| charge_de_emploi_CMJN |  |  |  |
| --- | --- | --- | --- |
| **Dossier-Projet** | | |
|  | | |
|  |  | ◢ |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| *Nom de naissance* | ▶ | Radouani |
| *Nom d’usage* | ▶ | Radouani |
| *Prénom* | ▶ | Yasmine |
| *Adresse* | ▶ | 14 rue Maurice Dodero, 38100 Grenoble |
|  |  |

| **Titre professionnel visé** | |
| --- | --- |
|  | |
| **Concepteur développeur d’application fullstack - Niveau VI** | |
|  | |

| **Sommaire** | | | |
| --- | --- | --- | --- |
|  | | | |

[**Remerciements**](#_heading=h.gjdgxs) **3**

[**Résumé du projet en anglais**](#_heading=h.30j0zll) **4**

[**Liste des compétences du référentiel**](#_heading=h.1fob9te) **5**

**1. E**[**xpressions des besoins de l’application à développer**](#_heading=h.3znysh7) **6**

1.1 Contexte du projet

1.2 Définition des besoins

**2. Gestion de projet 7**

2.1 Planification et gestion des tâches

2.2 Gestion des versions

3. [**Spécifications fonctionnelles**](#_heading=h.4d34og8) **10**

4. [**Spécifications techniques**](#_heading=h.2s8eyo1) **13**

4.1 *Front-end de l’application*

4.2 Back-end de l’application

5. [**Réalisations**](#_heading=h.17dp8vu) **16**

5.1 Conception

5.2 Maquettes

5.3 Architecture technique du site

5.4 Démarche de la réalisation de l’application

5.5 State Manager

5.6 Sécurité

5.7 Référencement naturel

**6. Présentation d’un jeu d’essai 38**

**7. Description de veille 43**

[**Conclusions**](#_heading=h.3rdcrjn) **48**

| Remerciements |
| --- |
|  |
| Je tiens à remercier chaleureusement toute l'équipe de Simplon ainsi que la région Auvergne Rhône Alpes pour m'avoir permis de suivre avec succès la formation "Concepteur développeur d'applications fullstack".  La méthodologie d'apprentissage par projet de Simplon a parfaitement convenu à ma manière d'apprendre, et cette opportunité a été un véritable tremplin pour mes compétences professionnelles.  Je suis également reconnaissante envers l'entreprise Visions nouvelles, où j'ai eu la chance de réaliser un stage de 5 mois. Mon tuteur a su m'accompagner avec patience et compétence dans le développement de thèmes et de modules WordPress. Cette expérience en entreprise a été une véritable mise en pratique de mes compétences et une occasion d'approfondir mon savoir-faire.  Durant cette année, j'ai eu le privilège de rencontrer des collègues qui partagent le même intérêt pour le domaine et avec qui j'ai tissé des liens. Les projets que nous avons réalisés ensemble nous ont permis de développer des techniques collaboratives ainsi que notre esprit d'équipe.  Je tiens à adresser un merci particulier à mes formateurs, Ali SEKMA et Catherine SCHWARTZ, pour leur encadrement efficace et bienveillant. Leur expertise et leur approche d'apprentissage ont grandement contribué à ma progression.  Cette formation a été une étape marquante de mon parcours professionnel.  Je suis enthousiaste pour la suite de ma carrière, soutenu par les compétences que j'ai acquises et les opportunités qui se profilent.  Merci à tous ceux qui ont participé à cette expérience enrichissante. |
|  |



| Résumé du projet en anglais |
| --- |
|  |
|  |

This project is centered around the development of a web application for online medical appointment scheduling. It utilizes React for front-end, Laravel for back-end, and MySQL for database.

All users have the capability to create accounts, access, and manage their profiles. They can easily search for doctors based on their specialty, name, or location. Additionally, doctors can open available time slots on their calendars.

Patients can book appointments with their preferred doctors and have the flexibility to cancel appointments when necessary. The development plan is divided into four phases, each focused on enhancing the functionality and user experience of the application.

The primary goal of this application is to offer improved accessibility to medical care. It aims to simplify the process of scheduling medical appointments, eliminating geographical and temporal barriers. This is particularly beneficial for patients who may have difficulty traveling or finding suitable appointment times.

Furthermore, the application reduces the risk of errors in appointment scheduling, as digital records minimize the chances of mistakes and simplify patient record management.

Overall, this initiative seeks to enhance the Patient Experience, providing them with the convenience of scheduling appointments online. Patients can select time slots that suit their schedules, thus improving their overall experience with healthcare services.



| Liste des compétences du référentiel |
| --- |
|  |
|  |

Compétences à valider pour l’obtention du titre “Concepteur Développeur d’Applications” :

**1. Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité :**

C1. Maquetter une application

C2. Développer une interface utilisateur de type desktop

C3. Développer des composants d’accès aux données

C4. Développer la partie front-end d’une interface utilisateur web

C5. Développer la partie back-end d’une interface utilisateur web

**2. Concevoir et développer la persistance des données en intégrant les**

**recommandations de sécurité :**

C6. Concevoir une base de données

C7. Mettre en place une base de données

C8. Développer des composants dans le langage d’une base de données

**3. Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité :**

C9. Collaborer à la gestion d’un projet informatique et à l’organisation de l’environnement de développement

C10. Concevoir une application

C11. Développer des composants métiers

C12. Construire une application organisée en couches

C13. Développer une application mobile

C14. Préparer et exécuter les plans de tests d’une application

C15. Préparer et exécuter le déploiement d’une application



| Cahier des charges ou expressions des besoins de l’application à développer |
| --- |
|  |
| ***CareConnect***  **1. E**[**xpressions des besoins de l’application à développer**](#_heading=h.3znysh7) 1.1 Contexte du projetL’objectif de ce projet est de développer une application web qui permettra aux patients de prendre des rendez-vous médicaux en ligne avec des professionnels de la santé. Il permettra à ces professionnels de santé de gérer leur emploi du temps de façon flexible et autonome, et de toucher une large patientèle.  **1.2 Définition des besoins**   * **Offrir une meilleure accessibilité :** l'application vise à faciliter l'accès aux soins en permettant aux patients de prendre des rendez-vous médicaux en ligne. Cela élimine les barrières géographiques et temporelles, ce qui est particulièrement utile pour les patients qui ont du mal à se déplacer ou à trouver un créneau de rendez-vous convenable. * **Flexibilité dans la gestion du planning :** les professionnels de santé bénéficient d’une flexibilité pour gérer leur emploi du temps de manière autonome. Ils peuvent mettre à jour leur disponibilité en temps réel, réduisant ainsi les temps morts et maximisant l'efficacité de leurs consultations. * **Réduction des coûts administratifs :** l'application réduit la charge administrative liée à la prise de rendez-vous. * **Réduction du risque d’erreur dans la prise des rendez-vous :** les enregistrements informatiques réduisent les risques d'erreurs et simplifient la gestion des dossiers des patients. * **Amélioration de l'Expérience Patient** : Les patients apprécient la commodité de la prise de rendez-vous en ligne. Cela leur permet de choisir des créneaux horaires qui conviennent à leur emploi du temps, ce qui améliore leur expérience globale.   ***2. Gestion de projet***  ***2.1 Planification et gestion des tâches***  Pour la planification et la gestion des tâches j’ai utilisé l’outil en ligne **Trello** qui permet de créer des tableaux de bord pour organiser son projet.  Les cartes Trello sont utilisées pour représenter des tâches que l’on peut attribuer à des membres de l'équipe.  On peut définir pour chaque carte une date d'échéance et une description.  J’ai défini 4 colonnes :   * To do : correspond aux tâches qui doivent être effectuées * Doing : correspond aux tâches en cours * Done : correspond aux tâches terminées * To correct : correspond aux tâches à corriger     J’ai également réalisé un **diagramme de Gantt** avec Canva, qui offre une vue d'ensemble du projet, montrant quand chaque tâche commence et se termine. Cela facilite la gestion du temps.  Un diagramme de Gantt peut être mis à jour en fonction des changements survenus pendant le projet. J’ai dû effectivement ajouter une semaine pour le temps de développement et quelques jours pour la conception, donc j’ai mis à jour mon diagramme.    **2.2 Gestion de Versions**  Pour la gestion de versions de mon application j’ai utilisé **GitHub** qui permet que chaque modification soit enregistrée comme un "commit", cela permet de retracer l'historique des modifications et de revenir à des versions précédentes si nécessaire.  Il permet également de stocker son code de façon sécurisé et de permettre à d'autres personnes de le consulter.  Avec Github, il est très intéressant de créer des branches pour chaque fonctionnalité, puis de merger ces branches une fois que leur développement est terminé.  Cette dernière fonctionnalité nous a été très utile durant toute la formation pour travailler en équipe. |



| Spécifications fonctionnelles |
| --- |
|  |

***3. Spécifications fonctionnelles***

***3.1 Spécifications fonctionnelles relatives aux utilisateurs***

1. ***Authentification***

***Inscription***

*Sur la première page de l’application, avant de pouvoir accéder à la page d’accueil, les utilisateurs devront s'inscrire s’ils n’ont pas encore de compte.*

*Pour des raisons de sécurité, les champs à remplir lors de l’inscription sont soumis à des contraintes. Exemple : le mot de passe doit contenir au moins 8 caractères, dont une lettre minuscule, une lettre majuscule, un chiffre et un caractère spécial.*

*Si tous les champs ne sont pas correctement remplis dans le respect des contraintes, le formulaire d’inscription ne peut pas être validé.*

***Connexion***

*Une fois le compte créé, les utilisateurs doivent se connecter à l’aide de leur adresse mail et de leur mot de passe.*

*Pour des raisons de sécurité, seules 3 tentatives de connexion par minute sont autorisées.*

1. ***Consulter son profil :***

*Les utilisateurs peuvent accéder à leur profil en cliquant sur l’onglet avec leur nom/prénom dans le menu de navigation.*

*Ils accèdent à la page de leur compte avec les informations suivantes : Nom, Prénom, numéro de téléphone, email et mot de passe grisé.*

1. ***Modifier son profil :***

*Dans la page de leur compte, les utilisateurs peuvent modifier chacune de leurs informations personnelles.*

*En cliquant sur l’information en question, un fenêtre s’ouvre et leur propose de modifier l’information. Si l’utilisateur entre un champ et valide, le champ est modifié.*

1. ***Déconnexion :***

*En cliquant sur l’onglet déconnexion présent sur la barre de navigation, l’utilisateur se déconnecte et retourne à la page de connexion.*

***3.1 Spécifications fonctionnelles relatives à la prise de rendez-vous***

1. ***Recherche de médecins*** *:*

*Sur la page d’accueil, les utilisateurs pourront rechercher des professionnels de santé par spécialité, localisation et/ou nom.*

*Deux champs de recherche sont présents : Nom/spécialité et localisation.*

*Ils sont aussi soumis à des contraintes par souci de sécurité.*

*Une fois la recherche lancée, une liste de résultats s’affiche sur la même page.*

*Chaque résultat contient le nom/prénom du professionnel, l’adresse du cabinet, la date et l’heure du prochain rendez-vous disponible et un bouton cliquable pour prendre rendez-vous avec ce professionnel.*

1. ***Prise de rendez-vous :***

*Quand un patient clique sur le bouton prendre rendez-vous (mentionné dans la fonctionnalité précédente), toutes les disponibilités du professionnel s’affiche.*

*S’il clique ensuite sur un des créneaux, une fenêtre s’ouvre pour lui proposer de réserver ce rendez-vous.*

*S’il clique sur “Oui”, le créneau est réservé pour cet utilisateur, et une pop-up l’informe que le rendez-vous a été pris avec succès. La page d’accueil se ré affiche sans le résultat de la recherche.*

*S’il clique sur “Annuler”, la fenêtre se ferme.*

1. ***Affichage des rendez-vous des patients*** *:*

*Dans l’onglet “Mes rendez-vous”, les patients peuvent consulter leurs rendez-vous à venir, avec pour chaque rendez-vous, des informations sur le professionnel de santé, sur la date, l’heure et l’adresse du cabinet.*

1. ***Calendrier du médecin :***

*Les professionnels de santé ayant a priori plus de rendez-vous que les patients, ont un affichage de leur rendez-vous sous la forme d’un calendrier hebdomadaire.*

*En bleu sont affichés les créneaux passés ou non disponibles et en vert les créneaux ouverts et disponibles (pas encore réservés par un patient).*

*Il est possible grâce aux flèches de navigation de consulter la semaine précédente et la semaine suivante.*

1. ***Ouvrir un nouveau créneau***

*Les professionnels peuvent ouvrir des créneaux horaires en fonction de leurs disponibilités.*

*Dans la section “Mes rendez-vous”, un bouton “Ouvrir un nouveau créneau” permet l’ouverture d’une fenêtre. Dans cette dernière, le professionnel de santé doit entrer :*

* *La date du créneau qu’il souhaite ouvrir*
* *L’heure du début du créneau*
* *L’heure de la fin du créneau*
* *La durée des rendez-vous en minute*

*Les champs sont encore une fois soumis à des contraintes pour des raisons de logique et des raisons de sécurité.*

*Pour des raisons logiques, la date saisie doit être antérieure à la date actuelle, le créneau horaire des rendez-vous ne doit pas être déjà ouvert par ce médecin et la plage horaire du créneau doit être divisible par la durée des rendez-vous.*

*Si toutes les contraintes sont validées, le professionnel peut “valider” mais il peut également à tout moment annuler la démarche.*

*Lorsqu’il valide, un pop-up confirme que le créneau a bien été ouvert et les rendez-vous s’affichent sur le planning.*

1. ***Supprimer un rendez-vous :***

*Les utilisateurs peuvent supprimer un rendez-vous en cliquant dessus, une fenêtre s’ouvre pour leur proposer de supprimer le rendez-vous.*

*S’ils cliquent sur “Oui, annuler”, le rendez-vous est supprimé lorsqu’il s’agit d’un professionnel de santé par contre il devient simplement disponible si c’est un patient qui le supprime.*

*S’ils cliquent sur “Non, conserver”, la fenêtre se ferme.*

1. ***Réception d’un email de confirmation***

*Dès lors qu’un patient réserve un rendez-vous avec un professionnel de santé, il reçoit un email qui lui confirme la réservation*



| Spécifications techniques |
| --- |
|  |

## *4. Spécifications techniques*

### *4.1 Front-end de l’application*

***Javascript/React :***

*J’ai utilisé le framework React de javascript pour la construction de l'interface utilisateur.*

*Pour éviter la redondance de code et pour faciliter la maintenance et l’évolution de l’application, j’ai mis en place des composants réutilisables.*

***Packages :***

*J’ai intégré plusieurs bibliothèques javascript via npm :*

* *Redux pour la gestion d'état,*
* *Axios pour les requêtes HTTP,*
* *React big calendar pour l’affichage de l’emploi du temps du professionnel de santé.*
* *SweetAlert pour la génération des pop-up*
* *Jwt\_decode pour la sécurité*
* *React-helmet pour le SEO*

*Pour la gestion de l'état de l’application, j’ai donc utilisé redux qui permet une mise à jour efficace et rapide de l'interface utilisateur sans avoir besoin de faire de nouvelles requêtes à la base de données ou de recharger complètement la page.*

***SASS :***

*Pour le style j’ai utilisé SASS essentiellement pour sa fonction de nesting que j'apprécie particulièrement pour une meilleure lisibilité et compréhension de la structure du code.*

### *4.2 Back-end de l’application*

***PHP/Laravel :***

*J’ai développé une API RESTful avec Laravel pour le back-end.*

*Laravel est un framework PHP qui repose sur l’architecture MVC. Il offre une structure organisée pour gérer les routes, les contrôleurs, les modèles, les migrations de base de données, la sécurité, l'authentification.*

*J’ai utilisé l’ORM Eloquent de Laravel qui simplifie les requêtes à la base de données.*

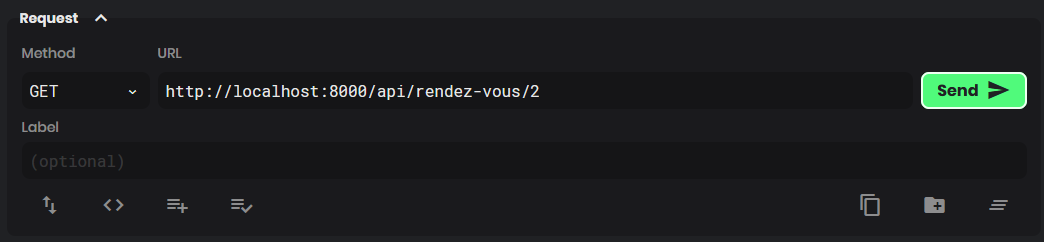
*Dès la connexion d’un utilisateur à la base de données, un token lui est attribué durant tout le temps de sa navigation.*

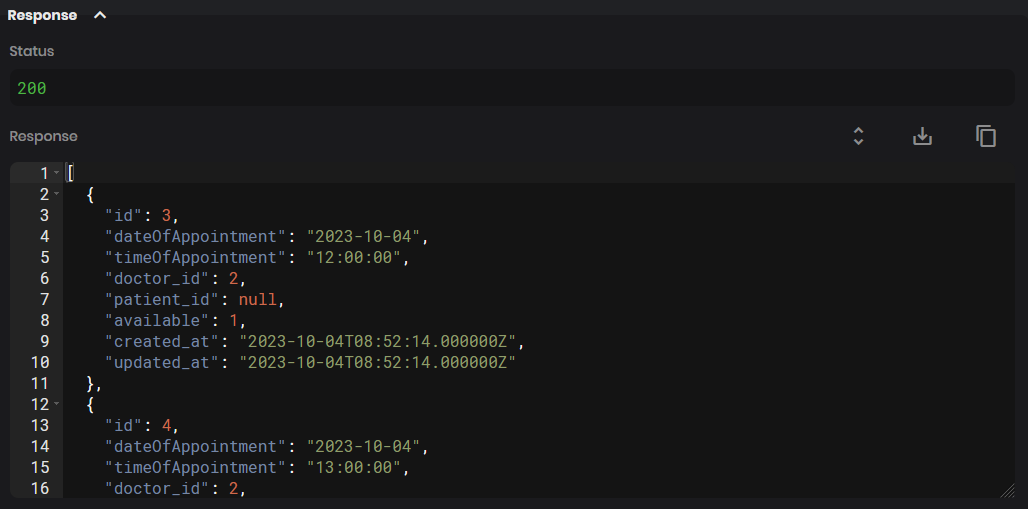
*Pour la sécurité des routes sensibles, j’ai ajouté une authentification avec JWTAuth, qui oblige l’utilisation du token pour accéder à certaines routes de l’API (comme la suppression d’un rendez-vous par exemple). Il permet aussi l’envoi des informations de l’utilisateur dans le token.*

***Postwoman :***

*Comme j’ai développé le back-end avant le front-end, je me suis servi de Postwoman pour tester les requêtes de mon API et corriger d’éventuelles erreurs.*

*Cette application web open source et gratuite permet d'envoyer des requêtes HTTP à l’API en spécifiant la méthode (GET, POST, PUT, DELETE, etc.), les en-têtes, les paramètres, et le corps de la requête.*

****

****

### *Base de données*

Pour la base de données, j’ai choisi le SGBDR MySQL pour stocker les informations relatives aux utilisateurs, aux rendez-vous, aux médecins et aux patients.

L’intérêt d’une base de données relationnelle est la présence d’une structure tabulaire qui facilite l'organisation et la gestion des données.

Les tables, les colonnes et les lignes permettent de stocker des informations de manière structurée.

Elle garantit l'intégrité des données grâce aux contraintes comme les clés primaires, les clés étrangères et les vérifications de données. Cela permet une cohérence des données.

J’ai mis en place 4 tables, conformément au diagramme de classe réalisé pour l’application: users, doctors, patients et appointments.

Les médecins et patients héritent de la classe user et les tables sont reliées grâce à l’utilisation du user\_id comme clé étrangère.

D’autres tables sont créées automatiquement par Laravel :

* La table **“**[**personal\_access\_tokens**](http://localhost:81/phpMyAdmin/index.php?route=/sql&db=careconnect&table=personal_access_tokens&pos=0)**”** stocke les token générés lors de chaque connexion
* La table **“password\_resets”** sert à gérer la réinitialisation des mots de passe des utilisateurs de l’application (elle stocke des informations temporaires permettant de valider et d'acheminer les demandes de réinitialisation de mot de passe).
* La table **“migrations”** garde une trace de toutes les modifications apportées au schéma de la base de données de votre application Laravel.

**Tests**

J’ai utilisé PhpUnit pour réaliser des tests unitaires sur les contrôleurs.

C’est un framework de test unitaire pour PHP très utilisé dans les projets basés sur le framework Laravel.

Les tests unitaires vérifient si chaque unité de l’application fonctionne correctement.

J’ai réalisé peu de tests par manque de temps, mais j’ai compris l’intérêt et le principe de ces tests.



| Réalisations |
| --- |
|  |
|  |

**5. Réalisations**

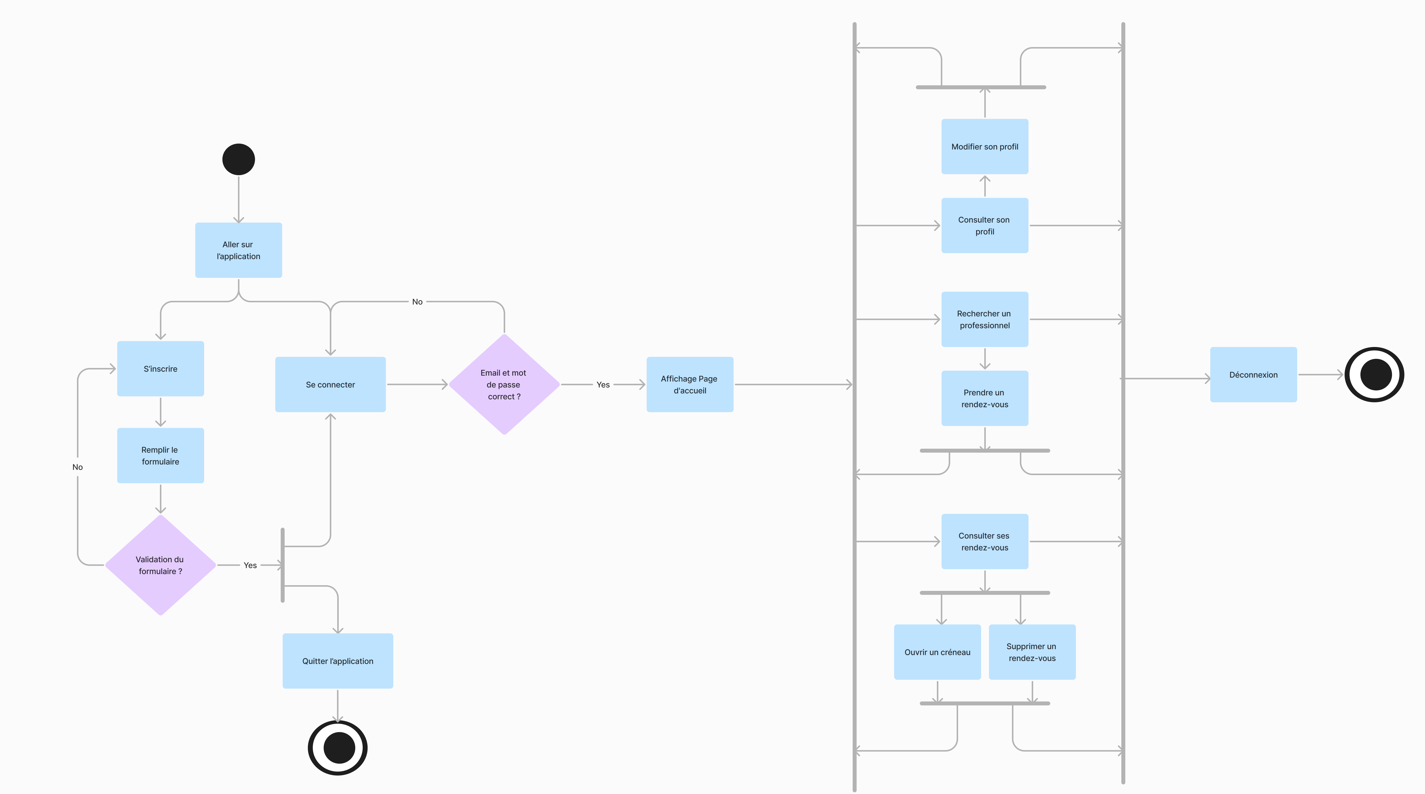
**5.1 Conception**

J’ai réalisé la conception de l'application afin de définir clairement les objectifs, les fonctionnalités, et l'architecture de l'application.

Concevoir permet d’identifier les problèmes potentiels, les lacunes de conception et les erreurs conceptuelles avant de commencer le développement réel.

J’ai réalisé un **diagramme d'activité** (LucidChart) qui représente le comportement de l’application. Il montre comment les composants interagissent pour réaliser une fonction spécifique.

<https://github.com/Yradouani/CareConnect/blob/main/Conception/CareConnect_activity_diagram.png>

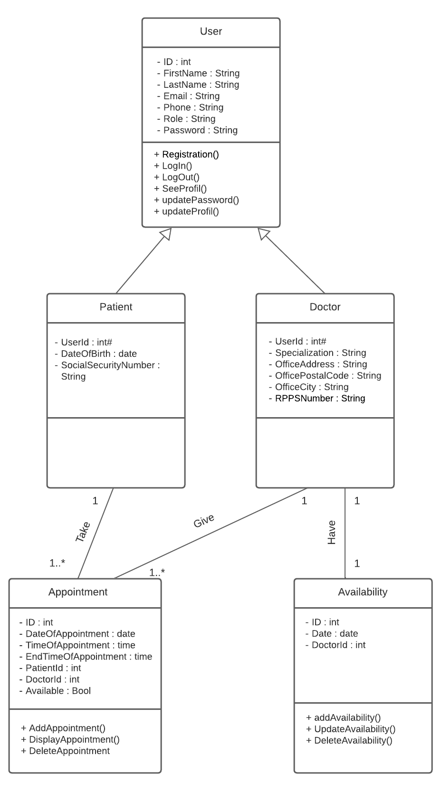


J’ai également créé un **diagramme de classe**, qui sert à modéliser la structure des données dans ma base de données.

Les classes représentent les entités (tables) et définissent les attributs, les relations entre les entités, les clés primaires et étrangères.

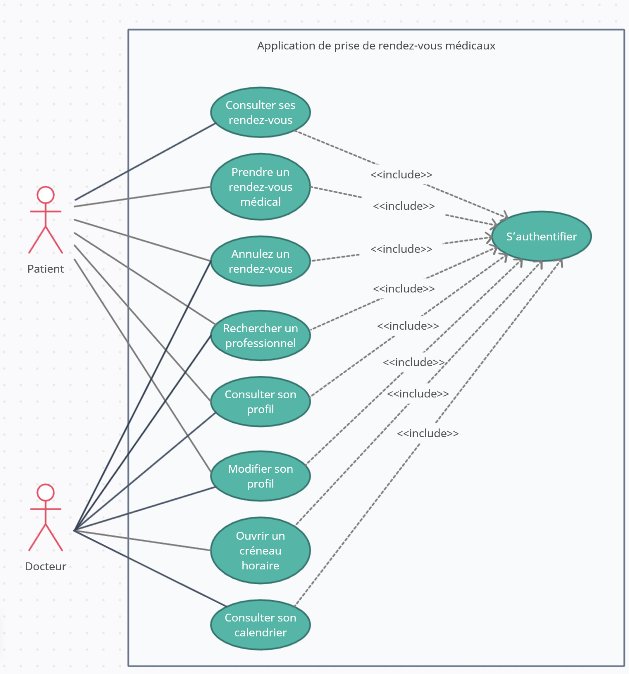
Ce diagramme me permet donc :

1. D’identifier les tables nécessaires dans la base de données.
2. De définir les attributs de chaque table et leurs types de données.
3. De modéliser les relations entre les tables (clés primaires et étrangères).
4. De comprendre comment les données seront stockées et organisées.
5. De fournir une vue d'ensemble claire de la structure de la base de données pour toute l'équipe de développement.



Enfin, j’ai réalisé un **diagramme de cas d’utilisation**, qui permet de comprendre comment les acteurs externes interagissent avec l’application.

Il fournit une vue d'ensemble des fonctionnalités du système du point de vue de l'utilisateur.



Tous les diagrammes se trouvent dans le dossier conception du GitHub :

<https://github.com/Yradouani/CareConnect/tree/main>

**5.2 Maquettes**

Les maquettes permettent de visualiser à quoi ressemblera l’application finale.

Cela aide à clarifier les idées et à obtenir une compréhension visuelle du concept.

Pour le design, j’ai opté pour un aspect simple ave*c une interface conviviale et responsive, pour une expérience utilisateur cohérente sur différents dispositifs.*

*J’ai choisi en couleur primaire un bleu qui reflète la confiance et en couleur secondaire un orange synonyme de dynamisme et d'énergie.*

Couleur primaire :

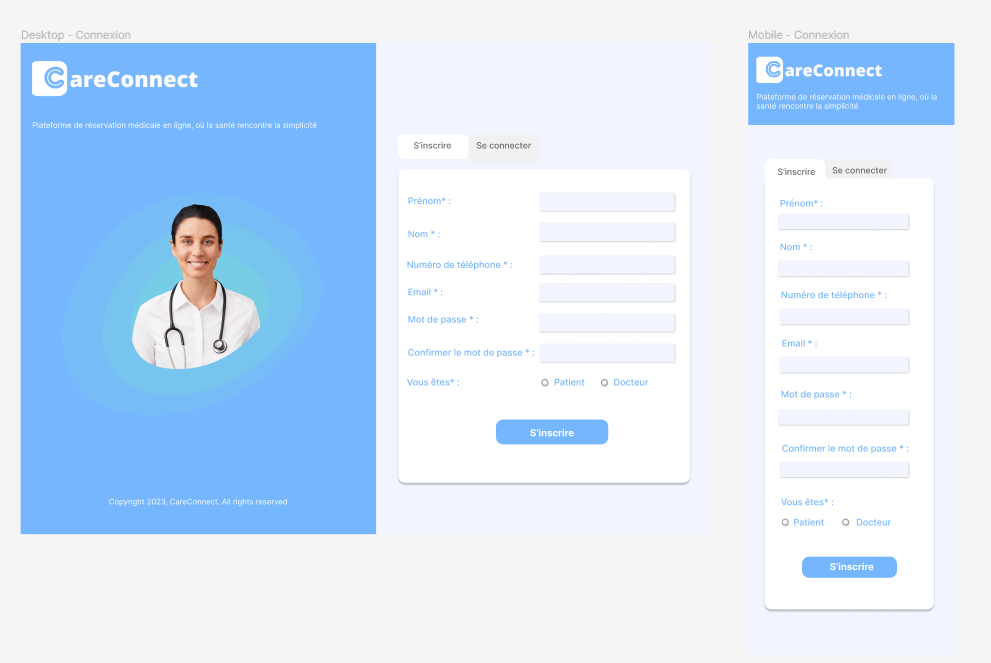


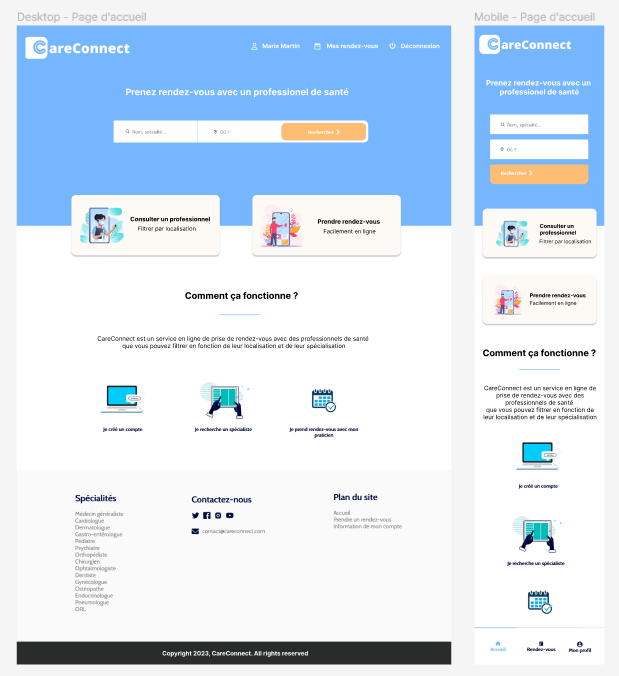
Couleur secondaire :



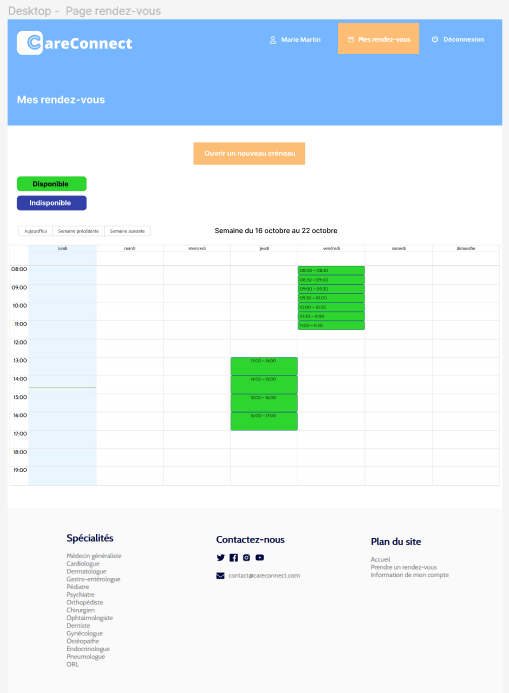
J’ai donc réalisé les maquettes de l’application avec Figma, au format desktop mais aussi mobile pour avoir le design du responsive.

Maquettes de la page de connexion :

Maquettes de la page d’accueil :

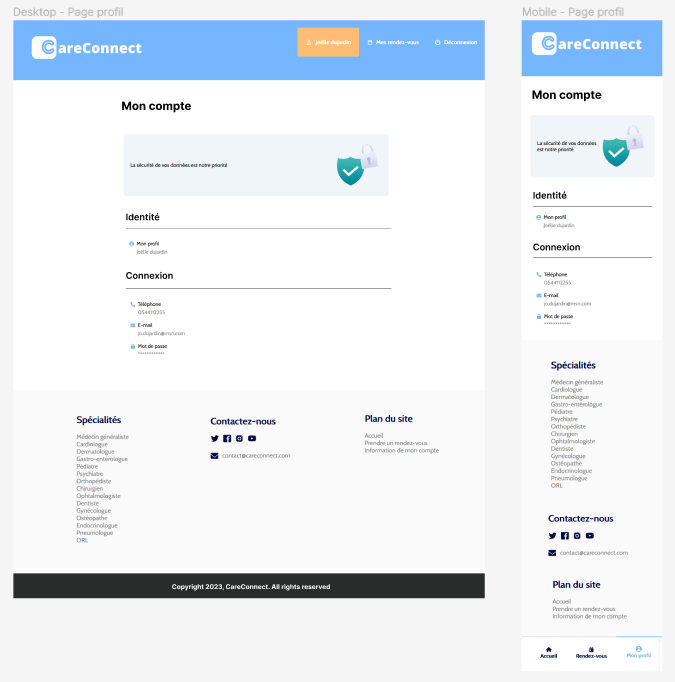


Maquette de la page des rendez-vous :





Maquette de la page profil :



**5.3 Architecture technique du site**

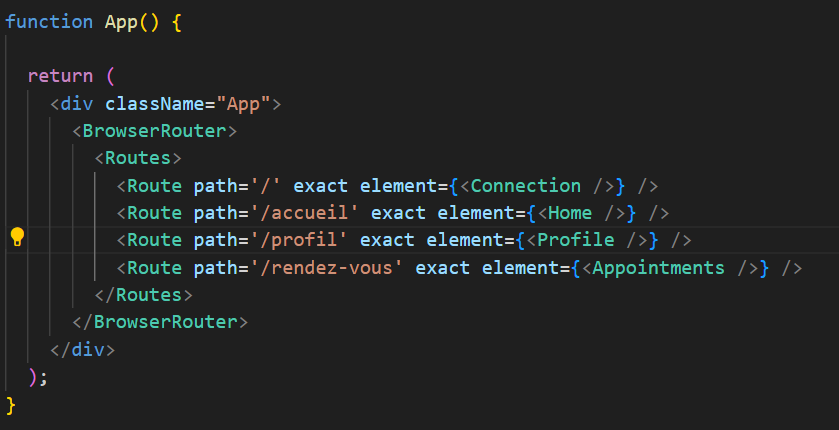
**Architecture front-end**

Pour mettre en place le template de base d’une application avec React, il faut entrer la commande : npx create-react-app “nom\_application”.

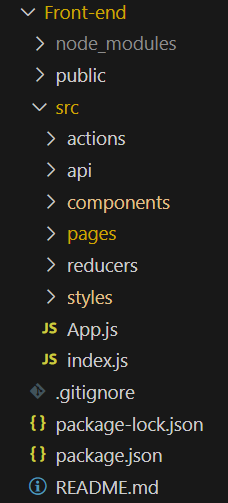
J’ai ensuite créé les dossiers “pages” et “components” pour y mettre les différentes pages et composants que je vais développer.

J’ai mis en place des dossiers actions et reducers pour gérer l’état de l’application avec Redux.

Enfin j’ai créé un routeur dans App.js pour permettre à l’utilisateur de naviguer entre les différentes pages en fonction des liens.



Une fois tous ces éléments mis en place, voici l’arborescence du front-end.



**Architecture back-end**

Pour mettre en place le template de base de l’API avec Laravel, il faut entrer la commande : composer create-project laravel/laravel “nom\_application”.

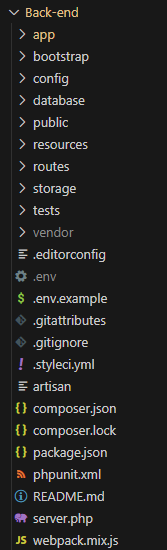
Il faut ensuite lancer le serveur apache émulé par Laravel pour faire tourner l’API avec la commande : “php artisan serve”.

On peut ensuite vérifier que l’API tourne sur le localhost avec le numéro de port que l’on a spécifié.

J’ai ensuite créé les routes dans le fichier déjà présent routes/api.php puis les contrôleurs associés à chaques routes.

J’ai créé les différents modèles que l’on retrouvera comme table en base de données et les migrations qui permettent la création des éléments de la base de données.

Une fois tous ces éléments mis en place, voici l’arborescence du back-end :

****

**5.4 Démarche de la réalisation de l’application**

Après l’architecture des frameworks mise en place, j’ai commencé par développer l’API.

L’application CareConnect est développée dans le respect de l’architecture MVC qui se divise en trois composants : les modèles, les vues et les contrôleurs.

1. **Les Modèles**

Les **Modèles** représentent la couche de données de l'application et s'occupent de la gestion et de la manipulation des données (présent dans le backend).

Dans un premier temps, j’ai créé mes 4 modèles User, Patient, Doctor et Appointment.

Voici un exemple du modèle Appointment :

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Appointment extends Model

{

use HasFactory;

protected $fillable = [

'dateOfAppointment',

'timeOfAppointment',

'endTimeOfAppointment',

'doctor\_id',

'patient\_id',

'available'

];

public function doctor()

{

return $this->belongsTo(Doctor::class, 'doctor\_id', 'user\_id');

}

}

Le modèle Appointment définit un tableau $fillable qui spécifie les attributs dateOfAppointment, timeOfAppointment, endTimeOfAppointment, doctor\_id, patient\_id, et available que l’on ajoutera à la base de données.

Grâce à Eloquent, je peux définir le lien entre deux modèles de données.

1. **Les Migrations**

Une migration utilise la définition du modèle qui lui correspond pour créer ou mettre à jour la structure de la base de données. Cela permet de garder la base de données en phase avec la structure du modèle de l'application.

J’ai donc ajouter les migrations pour créer ma base de données :

class CreateAppointmentsTable extends Migration

{

public function up()

{

Schema::create('appointments', function (Blueprint $table) {

$table->id();

$table->date('dateOfAppointment');

$table->time('timeOfAppointment');

$table->time('endTimeOfAppointment');

$table->unsignedBigInteger('doctor\_id');

$table->foreign('doctor\_id')->references('id')->on('users');

$table->unsignedBigInteger('patient\_id')->nullable();

$table->foreign('patient\_id')->references('id')->on('users');

$table->boolean('available')->default(true);

$table->timestamps();

});}

public function down()

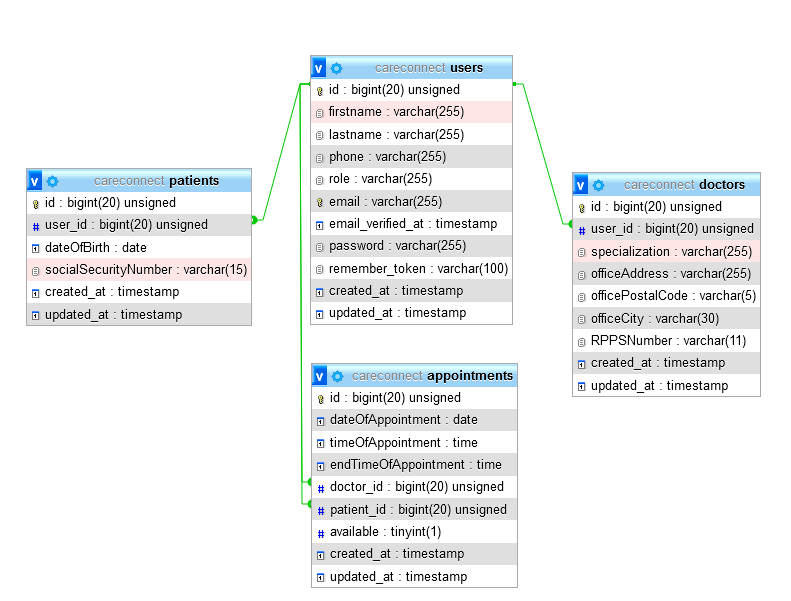
{

Schema::dropIfExists('appointments');

}

}

Une fois la migration faite, j’obtiens la base de donnée suivante :



1. **Les Routes**

J’ai ensuite mis en place les routes qui servent à définir les endpoints, c’est-à-dire les URL auxquelles le frontend enverra les requêtes HTTP pour effectuer des opérations spécifiques.

Chaque route correspond à une URL spécifique qui permet d’accéder à une fonctionnalité ou une ressource particulière de l'API.

<?php

use App\Http\Controllers\AppointmentController;

use App\Http\Controllers\UserController;

use Illuminate\Support\Facades\Route;

//Users

Route::middleware(['throttle:login'])->group(function () {

Route::post('/connexion', [UserController::class, 'logIn']);

});

Route::post('/inscription', [UserController::class, "registration"]);

Route::post('/mon-compte', [UserController::class, "getUserById"]);

Route::post('/recherche', [UserController::class, "getDoctorByNameSpecialityAndLocation"]);

//Routes protected by token authentication

Route::middleware(['jwt.auth'])->group(function () {

//Appointments

Route::post('/annuler-un-rendez-vous/{id}', [AppointmentController::class, "deleteAppointment"]);

Route::post('/ajouter-un-rendez-vous', [AppointmentController::class, "addAppointment"]);

Route::get('/rendez-vous/{id}', [AppointmentController::class, "getAllAppointmentsOfOneUser"]);

Route::put('/rendez-vous/{id}', [AppointmentController::class, "makeAppointment"]);

//Users

Route::put('/profile/{id}', [UserController::class, "updateProfil"]);

});

1. **Les Contrôleurs**

Les **Contrôleurs** agissent comme intermédiaires entre le Modèle et la Vue. Ils gèrent la logique d'acheminement des demandes de l'utilisateur et les réponses appropriées.

J’ai séparé les UserControllers et les AppointmentControllers qui contiennent chacun des fonctions responsables de traiter des demandes relatives aux utilisateurs et aux rendez-vous respectivement.

Par exemple, la fonction “getAllAppointmentsOfOneUser” présente dans AppointmentController qui sert à récupérer tous les rendez-vous d’un utilisateur avec les informations du docteur associé au rendez-vous :

public function getAllAppointmentsOfOneUser($id)

{

$user = User::find($id);

if ($user) {

$column = ($user->role === 'doctor') ? 'doctor\_id' : 'patient\_id';

try {

$appointments = Appointment::where($column, $id)

->orderBy('dateOfAppointment', 'asc')

->orderBy('timeOfAppointment', 'asc')

->with('doctor.user')

->get();

return response()->json($appointments, 200);

} catch (Exception $e) {

echo '</br> <b> Exception Message: ' . $e->getMessage() . '</b>';

}

} else {

return response()->json(['message' => 'Utilisateur introuvable.'], 404);

}

}

1. **Les Vues**

Après avoir développé le backend de l’application, j’ai réalisé la partie frontend avec React.

Les **Vues** correspondent à la partie de l'application qui gère l'interface utilisateur et l'affichage des données. Elles correspondent donc aux différentes pages dynamiques.

C’est à partir des vues que l’on envoie des requêtes à l’API avec une méthode et une URL spécifique.

Par exemple, le composant “Login” s’occupe de gérer la requête relative à la connexion de l’utilisateur.

const handleRegister = async (e) => {

e.preventDefault();

try {

const response = await axios.post(

'/connexion',

JSON.stringify({ email, password: pwd }),

{

headers: { 'Content-Type': 'application/json' },

}

);

const decodedToken = jwt\_decode(response?.data?.token);

const userData = {

id: decodedToken.id,

token: response?.data?.token,

firstname: decodedToken.firstname,

lastname: decodedToken.lastname,

phone: decodedToken.phone,

email: decodedToken.email,

role: decodedToken.role,

issued\_at: decodedToken.issued\_at

};

dispatch(setUser(userData));

try {

const response = await axios.get(

`/rendez-vous/${userData.id}`,

{

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${userData.token}`

}

}

);

const tab = [];

Object.entries(response.data).forEach(item => {

tab.push(item[1])

})

dispatch(setAppointments(tab));

} catch (err) {

console.error('Erreur lors de la récupération des rendez-vous :', err);

}

navigate('/accueil');

} catch (err) {

console.log(err);

if (!err?.response) {

setErrMsg('Pas de réponse du serveur');

} else if (err.response?.status === 400) {

setErrMsg("Veuillez entrer votre email ET votre mot de passe");

} else if (err.response?.status === 401) {

setErrMsg("Email et/ou mot de passe incorrect");

} else if (err.response?.status === 429) {

setErrMsg("Trop de tentatives de connexion, veuillez réessayer plus tard");

} else {

setErrMsg('Connexion échouée')

}

errRef.current.focus();

}

}

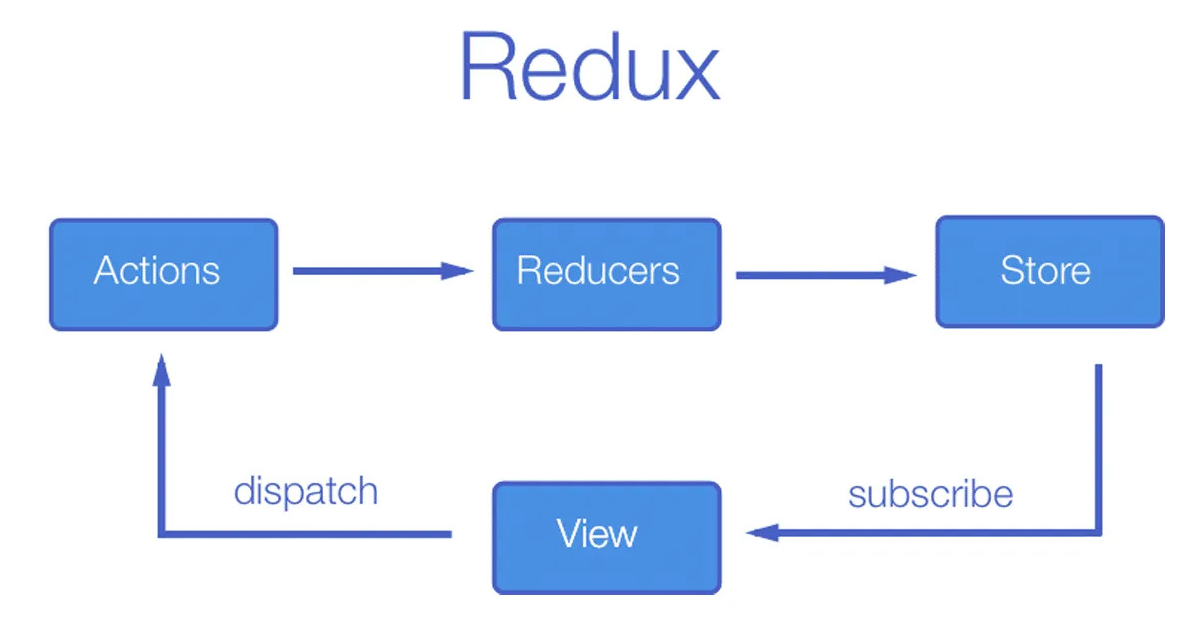
La fonction “handleRegister” est appelée lorsque l’utilisateur clique sur le bouton “se connecter”. Une requête asynchrone est alors envoyée avec la méthode “POST”. En premier paramètre de la requête, on spécifie l’URL de la route et en deuxième paramètre, on fournit les données attendues par le contrôleur.

On stocke enfin les données renvoyées par le serveur dans le state manager.

**5.5 State manager**

Pour gérer l’état de l’application, j’ai utilisé la bibliothèque Redux qui se compose de plusieurs éléments clés qui travaillent ensemble pour maintenir et mettre à jour l'état global de l'application.

1. **Dispatcher** : on l’utilise en réponse à un changement dans une page ou un composant. Il fait appel aux actions.
2. **Actions** : Les actions spécifient comment l'état global de l'application doit être modifié en réponse aux changements. Elles décrivent ce qui a eu lieu et contiennent des informations sur les modifications à apporter à l'état.
3. **Reducers** : Les reducers sont chargés de mettre à jour l'état de l'application en réponse aux actions. Ils prennent l'état actuel et l'action comme arguments, calculent le nouvel état en fonction de ces entrées, et retournent le nouvel état au store.
4. **Store** : Le store est un objet central qui contient l'état global de l'application. Il conserve toutes les données nécessaires pour le bon fonctionnement de l'application.



**5.6 Sécurité**

La sécurité représente une part très importante du métier de développeur d’application, c’est pourquoi les compétences à valider pour l’obtention du titre insistent sur l’intégration des recommandations de sécurité.

J’ai donc mis en place un certain nombre de mesures pour assurer la sécurité des données de l’application.

1. **Hachage du mot de passe**

Lorsque l’utilisateur s’inscrit, son mot de passe est hashé grâce au package bcrypt.

Cela consiste à prendre le mot de passe et à le transformer en une chaîne de caractères de longueur fixe (un "hash"), de façon irréversible.

$user = User::create([

"firstname" => $userInfo["firstname"],

"lastname" => $userInfo["lastname"],

"phone" => $userInfo["phone"],

"role" => $userInfo["role"],

"email" => $userInfo["email"],

"password" => bcrypt($userInfo["password"]),

]);

Lorsque l’utilisateur souhaite se connecter, pour vérifier que le mot de passe qui est entré correspond au hash stocké en base de données, on utilise la fonction Hash::check() de bcrypt pour comparer le mot de passe en clair avec le hash stocké. Si les deux correspondent, l'utilisateur est authentifié.

if (!Hash::check($userInfo["password"], $user->password)) return response(["message" => "Aucun utilisateur de trouver avec ce mot de passe"], 401);

1. **Token d'authentification**

Certaines routes de l’application nécessitent que l’utilisateur soit connecté. Pour vérifier cette contrainte, lors de la connexion, j’utilise le package jwt pour créer un jeton que l’utilisateur fournira à chacune de ses requêtes pour s’authentifier.

J’ai choisi d’insérer les informations de l’utilisateur dans le token pour plus de sécurité.

Ainsi les données renvoyées au front-end lors de la connexion ne seront pas en clair.

$customClaims = [

'id' => $user->id,

'firstname' => $user->firstname,

'lastname' => $user->lastname,

'phone' => $user->phone,

'email' => $user->email,

'role' => $user->role,

];

$token = JWTAuth::claims($customClaims)->fromUser($user);

return response([

"token" => $token

], 200);

Voici le token reçu dans le front-end :

"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjgwMDAvYXBpL2Nvbm5leGlvbiIsImlhdCI6MTY5NzczMzY3MSwiZXhwIjoxNjk3NzM3MjcxLCJuYmYiOjE2OTc3MzM2NzEsImp0aSI6Im9FOVNlaXROcElZenN5YlgiLCJzdWIiOiIxNyIsInBydiI6IjIzYmQ1Yzg5NDlmNjAwYWRiMzllNzAxYzQwMDg3MmRiN2E1OTc2ZjciLCJpZCI6MTcsImZpcnN0bmFtZSI6Ikpvw6tsbGUiLCJsYXN0bmFtZSI6Ik1hcnRpbiIsInBob25lIjoiMDY1NDExNzc4NSIsImVtYWlsIjoiam9lbGxlLm1hcnRpbkB5YWhvby5mciIsInJvbGUiOiJwYXRpZW50In0.t1\_0ObfJt-eKM3osJ50OrTatDFV2VUTXufAA6xrLdEA"

Le token est ensuite décodé dans le front-end grâce au package **"jwt-decode"**

const decodedToken = jwt\_decode(response?.data?.token);

const userData = {

id: decodedToken.id,

token: response?.data?.token,

firstname: decodedToken.firstname,

lastname: decodedToken.lastname,

phone: decodedToken.phone,

email: decodedToken.email,

role: decodedToken.role,

};

Le token a une durée de validité de 60 min, au-delà, l’utilisateur sera déconnecté et renvoyé vers la page d’accueil.

1. **Email unique**

L'unicité de l'adresse e-mail aide à réduire les abus comme la création de comptes factices.

Pour ce faire, lors de la création de la table user, dans la migration j’indique que l’email doit être unique :

Schema::create('users', function (Blueprint $table) {

$table->id();

$table->string('firstname');

$table->string('lastname');

$table->string('phone');

$table->string('role');

$table->string('email')->unique();

$table->timestamp('email\_verified\_at')->nullable();

$table->string('password');

$table->rememberToken();

$table->timestamps();

});

Puis lors de l’inscription, je vérifie à nouveau :

$userInfo = $request->validate([

"firstname" => ["required", "string", "min:2", "max:30"],

"lastname" => ["required", "string", "min:2", "max:30"],

"phone" => ["required", "string", "min:10", "max:13"],

"role" => ["required", "string", "in:patient,doctor"],

"email" => ["required", "email", "unique:users,email"],

"password" => ["required", "string", "min:8", "max:30"],

"confirm\_password" => ["required", "same:password"]

]);

1. **REGEX**

J’ai utilisé des expressions régulières pour assurer que l’utilisateur entre des données conformes à celles attendues mais aussi pour prévenir les **attaques par injection SQL** c'est-à-dire empêcher de potentiels attaquants d'injecter des commandes SQL malveillantes.

Cela permet également la protection contre les **attaques XSS** en évitant que du code Javascript malveillant ne soit intégré dans les données soumises.

Exemple de REGEX du formulaire d’inscription :

const USER\_REGEX = /^[a-zA-Z][a-zA-Z-éÉèÈàÀùÙâÂêÊîÎôÔûÛïÏëËüÜçÇ]{2,24}$/;

const PHONE\_REGEX = /^\d{10,13}$/;

const PWD\_REGEX = /^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[0-9])(?=.\*[!@#$%£§&]).{8,24}$/;

const SOCIAL\_SECURITY\_REGEX = /^[0-9]{15}$/;

const ADDRESS\_REGEX = /^\d+\s+[\w\s-]+$/;

const POSTAL\_CODE\_REGEX = /^[0-9]{5}$/;

const RPPS\_REGEX = /^[0-9]{11}$/;

const EMAIL\_REGEX = /^[a-zA-Z0-9.!#$%&’\*+/=?^\_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)\*$/;

const DATE\_OF\_BIRTH\_REGEX = /^\d{4}-\d{2}-\d{2}$/;

1. **Limitation des tentatives de connexion**

Pour protéger l’application des tentatives de connexion par force brute, j’ai limité le nombre des tentatives à 3 par minute.

Pour cela j’ai ajouté un middleware à la route de connexion :

RateLimiter::for('login', function (Request $request) {

return Limit::perMinute(3)->by($request->input('email'));

});

Route::middleware(['throttle:login'])->group(function () {

Route::post('/connexion', [UserController::class, 'logIn']);

});



**5.7 Référencement naturel**

Le SEO, ou Search Engine Optimization, est très important à prendre en compte.

Plus il sera bon, plus le site sera visible et potentiellement visité.

J’ai donc mis en place quelques stratégies pour augmenter le référencement naturel de l’application CareConnect.

1. **Balises title et meta description**

Afin de fournir une meilleure expérience utilisateur et inciter les clics sur l’application, j’ai ajouté une balise meta description à toutes mes pages React. Cette dernière est utilisée par les moteurs de recherche pour afficher un extrait de texte dans les résultats de recherche. Les utilisateurs peuvent ainsi avoir un aperçu du contenu de la page.

CareConnect étant un single page application, j’ai utilisé le package 'react-helmet'pour insérer cette balise à l’intérieur de mon rendu.

import { Helmet } from 'react-helmet';

<Helmet>

<title>CareConnect - Page de connexion</title>

<meta name="description" content="Connectez-vous à CareConnect pour prendre vos rendez-vous médicaux en ligne facilemen et rapidement" />

</Helmet>

1. **Optimisation des images**

J’ai compressé les images pour une meilleure vitesse de chargement et j’ai ajouté des balises alt.

La balise "alt" est principalement destinée aux lecteurs d'écran utilisés par les personnes aveugles ou malvoyantes. En fournissant un texte alternatif descriptif pour les images, on améliore l'accessibilité de l’application.

1. **Lighthouse**

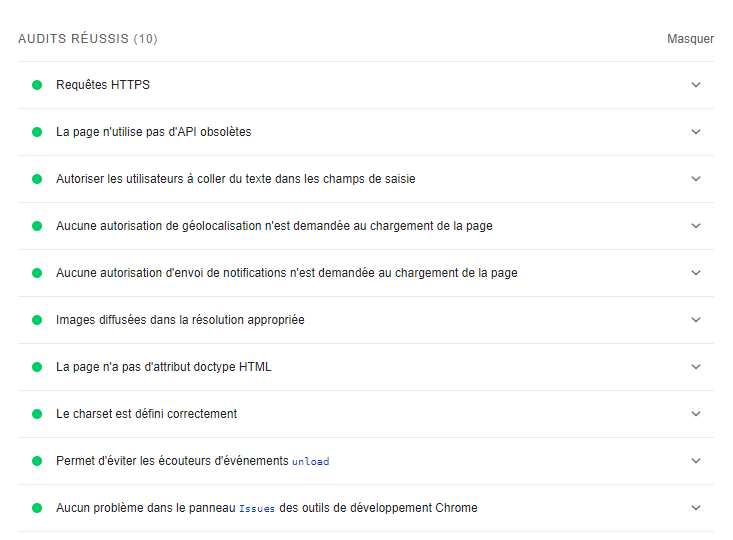
J’ai installé l’extention Lighthouse sur mon navigateur Google Chrome. Cette extension évalue la qualité d'une page web en fonction de divers critères. Il génère un rapport détaillé sur l'accessibilité, les bonnes pratiques SEO et la qualité du code de la page web.

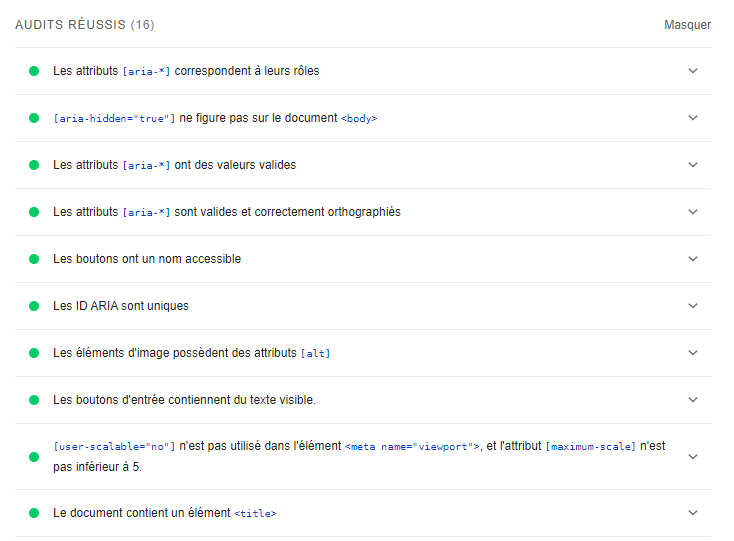
Voici le score que j’ai obtenu sur l’application :



Lighthouse fournit également un audit personnalisé qui aide à améliorer le SEO.

J’ai pu modifier plusieurs points qui m’ont aidé à améliorer le score, comme le changement de balise adéquat, la suppression de doublon au niveau d’id…





| Présentation d’un jeu d’essai |
| --- |
|  |

**6. Présentation d’un jeu d’essai**

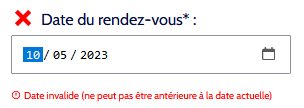
***Ouverture d’un créneau horaire***

Pour permettre aux patients de prendre rendez-vous, le professionnel de santé doit ouvrir un ou plusieurs créneaux de disponibilité.

## Données en Entrée

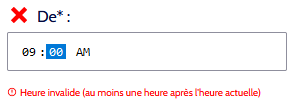
Lorsqu’un professionnel ouvre la fenêtre dédiée à cette fonctionnalité, il doit entrer plusieurs informations :

1. **La date du créneau**, obligatoirement postérieure ou égale à la date actuelle sinon un message d’erreur s’affiche et empêche la validation du formulaire.

****

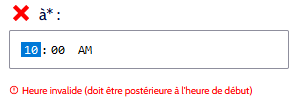
**Je choisis donc le Lundi 30 Novembre 2023 pour la date**

1. **L’heure du début du créneau** ou moins 1h après l’heure actuelle (pour une date similaire)

****

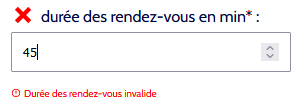
**Je choisis 8:30 AM pour l’heure de début**

1. **L’heure de fin du créneau**, obligatoirement postérieure à l’heure du début du créneau.

****

**Je choisis 10:30 AM pour l’heure de fin**

1. La durée de chaque rendez-vous, doit pouvoir diviser la durée du créneau

****

**Je choisi 30min pour la durée**

Chaque champ du formulaire est soumis à une expression régulière spécifique pour garantir le bon format attendu mais aussi pour des raisons de sécurité.

## Données Attendues

J’ai entré des données conformes à ce que le formulaire attend donc tous les champs devraient être validés avec un icône “checked”.

Cependant, malgré la validité des données, le back-end de mon application devrait refuser l’ouverture de ces créneaux car ils chevauchent d’autres créneaux déjà ouverts et stockés en base de données. J'attends l’ouverture de la pop-up qui indique que le créneau n’a pas pu être ouvert.

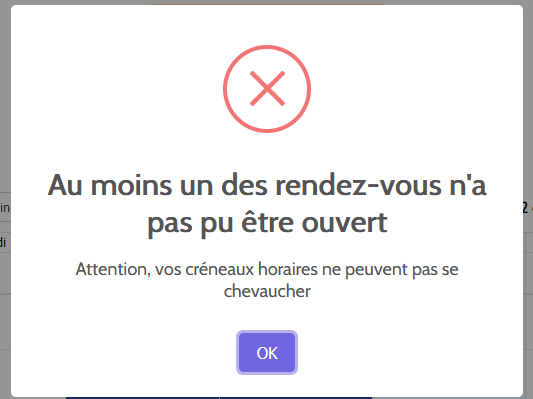
Il ne devrait pas y avoir d’ajout de rendez-vous dans le calendrier professionnel et il ne doit pas y avoir non plus d’ajout en base de données.

**Données Obtenues**

Concernant les données que j’ai entrées dans le formulaire elles ont toutes un icône “checked” comme prévu.

****

La pop-up d’erreur s’affiche correctement :



Aucun rendez-vous n’est ajouté au calendrier, ni en base de données.

Pour tester ce cas de figure au niveau du code, j’ai ajouté deux tests sur le contrôleur qui permet de créer un rendez-vous en base de données.

class AppointmentControllerTest extends TestCase

{

public function test\_it\_can\_create\_an\_appointment()

{

$user = User::factory()->create();

$token = JWTAuth::fromUser($user);

$response = $this->withHeaders([

'Authorization' => 'Bearer ' . $token,

])->json('POST', '/api/ajouter-un-rendez-vous', [

'dateOfAppointment' => '2023-10-28',

'timeOfAppointment' => '09:00',

'endTimeOfAppointment' => '10:00',

'doctor\_id' => 4,

'role' => 'doctor',

]);

$response->assertStatus(200);

$response->assertJson([

'dateOfAppointment' => '2023-10-28',

'timeOfAppointment' => '09:00',

'endTimeOfAppointment' => '10:00',

'doctor\_id' => 4,

'available' => true,

]);

}

public function test\_it\_can\_not\_create\_appointment\_with\_bad\_data()

{

$user = User::factory()->create();

$token = JWTAuth::fromUser($user);

$response = $this->withHeaders([

'Authorization' => 'Bearer ' . $token,

])->json('POST', '/api/ajouter-un-rendez-vous', [

'dateOfAppointment' => '2023-11-30',

'timeOfAppointment' => '09:00',

'endTimeOfAppointment' => '10:00',

'doctor\_id' => 2,

'role' => 'doctor',

]);

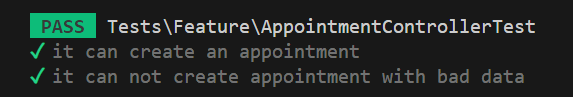
$response->assertStatus(400);

}

}

J’ai créé une fonction de référence qui doit ajouter un rendez-vous en base de données et une 2ème fonction qui contient les données que nous avons entrées dans le jeu d’essai. La 2ème fonction teste que que le status dois être 400 et que l’ajout en base de données ne doit pas se faire.

Les 2 tests sont validés :





| Description de la veille |
| --- |
|  |

**7. Description de la veille**

Lors du développement de mon projet, j’ai cherché les potentielles failles de sécurité présentes dans l’application CareConnect.

Ces failles représentent un enjeu de taille aux vues des données confidentielles stockées dans la base de données.

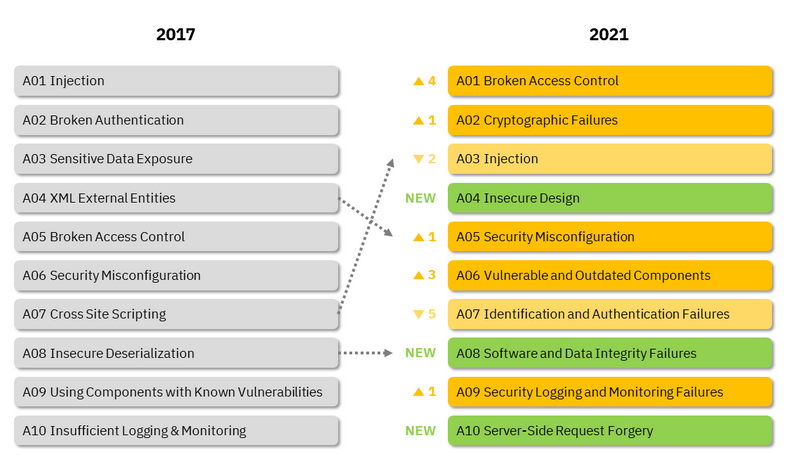
J’ai donc réalisé une veille pour mieux comprendre ces failles et comment m’en prémunir. J’ai recherché sur internet "Common Security Vulnerabilities in a Web Application" et un site en particulier à retenu mon attention :

<https://owasp.org/www-project-top-ten/>

L'OWASP (Open Web Application Security Project) est une communauté en ligne de professionnels de la sécurité informatique dédiée à la sensibilisation et à la promotion des meilleures pratiques en matière de sécurité des applications web.

L'organisation publie le “Top Ten Project” qui répertorie les dix vulnérabilités de sécurité les plus courantes dans les applications web. Ces informations aident à comprendre, prévenir et corriger les failles de sécurité courantes dans les applications.

Je me suis donc intéressée au top 10 des failles de sécurité des applications web de l’OWASP, et particulièrement aux numéros 1, 3, 5 et 7.



**Broken Access Control (numéro 1)**

C’est une vulnérabilité courante dans les applications web qui permet à un utilisateur non autorisé d'accéder à certaines données auxquelles il ne devrait pas avoir accès. Cette faille se produit lorsque les contrôles d'accès appropriés ne sont pas mis en place pour restreindre l'accès aux fonctionnalités ou aux données de l'application.

Pour contrer cette faille, j’ai mis en place dans l’application :

* Une **authentification avec mot de passe fort** : au moins 8 caractères dont une lettre majuscule, une lettre minuscule, un chiffre et un caractère spécial.
* Un **contrôle d'accès basé sur les rôles** : chaque utilisateur a un rôle spécifique (“doctor” ou “patient”) attribué et accède uniquement aux ressources associées à ce rôle
* Une utilisation de **token d’authentification** nécessaire pour accéder à certaines routes
* Une utilisation de **librairies reconnues** avec des **versions récentes**

**Injection (numéro 3)**

La faille d’injection se produit lorsqu'un attaquant parvient à injecter du code malveillant ou des données non validées dans une application. Les types d'injection les plus courants sont les injections SQL, les injections de commandes, et les injections de scripts.

Pour contrer cette faille, j’ai mis en place dans l’application :

* Des **regex stricts** pour n’autoriser que les caractères nécessaires
* L’utilisation de **l’ORM Éloquent** qui utilise des requêtes préparées pour interagir avec la base de données. De plus, Eloquent échappe automatiquement les données avant de les inclure dans des requêtes SQL. Donc les données provenant d'entrées utilisateur sont traitées de manière sécurisée et ne peuvent pas être interprétées comme du code SQL malveillant.
* L’utilisation de la **fonction “validate”** qui permet de valider les données d'entrée fournies par les utilisateurs et s'assurer que les données respectent les critères définis, comme la longueur minimale, la présence, le format, etc.

**Security Misconfiguration (numéro 5)**

Elle survient lorsque les paramètres de sécurité d'une application ne sont pas correctement configurés, exposant ainsi l'application à des risques de sécurité.

Pour contrer cette faille, j’ai mis en place dans l’application :

* Une **réduction des privilèges** : les utilisateurs ont uniquement les autorisations dont ils ont besoin
* Une **protection du fichier de configuration .env** : situé à l’extérieur du répertoire “public” pour ne pas être accessible via un navigateur

**Identification et Authentication Failures (numéro 7)**

Elle concerne les vulnérabilités liées à l'authentification des utilisateurs, aux sessions, aux mots de passe, et à tout ce qui touche à l'identification des utilisateurs dans une application.

Pour contrer cette faille, j’ai mis en place dans l’application :

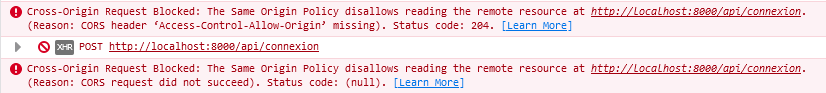
* Un **hachage du mot de passe** et **ajout d’un sel** avec Bcrypt
* Une **expiration du token d'authentification** au bout de 60 min avec obligation de reconnexion
* Des **messages d’erreur génériques** qui ne fournissent pas d’information sensible lors de l’authentification



| Description d’une situation de travail |
| --- |
|  |

**10. Description d’une situation de travail**

Durant le développement de mon application, j’ai rencontré une erreur CORS lorsque j’essayais d’envoyer des requêtes du frontend vers le backend.



Il a d’abord été nécessaire que je comprenne ce qu’est une erreur CORS.

J’ai donc cliqué sur le lien “Learn More” présent dans le message d’erreur du navigateur :

<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS/Errors/CORSMissingAllowOrigin?utm_source=devtools&utm_medium=firefox-cors-errors&utm_campaign=default>

Ce lien explique qu’il y a une erreur car la réponse à la requête ne contient pas dans son en-tête “Access-Control-Allow-Origin”.

Mais cela ne permet toujours pas de comprendre pourquoi cet en-tête est nécessaire. Donc je continue à naviguer sur le même site et cette page me fournit les informations que je cherche :

<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

Je comprends que Cross-Origin Resource Sharing (CORS) est un mécanisme de sécurité qui permet à un serveur d'indiquer à partir de quelles origines autres que les siennes un navigateur doit autoriser le chargement de ressources.

Une application web utilisant une API ne peut demander des ressources que depuis la même origine à partir de laquelle l'application a été chargée, sauf si la réponse des autres origines inclut les en-têtes CORS appropriés.

C'est pourquoi je dois ajouter des en-têtes CORS à mon API Laravel.

Je fais donc la recherche suivante dans Google “how to add access control allow origin header in laravel” et j’accède à ce site qui me semble pertinent :

https://www.stackhawk.com/blog/laravel-cors/

Il explique que je dois importer la librairie “fruitcake/laravel-cors” et que je dois configurer le fichier config/cors.php de la manière suivante :

<?php

return [

'paths' => ['api/\*', 'sanctum/csrf-cookie'],

'allowed\_methods' => ['\*'],

'allowed\_origins' => ['\*'],

'allowed\_origins\_patterns' => [],

'allowed\_headers' => ['\*'],

'exposed\_headers' => [],

'max\_age' => 0,

'supports\_credentials' => false,

];

“\*” signifie que toutes les sources sont acceptées. En phase de développement, on peut se permettre de laisser la configuration de cette manière par contre en production, on devra spécifier exactement les sources autorisées à faire des requêtes.

Ces recherches m’ont permis de débloquer le problème et de faire des requêtes depuis le frontend vers le backend.

| Conclusions |
| --- |
|  |
| En concevant et développant l'application CareConnect, j'ai eu l'opportunité d'approfondir mes connaissances, en particulier dans les domaines du maquettage, de la sécurité et de la gestion de l'état. L'application que j'ai créée représente un MVP (Minimum Viable Product) contenant les fonctionnalités essentielles pour répondre aux besoins identifiés lors de la phase de conception.  J'ai identifié plusieurs axes d'amélioration que je n'ai pas pu implémenter en raison de contraintes de temps : |

* Trouver une API qui me permette de vérifier le numéro RPPS du médecin avant d'accepter son inscription
* Ajouter un tableau de bord pour gérer les utilisateurs
* Envoyer une notification par mail au patient pour lui confirmer la réservation ou l’annulation de son rendez-vous
* Envoyer un mail 24h avant le rendez-vous pour faire un rappel au patient
* Utilisation de seeders pour peupler la bdd
* Explorer et utiliser davantage les fonctionnalités préexistantes dans laravel
* Séparer le code du frontend en plus petit composants
* Optimiser mon code

Les 4 derniers points font partie *de mes objectifs personnels pour fournir un meilleur code au quotidien.*

Beaucoup de vérifications étaient nécessaires pour que les créneaux ne se chevauchent pas pour un même médecin, pour trier les dates (passées, présentes et futures) et pour la vérification des droits.

Pendant le développement, j'ai rencontré des difficultés, notamment dans la manipulation des dates et la gestion des erreurs. De nombreuses vérifications étaient nécessaires pour que les créneaux ne se chevauchent pas pour un même médecin, pour trier les dates (passées, présentes et futures) et pour la vérification des droits.

En conclusion, j'ai apprécié chaque étape de ce projet. Chaque nouveau projet me permet de progresser et d'ajouter une couche de complexité supplémentaire, ce qui reste mon objectif constant. J'ai pu identifier mes erreurs et je m'efforce de ne pas les reproduire dans les projets futurs.

Merci pour votre attention.