

**DOSSIER PROJET – Mimouni Alice**  
**Période du 2 novembre 2021 au 11 janvier 2022**

**Société Hestia Energies**



# REMERCIEMENTS

Merci à Nom Prénom de m'avoir fait confiance pour la réalisation du site web de sa Société.

Merci à toute l'équipe de Human Booster de m'avoir permis de suivre cette formation.

Merci aux formateurs pour leur travail, et la transmission de leur savoir.

# COMPÉTENCES PROFESSIONNELLES MISES EN OEUVRE

CCP	Compétences Professionnelles	Appliquées	Techniques utilisées
CCP1 Développer la partie Frontend d'une application web ou web mobile en intégrant les recommandations de sécurité.	Maquetter une application	OUI	Figma
	Réaliser une interface utilisateur web statique et adaptable	OUI	HTML/CSS/SCSS/Bootstrap
	Développer une interface utilisateur web dynamique	OUI	CSS/SCSS/Bootstrap
	Réaliser une interface utilisateur avec une solution de gestion	OUI	WordPress
CCP2 Développer la partie Backend d'une application web ou web mobile en intégrant les recommandations de sécurité.	Créer une base de données	NON	
	Développer les composants d'accès aux données	NON	
	Développer la partie Backend d'une application Web ou Web mobile	NON	
	Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce	NON	

# SOMMAIRE

## PARTIE 1 : HESTIA ÉNERGIES

<b>1 - INTRODUCTION</b>	page 6
<b>2 - ENVIRONNEMENT DE STAGE</b>	page 6
2.1 Présentation de la Société	page 6
<b>3 - DÉFINITION DU PROJET</b>	page 7
3.1 Analyse des besoins	page 7
3.2 - Organisation des tâches	page 7
<b>4 - MAQUETTAGE</b>	page 8
4.1 - Design system	page 9
4.1.1 Couleurs	page 9
4.1.2 Composants	page 9
4.2 - Réalisation de la maquette	page 9
4.2.1 Page d'accueil	page 9
4.2.2 Page services	page 10
4.2.3 Page contact	page 11
4.2.4 Page article	page 11
4.2.5 - Page d'accueil version ordinateur	page 11
4.3 - Matériels et technologies utilisés	page 11
<b>5 - INTÉGRATION</b>	page 12
5.1 - Installation de la librairie Bootstrap et du préprocesseur SASS	page 12
5.1.1 Bootstrap	page 12
5.1.2 SASS	page 12
5.2 - Media Queries	page 13
<b>6 - WORDPRESS</b>	page 14
6.1 - Installation de WordPress	page 15
6.2 - Création du thème	page 15
6.2.1 Le fichier style.css	page 15
6.2.2 Le fichier functions.php	page 16
6.2.3 Le fichier screenshot.png	page 17
6.2.4 Création des templates	page 17

6.2.5 header.php et footer.php	page 18
6.2.6 Affichage de la page d'accueil	page 18
6.2.7 Affichage des pages et des articles	page 19
6.3 - Création d'extensions	page 20
6.4 - Référencement	page 22
6.4.1 La balise head	page 22
6.4.2 La balises h1 à h6	page 22
6.4.3 L'attribut alt des images	page 22
6.4.4 Utilisation de l'extension Yoast SEO	page 22
<b>7 - MISE EN LIGNE</b>	page 24
<b>8 - CONCLUSIONS</b>	page 27

## PARTIE 2 : LA NÎMES'ALERIE

<b>1 - BASE DE DONNÉES</b>	page 28
1.1 - Le MCD	page 28
1.2 - Crédation de la base de données	page 29
1.3 - Crédation des entités	page 30
1.3.1 Les relations	page 31
1.3.2 Migration en base de données	page 32
<b>2 - SYSTÈME D'AUTHENTIFICATION</b>	page 33
2.1 - Formulaire d'inscription	page 33
2.1.1 Crédation de l'entité User	page 33
2.1.2 Crédation du formulaire d'inscription	page 35
2.1.3 Crédation du formulaire de connexion	page 35
<b>3 - AFFICHAGE DES PRODUITS</b>	page 36
3.1 - Menu de navigation	page 36
3.2 - Affichage des produits par catégorie	page 37
3.3 - Page détail produit	page 38
<b>4 - VEILLE EN ANGLAIS</b>	page 39

# PARTIE 1 : HESTIA ÉNERGIES

## 1 - INTRODUCTION

Dans le cadre de ma formation « Développeur web et web mobile » chez Human Booster, j'ai effectué un stage de fin de formation d'une durée de 10 semaines.

J'ai intégré la Société Hestia Énergies, afin de réaliser son site Web.

L'entreprise Hestia Energies est spécialisée dans le dépannage, l'installation et l'entretien de systèmes de chauffage et de climatisation.

Je détaillerai dans ce rapport les moyens mis en œuvre pour la réalisation du projet, du maquettage à la mise en ligne du site.

Je présenterai les technologies que j'ai utilisées et décrirai mes réalisations en détail.

Cependant, mon stage ne couvrant pas toutes les compétences professionnelles demandées, je me permettrai de présenter un second projet afin de couvrir les compétences pour la partie Backend.

Je vous exposerai les étapes de la création d'un site e-commerce que j'ai pu réaliser.

## 2 - ENVIRONNEMENT DE STAGE

### 2.1 - Présentation de la Société



Créée en 2010, par Hervé Bénier, Hestia énergies est une société spécialisée dans les systèmes de chauffage et de climatisation.

Située à Feurs dans la Loire, l'entreprise compte 5 salariés.

Hestia énergies propose ses services de dépannage, d'entretien et d'installation d'équipements aux particuliers et aux collectivités.

Son activité concerne tous types de chaudières, pompes à chaleur, poêles à granulés, climatisations réversibles ...

Elle intervient pour ses clients essentiellement dans le département de la Loire.

## 3 - DÉFINITION DU PROJET

### 3.1 - Analyse des besoins

Afin de réaliser un site correspondant aux besoins de l'entreprise, j'ai dû échanger avec le dirigeant sur ses exigences.

Celui-ci souhaitait un site simple d'utilisation avec un menu contenant plusieurs onglets, un pour l'installation et le remplacement d'équipements, un pour l'entretien d'équipements, un pour le dépannage, un onglet contact avec un formulaire et un onglet actualités regroupant les dernières nouveautés comme les aides de l'état pour la rénovation énergétiques.

L'essentiel pour lui, était que le site soit bien référencé

Il avait aussi besoin de pouvoir modifier lui-même les articles contenus dans l'onglet actualités.

Concernant le choix des couleurs, celui-ci souhaitait adapter son site en fonction des couleurs du logo de l'entreprise.

### 3.2 - Organisation des tâches

J'ai utilisé l'outil de gestion de projet Trello, afin d'optimiser l'organisation des tâches de mon projet.

J'ai listé tous les besoins de l'entreprise et des visiteurs dans une première planche nommée Backlog.

J'ai ensuite créé une autre planche nommée "TO DO", listant les tâches techniques à effectuer ( maquettage, intégration ... ).

J'ai ajouté deux autres planches nommées "DOING" et "DONE", afin de trier les tâches.

Répartition des tâches avec l'outil Trello :

The screenshot shows a Trello board with the following structure:

- BACKLOG** (Leftmost column):
  - VISITEUR USER STORY: Trouver facilement ce que je recherche sur le web (bon référencement)
  - VISITEUR USER STORY: Voir le détail des produits et services de l'entreprise (pages de détail des produits et services)
  - VISITEUR USER STORY: Pouvoir contacter facilement l'entreprise (page de contact avec toutes les infos)
  - USER STORY ENTREPRISE: Création d'un espace admin pour que l'entreprise puisse ajouter des articles
  - VISITEUR USER STORY: Trouver une entreprise qualifiée et de confiance (page à propos, contenu, références, projets aboutis...)
  - USER STORY ENTREPRISE: Ajouter une carte
- TO DO** (Second column from left):
  - sprint 5/11 Maquettage TECHNIQUE: Figma: page contact / page renseignements entretien
  - sprint 5/11 Maquettage TECHNIQUE: page installation replacement
  - sprint 5/11 Maquettage TECHNIQUE: Figma : page entretien
  - sprint 5/11 Maquettage TECHNIQUE: figma, page accueil ordinateur
  - sprint 12/11 Intégration TECHNIQUE: Page accueil
  - sprint 12/11 Intégration TECHNIQUE: Page single type
  - + Ajouter une carte
- DOING** (Third column from left):
  - sprint 5/11 Maquettage TECHNIQUE: Figma : page dépannage mobile
  - + Ajouter une carte
- DONE** (Rightmost column):
  - sprint 5/11 Maquettage TECHNIQUE: Maquette Figma : page accueil mobile
  - + Ajouter une carte

Mes missions durant ce stage, étaient les suivantes:

- Réalisation de la maquette
- Intégration de la maquette
- Installation de WordPress
- Création d'un thème WordPress
- Création de plugins WordPress
- Réalisation du site
- Mise en ligne du site

## 4 - MAQUETTAGE

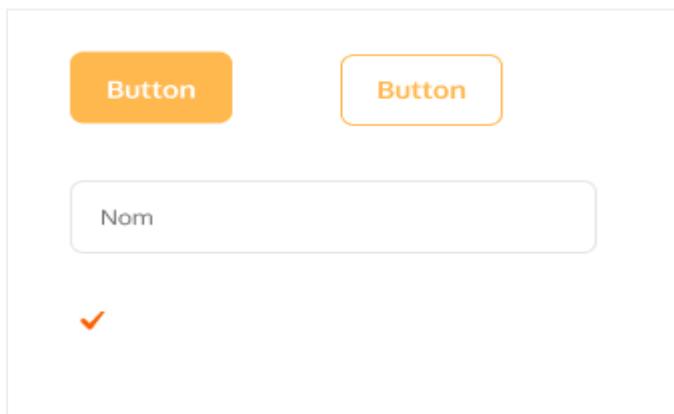
### 4.1 - Design system

#### 4.1.1 Couleurs

Le logo étant déjà existant, je me suis basée sur celui-ci pour les couleurs



#### 4.1.2 Composants



### **4.2 - Réalisation de la maquette**

#### 4.2.1 Page d'accueil

J'ai construit la maquette en suivant le concept du mobile first qui consiste à créer le site en version mobile puis de l'adapter par la suite à l'ordinateur.

Pour une version mobile, le design doit être simple, épuré de manière à faciliter la navigation.

La page d'accueil est composée d'un en-tête, avec le logo de l'entreprise, son numéro de téléphone et un menu de navigation.

Un carrousel présente les offres et les actualités, en dessous, se trouve le slogan de la Société.

On trouve ensuite trois cartes, chacune représentant un service avec un bouton pour accéder à la page du service concerné.

En dessous se trouve la section actualité qui affichera le dernier article publié, puis une section qualifications, une section contact et une section partenaires qui affiche les marques partenaires de l'entreprise.

Pour finir, le footer avec les coordonnées de contact de la Société, ainsi que le menu de navigation et les mentions légales.

Header



## Carte remplacement



### Remplacement

- ✓ Respect de votre budget et de vos envies
- ✓ Des équipements adaptés
- ✓ Des professionnels à votre écoute

[Remplacer mes équipements](#)

## Section contact

**Une question, une demande de devis ?**

Devis 100 % gratuit et sans engagement !

[Contactez-nous](#)

## 4.2.2 Page service

La page service est composée d'un titre en en-tête, avec une image en fond.

Puis en dessous, se trouve une description simple du service sous forme de liste, avec un bouton de contact.

Sur chaque page sera affiché le header et le footer, on retrouvera aussi les sections qualifications et partenaires.

### Contenu de la page entretien =>

**HESTIA vous accompagne pour l'entretien et la maintenance de vos équipements.**

Grâce à notre équipe de professionnels qualifiés sur toutes marques, nous réalisons l'entretien complet de vos équipements tels que :

- ✓ Chauffage au gaz, au fioul ...
- ✓ Chaudière ( gaz, fioul, bois granulés ... )
- ✓ Pompe à chaleur
- ✓ Climatisation
- ✓ Chauffage réversible

**Nous vous proposons des solutions adaptées à vos besoins !**

[Entretenir mes équipements](#)

#### 4.2.3 Page contact

Sur la page de contact se trouve un titre sur une image de fond comme pour chaque page du site, suivi d'un formulaire de contact.

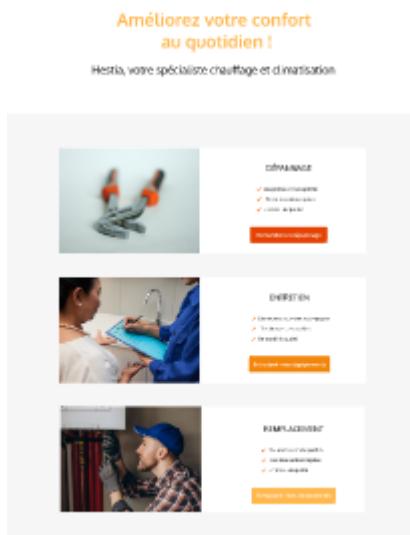
#### 4.2.4 Page article

Tout comme les pages du site, cette page contient le header, le titre, le contenu et le footer.

#### 4.2.5 - Page d'accueil version ordinateur

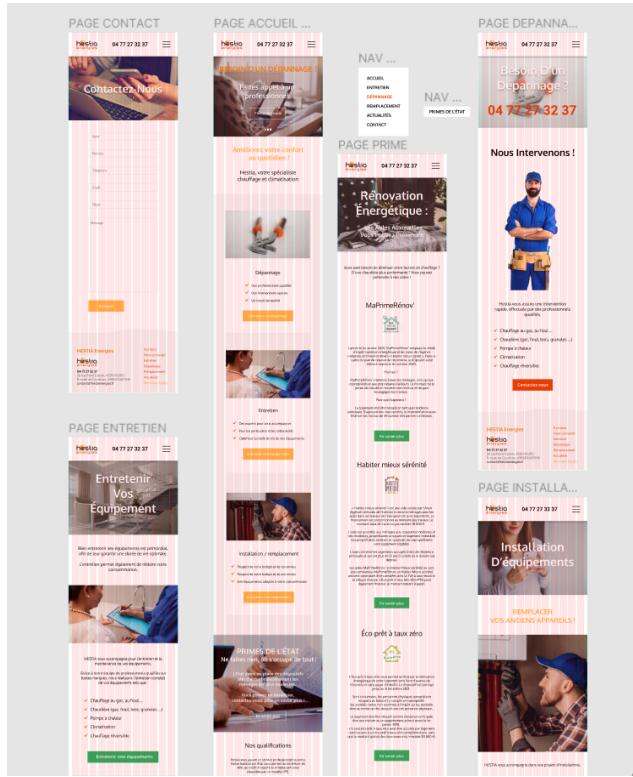
La page d'accueil version ordinateur reste semblable à la version mobile, il y a des modifications au niveau de l'alignement des cartes, du footer et du menu.

##### **Extrait de la page d'accueil version ordinateur**



#### **4.3 - Matériels et technologies utilisés**

Pour la réalisation de cette maquette j'ai utilisé l'outil de prototypage Figma, j'ai travaillé sur un ordinateur de bureau, et je disposais aussi d'un pc portable.



Un cadre sur Figma est appelé *frame*, pour chaque page du site j'ai défini une taille de frame, pour les pages en version mobile, je me suis basée sur une frame de 414 pixels de large, avec une grille de douze colonnes.

J'ai construit ma page d'accueil version ordinateur sur une frame de 1920 pixels de large, en utilisant également une grille de douze colonnes.

Un fois celle-ci terminée, je l'ai envoyée pour validation à Hestia Energies.

## 5 - INTÉGRATION

J'ai réalisé la partie Frontend du site en me basant sur la maquette, j'ai réalisé la page d'accueil, les pages services, l'article d'actualité et le formulaire de contact.

### 5.1 - Installation de la librairie Bootstrap et du préprocesseur SASS

#### 5.1.1 Bootstrap

Après avoir créé mon dossier *hestia-intégration*, j'ai créé le fichier *index.html*, puis un dossier *Bootstrap* dans lequel j'ai copié le fichier *bootstrap.css*, et le fichier *bootstrap.js* que

j'avais préalablement téléchargé depuis le site web de Bootstrap. J'ai utilisé Bootstrap 5, la dernière version de Bootstrap.

J'ai importé le fichier *bootstrap.js* dans mon *index.html* en bas de la balise *body*.

### 5.1.2 SASS

Ayant déjà Node.js et Npm installés sur mon pc, j'ai initialisé le fichier *package.json* avec la commande **npm init**. Puis j'ai installé SASS avec la commande **npm install sass**, après avoir créé mes fichiers scss. J'ai créé un script dans le fichier *package.json*, afin de pouvoir lancer la compilation avec la commande **npm run sass**.

```
"scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1",  
    "sass": "sass --watch ./sass/style.scss:./style.css --style compressed"  
},
```

Grâce au **--watch**, la mise à jour se fera automatiquement tant que le script tourne dans le terminal, et le **--style compressed** générera une version CSS compressée de mon SCSS.

J'ai importé tous mes fichiers SCSS et le CSS de Bootstrap dans le fichier que j'ai nommé *style.scss*.

### **5.2 - Media Queries**

Comme pour le maquettage, j'ai suivi la règle du mobile first, j'ai donc commencé à coder la page d'accueil du site en version mobile en me basant sur une taille minimale de 320 pixels. J'ai personnalisé les points de ruptures en fonction du contenu de chaque section.

Pour mettre en place un point de rupture, il suffit d'écrire une media query, et ayant commencé en version mobile, j'ai dû mettre en place des media queries pour les tailles supérieures.

```
@media screen and (min-width: 400px) {  
  
    nav {  
        padding: 0 1rem;  
        p {  
            font-size: large;  
            img {  
                width: 20px;  
            }  
        }  
    }  
}
```

Ici, un exemple de media query avec un point de rupture (breakpoint) de 400px pour l'adaptation de la barre de navigation.

Pour définir un point de rupture, j'ai utilisé le navigateur, et j'ai diminué petit à petit la fenêtre. Dès que j'observais un affichage devant être ajusté, j'appliquais les modifications dans le CSS.



**Rappel** : un point de rupture est la taille pour laquelle l'affichage change sur le site.

## 6 - WORDPRESS

### 6.1 - Installation de WordPress

Pour pouvoir faire mon site avec le CMS WordPress, il a fallu installer Wordpress. J'ai donc téléchargé la dernière version de WordPress (5.8.2) sur le site [wordpress.org](https://wordpress.org).

J'ai créé une base de données en me servant de MySql Workbench.

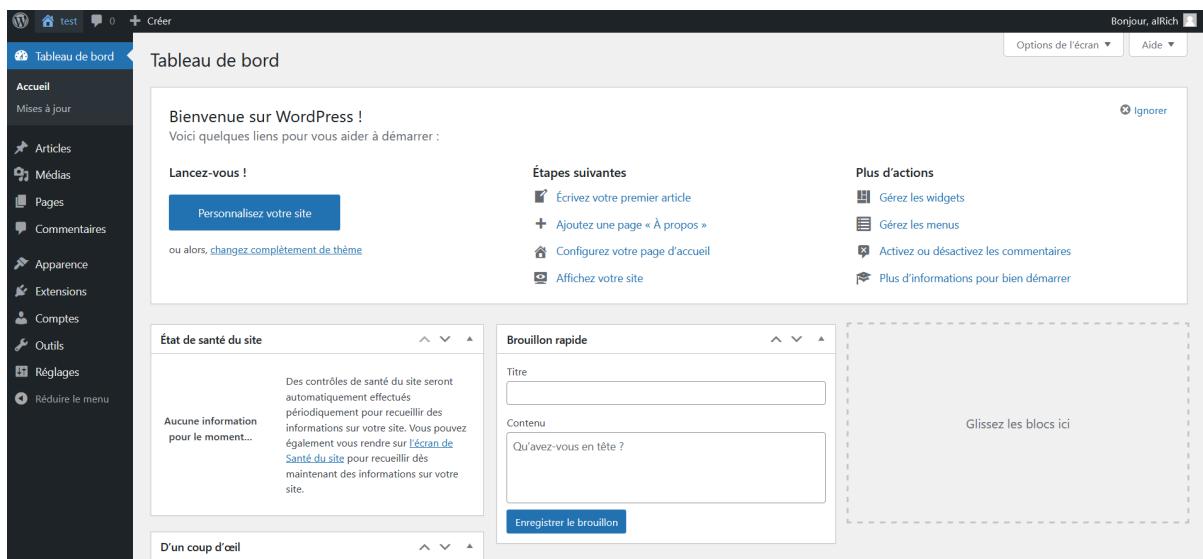


J'ai décompressé le fichier zip téléchargé plus tôt, et je l'ai renommé et placé dans l'espace www afin de pouvoir travailler en local grâce à la plateforme de développement Web WAMP.



**Rappel** : WAMP permet de faire fonctionner localement des scripts php, c'est un environnement qui comprend trois serveurs (Apache, MySql et MariaDB), un interpréteur de script (PHP), ainsi que phpMyAdmin pour l'administration Web des bases MySQL.

Pour accéder à mon site en local, je suis allée sur mon navigateur et me suis connectée à mon site WordPress, en tapant l'url `localhost/hestia..`, WordPress m'a redirigée vers un lien de configuration pour la connexion à la base de données. Une fois les informations de la base de données saisies, et les identifiants de connexion de l'administrateur renseignés, l'installation peut commencer, le site est prêt. On accède au tableau de bord par l'url `localhost/hestia/wp-admin`.



## 6.2 - Création du thème

Afin d'avoir un site 100% personnalisé, j'ai décidé de créer mon propre thème WordPress, cela fut une découverte pour moi car nous n'avions pas eu l'occasion de le voir durant les cours de WordPress.

Pour y parvenir je me suis servie de la documentation de WordPress, j'ai regardé le code des autres thèmes et stackoverflow m'a aussi beaucoup aidée.

J'ai commencé par créer le dossier de mon thème dans le dossier wp-content/themes. Afin qu'il puisse être activé, un thème a besoin de 4 fichiers :

- index.php
- style.css
- functions.php
- screenshot.png

### 6.2.1 Le fichier style.css

C'est le fichier style.css qui permet à WordPress de savoir que c'est un thème, grâce aux commentaires insérés en en-tête de la page.

```
wp-content > themes > hestia-theme > # style.css > ...
1  /*
2  Theme Name: Hestia Energies
3  Author: Alice Richardeau
4  Author URI: https://github.com/alicerichardeau
5  Description: Ce theme est réalisé spécialement pour le site web de la société hestia énergies.
6  Version: 1.0
7  */
```

On indique le nom du thème, son auteur, une description et le numéro de version.

## 6.2.2 Le fichier functions.php

Ce fichier est essentiel puisque c'est ici que l'on doit activer toutes les fonctionnalités nécessaires de WordPress.

```
<?php
// Add support for highlighted images
add_theme_support( 'post-thumbnails' );
// customise the size of the images
add_image_size('my-fun-size', 320, 150, true);

// Automatically add the site title to the site header
add_theme_support( 'title-tag' );

// SCRIPT LOADS
// #####
function wpdocs_theme_name_scripts() {
    wp_register_script( 'bootstrap-js', get_template_directory_uri() . '/bootstrap/bootstrap.js', [], false, true);
    wp_enqueue_script('bootstrap-js');
}
add_action( 'wp_enqueue_scripts', 'wpdocs_theme_name_scripts' );

// SETUP
// #####
function edn_theme_setup() {
    register_nav_menus(
        array(
            'nav' => __( 'navbar'),
            'nav_footer' => __('navbar_footer')
        )
    );
}

add_action( 'after_setup_theme', 'edn_theme_setup' );
```

Ci-dessus, le fichier *functions.php*.

La fonction **add\_theme\_support()** permet d'ajouter des images mises en avant sur les pages et articles.

**add\_image\_size()** permet la personnalisation de la taille des images. Elle prend en paramètre un nom, une largeur, une hauteur et un booléen, si faux, l'image sera mise à l'échelle (par défaut), si vrai, l'image sera recadrée aux dimensions spécifiées en utilisant les positions centrales.

La fonction **wpdocs\_theme\_name\_scripts()** est une fonction que j'ai créée afin d'ajouter les scripts JS de bootstrap.

**wp\_register\_script()** enregistre le script, elle prend en paramètres :

- le nom du script,
- la source,
- un tableau de scripts enregistrés, ici le tableau est vide,
- une chaîne spécifiant le numéro de version du script, ici la version est définie sur `false`, un numéro de version est automatiquement ajouté, égal à la version WordPress actuellement installée, s'il est défini sur `null`, aucune version n'est ajoutée.
- un booléen qui permet de placer le script ou non en bas du *body* plutôt que dans le *head*. Ici, `true` signifie qu'il sera placé en bas de la balise *body*.

`wp_enqueue_script()` enregistre le script, et prend le nom du script en paramètre.

La fonction `add_action()` ajoute l'action, elle prend en paramètres le nom de l'action et le nom de la fonction que l'on souhaite exécuter.

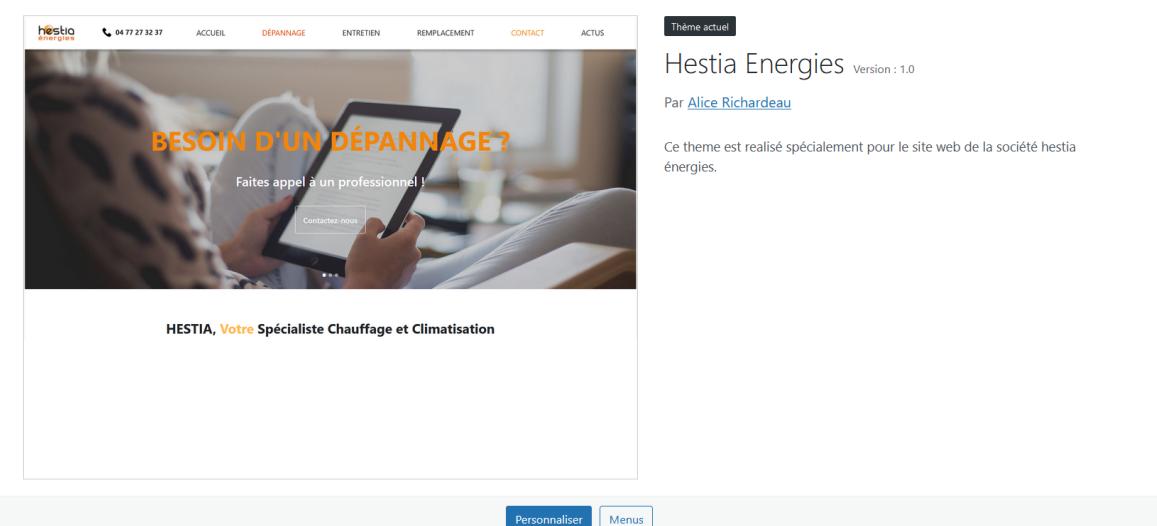
 **Rappel** : Les actions sont les crochets que WordPress lance à des points spécifiques pendant l'exécution.

J'ai créé la fonction `edn_theme_setup()` dans laquelle j'ai utilisé la fonction WordPress `register_nav_menus()` afin de définir les menus de navigation du site.

### 6.2.3 Le fichier screenshot.png

Il permet d'afficher un aperçu du thème dans l'interface d'administration de WordPress.

Une fois tous ces fichiers créés, on peut activer le thème dans le tableau de bord WordPress.



Une fois mon thème actif, il me restait à insérer mon HTML et mon CSS.

Pour le CSS, j'ai simplement copié les fichiers de mon intégration, ainsi que le dossier bootstrap.

Pour le contenu ce fut un peu plus difficile, j'ai dû me documenter pour savoir où et comment insérer mon HTML.

### 6.2.4 Crédit des templates

J'ai créé un dossier nommé template-parts dans lequel j'ai créé un fichier pour chaque section de mon site. A l'intérieur de chaque fichier, j'y ai collé le code HTML, j'ai créé un fichier `contact.php` pour la partie contact, un fichier `services.php`, qui regroupe le code HTML de la section services, un fichier `partenaires.php` qui contient la section partenaires ...

## 6.2.5 header.php et footer.php

A la racine de mon thème, j'ai créé les fichiers *header.php* et *footer.php*.

Dans le fichier *header.php*, on retrouve le début de notre document HTML, jusqu'à l'ouverture de la balise body.

```
<!DOCTYPE html>
<!-- Displays the language attributes for the 'html' tag -->
<html <?php language_attributes(); ?>>

<head>
    <!-- Displays information about the current site -->
    <meta charset=<?php bloginfo('charset'); ?>" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
    <link rel="stylesheet" type="text/css" href="<?php echo get_template_directory_uri();?>/style.css" />
    <!-- Starts the wp_head action which prints scripts or data in the head tag on the front end -->
    <?php wp_head(); ?>
</head>

<!-- ##### BODY ##### -->
<!-- get css class names -->
<body <?php body_class(); ?>>
    <div class="bloc-page">
        <!-- Starts the wp_body_open action triggered before the body is opened -->
        <?php wp_body_open(); ?>
```

**language\_attributes()** : Affiche les attributs de langue pour la balise HTML

**bloginfo()** : Affiche les informations du site, ici la valeur 'charset' fait référence à « UTF-8 », qui est l'encodage par défaut de WordPress.

**get\_template\_directory\_uri()** : Récupère l'URI du répertoire du thème.

 **Rappel** : Différence entre URI et URL ? L'URL est une spécialisation de l'URI. L'URI est un identifiant alors que l'URL fournit des informations sur la manière d'obtenir une ressource.

**wp\_head()** : Lance l'action *wp\_head* qui lance les scripts ou les données dans la balise *head*.

**body\_class()** : récupères les noms des classes CSS

**wp\_body\_open()** : Lance l'action *wp\_body\_open* déclenchée avant l'ouverture du *body*.

Le fichier *footer.php* affiche la fin du document HTML.

```
<?php wp_footer(); ?>
</div>
</body>
</html>
```

**wp\_footer()** : Lance l'action *wp\_footer* qui imprime des scripts ou des données avant la fermeture du *body*.

## 6.2.6 Affichage de la page d'accueil

J'ai créé un fichier *front-page.php* en me basant sur la documentation du site [wordpress.org](https://wordpress.org) sur la hiérarchie des templates. WordPress effectue une recherche hiérarchisée des templates jusqu'à ce qu'il trouve un fichier de modèle correspondant et le fichier *front-page.php* est utilisé pour la page d'accueil du site.

```
<?php
/*
Template Name: home page
*/
?>
<?php get_header(); ?>

<?php get_template_part( 'template-parts/header' ); ?>
<?php get_template_part( 'template-parts/carousel' ); ?>
<?php get_template_part( 'template-parts/slogan' ); ?>
<?php get_template_part( 'template-parts/services' ); ?>
<?php get_template_part( 'template-parts/actualities' ); ?>
<?php get_template_part( 'template-parts/qualifications' ); ?>
<?php get_template_part( 'template-parts/contact' ); ?>
<?php get_template_part( 'template-parts/partenaires' ); ?>
<?php get_template_part( 'template-parts/footer' ); ?>
<?php get_footer(); ?>
```

Dans ce fichier *front-page.php*, j'ai récupéré *header.php* et *footer.php*.  
Ensuite, la fonction wordpress [get\\_template\\_part\(\)](#) m'a permis de récupérer chaque partie de mon dossier *template-parts* que je souhaite voir afficher sur ma page d'accueil.

### 6.2.7 Affichage des pages et des articles

J'ai créé le fichier *page.php* pour le template des pages.

```
<?php /*
Template Name: Page standard
*/
?>
<?php get_header(); ?>

<?php get_template_part( 'template-parts/header' ); ?>
<?php get_template_part( 'template-parts/title' ); ?>
<?php the_content(); ?>

<div class="grey-light-background">
    <?php get_template_part( 'template-parts/qualifications'); ?
</div>

<?php get_template_part( 'template-parts/partenaires'); ?>
<?php get_template_part( 'template-parts/footer' ); ?>
<?php get_footer(); ?>
```

Et pour les articles, le fichier *category.php*

```
wp-content > themes > hestia-theme > category.php
1  <?php
2  /*
3  Template Name: Article standard
4  */
5  ?>
6  <?php get_header(); ?>
7
8  <?php get_template_part( 'template-parts/header' ); ?>
9
10 <?php get_template_part( 'template-parts/title' ); ?>
11
12 <?php the_content(); ?>
13
14 <?php get_template_part( 'template-parts/footer' ); ?>
15
16 <?php get_footer(); ?>
```

*the\_content()* est la fonction WordPress qui permet l'affichage du contenu de chaque page.

### 6.3 - Crédation d'extensions

J'ai réalisé une première extension WordPress qui permet d'afficher un formulaire de contact, une seconde qui modifie l'interface de connexion et une dernière qui adapte le tableau de bord en fonction du rôle de l'utilisateur.

Pour créer une extension, il faut se rendre dans le dossier *plugins*, créer un dossier du nom de son extension. Il faut ensuite, un fichier .php, dans lequel on y insère un commentaire. Sans ce commentaire, l'extension ne fonctionne pas.

```
<?php
/*
Plugin Name: Alice Dashboard Interface
Plugin URI: http://wordpress.org/plugins/interface-customer/
Description: Customer interface
Author: Alice Richardau
Version: 1.0
Author URI:
*/
```

Ci-dessus, un extrait du fichier *interface-customer.php*, situé dans le dossier *interface-customer*, le plugin qui permet de modifier l'interface du tableau de bord du client.

Ensuite, il suffit de placer le code dans ce même fichier.

```
//CREATION OF THE OWNER ROLE
//#####
function owner_roles() {
    |   add_role( 'owner_role', 'Propriétaire', get_role( 'administrator' )->capabilities );
}
add_action( 'init', 'owner_roles' );
```

Ici, la fonction `owner_roles()` ajoute le rôle propriétaire, on lui attribue les mêmes capacités qu'un administrateur. On ajoute l'action, 'init' signifie que l'action se déclenche lors de l'initialisation de l'ensemble du script WordPress.

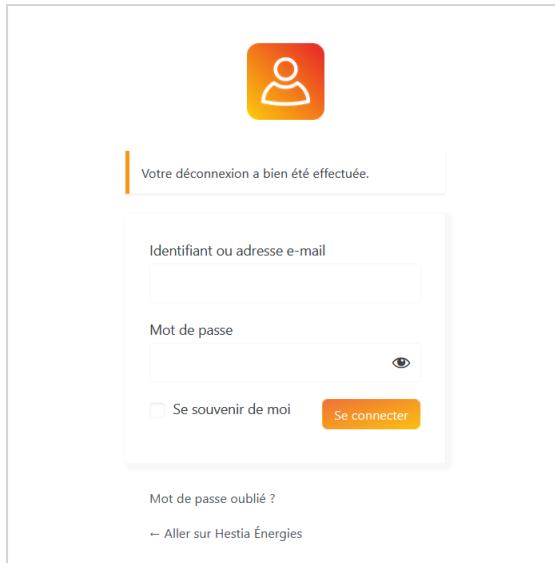
```
//DELETING ACCESS TO UNNECESSARY PAGES
//#####
function remove_design() {
    if (!current_user_can('administrator') ) [
        // CHANGE CSS
        // #####
        >
        <style type="text/css">
            #footer-thankyou, #footer-upgrade, #contextual-help-link, .user-comment-shortcuts-wrap, .user-language-wrap, .user-admin-color-wrap,
            display: none;
        </style>
    </?php
        // DELETE MENUS
        // #####
        remove_menu_page('edit-comments.php');
        remove_menu_page('themes.php');
        remove_menu_page('plugins.php');
        remove_menu_page('user-new.php');
        remove_menu_page('profile.php?wp_http_referer=%2Fhestia%2Fwp-admin%2Fusers.php');
        remove_menu_page('tools.php');
        remove_menu_page('options-general.php');
        remove_menu_page('edit.php?post_type=page');
        remove_menu_page('admin.php');
    ]
}

add_action( 'admin_menu', 'remove_design' );
```

Un fois le rôle défini, j'ai créé la fonction `remove_design()` qui supprime les éléments inutiles. J'ai indiqué dans une condition, que si l'utilisateur n'est pas administrateur, j'applique le CSS dans la balise style et je supprime les pages auxquelles l'accès est refusé.

<input type="checkbox"/> <a href="#">Alice Dashboard Interface</a>	Customer interface
<a href="#">Désactiver</a>	Version 1.0   Par Alice Richardeau   <a href="#">Aller sur le site de l'extension</a>
<input type="checkbox"/> <a href="#">Alice Form</a>	Creation of contact forms for wordpress
<a href="#">Désactiver</a>	Version 1.0   Par Alice Richardeau   <a href="#">Aller sur le site de l'extension</a>
<input type="checkbox"/> <a href="#">Alice Login Style</a>	Customise the login interface
<a href="#">Désactiver</a>	Version 1.0   Par Alice Richardeau   <a href="#">Aller sur le site de l'extension</a>

Ci-dessus, les trois extensions dans le tableau de bord, dans le menu extensions.



Ici, l'interface de connexion personnalisée avec l'extension *Login Style*.

## 6.4 - Référencement

Bien référencer son site Web est primordial. Afin d'apparaître le plus haut possible dans les résultats de recherche des internautes, il est nécessaire de bien structurer son code, d'utiliser les bons mots clés et les bonnes balises HTML.

### 6.4.1 La balise *head*

À l'intérieur de la balise <head> d'un document HTML, on retrouve de nombreuses informations avec notamment des balises de métadonnées et des scripts. Pour le référencement, il faut veiller à bien renseigner le titre dans la balise title et bien remplir la balise *meta description*.

### 6.4.2 La balises *h1* à *h6*

Il faut veiller à bien utiliser les balises *h1* à *h6*, la hiérarchie des rubriques permet aux moteurs de recherche de mieux comprendre le contenu de la page. Il est conseillé d'utiliser un *h1* par page et de diviser davantage le contenu de manière logique en sections avec *h2*. Si nécessaire, on peut encore segmenter les sections *h2* avec *h3*, *h4*, etc.

### 6.4.3 L'attribut *alt* des images

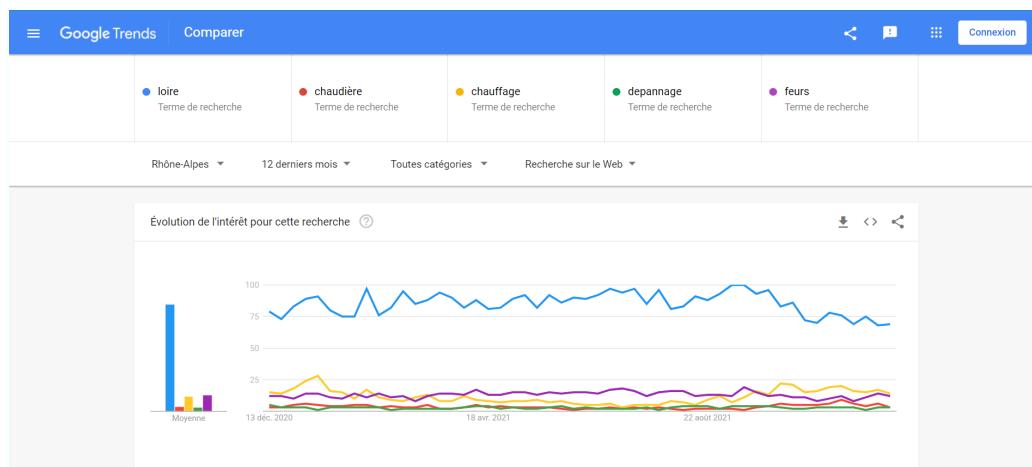
Le texte alternatif joue un rôle majeur dans l'optimisation de l'image. Il rend les images accessibles à la fois aux moteurs de recherche et aux personnes.

### 6.4.4 Utilisation de l'extension Yoast SEO

Pour ce projet WordPress, j'ai utilisé l'extension Yoast SEO, afin d'optimiser le référencement du site. Cette extension est très pratique car elle se charge de remplir les balises meta pour chaque page et articles du site.



« Google Trends » permet de comparer les évolutions des demandes sur tel ou tel produit ou concept. Il indique pour une à cinq requêtes la fréquence à laquelle elles ont été cherchées dans le moteur de recherche.



Ici, j'ai effectué une comparaison sur plusieurs mots clés en relation avec le dépannage.

Avec Yoast, j'ai défini la requête cible ci-dessous en fonction des résultats obtenus avec Google Trends.

Requête cible ?

Obtenir des requêtes cibles liées

Puis, j'ai renseigné le titre SEO (le titre qui apparaît dans les résultats de recherche du navigateur), le slug (slug de l'url) et la meta description (texte descriptif qui apparaît dans les résultats de recherche du navigateur).

Titre SEO	<input type="button" value="Insert variable"/>
Dépannage chauffage à Feurs   42 Loire	
	
Slug	
depannage-chauffage-feurs-loire	
Méta description	<input type="button" value="Insert variable"/>
Besoin d'un dépannage pour votre chauffage ? Faites appel à un chauffagiste professionnel dans le département de la Loire 42 ...	
	

Voici, le code HTML généré dans la balise head :

```
<!--This site is optimized with the Yoast SEO plugin v17.7 - https://yoast.com/wordpress/plugins/seo/-->
<title>Dépannage chauffage à Feurs | 42 Loire</title>
<meta name="description" content="Besoin d'un dépannage pour votre chauffage ? Faites appel à un chauffagiste professionnel dans le département de la Loire 42 ...">
<link rel="canonical" href="http://localhost/hestia/depannage-chauffage-feurs-loire/">
<meta property="og:locale" content="fr_FR">
<meta property="og:type" content="article">
<meta property="og:title" content="Dépannage chauffage à Feurs | 42 Loire">
<meta property="og:description" content="Besoin d'un dépannage pour votre chauffage ? Faites appel à un chauffagiste professionnel dans le département de la Loire 42 ...">
<meta property="og:url" content="http://localhost/hestia/depannage-chauffage-feurs-loire/">
<meta property="og:site_name" content="Hestia Énergies">
<meta property="article:modified_time" content="2021-12-13T11:03:46+00:00">
<meta property="og:image" content="http://localhost/hestia/wp-content/uploads/2021/11/depannage-plombier-outil.jpg">
<meta property="og:image:width" content="320">
<meta property="og:image:height" content="180">
<meta name="twitter:card" content="summary_large_image">
<meta name="twitter:label1" content="Durée de lecture est.">
<meta name="twitter:data1" content="2 minutes">
▶ <script class="yoast-schema-graph" type="application/ld+json">...</script>
<!-- Yoast SEO plugin.-->
```

On retrouve bien la balise *meta description* renseignée plus haut. On remarque aussi des balises *meta OG*. Ce sont les balises qui permettent aux réseaux sociaux, notamment Facebook et LinkedIn, de générer des aperçus de lien.

## 7 - MISE EN LIGNE

Mon site prêt, je devais le présenter à l'entreprise. J'ai décidé de le mettre en ligne avec un formulaire de connexion sécurisé pour le montrer en guise de démonstration.

J'ai choisi l'hébergeur PlanetHoster, et j'ai créé le nom de domaine *test-site.go.yj.fr*. Pour déplacer les fichiers de mon ordinateur vers le serveur, je n'ai pas eu besoin de client FTP (File transfert protocole).

En effet, avec PlanetHoster, il suffit de compresser le dossier de son site web, puis de le télécharger, c'est vraiment pratique, et cela permet de gagner du temps, car Filezilla n'est pas toujours très rapide.

Une fois les fichiers de mon site téléchargés dans le dossier *public\_html*, il fallait importer ma base de données. Pour cela, j'ai créé une nouvelle base de données depuis PlanetHoster, et j'ai importé mon ancienne base de données.

Dans celle-ci j'ai changé les url dans la table wp-options.

J'ai ensuite modifié l'utilisateur et le mot de passe dans le fichier *config.php*.

```
22 | // ** Réglages MySQL - Votre hébergeur doit vous fournir ces informations. ** //
23 | /** Nom de la base de données de WordPress. */
24 | define( 'DB_NAME', 'cgfpkgcr_hestia' );
25 |
26 | /** Utilisateur de la base de données MySQL. */
27 | define( 'DB_USER', 'cgfpkgcr_alRich' );
28 |
29 | /** Mot de passe de la base de données MySQL. */
30 | define( 'DB_PASSWORD', 'BB;~1%^B%1~F^Q1vFz' );
31 |
32 | /** Adresse de l'hébergement MySQL. */
33 | define( 'DB_HOST', 'localhost' );
34 |
--
```

### Sécuriser l'accès :

Maintenant, il fallait que je sécurise l'accès au site par un système de connexion sécurisé. J'ai donc créé un fichier *.htaccess* à la racine du répertoire qui se nomme */home* chez PlanetHoster.

```
1 AuthType Basic
2 AuthName "Accéder au site web"
3 AuthUserFile /home/cgfpkgcr/.htpasswd
4 require valid-user
```

Ici, j'ai mis en place une authentification de type Basic, dont le nom est "Accéder au site web", AuthUserFile permet de récupérer les identifiants et les mots de passe créés dans le fichier *.htpasswd* (que nous allons voir tout de suite). Sans les bons identifiants, l'accès sera refusé.

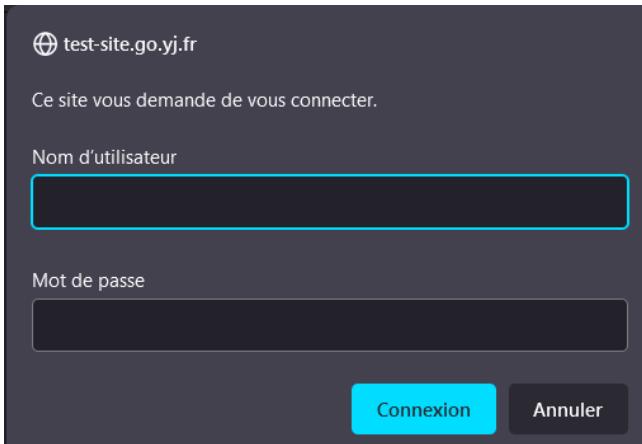
J'ai ensuite créé le fichier *.htpasswd* à la racine du répertoire *home*.

```
1 demo-hestia:$2y$10$bUqaV0u50fbKZ5lJrJT3Yh03mTH1bI/enjy62Yjd3yPrD.H86f5HtC
```

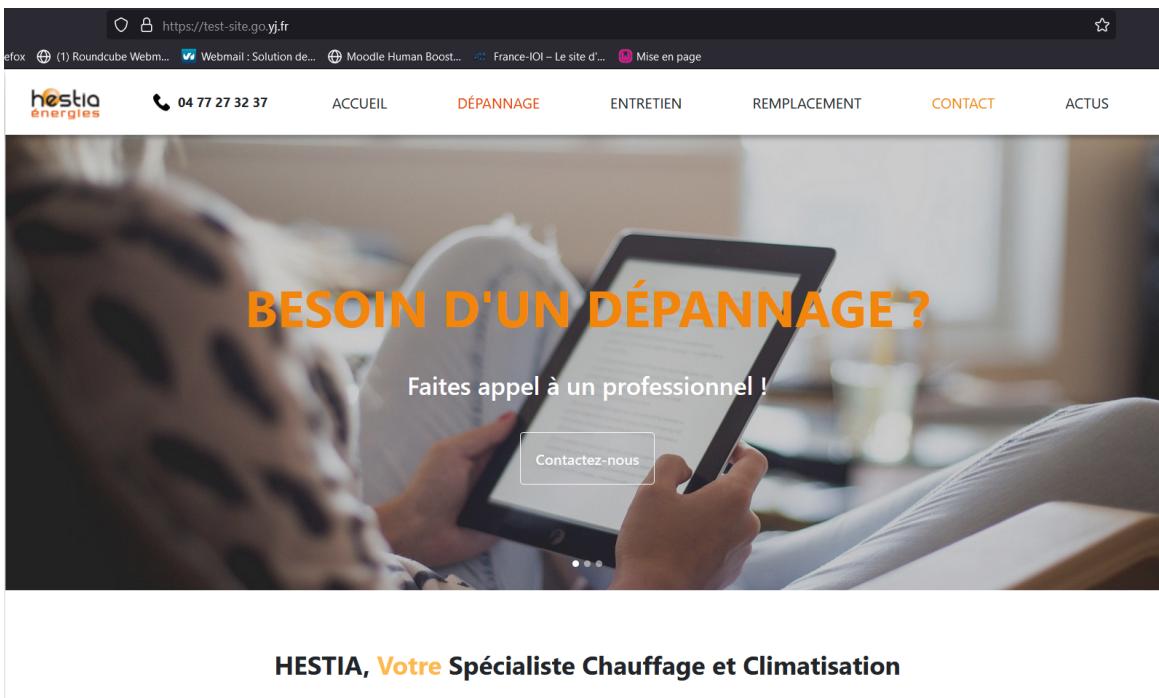
Ce fichier contient l'identifiant et le mot de passe de connexion.

Ici, j'ai hashé le mot de passe à l'aide du site <https://www.bcrypt.fr/> pour sécuriser le mot de passe.

 **Rappel** : Une fonction de hash cryptographique est un algorithme ou une série d'opérations mathématiques effectuées par un ordinateur. La division du hash se fait en 10 morceaux, dont chacun est de 4 caractères, ce qui donne une totalité de 40 caractères. Bien que le format soit différent, le hash est toujours identique.



The screenshot shows a dark-themed login page. At the top left is the URL "test-site.go.yj.fr". Below it, a message reads "Ce site vous demande de vous connecter.". There are two input fields: one for "Nom d'utilisateur" (username) and one for "Mot de passe" (password), both with blacked-out content. At the bottom are two buttons: a blue "Connexion" (Login) button and a grey "Annuler" (Cancel) button.



The screenshot shows the homepage of the Hestia Energies website. The URL in the address bar is "https://test-site.go.yj.fr". The header includes the Hestia Energies logo, a phone number "04 77 27 32 37", and navigation links for "ACCUEIL", "DÉPANNAGE" (highlighted in orange), "ENTRETIEN", "REPLACEMENT", "CONTACT", and "ACTUS". The main visual is a blurred image of a person holding a tablet. Overlaid text in large orange letters says "BESOIN D'UN DÉPANNAGE ?" and below it, in smaller white text, is "Faites appel à un professionnel !". A "Contactez-nous" button is visible at the bottom of the image area.

**HESTIA, Votre Spécialiste Chauffage et Climatisation**

La connexion était protégée, et je pouvais envoyer la démonstration du site web à l'entreprise.

## 8 - CONCLUSIONS

Pour conclure, cette expérience fut extrêmement enrichissante pour moi. Cela m'a permis de tester mes capacités à pouvoir travailler seule sur un projet du début à la fin.

J'appréhendais la création du thème WordPress, car c'était une première pour moi et au début, cela me paraissait compliqué. J'ai rencontré quelques difficultés, pour le thème et pour le plugin qui affiche le formulaire mais après des recherches, je m'en suis finalement sortie.

Une autre chose que j'appréhendais était la mise en ligne du site, qui au final n'est pas si difficile que ça.

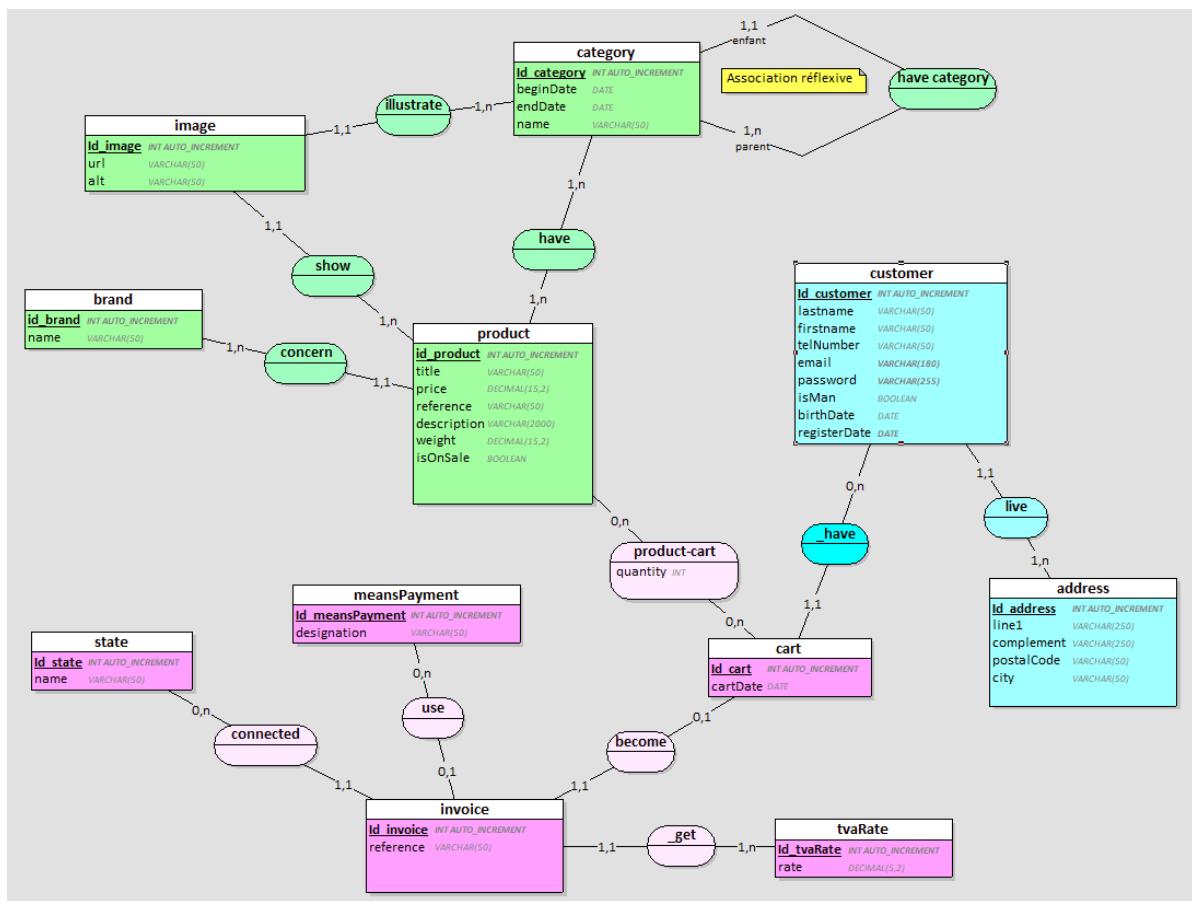
Je suis ravie de cette expérience, qui me donne l'envie d'évoluer dans cette voie et d'en apprendre encore et encore.

# PARTIE 2 : LA NÎMES'ALERIE

Mon stage ne couvrant pas tous les CCP demandés, je vous présente en parallèle la partie Backend d'un site e-commerce fictif que j'ai réalisé avec Symfony.

## 1 - BASE DE DONNÉES

### 1.1 - Le MCD





**Rappel :** Le MCD est une représentation graphique qui permet facilement et simplement de comprendre comment les différents éléments sont liés entre eux à l'aide de diagrammes codifiés dont les éléments suivants font partie :

- Les entités
- Les propriétés
- Les relations
- Les cardinalités

La conception de la base de données a débuté par la réalisation du Modèle Conceptuel des Données (MCD) à l'aide du logiciel Looping.

Les tables en vert sont celles associées à l'entité *product*.

*Category* : un produit peut avoir une ou plusieurs catégories et une catégorie peut avoir à un ou plusieurs produits.

Aussi, une catégorie enfant ne peut avoir qu'un et un seul parent, de même qu'une catégorie parent peut avoir un ou plusieurs enfants.

*Image* : est reliée à *product* et à *category*, une image illustre une et une seule catégorie, et une catégorie est illustrée par une ou plusieurs images, aussi, un produit est illustré par une ou plusieurs images et une image illustre un et un seul produit. Un produit est rattaché à une marque et une marque est rattachée à un ou plusieurs produits

J'ai créé une association *product-card* qui prend en paramètre une quantité, nécessaire pour connaître le nombre de produits dans un panier.

Les tables en bleu sont associées à l'entité *customer*.

Un client vit à une adresse et à une adresse peuvent vivre un ou plusieurs clients. Un client est aussi relié à un panier, un client est rattaché à aucun, un ou plusieurs paniers, et un panier est rattaché à un et un seul client.

Les tables en rose sont associées à l'entité *invoice*.

Une facture a un et un seul état, de même qu'un état est rattaché à aucune, une ou plusieurs factures. Une facture est aussi reliée à un et un seul moyen de paiement, un moyen de paiement est relié à aucune, une ou plusieurs factures. Une facture est associée à un et un seul taux de TVA, un taux de TVA est associé à une ou plusieurs factures. Enfin, un panier est rattaché à aucune ou une seule facture, et une facture est rattachée à un seul panier.

## 1.2 - Création de la base de données

Pour créer la base de données, Symfony met à disposition une commande. Les informations de connexion s'inscrivent dans le fichier *.env* situé à la racine du projet.

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/animalerie?serverVersion=5.7"
```

*root* est l'identifiant de l'utilisateur et si un mot de passe existe, il suffit de l'indiquer à la suite des “:”. Après le “@” il faut indiquer le serveur de la base de données, puis le nom de la base de données doit être indiqué après le “/”.

Après, il suffit de taper la commande suivante : *php bin/console doctrine:database:create*.

### 1.3 - Création des entités

Les tables sont créées dans le projet Symfony par la commande *php bin/console make:entity*.

Plusieurs champs sont à renseigner, le nom, puis les propriétés, leur type ...

Cela génère dans le dossier *Entity* de notre projet Symfony, un fichier :

```
/**  
 * Product  
 * @ORM\Entity(repositoryClass=ProductRepository::class)  
 */  
class Product  
{  
    /**  
     * @var int  
     *  
     * @ORM\Column(name="id", type="integer", nullable=false)  
     * @ORM\Id  
     * @ORM\GeneratedValue(strategy="IDENTITY")  
     */  
    private $id;  
  
    /**  
     * @var string|null  
     *  
     * @ORM\Column(name="title", type="string", length=50)  
     */  
    private $title;  
  
    /**  
     * @var string|null  
     *  
     * @ORM\Column(name="price", type="decimal", precision=15, scale=2)  
     */  
    private $price;  
  
    /**  
     * @var string|null  
     *  
     * @ORM\Column(name="reference", type="string", length=50)  
     */  
    private $reference;
```

Ici, un extrait de l'entité *Product*, la création de l'entité a généré automatiquement un fichier *ProductRepository*.

```

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getTitle(): ?string
    {
        return $this->title;
    }

    public function setTitle(?string $title): self
    {
        $this->title = $title;

        return $this;
    }

```

Les getters et les setters sont aussi automatiquement générés.

### 1.3.1 Les relations :

```

/**
 * @ORM\OneToMany(targetEntity=Image::class, mappedBy="product")
 */
private $images;

/**
 * @ORM\ManyToOne(targetEntity=Brand::class, inversedBy="products")
 */
private $brand;

/**
 * @ORM\Column(type="date")
 */
private $startDate;

/**
 * @ORM\Column(type="date", nullable=true)
 */
private $endDate;

/**
 * @ORM\ManyToMany(targetEntity=Category::class, inversedBy="products")
 */
private $categories;

```

Ci-dessus, un extrait de l'entité *Product*.

*Product* est relié à *images*, *brand* et *category*. J'ai donc créé trois propriétés avec le *make:entity*, pour le *type*, j'ai indiqué qu'il s'agissait d'une *relation*.

*images* : C'est une relation **OneToMany** car une image ne concerne qu'un seul produit et un produit contient plusieurs images.

*brand* : C'est une relation **ManyToOne** car une marque peut concerner plusieurs produits et un produit ne concerne qu'une marque.

*category* : C'est une relation **ManyToMany** car une catégorie peut contenir plusieurs produits et un produit peut être rattaché à plusieurs catégories.

### Rappel :

- `mappedBy` doit être spécifié sur le côté inversé d'une association (bidirectionnelle)
- `inversedBy` doit être spécifié sur le côté propriétaire d'une association (bidirectionnelle)

Pour savoir :

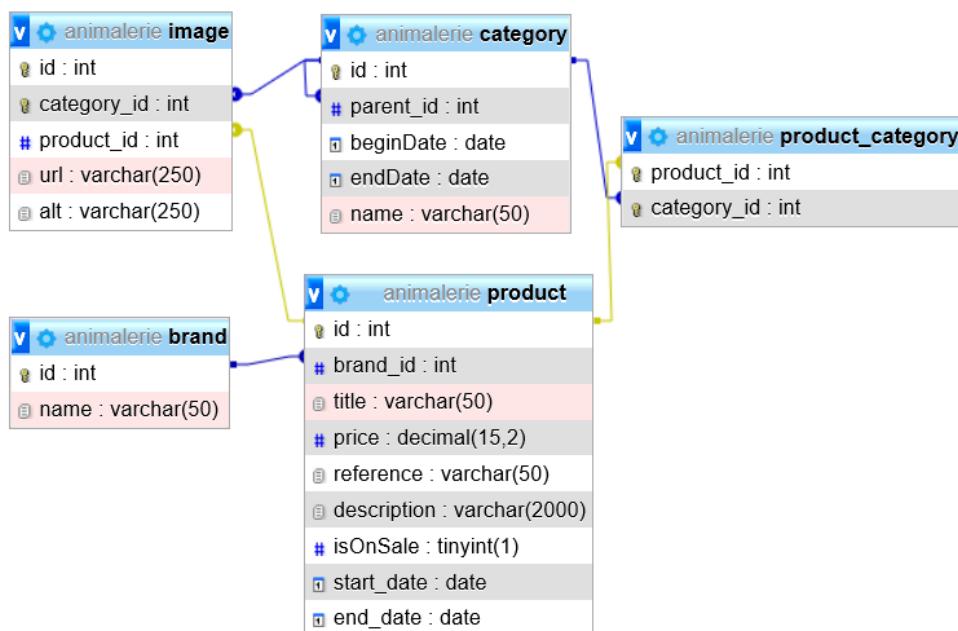
- `ManyToOne` est toujours le côté propriétaire d'une association.
- `OneToMany` est toujours le côté inversé d'une association
- Le côté propriétaire d'une association `OneToOne` est l'entité avec la table contenant la clé étrangère.
- On peut choisir soi-même le côté propriétaire d'une association `ManyToMany`.

### 1.3.2 Migration en base de données

Il faut ensuite effectuer la migration vers la base de données, deux commandes sont nécessaires :

- `php bin/console make:migration` : crée le fichier nécessaire à la transformation du code php en fonction sql dans le projet Symfony et permet de vérifier et modifier si besoin la requête à envoyer avant d'effectuer les modifications en base de données.
- `php bin/console doctrine:migrations:migrate` : exécute la requête créée dans le fichier « migration » et crée les tables dans la base de données.

Extrait du concepteur dans phpMyAdmin :



On voit ici, que les relations ont bien été créées, la relation ManyToMany entre *product* et *category*, a engendré la création d'une nouvelle table *product\_category*.

## 2 - SYSTÈME D'AUTHENTIFICATION

J'ai mis en place un système d'authentification des utilisateurs du site qui sert à la fois d'authentification pour les clients et pour le personnel de l'entreprise.

### 2.1 - Formulaire d'inscription

#### 2.1.1 Crédation de l'entité User

J'ai utilisé la commande *php bin/console make:user* qui fonctionne comme ceci :  
On nous demande le nom de l'entité souhaitée, si le mot de passe doit-être hashé, avec quel identifiant l'utilisateur va se connecter et si l'on souhaite que tout ceci soit stocké en base de données.

```
/**
 * @ORM\Entity(repositoryClass=UserRepository::class)
 */
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer", name="id")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=180, unique=true)
     */
    private $email;

    /**
     * @ORM\Column(type="json")
     */
    private $roles = [];

    /**
     * @var string The hashed password
     * @ORM\Column(type="string")
     */
    private $password;

    /**
     * @var string|null
     *
     * @ORM\Column(name="lastname", type="string", length=50, nullable=true)
     */
    private $lastname;
```

Ci-dessus, un extrait de l'entité *User*, on peut remarquer qu'une propriété *rôle* a été créée. Outre la création de l'entité, la commande modifie également le fichier *security.yaml*.

```
/**  
 * @see UserInterface  
 */  
public function getRoles(): array  
{  
    $roles = $this->roles;  
    // guarantee every user at least has ROLE_USER  
    $roles[] = 'ROLE_USER';  
  
    return array_unique($roles);  
}  
  
public function setRoles(array $roles): self  
{  
    $this->roles = $roles;  
  
    return $this;  
}
```

La propriété *rôles* est un tableau, par défaut, le rôle user est attribué.

```
providers:  
    # used to reload user from session & other features (e.g. switch_user)  
    app_user_provider:  
        entity:  
            class: App\Entity\User  
            property: email
```

*app\_user\_provider* permet de charger les utilisateurs à partir d'un stockage en base de données sur la base d'un identifiant qui est ici un email.

Symfony avec le SecurityBundle fournit des fonctionnalités de hashage et de vérification de mots de passe. Avec le *make:user* tout se génère automatiquement.

```
/**  
 * @ORM\Entity(repositoryClass=UserRepository::class)  
 */  
class User implements UserInterface, PasswordAuthenticatedUserInterface  
{  
    /**  
     * @ORM\Id  
     * @ORM\GeneratedValue  
     * @ORM\Column(type="integer", name="id")  
     */  
    private $id;
```

La class *User* implémente *PasswordAuthenticatedUserInterface* qui retourne le mot de passe hashé utilisé pour authentifier l'utilisateur.

```
password_hashers:  
    Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
```

Et dans le fichier `security.yaml`, le hash de mot de passe est configuré.

### 2.1.2 Crédation du formulaire d'inscription

Une fois l'entité `User` créée, j'ai réalisé le formulaire d'inscription à l'aide de la commande `php bin/console make:registration-form`.

Un fichier `RegistrationFormType.php` a été généré dans le dossier `Form`, un dossier `Registration` est créé dans le dossier `Template` et un `RegistrationController.php` est généré dans le dossier `Controller`.

```
/**  
 * @Route("/register", name="app_register")  
 */  
public function register(Request $request, UserPasswordHasherInterface $userPasswordHasherInterface): Response  
{  
    $user = new User();  
    $form = $this->createForm(RegistrationFormType::class, $user);  
    $form->handleRequest($request);  
  
    if ($form->isSubmitted() && $form->isValid()) {  
        // encode the plain password  
        $user->setPassword(  
            $userPasswordHasherInterface->hashPassword(  
                $user,  
                $form->get('plainPassword')->getData()  
            )  
        );  
  
        $entityManager = $this->getDoctrine()->getManager();  
        $entityManager->persist($user);  
        $entityManager->flush();  
        // do anything else you need here, like send an email  
  
        return $this->redirectToRoute('default');  
    }  
  
    return $this->render('registration/register.html.twig', [  
        'registrationForm' => $form->createView(),  
    ]);  
}
```

Ici, on retrouve un extrait du `RegisterController`, on remarque que Symfony a généré une fonction `register` avec un paramètre `UserPasswordHasherInterface`, qui permet d'enregistrer le hash du mot de passe en base de données.

`hashPassword()` hash le mot de passe en clair de l'utilisateur.

### 2.1.3 Crédation du formulaire de connexion

Pour générer le formulaire de connexion, il faut utiliser la commande `php bin/console make:auth`.

Cette commande génère un fichier `login.html.twig` situé dans un dossier `security` dans le dossier `template`, un controller est aussi créé nommé `SecurityController` et un dossier `Security` comprenant un fichier `UserAuthenticator.php` qui se chargera de traiter le formulaire soumis.

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

    // For example:
    $user = $token->getUser();
    if(in_array("ROLE_ADMIN", $user->getRoles())) {
        return new RedirectResponse($this->urlGenerator->generate('admin'));
    }
    else [
        return new RedirectResponse($this->urlGenerator->generate('default'));
    }
}
```

Dans la fonction `onAuthenticationSuccess`, il ne faut pas oublier de modifier la route de redirection, ici en fonction du rôle, la redirection est différente, l'administrateur sera redirigé vers l'administration du site tandis qu'un utilisateur simple sera redirigé vers l'accueil du site.

Ci-dessous, le formulaire de connexion :

The image shows a login form with a yellow border. At the top center, it says "CONNEXION". Below that are two input fields: one for "Email" and one for "Mot de passe", both with placeholder text. At the bottom is a large orange button labeled "Connexion". Below the button is a blue link labeled "S'inscrire".

## 3 - AFFICHAGE DES PRODUITS

### 3.1 - Menu de navigation

Pour créer le menu du site, j'ai commencé par créer une route dans le DefaultController

```
// NAVBAR
// #####
/*
 * @Route("/menu", name="category_menu", methods={"GET"})
 */
public function menuCategories(CategoryRepository $categoryRepository): Response
{
    return $this->render('parts/header.html.twig', [
        'categories' => $categoryRepository->findAll(),
    ]);
}
```

Cette fonction va récupérer toutes les catégories dans le *CategoryRepository*, elle retourne le *header.html.twig* situé dans le dossier parts et la propriété categories servira dans le twig pour récupérer les catégories.

Dans le *header.html.twig* se trouve la partie en tête de mon site, dont le menu.

```
CHAT CHIEN POISSON RONGEUR OISEAU PROMOS CONTACT
```

```
<!--NAVBAR PC-->
<nav class="navbar navPc navbar-expand-lg navbar-light bg-white dNone">
    <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
        <div class="navbar-nav text-uppercase fs-4">
            {% for category in categories %}
                {% if category.parent == false %}
                    <a class="nav-item nav-link active" href="{{ path('products_by_category', {'category': category.id}) }}> {{ category.name }}
                {% endif %}
            {% endfor %}
            <a class="nav-item nav-link colorBaseText" href="#">Promos</a>
            <a class="nav-item nav-link" href="#">Contact</a>
        </div>
    </div>
</nav>
```

Ici, un extrait du *header.html.twig*, plus précisément la balise *nav*. Afin d'afficher toutes les catégories qui n'avaient pas de catégorie parent, j'ai créé une boucle *for* en précisant avec une condition *if*, que si la catégorie ne possède pas de catégorie parente, alors je peux l'afficher sous forme de lien. Sur chaque catégorie, au click, l'utilisateur sera redirigé vers la route *products\_by\_category*, qui prend en paramètre l'identifiant de la catégorie.

### 3.2 - Affichage des produits par catégorie

Toujours dans le *DefaultController*, j'ai créé une route *products\_by\_category*.

```

// ALL PRODUCTS BY CATEGORY
// #####
/**
 * @Route("/categorie/{category}", name="products_by_category", methods={"GET"})
 */
public function findByCategory($category, CategoryRepository $categoryRepository): Response
{
    $categories = $categoryRepository->findBy(['id' => $category]);
    return $this->render('product/product_by_category.html.twig', [
        'categories' => $categories
    ]);
}

```

J'ai créé une fonction `findByCategory()` qui permet de retrouver tous les produits de la catégorie concernée. La méthode `findBy()` du `CategoryRepository` permet cela. Ensuite, est retourné le fichier `product_by_catery.html.twig` que j'ai créé dans le dossier `product` dans les templates.

```

{% extends 'base-site.html.twig' %}

{% block body %}

<div class="container mt-5 mb-5">
    {% for category in categories %}
        <div class="card m-3 row" style="width: 18rem;">
            {% for product in category.products %}
                {% for image in product.images %}
                    
                {% endfor %}
                <div class="card-body text-center">
                    <h5 class="card-title">{{ product.title }} </h5>
                    <p class="card-text">{{ product.price }} €</p>
                    <a href="{{ path('detail_product', {'product': product.id }) }}" class="btn myBtn m-3">Détail de l'article</a>
                {% endfor %}
            </div>
        {% endfor %}
    </div>
    {% endblock body %}

```

Ci-dessus le fichier `product_by_category.html.twig`, qui affiche les cartes produits de la catégorie sélectionnée. Ici, pour récupérer les produits, j'ai d'abord créé une boucle `for` pour récupérer les catégories puis à l'intérieur de celle-ci une autre boucle `for` pour afficher les produits un à un . J'ai réalisé une autre boucle `for` pour récupérer la propriété `images` dans `product`, afin d'afficher les images pour chaque produit.

J'ai créé un lien pour voir le détail d'un produit avec la route `detail_product` qui prend l'identifiant du produit en paramètre.

### 3.3 - Page détail produit

Comme précédemment, j'ai créé une route `detail_product` qui affiche le détail d'un produit. Elle retourne le fichier `detail_product.html.twig`.



[Retour à la liste des produits](#)



## Sac à foin Bambi

23.90 €

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

[Ajouter au panier](#)

- ✓ Chez vous en 3 à 5 jours ouvrés
- ✓ Livraison offerte dès 49 €

## 4 - VEILLE EN ANGLAIS

Après avoir créé la page qui affiche tous les produits par catégorie, il était impossible d'accéder au formulaire de connexion.

Manquant d'expérience, je n'ai pas réussi à trouver par moi même pourquoi cela ne fonctionnait pas.

J'ai donc cherché via le moteur de recherche de Google "symfony displays the template of the wrong route", et j'ai obtenu les résultats suivants :

symfony displays the template of the wrong route

🔍 Tous 🖼 Images 🎥 Vidéos 📰 Actualité 🗺 Carte 💳 Shopping Préférences ▾

🇫🇷 France ▾ Filtre parental : modéré ▾ À tout moment ▾

🔗 <https://symfony.com/doc/current/routing.html>

## Routing (Symfony Docs)

The value of the condition option is any valid ExpressionLanguage expression and can use any of these variables created by Symfony: context An instance of RequestContext, which holds the most fundamental information about the route being matched. request The Symfony Request object that represents the current request. Behind the scenes, expressions are compiled down to raw PHP.

🔗 <https://symfony.com/doc/current/templates.html>

## Creating and Using Templates (Symfony Docs)

Creating and Using Templates. A template is the best way to organize and render HTML from inside your application, whether you need to render HTML from a controller or generate the contents of an email. Templates in Symfony are created with Twig: a flexible, fast, and secure template engine.

📌 <https://stackoverflow.com/questions/50747664/symfony4-annotation-routing-does-not-work>

### symfony - Symfony4 Annotation routing does not work ...

I found out my mistake. I used the **wrong** namespace for routing. use Symfony\Component\Annotation\Route; It should have been: use Symfony\Component\Routing\Annotation\Route; EDIT: I wanted to delete this question but the system wouldn't let me.

Les premiers résultats renvoyaient la documentation de Symfony, j'ai décidé d'aller voir la documentation sur les routes, car il s'agissait à coup sûr d'un problème dans un des controllers.

Une fois sur le site de Symfony, je me suis dirigée directement sur la partie concernant le debug de route.

## Debugging Routes

As your application grows, you'll eventually have a *lot* of routes. Symfony includes some commands to help you debug routing issues. First, the `debug:router` command lists all your application routes in the same order in which Symfony evaluates them:

```
$ php bin/console debug:router
```

On nous explique ici que Symfony inclut des commandes afin de nous aider à déboguer les problèmes de routes. Ici, la commande `debug:router`, qui renvoie la liste des routes de l'application.

Pass the name (or part of the name) of some route to this argument to print the route details:

```
$ php bin/console debug:router app_lucky_number
```

Ci-dessus, on nous explique comment afficher le détail d'une route.

The other command is called `router:match` and it shows which route will match the given URL. It's useful to find out why some URL is not executing the controller action that you expect:

```
$ php bin/console router:match /lucky/number/8
```

Cette dernière commande affiche la route qui correspond à l'url. Il est précisé que cette commande est très utile pour trouver pourquoi l'url n'exécute pas l'action du controller désiré. Exactement mon souci.

```
+-----+
PS C:\wamp64\www\lanimesalerie> php bin/console router:match /login

[OK] Route "products_by_category" matches

+-----+
| Property      | value
+-----+
| Route Name    | products_by_category
| Path           |/{category}
| Path Regex    |^/(?P<category>[^/]+)$}sDu
| Host          | ANY
| Host Regex   |
| Scheme        | ANY
| Method        | GET
| Requirements  | NO CUSTOM
| Class          | Symfony\Component\Routing\Route
| Defaults       | _controller: App\Controller\DefaultController::findByCategory()
| Options        | compiler_class: Symfony\Component\Routing\RouteCompiler
|               | utf8: true
+-----+
PS C:\wamp64\www\lanimesalerie>
```

J'ai donc exécuté la commande, et on voit bien que l'url /login correspond à la route *products\_by\_category*. Mais d'où venait ce problème ?

Je me suis donc tournée vers le controller qui renvoie la route *products\_by\_category*, c'est une route avec paramètres, et justement, dans la documentation de Symfony, j'ai trouvé ceci :

## Route Parameters

The previous examples defined routes where the URL never changes (e.g. `/blog`). However, it's common to define routes where some parts are variable. For example, the URL to display some blog post will probably include the title or slug (e.g. `/blog/my-first-post` or `/blog/all-about-symfony`).

In Symfony routes, variable parts are wrapped in `{ ... }` and they must have a unique name. For example, the route to display the blog post contents is defined as `/blog/{slug}`:

Annotations	Attributes	YAML	XML	PHP
-------------	------------	------	-----	-----

```
// src/Controller/BlogController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class BlogController extends AbstractController
{
    // ...

    /**
     * @Route("/blog/{slug}", name="blog_show")
     */
}
```

Ici, on nous dit que ce que nous avons vu avant concernait les routes dont l'url ne changeait jamais. Il y a parfois des routes avec des parties variables, par exemple l'url qui affiche des posts de blog change en fonction du nom du post. Dans Symfony, la partie variable est entourée de {...} et le nom de celle-ci doit-être unique. Par exemple, la route qui affiche un post de blog est : `/blog/{slug}`.

The name of the variable part (`{slug}` in this example) is used to create a PHP variable where that route content is stored and passed to the controller. If a user visits the `/blog/my-first-post` URL, Symfony executes the `show()` method in the `BlogController` class and passes a `$slug = 'my-first-post'` argument to the `show()` method.

Routes can define any number of parameters, but each of them can only be used once on each route (e.g. `/blog/posts-about-{category}/page/{pageNumber}`).

## Parameters Validation

Imagine that your application has a `blog_show` route (URL: `/blog/{slug}`) and a `blog_list` route (URL: `/blog/{page}`). Given that route parameters accept any value, there's no way to differentiate both routes.

If the user requests `/blog/my-first-post`, both routes will match and Symfony will use the route which was defined first. To fix this, add some validation to the `{page}` parameter using the `requirements` option:

Annotations   Attributes   YAML   XML   PHP

Le nom de la partie variable est utilisé pour créer une variable PHP dont le contenu est stocké et transmis au controller. Si un visiteur visite l'url `/blog/my-first-post`, Symfony exécute la méthode `show()` dans le `BlogController` et passe en argument 'my-first-post' dans cette méthode.

Les routes peuvent définir plusieurs paramètres, mais chacun d'eux peut-être utilisé une fois par route.

Bon, jusque-là, je n'apprenais rien de nouveau, mais la suite de la documentation concernait la validation des paramètres, et cette partie m'a aiguillée.

En effet, il est dit que si dans mon application, plusieurs routes ne peuvent être différencierées, Symfony utilisera la route définie en premier. Pour résoudre ce problème, il suffit d'ajouter l'option requirements :

```
/**  
 * @Route("/blog/{page}", name="blog_list", requirements={"page"="\d+"})  
 */
```

The `requirements` option defines the PHP regular expressions ↗ that route parameters must match for the entire route to match. In this example, `\d+` is a regular expression that matches a *digit* of any length. Now:

L'option *requirements* définit les expressions régulières PHP auxquelles les paramètres de l'itinéraire doivent correspondre pour que la route entière corresponde. Dans cet exemple, `\d+` correspond à un chiffre de n'importe quelle longueur.

Bien que cela ne répondait pas totalement à ma question, je savais que la route `products_by_category` prenait la place de la route `app_login`.

```
/**  
 * @Route("/{category}", name="products_by_category", methods={"GET"})  
 */
```

Ici, la route `products_by_category`.

```
/**  
 * @Route("/login", name="app_login")  
 */
```

Et la route qui ne s'affiche pas.

J'ai testé et modifié l'url comme ceci :

```
/**  
 * @Route("/categorie/{category}", name="products_by_category", methods={"GET"})  
 */
```

Et tout fonctionnait à nouveau.

