

DÉVELOPPEUR WEB ET WEB MOBILE

DOSSIER DE PROJET : KAYBEAUTY

Sommaire

I.	Remerciements.....	2
II.	Introduction.....	3
III.	Kaybeauty.....	4
1.	Conception de mon projet	4
	Planning.....	4
	Cahier des charges.....	12
	Terminal.....	12
	Mamp.....	12
	MySQL.....	12
	Visual Studio Code.....	13
	Git.....	13
	Backend.....	13
	Frontend.....	13
2.	Développement du projet.....	14
A.	Mise en place.....	14
	Installation de Mamp, Homebrew et de Composer.....	14
	Installation de Symfony.....	15
	Initialisation du dépôt Git.....	15
	Importation de Bootstrap.....	17
	Structure de Symfony.....	17
B.	La base de données.....	18
	Création de la base de données.....	18
	Les Entités.....	19
	Les CRUD.....	21
C.	Les Routes.....	24
D.	Intégration des vues.....	26
	Les vues statiques.....	26
	Les vues dynamiques.....	33
E.	Sécurité.....	36
	Sécurisations des routes et rôles.....	36
	Authentification.....	38
	Encodage du mot de passe.....	39
F.	Contraintes de langages.....	39
IV.	Conclusion.....	41
	La formation.....	41
	Le projet.....	41

I.**Remerciement**

Dans un premier temps, je voudrais remercier Romain Erard, conseiller pédagogique chez NEXTFORMATION, pour m'avoir assistée dans la réalisation de mon dossier « Transitionpro ».

Je voudrais remercier également les intervenants de NEXFORMATION, qui par leurs qualités pédagogiques nous ont transmis leurs connaissances durant ces mois de formations, particulièrement Christian Meneux, qui a su nous expliquer PHP et son Framework Symfony.

Je voudrais aussi remercier Monsieur Patrick Naitali Directeur de la société MANAIS SAS, pour m'avoir accueilli au sein de son entreprise dans le cadre de ma période de stage.

Je voudrais également remercier Raphael alternant dans la société MANAIS SAS, pour avoir partagé ces connaissances, qui m'ont permis d'enrichir mon expérience professionnelle.

Je voudrais également remercier mes proches, qui m'ont soutenus et encouragés dans la réalisation de mon projet.

II.**Introduction**

Après avoir travaillé sept ans en tant que vendeur dans un aéroport, j'ai décidé d'entreprendre une reconversion professionnelle.

Depuis toujours je suis attiré par les jeux vidéo et l'informatique, j'ai cherché un métier pouvant me correspondre dans ces domaines.

J'ai donc commencé par me former sur mon temps libre en suivant des cours de HTML/CSS et PHP sur le site Openclassroom.

Après avoir commencé à me former sur le site Openclassroom, je me suis aperçu que la logique de programmation me séduisait et c'est à partir de ce moment-là que j'ai décidé de me lancer et de faire une reconversion professionnelle dans ce métier.

J'ai déposé ma demande de financement auprès de Transition Pro au mois de septembre 2022, le financement a été accepté en novembre 2022 et j'ai commencé ma formation chez NEXTFORMATION en janvier 2023.

C'est grâce à tout ce que j'ai appris durant cette formation que je peux vous présenter mon projet.

« Kaybeauty » est un site de e-commerce, réalisé en Symfony/Twig avec MVC (Model, Vue, Controller), et qui a pour objectif de vendre des produits pour la peau et des prestations.

III.

Kaybeauty

1. Conception de mon projet

Planning

Afin de travailler dans de meilleures conditions, il était important pour moi de définir un planning de répartition des tâches.

J'ai donc décidé de consacrer soixante pourcents de mon temps au code et quarante pourcents à la rédaction du dossier.

La « deadline » imposée pour rendre le dossier m'a amené à créer un cahier des charges afin de repérer les fonctionnalités prioritaires et de créer l'architecture du site.

Cahier des charges

Description de l'existant :

Durant la conception du site, j'ai pu regarder différents sites sur Internet et le site que j'ai trouvé qui se rapproche le plus de « Kaybeauty » c'est le site l'Atelier du Sourcil :

<https://www.atelierdusourcil.com/>

Public visé :

Le site « Kaybeauty » s'adresse à toutes les personnes qui souhaite acheter des produits pour la peau ou faire des soins pour le visage

En ce qui concerne la région, pour les prestations la boutique se situera à Paris, mais pourrait s'implanter dans différentes villes en France plus tard, quand aura lieu la livraison pour les produits, cela sera limité à la France métropolitaine et au DOM-TOM.

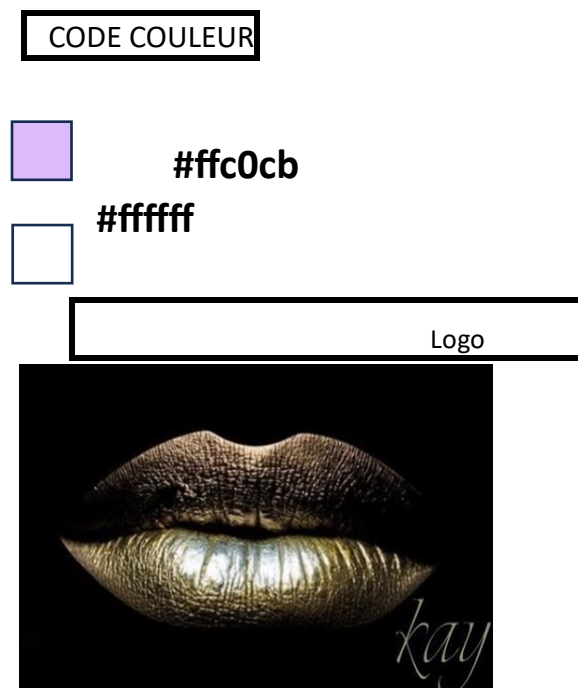
Périmètre du projet :

Le site « Kaybeauty » est conçu en français, et doit permettre à un utilisateur de créer un compte ou l'utilisateur devra renseigner son nom, son prénom et son email pour pouvoir consulter les produits ou les prestations qui sont sur le site, et de pouvoir si c'est un produit consulter sa fiche produit ou de l'ajouter à un panier, si c'est une prestation il pourra soit consulter la prestation ou prendre une réservation.

Si l'utilisateur a un panier et qu'il souhaite le valider il devra renseigner en plus son adresse et son numéro de téléphone.

J'ai également réfléchi à d'autres moyens de paiement (exemple : Apple Pay, PayPal, etc...), je suis également en train de réfléchir à un moyen de pouvoir sélectionner un Point Relais si le client ne désire pas être livré chez lui.

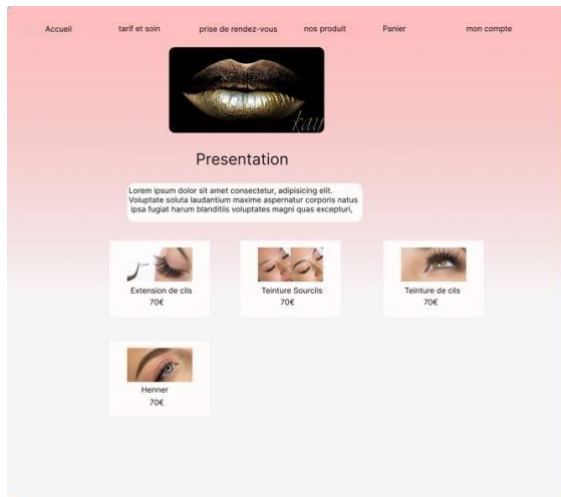
Charte graphique :



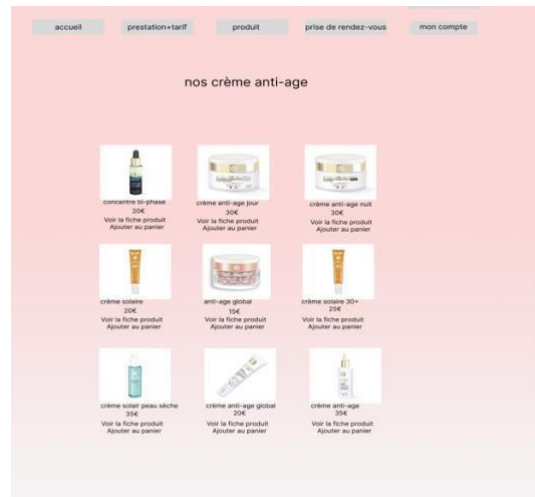
Maquette

Les maquettes du site ont été réalisé avec le logiciel « Figma »

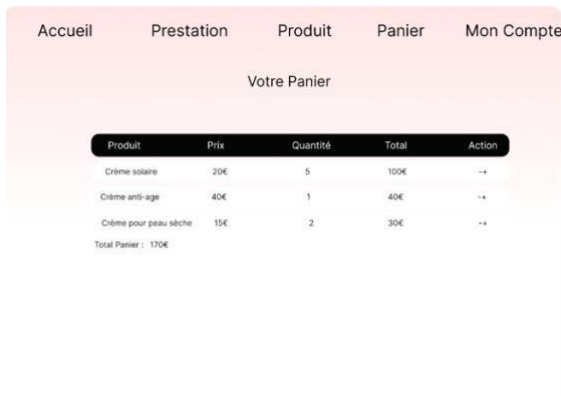
Page d'accueil



Page produits



Page panier



Page profil



Page gestion

Gestion des Prestation						
ID	Nom	Prix(€)	Durée	Description	Modifier	Supprimer
1	micro blading	100€	30min	Le microblading est une technique de pigmentation ancestrale originaire d'Asie qui consiste à imiter les poils des sourcils grâce à des pigments placés sous les couches superficielles de la peau. Le trait, ultra-fin, permet de combler discrètement les sourcils clairsemés.	Modifier	Supprimer
2	brow lift	100€	2h	Cette nouvelle technique est une méthode nettement moins invasive que le microblading. En effet, il ne s'agit pas de maquillage permanent mais d'une technique pour révéler tout le potentiel des sourcils.	Modifier	Supprimer
3	henné	200€	30min	Cette technique remplace le tracé quotidien du crayon à sourcils, grâce à sa tenue dans le temps, de l'ordre de 2 à 3 semaines selon le type de peau. Le henné végétal est une "technique nouvelle génération", qui donne un rendu très naturel avec des nuances de couleurs allant du blond foncé au châtain foncé.	Modifier	Supprimer
4	Rehaussement de Cils	50€	3h	Rehaussement de cils	Modifier	Supprimer
5	Épilation de Restructuration	90€	1h	Épilation de restructuration	Modifier	Supprimer
6	Teinture des cils	20€	2h	Teinture de cils	Modifier	Supprimer
7	Teinture des sourcils	80€	40min	Teinture des sourcils	Modifier	Supprimer
8	Extension de cils	70€	2h	Extension de cils	Modifier	Supprimer

Description fonctionnelle et technique :

Description du site

L'utilisateur aura la possibilité de créer un compte, il aura aussi la possibilité de consulter les produits et les prestations et si l'utilisateur le souhaite il pourra ajouter des produits dans son panier ou de prendre rendez-vous pour une prestation.

Une fois que l'utilisateur aura créé son compte il pourra gérer ces informations personnelles, consulter les commandes qu'il aura passé et consulter les prestations qu'il aura réserver.

L'administrateur pourra quand à lui ajouter, modifier, supprimer des produits.

Ces produits étant classifié par catégories, l'administrateur pourra soit ajouter des catégories ou les supprimer. Il aura accès aussi à l'onglet « Gestion du logo » ou il pourra changer le logo du site

L'administrateur aura aussi accès aux commandes et à leur informations (adresse, produit, etc...) qui ont été faite sur le site et aussi aux réservations et à leur information (l'utilisateur, l'horaire, la date, le nom de la prestation).

Spécifications fonctionnelles

Page d'accueil : La page sur laquelle sera dirigé l'utilisateur en arrivant sur le site. On pourra y découvrir différents éléments tels que, un texte de présentation de du site, le logo du site, les prestations qui seront proposer ainsi qu'une barre de navigation.

Page d'inscription : L'utilisateur aura accès au formulaire pour créer son compte, il devra renseigner son nom, son prénom, son email et son mot de passe.

Page de connexion : La page pour se connecter ou figurera un formulaire et on lui demandera de renseigner son email et son mot de passe.

Page de gestion du compte : sur cette page il y aura 3 onglets.

« Connexion et sécurité » dans cette onglet on retrouvera les informations de l'utilisateur (nom, prénom, email) et ou il pourra les modifier.

« Commande Passé » dans cette onglet l'utilisateur pourra retrouver toutes les commandes qu'il aura passé.

« Mes Réservations » dans cette onglet l'utilisateur pourra retrouver toutes les réservations que l'utilisateur aura passer.

Page des produits : une page ou sera afficher une liste de produits avec leur nom, et le prix et une image, ainsi que deux liens.

« Voir la fiche produit » se lien redirigera vers une page ou sera afficher l'image du produit, son nom, son prix, sa description et aussi les commentaires sur le produit sur cette page l'utilisateur pourra aussi ajouter sont commentaire s'il est connecté.

« Ajouter au panier » qui permettra d'ajouté le produit au panier de l'utilisateur.

Étant donné que les produit sont classifié par catégories il existera aussi une page par catégorie de produits.

Page panier : la page représentant le panier de l'utilisateur. Le panier sera sous forme de tableau qui listera les produits présent dans le panier ainsi que leur prix, leur quantités et le prix total du panier, il y aura aussi des bouton « Ajouter » « Retirer » qui permettront a l'utilisateur d'augmenter la quantité d'un produit ou de la réduire. Il y aura aussi un bouton pour valider le panier.

Page adresse : cette page permettra a l'utilisateur de renseigner sont adresse, et son numéro de téléphone.

Page de paiement : la page pou procéder au paiement de la commande.

Description technique

Le site doit être compatible avec Google Chrome, Safari et Mozilla Firefox.

Bien que le site ne soit pas mis en ligne tous de suite après sa conception, j'ai déjà fait mon choix quand a l'hébergeur, mon choix pour l'hébergement c'est porté sur « o2switch ».

Le fait que l'hébergeur soit Français, qu'il propose un certificat « SSL » et une protection « Anti-DDoS » mon conforté dans mon choix

Diagramme d'utilisation

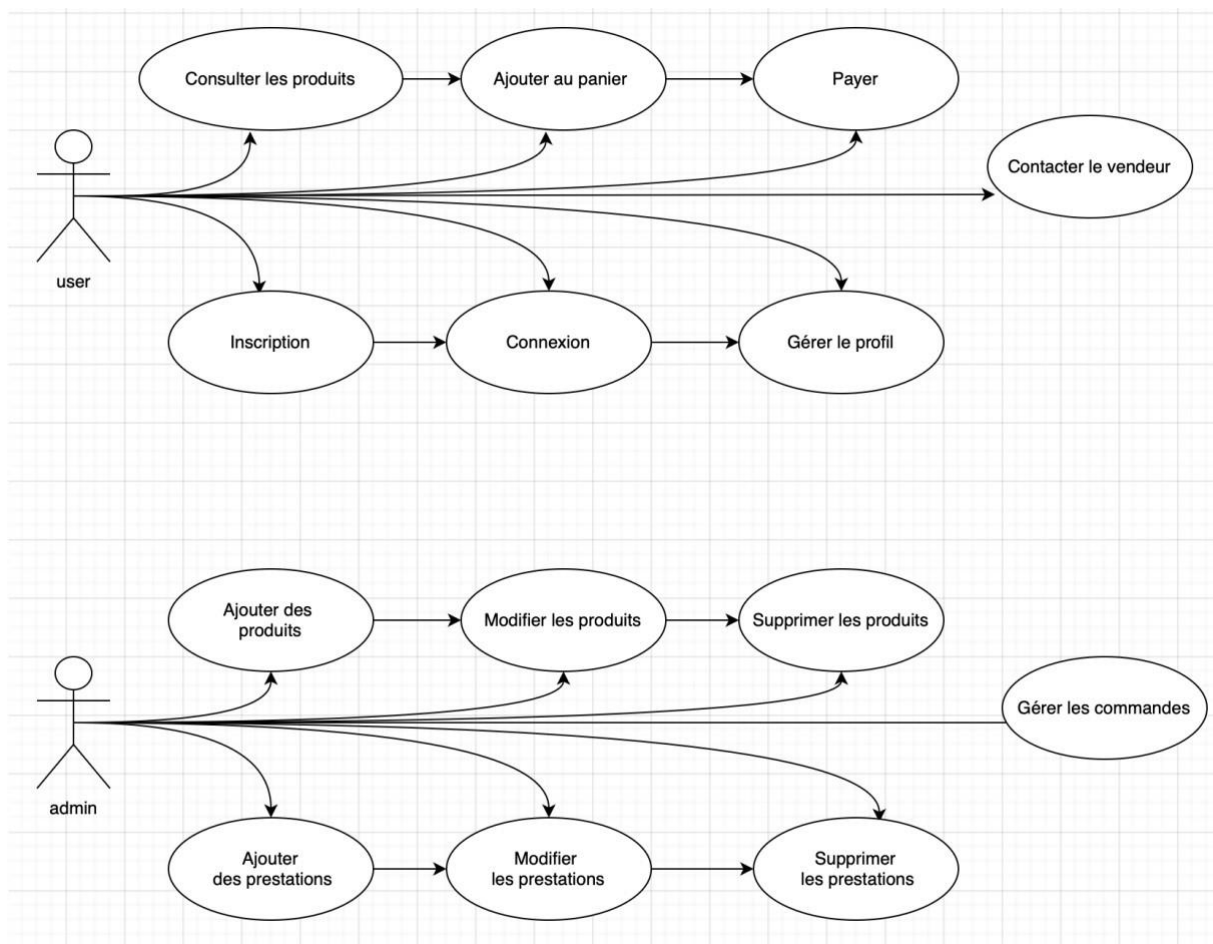
Le diagramme « UML » (cf. annexe 1) de « Kaybeauty » a été réalisé sur <https://app.iagrams.net/>.

« UML » signifie Langage de Modélisation Unifié, cela permet de visualiser la conception d'un site à l'aide de pictogramme.

De ce fait nous pouvons voir les parties du site accessible en fonction du rôle de l'utilisateur.

Éléments du diagramme d'utilisation :

- L'acteur : un acteur représente un rôle de l'utilisateur qui interagit avec le système que l'on modélise.
- Le cas d'utilisation : un cas d'utilisation décrit une fonction qu'un système exécute pour atteindre l'objectif de l'utilisateur.
- La relation : une relation est une connexion entre des éléments de modèle.



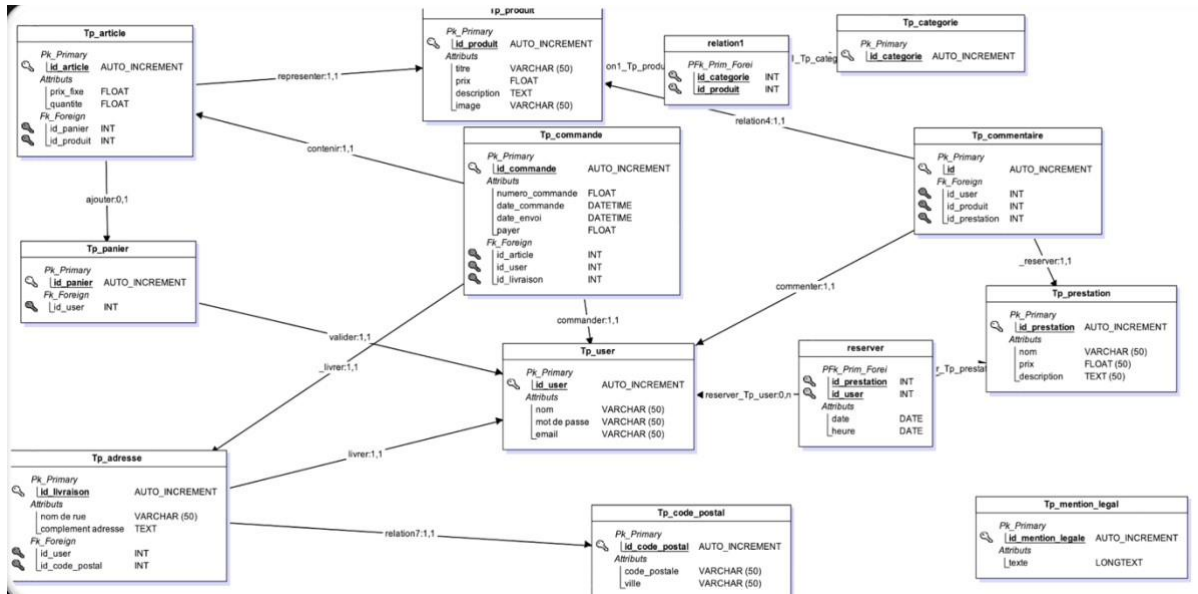
Annexe 1 – Diagramme d'utilisation

Modèle de données

Modèle Conceptuel de données :

Le « MCD » ou « Modèle Conceptuel de données » (cf. annexe 2) est une représentation graphique tirée de la méthode « Merise » qui permet de comprendre facilement comment les différentes tables d'une base de données sont liées entre elles à l'aide du code suivant.

MLD



Annexe 3 - MLD

Environnement technique



L'invite de commande que j'ai utilisée est « Terminal », « Terminal » donne aux utilisateur un accès à la commande « UNIX » (Les commandes « UNIX » sont des commandes que l'on peut utiliser dans une interface de terminal pour interagir avec un système d'exploitation de UNIX ou UNIX-like (comme ios)), car il offre différente fonction intéressante comme par exemple :

- **Intégration d'autre outils** : « Terminal » peut être utilisé pour interagir avec d'autre outils ou applications.
- **Support des scripts** : Il permet aussi d'exécuter des script « Shell » pour automatiser des tâches.
- **Support de plusieurs Shell** : Bien que « Terminal » ait déjà un Shell (zsh), il peut en supporter d'autre comme « fish » ou « tcsh ».

Mamp

« Mamp » est un environnement qui permet de faire fonctionner en local des script PHP et ainsi d'afficher un site web

MySQL

« MySQL » est le système de gestion de base de données utilisé pour ce projet. Ce système est très répandu dans le développement web et de nombreuse documentation sont disponible sur Internet. Il permet entre autres de mettre en place des relations entres les tables et fonctionne parfaitement avec les outils techniques utilisé pour ce projet.

Visual Studio Code

« Visual Studio Code » est l'éditeur de code utilisé pour se projet. C'est un éditeur de code gratuit, puissant et il est possible d'y ajouter de nombreuse extension qui permette de code plus rapidement et qui rendent le code plus lisible.

Git

L'utilisation d'un dépôt « Git » est très intéressante lors de la création d'un projet. Il permet à la fois de créer une sauvegarde de son projet et de développer de nouvelle fonctionnalité sur différentes (versions) pour pouvoir ensuite les ajouter à la version principale.

Backend

Pour réaliser ce projet j'ai décidé d'utiliser le « Framework » Symfony. Car c'est un framework très efficace, qui permet de développer une application rapidement notamment avec son système de packages PHP que l'on peut installer via la commande « Composer » dans « Terminal ».

Grâce à la dépendance Doctrine, la gestion de la base de données et la création d'un « CRUD », sont très simplifiées, mais reste tout fois très modulable notamment via les annotations.

Le système de sécurisation des routes permet d'avoir un accès administrateur protégé.

De plus le modèle « MVC » (Model View Controller) utilisé par Symfony permet de développer un projet clair et bien structuré.

Frontend

Pour l'affichage, j'ai utilisé Twig, le moteur de « Template » de Symfony. Car grâce à sa syntaxe il permet grâce aux `{{ }}` de faire appel à des variable PHP, il permet aussi d'écrire des commentaire grâce `{##}` et aussi d'écrire des commandes avec `{% %}`.

En se qui concerne la gestion du style, je me suis tourné vers le « Framework » CSS « Bootstrap », car sont grand nombre de classe m'a permis de développer la mise en place et le style de mes éléments plus rapidement.

Cependant, j'ai créé une fiche de style pour ajouter les couleurs d a charte graphique.

2. Développement du projet

A. Mise en place

Installation de Mamp, Homebrew et Composer

Avant de pouvoir commencer à développer mon projet, il fallait créer un environnement de travail sur ma machine.

Pour faire ça j'ai dû commencer par installer « Mamp » sur ma machine en allant sur le lien ci-dessous :

<https://www.mamp.info/en/mamp/mac/> .

Ensuite j'ai dû installer « Homebrew » sur ma machine en allant sur le lien ci-dessous :

https://brew.sh/index_fr .

Et pour finir, j'ai installé « Composer » sur ma machine, avec la commande « brew install composer ».

Installation de Symfony

J'ai ensuite installé le client Symfony qui me donnait accès aux commandes « Symfony » dans la console.

J'ai ainsi pu lancer la commande « symfony new Kaybeauty-full » qui permet de créer un projet Symfony.

Le « --full » permet d'importer les « bundles » les plus utiles de Symfony tels que Doctrine et Twig.

Le client de Symfony permet aussi de lancer la commande « symfony server :start » qui démarre le serveur

Initialisation du dépôt Git

Pour sauvegarder mon projet et avoir accès à une plateforme de « versioning », j'ai créé un compte chez « GitHub » sur lequel j'ai créé un dépôt « Kaybeauty » pour mon projet (cf. annexe 4).

J'ai ensuite converti mon projet en dépôt « Git » via la commande « git init ».


J'ai ensuite indexé les ajouts et les modifications qui ont été faites dans les fichiers du répertoire de travail via la commande « git add » et préparé ainsi le premier « commit » que j'ai exécuté via la commande « git commit -m "premier message" »

J'ai ensuite relié ce dépôt au dépôt distant de « GitHub » via la commande « git remote add origin <https://github.com/Sebastien0506/kayBeauty.git> ».

Il me suffisait ensuite d'envoyer les fichiers du dépôt local au dépôt distant via la commande « git push -u origin main ».

À partir de ce moment, j'exécutais la suite de commande « git add . , git commit -m "mon message", git push » pour envoyer les modifications vers le dépôt distant. Pour le « versionning », la commande « git branch essais » à créer une branche nommée « essais » sur lequel je pouvais enregistrer les nouvelles fonctionnalités en gardant la branche main inchangée.

Une fois que les nouvelles fonctionnalités étaient abouties, j'exécutais la commande « git merge » pour les ajouter à la branche main.

 Sebastien0506 Merge branch 'main' of https://github.com/Sebastien0506/kayBeauty 802e5f1 3 hours ago 🕒 4 commits		
bin	Ajout de plusieurs fonctionnalités	4 hours ago
config	Ajout de plusieurs fonctionnalités	4 hours ago
migrations	Ajout de plusieurs fonctionnalités	4 hours ago
public	Ajout de plusieurs fonctionnalités	4 hours ago
src	Ajout de plusieurs fonctionnalités	4 hours ago
templates	Ajout de plusieurs fonctionnalités	4 hours ago
tests	Ajout de plusieurs fonctionnalités	4 hours ago
translations	Ajout de plusieurs fonctionnalités	4 hours ago
.DS_Store	Ajout de plusieurs fonctionnalités	4 hours ago
.env	Ajout de plusieurs fonctionnalités	4 hours ago
.env.test	Ajout de plusieurs fonctionnalités	4 hours ago
.gitignore	Ajout de plusieurs fonctionnalités	4 hours ago
README.md	Update README.md	3 months ago
base de données kaybeauty	Ajout de plusieurs fonctionnalités	4 hours ago
composer.json	Ajout de plusieurs fonctionnalités	4 hours ago
composer.lock	Ajout de plusieurs fonctionnalités	4 hours ago
phpunit.xml.dist	Ajout de plusieurs fonctionnalités	4 hours ago
symfony.lock	Ajout de plusieurs fonctionnalités	4 hours ago

Annexe 4 – Dépôt Git

Importation de Bootstrap

Avant de pouvoir utiliser « Bootstrap » j'ai dû intégrer son « CDN » (cf. annexe 5) (Content Delivery Network) disponible sur le site <https://getbootstrap.com> à mon fichier « base.html.twig » qui est situé dans mon projet. Cela permet d'injecter le contenu de la bibliothèque sans avoir à la télécharger.


```

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{% block title %}Kay Beauty{% endblock %}</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css" rel="stylesheet"
  integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJ0Z" crossorigin="anonymous">
  <link rel="stylesheet" href="{{ asset('css/style.css') }}">
  {# Run `composer require symfony/webpack-encore-bundle` to start using Symfony UX #}
  {% block stylesheets %}
  {% endblock %}
</head>
<body>
<header>
  {% include '_nav.html.twig' %}
</header>
<main class="col-md-10 mx-auto">
  <h1 class="display:flex;justify-content:center;">{% block h1 %}{% endblock %}</h1>
  {% block body %}{% endblock %}
</main>
<footer></footer>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"
  integrity="sha384-w76AqPfDkMBDXo30jS15geZ6pr3x5MlQ1ZAGC+nuZB+EYdgRZGiwxhTBTkF7CXvN" crossorigin="anonymous"></script>
<script src="{{ asset('js/script.js') }}"></script>
{% block javascripts %}{% endblock %}
</body>

```

Annexe 5 - CDN

Structure de Symfony

Un projet Symfony est structuré en différents dossiers :

- **Bin** : ce dossier contient les fichiers de commande qu'on va utiliser tout au long du projet notamment pour modifier la base de données, nettoyer le cache ou encore lancer le serveur.
- **Config** : contient les différents packages installés, au cours de la conception du projet, comme par exemple les routes et le système de sécurité. Les fichiers sont de base en YALM ou en PHP. Ces fichiers sont personnalisables afin de s'adapter aux exigences du projet.
- **Migration** : contient les fichiers de migration créé par Doctrine.
- **Public** : on y retrouve les fichiers du projet accessibles par tous, tel que les images, les fiche de styles et les différentes pages du site.
- **Src** : C'est ici que le développeur intègre sa logique d'application, et ce dossier est structuré de manière précise. Le « **Controller** » va lire les informations d'une requête et va renvoyer une réponse, cette réponse peut être une page html, un objet.... L'« **Entity** » est le fichier qui va représenter une table dans la base de données avec ses propriétés. Avec l'« **Entity** » il y a toujours un « **Repository** », le « **Repository** »

permet de créer les requêtes vers l'entité et donc vers les tables de la base de données. « Src » contient aussi un sous-dossier « **Form** » qui contient les fichiers « **FormType** » du projet.

- **Templates** : c'est ici que seront regroupées les différentes vues du projet, elles sont au format Twig.
- **Test** : c'est ici que se font les tests unitaires.
- **Translation** : on peut y mettre des fichiers de traduction généralement au format XML.
- **Var** : on y trouve les fichiers « cache » et les « log ».

B. La base de données

Création de la base de données

Pour créer la base de données de mon projet, j'ai modifié le fichier « .env » qui est fourni par Symfony afin que le « bundle » Doctrine puisse communiquer avec la base de MySQL.

Pour faire ça, j'ai décommenté la ligne correspondant aux bases MySQL en indiquant le nom d'utilisateur de « phpMyAdmin », le mot de passe, et le nom que je souhaite donner à ma base de données.

Ensuite, j'ai fait la commande « php bin/console doctrine:database:create », ce qui crée la base de données dans « phpMyAdmin ».

Les Entités

Le bundle « security-bundle » donne accès à la ligne de commande « php bin/console make:user » avec laquelle on peut créer un utilisateur en indiquant le nom de l'entité, si l'on souhaite enregistrer l'utilisateur dans la base de données, le champ servira d'identifiant de connexion et si l'on souhaite utiliser le système de cryptage de Doctrine.

Doctrine nous permet, grâce à la ligne de commande « `php bin/console make:entity` » de créer rapidement de nouvelles entités. Une fois que cette ligne de commande sera faite et validée, Symfony nous demandera le nom de la nouvelle propriété qui sera dans l'entité, ainsi que son type (String, Integer, Relation...) (cf. annexe 6).

```
● sebastien@MacBook-Pro-de-sebastien Kaybeauty % php bin/console make:entity

Class name of the entity to create or update (e.g. AgreeableChef):
> Boutique

created: src/Entity/Boutique.php
created: src/Repository/BoutiqueRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.

New property name (press <return> to stop adding fields):
> nom

Field type (enter ? to see all types) [string]:
>

Field length [255]:
>

Can this field be null in the database (nullable) (yes/no) [no]:
>

updated: src/Entity/Boutique.php

Add another property? Enter the property name (or press <return> to stop adding fields):
> adresse

Field type (enter ? to see all types) [string]:
>

Field length [255]:
>

Can this field be null in the database (nullable) (yes/no) [no]:
>

updated: src/Entity/Boutique.php
```

Annexe 6 – Make Boutique

Dans cet exemple (cf. annexe 7) j'ai créé une relation « ManyToMany » entre les entités « Produit » et « Catégorie » car un produit peut avoir plusieurs catégories et une catégorie peut avoir plusieurs produits.

```
#[ORM\ManyToMany(targetEntity: Catégorie::class, inversedBy: 'produits')]
private Collection $categorie;
```

Annexe 7 - ManyToMany

Pour les relations « OneToMany » et « ManyToMany », les « Getter » et « Setter » sont remplacés par une collection (cf. annexe 8).

```

/**
 * @return Collection<int, Categorie>
 */
public function getCategorie(): Collection
{
    return $this->categorie;
}

public function addCategorie(Categorie $categorie): self
{
    if (!$this->categorie->contains($categorie)) {
        $this->categorie->add($categorie);
    }

    return $this;
}

```

Annexe 8 – Entity Catégorie

Une fois toutes les entités créées, on peut exécuter la commande « php bin/console make:migration », qui sert à créer des fichiers de migration dans le dossier « migrations » et ensuite on peut exécuter la deuxième commande « php bin/console doctrine:migrations:migrate » (on peut aussi l'écrire « php bin/console d:m:m ») qui sert à créer les tables correspondantes dans la base de données.

Le CRUD

Le « CRUD » (Create, Read, Update, Delete) est une notion fondamentale du développement web. En effet ce sont les quatre fonctions de base pour gérer les données.

Pour faire un « CRUD » Symfony nous permet via Doctrine d'exécuter la commande « php bin/console make:crud 'Nom de l'entité' », grâce à cette ligne de commande Symfony va créer un « Controller », un « FormType », et les « Templates » des pages index, new, show et update ainsi que les « includes » du formulaire et du bouton supprimer.

Il est tout à fait possible de modifier ces fichiers afin de personnaliser ce CRUD, suivant les besoins du projet.

Par exemple pour ce projet, j'ai voulu créer un formulaire imbriqué afin que l'administrateur puisse ajouter une image représentant la prestation directement dans le formulaire de créations ou de modifications des prestations.

Pour faire ça, je me suis servi du bundle « VichUploader » que l'on peut installer grâce à la commande « `composer require vich/uploader-bundle` ».

J'ai ensuite configuré « VichUploader », pour lui dire où seront stocker les images qui seront ajoutées à la prestation, pour faire ça je suis allé dans le dossier « Config », ensuite dans le dossier « packages », et dans le fichier « `vich_uploader.yaml` », dans ce fichier je lui est rajouté tous les paramètres dont « VichUploader » aura besoin pour savoir où stocker les images liées aux prestations (cf. annexe 9).

```
image_prestation:      You, il y a 24 heures • Ajoute de plusieurs fonctionnalités
    uri_prefix: /image/prestation
    upload_destination: '%kernel.project_dir%/public/image/prestation'
    namer: Vich\UploaderBundle\Naming\SmartUniqueNamer
```

Annexe 9 – Config VichUploader

Ensuite j'ai modifié le fichier « `Prestation.php` » (cf. annexe 10, 11, 12), qui se situe dans le dossier « Entity », dans ce fichier j'ai rajouté tous les paramètres dont « VichUploader » aura besoin pour fonctionner.

```
use Vich\UploaderBundle\Mapping\Annotation as Vich;
use Symfony\Component\Validator\Constraints as Assert;

You, il y a 3 minutes | 1 author (You)
#[ORM\Entity(repositoryClass: PrestationRepository::class)]
#[Vich\Uploadable]
```

Annexe 10 – Config VichUploader Prestation.php

```
#[Vich\UploadableField(mapping: 'image_prestation', fileNameProperty: 'imageName')]
#[Assert\Image(
    maxSize: '25K',
    maxSizeMessage: "L'image est trop volumineuse ({{ size }} {{ suffix }}).",
    La taille maximum autorisée est de {{ limit }} {{ suffix }}",
    mimeTypes: ["image/jpeg", "image/png", "image/jpg"],
    mimeTypesMessage: "Le format de l'image est invalide. Seul les formats JPEG, PNG et JPG sont acceptés."
)]
private ?File $imageFile = null;

#[ORM\Column(nullable: true)]
private ?string $imageName = null;

#[ORM\Column(nullable: true)]
private ?DateTimeImmutable $updatedAt = null;
```

Annexe 11 – Config VichUploader Prestation.php

```

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;
}

public function setImageName(?string $imageName): void
{
    $this->imageName = $imageName;
}

public function getImageName(): ?string
{
    return $this->imageName;
}

```

Annexe 12 – Config VichUploader Prestation.php

Ensuite, j'ai modifié le fichier « PrestationType » (cf. annexe 13) présent dans le dossier 'src/Form' en ajoutant le champ « add('imageFile', VichImageType::class) »

```

->add('imageFile', VichImageType::class, [
    'label' => 'Image de la prestation',
    'label_attr' => [
        'class' => 'form-labe mt4'
    ],
    'constraints' => [
        new NotBlank([
            'message' => 'Veuillez selectionnez une image'
        ])
    ]
])

```

Annexe 13 – VichUploader PrestationType

J'ai ensuite édité le fichier « PrestationController » (cf. annexe 14) présent dans le dossier « src/Controller », afin de créer le code qui permettra l'ajout de la prestation.

```
/**
 * @Route("/ajouter", name="ajouter_prestation")
 */
public function prestation_ajouter(Request $request, PrestationRepository $prestationRepository):Response
{
    $prestation = new Prestation();

    $form = $this->createForm(PrestationType::class, $prestation);

    $form->handleRequest($request);

    if($form->isSubmitted() && $form->isValid())
    {
        $dureMinutes = $prestation->getDureMinutes();
        $dureHeures = $dureMinutes / 60;
        $prestation->setDureHeure($dureHeures);

        $prestationRepository->save($prestation, true);

        $this->addFlash('succes', 'La prestation a bine été ajouté');

        return $this->redirectToRoute('ajouter_prestation');
    }
    return $this->render('prestation/ajouter_prestation.html.twig', [
        'formPrestation' => $form->createView()
    ]);
}
```

Annexe 14 - PrestationController

Dans ce code on ne voit pas l'ajout de l'image lier a la prestation, car « VichUploader » gère automatiquement le téléchargement de l'image et met a jour le champ de l'entité correspondant.

J'ai ensuite, éditer le fichier « FrontController » (cf. annexe 15) présent dans le dossier « src/Controller », afin de créer le code qui permettra de recuperer toutes les prestations, ainsi que leur image et de les donner a ma vue grace au code situer dans l'encadrer rouge .

```
/**      You, il y a 2 minutes • Uncommitted changes
 * @Route("prestation", name="prestation")
 */
public function prestation(PrestationRepository $prestationRepository):Response
{
    $prestation = $prestationRepository->findAll();

    return $this->render('front/prestation.html.twig', ['prestation' => $prestation]);
}
```

Annexe 15 - FrontController

Ensuite, je suis allé dans le dossier « Template », ensuite dans le dossier « front » que j'avais créé, et dans se dossier j'ai créé le fichier « prestation.html.twig » dans lequel j'ai créé le code qui me permettra d'afficher toutes les prestations, ainsi que leur image que j'aurais récupérer grâce au code situer dans l'encadrer rouge (cf. annexe 16).

```

{% for prestation in prestation %}
<div class="col-6 col-md-4 mt-5">
    
    <div class="text-center">
        {{prestation.nom|capitalize}}
        <h3>{{prestation.prix}}€</h3>
        You, avant-hier • Ajoute de plusieurs fonctionnalite
        {% if prestation.dureHeure > 0 %}
            {% if prestation.dureMinutes > 0 %}
                <p>{{ prestation.dureHeure}}h{{prestation.dureMinutes}}min</p>
            {% else %}
                <p>{{ prestation.dureHeure }}h</p>
            {% endif %}
        {% else %}
            <p>{{ prestation.dureMinutes }}min</p>
        {% endif %}
        <a href="{{ path('fiche_prestation', {'id' : prestation.id}) }}">Voir la fiche de la prestation</a><br>
        <a href="{{ path('prestation_reservation', {'id' : prestation.id}) }}">Prendre rendez-vous</a>
    </div>
</div>

```

Annexe 16 – Prestation.html.twig

C. Route

Les routes sont les chemins URL d'un site web. Pour mon projet, c'est dans « Controller » que j'ai déclaré ces routes grâce au bundle « annotations » installé automatiquement.

Ces annotations peuvent se placer avant la déclaration de la classe et s'applique ainsi à toutes les méthodes de la classe, ou les annotations peuvent être placer avant une méthode et ne s'appliquer qu'à cette méthode.

Toutes les méthodes de la classe « ProfilController » hériteront du préfix '/profil' via l'annotation "@Route('/profil')" (cf. annexe 17).

La méthode « mon_compte », ayant pour préfix '/profil' et pour annotations "@Route('/mon_compte')", sera donc déclenchée lorsque l'on ira sur l'URL 'https://Kaybeauty/profil/mon_compte'.


```

/**
 * @Route("/profil")
 */
class ProfilController extends AbstractController
{
    /**
     * @Route("/mon_compte", name="mon_compte")
     */
    public function mon_compte()
    {
        return $this->render('profil/mon_compte.html.twig');
    }

    /**
     * @Route("/parametre", name="connexion_securite")
     */
    public function connexion_securite():Response
    {
        $user = $this->getUser();
        // You, hier • Ajoue de plusieurs fonctionnalite ...
        return $this->render('profil/connexion_securite.html.twig', [
            'user' =>$user,
        ]);
    }
}

```

Annexe 17 - ProfilController

Il faudra ensuite donner un nom à la route (sur l'annexe 17, le nom de la route « mon_compte »).

Ce nom est ensuite utilisé dans le fichier Twig pour hydrater les « href » (cf. annexe 18).

```

<a class="navbar-band" href="{{ path('mon_compte') }}">Mon Compte </a>

```

Annexe 18 – Path

D. Intégrations des vues

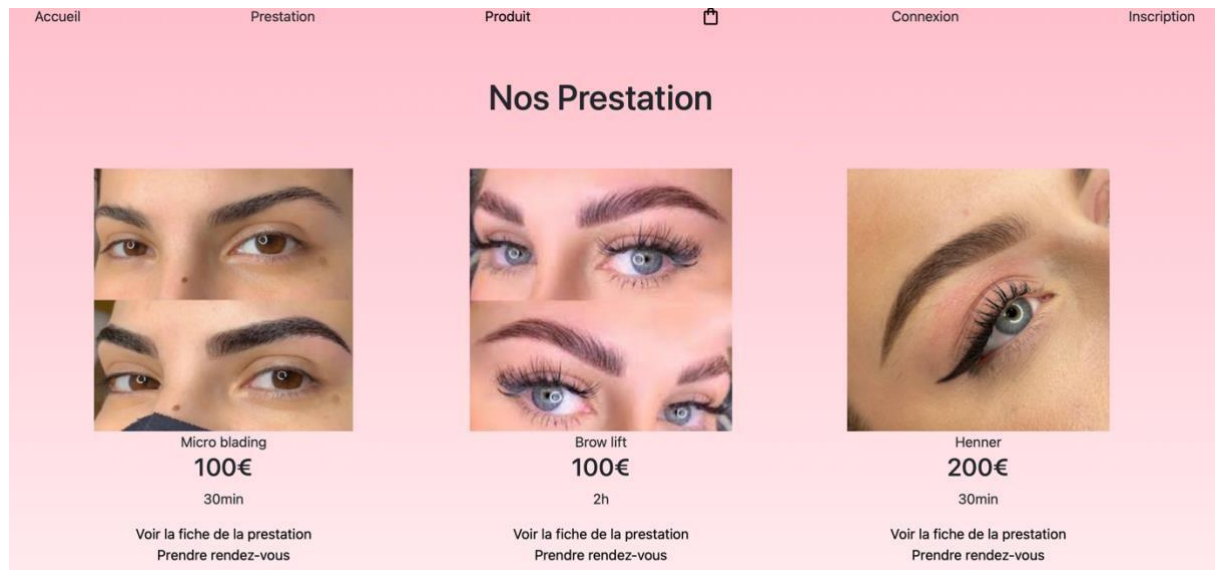
Il peut exister deux types de vue sur un site web : les vues statiques et les vues dynamiques.

La page statique est la même pour tous les utilisateurs.

La page dynamique, est quand t'a-elle différente en fonction de l'utilisateur connecté.

Les vues statiques

Pour donner un exemple d'une vue statique je vais présenter la page Prestation de mon projet (cf. annexes 19 et 20).



Annexe 19 – Page prestation – Version Desktop



Nos Prestation



Micro blading

100€

30min

[Voir la fiche de la
prestation](#)

[Prendre rendez-vous](#)



Brow lift

100€

2h

[Voir la fiche de la
prestation](#)

[Prendre rendez-vous](#)

Annexe 20 – Page prestation – Version Mobile

Elle est composée comme tout les pages du site, d'un menu de navigation « Header » et d'un pied de page « footer ».

À cela s'ajoute les prestations proposer en affichant leur nom, leur prix et leur durée, ainsi que deux lien pour soit avoir plus d'information sur la prestation ou pour prendre directement rendez-vous.

Sur la ligne 1, on retrouve le code `{% extends 'base.html.twig' %}`, qui permet d'inclure dans la page 'prestation.html.twig' (cf. annexe 21), le fichier 'base.html.twig' (cf. annexe 22)

```

1  {% extends 'base.html.twig' %}
2
3  {% block h1 %}Nos Prestation{% endblock %}
4
5  {% block body %}
6  <div class="container">
7      <div class="row">
8          {% for prestation in prestation %}
9              <div class="col-6 col-md-4 mt-5">
10                 
11                 <div class="text-center">
12                     {{prestation.nom|capitalize}}
13
14                     <h3>{{prestation.prix}}€</h3>
15
16                     {% if prestation.dureHeure > 0 %}
17                         {% if prestation.dureMinutes > 0 %}
18                             <p>{{ prestation.dureHeure }}h{{prestation.dureMinutes}}min</p>
19                         {% else %}
20                             <p>{{ prestation.dureHeure }}h</p>
21                         {% endif %}
22                     {% else %}
23                         <p>{{ prestation.dureMinutes }}min</p>
24                     {% endif %}
25                     <a href="{{ path('fiche_prestation', {'id' : prestation.id}) }}">Voir la fiche de la prestation</a>
26                     <a href="{{ path('prestation_reservation', {'id' : prestation.id}) }}">Prendre rendez-vous</a>
27                 </div>
28             </div>
29         {% endfor %}
30     </div>
31 </div>
32 {% endblock %}

```

Annexe 21 – prestation.html.twig

```

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Kay Beauty{% endblock %}</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJ0Z" crossorigin="anonymous">
    <link rel="stylesheet" href="{{ asset('css/style.css') }}">
    {# Run `composer require symfony/webpack-encore-bundle` to start using Symfony UX #}
    {% block stylesheets %}
    {% endblock %}
</head>
<body>
<header>
    {% include '_nav.html.twig' %}
</header>
<main class="col-md-10 mx-auto">
    <h1 class="display:flex;justify-content:center;">{% block h1 %}{% endblock %}</h1>
    {% block body %}{% endblock %}
</main>
<footer></footer>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-w76AqPFDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB+EYdgRZgiwkhTBTkF7CXvN" crossorigin="anonymous"></script>
<script src="{{ asset('js/script.js') }}"></script>
{% block javascripts %}{% endblock %}
</body>

```

Annexe 22 – base.html.twig

Twig fonctionne avec des blocks. Ces blocks sont définis dans un fichier parent, et peuvent ensuite être réutilisés ou redéfinis dans tous les fichiers enfants. Prenons, par exemple, le block `{% block javascript %}`. Celui-ci est déclaré dans le fichier 'base.html.twig' (cf. annexe 22). Lorsqu'un fichier, comme 'prestation.html.twig' (cf. annexe 21), étend 'base.html.twig', il

peut redéfinir le contenu de ce bloc pour y insérer du Javascript spécifique ou d'autre éléments. Pour indiquer la fin du contenu personnalisé, nous utilisons '{% endblock %}'.

Dans le cadre de mon projet, j'ai voulu intégrer des données du serveur dans du JavaScript avec Twig

Voici comment j'ai procédé :

J'ai commencé par définir une route nommée « prestation_reservation » qui prend un ID de prestation en paramètre (cf. annexe 23).

Ensuite grâce à la méthode 'find' du 'PrestationRepository', j'ai récupéré la prestation correspondant à l'ID fournit.

Une fois la prestation obtenue, j'ai extrait sa durée en heures et en minutes. J'ai ensuite vérifié si cette durée en heures était supérieure à zéro. Si c'était le cas, je converti ces heures en minutes et je les ajoute à la durée totale déjà exprimée en minutes.

Avec les données récupérées, j'ai construit un tableau nommé 'event'. Ce tableau encapsule les informations essentielles de la prestation, notamment sont identifiant, son nom, et sa durée total en minutes.

J'ai ensuite stocké ce tableau 'event' dans la session utilisateur. Cela me permet d'utiliser ces informations dans d'autre parties de mon projet sans avoir a interroger de nouveau la base de donnée.

Et je redirige l'utilisateur vers la route « reservation_prestation », en passant l'ID de la prestation comme paramètre. Cette redirection amènera l'utilisateur sur la page pour finalisez sa réservation.

Dans un second temps ce que j'ai fait, c'est de créer la page sur laquelle sera redirigé l'utilisateur pour valider la réservation.

Pour commencer j'ai initialisé deux variables principales : 'event' et 'selectEvent'. La première contient les données de la prestation choisie par l'utilisateur, récupérées du serveur Twig, tandis que la seconde, initialisée à null, servira à suivre les sélections de l'utilisateur sur le calendrier.

Ensuite avec la bibliothèque « Fullcalendar », j'ai généré un calendrier, pour faire ça, j'ai sélectionné l'élément HTML ou le calendrier sera affiché grâce à son identifiant, '#calendrier'(cf. annexe 24 ligne 20).

J'initialise ensuite le calendrier en lui passant une série d'options. Parmi ces options, j'ai défini les heures de travail 'businessHours', ce qui empêche les utilisateurs de sélectionner des créneaux en dehors des horaires définis (cf. annexe 24 ligne 36) a (cf. annexe 25 ligne 46).

Ensuite j'ai activé la possibilité de sélectionner des créneaux ('selectable') et de cliquer sur des évènements déjà existant ('eventClick') (cf. annexe 24 ligne 24).

J'ai ensuite spécifié la durée minimale d'un créneau ('slotDuration') à 30 minutes (cf. annexe 24 ligne 27).

Et j'ai également ajouté des contraintes sur les sélections ('selectConstraint') (cf. annexe 24 ligne 26)pour assure qu'elles correspondent aux horaires de travail.

Ensuite j'ai paramétré les interactions de l'utilisateur pour faire en sorte que quand l'utilisateur sélectionne ('select') (cf. annexe 26), un créneau je vérifie sa durée. Si elle est plus courte que la durée de la prestation, je l'ajuste en conséquence. Ensuite, j'ajoute un nouvel évènement au calendrier avec les détails de la prestation. Cette étape permet de s'assurer que les réservations correspondent bien à la durée des prestations choisies.

Et pour finir, j'effectue le rendu du calendrier pour l'afficher a l'utilisateur (cf. annexe 27).

```
#[Route("/prestation_reservation/{id}", name: "prestation_reservation")]
public function prestation_reservation($id, PrestationRepository $prestationRepository, SessionInterface $session):Response
{
    $prestationId = $prestationRepository->find($id);
    $prestation = $prestationId;
    // You, il y a 3 jours * Ajoue de plusieurs fonctionnalite
    // On récupère la durée de la prestation en heure et minute
    $dureHeures = $prestationId->getDureHeure();
    $dureMinutes = $prestationId->getDureMinutes();

    if($dureHeures > 0){
        $dureMinutes += $dureHeures * 60;
    }
    $event = [
        'id' => $prestation->getId(),
        'nom' => $prestation->getNom(),
        'dureMinutes' => $dureMinutes,
    ];
    $session->set('event', $event);

    return $this -> redirectToRoute("reservation_prestation", [
        'id' => $id
    ]);
}
```

Annexe 23 -Route prestation_resrevation


```

14  {% block javascripts %}
15  <script>
16      let event = {{ event|json_encode|raw }};
17      let selectEvent = null;
18      //Sers a générer un calendrier
19      window.onload = () => {
20          let calendarElt = document.querySelector("#calendrier")
21          let calendar = new FullCalendar.Calendar(calendarElt, {
22              initialView: 'timeGridWeek',
23              editable: true,
24              selectable: true,
25              eventConstraint: 'businessHours',
26              selectConstraint: 'businessHours',
27              slotDuration: '00:30:00', //durée minimum pour une prestation
28
29              locale: 'fr',
30              timeZone: 'Europe/Paris',
31              headerToolbar: {
32                  start: 'prev, next today',
33                  center: 'title',
34                  end: 'timeGridWeek',
35              },
36              businessHours: [
37                  {
38                      daysOfWeek: [1, 2, 3, 4, 5], // Monday to Friday
39                      startTime: '08:00',
40                      endTime: '12:00',
41                  },

```

Annexe 24 - calendrier

```

42  { daysOfWeek: [1, 2, 3, 4, 5], // Monday to Friday
43      startTime: '13:00',
44      endTime: '17:00'
45  },
46  ],
47  selectOverlap: false,
48  selectOverlap: function(event) {
49      return event.rendering === 'background';
50  },
51  eventClick: function(info){
52      if(confirm("Etes vous sûr de vouloir supprimer cette reservation ?")){
53          info.event.remove(); //On supprime la reservation
54      }
55  },

```

Annexe 25 - calendrier

```

56 select: function(info){
57   let prestationDuration = event.dureMinutes;
58   let selectDuration = (info.end - info.start) / 1000 / 60;
59
60   if(selectDuration < prestationDuration){
61     //Calculer la nouvelle date de fi
62     let newEnd = new Date(info.start.getTime() + prestationDuration * 60 * 1000);
63
64     //Ajouter l'évènement au calendrier
65     calendar.addEvent({
66       title: event.nom,
67       start: info.start,
68       end: newEnd,
69       color: 'red',
70       textColor: 'white'
71     });
72   }else{
73     //Si la durée de la sélection est suffisant on ajoute la prestation
74     calendar.addEvent({
75       title: event.nom,
76       start: info.start,
77       end: info.end,
78       color: 'red',
79       textColor: 'white'
80     });
81   }
82   selectEvent = {
83     prestation: event.nom,
84     date: info.start,
85     start: info.start,
86     end: info.end
87   };
88 }

```

You, il y a 3 jours • Ajoute de plusieurs fonctionnalités

Annexe 26 – calendrier

```

calendar.render();
{% endblock %}

```

Annexe 27 – calendrier

Les vues dynamiques

Pour les vues dynamiques, j'ai vais présenter deux page la première page c'est la page « Connexion_sécurité » (cf. annexe 27), sur cette page on va retrouver le nom de l'utilisateur, son prénom et son email, ainsi qu'un bouton modifier, ces informations seront différent suivant l'utilisateur connecté.

Connexion et sécurité

Profil de l'utilisateur
Nom : Tom
Prénom : Cruise
Email : tomcruise@gmail.com
Modifier

Annexe 27 – Connexion_sécurité

La deuxième, sera la page où l'utilisateur pourra consulter les commandes qu'il aura passé sur cette page on retrouvera le numéro de commande, la date de commande, l'adresse de l'utilisateur, le nom du produit, le prix du produit et le prix total du panier (cf. annexe 28)

Commande Passé		
Numéro de commande:CMD-01H82F03		Total:20€
Date de commande:17/08/2023 19:19		
Adresse: 2, Allée du midi, 94310, Orly		
	Nom du produit: peau sèche Prix du produit: 20€	

Annexe 28 – Commande

L'implémentation d'un système de commandes dans une application est une étape cruciale pour garantir une expérience utilisateur fluide, surtout pour un projet e-commerce. Voici une explication détaillée de la manière dont j'ai procédé pour créer ce mécanisme.

La fonctionnalité que je vais décrire (cf. annexe 29) a pour objectif de transformer le panier d'achat d'un utilisateur en une commande formelle, tout en enregistrant tous les détails nécessaires.

Au début j'ai commencé par instancier un nouvel objet « Ulid », qui est une approche moderne pour générer des identifiants uniques. Cette valeur « ulid » a ensuite été formatée et préfixée par « CMD- », pour créer un identifiant de commande distinctif et mémorable.

J'ai également capturé la date et l'heure de la commande à l'aide de l'objet « DateTime », pour savoir précisément quand la commande a été passée.

Ensuite j'ai associé la commande à l'utilisateur actuellement connecté. Cela garanti que chaque commande peut être reliée directement à un utilisateur spécifique, facilitant ainsi le suivi et la gestion des commandes.

Le panier de l'utilisateur, qui est stocker temporairement dans la « session », contient tous les produits que l'utilisateur souhaite acheter. En parcourant chaque produit du panier, j'ai multiplié le prix du produit par sa quantité pour obtenir le total pour cet article.

Chaque total est alors ajouté à une somme cumulative pour obtenir le coût total de la commande.

Pour chaque produit dans le panier, j'ai créé un nouvel objet « Article ». Cet objet stock les détails du produit, tels que le produit lui-même, la quantité, et le prix, au moment de l'achat. Cela permet de garder une trace très exacte de ce qui a été commandé, même si les détails du produit ou son prix venait à changer ultérieurement.

Après avoir calculé le total, je l'ai associé à l'objet de commande, j'ai ensuite sauvegardé toutes ces informations dans la base de données. Enfin le panier de l'utilisateur a été vider, signifiant que sa commande a été finalisé et qu'il peut recommencer un nouveau processus d'achat si nécessaire.

Après avoir terminé la création de la commande, l'utilisateur est redirigé vers une page où il pourra renseigner son adresse de livraison.

```

public function index( ArticleRepository $articleRepository, ProduitRepository $produitRepository,
SessionInterface $session, CommandeRepository $commandeRepository, EntityManagerInterface $em): Response
{
    //Permet de valider la commande et de passer a l'étape pour enregistrer ces information personnel
    $ulid = new Uuid();
    $numeroDeCommande = 'CMD-' . strtoupper(substr($ulid->toBase32(), 0, 8));
    $dateDeCommande = new DateTime();
    $user = $this->getUser();
    $commande = new Commande();
    $commande->setUser($user);
    $panierSession = $session->get("panier");
    //On calcule le total de la commande
    $total = 0;
    $commande->setNumeroDeCommande($numeroDeCommande);
    $commande->setDateDeCommande($dateDeCommande);
    $commandeRepository->save($commande, false);

    foreach($panierSession as $key=>$value){//On fait une boucle sur chaque produit du panier

        $article = new Article();//On définit les propriété de l'instance Article en utilisant les données récupérées
        $produit = $produitRepository->find($key);
        $total += $produit->getPrix() * $value;
        $article->setCommande($commande);
        $article->setProduit($produit);
        $article->setQuantite($value);
        $article->setPrixFixe($produit->getPrix());
        $articleRepository->save($article, false);
    }
    $commande->setTotal($total);
    $em->flush();
    $session->remove('panier');
    return $this->redirectToRoute('adresse_commande', ['id' => $commande->getId()]);
}

```

Annexe 29 – Création commande

La fonction que je vais décrire, sert à expliquer comment j'ai fait pour récupérer l'adresse de l'utilisateur (cf. annexe 30).

J'ai commencé par instancier un nouvel objet « Adresse ». Cet objet servira à stocker les informations sur l'adresse que l'utilisateur saisira.

Ensuite, j'ai créé un formulaire à partir de la classe « AdresseType », qui a pour but de définir la structure et les champs du formulaire (cf. annexe 31). Ce formulaire est associé à l'objet « Adresse » nouvellement créé. Cela signifie que les données soumises via le formulaire rempliront cet objet.

Ensuite, j'utilise « handleRequest » pour capturer et traiter les données envoyées via le formulaire lorsque l'utilisateur le soumet. Je vérifie ensuite si le formulaire été soumis et si les données sont valides. Si le formulaire est valide, on récupère l'utilisateur actuellement connecté, ensuite l'adresse saisie est associée à l'utilisateur grâce à la méthode « setUser ». L'adresse est aussi liée à la commande en cours, en utilisant la méthode « addCommande ».

L'adresse complétée est ensuite enregistrée dans la base de données. C'est là que le « AdresseRepository » intervient grâce à sa méthode « save ».

Après avoir sauvegardé l'adresse l'utilisateur sera redirigé vers la page ou il devra renseigner son moyen de paiement.

```

/**      You, il y a 3 jours + Ajout de plusieurs fonctionnalités
 * @Route("/commande/adresse/{id}", name="adresse_commande")
 */
public function adresse(Commande $commande, Request $request, AdresseRepository $adresseRepository):Response
{
    $adresse = new Adresse;
    $form = $this->createForm(AdresseType::class, $adresse);

    $form->handleRequest($request);

    if($form->isSubmitted() AND $form->isValid())
    {
        $user = $this->getUser();

        $adresse->setUser($user);
        $adresse->addCommande($commande);

        $adresseRepository->save($adresse, true);
        return $this->redirectToRoute('payement');
    }
    return $this->render('adresse/index.html.twig', [
        'form' => $form->createView()
    ]);
}

```

Annexe 30 – Création adresse

E. Sécurité

Sécurisations des routes et rôles

Afin de sécuriser le contenu du site, j'ai décidé que les routes précédées de « /admin » ne devaient être accessibles qu'aux utilisateurs ayant le rôle administrateur.

C'est pour ça que lorsque 'on crée un projet en « --full », Symfony installe le bundle « security-bundle », qui crée une propriété « rôles » dans lequel on enregistre le rôle de l'utilisateur.

J'ai donc décidé de donner le rôle 'ROLE_ADMIN' au compte de l'administrateur. Ce bundle crée aussi dans le dossier config, le fichier 'security.yaml' dans lequel on peut gérer les accès aux routes.

Il suffit de « décommenter » (cf. annexe 31, ligne 34) pour protéger les routes admin.

```

1 security:
2   # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
3   password_hashers:
4     Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
5   # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
6   providers:
7     # used to reload user from session & other features (e.g. switch_user)
8     app_user_provider:
9       entity:
10         class: App\Entity\User
11         property: email
12   firewalls:      You, il y a 4 jours • Ajoute de plusieurs fonctionnalités
13     dev:
14       pattern: ^/(_(profiler|wdt)|css|images|js)/
15       security: false
16     main:
17       lazy: true
18       provider: app_user_provider
19       custom_authenticator: App\Security\EmailAuthenticator
20       logout:
21         path: app_logout
22         # where to redirect after logout
23         target: home
24
25       # activate different ways to authenticate
26       # https://symfony.com/doc/current/security.html#the-firewall
27
28       # https://symfony.com/doc/current/security/impersonating\_user.html
29       # switch_user: true
30
31   # Easy way to control access for large sections of your site
32   # Note: Only the *first* access control that matches will be used
33   access_control:
34     - { path: ^/admin, roles: ROLE_ADMIN }
35     #- { path: ^/profil, roles: ROLE_USER }

```

Annexe 31 – Security.Yaml

Authentication

Afin d'avoir un système de connexion simplifié, j'ai utilisé la commande « php bin/console make :auth ».

Cette commande permet de créer un « SecurityController » qui, via les routes login et logout, gère la connexion et la déconnexion des utilisateurs.

De plus, il crée la vue 'login.html.twig', qui comporte un formulaire de connexion et la classe « UserAuthenticator ».

On retrouve dans cette classe la fonction « OnAuthenticationSuccess » qui permet de contrôler le comportement du site notamment, la redirection après une connexion réussie.

```
public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
{
    if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
        return new RedirectResponse($targetPath);
    }

    //recuperer le user, verifier si il a un panier en base de donnée et s'il y'en a un le mettre dans la session.
    //Faire une boucle sur le panier pour afficher chaque produit, ensuite dissocié les produit du panier
    $user = $token->getUser();
    $session = $this->requestStack->getCurrentRequest()->getSession();//permet de recuperer la session de l'utilisateur connecter
    //You, maintenant * Uncommitted changes
    $panier = [];
    if($user->getPanier()){
        foreach($user->getPanier()->getArticles() as $article){
            $panier[$article->getProduit()->getId()] = $article->getQuantite();
        }
    }
    $session->set("panier", $panier);
    // For example:
    return new RedirectResponse($this->urlGenerator->generate('home'));
    throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
}

protected function getLoginUrl(Request $request): string
{
    return $this->urlGenerator->generate(self::LOGIN_ROUTE);
}
```

Annexe 32 – OnAuthenticationSuccess

Encodage du mot de passe

Afin d'encoder le mot de passe, j'ai utilisé la dépendance "UserPasswordHasherInterface » que j'utilise pour encoder les mots de passe saisie dans le formulaire avant de les envoyer dans la base de données.

```
$user = new User();
$form = $this->createForm(RegistrationFormType::class, $user);
$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {

    $user->setPassword(
        $userPasswordHasher->hashPassword(
            $user,
            $form->get('Password')->getData()
        )
    );
}
```

Annexe 33 - UserPasswordHasherInterface

F. Contraintes de langage

Tout au long de création de mon projet, il m'a fallu effectuer des recherches sur internet et regarder des vidéos pour trouver et comprendre des fonctions qui me permettrait de faire ce que je voulais.

Cependant, la majeure partie des sites sur lesquels je trouvais des réponses était en anglais, comme le site <https://stackoverflow.com> sur lequel j'ai passé beaucoup de temps.

J'ai notamment eu beaucoup de problème lors de la création de mon calendrier avec Fullcalendar, notamment pour créer des événements, pour définir les contraintes pour que l'utilisateur ne puisse pas sélectionner des horaires en dehors des heures de travail, pour que l'utilisateur puisse supprimer sa réservation. Pour résoudre ces problèmes j'ai consulté le site <https://fullcalendar.io/docs>.

Mais le plus gros problème que j'ai eu ça a été avec « slotDuration » car « slotDuration » a une fréquence d'affichage des horaires de '30minute' (cf. annexe 34), alors que mes prestations peuvent aller au-delà de 30 minute, c'est pourquoi j'ai dû créer une fonction qui fait en sorte que lorsque l'utilisateur sélectionne un créneau horaire dans le calendrier pour ajouter un événement, le code vérifie l'utilisateur a choisi suffisamment de temps pour l'événement prévu.

Si la durée est trop courte, le code ajuste automatiquement la fin de l'événement pour qu'il ait la durée correcte. Sinon il ajoute l'événement tel quel.

slotDuration

The frequency for displaying time slots.

Duration, *default*: `'00:30:00'` (30 minutes)

Annexe 34 – SlotDuration

If "businessHours" is given, events being dragged or resized must be fully contained within the week's business hours (Monday-Friday 9am-5pm by default). A custom `businessHours` value will be respected.

A custom time-window, an object identical to what `businessHours` accepts, can also be given:

```
{
  startTime: '10:00', // a start time (10am in this example)
  endTime: '18:00', // an end time (6pm in this example)

  daysOfWeek: [ 1, 2, 3, 4 ]
  // days of week. an array of zero-based day of week integers (0=Sunday)
  // (Monday-Thursday in this example)
}
```

Annexe 35 – businessHours

Ce code explique que si « `businessHours` » (cf. annexe 24, 25 ligne 36 à 45), est fourni, les évènements qui sont déplacés ou redimensionnés doivent être entièrement contenus dans les heures d'ouverture de la semaine, il précise aussi que une fenêtre temporelle personnalisée, qui contient un objet identique à ce que « `businessHours` » accepte peut être fournie

IV. Conclusion

La formation

Les cinq mois passé à suivre les cours de « NEXTFORMATION » m'ont conforté dans mon choix de me reconvertir dans le développement web.

En fait, j'ai vite compris que ce métier était fait pour moi. Mon sens de la logique et de la créativité se mariant parfaitement avec celui du développement web ; j'ai su maîtriser les différents langages malgré leur complexité.

Le projet

La conception de mon projet m'a suivi tout au long de ma formation. J'ai donc essayé d'apporter des améliorations au fur et à mesure que mes compétences on évoluer. Avec le recul et l'expérience acquise depuis le début, je peux dire que certaine chose aurait pu être différente.

Le stage

Au cours de ma période de mise en pratique au sein de l'entreprise « Odpo », j'ai pu développer mes compétences et en acquérir de nouvelles, comme la mise en place d'un WebDav ou le fait de communiquer avec des API. J'ai aussi apporté un soutien à l'équipe en étant disponible et réactif à leur demande.

Cette collaboration m'a permis à la fois d'avoir un support sur leur travail et dans un même temps de consolider mes compétences et enrichir mes connaissances techniques. Cette réussite m'a encouragé à poursuivre dans ce métier.