

Atelier : Création d'un site e-commerce avec Symfony 6

Prérequis :

- PHP 8 installé
- Composer installé
- Symfony CLI installé
- maildev

Si le projet est existant :

- `composer update`
- `.env.local`
- `Php bin/console doctrine:database:create`
- `Php bin/console doctrine:migrations:migrate`
- `npm i`
- `npm run watch`
- `Symfony serve`
- `Symfony server:ca:install`

Objectifs de l'atelier :

- Comprendre l'architecture MVC de Symfony
- Configurer un projet Symfony
- Créer des entités et interagir avec une base de données
- Développer des contrôleurs et gérer le routage
- Utiliser Twig pour le rendu des vues
- Gérer les assets avec Webpack Encore
- Mettre en place un système d'authentification
- Mettre en place une navigation dynamique

Plan de l'atelier :

1. Présentation de l'architecture MVC

- Explication du modèle MVC (Modèle-Vue-Contrôleur) et son application dans Symfony.

2. Initialisation du projet Symfony

- Créer un nouveau projet Symfony :
`symfony new my_project_directory --version="6.2.*" --webapp`
- Naviguer dans le répertoire du projet :
`cd my_ecommerce`

3. Configuration du projet

- Dupliquer le fichier `.env` en `.env.local`.
- Modifier le fichier `.env.local` pour définir la variable `DATABASE_URL`.
- Ajouter les fichiers sensibles au fichier `.gitignore`.

4. Installation des dépendances backend

- Installer les bundles nécessaires :
composer require symfony/maker-bundle
composer require doctrine

5. Création des entités et de la base de données

- Créer une entité Product :
php bin/console make:entity Product
- Ajouter les propriétés suivantes : name, description, price, image.
- Créer la base de données :
php bin/console doctrine:database:create
- Générer et exécuter les migrations :
php bin/console make:migration
php bin/console doctrine:migrations:migrate

6. Création des contrôleurs et des routes

- Créer un contrôleur pour les produits :
- Définir les routes pour les actions CRUD
php bin/console make:crud
- Utiliser l'injection de dépendances pour accéder à EntityManagerInterface.

7. Utilisation de Twig pour les vues

- Utiliser la fonction `render ()` dans le contrôleur pour retourner les vues Twig.
- Utiliser les fonctions `path ()` les liens et `asset ()` pour les images dans les templates Twig.
- Implémenter des conditions et des boucles pour afficher dynamiquement les données.

8. Gestion des assets avec Webpack Encore

- Installer Webpack Encore :
composer require symfony/webpack-encore-bundle
- Désinstaller AssetMapper
composer remove symfony/asset-mapper
- Installer Bootstrap
npm install bootstrap --save-dev
- Installer les dépendances front-end :
npm install bootstrap sass-loader@^16.0.1 sass --save-dev
- Configurer Webpack Encore dans le fichier `webpack.config.js` et décommenter la partie `saas`
- Configurer le framework JS stimulus avec WebPackEncore
npm install @symfony/webpack-encore @hotwired/stimulus --save-dev
- Configurer Webpack Encore dans le fichier `webpack.config.js`.
- Générer les assets :
npm run watch

9. Générer des formulaire

- Créer une entité contact pour créer un formulaire de contact :
php bin/console make:entity
- Créer le formulaire (ContactType)
php bin/console make:form
- Configurer les champs de formulaire dans ContactType
- Créer la vue qui affiche le formulaire :
php bin/console make:controller ContactController
- Afficher le formulaire dans la vue générée
- Appliquer le thème Bootstrap à votre formulaire
{% form_theme contactForm 'bootstrap_5_layout.html.twig' %}
- Installer Symfony Validator pour la validation des champs au niveau des classes ou de ContactType :
composer require symfony/validator
- Afficher un message flash si la soumission du champ est valide sinon afficher les erreurs
- Envoyer un email si aucune erreur n'est présente sur le formulaire de contact
composer require symfony/mailer
composer require symfony/twig-bundle

10. Mise en place de la sécurité et de l'authentification

- Installer le bundle de sécurité :
composer require symfony/security-bundle
- Créer une entité User :
php bin/console make:user
- Configurer le système de sécurité dans le fichier `config/packages/security.yaml`.
- Générer les formulaires de connexion et d'inscription :
composer require symfonycasts/verify-email-bundle
php bin/console make:registration-form
php bin/console make:security:form-login
 - Assurer les redirection après connexion et déconnexion :
form_login:
default_target_path: app_profile
logout:
target: app_home
- Vérifier l'utilisateur connecté avec `app.user` et `is_granted()`.
- Installer maildev en local pour recevoir des emails
npm i -g maildev
.env.local : MAILER_DSN=smtp://localhost:1025?verify_peer=0
maildev -v --ip 127.0.0.1
- Sécuriser la route `/profile` et ne la rendre accessible qu'aux personnes connectées :
php bin/console make:user (config/packages/security.yaml)
- { path: ^/profile, roles: ROLE_USER }

11. Créer une navigation dynamique

- Afficher inscription/connexion si pas connecté ou profile si connecté dans votre nav
`{% if app.user %}`

12. Créer un backOffice avec EasyAdmin

- Afficher inscription/connexion si pas connecté ou profile si connecté dans votre nav
composer require easycorp/easyadmin-bundle
php bin/console make:admin:dashboard
php bin/console make:admin:crud

13. Sécurisez vos routes

- Sécurisez la route qui mène à votre backoffice pour n'accorder que l'accès à un administrateur (config/packages/security.yaml)
access_control:
- { path: ^/admin, roles: [ROLE_ADMIN] }
- Pensez à modifier le rôle d'un de vos user en lui donnant le ROLE_ADMIN

14. Ajouter des produits au panier

- Créer CartController
- Créer une route pour afficher le panier
\$cart = \$session->get('cart', []); return \$this->render('cart/show.html.twig', ['cart' => \$cart]);
- Créer la méthode pour ajouter un produit à la session panier
\$product = \$productRepo->find(\$id);
if (\$product) {
\$cart[\$id] = (\$cart[\$id] ?? 0) + 1;
\$session->set('cart', \$cart);
}
- Créer une route pour vider le panier
\$session->remove('cart');
- Créer le fichier Twig cart/show.html.twig

15. Générer les commandes et détails de commande si le panier est valider

- Créer une méthode pour valider le panier dans le contrôleur
- Créer une route pour afficher la commande
- Créer la vue twig order/show.html.twig
- Alimenter votre backOffice pour administrer les commandes et leur status

16. Mettre en place des données mock avec DataFaker (Fixtures)

- Créer une méthode pour valider le panier dans le contrôleur
composer require --dev orm-fixtures
- Créer une route pour afficher la commande
composer require fakerphp/faker
- Créer une route pour afficher la commande
php bin/console doctrine:fixtures:load --append

17. Générer des PDF et les envoyer par email avec DomPDF

- Installer la librairie DomPDF
composer require dompdf/dompdf
- Créer le pdf avec les options
\$dompdf = new Dompdf(\$pdfOptions);
- Préparer le twig qui sera envoyé en pdf
\$html = \$this->renderView();
- Transformer le twig en pdf avec les options de format
\$dompdf->loadHtml(\$html);
- Enregistrer le pdf dans une variable
\$dompdf->render();
\$finalInvoice = \$dompdf->output();
- Attacher le pdf à l'envoi d'email
->attach(\$finalInvoice)

18. Créer un service d'envoi de mail et de génération de pdf

- Installer la librairie DomPDF
composer require dompdf/dompdf
- Créer le pdf avec les options
\$dompdf = new Dompdf(\$pdfOptions);

19. Traduire site en multilingues

- Installer la librairie DomPDF
composer require dompdf/dompdf
- Créer le pdf avec les options
\$dompdf = new Dompdf(\$pdfOptions);

19. Personnaliser les messages d'erreur (404, 500...)

- Installer la librairie DomPDF
composer require dompdf/dompdf
- Créer le pdf avec les options
\$dompdf = new Dompdf(\$pdfOptions);

20. Importer un template html dans une page (message flash)

- Installer la librairie DomPDF
composer require dompdf/dompdf
- Créer le pdf avec les options
\$dompdf = new Dompdf(\$pdfOptions);

21. Générer un paiement via une API externe bancaire (Stripe)

- Créer une méthode pour valider le panier dans le contrôleur
composer require --dev orm-fixtures
- Créer une route pour afficher la commande
composer require fakerphp/faker
- Créer une route pour afficher la commande
php bin/console doctrine:fixtures:load --append

22. Appeler un controller Symfony en asynchrone avec Javascript

- Créer une méthode pour valider le panier dans le contrôleur
composer require --dev orm-fixtures
- Créer une route pour afficher la commande
composer require fakerphp/faker
- Créer une route pour afficher la commande
php bin/console doctrine:fixtures:load --append

23. Utiliser le framework Stimulus JS dans Symfony

- Créer une méthode pour valider le panier dans le contrôleur
composer require --dev orm-fixtures
- Créer une route pour afficher la commande
composer require fakerphp/faker
- Créer une route pour afficher la commande
php bin/console doctrine:fixtures:load --append

Une interface pour la gestion des commandes
easyadmin => gerer status des commandes
envoyé un email quand le status change

Tirer par ordre les produits
Faire une recherche

Explique les sujets de fonds
Request
EntityManager
SecurityBundle
ORM Doctrine
KernelInterface

Ressources supplémentaires :

- Documentation officielle de Symfony : <https://symfony.com/doc/current/index.html>