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="6.4.*" --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 formulaires

- 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

- Créer un service
- Injecter le service dans votre controller pour appeler la méthode qui vous intéresse

19. Traduire le site en multilangues

- Installer Symfony translation
  composer require symfony/translation
- Créer translations/messages.fr.yaml, translations/messages.en.yaml
- Dans services.yaml (parameters: locale:'fr')
- Créer EventSubscriber/LocaleSubscriber.php
- Créer le controlleur qui intercepte le choix de la langue LanguageController.php
- Dans services.yaml, configurer App\EventSubscriber\LocaleSubscriber

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

- Installer la librairie Twig-pack
  composer require symfony/twig-pack
- Créer les fichiers d'erreur dans templates/bundles/TwigBundle/Exception/ error404.html.twig et les autres types d'erreur que vous souhaitez gérer

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

- Définir un fichier nav.html.twig
- Importer le template dans votre template twig
  {% include 'nav.html.twig' %}

22. Mettre en place une pagination

- Installer knp-paginator-bundle
  composer require knplabs/knp-paginator-bundle
- Créer le fichier config/package
  config/package/knp_paginator.yaml
- Injecter PaginatorInterface dans votre controller
- Utiliser knp_pagination_render()

23. Mettre en place une barre de recherche

- Mettre en place un formulaire en GET dans votre nav, avec un input text name search
- Créer une route app_search
- Récupérer le paramètre GET
  $search = $request->query->get('search');
- Créer une méthode dans la couche repository ArticleRepository avec le queryBuilder de Symfony

24. Appeler un controller Symfony en asynchrone avec Javascript

- Créer une méthode dans ArticleController avec une route qui renvoie une response Json
  getArticleJson(int $id, ArticleRepository $articleRepository): JsonResponse

- Appeler la route en asynchrone avec l'api fetch de Javascript
  fetch(`/article/${articleId}/json`)
- Injecter les données récupérées dans le DOM avec Javascript

25. Trier les produits par ordre croissant de prix

- Mettre en place un menu déroulant avec des li a qui appelle la route responsable de trier et ordonner lers données
- Récupérer le paramètre GET 'sort' renvoyé par l'url appelé
  $sort = $request->query->get('sort', 'price_asc');
- Mettre en place une requête personnalisée pour récupérer les données en triant par la colonne et l'ordre définis par le paramètre post

26. Mot de passe oublié

- Installer le bundle Reset Password de Symfony
  composer require symfonycasts/reset-password-bundle
- Créer la vue pour reset le mot de passe
  php bin/console make:reset-password
- Créer un fichier de migration
  php bin/console make:migration
- Mettez à jour la base de données
  php bin/console doctrine:migrations:migrate
- Intégrer la route dans votre formulaire de connexion
  app_forgot_password_request
- Configurer le MAILER_DSN pour l'envoie du mail
  MAILER_DSN

26. Afficher le nombre de commentaires par produit

- Dans la vue qui affiche vos articles
  {{article.reviews|length}} * VOIR LAZY LOADING

27. 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

28. 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

Explications et Jargon

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 »).

🔐 1. Pourquoi les formulaires Symfony sont sécurisés ?

a) Protection contre les attaques CSRF (Cross-Site Request Forgery)

L'attaque CSRF consiste à tromper un utilisateur authentifié pour qu'il effectue **une action non souhaitée** sur un site (exemple : changer un mot de passe, supprimer un compte, etc.).

➡**Symfony protège automatiquement les formulaires contre les attaques CSRF** en ajoutant un **jeton CSRF** (`csrf_token`) qui **valide l'origine de la requête**.

⚙️ 2. Comment fonctionne la protection CSRF dans Symfony ?

a) Ajout automatique du jeton CSRF

Quand tu crées un formulaire avec le Form Builder de Symfony, tu peux activer la protection **avec l'option `csrf_protection: true`** :

b) Comment le jeton CSRF est généré et vérifié ?

1. **Symfony génère un jeton unique** basé sur l'utilisateur et l'identifiant du formulaire (`csrf_token_id`).
2. **Ce jeton est stocké côté serveur** et ajouté automatiquement dans le formulaire sous la forme d'un champ caché.
3. **Quand l'utilisateur soumet le formulaire**, Symfony vérifie si le jeton envoyé correspond à celui stocké côté serveur.
4. **Si le jeton est invalide**, Symfony rejette la requête.

✅ Résumé

- **Le Form Builder de Symfony ajoute automatiquement un jeton CSRF**.
- **Ce jeton protège les actions sensibles (ex: changement de mot de passe)**.
- **Si un attaquant tente de soumettre un formulaire sans jeton valide, Symfony bloque la requête**.
- **C'est une protection essentielle contre les attaques CSRF**.

💡 **En utilisant les formulaires Symfony avec `csrf_protection: true`, tu t'assures que seules les requêtes légitimes sont traitées.**

1️⃣ Principe d'une attaque CSRF

Lorsqu'un utilisateur est connecté à un site (ex : une banque en ligne), le navigateur **conserve son cookie de session** pour éviter qu'il ait à se reconnecter à chaque requête.

🚨 **Problème** : Si le site **n'utilise pas de protection CSRF**, un attaquant peut **forcer l'utilisateur à exécuter une action dangereuse** simplement en le faisant visiter une page piégée.

🚀 Étape par étape : Comment l'attaque fonctionne

1. L'utilisateur est connecté à **banque.com** et son **cookie de session est actif**.
2. Il visite **hacker-site.com**, qui contient une requête cachée (`<img>` ou `<iframe>`).
3. **Son navigateur envoie automatiquement la requête vers banque.com avec son cookie de session.**
4. **Résultat :** 💸 Un virement de 1000€ est effectué vers l'attaquant **sans que l'utilisateur le sache**.

LAZY LOADING

Quand tu fais `{{ article.reviews }}` en Symfony (avec Twig), voici ce qui se passe :

Relation Doctrine :

- Ton entité `Article` a probablement une relation **OneToMany** avec `Review` :

```
#[OneToMany(mappedBy: 'article', targetEntity:
Review::class)]
private Collection $reviews;
```

- Cela signifie qu'un **article** peut avoir plusieurs **reviews**.

Lazy Loading (chargement à la demande) :

- Symfony utilise **Doctrine** qui retourne un objet `Collection` (généralement un `PersistentCollection`).
- Quand `{{ article.reviews }}` est utilisé, Doctrine **charge les reviews** de cet article **seulement quand elles sont utilisées**.

6. **Utilisation en Twig** :

   - Comme `reviews` est une `Collection`, tu peux **boucler dessus** :

```
{% for review in article.reviews %}
    <p>{{ review.content }} - {{ review.author }}</p>
{% endfor %}
```

💡 **Pourquoi ça marche directement ?**

Parce que **Doctrine utilise un proxy** qui permet de récupérer automatiquement les données liées **sans écrire de requête SQL manuelle**. C'est le **lazy loading**.

Petit défis

- Gérer l'ajout d'image sur le serveur
- Vérifier/mettre à jour le stock/quantité lors d'un achat et gerer les msg d'erreur
- Ajouter une session de navigation
- darkmode
- Ajouter des boutons de partage sur les réseaux sociaux dans la page fiche produit

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

- Ne pas pouvoir valider le panier si je suis pas connecté — —
- Ne pas pouvoir avoir accès au profile si je suis pas co — —
- 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) —
- Vider le panier après avoir passer commande—
- Rajouter un lien entre order et user—
- Trier les produits par catégorie —
- Pouvoir retrouver les factures dans la page profil avec historique des achats—
- Traduire le site —
- Pouvoir modifier le profil et le mdp dans la page profile —
- Avoir une barre de recherche pour rechercher un produit —
- Pagination —

- Trier les produits par ordre croissant de prix/date —
- Pouvoir commenter les produits et afficher les commentaires —
- Mot de passe oublié—
- Afficher le nombre de notes par produit ——

Ressources supplémentaires :

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