



2023

Rapport de Projet

**Titre Professionnel de Développeur web et web mobile
Niveau 5 (BTS/DUT)**



FLOREIL

Konstantin Shilkov
Développeur web et web mobile chez Floreil

BTS/DUT DWWM - Projet 2023 – Shilkov

Konstantin – Floreil (www.Floreil.com)

Formateur: Patrick Croquet

05/09/2022 - 28/04/2023

Table des matières

| | | |
|---------------|--|-----------|
| 1. | Introduction..... | 2 |
| 2. | Projet | 3 |
| 2.1. | Cahier des charges..... | 3 |
| 2.1.1 | La couverture :..... | 3 |
| 2.1.2 | Les cibles : | 4 |
| 2.1.3 | Les objectifs quantitatifs : | 5 |
| 2.1.4 | Les perimetros du projet : | 5 |
| 2.2. | Graphisme et ergonomie | 6 |
| 2.2.1 | La planche de tendance :..... | 6 |
| 2.2.2 | Le logo : | 6 |
| 2.2.3 | La typographie : | 7 |
| 2.2.4 | La palette de couleurs : | 8 |
| 2.2.5 | La Maquette : | 9 |
| 2.3. | Activité n°1 : Développer La Partie Front-End..... | 10 |
| 2.4.1 | Competence n°1 : Maquetter une application..... | 10 |
| 2.4.2 | Competence n°2 : Realiser une interface utilisateur web statique et adaptable | 13 |
| 2.4.3 | Competence n°3 : Developper une interface utilisateur web dynamique | 17 |
| 2.4. | Activité n°2 : Développer La Partie Back-End..... | 39 |
| 2.5.1. | Competences n°5 : Créer une base de données | 39 |
| 2.5.2. | Competences n°6 : Développer les composants d'accès aux données | 46 |
| 2.5.3. | Competences n°7 : Développer la partie back-end..... | 52 |
| 2.5. | La sécurité..... | 60 |
| 3. | L'utilisation de l'anglais dans le travail sur le projet..... | 69 |
| 4. | Conclusion | 72 |
| 5. | Remerciements..... | 73 |

1. Introduction

Pendant ces deux mois consacrés au projet **Floreil**, j'ai choisi de réaliser un site e-commerce pour la vente de fleurs, afin de mettre à profit mes connaissances et d'en acquérir de nouvelles. Le projet Floreil est une très petite entreprise (TPE) qui vise à fournir des services de qualité aux clients à la recherche de fleurs de qualité.

Pour la réalisation de ce projet, j'utilise les langages de programmation suivants: **HTML, CSS, PHP, SQL**, ainsi que **JavaScript**

Les principaux outils que j'ai choisis pour la réalisation de mon projet sont **Symfony 6** et **Bootstrap 5**.

Symfony 6 est un framework de développement web open source écrit en PHP. Il fournit des outils et des bibliothèques pour accélérer le développement de sites web.

Bootstrap 5 est une bibliothèque de composants d'interface utilisateur open source pour le développement web. Elle offre des composants prêts à l'emploi tels que des boutons, des formulaires, des modèles de mise en page, etc., qui peuvent être facilement intégrés à un site web.

J'utilise également les outils suivants: **PhpMyAdmin** et pour la gestion de la base de données, ainsi que **Composer** et **Nodejs** pour la gestion des dépendances.

Les **logiciels et applications** supplémentaires :

InkScape, Figma, Visual Studio Code, WAMP.

En tant que chef et créateur de l'entreprise Floreil, mon objectif est de développer un site web original, en effectuant des missions plus intéressantes les unes que les autres. Ayant une grande passion pour l'entreprenariat, ce choix s'est imposé naturellement.

2. Projet

2.1. Cahier des charges

2.1.1 La couverture :

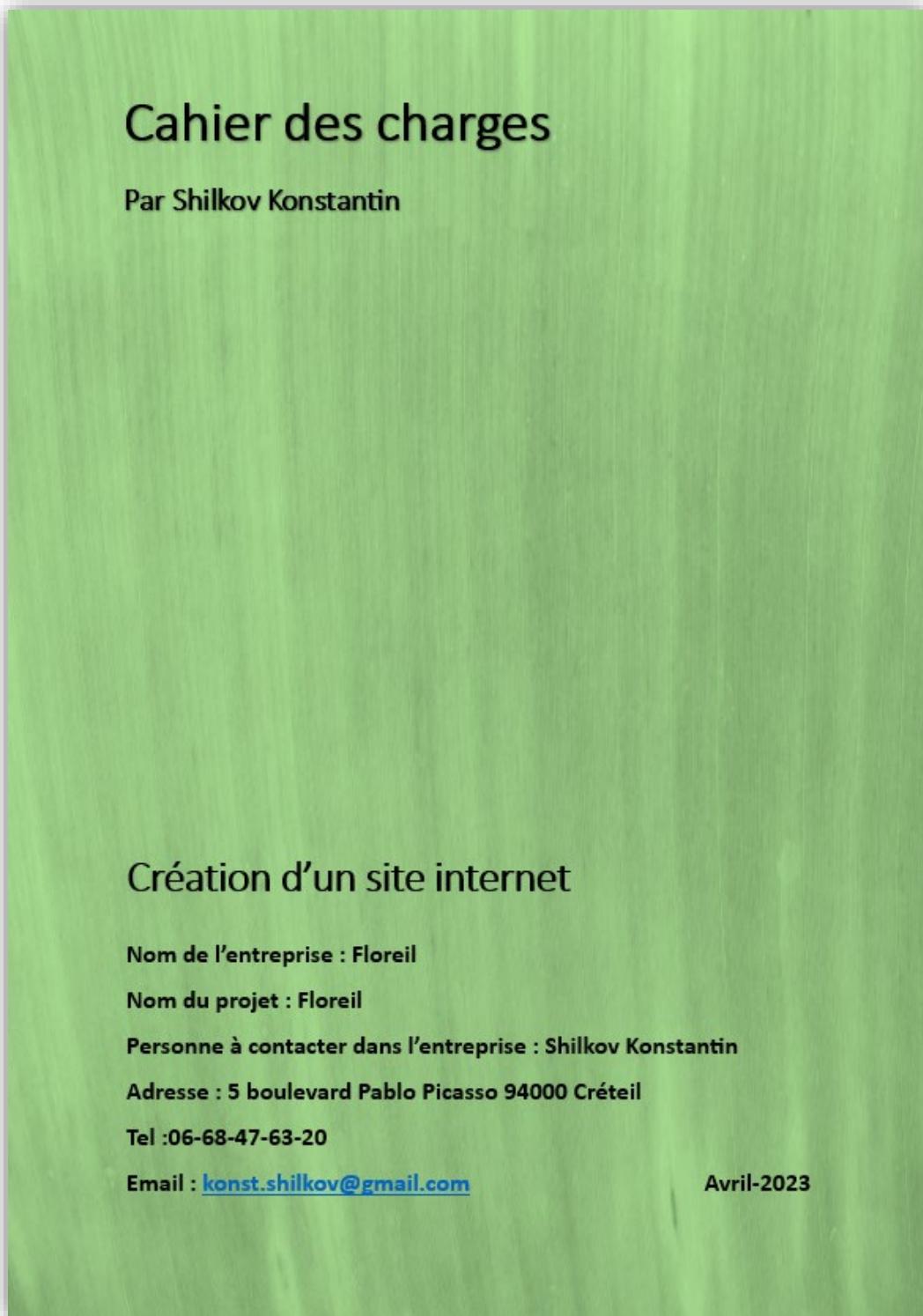


Fig. 1 Couverture cahier des charges

2.1.2 Les cibles :

Nous visons un public de particuliers, voici le profil type de nos prospects. J'ai créé ce Persona avec le logiciel **Xtensio** et la banque de photos **unsplash**.

Amelie Delorme



Age: 45
Travail: Médecin
Family: Divorcée, enfants adultes
Location: Deauville, Calvados

Motivation

- Site ergonomique adapté aux mobiles
- Design du site
- Description des produits
- Photos du site
- Commentaires des visiteurs

Brands & Influencers







Personality

| | |
|---------------|----------------|
| Introverti(e) | Extroverti(e) |
| Rationnel(l)e | Emotionnel(l)e |
| Jugement | Perception |

Buts

- Elle veut meubler sa nouvelle maison
- Elle veut faire un grand jardin avec ses mains
- Passer du temps libre sans agitation ni stress

Frustrations

- Stress dû à une lourde charge de travail
- Il n'y a pas de personne proche à proximité
- Elle est trop timide pour se faire de nouveaux amis

Bio

Née dans une petite banlieue, Amélie voulait être médecin depuis son enfance, comme ses parents. Après avoir été diplômée d'une université de médecine, elle a obtenu un emploi dans l'un des hôpitaux de Paris. Après 10 ans, Amélie s'installe en Normandie et part travailler dans le privé. Elle est divorcée et ses enfants ont grandi et sont assez indépendants.

Utilisation écrans
(% d'utilisation lors des recherches sur le web)



43%

Utilise son smartphone très régulièrement pour faire des recherches sur le web



0%

Ne possède pas ce type d'objet connecté



57%

Utilise à temps équivalent l'ordinateur pour la navigation web

Mathieu Leiris



Age: 36
Travail: Jardinier
Family: Marié, 2 enfants
Location: Crétteil, Île-de-France

Motivation

- Site ergonomique adapté aux mobiles
- Design du site
- Description des produits
- Photos du site
- Commentaires des visiteurs

Brands & Influencers







Personality

| | |
|---------------|----------------|
| Introverti(e) | Extroverti(e) |
| Rationnel(l)e | Emotionnel(l)e |
| Jugement | Perception |

Buts

- Il veut développer son activité
- Exécution des travaux qualitativement et sans délais
- Passer du temps libre avec sa famille

Frustrations

- Retard des fournisseurs
- Il n'y a pas beaucoup de temps libre
- Concurrence intense dans ce domaine

Bio

Mathieu a toujours eu une passion pour l'aménagement paysager. Après avoir quitté l'école, il a commencé à travailler comme jardinier dans une entreprise réputée, et après 5 ans, il a ouvert sa propre entreprise et maintenant 5 personnes travaillent sous lui. Ses enfants partagent sa passion et l'aident souvent dans son travail.

Utilisation écrans
(% d'utilisation lors des recherches sur le web)



28%

Utilise son smartphone régulièrement pour faire des recherches sur le web



16%

Utilise son smartphone régulièrement pour faire des recherches sur le web



56%

Utilise très régulièrement l'ordinateur pour la navigation web

BTS/DUT DWWM - PROJET 2023 – SHILKOV KONSTANTIN – FLOREIL (WWW.FLOREIL.COM)

4

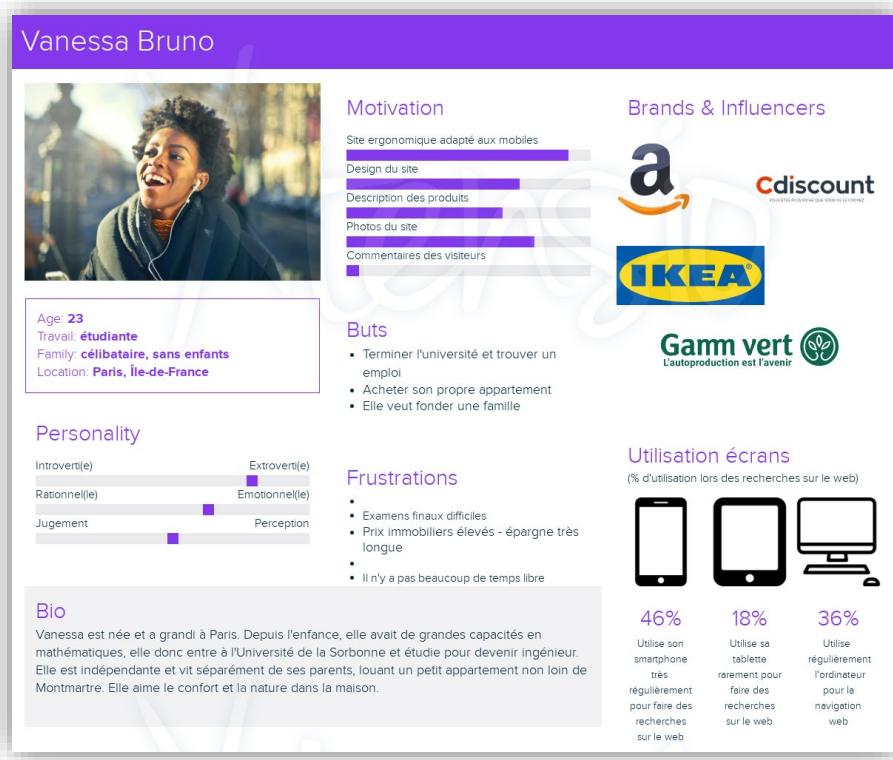


Fig.2 Les personas

2.1.3 Les objectifs quantitatifs :

- Nous visons un volume de trafic d'environ 200 000 par an.
- Avec un volume de contact d'environ 20 000 par an.
- Notre site va contenir entre 10 et 15 pages.

2.1.4 Les perimètres du projet :

- Notre site doit être Francophone.
- Utiliser ces langages de préférence : **Symfony6, Bootstrap5, Php8**
- Notre site va contenir une page boutique e-commerce.
- Nous allons utiliser « Stripe » comme méthode de paiement en ligne.
- Nous adapterons notre site sur version mobile et également sur application.
- Nous allons utiliser création de compte, système de paiement en ligne.

2.2. Graphisme et ergonomie

2.2.1 La planche de tendance :

Création de la Planche de tendance avec les logiciels **InkScape**. On y retrouve le logo du site, les couleurs, le slogan et quelques illustrations venant de ma banque de photos personnelle qui seront intégrées au projet.



Fig.3 La planche de tendance (moodboard)

2.2.2 Le logo :

J'ai créé ce Logo sur mon ordinateur avec le logiciel **InkScape**.



Fig.4 Le logo

2.2.3 La typographie :

J'ai choisi cette police pour son effet rustique que je recherchais pour mon projet.

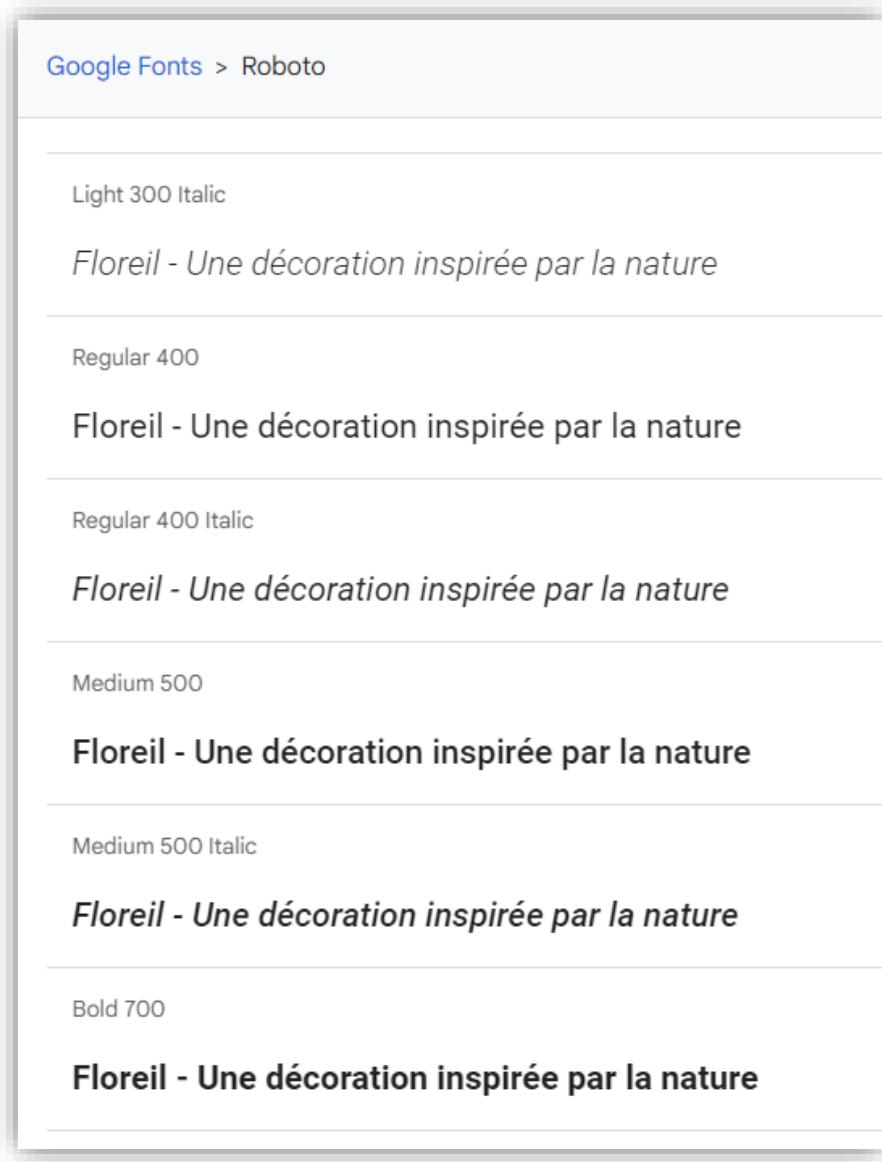


Fig.5 La police

2.2.4 La palette de couleurs :

J'ai choisi ces couleurs car elles correspondent parfaitement avec le thème de mon site.

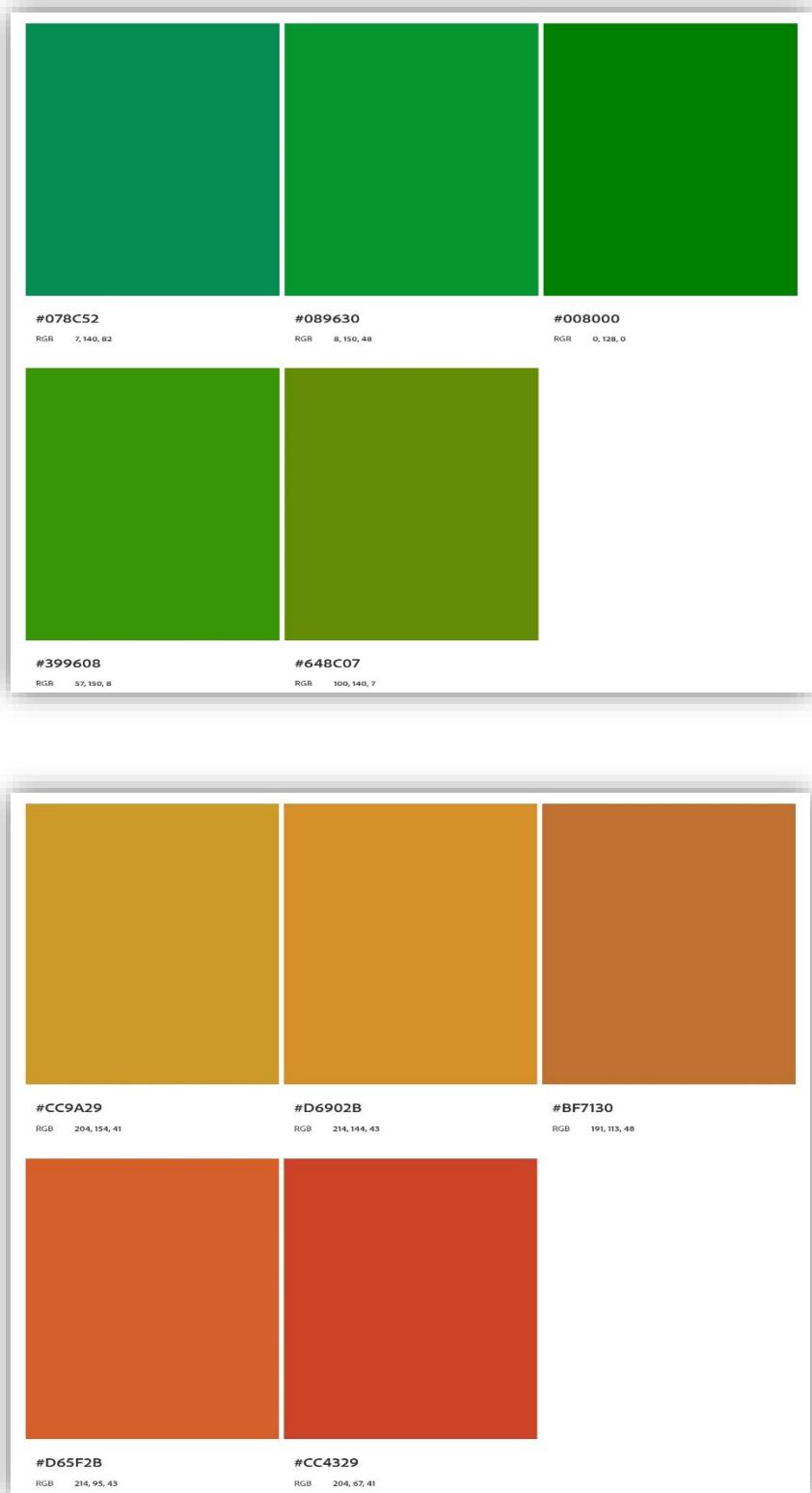


Fig.5 La palette de couleurs

2.2.5 La Maquette :

J'ai créé cette Maquette en format mobile, tablette et desktop avec le logiciel **Figma**.

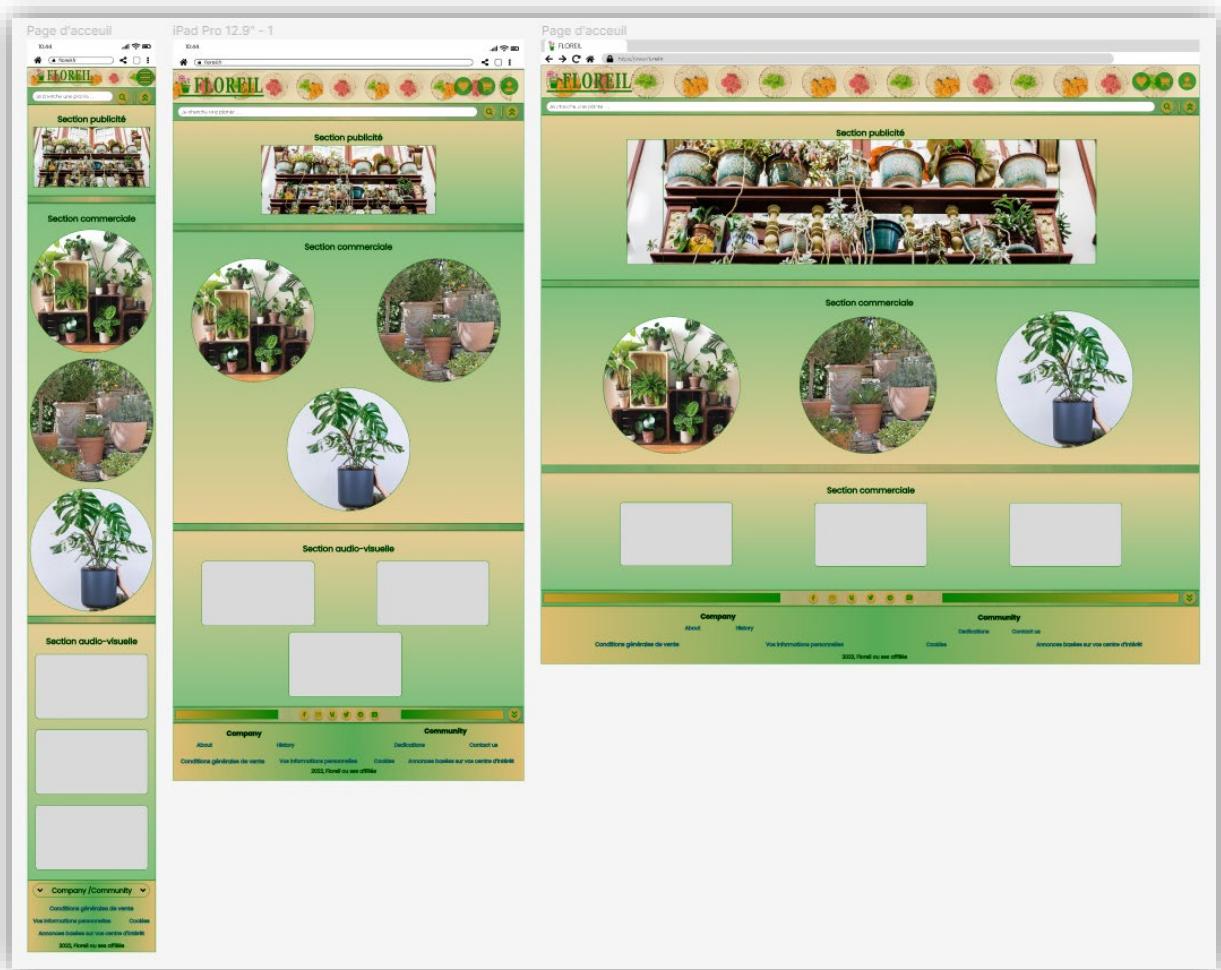


Fig.6 La maquette

2.3. Activité n°1 : Développer La Partie Front-End

2.3.1 Competence n°1 : Maquetter une application

J'ai débuté par créer une planche de tendance (**Moodboard**), avec le logiciel **InkScape**, où l'on retrouve le logo, les couleurs, le slogan et quelques illustrations qui seront intégrées au projet.

Ce qui va permettre aux clients(es) d'avoir un rendu visuel qui correspond à leurs attentes sur le contenu de leur projet.



Fig.7 La planche de tendance(moodboard)

Ensuite j'ai cree une charte graphique avec les logiciels **InkScape** et **Figma**, ce qui consiste a detailler tous les points a respecter qui regroupe l'ensemble des codes definissant l'identite visuelle du projet.

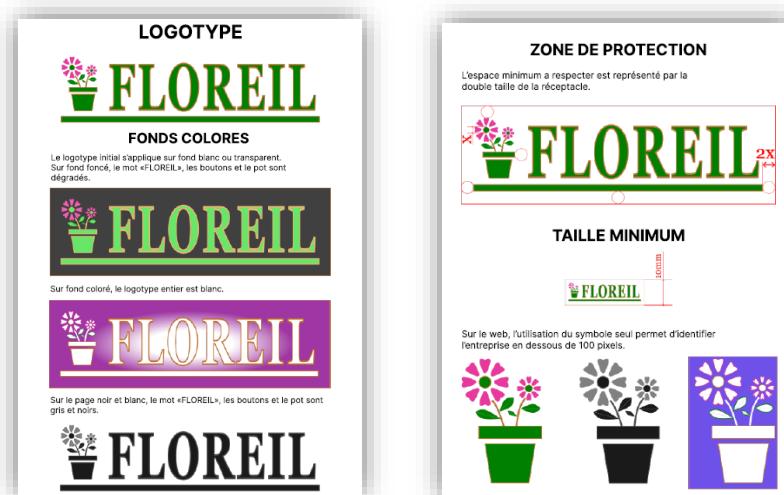


Fig.8 La charte graphique

Dans mon introduction j'ai commencé par présenter l'objectif du projet qui est de créer un site web pour un magasin en ligne de vente de plantes décoratives. Ce site permettra aux clients cibles, des femmes avec un niveau d'éducation élevé, des jardiniers et des étudiantes, de trouver toutes sortes d'informations sur les plantes proposées, telles que :

- Les soins appropriés pour les différentes plantes en vente, afin de garantir leur santé et leur développement optimal.
- Les spécificités de chaque type de plante, afin que les clients puissent choisir les plantes les mieux adaptées à leurs besoins et à leur environnement.
- Les conditions de livraison et les politiques de retour, pour une expérience d'achat en ligne agréable et sans stress.

A la suite, j'ai présenté le slogan du projet ainsi que le logo avec ses dimensions, ses couleurs, et j'ai indiqué la typographie utilisée dans le projet, la palette de couleurs avec ses codes tels que RGB, HEXADECIMAL et CMJN, et j'ai également inclus des exemples d'icônes et de graphiques utilisés dans le site. Enfin, j'ai créé la maquette de la page d'accueil du site web en utilisant des logiciels de mise en page tels que **Figma**, en respectant les codes définis dans la charte graphique du site.

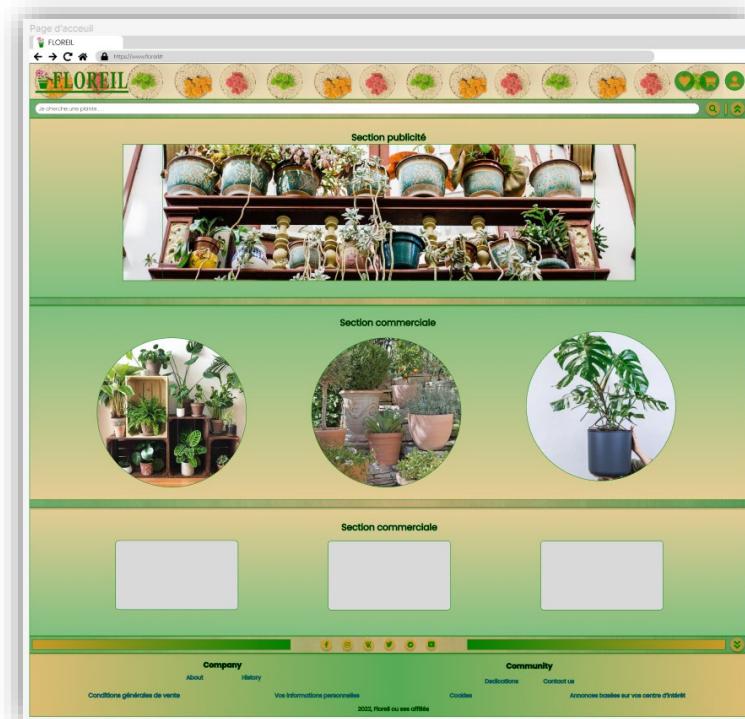


Fig.9 La maquette de la page d'accueil de mon projet

Pour finir avec cette compétence j'ai commencé à coder mon projet en **HTML 5** et en **CSS 3** avec l'éditeur de code **Visual Studio Code** et pour sauvegarder l'avancée de mon travail en créant des dépôts j'ai utilisé **Github Desktop**.

```

index.html
1 <!DOCTYPE html>
2 <html lang="zxx">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge"
6   <meta name="viewport" content="width=device-width,
7   <link rel="stylesheet" href="css/style.css">
8   <link rel="stylesheet" href="https://fonts.googleapis.com/
9   <script src="https://kit.fontawesome.com/0461c0955a
10

File Edit View Repository Branch Help
Current repository Project-Floreil Current branch master Fetch origin Last fetched just now
Changes History all_done_v0.1
ShilkovKonst 38015cb 1 changed file +7 -4
...style.css @@ -186,7 +186,7 @@ header>e{
186 186 #dropdown_menu_bouton:hover ~ #dropdown_menu_list, #dropdown_menu_list:hover {
187 187 visibility: visible;
188 188 position: sticky;
189 189 - top: 198px;
190 190 + top: 200px;
191 191 opacity: 1;
192 192 transition: all 0.3s;
193 193 }
194 194 @@ +320,15 +320,17 @@ iframe{
320 320 /*Секция правого бокового стационарного меню*/
321 321 aside {
322 322 padding: 1%;
323 323 - flex-shrink: 1;
324 324 + /*flex-shrink: 1;*/
325 325 width: 250px;

```

Fig.10 : Code HTML / Code CSS / Dépôt de Visual Studio Code vers GitHub Desktop

Pour sauvegarder l'avancée de mon travail j'ai utilisé **Github Desktop** en créant des dépôts, j'ai donc commencé par créer un compte sur le site de **Github.com**. J'ai poursuivi sur le téléchargement de GithubDesktop.

Je l'ai installé sur ma machine, puis j'ai fait la relation avec mon éditeur de code **Visual Studio Code**, en installant l'extension de Github dans celui-ci.

J'ai vérifié si la relation entre Visual Studio Code et mon compte Github était bien Active.

J'ai créé un dépôt vers GithubDesktop en lui donnant un Nom.

Puis je Push mes commits, vers mon compte sur la plateforme Github.com, là où je vais pouvoir

retrouver tous les fichiers de mon projet, je pourrais alors les récupérer de n'importe où, pour

reprendre mon projet et avancé, et également les partager avec de futurs intervenants.

2.3.2 Competence n°2 : Realiser une interface utilisateur web statique et adaptable

En développement web, une **interface utilisateur statique** fait référence à un site web ou à une application web qui possède une mise en page, un design et un contenu fixe qui ne change pas à moins que le développeur web ne le mette à jour manuellement. Une interface statique est généralement codée en HTML et CSS et ne comprend pas d'éléments dynamiques tels que des animations, du contenu généré par l'utilisateur ou des mises à jour en temps réel.

Une **interface utilisateur adaptable** est conçue pour fonctionner efficacement sur différents appareils et tailles d'écran. Avec l'utilisation croissante des appareils mobiles pour accéder au web, il est important pour les développeurs web de s'assurer que leurs interfaces peuvent s'adapter aux écrans plus petits sans perdre de fonctionnalités ou d'utilisabilité. Les interfaces adaptables utilisent généralement des techniques de conception réactive telles que des mises en page fluides, des requêtes multimédias et des images et du texte flexibles pour ajuster la mise en page et le contenu en fonction de la taille et de l'orientation de l'écran de l'appareil.

Pour cette deuxième compétence j'ai commencé par maquetter mon projet en format mobile, tablette et desktop toujours avec le logiciel **Figma**.

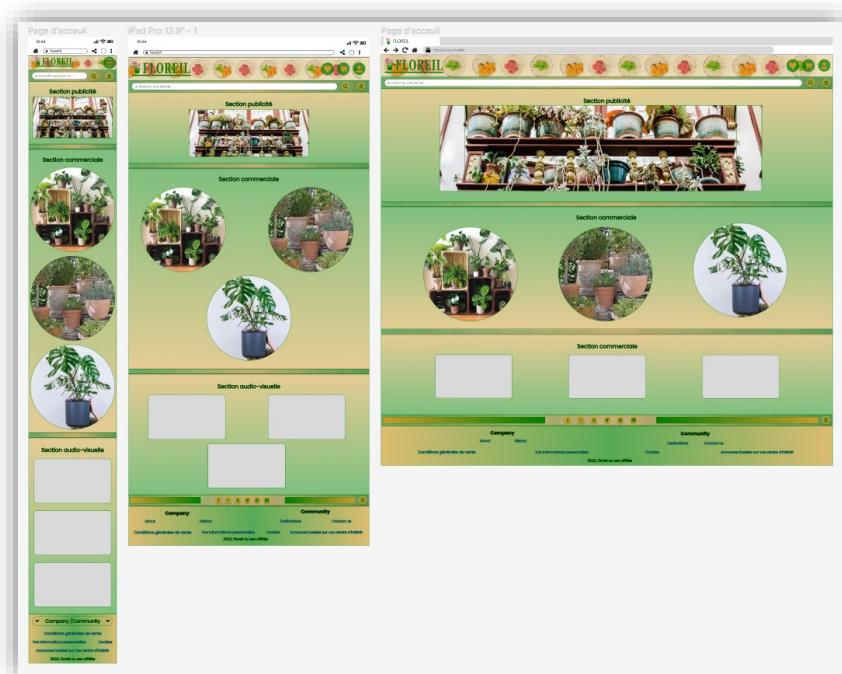


Fig.11 : La maquette au format Mobile, Tablette et Desktop

Et par la suite avec l'éditeur de code Visual Studio Code, j'ai continué le codage de la page d'accueil de mon projet, en y ajoutant les média queries pour le rendre adaptable sur tous les types de supports : Mobile, Tablette et Desktop.

Mais après j'ai décidé refaire mon projet en utilisant **Bootstrap** (un framework front-end puissant) car il a une fonctionnalité de conception adaptive que media queries. En plus ce framework a beaucoup de fonctionnalité qui me donnent possibilité d'améliorer apparence et caractéristiques de mon projet.

Grid options

Bootstrap's grid system can adapt across all six default breakpoints, and any breakpoints you customize. The six default grid tiers are as follows:

- Extra small (xs)
- Small (sm)
- Medium (md)
- Large (lg)
- Extra large (xl)
- Extra extra large (xxl)

As noted above, each of these breakpoints have their own container, unique class prefix, and modifiers. Here's how the grid changes across these breakpoints:

| | xs <576px | sm ≥576px | md ≥768px | lg ≥992px | xl ≥1200px | xxl ≥1400px |
|---|-----------------------------------|-----------------------|-----------------------|-----------------------|-----------------------|------------------------|
| Container <code>max-width</code> | None (auto) | 540px | 720px | 960px | 1140px | 1320px |
| Class prefix | <code>.col-</code> | <code>.col-sm-</code> | <code>.col-md-</code> | <code>.col-lg-</code> | <code>.col-xl-</code> | <code>.col-xxl-</code> |
| # of columns | 12 | | | | | |
| Gutter width | 1.5rem (.75rem on left and right) | | | | | |
| Custom gutters | Yes | | | | | |
| Nestable | Yes | | | | | |
| Column ordering | Yes | | | | | |

Fig.12 Le système de grille de Bootstrap





Fig.13 Page d'accueil illustrant les média quérés, Mobile, Tablette et Desktop

J'ai intégré du contenu multimédia, exemple une vidéo provenant de «Youtube» pour accentuer la visibilité de mon site dans les moteurs de recherches par les utilisateurs.

```
<section id="video_section">
    <div class="container">
        <div class="row">
            <div class="col-12">
                <h2 class="text-center fw-bold">Section audio-visuelle</h2>
            </div>
        </div>
        <div class="row justify-content-evenly">
            <div class="col-12 col-md-6 col-lg-4 mt-4 mt-lg-0">
                <div class="embed-container rounded-4">
                    <iframe title='1' src="https://www.youtube.com/embed/8ZXddFPscfa" frameborder='1' allowfullscreen></iframe>
                </div>
            </div>
            <div class="col-12 col-md-6 col-lg-4 mt-4 mt-lg-0">
                <div class="embed-container rounded-4">
                    <iframe title='2' src="https://www.youtube.com/embed/oFeQ8eDhtag" frameborder='2' allowfullscreen></iframe>
                </div>
            </div>
            <div class="col-12 col-md-6 col-lg-4 mt-4 mt-lg-0">
                <div class="embed-container rounded-4">
                    <iframe title='3' src="https://www.youtube.com/embed/cz0NbjmJyc" frameborder='3' allowfullscreen></iframe>
                </div>
            </div>
        </div>
    </div>
</section>
```



Fig.14 Code de la video youtube et section audio-visuelle de la page d'accueil

J'ai aussi intégré un **menu hamburger** à mon bandeau de navigation (*nav-bar*), pour faciliter l'accès aux informations disponibles sur le site.



Fig.15 Menu hamburger

J'ai également intégré des icônes cliquables dans mon bandeau de pied de page (*footer*), qui redirigent vers les **réseaux sociaux** du type : Facebook, Twitter, Instagram, Telegram, VK et Youtube pour offrir une plus grande visibilité à ce projet sur le web.

```

<div id="social" class="col-12 d-flex gap-1 gap-sm-3">
    <div class="floreil_brun_separator separator_adjustable grad-to-left h-75 align-self-center mx-auto mx-sm-0 ms-0 ms-sm-auto"></div>
    [% for icon in [ %]
        {
            "link": "https://www.facebook.com/",
            "icon": "fa fa-brands fa-facebook-f"
        },
        {
            "link": "https://www.instagram.com/",
            "icon": "fa fa-brands fa-instagram"
        },
        {
            "link": "https://vk.com/",
            "icon": "fa fa-brands fa-vk"
        },
        {
            "link": "https://twitter.com/",
            "icon": "fa fa-brands fa-twitter"
        },
        {
            "link": "https://web.telegram.org/",
            "icon": "fa fa-brands fa-telegram"
        },
        {
            "link": "https://www.youtube.com/",
            "icon": "fa fa-brands fa-youtube"
        }
    ] %
    <a href="{{ icon.link }}" target="_blank" rel="noreferrer" class="social_links d-flex justify-content-center align-self-center rounded-circle mx-auto mx-sm-0">
        <button class="social_buttons d-flex justify-content-center align-self-center btn btn-social-media rounded-circle flex-shrink-0 icon_resize" type="submit">
            | <i class="{{ icon.icon }} social_icons fa-lg align-self-center" aria-hidden="true"></i>
            </button>
        </a>
    (% endfor %)

```

Fig.27 Code icônes réseaux sociaux



Fig.16 Icône réseaux sociaux

2.3.3 Competence n°3 : Développer une interface utilisateur web dynamique

Le développement **d'une interface utilisateur web dynamique** est un aspect crucial de la conception de sites web modernes et conviviaux. Les utilisateurs s'attendent à des sites web réactifs et interactifs qui leur permettent de naviguer facilement et efficacement. Pour répondre à ces attentes, il est

important de développer des interfaces utilisateur qui sont visuellement attrayantes, intuitives et engageantes.

Pour développer une interface utilisateur web dynamique, vous devez avoir une bonne compréhension de **HTML**, **CSS** et **JavaScript**. Ces langages sont essentiels pour créer des pages web dynamiques et interactives. Vous pouvez utiliser une variété d'outils et de frameworks pour simplifier le processus de développement, tels que **React**, **Vue.js** ou **Angular**.

Pour la première partie de cette compétence, toujours à partir de l'éditeur de code **Visual Studio Code** j'ai utilisé **Bootstrap** qui est un framework qui permet de simplifier l'action du développeur web en lui donnant accès à une multitude de codes déjà prêts à être intégrés à son projet pour la navigation et les éléments interactifs de celui-ci.

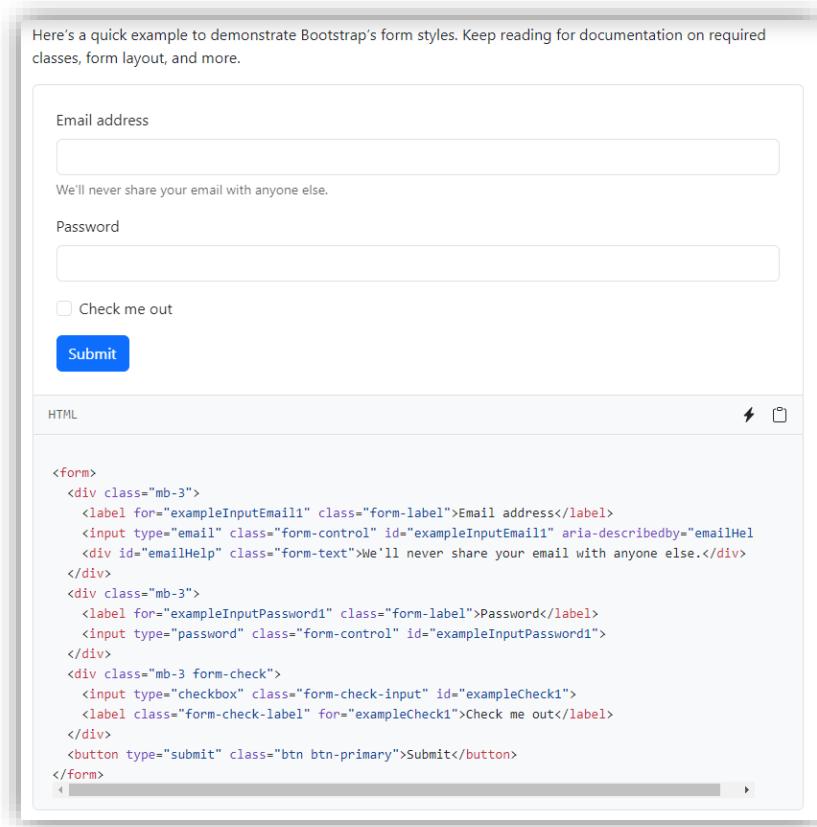


Fig.17 Un exemple de code pour une formulaire

Cela m'a permis, entre autres, de rendre mon projet dynamique en y intégrant un formulaire de "sign in" interactif à partir de **Bootstrap**. Voici un exemple du code que j'ai intégré sur la page "sign in" de mon projet, après l'avoir récupéré sur la documentation du site de **Bootstrap**

```

<section id="login_section">
    <div class="container">
        <div class="row">
            <div class="col-12">
                <h3 class="fw-bold text-center">Sign in</h3>
            </div>
        </div>
        <div class="row">
            <div class="col-12 ">
                <div id="login">
                    <form action="{{ path('app_login') }}" method="post">
                        <label for="username" class="col-form-label">Email:</label>
                        <input type="text" class="form-control rounded-pill" id="username" name="_username" value="{{ last_username }}>

                        <label for="password" class="col-form-label">Mot de passe:</label>
                        <input type="password" class="form-control rounded-pill" id="password" name="_password">
                        <div>Mot de passe oublié?
                            |   <a href="{{ path('app_request_reset_pass') }}">Cliquez ici!</a>
                        </div>
                        <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}>
                        <input type="hidden" name="_target_path" value="{{ path('app_main') }}>
                        <button type="submit" class="btn btn-body fw-bold text-shadow rounded-pill w-100 mt-3 bodybtn_text_shadow">Sign in!</button>
                    </form>
                    <div>Besoin d'un compte?
                        |   <a href="{{ path('app_register') }}">Sign up!</a>
                    </div>
                </div>
            </div>
        </div>
    </div>
</section>

```

Fig.18 Code snippet de la formulaire



Fig.19 Section «sign in»

Pour poursuivre mon projet je lui ai intégré le framework **Symfony 6** qui utilise par défaut un moteur de templates : **Twig** pour le langage de programmation **Php**, et qui offre plusieurs fonctionnalités qui faciliteront la réalisation de mon projet. Ainsi qu'un ensemble de composants Php et d'un framework **MVC** libre écrit en Php. Il fournit également des fonctionnalités modulables et adaptables qui permettent de faciliter et d'accélérer le développement d'un site web).

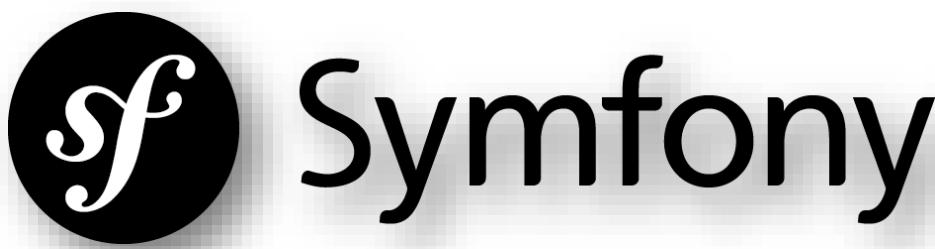


Fig .20 Logo de Symfony

Avant de créer votre première application Symfony, vous devez :

- Installer **PHP 8.1** ou une version supérieure ainsi que les extensions PHP suivantes (qui sont installées et activées par défaut dans la plupart des installations PHP 8) : Ctype, iconv, PCRE, Session, SimpleXML et Tokenizer;
- Installer **Composer** (un gestionnaire de dépendances pour PHP), qui est utilisé pour installer des packages PHP.

Composer n'est pas un gestionnaire de paquets au sens où Yum ou Apt le sont. Oui, il gère des "paquets" ou des bibliothèques, mais il les gère au niveau de chaque projet, en les installant dans un répertoire (par exemple, "vendor") à l'intérieur de votre projet. Par défaut, il n'installe rien de manière globale. C'est donc un gestionnaire de dépendances. Cependant, il prend en charge un projet "global" pour plus de commodité via la commande "global".

Cette idée n'est pas nouvelle et Composer s'inspire fortement de **npm** de node et de Bundler de Ruby.

Supposons que :

- Vous avez un projet qui dépend de plusieurs bibliothèques.

- Certaines de ces bibliothèques dépendent d'autres bibliothèques.

Composer :

- Vous permet de déclarer les bibliothèques dont vous avez besoin.
 - Découvre quelles versions de quels paquets doivent et peuvent être installées, et les installe (ce qui signifie qu'il les télécharge dans votre projet).
 - Vous pouvez mettre à jour toutes vos dépendances en une seule commande.
 - En option, vous pouvez également installer **Symfony CLI**. Cela crée un binaire appelé `symfony` qui fournit tous les outils dont vous avez besoin pour développer et exécuter votre application Symfony localement.

Le **Symfony CLI** est un outil pour les développeurs qui vous aide à construire, exécuter et gérer vos applications Symfony directement à partir de votre terminal. C'est open source, ça fonctionne sur macOS, Windows et Linux, et vous n'avez à l'installer qu'une fois sur votre système.

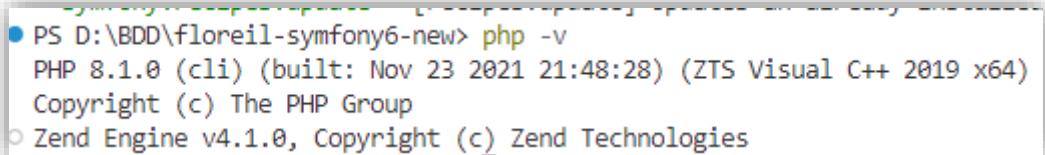
Après avoir vérifié les versions de Symfony, Composer et Php, avec les commandes : ***symfony -v, composer, php -v*** :

```
PS D:\BDD\floreil-symfony6-new> symfony -v

INFO A new Symfony CLI version is available (5.5.2, currently running 5.4.21).

If you installed the Symfony CLI via a package manager, updates are going to be automatic.
If not, upgrade by downloading the new version at https://github.com/symfony-cli/symfony-cli/releases
And replace the current binary (symfony.exe) by the new one.

Symfony CLI version 5.4.21 (c) 2021-2023 Fabien Potencier #StandWithUkraine Support Ukraine (2023-01-10T10:24:26Z - stable)
Symfony CLI helps developers manage projects, from local code to remote infrastructure
```



```
PS D:\BDD\floreil-symfony6-new> php -v
PHP 8.1.0 (cli) (built: Nov 23 2021 21:48:28) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.1.0, Copyright (c) Zend Technologies
```

Fig. 21 Les Resultats des commandes

J'ai poursuivi sur la création du projet Symfony en exécutant dans le terminal la commande :

symfony new my_project_directory --version="6.2.*" –webapp

avec laquelle je vais obtenir la structure de Symfony, basée sur une amélioration et une automatisation du **MVC** (Modèle View Controller) qui fait partie de la **POO** (Programmation orienté objet).

Voici l'architecture que l'on retrouve dans la structure de **Symfony** dans son dossier « src/ » :

- **Controller** – réceptionner une requête (triée par une route) et de définir la réponse appropriée
- **Entity** – classe qui représente la structure de nos tables et dont les attributs représentent les champs de notre table
- **Repository** – centralise tout ce qui touche à la récupération de vos entités

On a également le dossier « templates », qui se trouve dans la structure générée automatiquement par **Symfony**, dans lequel on peut stocker nos **Vue**, et se servir de celles-ci pour les créer avec le moteur de template **Twig**.

Ensuite je lance mon projet **Symfony** en tapant en ligne de commande :

symfony serve

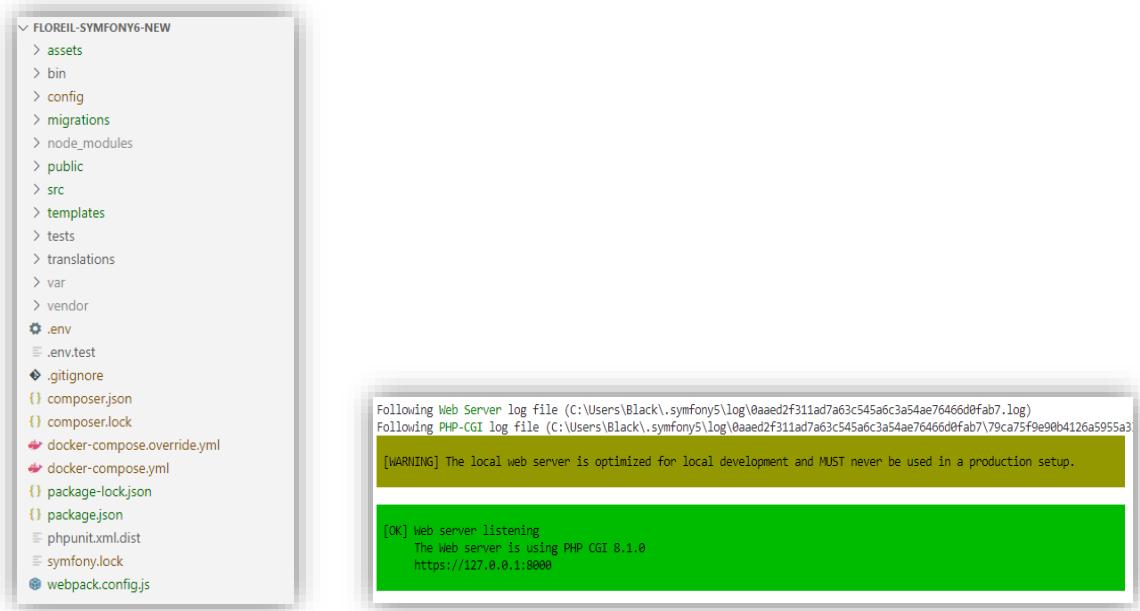


Fig. 22 La structure de Symfony obtenue dans Visual studio code

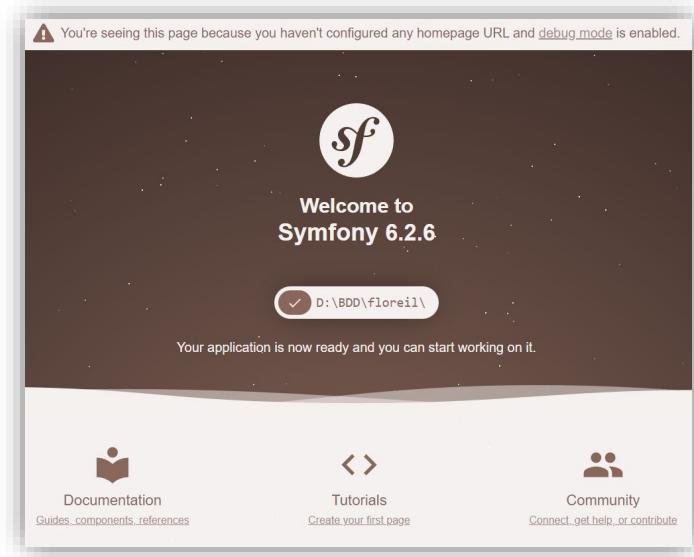


Fig. 23 Le premier démarrage du projet Symfony

Maintenant que Symfony 6 est opérationnel, j'ai débuté la réalisation du **Layout** de mon site. Pour cela, j'ai créé quelques composants dans le dossier « templates/_partials » : **_header.html.twig**, **_footer.html.twig**, **_flash.html.twig**, **_searchbar.html.twig**, **_prefooter.html.twig**, que j'ai ensuite implémentés dans le modèle de base Twig - **base.html.twig**.

Ainsi, ce **Layout** sera désormais présente sur toutes les pages de mon projet.

The screenshot shows a file structure on the left and a code editor on the right.

File Structure:

```
templates > _partials > _flash.html.twig
```

- templates
 - _partials
 - _addToCart_form.html.twig
 - _flash.html.twig
 - _footer.html.twig
 - _header.html.twig
 - _intersection.html.twig
 - _prefooter.html.twig
 - _searchbar.html.twig

Code Editor ('_flash.html.twig'):

```

1  {% for label, messages in app.flashes %}
2  {% for message in messages %}
3    <div class="alert alert-{{ label }} alert-dismissible rounded-0 mb-0" role="alert">
4      <button type="button" class="btn btn-close" data-bs-dismiss="alert" aria-label="close">
5        </button>
6        <div class="alert-message">
7          {{ message|raw }}
8        </div>
9      </div>
10     {% endfor %}
11   {% endfor %}

```

Fig. 24 La structure de dossier « `_partials` » et l'exemple de code de « `_flash` » component

The screenshot shows a code editor displaying the 'base.html.twig' template.

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>
6        {% block title %}Welcome!
7        {% endblock %}
8      </title>
9      <meta name="viewport" content="width=device-width, initial-scale=1"/>
10     <link
11       rel="icon" href="data:image/svg+xml,<svg xmlns=%22http://www.w3.org/2000/svg%22 viewBox=%220 0 128 128%22
12       %# Run `composer require symfony/webpack-encore-bundle` to start using Symfony UX #
13       {% block stylesheets %}
14         {{ encore_entry_link_tags('app') }}
15         <script defer src="https://js.stripe.com/v3/"></script>
16       {% endblock %}
17
18       {% block javascripts %}
19         {{ encore_entry_script_tags('app') }}
20       {% endblock %}
21     </head>
22     <body>
23       {% include "_partials/_header.html.twig" %}
24       {% include "_partials/_searchbar.html.twig" %}
25       {% block breadcrumb %}{% endblock %}
26       {% if app.user and app.user.isVerified == false %}
27         <div class="alert alert-warning alert-dismissible mb-0" role="alert">
28           <button type="button" class="btn btn-close" data-bs-dismiss="alert" aria-label="close"></button>
29           <div class="alert-message">
30             <strong>Votre compte n'est pas activé</strong>,
31             <a href="{{ path('app_resend_verify') }}">envoyer le lien d'activation</a>
32           </div>
33         </div>
34       {% endif %}
35       {% include "_partials/_flash.html.twig" %}
36       {% block body %}{% endblock %}
37       {% include "_partials/_prefooter.html.twig" %}
38       {% include "_partials/_footer.html.twig" %}
39     </body>
40   </html>

```

Fig. 25 base.html.twig

J'ai continué avec la réalisation du page d'accueil, avec ce Framework.

Pour commencer je me suis servi de la commande : **symfony console make:controller**, ce qui permet à Symfony de générer un Controller auquel nous allons donner un nom : (**MainController**), son rôle sera de retourner une Vue grâce à la Route que nous lui donnerons :(« / ») dans le dossier Template.

```
src > Controller > MainController.php > ...
1  <?php
2
3  namespace App\Controller;
4
5  use App\Repository\CategoryRepository;
6  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
7  use Symfony\Component\HttpFoundation\Response;
8  use Symfony\Component\Routing\Annotation\Route;
9
10 class MainController extends AbstractController
11 {
12     #[Route('/', name: 'app_main')]
13     public function index(CategoryRepository $categories): Response
14     {
15         return $this->render('main/index.html.twig', [
16             'user' => $this->getUser(),
17             'categories' => $categories->findBy([]),
18         ]);
19     }
20 }
21
```

Fig. 26 Le code php dans MainController.php

```
templates > main > index.html.twig
1  {% extends 'base.html.twig' %}
2
3  {% block title %}Floreil - Accueil
4  {% endblock %}
5
6  {% block body %}
7      {% include "main/_carousel.html.twig" %}
8      {% include "_partials/_intersection.html.twig" %}
9      {% include "main/_categories.html.twig" %}
10     {% include "_partials/_intersection.html.twig" %}
11     {% include "main/_video.html.twig" %}
12  {% endblock %}
13
```

Fig. 27 Dossier « main » et son fichier index.html.twig

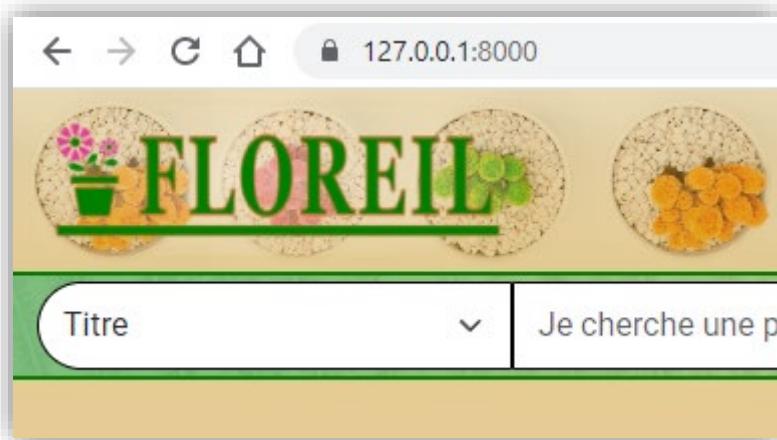


Fig. 28 Renseignement de la route dans la barre d'adresse

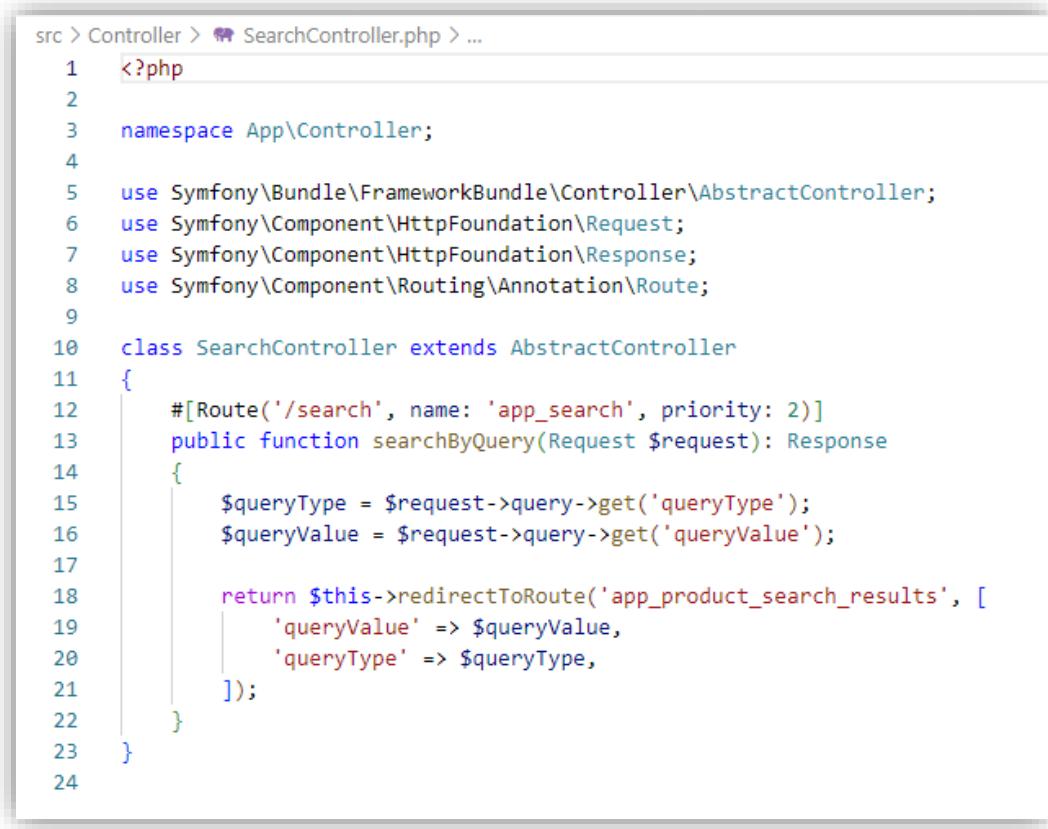
On peut également se servir de **Twig**, qui est un moteur de Templates moderne pour **Php**, il compile les templates de façon à optimiser le code Php. Il dispose également d'un mode Sandbox pour évaluer le code des templates non fiable. Pour sa **Fléxibilité** il est alimenté par un Lexer et un Parser, cela permet au développeur de définir ses propres balises et filtres et de créer son propre **DSL**.

DSL (Domain-Specific Language) est un langage de programmation spécialisé pour un domaine spécifique ou une tâche de domaine. Dans ce contexte, cela signifie que **Twig** permet aux développeurs de créer leurs propres langages de programmation spécialisés pour des tâches et des domaines spécifiques. Cela se fait en définissant des bibliothèques de filtres et de fonctions personnalisées, ainsi qu'en créant des balises et des blocs personnalisés qui peuvent être utilisés pour définir des constructions syntaxiques uniques à l'intérieur des modèles. Ainsi, **Twig** offre la possibilité de créer son propre **DSL** pour résoudre des tâches spécifiques de manière plus efficace et pratique.

Pour se faire nous allons appeler le templates de base qui correspond pour ma part au header, body et footer qui se trouve dans le fichier : (**base.html.twig**), et que l'on souhaite étendre sur chaque page de notre site en se servant du Child Template avec La commande : { % extends 'base.html.twig' %} que l'on place en haut du fichier : (**index.html.twig**) souhaité. On utilise l'accolade pourcentage pour appeler une structure de contrôle.

En tant que l'exemple on peut reviser *fig.25* et *fig.27* où se trouve des structures de mon **base.html.twig** et **main/index.html.twig** , qui est Child template de **base.html.twig**.

Pour finaliser mon **Layout**, j'ai créé un contrôleur et un template Twig pour searchbar.



```

src > Controller > SearchController.php > ...
1  <?php
2
3  namespace App\Controller;
4
5  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
6  use Symfony\Component\HttpFoundation\Request;
7  use Symfony\Component\HttpFoundation\Response;
8  use Symfony\Component\Routing\Annotation\Route;
9
10 class SearchController extends AbstractController
11 {
12     #[Route('/search', name: 'app_search', priority: 2)]
13     public function searchByQuery(Request $request): Response
14     {
15         $queryType = $request->query->get('queryType');
16         $queryValue = $request->query->get('queryValue');
17
18         return $this->redirectToRoute('app_product_search_results', [
19             'queryValue' => $queryValue,
20             'queryType' => $queryType,
21         ]);
22     }
23 }
24

```



```

templates > _partials > _searchbar.html.twig
1  <section id="search_section" class="sticky-top">
2      <div class="container-fluid">
3          <div class="row">
4              <div class="col-12 d-flex gap-2 navbar-expand-sm justify-content-end">
5                  <div class="collapse navbar-collapse w-100" id="searchGroup">
6                      <form method="get" action="{{ path('app_search') }}" class="d-flex flex-fill align-self-center flex-column flex-sm-row" id="form">
7                          <select name="queryType" class="form-select order-2 order-sm-1 input_type" aria-label="Default select example">
8                              <option value="title">titre</option>
9                              <option value="commonName">Nom Commun</option>
10                             <option value="genre">Genre</option>
11                         </select>
12                         <input name="queryValue" class="form-control order-1 order-sm-2 me-1 input_search" type="search" placeholder="Je cherche une planète..." aria-label="Search input" />
13                         <button class="btn btn-additional order-3 align-self-center search_button" for="search" type="submit">
14                             <i class="fa fa-search fa-lg align-self-center" aria-hidden="true"></i>
15                         </button>
16                     </form>
17                 </div>
18
19                 <div class="floreil_brun_separator h-75 align-self-center"></div>
20                 <a href="#" class="btn btn-additional switch_button d-flex justify-content-center align-self-center">
21                     <i class="fa fa-angle-double-down fa-lg align-self-center" aria-hidden="true"></i>
22                 </a>
23                 <button type="button" class="btn btn-additional align-self-center collapse_button d-sm-none dropdown-toggle triangle-sup" data-bs-toggle="dropdown">
24                     <i class="fa fa-angle-down fa-lg align-self-center" aria-hidden="true"></i>
25                 </button>
26             </div>
27         </div>

```

Fig. 29 SearchController.php et _searchbar.html.twig

Dans ce cas, j'ai utilisé un formulaire HTML car il est assez simple. Cependant, Symfony 6 dispose d'une fonctionnalité distincte pour la création et la gestion de formulaires, que j'ai utilisée pour créer le formulaire d'inscription .

Pour créer un **formulaire** dans Symfony 6, il faut entrer la commande console "**symfony console make:form**" et indiquer l'entité avec laquelle le formulaire sera associé.

```
class RegistrationFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('email', EmailType::class)
            ->add('agreeTerms', CheckboxType::class, [
                'mapped' => false,
                'constraints' => [
                    new IsTrue([
                        'message' => 'You should agree to our terms.',
                    ]),
                ],
            ])
            ->add('plainPassword', PasswordType::class, [
                // instead of being set onto the object directly,
                // this is read and encoded in the controller
                'mapped' => false,
                'attr' => ['autocomplete' => 'new-password'],
                'constraints' => [
                    new NotBlank([
                        'message' => 'Please enter a password',
                    ]),
                    new Length([
                        'min' => 6,
                        'minMessage' => 'Your password should be at least {{ limit }} characters',
                        // max length allowed by Symfony for security reasons
                        'max' => 4096,
                    ]),
                ],
            ])
            ->add('surname', TextType::class)
            ->add('name', TextType::class)
    }
}
```

```
public function configureOptions(OptionsResolver $resolver): void
{
    $resolver->setDefaults([
        'data_class' => User::class,
    ]);
}
```

Fig. 30 Fragment de code de RefistrationForm

Ensuite, le formulaire créé peut être plus profondément configuré dans son fichier PHP en définissant les contraintes et les vérifications nécessaires pour les champs de saisie.

Ensuite, le formulaire est envoyé au **contrôleur**, d'où il est rendu dans le **template Twig** dans sa forme finale. La configuration supplémentaire du formulaire est également possible dans le template Twig. En réalité, nous pouvons choisir où il est plus pratique pour nous de configurer le formulaire.

```
class RegistrationController extends AbstractController
{
    #[Route('/register', name: 'app_register', priority: 2)]
    public function register(
        Request $request,
        UserPasswordHasherInterface $userPasswordHasher,
        EntityManagerInterface $entityManager,
        JwtService $jwt,
        SendMailService $mail,
        UserAuthenticatorInterface $userAuthenticator,
        UserAuthenticator $authenticator
    ): Response {
        $user = new User();
        $form = $this->createForm(RegistrationFormType::class, $user);
        $form->handleRequest($request);
    }
}
```

Fig. 31 Fragment de code de RefistrationController

```
<div class="row">
    {{ form_start(registrationForm, {'attr': {'class': 'row'}}) }}
    <h3 class="col-12">Information principale</h3>
    <div class="col-12 col-md-6">
        {{ form_label(registrationForm.email, 'Email', {'label_attr': {'class': 'col-form-label'}}) }}
        {{ form_widget(registrationForm.email, {'attr': {'class': 'form-control rounded-pill', 'required': 'true'}}) }}
    </div>
    <div class="col-12 col-md-6">
        {{ form_label(registrationForm_plainPassword, 'Mot de passe', {'label_attr': {'class': 'col-form-label'}}) }}
        {{ form_widget(registrationForm_plainPassword, {'attr': {'class': 'form-control rounded-pill', 'required': 'true'}}) }}
    </div>
    <div class="col-8 form-check ms-3">
        {{ form_label(registrationForm_agreeTerms, 'Accepter les termes ', {'label_attr': {'class': 'form-check-label'}}) }}
        {{ form_widget(registrationForm_agreeTerms, {'attr': {'class': 'form-check-input rounded-1', 'required': 'true'}}) }}
    </div>
    <div class="border border-bottom my-3 border-success"></div>
    <h3 class="col-12">Coordonnées</h3>
    <div class="col-12 col-md-6 col-lg-4">
        {{ form_label(registrationForm_surname, 'Nom', {'label_attr': {'class': 'col-form-label'}}) }}
        {{ form_widget(registrationForm_surname, {'attr': {'class': 'form-control rounded-pill', 'required': 'true'}}) }}
    </div>
    <div class="col-12 col-md-6 col-lg-4">
        {{ form_label(registrationForm_name, 'Prenom', {'label_attr': {'class': 'col-form-label'}}) }}
        {{ form_widget(registrationForm_name, {'attr': {'class': 'form-control rounded-pill', 'required': 'true'}}) }}
    </div>
    <div class="col-12 col-md-6 col-lg-4">
        {{ form_label(registrationForm_mobTel, 'Téléphone portable', {'label_attr': {'class': 'col-form-label'}}) }}
        {{ form_widget(registrationForm_mobTel, {'attr': {'class': 'form-control rounded-pill', 'required': 'true'}}) }}
    </div>
    <div class="col-12 col-md-6 col-lg-2">
        {{ form_label(registrationForm_codeZIP, 'Code postale', {'label_attr': {'class': 'col-form-label'}}) }}
        {{ form_widget(registrationForm_codeZIP, {'attr': {'class': 'form-control rounded-pill', 'required': 'true', 'minlength': 5}}) }}
        <p id="errorZIP" class="d-none text-danger">Votre code postale est incorrect!</p>
    </div>
</div>
```

Fig. 32 Fragment de code de register.html.twig

En plus d'utiliser la fonctionnalité Symfony pour créer un formulaire d'inscription utilisateur, j'ai également utilisé **l'API géographique** du gouvernement français (<https://geo.api.gouv.fr>) pour le formulaire d'inscription. Cette API permet d'obtenir des informations sur les villes, les départements, les régions de France et d'autres informations en fonction de la requête. Dans mon cas, j'utilise cette API pour déterminer la ville et le département de l'utilisateur à partir de son code postal.

The screenshot shows two main sections of the GeoAPI documentation:

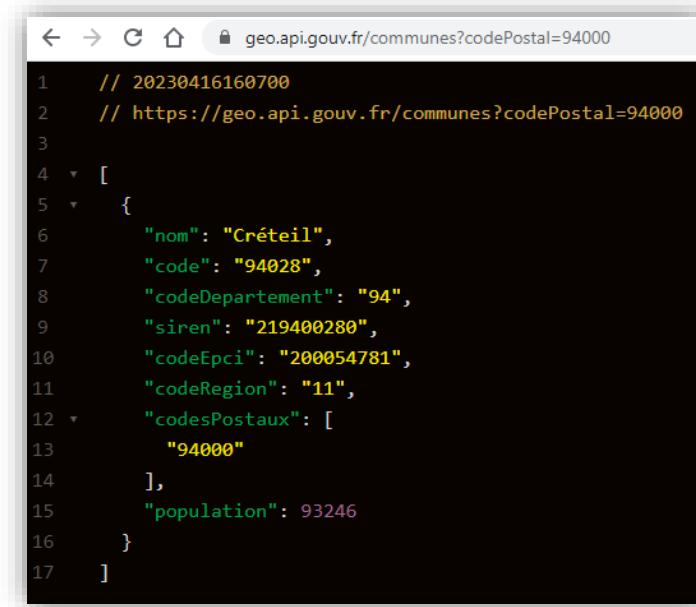
- API Découpage administratif > Communes**: This section includes a search bar and five search options:
 - Recherche par code postal (Icon: envelope)
 - Recherche par nom (Icon: tag)
 - Liste de communes (Icon: list)
 - Recherche géographique (Icon: location)
 - Recherche avancée (Icon: search)
- Documentation technique**: This section provides detailed API documentation for the /communes endpoint:
 - GET /communes**: Rechercher des communes
 - GET /communes/{code}**: Récupérer les informations concernant une commune
 - GET /epcis/{code}/communes**: Renvoi les communes d'un EPCI
 - GET /departements/{code}/communes**: Renvoi les communes d'un département
 Below this, there is a table titled "Attributs" listing the attributes of a commune:

| Nom | Description | Type |
|-----------------|---|--------|
| code | Code (INSEE) de la commune | string |
| nom | Nom de la commune | string |
| siren | Code SIREN de la commune | string |
| codesPostaux | Liste des codes postaux de la commune | array |
| codeEPCI | Code de l'EPCI associé | string |
| codeDepartement | Code du département associé | string |
| codeRegion | Code de la région associée | string |
| population | Nombre d'habitants de la commune | number |
| _score | Score de correspondance allant de 0 à 1 | number |

Fig. 33 Page de GeoAPI

Pour ce faire, j'ai ajouté des **listeners** à l'entrée de code postal et au nom de la ville avec **JavaScript** et défini des conditions pour lesquelles une requête serait envoyée via l'API géographique lors de la saisie du code postal. Si une correspondance est trouvée dans la base de données, le nom de la ville et le

nom du département du gouvernement français seront ajoutés au champ de saisie de la ville.



```
// 20230416160700
// https://geo.api.gouv.fr/communes?codePostal=94000
[
  {
    "nom": "Créteil",
    "code": "94028",
    "codeDepartement": "94",
    "siren": "219400280",
    "codeEPCI": "200054781",
    "codeRegion": "11",
    "codesPostaux": [
      "94000"
    ],
    "population": 93246
  }
]
```



```
const zipInput = document.getElementById("registration_form_codeZIP");
const cityInput = document.getElementById("registration_form_city");
const flashMessage = document.getElementById("errorZIP")
zipInput.addEventListener("input", async () => {
  flashMessage.classList.add('d-none')
  if (zipInput.value.length === 5) {
    try {
      const response = await fetch(
        `https://geo.api.gouv.fr/communes?codePostal=${zipInput.value}&fields=nom,departement`
      );
      const data = await response.json();
      // console.log(data)
      if (data.length !== 0) {
        const city = data[0].nom;
        const departement = data[0].departement.nom;
        cityInput.value = city + ", " + departement;
        // console.log(cityInput.value)
      } else {
        cityInput.value = "";
        flashMessage.classList.remove('d-none')
      }
    } catch (error) {
      console.error(`Error fetching city for zip code ${zipInput.value}:`, error)
    }
  } else {
    cityInput.value = "";
  }
});
```

Fig. 34 L'exemple de req/res de l'API et code js

| | |
|-------------|-----------------------|
| Code postal | Ville |
| 94000 | Créteil, Val-de-Marne |

| | |
|-------------|------------------------------|
| Code postal | Ville |
| 94005 | Saisissez code postal valide |

Votre code postal est incorrect!

Fig. 35 L'exemple d'execution de code

Si l'utilisateur a saisi un code postal incorrect, il recevra une notification sous forme de message flash.

En plus de cela, pour la vérification de l'adresse e-mail de l'utilisateur, j'utilise la fonctionnalité de la bibliothèque **symfony-mailer** en combinaison avec le service de test de réception de courrier **schickling/mailcatcher**, que je lance via **Docker**.

```
###> symfony/mailers ###
MAILER_DSN=smtp://0.0.0.0:1025
###< symfony/mailers ###
```

```
docker-compose.yml
version: '3.8'
services:
  mailer:
    image: schickling/mailcatcher
    ports:
      - 1080:1080
      - 1025:1025
```

Fig. 36 Fragment de code .env et docker-composer.yml

Docker est une plateforme open-source de développement, de livraison et d'exécution d'applications. Elle permet d'emballer une application avec toutes ses dépendances dans un conteneur, qui pourra être exécuté sur n'importe quel

environnement. Cela permet une grande portabilité et une gestion simplifiée des applications, ainsi qu'une isolation efficace des processus.

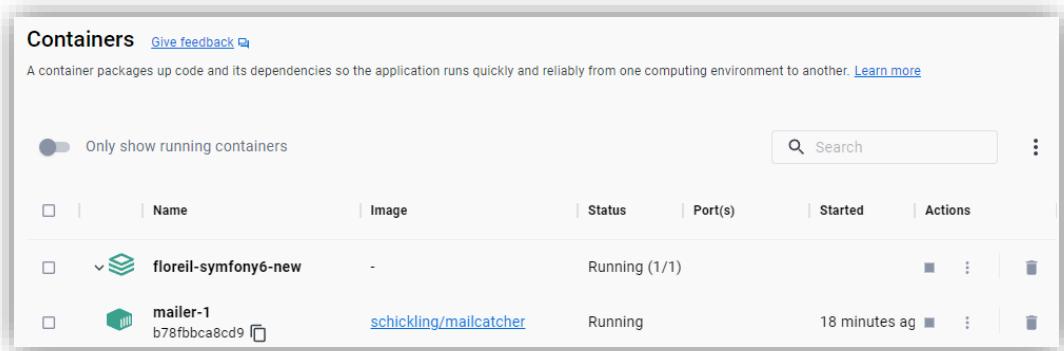


Fig. 37 Docker conteneur de schickling/mailcatcher

Le processus de **vérification** se déroule comme suit. Dans le **RegistrationController**, après avoir rempli et validé le formulaire, le système vérifie la conformité des informations saisies avec les modèles de données de l'entité **User** et, si tout est correct, crée ensuite un **e-mail** avec un **jeton JWT** généré à partir de l'id et de l'adresse e-mail de l'utilisateur (également en utilisant une **clé secrète** de projet unique définie dans le fichier **.env**), puis l'envoie à l'adresse e-mail spécifiée.

```
//generating users JWT
//creating Header
$header = [
    'typ' => 'JWT',
    'alg' => 'HS256'
];
//creating Payload
	payload = [
        'user_id' => $user->getId(),
        'user_email' => $user->getEmail(),
    ];
//generating token
$token = $jwt->generate($header, $payload, $this->getParameter('app.jwtsecret'));

$mail->send(
    'no-reply@floreil.net',
    $user->getEmail(),
    'Activation votre compte sur le site e-commerce',
    'register',
    [
        'user' => $user,
        'token' => $token
    ]
);
```

Fig. 38 Fragment de code de RegistrationController concernant création et envoi une lettre

L'utilisateur reçoit cet e-mail et clique sur le lien qui inclut le jeton précédemment généré, qui mène au contrôleur de confirmation de l'e-mail.

```
'use strict';
#[IsGranted('ROLE_USER')]
#[Route('/verify/{token}', name: 'app_verify_email', priority: 2)]
public function verifyUserEmail(
    $token,
    JWTService $jwt,
    UserRepository $users,
    EntityManagerInterface $entityManager,
): Response {
    $this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');
    if ($jwt->isValid($token) && !$jwt->isExpired($token) && $jwt->checkSignature($token, $this->getParameter('app.jwtsecret'))) {
        $payload = $jwt->getPayload($token);
        $user = $users->find($payload['user_id']);
    } else {
        $this->addFlash('danger', "Le token est invalide ou a expiré");
        return $this->redirectToRoute('app_login');
    }

    if ($user && !$user->isVerified()) {
        $user->setIsVerified(true);
        $entityManager->flush($user);
        $this->addFlash('success', "Votre compte est activé!");
        return $this->redirectToRoute('app_main');
    }

    //if token isn't valid or changed
    $this->addFlash('danger', "Le token est invalide ou a expiré");
    return $this->redirectToRoute('app_login');
}
```

Fig. 39 Fragment de code de RegistrationController concernant vérification de jeton JWT

Dans celui-ci, le jeton est vérifié pour son format **JWT**, sa signature correspondant à la clé secrète de projet unique et l'id utilisateur du jeton est comparé à ceux de la base de données. Si un tel utilisateur existe, il se voit attribuer le statut "vérifié".



Fig. 40 Schickling/mailcatcher

Le jeton **JWT** (**JSON Web Token**) est un format compact, autoportant et sécurisé pour représenter des informations échangeables entre des parties en tant qu'objet **JSON**. Ce jeton est généralement utilisé pour authentifier les utilisateurs et échanger des informations en toute sécurité entre différents systèmes. Il est composé de trois parties: une entête qui décrit le type de jeton et l'algorithme de chiffrement utilisé, une charge utile qui contient les informations nécessaires à l'application et une signature qui permet de vérifier l'intégrité du jeton. Le **JWT** est souvent utilisé dans les applications Web modernes pour gérer l'authentification et l'autorisation des utilisateurs.



Fig. 41 Message flash

Dans le cas où l'utilisateur n'a pas reçu l'e-mail avec le jeton, il peut toujours demander un nouvel envoi, car tant qu'il n'est pas vérifié, il verra toujours un message flash avec un lien pour renvoyer l'e-mail.

Un autre sujet sur lequel je voulais m'arrêter est l'intégration d'un **système de paiement en ligne** pour les achats. Pour tout projet de commerce électronique, cela est extrêmement important. Pour fournir cette fonctionnalité dans mon projet, j'ai décidé d'utiliser le service **Stripe**.



Fig. 42 Page d'accueil de Stripe

Stripe est une plateforme de paiement qui permet d'intégrer rapidement et facilement les paiements en ligne sur votre site Web ou votre application mobile. Stripe fournit un ensemble d'outils pour le traitement des paiements, y compris des API, des bibliothèques pour différents langages de programmation, des plugins pour différentes plateformes CMS et de nombreux autres outils. Stripe assure également un niveau élevé de sécurité des paiements, notamment la protection contre la fraude et la conformité aux normes de sécurité des cartes de paiement (**PCI DSS**). Stripe est l'un des services de paiement en ligne les plus populaires et fiables au monde.

Tout d'abord, il est nécessaire d'installer la bibliothèque "**stripe/stripe-php**". Il est également nécessaire d'avoir un compte **Stripe** et une paire de clés (**clé publique** et **clé secrète**) pour assurer le traitement correct des paiements, que nous stockerons dans le fichier **.env**.

Ensuite, nous créons un service dans notre projet que nous utiliserons ultérieurement dans notre contrôleur de paiement.

Pour le côté client, j'ai décidé d'utiliser **Stripe.js**.

Stripe.js est une bibliothèque client fournie par le système de paiement Stripe, qui permet d'intégrer un formulaire de paiement sécurisé directement sur votre site web. Stripe.js assure la sécurité des paiements, la protection contre la fraude et la conformité aux normes de sécurité des cartes de paiement (**PCI DSS**), ainsi qu'une interface simple et conviviale pour interagir avec les éléments du formulaire de paiement. Avec **Stripe.js**, vous pouvez créer et traiter des transactions, y compris payer des abonnements, effectuer des paiements récurrents, etc.



```

templates > cart > index.html.twig
  51   <div class="col-12 col-md-3 h5 fw-bold text-center">
  52     <p>Sous-total, €:<br/>{{ totalPrice }}</p>
  53     <p>{{ userCartProducts|length }}<br/>article(s)</p>
  54   <form action="{{ path('app_checkout', { 'amount': totalPrice }) }}" method="POST">
  55     <script>
  56       src="https://checkout.stripe.com/checkout.js" class="stripe-button "
  57       data-key="{{ stripe_public_key }}"
  58       data-amount="{{ totalPrice * 100 }}"
  59       data-name="Floreil"
  60       data-description="Widget"
  61       data-image="/images/logos/Logo_noname.png"
  62       data-locale="auto"
  63     </script>
  64   </form>
  65
  66

```

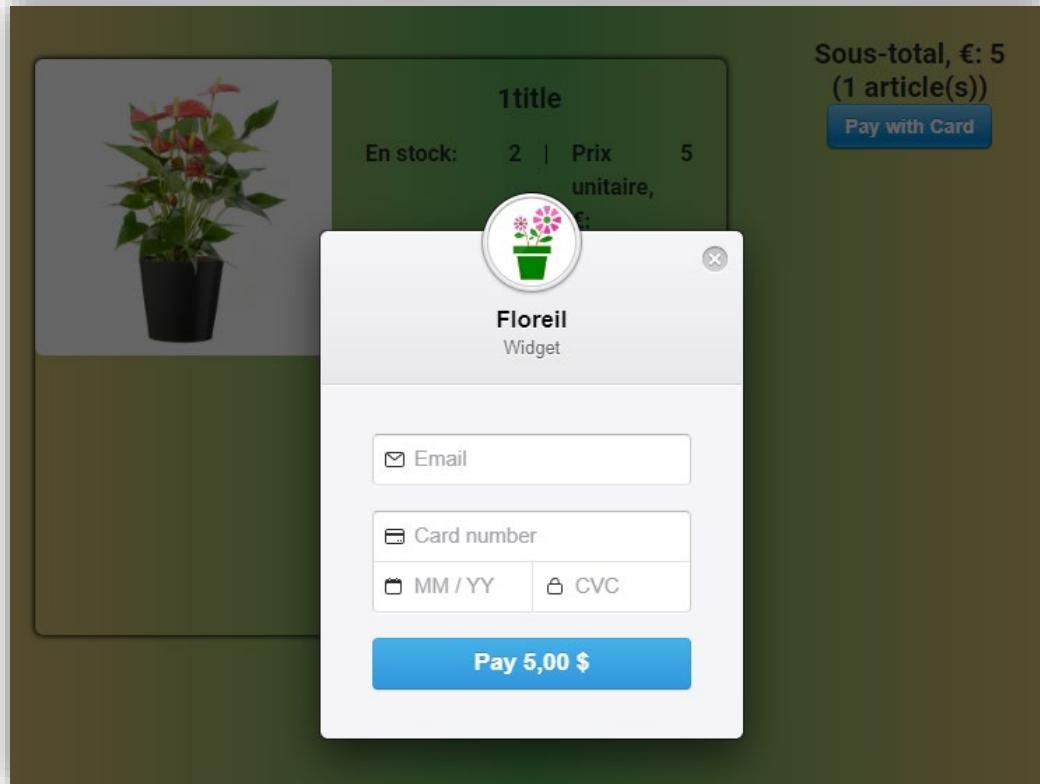


Fig. 43 Le code et affichage de Stripe.js

Ainsi, la séquence d'actions est la suivante. Le client décide de procéder au paiement dans son panier et utilise le formulaire **stripe.js** pour saisir les informations de sa carte et appuyer sur le bouton Envoyer. Ensuite, le formulaire envoie une demande au contrôleur de paiement (**CheckoutController**), où une vérification finale de la validité de la quantité d'articles est effectuée. S'il y a des erreurs, le paiement est interrompu et l'utilisateur est redirigé vers le panier avec un message flash correspondant.

```

if (count($errors) > 0 || $invalidQty) {
    // dd($errors, $invalidQty);
    $this->addFlash('danger', 'La quantité des produits est insuffisante. Le payment est annulé. Vérifiez la quantité des produits.');
    $this->redirectToRoute('app_cart', [
        'stripe_public_key' => $stripePublicKey,
        'amount' => $amount * 100,
    ]);
} else {
    if ($request->isMethod('POST')) {
        $token = $request->get('stripeToken');
        $stripe = new StripeClient($this->getParameter('stripe_api_key'));
        $stripe->charges->create([
            'amount' => $amount * 100,
            'currency' => $currency,
            'source' => $token,
            'description' => 'My First Test Charge (created for API docs at https://www.stripe.com/docs/api)',
        ]);

        $this->addFlash('success', 'Your payment goes well for now');
        return $this->redirectToRoute('app_cart_validate', [
            'price' => $amount
        ]);
    }
}

```

Fig. 44 Fragment de code CheckoutController

Si tout est en ordre, une instance de **StripeClient** est créée, dans laquelle les informations de paiement (montant, devise, description, source) sont enregistrées et envoyées sous forme de requête à **l'API Stripe**, et le méthode de validation du panier de **CartController** est activé, où la mise à jour des articles et la vidange du panier sont effectuées, ainsi qu'un rapport sur cet achat est généré et enregistré dans la base de données.

Ces informations de paiement sont stockées dans le compte **Stripe**. Il est possible de consulter l'historique des paiements effectués, les montants et les dates de ces paiements, ainsi que d'autres informations connexes. Il est également possible de vérifier les paiements qui ont échoué et la raison de cet échec.

2.4. Activité n°2 : Développer La Partie Back-End

2.4.1 Competences n°5 : Créer une base de données

Pour la partie **Back-end**, j'ai commencé par comprendre le fonctionnement d'un site web et savoir faire la différence entre la partie clients et la partie serveurs, j'ai également appris à différencier un site **Statique** d'un site **Dynamique**.

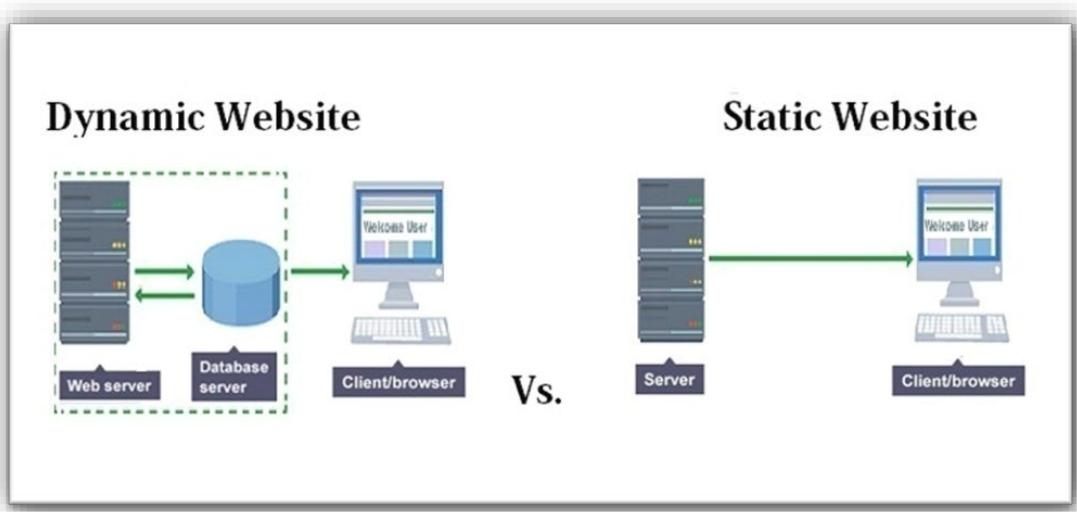


Fig. 45 La différence entre un site statique et un site dynamique

Sur un site **Statique** le serveur stocke des pages web et les envoie aux clients qui les demandent sans les modifier. Alors que sur un site **Dynamique** la page web est générée à chaque fois qu'un client la réclame, cela rend son contenu vivant il peut changer d'un instant à l'autre. Tout cela est possible grâce au langage PHP et au **SGDBr** (système de gestion de base de données relationnelle) **MySql**.

J'ai commencé création de ma base de données de création de **Model Concepteur de Données** (MCD) en utilisant **méthode Merise**. Pour réaliser ça, on peut utiliser des applications spécialisées - **JMerise** ou **AnalyseSI**. J'ai choisi AnalyseSI.

AnalyseSI est un outil logiciel gratuit et open-source pour la conception et la modélisation de systèmes d'information. C'est un outil de modélisation visuelle qui prend en charge une gamme de techniques de modélisation, y compris les **diagrammes Entité-Relation** (ER), les **diagrammes UML**.

AnalyseSI permet aux utilisateurs de créer des modèles de systèmes d'information, y compris des bases de données, des processus métier et des applications logicielles. Il fournit une interface graphique pour créer et modifier des modèles, ainsi que des fonctionnalités pour générer de la documentation et du code à partir des modèles. Le logiciel inclut également des outils pour effectuer des vérifications de cohérence, générer des rapports et exporter des modèles dans différents formats.

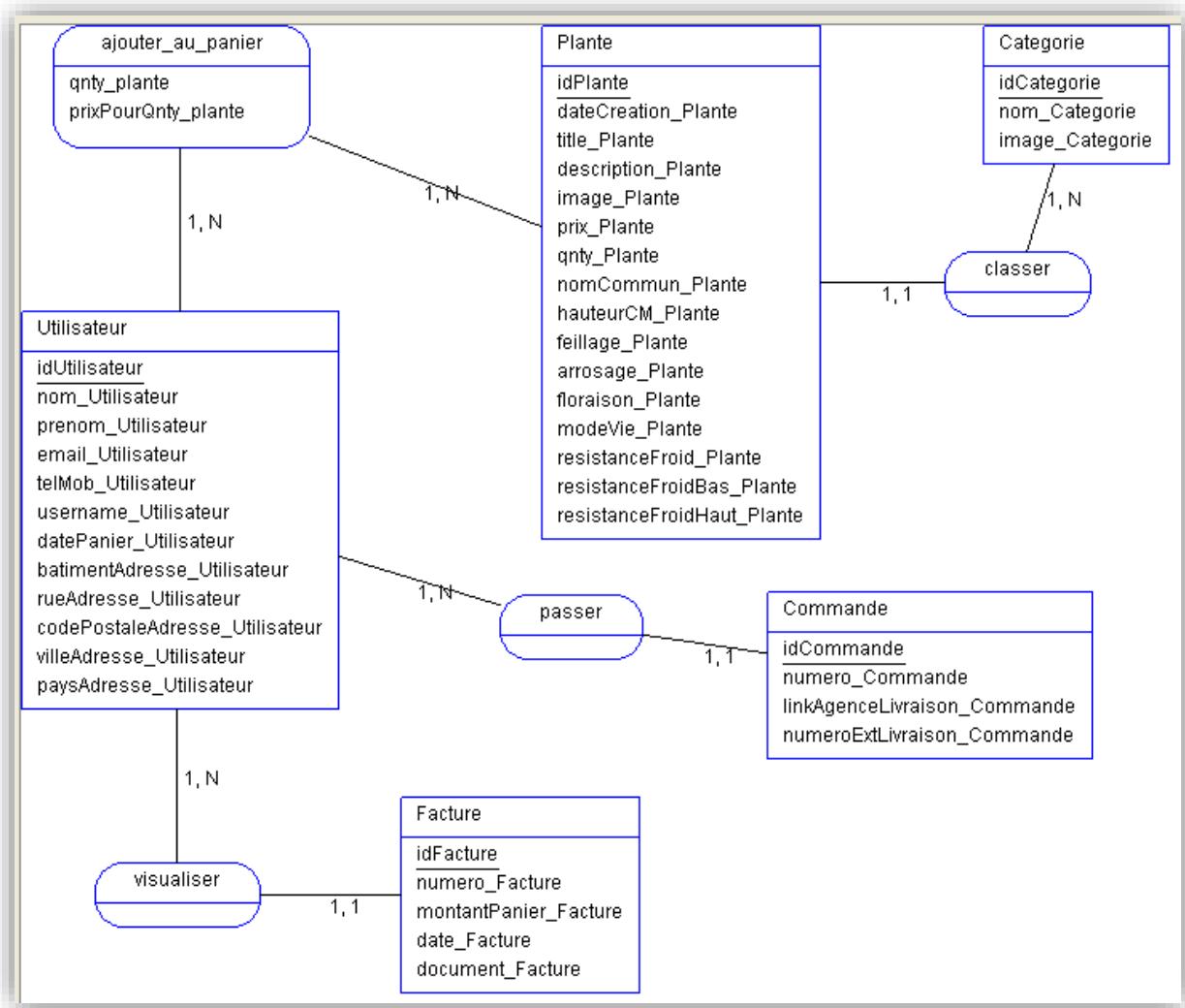


Fig. 46 L'interface de AnalyseSI

Ensuite j'ai installé **WAMP** pour accéder à **PhpMyAdmin** ce qui m'a permis de créer une base de données pour faire fonctionner mon code Php.

WAMP signifie "**Windows, Apache, MySQL et PHP**". C'est un ensemble de logiciels open-source populaire utilisé pour le développement web sur les systèmes d'exploitation Windows.

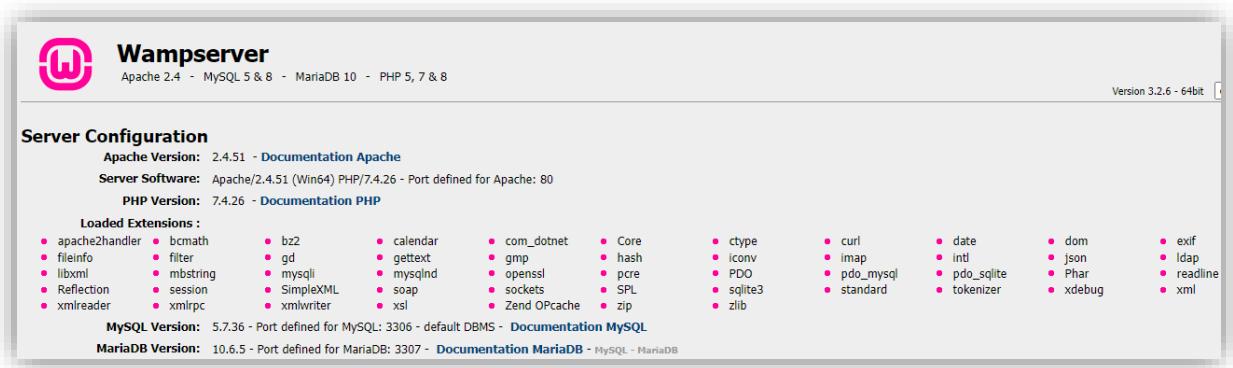


Fig. 47 WAMP

Grâce aux fonctionnalités d'**Analyse-SI**, nous pouvons obtenir le code **SQL** à partir du modèle créé, puis l'importer dans **PhpMyAdmin** pour créer une base de données opérationnelle pour le projet.

```

DROP TABLE IF EXISTS Utilisateur ;
CREATE TABLE Utilisateur (idUtilisateur INTEGER(10) AUTO_INCREMENT NOT NULL,
nom_Utilisateur VARCHAR(45),
prenom_Utilisateur VARCHAR(45),
email_Utilisateur VARCHAR(45),
telMob_Utilisateur INTEGER(20),
username_Utilisateur VARCHAR(45),
datePanier_Utilisateur DATE,
batimentAdresse_Utilisateur INTEGER(5),
rueAdresse_Utilisateur VARCHAR(45),
codePostaleAdresse_Utilisateur INTEGER(6),
villeAdresse_Utilisateur VARCHAR(45),
paysAdresse_Utilisateur VARCHAR(45),
PRIMARY KEY (idUtilisateur)) ENGINE=InnoDB;

```

Fig. 48 Fragment de code SQL généré par AnalyseSI

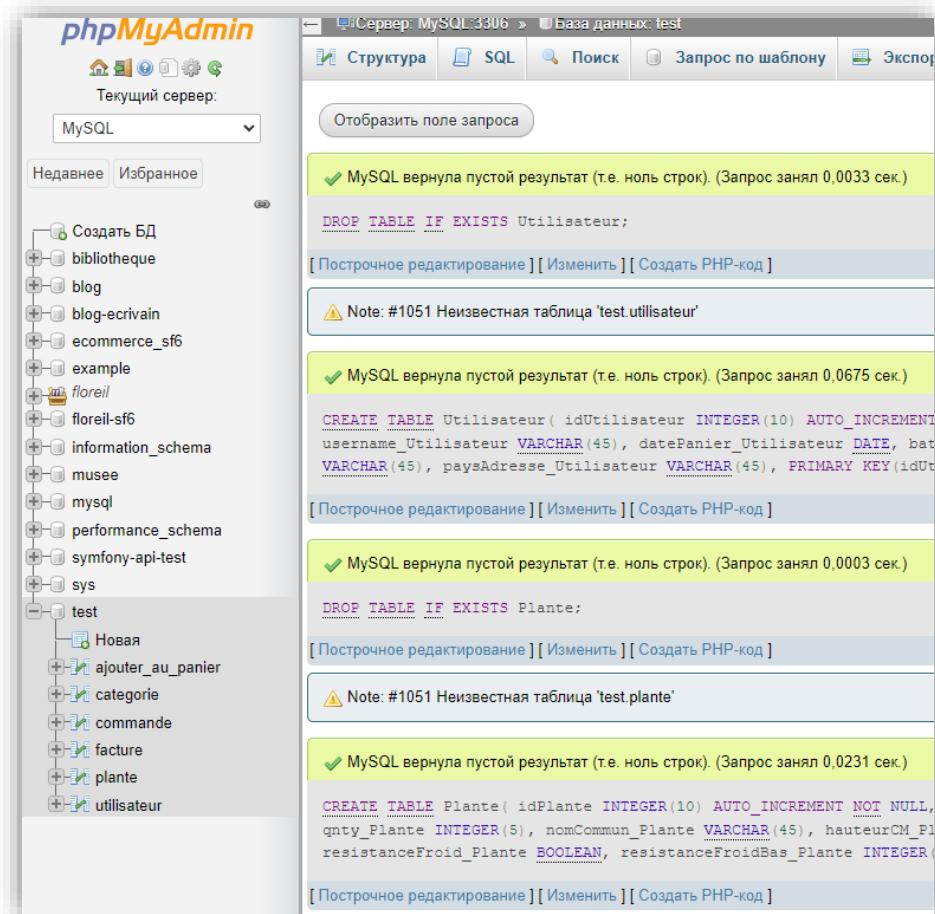
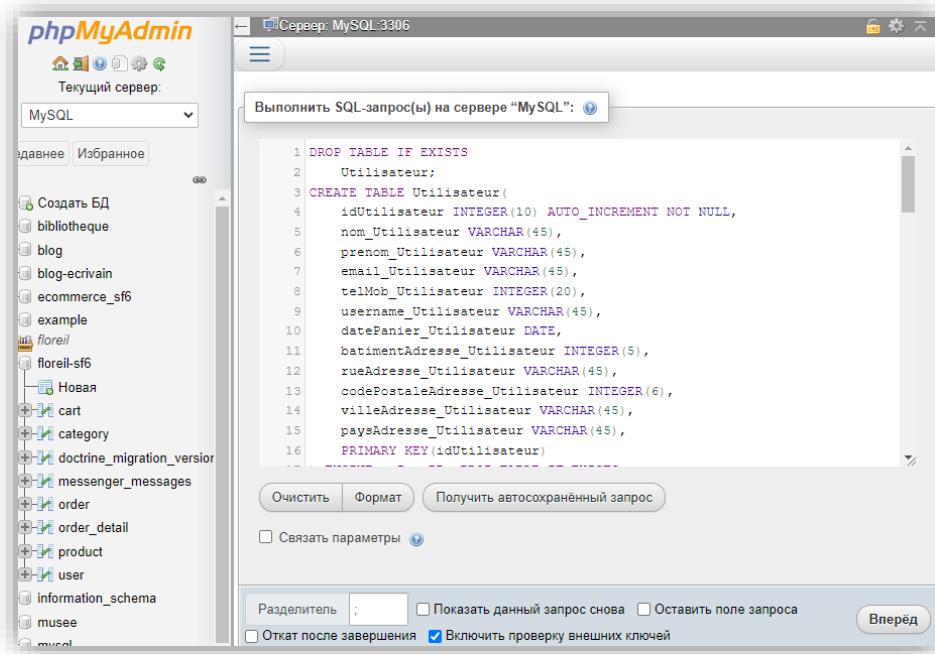


Fig. 49 Creation de bdd à la base de code SQL généré par AnalyseSI

Voici le dictionnaire de données :

| Table:product | |
|---------------------------|---------------|
| Columns: | Type: |
| <u>id</u> | int(11) AI PK |
| <u>category_id</u> | int(11) |
| title | varchar(255) |
| description | longtext |
| image | varchar(255) |
| common_name | varchar(255) |
| genre | varchar(255) |
| size | varchar(255) |
| foliage | varchar(255) |
| watering | longtext |
| bloom | varchar(255) |
| is_fragrant_bloom | tinyint(1) |
| price | double |
| in_stock_qty | int(11) |
| created_at | datetime |
| slug | varchar(255) |
| is_resisted_to_cold | tinyint(1) |
| cold_resistance | int(11) |

| Table:category | |
|------------------|---------------|
| Columns: | Type: |
| <u>id</u> | int(11) AI PK |
| name | varchar(100) |
| image | varchar(255) |
| slug | varchar(255) |
| description | longtext |

| Table:user | |
|-----------------|---------------|
| Columns: | Type: |
| <u>id</u> | int(11) AI PK |
| <u>email</u> | varchar(180) |
| roles | longtext |
| password | varchar(255) |
| surname | varchar(255) |
| name | varchar(255) |
| mob_tel | varchar(255) |
| city | varchar(255) |
| code_zip | varchar(10) |
| street | varchar(255) |
| build_num | varchar(10) |
| created_at | datetime |
| avatar_image | varchar(255) |
| is_verified | tinyint(1) |
| reset_token | varchar(255) |
| additional_info | longtext |

| Table:order | |
|----------------|---------------|
| Columns: | Type: |
| <u>id</u> | int(11) AI PK |
| <u>user_id</u> | int(11) |
| reference | varchar(255) |
| created_at | datetime |
| price | double |

| Table:order_detail | |
|--------------------|------------|
| Columns: | Type: |
| <u>orders id</u> | int(11) PK |
| <u>products id</u> | int(11) PK |
| qnty | int(11) |

| Table:cart | |
|--------------------|------------|
| Columns: | Type: |
| <u>products id</u> | int(11) PK |
| <u>users id</u> | int(11) PK |
| qnty | int(11) |
| created_at | datetime |

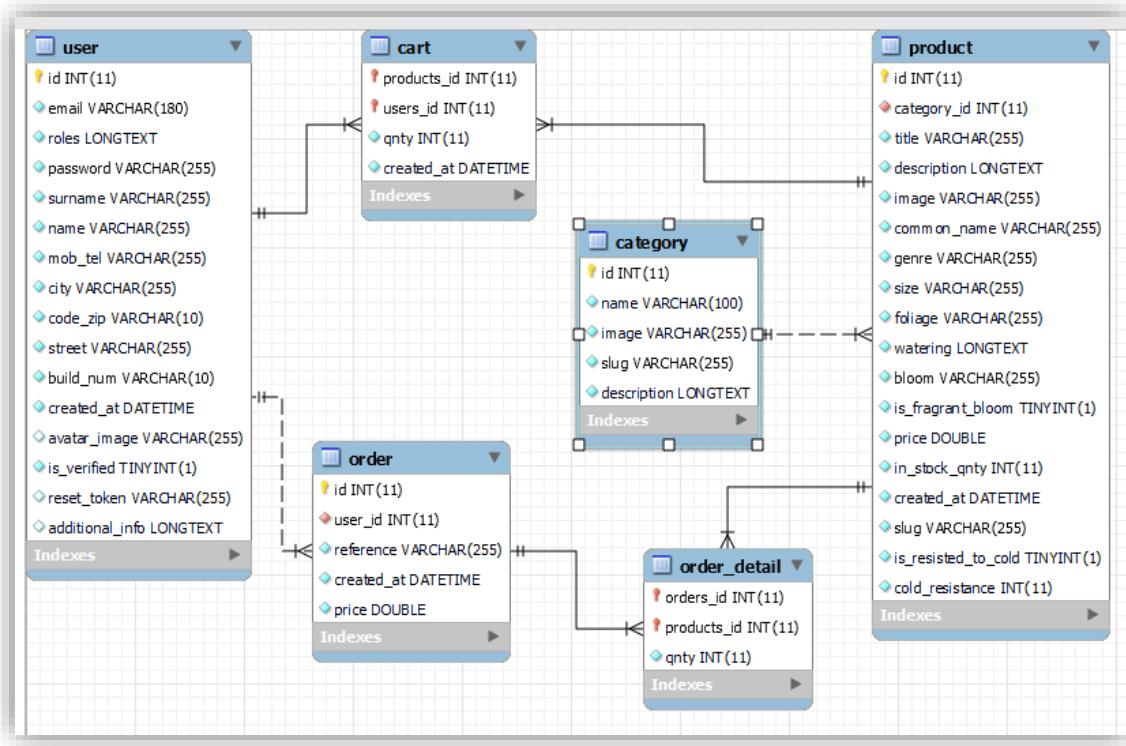


Fig. 50 Le modèle MCD et ses associations

J'ai défini chaque cardinalité pour chaque relation :

- Pour la relation **user-cart**: J'ai choisi la cardinalité **OneToMany**, car chaque **user** peut avoir plusieurs **cart**s et chaque **cart** peut avoir uniquement un **user**.
- Pour la relation **product-cart**: J'ai choisi la cardinalité **OneToMany**, car chaque **product** peut avoir plusieurs **cart**s et chaque **cart** peut avoir uniquement un **product**.
- Pour la relation **product-category**: J'ai choisi la cardinalité **ManyToOne**, car chaque **product** peut avoir un **category** et chaque **category** peut avoir plusieurs **products**.
- Pour la relation **user-order**: J'ai choisi la cardinalité **OneToMany**, car chaque **user** peut avoir plusieurs **orders** et chaque **order** peut avoir uniquement un **user**.
- Pour la relation **order-order_detail**: J'ai choisi la cardinalité **OneToMany**, car chaque **order** peut avoir plusieurs **order_details** et chaque **order_detail** peut avoir uniquement un **order**.
- Pour la relation **product-order_detail**: J'ai choisi la cardinalité **OneToMany**, car chaque **product** peut avoir plusieurs **order_details** et chaque **order_detail** peut avoir uniquement un **product**.

2.4.2 Compétences n°6 : Développer les composants d'accès aux données

Je vais utiliser l'**ORM Doctrine** : qui est comme son nom l'indique un **ORM**: (**Object Relational Mapping**). L'ORM est une classe qui permet de manipuler les tables d'une base de données comme des objets (**Entities**) et leurs relations. Doctrine ajoute également des fonctionnalités à la **PDO**. (PHP Data Objects).

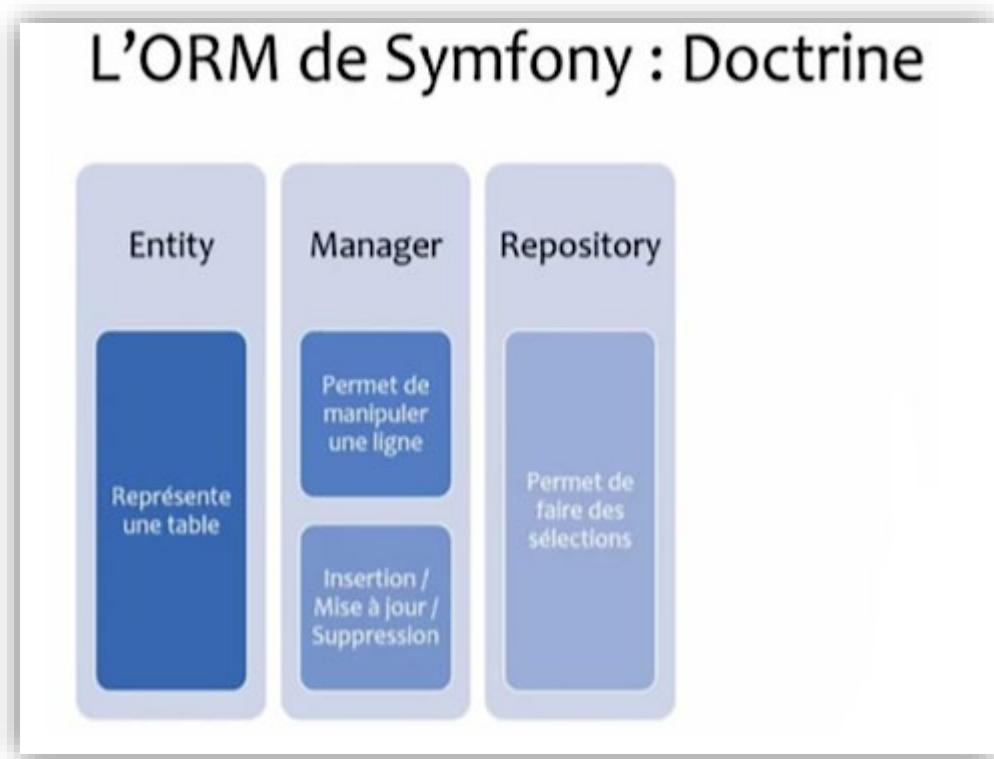


Fig. 51 L'ORM de symfony

J'ouvre le fichier .env comme dans l'exemple ci-dessous et je modifie la ligne 28 en y ajoutant le nom de ma base de données (floreil-sf6) et en vérifiant le port du serveur MySql qui est le 3306 ainsi que le nom d'utilisateur de la machine et le mot de passe.

```
26 # DATABASE_URL="sqlite:///kernel.project_dir/var/data.db"
27 # DATABASE_URL="mysql://root:root@127.0.0.1:3306/floreil-sf6?serverVersion=5.7&unix_socket=/cl
28 DATABASE_URL="mysql://root:@127.0.0.1:3306/floreil-sf6?serverVersion=5.7"
29 # DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=15&charset=utf8"
30 ##### doctrine/doctrine-bundle #####
```

Fig. 52 Modification du fichier .env

Si on n'a pas de base de données on peut la créer avec la commande de Symfony: **symfony console doctrine:database:create**.

Notre base de données peut être trouver dans PhpMyAdmin.

Je vais commencer par créer l'entité **User** pour mieux sécuriser mon back-end, ce qui va me permettre de stocker les informations des utilisateurs quand ils vont se retrouver sur notre site et qu'ils vont accéder à la page inscription. Je souhaite que ces informations soient stockées dans ma base de données pour qu'ils puissent avoir accès à l'application quand ils le souhaitent.

J'utilise la commande spéciale : **symfony console make :user**.

Cette commande a crée deux nouveaux fichiers une **entité User** (qui représente la table des Utilisateurs inscrit sur le site) et un **repository UsersRepository** (qui permet de faire des sélections sur les données de la table des utilisateurs).

```
#[ORM\Entity(repositoryClass: UserRepository::class)]
#[UniqueEntity(fields: ['email'], message: 'There is already an account with this email')]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    use CreatedAtTrait;

    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 180, unique: true)]
    private ?string $email = null;

    #[ORM\Column]
    private array $roles = [];

    /**
     * @var string The hashed password
     */
    #[ORM\Column]
    private ?string $password = null;

    #[ORM\Column(length: 255)]
    private ?string $surname = null;

    #[ORM\Column(length: 255)]
    private ?string $name = null;

    #[ORM\Column(length: 255)]
    private ?string $mobTel = null;

    #[ORM\Column(length: 255)]
    private ?string $city = null;
```

Fig. 53 Le fichier User.php

Et à également créer dans mon entité User une **propriété unique** pour mes utilisateurs, exemple avec leurs : (*emails*). Ce qui permettra à un utilisateur de se connecter qu'une seule fois avec son adresse email, ce sera son **identifiant unique**. Si une autre personne essaye de se connecter avec la même adresse email elle obtiendra un **message d'erreur** (« L'adresse email déjà été utilisée »).

Puis m'a proposé de stocker les données de cette entité dans ma base de données via doctrine et m'a également proposé de **hasher, encoder, crypté** les mots de passe de mes utilisateurs pour ne pas les stocker en clair.

Le fichier **UserRepository.php**, permet de récupérer les données de l'entité User, cela remplace les requête **SQL** préparées, du type : (`$sql = "SELECT * FROM 'user' WHERE id ?";`) qui permettent de récupérer un utilisateur en base de données.

Avec **Symfony** l'idée c'est de pouvoir récupérer toutes les datas d'un utilisateur grâce à son Id avec la méthode `->findOneById()`; du coup on a plus besoin de taper de requête, tout ça est possible grâce à l'ORM Doctrine, il nous permet d'intéragir avec notre base de données.

```
/**  
 * @extends ServiceEntityRepository<User>  
 *  
 * @method User|null find($id, $lockMode = null, $lockVersion = null)  
 * @method User|null findOneBy(array $criteria, array $orderBy = null)  
 * @method User[] findAll()  
 * @method User[] findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)  
 */  
class UserRepository extends ServiceEntityRepository implements PasswordUpgraderInterface  
{  
    public function __construct(ManagerRegistry $registry)  
    {  
        parent::__construct($registry, User::class);  
    }  
  
    public function save(User $entity, bool $flush = false): void  
    {  
        $this->getEntityManager()->persist($entity);  
  
        if ($flush) {  
            $this->getEntityManager()->flush();  
        }  
    }  
  
    public function remove(User $entity, bool $flush = false): void  
    {  
        $this->getEntityManager()->remove($entity);  
  
        if ($flush) {  
            $this->getEntityManager()->flush();  
        }  
    }  
}
```

Fig. 54 Le fichier User.php

Ensuite on finalise en effectuant une migration vers la base de données avec la commande : **symfony console make:migration**, qui permet de créer une migration SQL à partir des entités présentes. Et pour faire une mise à jour on se sert de la commande : **symfony console doctrine:migrations:migrate**, qui permet de lancer les scripts de migrations afin de mettre à jour la base de données.

```

final class Version20230225212745 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('CREATE TABLE cart (products_id INT NOT NULL, users_id INT NOT NULL, q
        $this->addSql('CREATE TABLE category (id INT AUTO_INCREMENT NOT NULL, name VARCHAR(1
        $this->addSql('CREATE TABLE `order` (id INT AUTO_INCREMENT NOT NULL, user_id INT NOT
        $this->addSql('CREATE TABLE order_detail (orders_id INT NOT NULL, products_id INT NO
        $this->addSql('CREATE TABLE product (id INT AUTO_INCREMENT NOT NULL, category_id INT
        $this->addSql('CREATE TABLE user (id INT AUTO_INCREMENT NOT NULL, email VARCHAR(180)
        $this->addSql('CREATE TABLE messenger_messages (id BIGINT AUTO_INCREMENT NOT NULL, b
        $this->addSql('ALTER TABLE cart ADD CONSTRAINT FK_BA388B76C8A81A9 FOREIGN KEY (produ
        $this->addSql('ALTER TABLE cart ADD CONSTRAINT FK_BA388B767B3B43D FOREIGN KEY (users_
        $this->addSql('ALTER TABLE `order` ADD CONSTRAINT FK_F5299398A76ED395 FOREIGN KEY (u
        $this->addSql('ALTER TABLE order_detail ADD CONSTRAINT FK_ED896F46CFFE9AD6 FOREIGN K
        $this->addSql('ALTER TABLE order_detail ADD CONSTRAINT FK_ED896F466C8A81A9 FOREIGN K
        $this->addSql('ALTER TABLE product ADD CONSTRAINT FK_D34A04AD12469DE2 FOREIGN KEY (c
    }

    public function down(Schema $schema): void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->addSql('ALTER TABLE cart DROP FOREIGN KEY FK_BA388B76C8A81A9');
        $this->addSql('ALTER TABLE cart DROP FOREIGN KEY FK_BA388B767B3B43D');
        $this->addSql('ALTER TABLE `order` DROP FOREIGN KEY FK_F5299398A76ED395');
        $this->addSql('ALTER TABLE order_detail DROP FOREIGN KEY FK_ED896F46CFFE9AD6');
        $this->addSql('ALTER TABLE order_detail DROP FOREIGN KEY FK_ED896F466C8A81A9');
        $this->addSql('ALTER TABLE product DROP FOREIGN KEY FK_D34A04AD12469DE2');
        $this->addSql('DROP TABLE cart');
        $this->addSql('DROP TABLE category');
    }
}

```

Fig. 55 L'exemple de fichier migration

Dans le dossier « migrations », toutes les **migrations** créées avec la liste des instructions SQL sont stockées. Si nécessaire, chaque migration peut être annulée, ce qui annulera tous les changements apportés à la base de données par cette migration.

De manière similaire à création d'entity **User**, nous créons les autres entités de notre base de données à l'aide de la commande console :

symfony console make:entity <nom de l'entité>. Ainsi, nous définissons les entités **Category**, **Product**, **Order**, **OrderDetails** et **Cart**. Pour chacune de ces entités, nous définissons les champs de la même manière que pour l'entité **User**. Par exemple, dans l'entité **Category**, nous avons défini les champs **id**, **name**, **image**, **description**, **slug** et avons également défini la **relation** avec l'entité **Product** (comme je l'ai déjà décrit dans la section précédente, la relation **Product-Category** est **ManyToOne**, donc la relation **Category-Product** sera **OneToMany**).

```
#[ORM\Entity(repositoryClass: CategoryRepository::class)]
class Category
{
    use SlugTrait;

    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 100)]
    private ?string $name = null;

    #[ORM\OneToMany(mappedBy: 'category', targetEntity: Product::class)]
    private Collection $products;

    #[ORM\Column(length: 255)]
    private ?string $image = null;

    #[ORM\Column(type: Types::TEXT)]
    private ?string $description = null;
```

Fig. 56 Entité Category

Certains champs (comme **createdAt** ou **slug**) sont identiques pour certaines entités. Ainsi, pour éviter la duplication de code, nous pouvons les extraire dans un **trait** distinct et l'utiliser en cas de besoin dans les entités concernées.

Un **trait** est un mécanisme permettant de réutiliser du code qui n'est pas hérité de la classe parent, mais peut être utilisé dans plusieurs classes. Dans Symfony 6, le trait est utilisé pour éviter la duplication de code dans plusieurs entités de la base de données, par exemple lorsque plusieurs entités ont des champs identiques. Le trait permet de regrouper ces champs dans un module distinct et de l'utiliser dans les entités concernées.

```
src > Entity > Trait > SlugTrait.php > ...
1  <?php
2  namespace App\Entity\Trait;
3
4  use Doctrine\ORM\Mapping as ORM;
5
6  trait SlugTrait
7  {
8      #[ORM\Column(length: 255)]
9      private ?string $slug = null;
10
11     public function getSlug(): ?string
12     {
13         return $this->slug;
14     }
15
16     public function setSlug(string $slug): self
17     {
18         $this->slug = $slug;
19
20         return $this;
21     }
22 }
```

```
src > Entity > Trait > CreatedAtTrait.php > ...
1  <?php
2  namespace App\Entity\Trait;
3
4  use Doctrine\ORM\Mapping as ORM;
5
6  trait CreatedAtTrait
7  {
8      #[ORM\Column(options:['default' => 'CURRENT_TIMESTAMP'])]
9      private ?\DateTimeImmutable $created_at = null;
10
11     public function getCreatedAt(): ?\DateTimeImmutable
12     {
13         return $this->created_at;
14     }
15
16     public function setCreatedAt(\DateTimeImmutable $created_at): self
17     {
18         $this->created_at = $created_at;
19
20         return $this;
21     }
22 }
23
```

Fig. 57 SlugTrait et CreatedAtTrait

2.4.3 Compétences n°7 : Développer la partie back-end d'une application web ou web mobile

J'intègre **EasyAdmin** bundle à mon projet pour créer la partie administrative(**backoffice**) de mon site, cela va me permettre de contrôler plus facilement toutes les données recueillies pour chaque catégorie concernée. Je récupère le dépôt sur la documentation de symfony et j'exécute la commande : **composer require easycorp/easyadmin-bundle**, pour l'installer. Puis j'ai créé un dashboard avec la commande : **symfony console make:admin:dashboard**. Le **DashboardController** se retrouve dans le dossier Controller/Admin/.

```
class DashboardController extends AbstractDashboardController
{
    #[Route('/admin', name: 'admin')]
    #[IsGranted('ROLE_ADMIN')]
    public function index(): Response
    {
        return $this->render('admin/index.html.twig');
    }

    public function configureDashboard(): Dashboard
    {
        return Dashboard::new()
            ->setTitle('Floreil Symfony6');
    }

    public function configureMenuItems(): iterable
    {
        yield MenuItem::linkToDashboard('Dashboard', 'fa fa-dashboard');
        yield MenuItem::linkToUrl('Floreil', 'fas fa-home', $this->generateUrl('app_main'));

        yield MenuItem::section('Shop');
        yield MenuItem::linkToCrud('Categories', 'fas fa-list', Category::class);
        yield MenuItem::linkToCrud('Products', 'fas fa-list', Product::class);

        yield MenuItem::section('Users');
        yield MenuItem::linkToCrud('Users', 'fas fa-users', User::class);
        yield MenuItem::linkToCrud('Orders', 'fas fa-list', Order::class);
    }
}
```

Fig. 58 Fragment de code de DashboardController

Dans ce **DashboardController** il y a une route (« **/admin** ») qui nous renvoie sur la Vue du dashboard.

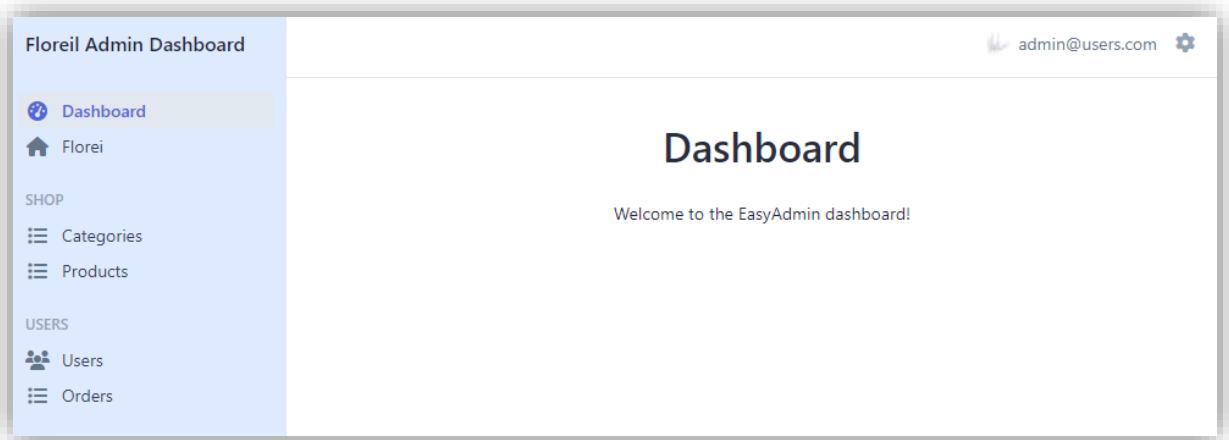


Fig. 59 La Vue du dashboard

Je créer le **CRUD** (create, read, update, delete) pour mon entité **User**, avec la commande : **symfony console make:admin:crud**.

```
class UserCrudController extends AbstractCrudController
{
    public static function getEntityFqcn(): string
    {
        return User::class;
    }

    public function configureFields(string $pageName): iterable
    {
        yield ImageField::new('avatarImage')
            ->setBasePath("images/profiles")
            ->setUploadDir("public/images/profiles")
            ->setUploadedFileNamePattern('[slug]-[timestamp].[extension]');
        yield IdField::new('id')
            ->hideOnForm();
        yield EmailField::new('email');
        yield BooleanField::new('isVerified')
            ->renderAsSwitch(false);
        $roles = ['ROLE_SUPER_ADMIN', 'ROLE_ADMIN', 'ROLE_MODERATOR', 'ROLE_USER'];
        yield ArrayField::new('roles')
            ->hideOnForm();
        yield ChoiceField::new('roles')
            // ->setHelp(' Available roles: ROLE_SUPER_ADMIN, ROLE_ADMIN, ROLE_MODERATOR, ROLE_USER')
            ->setFormTypeOptions([
                'choices' => array_combine($roles, $roles),
                'multiple' => true,
                'expanded' => true
            ])
            ->onlyOnForms();
        yield TextField::new('name');
        yield TextField::new('surname');
        yield TextField::new('mobTel');
        yield TextField::new('city');
        yield TextField::new('codeZIP');
        yield TextField::new('street');
        yield TextField::new('buildNum');
```

Fig. 60 Le code du UserCrudController

Avec le **CRUD** je peux créer, modifier, mettre à jour et supprimer des utilisateurs, je peux déployer cette fonctionnalité pour toutes mes entités **Order, Product, Category** etc.. Cela me permet de pouvoir contrôler toute ma partie administrative facilement.

The screenshot shows the Floreil Admin Dashboard with the 'Users' section selected. The main area displays a table titled 'User' with 8 results. The columns include: Avatar Image, ID, Email, Is Verified, Roles, Name, Surname, Mob Tel, City, and Code ZIP. The data rows show various user entries with their respective details like email, role (ROLE_ADMIN, ROLE_USER), and city.

| | Avatar Image | ID | Email | Is Verified | Roles | Name | Surname | Mob Tel | City | Code ZIP |
|--------------------------|--------------|----|-----------------------------|-------------|-----------------------|------------|------------|-------------|-----------------------|----------|
| <input type="checkbox"/> | | 21 | admin@users.com | YES | ROLE_ADMIN, ROLE_USER | Admin | Adminovich | +123456789 | City No.0 | 00000 |
| <input type="checkbox"/> | Null | 22 | 1user@users.com | YES | ROLE_USER | 1User | 1Userovich | 1+123456789 | City No.1 | 10000 |
| <input type="checkbox"/> | Null | 23 | 2user@users.com | NO | ROLE_USER | 2User | 2Userovich | 2+123456789 | City No.2 | 20000 |
| <input type="checkbox"/> | Null | 24 | 3user@users.com | NO | ROLE_USER | 3User | 3Userovich | 3+123456789 | City No.3 | 30000 |
| <input type="checkbox"/> | Null | 25 | 4user@users.com | NO | ROLE_USER | 4User | 4Userovich | 4+123456789 | City No.4 | 40000 |
| <input type="checkbox"/> | Null | 26 | 5user@users.com | NO | ROLE_USER | 5User | 5Userovich | 5+123456789 | City No.5 | 50000 |
| <input type="checkbox"/> | Null | 27 | blacktormentoro@gmail.com | NO | ROLE_USER | Konstantin | Shilkov | 0668476320 | Créteil | 94000 |
| <input type="checkbox"/> | Null | 28 | blacktormentoro+1@gmail.com | NO | ROLE_USER | Konstantin | Shilkov | 0668476320 | Créteil, Val-de-Marne | 94000 |

8 results

Fig. 61 Vue du Crud User avant la création d'un nouvel utilisateur

The screenshot shows the Floreil Admin Dashboard with the 'Create User' form open. The form fields include: Avatar Image (choose file), Email* (input field), Is Verified (checkbox), Roles* (checkboxes for ROLE_SUPER_ADMIN, ROLE_ADMIN, ROLE_MODERATOR, and ROLE_USER, with ROLE_USER checked), Name* (input field), Surname* (input field), Mob Tel* (input field), and City* (input field). There are also 'Create and add another' and 'Create' buttons at the top right.

Fig. 62 Crédation d'un nouvel utilisateur

User

| <input type="checkbox"/> Avatar Image | ID | Email | Is Verified | Roles | |
|---------------------------------------|---|-------|----------------------------|------------------|--------------------------|
| <input type="checkbox"/> |  | 21 | admin@users.com | YES | ROLE_ADMIN, ROLE_USER |
| <input type="checkbox"/> | Null | 22 | 1user@users.com | YES | ROLE_USER |
| <input type="checkbox"/> | Null | 23 | 2user@users.com | NO | ROLE_USER |
| <input type="checkbox"/> | Null | 24 | 3user@users.com | NO | ROLE_USER |
| <input type="checkbox"/> | Null | 25 | 4user@users.com | NO | ROLE_USER |
| <input type="checkbox"/> | Null | 26 | 5user@users.com | NO | ROLE_USER |
| <input type="checkbox"/> | Null | 27 | blacktormentor@gmail.com | NO | ROLE_USER |
| <input type="checkbox"/> | Null | 28 | blacktormentor+1@gmail.com | NO | ROLE_USER |
| <input type="checkbox"/> |  | 29 | konst.shilkov@google.com | YES | ROLE_USER |

9 results

Fig. 63 Nouvel utilisateur créé

Le nouvel utilisateur peut désormais se connecter au site et peut avoir accès à son compte personnel pour modifier son mot de passe ou consulter le détail et l'historique de ses commandes.

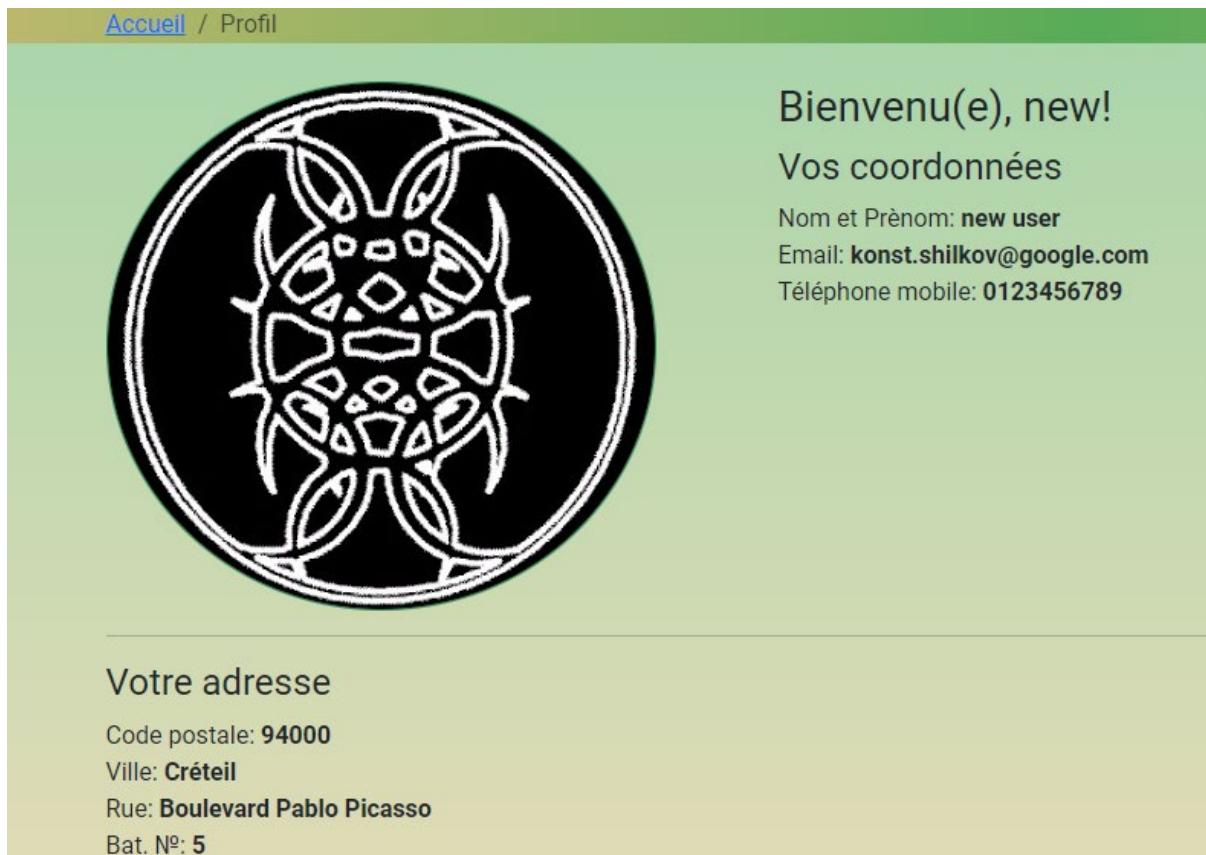


Fig. 64 Espace personnel de l'utilisateur

A présent je souhaite modifier la page d'accueil de **EasyAdmin**, y afficher des statistiques, exemple le nombre d'utilisateurs inscrits sur le site, aussi le nombre de commandes passées en temps réel pour fluidifier la préparation des commandes.

J'ai découvert que la partie centrale de la page d'accueil (**index.html.twig**) d'EasyAdmin est gérée par le bloc "**page_content**". À titre de test, j'ai modifié ce bloc en écrivant un petit message de bienvenue.

```
templates > admin > index.html.twig
1  {% extends "base.html.twig" %}
2  {% extends "@EasyAdmin/page/content.html.twig" %}
3
4  {% block content_title %}
5  |   <h1 class='text-center'>Dashboard</h1>
6  {% endblock %}
7
8  {% block page_content %}
9  |   <p class='text-center'>Welcome to the EasyAdmin dashboard!</p>
10 {% endblock %}
```

Fig. 65 Modification de page d'accueil

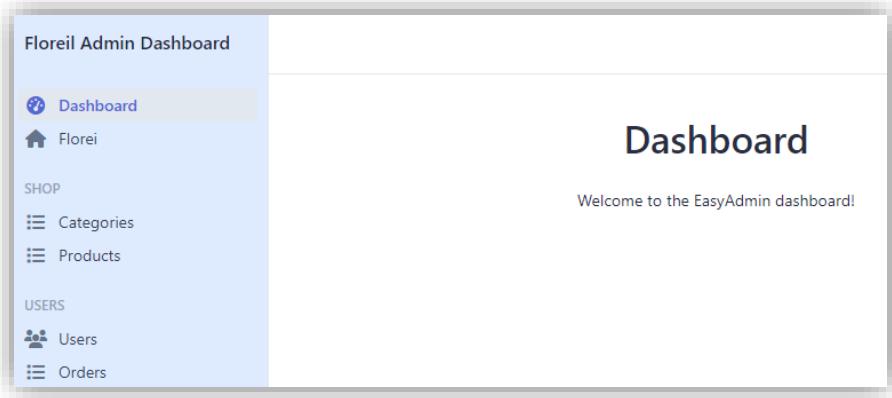


Fig. 66 La vue du fichier index.html.twig après sa modification

Je retourne dans mon DashboardController.php pour créer un constructeur avec la méthode magique (`__construct()`) et je lui passe les repository User, Category, Product et Order, ensuite je déclare les propriétés de User, Category, Product et Order puis je fais un `$this->` de User, de Category, de Product et de Order.

```
class DashboardController extends AbstractDashboardController
{
    protected $userRepository;
    protected $categoryRepository;
    protected $productRepository;
    protected $orderRepository;

    public function __construct(
        UserRepository $userRepository,
        CategoryRepository $categoryRepository,
        ProductRepository $productRepository,
        OrderRepository $orderRepository,
    ) {
        $this->userRepository = $userRepository;
        $this->categoryRepository = $categoryRepository;
        $this->productRepository = $productRepository;
        $this->orderRepository = $orderRepository;
    }
}
```

Fig. 67 Crédit à la construction

Maintenant que mes quatre repository sont injectés, je vais dans les fichiers UserRepository, CategoryRepository, ProductRepository et OrderRepository, pour créer des requêtes qui me permettent de faire la somme des valeurs des User, Category, Product et des Order.

```

public function countAllUsers() {
    $queryBuilder = $this->createQueryBuilder('o');
    $queryBuilder->select('COUNT(o.id) as value');

    return $queryBuilder->getQuery()->getOneOrNullResult();
}

```

Fig. 68 Requête pour récupérer la somme des User

Je retourne dans le DashboardController.php, pour y configurer mon tableau, en lui plaçant les fonctions (**countAllUsers()**, **countAllCategories()**, **countAllProducts()** et **countAllOrders()**), créées précédemment.

```

#[Route('/admin', name: 'admin')]
#[IsGranted('ROLE_ADMIN')]
public function index(): Response
{
    return $this->render('admin/index.html.twig', [
        'countUsers' => $this->userRepository->countAllUsers(),
        'countCategories' => $this->categoryRepository->countAllCategories(),
        'countProducts' => $this->productRepository->countAllProducts(),
        'countOrders' => $this->orderRepository->countAllOrders(),
    ]);
}

```

Fig. 69 Configuration de mon tableau dans le DashboardController

A présent je peux récupérer les données qui se trouvent dans mon tableau pour les afficher dans ma Vue index.html.twig, avec l'interpolation, la double-accolade pour afficher le contenu d'une variable :

(***{ { countUser['value'] }}, { { countCategories['value'] }}, { { countProducts['value'] }}, et { { countOrder['value'] } }***).

```

{% block page_content %}
    <div class='container'>
        <div class='row justify-content-evenly'>
            <div class='col-12 col-md-6 col-lg-3 p-3'>
                <div class='bg-success bg-opacity-75 p-0 m-0'>
                    <a href='?crudAction=index&crudControllerFqcn=App%5CController%5CAdmin%5CUserCrudController' class='text-decoration-none'>
                        <p class='h3'>Utilisateurs</p>
                        <p class='display-3 fw-bold'>{{ countUsers['value'] }}</p>
                    </a>
                </div>
            </div>
            <div class='col-12 col-md-6 col-lg-3 p-3'>
                <div class='bg-danger bg-opacity-75 p-0 m-0'>
                    <a href='?crudAction=index&crudControllerFqcn=App%5CController%5CAdmin%5COrderCrudController' class='text-decoration-none'>
                        <p class='h3'>Commandes</p>
                        <p class='display-3 fw-bold'>{{ countOrders['value'] }}</p>
                    </a>
                </div>
            </div>
            <div class='col-12 col-md-6 col-lg-3 p-3'>
                <div class='bg-warning bg-opacity-75 p-0 m-0'>
                    <a href='?crudAction=index&crudControllerFqcn=App%5CController%5CAdmin%5CCategoryCrudController' class='text-decoration-none'>
                        <p class='h3'>Categories</p>
                        <p class='display-3 fw-bold'>{{ countCategories['value'] }}</p>
                    </a>
                </div>
            </div>
            <div class='col-12 col-md-6 col-lg-3 p-3'>
                <div class='bg-info bg-opacity-75 p-0 m-0'>
                    <a href='?crudAction=index&crudControllerFqcn=App%5CController%5CAdmin%5CProductCrudController' class='text-decoration-none'>
                        <p class='h3'>Produits</p>
                        <p class='display-3 fw-bold'>{{ countProducts['value'] }}</p>
                    </a>
                </div>
            </div>
        </div>
    </div>{%
        endblock %

```

Fig. 70 Récupération des données de mon tableau dans la Vue index.html.twig

Voici le rendu de la page d'accueil de EasyAdmin personnalisée avec un tableau de statistique sur lequel on retrouve le nombre d'**Utilisateurs** inscrits, **Categories**, **Produits** existants et le nombre de **Commandes** passées.

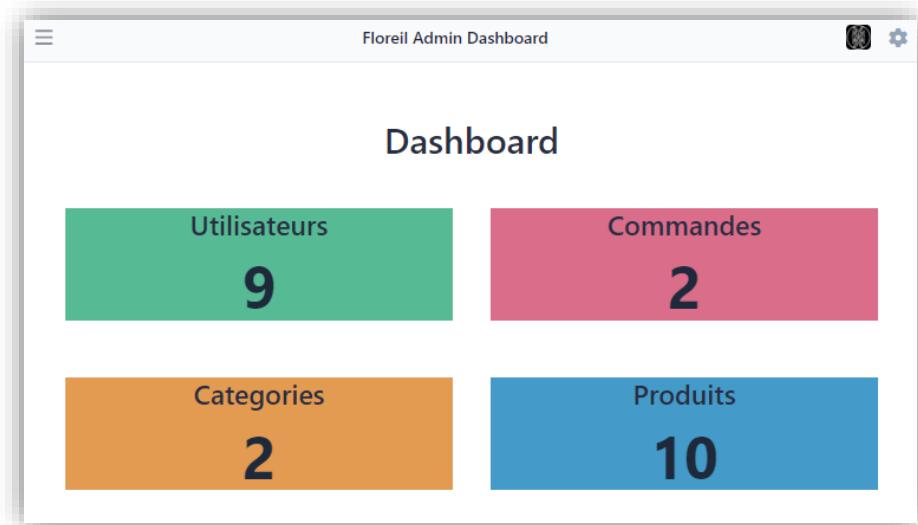


Fig. 71 Le tableau avec ses statistique dans la Vue du fichier index.html.twig

2.5. La sécurité

Les failles **XSS** (cross site scripting), font parties des failles les plus communes mais cela ne les rend pas pour autant moins dangereuses. C'est un type de **faille de sécurité** des sites web permettant d'injecter du contenu dans une page, provoquant ainsi des actions sur les navigateurs web visitant la page. Les possibilités des XSS sont très larges puisque l'attaquant peut utiliser tous les langages pris en charge par le navigateur (JavaScript, Java...) et de nouvelles possibilités sont régulièrement découvertes notamment avec l'arrivée de nouvelles technologies comme **HTML5**.

Il est par exemple possible de réaliser les opérations suivantes :

- Redirection (parfois de manière transparente) de l'utilisateur (souvent dans un but d'hameçonnage)
- Vol d'informations, par exemple **sessions** et **cookies**.
- Actions sur le site faillible, à l'insu de la victime et sous son identité (envoi de messages, suppression de données...)
- Rendre la lecture d'une page difficile (boucle infinie d'alertes par exemple).

Les types de failles :

- La faille **XSS Réflective** (non permanente), se sert du code javascript déjà existant dans une page web, pour y exécuter du code à l'insu de l'utilisateur.
- La faille **XSS Stored** (permanente), se sert d'une page HTML type blog avec du contenu que l'on retrouve en général dans un blog exemple des commentaires. Imaginons qu'un pirate écrit un commentaire sur ce blog en y intégrant du JS dedans, cela crée une XSS dans ce commentaire. La prochaine fois qu'un utilisateur non malveillant viendra charger cette page, il pourra écrire son commentaire, et en dessous de celui-ci il y aura les autres commentaires postés auparavant dont celui du pirate avec la XSS. En conclusion la XSS se retrouve stockée, quand l'utilisateur non malveillant chargera sa page pour enregistrer son nouveau commentaire, il stockera la XSS qui se trouve dans le commentaire précédent et cela se déclenchera pour chaque utilisateur à chaque fois qu'il postera un nouveau commentaire.
- Il y a également une autre faille **XXS**, imaginons une page HTML **côté client**, avec du contenu. De l'autre côté nous avons le **serveur**, entre le client et le

serveur il y a des échanges qui se font, il est possible qu'au milieu de cet échange une personne malveillante **écoute le trafic** et le **modifie** en prenant la page HTML et en y intégrant du JS ce qui va créer une XSS, c'est cette personne malveillante qui l'aura créée.

Plusieurs **techniques** permettent d'éviter le XSS :

- La **détection** de la présence d'une faille XSS peut se faire par exemple en entrant un script Javascript dans un champ de formulaire ou dans une URL :

```
<script>alert('bonjour')</script>
```

Si une boîte de dialogue apparaît, on peut en conclure que l'application Web est sensible aux attaques de type XSS.

- **Retraiter** systématiquement le code HTML produit par l'application avant l'envoi au navigateur ;

- **Filtrer** les variables affichées ou enregistrées avec des caractères '<' et '>' (en CGI comme en PHP). De façon plus générale, donner des noms préfixés (avec par exemple le préfixe "us" pour user string) aux variables contenant des chaînes venant de l'extérieur pour les distinguer des autres, et ne jamais utiliser aucune des valeurs correspondantes dans une chaîne exécutable (en particulier une chaîne SQL, qui peut aussi être ciblée par une injection SQL d'autant plus dangereuse) sans filtrage préalable.

- En **PHP** :

- On peut utiliser la fonction : **htmlspecialchars()** qui filtre les '<' et '>';
- On peut utiliser la fonction : **htmlentities()** qui est identique à **htmlspecialchars()** sauf qu'elle filtre tous les caractères équivalents au codage **HTML** ou **JavaScript**.

Les injections de dépendance SQL :

Une injection SQL consiste en l'insertion d'un code malveillant dans des sites et applications web dans le but de compromettre le site ciblé et de recueillir les données de l'utilisateur. Comme son nom l'indique, une **injection SQL** attaque les bases de données **SQL** (Structured Query Language), qui se trouve être la colonne vertébrale d'un site web.

Mais elles peuvent aussi, enfreindre la sécurité des données, envoyer de fausses informations dans la base de données de l'application, en supprimer les informations importantes ou empêcher l'accès aux propriétaires et créateurs de l'application.

SQL est un langage spécialement conçu pour saisir des données et modifier le contenu de base de données. Les sites et applications web utilisent ces bases de données pour stocker toutes leurs

données et fournir leurs services aux utilisateurs. SQL joue un rôle primordial dans ce processus, en permettant aux utilisateurs de localiser un contenu spécifique au sein de la base de données.

Comme son nom l'indique, les attaques par **injection SQL** ciblent ces bases de données SQL. Le pirate à l'origine de l'attaque utilise un manque de **filtres de validation** de saisie au niveau des caractères d'échappement (par exemple la barre oblique inversée) pour injecter son propre code dans le système. En fonction de leurs objectifs, les pirates peuvent écrire le code de manière à ce qu'à chaque saisie de recherche d'un utilisateur, ils puissent avoir accès à ses identifiants ou détruire une partie de la base de données. Les **injections SQL** peuvent même être utilisées pour distribuer des malwares à travers des sites web infectés.

CSRF (Cross-Site Request Forgery) est une attaque sur les applications web qui est utilisée pour effectuer des actions non désirées au nom d'un utilisateur authentifié. Dans cette attaque, un attaquant crée une fausse requête qui ressemble à une requête légitime de l'utilisateur authentifié et l'envoie sur le site.

Pour se protéger contre de telles attaques, un mécanisme de **jetons CSRF** est utilisé. Lorsqu'un formulaire est envoyé au serveur, un jeton unique est généré et stocké dans la session de l'utilisateur. Ce jeton est ensuite inséré dans un champ masqué du formulaire. Lorsqu'une requête est envoyée au serveur, ce jeton est comparé au jeton stocké dans la session. Si les jetons ne correspondent pas, la requête est rejetée comme suspecte et non autorisée.

Ainsi, l'utilisation de **jetons CSRF** permet de protéger les applications web contre de telles attaques en empêchant l'exécution d'actions non désirées au nom d'un utilisateur authentifié.

Les types d'injections SQL :

- **L'injection SQL Union :**

Une injection SQL Union est un type d'attaque par injection SQL in band qui utilise l'opérateur UNION SQL pour extraire aisément les informations requises de la base de données ciblée.

L'opérateur UNION permet à l'utilisateur de retirer des données simultanément sur de multiples tableaux constitués du même nombre de colonnes et types de données identiques. Les pirates

peuvent collecter les informations dont ils ont besoin en injectant une commande SELECT, mais pour que l'attaque réussisse ils doivent connaître le nom exact du tableau, le nombre de colonnes et le type des données.

- **L'injection SQL error-based :**

Une injection SQL error-based est un type d'attaque par injection SQL in band. Cette technique permet aux pirates de pondérer les messages d'erreurs retournés par les serveurs pour obtenir

des informations sur la structure du serveur ciblé. Les pirates saisissent volontairement des demandes invalides pour déclencher des messages d'erreurs. Ces messages contiennent cependant souvent soit le résultat complet de la demande, soit des informations sur la manière d'améliorer la demande pour obtenir les résultats désirés, qui peuvent aider les pirates à mener à bien leur attaque.

- **L'injection SQL blind time-based :**

Une injection SQL blind time-based est une technique qui repose sur l'envoi d'une demande SQL à une base de données pour évaluer les résultats d'une demande. La demande en question force la base de données à patienter avant de retourner un résultat, qui sera soit TRUE soit FALSE. En fonction du temps de réponse et de la réponse elle-même, un pirate peut voir si sa charge active a bien été transmise. L'inconvénient principal de ce type d'injection SQL est le temps qu'elle

prend à réaliser car le pirate doit énumérer la base de données caractère par caractère.

- **L'injection SQL boolean-based :**

Un injection SQL blind boolean-based est une technique d'injection inférentielle (conclusions à partir d'une assertion considérée comme vraie) très similaire à l'injection SQL blind time-based par laquelle les pirates envoient une commande SQL à la fois pour tenter d'énumérer la base de données. En fonction de la réponse qu'ils reçoivent, ils vérifient si leur charge utile a été envoyée correctement. Mais plutôt que de temporiser leurs commandes, ils combinent des expressions

TRUE et FALSE. Comme avec les injections SQL time-based, ces attaques peuvent être très lentes, en particulier lorsqu'un pirate cible une large base de données.

- **L'injection SQL out-of-band:**

L'injection SQL out-of-band est une technique utilisée par les pirates pour générer des commandes DNS (Domain Name System) et/ou http, destinée à leur envoyer directement des données. Elle est parfois utilisée comme alternative aux attaques par injection SQL blind timebased, généralement lorsque le serveur est lent à répondre ou lorsqu'il est impossible de collecter des données par le canal utilisé pour lancer l'attaque. Leur réussite dépendant des fonctions susceptibles d'être activées par l'administrateur du serveur, les attaques par injection SQL out-of-band sont très rares.

Pour éviter les attaques par injection SQL il est conseillé de surveiller en permanence les commandes SQL de toutes les applications reliées à la base de données, pratiquer une mise à jour régulière des bases de données et correctif, également l'achat d'un bon logiciel de cybersécurité pour protéger la base de données.

Ces attaques ciblant des sites web utilisant un SQL dynamique, vous devez prendre les mesures nécessaires pour minimiser l'action de l'utilisateur dans la construction de vos commandes.

Lorsque cela est possible, offrez aux utilisateurs des commandes prêtes ainsi qu'une liste d'options parmi lesquelles choisir plutôt que la possibilité de saisir leur propre commande. Il est aussi important d'utiliser la validation de saisie pour éviter les problèmes liés aux caractères d'échappement. N'oubliez pas non plus d'activer le filtrage des données sur le contexte.

Autorisez par exemple uniquement des chiffres pour les numéros de téléphone. Il existe aussi dans quelques cas rares, que les pirates utilisent des attaques par

injection SQL pour compromettre des sites fiables avec des malwares. Dès que vous visitez un site infecté, le

malware se télécharge sans votre consentement. Une fois installé, les pirates peuvent accéder à votre historique de navigation, informations personnelles et même vos frappes de clavier, pour éviter cela il faut se munir d'un bon logiciel antivirus.

Pour prévenir les **attaques CSRF**, j'utilise le contrôleur **UserAuthenticator**.

Ce contrôleur utilise le **mécanisme d'authentification utilisateur** avec la classe **Passport** du composant Security de Symfony.

Le **Passport** est un conteneur qui contient divers "badges" (**Badge**) - des composants qui fournissent des informations sur la manière d'authentifier un utilisateur. Dans ce cas, les badges suivants sont utilisés :

- **UserBadge** - fournit des informations sur l'utilisateur qui tente de s'authentifier. Il prend le nom d'utilisateur en argument.
- **PasswordCredentials** - contient le mot de passe saisi par l'utilisateur.
- **CsrfTokenBadge** - vérifie que le jeton CSRF transmis est valide. Il prend en argument le nom du formulaire pour lequel il faut vérifier le jeton CSRF et la valeur du jeton.

La méthode **authenticate()** utilise ces badges pour créer une instance de la classe Passport, qui représente une demande d'authentification.

Dans la méthode **onAuthenticationSuccess()**, on vérifie si une URL de redirection a été définie après une authentification réussie, et si c'est le cas, on redirige vers cette URL. Sinon, on redirige vers la page d'accueil de l'application.

La méthode **getLoginUrl()** renvoie l'URL de la page de connexion qui sera utilisée si l'utilisateur n'est pas authentifié et doit être redirigé vers la page de connexion.

Dans l'ensemble, l'utilisation de **Passport** avec **UserBadge**, **PasswordCredentials** et **CsrfTokenBadge** permet de vérifier que l'utilisateur entre les bonnes informations d'identification et que les données n'ont pas été falsifiées ou volées par un attaquant lorsqu'elles sont transmises via un formulaire.

```

src > Security > UserAuthenticator.php > UserAuthenticator
18  class UserAuthenticator extends AbstractLoginFormAuthenticator
19  {
20      use TargetPathTrait;
21
22      public const LOGIN_ROUTE = 'app_login';
23
24      public function __construct(private UrlGeneratorInterface $urlGenerator)
25      {
26      }
27
28      public function authenticate(Request $request): Passport
29      {
30          $email = $request->request->get('_username', '');
31
32          $request->getSession()->set(Security::LAST_USERNAME, $email);
33          return new Passport(
34              new UserBadge($email),
35              new PasswordCredentials($request->request->get('_password', '')),
36              [
37                  new CsrfTokenBadge('authenticate', $request->request->get('_csrf_token'))
38              ]
39          );
40      }
41
42      public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
43      {
44          if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
45              return new RedirectResponse($targetPath);
46          }
47
48          // For example:
49          return new RedirectResponse($this->urlGenerator->generate('app_main'));
50          // throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
51      }
52
53      protected function getLoginUrl(Request $request): string
54      {
55          return $this->urlGenerator->generate(self::LOGIN_ROUTE);
56      }
57  }

```

Fig. 72 Le contrôleur UserAuthenticator

J'ouvre dans le dossier config, le dossier packages puis j'ouvre le fichier **security.yaml**, il schématise le composant sécurité de symfony, dans la catégorie Component de la documentation de Symfony on peut retrouver le security component, à lui seul il peut construire notre espace membre, encoder nos mots de passe, de restreindre l'accès à certaine Url uniquement si on a le bon Rôle.

```

config > packages > ! security.yaml
  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:
 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\UserAuthenticator
 20             # activate different ways to authenticate
 21             # https://symfony.com/doc/current/security.html#the-firewall
 22
 23             # https://symfony.com/doc/current/security/impersonating_user.html
 24             # switch_user: true
 25             logout:
 26                 path: app_logout
 27                 # where to redirect after logout
 28                 target: app_main
 29             # Easy way to control access for large sections of your site
 30             # Note: Only the *first* access control that matches will be used
 31             access_control:
 32                 - { path: ^/admin, roles: ROLE_ADMIN }
 33                 # - { path: ^/profile, roles: ROLE_USER }
 34
 35     when@test:
 36         security:
 37             password_hashers:
 38                 # By default, password hashers are resource intensive and take time. This is
 39                 # important to generate secure password hashes. In tests however, secure hashes
 40                 # are not important, waste resources and increase test times. The following
 41                 # reduces the work factor to the lowest possible values.
 42                 Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
 43                     algorithm: auto
 44                     cost: 4 # Lowest possible value for bcrypt
 45                     time_cost: 3 # Lowest possible value for argon
 46                     memory_cost: 10 # Lowest possible value for argon

```

Fig. 73 Le Fichier security.yaml

La première partie définit la façon dont les mots de passe sont hachés et dont les utilisateurs sont authentifiés. Elle spécifie que l'interface ***PasswordAuthenticatedUserInterface*** doit être utilisée pour le hachage automatique des mots de passe et que le fournisseur ***app_user_provider*** doit être utilisé pour l'authentification des utilisateurs.

Ensuite, différents "pare-feu" (***firewalls***) sont définis pour contrôler l'accès à différentes parties de l'application. Dans ce cas, deux pare-feu sont définis : dev pour le développement et main pour l'utilisation principale.

Enfin, un contrôle d'accès (***access_control***) est défini pour différents chemins dans l'application. Par exemple, si un chemin commence par /admin, l'accès ne sera autorisé qu'aux utilisateurs ayant le rôle ***ROLE_ADMIN***.

Dans certains cas (accès au panier, système de paiement ou à l'espace utilisateur), j'utilise l'accès aux attributs (***attribute access***) de Symfony, qui vérifie si l'utilisateur actuel a un statut d'authentification complet (***IS_AUTHENTICATED_FULLY***).

```
#[IsGranted("IS_AUTHENTICATED_FULLY")]
class ProfileController extends AbstractController
{
```

Fig. 74 Utilisation de l'attribut access

L'**attribut access** est un moyen de contrôler l'accès à des actions ou des ressources spécifiques dans Symfony, basé sur les droits et rôles de l'utilisateur. En utilisant des attributs d'accès dans Symfony, vous pouvez restreindre l'accès à des parties de votre application en fonction des droits de l'utilisateur.

Dans ce cas, si l'utilisateur actuel n'a pas réussi une **authentification complète** (par exemple, si son adresse e-mail n'a pas été vérifiée), il n'aura pas accès à la méthode ou au contrôleur contenant cet attribut d'accès.

Généralement, ce code est placé au début de la méthode ou du contrôleur pour s'assurer que l'utilisateur qui essaie d'accéder à cette méthode ou à ce contrôleur a un statut d'authentification complet.

3. L'utilisation de l'anglais dans le travail sur le projet

Dans le métier de développeur d'applications web, il est très important de connaître l'anglais, car de nombreuses documentations et instructions pour travailler avec différents frameworks, bibliothèques et API sont disponibles uniquement en anglais. De plus, le plus grand site d'aide aux développeurs, **StackOverflow**, est en anglais.

Il n'est pas surprenant que lors de la réalisation d'un projet, j'ai souvent utilisé des sources en anglais. Par exemple, pour implémenter le système de paiement Stripe dans mon projet Symfony, j'ai étudié la documentation et regardé un tutoriel en anglais.

De plus, j'ai souvent utilisé l'aide de **ChatGPT**, avec qui j'ai également communiqué en anglais, car ma langue maternelle, le russe, fonctionne souvent avec des dysfonctionnements.

Ci-dessous, je présente un extrait de la documentation Stripe en version originale et en traduction en français :

Stripe.js reference

This reference documents every object and method available in Stripe's browser-side JavaScript library, Stripe.js. Use our React Stripe.js reference if you want to add Elements to your React based app.

You can use Stripe.js' APIs to tokenize customer information, collect sensitive payment details using customizable Stripe Elements, and accept payments with browser payment APIs like Apple Pay and the Payment Request API.

Including Stripe.js

Include the Stripe.js script on each page of your site—it should always be loaded directly from <https://js.stripe.com>, rather than included in a bundle or hosted yourself.

To best leverage Stripe's advanced fraud functionality, include this script on every page, not just the checkout page. This allows Stripe to detect suspicious behavior that may be indicative of fraud as customers browse your website.

Comme le français n'est pas ma langue maternelle et afin d'éviter des erreurs ridicules, j'ai envoyé ma traduction pour vérification et correction à **ChatGPT** (on ne peut pas le qualifier de traducteur automatique, donc je n'ai pas violé formellement la condition). Ci-dessous se trouve sa traduction corrigée et la liste des erreurs que j'ai commises.

En fait, j'ai constamment utilisé l'aide de ChatGPT pour rédiger ce rapport.

Traduction :

Stripe.js référence

Cette référence documente chaque objet et méthode disponible dans la bibliothèque JavaScript côté navigateur de Stripe, Stripe.js. Utilisez notre référence React Stripe.js si vous voulez ajouter des éléments à votre application basée sur React.

Vous pouvez utiliser les API de Stripe.js pour tokeniser les informations des clients, collecter des détails de paiement sensibles à l'aide d'éléments Stripe personnalisables et accepter les paiements avec les API de paiement de navigateur comme Apple Pay et l'API Payment Request.

Inclure Stripe.js

Incluez le script Stripe.js sur chaque page de votre site - il doit toujours être chargé directement depuis <https://js.stripe.com>, plutôt que d'être inclus dans un bundle ou hébergé vous-même.

Pour tirer le meilleur parti de la fonctionnalité de détection de fraude avancée de Stripe, incluez ce script sur chaque page, pas seulement sur la page de paiement. Cela permet à Stripe de détecter les comportements suspects qui pourraient indiquer une fraude alors que les clients naviguent sur votre site Web.

Mes erreurs :

- "Cette references" devrait être "Cette référence".
- "JavaScript bibliothèque" devrait être "bibliothèque JavaScript".

- "*côte navigateur*" devrait être "côté navigateur".
- "*si vous voulez à ajouter*" devrait être "si vous voulez ajouter".
- "*les Elements*" devrait être "les éléments".
- "*l'information d'utilisateur*" devrait être "les informations utilisateur".
- "*ramener les détails sensitifs de payment*" devrait être "collecter les détails de paiement sensibles".
- "*accepter des payments*" devrait être "accepter des paiements".
- "*à l'aide des API de payment de navigateur*" devrait être "à l'aide des API de paiement du navigateur".
- "*Ce permet Stripe à détecter comportement suspect*" devrait être "Cela permet à Stripe de détecter les comportements suspects".
- "*lequel peut être indication de fraud*" devrait être "qui peuvent être une indication de fraude".
- "*lorsque les clients parcourront votre site Web*" devrait être "pendant que les clients naviguent sur votre site Web".

4. Conclusion

En résumé, je voudrais dire que c'est ma première expérience en développement web et web mobile. Avant le cours, j'ai travaillé un peu avec **C#** et **Unity**, donc les langages de programmation et les technologies enseignés sur le cours étaient nouveaux pour moi et leur apprentissage était une expérience intéressante et utile.

Pendant l'apprentissage, j'ai essayé plusieurs méthodes pour développer mon projet, j'ai commencé avec **React**, mais j'ai décidé de me concentrer sur **Symfony** pour plusieurs raisons. Tout d'abord, Symfony est basé sur **PHP**, qui est un **langage de programmation orienté objet**, dont les principes me sont familiers grâce à **C#**. Deuxièmement, **Symfony** est un framework très puissant avec une large gamme de fonctions intégrées. Troisièmement, il est agréable et facile à utiliser, il est très visuel et compréhensible.

Bien sûr, mon projet n'est pas prêt pour une sortie, il y a beaucoup de fonctions qui peuvent et devraient être ajoutées à la version finale, mais même avec un petit ensemble de fonctionnalités, je pense que le projet est bien construit et fiable et occupera une place méritée dans mon portfolio.

Mes objectifs principaux lors de la rédaction de ce projet étaient les suivants : apprendre à écrire du code clair et efficace ; me familiariser avec les technologies actuelles de la programmation web ; acquérir l'habitude de penser de manière structurée pour bien construire mes projets futurs ; apprendre à travailler de manière autonome et à trouver des solutions aux problèmes qui surgissent au cours du développement.

Je pense avoir atteint ces objectifs, bien qu'il reste encore beaucoup à apprendre. Par exemple, lors de mon stage, j'ai travaillé avec mon collègue étudiant Daniel Gimenez sur un projet réel de location de photomatons en **Next.js** et **MongoDB** (vous pouvez trouver la version de développement <https://next-photo-booth.vercel.app>), et c'était une expérience complètement différente. J'ai appris beaucoup de nouvelles choses sur la manipulation des bibliothèques **Next** et **React**. Ce projet n'est pas encore terminé et nous prévoyons de le terminer après avoir obtenu notre diplôme.

Mes plans futurs consistent à chercher du travail et à approfondir mes connaissances en **PHP** et en **JavaScript**, ainsi que dans leurs frameworks populaires. Je ne voudrais pas non plus oublier complètement **C#**.

5. Remerciements

Je voudrais exprimer ma gratitude à notre Tuteur de Formation, M. **Patrick Croquet**, et à toute l'équipe de l'AFPA pour leur aide dans la réalisation de ce projet et leur soutien constant. Je remercie également mes camarades étudiants pour leur atmosphère amicale et leur aide à résoudre les tâches d'apprentissage assignées.