

***TITRE PROFESSIONNEL DWWM***

DEVELOPPEUR WEB ET WEB MOBILE



***VETOTOIL***

regnier sylvain

***Sommaire***

1. Compétence du référentiel couverte
2. Présentation du Projet
3. Spécifications Techniques
4. Cahier des charges
5. Compétences transversales et méthode agile
6. Architecture du projet
7. Réalisation de la partie Front-End

6.1- Wireframes et Maquette

6.2- UML

6.3- Développement des fonctionnalités

6.4- Rendu final

1. Réalisation de la partie Back-End

7.1-Base de données

7.1-Modélisation(Merise)

7.1-Création de la BDD et des requêtes

7.4-Développement

1. Déploiement du site
2. Conclusion

**01| Compétence du référentiel couverte**

*1.1 Développer la partie Front-End d’une application web ou web mobile en intégrant les recommandations de sécurité.*

• LES MAQUETTES

• REALISER UNE INTERFACE UTILISATEUR WEB OU MOBILE STATIQUE ET ADAPTABLE

• DÉVELOPPER UNE INTERFACE UTILISATEUR WEB OU MOBILE DYNAMIQUE

Pour la réalisation du site web « VETOTOIL », j’ai créé des maquettes afin de développer le site. Ensuite, elles ont été rendue dynamique et fonctionnelle en suivant les bonnes pratiques de développement.

*1.2 Développer la partie Back-End d'une application Web ou Web-mobile en intégrant les recommandations de sécurité.*

-En développement la partie back-end du site, j’ai intégré les mesures nécessaires de sécurité pour la protection des données ainsi que des attaques malveillantes comme par exemple en vérifiant chaque donnée insérée par l’utilisateur (injection SQL, injection de script …), mais aussi en me servant de UuidV7 unique, pour rendre chaque ID aléatoire au lieu d’avoir des chiffres qui se suivent, captchaV3 de Google pour éviter les inscriptions des robots.

-J’ai aussi fait le contrôle sur l’accès au donnée et aux fonctionnalité du site via le système des rôles.

-La partie back-end présenté est faite sous Symfony mais une version en MVC a aussi été faite.

-Une partie envoi de mail à l’administrateur est aussi proposé, cela a été fait avec PHPMailer

*1.3 Compétence Transversale*

• MÉTHODE AGILE (AGILE SOLO) : Dans le cadre de la réalisation de VETOTOIL j’ai appliqué à moi-même la méthode AGILE, je m’appliquais des taches journalières, des étapes et la durée de chacune des étapes pour la conception et la réalisation.

**Présentation du projet**

VETOTOIL est une application qui a pour but de pouvoir prendre des rendez-vous chez le professionnel de votre choix, vétérinaire, toiletteur (liste non exhaustive) sans avoir à rechercher le professionnel sur internet puis de le contacter ou par moment il n’y a pas de réponse à l’appel et nous devons réitérer l’appel, puis ensuite une fois les avoir au téléphone de pouvoir trouver une date et une heure qui conviennent à tout le monde.

VETOTOIL a pour but de facilité cela car après l’inscription sur le site, l’utilisateur pourra rechercher les professionnels qui sont inscrit dans son secteur, et en choisissant son domaine d’activité et ainsi de voir les premières disponibilités pour une petite urgence mais aussi la possibilité de voir par professionnel ses disponibilités au dates souhaité, cela permet d’être devant son téléphone ou son ordinateur et de voir directement les rdv disponibles. Le patient pourra aussi voir ses rdv en cours et ses rdv passés ainsi que le suivi de traitement s’il y en a pour son animal. Il pourra aussi ajouter des animaux a sa fiche, pour le moment chien ou chat avec une liste de race pour chacun d’entre eux prédéfini.

Pour des raisons de sécurité a société quand a elle pourra s’inscrire, mais il devra forcément avoir un Siret valide et sera soumis à validation par l’administrateur du site. Ensuite il pourra rattacher des employés déjà crée ou en crée des nouveaux, tous cela sera de la responsabilité de la société.

*Le but principal de VETOTOIL est de faire un Doctolib mais pour nos animaux*

**Spécifications Technique**

*Général :*

- Editeur de code : Visual Studio Code

-extension utilisées :

\*Composer for Visual Studio Code : permet de de faire des commandes rapides pour composer

\* IntelliPHP for Visual Studio Code : permet de travailler plus rapidement car l’IA essaie de prédire la suite du code

\*Live Server for Visual Studio Code : permet d’ouvrir une page HTML ou JavaScript en local

\*Prettier Formatter for Visual Studio Code : formatage du code

\*Twig Formatter for Visual Studio Code : formatage du code pour twig

- Outil de versionning : GitHub

- Maquettages : Figma

- Modèle conceptuel et logique de donnée : Looping

- Serveur php : Xampp

- Modele physique de données : Interface visuelle de PhpMyAdmin

**Projet VETOTOIL(version 2)**

-Language et balisage : HTML,Bootstrap et CSS

-Language de programmation : PHP, Javascript et twig

-Fait sous Symfony

Librairie :

-PHPMailer pour l’envoie de mail (‘https://github.com/PHPMailer/PHPMailer’)

-VichUploaderBundle pour insérer des images(ou fichier) (‘https://symfony.com/doc/current/controller/upload\_file.html’)

-Karser pour faire le RecaptchaV3 de google pour sécuriser les envoie de mail et d’inscription (‘https://github.com/karser/KarserRecaptcha3Bundle’)

-UuidV7 (fonction unique) pour générer des id aléatoire et « unique » pour certaine base de donnée sensible.(‘ https://symfony.com/doc/current/components/uid.html#working-with-uuids’)

- password-hasher pour crypter un mot de passe pour qu’il ne soit pas récupéré(‘https://symfony.com/doc/current/security/passwords.html’)

**Cahier des charges**

*Objectif :*

-Etablir un programme utile pour simplifier la prise de rdv chez les professionnels pour les animaux.

Pour les Professionnels :

-Gagner du temps, car il y aura moins d'appels téléphoniques pour la prise/modification/annulation de rendez-vous.

- Possibilité de ne plus avoir de créneaux libres dans leurs plannings.

-Pour les Particuliers :

-Ne plus attendre d'avoir des dates fournies par le professionnel et de vérifier ses propres disponibilités, car les disponibilités seront disponibles immédiatement sur le site internet.

-Avoir un suivi des rendez-vous et chez quel professionnel, ainsi que les traitements dans l'historique.

-Il doit être responsive et s’adapter aux différente taille d’écran (smartphone, tablette, pc)

-Le site aura aussi bien du Back-End et du Front-end

*Fonctionnalités principales :*

*Pour tous :*

Partie inscription users ou tout le monde s’inscrira avec les mêmes information demandé (nom, prénom, email(ne peux avoir deux emails identique), password) ensuite à la connexion ils seront redirigés vers une autre page des complément d’information.

*Pour les Sociétés :*

-création d’une fiche société (information demandé Siret, nom de la société, profession de la société, adresse de la société, complément adresse de la société, code postal de la société, ville de la société, téléphone de la société, téléphone du dirigeant, image société, date de création, date de résiliation, date de validation)

-Doit pouvoir crée un employé (fiche user normale)

-Doit pouvoir rechercher un employé par son email s’il est déjà inscrit

-Doit pouvoir ajouter un employé a son équipe et définir les jours travaillé ainsi que la pause repas et les vacances

-Sur le compte de la société doit pouvoir voir les informations de ses employés ainsi que leur planning, doit pouvoir aussi retirer un employé.

*Pour les employés :*

-Doit pouvoir compléter sa fiche (adresse employer, complément adresse employer, code postal employer, ville employer, téléphone employer, profession employer, image employer, date de création employer)

-Ne peux être rechercher que si la partie précédente est faite

-Doit pouvoir voir ses rdv du jour et de la semaine en fonction de chaque heure et de voir la fiche du patient et animal en fonction de ses rdv (consultation uniquement), doit pouvoir prendre un rdv en « live », possibilité d’annuler un rdv, possibilité, valider ou annuler le rdv.

-Doit pouvoir inscrire pour les patients les médicaments et la posologie

*Pour les particuliers :*

-Doit pouvoir compléter sa fiche patient (adresse patient, complément adresse patient, code postal patient, ville patient, téléphone patient, image, date de création, date fin patient)

-Doit pouvoir inscrire un ou plusieurs animal (prénom, date de naissance, type, race)

-Rechercher tous les professionnels suivant leurs disponibilités dans une zone géographique de 10km

-Prendre un ou plusieurs rdv

-Pouvoir consulter ses rdv futur comme passé

-Si possible envoyé un mail 48h avant le rdv

*CONCEPTION VISUEL :*

*Pour le Header et le Footer*

Un dégrader de white au purple (#800080) 

*Dans le body*

Les mots importants seront en purple.

*Couleur Générale d’écriture :*

Black

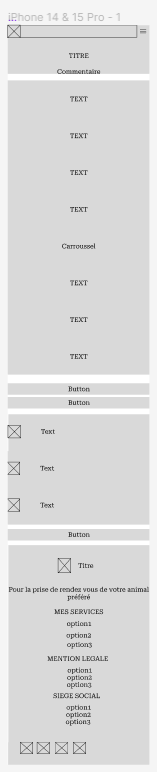
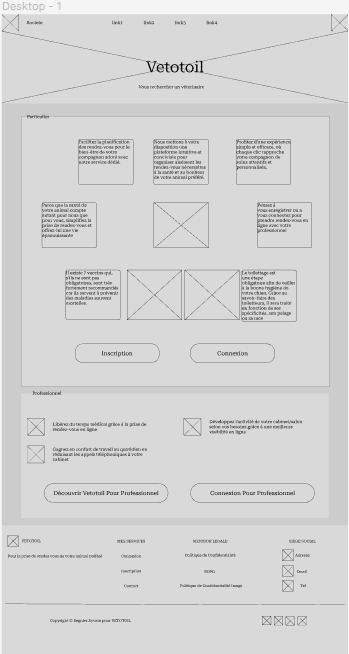
*Icones :*

Purple sauf dans le header et le footer

*Footer icone :*

Facebook , Twitter(X), instagram,linkedin

**Création des WireFrames :**



**Réalisation des maquettes :**

*Version PC*

**

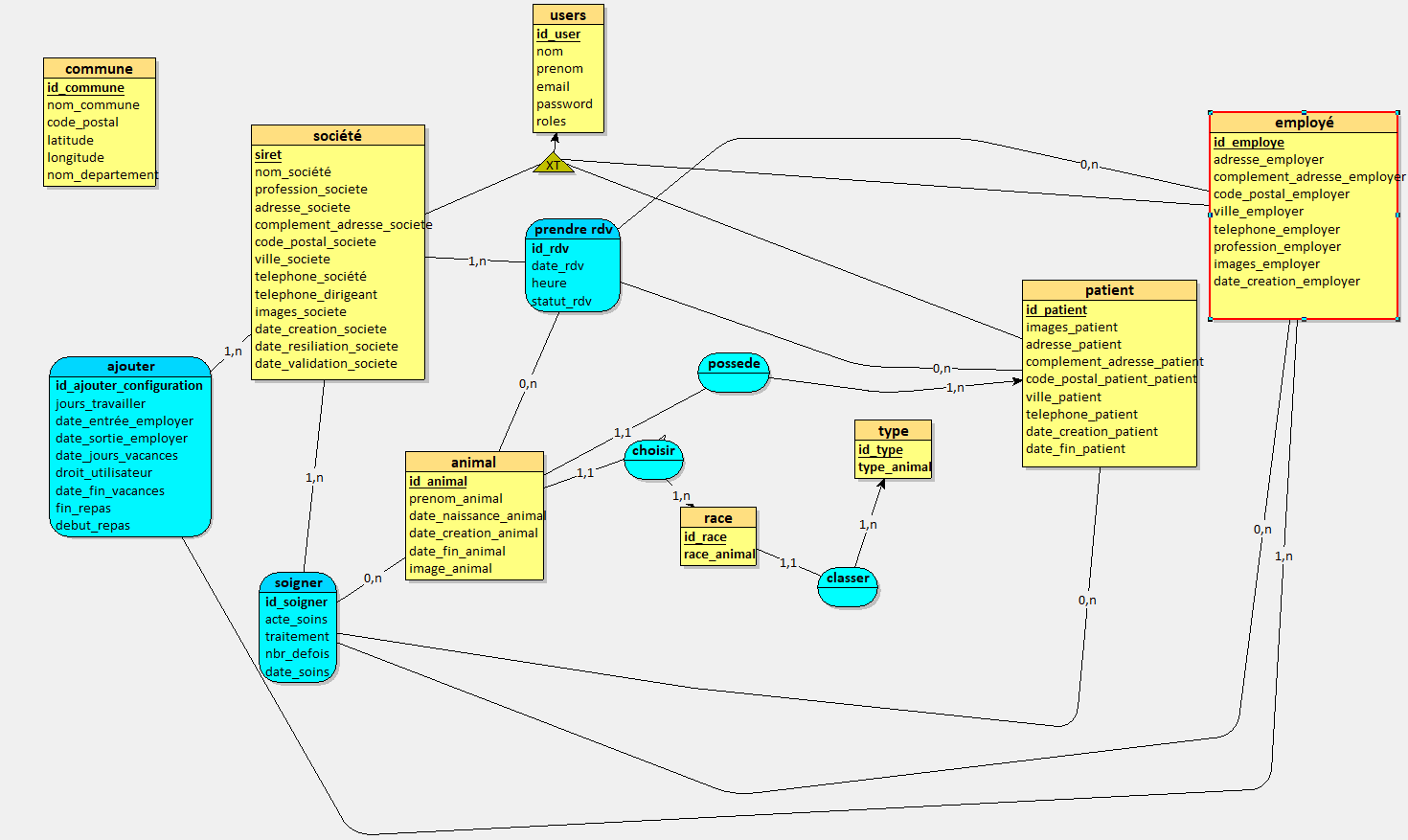
***Faire un UML***

*Version mobile :*

**

*MCD :*

J’ai donc élaboré un mcd pour pouvoir construire ensuite le mld qui sera ma référence pour ma bdd et mes tables. Au départ je les ai fait a la main sur une feuille puis j’ai utilisé looping pour un meilleur rendu.



Pour élaborer le MCD je suis partie d’une entité user qui a un héritage sur l’entité société, patient et employer.

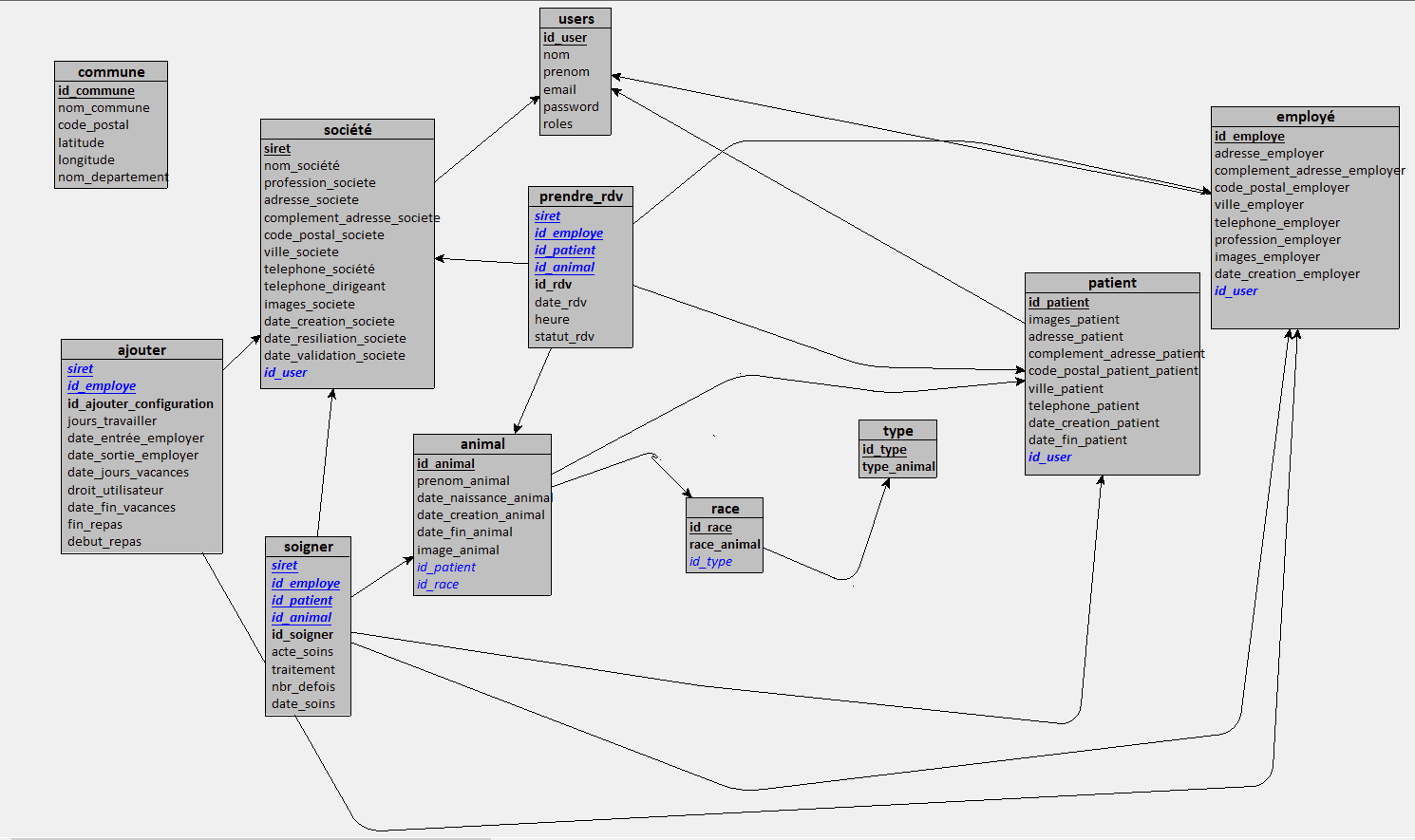
De la partie société une association a été créé pour ajouter des employés. De l’entité société à ajouter j’ai mis une cardinalité de 1, N car une société peut avoir un ou plusieurs employer et d’employé à ajouter la cardinalité est de 1, N car un employé peut être affecté à une ou plusieurs sociétés. Sur le MLD cette association deviendra une table car nous avons d’un côté N et de l’autre côté N.

Entre l’entité animal et race l’association choisir ne deviendra pas une table au MLD car l’association est d’animal à choisir avec la cardinalité 1,1 ce qui correspond que l’on peut choisir 1 seule race pour un animal. De race à choisir la relation est du 1, N car une race peut être choisit pour 1 ou plusieurs animaux et donc une clef étrangère dans l’entité animal suffira car c’est une relation directe.

L’association soigner et prendre rdv qui a une cardinalité N N sera modifié pour les mêmes raison dans le MLD

L’association possède et classer sera du même ordre que l’association choisir car nous avons du 1,1 en cardinalité

*MLD :*

**

Pour le MCD j’ai donc ajouter les clefs étrangères et fait les tables en fonction du mcd ce qui m’a permis de pouvoir commencer le développement du projet

**Introduction et structure :**

Pour la partir Front-End, j’ai décidé d’utilisé du HTML,le framework Bootstrap ,Javascript ,twig et un peu de CSS le tout fait avec le Framework Symfony et son architecture.

J’ai aussi utilisé des librairies qui sont à la fois Front-End et Back-End comme :

-Le recaptchaV3 de Google fournit par Karser (<https://github.com/karser/KarserRecaptcha3Bundle>)

-VichUploaderBundle pour insérer des photos et les affichés (<https://symfony.com/doc/current/controller/upload_file.html>)

J’ai aussi utilisé l’API ADRESSE de l’Etat pour permettre de l’utilisateur de trouver celle-ci en fonction de ce qu’il va taper et ainsi renseigner la ville et le code postal automatiquement

Pour pouvoir avoir un rendu en mode développent il faut au préalable lancer le serveur de Symfony avec la commande :

symfony server:start

Puis on accède au site a l url suivante http :://localhost :8000

Voici la structure de la partie Front-End

|  |  |
| --- | --- |
|  | Dans les assets nous retrouvons un dossier js ou tous les scripts sont inséré dedans.  Nous trouvons aussi le dossier styles ou il y a le fichier css |

|  |  |
| --- | --- |
|  | Dans cette partie dans le dossier public, j’ai créé un dossier images dans lequel il y a 4 autres dossiers.  Le dossier img est pour toutes les images du site  Les 3 autres dossiers sont lorsque les utilisateurs s’inscrivent ils peuvent ajouter une photo à leurs profils via vichupload et sont trié par rapport à leurs rôles.  Ensuite l’on peut voir le dossier templates dans lequel il y a les vues sous l’extension .html.twig qui indique que l’on peut mettre du code html dedans et le twig permet de récupérer des variables pour rendre le site plus dynamique et non statiques  Le fichier base.html.twig est le fichier de référence pour les autres pages c’est dedans que nous allons faire appel au script Bootstrap et importmap App qui fait référence au fichier app.js dans lequel le chemin du style/app.css est déclaré, cela est fait automatiquement par Symfony |

Voici le fichier base.html.twig

|  |
| --- |
|  |

Dans ce fichier nous voyons donc les liens pour accéder à la bibliothèque Bootstrap ainsi qu’à Bootstrap icon

De plus Symfony génère automatiquement le block JavaScript dans lequel il y a l importation de (‘app’) ou l’on peut retrouver le chemin du fichier css pour qu’il soit actif sur toutes les pages.

Ensuite toutes les pages seront traitées dans la balise <body> importer le header et le footer permet de ne plus avoir besoin de l’importer et sera visible sur toutes les pages du site.

En ce qui concerne le {% block body %} {% endblock %} cela sera à mettre dans toutes les pages pour informer le navigateur ou il doit l affiché sur la page.

Interface web adaptable

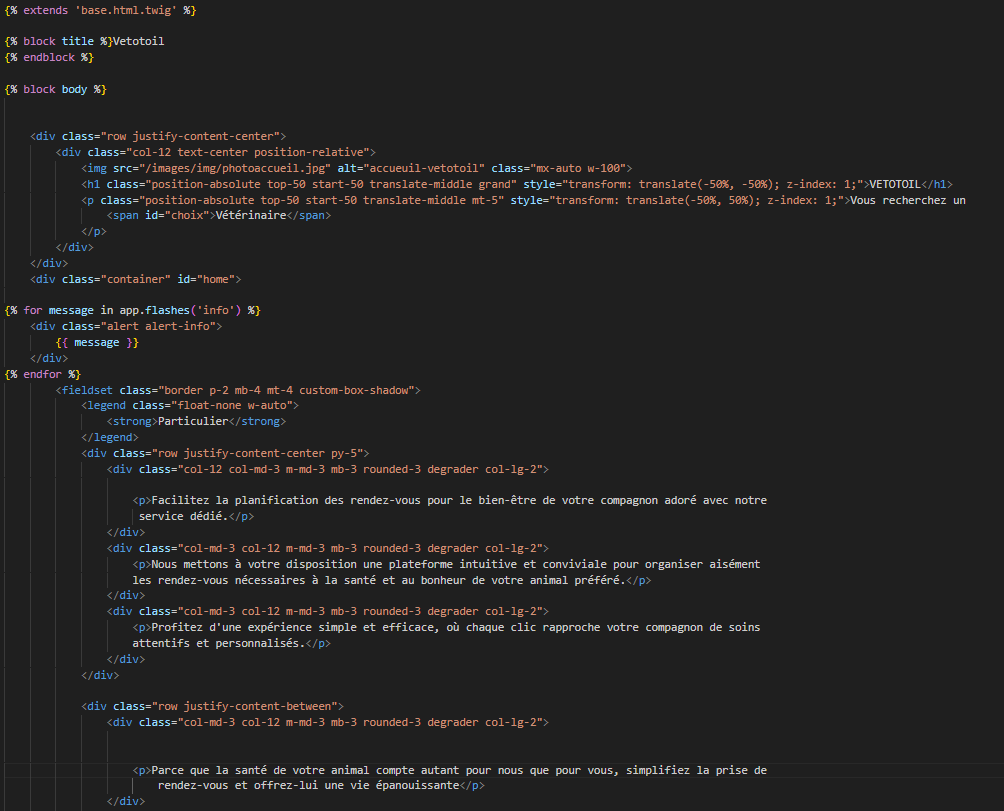
Nous avons vu le fichier principal préalablement et pour faire de notre site un sire responsive nous avons besoin de cette ligne.

<meta name="viewport" content="width=device-width, initial-scale=1">

Qui permet de dire au navigateur qu’il doit s’adapter à la largeur de l’appareil le viewport définissant les propriétés d’affichages.

Grace à la bibliothèque Bootstrap , le responsive est plus facile à gérer et nous avons besoin de moins coder dans le fichier css .

Voici une partie du [home.html.twig](http://home.html.twig)



Dans un premier temps on étend le fichier « base.html.twig » pour permettre d’avoir l’affichage prédéfinit ,puis ensuite nous mettons le code dans la balise {% block body %} et à la fin de la page il faut refermer cette balise avec {% endblock %}

Dans cette partie de code et avec bootstrap tous se fait au niveau des class.

Les row définissent une ligne et les colonnes sont une division de cette ligne, qui ne peux excéder 12 colonnes et chaque colonne peux encore avoir 12 colonnes.

Cette ligne veut dire qu’il y a une ligne et qu’elle est centré horizontalement dans son conteneur parent :

<div class="row justify-content-center">

Ici j’informe le navigateur que je veux que la colonne est une taille de 12 qui correspond à une colonne de 100% de largeur par rapport au paramètre précédent (s’il y a une colonne de 1 alors il prendra 100% de la colonne et non de la page).

Le text-center position-relative permet au contenu de cette colonne d’être centre horizontalement et il est positionné de façon relative par rapport à sa position normale dans le flux du document, je peux donc ajuster son positionnement à ma guise.

<div class="col-12 text-center position-relative">

<img src="/images/img/photoaccueil.jpg" alt="accueil-vetotoil" class="mx-auto w-100">

Ici j’insère l’image avec la balise img, le paramètre « alt » a plusieurs utilités qui sont, en cas d’erreur de chargement de l’image c’est l’annotation qui est indiqué qui sera vu à l’écran, il permet aussi aux personnes ayant des dispositifs de vue comme des lecteur d’écran de leur signifié ce que c’est et le dernier point est qu’il sert aussi pour les moteurs de recherche pour indexer les images (SEO).

Le src permet d’indiqué ou est situé le chemin de l’image et la class bootstrap mx auto défini qu’il faut centrer horizontalement l’image et le w-100 d’utiliser 100% de la largeur du conteneur.

La balise <h1> est la balise utilisé généralement pour les titres c’est l’une des balises les plus importantes pour le SEO.

*Le responsive sous bootstrap*

Bootstrap permet de définir dans ses classes les tailles d’ecran comme ceci

<div class="col-12 col-md-3 m-md-3 mb-3 rounded-3 degrader col-lg-2">

Ici on définit que la taille initiale est de de 12 colonnes donc 100% de l’écran, mais dès que l’on passe sur des écrans de taille moyenne la taille est réduite à 3 colonnes donc ¼ de celui-ci le m-md-3 permet de faire un margin de 3 sur tous les coté et le mb-3 rajoute une marge en bas ensuite le col-lg-2 indique que pour les écrans larges cela occupera 2 colonnes sur 12.

Comme on peut le voir bootstrap permet de gérer beaucoup de chose mais nous devons quand même passer par du css lorsque ce que l’on demande sort du type bootstrap par exemple pour faire un dégradé avec mes propres couleurs j’ai dû faire du css pour le header et le footer

.navbar,.footer {

    background: linear-gradient(white, purple);

}

Tous comme pour faire un hover sur un bouton sur les taille d’ecran supérieur a 1200px j ai du le faire en css en utilisant les média querie

<button type="button" class="btn btn-custom rounded-pill">Connexion</button>

@media only screen and (min-width: 1200px) {

    .btn-custom:hover {

        background-color:pink !important;

        color: purple !important;

    }

    .grand{

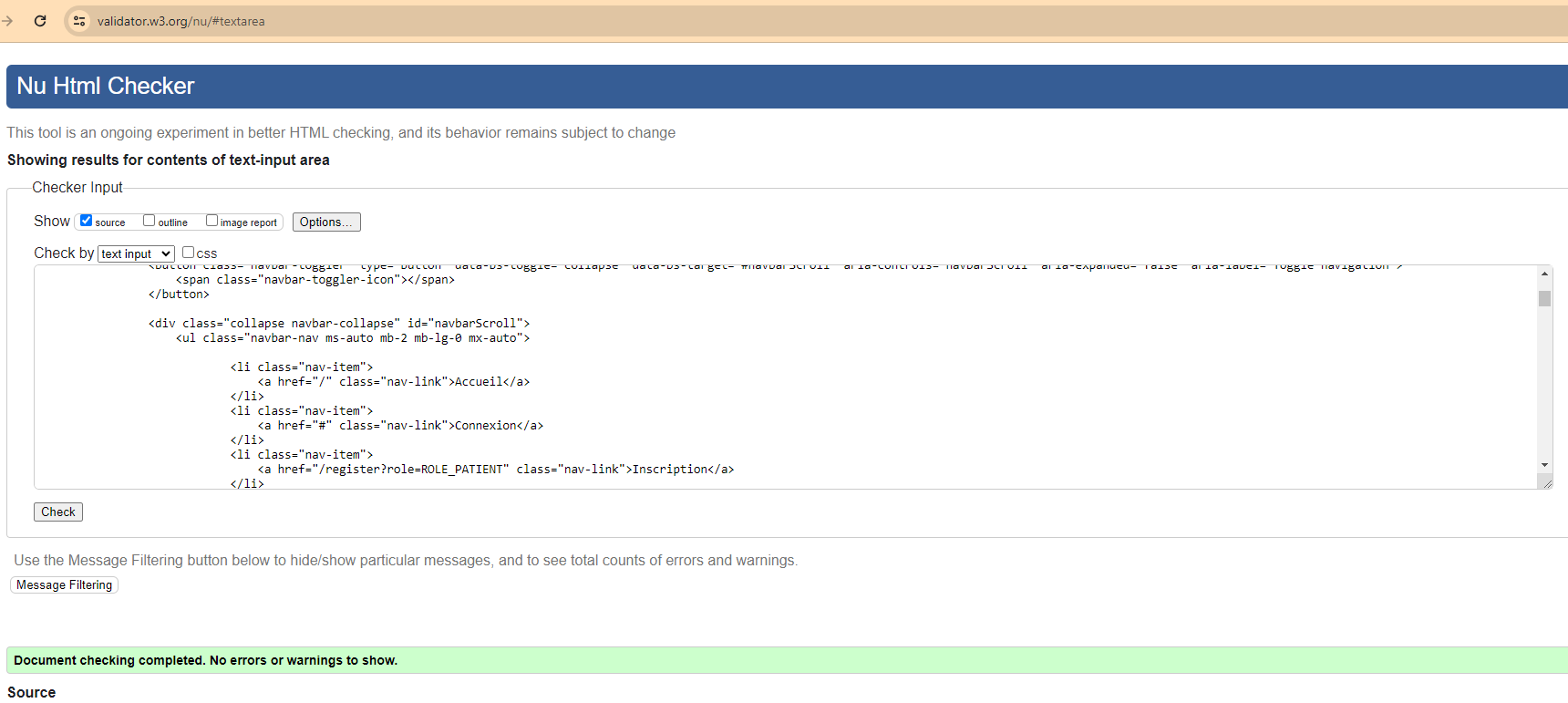
        font-size: 500%;

    }

}

Donc pour établir ce site j’ai donc commencé par le mobile first puis adapter mes class bootstrap en fonction des tailles d’ecran.

J’ai passé mon code de la page d’accueil sur le validateur W3 Validator, étant donnée qu’il y a du twig dedans j’ai dû copier la balise « Body » dans la console pour le tester le document n’a pas d’erreur ou d’alerte ce qui est pour le SEO l’un des critères



*Partie dynamique :*

Ayant la partie statique j’ai donc ajouter du JavaScript pour le rendre plus dynamique sur certaine fonctionnalité, dans un premier temps sur la saisie du mot de passe j’impose un nombre minimum de chiffre ET lettre, une majuscule et un caractère spécial dès que l’utilisateur commence à taper une « ul » s’affiche avec la liste qui se valide au fur et à mesure, si un champ est manquant la validation ne se fait pas

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | Dans un premier temps je mets un écouteur pour confirmer le chargement du DOM  Ensuite je recherche l’id du submit  Met un écouteur dessus avec la fonction « click »  Vérifie, même si symfony le gère la concordance entre les 2 mots de passe  Vérifie la longueur du mot de passe avec .length  Vérifie s’il y a une majuscule  S’il y a un chiffre minimum  Si un caractère spécial est bien présent  Et pour finir l’alert dialog si un champs n’a pas été respecté en utilisant la variable déclaré errors en tableau et qui est push (insérer) au fur et à mesure des erreur détecté |

|  |  |
| --- | --- |
|  | Dans cette suite de code je refais les vérifications en direct pour afficher ou masquer les champs qui ne respecte pas la chartre des mots de passe en rendant visible l’image ou en la rendant invisible mais aussi en rendant invisible l’ « ul » si le mot de passe est vide |

Pour la partie société J’ai fait appel à l’API SIRENE proposé par le gouvernement pour permettre une facilité de saisie en fonction du numéro de siret de la société le tout fais en javascript avec la method fetch

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | Définit l’url de l’api  Clef d’accès de test de l’api(version de test)  J’intercepte les lettres qui sont appuyé pour autoriser uniquement les chiffres et certaine touche  Je renseigne le lien de l’url enregistré + la valeur du champs input  Method fetch ou je passe les paramètres ou je passe les paramètres obligatoires dans le header demandé par l api  Je récupère en json le résultat  Je récupère les informations nécessaires  J’insère les information récupéré dans l input |

**Back-End du Projet**

Pour cette partie j’ai utilisé la structure de Symfony , c’est-à-dire un Controller,Entity,Form,Repository a cela j’ai ajouté un dossier function pour mettre des fonctions et pour insérer des données personnel dans la base de de donnée(bdd) j’ai utilisé la fonction de symfony, dedans j’ai mis des données statique pour les races d’animaux , les types ainsi que tous les renseignements pour les communes de France(info récupéré sur le site du gouvernement) pour faire un système de zonage.

composer require --dev orm-fixtures

Voici le squelette de l’architecture de symfony pour la partie Back-End

|  |  |
| --- | --- |
|  | Dans le dossier src  Nous avons la partie controlleur qui réceptionne les requêtes des utilisateurs, traite la logique, permet de manipuler les données et de sélectionner et préparer la vue, il sert de lien entre l’utilisateur, les données de l’application et l’interface utilisateur  DataFixture pour intégrer les données fixe ou fake dans les tables.  La partie entity permet de manipuler les données dans la base de donnée avec les getter et setter comme des objets. Une entité correspond aux colonnes d’une table, dedans nous avons les annotations lier au colonnes de la table le type les relations  Le Form permet de crée, traiter et valider des formulaires plus rapidement, ils peuvent être réutilisable sur différente vues  Le dossier function que j’ai créé pour mettre certaine fonction de traitement réutilisable  Le Repository sert de relation entre l’application et la base de donnée c’est dedans que nous avons les requêtes prédéfini par Symfony mais l’on peut aussi ajouter des requêtes complexe que Symfony ne gère pas |

Il y a aussi les dossiers indispensables à Symfony qui sont créé automatiquement comme le dossier vendor.

Sous symfony pour crée la base de donnée nous avons besoin de phpMyAdmin, j’ai utilisé celui de XAMPP.

Pour configurer crée la base de donnée il faut renseigner dans le fichier. env au niveau de la partie

###> doctrine/doctrine-bundle ###

Et j’ai rajouté la ligne pour dire a Symfony comment se connecté à la base de donnée

 DATABASE\_URL="mysql://root:@127.0.0.1:3306/vetotoil\_symfony?charset=utf8"

Mysql est le type de donnée utilisé, root et le nom d’utilisateur pour me connecter à la base de donnée, root est par défaut mais il peut être différent si lors de la configuration a la base de donnée nous avons modifier le paramètre utilisateur. Ensuite nous avons l’adresse ip pour mon cas le serveur étant en local 127.0.0.1 est l’adresse local, ensuite nous avons le port 3306 qui est le port par défaut puis le nom de la base de donnée et le jeu de caractère qui permet une compatibilité entre les différentes langues et caractère.

Une fois cela fait nous pouvons lancé la commande pour créer la bdd dans phpmyadmin avec cette ligne de commande

php bin/console doctrine:database:create

ceci crée la base de donnée mais sans table à l’intérieur

*Sous le MVC*

la procédure est différente dans un premier temps il faut crée la table directement dans phpmyadmin puis créer manuellement les tables et les colonnes

Une fois cette étape faite nous devons faire un constructeur dans le model du MVC comme ceci



La fonction new PDO() permet de se connecter a la base de donnée en lui informant que nous sommes en local avec localhost ensuite le dbname pour se connecté a la base de donnée vetotoil en paramètre le nom d’utilisateur root et en dernier champs le mot de passe à vide (car j’en ai pas mis)

Voici les bases de données suivant les types de méthodes utilisé

|  |  |
| --- | --- |
| Symfony | MVC |
|  |  |

***Création des entity et des controllers sous symfony :***

Sous Symfony pour crée les tables nous passons par Doctrine qui permet de crée des tables ses relations, ses attributs mais aussi de manipuler les données car il facilite le CRUD et de faire des requêtes DQL qui permette de faire des requêtes en utilisant les concepts orienté objet

Pour crée une table il faut dans un premier temps crée une entity avec cette commande

php bin/console make:entity

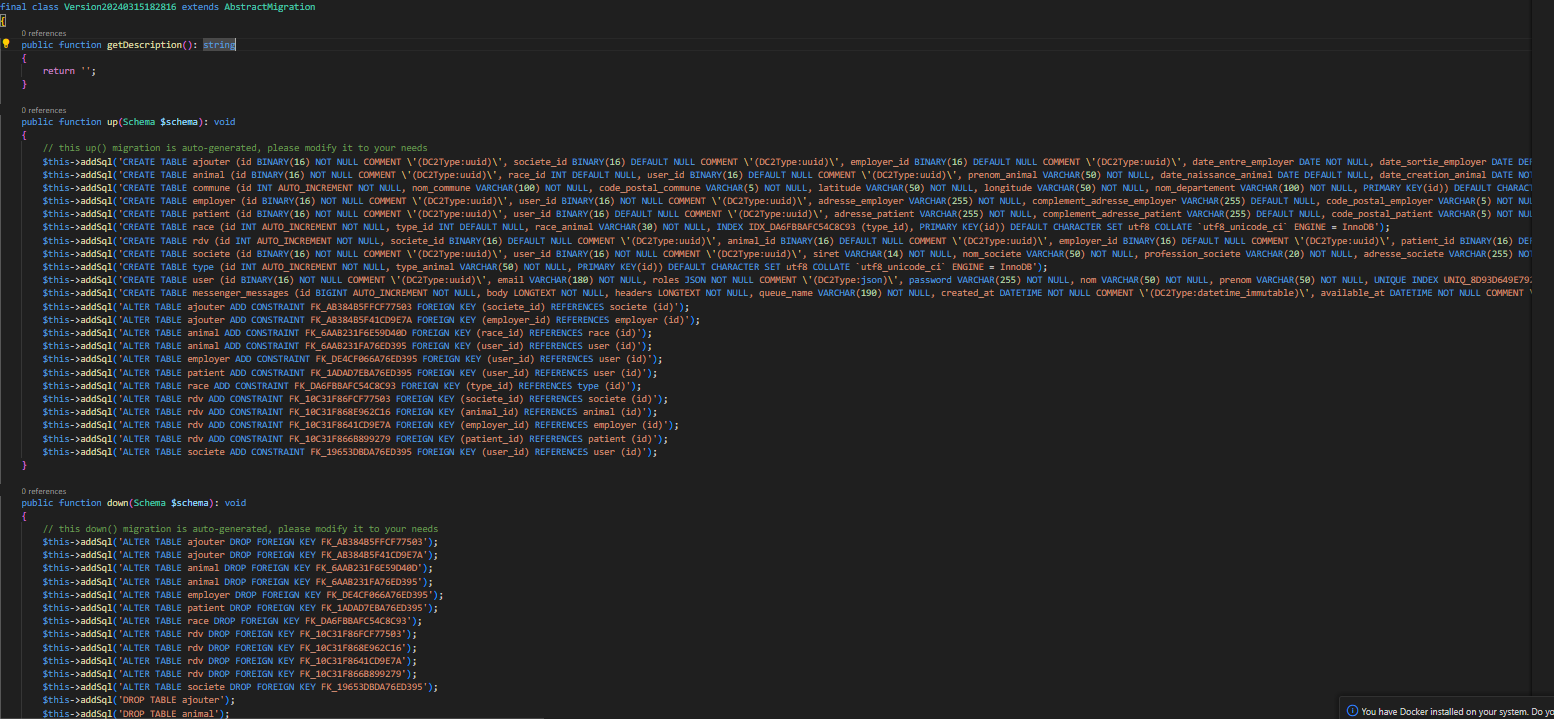
Grace à cette commande nous pouvons crée une table avec des getters ou setters suivant les renseignements et crée aussi le repository de son entity voici un exemple d’entity :

|  |  |
| --- | --- |
|  | Dans un premier temps il nous demande ne nom de l’entity qui sera le nom de notre table en nous donnant un exemple de convention avec la majuscule  Il nous demande le nom du champ  Puis le type de champs  Vu que c’est un string il nous demande la longueur  Et pour finir il nous demande si le champ est Null ou non.  L’id étant créé automatiquement il n’est judicieux d’en crée un |

Une fois crée nous pouvons faire un

php bin/console make:migration

qui va générer un fichier de migration ci-joint un exemple de fichier de migration



Dans celui-ci on peut voir qu’il a créé des commandes SQL avec CREATE TABLE il met le nom de l entity que j’ai créé qui sera utilisé pour la table dans phpmyadmin avec tous les champs renseigner et ses descriptifs qui sont dans le dossier Src/Entity/ il reprend toutes les entity les relations les clef … pour pouvoir insérer les données dans la bdd.

Une fois le fichier généré il faut faire un migrate pour insérer les tables et les champs dans la base de donnée avec cette commande

php bin/console doctrine:migrations:migrate

ci-joint mon entity user crée avec la commande

php bin/console make:user

|  |  |
| --- | --- |
|  | Symfony utilise le « namespacing » pour importer des class de son squelette ainsi que des classes que j’ai importées.  Il renseigne aussi le repository affecté pour son entité  Ici on informe que l’email est un champ unique  Il définit le type, le nom des champs et s’il peut être Null ou non  Comme l’on peut le voir j’ai importé la librairie Uuid pour me servir de l’Uuid V7 avec cette commande  composer require symfony/uid  Une fois importé je l’ai intégré a quasiment toutes les tables sensibles pour que les id soit généré de façon aléatoire et que sur chaque Entity il soit aussi unique  Et symfony gère automatiquement le hashage du mot de passe mais du code dans le controlleur est nécessaire |

|  |  |
| --- | --- |
|  | Il est aussi précisé le type de relation entre les entity dans ce cas l’entity user peux avoir plusieurs animaux car c’est du OneToMany  Sur cette partie nous avons les getters (lire)  Ici nous avons les setters (envoyer)  Il crée aussi un champs rôles (droit pour les utilisateurs du site ) automatiquement  Ensuite il faut définir les roles dans config/packages/security.yaml     access\_control:           - { path: ^/patient, roles: ROLE\_PATIENT }           - { path: ^/societe, roles: ROLE\_SOCIETE }           - { path: ^/employer, roles: ROLE\_EMPLOYER }           - { path: ^/administrateur, roles: ROLE\_ADMINISTRATEUR } |

Maintenant pour pouvoir utiliser cette entity j’ai besoin de crée un controlleur avec la commande

php bin/console make:controller

|  |  |
| --- | --- |
|  | Encore une fois symfony nous aide sur la convention il faut mettre chaque première lettre en majuscule et finir avec Controller même si sur la version 7 que j’utilise symfony le rajoute de lui-même.  Renseigne le nom  Puis symfony crée le controlleur mais aussi un dossier du nom du Controller avec la vue index.html.twig |

Voici le controller crée

|  |  |
| --- | --- |
|  | Ici on peut voir que symfony extends la class animal à AbstractController en réalité on hérite de la class crée par symfony pour simplifier le code et réduit le code répétitif  Il crée aussi une route par défaut pour notre controlleur qui nous sera utile pour afficher les vues ou faire des traitements sans vue  Puis dans la fonction il insère directement le render de la vue qu’il a créé ce qui permet un gain de rapidité |

***Création des entity et des controllers sous le mvc (model vue controller) :***

Sur le mvc c’est un peu plus complexe car il faut d’abord avoir le fichier index qui sera le gestionnaire des controlleurs ainsi que des models et des autres fichiers.

|  |  |
| --- | --- |
|  | Les require\_once permette de charger qu’une seule fois la page demandée sans avoir à la recharger constamment ici ce sont le header le controlleur qui va gérer le fonctionnement des autres Controller et le model  A chaque nouveau Controller il faut le renseigner dans la variable Controller sinon ce Controller ne sera pas reconnu  On lui précise le Controller par défaut  On commence par vérifier si un paramètre nommé controller est présent dans l'URL en utilisant **isset($\_GET['controller'])**, ce qui assure que le paramètre n'est pas null. Si ce paramètre est présent, le code récupère sa valeur. je vérifie ensuite si cette valeur se trouve dans le tableau **$controllers**, qui contient une liste de noms de contrôleurs valides autorisés par l'application. Si la valeur de **$\_GET['controller']** correspond à un élément dans le tableau **$controllers**, alors la variable **$nom\_controller** est définie avec le nom du contrôleur récupéré de l'URL.  On recrée une nouvelle variable $nom\_classe en concaténant Controlleur\_ car tous les nom des controlleur commence par ceci puis en ajoutant la variable récupéré préalablement pour ensuite accéder au controller |



Ceci est le controlleur principale qui va gérer tous les autres Controller car ceux-ci seront extends a celui-ci. Ce Controller principale permet de de récupérer l’action qui est renseigner dans le Controller enfants puis d’afficher les view renseigner dans le controller enfant.

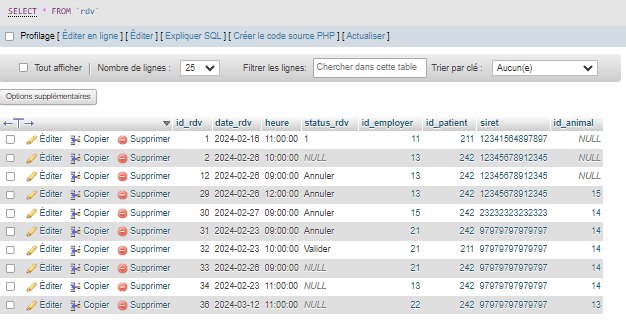
Voici le type de controller qui a pour nom Controller\_employer

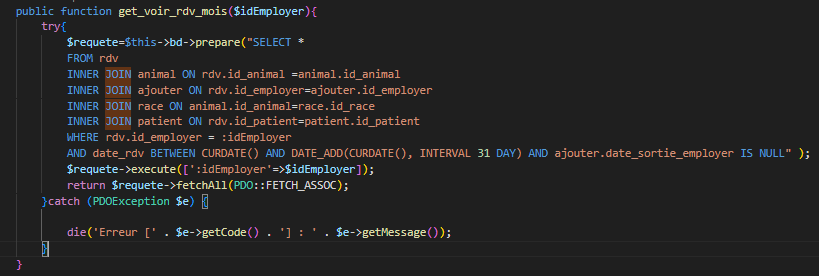
|  |  |
| --- | --- |
|  | On fait appel à la class qui est étendu à Controller  Sur la function action\_home lorsque l’on appel ce Controller avec pour action home on retourne sur la vue home .  Sur la function action\_session\_connect\_employer on fait appel au model grâce Model ::get\_model qui est une instance de la class Model    public static function get\_model()      {          if (is\_null(self::$instance)) {              self::$instance = new Model();          }          return self::$instance;      }  Ensuite je crée une variable $connecteEmployer sous forme de tableau avec pour clef connectEmployer ou j’envoie au model $m à la function get\_connexion\_employer les valeurs du formulaire envoyé par la méthode POST du form .  Puis après le traitement du model je retourne la vue session avec comment variable de traitement $connectEmployer |

Voici le code dans le model

|  |  |
| --- | --- |
|  | Dans cette fonction on commence a récupéré le tableau envoyé précédemment (array $data).  Je récupère l’email de connexion ainsi que le mot de passe qui sont dans mon tableau $\_POST[‘email’]et[‘password’]  Ensuite, je procède à une requête préparée pour rechercher tous les champs de la table employer où la colonne email correspond à l'email saisi. J'utilise un paramètre nommé :email dans ma requête préparée, et je lie la valeur de la variable $email à ce paramètre lors de l'exécution de la requête, ce qui me permet d'effectuer la recherche de manière sécurisée.  J’exécute la requête sous forme de tableau en associant le paramètre à sa variable  Puis je lance le fetch pour retrouver le premier résultat trouvé.  Et pour pouvoir retourné le résultat je vérifie que le hashage du mot de passe correspond bien au mot de passe avec password\_verify  Si le résultat est correct il renvoi au controller le résultat de $requete tandis que si c’est faux il renvoi false  Et bien sûr il y a la gestion des erreurs si la requête c est mal exécuté |

Exemple de requête SQL avec jointure et de la table Rdv



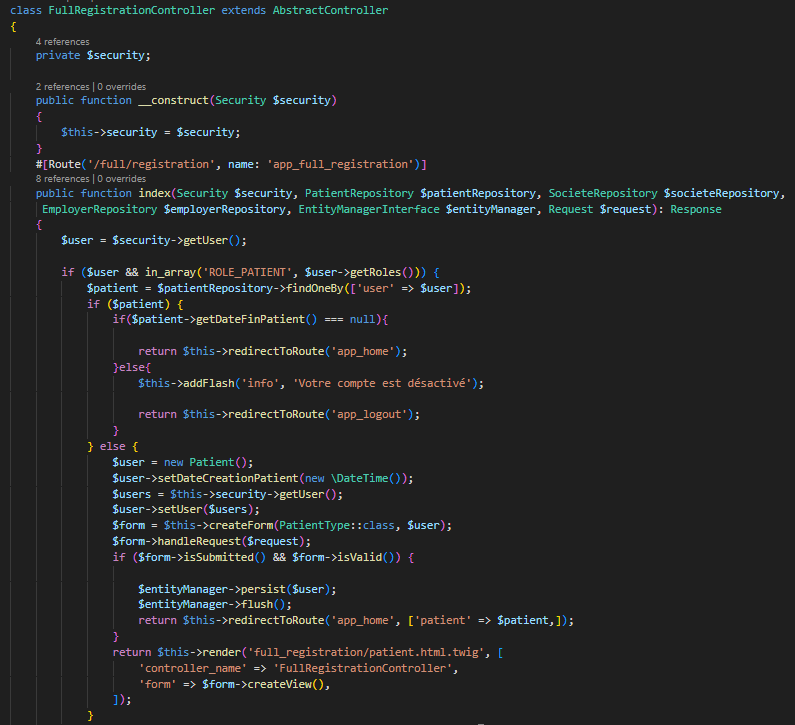


Dans la table rdv j’ai les clef étrangère suivante id\_employer , id \_patient, le Siret de l’entreprise car un employé peux travailler dans plusieurs entreprises(mais cette requête est pour le gestionnaire de la société) et j’ai aussi besoin de l’id animal car un patient peux avoir plusieurs animaux .

Donc pour avoir un affichage des nom prénom j’ai dû faire une requête ou dans la table rdv je joins la table animal pour retrouver le nom de l’animal mais aussi afin de récupéré l id race de l’animal pour que le professionnel connaisse les informations dessus le prénom, date de naissance de cette table animal j’ai la clef étrangère de la race, je fais donc une nouvelle jointure pour pouvoir afficher la race de celui-ci, dans la table ajouter je recherche en fonction de l’id employé récupéré préalablement ainsi que la jointure sur la table patient pour récupéré les information de la table patient le tout sur la condition de recherche que rdv.id\_employer corresponde a l idEmployer récupéré précédemment et dans cette requete pour j’ai définis un espace-temps d’un mois à la date du jours donc je fais un Between pour filtre les dates entre le jours « j » et un intervalle de 31 jours le tout agrémenté d’un condition supplémentaire que l’employé ne soit pas sortie de l’entreprise car ce champs est initialement null mais dès qu’il quitte une société le champs prend la date de sortie validé par l’employeur.

***Symphonie :***

Grace au DQL les requêtes sont géré plus facilement et donc la lecture du code est simplifiée



Au départ je crée un constructeur qui fait appel a Security bundle de symfony ,cela me permet de récupérer les information de l’user connecté

Cela me permet de pouvoir mettre des conditions en fonction de ma variable $user ,sur cet exemple je recherche le role de l’utilisateur et donc si il est patient on continue

Ensuite je crée une variable patient ou je fait appel au repository de patient pour pouvoir effectuer une requete DQL avec le findByOne car dans cette partie il ne peux y avoir qu un seul utilisateur comme vu precedemment par rapport a l adresse email d inscription .

Ensuite je vérifie que patient n’est pas nul puis que son compte ne soit pas désactivé avec

if($patient->getDateFinPatient() === null){

ensuite il est redirigé soit vers l accueil avec

 return $this->redirectToRoute('app\_home');

sinon il aura un message sur la page d accueil avec addFlash

 $this->addFlash('info', 'Votre compte est désactivé');

                    return $this->redirectToRoute('app\_logout');

et ne pourra se connecté.

Si le patient est n’existe pas c’est qu il vient de s’enregistrer a partir de ce moment je fait appel a l’objet Patient() avec cette fonction

$user = new Patient();

J’envoie la date du jours a $user

  $user->setDateCreationPatient(new \DateTime());

Ses deux lignes me permettent de passer l id user

Et de l envoye à $user

 $users = $this->security->getUser();

                $user->setUser($users);

Ensuite je fais appel au formulaire crée par rapport a l entity qui a pour nom PatientType

 $form = $this->createForm(PatientType::class, $user);

                $form->handleRequest($request);