Documentation du projet - Boutique en ligne Stubborn

. Introduction

Description du projet

Le projet consiste à créer une boutique en ligne pour la marque de sweat-shirts **Stubborn**. Cette application web permettra aux utilisateurs de consulter les produits, de les ajouter à leur panier et de procéder à des achats en ligne. Le système inclut des fonctionnalités d'authentification, une gestion de panier, un back-office pour l'administration et un paiement sécurisé via Stripe.

Objectifs

- Créer une application Symfony connectée à une base de données MySQL.
- Implémenter un système d'authentification avec des rôles client et administrateur.
- Permettre la gestion des produits, des utilisateurs et des commandes.
- Intégrer une solution de paiement avec Stripe.

Technologies utilisées

- **Backend** : Symfony (framework PHP)
- **Base de données** : MySQL
- **Frontend** : HTML, CSS (avec un framework CSS Bootstrap)
- **Système de paiement** : Stripe
- **Authentification** : Symfony Security

Prérequis

- PHP 8.x
- Composer
- Docker et Docker Compose
- Node.js et npm (pour la compilation des assets)
- XAMPP v3.3.0

Installation

- Cloner le repository :
  1. git clone https://github.com/IekoTsu/stubborn.git
  2. cd stubborn
- Installer les dépendances

1. composer install

- Configuration de l'environnement :
  1. vérifier que toutes les variables sont correctes.

- Exécutez Apache et MySQL à partir du panneau de configuration XAMPP.

- Étapes pour importer la base de données (pour tester le projet avec des données déjà fourni):
  1. Une base de données `stubborn.sql` est incluse dans le répertoire `database/` du dépôt.
  2. Créez une base de données nommée `stubborn`
  3. Allez dans l'onglet **Importer** et téléchargez le fichier `stubborn.sql`
  4. Cliquez sur **Exécuter** pour importer les données.
- Ou créer la base de données avec Doctrine :
  1. Si vous préférez générer la base de données à partir des entités Symfony, exécutez la commande suivante :
     - php bin/console doctrine:database:create
     - php bin/console doctrine:migrations:migrate
- Créer la base de donner pour les tests :
  1. php bin/console doctrine:database:create --env=test
  2. php bin/console doctrine:migrations:migrate --env=test

Note : Dans les deux cas si vous rencontrez des probleme de shemas avec les migration exécutez cette commande :

php bin/console doctrine:schema:update --force

php bin/console doctrine:schema:update --force –env=test (Pour la base de donner test)

- **Démarrez le serveur Symfony** :
  - symfony serve

L'application sera maintenant accessible sur `http://localhost:8000`.

Note : **Si vous avez importé la base de données depuis le répertoire, un utilisateur avec le rôle administrateur existe déjà pour tester le back-office.**

Nom utilisateur : **test**

mot de passe : **123qwe**

Sinon, créez un utilisateur et ajoutez-lui le rôle `"ROLE_ADMIN"` via phpMyAdmin

3. Fonctionnalités

3.1. Authentification et gestion des utilisateurs

- **Page de connexion (`/login`)** : Permet aux utilisateurs de se connecter avec leurs identifiants.
- **Page d'inscription (`/register`)** : Permet aux utilisateurs de s'inscrire avec confirmation par email.
- **Rôles** : Il y a deux rôles : **client** (utilisateur standard) et **administrateur** (pour gérer les produits).

Processus d'inscription :

1. L'utilisateur remplit le formulaire d'inscription.
2. Un email de confirmation est envoyé à l'utilisateur.
3. En cliquant sur le lien dans l'email, l'utilisateur est redirigé vers la page d'accueil et connecté automatiquement.

3.2. Page d’accueil (`/`)

- Affiche des informations sur la société **Stubborn** et les produits phares de la marque.
- Si l'utilisateur est connecté, un menu avec des liens vers **Boutique**, **Panier**, et **Se déconnecter** apparaît.
- Si l'utilisateur n'est pas connecté, seuls les liens **Accueil**, **S’inscrire**, et **Se connecter** sont visibles.

3.3. Pages de produits

- **Page de produits (`/products`)** : Liste tous les produits disponibles avec des filtres de prix.

    - **Filtres de prix** : 10-29 €, 29-35 €, 35-50 €.
- **Page d’un produit individuel (`/product/{id}`)** : Affiche les détails d'un produit et permet de l'ajouter au panier.

3.4. Panier (`/cart`)

- Permet à l'utilisateur de voir le contenu de son panier, de retirer un produit, de calculer le total de la commande et de valider la commande.

3.5. Intégration de Stripe

- **Stripe** est intégré pour simuler un paiement en mode développement (bac à sable).
- Un service Symfony gère l'interaction avec l'API Stripe pour simuler un règlement de commande.

3.6. Back-office Admin (`/admin`)

- Cette page est accessible uniquement par les administrateurs.
- Les administrateurs peuvent ajouter, modifier ou supprimer des sweat-shirts dans la boutique.

Documentation des Contrôleurs Symfony

1. BackOfficeController

Routes et Fonctions

- **`/admin` (back_office)**
    - **Accès :** ROLE_ADMIN
    - **Description :** Permet aux administrateurs de gérer les sweat-shirts (ajout, modification, visualisation).
    - **Fonction principale :** Affiche un formulaire pour ajouter un nouveau sweat-shirt et des formulaires de modification pour chaque article existant.
- **`/admin/edit/{id}` (app_sweatshirt_edit)**
    - **Accès :** ROLE_ADMIN
    - **Description :** Permet de modifier un sweat-shirt existant.
    - **Détails :** Lors de la soumission du formulaire, si une nouvelle image est uploadée, l'image précédente est supprimée du système de fichiers et remplacée par la nouvelle.
- **`/admin/delete/{id}` (app_sweatshirt_delete)**
    - **Accès :** ROLE_ADMIN
    - **Description :** Supprime un sweat-shirt spécifique de la base de données.
    - **Retour :** Redirige vers `/admin` avec un message flash confirmant la suppression.

Méthodes Utiles

- **`uploadImage()`** :
    - **Paramètres :** `$imageFile`, `$slugger`
    - **Description :** Gère l'upload de l'image et retourne le nom du fichier.
    - **Gestion des exceptions :** En cas d'échec de l'upload, un message d'alerte est affiché.

2. CartController

Routes et Fonctions

- **`/cart` (app_cart)**
    - **Accès :** ROLE_USER

    - **Description :** Affiche le contenu du panier de l'utilisateur.
- **`/cart/add/{id}` (cart_add)**
    - **Accès :** ROLE_USER
    - **Description :** Ajoute un sweat-shirt au panier.
    - **Validation :** Vérifie si une taille est sélectionnée avant d'ajouter au panier.
- **`/cart/remove/{id}/{size}` (cart_remove)**
    - **Accès :** ROLE_USER
    - **Description :** Retire un article du panier en fonction de l'identifiant et de la taille spécifiée.

3. HomeController

Routes et Fonctions

- **`/` (app_home)**
    - **Description :** Affiche la page d'accueil avec une liste de sweat-shirts en vedette récupérés depuis la base de données.

4. PaymentController

Routes et Fonctions

- **`/payment` (app_payment)**
    - **Accès :** ROLE_USER
    - **Description :** Initialise une session de paiement Stripe avec les articles du panier.
    - **Validation :** Redirige l'utilisateur vers le panier si celui-ci est vide.
- **`/payment/success` (payment_success)**
    - **Accès :** ROLE_USER
    - **Description :** Met à jour le stock des sweat-shirts achetés après un paiement réussi.
    - **Détails :** Réduit le stock de l'article en fonction de la taille sélectionnée.

Configuration Stripe

- Utilisation de l'API Stripe avec la clé secrète (`sk_test_...`).
- Gestion des sessions de paiement avec `Stripe\Checkout\Session`.

- URLs de succès et d'annulation définies sur `http://localhost:8000/payment/success` et `http://localhost:8000/payment/cancel`.

Note : Pour tester un paiement avec **Stripe**, utilisez :

la carte bancaire suivante : **4242 4242 4242 4242**

avec une date d'expiration valide, un code **CVC**, et un **nom** aléatoire.

5. ProductController

Routes et Fonctions

- **/product/{id}** (app_product)
  - **Accès** : ROLE_USER
  - **Description** : Affiche les détails d'un sweat-shirt spécifique.
  - **Fonction principale** : Permet aux utilisateurs connectés de consulter les informations détaillées d'un produit (sweat-shirt), telles que le nom, le prix, et la taille.
  - **Détails** : La page utilise le template `product/index.html.twig` pour le rendu.

6. ProductsController

Routes et Fonctions

- **/products** (app_products)
  - **Accès** : ROLE_USER
  - **Description** : Affiche une liste de tous les sweat-shirts disponibles.
  - **Fonction principale** : Permet aux utilisateurs connectés de voir l'ensemble des produits et de les parcourir.
  - **Détails** : Récupère tous les produits via le `SweatShirtsRepository` et les affiche dans le template `products/index.html.twig`.

7. RegistrationController

Routes et Fonctions

- **/register** (app_register)
  - **Accès** : Ouvert à tous
  - **Description** : Permet aux utilisateurs de s'inscrire sur le site.
  - **Fonction principale** : Affiche un formulaire d'inscription. Vérifie si les mots de passe correspondent et crée un nouvel utilisateur en cas de succès.
  - **Détails** :
    - Si le formulaire est valide, enregistre l'utilisateur avec un mot de passe haché.

        - Envoie un email de confirmation via `EmailVerifier`.
        - Connecte automatiquement l'utilisateur après inscription.
- **/verify/email** (app_verify_email)
    - **Accès** : Ouvert à tous
    - **Description** : Vérifie le lien d'activation envoyé par email pour confirmer l'adresse de l'utilisateur.
    - **Fonction principale** : Valide le compte utilisateur en activant le champ `isVerified`.
    - **Détails** :
        - Redirige vers la page d'inscription en cas d'erreur.
        - Affiche un message de succès si l'email est vérifié avec succès.

8. SecurityController

Routes et Fonctions

- **/login** (app_login)
    - **Accès** : Ouvert à tous
    - **Description** : Gère l'authentification des utilisateurs.
    - **Fonction principale** : Affiche un formulaire de connexion et traite les erreurs d'authentification.
    - **Détails** :
        - Récupère le dernier nom d'utilisateur entré et les erreurs d'authentification via `AuthenticationUtils`.
        - Affiche un message d'erreur personnalisé en cas d'identifiants incorrects.
- **/logout** (app_logout)
    - **Accès** : Utilisateurs connectés
    - **Description** : Gère la déconnexion des utilisateurs.
    - **Fonction principale** : Cette route est interceptée par le système de sécurité Symfony pour effectuer la déconnexion.
    - **Détails** : Ne contient pas de logique, la déconnexion est gérée par le `firewall` de Symfony.

Explication des rôles et permissions

- **ROLE_USER** : Les utilisateurs avec ce rôle peuvent accéder aux pages des produits et voir les détails des articles.
- **ROLE_ADMIN** : Accès aux fonctionnalités du back-office pour la gestion des produits (non inclus dans cette documentation mais généralement réservé à la gestion des inventaires, des commandes, etc.).

Le Event Listener `RunTestsOnAppStartListener:`

Description

L'EventListener **`RunTestsOnAppStartListener`** est utilisé pour exécuter automatiquement une suite de tests PHPUnit lors du démarrage de l'application Symfony. Il est déclenché à chaque requête entrante au serveur web, mais avec une vérification en cache pour éviter des exécutions répétées dans un court laps de temps.

Fonctionnalités

- **Exécution Automatique des Tests** : L'écouteur exécute une série de tests PHPUnit dès que l'application démarre, à moins qu'il ait déjà été exécuté récemment.
- **Mise en Cache** : Utilisation d'un cache pour marquer si les tests ont été exécutés afin de les éviter pendant une période définie (30 minutes).
- **Journalisation** : Tous les résultats des tests, ainsi que les erreurs éventuelles, sont enregistrés dans les logs pour faciliter le suivi des tests.

Dépendances

- `KernelInterface` : Utilisé pour obtenir le répertoire racine du projet.
- `LoggerInterface` : Pour enregistrer des messages dans les logs (info, debug, erreurs).
- `FilesystemAdapter` (Cache Symfony) : Pour stocker temporairement un indicateur si les tests ont déjà été exécutés.
- `Process` (Symfony) : Pour exécuter des commandes shell, en l'occurrence la commande PHPUnit.

Routes et Événements

- **Événement Écouté** : `kernel.request`
    - Cet écouteur est lié à l'événement `RequestEvent` qui est déclenché au début de chaque requête HTTP.

Tests Associés

Les tests PHPUnit exécutés par cet EventListener se trouvent principalement dans deux fichiers :

- `PaymentControllerTest.php` : Teste la logique de paiement, incluant les redirections Stripe, la mise à jour des stocks, et la gestion du panier.
- `CartServiceTest.php` : Teste les fonctionnalités du service de gestion du panier (ajout, suppression, récupération d'articles).

1. Tests pour le Contrôleur de Paiement (`PaymentControllerTest`)

- **testStripeCheckoutRedirectsToSessionUrl()** : Vérifie que l'utilisateur est redirigé vers la page de paiement Stripe.
- **testSuccessUpdatesStockAndClearsCart()** : Vérifie que les stocks sont mis à jour et que le panier est vidé après un paiement réussi.
- **testCancelAddsFlashMessageAndRedirects()** : Vérifie qu'un message flash est ajouté et que l'utilisateur est redirigé après une annulation de paiement.

2. Tests pour le Service de Panier (`CartServiceTest`)

- **testAddProduct()** : Vérifie l'ajout de produits au panier.
- **testGetCartItems()** : Vérifie la récupération correcte des articles du panier.
- **testRemoveProduct()** : Vérifie la suppression d'un produit spécifique du panier.
- **testClearCart()** : Vérifie que le panier est vidé

Pour exécuter les tests manuellement en dehors de cet EventListener :

- php vendor/bin/phpunit --configuration phpunit.xml.dist --testdox