Créer une application e-commerce avec Symfony 8.0

Guide complet pour créer une application e-commerce avec Symfony 8.0 : produits, catégories, panier, paiement Stripe, commandes, authentification et interface d'administration.

Symfony Programmation 120 min de lecture Niveau Intermédiaire

Document généré le 11/07/2026 à 20h42 · nouvy.fr/wiki/symfony-ecommerce-application

Sommaire

71 section(s) · 120 min de lecture

Étape 7 : Créer le template de base

Étape 8 : Migrations supplémentaires

Étape 9 : Générer des données de test avec Doctrine Fixtures

- ↳ Installer les packages nécessaires
- ↳ Créer les fixtures avec make:fixtures
- ↳ Structure des fixtures
- ↳ Gérer les dépendances entre fixtures
- ↳ Charger les fixtures
- ↳ Concepts importants
- ↳ Exemple de fixture complète

Étape 10 : Créer la page d'accueil publique

- ↳ Créer le controller Accueil
- ↳ Créer le template de base pour l'accueil
- ↳ Créer la vue accueil
- ↳ Différences entre base.html.twig et base_accueil.html.twig
- ↳ Points importants

Étape 11 : Créer la page produit détaillée

- ↳ Générer le contrôleur
- ↳ Créer le template product/show.html.twig
- ↳ Mettre à jour le lien sur la page d'accueil

Étape 12 : Système de panier et page panier

- ↳ Concept du panier par session
- ↳ Générer le CartController
- ↳ Créer le template cart/index.html.twig
- ↳ Ajouter le lien panier dans la navigation

Étape 13 : Page de validation de commande

- ↳ Générer le CheckoutController
- ↳ Créer le template checkout/index.html.twig

Étape 14 : Intégration Stripe pour le paiement par carte bancaire

- ↳ Créer un compte Stripe de test
- ↳ Installer le SDK Stripe

↳ Configurer les clés API

↳ Générer le PaymentController

↳ Créer les templates de paiement

↳ Générer le contrôleur Webhook Stripe

↳ Désactiver le firewall pour le webhook

↳ Installer Stripe CLI

↳ Se connecter à Stripe CLI

↳ Transférer les webhooks vers votre serveur local (forward-to)

↳ Déclencher des événements de test

↳ Flux de test complet

↳ Cartes de test Stripe

Ce guide vous explique comment créer une application e-commerce complète avec Symfony 8.0, incluant la gestion des produits, catégories, commandes, authentification utilisateur et interface d'administration.

Prérequis : Avoir Symfony 8.0 installé et configuré.

□ Vue d'ensemble

Ce guide vous permettra de créer :

- □ **Catalogue produits** avec catégories (relation ManyToMany)
- □ **Gestion des commandes** (Order et OrderItem)
- □ **Authentification utilisateur** (inscription, connexion, vérification email)
- □ **Interface d'administration** (CRUD pour toutes les entités)
- □ **Sécurité** (protection des routes par rôles)

□ Étape 1 : Créer le projet Symfony

Créer un nouveau projet

```
# Créer un nouveau projet Symfony 8.0 avec webapp
symfony new mon-ecommerce --version='8.0.*' --webapp

# Ou avec Composer
composer create-project symfony/skeleton:'8.0.*' mon-ecommerce
cd mon-ecommerce
composer require webapp
```

Copier

Installer les packages nécessaires

```
cd mon-ecommerce

# ORM Doctrine (base de données)
composer require symfony/orm-pack

# Maker Bundle (générateur de code)
composer require --dev symfony/maker-bundle

# Security Bundle (authentification)
composer require symfony/security-bundle

# Formulaires
composer require symfony/form

# Validation
composer require symfony/validator

# Twig (templates)
composer require symfony/twig-pack

# Mailer (emails)
composer require symfony/mailer

# Vérification d'email
composer require symfonycasts/verify-email-bundle

# Profiler (débogage)
composer require --dev symfony/profiler-pack
```

Copier

Étape 2 : Configuration de la base de données

Configurer la connexion

Éditez le fichier `.env` :

```
DATABASE_URL='mysql://root:password@127.0.0.1:3306/ecommerce_db?serverVersion=8.0.32&charset
=utf8mb4'
```

Copier

Créer la base de données

```
php bin/console doctrine:database:create
```

Copier

Étape 3 : Créer les entités

Entité Category (Catégorie)

```
php bin/console make:entity Category
```

Copier

Remplissez les champs suivants :

- `name` (string, 255, not null)
- `description` (string, 255, nullable)
- `slug` (string, 255, not null)
- `createdAt` (datetime_immutable, nullable)
- `updatedAt` (datetime_immutable, nullable)

Entité Product (Produit)

```
php bin/console make:entity Product
```

Copier

Remplissez les champs suivants :

- `name` (string, 255, not null)
- `description` (string, 255, nullable)
- `price` (decimal, precision: 10, scale: 2, not null)
- `stock` (integer, not null)
- `image` (string, 255, nullable)
- `slug` (string, 255, not null)
- `createdAt` (datetime_immutable, nullable)
- `updatedAt` (datetime_immutable, nullable)
- `categories` (relation ManyToMany vers Category)

Important : Lors de la création de la relation ManyToMany, choisissez :

- Type de relation : `ManyToMany`
- Entité cible : `Category`
- Propriété dans Category : `products`
- Voulez-vous ajouter une nouvelle propriété dans Category ? : `yes`
- Relation propriétaire : `Product` (côté Product)

Entité User (Utilisateur)

```
php bin/console make:user User
```

Copier

Choisissez :

- Utiliser un email comme identifiant : `yes`
- Stocker le mot de passe dans la base de données : `yes`

Ensuite, ajoutez des champs supplémentaires :

```
php bin/console make:entity User
```

Copier

Ajoutez :

- `firstName` (string, 255, not null)
- `lastName` (string, 255, not null)
- `address` (string, 255, nullable)
- `city` (string, 255, nullable)
- `postalCode` (string, 20, nullable)
- `phone` (string, 50, nullable)
- `isVerified` (boolean, not null, default: false)

Entité Order (Commande)

```
php bin/console make:entity Order
```

Copier

Remplissez les champs suivants :

- `orderNumber` (string, 50, not null)
- `status` (string, 50, not null)
- `total` (decimal, precision: 10, scale: 2, not null)
- `createdAt` (datetime_immutable, nullable)
- `updatedAt` (datetime_immutable, nullable)
- `user` (relation ManyToOne vers User, not null)

Note : Doctrine créera automatiquement une table nommée `` `order` `` (avec backticks car "order" est un mot réservé SQL).

Entité OrderItem (Article de commande)

```
php bin/console make:entity OrderItem
```

Copier

Remplissez les champs suivants :

- `quantity` (integer, not null)
- `price` (decimal, precision: 10, scale: 2, not null)
- `orderId` (relation ManyToOne vers Order, not null)
- `product` (relation ManyToOne vers Product, not null)

Explication de la relation ManyToMany

La relation ManyToMany entre Product et Category permet :

- **Un produit peut appartenir à plusieurs catégories** : Un produit peut être classé dans plusieurs catégories simultanément

- **Une catégorie peut contenir plusieurs produits** : Une catégorie peut regrouper de nombreux produits
- **Table de jointure automatique** : Doctrine crée automatiquement la table `product_category` pour gérer la relation

Structure de la table de jointure :

- `product_id` (clé étrangère vers `product`)
- `category_id` (clé étrangère vers `category`)
- Clé primaire composite : `(product_id, category_id)`

Générer les migrations

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

Copier

□ Étape 4 : Configuration de l'authentification

Créer l'authentification

```
php bin/console make:auth
```

Copier

Choisissez :

- Type d'authentification : `Login form authenticator`
- Nom de la classe : `AppAuthenticator`
- Route de redirection après connexion : `/home`

Configurer Security

Le fichier `config/packages/security.yaml` est automatiquement configuré. Vérifiez qu'il contient :

```
security:
    password_hashers:
        SymfonyComponentSecurityCoreUserPasswordAuthenticatedUserInterface: 'auto'
    providers:
        app_user_provider:
            entity:
                class: AppEntityUser
                property: email
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|assets|build)/
            security: false
        main:
            lazy: true
            provider: app_user_provider
            form_login:
                login_path: app_login
                check_path: app_login
                csrf_token_id: authenticate
                username_parameter: email
                password_parameter: password
            logout:
                path: app_logout
                target: app_home
    access_control:
        - { path: ^/admin, roles: ROLE_ADMIN }
        - { path: ^/profile, roles: ROLE_USER }
```

Copier

Créer le contrôleur d'authentification

```
php bin/console make:controller SecurityController
```

Copier

Le contrôleur généré contient les méthodes `login()` et `logout()`.

Créer le contrôleur d'inscription

```
php bin/console make:registration-form
```

Copier

Choisissez :

- Voulez-vous ajouter la vérification d'email ? : `yes`
- Classe EmailVerifier : `AppSecurityEmailVerifier`

Le système génère automatiquement :

- `RegistrationController` avec gestion de l'inscription et vérification d'email
- `RegistrationFormType` avec les champs : firstName, lastName, email, agreeTerms, plainPassword
- `EmailVerifier` pour la vérification d'email
- Templates Twig pour l'inscription et la confirmation d'email

Adapter le formulaire d'inscription

Le formulaire `RegistrationFormType` est généré automatiquement avec les champs de base. Il inclut :

- `firstName` et `lastName` (mappés à l'entité User)
- `email` (mappé à l'entité User)
- `agreeTerms` (non mappé, avec validation IsTrue)
- `plainPassword` (non mappé, avec validation NotBlank et Length)

Le mot de passe est automatiquement hashé dans le contrôleur avant la persistance.

□ Étape 5 : Créer les CRUD avec make:crud

CRUD pour Category

```
php bin/console make:crud Category
```

Copier

Choisissez :

- Voulez-vous générer les routes avec des préfixes ? : `no`
- Classe du contrôleur : `CategoryController`

Le système génère automatiquement :

- `CategoryController` avec les actions : index, new, show, edit, delete
- `CategoryType` (formulaire)
- Templates Twig : index, new, show, edit, _form, _delete_form

CRUD pour Product

```
php bin/console make:crud Product
```

Copier

Le système génère automatiquement :

- `ProductController` avec les actions : index, new, show, edit, delete
- `ProductType` (formulaire)
- Templates Twig : index, new, show, edit, _form, _delete_form

CRUD pour Order

```
php bin/console make:crud Order
```

Copier

Le système génère automatiquement :

- `OrderController` avec les actions : index, new, show, edit, delete
- `OrderType` (formulaire)
- Templates Twig : index, new, show, edit, _form, _delete_form

CRUD pour OrderItem

```
php bin/console make:crud OrderItem
```

Copier

Le système génère automatiquement :

- `OrderItemController` avec les actions : index, new, show, edit, delete
- `OrderItemType` (formulaire)
- Templates Twig : index, new, show, edit, _form, _delete_form

Adapter les formulaires

Les formulaires générés par `make:crud` incluent tous les champs de l'entité. Pour les relations ManyToMany et ManyToOne, adaptez les formulaires :

Dans `ProductType.php` :

- Le champ `categories` est automatiquement généré en `EntityType` avec `multiple => true`
- Vous pouvez améliorer l'affichage en changeant `choice_label` de `id` à `name`

Dans `OrderType.php` :

- Le champ `user` est automatiquement généré en `EntityType`
- Vous pouvez améliorer l'affichage en changeant `choice_label` de `id` à `email`

Dans `OrderItemType.php` :

- Les champs `orderId` et `product` sont automatiquement générés en `EntityType`
- Vous pouvez améliorer l'affichage en changeant `choice_label` de `id` à un champ plus lisible

Protéger les routes CRUD

Ajoutez l'attribut `#[IsGranted('ROLE_ADMIN')]` au-dessus de chaque classe de contrôleur CRUD :

```
use SymfonyComponentSecurityHttpAttributeIsGranted;

#[IsGranted('ROLE_ADMIN')]
#[Route('/product')]
final class ProductController extends AbstractController
{
    // ...
}
```

Copier

Cela protège toutes les routes du contrôleur et nécessite le rôle `ROLE_ADMIN` pour y accéder.

Étape 6 : Créer le HomeController

```
php bin/console make:controller HomeController
```

Copier

Créez une route protégée par `ROLE_USER` :

```
use SymfonyComponentSecurityHttpAttributeIsGranted;

#[IsGranted('ROLE_USER')]
#[Route('/home', name: 'app_home')]
public function index(): Response
{
    return $this->render('home/index.html.twig');
}
```

Copier

Étape 7 : Créer le template de base

Créez `templates/base.html.twig` avec une structure de sidebar pour l'interface d'administration, incluant :

- Navigation vers les différentes sections (Catégories, Produits, Commandes, etc.)
- Bouton de déconnexion
- Intégration Bootstrap et Font Awesome

Étape 8 : Migrations supplémentaires

Si vous ajoutez des champs après la création initiale (comme `isVerified` pour User), créez une nouvelle migration :

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

Copier

Étape 9 : Générer des données de test avec Doctrine Fixtures

Installer les packages nécessaires

```
composer require --dev orm-fixtures
composer require --dev fakerphp/faker
```

Copier

Créer les fixtures avec make:fixtures

```
php bin/console make:fixtures CategoryFixtures
php bin/console make:fixtures ProductFixtures
php bin/console make:fixtures UserFixtures
php bin/console make:fixtures OrderFixtures
php bin/console make:fixtures OrderItemFixtures
```

Copier

Structure des fixtures

Les fixtures sont créées dans `src/DataFixtures/` et permettent de générer des données de test pour votre application.

CategoryFixtures : Crée 8 catégories prédéfinies (Électronique, Vêtements, Maison & Jardin, etc.)

ProductFixtures : Génère 50 produits avec Faker

- Nom, description, prix, stock générés aléatoirement
- Images optionnelles (70% de chance)
- Association à 1 catégorie aléatoire
- Dates de création et mise à jour réalistes

UserFixtures : Génère 21 utilisateurs (1 admin + 20 utilisateurs)

- Admin : `admin@example.com` / `admin123` avec `ROLE_ADMIN`
- Utilisateurs : emails, noms, adresses générés avec Faker
- Mots de passe hashés avec `UserPasswordHasherInterface`
- 80% des utilisateurs vérifiés

OrderFixtures : Génère 30 commandes

- Numéros de commande au format `CMD-####-####`
- Statuts aléatoires : pending, processing, shipped, delivered, cancelled
- Dates entre les 6 derniers mois
- Total initialisé à 0 (recalculé dans OrderItemFixtures)

OrderItemFixtures : Génère les articles de commande

- 1 à 5 articles par commande
- Quantité entre 1 et 3 par article
- Prix récupéré du produit
- Recalcule le total de chaque commande

Gérer les dépendances entre fixtures

Utilisez `DependentFixtureInterface` pour définir l'ordre de chargement :

```
use DoctrineCommonDataFixturesDependentFixtureInterface;

class ProductFixtures extends Fixture implements DependentFixtureInterface
{
    public function getDependencies(): array
    {
        return [
            CategoryFixtures::class,
        ];
    }
}
```

Copier

Charger les fixtures

```
# Charger toutes les fixtures (vide la base avant)
php bin/console doctrine:fixtures:load

# Ajouter les fixtures sans vider la base
php bin/console doctrine:fixtures:load --append

# Charger un groupe spécifique
php bin/console doctrine:fixtures:load --group=CategoryFixtures
```

Copier

Concepts importants

`$manager->persist($entity)` : Prépare l'entité pour la sauvegarde (mise en file d'attente)

`$manager->flush()` : Exécute toutes les requêtes SQL en base de données

`$this->addReference('name', $entity)` : Stocke une référence à une entité pour la récupérer dans d'autres fixtures avec `$this->getReference('name')`

Faker : Bibliothèque pour générer des données réalistes (noms, adresses, dates, etc.)

Exemple de fixture complète

```php
<?php

namespace AppDataFixtures;

use AppEntityCategory;
use DoctrineBundleFixturesBundleFixture;
use DoctrinePersistenceObjectManager;

class CategoryFixtures extends Fixture
{
    public function load(ObjectManager $manager): void
    {
        $categories = [
            ['name' => 'Électronique', 'description' => 'Appareils électroniques'],
            ['name' => 'Vêtements', 'description' => 'Mode et habillement'],
        ];

        foreach ($categories as $categoryData) {
            $category = new Category();
            $category->setName($categoryData['name']);
            $category->setDescription($categoryData['description']);
            $category->setSlug(strtolower(str_replace(' ', '-', $categoryData['name'])));
            $category->setCreatedAt(new DateTimeImmutable());
            $category->setUpdatedAt(new DateTimeImmutable());

            $manager->persist($category);
        }

        $manager->flush();
    }
}
```

Copier

Étape 10 : Créer la page d'accueil publique

Créer le controller Accueil

Créez `src/Controller/AccueilController.php` :

```php
<?php

namespace AppController;

use AppRepositoryProductRepository;
use SymfonyBundleFrameworkBundleControllerAbstractController;
use SymfonyComponentHttpFoundationResponse;
use SymfonyComponentRoutingAttributeRoute;

final class AccueilController extends AbstractController
{
    #[Route('/', name: 'app_accueil')]
    public function index(ProductRepository $productRepository): Response
    {
        $products = $productRepository->findAll();

        return $this->render('accueil/index.html.twig', [
            'products' => $products,
        ]);
    }
}
```

Copier

Points importants :

- Route `/` : Page d'accueil accessible à tous (pas de protection)
- Injection de `ProductRepository` : Récupère tous les produits
- Template `accueil/index.html.twig` : Affiche les produits en grille

Créer le template de base pour l'accueil

Créez `templates/base_accueil.html.twig` :

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Accueil{% endblock %}</title>
    {% block stylesheets %}
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
        <link
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css"
rel="stylesheet">
        <style>
            body.page-accueil {
                min-height: 100vh;
                display: flex;
                flex-direction: column;
                margin: 0;
                background-color: #f8f9fa;
            }
            body.page-accueil .navbar {
                background-color: #fff;
                box-shadow: 0 2px 4px rgba(0,0,0,0.1);
            }
            body.page-accueil .product-card {
                transition: transform 0.3s ease, box-shadow 0.3s ease;
                height: 100%;
```

```
            }
            body.page-accueil .product-card:hover {
                transform: translateY(-5px);
                box-shadow: 0 4px 12px rgba(0,0,0,0.15);
            }
            body.page-accueil .product-image {
                height: 200px;
                object-fit: cover;
                width: 100%;
            }
            body.page-accueil main {
                flex: 1;
            }
            body.page-accueil .footer {
                background-color: #343a40;
                color: #fff;
                padding: 40px 0;
                margin-top: auto;
            }
        </style>
    {% endblock %}
</head>
<body class="page-accueil">
    <nav class="navbar navbar-expand-lg navbar-light">
        <div class="container">
            <a class="navbar-brand fw-bold" href="{{ path('app_accueil') }}">E-Commerce</a>
            <div class="collapse navbar-collapse">
                <ul class="navbar-nav ms-auto">
                    <li class="nav-item">
                        <a class="nav-link" href="{{ path('app_accueil') }}">Accueil</a>
                    </li>
                    {% if app.user %}
                        <li class="nav-item">
                            <a class="nav-link" href="{{ path('app_home') }}">Dashboard</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" href="{{ path('app_logout')
}}">Déconnexion</a>
                        </li>
                    {% else %}
                        <li class="nav-item">
                            <a class="nav-link" href="{{ path('app_login') }}">Connexion</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" href="{{ path('app_register')
}}">Inscription</a>
                        </li>
                    {% endif %}
                </ul>
            </div>
        </div>
    </nav>

    <main>
        {% block body %}{% endblock %}
    </main>

    <footer class="footer">
        <div class="container text-center">
            <p class="mb-0">© 2026 E-Commerce. Tous droits réservés.</p>
        </div>
    </footer>
</body>
</html>
```

Copier

Caractéristiques du template :

- **Classe `page-accueil`** : Scopage des styles CSS pour éviter les conflits avec `base.html.twig`
- **Flexbox layout** : Footer toujours en bas de page même si le contenu est court
- **Navigation conditionnelle** : Affiche Dashboard/Déconnexion si connecté, Connexion/Inscription sinon
- **Responsive** : Bootstrap pour l'adaptation mobile

Créer la vue accueil

Créez `templates/accueil/index.html.twig` :

```
{% extends 'base_accueil.html.twig' %}

{% block title %}Accueil - Produits{% endblock %}

{% block body %}
<div class="container my-5">
    <div class="row mb-4">
        <div class="col-12">
            <h1 class="display-4 text-center mb-4">Nos Produits</h1>
            <p class="text-center text-muted">Découvrez notre sélection de produits</p>
        </div>
    </div>

    <div class="row g-4">
        {% for product in products %}
            <div class="col-md-4 col-lg-3">
                <div class="card product-card">
                    {% if product.image %}
                        <img src="{{ product.image }}" class="card-img-top product-image"
alt="{{ product.name }}">
                    {% else %}
                        <div class="card-img-top product-image bg-secondary d-flex align-
items-center justify-content-center">
                            <i class="fas fa-image fa-3x text-white"></i>
                        </div>
                    {% endif %}
                    <div class="card-body d-flex flex-column">
                        <h5 class="card-title">{{ product.name }}</h5>
                        <p class="card-text text-muted flex-grow-1">
                            {% if product.description %}
                                {{ product.description|length > 100 ?
product.description|slice(0, 100) ~ '...' : product.description }}
                            {% else %}
                                Aucune description disponible
                            {% endif %}
                        </p>
                        <div class="mt-auto">
                            <div class="d-flex justify-content-between align-items-center
mb-3">
                                <span class="h5 text-primary mb-0">{{ product.price }}
€</span>
                                {% if product.stock > 0 %}
                                    <span class="badge bg-success">En stock ({{
product.stock }})</span>
                                {% else %}
                                    <span class="badge bg-danger">Rupture de stock</span>
                                {% endif %}
```

```
                            </div>
                            <div class="d-grid gap-2">
                                <a href="{{ path('app_product_show', {'id': product.id}) }}"
class="btn btn-primary">
                                    <i class="fas fa-eye me-2"></i>Voir les détails
                                </a>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        {% else %}
            <div class="col-12">
                <div class="alert alert-info text-center">
                    <i class="fas fa-info-circle me-2"></i>
                    Aucun produit disponible pour le moment.
                </div>
            </div>
        {% endfor %}
    </div>
</div>
{% endblock %}
```

Copier

Fonctionnalités de la vue :

- **Grille de produits** : Affichage en cartes Bootstrap (4 colonnes sur desktop, responsive)
- **Images produits** : Affiche l'image si disponible, sinon icône placeholder
- **Description tronquée** : Limite à 100 caractères avec "..."
- **Badge de stock** : Vert si en stock, rouge si rupture
- **Lien vers détails** : Bouton vers la page de détail du produit
- **Message si vide** : Affiche un message si aucun produit

Différences entre base.html.twig et base_accueil.html.twig

`base.html.twig` (Interface d'administration) :

- Sidebar fixe à gauche
- Navigation verticale
- Protégé par `ROLE_USER`
- Utilisé pour les pages CRUD

`base_accueil.html.twig` (Page publique) :

- Navigation horizontale en haut
- Footer en bas
- Accessible à tous (pas de protection)
- Utilisé pour la page d'accueil publique

Points importants

- **Scopage CSS** : Utilisez `body.page-accueil` pour éviter les conflits de styles
- **Flexbox pour footer** : `display: flex` + `flex-direction: column` + `margin-top: auto` sur le footer

- **Responsive** : Utilisez les classes Bootstrap `col-md-4 col-lg-3` pour l'adaptation mobile
- **Performance** : Considérez la pagination si vous avez beaucoup de produits

---

□ Étape 11 : Créer la page produit détaillée

Générer le contrôleur

```
php bin/console make:controller ProductShowController
```

Copier

Modifiez `src/Controller/ProductShowController.php` pour afficher les détails d'un produit. Ce contrôleur est **public** (pas de protection admin) :

```
<?php

namespace App\Controller;

use App\Entity\Product;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

final class ProductShowController extends AbstractController
{
    #[Route('/produit/{slug}', name: 'app_product_detail')]
    public function show(Product $product): Response
    {
        return $this->render('product/show.html.twig', [
            'product' => $product,
        ]);
    }
}
```

Copier

Points importants :

- **Route** `/produit/{slug}` : Utilise le slug du produit pour une URL SEO-friendly
- **ParamConverter automatique** : Symfony convertit automatiquement le `{slug}` en objet `Product` grâce à Doctrine
- **Pas de protection** : La page produit est accessible à tous les visiteurs

Créer le template product/show.html.twig

Créez `templates/product/show.html.twig` :

```
{% extends 'base_accueil.html.twig' %}

{% block title %}{{ product.name }} - E-Commerce{% endblock %}

{% block body %}
<div class="container my-5">
```

```
    <nav aria-label="breadcrumb">
        <ol class="breadcrumb">
            <li class="breadcrumb-item"><a href="{{ path('app_accueil') }}">Accueil</a></li>
            <li class="breadcrumb-item active">{{ product.name }}</li>
        </ol>
    </nav>

    <div class="row">
        <div class="col-md-6">
            {% if product.image %}
                <img src="{{ product.image }}" class="img-fluid rounded shadow" alt="{{
product.name }}">
            {% else %}
                <div class="bg-light d-flex align-items-center justify-content-center
rounded shadow" style="height: 400px;">
                    <i class="fas fa-image fa-5x text-secondary"></i>
                </div>
            {% endif %}
        </div>
        <div class="col-md-6">
            <h1 class="mb-3">{{ product.name }}</h1>

            <div class="mb-3">
                {% for category in product.categories %}
                    <span class="badge bg-info me-1">{{ category.name }}</span>
                {% endfor %}
            </div>

            <p class="text-muted fs-5">{{ product.description }}</p>

            <h2 class="text-primary mb-3">{{ product.price|number_format(2, ',', ' ') }}
€</h2>

            {% if product.stock > 0 %}
                <span class="badge bg-success fs-6 mb-3">En stock ({{ product.stock }}
disponibles)</span>

                <form action="{{ path('app_cart_add', {id: product.id}) }}" method="POST"
class="mt-3">
                    <div class="input-group mb-3" style="max-width: 200px;">
                        <label class="input-group-text" for="quantity">Qté</label>
                        <input type="number" class="form-control" id="quantity"
name="quantity"
                               value="1" min="1" max="{{ product.stock }}">
                    </div>
                    <button type="submit" class="btn btn-primary btn-lg">
                        <i class="fas fa-cart-plus me-2"></i>Ajouter au panier
                    </button>
                </form>
            {% else %}
                <span class="badge bg-danger fs-6 mb-3">Rupture de stock</span>
            {% endif %}

            <hr>
            <a href="{{ path('app_accueil') }}" class="btn btn-outline-secondary">
                <i class="fas fa-arrow-left me-2"></i>Retour aux produits
            </a>
        </div>
    </div>
</div>
{% endblock %}
```

Copier

Fonctionnalités de la page produit :

- **Fil d'Ariane (breadcrumb)** : Navigation Accueil > Produit
- **Image ou placeholder** : Affiche l'image du produit ou une icône par défaut
- **Catégories** : Badges pour chaque catégorie du produit (relation ManyToMany)
- **Prix formaté** : Affichage avec 2 décimales et séparateur
- **Gestion du stock** : Formulaire d'ajout au panier si en stock, badge "Rupture" sinon
- **Sélecteur de quantité** : Champ numérique limité au stock disponible

Mettre à jour le lien sur la page d'accueil

Dans `templates/accueil/index.html.twig`, modifiez le bouton "Voir les détails" pour pointer vers la nouvelle route basée sur le slug :

```
<a href="{{ path('app_product_detail', {'slug': product.slug}) }}" class="btn btn-primary">
    <i class="fas fa-eye me-2"></i>Voir les détails
</a>
```

Copier

□ Étape 12 : Système de panier et page panier

Concept du panier par session

Le panier est stocké dans la **session PHP** de l'utilisateur. Pas besoin de base de données pour le panier temporaire. On utilise directement `$request->getSession()` dans les contrôleurs, sans créer de service supplémentaire.

La structure en session est un tableau associatif simple :

```
// Structure de la session 'cart'
[
    product_id => quantity,
    // ex: 5 => 2   (2 exemplaires du produit #5)
    // ex: 12 => 1  (1 exemplaire du produit #12)
]
```

Copier

Avantages :

- Pas de requête SQL pour gérer le panier
- Le panier persiste tant que la session est active
- Fonctionne même pour les utilisateurs non connectés
- Utilise uniquement les composants natifs de Symfony (`Request`, `Session`)

Générer le CartController

```
php bin/console make:controller CartController
```

Copier

Modifiez `src/Controller/CartController.php` pour gérer toutes les opérations du panier :

```
<?php

namespace App\Controller;

use App\Repository\ProductRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

#[Route('/panier')]
final class CartController extends AbstractController
{
    #[Route('', name: 'app_cart', methods: ['GET'])]
    public function index(Request $request, ProductRepository $productRepository): Response
    {
        $cart = $request->getSession()->get('cart', []);
        $cartData = [];
        $total = 0;

        foreach ($cart as $productId => $quantity) {
            $product = $productRepository->find($productId);
            if ($product) {
                $subtotal = $product->getPrice() * $quantity;
                $cartData[] = [
                    'product' => $product,
                    'quantity' => $quantity,
                    'subtotal' => $subtotal,
                ];
                $total += $subtotal;
            }
        }

        return $this->render('cart/index.html.twig', [
            'cart' => $cartData,
            'total' => $total,
        ]);
    }

    #[Route('/ajouter/{id}', name: 'app_cart_add', methods: ['POST'])]
    public function add(int $id, Request $request): Response
    {
        $session = $request->getSession();
        $cart = $session->get('cart', []);
        $quantity = $request->request->getInt('quantity', 1);

        if (isset($cart[$id])) {
            $cart[$id] += $quantity;
        } else {
            $cart[$id] = $quantity;
        }

        $session->set('cart', $cart);
        $this->addFlash('success', 'Produit ajouté au panier !');

        return $this->redirectToRoute('app_cart');
    }
```

```
    #[Route('/supprimer/{id}', name: 'app_cart_remove', methods: ['POST'])]
    public function remove(int $id, Request $request): Response
    {
        $session = $request->getSession();
        $cart = $session->get('cart', []);
        unset($cart[$id]);
        $session->set('cart', $cart);

        $this->addFlash('success', 'Produit retiré du panier.');

        return $this->redirectToRoute('app_cart');
    }

    #[Route('/modifier/{id}', name: 'app_cart_update', methods: ['POST'])]
    public function update(int $id, Request $request): Response
    {
        $session = $request->getSession();
        $cart = $session->get('cart', []);
        $quantity = $request->request->getInt('quantity', 1);

        if ($quantity <= 0) {
            unset($cart[$id]);
        } else {
            $cart[$id] = $quantity;
        }

        $session->set('cart', $cart);
        $this->addFlash('success', 'Panier mis à jour.');

        return $this->redirectToRoute('app_cart');
    }

    #[Route('/vider', name: 'app_cart_clear', methods: ['POST'])]
    public function clear(Request $request): Response
    {
        $request->getSession()->remove('cart');

        $this->addFlash('success', 'Panier vidé.');

        return $this->redirectToRoute('app_cart');
    }
}
```

Copier

Explication des méthodes :

Méthode	Rôle
`index()`	Lit la session, récupère les objets `Product` via le `ProductRepository`, calcule les sous-totaux et le total
`add()`	Ajoute un produit à la session ou incrémente la quantité s'il existe déjà
`remove()`	Supprime un produit de la session avec `unset()`
`update()`	Met à jour la quantité (supprime si ≤ 0)
`clear()`	Vide le panier avec `$session->remove('cart')`

Points importants :

- **`$request->getSession()`** : Accès natif à la session Symfony, pas besoin de service

custom

- **`ProductRepository`** : Injecté par Symfony grâce à l'autowiring pour récupérer les objets `Product`
- **Préfixe `/panier`** : L'attribut `#[Route('/panier')]` au niveau de la classe préfixe toutes les routes
- **Méthode POST** : Toutes les actions de modification utilisent POST (sécurité contre les modifications accidentelles via URL)
- **Flash messages** : `$this->addFlash()` stocke un message en session, affiché une seule fois au prochain chargement de page
- **Redirection PRG** : Pattern Post/Redirect/Get pour éviter la re-soumission du formulaire au rafraîchissement

Créer le template cart/index.html.twig

Créez `templates/cart/index.html.twig` :

```
{% extends 'base_accueil.html.twig' %}

{% block title %}Mon Panier - E-Commerce{% endblock %}

{% block body %}
<div class="container my-5">
    <h1 class="mb-4"><i class="fas fa-shopping-cart me-2"></i>Mon Panier</h1>

    {% for message in app.flashes('success') %}
        <div class="alert alert-success alert-dismissible fade show">
            {{ message }}
            <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
        </div>
    {% endfor %}

    {% if cart|length > 0 %}
        <div class="table-responsive">
            <table class="table table-hover align-middle">
                <thead class="table-dark">
                    <tr>
                        <th>Produit</th>
                        <th>Prix unitaire</th>
                        <th style="width: 180px;">Quantité</th>
                        <th>Sous-total</th>
                        <th>Actions</th>
                    </tr>
                </thead>
                <tbody>
                    {% for item in cart %}
                        <tr>
                            <td>
                                <div class="d-flex align-items-center">
                                    {% if item.product.image %}
                                        <img src="{{ item.product.image }}" alt="{{
item.product.name }}"
                                             class="rounded me-3" style="width: 60px;
height: 60px; object-fit: cover;">
                                    {% else %}
                                        <div class="bg-light rounded me-3 d-flex align-
items-center justify-content-center"
                                             style="width: 60px; height: 60px;">
                                            <i class="fas fa-image text-secondary"></i>
                                        </div>
                                    {% endif %}
```

```
                                    <div>
                                        <h6 class="mb-0">{{ item.product.name }}</h6>
                                    </div>
                                </div>
                            </td>
                            <td>{{ item.product.price|number_format(2, ',', ' ') }} €</td>
                            <td>
                                <form action="{{ path('app_cart_update', {id:
item.product.id}) }}" method="POST"
                                      class="d-flex align-items-center">
                                    <input type="number" name="quantity" value="{{
item.quantity }}"
                                           min="1" max="{{ item.product.stock }}"
                                           class="form-control form-control-sm"
style="width: 70px;">
                                    <button type="submit" class="btn btn-sm btn-outline-
primary ms-2"
                                            title="Mettre à jour">
                                        <i class="fas fa-sync-alt"></i>
                                    </button>
                                </form>
                            </td>
                            <td class="fw-bold">{{ item.subtotal|number_format(2, ',', ' ')
}} €</td>
                            <td>
                                <form action="{{ path('app_cart_remove', {id:
item.product.id}) }}" method="POST"
                                      class="d-inline">
                                    <button type="submit" class="btn btn-sm btn-outline-
danger" title="Supprimer">
                                        <i class="fas fa-trash"></i>
                                    </button>
                                </form>
                            </td>
                        </tr>
                    {% endfor %}
                </tbody>
                <tfoot>
                    <tr class="table-light">
                        <td colspan="3" class="text-end fw-bold fs-5">Total :</td>
                        <td class="fw-bold fs-5 text-primary">{{ total|number_format(2, ',',
' ') }} €</td>
                        <td></td>
                    </tr>
                </tfoot>
            </table>
        </div>

        <div class="d-flex justify-content-between mt-4">
            <form action="{{ path('app_cart_clear') }}" method="POST">
                <button type="submit" class="btn btn-outline-danger"
                        onclick="return confirm('Êtes-vous sûr de vouloir vider le panier
?')">
                    <i class="fas fa-trash-alt me-2"></i>Vider le panier
                </button>
            </form>
            <div>
                <a href="{{ path('app_accueil') }}" class="btn btn-outline-secondary me-2">
                    <i class="fas fa-arrow-left me-2"></i>Continuer les achats
                </a>
                <a href="{{ path('app_checkout') }}" class="btn btn-success btn-lg">
                    <i class="fas fa-credit-card me-2"></i>Passer à la caisse
                </a>
            </div>
        </div>
```

```
    {% else %}
        <div class="text-center py-5">
            <i class="fas fa-shopping-cart fa-4x text-muted mb-3 d-block"></i>
            <h3 class="text-muted">Votre panier est vide</h3>
            <p class="text-muted mb-4">Découvrez nos produits et ajoutez-les à votre
panier.</p>
            <a href="{{ path('app_accueil') }}" class="btn btn-primary btn-lg">
                <i class="fas fa-shopping-bag me-2"></i>Voir les produits
            </a>
        </div>
    {% endif %}
</div>
{% endblock %}
```

Copier

Fonctionnalités de la page panier :

- **Tableau produits** : Image, nom, prix unitaire, quantité modifiable, sous-total
- **Modification en ligne** : Champ de quantité avec bouton de mise à jour
- **Suppression** : Bouton poubelle pour retirer un produit
- **Total calculé** : Affiché dans le pied de tableau
- **Actions** : Vider le panier (avec confirmation), continuer les achats, passer à la caisse
- **État vide** : Message et lien vers les produits si le panier est vide
- **Flash messages** : Affichage des messages de confirmation

Ajouter le lien panier dans la navigation

Dans `templates/base_accueil.html.twig`, ajoutez un lien vers le panier dans la navbar, avant les liens de connexion :

```
<li class="nav-item">
    <a class="nav-link" href="{{ path('app_cart') }}">
        <i class="fas fa-shopping-cart"></i> Panier
    </a>
</li>
```

Copier

□ Étape 13 : Page de validation de commande

Générer le CheckoutController

```
php bin/console make:controller CheckoutController
```

Copier

Modifiez `src/Controller/CheckoutController.php` :

```php
<?php

namespace App\Controller;

use App\Repository\ProductRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;
use Symfony\Component\Security\Http\Attribute\IsGranted;

#[IsGranted('ROLE_USER')]
final class CheckoutController extends AbstractController
{
    #[Route('/commande', name: 'app_checkout')]
    public function index(Request $request, ProductRepository $productRepository): Response
    {
        $cart = $request->getSession()->get('cart', []);

        if (empty($cart)) {
            $this->addFlash('warning', 'Votre panier est vide.');
            return $this->redirectToRoute('app_cart');
        }

        $cartData = [];
        $total = 0;

        foreach ($cart as $productId => $quantity) {
            $product = $productRepository->find($productId);
            if ($product) {
                $subtotal = $product->getPrice() * $quantity;
                $cartData[] = [
                    'product' => $product,
                    'quantity' => $quantity,
                    'subtotal' => $subtotal,
                ];
                $total += $subtotal;
            }
        }

        return $this->render('checkout/index.html.twig', [
            'cart' => $cartData,
            'total' => $total,
            'user' => $this->getUser(),
        ]);
    }
}
```

Copier

Points importants :

- **`#[IsGranted('ROLE_USER')]`** : L'utilisateur doit être connecté pour accéder à la page de validation
- **Lecture de la session** : On lit directement `$request->getSession()->get('cart', [])`, même logique que dans le `CartController`
- **Vérification panier vide** : Redirige vers le panier avec un message d'avertissement si le panier est vide
- **Données utilisateur** : Les informations du profil (adresse, ville, etc.) sont envoyées au template pour pré-remplir le formulaire

Créer le template checkout/index.html.twig

Créez `templates/checkout/index.html.twig` :

```
{% extends 'base_accueil.html.twig' %}

{% block title %}Validation de commande - E-Commerce{% endblock %}

{% block body %}
<div class="container my-5">
    <h1 class="mb-4"><i class="fas fa-clipboard-check me-2"></i>Validation de commande</h1>

    <div class="row">
        <div class="col-md-7">
            <div class="card shadow-sm mb-4">
                <div class="card-header bg-dark text-white">
                    <h5 class="mb-0"><i class="fas fa-truck me-2"></i>Informations de
livraison</h5>
                </div>
                <div class="card-body">
                    <div class="row g-3">
                        <div class="col-md-6">
                            <label class="form-label fw-bold">Prénom</label>
                            <input type="text" class="form-control" value="{{ user.firstName
}}" readonly>
                        </div>
                        <div class="col-md-6">
                            <label class="form-label fw-bold">Nom</label>
                            <input type="text" class="form-control" value="{{ user.lastName
}}" readonly>
                        </div>
                        <div class="col-12">
                            <label class="form-label fw-bold">Email</label>
                            <input type="email" class="form-control" value="{{ user.email
}}" readonly>
                        </div>
                        <div class="col-12">
                            <label class="form-label fw-bold">Adresse</label>
                            <input type="text" class="form-control"
                                   value="{{ user.address is defined and user.address ?
user.address : '' }}" readonly>
                        </div>
                        <div class="col-md-4">
                            <label class="form-label fw-bold">Code postal</label>
                            <input type="text" class="form-control"
                                   value="{{ user.postalCode is defined and user.postalCode
? user.postalCode : '' }}" readonly>
                        </div>
                        <div class="col-md-4">
                            <label class="form-label fw-bold">Ville</label>
                            <input type="text" class="form-control"
                                   value="{{ user.city is defined and user.city ? user.city
: '' }}" readonly>
                        </div>
                        <div class="col-md-4">
                            <label class="form-label fw-bold">Téléphone</label>
                            <input type="text" class="form-control"
                                   value="{{ user.phone is defined and user.phone ?
user.phone : '' }}" readonly>
                        </div>
                    </div>
                </div>
            </div>
        </div>
```

```
        <div class="col-md-5">
            <div class="card shadow-sm">
                <div class="card-header bg-dark text-white">
                    <h5 class="mb-0"><i class="fas fa-receipt me-2"></i>Récapitulatif de la
commande</h5>
                </div>
                <div class="card-body">
                    {% for item in cart %}
                        <div class="d-flex justify-content-between align-items-center mb-2">
                            <div>
                                <strong>{{ item.product.name }}</strong>
                                <br><small class="text-muted">{{
item.product.price|number_format(2, ',', ' ') }} € × {{ item.quantity }}</small>
                            </div>
                            <span class="fw-bold">{{ item.subtotal|number_format(2, ',', '
') }} €</span>
                        </div>
                        {% if not loop.last %}<hr class="my-2">{% endif %}
                    {% endfor %}

                    <hr>
                    <div class="d-flex justify-content-between align-items-center">
                        <strong class="fs-5">Total à payer</strong>
                        <strong class="fs-4 text-primary">{{ total|number_format(2, ',', '
') }} €</strong>
                    </div>
                </div>
                <div class="card-footer">
                    <form action="{{ path('app_payment_checkout') }}" method="POST">
                        <button type="submit" class="btn btn-success btn-lg w-100">
                            <i class="fas fa-lock me-2"></i>Payer {{ total|number_format(2,
',', ' ') }} € par carte
                        </button>
                    </form>
                    <p class="text-center text-muted mt-2 mb-1">
                        <i class="fas fa-shield-alt me-1"></i>Paiement sécurisé par Stripe
                    </p>
                    <a href="{{ path('app_cart') }}" class="btn btn-outline-secondary w-100
mt-2">
                        <i class="fas fa-arrow-left me-2"></i>Modifier le panier
                    </a>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

Copier

Fonctionnalités de la page de validation :

- **Colonne gauche** : Informations de livraison pré-remplies depuis le profil utilisateur (en lecture seule)
- **Colonne droite** : Récapitulatif de commande avec détail des articles, prix × quantité, et total
- **Bouton de paiement** : Lance le processus de paiement Stripe Checkout
- **Mention sécurité** : Rassure l'utilisateur sur la sécurité du paiement
- **Retour panier** : Lien pour modifier le panier avant de payer

🔟 Étape 14 : Intégration Stripe pour le paiement par carte bancaire

Créer un compte Stripe de test

1. Rendez-vous sur https://dashboard.stripe.com/register
2. Créez un compte (gratuit, aucune carte bancaire requise)
3. Une fois connecté, activez le **mode test** avec le toggle en haut à droite du dashboard
4. Allez dans **Développeurs > Clés API** pour récupérer :
    - **Clé publique** (publishable key) : commence par `pk_test_...`
    - **Clé secrète** (secret key) : commence par `sk_test_...`

⚠ **Ne partagez jamais votre clé secrète** et ne la committez pas dans Git.

Installer le SDK Stripe

```
composer require stripe/stripe-php
```

Copier

Configurer les clés API

Ajoutez les variables d'environnement dans `.env` (et `.env.local` pour les vraies valeurs) :

```
# .env (valeurs par défaut / documentation)
STRIPE_SECRET_KEY=sk_test_VOTRE_CLE_SECRETE
STRIPE_PUBLIC_KEY=pk_test_VOTRE_CLE_PUBLIQUE
STRIPE_WEBHOOK_SECRET=whsec_VOTRE_SECRET_WEBHOOK
```

Copier

Puis référencez-les dans `config/services.yaml` :

```
# config/services.yaml
parameters:
    app.stripe_secret_key: '%env(STRIPE_SECRET_KEY)%'
    app.stripe_public_key: '%env(STRIPE_PUBLIC_KEY)%'
    app.stripe_webhook_secret: '%env(STRIPE_WEBHOOK_SECRET)%'
```

Copier

Générer le PaymentController

```
php bin/console make:controller PaymentController
```

Copier

Modifiez `src/Controller/PaymentController.php` :

```php
<?php

namespace App\Controller;

use App\Entity\Order;
use App\Entity\OrderItem;
use App\Repository\ProductRepository;
use Doctrine\ORM\EntityManagerInterface;
use Stripe\Checkout\Session as StripeSession;
use Stripe\Stripe;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;
use Symfony\Component\Security\Http\Attribute\IsGranted;

#[IsGranted('ROLE_USER')]
final class PaymentController extends AbstractController
{
    #[Route('/paiement/checkout', name: 'app_payment_checkout', methods: ['POST'])]
    public function checkout(Request $request, ProductRepository $productRepository):
Response
    {
        $cart = $request->getSession()->get('cart', []);

        if (empty($cart)) {
            $this->addFlash('warning', 'Votre panier est vide.');
            return $this->redirectToRoute('app_cart');
        }

        Stripe::setApiKey($this->getParameter('app.stripe_secret_key'));

        $lineItems = [];
        foreach ($cart as $productId => $quantity) {
            $product = $productRepository->find($productId);
            if ($product) {
                $lineItems[] = [
                    'price_data' => [
                        'currency' => 'eur',
                        'product_data' => [
                            'name' => $product->getName(),
                        ],
                        'unit_amount' => (int) ($product->getPrice() * 100),
                    ],
                    'quantity' => $quantity,
                ];
            }
        }

        $stripeSession = StripeSession::create([
            'payment_method_types' => ['card'],
            'line_items' => $lineItems,
            'mode' => 'payment',
            'success_url' => $this->generateUrl(
                'app_payment_success',
                [],
                UrlGeneratorInterface::ABSOLUTE_URL
            ) . '?session_id={CHECKOUT_SESSION_ID}',
            'cancel_url' => $this->generateUrl(
                'app_payment_cancel',
                [],
                UrlGeneratorInterface::ABSOLUTE_URL
            ),
            'customer_email' => $this->getUser()->getEmail(),
```

```
        ]);

        return $this->redirect($stripeSession->url);
    }

    #[Route('/paiement/succes', name: 'app_payment_success')]
    public function success(Request $request, ProductRepository $productRepository,
EntityManagerInterface $em): Response
    {
        $session = $request->getSession();
        $cart = $session->get('cart', []);

        if (!empty($cart)) {
            $order = new Order();
            $order->setUser($this->getUser());
            $order->setOrderNumber('CMD-' . strtoupper(substr(uniqid(), -8)));
            $order->setStatus('paid');
            $order->setCreatedAt(new \DateTimeImmutable());
            $order->setUpdatedAt(new \DateTimeImmutable());

            $total = 0;
            foreach ($cart as $productId => $quantity) {
                $product = $productRepository->find($productId);
                if ($product) {
                    $orderItem = new OrderItem();
                    $orderItem->setOrderId($order);
                    $orderItem->setProduct($product);
                    $orderItem->setQuantity($quantity);
                    $orderItem->setPrice($product->getPrice());
                    $em->persist($orderItem);
                    $total += $product->getPrice() * $quantity;
                }
            }

            $order->setTotal($total);
            $em->persist($order);
            $em->flush();

            $session->remove('cart');
        }

        return $this->render('payment/success.html.twig');
    }

    #[Route('/paiement/annulation', name: 'app_payment_cancel')]
    public function cancel(): Response
    {
        return $this->render('payment/cancel.html.twig');
    }
}
```

Copier

Explication du flux de paiement :

1. `checkout()` : Crée une **Stripe Checkout Session** avec les articles du panier
    - Lit la session pour récupérer le panier `[productId => quantity]`
    - Récupère chaque `Product` via le `ProductRepository`
    - `unit_amount` : Stripe attend le montant en **centimes** (19,99 € → 1999)
    - `{CHECKOUT_SESSION_ID}` : Placeholder remplacé automatiquement par Stripe dans l'URL de succès

- `customer_email` : Pré-remplit l'email dans le formulaire Stripe
- Redirige le navigateur vers la page de paiement hébergée par Stripe

2. **`success()`** : Appelé après un paiement réussi
   - Lit la session pour récupérer le panier
   - Crée l'entité `Order` avec le statut `paid`
   - Crée les `OrderItem` pour chaque produit du panier
   - Vide le panier avec `$session->remove('cart')`

3. **`cancel()`** : Appelé si l'utilisateur annule le paiement
   - Affiche un message, le panier reste intact en session

Créer les templates de paiement

`templates/payment/success.html.twig` :

```
{% extends 'base_accueil.html.twig' %}

{% block title %}Paiement réussi - E-Commerce{% endblock %}

{% block body %}
<div class="container my-5 text-center">
    <div class="py-5">
        <i class="fas fa-check-circle fa-5x text-success mb-4 d-block"></i>
        <h1 class="text-success mb-3">Paiement réussi !</h1>
        <p class="fs-5 text-muted">Merci pour votre commande. Vous recevrez un email de
confirmation sous peu.</p>
        <div class="mt-4">
            <a href="{{ path('app_accueil') }}" class="btn btn-primary btn-lg me-2">
                <i class="fas fa-home me-2"></i>Retour à l'accueil
            </a>
            <a href="{{ path('app_home') }}" class="btn btn-outline-secondary btn-lg">
                <i class="fas fa-list me-2"></i>Mes commandes
            </a>
        </div>
    </div>
</div>
{% endblock %}
```

Copier

`templates/payment/cancel.html.twig` :

```
{% extends 'base_accueil.html.twig' %}

{% block title %}Paiement annulé - E-Commerce{% endblock %}

{% block body %}
<div class="container my-5 text-center">
    <div class="py-5">
        <i class="fas fa-times-circle fa-5x text-warning mb-4 d-block"></i>
        <h1 class="text-warning mb-3">Paiement annulé</h1>
        <p class="fs-5 text-muted">Le paiement a été annulé. Votre panier est toujours
disponible.</p>
        <div class="mt-4">
            <a href="{{ path('app_cart') }}" class="btn btn-primary btn-lg me-2">
                <i class="fas fa-shopping-cart me-2"></i>Retour au panier
            </a>
            <a href="{{ path('app_accueil') }}" class="btn btn-outline-secondary btn-lg">
                <i class="fas fa-home me-2"></i>Retour à l'accueil
            </a>
        </div>
    </div>
</div>
{% endblock %}
```

Copier

Générer le contrôleur Webhook Stripe

Les **webhooks** permettent à Stripe de notifier votre serveur quand un événement se produit (paiement réussi, échoué, remboursement, etc.).

```
php bin/console make:controller StripeWebhookController
```

Copier

Modifiez `src/Controller/StripeWebhookController.php` :

```
<?php

namespace App\Controller;

use App\Repository\OrderRepository;
use Doctrine\ORM\EntityManagerInterface;
use Stripe\Event;
use Stripe\Stripe;
use Stripe\Webhook;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

final class StripeWebhookController extends AbstractController
{
    #[Route('/webhook/stripe', name: 'app_stripe_webhook', methods: ['POST'])]
    public function handleWebhook(
        Request $request,
        OrderRepository $orderRepository,
        EntityManagerInterface $em
    ): JsonResponse {
        Stripe::setApiKey($this->getParameter('app.stripe_secret_key'));
```

```
        $payload = $request->getContent();
        $sigHeader = $request->headers->get('Stripe-Signature');
        $webhookSecret = $this->getParameter('app.stripe_webhook_secret');

        try {
            $event = Webhook::constructEvent($payload, $sigHeader, $webhookSecret);
        } catch (\UnexpectedValueException $e) {
            return new JsonResponse(
                ['error' => 'Payload invalide'],
                Response::HTTP_BAD_REQUEST
            );
        } catch (\Stripe\Exception\SignatureVerificationException $e) {
            return new JsonResponse(
                ['error' => 'Signature invalide'],
                Response::HTTP_BAD_REQUEST
            );
        }

        switch ($event->type) {
            case 'checkout.session.completed':
                $session = $event->data->object;
                // Le paiement a été accepté
                // $session->customer_email contient l'email du client
                // $session->amount_total contient le montant total en centimes
                // $session->payment_status contient le statut ('paid')
                break;

            case 'payment_intent.succeeded':
                $paymentIntent = $event->data->object;
                // Le paiement a été effectivement encaissé
                break;

            case 'payment_intent.payment_failed':
                $paymentIntent = $event->data->object;
                // Le paiement a échoué
                // $paymentIntent->last_payment_error->message contient le message d'erreur
                break;
        }

        return new JsonResponse(['status' => 'success']);
    }
}
```

Copier

Points importants sur les webhooks :

- **Vérification de signature** : `Webhook::constructEvent()` vérifie que la requête vient réellement de Stripe (protection contre les requêtes falsifiées)
- **Pas d'authentification** : L'endpoint webhook ne doit pas être protégé par le firewall Symfony
- **Réponse JSON** : Stripe attend une réponse HTTP 200 pour confirmer la réception
- **Idempotence** : Les webhooks peuvent être envoyés plusieurs fois, votre code doit le gérer

Désactiver le firewall pour le webhook

Dans `config/packages/security.yaml`, ajoutez un firewall spécifique pour le webhook **avant** le firewall `main` :

```
# config/packages/security.yaml
security:
    firewalls:
        webhook:
            pattern: ^/webhook
            security: false
        dev:
            pattern: ^/(_(profiler|wdt)|assets|build)/
            security: false
        main:
            lazy: true
            # ... reste de la configuration
```

Copier

Important : Le firewall `webhook` doit être déclaré **avant** `main` car Symfony les évalue dans l'ordre.

Installer Stripe CLI

Stripe CLI est un outil en ligne de commande pour tester les webhooks en local.

```
# macOS (Homebrew)
brew install stripe/stripe-cli/stripe

# Linux (téléchargement direct)
curl -s https://packages.stripe.dev/api/security/keypair/stripe-cli-gpg/public | gpg --
dearmor | sudo tee /usr/share/keyrings/stripe.gpg
echo "deb [signed-by=/usr/share/keyrings/stripe.gpg]
https://packages.stripe.dev/stripe-cli-debian-local stable main" | sudo tee
/etc/apt/sources.list.d/stripe.list
sudo apt update && sudo apt install stripe

# Windows (Scoop)
scoop install stripe
```

Copier

Se connecter à Stripe CLI

```
stripe login
```

Copier

Cette commande ouvre une page dans le navigateur pour autoriser l'accès à votre compte Stripe.

Transférer les webhooks vers votre serveur local (forward-to)

C'est la commande clé pour tester les webhooks en développement :

```
stripe listen --forward-to http://localhost:8000/webhook/stripe
```

Copier

Ce que fait cette commande :

1. Se connecte au serveur Stripe en temps réel
2. Intercepte **tous les événements** Stripe (paiements, remboursements, etc.)
3. Les redirige vers votre endpoint local `http://localhost:8000/webhook/stripe`
4. Affiche en temps réel chaque événement reçu et la réponse de votre serveur

Au lancement, la commande affiche un webhook signing secret :

```
> Ready! Your webhook signing secret is whsec_xxxxxxxxxxxxxxxxxxxxxxxx
```

Copier

⚠ **Copiez ce `whsec_...`** et mettez-le dans votre fichier `.env.local` :

```
STRIPE_WEBHOOK_SECRET=whsec_xxxxxxxxxxxxxxxxxxxxxxxx
```

Copier

Déclencher des événements de test

Dans un **second terminal**, vous pouvez simuler des événements Stripe :

```
# Déclencher un checkout complet (le plus utile pour tester)
stripe trigger checkout.session.completed

# Déclencher un paiement réussi
stripe trigger payment_intent.succeeded

# Déclencher un échec de paiement
stripe trigger payment_intent.payment_failed

# Déclencher un remboursement
stripe trigger charge.refunded

# Lister tous les événements disponibles
stripe trigger --list
```

Copier

Flux de test complet

Ouvrez **3 terminaux** pour tester l'intégralité du flux de paiement :

Terminal 1 — Lancez le serveur Symfony :

```
symfony serve
```

Copier

Terminal 2 — Lancez le forwarding Stripe CLI :

```
stripe listen --forward-to http://localhost:8000/webhook/stripe
```

Copier

Terminal 3 — (Optionnel) Déclenchez des événements manuellement :

```
stripe trigger checkout.session.completed
```

Copier

Dans le navigateur :

1. Allez sur `http://localhost:8000`
2. Connectez-vous avec un compte utilisateur
3. Ajoutez des produits au panier
4. Allez sur la page panier, cliquez "Passer à la caisse"
5. Sur la page de validation, cliquez "Payer par carte"
6. Vous êtes redirigé vers **Stripe Checkout** (page hébergée par Stripe)
7. Utilisez une **carte de test** (voir ci-dessous)
8. Après le paiement, vous êtes redirigé vers la page de succès
9. Dans le Terminal 2, vous voyez les événements reçus en temps réel

Cartes de test Stripe

Numéro de carte	Comportement