📚 Documentation Technique - Stubborn E-commerce

Table des matières

Architecture

Pattern utilisé : Action Controller

Chaque controller ne contient qu'**une seule action** grâce à la méthode `__invoke()`.

Avantages :

- Code plus maintenable
- Responsabilité unique (SOLID)
- Tests plus faciles
- Routes plus explicites

Exemple :

```
// src/Controller/Product/ProductShowController.php
class ProductShowController extends AbstractController
{
    #[Route('/product/{id}', name: 'app_product_show')]
    public function __invoke(Product $product, Request $request): Response
    {
        // Une seule responsabilité : afficher un produit
    }
}
```

Entités et Relations

Schéma de base de données

```
User (utilisateur)
├── id (PK)
├── email (unique)
├── password (hashed)
├── name
├── deliveryAddress
├── roles (JSON)
├── isVerified (boolean)
├── verificationToken
└── orders (OneToMany → Order)

Product (produit)
├── id (PK)
├── name
├── price (DECIMAL)
├── image (string)
├── featured (boolean)
├── stocks (OneToMany → Stock)
└── orderItems (OneToMany → OrderItem)

Stock (stock par taille)
├── id (PK)
├── product_id (FK → Product)
├── size (XS, S, M, L, XL)
└── quantity (integer)

Order (commande)
├── id (PK)
├── user_id (FK → User)
├── totalPrice (DECIMAL)
├── status (pending, paid, cancelled)
├── createdAt (DateTime)
├── stripeSessionId
└── orderItems (OneToMany → OrderItem)

OrderItem (ligne de commande)
├── id (PK)
├── order_id (FK → Order)
├── product_id (FK → Product)
├── size
├── quantity
└── unitPrice (DECIMAL)
```

Relations

- **User ↔ Order** : One-to-Many
  Un utilisateur peut avoir plusieurs commandes
- **Product ↔ Stock** : One-to-Many
  Un produit a plusieurs stocks (un par taille)
- **Order ↔ OrderItem** : One-to-Many
  Une commande contient plusieurs lignes
- **Product ↔ OrderItem** : One-to-Many
  Un produit peut apparaître dans plusieurs commandes

Services

CartService

Responsabilité : Gestion du panier en session

Méthodes principales :

```
add(int $productId, string $size): void
remove(int $productId, string $size): void
updateQuantity(int $productId, string $size, int $quantity): void
getCartWithDetails(): array
getTotal(): float
getItemCount(): int
clear(): void
isEmpty(): bool
```

Stockage : Session Symfony (`cart` key)

Structure de données :

```
[
    '1_M' => [
        'productId' => 1,
        'size' => 'M',
        'quantity' => 2,
    ],
    '2_L' => [
        'productId' => 2,
        'size' => 'L',
        'quantity' => 1,
    ],
]
```

OrderService

Responsabilité : Gestion des commandes

Méthodes principales :

```
createOrderFromCart(User $user, array $cartItems, ? string $stripeSessionId
= null): Order
finalizeOrder(Order $order): void  // Change status + décrémente stock
cancelOrder(Order $order): void
```

Cycle de vie d'une commande :

1. **PENDING** : Commande créée, paiement en cours
2. **PAID** : Paiement validé, stock décrémenté
3. **CANCELLED** : Commande annulée

StripeService

Responsabilité : Intégration avec l'API Stripe

Méthodes principales :

```
createCheckoutSession(array $cartItems, User $user): Session
retrieveSession(string $sessionId): Session
```

Configuration requise :

```
STRIPE_PUBLIC_KEY=pk_test_xxxxx
STRIPE_SECRET_KEY=sk_test_xxxxx
```

URLs de redirection :

- Success : `/payment/success? session_id={CHECKOUT_SESSION_ID}`
- Cancel : `/payment/cancel`

EmailService

Responsabilité : Envoi d'emails

Méthodes principales :

```
sendVerificationEmail(User $user): void
sendOrderConfirmation(User $user, string $orderNumber, float $total): void
```

Templates utilisés :

- `emails/verification.html. twig`
- `emails/order_confirmation.html.twig`

Sécurité

Configuration (security.yaml)

```
access_control:
    - { path: ^/login, roles: PUBLIC_ACCESS }
    - { path: ^/register, roles: PUBLIC_ACCESS }
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/cart, roles: ROLE_USER }
    - { path: ^/product, roles: ROLE_USER }
```

Rôles

- **ROLE_USER** : Client connecté (accès boutique + panier)
- **ROLE_ADMIN** : Administrateur (accès back-office)

Authenticator

LoginFormAuthenticator gère :

- Connexion par email/password
- CSRF protection
- Remember Me
- Redirection après login

Vérification email

Processus :

1. Inscription → génération d'un token aléatoire
2. Email envoyé avec lien `/verify/{token}`
3. Clic sur le lien → `isVerified = true` + connexion auto

Formulaires

LoginType

```
email (EmailType)
password (PasswordType)
_remember_me (CheckboxType)
```

RegistrationType

```
name (TextType)
email (EmailType)
password (RepeatedType: PasswordType)
deliveryAddress (TextareaType)
```

AddToCartType

```
size (ChoiceType: XS, S, M, L, XL)
```

ProductType (Admin)

```
name (TextType)
price (MoneyType)
image (FileType)
featured (CheckboxType)
stock_XS, stock_S, stock_M, stock_L, stock_XL (IntegerType)
```

Gestion d'upload : Les images sont stockées dans `public/uploads/products/`

Flux de paiement Stripe

Étapes

1. **Utilisateur clique "Finaliser ma commande"**
   → `CheckoutController` est appelé
2. **Création de la commande en BDD**
   → Status: `PENDING`
3. **Création d'une Checkout Session Stripe**

```
4. $session = Session::create([
5.     'payment_method_types' => ['card'],
6.     'line_items' => [... ],
7.     'mode' => 'payment',
8.     'success_url' => '...  /payment/success?
   session_id={CHECKOUT_SESSION_ID}',
9.     'cancel_url' => '. ../payment/cancel',
10.  ]);
```

11. **Redirection vers Stripe Checkout**
    → L'utilisateur entre ses informations de paiement
12. **Stripe redirige vers success_url**
    → `PaymentSuccessController` vérifie la session
13. **Finalisation de la commande**
    - Status passe à `PAID`
    - Stock décrémenté
    - Panier vidé
    - Email de confirmation envoyé

Annulation

Si l'utilisateur annule → redirection vers `/payment/cancel`
Le panier est conservé.

Tests

Structure

```
tests/
├── Service/
│   ├── CartServiceTest.php
│   └── OrderServiceTest.php
└── Functional/
    └── CheckoutTest.php
```

Exécution

```
php bin/phpunit
```

Coverage

- **CartService** : 100%
  - Ajout/suppression
  - Calcul du total
  - Gestion des quantités
- **OrderService** : 80%
  - Création de commande
  - Finalisation
  - Annulation
- **Checkout** : Fonctionnel (simulation)

API Reference

Repositories personnalisés

ProductRepository

```
// Récupérer les produits mis en avant
findFeaturedProducts(): array

// Filtrer par fourchette de prix
findByPriceRange(? float $minPrice, ?float $maxPrice): array
```

UserRepository

```
// Mettre à jour le mot de passe (implémente PasswordUpgraderInterface)
upgradePassword(PasswordAuthenticatedUserInterface $user, string
$newHashedPassword): void
```

Bonnes pratiques appliquées

SOLID

- ✅ **S**ingle Responsibility : Action Controllers
- ✅ **O**pen/Closed : Services extensibles
- ✅ **L**iskov Substitution : Interfaces respectées
- ✅ **I**nterface Segregation : Services spécialisés
- ✅ **D**ependency Inversion : Injection de dépendances

DRY (Don't Repeat Yourself)

- Services réutilisables (Cart, Email, Stripe)
- Template de base (`base.html.twig`)
- Fixtures centralisées

Sécurité

- ✅ Mots de passe hashés (bcrypt)
- ✅ Protection CSRF
- ✅ Validation des formulaires
- ✅ Vérification des rôles
- ✅ Emails de vérification

Version du projet : 1.0.0
Symfony Version : 7.0^