

Rapport de stage

Mars-Avril 2021

TP Développeur Web / Web Mobile

Yann DECORCE

15 septembre 2020 – 12 mai 2021

6ONLINE < ACCESS **CODE** >
FORMAPRO SCHOOL

dm
web

REMERCIEMENTS

En préambule à ce rapport, je tiens particulièrement à remercier Marian Denys Dirigeant de DM Web, de m'avoir accueilli dans son entreprise et de m'avoir confié des tâches variées pendant ses 2 mois de stage.

Je tiens aussi à remercier Igor, développeur Web chez DM Web, de m'avoir encadrée au quotidien, d'avoir répondu à mes nombreuses sollicitations et d'avoir fortement contribué dans ma montée en compétence.

Enfin, ma reconversion professionnelle n'aurait pas été possible sans l'implication de Alain Merucci, Coach-Formateur, de l'Access Code School de Lons le Saunier qui m'a patiemment fait découvrir et formé aux technologies actuelles du Web.

Merci à tous.

Yann.

<u>1</u>	<u>INTRODUCTION</u>	<u>4</u>
<u>2</u>	<u>LISTE DES COMPETENCES DU REFERENTIEL QUI SONT COUVERTES PAR LE PROJET</u>	<u>4</u>
<u>3</u>	<u>RESUME DU PROJET</u>	<u>4</u>
<u>4</u>	<u>CAHIER DES CHARGES, EXPRESSION DES BESOINS</u>	<u>5</u>
<u>5</u>	<u>SPECIFICATIONS TECHNIQUES DU PROJET</u>	<u>5</u>
5.1	LARAVEL 8	5
5.1.1	LARAVEL MEDIA	6
5.1.2	LARAVEL EXCEL	6
5.2	BOOTSTRAP 4	7
<u>6</u>	<u>REALISATIONS DU CANDIDAT</u>	<u>7</u>
6.1	RECUPERATION DES DONNEES EXISTANTES.	7
6.2	CREATION DU PROJET SOUS LARAVEL	7
6.3	RECUPERATION DE LA BASE DE DONNEES EXISTANTE	8
6.4	STRUCTURATION DU PROJET LARAVEL.	9
6.5	STRUCTURATION DE LA PARTIE FRONT	11
6.5.1	LE TEMPLATE DE BASE	11
6.5.2	LA BARRE DE NAVIGATION	12
6.5.3	UNE VUE STATIQUE : LA HOME PAGE	13
6.5.4	UNE VUE DYNAMIQUE : LE RESULTAT D'UNE RECHERCHE	15
6.6	GESTION DE L'AUTHENTIFICATION	16
6.6.1	INSTANCIATION DU PACKAGE	17
6.6.2	ADAPTATION DES VUES	17
6.6.3	MODIFICATION DU CODE DU LOGINCONTROLLER	17
6.7	MODIFICATION, IMPORT ET MISE A JOUR DE LA TABLE USER	19
6.7.1	MODIFICATION DE LA TABLE USER	19
6.7.2	IMPORT ET MISE A JOUR DES DONNEES	20
6.8	UTILISATION DE LARAVEL MEDIA	22
6.9	SECURISATION DES ROUTES.	22
6.10	CREATION DES CRUD	24
6.10.1	METHODE DOCTORSINDEX	25
6.10.2	METHODE EDIT	27
6.10.3	METHODE UPDATE	29
6.10.4	METHODE DESTROY	30
6.10.5	METHODE SEARCH_DOCTOR	30
6.11	DEVELOPPEMENT DU MOTEUR DE RECHERCHE	32
<u>7</u>	<u>PRESENTATION DU JEU D'ESSAI</u>	<u>33</u>
<u>8</u>	<u>DESCRIPTION DE LA VEILLE,</u>	<u>35</u>
<u>9</u>	<u>DESCRIPTION D'UNE SITUATION DE TRAVAIL AYANT NECESSITE UNE RECHERCHE</u>	<u>36</u>
<u>10</u>	<u>CONCLUSION</u>	<u>38</u>

1 INTRODUCTION

Dans le cadre de ma formation, j'ai intégré l'entreprise DMWeb à Lons le Saunier en tant que stagiaire Full Stack. J'ai eu l'occasion de travailler sur de nombreux et variés projets en lien avec l'activité de l'entreprise. J'ai ainsi pu réaliser des intégrations HTML/CSS pour des besoins d'une plateforme d'e-learning, participer à la création de deux sites web utilisant le CMS Wordpress et développer et participer à des solutions utilisant le Framework Laravel. Mon dossier de projet en présente d'ailleurs un, la refonte du site Zestetik.fr

2 LISTE DES COMPETENCES DU REFERENTIEL QUI SONT COUVERTES PAR LE PROJET

Activités Types	Compétences professionnelles	
Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité	Maquetter une application	
	Réaliser une interface utilisateur web statique et adaptable	✓
	Développer une interface utilisateur web dynamique	✓
	Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce	
Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité	Créer une base de données	✓
	Développer les composants d'accès aux données	✓
	Développer la partie back-end d'une application web ou web mobile	✓
	Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce	

3 RESUME DU PROJET

Zestetik.fr est un site web créé par l'agence il y a 7 ans. Ce site référence des spécialistes de la chirurgie esthétique pour leur permettre une mise en relation plus aisée avec des patients potentiels.

Dans le cadre de ce projet, l'agence DM Web m'a demandé, suite à la demande du client, de mener une refonte complète du site. Le site avait été réalisé sous Cake PHP. La commande formulée était de récupérer la partie front-end existante et de l'intégrer dans un nouveau projet sous LARAVEL.

Mon travail a donc suivi les étapes suivantes :

- Clonage de la base de données actuelle
- Récupération du code source du site.
- Création d'un nouveau projet LARAVEL,
- Intégration des pages existantes dans LARAVEL
- Développement d'une interface back-office permettant l'administration des données du site.
- Adaptation de l'existant aux demandes du client : Nettoyage de la base de données, modification de l'interface du moteur de recherches, modification des données liées au médecin

4 CAHIER DES CHARGES, EXPRESSION DES BESOINS



Zestetik.fr est un site web existant depuis 7 ans.

La demande du client est de conduire une refonte du site web actuel pour lui apporter quelques évolutions.

Le site étant sur une technologie « obsolète », le premier besoin est de le moderniser pour le rendre maintenable. Pour ceci, DM Web a fait le choix d'utiliser le Framework LARAVEL pour la gestion du back-office, associé au Framework Bootstrap4 pour l'affichage responsive de la partie Front.

Par ailleurs, le client souhaite apporter des évolutions aux contenus de la base de données mais aussi à la fonctionnalité du moteur de recherche.

Il souhaite aussi que sa base de données User soit mise à jour pour les données ayant un rôle « médecin » (passage d'environ 5000 médecins à 500 médecins actifs)

Enfin, le client souhaite que l'affichage des fiches médecins soit adapté à la nouvelle structure de la base de données.

Afin de rendre le site maintenable, l'agence souhaite aussi que soit développé un back office permettant à l'aide de CRUD d'agir sur les données présente en base.

Le travail sur ce projet a été séquencé de la façon suivante :

- Export de la base donnée distante existante pour pouvoir créer un nouveau projet en local.
- Récupération en local des fichiers du site afin de pouvoir « récupérer » les morceaux de code réutilisables
- Création d'un nouveau projet LARAVEL
- Création de la vue index.php sur la base la nouvelle technologie
- Création du back office sous Laravel.
- Modification de la table USER et création d'une table média afin de pouvoir gérer les photos des médecins avec la librairie Laravel Media.
- Création d'un Controller dédié à la mise à jour de la table User avec la librairie Laravel Excel.
- Création du moteur de recherche
- Création des différentes vues disponibles sur la parties front.

Afin de permettre le suivi du projet, celui-ci a été versionné sur GitHub sur le compte Professionnel de la société.

5 SPECIFICATIONS TECHNIQUES DU PROJET

5.1 Laravel 8

Sources : [Wikipédia.org](https://fr.wikipedia.org/wiki/Laravel), [Laravel.sillo.org](https://laravel.com)

Laravel est un Framework web open-source écrit en PHP respectant le principe modèle-vue-contrôleur et entièrement développé en programmation orientée objet.

Laravel a été créé par Taylor Otwell en juin 2011.

Laravel 8.26 (utilisé dans ce projet), est la dernière version stable publiée ; la LTS étant la version 6 amenée à être remplacée en septembre 2021. Laravel 8.26 nécessite une version minimum 7.3 de PHP.

Laravel initie une nouvelle façon de concevoir un Framework en utilisant ce qui existe de mieux pour chaque fonctionnalité. Par exemple toute application web a besoin d'un système qui gère les requêtes HTTP. Plutôt que de réinventer quelque chose, le concepteur de Laravel a tout simplement utilisé celui de Symfony en l'étendant pour créer un système de routage efficace. De la même manière, l'envoi des emails se fait avec la bibliothèque SwiftMailer. En quelque sorte Otwell a fait son marché parmi toutes les bibliothèques disponibles. Laravel ce n'est pas seulement le regroupement de bibliothèques existantes, c'est aussi de nombreux composants originaux et surtout une orchestration de tout ça.

Laravel disponible, par exemple des fonctionnalités suivantes :

- Un système de routage (RESTful et ressources),
- Un créateur de requêtes SQL et un ORM,
- Un moteur de template, Blade
- Un système d'authentification pour les connexions,
- Un système de validation,
- Un système de pagination,
- Un système de migration pour les bases de données,
- Un système d'envoi d'emails,
- Un système de cache,
- Un système de gestion des événements,
- Un système d'autorisations,
- Une gestion des sessions,
- Un système de localisation,
- Un système de notifications...

L'ensemble des fonctionnalités ne sera pas utilisé dans le projet, mais il a été nécessaire d'ajouter 2 librairies supplémentaires :

- Laravel Media
- Laravel Excel

5.1.1 *Laravel Media*

Sources : spatie.be

Ce package permet d'associer toutes sortes de fichiers aux modèles Eloquent. Il permet au travers d'une table d'associer des fichiers uploadés avec d'autres tables et de les utiliser facilement grâce à des méthodes fournies. Le package permet de gérer les téléchargements vers la médiathèque et d'administrer le contenu d'une collection medialibrary.

5.1.2 *Laravel Excel*

Sources : <https://docs.laravel-excel.com/>

Laravel Excel utilise la bibliothèque PhpSpreadsheet dans le but de simplifier les exportations et les importations de feuilles de calcul

PhpSpreadsheet est une bibliothèque écrite en PHP pur et fournissant un ensemble de classes qui permettent de lire et d'écrire dans différents formats de fichiers de feuilles de calcul, comme Excel et LibreOffice Calc.

Les principales fonctionnalités de Laravel Excel sont de :

- Exporter facilement des collections vers Excel.
- Exporter les requêtes avec segmentation automatique pour de meilleures performances.
- Exporter facilement les vues Blade vers Excel.

- Importer facilement dans les collections.
- Lire le fichier Excel en morceaux.

5.2 Bootstrap 4

Sources : <https://www.pierre-giraud.com/bootstrap-apprendre-cours/introduction/>

Bootstrap est un Framework CSS. Un Framework correspond à un ensemble de bibliothèques regroupées dans un but précis et possédant des règles internes que doivent suivre les utilisateurs.

Bootstrap est donc un ensemble de fichiers CSS et JavaScript fonctionnant ensemble et qui peuvent être utilisés pour créer des designs complexes de manière relativement simple, notamment grâce à la grille en 12 colonnes permettant de gérer très facilement la responsivité.

Le Framework Bootstrap est donc un ensemble de fichiers CSS et JavaScript qui contiennent des règles prédéfinies et qui définissent des composants. Ces ensembles de règles sont enfermés dans des classes et nous n'aurons donc qu'à utiliser les classes qui nous intéressent afin d'appliquer un ensemble de styles à tel ou tel élément HTML.

De plus, Bootstrap utilise également des fichiers JavaScript et notamment des bibliothèques JavaScript externes comme jQuery ou Popper pour définir des composants entiers comme des barres de navigation, des fenêtres modales, etc. qu'on va pouvoir également directement implémenter.

6 REALISATIONS DU CANDIDAT

6.1 Récupération des données existantes.

A partir des codes, fournis par le dirigeant de DM Web, j'ai pu dans un premier temps récupérer les données existantes pour démarrer le projet.

Pour ceci, à l'aide d'un logiciel FTP, j'ai dans un premier temps récupéré en local l'ensemble des fichiers constituant le site initial.

Dans un second temps, après m'être connecté à la base de données (via PhpMyAdmin) du site distant j'ai réalisé un export de la base de données, afin de conserver les données existantes sur le nouveau site.

6.2 Création du projet sous Laravel

L'instanciation d'un projet Laravel se réalise à partir d'une commande Composer :

```
acs@PORT201910-090 MINGW64 /c/laragon/www
$ composer create-project laravel/laravel Zestetik
```

Par cette commande, un nouveau projet utilisant la dernière version de Laravel, soit la 8.26 est instanciée en local. Après téléchargement et installation des différents composants, le projet est prêt à être lancé. Il faut alors aller dans le répertoire créé avec la commande suivante :

```
acs@PORT201910-090 MINGW64 /c/laragon/www
$ cd zestetik
```

Ensuite, l'ensemble des commandes qui seront passées vont commencer par « PHP artisan » qui permet d'appeler les commandes Laravel. La première, qui peut être utilisée est la suivante :


```
acs@PORT201910-090 MINGW64 /c:/laragon/www/zestetikv2 (master)
$ php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
[Sun Apr 18 14:55:55 2021] PHP 7.4.1 Development Server (http://127.0.0.1:8000) started
```

Elle permet de démarrer un serveur PHP de développement permettant de visualiser les résultats du code.

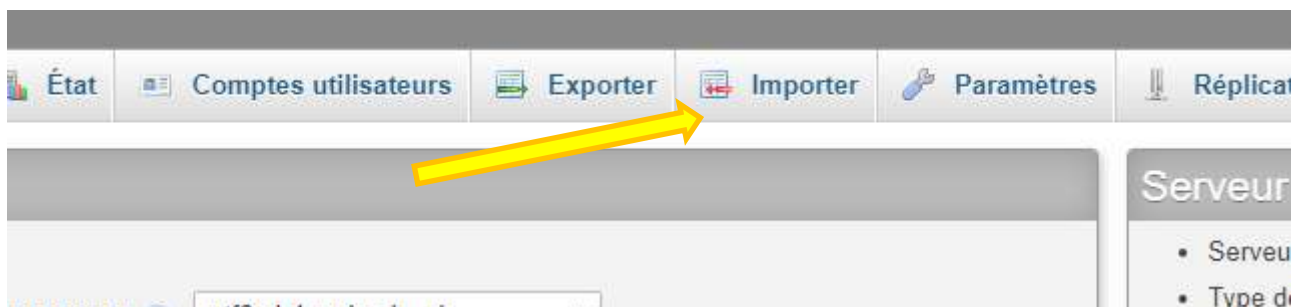
La première étape avant de continuer le code est de modifier le fichier .env qui définit les variables d'environnement de Laravel et en particulier les paramètres de connexion à la base de données.

```
.env
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:Y0xFS38Yeo7tucPk5IeS047Uk6zbXG1AHsJwQnGodAI=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_LEVEL=debug
9
10 DB_CONNECTION=mysql
11 DB_HOST=127.0.0.1
12 DB_PORT=3306
13 DB_DATABASE=c1ubddzes55
14 DB_USERNAME=root
15 DB_PASSWORD=
16
17 BROADCAST_DRIVER=log
```

Les lignes 10 à 15 permettent de définir les paramètres de connexions à la base de données. Ils sont définis une fois la base de données créée avec phpMyAdmin.

6.3 Récupération de la base de données existante

A partir de l'export réalisé préalablement, j'ai pu réaliser un import de la base SQL



En raison de la taille du fichier d'import (28Mo) plusieurs modifications dans le fichier php.ini ont dû être réalisées, en particulier :

- La taille maximum du fichier uploadé


```

; Maximum allowed size for uploaded files.
; http://php.net/upload-max-filesize
upload_max_filesize = 36M

; Maximum number of files that can be uploaded via a single request
max_file_uploads = 20

```

- La taille maximale du fichier accepter en méthode post.

```

; Maximum size of POST data that PHP will accept.
; Its value may be 0 to disable the limit. It is ignored if POST data reading
; is disabled through enable_post_data_reading.
; http://php.net/post-max-size
post_max_size = 48M

; Automatically add files before PHP document.

```

C'est modification ont été trouvé en suivant les erreurs affichées en essayant plusieurs fois l'import.

Une fois l'import réalisé, et le fichier .env modifié, le projet est prêt pour être codé.

6.4 Structuration du projet Laravel.

Laravel respecte strictement l'architecture MVC (Modèle Vue Controller). Aussi il est donc nécessaire de créer par des commandes PHP artisan, les différents éléments de l'architecture MVC :

- PHP artisan make :model User: permet de créer un modèle de class User.
- PHP artisan make :controller UserController --resource : permet de créer un controller User avec l'ensemble des méthodes du CRUD attenantes au modèle User : (index, create, store, show, edit, update, destroy)

Les vues sont créées avec le moteur de Template blade.

Afin de mettre le fonctionnement de l'ensemble de l'architecture, un fichier de routes appelé web.php permet d'écrire l'ensemble des routes nécessaires au fonctionnement du site.

```

/*
Auth::routes();

Route::get('/', [HomeController::class, 'index'])->name('home');
Route::get('pages/faq', [HomeController::class, 'faq'])->name('faq');
Route::get('pages/cgu', [HomeController::class, 'cgu'])->name('cgu');
Route::get('soins/dermatologue', [SearchController::class, 'search_dermatologue'])->name('search_dermatologue');
Route::get('soins/chirurgien-esthetique', [SearchController::class, 'search_chirurgien'])->name('search_chirurgien');
Route::get('soins/medecin-anti-age', [SearchController::class, 'search_antiage'])->name('search_antiage');
Route::get('soins/medecin-esthetique', [SearchController::class, 'search_medecin'])->name('search_medecin');
Route::get('medecin/{location}', [SearchController::class, 'search_by_location'])->name('search_location');

Route::prefix('user')->name('users.')->group(function () {
    Route::get('search', [SearchController::class, 'search'])->name('search');
    Route::get('{user}/show', [HomeController::class, 'show'])->name('show');
    Route::post('store_contact', [FrontContactController::class, 'store'])->name('store_contact');
});

//Route de l'administration

```

La structure d'une route est la suivante :

- La class Route
- La méthode http utilisé par la route (get, post, put, delete)
- L'URI de la route qui sera affiché dans l'URL du site,

- Le Controller appelé par la route et la fonction publique concernée.
- Enfin le nom donné par la route qui permettra de l'appeler dans le code

L'ensemble des routes peut être regroupé, préfixé et éventuellement être concerné par un middleware (expliqué plus tard).

Pour ce projet, j'ai été amené à instancier les modèles suivants :

- User : ensemble des comptes stockés dans la base
- Speciality : ensembles des spécialités des médecins
- Contact : ensemble des contacts enregistrés par le site
- Country ; table des pays possible pour la table user ;
- Healthcare : table des soins possibles.

Les modèles m'ont permis en autres de définir les relations One to Many ou Many to Many entre chaque table mais aussi de déclarer les champs de tables qui peuvent faire l'objet d'une assignation de masse.

Par exemple la relation Many to Many entre la table User et la table Healthcare est codé de la façon suivante ; la table de jointure ayant été créée dans la première version du projet :

```
public function healthcares()
{
    return $this->belongsToMany(Healthcare::class);
}
```

Coté modèle User.

```
public function users()
{
    return $this->belongsToMany(User::class);
}
```

Coté modèle Healthcare

J'ai de même instancié un certain nombre de Controller, organisé selon 4 répertoires :

- FrontControllers : ensemble des Controllers du front
 - o HomeController : gestion de la homepage (page d'accueil) et de la page détail médecin
 - o SearchController : ensemble des fonctions de recherche
 - o ContactController ; fonction d'ajout un nouveau contact
- AdminController : ensemble des Controller du Backoffice (soumis à autorisation)
 - o UserController : CRUD de la table User
 - o ContactController CRUD de la table Contact
 - o HealthcareControlller : CRUD de la table Healthcare
 - o DashBoradController : Affichage de la page d'accueil du Back Office
- Auth : ensemble des Controller de l'authentification (expliqué plus tard)
- DoctorController : ensemble des Controller liés au profil Docteur de la table User.
 - o UserController : ensemble des fonctions de mise à jour du profil médecin par l'utilisateur
 - o ContactController : ensemble des fonctions de gestion des contacts pour un médecin.

Dans la suite de ce rapport je vais exposer certains points de réalisations particuliers :

- La structuration de la partie Front.
- La gestion de l'authentification
- La modification de la table User et l'import des données mise à jour fournies par le client.
- L'utilisation de Laravel Media pour gérer les photos des médecins.
- La sécurisation des routes
- La création d'un CRUD
- L'ensemble du code du moteur de recherche.

6.5 Structuration de la partie Front

Laravel exploite le moteur de Template Blade pour la génération des vues. Comme évoqué au début du rapport, j'y ai ajouté le Framework Bootstrap afin de pouvoir exploiter la grille sur 12 colonnes. La partie front du site web est constitué d'assez peu de vues :

- 3 vues statiques pour la home page, la FAQ et les CGU
- 2 vues dynamiques pour le résultat de la recherche et l'affichage du détail d'un médecin.

Mon travail dans cette partie a consisté à récupérer les parties de code existantes sur le site actuel, le client ne souhaitant de changement d'aspect de son site.

Dans cette partie, je vais vous présenter l'organisation du Template que j'ai créé, la barre de navigation, une vue statique et une vue dynamique

6.5.1 Le Template de base

Laravel utilise le générateur de Template Blade pour l'affichage des vues. La première chose à mettre en place est un Template qui sera ensuite appelé dans chaque vue afin de structurer la page.

Un Template de base reprend d'architecture de la page HTML avec l'ensemble des balises structurantes et dans lequel j'ai placé le code qui ne changera pas d'une page à l'autre et des éléments de code spécifiques à Blade pour injecter le contenu de la page

```
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkv"  
  
<title>@yield('title')</title>  
  
</head>  
  
<body class="home fp-viewing-false" style='overflow:visible; height:initial'>  
    @if ($message= Session::get('success'))  
        <div class="alert alert-success alert-block">  
            <button type="button" class="close" data-dismiss="alert">x</button>  
            <strong>{{$message}}</strong>  
        </div>  
    @endif  
  
    <header class="header-site navbar-fixed-top">  
        <div class="container">  
            @include('incs/menu-nc')  
        </div>  
    </header>  
  
    <main class="container-fluid">  
        @yield('content')  
    </main>  
    <section class="block-logo-marques">...</section>  
    <section class="block-contact-expert">...</section>  
  
    </main>  
    <footer>  
        <div class="footer-social" style="height:150px!important">  
            <div class="container">  
                <div class="block-nl col-12 col-lg-6 d-none d-lg-block">
```

La partie de code ci-dessus permet de visualiser ces zones d'injection.

- Le champ `@yield('title')` entouré de la balise `<title>` permet de modifier le titre de chaque page avec un contenu (`@content`) de la page à afficher.
- Le code `@include('incs/menu-nc')`, permet d'inclure la navbar, codée dans un fichier à part, dans la balise `<header>` du fichier.
- Le champ `@yield('content')` contenu dans la balise `<main>` va permettre d'injecter l'ensemble du contenu de la page.

Le fichier Template se termine par un footer reprenant le contenu du site actuel.

6.5.2 La barre de navigation

La barre de navigation comme l'ensemble du site doit être responsive. Les vues ci-dessous présentent la barre de navigation dans différents états :

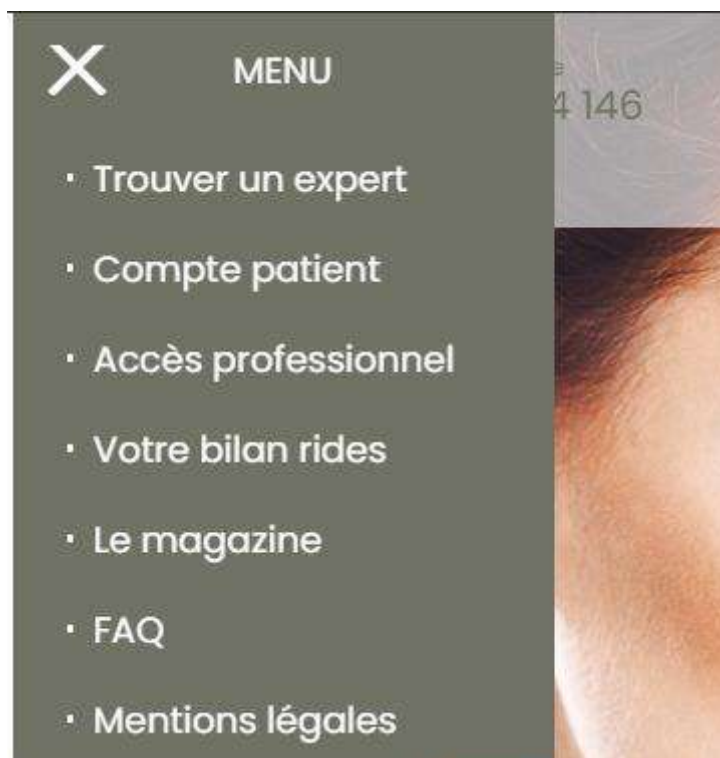
- Sur mobile



- Sur ordinateur de bureau



- Quand le menu burger est déployé.



Le code se présente de la façon suivante :

```

<div class="row justify-content-between" style="display:block">
  <div class="col-12 col-md-12 col-lg-4 d-flex justify-content-between order-1 order-lg-0">
    <div class="block-menu-site">
      <div class="menu-title pull-left">
        <button class="hamburger hamburger-x pull-right">
          <span>toggle menu</span>
        </button>
        <span>menu</span>
      </div>

      <div class="block-menu">
        <div class="text-center">Menu</div>
        <ul class="nav">
          <li>
            <a href="{{route('home')}}" title="">Trouver un expert</a>
          </li>
          <li>
            <a href="{{route('login')}}" title="">Compte patient</a>
          </li>
          <li>
            <a href="{{route('login')}}" title="">Accès professionnel</a>
          </li>
          <li>
            <a href="http://bilan-rides.zestetik.fr/" target="_blank" title="">Votre
          </li>

```

Il est basé sur la grille Bootstrap dans une <div> de classe row avec dedans des <div> de class « col » ajustées en fonction de la taille de l'écran. J'ai aussi utilisé la classe order de Bootstrap pour modifier l'ordre des blocs.

L'ouverture et la fermeture du menu est géré par un bout de code JQuery qui existait déjà sur le site qui ajoute une classe 'is-active' en toggle

```

$(document).ready(function() {
  $( ".menu-title" ).click(function() {
    $( this ).parents('.block-menu-site').toggleClass( "is-active" );
  });
});

```

6.5.3 Une vue statique : la home page

La home page est une page statique qui doit afficher le moteur de recherche d'un médecin et une quantité importante d'information statiques qui n'ont pas été modifiée depuis la création du site.

```

@extends('template')

@section('title')
Zestetik, Téléconseil et information en esthétique médicale – médecine, chirurgie esthétique, dermatologie : Prise de
rendez-vous médecins, chirurgiens, dermatologues, conseils, news, 24H/24 et 7j/7
@endsection

@section('content')

```

Comme expliqué dans la partie Template, on retrouve dans le code de la page Home les éléments suivants :

- Le champ @extends('template') qui permet d'ajouter le Template à la page
- Le champ @section('title'), fermé par un champ @endsection qui précise le nom de la page
- Le champ @section('content') qui contient l'ensemble du code destiné au corps de la page web.

Sur cette page, je vais vous présenter le moteur de recherche d'un médecin.

Les captures d'écrans suivantes présentent le moteur de recherche en 2 résolutions

- Sur ordinateur de bureau

- En mobile

Ce code de cette barre de recherche est présenté ci-dessous

```
<section class="block-search-page">
  <div class="container d-flex justify-content-center">
    <div class="row w-100 justify-content-between">
      <div class="col-md-12 col-lg-2 p-0">
        Rencontrez votre<br>spécialiste
      </div>
      <form action="{{route('users.search')}}" class="col-md-12 col-lg-10 d-flex p-0 align-items-center">
        <div class="form-group p-0">
          <input type="text" class="form-control p-0" placeholder="Médecin"
            name="name-search">
        </div>
        <span class="or">ou</span>
        <div class="form-group p-0 search_group">
          <select name="speciality_id" id="speciality" class="form-control p-0">
            <option value="">Spécialité</option>
            @foreach ($specialities as $speciality)
              <option value="{{ $speciality->id }}">{{ $speciality->name }}</option>
            @endforeach
          </select>
        </div>
        <div class="form-group p-0 search_group">
          <input class="form-control p-0" id="adresse-zestetik-search" type="text" name="lieu" value=""
            placeholder="Code postal">
        </div>
        <div class="form-group p-0 search_group">
          <select name="country" class="form-control p-0" placeholder="Pays">
            <option value="">Pays</option>
            <option value="fr" selected>France</option>
            <option value="gb">Royaume-Uni</option>
            <option value="ch">Suisse</option>
            <option value="be">Belgique</option>
            <option value="lu">Luxembourg</option>
            <option value="nl">Pays-Bas</option>
            <option value="es">Espagne</option>
          </select>
        </div>
        <button type="submit" class="btn btn-primary" style='margin-bottom:12px'>
          <span>Rechercher</span>
          <span class="zestetik-search"></span>
        </button>
      </form>
    </div>
  </div>
</section>
```



Il s'agit d'un formulaire qui appelle la route 'users.search' en méthode 'get' qui ne modifiant pas la base de données, n'a pas besoin d'un champ @csrf. Il est ensuite constitué de 2 inputs de type texte et de 2 selects dont un généré à partir de données issues de la table specialities dans la base de données.

La requête est transmise au Controller SearchController et appelle la méthode 'search' présentée, plus tard dans le rapport et va générer la vue dynamique détaillée de suite.

6.5.4 Une vue dynamique : le résultat d'une recherche

Une fois la requête passée, le Controller retourne une vue dynamique qui sera construite à partir d'une collection \$doctors et d'une collection \$specialities. La vue se présente :


- Sur ordinateur de bureau



L'expert esthétique pour trouver le soin ou l'intervention qui vous correspond

22 résultats pour votre recherche

1 Dr. Philippe Abimelech



Dermatologue
121 Rue Caulaincourt
75018 Paris
France

Appeler

Demande de devis

Itinéraire

voir son profil

2 Dr. JEAN-ALBERT AMAR



Dermatologue
12 RUE TRONCHET
75008 PARIS
France

Appeler

Demande de devis

Itinéraire

voir son profil

- En mobile

LES EXPERTS EN ESTHETIQUE MEDICALE

2 Dr. JEAN-ALBERT AMAR



Dermatologue
12 RUE TRONCHET
75008 PARIS
France

Appeler

Demande de devis

Itinéraire

Le bandeau affichant le nombre de résultats de la recherche est généré par le code suivant :


```

<section class="block-filter-page-result">
    <div class="container">
        @if ($doctors->total())>0
        <h3>{{ $doctors->total() .' résultats pour votre recherche' }}</h3>
        @else
        <h3>Pas de résultat pour votre recherche</h3>
        @endif
    </div>
</section>

```

La méthode 'total()' est appliquée sur la collection \$doctors pour connaître le nombre de résultats et l'afficher dans une balise <h3> avec une concaténation dans une syntaxe Blade ({{ }}) ;

La collection \$doctors est ensuite parcourue avec une boucle Foreach pour générer les fiches médecins selon le code suivant :

```

<div class="row">
    <?php $i=0?>
    @foreach ($doctors as $item)

    <?php $i++?>
    <div class="col-12 col-lg-6 px-3">
        <div class="expert-item desc-expert p-4">
            <div class="row align-items-center">
                <div class="col-12 col-lg-8">
                    <h2 class="heading-name">
                        <span class="number">{{ $i }}</span>
                        <span class="name"> {{ 'Dr. ' . $item->firstname . ' ' . $item->lastname }}</span>
                    </h2>
                    <div class="row justify-content-center justify-content-sm-between align-items-start">
                        <div class="col-12 col-lg-5 d-flex justify-content-center">
                            @if (count($item->getMedia('doctors'))>0)
                            
                            @else
                            
                            @endif
                        </div>
                        <div class="col-12 col-lg-7 d-flex align-items-center flex-column">
                            <h4 class="speciality">{{ $item->speciality->name }}</h4>
                            <p class="adress">{{ $item->adress1 }}</p>
                            <p class="adress">{{ $item->postcode . ' ' . $item->city }}</p>
                            <p class="adress">{{ $item->country->name_fr }}</p>
                        </div>
                    </div>
                </div>
                <div class="col-12 col-lg-4">
                    <a href="#" class="btn btn-default btn-lg">Appeler</a>
                    <button type="button" class="btn btn-default btn-lg" style="width:100%" data-toggle="modal"
                        data-target="{{ '#estimate.' . $item->id }}">Demande de devis</button>
                    <a href="#" class="btn btn-default btn-lg" data-toggle="modal" data-target="{{ '#itinerary.' . $item->id }}">Itinéraire</a>
                    <a href="{{ route('users.show', $item->id) }}" class="more-info btn btn-primary btn-lg">voir son
                        profil</a>
                </div>
            </div>
        </div>
    </div>
    </div>
</div>

```

La fiche médecin est construite en utilisant la grille Bootstrap. La gestion de l'image du médecin est réalisée avec la librairie LaravelMedia qui sera détaillée au point 6.8 du rapport ; la base ayant été modifiée et donc vide, seul l'image par défaut des médecins est affichée.

En l'état actuel du travail demandé, les boutons permettent :

- D'afficher une vue détaillée du médecin avec l'ensemble des données qu'il aura complété
- D'ouvrir une modal (« demande de devis ») pour générer un formulaire dont le résultat sera disponible dans le back-office du médecin concerné.

6.6 Gestion de l'authentification

Le projet Laravel est riche de plusieurs packages disponibles permettant de gérer l'authentification. Après échange avec mon maître de stage, j'ai utilisé un package Laravel UI qui présente l'avantage d'être consolidé depuis plusieurs versions.

6.6.1 Instanciation du package

L'instanciation de Laravel UI se fait par une commande Composer :

```
composer require laravel/ui
```

Une fois le package téléchargé et installé, une simple commande PHP artisan permet de choisir comment l'authentification doit être gérée.

Dans mon cas, j'ai choisi une authentification complète (Login et Création de compte) en utilisant Bootstrap soit la commande :

```
php artisan ui bootstrap --auth
```

Cette commande va créer l'ensemble du système d'authentification, il reste juste à adapter le code pour qu'il respecte les besoins du site.

6.6.2 Adaptation des vues

Afin d'adapter les vues, il m'a juste suffi de préciser au début du fichier login.blade.php que cette vue utilise le Template 'template' afin que le code s'adapte au design du site.

```
@extends('template')

@section('content')
<div class="container mb-5" style="margin-top:150px">
  <div class="row justify-content-center">
    <div class="col-12 col-md-10">
      <div class="card">
        <div class="card-header">Accédez à votre espace personnel</div>
        <div class="card-body">
          <form method="POST" action="{{ route('login') }}">
            @csrf
            <div class="form-group row">
              <label for="email" class="col-md-4 col-form-label text-md-right">{{ __('E-Mail Address') }}</label>
              <div class="col-md-8">
                <input id="email" type="email" class="form-control @error('email') is-invalid @enderror" name="email" value="{{ old('email') }}">
                @error('email')
                  <span class="invalid-feedback" role="alert">
                    <strong>{{ $message }}</strong>
                  </span>
                @enderror
              </div>
            </div>
          </form>
        </div>
      </div>
    </div>
  </div>
</div>
```

6.6.3 Modification du code du LoginController

Le login Controller devant pouvoir faire la distinction entre User, Doctor et Admin, le code a dû être profondément modifié dans le Controller LoginController :

```

namespace App\Http\Controllers\Auth;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\Providers\RouteServiceProvider;
use Illuminate\Foundation\Auth\AuthenticatesUsers;

class LoginController extends Controller
{
    /**
     * Login Controller
     *
     * This controller handles authenticating users for the application and
     * redirecting them to your home screen. The controller uses a trait
     * to conveniently provide its functionality to your applications.
     */

    use AuthenticatesUsers;

    protected function authenticated(Request $request, $user) {
        if ($user->role == 'admin') {
            return redirect(RouteServiceProvider::DASHBOARD);
        }
        elseif ($user->role == 'patient') {
            return redirect(RouteServiceProvider::HOME);
        }
        elseif($user->role == 'doctor'){
            return redirect(RouteServiceProvider::DOCTOR);
        }
        else {
            return redirect(RouteServiceProvider::HOME);
        }
    }
}

```

J'ai donc mis en place une imbrication de boucle If-Else permettant d'affecter une constante différente en fonction du profil identifié dans le rôle du User au RouteServiceProvider, modèle chargé de router les User après authentification.

Les constantes sont définies au début du modèle RouteServiceProvider de la façon suivante :

```

class RouteServiceProvider extends ServiceProvider
{
    /**
     * The path to the "home" route for your application.
     *
     * This is used by Laravel authentication to redirect users after login.
     *
     * @var string
     */
    public const HOME = '/';
    public const DASHBOARD = '/admin/gestion';
    public const DOCTOR = '/doctor/index';
}

```

Chaque constante donne le chemin désiré pour chaque profil.

6.7 Modification, Import et mise à jour de la table user

Evolution demandée par le client, la modification de la table User ainsi que sa mise à jour a nécessité plusieurs opérations :

- Modification des champs de la table User
- Création d'une méthode permettant l'import d'un fichier Excel, contenant soit des Users à mettre à jour, soit des nouveaux Users

6.7.1 Modification de la table User

Laravel permet de réaliser de multiples migrations pour créer mais aussi modifier des tables.

Dans ce cas précis, j'ai exécuté plusieurs commandes PHP artisan make :migration afin de modifier les champs de la table

```
<?php

use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class AddCentreToUser extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->string('centre')->after('lastname')->nullable()->default('null');
            $table->integer('speciality_2_id')->after('speciality_id')->nullable();
            $table->string('page_instagram',250)->after('page_facebook')->nullable()->default('null');
            $table->dateTime('created_at')->default(DB::raw('CURRENT_TIMESTAMP'));
            $table->dateTime('updated_at')->nullable();
        });
    }
}
```

L'exemple de code ci-dessus, présente une modification de la table User pour ajouter 5 champs :

- Le champ « Centre) de type Varchar(255), placé après le champ « lastname » qui peut être nul et à pour valeur par défaut « null »
- Le champ « speciality_2_id » de type Integer placé après le champ « speciality_id »
- Le champ 'page_instagram de type varchar(250), placé après le champ « Facebook » » qui peut être nul et à pour valeur par défaut « null »
- Les champs « created_at » et « updated_at » qui sont des champs date de format TimeStramp qui permettent de savoir quand l'enregistrement a été créé et modifié

Ainsi de nombreux champs ont été manipulés pour arriver à la structure de la table voulue par le client et de nombreux champs ont été aussi enlevés avec le code suivant :

```

<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class Drop2FromUser extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn(['adress2','civility','photo','photo_dir','ranking','cartevitale','cmuame','teldomicile',
            'telwork','datenaissance','profil_zestetik','compte_twitter','profil_google',
            'presentationzes','presentation_en','presentationzes_en','oder_number','created','modified']);
        });
    }
}

```

6.7.2 Import et mise à jour des données

Afin de réaliser la mise à jour de la base de données, j'ai utilisé une librairie supplémentaire Laravel-Excel. Cette librairie s'installe via une commande composer :

```
1 composer require maatwebsite/excel
```

Une fois installée, pour créer un import de mes Users j'ai passé la commande suivante :

```

acs@PORT201910-090 MINGW64 /c/laragon/www/zestetikV2 (master)
$ php artisan make:import UserImport --model=User

```

Cette commande m'a créé un fichier UserImport.php qu'il a fallu modifier afin de réaliser la commande voulue, savoir vérifier si le User existe, s'il existe, le mettre à jour sinon le créer.

Laravel_Excel, propose 2 modèles pour réaliser les imports : ToModel ou ToCollection. Le modèle ToCollection permet de réaliser des tests ligne par ligne dans le fichier ce que ne permet pas ToModel qui réalise une assignation de masse selon le modèle User avec Eloquent. J'ai donc créé une méthode utilisant ToCollection pour faire l'import.

Le code est présenté en annexe.

Il est constitué d'une boucle Foreach parcourant chaque ligne du fichier. Chaque email est recherché dans la base

```

$dejaenbase=User::where('role','doctor')
->where('email',$row[9])
->first();

```

Une condition If Else est ensuite réalisée : soit l'e-mail existe et on procède à l'update,

```

$dejaenbase->update([
    'imported'=>true,
    'speciality_2_id'=>null,
    'lastname'=> $row[0],
    'firstname'=> $row[1],
    'centre'=>$row[4],
    'phone'=>$row[8],
    'adress1'=> $row[5],
    'postcode'=> $row[6],
    'city'=> $row[7],
    'page_facebook'=> $row[10],
    'profil_linkedin'=> $row[12],
    'page_instagram'=> $row[11],
    'site_internet'=> $row[13],
    'chaine_youtube'=> $row[14],
]);

```

Soit c'est un nouveau médecin est on procède à une création

```

$password=$row[0].rand(100,5000);

$user=User::create([
    'role'=>'doctor',
    'lastname'=> $row[0],
    'firstname'=> $row[1],
    'centre'=>$row[4],
    'phone'=>$row[8],
    'country_code'=>'fr',
    'speciality_id'=>$speciality,
    'speciality_2_id'=>$speciality2,
    'adress1'=> $row[5],
    'postcode'=> $row[6],
    'city'=> $row[7],
    'email'=> $row[9],
    'page_facebook'=> $row[10],
    'profil_linkedin'=> $row[12],
    'page_instagram'=> $row[11],
    'site_internet'=> $row[13],
    'chaine_youtube'=> $row[14],
    'password'=> Hash::make($password),
    'imported'=>true,
]);

```

Une condition switch case permet aussi de traiter la codification utilisée par le client pour les spécialités des médecins et de les convertir en code adapté à la base de données.

Le mot de passe du nouveau médecin est créé en concaténant son nom et un nombre aléatoire compris entre 100 et 5000.

Cette fonction d'import est appelée depuis le Controller AdminControllers/UserController avec la fonction import, qui précise en plus le nom du fichier à importer ; celui -ci ayant été placé dans le dossier public par défaut.

```

class UserController extends Controller
{
    public function import()
    {
        ini_set('memory_limit', '1024M');

        Excel::import(new UsersImport, 'import.xlsx');

        return redirect('/')->with('success', 'All good!');
    }
}

```

6.8 Utilisation de Laravel Media

Afin de pouvoir gérer les photos des médecins sur le site, mon maitre de stage m'a demandé d'utiliser la librairie Laravel-Media.

Pour ce faire, comme les librairies précédentes, elle s'instancie via une commande composer :

```
composer require "spatie/laravel-medialibrary:^9.0.0"
```

Ensuite la documentation demande de publier une migration par une commande PHP artisan vendor:publish qui permet de générer une table media spécifique pour y stocker les relations entre les fichiers et le modèle qui doit se servir de Laravel Media

La migration ainsi créée est déployée par une commande PHP artisan migrate.

Cette librairie doit être ensuite associée à une classe, dans mon cas la classe User par l'ajout du code suivant :

```

class User extends Authenticatable implements HasMedia
{
    use HasFactory, Notifiable, InteractsWithMedia;
}

```

La librairie peut être utilisée et il suffira de l'appeler pour enregistrer un media :

```

if (isset($request['profile_img'])) {
    $file = $request->file('profile_img');
    $user->addMedia($file)
        ->preservingOriginal()
        ->toMediaCollection('doctors');
}

```

Ou afficher un media associé à un User

```

@if (count($item->getMedia('doctors')))

@else

@endif

```

6.9 Sécurisation des routes.

La sécurisation des routes, et donc la limitation d'accès à certains contenus soumis à authentification se réalise dans Laravel par l'intermédiaire d'un Middleware (Logiciel médiateur en français)

Afin de sécuriser l'ensemble des routes de l'application j'ai créé 2 middlewares :

- Le middleware de l'administrateur : AdminMiddleware
- Le middleware des profils Docteurs : DoctorMiddleware

La création d'un middleware se fait par la commande :

```
acs@PORT201910-090 MINGW64 /c:/laragon/www/zestetikV2 (master)
$ php artisan make:middleware AdminMiddleware
```

Celui crée un fichier AdminMiddleware.php dans le répertoire Middleware qu'il suffit de modifier pour obtenir la « médiation » recherchée. Par exemple pour l'AdminMiddleware

```
<?php

namespace App\Http\Middleware;

use Closure;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class AdminMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request
     * @param \Closure $next
     * @return mixed
     */
    public function handle(Request $request, Closure $next)
    {
        $user=Auth::user();
        if(!$user){
            return redirect()->route('home');
        }

        if($user->role !== User::ADMIN_ROLE){
            return redirect()->route('home');
        }
        return $next($request);
    }
}
```

Dans ce cas précis, il y a 2 conditions successives, la première vérifie l'existence d'un utilisateur authentifié ; si ce n'est pas le cas il est renvoyé à la route 'home', dans ce projet, la page d'accueil.

Ensuite s'il est authentifié mais que son rôle n'est pas un rôle Admin (Constante définie dans la classe User), il est de même retourné vers la page d'accueil. Sinon le Middleware autorise l'accès à la route.

Le middleware ainsi créé doit être déclaré dans le fichier kernel.php du répertoire Controller

```
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    'admin' => \App\Http\Middleware\AdminMiddleware::class,
    'doctor' => \App\Http\Middleware\DoctorMiddleware::class,
];
```

Le middleware ainsi appelé 'admin' peut ensuite être ajouté aux routes dans le fichier web.php

```
//Route de l'administration
Route::middleware('admin')->prefix('admin')->name('admin.')->group(function () {
    Route::get('import', [UserController::class, 'import']);
    Route::delete('deleteafterimport', [UserController::class, 'deleteafterimport'])->name('deleteafterimport');

    Route::get('gestion', [DashBoardController::class, 'index'])->name('gestion');
    Route::get('crud_doctors', [UserController::class, 'doctorsindex'])->name('crud_doctors');
    Route::get('crud_patients', [UserController::class, 'patientsindex'])->name('crud_patients');
    Route::delete('{user}/delete_patient', [UserController::class, 'destroy_patient'])->name('delete_patient');
    Route::get('{user}/edit', [UserController::class, 'edit'])->name('edit');
    Route::put('{user}/update', [UserController::class, 'update'])->name('update');
    Route::get('crud_contacts', [ContactController::class, 'index'])->name('crud_contacts');
    Route::delete('{contact}/delete_contact', [ContactController::class, 'destroy'])->name('delete_contact');
    Route::get('{user}/show_doctor', [UserController::class, 'show'])->name('show_doctor');
    Route::delete('{user}/destroy', [UserController::class, 'destroy'])->name('destroy');
    Route::get('search', [UserController::class, 'search_doctor'])->name('search_doctor');
    Route::get('healthcare/index', [HealthcareController::class, 'index'])->name('healthcares.index');
    Route::put('healthcare/{healthcare}/update', [HealthcareController::class, 'update'])->name('healthcares.update');
    Route::post('healthcare/store', [HealthcareController::class, 'store'])->name('healthcares.store');
});
```

Toutes les routes présentées ci-dessus sont groupées avec pour Middleware 'Admin'. Ceci sous-entend que les 16 routes ne seront accessibles qu'à un utilisateur authentifié ayant le rôle Admin

6.10 Création des CRUD

Pour la partie Backoffice, il m'a été demandé de mettre en place des CRUD (Create Read Update Delete) pour les tables disponibles dans le projet.

J'ai ainsi créé les vues suivantes :

- CRUD Médecins avec un moteur de recherche pour un accès rapide
- CRUD Patients pour pouvoir simplement les supprimer
- CRUD Messages pour pouvoir simplement les supprimer
- CRUD Soins avec la création et l'édition (pas de suppression pour ne pas altérer les relations la base de données.

Dans cette partie je vais détailler la création du CRUD Médecin.

Liste des médecins actifs

Nom, Prénom, Ville

Choisir la spécialité

Rechercher

Nom	Prénom	Spécialité	Adresse	CP	Ville	Pays	Action	
De Goursac	Catherine	Médecin esthétique	81 Avenue Niel	75017	Paris	fr	Editer	Supprimer
Slous	Maryse	Médecin esthétique	106 Avenue Victor Hugo	75016	Paris	fr	Editer	Supprimer
BERSAND	Gerard	Médecin anti-âge	57, Promenade des Anglais	06000	NICE	fr	Editer	Supprimer
BARON	NADINE	Médecin esthétique	4 Rue Julien Forgues	31100	TOULOUSE	fr	Editer	Supprimer
Chardonneau	Jean-Marc	Médecin esthétique	5 Rue de Budapest	44000	Nantes	fr	Editer	Supprimer

L'ensemble des méthodes gérant ce CRUD, sont regroupées dans le Controller AdminControllers/UserController. Elles sont soumises au middleware 'Admin'

J'ai ainsi créé les méthodes suivantes :

- doctorsindex : requête de la table user retournée à la vue cruddoctor
- edit : requête sur la table User pour éditer le médecin en modification
- Update : requête en mode put pour mettre à jour les données du User médecin
- Destroy ; requête en mode delete pour effacer une entrée
- search_doctor : requête en mode get pour rechercher un médecin ou un ensemble de médecin selon une spécialité

6.10.1 Méthode doctorsindex

```
public function doctorsindex()
{
    $doctors = User::where('role', 'doctor')
        ->where('imported', true)
        ->paginate(30);
    return view('admin.cruddoctors', [
        'doctors' => $doctors,
        'specialities' => Speciality::all(),
    ]);
}
```

Cette méthode utilise Eloquent pour réaliser une requête sur la table users. Elle crée une collection de class USER contenant les utilisateurs ayant le rôle 'doctor' et qui ont été importé, ou mis à jour, dans la base de données. Les données sont paginées par 30 lignes ce qui permettra de mettre en place une navigation dans les pages du CRUD. L'objet \$doctors est retourné à la vue admin.cruddoctors avec un autre objet, \$specialities, contenant l'ensemble des valeurs de la table 'specialities'.

La vue générée utilise un Template spécifique 'admin'. La vue comprend un moteur de recherche et un tableau CRUD affichant la liste des médecins et les actions « Editer » et « supprimer »

Le moteur de recherche est présenté dans la capture d'écran ci-dessous, Il est constitué d'un champ input et d'un champ select construit à partir de l'objet \$specialities. Comme expliqué précédemment, la vue utilise le générateur de template Blade. Les deux champs font partie d'un div <form> appelant la route 'admin.search.doctor', qui sera détaillée plus loin.

```

@extends('template_admin')

@section('title')
Adminstration Zestetik
@endsection

@section('content')


<h2>Liste des m  d  cins actifs</h2>
</div>

<div class="row my-2">
    <form action="{{route('admin.search_doctor')}}" class="d-flex">
        <div class="form-group mb-0 mr-2">
            <input type="text" name="q" class="form-control" placeholder="Nom, Pr  nom, Ville" value="{{request()->q ?? ''}}">
        </div>
        <div class="form-group mb-0 mr-2">
            <select name="speciality_id" class="form-control">
                <option value="">Choisir la sp  cialit  </option>
                @foreach ($specialities as $speciality)
                <option value="{{ $speciality->id }}">{{ $speciality->name }}</option>
            @endforeach
        </select>
    </div>
    <button type="submit" class="btn btn-primary">Rechercher</button>
</form>
</div>
<div class="row w-100">
    <table class="table">
        <thead class="thead-light">
            <tr>...
        </tr>
        </thead>
        <tbody>


```

Le tableau CRUD est de m  me r  alis   avec du code Blade, il s'agit d'une boucle Foreach, qui pour chaque ligne de l'objet Doctors, va permettre d'afficher les donn  es d'un m  decin :

- Nom,
- Pr  nom,
- Sp  cialit  , s'il en a une de d  finir (condition @if @else @endif)
- Adresse compl  te du m  decin

Chaque ligne est compl  t  e par 2 actions :

- Editer : qui redirige vers la route 'admin.edit' avec la valeur de l'ID du m  decin cibl  
- Supprimer : qui ouvre une modal de confirmation de la suppression de la ligne.

La suppression, dans Laravel, est une route particuli  re, utilisant la m  thode http, delete. Pour ceci, j'ai cr     un formulaire, appelant la route 'admin.destroy'. Dans ce formulaire, je passe obligatoirement la commande '@csrf' qui, par un token d'authentification, va s  curiser la requ  te de modification de la base de toute forme d'attaque. Par ailleurs, une balise <form> ne prenant que des m  thodes 'get' ou 'post', il faut pr  ciser, par la commande blade @method, que ce formulaire utilisera la m  thode 'delete'.

Le code est pr  sent   ci-dessous et il va g  n  rer jusqu'  30 lignes dans la table et les 30 modales associ  es    la suppression.


```

@extends('template_admin')

@section('title')
Administration Zestetik
@endsection

@section('content')


# {{ 'Edition : ' . $user->firstname . ' ' . $user->lastname }}



<form method="POST" action="{{route('admin.update', $user->id)}}" enctype="multipart/form-data">
  @method("PUT")
  @csrf
  <input hidden name='id' value="{{ $user->id }}">
  <div class="row">
    <div class="col-12 col-sm-6">
      <div class="form-group">
        <label>Nom</label>
        <input type="text" name="lastname" class="form-control" value="{{ $user->lastname }}" />
      </div>
    </div>
    <div class="col-12 col-sm-6">
      <div class="form-group">
        <label>Prénom</label>
        <input type="text" name="firstname" class="form-control" value="{{ $user->firstname }}" />
      </div>
    </div>
  </div>
</div>
</div>


```

La vue edit_user est un formulaire permettant de mettre à jour les données du user. Pour ceci, l'ensemble des inputs sont placés dans une balise <form> dont la méthode post est modifiée par la commande @method en méthode 'put', qui va correspondre la méthode http utilisée par la route 'admin.update'. Comme précédemment, la requête SQL est sécurisée par l'envoi un token via la commande @csrf.

L'ensemble des champs sont positionnés dans une grille Bootstrap, un champ devant utilisé la totalité des 12 colonnes en mobile (col-12) et être positionné 2 champs par ligne à partir de la tablette (col-sm-6).

Sur chaque input, la valeur par défaut sera récupérée dans l'objet \$user passée à la vue comme par exemple

`value="{{ $user->lastname }}" />` qui permet d'assigner le nom du user au champ name='lastname'.

Le site prévoit que les médecins peuvent afficher quels soins ils peuvent proposer aux patients. L'ensemble des soins possibles est stocké dans une table healthcares l'affichage final doit s'afficher comme suit : les soins classés selon des catégories avec des checkbox actives si le médecin a déjà complété sa fiche.

Sélectionner les prestations possibles

Anti-âge

- ☒ Bilan anti-âge et hormonal
- ☐ Hormonothérapie
- ☐ Sexualité
- ☐ Nutrition
- ☐ Immunothérapie
- ☐ Cosmécéutiques
- ☐ Compléments alimentaires

Chirurgie esthétique

- ☐ Augmentation ou réduction mammaire
- ☐ Chirurgie du nez (rhinoplastie)
- ☐ Chirurgie des paupières (blépharoplastie)

Autre soin

- ☐ Injections de graisse autologue (fat-grafting) dans le visage
- ☐ Radiofréquence
- ☒ Ultrasons focalisés
- ☐ Rajeunissement des mains (surface et comblement)

Esthétique génitale

- ☐ Pénoplastie
- ☐ Sécheresse vaginale
- ☐ Augmentation point G

Cheveux et pilosité

- ☐ Epilation laser
- ☐ Microgreffes capillaires
- ☐ Sudation
- ☐ PRP
- ☐ Chutes de cheveux

Injectons de fillers

- ☒ Mésolift
- ☒ Mésothérapie
- ☒ Acide hyaluronique rides

Le code générant cette partie de la vue est affiché ci-dessous.

```

<h4>Sélectionner les prestations possibles</h4>
<div class="row">
  @foreach ($categoryHC as $cat)
    <div class="col-12 col-sm-4 my-2 px-2">
      <h5>{{ $cat->category }}</h5>
      @foreach ($healthcares as $item)
        @if ($item->category==$cat->category)
          <div class="col-12">
            <input type="checkbox" name="healths[]" value="{{ $item->id }}" class="mr-2"
            @foreach ($user->healthcares as $cares)
              @if ($cares->id==$item->id)
                {{ 'checked' }}
              @endif
            @endforeach
            >{{ $item->name }}
          </div>
        @endif
      @endforeach
    </div>
  @endforeach
</div>

```

Une boucle Foreach, permet de parcourir l'ensemble des catégories de soins, passé dans l'objet \$categoryHC.

Ensuite pour chaque catégorie, une autre boucle foreach, parcourt l'ensemble de l'objet \$healthcares et :

- Compare (condition if) si la catégorie du soin correspond à la catégorie qui est en train d'être affiché ; auquel cas un input de type checkbox est créé avec le nom du soin
- Compare (condition foreach et if) si le soin a été déjà sélectionné par le médecin, auquel cas la checkbox est déclarée 'checked'

```

    @endif
    <div class="col-12 d-flex justify-content-center py-2">
      <button type="submit" class="btn btn-light">Mettre à jour mon profil</button>
    </div>
  </form>

```

La fin du code se termine par une balise <button> qui permet de soumettre le formulaire à la route 'admin.update'.

6.10.3 Méthode update

La méthode update doit mettre la table users, la table media query pour stocker l'image du médecin et la table de jonction healthcare_user.

Pour ceci, le code est ci -dessous.

Je commence par utiliser une méthode Eloquent detach() qui permet de supprimer les valeurs dans la table de jonction pour le user donné.

J'ai ensuite réalisé une méthode update() avec l'ensemble de la requête passé au contrôleur (l'ensemble des champs n'est pas affiché).

Pour gérer l'ajout éventuel d'une photo, je commence par vérifier si le champ 'profile_img' de la requête existe. Dans ce cas, je récupère le fichier (\$file) et à l'aide des méthodes spécifiques à LavarelMedia, j'ajoute ce fichier à la collection('doctors) en réalisant une copie du fichier original (->preservingOriginal, non destructif)


```

public function update(Request $request, User $user)
{
    // dd($request);
    $user->healthcares()->detach();

    $user->update([ ...
    ]);

    if (isset($request['profile_img'])) {
        $file = $request->file('profile_img');
        $user->addMedia($file)
            ->preservingOriginal()
            ->toMediaCollection('doctors');
    }

    $healths = $request->input('healths');
    $iduser = $user->id;
    $healths_user = [];
    if ($healths !== null) {
        foreach ($healths as $health => $value) {
            array_push($healths_user, [
                'healthcare_id' => $value,
                'user_id' => $iduser
            ]);
        }

        $user->healthcares()->attach($healths_user);
    }
    return redirect()->route('admin.crud_doctors')->with('success','Médecin mis à jour!');
}

```

Enfin je récupère l'array \$healths passé par la requête. J'instancie un array vide (\$healths_user) et j'affecte l'id du user à la variable \$iduser.

Si l'array \$healths n'est pas nul, alors pour chaque ligne (boucle foreach), j'ajoute le couple Id_User/Id_Healthcare à l'array \$healths_user.

Ce tableau associatif est ensuite attaché (méthode attach()) à l'objet \$user.

Au final, l'utilisateur est redirigé vers la route 'admin.crud_doctors' avec un message flash de 'success' qui sera affiché en arrivant sur la vue.

6.10.4 Méthode destroy

```

public function destroy(User $user)
{
    $destroyuser=User::where('id', $user->id)->first();
    $destroyuser->delete();
    return redirect()->route('admin.crud_doctors');
}

```

Cette méthode est appelée dans la modale du CRUD. Elle permet simplement d'effacer définitivement un médecin dont l'id a été passé avec la route.

6.10.5 Méthode search_doctor

Pour faciliter la recherche dans le CRUD « médecin », j'ai ajouté un moteur de recherche simple à la vue. Il est composé de 2 champs :

- Un champ texte
- Une liste déroulante de spécialité

Liste des médecins actifs

Ce formulaire appelle la méthode `search_doctor` qui est présenté ci-dessous.

Comme pour chaque formulaire, les valeurs saisies arrivent dans un objet `$request`.

Après avoir récupérer les données dans 2 variables (`$q` et `$speciality`), j'ai effectué 3 conditions (if else) imbriqué pour effectuer les 4 possibilités de recherche :

- Aucun champ n'a été complété : la méthode retrouve l'ensemble des medecins
- Seul le champs spécialité a été complété, la requête retourne les medecins ayant la spécialité visée
- Seul le champ texte a été complété, la requête recherche sur 3 colonnes de la table (lastname, firstname, city) si le champ est présent. Des '%' sont ajoutés autour de `$q` afin de pouvoir trouver tous les champs n'étant pas strictement égaux à la requête.
- Les 2 champs sont complétés : une requête cumulative est réalisée.

Les données sont retournées à la vue 'admin.cruddoctor' par page de 30 medecins. Afin que la requête persiste dans la navigation dans le page, une méthode Eloquent '`->withQueryString()`' est ajoutée en fin de requête.

```
public function search_doctor()
{
    $q=request()->input('q');
    $speciality=request()->input('speciality_id');

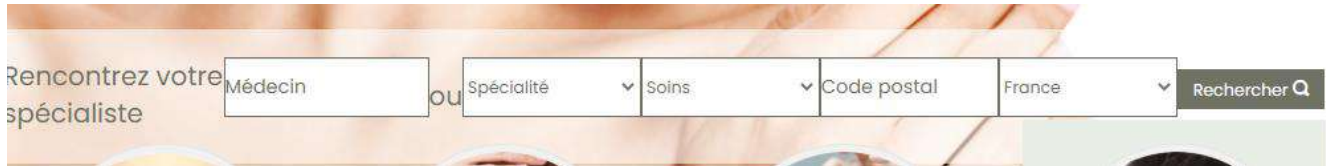
    if (($speciality != null)&&($q==null))
    {
        $doctors=User::where('role', 'doctor')
            ->where('speciality_id', $speciality)
            ->paginate(30)->withQueryString();
    }
    elseif (($speciality == null)&&($q!=null))
    {
        $doctors=User::where('role', 'doctor')
            ->where('lastname', 'like', "%$q%")
            ->orWhere('firstname', 'like', "%$q%")
            ->orWhere('city', 'like', "%$q%")
            ->paginate(30)->withQueryString();
    }
    elseif (($speciality != null)&&($q!=null))
    {
        $doctors=User::where('role', 'doctor')
            ->where('speciality_id', $speciality)
            ->where(function($query) use($q){
                $query->where('lastname', 'like', "%$q%")
                ->orWhere('firstname', 'like', "%$q%")
                ->orWhere('city', 'like', "%$q%");
            })
            ->paginate(30)->withQueryString();
    }
    else
    {
        $doctors=User::where('role', 'doctor')
            ->paginate(30)->withQueryString();
    }

    return view('admin.cruddoctors', [
        'doctors' => $doctors,
        'specialities' =>Speciality::all(),
    ]);
}
```

6.11 Développement du moteur de recherche

Dans la vue front, un moteur de recherche est proposé à l'utilisateur pour trouver un praticien :

- Nom
- Spécialité
- Code postal
- Pays



Rencontrez votre spécialiste

Médecin ou Spécialité Soins Code postal France Rechercher

Le code de la méthode search() est présenté ci-dessous.

```
/**
 * Search doctor in User database.
 *
 */
public function search()
{
    $name=request()->input('namesearch');
    $speciality=request()->input('speciality_id');
    $postcode=request()->input('lieu');
    $country=request()->input('country');

    // Définition des opérateurs en fonction des champs remplis par l'utilisateur
    $speciality!=null ? $op_spe='=' : $op_spe='!=';
    $name!=null ? $op_name='like' : $op_name='!=';
    $postcode!=null ? $op_pc='like' : $op_pc='!=';

    $country!=null ? $op_country='=' : $op_country='!=';

    // Réalisation de la requête dans la table user
    $doctors=User::where('role', 'doctor')
        ->where('imported', true) // N'affiche que les médecins importés //
        ->where(function($query) use($op_spe, $speciality){
            $query->where('speciality_id', $op_spe, $speciality)
                ->orWhere('speciality_2_id', $op_spe, $speciality);
        })
        ->where(function($query) use($op_name, $name){
            $query->where('lastname', $op_name, "%$name%")
                ->orWhere('centre', $op_name, "%$name%");
        })
        ->where(function($query) use($op_pc, $postcode){
            $query->where('postcode', $op_pc, "$postcode")
                ->orWhere('city', $op_pc, "%$postcode%");
        })
        ->where('country_code', $op_country, $country)
        ->orderBy('lastname','asc')
        ->paginate(10)->withQueryString();

    return view('user.search', [
        'doctors' => $doctors,
        'specialities' =>Speciality::all(),
        'soins'=>Healthcare::all(),
    ]);
}
```

Les valeurs de l'objet request sont affectées à leurs variables respectives.

Ensuite, par un jeu de condition ternaire, des opérateurs sont définis pour chaque variable à rechercher dans la base de données. Par exemple :

```
$speciality!=null ? $op_spe='=' : $op_spe='!=';
```

Si la variable \$speciality n'est pas nul alors l'opérateur de recherche sera 'égal à ' sinon il sera 'pas égal à '.

L'ensemble des variables et opérateurs sont ensuite utilisés dans la requête Eloquent.

Par exemple pour la spécialité :

```
->where(function($query) use($op_spe, $speciality){  
    $query->where('speciality_id', $op_spe, $speciality)  
    ->orWhere('speciality_2_id', $op_spe, $speciality);  
})
```

La spécialité pouvant être sur 2 colonnes, j'ai utilisé la propriété function(\$query) qui permet d'isoler l'opérateur OR entre les 2 colonnes (comme des parenthèses dans une requête SQL classique).

Codé de cette façon le moteur est DRY et autorise les recherches cumulatives sans que tous les champs soient pour autant remplis

7 PRESENTATION DU JEU D'ESSAI

Pour le jeu d'essai j'ai décidé de vous présenter le résultat d'une recherche sur la page d'accueil.

J'ai choisi de ne remplir que le champ spécialité à « médecin esthétique » et le code postal de façon partiel à 75 (donc à priori tous les arrondissements de Paris). Selon, la capture d'écran ci-dessous :

La requête Search retourne la vue suivante :



Numéro de téléphone
+ (33) 147 234 146

Zestetik®

LES EXPERTS EN ESTHÉTIQUE MÉDICALE

Connexion

Inscription Médecin

L'expert esthétique pour trouver
le soin ou l'intervention
qui vous correspond

53 résultats pour votre recherche

1 Dr. Cati Albou-ganem



Médecin esthétique

230 Rue du Faubourg
Saint-Honoré
75008 Paris
France

Appeler

Demande de
devis

Itinéraire

voir son profil

2 Dr. Jean Luc Allemandi



Médecin esthétique

6 Rue Brey
75017 Paris
France

Appeler

Demande de
devis

Itinéraire

voir son profil

J'ai bien trouvé 53 médecins esthétiques donc le code postal commence par 75. La page s'affiche d'ailleurs avec une navigation sur 6 pages (ce qui correspond au paginate(10) mis dans la méthode Search)



France

voir son profil



France

voir son profil



L'OFFICIEL



ANTI-AGE

PRIME

EuroMediCom

Teads

ELLE TAILLER

Expert c'est à vous !



Cette recherche dans le front équivaut à saisir cette requête SQL suivante dans phpMyAdmin :

Exécuter une ou des requêtes SQL sur la table « c1ubddzes55.users »:

```
1 SELECT * FROM `users` WHERE (`role`='doctor') AND (`speciality_id`=1 OR `speciality_2_id`=1) AND (`postcode` LIKE '75%')
```

Colonnes

id
active
role
imported

Le résultat de la requête donne bien 53 lignes au total.

Afficher la zone SQL

✓ Affichage des lignes 0 - 24 (total de 53, traitement en 0,0223 seconde(s).)

SELECT * FROM `users` WHERE (`role`='doctor') AND (`speciality_id`=1 OR `speciality_2_id`=1) AND (`postcode` LIKE '75%')

☐ Profilage [Éditer en ligne] [Éditer] [Expliquer]

1 > >>

☐ Tout afficher

Nombre de lignes : 25

Filtrer les lignes: Chercher dans cette table

Trier par clé : Aucun(e)

+ Options

	id	active	role	imported	email	password
<input type="checkbox"/> Éditer <input type="checkbox"/> Copier <input type="checkbox"/> Supprimer	14	0	doctor	1	centremedicalniel@gmail.com	097cafc46ee86d7bb4c4241404de857c6ddc1627

Le moteur de recherche fonctionne de façon conforme.

8 DESCRIPTION DE LA VEILLE,

Tout au long de la formation, j'ai entrepris une veille à partir de plusieurs sites.

Ainsi sur le réseau social LinkedIn, je me suis inscrit à de nombreux groupes thématiques liés au développement web :

- PHP Developpers (152 000 membres)
- Javascript (189 000 membres)
- Wordpress Developpers (25 500 membres)
- FrenchWeb, le network des acteurs de la tech (65 500 membres)
- CSS3/HTML5 The future of front end (59 500 membres)
- ...

Afin de me tenir au courant de l'actualité de ses technologies, y compris les éventuels problèmes de sécurité.

Par ailleurs, j'ai développé mon réseau LinkedIn en invitant des acteurs locaux du développement web au fur et à mesure de mes rencontres. J'ai pu ainsi avoir un accès régulier aux post de la déléguée régionale de ANSSI (Agence Nationale de la Sécurité des Systèmes Informatiques)

Je reçois par ailleurs régulièrement les newsletters de :

Codeur.com : Blog consacré au développement web qui peut être amené à signaler des problèmes de sécurité

Silicon.fr : site dédié à la sécurité numérique.

Concernant le projet Zestetik, l'utilisation de Laravel m'a permis de me prémunir de nombreux problèmes de sécurité, ce Framework embarquant de base des fonctionnalités facilitant la sécurisation du site. Ainsi,

- L'existence de package dédié à l'authentification sécurisée des utilisateurs permet de réduire le risque d'intrusion.
- Eloquent utilise largement PDO ce qui immunise contre les injections SQL
- La mise en place des Tokens CSRF sur chaque formulaire permet de se protéger des attaques XSS
- Enfin l'existence des middlewares permet de sécuriser les routes en empêchant l'accès à certaines pages.

9 DESCRIPTION D'UNE SITUATION DE TRAVAIL AYANT NECESSITE UNE RECHERCHE

Pendant ma période de stage, j'ai eu à utiliser des packages complémentaires dans mon projet Zestetik, Ne connaissant pas ces packages j'ai donc eu recours à la documentation disponible sur Internet. Un package intéressant fut Laravel Excel qui permet de gérer des imports et des exports de fichiers Excel. Je l'ai par ailleurs utilisé sur 2 autres développements.

Dans le présent projet, la consigne était de mettre à jour la table User à partir d'un fichier xls fourni par le client. J'ai donc eu à consulter cette documentation :

<https://docs.laravel-excel.com/3.1/imports/basics.html>

Cette page explique comment utiliser les fonctionnalités basiques d'import.

Le contenu en anglais est le suivant :

« [Importing basics](#)

If you have followed the 5 minute quick start, you'll already have a `UsersImport` class.

```
1  <?php
2
3  namespace App\Imports;
4
5  use App\User;
6  use Illuminate\Support\Facades\Hash;
7  use Maatwebsite\Excel\Concerns\ToModel;
8
9  class UsersImport implements ToModel
10 {
11     /**
12      * @param array $row
13      *
14      * @return User|null
15      */
16     public function model(array $row)
17     {
18         return new User([
19             'name' => $row[0],
20             'email' => $row[1],
21             'password' => Hash::make($row[2]),
22         ]);
23     }
24 }
```

Importing from default disk

Passing the `UsersImport` object to the `Excel::import()` method, will tell the package how to import the file that is passed as second parameter. The file is expected to be located in your default filesystem disk (see `config/filesystems.php`).

```
1  Excel::import(new UsersImport, 'users.xlsx');
```

Importing from another disk

You can specify another disk with the third parameter like your Amazon s3 disk. (see `config/filesystems.php`)

```
1  Excel::import(new UsersImport, 'users.xlsx', 's3');
```

Importing uploaded files

If you let your user upload the document, you can also just pass the uploaded file directly.

1

```
Excel::import(new UsersImport, request()->file('your_file'));
```

php

Importing full path

If you want to specify the path where your file is, without having to move it to a disk, you can directly pass that file path to the import method.

1

```
Excel::import(new UsersImport, storage_path('users.xlsx'));
```

php

Importing to array or collection

If you want to bypass the `ToArray` or `ToCollection` concerns and want to have an array of imported data in your controller (beware of performance!), you can use the `::toArray()` or `::toCollection()` method.

1

```
$array = Excel::toArray(new UsersImport, 'users.xlsx');
```

2

3

```
$collection = Excel::toCollection(new UsersImport, 'users.xlsx');
```

php

Specifying a reader type

If the reader type is not detectable by the file extension, you can specify a reader type by passing it as fourth parameter. »

1

```
Excel::import(new UsersImport, 'users.xlsx', 's3', \Maatwebsite\Excel\Ex
```

php

La traduction qu'on peut en faire est la suivante :

Les basiques de l'import

Si vous avez suivi la mise en route rapide en 5 minutes, vous avez déjà une classe `UserImport`.

Importer depuis le disque par défaut :

Le fait de passer l'objet `UserImport` à la méthode `Excel::import` va dire au paquet comment importer le fichier excel qui est passé en second argument. Le fichier est supposé être localisé dans votre disque système par défaut (voir dans le fichier `config/filesystems.php`)

Importer depuis un autre disque :

Vous pouvez spécifier un autre disque en passant un troisième paramètre comme par exemple un disque Amazon S3 (Service Simple de Stockage) (voir dans le fichier `config/filesystems.php`)

Importer des fichiers téléchargés

Si vous voulez laisser l'utilisateur télécharger directement le fichier, vous pouvez juste aussi passer le fichier téléchargé directement.

Importer un chemin complet :

Si vous voulez spécifier le chemin où se trouve votre fichier sans avoir à le déplacer, vous pouvez directement passer le chemin du fichier dans la méthode d'import.

Importer dans un tableau ou une collection

Si vous voulez contourner les soucis des méthodes toArray ou toCollection et que vous voulez un tableau des données importées directement dans votre contrôleur (attention aux performances), vous pouvez utiliser directement ces méthodes.

Spécifier le type de lecteur :

Si le type de lecteur n'est pas détectable par l'extension du fichier, vous pouvez spécifier un type de lecteur en le passant en 4^{ème} paramètre.

10 CONCLUSION

Ce projet et l'ensemble des missions qui m'ont été confiées par la société DM Web m'a permis de confronter aux réalités du métier. J'ai pu y approfondir mes connaissances dans l'utilisation du Framework Laravel mais aussi m'exercer sur un de mes points faibles l'utilisation du CSS pour améliorer l'expérience utilisateur.

Cette période de stage et l'ensemble de la formation ont confirmé mon projet de reconversion professionnelle.