Documentation de l'application Stubborn

Boutique en ligne de sweat-shirts

Version 1.0

Date : 5 mai 2025

Développé par : Antoine Papin

Table des matières

1 Introduction

L'application Stubborn est une boutique en ligne développée avec le framework Symfony 7.2, conçue pour vendre des sweat-shirts personnalisés. Elle offre une interface utilisateur intuitive pour les clients et un back-office robuste pour les administrateurs. Ce document présente les fonctionnalités principales, l'architecture technique, et les instructions pour installer et utiliser l'application.

1.1 Objectifs

L'objectif principal de Stubborn est de fournir une plateforme e-commerce simple et efficace, permettant aux utilisateurs de :

— Parcourir et acheter des sweat-shirts disponibles en différentes tailles (XS à XL).
— Gérer leur panier et effectuer des paiements sécurisés via Stripe.
— Administrer les produits (ajout, modification, suppression) via un back-office.
— Gérer les comptes utilisateurs avec authentification et vérification d'e-mail.

2 Fonctionnalités principales

L'application Stubborn propose les fonctionnalités suivantes, organisées en deux interfaces : client et administrateur.

2.1 Interface client

— **Boutique** : Affiche la liste des sweat-shirts avec leurs images, noms, prix, et tailles disponibles. Les sweat-shirts mis en avant sont highlightés sur la page d'accueil.
— **Fiche produit** : Permet de consulter les détails d'un sweat-shirt et d'ajouter au panier en sélectionnant une taille.
— **Panier** : Permet d'ajouter, supprimer des articles, et visualiser le total avant le paiement.
— **Paiement** : Intègre Stripe pour des paiements sécurisés. En mode test, utilise la carte 4242 4242 4242 4242.
— **Compte utilisateur** : Permet aux utilisateurs de s'inscrire, se connecter, vérifier leur e-mail, et gérer leur profil (nom, adresse de livraison).

2.2 Back-office administrateur

— **Gestion des produits** : Permet de modifier le nom, le prix, l'image, les stocks (XS, S, M, L, XL), et l'état "mis en avant" des sweat-shirts existants.
— **Ajout de produits** : Formulaire pour ajouter de nouveaux sweat-shirts avec les mêmes champs que pour la modification.
— **Suppression** : Suppression sécurisée des produits avec vérification CSRF.

3 Architecture technique

L'application est construite sur une architecture MVC avec Symfony 7.2, utilisant des technologies modernes pour une expérience fluide.

3.1 Technologies utilisées

— **Backend** : Symfony 7.2 (PHP 8.2+), Doctrine ORM pour la gestion des entités.

— **Frontend** : Twig pour les templates, Bootstrap pour les styles, Webpack Encore pour la gestion des assets JavaScript et CSS, Symfony UX Turbo pour une navigation dynamique sans rechargement complet des pages.
— **Paiement** : API Stripe pour le traitement des paiements.
— **Base de données** : MySQL/PostgreSQL (configurable via Doctrine).
— **Sécurité** : Symfony Security pour l'authentification, gestion des rôles, et vérification CSRF.
— **E-mails** : Symfony Mailer pour l'envoi de confirmations (inscription, commande).
— **Vérification d'e-mail** : SymfonyCasts VerifyEmail pour la vérification des adresses e-mail.

3.2 Structure des entités

Les principales entités de l'application sont :
— **Sweatshirt** : Représente un produit sweat-shirt. Champs : `id`, `name`, `price`, `isFeatured`, `image`, `stocks` (relation One-to-Many avec Stock).
— **Stock** : Représente une entrée de stock pour une taille spécifique. Champs : `id`, `size` (XS à XL), `quantity`, `sweatshirt` (relation Many-to-One avec Sweatshirt).
— **User** : Représente un utilisateur. Champs : `id`, `email`, `roles`, `password`, `name`, `deliveryAddress`, `isVerified`.

3.3 Contrôleurs principaux

Les contrôleurs gèrent la logique métier et les interactions utilisateur :
— **AdminController** : Gère le back-office, permettant l'ajout, la modification, et la suppression des sweat-shirts. Inclut la gestion des formulaires et des images.
— **CartController** : Gère le panier (ajout, suppression, affichage), le paiement via Stripe, et l'envoi d'e-mails de confirmation.
— **HomeController** : Affiche la page d'accueil avec les sweat-shirts mis en avant.
— **LoginController** : Gère la connexion et la déconnexion des utilisateurs.
— **ProductController** : Gère l'affichage des détails d'un sweat-shirt et l'ajout au panier.
— **ProductsController** : Affiche la liste des sweat-shirts avec un filtre par fourchette de prix.
— **RegistrationController** : Gère l'inscription des utilisateurs et la vérification d'e-mail.

3.4 Formulaires

Les formulaires définissent les interfaces pour la saisie des données :
— **SweatshirtType** : Formulaire pour ajouter un nouveau sweat-shirt, avec les champs `name`, `price`, `isFeatured`, et `image`.
— **SweatshirtInlineType** : Formulaire pour modifier un sweat-shirt existant dans le back-office, avec les mêmes champs et une gestion des stocks.
— **RegistrationFormType** : Formulaire pour l'inscription des utilisateurs, avec les champs `name`, `email`, `plainPassword`, et `deliveryAddress`.

3.5 Services

Les services encapsulent la logique métier réutilisable :
— **StripeService** : Gère les interactions avec l'API Stripe, incluant la création, la confirmation, et la récupération des PaymentIntents pour les paiements.

3.6 Listeners d'événements

Les listeners gèrent les événements spécifiques :

— **ExceptionListener** : Intercepte les exceptions d'accès interdit (`AccessDeniedHttpException`) pour rediriger les utilisateurs non connectés vers la page de connexion ou afficher un message d'erreur pour les utilisateurs connectés sans autorisation.

3.7 Repositories

Les repositories fournissent des méthodes personnalisées pour accéder aux données :

— **SweatshirtRepository** : Fournit des méthodes pour récupérer les sweat-shirts, notamment `findByCriteria` pour filtrer par critères (par exemple, fourchette de prix).
— **StockRepository** : Gère les entrées de stock, avec des méthodes standard pour accéder aux données.
— **UserRepository** : Gère les utilisateurs, avec une méthode `upgradePassword` pour mettre à jour les mots de passe hachés.

3.8 Sécurité

Les composants de sécurité assurent l'authentification et la vérification des utilisateurs :

— **CustomFormLoginAuthenticator** : Gère l'authentification par formulaire, vérifiant les identifiants, l'état de vérification de l'e-mail, et redirigeant après connexion.
— **EmailVerifier** : Gère l'envoi et la validation des e-mails de confirmation pour vérifier les adresses des utilisateurs.

3.9 Routes principales

— `/` : Page d'accueil (`app_home`).
— `/products` : Liste des sweat-shirts (`app_products`).
— `/product/{id}` : Détails d'un sweat-shirt (`app_product`).
— `/cart` : Affichage du panier (`app_cart`).
— `/cart/add/{id}` : Ajout au panier (`app_cart_add`).
— `/cart/remove/{index}` : Suppression d'un article du panier (`app_cart_remove`).
— `/cart/payment` : Page de paiement (`app_cart_payment`).
— `/cart/checkout` : Validation du paiement (`app_cart_checkout`).
— `/cart/create-payment-intent` : Création d'un PaymentIntent Stripe (`app_cart_create_payment_int`
— `/admin` : Back-office pour la gestion des produits (`app_admin`).
— `/admin/delete/{id}` : Suppression d'un sweat-shirt (`app_admin_delete`).
— `/login` : Connexion utilisateur (`app_login`).
— `/logout` : Déconnexion utilisateur (`app_logout`).
— `/register` : Inscription utilisateur (`app_register`).
— `/verify/email` : Vérification de l'e-mail (`app_verify_email`).

4 Instructions d'installation

Pour installer et exécuter l'application localement, suivez ces étapes dans l'ordre. Les assets JavaScript et CSS doivent être compilés avant de démarrer le serveur.

4.1 Prérequis

— PHP 8.2 ou supérieur
— Composer
— Node.js (version 14.x ou supérieure) et npm

— MySQL/PostgreSQL
— Clé API Stripe (publique et secrète)

4.2 Étapes d'installation

1. Cloner le dépôt : `git clone https://github.com/Papinte/stubborn`
2. Installer les dépendances PHP : `composer install`
3. Installer les dépendances JavaScript (inclut `@symfony/stimulus-bridge` et `@symfony/ux-turbo`) : `npm install`
4. Compiler les assets avec Webpack Encore pour générer `public/build/entrypoints.json` : `npm run dev`
5. Configurer la base de données dans `.env` :
   `DATABASE_URL="mysql://root:@127.0.0.1:3306/stubborn?serverVersion=10.4.32-MariaDBchars`
6. Créer la base de données : `php bin/console doctrine:database:create`
7. Exécuter les migrations : `php bin/console doctrine:migrations:migrate`
8. Configurer les clés Stripe dans `.env` :
   `STRIPE_PUBLIC_KEY=pk_test_...`
   `STRIPE_SECRET_KEY=sk_test_...`
9. Lancer le serveur :
   — Sur Windows, exécuter le script : `.\run-tests-and-start.bat` (après avoir compilé les assets avec `npm run dev`).
   — Sur d'autres systèmes : `symfony server:start`

4.3 Accès

— Boutique : `http://localhost:8000/products`
— Back-office : `http://localhost:8000/admin`
— Connexion/Inscription : `http://localhost:8000/login`, `http://localhost:8000/register`

5 Conclusion

L'application Stubborn est une solution e-commerce complète pour la vente de sweat-shirts, avec une interface utilisateur moderne, un back-office fonctionnel, et une gestion sécurisée des utilisateurs. Grâce à Symfony UX Turbo, la navigation est fluide et rapide. Les futures évolutions pourraient inclure l'ajout de filtres avancés dans la boutique, la gestion des commandes, ou l'intégration de promotions. Pour toute question, contactez Antoine Papin à antoine.papin.dev@gmail.com.