135 rue du Madrillet
Saint du Etienne du Rouvray

DOSSIER PROJET
SITE WEB

L'ATELIER BARBERSHOP ROUEN

PAR

M. BEAUROY-EUSTACHE JOËL

promotion Grace Hopper 2021

Table des matières

Compétences du *Référentiel Emploi Activités Compétences du Titre Professionnel (REAC)* **couvertes durant le stage**

- 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

- 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

Résumé du projet :

Stage pratique DWWM chez un coiffeur barbier

Durant la période de stage pratique de 9 semaines, j'ai réalisé un site vitrine pour un salon de coiffure du centre-ville de Rouen. Le projet s'est déroulé en plusieurs phases :

- Élaboration du cahier des charges avec le gérant du salon
- Détail des fonctionnalités
- Conception des use cases
- Conception des wireframes
- Établissement du schéma de base de données
- Programmation de la partie back-end
- Programmation de la partie front-end

Dans ce secteur d'activité, la concurrence est très représentée sur le web. Le salon qui dispose déjà d'un Facebook et d'un Instagram doit donc également avoir son propre site. Le gérant souhaite ainsi accroître sa visibilité et par ce biais gagner une nouvelle clientèle.

Le salon de coiffure n'ayant pas d'ordinateur ni la place au sein du salon, le stage s'est entièrement déroulé à distance. Cela m'a permis d'appréhender le métier depuis une situation se rapprochant de celle du métier de développeur web en free-lance. Durant ce stage, j'ai dû être à la fois le commercial, le graphiste et le développeur full stack du projet.

Cahier des charges du projet

Le cahier des charges a été établi lors d'un rendez-vous avec le gérant et un employé. Les besoins exprimés au cours de cet entretien ont été :

- Un site avec les coordonnées du salon (adresse, téléphone, Google maps intégrée)
- Mise en avant des marques utilisées et vendues par le salon
- Possibilité d'ajouter des photos de réalisations
- Prestations et tarifs du salon
- Un site représentatif du salon (couleurs et ambiance)
- Création d'un logo

Le gérant souhaite un site vitrine avec une mise en avant des coordonnées du salon afin d'être plus facilement accessible pour la prise de rendez-vous mais il ne désire pas une prise de rendez-vous en ligne. Le salon étant situé au cœur du centre ville mais dans une rue peu commerçante il souhaite mettre une photo de la devanture du salon et également une Google maps intégrée.

Une partie de sa clientèle est sensible à certaines marques utilisées et vendues par le salon. Il est donc nécessaire de faire connaître à un plus large publique la possibilité d'acheter ces produits sur place.

De nombreux salons de coiffure affichent la tarification des services et peuvent ainsi offrir la possibilité aux clients de comparer les salons. Le gérant du salon souhaite donc faire de même.

La concurrence étant très présente sur le web, il faut se démarquer en affichant certaines des réalisations faites au salon sur le site. Instagram et Facebook sont très utiles pour diffuser du contenu de ce type mais ne touchent pas forcément les clients potentiels. C'est pourquoi l'ajout ponctuel de photos est demandé.

La décoration du salon de coiffure lui donne un caractère ancien et moderne à la fois. Cela doit se refléter sur le thème du site.
L'entreprise ne dispose pas de logo personnalisé. La demande de création d'un logo est prise en compte.

Spécifications fonctionnelles

- Voir les coordonnées :

L'utilisateur doit voir clairement toutes les coordonnées du salon. Les numéros de téléphone sont cliquables pour lancer la communication. L'adresse du salon située dans le footer renvoie à un ancrage interne sur la Google maps de la page.

- Voir la Google maps :

La Google maps pointe sur le salon et affiche l'adresse. Le lien est généré depuis le site Google maps et inséré dans le code HTML. Il est responsive.

- Voir les marques utilisées et vendues :

L'utilisateur doit pouvoir voir les marques en naviguant.

- Ajouter des photos :

Dans un espace sécurisé par un mot de passe, l'utilisateur loggé peut ajouter et supprimer des images depuis un téléphone mobile.

- Voir les photos :

L'utilisateur voit les réalisations du salon dans des blocs distincts (coupes femme, coupes homme et tailles de barbe).

- Avoir des liens vers Instagram et Facebook :

L'utilisateur voit et peut être redirigé vers les réseaux sociaux du salon via des icônes représentant les logos Instagram et Facebook. Les liens ouvriront une nouvelle fenêtre afin de garder la possibilité de revenir facilement sur la page du salon.

- Voir les prix et les prestations du salon :

L'utilisateur voit les tarifs pratiqués et les prestations pratiquées par le salon dans des blocs distincts (coupes, tailles de barbe et colorations). Dans un espace sécurisé par un mot de passe, sur une page dédiée, l'utilisateur loggé peut ajouter, supprimer ou modifier les prestations et les tarifs.

Arborescence fonctionnelle

Arborescence fonctionnelle **fichier annexe :**

- Arborescence fonctionnelle de l'Atelier.PDF

Spécifications techniques

- Framework Symfony
- SGBD MySql
- Easy Admin bundle
- Vich Uploader bundle
- Mobile first
- Bootstrap V 5
- HTML5 CSS3 Javascript
- Serveur : Apache2

Logiciels et outils utilisés

- Visual Studio Code
- Git / Github
- Xampp
- Libre Office
- Gimp

Wireframes

Wireframe mobile **fichier annexe** :

- Wireframe Mobile l'Atelier.PDF

Wireframe Tablette **fichier annexe** :

- Wireframe Tablette l'Atelier.PDF

Wireframe PC **fichier annexe** :

- Wireframe PC l'Atelier.PDF

FRONT-END

Interface statique et adaptable

Développement en Mobile first

Afin de répondre au mieux aux attentes des clients et de s'adapter aux différents outils de navigation, l'application a été développée en **mobile first**. Cela se traduit par la nécessité de commencer la conception de l'application en concevant des **wireframes** (fichiers annexe) pour mobile, puis d'agrandir les formats en adaptant la taille des éléments à afficher. Lors du développement de l'application, l'utilisation de la vue adaptative du navigateur qui simule différents types de taille d'écran ainsi que leur orientation (paysage ou portrait) est indispensable.

Application responsive

Pour rendre l'application responsive je me suis servi de **Bootstrap 5** pour son système de grille et de **Medias queries** qui m'ont permis de créer des points de ruptures en fonction des dimensions des écrans. J'ai utilisé les mêmes points de ruptures pour les **Media queries** que ceux de **Bootstrap**. Pour le redimensionnement des éléments, les unités de mesure relative telles que **Viewport Height** et **Viewport Width** ont été adoptées dans le but de garder une cohérence d'affichage.

Ratio des images

Pour conserver le ratio des images, j'ai combiné les propriétés « **max-width** », « **max-height** » avec les propriétés « **width** » et « **height** » en **CSS.** Comme leur nom l'indique, les propriétés « max » définissent les dimensions maximales de l'élément. Elles sont ici définies en « **vh** » et « **vw** » pour prendre le même pourcentage de place sur chaque taille de viewport . Puis j'ai ajouté « **auto** » aux propriétés « **width** » et « **height** ».

Développer une interface utilisateur web dynamique

Afficher et cacher un élément HTML en fonction de la hauteur de scroll

Afin de ne pas cacher le logo de l'entreprise lors du scroll sur la page d'accueil, j'ai utilisé **JavaScript**. Pour cela, j'ai ajouté un « id » « hiddenNavabr » sur la barre de navigation et un autre « id » « logoContainer » sur l'élément HTML contenant le logo. Puis avec la méthode « getElementById », j'ai récupéré chacun des éléments et les stocks dans des variables. J'ai stocké également dans une autre variable la propriété « **clientHeight** » qui mesure la hauteur de l'élément ciblé (le bloc contenant le logo) au moment du chargement de la page. La valeur sera variable en fonction du **viewport**. Sur l'objet « **window** » qui représente la page web chargée, j'ai placé la méthode « addEventListener » qui écoute l'événement spécifié (ici le scroll) et qui déclenche la fonction « showNavbar ». Cette fonction compare la valeur du scroll vertical parcouru à la hauteur du logo. Lorsque la valeur du scroll est supérieure à celle du logo, avec la méthode « add » et la propriété « classList », j'ai ajouté une classe définie au préalable dans la feuille de style en cascade(CSS). Cette classe ajoute une opacité de 0,8 sur la barre de navigation qui avait auparavant une opacité de 0,0. Lorsque la valeur devient inférieure à la hauteur du logo, cette classe est retirée afin que la barre de navigation disparaisse à nouveau.

```
window.addEventListener('scroll', function showNavbar(){
    if(window.scrollY > logoContainer_Height){
        hiddenNav.classList.add('show');
    }
    else{
        hiddenNav.classList.remove('show');
    }
});
```

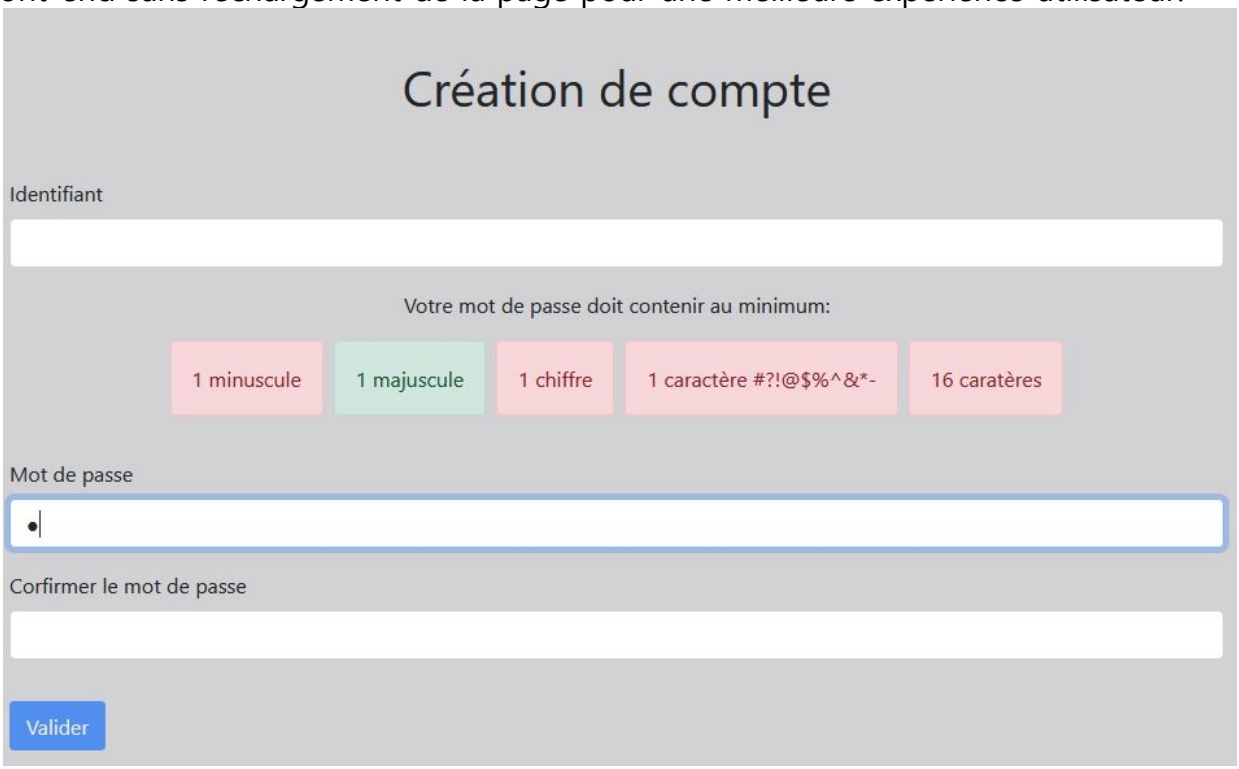
Fonction JavaScript

```
.navbar-dark{
  background-color: #000000;
  opacity: 0.0;
  font-family:'Open Sans Condensed';
  color: white;
  transition: all 0.5s ease-in-out;
}
.navbar-dark.show{
  opacity: 0.8;
}
```

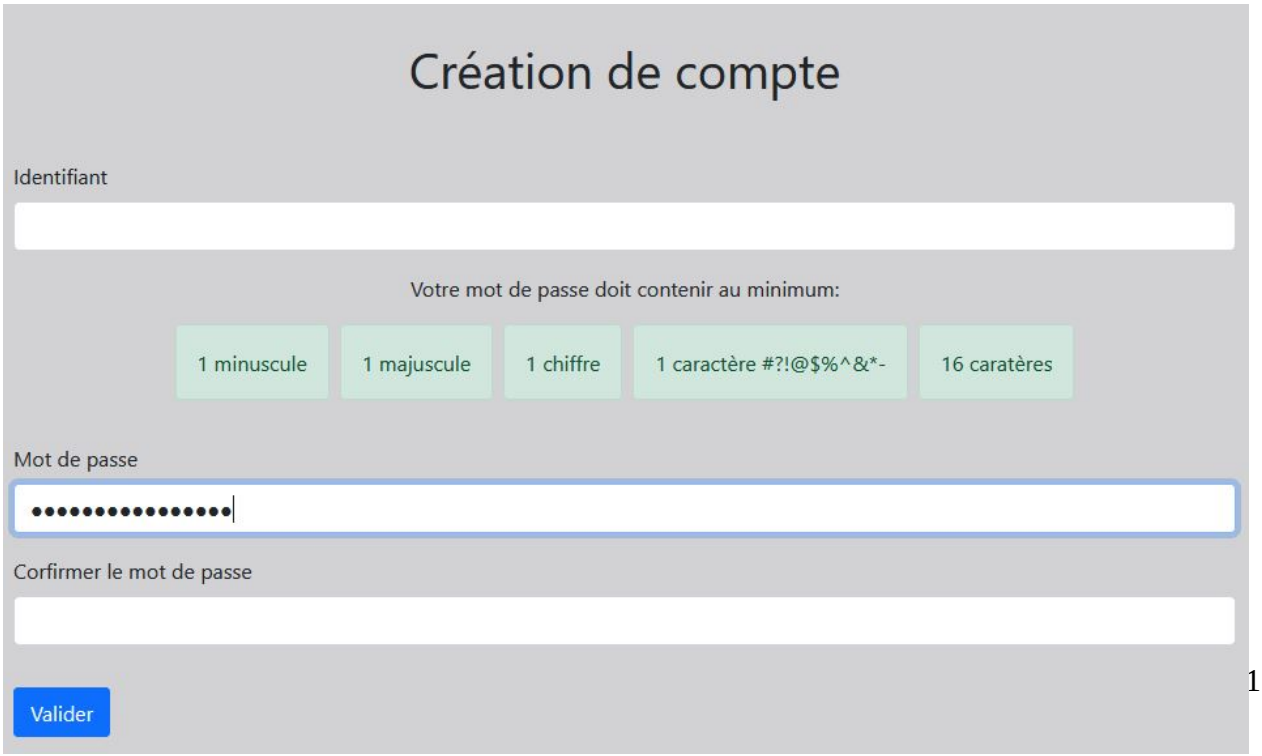
CSS

Expressions Régulières pour valider la force d'un mot passe

Pour contraindre les utilisateurs à utiliser un mot de passe fort lors de la création de leur compte utilisateur, j'ai développé une fonction JavaScript qui se sert d'expressions régulières pour contrôler le mot de passe saisi. Cela se fait sur la partie front-end sans rechargement de la page pour une meilleure expérience utilisateur.



Les contraintes sont indiquées clairement en rouge et changent de couleur et passent au vert lorsque celles-ci ont été respectées. Une fois que toutes les contraintes ont été levées, le bouton « Valider » passe de la propriété « disable » « true » à «false » etpermet l'envoi du formulaire.



```
/*Html elements*/
let passwordForm = document.getElementById('registration_form_plainPassword_first');
let button = document.getElementById('button');
let lowercase = document.getElementById('lowercase');
let uppercase = document.getElementById('uppercase');
let number = document.getElementById('number');
let special = document.getElementById('special');
let passwordlength = document.getElementById('length');
 /*Initialize parameter*/
let passwordMinLength = 16;
```

Dans un premier temps, j'ai récupéré par leur « Id » chaque élément HTML qui devra changer de couleur, que j'ai stocké dans des variables portant le nom de la contrainte à vérifier. Le premier champ « input » du mot de passe est également récupéré pour vérifier la valeur donnée saisie.

```
passwordForm.addEventListener('input', ()=>{

    let inputPassword = passwordForm.value;

    /*Lowercase control*/
    if(/[a-z]/.test(inputPassword)){
        /*Change Class*/
        lowercase.classList.replace('alert-danger','alert-success');
    }else{
        lowercase.classList.replace('alert-success','alert-danger');
    }
    /*Upperercase control*/
    if(/[A-Z]/.test(inputPassword)){
        uppercase.classList.replace('alert-danger','alert-success');
    }else{
        uppercase.classList.replace('alert-success','alert-danger');
    }
    /*Number control*/
    if(/[0-9]/.test(inputPassword)){
        number.classList.replace('alert-danger','alert-success')
    }else{
        number.classList.replace('alert-success','alert-danger');
    }
    /* Special character control */
    if(/[#?!@$%^&*-]/.test(inputPassword)){
        special.classList.replace('alert-danger','alert-success')
    }else{
        special.classList.replace('alert-success','alert-danger');
    }
    /* Lenght control */
    if(inputPassword.length >= passwordMinLength){
        passwordlength.classList.replace('alert-danger','alert-success');
    }else{
        passwordlength.classList.replace('alert-success','alert-danger');
    }
    /*Verify all controls are true and active validation button */
    if(/^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-]).{16,}$/.test(inputPassword)){
        button.disabled = false;
    }else{
        button.disabled = true;
    }
})
```

Une fonction fléchée est déclenchée par l'écouteur d 'éventement de type « input » sur le premier champ du mot de passe stocké dans la variable « passwordForm ».
La propriété « value » retourne la valeur du champ texte de « passwordForm ».
Dans une succession d'instructions « if » « else » qui sert à exécuter du code de manière conditionnelle, et avec la méthode « test() » qui vérifie la correspondance entre les valeurs renvoyées par « inputPassword » et les expressions régulières définies, la méthode « test() » retourne un booléen, « true » en cas de correspondance et «false » dans le cas contraire.
Lorsque la méthode « test() » renvoie « true », avec la méthode « replace() » et la propriété « classList », je remplace la classe « alert-danger » par la classe « alert-sucess ». Cela a pour effet de changer la couleur de l'élément HTML et de le faire passer du rouge au vert.
Dans le cas ou la méthode « test() » renvoie « false », l'élément HTML reste rouge.
Pour vérifier la longueur du mot de passe, j'ai utilisé la propriété « length » qui mesure la longueur des valeurs d'« inputPassword ». Puis j'ai comparé cette valeur à celle initialisée dans la variable « passwordMinLength » qui sert de valeur de référence. Ensuite, comme précédemment, j'ai échangé les classes sur l'élément HTML afin de le faire changer de couleur lorsque la valeur est supérieure ou égale à 16 caractères.

Pour activer le bouton de validation du formulaire, j'ai recouru à une seule expression régulière qui reprend toutes les vérifications précédentes et qui, dans une instruction conditionnelle, active le bouton si l'expression est vérifiée. Dans le cas inverse, il lui conserve l'attribut « disabled ».

Animation en CSS

Pour inviter l'utilisateur à scroller vers le bas lorsqu'il est sur la page d'accueil, j'ai animé un chevron pointant vers le bas. Ce chevron est cliquable et amène a un ancrage plus bas dans la page. Le chevron est une image PNG importée dans une balise « img ». Avec une règle « **Keyframes** » qui définit un pourcentage de réalisation de l'animation, j'ai appliqué un mouvement vertical de 25px. L'animation nommée « **moveUpDown** » est définie pour une durée totale de 0,8 secondes se répétant à l'infini. La valeur « ease in out » indique qu'elle commence lentement puis s'accélère et ralentit à la fin. La propriété « alternate » définit que l'animation va se jouer dans un cycle qui va s'inverser une fois le premier cycle terminé.
Ce chevron étant inutile sur l'affichage d'un mobile, il comporte la classe « **d-sm-block** » pour n'apparaître qu'à partir d'un écran supérieur à 576px.

```css
@keyframes moveUpDown {
  100% {
    transform: translateY(25px);
  }
}
.logoContainer_Chevron_Img{
  width: 1.5rem;
  height: 1.5rem;
  animation: moveUpDown .8s infinite ease-in-out alternate;
}
```

Sur viewport inférieur à 576px

Sur viewport supérieur à 576px

14

BACK-END

Schéma base de données **fichier annexe** :

- schéma BDD l'Atelier.PDF

Création d'un compte utilisateur avec un formulaire

Pour que des utilisateurs puissent accéder à la partie d'administration du site, j'ai généré un formulaire en ligne de commande, il est nommé : « **RegistrationController** ».

```php
namespace App\Controller;

use App\Entity\Admin;
use App\Form\RegistrationFormType;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface;

class RegistrationController extends AbstractController
{
    #[Route('/registerFormWebsite76', name: 'app_register')]
    public function register(Request $request, UserPasswordHasherInterface $passwordEncoder): Response
    {
        $user = new Admin();
        $form = $this->createForm(RegistrationFormType::class, $user);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            // encode the plain password
            $user->setPassword(
                $passwordEncoder->hashPassword(
                    $user,
                    $form->get('plainPassword')->getData()
                )
            );

            $entityManager = $this->getDoctrine()->getManager();
            $entityManager->persist($user);
            $entityManager->flush();
            // do anything else you need here, like send an email

            return $this->redirectToRoute('index');
        }

        return $this->render('registration/register.html.twig', [
            'registrationForm' => $form->createView(),
        ]);
    }
}
```

- Ce formulaire hérite des propriétés et des méthodes de la classe « **AbastractController** ». Une classe **abstraite** ne peut être instanciée et permet l'utilisation des méthodes qu'elle contient et oblige à définir les méthodes abstraites qu'elle contiendrait. Ici, est utilisée la méthode « getDoctrine » qui sert à appeler la méthode « getManager », qui **synchronise** elle-même les objets dans la base de données.
- Le formulaire est accessible à l'URL « /registerFormWebsite76 » qui porte le « **name** »(**URI**) « app_register ». Le « name » est utile pour faire référence à cette URL sans avoir à l'écrire entièrement.

- Le « RegistrationController » ne contient qu'une seule méthode « register » qui instancie et prend en paramètre l'objet **Request** (contient les informations sur la requête) et l'interface **UserPasswordHasherInterface** (Ici pour hasher le mot de passe afin qu'il soit protégé en BDD). La fonction « register » renvoie un objet **Response** (la réponse du serveur).

- Dans la variable « $user » est stocké l'objet « Admin ».

- Dans la variable « $form » est stocké le formulaire « RegistrationFormType » qui est généré sur les bases de l'objet « Admin ».

- Formulaire est ensuite récupéré puis traité.

- Dans une instruction « if » qui permet une exécution **conditionnelle** de la suite du code, on vérifie que le formulaire a été **soumis** et que toutes les **contraintes** de validation de « RegistrationFormType » ont été respectées. Si non, un objet « **Response** » avec la méthode « **render** » renvoie sur la page du formulaire. Si le formulaire soumis est valide, le code qui suit est exécuté.

- La méthode « setPassword » de l'entité Admin « hash » le mot de passe saisi dans le formulaire.

- « Entity Manager » manipule les données provenant de l'entité Admin afin qu'elle soient interprétable par la base de données.

- La méthode « **persist** » informe Doctrine qu'un objet de type Admin va être ajouté à la BDD.

- Le « **flush** » enregistre les données dans la base sous forme de **transaction** (permet l'annulation de l'enregistrement des données en cas d'échec). Une fois le code exécuté, la fonction renvoie sur la page qui porte le « name » « index » .

Création d'une base de données

Avec Symfony, la création d'une base de données s'effectue en ligne commande par l'intermédiaire de **Doctrine ORM**. ORM siginifie « Object Relational Mapping », Doctrine ORM permet de faire le **lien** entre les objets dans le code, les tables et les champs de la base de données.

A la racine du projet, dans un ficher nommé « .env » ou « .env.local » pour une application développée sur un « repository » à plusieurs contributeurs, il faut paramétrer l'URL qui permettra la création, puis la connexion à la base de données(BDD). Pour ce projet, j'ai travaillé sur une BDD de type MySql. La ligne à dé-commenter se décompose comme ceci :

Avec la commande « doctrine:database:create », la base se crée.

Développer les composants d'accès aux données

Avant de pouvoir créer, ajouter, modifier, supprimer des données, il faut créer les entités qui regroupent des attributs et des méthodes. Ces entités sont des objets qui seront représentés en base de données par les tables et les champs.
Pour générer une entité, on peut utiliser la ligne de commande en tapant « make:entity ». A la suite de cela, il faut répondre à une série de questions qui créeront les **attributs** avec leur type dans des **annotations** que Doctrine utilisera par la suite pour **mapper** les données. Lors de la migration, les attributs deviendront les champs. Les « getters » et les « setters » de ses attributs seront également créés dans l'entité.

Pour contrôler ce qui va être généré en base de données, la commande «**doctrine:schema:update --dump-sql** » permet de voir la migration qui sera produite. De cette manière, on peut intervenir pour faire des modifications en amont si nécessaire. La commande « **make:migration** » crée le fichier de migration dans lequel il est pareillement possible de faire des modifications avant de taper la commande « **doctrine:migrations:migrate** » qui elle-même créera la table avec ses champs et leur type.

A présent, il est essentiel de créer un « **controller** » dans lequel le code logique est exécuté. Un « controller » récupère un objet « **Request** » et renvoie un objet « **Response** ». C'est dans un « controller » que l'on peut effectuer les opérations du « **CRUD** ». Le CRUD est l'acronyme de Create, Read, Update et Delete qui sont les termes pour la manipulation des données.
C'est en se servant des méthodes des « **getters** » et des « **setters** » d'une entité que l'ont peut exécuter le « CRUD » dans un « controller ».

1- Récupérer les données

Pour cette application, le « controller » gérant la page d'accueil du site s'appelle « IndexController ».

```php
class IndexController extends AbstractController
{
    #[Route('/', name: 'index')]
    public function index(ServiceRepository $ServiceRepository, ImageRepository $imageRepository,OpeningHoursRepository $openingHoursRepository,InformationRepository
    {
        $services = $ServiceRepository->findall();
        $opening = $openingHoursRepository->findAll();
        $info = $informationRepository->findAll();
        $img = $imageRepository->findall();

        return $this->render('index/index.html.twig', [
            'services' => $services,
            'images' => $img,
            'openingHours' => $opening,
            'informations' => $info,
        ]);
    }
}
```

C'est à partir de celui-ci que les demandes de requête partent.

- L'« IndexController » est à la racine de l'application sur la page qui porte le « name » « index ».

- La fonction index instancie et prend en paramètre les « Repository » des entités requises. Les « Repository » sont des classes qui héritent d'une classe «ServiceEntityRepository » qui hérite de la classe «EntityRepository » qui elle-même possède les méthodes « find, findAll, findOneBy» etc.
- La fonction « index » renvoie un objet de type « Response ».

- La méthode « findAll » est appelée sur chaque « Repository ». Puis elle est stockée dans une variable qui est renvoyée dans un tableau de la méthode « render » de l'objet « Response ». La méthode « render » prend en paramètre le fichier contenant « template » Twig et le tableau de type associatif (clef, valeur) qui contient toutes les variables. La clef est une chaîne de caractères qui sera utilisée dans le « template » pour l'affichage des données par le biais d'une boucle. La valeur stocke le contenu récupéré par les requêtes.

2- Afficher les données

Depuis le template de la page « index », avec une boucle « for » qui affiche les valeurs stockées dans la variable « openingHours » renvoyée par la méthode « render », la boucle opère des itérations sur chaque entité récupérée par la requête et affiche les valeurs correspondantes aux propriétés demandées dans le template.

```
<div class="container-fluid infoContainer pt-5" id="informationsContainer">
  <div class="row d-flex justify-content-evenly ">
    <div class="col-9 col-sm-5 col-md-4 col-lg-4 col-xl-3 mb-2 infoContainer_Opening">
      <div class="text-center">
      <h3 class="text-center">Horaires</h3>
        {% for opening in openingHours %}
          <span class="infoContainer_Opening_Day">{{ opening.day }}</span>
          <span class="infoContainer_Opening_OpenClose">{{ opening.open }}{{ opening.close }}</span>
        {% endfor %}
        </div>
    </div>
```

3- Insérer des données

Pour cette application, l'envoi d'image sur le serveur est nécessaire. Les images ne sont pas directement stockées dans la base de données mais uniquement certaines de leurs propriétés dont le chemin d'accès au fichier. Le dossier où sont stockées les images est dans le dossier « public » de l'application afin que les images soient accessibles pour les visiteurs. Pour la gestion de l'envoi des images, j'ai utilisé le bundle Vich Uploader. Pour l'upload des images, il faut paramétrer le mapping de l'envoi dans le fichier « config/packages/vich_uplaoder.yaml ».

```yaml
vich_uploader:
    db_driver: orm
    mappings:
        upload_img:
            uri_prefix: '%upload_img%'
            upload_destination: '%kernel.project_dir%/public/image'
            namer: Vich\UploaderBundle\Naming\SmartUniqueNamer

            inject_on_load: false
            delete_on_update: true
```

- Nom du mapping : upload_img.

- URI prefix : Paramètre mis entre deux signes « % » afin d'être interprété.

- Upload_destination: Qui désigne le chemin complet (depuis la racine) pour le stockage des fichiers.

- Namer : SmartUniqueNamer : Méthode qui va ajouter au nom du fichier envoyé une série de caractères de manière à le rendre unique pour éviter le stockage de fichiers portant le même nom.

- Inject_on_load : false : Pour que le fichier envoyé ne soit pas une instance de « HttpFoudation » mais une instance de « UploadedFile ».

- Delete_on_update : true : Pour supprimer le fichier si un autre prend sa place dans l'entité. Ceci préserve l'espace de stockage.

Dans le fichier « config/services.yaml », il faut ajouter aux paramètres le mapping

utilisé qui pointe vers le dossier où seront stockées les images.

```
parameters:
    upload_img: /image
```

L'entité Image reçoit en annotation la classe Uploadable avec l'alias Vich pour faire référence au bundle.

```
/**
 * @ORM\Entity(repositoryClass=ImageRepository::class)
 * @Vich\Uploadable
 */
class Image
{
```

L'attribut « imageFile » reçoit une variable de type « File », qui est un objet de la classe « HttpFoundation » et qui mappe les données des images avec les paramètres définis et renvoie le chemin d'accès aux fichiers.

```
     * @Vich\UploadableField(mapping="upload_img", fileNameProperty="imageName", size="imageSize")
     *
     * @var File|null
     */
    private $imageFile;
```

Le setter reçoit un objet nullable par défaut de type « File » (le paramétrage « (= null) » n'est plus recommandé depuis PHP 8.0). La méthode n'a pas de valeur de retour. L'attribut « imageFile » reçoit la variable « $imageFile ». Dans une instruction conditionnelle il est vérifié que, si la variable « $imageFile » n'est pas « null » il faut alors ajouter l'objet « DateTimeImmutable » qui à son instanciation renvoie la date et l'heure de la zone géographique dans laquelle on se trouve.

Le getter renvoie un objet de type « File » dans l'attribut « imageFile » ou « null ».

```
     * @param File|\Symfony\Component\HttpFoundation\File\UploadedFile|null $imageFile
     */
    public function setImageFile(?File $imageFile = null): void
    {
        $this->imageFile = $imageFile;

        if (null !== $imageFile) {
            // It is required that at least one field changes if you are using doctrine
            // otherwise the event listeners won't be called and the file is lost
            $this->updatedAt = new \DateTimeImmutable();
        }
    }

    public function getImageFile(): ?File
    {
        return $this->imageFile;
    }
```

Depuis l'interface graphique de l'application générée par Easy Admin j'ai testé manuellement l'ajout d'une image. Puis j'ai contrôlé sur la page « index » de l'entité « Image » si tous les champs étaient complets et la miniature m'a assuré que l'image avait bien été retrouvée après l'envoi. Puis j'ai contrôlé les champs directement dans la base de données avec « PhpMyAdmin ».

Ajout d'image depuis Easy Admin

Page « index » de Easy Admin affichant les valeurs de l'entité « Image »

Base de données avec Php My Admin

Configuration des champs dans Easy Admin

Easy Admin nécessite sa configuration. La fonction « configureFields » permet de paramétrer les types de champs que l'on retrouve sur les pages du « CRUD ».

```php
public function configureFields(string $pageName): iterable
{
    return [
        TextField::new('imageName', 'Nom du fichier')
            ->OnlyOnIndex(),
        TextareaField::new('imageFile', 'Image')
            ->setFormType(VichImageType::class)
            ->OnlyWhenCreating()
            ->setRequired(true),
         /*Create thumbnail after uploalding */
        ImageField::new('imageName', 'Image')
            ->onlyOnIndex()
            ->setBasePath('/image'),
        /*Get Category data*/
        AssociationField::new('category', 'Categorie')
            ->setRequired(true),
        TextField::new('imageAlt', 'Attribut HTML "ALT"(courte description pour référencement naturel)'),

        ];
}
```

- Cette fonction prend en paramètre le nom de la page sur laquelle on se trouve (Edit, New, Detail).

- Elle retourne un iterable (tableau).

Pour chaque champ, un objet de type « Field » est instancié et prend en paramètre le nom de la propriété et ici un nouveau label.

- « ImageFile» reprend les propriétés de « VichImageType » lors de l'ajout d'une nouvelle image. « VichImageType » indique à EasyAdmin que c'est un champ pour l'envoi de fichier vers le dossier désigné dans le mapping.

- L'objet « ImageField » prend en paramètre « imageName ». Il sera visible uniquement sur la page index et pointe vers le chemin où est stockée l'image de l'objet « Image » afin de créer une miniature.

- L'objet « AssociationField » prend en paramètre l'entité « Category » et permet d'afficher un dropdown avec les données contenues dans les entités « Category ».

Sécurité

Login

Afin de se prémunir des attaques par **force brute** j'ai paramétré le «firewall» dans le fichier security.yaml. Le « Firewall » gère l'authentification. J'ai limité à 4 le nombre de tentatives de connexions sur la page « login ». Cette limitation prend en compte le nom de l'utilisateur et son adresse IP. Au delà de 4 tentatives, l'adresse IP et le nom d'utilisateur sont bloqués pendant 3 heures.

```
    login_throttling:
        max_attempts: 4
        interval: '180 minutes'
```

Les routes

Pour interdire l'accès aux pages d'administration du site aux utilisateurs **non loggés** et n'ayant pas le « role » requis, j'ai utilisé les paramètres d'« access_control » dans le fichier«security,yaml » . « access_control » permet d'interdire des routes en fonction du «**role**» de l'utilisateur. La hiérarchie des

```
role_hierarchy:
    ROLE_ADMIN:       ROLE_USER
    ROLE_SUPER_ADMIN: [ROLE_USER, ROLE_ADMIN]
```

```
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    # - { path: ^/profile, roles: ROLE_USER }
```

- Dans une expression régulière il est défini que toutes les **routes** débutant par « admin » ne sont accessibles qu'aux utilisateurs ayant au minimum le « ROLE_ADMIN ».

Les controllers

Il y a une autre façon de restreindre l'accès : en utilisant les **annotations**. Ici, c'est la **totalité** des « controllers » de la classe « DashboardController » qui ne sont accessibles qu'aux utilisateurs ayant au moins le « ROLE_ADMIN ».

```
use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;

/**
 * @IsGranted("ROLE_ADMIN")
 */
class DashboardController extends AbstractDashboardController
{
```

L'authentification

Pour que les utilisateurs puissent se connecter via le formulaire de « login », il est nécessaire de les authentifier. C'est la classe « Authenticator » qui s'en occupe. Elle hérite de la classe « AbstractLoginFormAuthenticator ». Cette classe sert à vérifier que la requête vient bien de l'URL de « login » et que c'est une méthode « POST ». C'est également elle qui redirige vers « login » en cas d'échec.

```php
class Authenticator extends AbstractLoginFormAuthenticator
{
    use TargetPathTrait;

    public const LOGIN_ROUTE = 'app_login';

    private UrlGeneratorInterface $urlGenerator;

    public function __construct(UrlGeneratorInterface $urlGenerator)
    {
        $this->urlGenerator = $urlGenerator;
    }

    public function authenticate(Request $request): PassportInterface
    {
        $username = $request->request->get('username', '');

        $request->getSession()->set(Security::LAST_USERNAME, $username);

        return new Passport(
            new UserBadge($username),

            new PasswordCredentials($request->request->get('password', '')),
            [
                new CsrfTokenBadge('authenticate', $request->get('_csrf_token')),
            ]
        );
    }
```

- La classe « Authenticator » utilise un **trait**. Un trait est une forme d'héritage de méthode sans qu'il soit nécessaire d'« extends » cette classe. Ici, pour la méthode « getTargetPath » dans la fonction « onAuthenticationSuccess ».

- Elle contient une constante qui indique la route « login ».

- « Authenticator » reçoit une **injection de dépendance** de l'objet interface « UrlGeneratorInterface » pour créer les différentes ressources utiles dans la génération de route. Ici, dans la fonction « onAuthenticationSuccess ».

- La fonction « authenticate » prend en paramètre l'objet « Request » et retourne un objet « PassportInterface ».

- Dans l'objet « Request », est récupéré l'identifiant de l'utilisateur, ici, le « username ».

- Dans la session, on stocke l'identifiant.

- Est retourné un « **Passport** », dans lequel il y a :

  - Un « **UserBadge** » qui stocke l'identifiant et qui pourra être utilisé par le « firewall » pour la limitation de connexion du « loggin_trottling ».

  - Un « **PasswordCredentials** » qui stocke le mot de passe.

  - Un « **CsrfTokenBadge** » qui vérifie que le « token » est valide.

Si l'authentification est validée, la fonction « onAuthenticationSuccess » prend en paramètre l'objet « Request » dans lequel sont stockés les éléments de session (« Passport »etc .) et une chaîne de caractères «$firewallName » représentant le nom du firewall qui contient les paramètres édités dans le « security.yaml ».

Puis, dans une exécution de code conditionnelle, il est vérifié que dans la variable « $targetPath », il y a une méthode « getTargetPath » qui contient les éléments de la session et les paramètres du « firewall ».

Puis instancie un objet « RedirectResponse » ayant pour paramètre l'objet « UrlGeneratorInterface » qui appelle la méthode « generate » qui crée une route ayant les paramètres contenus dans la variable « targetPath ».

La fonction « onAuthenticationSuccess » retourne un objet Response qui redirige vers la route « admin ».

CSRF (Cross Site Request Forgery)

Les CSRF sont des requêtes indésirables effectuées à l'insu d'un utilisateur connecté. Dans l'espace d'administration géré par Easy Admin et les formulaires générés par symfony(« login » et « register »), les token CSRF sont vérifiés par défaut dans les formulaires. Ceci prémunit donc de ce type d'attaque.

XSS (Cross Site Scripting)

Une faille XSS consiste à injecter du code que le navigateur pourra lire directement. Les conséquences sont diverses mais on peut noter la redirection sur un autre site qui pourrait être semblable à celui que vous visitez dans le but de récupérer vos identifiants.

Le moteur de templating « Twig » qui est utilisé dans l'application échappe par défaut les caractères spéciaux avec la méthode « htmlspecialchars ».

Injection SQL

Une injection SQL est l'envoi de mots-clefs SQL pour modifier les requêtes envoyées au serveur. Pour parer à cela Symfony via Doctrine utilise des requêtes préparées, c'est à dire que le serveur est averti du type de requête qu'il va recevoir. Cela a pour conséquence que les paramètres de la requête ne peuvent être que des valeurs et non des mots-clefs SQL ou des noms de table.

Situation de travail ayant nécessité une recherche à partir de site anglophone

Afin de restreindre l'accès des pages d'administration aux utilisateurs n'ayant pas le statut adéquat, j'ai utilisé la documentation officielle des bundles de Symfony. Dans la page d'accueil de la documentation du bundle je me suis intéressé à l'onglet « security ». En suivant les instructions données dans cette documentation, j'ai pu sécuriser l'accès aux pages sensibles de l'application. *Voici l'article contenu dans cette page :*

Security

Version: current
Edit this page

- *Logged in User Information*
- *Restrict Access to the Entire Backend*
- *Restrict Access to Menu Items*
- *Restrict Access to Actions*
- *Restrict Access to Fields*
- *Custom Security Voters*

EasyAdmin relies on Symfony Security for everything related to security. That's why before restricting access to some parts of the backend, you need to properly setup security in your Symfony application:

1. *Create users in your application and assign them proper permissions (e.g. `ROLE_ADMIN`);*
2. *Define a firewall that covers the URL of the backend.*

Logged in User Information

When accessing a protected backend, EasyAdmin displays the details of the user who is logged in the application and a menu with some options like "logout". Read the user menu reference for more details.

Restrict Access to the Entire Backend

Using the access_control option, you can tell Symfony to require certain permissions to browse the URL associated to the backend. This is simple to do because each dashboard only uses a single URL

Restrict Access to Menu Items

Use the `setPermission()` method to define the security permission that the user must have in order to see the menu item:

Traduction

Sécurité

Version : actuelle

- Information de l'utilisateur connecté
- Restreindre l'accès à toutes la partie back-end
- Restreindre l'accès aux éléments du Menu
- Restreindre l'accès aux Actions
- Restreindre l'accès aux champs
- Personnaliser les Voters de sécurité

Easy Admin repose sur la Symfony Security pour tout ce qui concerne la sécurité. C'est pourquoi avant de restreindre l'accès à certaines parties du back-end, vous devez configurer convenablement la sécurité de votre application Symfony.

1 Créez des utilisateurs dans votre application et assignez leur des permissions appropriées.

2 Définissez un firewall qui concerne les URL du back-end.

Information de l'utilisateur connecté

Quand l'accès au back-end est protégé, Easy Admin affiche les détails de l'utilisateur loggé et un menu avec quelques options comme « déconnexion ». Lisez le Menu de référence de l'utilisateur pour plus de détails.

Restreindre l'accès à toute la partie back-end

En vous servant de l'« access_control option », vous pouvez dire à Symfony d'exiger certaines permissions pour naviguer sur les URL associés au back-end.

Restreindre l'accès aux éléments du Menu

Utilisez la méthode « setPermission » pour définir l'autorisation de sécurité que l'utilisateur doit avoir pour voir l'élément du menu.

Conclusion

La mise en œuvre de ce projet m'a permis de poursuivre mon apprentissage du développement d'application web et plus particulièrement du framework Symfony. Il est très intéressant par ses nombreuses fonctionnalités et ses possibilités de configuration. Néanmoins, il me sera nécessaire d'approfondir davantage mes connaissances. Je dois également élargir mon champ de compétences en abordant d'autres frameworks ou librairies.

Durant cette période de travail à domicile, j'ai pu éprouver une organisation rigoureuse de ma journée au sein du foyer et pour les étapes de conception de l'application. L'absence d'encadrement s'est fait ressentir pour l'aide à la prise de décision à certains moments de l'élaboration du projet du fait du manque d'expérience dans le domaine.

Les conseils reçus pendant la formation m'ont notamment aidé dans la manière de rechercher des solutions aux problèmes rencontrés. Ils m'ont aussi permis de porter une attention particulière sur le nommage dans les diverses parties du code en tentant d'être suffisamment explicite.

Le cahier des charges a été respecté dans son intégralité et l'application est fonctionnelle. Cependant, j'estime que des améliorations sont à apporter pour que l'application soit à un niveau d'exigence professionnel.