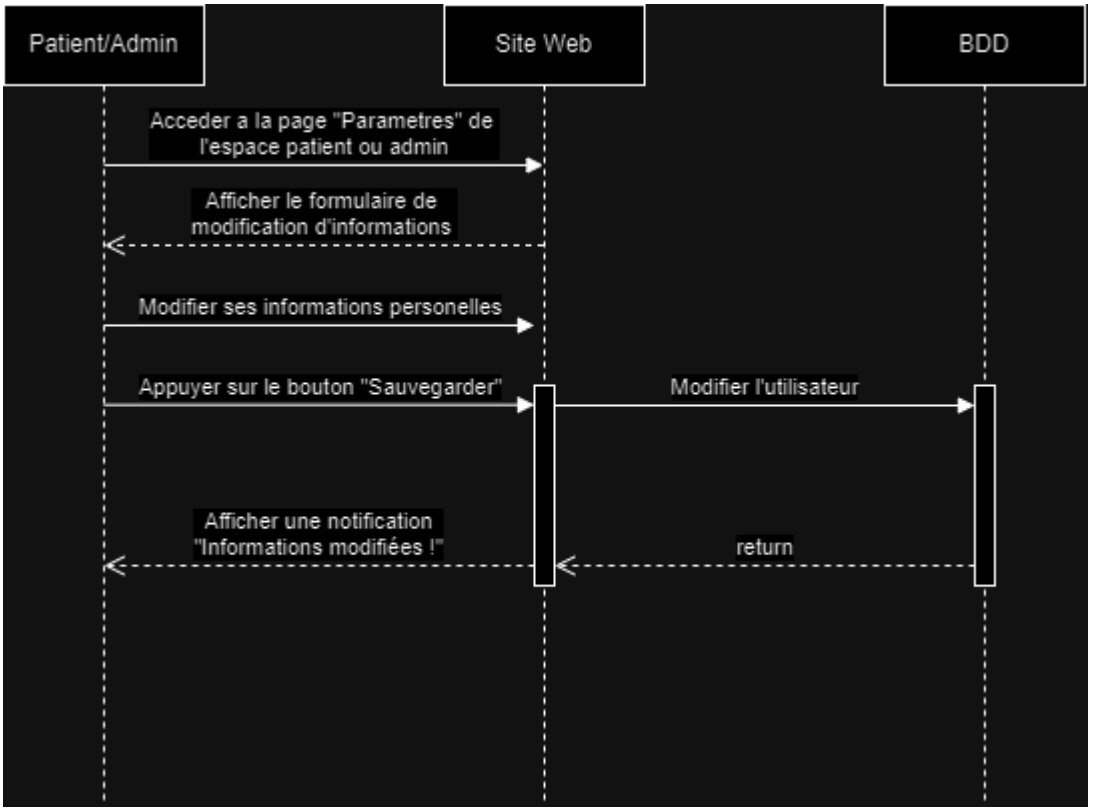
1.2. Connexion



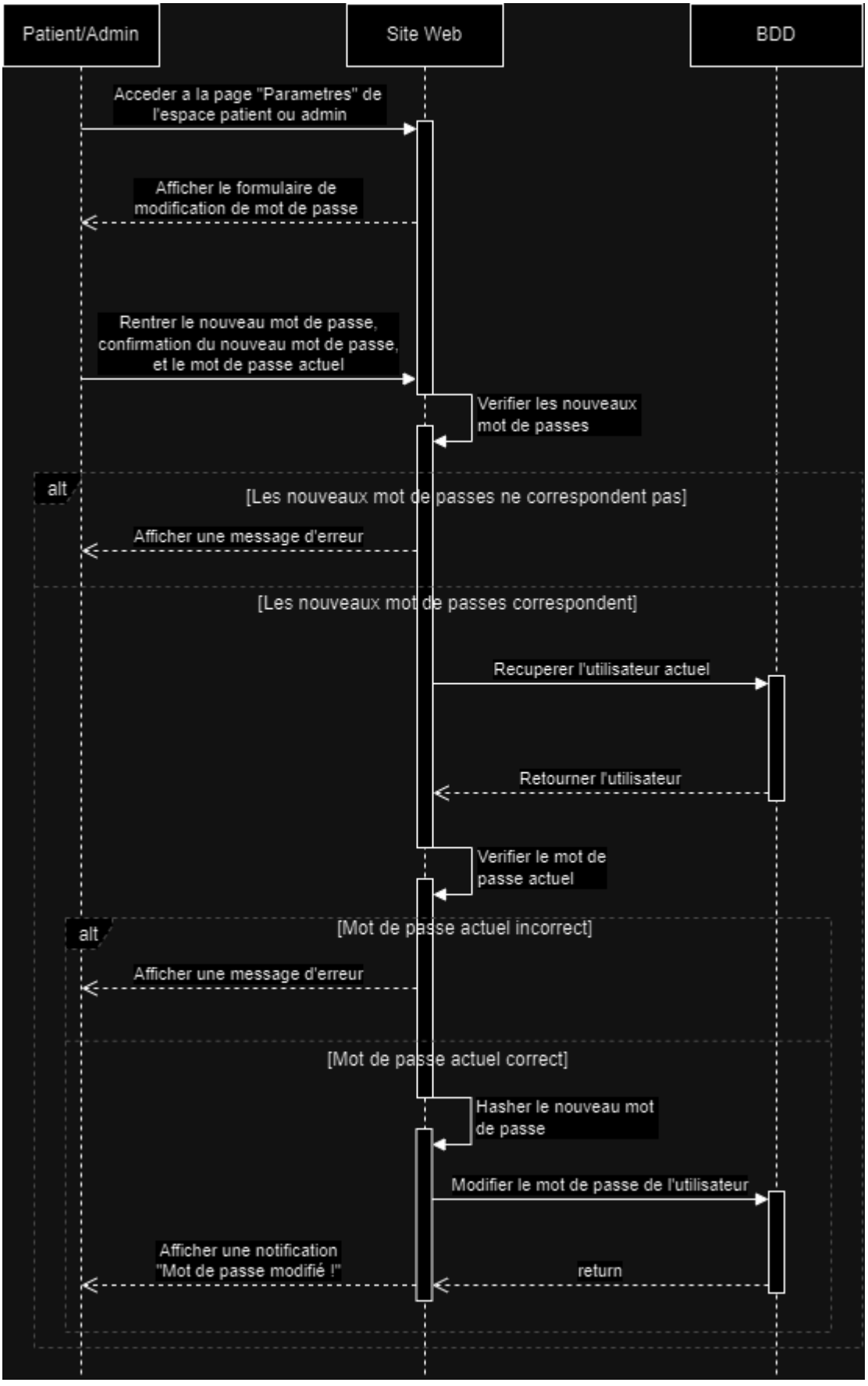
1.3. Inscription



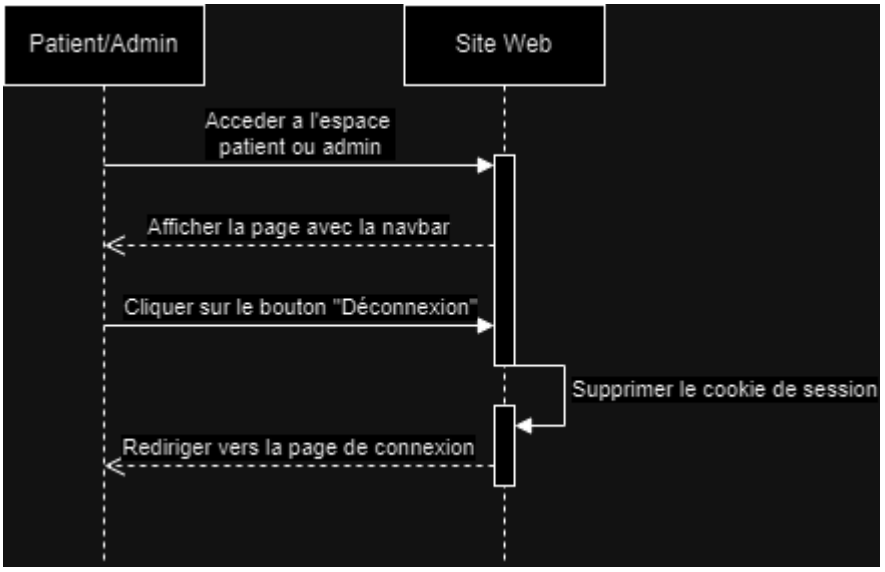
1.4. Modifier ses informations personnelles



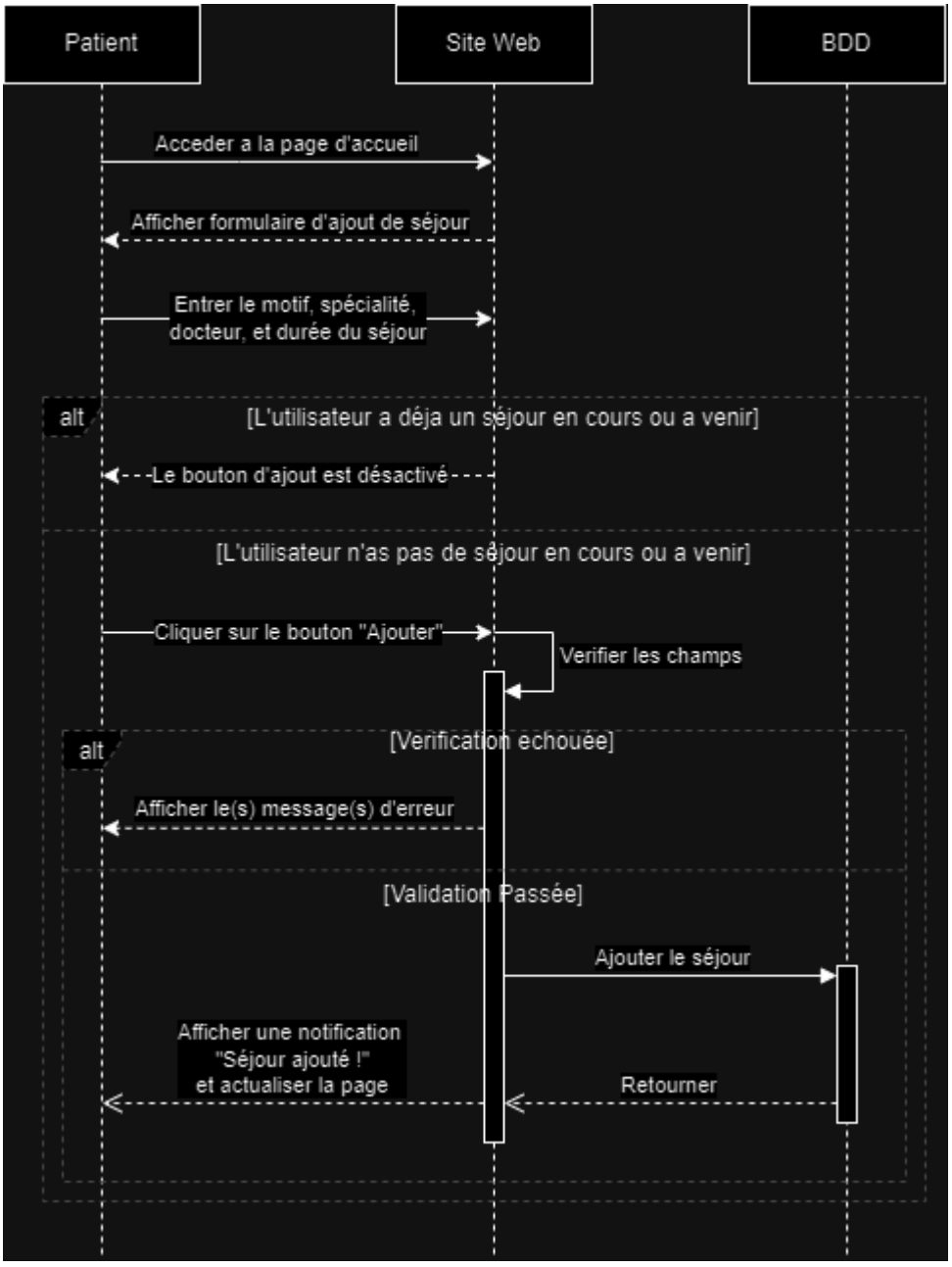
1.5. Modifier son mot de passe



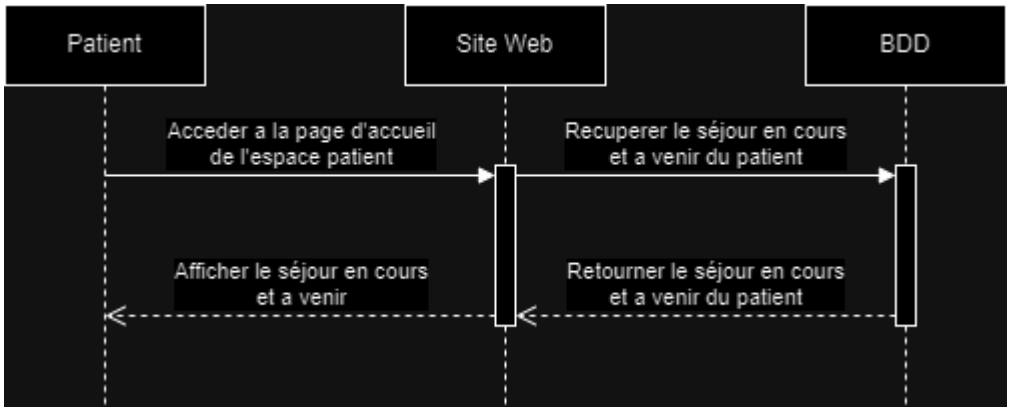
1.6. Déconnexion



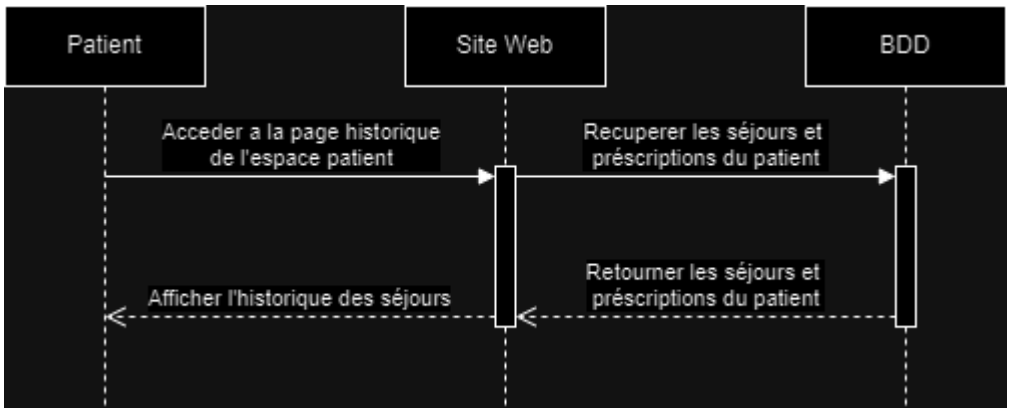
1.7. Ajouter un séjour



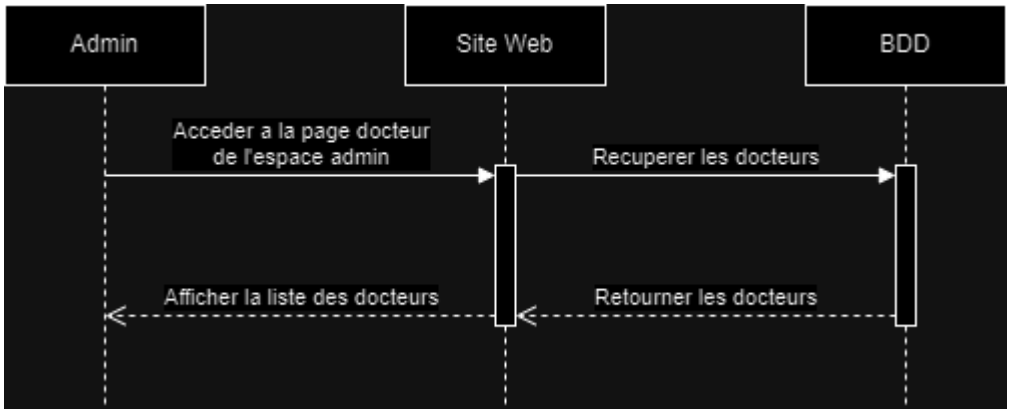
1.8. Voir son séjour actuel et a venir



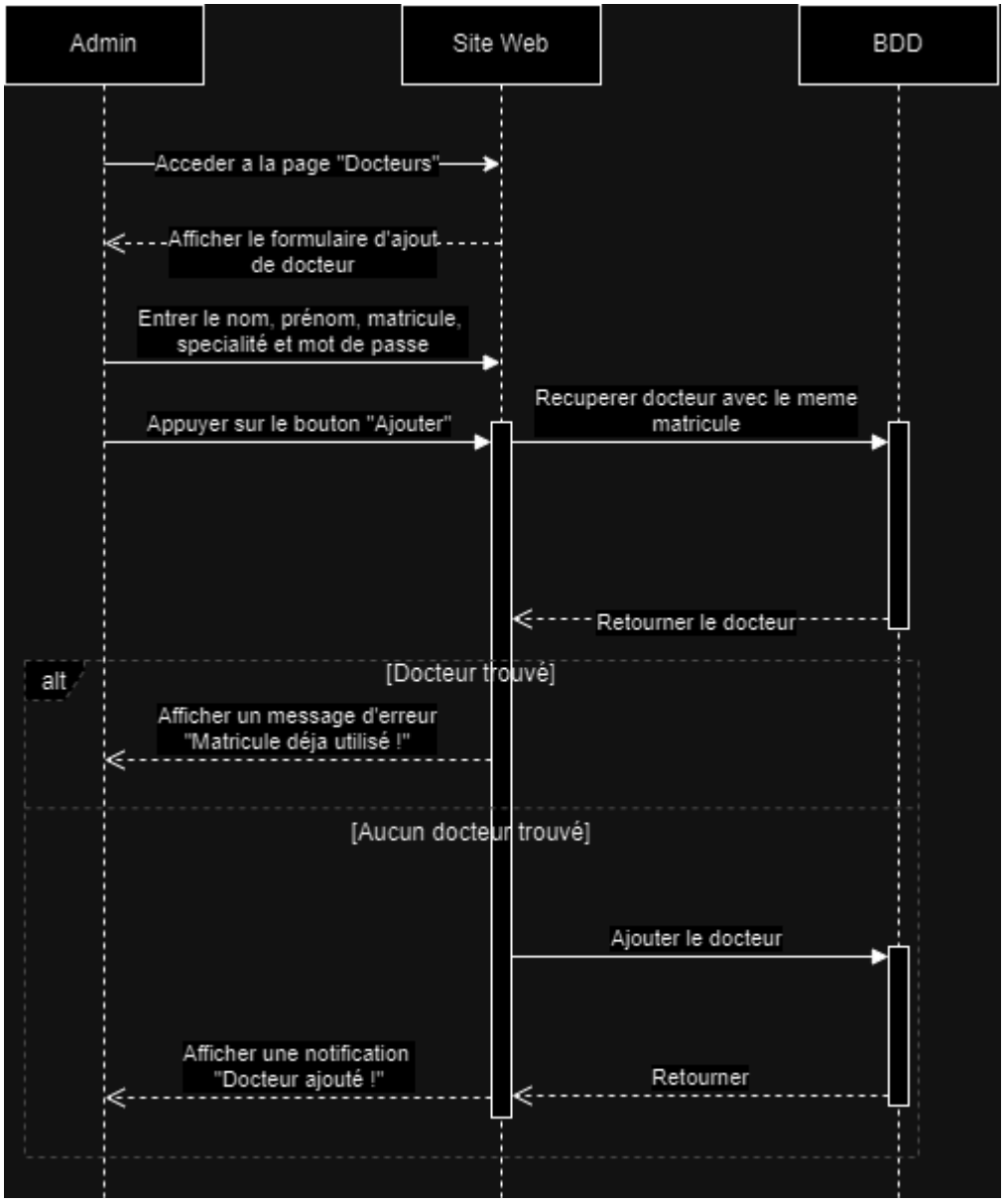
1.9. Voir son historique des séjours



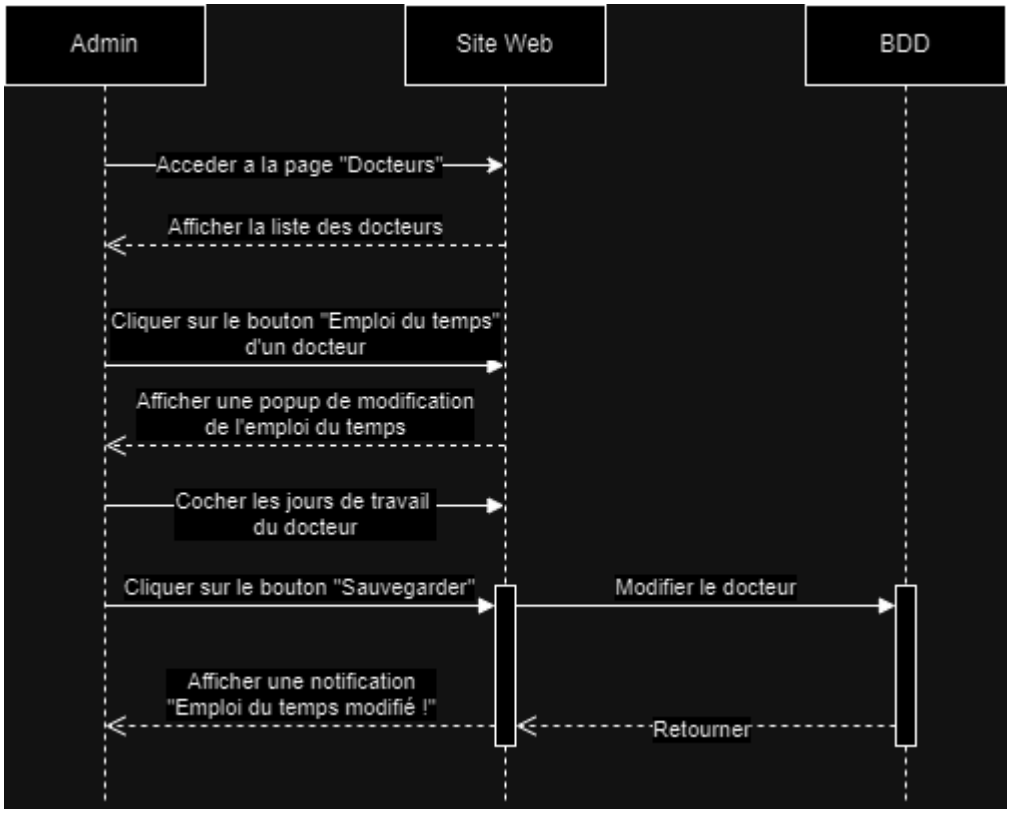
1.10. Voir la liste des docteurs



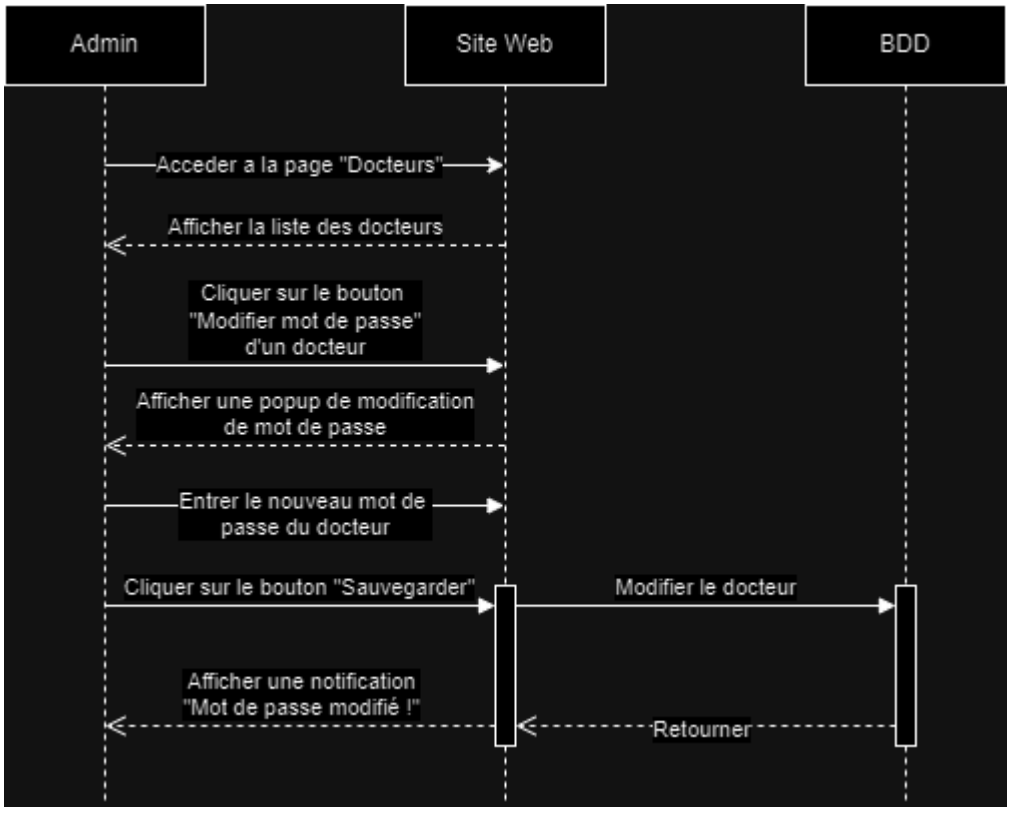
1.11. Ajouter un docteur



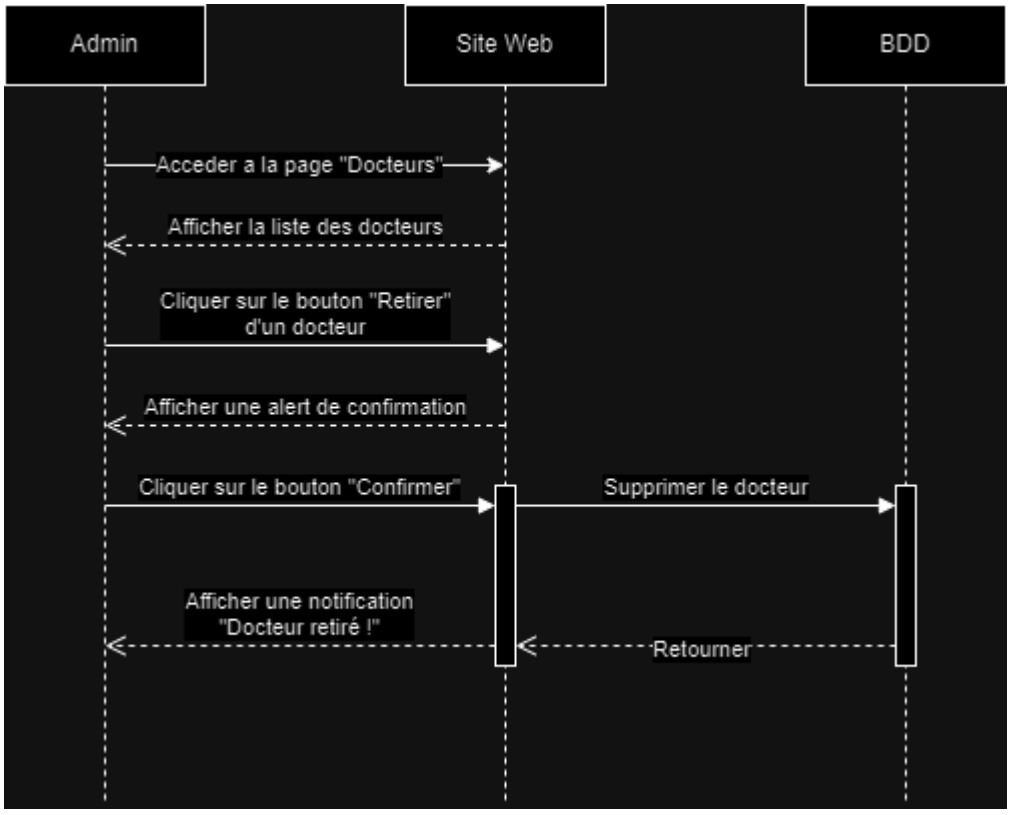
1.12. Modifier l’emploi du temps d’un docteur



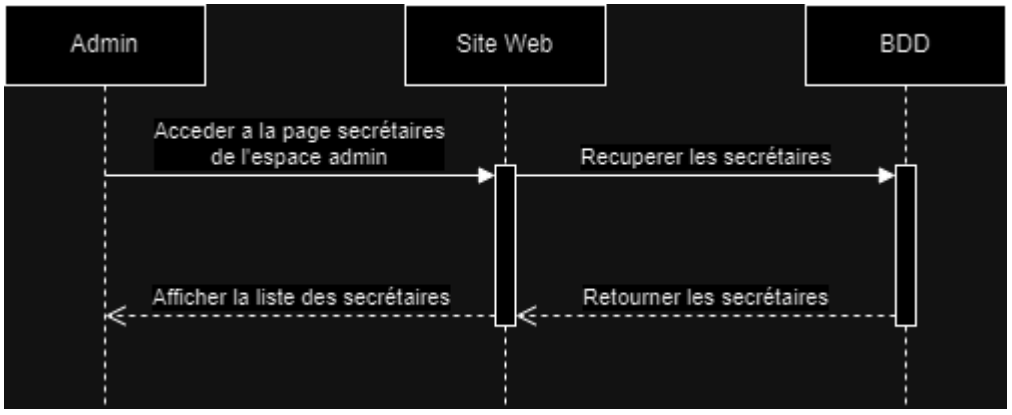
1.13. Modifier le mot de passe d'un docteur



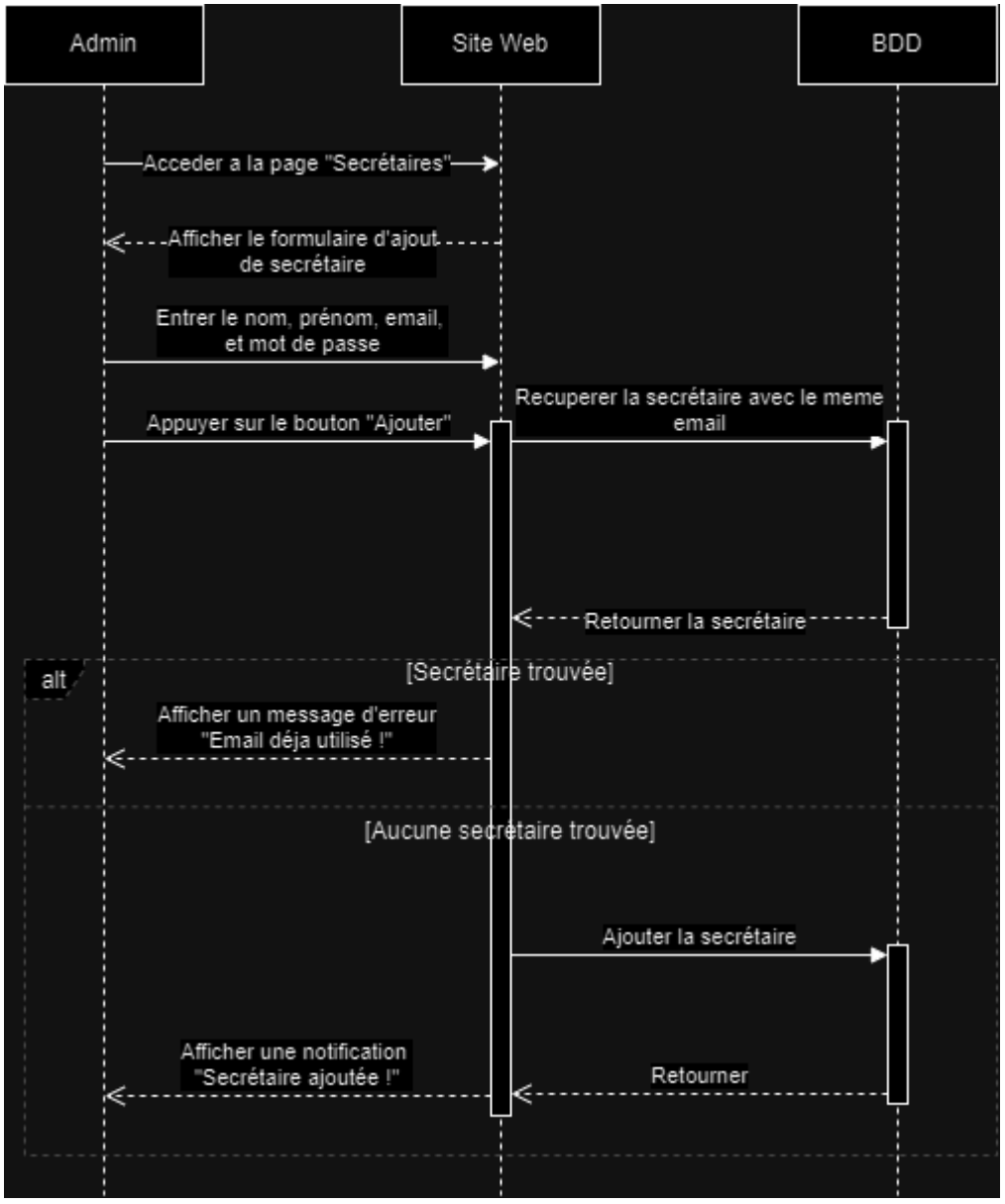
1.14. Retirer un docteur



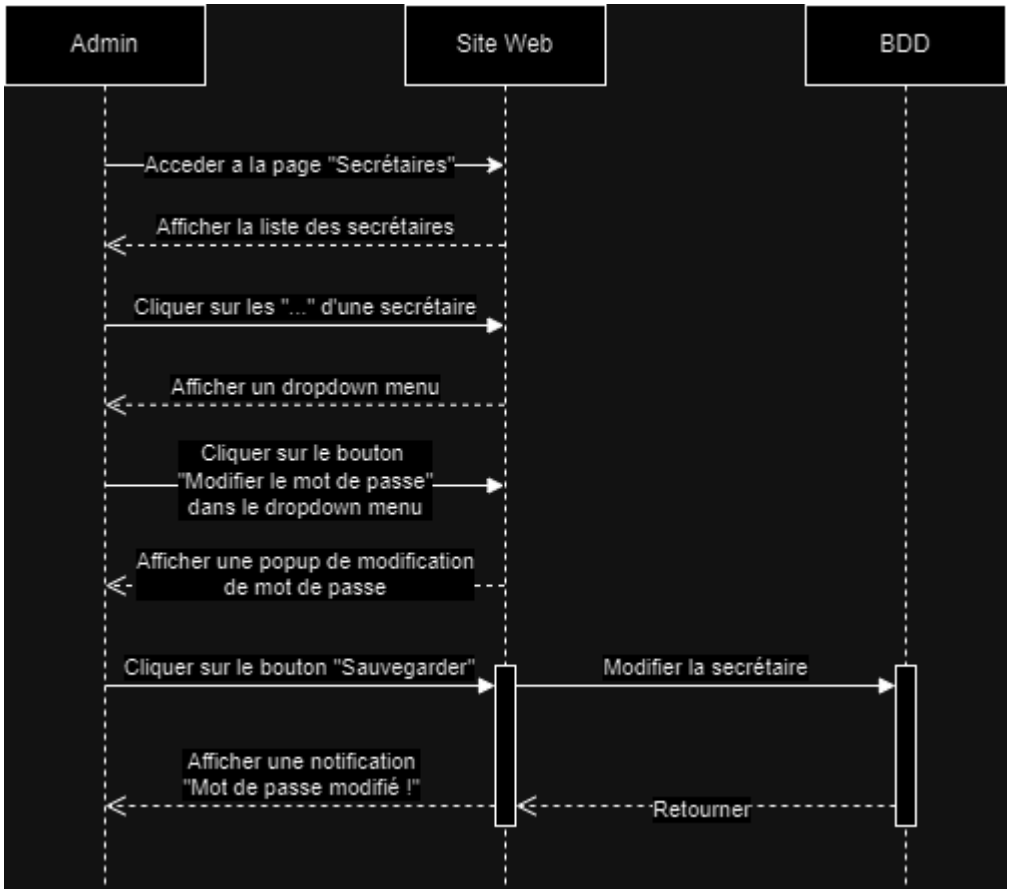
1.15. Voir la liste des secrétaires



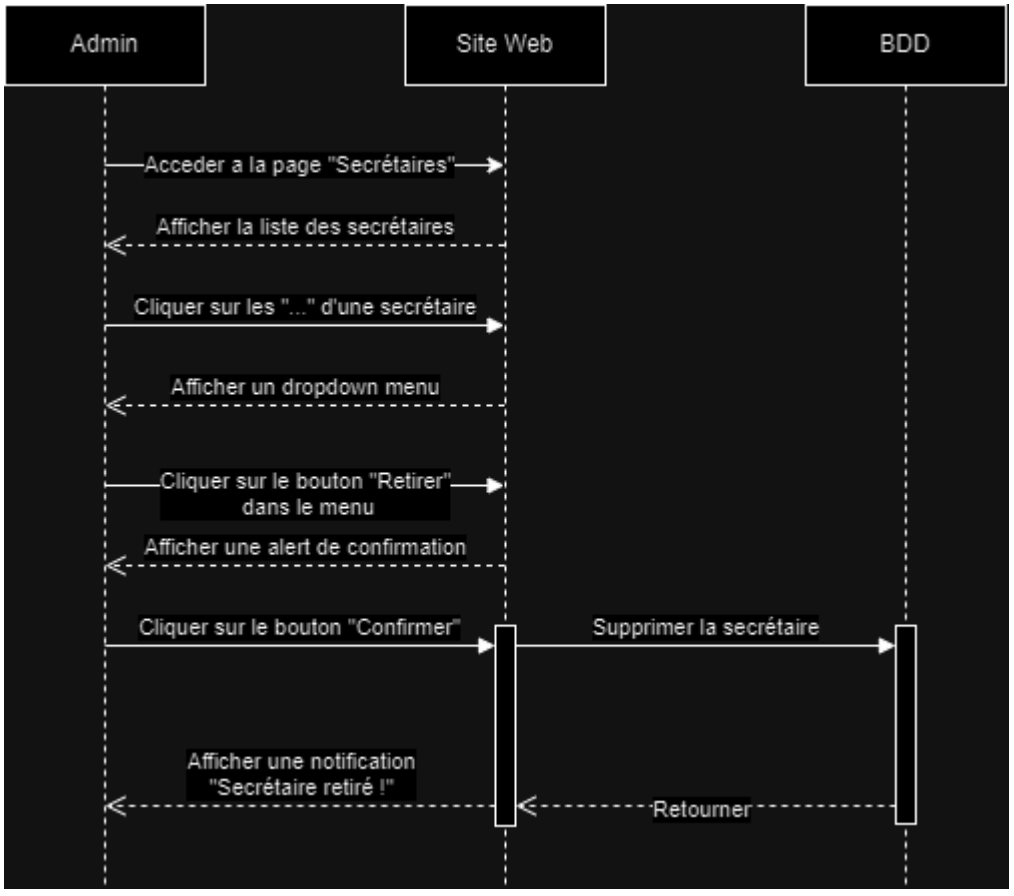
1.16. Ajouter une secrétaire



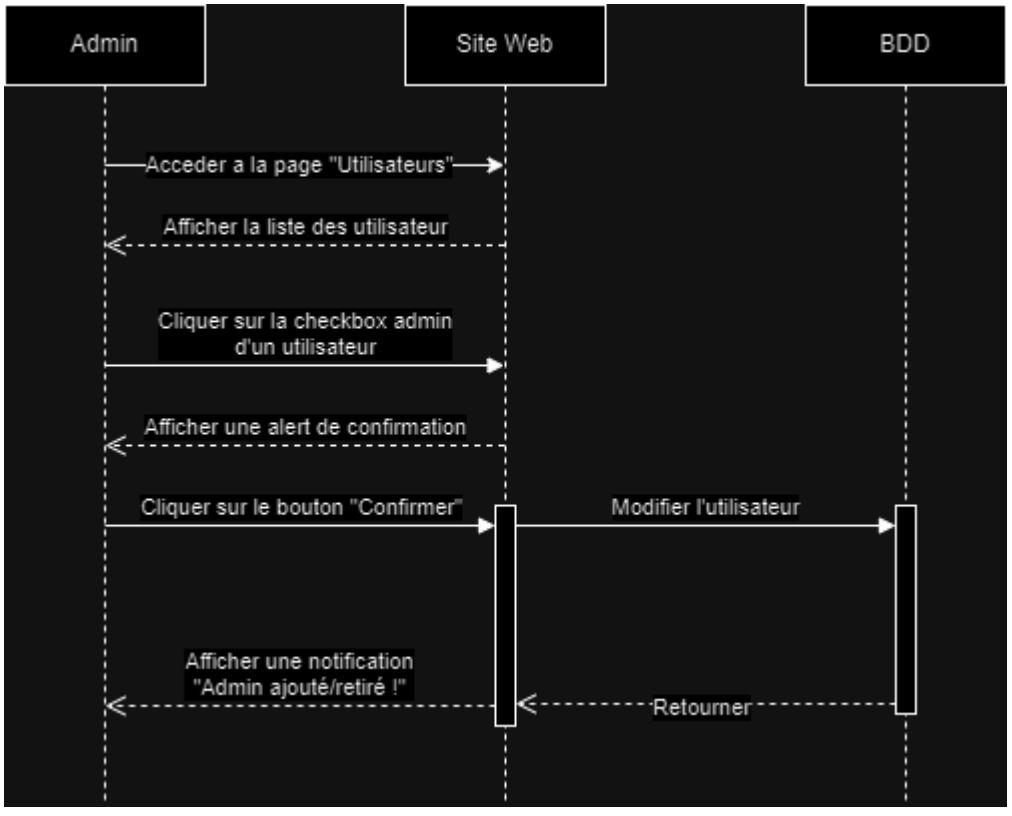
1.17. Modifier le mot de passe d’une secrétaire



1.18. Retirer une secrétaire

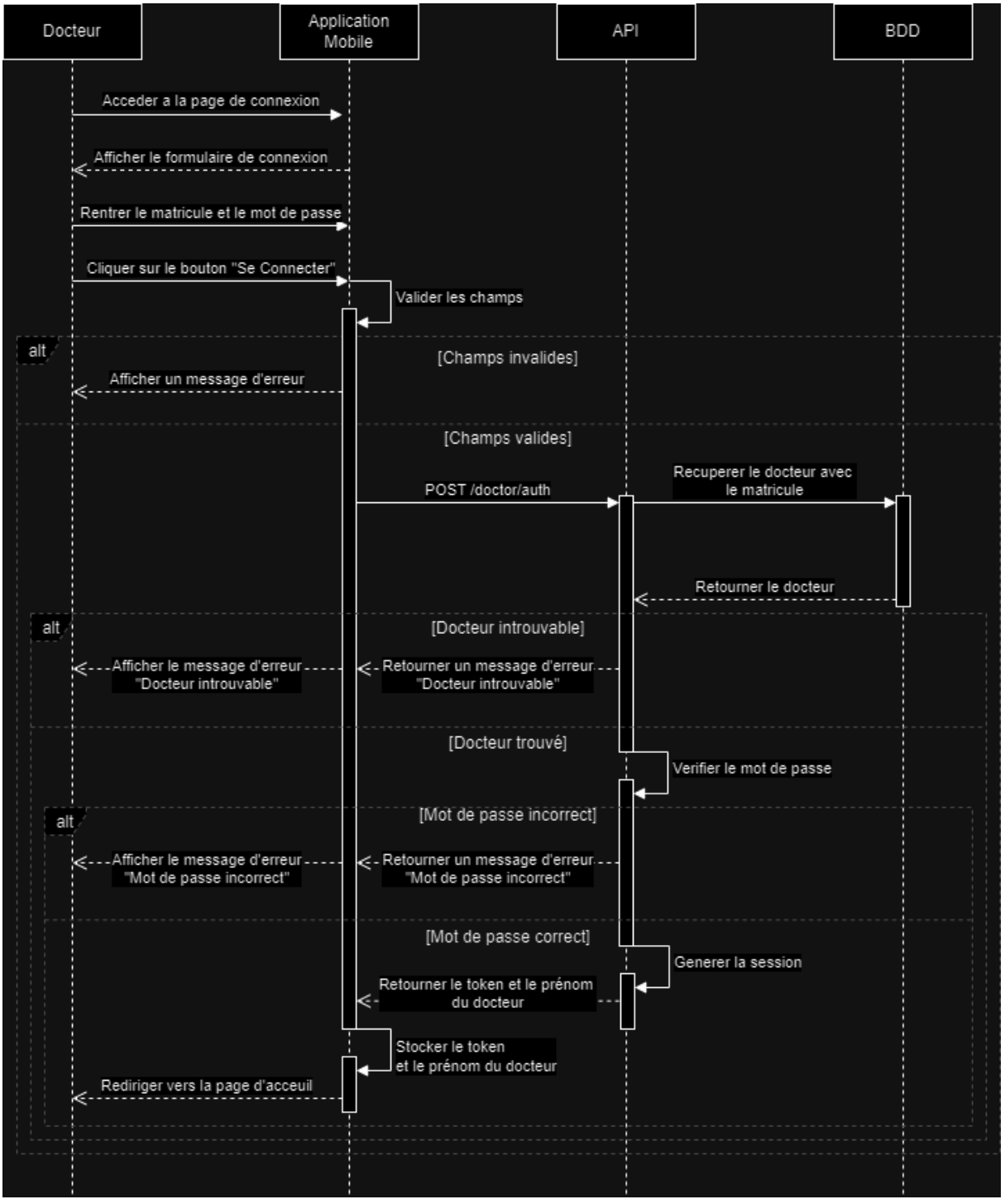


1.19. Ajouter/Retirer un administrateur

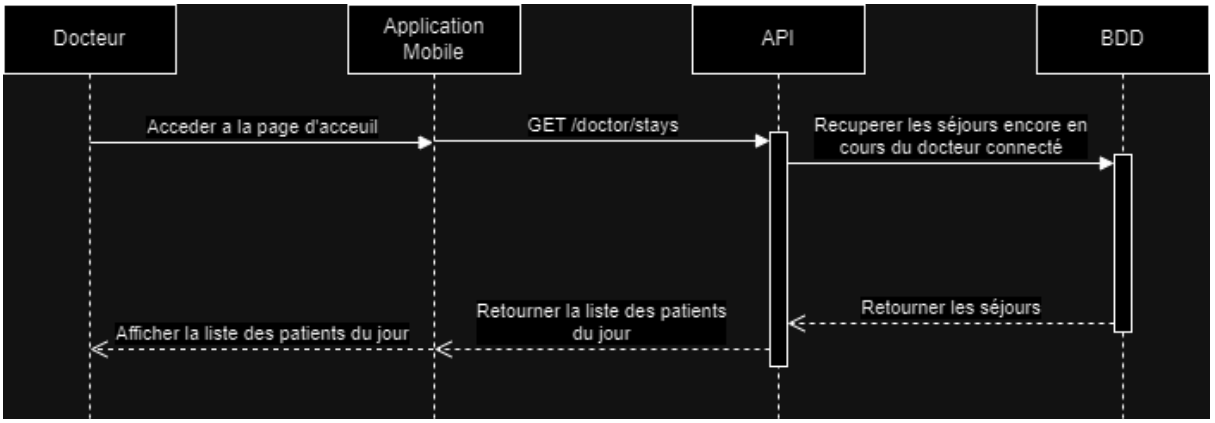


2. Application Mobile

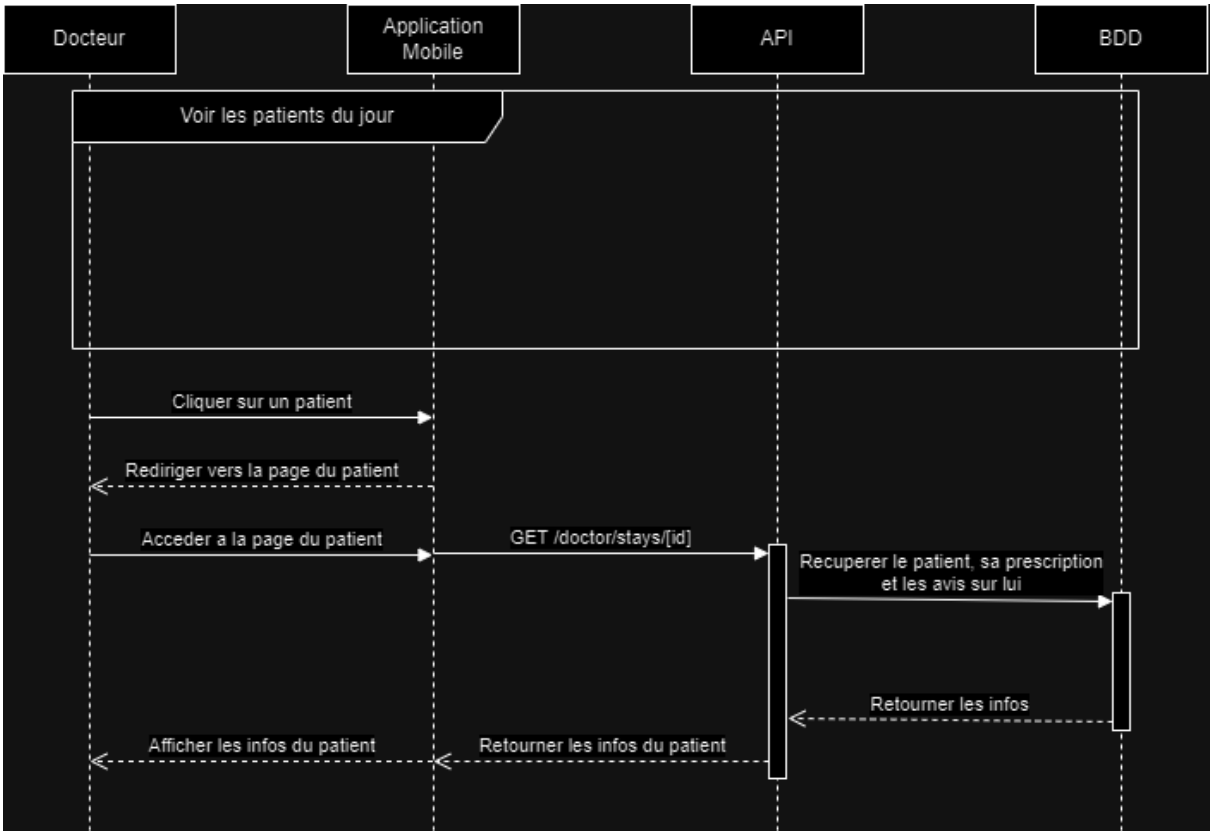
2.1. Connexion



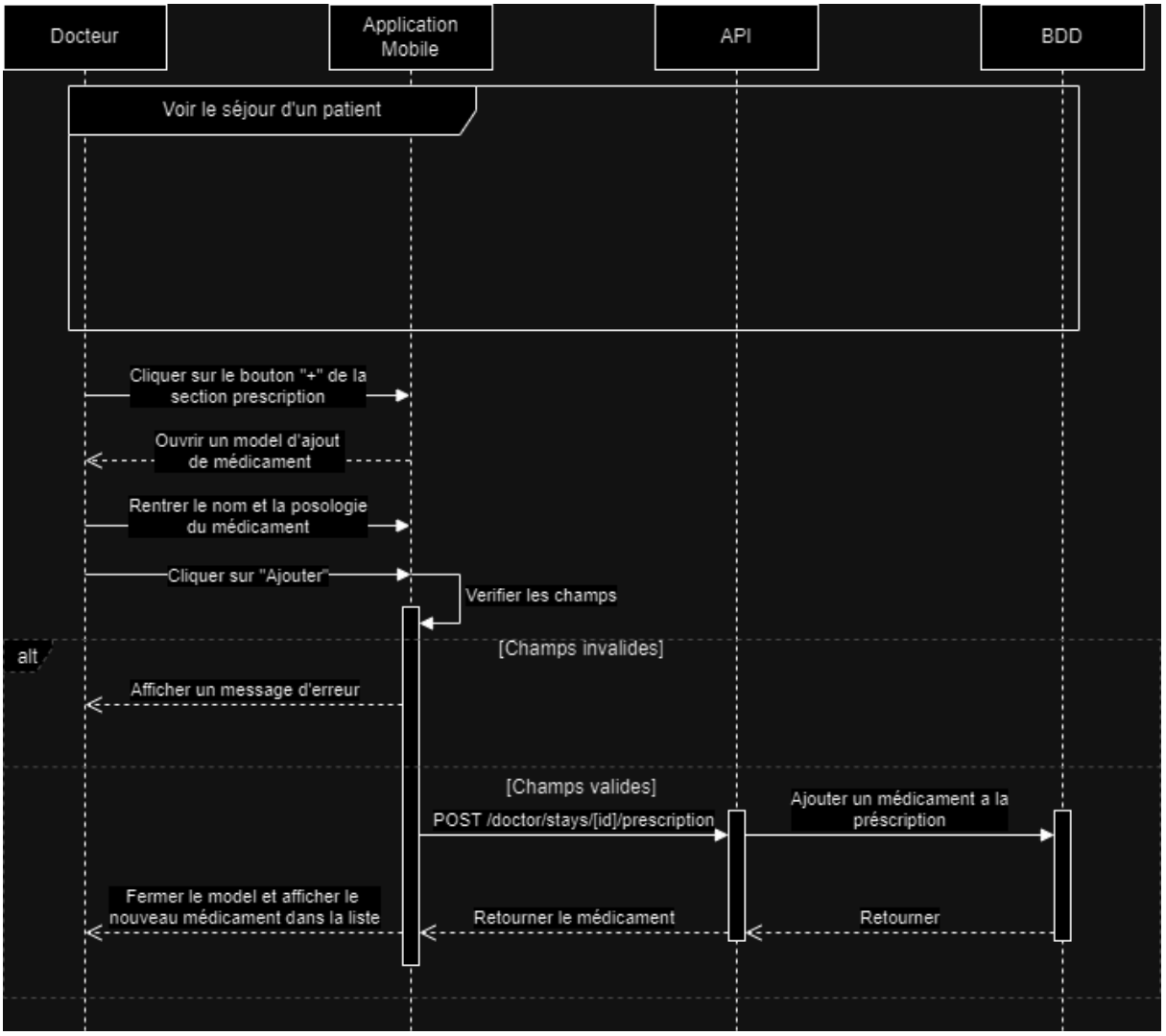
2.2. Voir les patients du jour



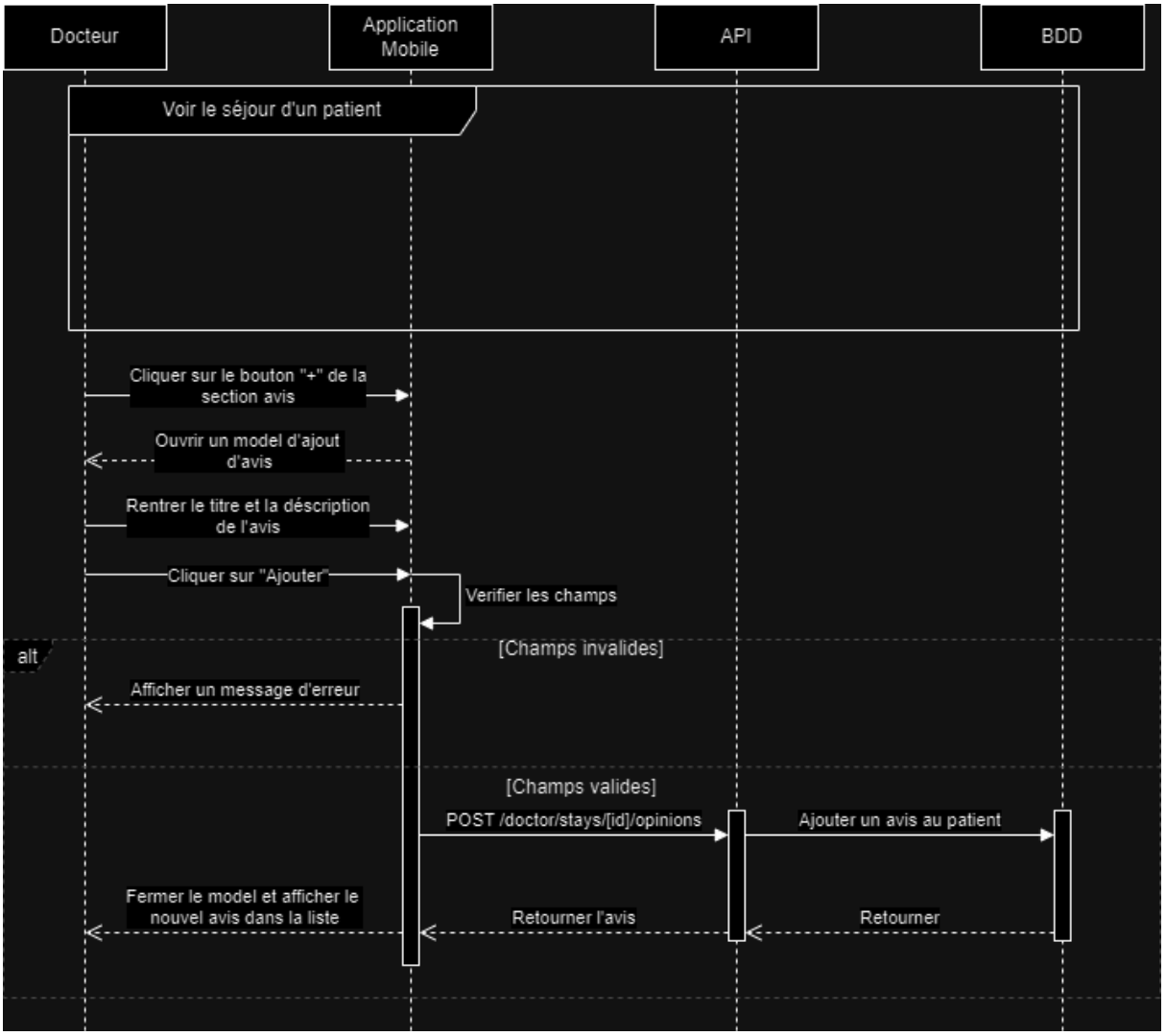
2.3. Voir le séjour d'un patient



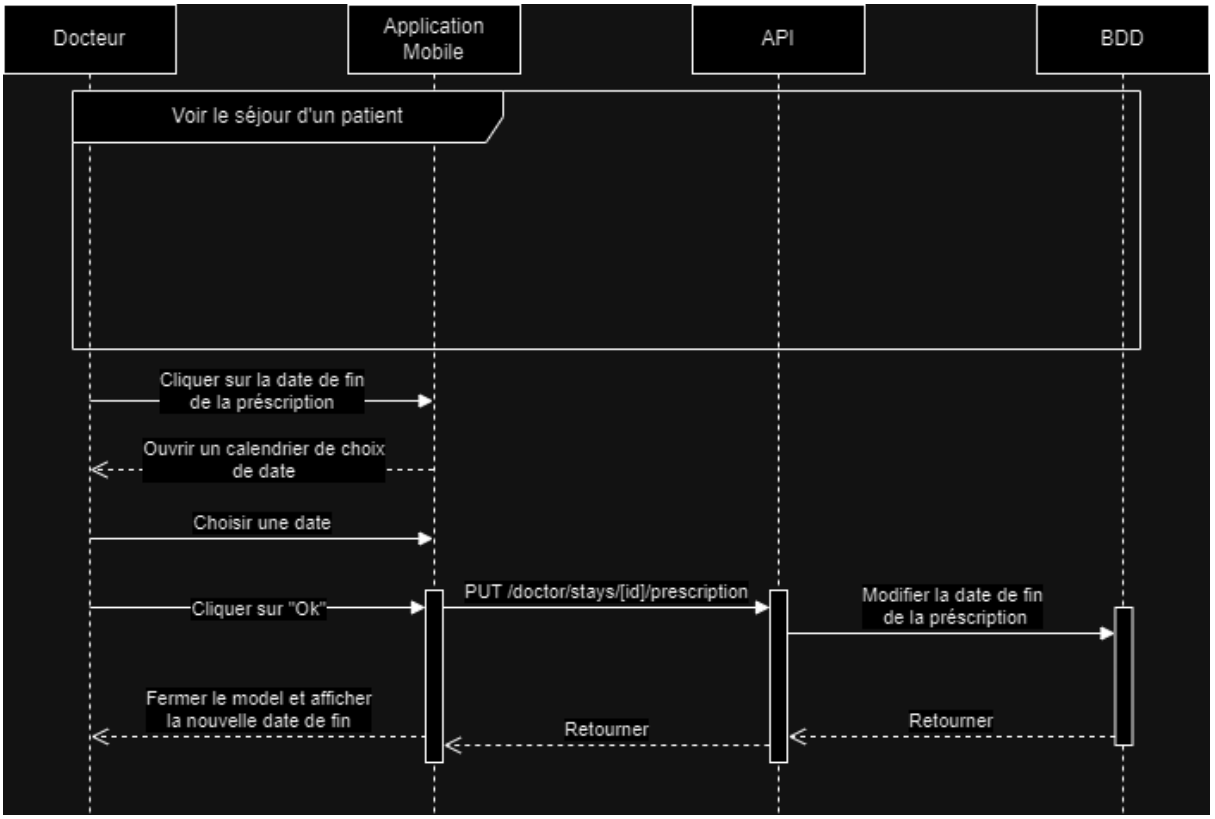
2.4. Ajouter un médicament



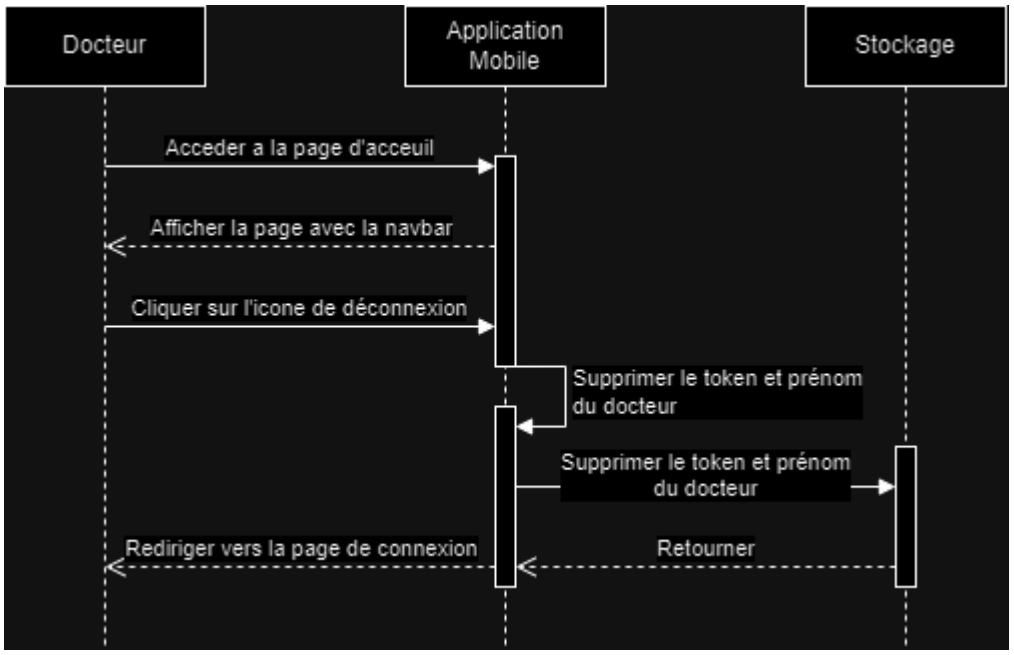
2.5. Ajouter un avis



2.6. Modifier la date de fin d’une prescription

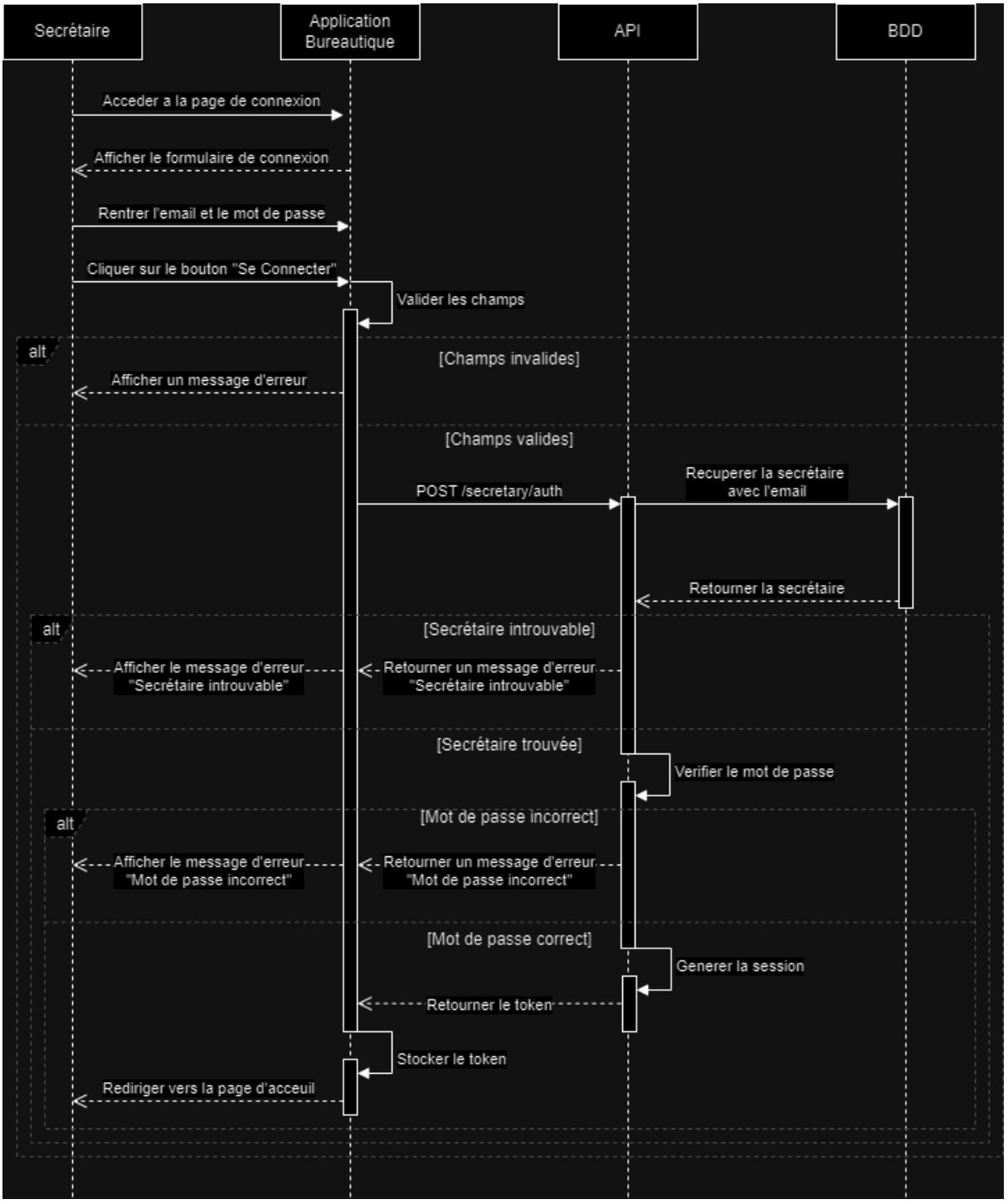


2.7. Déconnexion

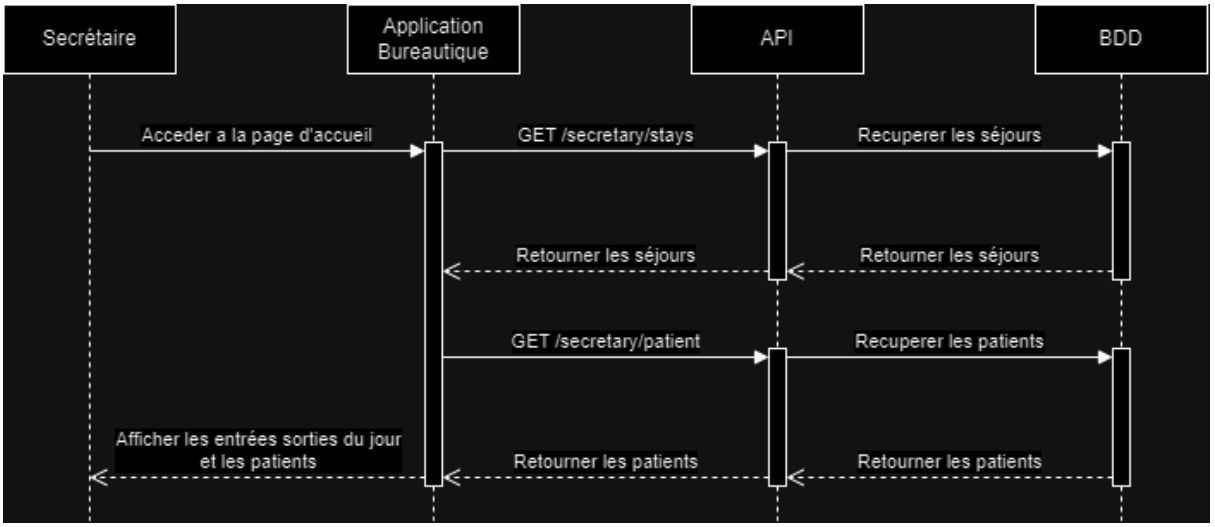


3. Application Bureautique

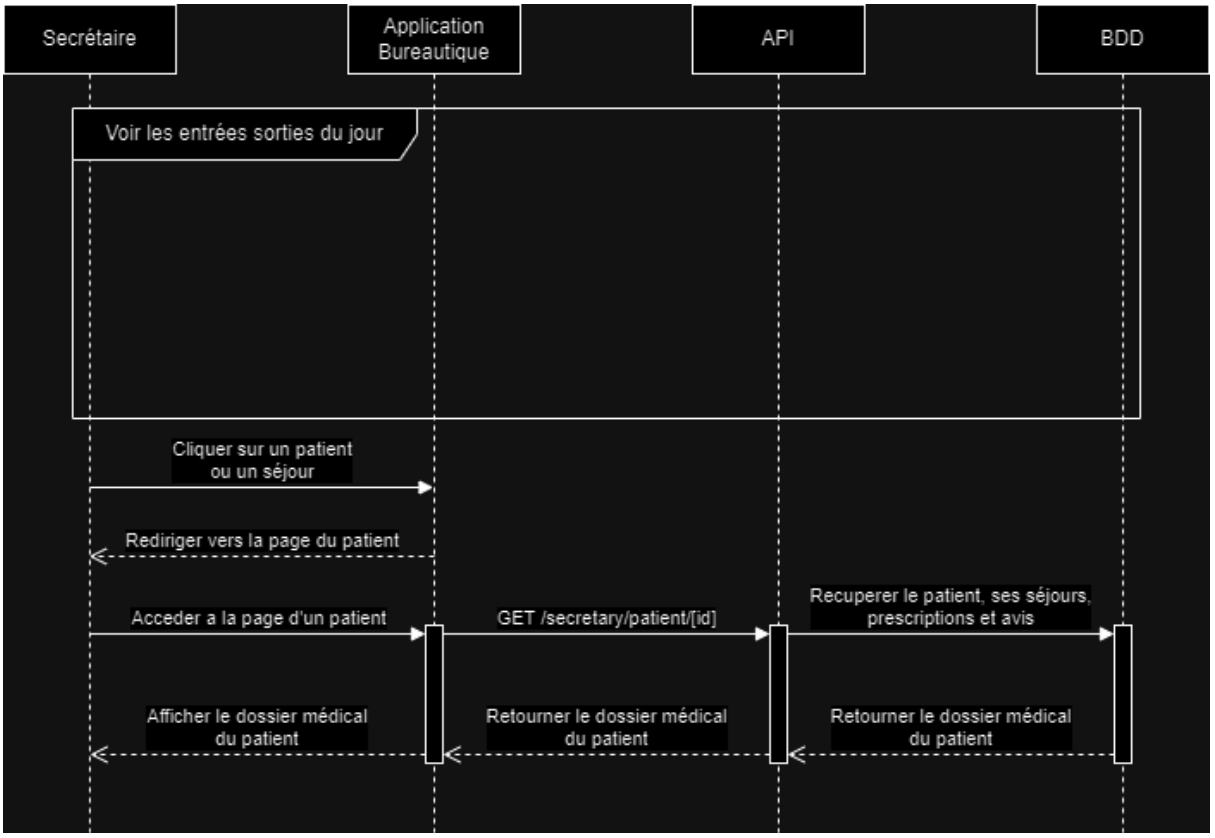
3.1. Connexion



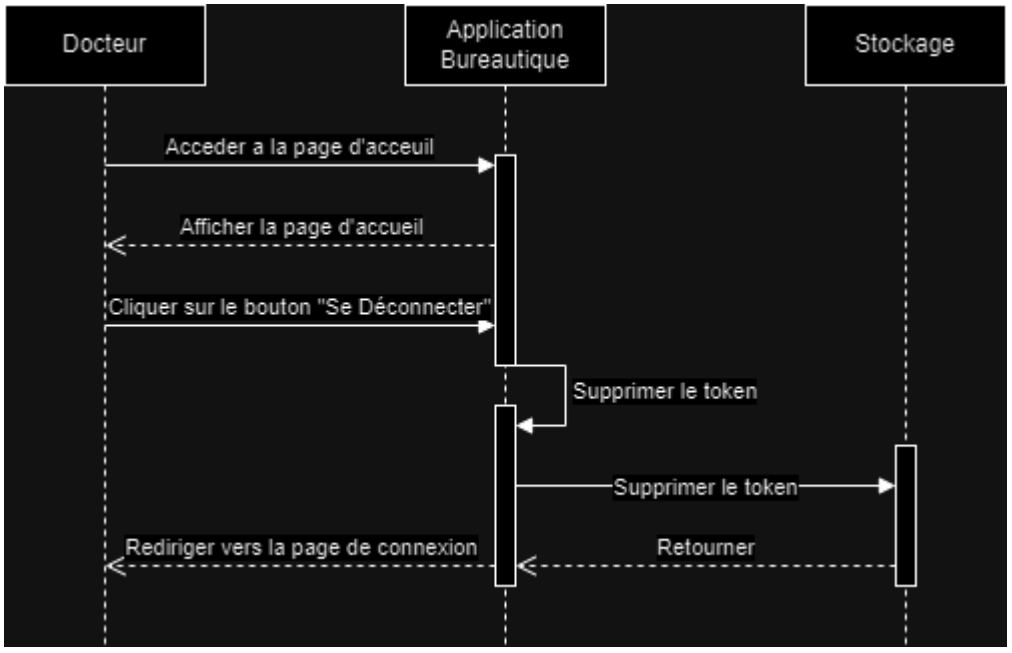
3.2. Voir les entrées/sorties du jour



3.3. Voir le dossier médical d'un patient



3.4. Déconnexion



Explication du plan de test

1. Objectif du plan de test

L'objectif des tests et de tester les différentes parties du code qui peuvent subir des changements ou des bugs dans le but d'assurer le bon fonctionnement de toutes les fonctionnalités et de s'assurer de ne pas casser une fonctionnalités lors d'un refactor ou d'un ajout de fonctionnalités.

2. Portée des tests

J'ai mis en place des tests unitaires sur les composants du site web pour vérifier que tous marche comme attendu et pour attraper les bugs des qu'ils arrivent

3. Outils de test

J'utilise Jest et Testing library dans le but de tester les composants web du site car c'est la combinaison la plus adapté, fiable, et recommandée par vercel (les créateurs de nextjs)