Documentation Technique de SoigneMoi

Introduction

Cette documentation technique présente en détail la conception, le développement et le déploiement d'une application de gestion d'hôpital. L'application se compose d'un back-end développé avec Symfony, d'un front-end utilisant HTML/Twig/JS et Tailwind, d'une application mobile en Flutter, et d'une application bureautique en Electron.

Réflexions Initiales Technologiques sur le Sujet

L'objectif principal de cette application est de créer une solution intégrée pour la gestion des séjours hospitaliers. Elle permet aux patients de réserver des séjours avec des docteurs, aux docteurs de suivre les séjours et de laisser des prescriptions et avis, et aux secrétaires de gérer les entrées et sorties des patients. Les technologies sélectionnées pour ce projet incluent Symfony pour le back-end, HTML/Twig/JS et Tailwind CSS pour le front-end, Flutter pour le développement mobile, et Electron pour l'application bureautique.

Symfony a été choisi pour sa robustesse et ses capacités en gestion de grandes bases de données. HTML/Twig/JS et Tailwind CSS permettent de créer des interfaces utilisateur modernes et réactives. Flutter est une plateforme performante pour le développement mobile multiplateformes, tandis qu'Electron offre une solution flexible pour créer des applications de bureau multiplateformes. Docker a été sélectionné pour simplifier le déploiement et la gestion des environnements.

Configuration de l'Environnement de Travail

Prérequis

Assurez-vous d'avoir les éléments suivants installés :

- **Docker** : Pour la conteneurisation des services.
- **Composer** : Pour la gestion des dépendances PHP.
- **PHP** : Version 7.4 ou supérieure.
- **Node.js et npm** : Pour la gestion des dépendances front-end.
- **Symfony CLI** : Pour faciliter le développement avec Symfony.
- **PHPStorm** : IDE recommandé pour le développement PHP.
- **WebStorm** : IDE recommandé pour Electron.
- **AndroidStudio** : IDE recommandé pour développer avec Flutter.

Installation Locale

Installation des dépendances

Pour Linux :

1. **Installer Symfony CLI** :

```bash
curl -sS https://get.symfony.com/cli/installer | bash
export PATH="$HOME/.symfony/bin:$PATH"
```

2. **Installer Node.js et npm** : Suivez les instructions sur le site officiel de Node.js pour installer la version appropriée pour votre système.

Pour Windows :

1. **Installer Symfony CLI** : Téléchargez et exécutez l'installateur depuis Symfony CLI.
2. **Installer Node.js et npm** : Téléchargez et installez depuis le site officiel de Node.js.

Symfony

1. **Démarrer le conteneur de la base de données Docker** :

```bash
docker-compose up -d database
```

2. **Installer les dépendances Composer** :

```bash
composer install
```

3. **Installer les dépendances npm** :

```bash
npm install
```

4. **Créer et exécuter les migrations de la base de données** :

```bash
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

5. **Démarrer le serveur Symfony** : Ouvrez une nouvelle fenêtre/onglet de terminal et exécutez :

Déploiement avec Docker sur un VPS

Pour déployer l'application sur un VPS en utilisant Docker avec Portainer et un proxy inverse, suivez ces étapes :

1. **Configurer Docker et Docker Compose sur votre VPS** : Assurez-vous que Docker et Docker Compose sont installés sur votre serveur. Vous pouvez suivre les instructions officielles de Docker pour votre système d'exploitation.
2. **Installer Portainer** : Portainer est une interface utilisateur pour gérer les conteneurs Docker. Installez Portainer en suivant les instructions officielles. Une fois installé, accédez à l'interface Web de Portainer pour configurer vos conteneurs Docker.
3. **Déployer l'application via Portainer** : Utilisez l'interface Portainer pour créer et gérer vos conteneurs Docker. Vous pouvez utiliser un fichier `docker-compose.yml` pour définir les services nécessaires à votre application, comme la base de données, PHP-FPM, et Nginx.
4. **Configurer un proxy inverse** : Un proxy inverse est utilisé pour diriger les requêtes HTTP vers les conteneurs appropriés. J'ai utilisé NGINX pour réaliser cela. Vous devrez configurer le proxy pour qu'il dirige les requêtes vers votre application Symfony en fonction du nom de domaine ou du chemin URL.
5. **Sécuriser votre déploiement** : Assurez-vous que toutes les communications entre votre proxy inverse et les conteneurs Docker sont sécurisées. Utilisez HTTPS pour sécuriser les communications entre le client et le proxy inverse. Vous pouvez obtenir des certificats SSL gratuits de Let's Encrypt.
6. **Gestion des migrations et des données** : Une fois les conteneurs déployés, vous devez exécuter les migrations de la base de données pour créer les tables nécessaires. Utilisez les commandes Symfony pour créer et exécuter les migrations.

En utilisant Portainer, vous simplifiez la gestion de vos conteneurs Docker et en configurant un proxy inverse, vous assurez que les requêtes sont correctement dirigées vers vos services Docker. Ce processus facilite le déploiement et la maintenance de votre application sur un VPS.

Application Mobile et Bureautique

Pour déployer l'application mobile et bureautique, il vous suffit simplement d'installer node.js avec NPM, flutter et leurs IDE respectifs. Tout le back est gérer par Symfony.
Pour plus d'informations concernant l'installation de ces projets, rendez-vous dans le readme de leur repository respectif.

Explication de mon transaction.sql

Objectif

Le but de cette transaction est d'ajouter une nouvelle spécialité dans la base de données et de mettre à jour la spécialité d'un utilisateur spécifique de manière atomique.

Description des Étapes

1. **Insertion de la Nouvelle Spécialité** :
   - La première requête insère une nouvelle spécialité dans la table `specialty` avec un ID et un nom. Dans cet exemple, la spécialité ajoutée est "Cardiologie".
2. **Mise à Jour de la Spécialité de l'Utilisateur** :
   - La deuxième requête met à jour la spécialité de l'utilisateur avec l'ID 7 pour qu'elle soit la nouvelle spécialité ajoutée (ID 10).
3. **Validation ou Annulation de la Transaction** :
   - Si toutes les requêtes réussissent, la transaction est validée avec `COMMIT`.
   - Si une requête échoue, la transaction est annulée avec `ROLLBACK`, ce qui annule toutes les modifications effectuées par les requêtes précédentes.

```sql
START TRANSACTION;

INSERT INTO specialty (id, name)
VALUES (10, 'Cardiologie');

UPDATE user
SET specialty_id = 10
WHERE id = 7;

COMMIT;

ROLLBACK;
```

Modèle Conceptuel de Données (MCD)

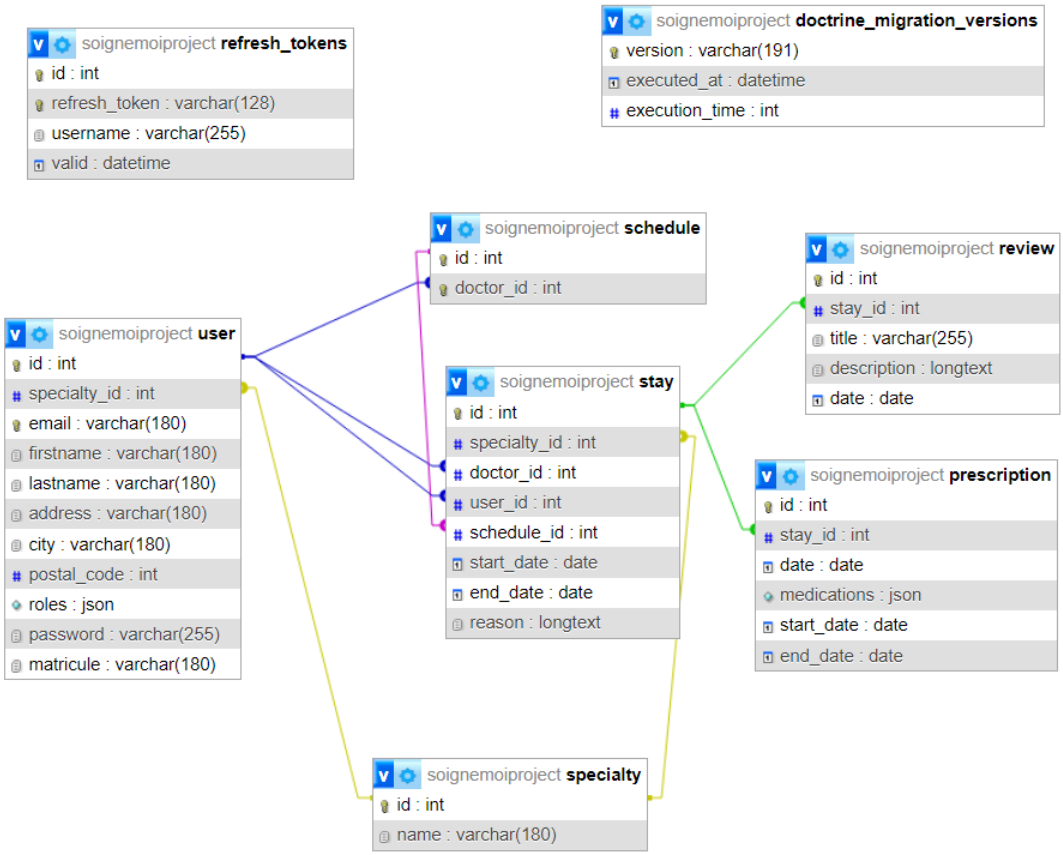
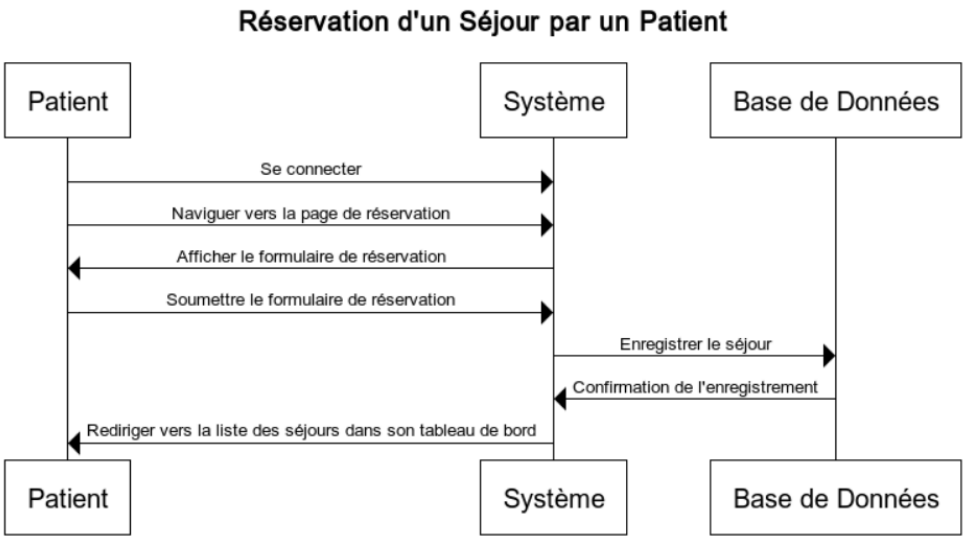


Diagramme d'Utilisation

Exemple d’un Diagramme de Séquence



Plan de Test

Test Unitaire Réalisé

Un test unitaire a été réalisé pour vérifier la fonctionnalité de création d'un séjour. Ce test permet de s'assurer que les objets `User`, `Specialty`, et `Schedule` sont correctement assignés à un séjour et que les dates de début et de fin du séjour sont correctement définies. Voici le code du test unitaire :

```php
<?php

namespace App\Tests\Service;

use App\Entity\Schedule;
use App\Entity\Specialty;
use App\Entity\Stay;
use App\Entity\User;
use PHPUnit\Framework\TestCase;

class StayServiceTest extends TestCase
{
    public function testCreateStay()
    {
        $patient = new User();
        $patient->setId(5); // Assurez-vous que votre entité User a une méthode setId()

        $doctor = new User();
        $doctor->setId(7); // Assurez-vous que votre entité User a une méthode setId()

        $specialty = new Specialty();
        $specialty->setId(2); // Assurez-vous que votre entité Specialty a une méthode se

        $schedule = new Schedule();
        $schedule->setId(1); // Assurez-vous que votre entité Schedule a une méthode setI

        $stay = new Stay();
        $stay->setStartDate(new \DateTime('2024-08-01'));
        $stay->setEndDate(new \DateTime('2024-08-15'));
        $stay->setDoctor($doctor);
        $stay->setUser($patient);
        $stay->setSchedule($schedule);
        $stay->setSpecialty($specialty);
        $stay->setReason('Mal de jambes');

        $this->assertEquals($patient, $stay->getUser());
        $this->assertEquals($doctor, $stay->getDoctor());
        $this->assertEquals('2024-08-01', $stay->getStartDate()->format('Y-m-d'));
        $this->assertEquals('2024-08-15', $stay->getEndDate()->format('Y-m-d'));
        $this->assertEquals($schedule, $stay->getSchedule());
        $this->assertEquals($specialty, $stay->getSpecialty());
        $this->assertEquals('Mal de jambes', $stay->getReason());
    }
}
```

Ce test vérifie que les propriétés d'un séjour sont correctement assignées et récupérées. Il assure que les objets nécessaires (patient, docteur, spécialité, horaire) sont correctement créés et attribués au séjour.