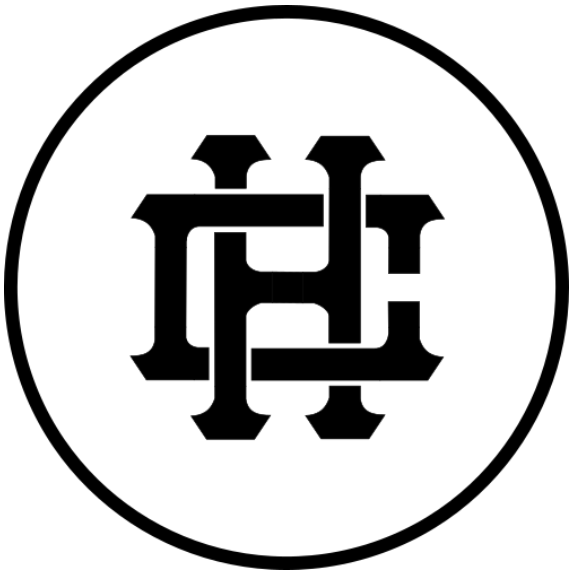
Documentation Technique

Projet Symfony HoodzClothes



Réalisé par : Vadier Mathis

Présentation et Objectifs

Nous sommes le groupe 2 composée de **Pichard Simon** et **Vadier Mathis**. Nous devions réaliser un site de e-commerce.

Afin de réaliser notre mission, nous avions un début de code source GitHub ainsi qu'une série de 19 vidéos pour nous expliquer le fonctionnement de Symfony 6.

Nous avions donc comme tâches :

1 - La gestion des utilisateurs : Cette tâche doit permettre l'inscription, connexion à un compte vérifié

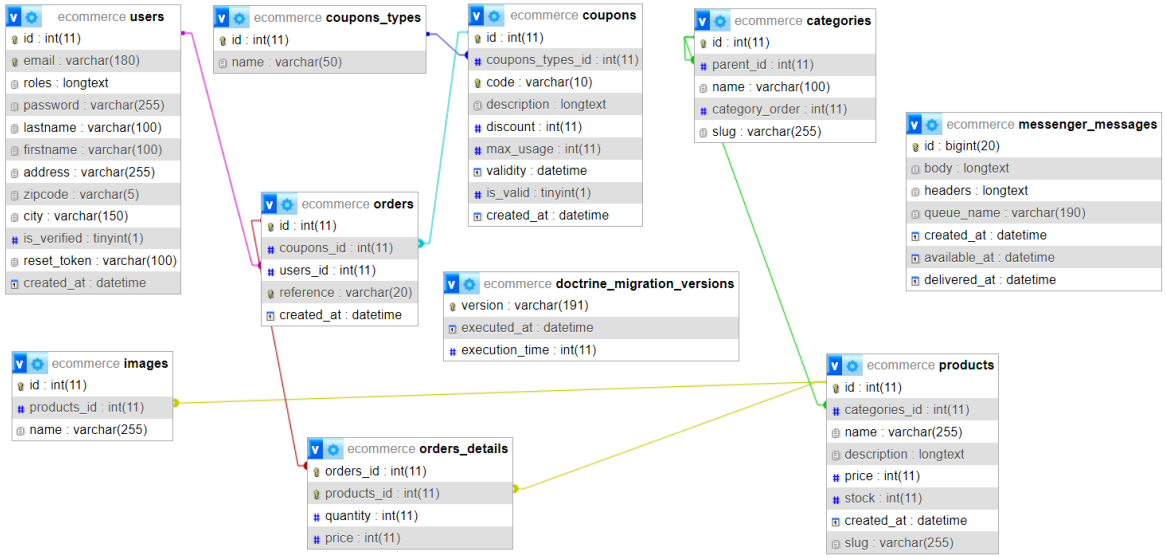
2 - La gestion des produits et des catégories : Cette tâche doit permettre l'affichage, création, suppression et modification de produits selon les catégories.

3 - La gestion du panier : Cette tâche doit permettre à l'utilisateur de gérer leur panier de produits

Environnement de travail

Avant de découvrir notre travail, laissez nous vous présenter notre environnement. Nous avons travaillé à l'aide d'une plateforme de développement Web **XAMPP** et d'une base de données **MySQL** que nous avons dû créer pour le projet.

Voici le schéma relationnel :



Par la suite, pour réaliser notre site de e-commerce nous avons utilisé l'IDE **PHP Storm** avec lequel nous avons utilisés les langages de programmation : **HTML**, **CSS** et **PHP** avec le Framework Symfony 6 tout en respectant une architecture **MVC** (Modèle / Vue / Contrôle).

Connexion à la BDD

Pour réaliser la connexion à la base de données nous avons dû modifier le fichier **.env** pour y ajouter une variable DATABASE_URL avec une valeur pour MySQL.

```
###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/co
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
#
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
DATABASE_URL="mysql://root:123@127.0.0.1:3306/ecommerce?serverVersion=8"
#DATABASE_URL="postgresql://symfony:ChangeMe@127.0.0.1:5432/app?serverVersion=13&charset=utf8"
###< doctrine/doctrine-bundle ###
```

Il faut ensuite taper la commande qui permet de migrer la base de données

```
symfony console make:migration
```

Détails de mon travail

- Affichage des produits
- Réalisation pied de page
- Gestion de l'inscription et du compte

Affichage des produits

Affichage de la page d'accueil

Pour réaliser l'affichage des produits sur la page d'accueil, j'ai utilisé dans le MainController une fonction **index(ManagerRegistery $doctrine)** qui permet de récupérer les catégories de produits ainsi que tous les produits, à partir de la base de données et d'envoyer ces données à la vue Twig.

```php
#[Route('/', name: 'main')]
public function index(ManagerRegistry $doctrine): Response
{
    $categorie = $doctrine->getRepository( persistentObject: Categories::class)->findBy([], ['categoryOrder' => 'asc']);
    $products = $doctrine->getRepository( persistentObject: Products::class)->findAll();

    return $this->render( view: 'main/index.html.twig', [
        'produits' => $products,
        'categories'=> $categorie
    ]);
}
```

Voici la vue index.html.twig qui permet d'afficher tous les produits.

```twig
{% extends 'base.html.twig' %}
{% block body %}
    <main class="container">
        <section class="row">
            <div class="col-12">
                <div class="presentation">
                <h1 class="slogan">VENEZ GLOW UP AVEC HOODZCLOTHES</h1>
                <h2 class="description">Booste ton style avec notre selection de vêtements et accessoires</h2>
                <a class="ex" href="#prod">Explorer</a></div>
            </div>
            <p id="prod" class="prod">Tous les produits</p>
            {% for unProduit in produits %}
                <div class="col-3 mb-3">
                    <article class="card h-100">
                        <img src="{{ asset('assets/uploads/products/mini/300x300-' ~ unProduit.images[0].name ) }}"
                             class="card-img-top" alt="{{ unProduit.name }}">
                        <div class="card-body">
                            <h2 class="card-title">{{ unProduit.name }}</h2>
                            <p class="card-text">{{ unProduit.price / 100 }} €</p>
                            <a href="{{ path('products_details', {"slug": unProduit.slug }) }}"
                               class="btn btn-primary">Voir les détails</a>
                        </div>
                    </article>
                </div>
            {% endfor %}
        </section>
    </main>
{% endblock %}
```

Affichage de la page pour chaque catégorie

Pour réaliser l'affichage des produits dans chaque catégorie, j'ai utilisé dans le CategoriesController la fonction **list(Categories $category . . . $request)** qui permet de récupérer le numéro de page à partir de la requête HTTP, puis utilise un référentiel pour récupérer une liste paginée de produits pour une catégorie spécifique et envoie ces données à la vue Twig.

```php
#[Route('/categories', name: 'categories_')]
class CategoriesController extends AbstractController
{
    Benoit Gambier *
    #[Route('/{slug}', name: 'list')]
    public function list(Categories $category, ProductsRepository $productsRepository, Request $request): Response
    {
        //On va chercher le numéro de page dans l'url
        $page = $request->query->getInt( key: 'page',  default: 1);

        //On va chercher la liste des produits de la catégorie
        $products = $productsRepository->findProductsPaginated($page, $category->getSlug(),  limit: 4);

        return $this->render( view: 'categories/list.html.twig', compact( var_name: 'category',  ...var_names: 'products'));

    }
}
```

Voici la vue list.html.twig qui permet d'afficher la liste des produits d'une catégorie donnée.

```twig
{% extends 'base.html.twig' %}
{% block title %}Liste des produits de la catégorie {{ category.name }}{% endblock %}
{% block body %}
    <main class="container">
        <section class="row">
            <div class="col-12">
                <h1>Liste des produits de la catégorie {{ category.name }}</h1>
            </div>
            {% for product in products.data %}
                <div class="col-3 mb-3">
                    <article class="card h-100">
                        <img src="{{ asset('assets/uploads/products/mini/300x300-' ~ product.images[0].name ) }}"
                             class="card-img-top" alt="{{ product.name }}">
                        <div class="card-body">
                            <h2 class="card-title">{{ product.name }}</h2>
                            <p class="card-text">{{ product.price / 100 }} €</p>
                            <a href="{{ path('products_details', {"slug": product.slug }) }}"
                               class="btn btn-primary">Voir les détails</a>
                        </div>
                    </article>
                </div>
            {% endfor %}
            {% set path = 'categories_list' %}
            {% set slug = category.slug %}
            {% set pages = products.pages %}
            {% set currentPage = products.page %}
            {% include "_partials/_pagination.html.twig" %}
        </section>
    </main>
{% endblock %}
```

Réalisation pied de page

J'ai utilisé dans l'InfoController, une fonction pour chaque page que comporte le pied de page qui par le biais renvoie à une page spécifique.

```php
#[Route('/faq', name: 'faq')]
public function faq(ManagerRegistry $doctrine): Response
{
    return $this->render( view: 'info/faq.html.twig', []);
}
```

```php
#[Route('/contact', name: 'contact')]
public function contact(ManagerRegistry $doctrine): Response
{
    return $this->render( view: 'info/contact.html.twig', []);
}
```

```php
#[Route('/cgu', name: 'cgu')]
public function cgu(ManagerRegistry $doctrine): Response
{
    return $this->render( view: 'info/cgu.html.twig', []);
}
```

```php
#[Route('/cgv', name: 'cgv')]
public function cgv(ManagerRegistry $doctrine): Response
{
    return $this->render( view: 'info/cgv.html.twig', []);
}
```

```php
#[Route('/chartecookies', name: 'chartecookies')]
public function chartecookies(ManagerRegistry $doctrine): Response
{
    return $this->render( view: 'info/chartecookies.html.twig', []);
}
```

```php
#[Route('/mentionslegales', name: 'mentionslegales')]
public function mentionslegales(ManagerRegistry $doctrine): Response
{
    return $this->render( view: 'info/mentionslegales.html.twig', []);
}
```

Voici la vue _footer.html.twig qui permet d'afficher le pied de page et permet de faire le lien avec les différentes routes du contrôleur.

```
<footer class="piedpage">
    <div class="row">
        <section class="col centre">
            <h1 class="titre-footer">A propos de nous</h1>
            <br>
            <a class="lien-footer" href="{{ path('faq') }}">FAQ</a><br>
            <a class="lien-footer" href="{{ path('contact') }}">Contact</a><br>
        </section>
        <section class="col">
            <h1 class="titre-footer">Informations légales</h1>
            <br>
            <a class="lien-footer" href="{{ path('cgu') }}">CGU</a><br>
            <a class="lien-footer" href="{{ path('cgv') }}">CGV</a><br>
            <a class="lien-footer" href="{{ path('mentionslegales') }}">Mentions Légales</a><br>
            <a class="lien-footer" href="{{ path('chartecookies') }}">Charte Cookies</a><br>
        </section>
        <section class="col">
            <h1 class="titre-footer">Paiements acceptés</h1>
            <br>
            <a class="logo-footer"><i class='fa fa-cc-mastercard'></i></a>
            <a class="logo2-footer"><i class='fa fa-cc-paypal'></i></a>
            <a class="logo2-footer"><i class='fa fa-cc-visa'></i></a>
            <a class="logo2-footer"><i class='fab fa-cc-apple-pay'></i></a>
            <br>
        </section>
        <section class="col">
            <h1 class="titre-footer">Suivez-nous</h1>
            <br>
            <a class="logo-footer"><i class='bx bxl-twitter'></i></a>
            <a class="logo2-footer"><i class='bx bxl-instagram' ></i></a>
            <a class="logo2-footer"><i class='bx bxl-youtube'></i></a>
            <a class="logo2-footer"><i class='bx bxl-tiktok' ></i></a>
        </section>
    </div>
</footer>
```

Gestion de l'inscription et du compte

Inscription

Afin de gérer l'inscription d'un utilisateur au site, j'ai utilisé dans le RegistrationController une fonction **register(Resquest $request . . . $jwt)** qui gère le processus d'inscription des utilisateurs sur le site. Elle crée un formulaire d'inscription, valide les données soumises, hache le mot de passe de l'utilisateur, enregistre l'utilisateur dans la base de données, génère un token JWT, envoie un mail d'activation du compte et enfin authentifie automatiquement l'utilisateur nouvellement inscrit.

```php
#[Route('/inscription', name: 'app_register')]
public function register(Request $request, UserPasswordHasherInterface $userPasswordHasher,
                         UserAuthenticatorInterface $userAuthenticator, UsersAuthenticator $authenticator,
                         EntityManagerInterface $entityManager, SendMailService $mail, JWTService $jwt): Response
{
    $user = new Users();
    $form = $this->createForm( type: RegistrationFormType::class, $user);
    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        // encode the plain password
        $user->setPassword(
        $userPasswordHasher->hashPassword(
                $user,
                $form->get('plainPassword')->getData()
            )
        );
        $entityManager->persist($user);
        $entityManager->flush();
```

```php
        // On génère le JWT de l'utilisateur
        // On crée le Header
        $header = [
            'typ' => 'JWT',
            'alg' => 'HS256'];
        // On crée le Payload
        $payload = [
            'user_id' => $user->getId()];
        // On génère le token
        $token = $jwt->generate($header, $payload, $this->getParameter( name: 'app.jwtsecret'));
        // On envoie un mail
        $mail->send(
             from: 'mathis.ecom@gmail.com',
            $user->getEmail(),
             subject: 'Activation de votre compte sur le site e-commerce',
             template: 'register',
            compact( var_name: 'user',  ...var_names: 'token')
        );
        return $userAuthenticator->authenticateUser(
            $user, $authenticator, $request);
    }
    return $this->render( view: 'registration/register.html.twig', [
        'registrationForm' => $form->createView(),]);
}
```

Ensuite, j'utilise la fonction **verifyUser(JWTService $jwt . . . $em)** qui permet de vérifier la validité et l'expiration d'un token JWT, ainsi que son intégrité. Si le token est valide, elle vérifie si l'utilisateur correspond.

```php
#[Route('/verif/{token}', name: 'verify_user')]
public function verifyUser($token, JWTService $jwt, UsersRepository $usersRepository,
                           EntityManagerInterface $em): Response
{
    //On vérifie si le token est valide, n'a pas expiré et n'a pas été modifié
    if($jwt->isValid($token) && !$jwt->isExpired($token) && $jwt->check($token,
            $this->getParameter( name: 'app.jwtsecret'))){
        // On récupère le payload
        $payload = $jwt->getPayload($token);

        // On récupère le user du token
        $user = $usersRepository->find($payload['user_id']);

        //On vérifie que l'utilisateur existe et n'a pas encore activé son compte
        if($user && !$user->getIsVerified()){
            $user->setIsVerified( is_verified: true);
            $em->flush($user);
            $this->addFlash( type: 'success', message: 'Utilisateur activé');
            return $this->redirectToRoute( route: 'profile_index');
        }
    }
    // Ici un problème se pose dans le token
    $this->addFlash( type: 'danger', message: 'Le token est invalide ou a expiré');
    return $this->redirectToRoute( route: 'app_login');
}
```

Après que l'utilisateur se soit inscrit, il doit vérifier son mail, j'ai donc utilisé la fonction **resendVerif(JWTService $jwt . . . $usersRepository)** qui permet à un utilisateur connecté, mais non vérifié de renvoyer le mail de vérification. Elle génère un nouveau token JWT, envoie le mail de vérification et informe l'utilisateur avec un message flash.

```php
#[Route('/renvoiverif', name: 'resend_verif')]
public function resendVerif(JWTService $jwt, SendMailService $mail, UsersRepository $usersRepository): Response
{
    $user = $this->getUser();
    if(!$user){
        $this->addFlash( type: 'danger', message: 'Vous devez être connecté pour accéder à cette page');
        return $this->redirectToRoute( route: 'app_login');
    }
    if($user->getIsVerified()){
        $this->addFlash( type: 'warning', message: 'Cet utilisateur est déjà activé');
        return $this->redirectToRoute( route: 'profile_index');
    }
    // On génère le JWT de l'utilisateur et On crée le Header
    $header = [
        'typ' => 'JWT',
        'alg' => 'HS256'];
    // On crée le Payload
    $payload = [
        'user_id' => $user->getId()];
    // On génère le token
    $token = $jwt->generate($header, $payload, $this->getParameter( name: 'app.jwtsecret'));
    // On envoie un mail
    $mail->send(
         from: 'no-reply@monsite.net',
        $user->getEmail(),
         subject: 'Activation de votre compte sur le site e-commerce',
         template: 'register', compact( var_name: 'user', ...var_names: 'token'));
    $this->addFlash( type: 'success', message: 'Email de vérification envoyé');
    return $this->redirectToRoute( route: 'profile_index');
}
```

Voici la vue register.html.twig qui permet d'afficher le formulaire d'inscription.

```twig
{% extends 'base.html.twig' %}
{% block title %}Inscription{% endblock %}
{% block body %}
    <section class="container">
        <div class="row">
            <div class="col">
                <h1>Inscription</h1>
                {{ form_start(registrationForm) }}
                    <fieldset class="mb-3">
                        <legend>Mon identité</legend>
                        {{ form_row(registrationForm.lastname) }}
                        {{ form_row(registrationForm.firstname) }}
                        {{ form_row(registrationForm.email) }}
                    </fieldset>
                    <fieldset class="mb-3">
                        <legend>Mes coordonnées</legend>
                        {{ form_row(registrationForm.address) }}
                        {{ form_row(registrationForm.zipcode) }}
                        {{ form_row(registrationForm.city) }}
                    </fieldset>
                    {{ form_row(registrationForm.plainPassword, {label: 'Password'}) }}
                    {{ form_row(registrationForm.RGPDConsent) }}
                    <button type="submit" class="btn btn-primary btn-lg my-3">M'inscrire</button>
                    <a href="{{ path('app_login') }}" class="btn btn-secondary">Me connecter</a>
                {{ form_end(registrationForm) }}
            </div>
        </div>
    </section>
{% endblock %}
```

Connexion

Afin de gérer la connexion d'un utilisateur au site, j'ai utilisé dans le SecurityController une fonction **login(AuthenticationUtils $authenticationUtils)** qui permet de gérer l'affichage du formulaire de connexion. Elle récupère l'éventuelle erreur d'authentification et le dernier nom d'utilisateur saisi, puis rend la vue Twig en passant ces informations en tant que variables. La vue peut ensuite utiliser ces variables pour afficher des messages d'erreur ou pré remplir le champ du nom d'utilisateur.

```php
#[Route('/connexion', name:'app_login')]
public function login(AuthenticationUtils $authenticationUtils): Response
{
    // if ($this->getUser()) {
    //     return $this->redirectToRoute('target_path');
    // }

    // get the login error if there is one
    $error = $authenticationUtils->getLastAuthenticationError();
    // last username entered by the user
    $lastUsername = $authenticationUtils->getLastUsername();

    return $this->render( view: 'security/login.html.twig', [
        'last_username' => $lastUsername,
        'error' => $error
    ]);
}
```

Voici la vue .html.twig qui permet d'afficher le formulaire de connexion.

```twig
{% extends 'base.html.twig' %}
{% block title %}Connexion{% endblock %}
{% block body %}
<section class="container my-3">
    <div class="row">
        <div class="col">
            <form method="post">
                {% if error %}
                    <div class="alert alert-danger">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
                {% endif %}
                {% if app.user %}
                    <div class="mb-3">
                        Vous êtes connecté(e) comme {{ app.user.userIdentifier }},
                        <a href="{{ path('app_logout') }}">Me déconnecter</a></div>
                {% endif %}
                <h1 class="h3 mb-3 font-weight-normal">Me connecter</h1>
                <label for="inputEmail">Email</label>
                <input type="email" value="{{ last_username }}" name="email" id="inputEmail" class="form-control"
                       autocomplete="email" required autofocus>
                <label for="inputPassword">Mot de passe</label>
                <input type="password" name="password" id="inputPassword" class="form-control"
                       autocomplete="current-password" required>
                <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}">
                <button class="btn btn-lg btn-primary mt-3" type="submit">Me connecter</button>
                <a href="{{ path('app_register') }}" class="btn btn-secondary mt-3">M'inscrire</a>
                <a href="{{ path('forgotten_password') }}" class="btn btn-secondary mt-3">Mot de passe oublié</a>
            </form>
        </div></div>
</section>
{% endblock %}
```

Compte

Lorsqu'un utilisateur est connecté il a alors accès à l'onglet compte dans la barre de navigation. J'ai utilisé la fonction **index()** qui permet d'accès à la vue Twig du compte et transmettre la variable controller_name avec la valeur profile de l'utilisateur.

```php
#[Route('/', name: 'index')]
public function index(): Response
{
    return $this->render( view: 'profile/index.html.twig', [
        'controller_name' => 'Profil de l\'utilisateur',
    ]);
}
```

Voici la vue profile/index.html.twig qui permet d'afficher la page de compte de l'utilisateur.

```twig
{% extends 'base.html.twig' %}

{% block title %}Hello ProfileController!{% endblock %}

{% block body %}
<style>
    .example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }
    .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
</style>

<div class="example-wrapper">
    <h1>Bienvenue vous êtes connecté ✅</h1>

    Vous pouvez accéder a votre panier <a href="{{ path('profile_indexpanier') }}">Ici</a>
</div>
{% endblock %}
```