Équipe 10 - LEGRAIN-HORAIRES

Visualisation et construction d’horaires d’infirmières
Document d'architecture logicielle

Version 7.0

Historique des révisions

Date	Version	Description	Auteur
2023-01-30	1.0	Première version du document d'architecture logicielle	Caroline Des Rochers
2023-02-06	2.0	Deuxième version du document d'architecture l -Mise à jour vue logique (Section 4) -Mise à jour vue de déploiement (Section 6)	Caroline Des Rochers Mark Bekhet Kyrollos Bekhet
2023-02-2	3.0	Troisième version du document d'architecture -Mise à jour vue logique (Section 4) -Mise à jour taille et performance (Section 7)	Caroline Des Rochers Mark Bekhet
2023-02-16	4.0	Quatrième version du document d'architecture - Révision document d'architecture logicielle	Caroline Des Rochers Mark Bekhet Kyrollos Bekhet Zoé Paradis Malgorzata Niczyporuk
2023-04-12	5.0	Cinquième version du document d'architecture - Mise à jour suite aux corrections de mis session	Caroline Des Rochers Mark Bekhet
2023-04-15	6.0	Sixième version du document d'architecture - Mise à jour de la vue logique et de déploiement de l'application suite à l'évolution de l'architecture	Caroline Des Rochers Mark Bekhet
2023-04-18	7.0	Septième version du document d'architecture - Révision final du document d'architecture logicielle	Caroline Des Rochers Mark Bekhet Kyrollos Bekhet Zoé Paradis Malgorzata Niczyporuk

Table des matières

Document d'architecture logicielle

1. Introduction

Le document qui suit présente l'architecture logicielle de l'application web de visualisation et construction d'horaires d'infirmières, soit les différents éléments architecturaux ainsi que la manière dont ils interagissent entre eux. L'ensemble des diagrammes contenus dans ce document respectent UML et celui-ci est organisé de la façon suivante. La section 2 décrit les objectifs et les contraintes possédant un impact architectural. Ensuite, la vue des cas d'utilisation contenant les diagrammes de cas d'utilisation, la vue logique contenant les diagrammes de paquetages , et la vue des processus contenant les diagrammes de séquences sont présentées dans les sections 3, 4 et 5. Quant à elle, la section 6 présente la vue logique en indiquant les nœuds physiques et virtuels qui exécutent le logiciel, ainsi que leurs interconnexions. Finalement, on retrouve dans la section 7 les caractéristiques de taille et de performance pouvant avoir un impact sur l'architecture et le design du logiciel.

2. Objectifs et contraintes architecturaux

2.1 Objectifs

2.1.1 Réutilisabilité

Le logiciel doit avoir des classes et des modules avec un faible couplage pour limiter les dépendances entre les classes, pour faciliter le changement d'une classe et sa réutilisabilité. Aussi, les classes doivent utiliser le plus possible l'héritage et le polymorphisme pour faciliter l'utilisation des patrons de conception lors de l'ajout d'une nouvelle fonctionnalité.

2.1.2 Sécurité et confidentialité

Afin d'assurer la sécurité des entrées de l'utilisateur, nous allons utiliser le protocole HTTPS plutôt que le protocole HTTP afin d'assurer l'encryption des entrées de l'utilisateur surtout si les entrées contiennent des données confidentielles comme des mots de passe. En plus de l'encryption des communications entre le client et le front-end, ainsi que l'encryption entre le *front-end* et le *back-end*, nous allons sauvegarder les mots de passe chiffrés dans la base de données.

2.1.3 Fiabilité

En cas de panne le système doit être en mesure de redémarrer sans aucune perte d'information. L'objectif est d'assurer la gestion des pannes en garantissant l'intégrité des données, et ce dans un délai bref. De plus, le logiciel doit respecter la règle du 80/20, c'est-à-dire que 80% des données présentées sur l'écran de l'utilisateur doivent venir du serveur et l'autre 20% est géré du côté client.

2.2 Contraintes

2.2.1 Les outils de développement

Le développement du *front-end* se fait avec Visual Studio Code qui présente un bon support pour le langage Typescript. D'ailleurs pour le développement de back-end, nous avons décidé d'utiliser l'IDE Pycharm offert par la compagnie JetBrain, car il offre un bon support au langage de programmation Python. Afin de tester l'application en cours de développement, nous allons utiliser un modèle de déploiement avec Docker et Docker-compose. Ceci a lieu, car nous allons utiliser beaucoup de conteneurs dans l'application.

2.2.2 Le langage de développement

Le code actuel de l'outil d'optimisation d'horaires d'infirmières est codé en C++. Ainsi pour assurer que l'application web s'intègre bien avec le code actuel, celui-ci sera appelé en ligne de commande par un serveur Python. L'application web sera développée en Typescript avec le cadriciel Angular. Pour sa part, le serveur sera codé en Python dans l'environnement en utilisant le cadriciel Flask.

2.2.3 Portabilité

L'application web de visualisation et de constructions d'horaire doit pouvoir fonctionner dans tous navigateurs

modernes pour favoriser un accès facile aux utilisateurs. En plus, afin de pouvoir déployer nos serveurs, nous utilisons Docker et Docker-compose afin de pouvoir fonctionner sur n'importe quelle machine.

2.2.4 Échéancier et coût

Étant donné que le logiciel doit être développé en 3 mois, l'architecture sera développée de façon continue selon les besoins des fonctionnalités à implémenter. Dans chacun des sprints prévus, le prototype de l'application va évoluer pour comprendre les fonctionnalités décrites dans l'échéancier afin d'obtenir un produit complet après 3 mois de développement. Les coûts de développement doivent être nuls.

3. Vue des cas d'utilisation

3.1 Cas d'utilisation général

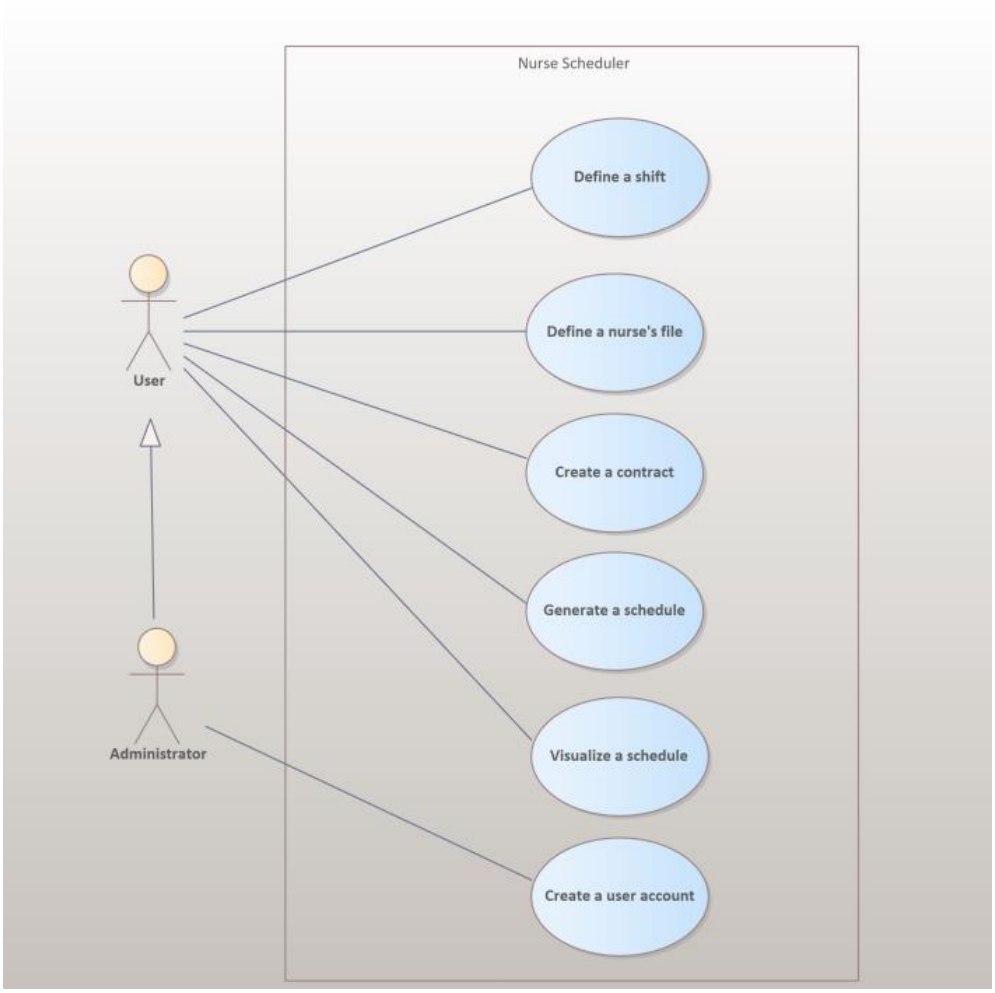

Figure 1 : Cas d'utilisation général de l'application *Nurse Scheduler*

3.1.1 Cas d'utilisation création de compte

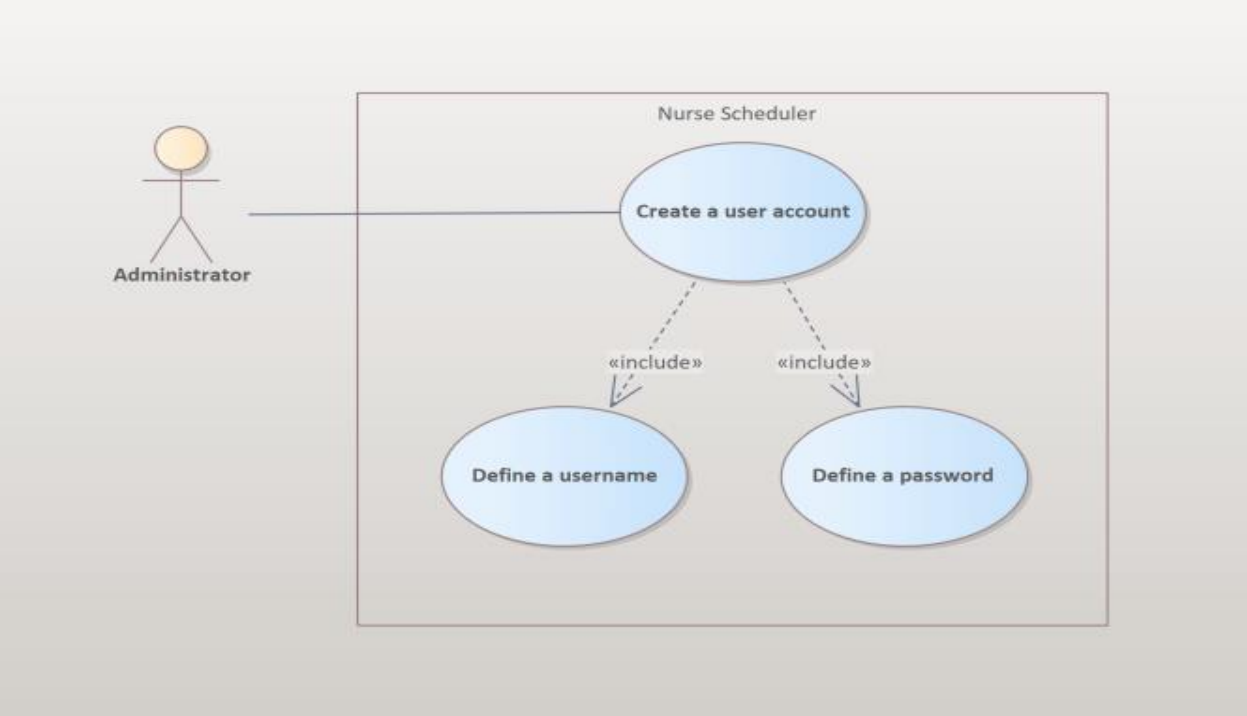

Figure 2 : Cas d'utilisation de la création d'un compte

3.1.2 Cas d'utilisation définition d'un quart de travail

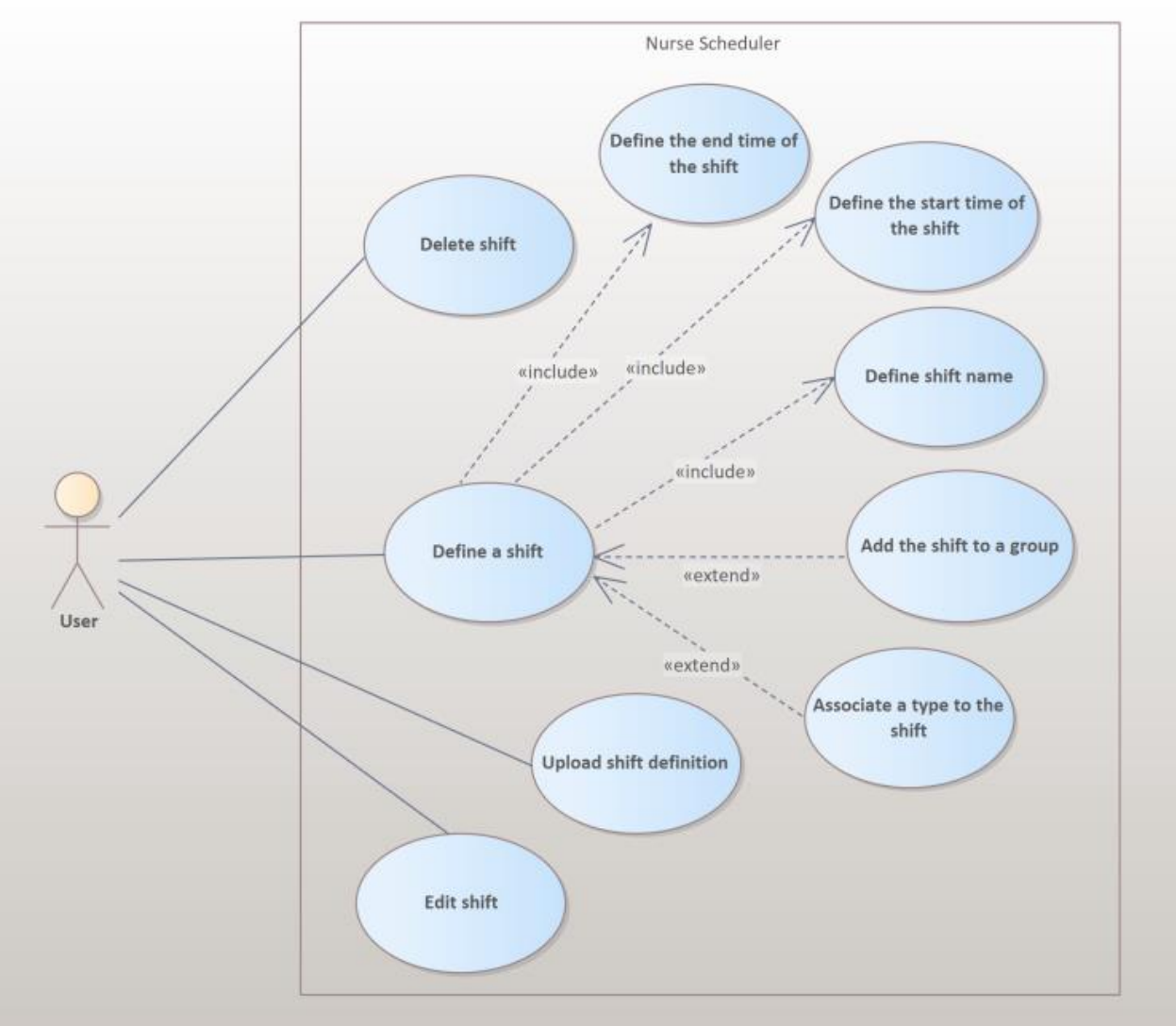

Figure 3 : Cas d'utilisation de la définition d'un quart de travail

3.1.3 Cas d'utilisation création de contrat

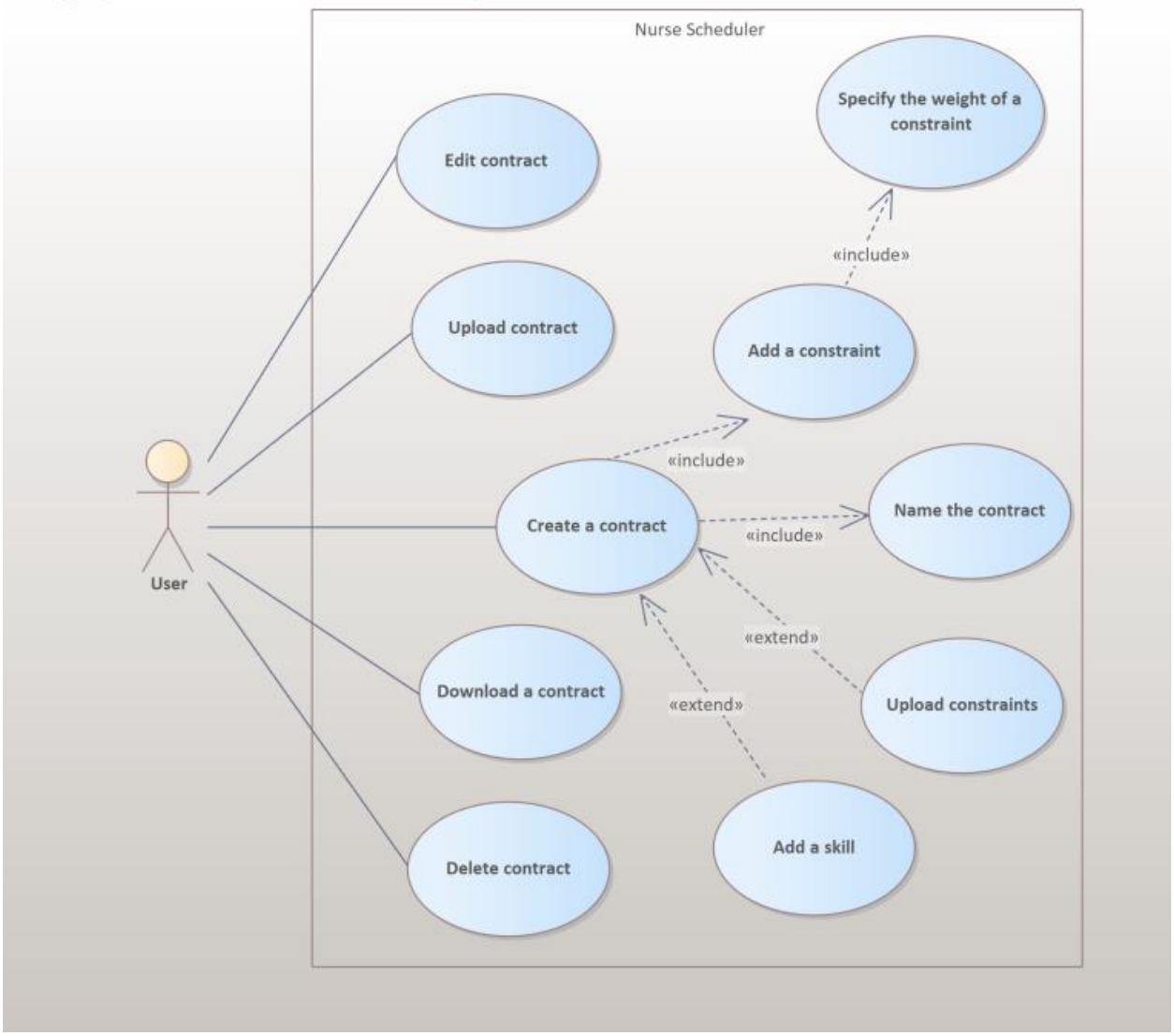

Figure 4 : Cas d'utilisation de la création d'un contrat

3.1.4 Cas d'utilisation création dossier d'infirmière

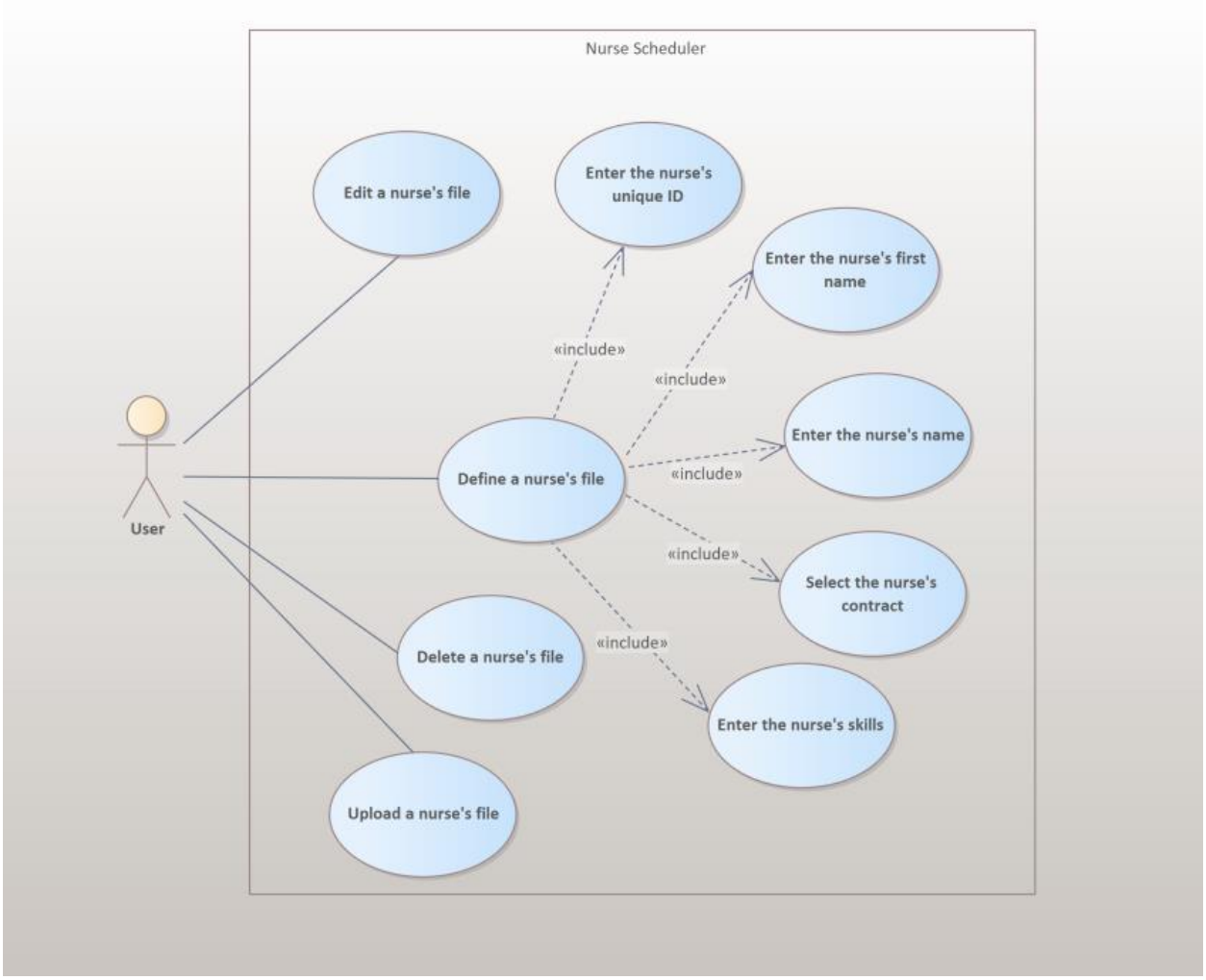

Figure 5 : Cas d'utilisation de la création d'un dossier d'infirmière

3.1.5 Cas d'utilisation génération d'horaire

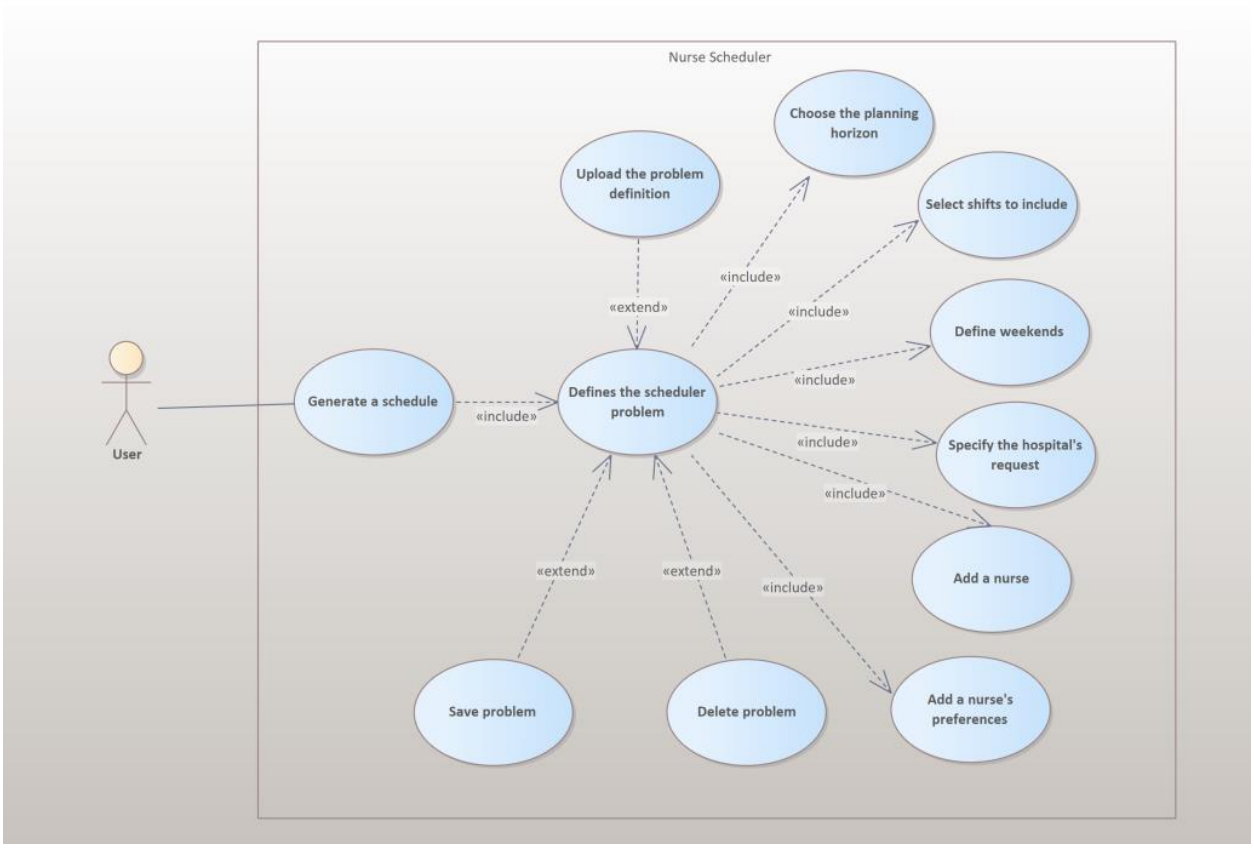

Figure 6 : Cas d'utilisation de la génération d'un horaire

3.1.6 Cas d'utilisation visualisation d'horaire

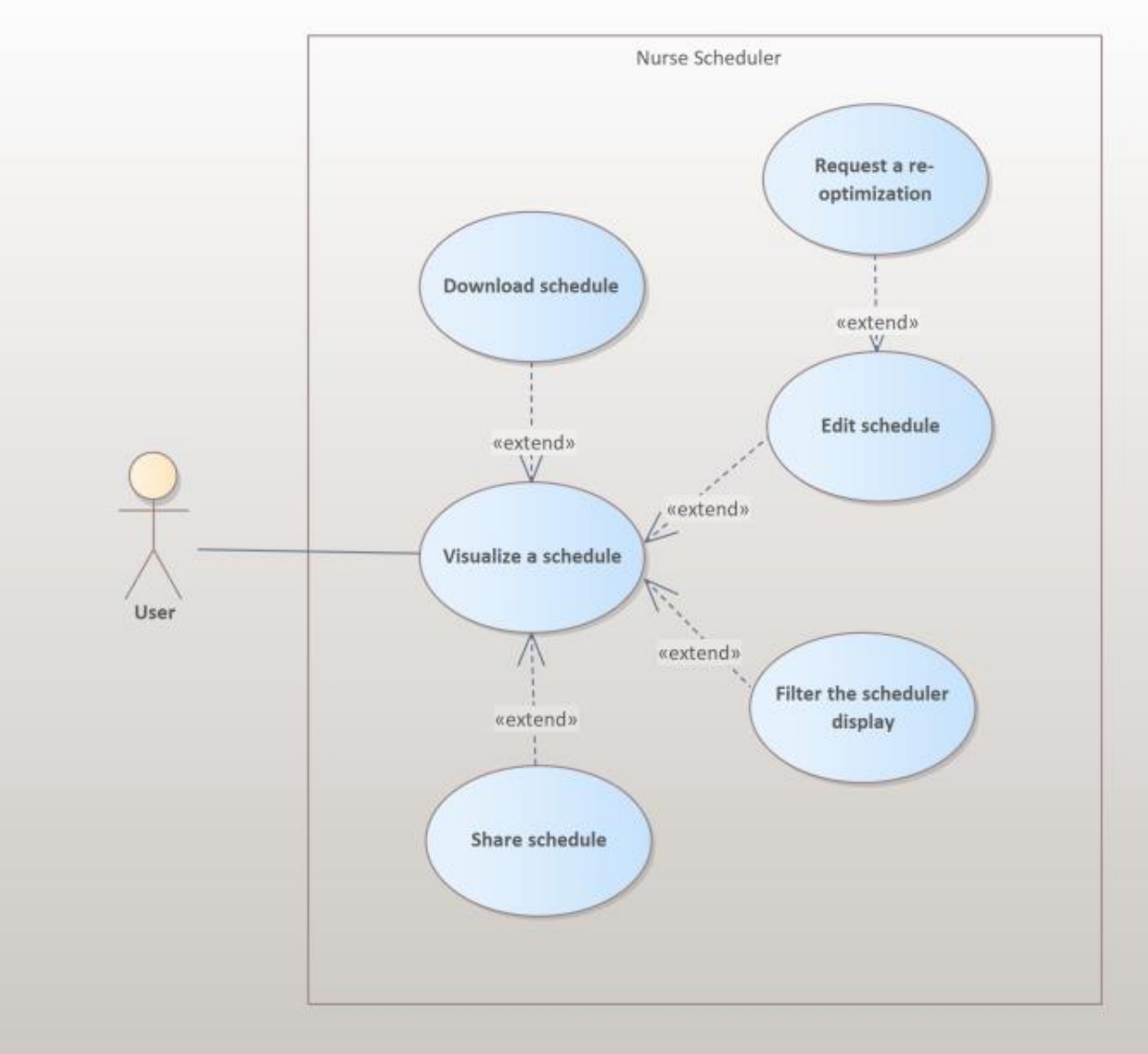

Figure 7 : Cas d'utilisation de la visualisation d'un horaire

4. Vue logique

4.1 Diagramme de paquetage du système complet

Nurse Scheduler
Contiens l'ensemble des composantes principales du projet Legrain_Horaire. Il s'agit d'un dossier qui rassemble le paquetage *nurse_Scheduler_app* et le *front_facing_server*.

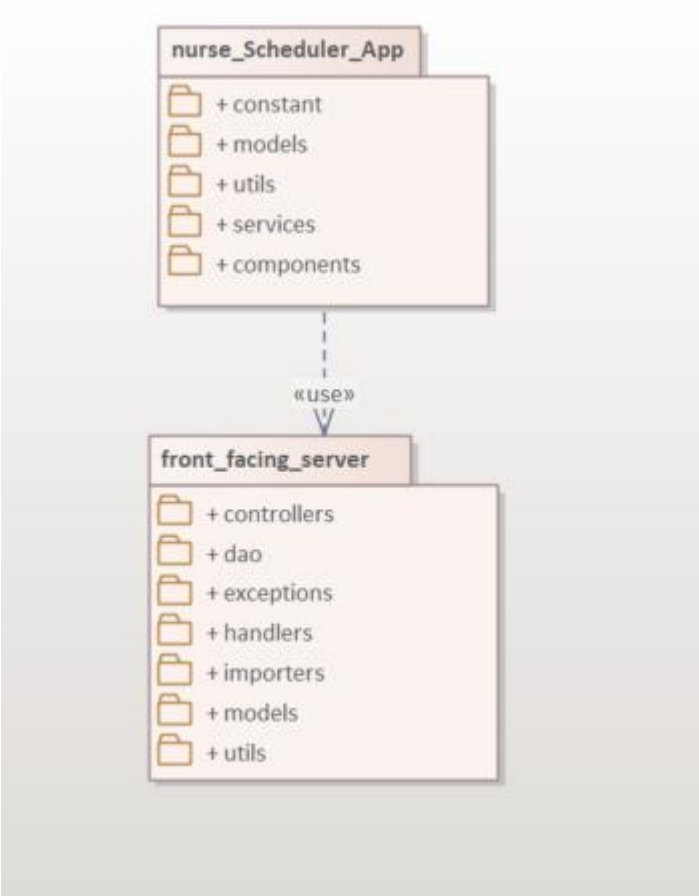

Figure 8: Diagramme de paquetage de haut niveau du système Nurse Scheduler

4.1.1 Diagramme de paquetage de nurse_Scheduler_App

nurse_Scheduler_App
Ce paquetage contient l'ensemble des fichiers et des dossiers qui permettent l'exécution de nurse_Scheduler_App. Il regroupe les fichiers mains, ainsi que les dossiers constant, models, utils, services et components.

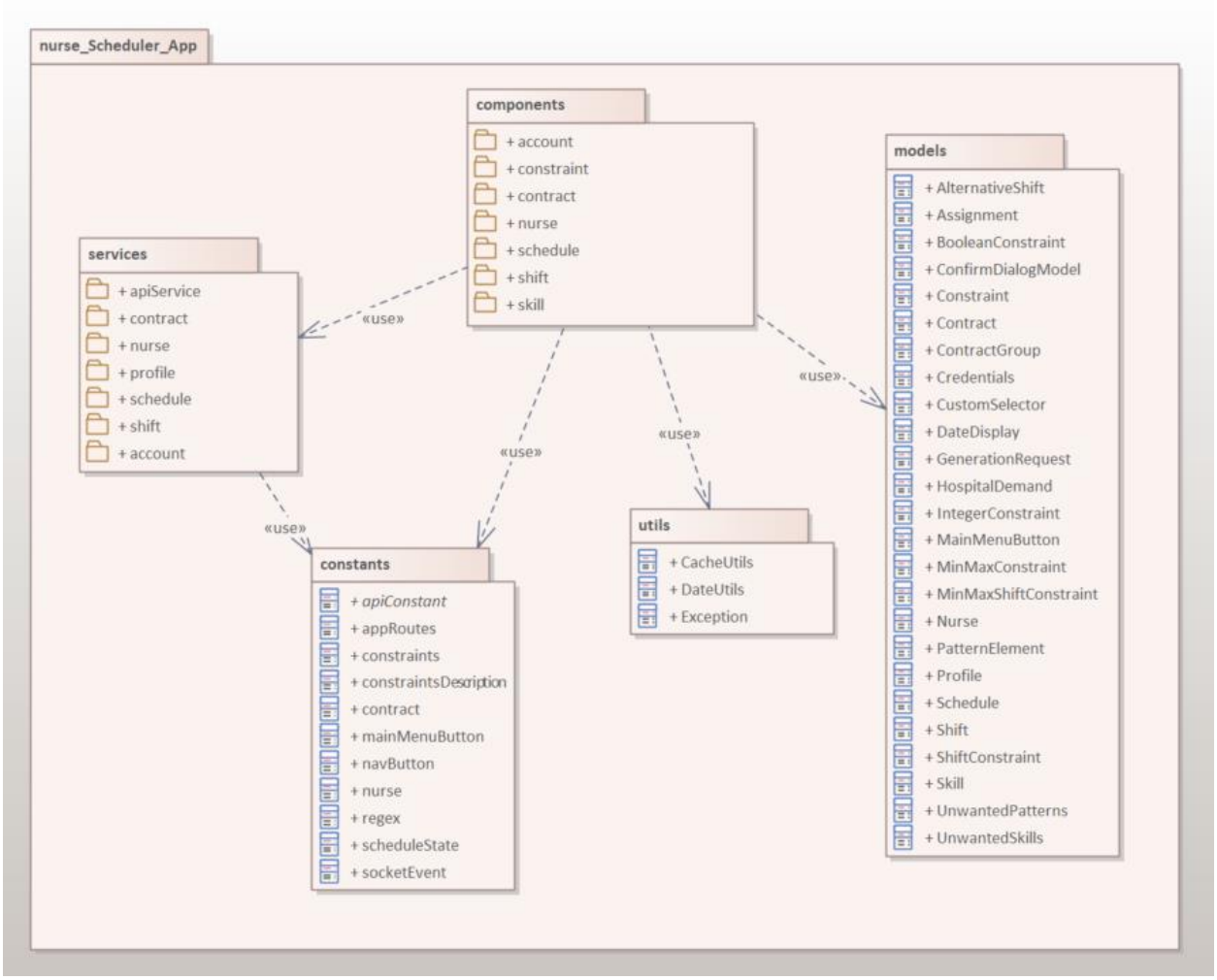

Figure 9: Diagramme de paquetage nurse_Scheduler_App

4.1.1.1 Paquetage components

nurse_Scheduler_App:components
Ce paquetage contient tous les paquetages qui gèrent les interfaces de l'application, soit tout ce qui est affiché à l'utilisateur.Ainsi on retrouve l'ensemble du code générant l'interface pour les contraintes, les contrats, les infirmières, les quarts de travail et les horaires.

nurse_Scheduler_App:components:account
Ce paquetage contient l'ensemble des dossiers reliés à la création d'un compte usager ainsi qu'à l'authentification sur l'application et son affichage.

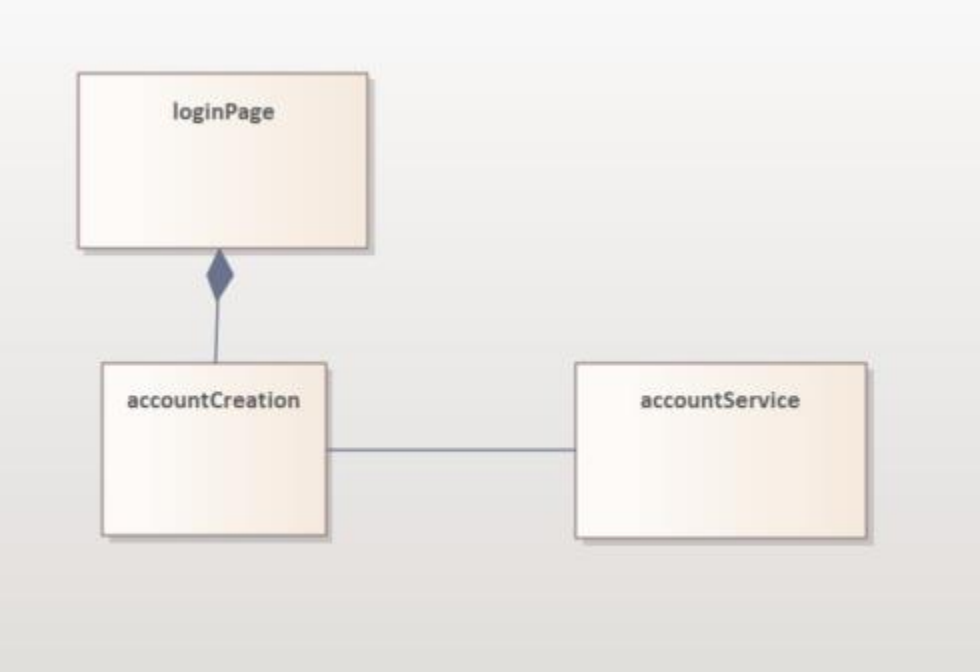

Figure 10: Diagramme de classe du paquetage authentification

nurse_Scheduler_App:components:constraint
Ce paquetage contient l'ensemble des dossiers reliés à la définition des contraintes et leurs affichages.

Figure 10: Diagramme de paquetage constraint

nurse_Scheduler_App:components:contract
Ce paquetage contient le composant qui s'occupe de l'affichage et de la création d'un contrat.

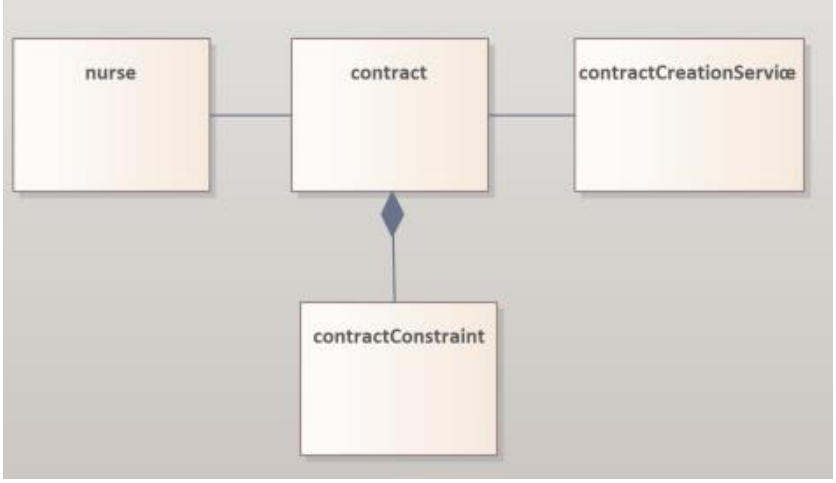

Figure 11: Diagramme de classe du paquetage contract

nurse_Scheduler_App:components:nurse
Ce paquetage contient le composant qui s'occupe de l'affichage d'une infirmière.

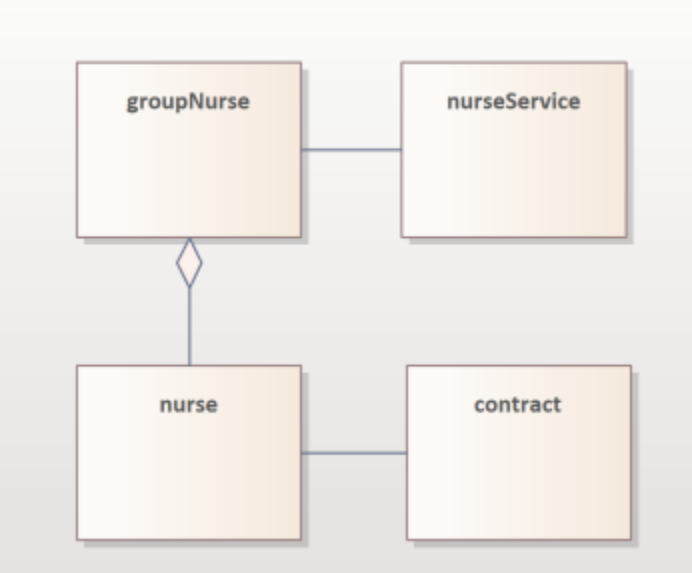

Figure 12: Diagramme de classe du paquetage nurse

nurse_Scheduler_App:components:schedule

Ce paquetage contient le composant qui s'occupe de l'affichage d'un calendrier permettant la sélection de date.

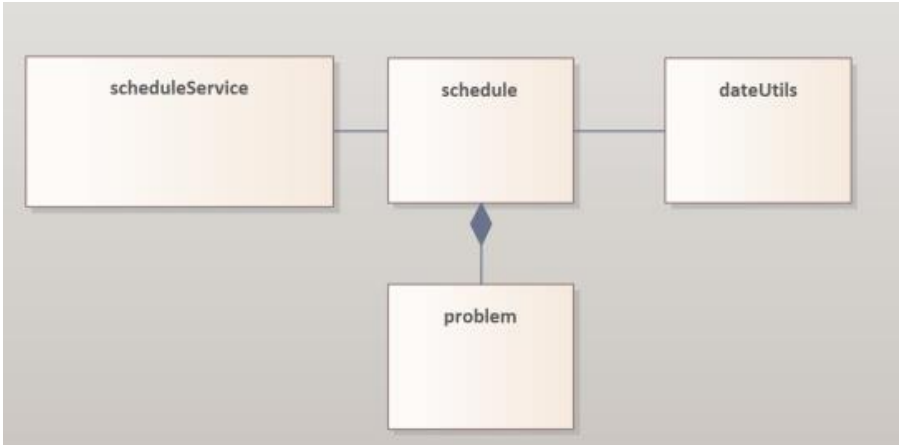

Figure 13: Diagramme de classe du paquetage schedule

nurse_Scheduler_App:components:shift
Ce paquetage contient le composant qui s'occupe de l'affichage des *shift*

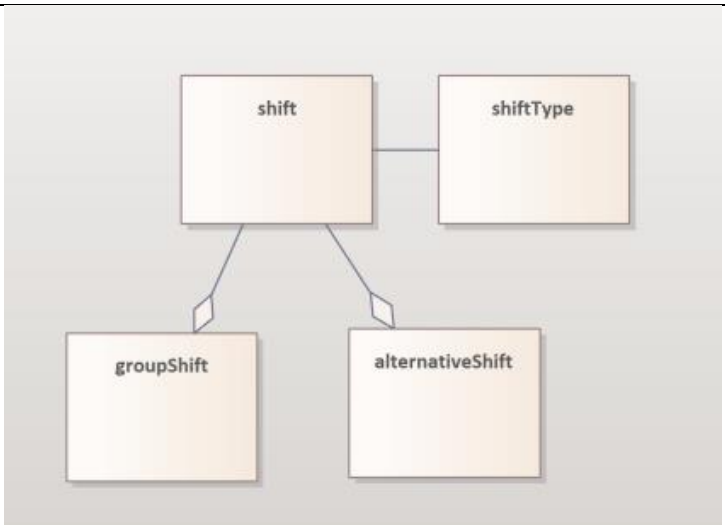

Figure 14: Diagramme de classe du paquetage shift

nurse_Scheduler_App:components:skill
Ce paquetage contient le composant qui s'occupe de l'affichage des *skills*

Figure 14: Diagramme de classe du paquetage skill

4.1.1.2 Paquetage Services

Nurse_Scheduler_App:Services

Ce paquetage contient tous les dossiers de services utilisés par l'application web. Ainsi, ce paquet représente la couche logique de l'application.

Nurse_Scheduler_App:Services:APIService
Ce service s'occupe de la communication avec le front facing server en utilisant le protocole HTTP

Nurse_Scheduler_App:Services:contract
Ce service s'occupe de la communication entre le component contrat et le front facing server en utilisant le protocole HTTP

Nurse_Scheduler_App:Services:nurse
Ce service s'occupe de la communication entre le component nurse avec le front facing server en utilisant le protocole HTTP

Nurse_Scheduler_App:Services:profile
Ce service s'occupe de la communication entre le component profile avec le front facing server en utilisant le protocole HTTP

Nurse_Scheduler_App:Services:schedule
Ce service s'occupe de la communication le component schedule avec le front facing server en utilisant le protocole HTTP

Nurse_Scheduler_App:Services:shift
Ce service s'occupe de la communication entre le component shift avec le front facing server en utilisant le protocole HTTP

Nurse_Scheduler_App:Services:account
Ce service s'occupe de la communication entre le component account avec le front facing server en utilisant le protocole HTTP

4.1.1.3 Paquetage constants

Nurse_Scheduler_App:constants
Ce paquetage contient le fichier *constant* et *enum.*

4.1.1.4 Paquetage utils

Nurse_Scheduler_App:utils
Ce paquetage contient les différents utilitaires par exemple la classe *Dateutils* qui offre une multitude de méthodes pour faciliter la manipulation des dates.

4.1.1.5 Paquetage models

nurse_Scheduler_App:models

Ce paquetage contient les différentes classes et interfaces nécessaires pour assurer la communication entre les *components* et le *front_facing_server*

4.1.2 Diagramme de paquetage du front_facing_server

front_facing_server
Le paquetage du front_facing_server englobe les paquetages suivants nécessaires afin de faire fonctionner le serveur, soit le paquetage controllers, utils, cpp_utils, dao et exceptions

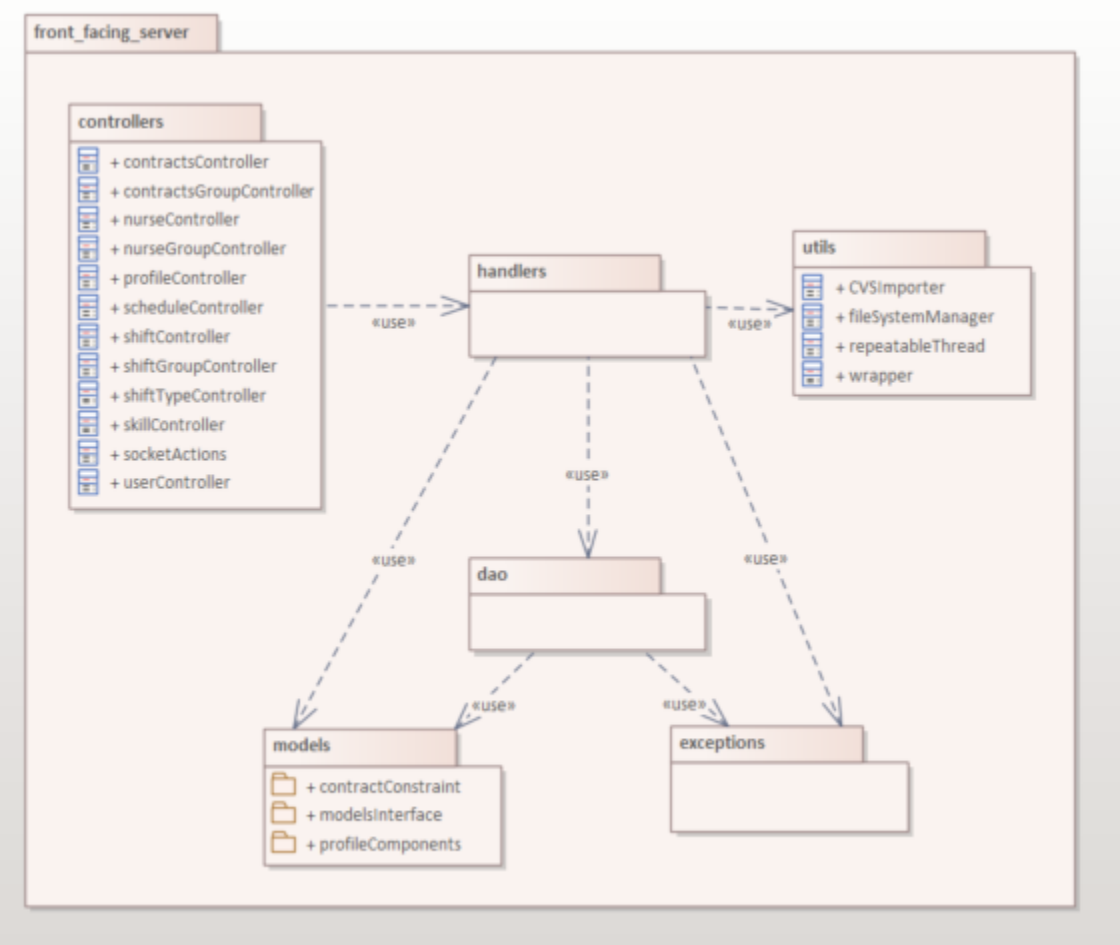

Figure 15: Diagramme de classe du paquetage front_facing_server

4.1.2.1 Paquetage controllers

front_facing_server:controllers
Le paquetage controllers contient des fichiers qui ont les *endpoints* du serveur.

4.1.2.2 Paquetage utils

front_facing_server:utils
Le paquetage utils contient les classes qui seront utilitaires pour le code de serveur, comme le FileSystemManager qui va avoir la logique de la manipulation du volume.

4.1.2.3 Paquetage models

front_facing_server:models
Le paquetage models contient l'ensemble des classes qui sont utilisées lors de la communication avec le Front-End, comme la structure de la demande de génération d'horaire. En plus que les classes qui représente les structures qui vont être sauvegardées dans la base de données.

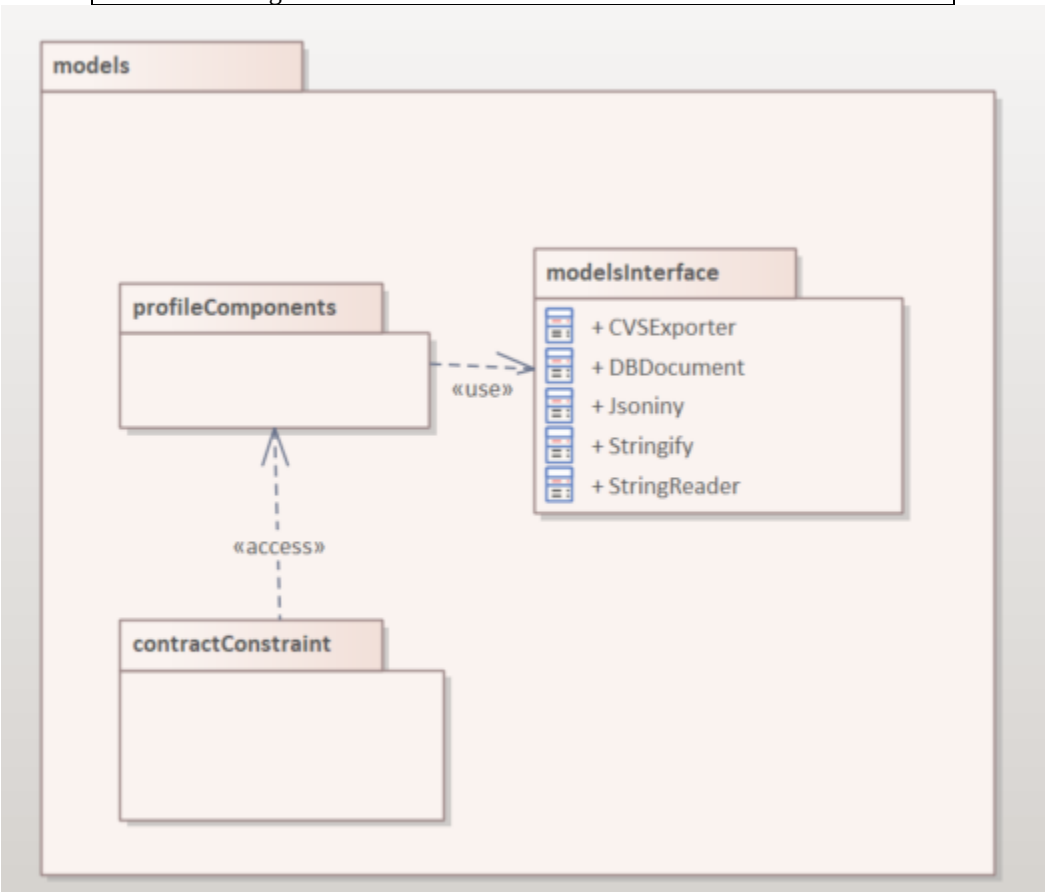

Figure 15: Diagramme de classe du paquetage front_facing_server

front_facing_server:models:ProfileComponents
Ce paquet est un sous paquets de models. Il contient les classes des éléments qui construisent un profil d'hôpital, comme les contrats, les infirmières ainsi que les quarts.Il est composé d'un autre paquet qui contient les contraintes du

contrats.

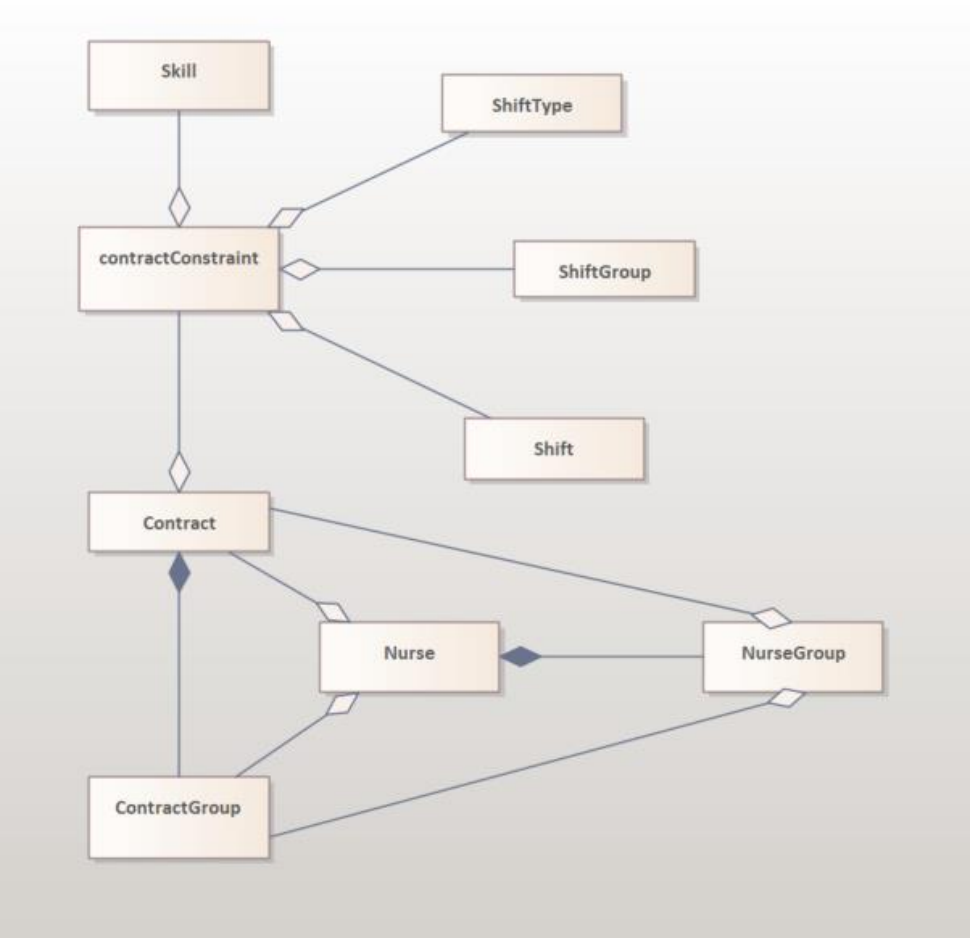

Figure 16: Diagramme de classe du paquetage front_facing_server

front_facing_server:models:ProfileComponents:ContractConstraint
Ce paquet contient les classes des différentes contraintes qui peuvent être dans un contrat

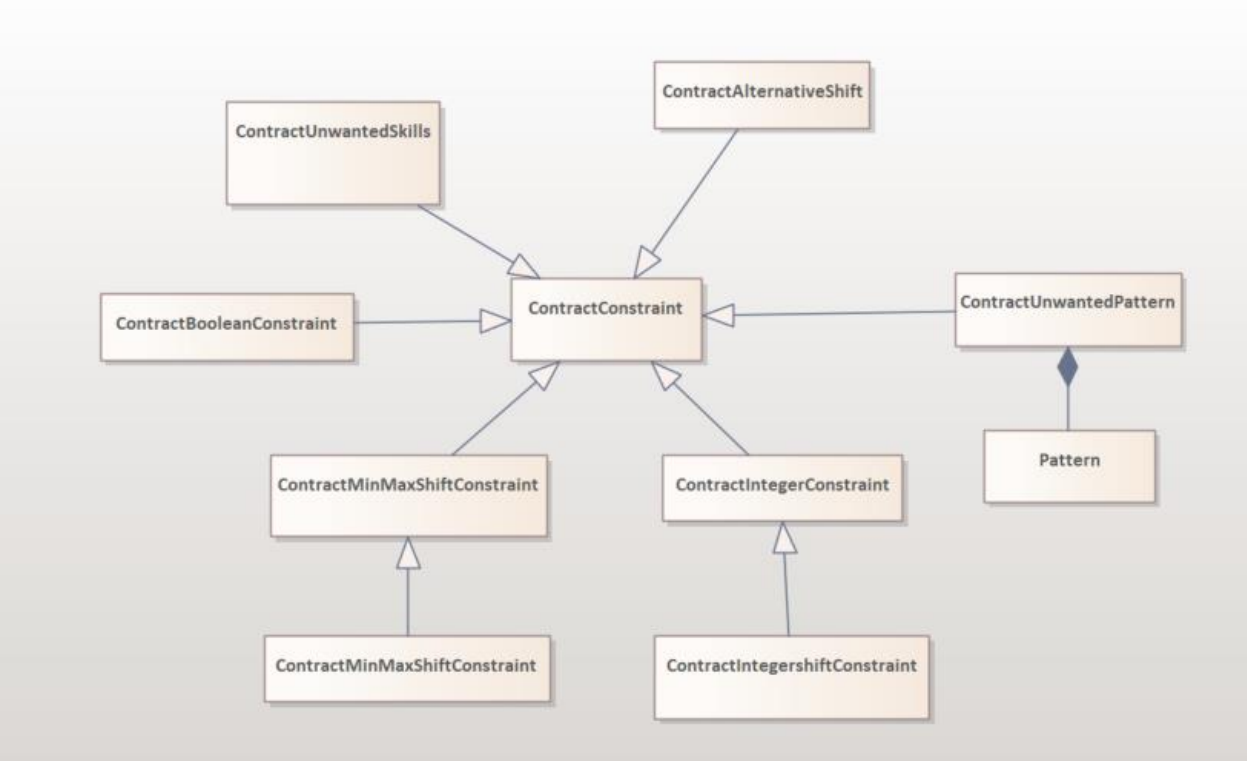

Figure 17: Diagramme de classe du paquetage front_facing_server

4.1.2.4 Paquetage dao

front_facing_server:dao
Ce paquetage contient toutes les classes nécessaires afin de pouvoir communiquer avec les différentes collections de la base de données. Chaque classe de ce paquetage représente une façade des opérations possibles sur une collection de MongoDB.

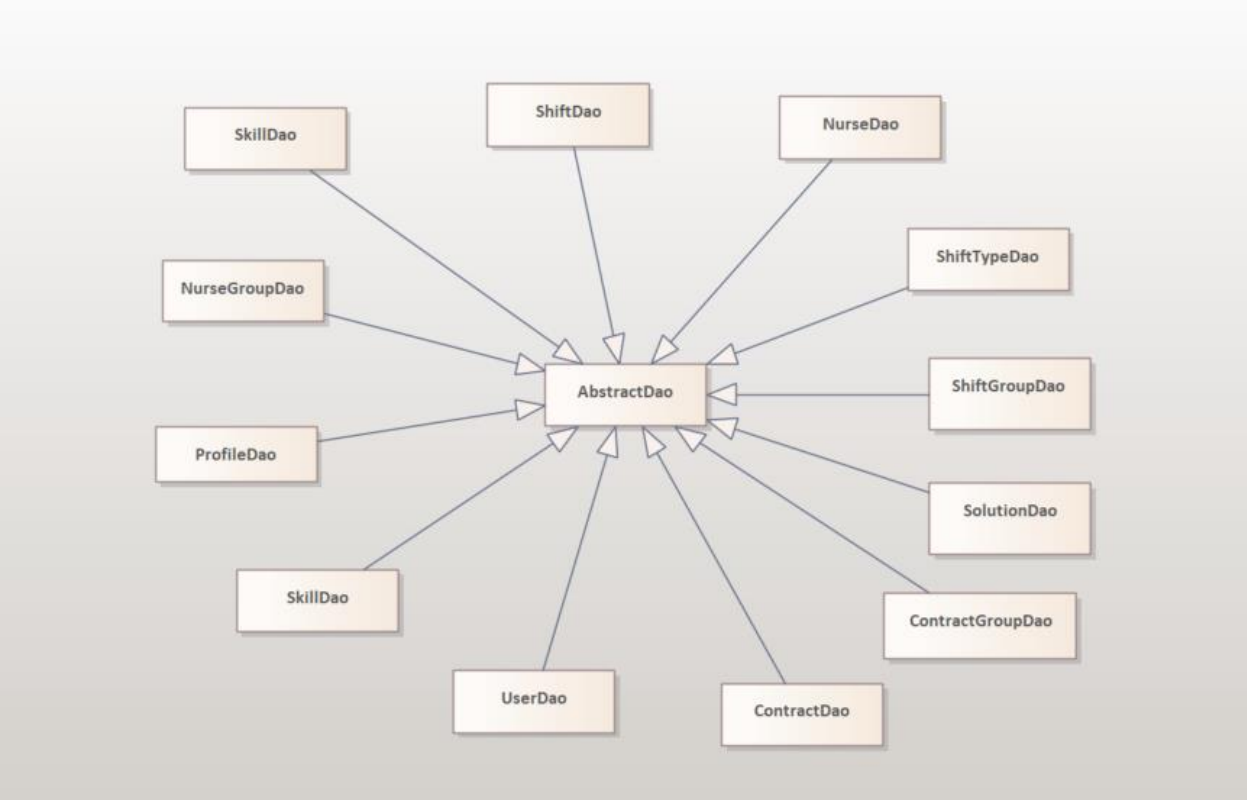

Figure 18: Diagramme de classe du paquetage front_facing_server

4.1.2.5 Paquetage exceptions

front_facing_server:exceptions
Ce paquetage contient les différentes classes des exceptions que le serveur peut lancer.

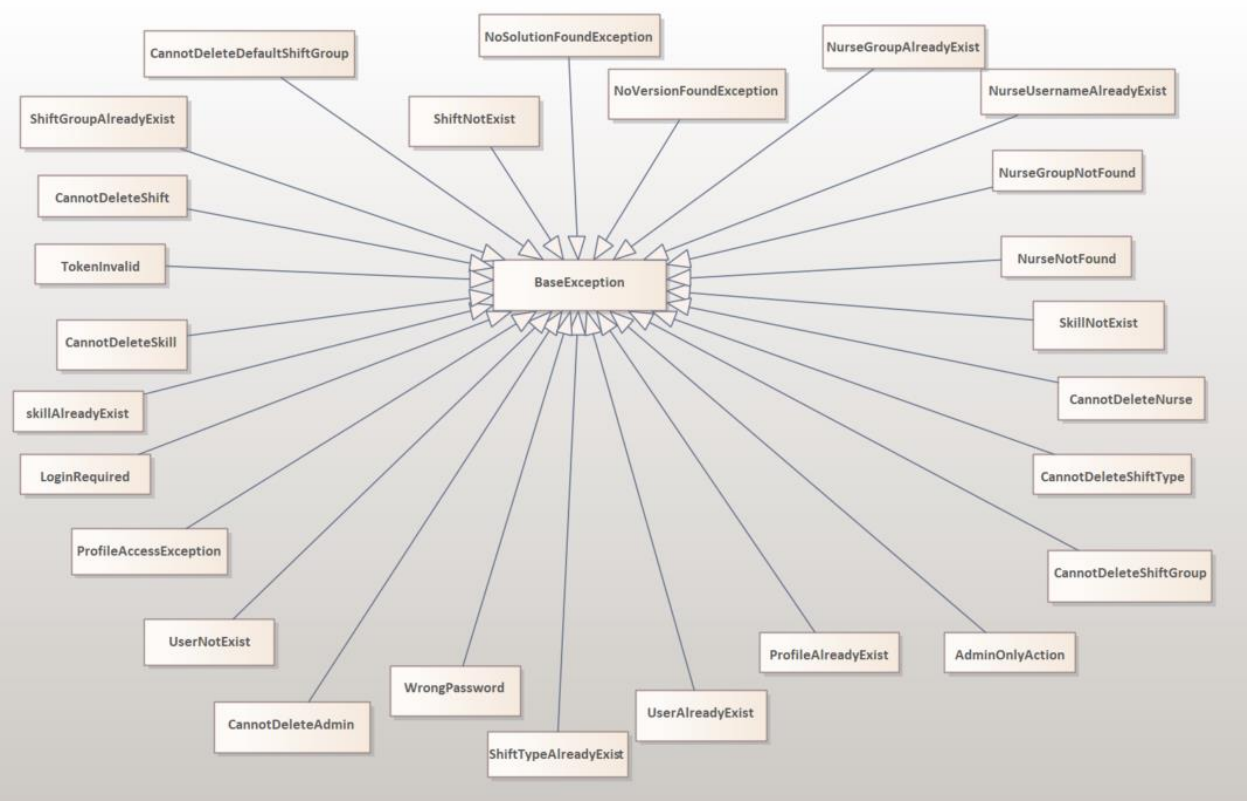

Figure 19: Diagramme de classe du paquetage front_facing_server

4.1.2.5 Paquetage handlers

front_facing_server:handlers
Ce paquet contient les classes de logiques. Chaque contrôleur utilise une seule classe de ce paquet. Le contrôleur extrait les arguments et le corps de la requête et ensuite exécute la méthode appropriée du handler.

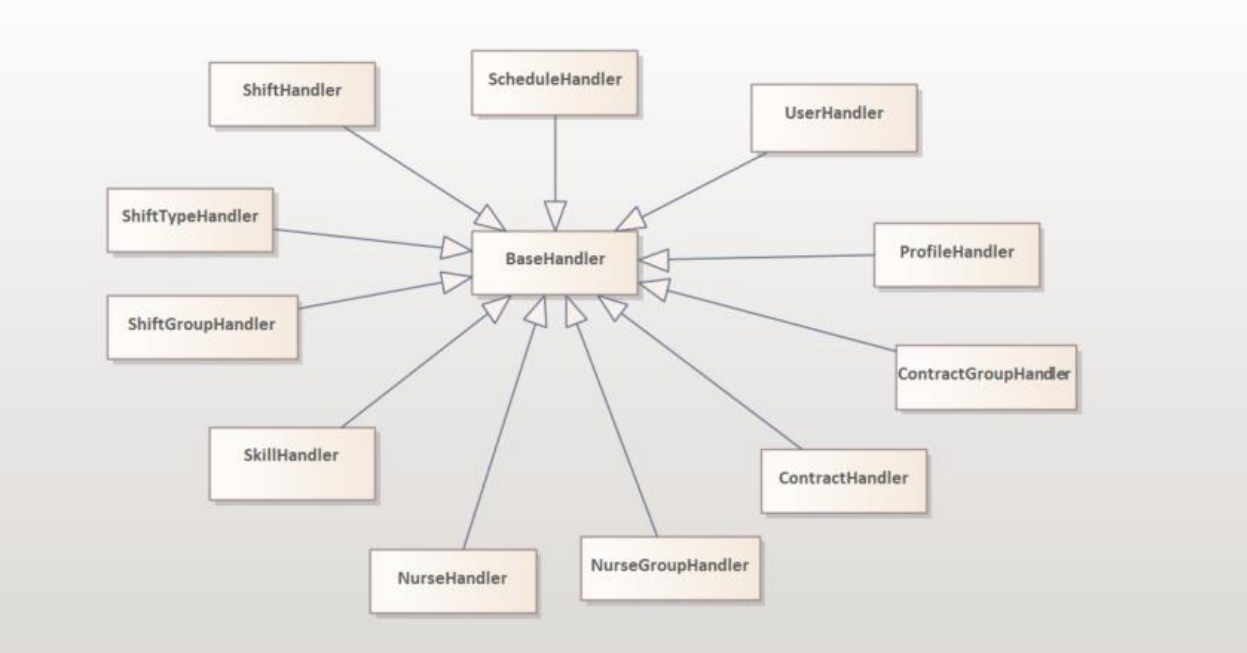

Figure 19: Diagramme de classe du paquetage front_facing_server

5. Vue des processus

5.1 Diagramme de séquence

5.1.1 Diagramme de séquence de la création d'un compte

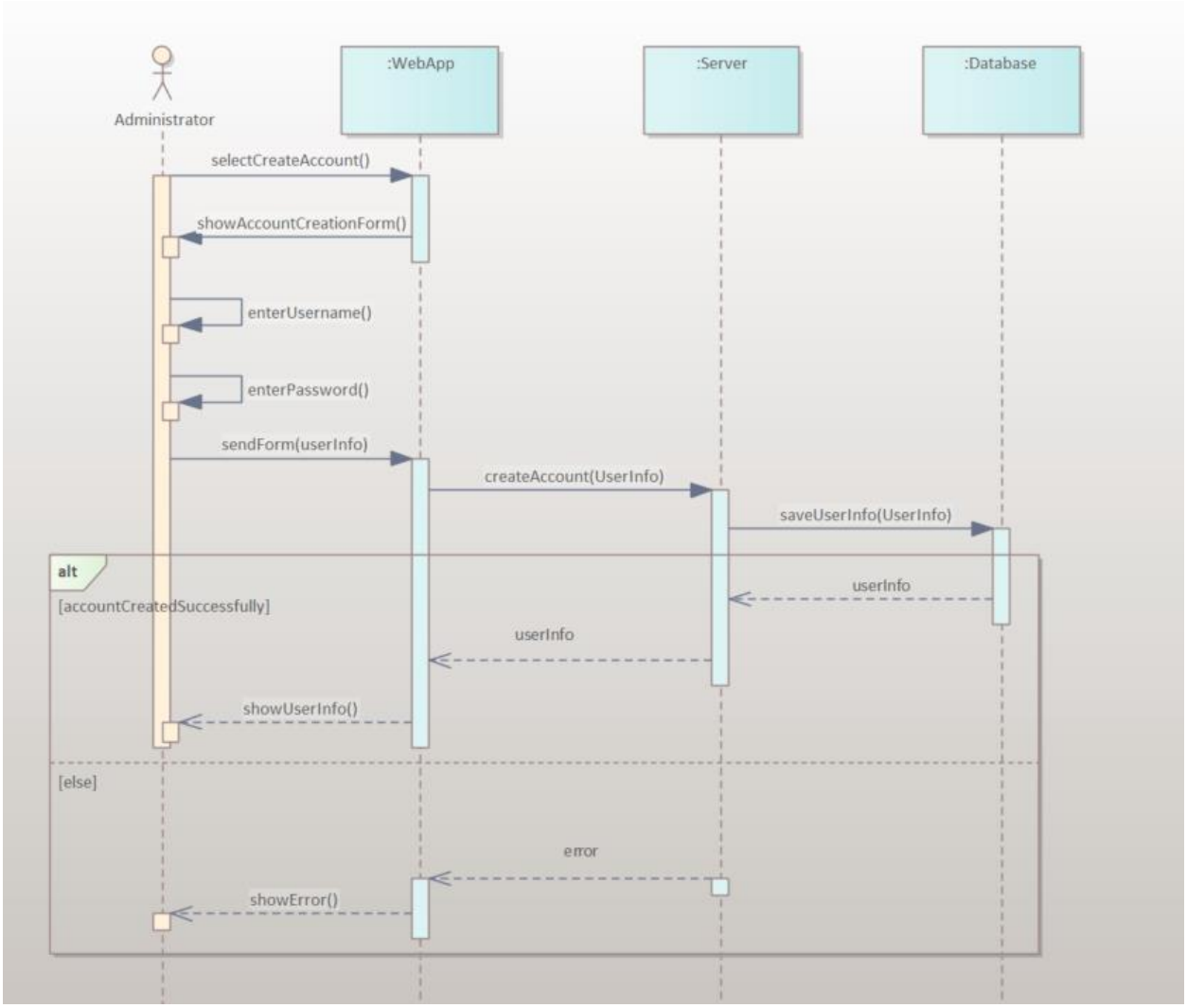

Figure 20 : Diagramme de séquence de la création d'un compte

5.1.2 Diagramme de séquence de la définition d'un shift

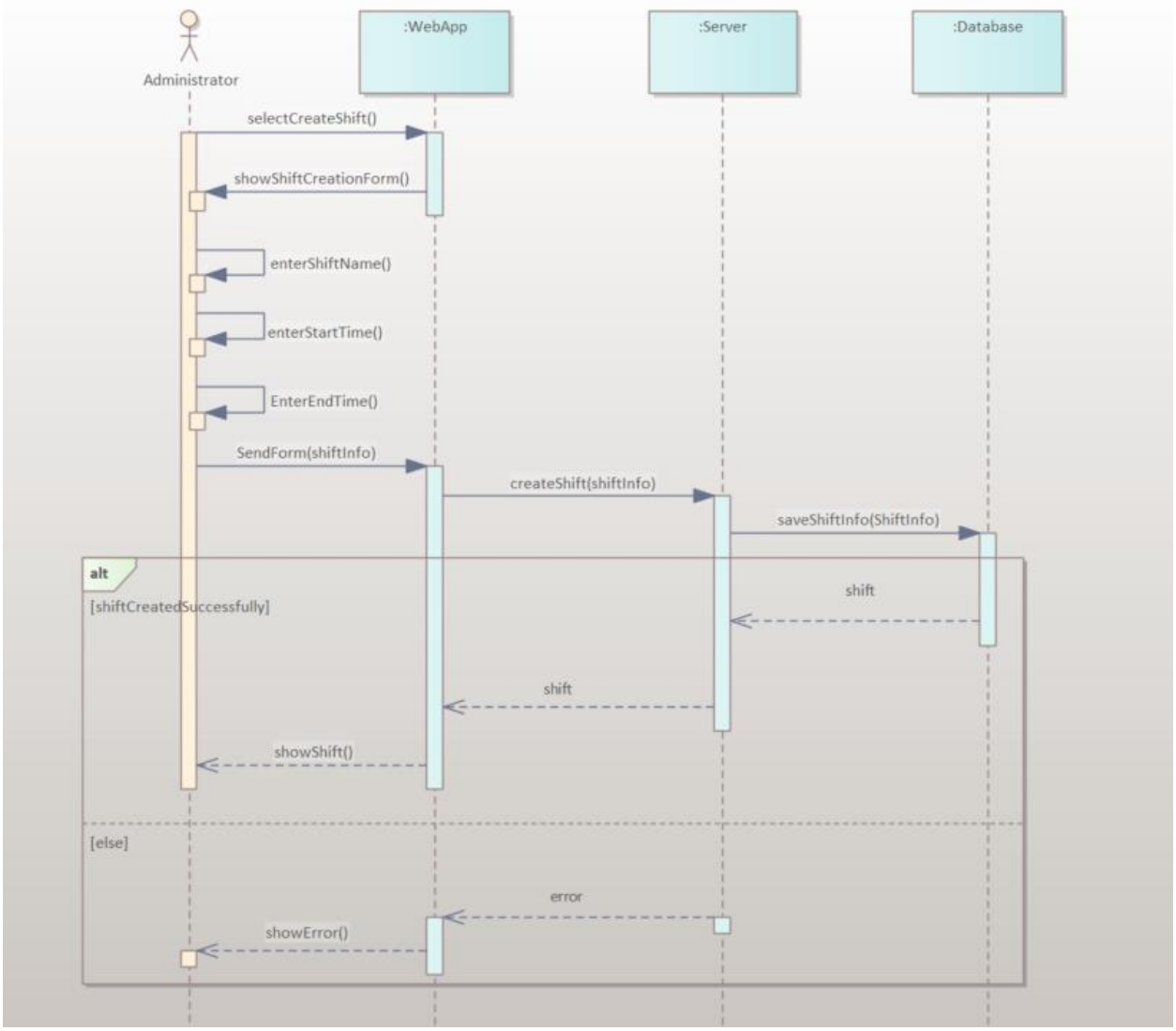

Figure 21 : Diagramme de séquence de la création de la définition d'un shift

5.1.3 Diagramme de séquence de la création d'un contrat

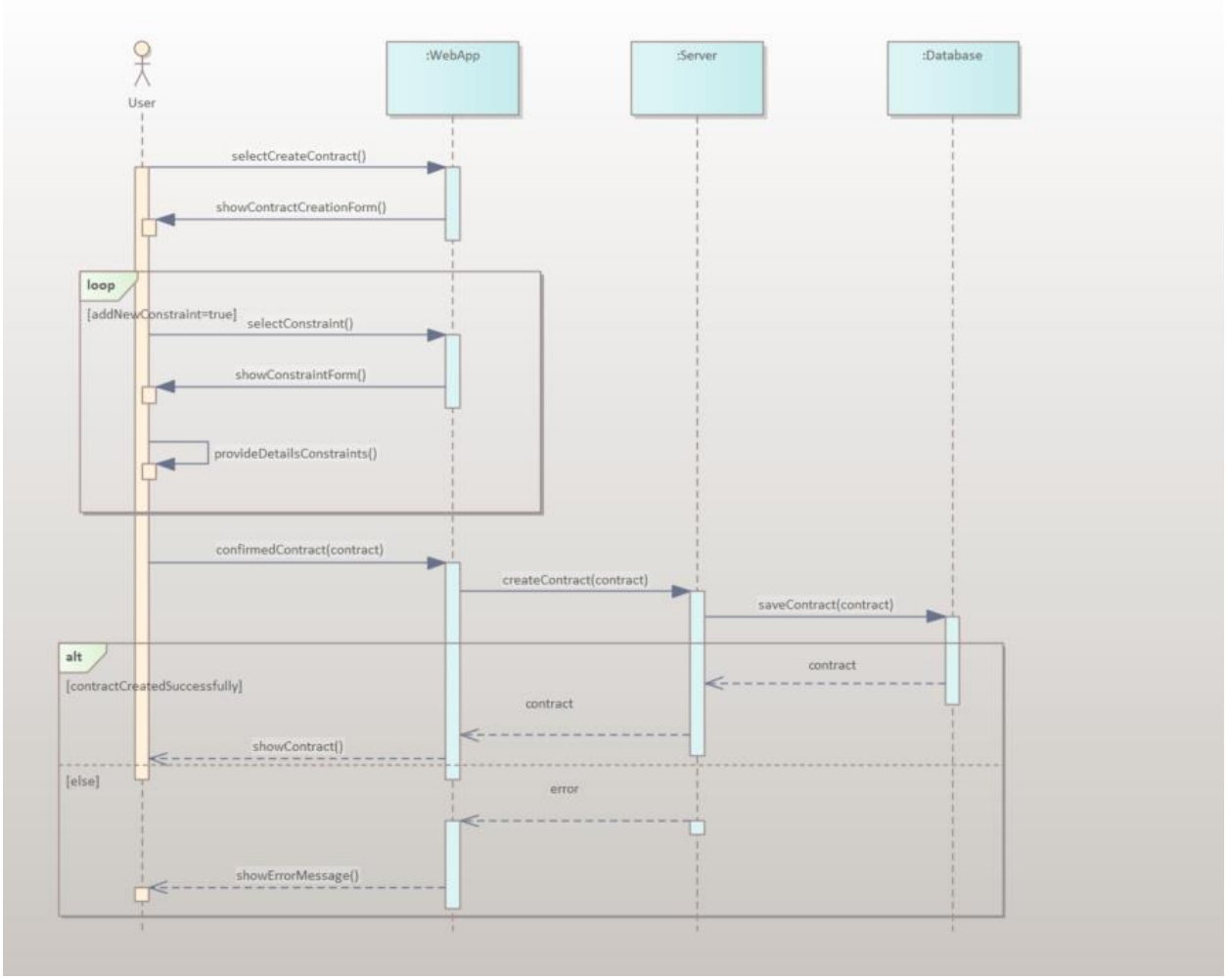

Figure 22 : Diagramme de séquence de la création de la création d'un contrat

5.1.4 Diagramme de séquence de la création d'un dossier d'infirmière

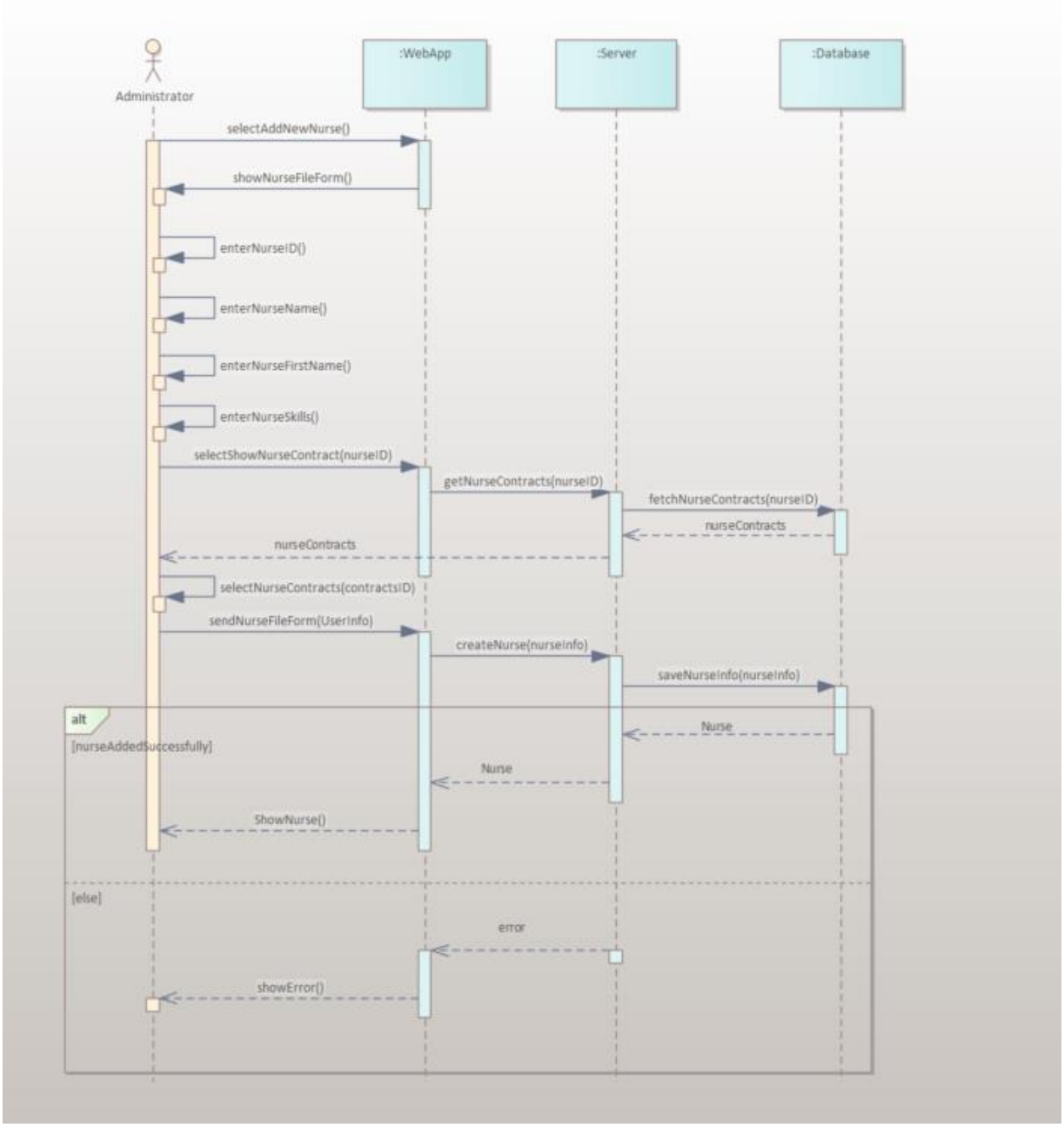

Figure 23 : Diagramme de séquence de la création d'un dossier d'infirmière

5.1.5 Diagramme de séquence de la génération d'un horaire

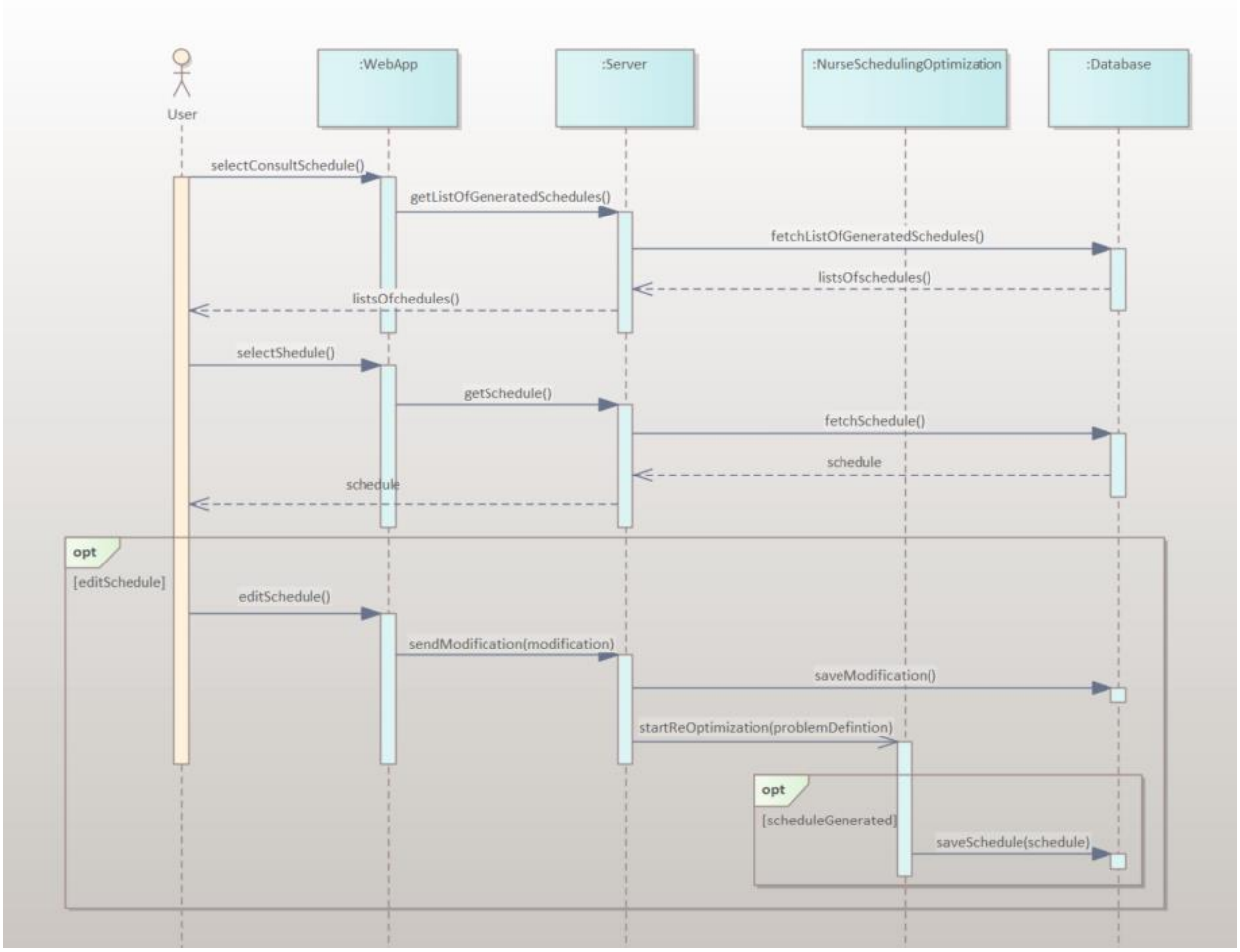

Figure 24 : Diagramme de séquence de la génération d'une horaire

5.1.6 Diagramme de séquence de la visualisation d'un horaire

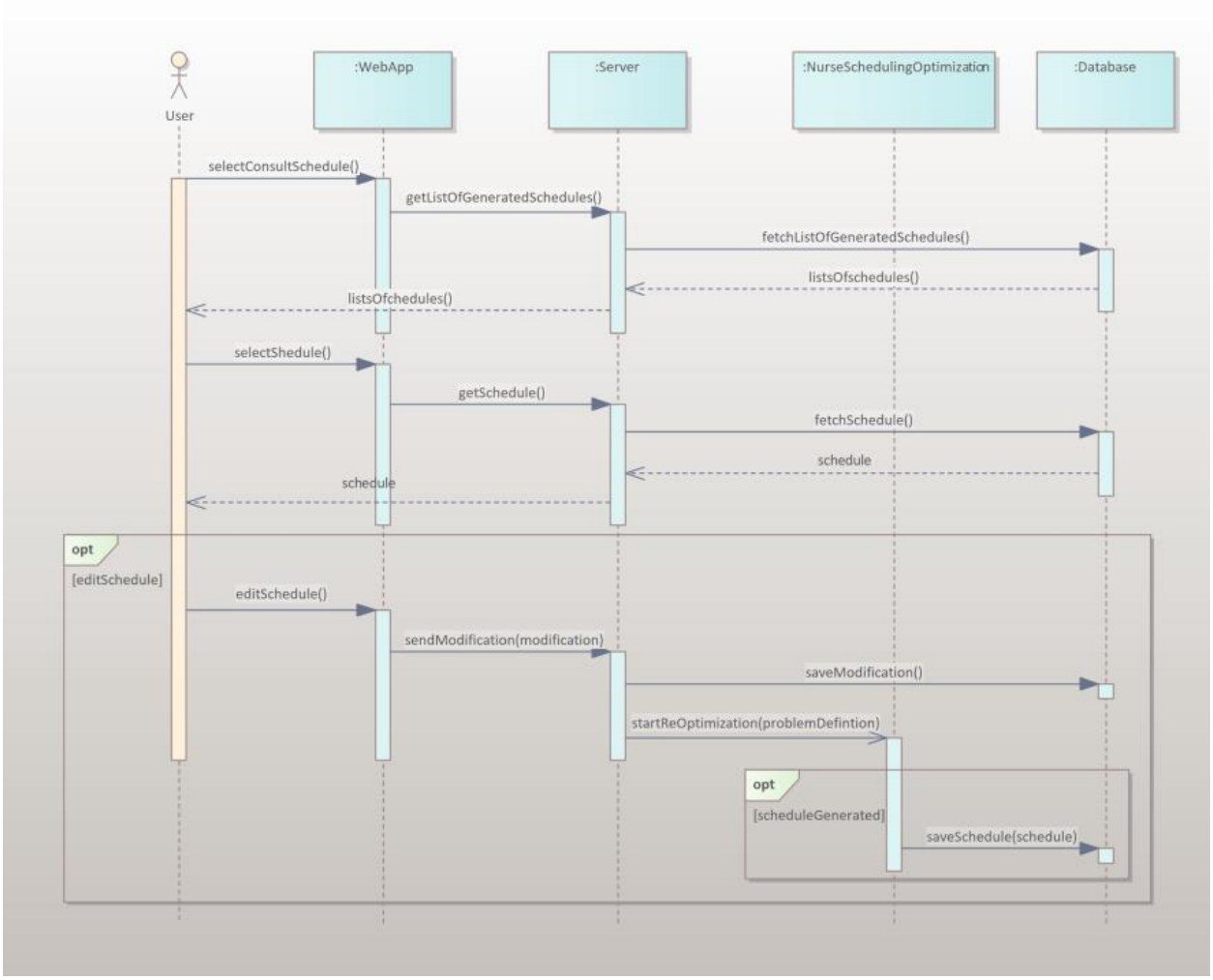

Figure 25 : Diagramme de séquence de la visualisation d'un horaire

6. Vue de déploiement

6.1 Diagramme de déploiement

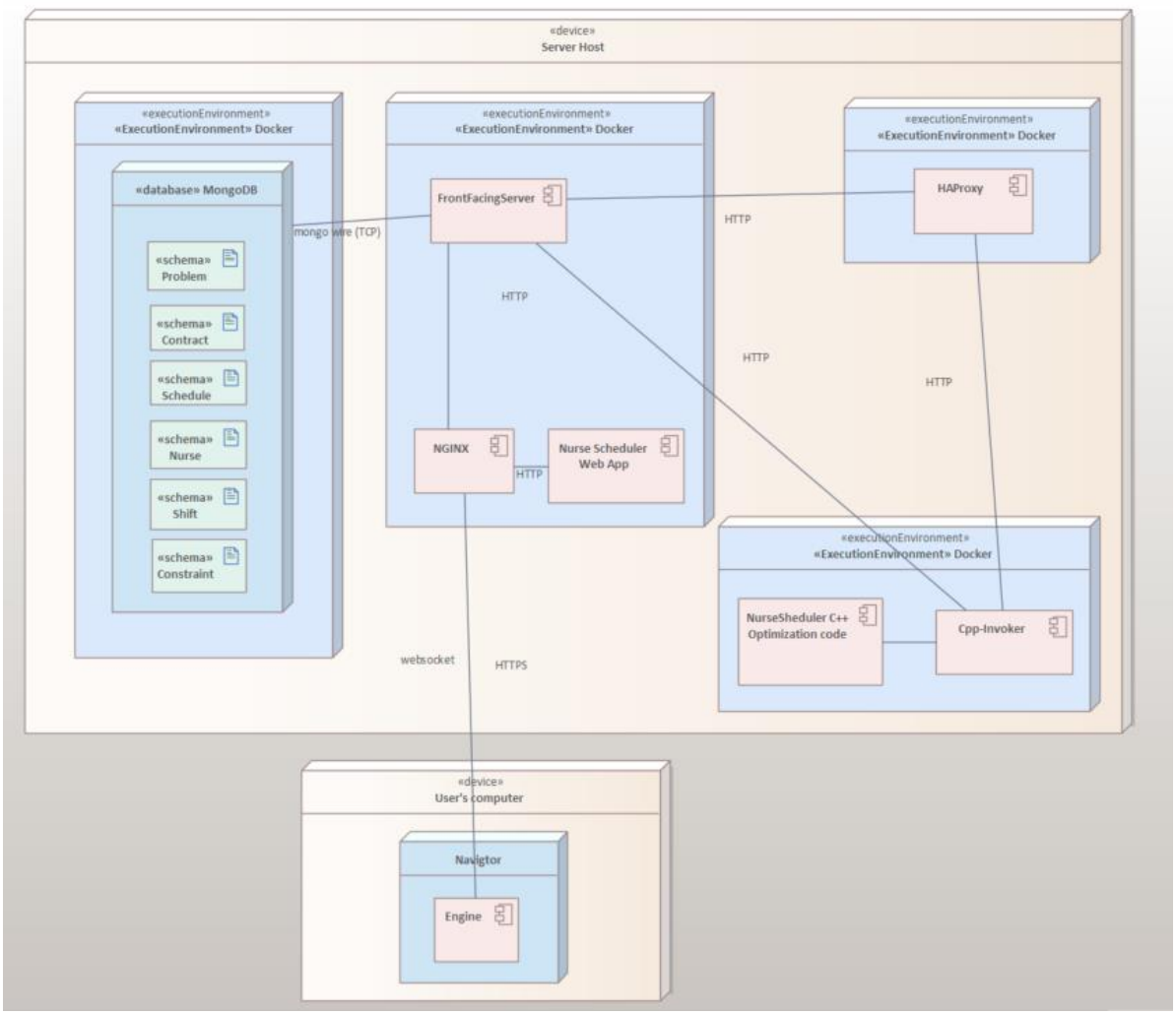

Figure 26 : Diagramme de déploiement

Le diagramme inclut deux blocs de type “device”, un englobe l’ensemble des serveurs et l’autre contient le navigateur de l’utilisateur. Il d’une division logique étant donné que notre application offre la possibilité à l’utilisateur de déployer les serveurs sur un ordinateur et de se connecter à partir d’un autre ordinateur. Il est toujours possible de naviguer vers l’application web à partir de l’ordinateur qui agit comme hôte des serveurs, car un des requis de notre SRS est que l’application tourne localement.

7. Taille et performance

Plusieurs caractéristiques de taille et de performance peuvent avoir un impact sur l'architecture et le design de notre application web. Tout d'abord, notre application roule sur l'ordinateur de l'utilisateur, ainsi la capacité de l'ordinateur, par exemple la mémoire RAM et le disque, devient un facteur limitant. En effet, notre application utilise plusieurs conteneurs Docker ce qui peut consommer beaucoup de mémoire. De plus, la gestion asynchrone de la génération des horaires peut demander beaucoup de ressources de l'ordinateur dépendamment de la complexité du problème en question. Les facteurs énumérés ci-dessus peuvent donc influencer le déploiement de l'application et surtout le nombre des instances du cpp-invoker que nous allons devoir déployé afin de répondre aux besoins de l'application.