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 dé-commenter 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 = $dompdf->output();
- Attacher le pdf à l'envoie d'email
  ->attach($finalInvoice)

18. Créer un service d'envoie 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 multilangues

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

20. 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);

21. 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 odre les produits
Faire une recherche

NPM : agent dans node qui permet d'installer les dépendances côté front

Composer : agent dans php qui permet d'installer les dépendances côté back-end

WebPack : compilateur qui permet de gérer les ressources d'un projet (css, js, images…)

Doctrine : ORM (Object-Relational Mapping) qui est une interface qui permet d'interagir avec la

BBD

MakerBundle : (php bin/console make:) Agent (dépendance composer) qui permet de générer controller, vue, crud etc…

Twig : moteur de template html qui permet d'avoir de la logique dans du HTML

SecurityBundle : dépendance qui nous a permis de créer un système d'authentification/inscription avec session et mot de passe hashé (B-CRYPT)

EntityManager : outil principale de Doctrine pour interragir avec la BDD

- **Créer** des enregistrements (`persist()` + `flush()`)
- **Lire** des données (`find()`, `findBy()`, `getRepository()`)
- **Mettre à jour** des entités (`persist()` + `flush()`)
- **Supprimer** des enregistrements (`remove()` + `flush()`)

KernelInterface: Interface de Symfony qui représente **le cœur de l'application Symfony**

Request : est une classe qui permet de récupérer et manipuler les données d'une requête HTTP (get, post, put…, session, cookies…)

REST:

- ✔ REST est une architecture pour créer des APIs web
- ✔ Utilise les **méthodes HTTP** (`GET`, `POST`, `PUT`, `DELETE`)
- ✔ Symfony permet de créer **une API RESTful avec `Request`, `JsonResponse` et Doctrine**
- ✔ **`FOSRestBundle` simplifie encore plus la gestion des routes et des réponses**

Injection de dépendance:

L'**injection de dépendances (Dependency Injection, DI)** est une méthode qui permet à **Symfony de gérer automatiquement les dépendances** d'une classe (services, repositories, paramètres...). Cela évite de créer manuellement des objets avec `new` et rend le code plus modulaire et maintenable.

Route: Le routing en Symfony permet de faire correspondre une URL à une action d'un contrôleur.

Service : Les services en Symfony sont des classes autonomes utilisées pour centraliser des fonctionnalités réutilisables.

FormBuilder: Symfony propose un **système de gestion de formulaires** avec validation et sécurité intégrée.

Fixtures: les fixtures permettent de remplir la base avec des données de test.

Message flash: Les **sessions** stockent temporairement des informations côté serveur (ex : utilisateur connecté), et les **flash messages** affichent des messages temporaires (ex : "Votre commande a bien été enregistrée").

Petit défis :

- Ne pas pouvoir valider le panier si je suis pas connecté
- Ne pas pouvoir avoir accès au profile si je suis pas co
- Pouvoir modifier le profil et le mdp dans la page profile
- Vider le panier en entier
- Ajouter un footer
- Externaliser le template pour les messages flash
- Externaliser le template pour la navigation
- Créer un template twig pour les message à afficher (flash)
- Avoir une barre de recherche pour rechercher un produit
- Trier les produits par ordre croissant de prix
- Pouvoir commenter les produits et afficher les commentaires
- Pouvoir retrouver les factures dans la page profil avec historique des achats
- Vérifier/mettre à jour le stock/quantité lors d'un achat

Ressources supplémentaires :

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