Cahier des Charges - Projet MossAir (Pages 9-39)

Cahier des Charges - Projet Moss (Pages 9-39)

Partie Frontend && Partie Backend

9. CSS et Media Queries : Design Responsive

Pour garantir que le site soit **responsive** et s’adapte à tous les types d’appareils, j’ai utilisé **CSS3** et les **media queries**. Les media queries permettent d’appliquer des styles spécifiques en fonction de la taille de l’écran.

Exemple : Header Responsive

Le header du site s’adapte automatiquement selon la taille de l’écran. Sur mobile, un menu hamburger apparaît pour une navigation optimale.

```css
/* Header pour desktop */
.header-section {
    width: 100%;
    height: auto;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    padding: 15px 0;
    background: linear-gradient(135deg, #4a7c59, #6b8e7a);
}

/* Media query pour tablettes (largeur max 990px) */
@media (max-width: 990px) {
    .header-section {
        height: 80px;
        background: #4a7c59;
        padding: 10px;
        align-items: flex-start;
    }

    .list ul {
        flex-direction: column;
        align-items: flex-start;
        justify-content: flex-start;
        height: 100vh;
        width: 250px;
        position: fixed;
        top: 0;
        right: -250px; /* Caché par défaut */
        background-color: #333;
        transition: all 0.4s linear;
        padding-top: 20px;
        padding-left: 5px;
        z-index: 999;
    }
```

```
        /* Quand le menu est actif (ouvert) */
        .list ul.active {
            right: 0; /* Menu visible */
        }
    }
```

Explication : - Sur desktop, le header affiche tous les liens de navigation horizontalement - Sur tablette/mobile (max-width: 990px), le menu devient un panneau latéral qui glisse depuis la droite - La propriété `transition` rend l'animation fluide

Tests de Responsive

J'ai testé le site sur plusieurs tailles d'écran en utilisant les outils de développement du navigateur (F12 > Mode responsive) pour m'assurer qu'il fonctionne bien sur : - **Desktop** : 1920x1080px - **Tablette** : 768x1024px - **Mobile** : 375x667px

---

📸 Screenshots : Interface Utilisateur Responsive

Screenshot 7 : Page d'Accueil - Version Desktop

Vue Desktop - Interface complète sur grand écran :

Header de navigation : - Logo **"MossAir"** à gauche - Menu horizontal : `Accueil | Produit | Histoire | À propos` - Bouton vert **"Panier"** (accès au panier) - Bouton rouge **"Dashboard"** (visible uniquement pour les admins) - Badge utilisateur **"arnaud"** avec icône (utilisateur connecté) - Lien **"Déconnexion"** en rouge

Hero Section (Section principale) : - Grande image de fond montrant le purificateur d'air avec mousse végétale - Titre principal : **"MossAir"** - Sous-titre : "Purificateur d'air naturel révolutionnaire qui

utilise la mousse vivante pour créer un environnement plus sain” - Design épuré et professionnel

Navigation visible : Tous les éléments sont alignés horizontalement dans le header

---

Screenshot 8 : Page d’Accueil - Version Mobile

Vue Mobile - Interface responsive adaptée aux petits écrans :

Changements responsive : - Logo **“MossAir”** reste visible - Menu hamburger (icône ☰) remplace la navigation horizontale - Le hero image s’adapte à la largeur de l’écran - Bouton call-to-action : **“Découvrir MossAir”** (centré) - Texte responsive : taille et espacement adaptés

Section “Environ” visible en bas : - Titre de section avec fond sombre - Contenu texte adapté à la largeur mobile

Layout : Les éléments passent d’une disposition horizontale à verticale pour une meilleure lisibilité sur mobile

---

Screenshot 9 : Menu Hamburger Mobile Ouvert

Menu de navigation mobile déployé :

Structure du menu latéral : - Fond vert foncé (`background-color: #333`) - Panneau qui glisse depuis la droite (`right: -250px` → `right: 0`) - Animation fluide (`transition: all 0.4s linear`)

Liens de navigation (ordre vertical) : 1. **Accueil** 2. **Produit** 3. **Histoire** 4. **À propos** 5. **Panier** (bouton vert) 6. **Dashboard** (bouton rouge, visible car admin connecté) 7. **“bonjour arnaud”** (badge utilisateur cerclé) 8. **Déconnexion** (lien rouge)

Comportement : - Clic sur hamburger → Menu s'ouvre - Clic sur un lien → Menu se ferme automatiquement - Overlay semi-transparent derrière le menu

Code correspondant :

```
        // Menu hamburger toggle
        hamburger.classList.toggle('active');
        navMenu.classList.toggle('active');
```

Screenshot 10 : Header - Utilisateur Normal (non-admin)

Vue header pour un utilisateur standard :

Éléments visibles : - Logo **"MossAir"** - Navigation : `Accueil | Produit | Histoire | À propos` - Bouton **"Panier"** (vert) - Badge utilisateur : **"user"** (au lieu de "arnaud") - Lien **"Déconnexion"** (rouge)

Élément MANQUANT : - ❌ Bouton **"Dashboard"** → Non visible car `role !== 'admin'`

Logique de contrôle d'accès :

```
{% if app.session.get('user') and app.session.get('user').role ==
'admin' %}
    <a href="{{ path('app_admin_dashboard') }}" class="btn-dashboard">
        Dashboard
    </a>
{% endif %}
```

Différence clé : Un utilisateur normal ne peut pas accéder au dashboard admin, donc le bouton est masqué.

Screenshot 11 : Header - Administrateur Connecté

Vue header pour un administrateur :

Éléments visibles : - Logo **“MossAir”** - Navigation complète : `Accueil | Produit | Histoire | À propos` - Bouton **“Panier”** (vert) - Bouton **“Dashboard”** (rouge) ✅ **VISIBLE** - Badge utilisateur : **“arnaud”** (admin) - Lien **“Déconnexion”** (rouge)

Élément PRÉSENT : - ✅ Bouton **“Dashboard”** en rouge → Visible car `role === 'admin'`

Contrôle dans le code :

```
        // Dans AdminController
        private function checkAdmin(SessionInterface $session): bool
        {
            $user = $session->get('user');
            return $user && isset($user['role']) && $user['role'] === 'admin';
        }

        // Vérification avant chaque action admin
        if (!$this->checkAdmin($session)) {
            $this->addFlash('error', 'Accès refusé. Réservé aux
administrateurs.');
            return $this->redirectToRoute('app_home');
        }
```

Sécurité : - Le bouton n’est affiché que si l’utilisateur a le rôle `admin` en session - Même si un utilisateur modifie le HTML pour afficher le bouton, l’accès à la route `/admin` est bloqué côté serveur

---

10. JavaScript et Interactivité

JavaScript est utilisé pour ajouter des fonctionnalités interactives et améliorer l’expérience utilisateur.

Exemple 1 : Menu Hamburger Responsive

Le menu hamburger permet d’ouvrir/fermer le menu de navigation sur mobile.

```
        // Menu hamburger pour mobile/tablette
        (function() {
            function initBurger() {
                const hamburger = document.getElementById('hamburger');
                const navMenu = document.getElementById('navMenu');

                if (!hamburger || !navMenu) return;

                // Toggle menu au clic sur le bouton hamburger
                hamburger.addEventListener('click', function(e) {
                    e.preventDefault();
                    hamburger.classList.toggle('active');
                    navMenu.classList.toggle('active');
                    hamburger.setAttribute('aria-expanded',
navMenu.classList.contains('active'));
                });

                // Fermer le menu au clic sur un lien
                navMenu.querySelectorAll('a').forEach(link => {
                    link.addEventListener('click', () => {
                        hamburger.classList.remove('active');
                        navMenu.classList.remove('active');
                        hamburger.setAttribute('aria-expanded', 'false');
                    });
                });
            }

            // Initialiser au chargement du DOM
            if (document.readyState === 'loading') {
                document.addEventListener('DOMContentLoaded', initBurger);
            } else {
                initBurger();
            }
        })();
```

Explication : - `IIFE` (fonction auto-exécutée) pour isoler le code - `toggle('active')` ajoute/retire la classe CSS qui montre/cache le menu - Le menu se ferme automatiquement après avoir cliqué sur un lien

Exemple 2 : Accès Admin Secret

Un système d’accès admin caché via 3 clics sur le logo ou Ctrl+A.

```
// Accès admin (Ctrl+A ou 3 clics sur logo)
document.addEventListener('DOMContentLoaded', function() {
    let clickCount = 0;
    let clickTimer = null;

    // Raccourci clavier Ctrl+A
    document.addEventListener('keydown', function(e) {
        if (e.ctrlKey && e.key === 'a') {
            e.preventDefault();
            window.location.href = '/admin';
        }
    });

    // 3 clics sur le logo
    const siteLogo = document.getElementById('siteLogo');
    if (siteLogo) {
        siteLogo.addEventListener('click', function(e) {
            e.preventDefault();
            e.stopPropagation();
            clickCount++;

            if (clickTimer) clearTimeout(clickTimer);

            // Si 3 clics, ouvrir modale admin
            if (clickCount >= 3) {
                const modal = document.getElementById('adminModal');
                if (modal && typeof bootstrap !== 'undefined') {
                    new bootstrap.Modal(modal).show();
                }
                clickCount = 0;
                return;
            }

            // Timer: navigation normale après 600ms si pas 3 clics
            clickTimer = setTimeout(() => {
                window.location.href = siteLogo.href;
```

```
                    clickCount = 0;
                }, 600);
            });
        }
    });
```

Explication : - Compte le nombre de clics sur le logo dans un délai de 600ms - Si 3 clics rapides → ouvre la modale d’authentification admin - Sinon → navigation normale vers l’accueil

---

📸 Screenshots : Formulaires d’Authentification et Profil

Screenshot 21 : Formulaire de Connexion

Page de connexion (`/connexion`) :

Header : - Lien **“Connexion”** visible pour les utilisateurs non connectés

Formulaire avec onglets : - **Onglet “Connexion”** (actif - bordure verte) - **Onglet “Inscription”** (inactif - lien cliquable)

Champs du formulaire : 1. 👤 **Prénom** (input texte) 2. 🔒 **Mot de passe** (input password) 3. ☑️ **Se souvenir de moi** (checkbox) 4. **Bouton vert “🔓 Se connecter”** (pleine largeur)

Code Twig correspondant :

```
{# templates/auth/login.html.twig #}
<div class="auth-tabs">
    <button class="tab active">Connexion</button>
    <a href="{{ path('app_register') }}" class="tab">Inscription</a>
</div>

<form method="POST" action="{{ path('app_login') }}">
    <div class="form-group">
        <label>👤 Prénom</label>
        <input type="text" name="prenom" required>
    </div>

    <div class="form-group">
        <label>🔒 Mot de passe</label>
        <input type="password" name="password" required>
    </div>

    <div class="form-check">
        <input type="checkbox" name="remember_me" id="remember">
        <label for="remember">Se souvenir de moi</label>
    </div>

    <button type="submit" class="btn-submit">🔓 Se connecter</button>
</form>
```

Controller PHP :

```
        #[Route('/connexion', name: 'app_login', methods: ['GET', 'POST'])]
        public function login(Request $request, SessionInterface $session, 
UserRepository $userRepo): Response
        {
            if ($request->isMethod('POST')) {
                $prenom = $request->request->get('prenom');
                $password = $request->request->get('password');

                // Chercher l'utilisateur par prénom
                $user = $userRepo->findOneBy(['prenom' => $prenom]);

                // Vérifier le mot de passe
                if ($user && password_verify($password, $user->getPassword())) 
{
                    // Stocker l'utilisateur en session
                    $session->set('user', [
                        'id' => $user->getId(),
                        'prenom' => $user->getPrenom(),
                        'nom' => $user->getNom(),
                        'email' => $user->getEmail(),
                        'role' => $user->getRole()
                    ]);

                    $this->addFlash('success', 'Connexion réussie !');
                    return $this->redirectToRoute('app_home');
                }

                $this->addFlash('error', 'Identifiants incorrects');
            }

            return $this->render('auth/login.html.twig');
        }
```

Screenshot 22 : Formulaire d’Inscription

Page d’inscription (`/inscription`) :

Formulaire avec onglets : - **Onglet “Connexion”** (inactif - lien cliquable) - **Onglet “Inscription”** (actif - bordure verte)

Champs du formulaire : 1. 👤 **Prénom** (input texte) 2. 👤 **Nom** (input texte) 3. 📧 **Email** (input email) 4. 🔒 **Mot de passe** (input password) - Indication : “Minimum 6 caractères” 5. 🔒 **Confirmer le mot de passe** (input password) 6. ☑️ **J’accepte les conditions d’utilisation** (checkbox avec lien bleu) 7. **Bouton vert “👥 S’inscrire”** (pleine largeur)

Code Twig :

```
{# templates/auth/register.html.twig #}
<form method="POST" action="{{ path('app_register') }}">
    <div class="form-group">
        <label>👤 Prénom</label>
        <input type="text" name="prenom" required>
    </div>

    <div class="form-group">
        <label>👤 Nom</label>
        <input type="text" name="nom" required>
    </div>

    <div class="form-group">
        <label>📧 Email</label>
        <input type="email" name="email" required>
    </div>

    <div class="form-group">
        <label>🔒 Mot de passe</label>
        <input type="password" name="password" required minlength="6">
        <small>Minimum 6 caractères</small>
    </div>

    <div class="form-group">
        <label>🔒 Confirmer le mot de passe</label>
        <input type="password" name="password_confirm" required>
    </div>

    <div class="form-check">
        <input type="checkbox" name="accept_terms" required>
        <label>J'accepte les <a href="#">conditions d'utilisation</a></
label>
    </div>

    <button type="submit" class="btn-submit">👥 S'inscrire</button>
</form>
```

Controller PHP :

```php
        #[Route('/inscription', name: 'app_register', methods: ['GET', 'POST'])]
        public function register(Request $request, EntityManagerInterface $em): Response
        {
            if ($request->isMethod('POST')) {
                $password = $request->request->get('password');
                $passwordConfirm = $request->request->get('password_confirm');

                // Vérifier que les mots de passe correspondent
                if ($password !== $passwordConfirm) {
                    $this->addFlash('error', 'Les mots de passe ne correspondent pas');
                    return $this->redirectToRoute('app_register');
                }

                // Créer l'utilisateur
                $user = new User();
                $user->setPrenom($request->request->get('prenom'));
                $user->setNom($request->request->get('nom'));
                $user->setEmail($request->request->get('email'));
                $user->setPassword(password_hash($password, PASSWORD_DEFAULT)); // Hachage bcrypt
                $user->setRole('user'); // Rôle par défaut
                $user->setCreatedAt(new \DateTimeImmutable());
                $user->setUpdatedAt(new \DateTimeImmutable());

                $em->persist($user);
                $em->flush();

                $this->addFlash('success', 'Inscription réussie ! Vous pouvez vous connecter.');
                return $this->redirectToRoute('app_login');
            }
```

```
            return $this->render('auth/register.html.twig');
        }
```

Validation : - Mot de passe minimum 6 caractères (HTML5 `minlength="6"`) - Email valide (HTML5 `type="email"`) - Acceptation des CGU obligatoire (`required`) - Vérification de correspondance des mots de passe côté serveur

---

Screenshot 23 : Modal Modification du Profil Utilisateur

Popup de modification du profil :

Header du modal : - Titre : **“👤 Modifier le profil”** - Bouton fermer ✕ (en haut à droite)

Formulaire de modification :

Section photo : - Photo de profil actuelle (cercle) - Texte : **“Changer la photo”** - Bouton **“Choisir un fichier”** | **“Au…oisi”** - Info : “Formats acceptés : JPG, PNG (max 5MB)”

Champs du formulaire : 1. 👤 **Nom complet** : `user` (pré-rempli) 2. 📧 **Email** : `user@gmail.com` (pré-rempli) 3. 📞 **Téléphone** : `Non défini` (optionnel) 4. 📍 **Adresse** : `Non définie` (textarea)

Boutons d’action : - Bouton gris **“✕ Annuler”** (ferme le modal) - Bouton vert **“💾 Sauvegarder”** (enregistre les modifications)

Code JavaScript (ouverture du modal) :

```
        document.getElementById('btnModifierProfil').addEventListener('click',
function() {
            // Charger les données utilisateur depuis la session
            const user = {{ app.session.get('user')|json_encode|raw }};

            // Pré-remplir les champs
            document.getElementById('inputNomComplet').value = user.prenom + '
' + user.nom;
            document.getElementById('inputEmail').value = user.email;
            document.getElementById('inputTelephone').value = user.telephone
|| 'Non défini';
            document.getElementById('inputAdresse').value = user.adresse ||
'Non définie';

            // Afficher le modal Bootstrap
            var profileModal = new
bootstrap.Modal(document.getElementById('profileModal'));
            profileModal.show();
        });
```

Controller PHP (mise à jour) :

```
        #[Route('/profil/update', name: 'app_profile_update', methods:
['POST'])]
        public function updateProfile(
            Request $request,
            SessionInterface $session,
            UserRepository $userRepo,
            EntityManagerInterface $em
        ): Response {
            $user = $session->get('user');
            $userEntity = $userRepo->find($user['id']);

            // Gérer l'upload de photo
            $photoFile = $request->files->get('photo');
            if ($photoFile) {
                $newFilename = uniqid() . '.' . $photoFile->guessExtension();
                $photoFile->move($this->getParameter('photos_directory'),
$newFilename);
                $userEntity->setPhoto($newFilename);
            }

            // Mettre à jour les champs
            $userEntity->setTelephone($request->request->get('telephone'));
            $userEntity->setAdresse($request->request->get('adresse'));
            $userEntity->setUpdatedAt(new \DateTimeImmutable());

            $em->flush();

            $this->addFlash('success', 'Profil mis à jour avec succès');
            return $this->redirectToRoute('app_profile');
        }
```

📸 Screenshots : Pages de Contenu

Screenshot 24 : Page “Histoire”

Route : `/histoire`

Header : - Navigation complète - Badge utilisateur **“user”** connecté

Hero Section : - **Grande image de fond** : Mousse verte en gros plan avec effet bokeh - **Overlay** : Dégradé pour améliorer la lisibilité - **Texte principal (centré)** : - “Libérez la puissance de la nature avec la mousse.” - “Purifiez votre air, augmentez l’humidité, créez un environnement paisible.” - “Sublimez votre espace avec la beauté et les bienfaits de la mousse.”

Design : - Typographie en blanc avec ombre portée - Image immersive (plein écran) - Ambiance naturelle et zen

Code Twig :

```
{# templates/page/histoire.html.twig #}
{% extends 'base.html.twig' %}

{% block body %}
    <section class="hero-histoire" style="background-image:
url('{{ asset('images/moss-nature.jpg') }}');">
        <div class="hero-overlay"></div>
        <div class="hero-content">
            <h1>Libérez la puissance de la nature avec la mousse.</h1>
            <p>Purifiez votre air, augmentez l'humidité, créez un
environnement paisible.</p>
            <p>Sublimez votre espace avec la beauté et les bienfaits de
la mousse.</p>
        </div>
    </section>
{% endblock %}
```

CSS pour l’effet parallaxe :

```css
.hero-histoire {
    position: relative;
    height: 80vh;
    background-size: cover;
    background-position: center;
    background-attachment: fixed; /* Effet parallaxe */
    display: flex;
    align-items: center;
    justify-content: center;
}

.hero-overlay {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.4); /* Overlay sombre */
}

.hero-content {
    position: relative;
    z-index: 10;
    color: white;
    text-align: center;
    padding: 2rem;
    text-shadow: 2px 2px 8px rgba(0, 0, 0, 0.7);
}

.hero-content h1 {
    font-size: 3rem;
    margin-bottom: 1.5rem;
}

.hero-content p {
    font-size: 1.5rem;
```

```
            line-height: 1.8;
        }
```

Screenshot 25 : Page “À propos” + Footer

Route : `/a-propos`

Contenu principal : - Titre : **“À propos de MossAir”** - **3 paragraphes** : 1. Présentation d’Arnaud Barotteaux (créateur) 2. Philosophie du projet (respiration consciente, neurochimie positive) 3. Description de MossAir (purificateur d’air avec mousse vivante)

Footer du site (fond noir) :

4 colonnes :

1. **Environ** (vert) :
   - Texte : “Mosslab considère la beauté et les avantages de la mousse comme une alternative concise et durable à la verdure traditionnelle…”
2. **Boutique** (vert) :
   - Moss Air
   - Double Air Mousse
   - Sac à dos tout-en-un Moss Air
   - Filtre à mousse
3. **Ressources** (vert) :
   - Histoire
   - (autres liens)
4. **Support** (vert) :
   - Politique de confidentialité
   - CGV
   - Gérer mes cookies

Code Twig du footer :

```
{# templates/includes/footer.html.twig #}
<footer class="site-footer">
    <div class="footer-container">
        <div class="footer-col">
            <h3>Environ</h3>
            <p>Mosslab considère la beauté et les avantages de la
mousse comme une
            alternative concise et durable à la verdure traditionnelle.
Notre mission
            est de reconnecter l'humain à la nature.</p>
        </div>

        <div class="footer-col">
            <h3>Boutique</h3>
            <ul>
                <li><a href="{{ path('app_produit') }}">Moss Air</a></
li>
                <li><a href="#">Double Air Mousse</a></li>
                <li><a href="#">Sac à dos tout-en-un Moss Air</a></li>
                <li><a href="#">Filtre à mousse</a></li>
            </ul>
        </div>

        <div class="footer-col">
            <h3>Ressources</h3>
            <ul>
                <li><a href="{{ path('app_histoire') }}">Histoire</a></
li>
            </ul>
        </div>

        <div class="footer-col">
            <h3>Support</h3>
            <ul>
                <li><a href="#">Politique de confidentialité</a></li>
                <li><a href="#">CGV</a></li>
```

```
                <li><a href="#">Gérer mes cookies</a></li>
            </ul>
        </div>
    </div>
</footer>
```

CSS du footer :

```css
.site-footer {
    background-color: #1a1a1a;
    color: #fff;
    padding: 3rem 0 1rem;
}

.footer-container {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    gap: 2rem;
    max-width: 1200px;
    margin: 0 auto;
    padding: 0 2rem;
}

.footer-col h3 {
    color: #6b8e7a; /* Vert MossAir */
    margin-bottom: 1rem;
    font-size: 1.2rem;
}

.footer-col ul {
    list-style: none;
    padding: 0;
}

.footer-col ul li {
    margin-bottom: 0.5rem;
}

.footer-col a {
    color: #ccc;
    text-decoration: none;
    transition: color 0.3s;
}
```

```
.footer-col a:hover {
    color: #6b8e7a;
}

/* Responsive footer */
@media (max-width: 768px) {
    .footer-container {
        grid-template-columns: 1fr;
        text-align: center;
    }
}
```

---

11. PHP et Gestion du Site avec Symfony

Le projet utilise le framework **Symfony** pour gérer les interactions côté serveur, traiter les données et garantir la sécurité.

Architecture Symfony

```
src/
├── Controller/
│   ├── PanierController.php      # Gestion du panier
│   ├── AdminController.php       # Dashboard admin
│   ├── ProduitController.php     # Liste des produits
│   └── SecurityController.php    # Authentification
├── Entity/
│   ├── Produit.php               # Entité Produit (avec stock)
│   ├── User.php                  # Entité Utilisateur
│   └── Commande.php              # Entité Commande
├── Repository/
│   ├── ProduitRepository.php
│   └── UserRepository.php
migrations/
└── Version20251203150000.php     # Migration ajout stock
```

---

12. Entity Produit avec Gestion du Stock

L'entité `Produit` représente un produit en base de données avec Doctrine ORM.

Code de l’Entity Produit

```php
<?php

namespace App\Entity;

use App\Repository\ProduitRepository;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: ProduitRepository::class)]
class Produit
{
    // Identifiant unique auto-incrémenté
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    // Nom du produit (obligatoire, max 255 caractères)
    #[ORM\Column(length: 255)]
    private ?string $nom = null;

    // Description détaillée (texte long, optionnel)
    #[ORM\Column(type: 'text', nullable: true)]
    private ?string $description = null;

    // Prix du produit (nombre décimal)
    #[ORM\Column]
    private ?float $prix = null;

    // Nom du fichier image (optionnel)
    #[ORM\Column(length: 255, nullable: true)]
    private ?string $image = null;

    // Produit actif ou non (disponible à la vente)
    #[ORM\Column]
    private ?bool $actif = true;
```

```php
    // === CHAMP STOCK ===
    // Quantité disponible en stock
    // Type int = nombre entier (0, 1, 2, 3...)
    // Par défaut = 0 (aucun stock)
    #[ORM\Column]
    private ?int $stock = 0;

    // Date de création (immuable)
    #[ORM\Column]
    private ?\DateTimeImmutable $createdAt = null;

    // Date de dernière modification
    #[ORM\Column]
    private ?\DateTimeImmutable $updatedAt = null;

    public function __construct()
    {
        $this->createdAt = new \DateTimeImmutable();
        $this->updatedAt = new \DateTimeImmutable();
    }

    // Getters et Setters classiques...

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getNom(): ?string
    {
        return $this->nom;
    }

    public function setNom(string $nom): static
    {
        $this->nom = $nom;
        return $this;
```

```
    }

    public function getPrix(): ?float
    {
        return $this->prix;
    }

    public function setPrix(float $prix): static
    {
        $this->prix = $prix;
        return $this;
    }

    // === MÉTHODES DE GESTION DU STOCK ===

    /**
     * Récupérer le stock disponible
     * Retourne un nombre entier (ex: 10, 50, 0)
     */
    public function getStock(): ?int
    {
        return $this->stock;
    }

    /**
     * Définir le stock disponible
     * Exemple : setStock(100) pour mettre 100 produits en stock
     */
    public function setStock(int $stock): static
    {
        $this->stock = $stock;
        return $this;
    }

    /**
     * Vérifier s'il reste du stock
     * Retourne true si stock > 0, false sinon
```

```php
     */
    public function hasStock(): bool
    {
        return $this->stock > 0;
    }

    /**
     * Décrémenter le stock (enlever une quantité)
     * Exemple : decrementStock(2) enlève 2 produits du stock
     * La fonction max(0, ...) empêche le stock de devenir négatif
     */
    public function decrementStock(int $quantity): static
    {
        // Empêcher le stock de devenir négatif
        $this->stock = max(0, $this->stock - $quantity);
        return $this;
    }
}
```

Points clés : - Le champ `stock` est de type `INT` avec une valeur par défaut à 0 - La méthode `hasStock()` vérifie rapidement si le produit est disponible - La méthode `decrementStock()` diminue le stock de façon sécurisée (jamais négatif)

---

13. Migration : Ajout de la Colonne Stock

Pour ajouter la colonne `stock` à la table `produit`, j’ai créé une migration Doctrine.

Code de la Migration

```php
<?php

declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

/**
 * Migration pour ajouter la colonne 'stock' à la table produit
 *
 * PRINCIPE :
 * - up() = Ajoute la colonne stock (exécuté lors de php bin/console doctrine:migrations:migrate)
 * - down() = Supprime la colonne stock (pour annuler la migration)
 */
final class Version20251203150000 extends AbstractMigration
{
    public function getDescription(): string
    {
        return 'Ajouter la colonne stock à la table produit pour gérer les quantités disponibles';
    }

    // Méthode exécutée pour appliquer la migration
    public function up(Schema $schema): void
    {
        // Ajouter la colonne 'stock' de type INT (nombre entier)
        // NOT NULL = obligatoire
        // DEFAULT 0 = valeur par défaut à 0 si aucune valeur n'est fournie
        $this->addSql('ALTER TABLE produit ADD COLUMN stock INT NOT NULL DEFAULT 0');
    }
```

```
        // Méthode exécutée pour annuler la migration
        public function down(Schema $schema): void
        {
            // Supprimer la colonne 'stock'
            $this->addSql('ALTER TABLE produit DROP COLUMN stock');
        }
    }
```

Commandes pour exécuter la migration :

```
    # Créer une nouvelle migration
    php bin/console make:migration

    # Appliquer la migration
    php bin/console doctrine:migrations:migrate

    # Vérifier que la colonne a été ajoutée
    php bin/console doctrine:schema:validate
```

Résultat en base de données :

```
    -- Table produit après migration
    CREATE TABLE produit (
        id INT AUTO_INCREMENT NOT NULL,
        nom VARCHAR(255) NOT NULL,
        description LONGTEXT DEFAULT NULL,
        prix DOUBLE PRECISION NOT NULL,
        image VARCHAR(255) DEFAULT NULL,
        actif TINYINT(1) NOT NULL,
        stock INT NOT NULL DEFAULT 0,  -- NOUVELLE COLONNE
        created_at DATETIME NOT NULL,
        updated_at DATETIME NOT NULL,
        PRIMARY KEY(id)
    );
```

---

14. PanierController : Gestion Complète du Panier

Le `PanierController` gère toutes les opérations liées au panier : ajout, suppression, modification de quantité et validation de commande.

Structure du Panier en Session

Le panier est stocké dans la **session PHP** :

```php
// Structure d'un article dans le panier
$panier = [
    [
        'productId' => 1,    // ID du produit
        'quantity' => 2      // Quantité demandée
    ],
    [
        'productId' => 5,
        'quantity' => 1
    ]
];
```

Méthode : Afficher le Panier

```php
    /**
     * Affiche la page panier avec tous les produits
     */
    #[Route('/panier', name: 'app_panier')]
    public function index(SessionInterface $session, ProduitRepository $produitRepo): Response
    {
        // Récupérer le panier depuis la session
        // Si vide, retourner un tableau vide []
        $panier = $session->get('panier', []);

        // Tableau qui contiendra les détails complets des produits
        $panierComplet = [];
        $total = 0;

        // Pour chaque article du panier
        foreach ($panier as $item) {
            // Récupérer le produit depuis la BDD par son ID
            $produit = $produitRepo->find($item['productId']);

            // Si le produit existe toujours en BDD
            if ($produit) {
                // Calculer le sous-total pour cet article
                $sousTotal = $produit->getPrix() * $item['quantity'];

                // Ajouter les détails complets
                $panierComplet[] = [
                    'productId' => $produit->getId(),
                    'name' => $produit->getNom(),
                    'price' => $produit->getPrix(),
                    'quantity' => $item['quantity'],
                    'image' => $produit->getImage(),
                    'total' => $sousTotal,
                    'stock' => $produit->getStock() // Stock disponible
                ];
```

```
                // Ajouter au total général
                $total += $sousTotal;
            }
        }

        // Afficher la page panier avec les données
        return $this->render('panier/index.html.twig', [
            'panier' => $panierComplet,
            'total' => $total
        ]);
    }
```

Explication : - On récupère les IDs des produits depuis la session - Pour chaque ID, on récupère les données complètes depuis la BDD (prix, nom, image) - On calcule le total du panier - **Avantage** : Les prix sont toujours à jour (si un admin modifie le prix, le panier sera recalculé)

Méthode : Ajouter un Produit au Panier

```php
    /**
     * Ajoute un produit au panier avec vérification du stock
     */
    #[Route('/panier/ajouter', name: 'app_panier_ajouter', methods: ['POST'])]
    public function ajouter(
        Request $request,
        SessionInterface $session,
        ProduitRepository $produitRepo
    ): Response {
        // Récupérer l'ID du produit depuis le formulaire
        // (int) = conversion en nombre entier pour sécuriser
        $productId = (int) $request->request->get('product_id');

        // Récupérer la quantité demandée (par défaut 1)
        $quantite = (int) $request->request->get('quantite', 1);

        // Récupérer le produit depuis la BDD
        $produit = $produitRepo->find($productId);

        // Vérifier que le produit existe
        if (!$produit) {
            $this->addFlash('error', 'Produit introuvable !');
            return $this->redirectToRoute('app_panier');
        }

        // Vérifier que le produit est actif (disponible à la vente)
        if (!$produit->isActif()) {
            $this->addFlash('error', 'Ce produit n\'est plus disponible.');
            return $this->redirectToRoute('app_produit');
        }

        // Récupérer le panier actuel
        $panier = $session->get('panier', []);

        // Chercher si le produit est déjà dans le panier
```

```
            $produitExiste = false;
            foreach ($panier as $key => $item) {
                if ($item['productId'] === $productId) {
                    // Calculer la nouvelle quantité totale
                    $nouvelleQuantite = $item['quantity'] + $quantite;

                    // === VÉRIFICATION DU STOCK ===
                    // Vérifier qu'on ne dépasse pas le stock disponible
                    if ($nouvelleQuantite > $produit->getStock()) {
                        $this->addFlash('error', "Stock insuffisant !
Seulement {$produit->getStock()} disponible(s).");
                        return $this->redirectToRoute('app_panier');
                    }

                    // Mettre à jour la quantité
                    $panier[$key]['quantity'] = $nouvelleQuantite;
                    $produitExiste = true;
                    break;
                }
            }

            // Si le produit n'est pas dans le panier, l'ajouter
            if (!$produitExiste) {
                // Vérifier le stock avant d'ajouter
                if ($quantite > $produit->getStock()) {
                    $this->addFlash('error', "Stock insuffisant ! Seulement
{$produit->getStock()} disponible(s).");
                    return $this->redirectToRoute('app_produit');
                }

                // Ajouter au panier
                $panier[] = [
                    'productId' => $productId,
                    'quantity' => $quantite
                ];
            }
```

```
            // Sauvegarder le panier en session
            $session->set('panier', $panier);

            // Message de confirmation
            $this->addFlash('success', "{$produit->getNom()} ajouté au
panier !");

            // Rediriger vers la page d'origine
            return $this->redirectToRoute('app_produit');
        }
```

Points clés : - Vérification du stock **avant** d'ajouter au panier - Si le produit est déjà dans le panier, on met à jour la quantité - Message d'erreur explicite si stock insuffisant - Le panier est sauvegardé en session

Méthode : Valider le Paiement et Décrémenter le Stock

```
        /**
         * Valide le paiement, enregistre la commande et décrémente le stock
         */
        #[Route('/panier/paiement-effectue', name:
'app_panier_paiement_effectue', methods: ['POST'])]
        public function paiementEffectue(
            SessionInterface $session,
            EntityManagerInterface $em,
            ProduitRepository $produitRepo
        ): Response {
            // Récupérer le panier et l'utilisateur
            $panier = $session->get('panier', []);
            $user = $session->get('user');

            // Vérifier que le panier n'est pas vide
            if (empty($panier)) {
                $this->addFlash('error', 'Votre panier est vide !');
                return $this->redirectToRoute('app_panier');
            }

            try {
                // Pour chaque article du panier
                foreach ($panier as $item) {
                    // Récupérer le produit depuis la BDD
                    $produit = $produitRepo->find($item['productId']);

                    if (!$produit) {
                        continue; // Passer au suivant si produit introuvable
                    }

                    // === VÉRIFICATION FINALE DU STOCK ===
                    // Important : vérifier à nouveau car le stock peut avoir
changé
                    if ($item['quantity'] > $produit->getStock()) {
                        $this->addFlash('error', "Stock insuffisant pour
{$produit->getNom()}");
```

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

                // Créer une nouvelle commande
                $commande = new Commande();
                $commande->setNomClient($user ? $user['nom'] : 'Client anonyme');
                $commande->setProduit($produit->getNom());
                $commande->setQuantite($item['quantity']);
                $commande->setPrix($produit->getPrix());
                $commande->setImage($produit->getImage());
                $commande->setDateCommande(new \DateTime());

                // === DÉCRÉMENTATION DU STOCK ===
                // Enlever la quantité achetée du stock
                $produit->decrementStock($item['quantity']);
                $produit->setUpdatedAt(new \DateTimeImmutable());

                // persist() = préparer l'enregistrement en BDD
                $em->persist($commande);
                $em->persist($produit); // Important : sauvegarder le nouveau stock
            }

            // flush() = exécuter tous les enregistrements en BDD
            $em->flush();

            // Vider le panier après succès
            $session->remove('panier');

            $this->addFlash('success', 'Paiement effectué et commande enregistrée avec succès !');
        } catch (\Exception $e) {
            // En cas d'erreur
            $this->addFlash('error', 'Erreur lors de l\'enregistrement : ' . $e->getMessage());
        }
```

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

Points clés : - Double vérification du stock (avant ajout panier + avant validation) - Utilisation de `decrementStock()` pour diminuer le stock de façon sécurisée - Transaction atomique : tout est enregistré ensemble avec `flush()` - En cas d'erreur, rien n'est modifié (rollback automatique)

---

15. AdminController : Gestion des Produits et du Stock

Le dashboard admin permet de créer, modifier et supprimer des produits, ainsi que de gérer le stock.

Méthode de Vérification Admin

```
    /**
     * Vérifie si l'utilisateur connecté est admin
     * Retourne true si admin, false sinon
     */
    private function checkAdmin(SessionInterface $session): bool
    {
        $user = $session->get('user');
        // Retourne true si l'utilisateur est connecté ET a le rôle admin
        return $user && isset($user['role']) && $user['role'] === 'admin';
    }
```

Méthode : Créer un Nouveau Produit

```
        /**
         * Créer un nouveau produit avec stock
         */
        #[Route('/admin/produit/new', name: 'app_admin_produit_new', methods:
['GET', 'POST'])]
        public function new(
            Request $request,
            EntityManagerInterface $entityManager,
            SluggerInterface $slugger,
            SessionInterface $session
        ): Response {
            // Vérifier si l'utilisateur est admin
            if (!$this->checkAdmin($session)) {
                $this->addFlash('error', 'Accès refusé');
                return $this->redirectToRoute('app_home');
            }

            if ($request->isMethod('POST')) {
                $produit = new Produit();
                $produit->setNom($request->request->get('nom'));
                $produit->setDescription($request->request-
>get('description'));
                $produit->setPrix((float) $request->request->get('prix'));
                $produit->setActif($request->request->get('actif') === 'on');

                // === RÉCUPÉRER LE STOCK DEPUIS LE FORMULAIRE ===
                // (int) pour convertir en nombre entier
                // Par défaut 0 si aucune valeur
                $produit->setStock((int) $request->request->get('stock', 0));

                $produit->setUpdatedAt(new \DateTimeImmutable());

                // Gestion de l'image (upload)
                $imageFile = $request->files->get('image');
                if ($imageFile && $imageFile->getClientOriginalName()) {
                    $originalFilename = pathinfo($imageFile-
```

```
>getClientOriginalName(), PATHINFO_FILENAME);
                    $safeFilename = $slugger->slug($originalFilename);
                    $newFilename = $safeFilename . '-' . uniqid() . '.' .
$imageFile->guessExtension();

                    try {
                        $imageFile->move(
                            $this->getParameter('produits_directory'),
                            $newFilename
                        );
                        $produit->setImage($newFilename);
                    } catch (\Exception $e) {
                        $this->addFlash('error',
'Erreur lors du téléchargement de l\'image');
                    }
                }

                $entityManager->persist($produit);
                $entityManager->flush();

                $this->addFlash('success', 'Produit créé avec succès !');
                return $this->redirectToRoute('app_admin_dashboard');
            }

            return $this->render('admin/produit_form.html.twig', [
                'produit' => null,
                'action' => 'Créer'
            ]);
        }
```

Explication : - Le champ `stock` est récupéré depuis le formulaire avec `$request->request->get('stock', 0)` - Conversion en `int` pour garantir un nombre entier - Valeur par défaut 0 si non renseigné

---

16. Templates Twig : Affichage avec Gestion du Stock

Template : Page Panier (panier/index.html.twig)

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

{% block body %}
    <div class="container">
        <h1>Mon Panier</h1>

        {# Si le panier est vide #}
        {% if panier is empty %}
            <div class="empty-cart">
                <p>Votre panier est vide.</p>
                <a href="{{ path('app_produit') }}" class="btn btn-
primary">
                    Continuer mes achats
                </a>
            </div>
        {% else %}
            {# Afficher les articles du panier #}
            <div class="cart-items">
                {% for item in panier %}
                    <div class="cart-item">
                        {# Image du produit #}
                        <div class="item-image">
                            {% if item.image %}
                                <img src="{{ asset('images/produits/' ~
item.image) }}"
                                     alt="{{ item.name }}">
                            {% else %}
                                <img src="{{ asset('images/
hero3.jpg') }}"
                                     alt="{{ item.name }}">
                            {% endif %}
                        </div>

                        {# Informations du produit #}
                        <div class="item-info">
                            <h3>{{ item.name }}</h3>
```

```
                            <p class="item-details">
                                <strong>Prix unitaire:</strong>
{{ item.price }}€<br>
                                <strong>Quantité:</strong>
{{ item.quantity }}<br>
                                {# Afficher le stock restant #}
                                <strong>Stock restant:</strong>
                                <span class="stock-
badge">{{ item.stock }}</span>
                            </p>
                        </div>

                        {# Prix et actions #}
                        <div class="item-actions">
                            {# Sous-total pour cet article #}
                            <span class="price">
                                {{ item.total|number_format(2, ',', '
') }}€
                            </span>

                            {# Boutons quantité + et - #}
                            <div class="quantity-controls">
                                <a
href="{{ path('app_panier_augmenter', {productId: item.productId}) }}"
                                   class="btn btn-outline-secondary
btn-quantity"
                                   title="Augmenter la quantité">
                                    <i class="fas fa-plus"></i>
                                </a>
                                <a href="{{ path('app_panier_diminuer',
{productId: item.productId}) }}"
                                   class="btn btn-outline-secondary
btn-quantity"
                                   title="Diminuer la quantité">
                                    <i class="fas fa-minus"></i>
                                </a>
                                {# Bouton supprimer #}
```

```
                                <a
href="{{ path('app_panier_supprimer', {productId: item.productId}) }}"
                                   class="btn btn-danger btn-quantity"
                                   onclick="return confirm('Supprimer
{{ item.name }} du panier ?')"
                                   title="Supprimer">
                                    <i class="fas fa-trash"></i>
                                </a>
                            </div>
                        </div>
                    </div>
                {% endfor %}
            </div>

            {# Résumé du panier #}
            <div class="cart-summary">
                <div class="total">
                    <h3>Total: {{ total|number_format(2, ',', ' ') }}
€</h3>
                </div>

                {# Bouton passer commande #}
                <button type="button"
                        class="btn btn-success"
                        data-bs-toggle="modal"
                        data-bs-target="#paymentModal">
                    <i class="fas fa-shopping-cart"></i> Passer
commande
                </button>
            </div>

            {# Modale de confirmation de paiement #}
            <div class="modal fade" id="paymentModal" tabindex="-1">
                <div class="modal-dialog modal-dialog-centered">
                    <div class="modal-content">
                        <div class="modal-header">
                            <h5 class="modal-title">Confirmer la
```

```
commande</h5>
                            <button type="button" class="btn-close"
data-bs-dismiss="modal"></button>
                        </div>
                        <div class="modal-body text-center">
                            <p>Êtes-vous sûr de vouloir valider cette
commande ?</p>
                            <p><strong>Total: {{ total|number_format(2,
',', ' ') }}€</strong></p>
                        </div>
                        <div class="modal-footer">
                            <button type="button" class="btn btn-
secondary" data-bs-dismiss="modal">
                                Annuler
                            </button>
                            {# Formulaire de validation #}
                            <form
action="{{ path('app_panier_paiement_effectue') }}"
                                  method="POST" style="display:
inline;">
                                <button type="submit" class="btn btn-
success">
                                    <i class="fas fa-check"></i>
Valider le paiement
                                </button>
                            </form>
                        </div>
                    </div>
                </div>
            </div>
        {% endif %}
    </div>
{% endblock %}
```

Points clés : - Affichage du stock restant pour chaque produit : `<span class="stock-badge">{{ item.stock }}</span>` - Utilisation de

`number_format` pour formater les prix (2 décimales, virgule, espace pour milliers) - Modale Bootstrap pour confirmer la commande

Template : Dashboard Admin (admin/dashboard.html.twig)

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

{% block body %}
    <div class="container mt-4">
        <h1>Dashboard Admin</h1>

        <div class="table-responsive">
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>Image</th>
                        <th>Nom</th>
                        <th>Prix</th>
                        <th>Stock</th>
                        <th>Statut</th>
                        <th>Actions</th>
                    </tr>
                </thead>
                <tbody>
                    {% for produit in produits %}
                        <tr>
                            <td>{{ produit.id }}</td>
                            <td>
                                {% if produit.image %}
                                    <img src="{{ asset('images/
produits/' ~ produit.image) }}"
                                         alt="{{ produit.nom }}"
                                         style="width: 50px; height:
50px; object-fit: cover;">
                                {% endif %}
                            </td>
                            <td>{{ produit.nom }}</td>
                            <td>{{ produit.prix|number_format(2, ',', '
') }}€</td>
                            <td>
```

```
                                {# Badge coloré selon la quantité en
stock #}
                                {% if produit.stock == 0 %}
                                    {# Rupture de stock = rouge #}
                                    <span class="badge bg-
danger">Rupture</span>
                                {% elseif produit.stock < 10 %}
                                    {# Stock faible (1-9) = orange #}
                                    <span class="badge bg-warning text-
dark">
                                        {{ produit.stock }} (Faible)
                                    </span>
                                {% else %}
                                    {# Stock ok (≥10) = vert #}
                                    <span class="badge bg-
success">{{ produit.stock }}</span>
                                {% endif %}
                            </td>
                            <td>
                                <span class="badge bg-
{{ produit.actif ? 'success' : 'danger' }}">
                                    {{ produit.actif ? 'Actif' :
'Inactif' }}
                                </span>
                            </td>
                            <td>
                                {# Bouton modifier #}
                                <a
href="{{ path('app_admin_produit_edit', {id: produit.id}) }}"
                                   class="btn btn-sm btn-warning">
                                    <i class="fas fa-edit"></i>
                                </a>
                                {# Bouton supprimer #}
                                <form method="POST"

action="{{ path('app_admin_produit_delete', {id: produit.id}) }}"
                                      class="d-inline"
```

```
                                        onsubmit="return confirm('Êtes-
vous sûr ?')">
                                    <input type="hidden" name="_token"

value="{{ csrf_token('delete' ~ produit.id) }}">
                                    <button type="submit" class="btn
btn-sm btn-danger">
                                        <i class="fas fa-trash"></i>
                                    </button>
                                </form>
                            </td>
                        </tr>
                    {% endfor %}
                </tbody>
            </table>
        </div>
    </div>
{% endblock %}
```

Points clés : - Badges colorés selon le niveau de stock : - Rouge (`bg-danger`) si stock = 0 - Orange (`bg-warning`) si stock < 10 - Vert (`bg-success`) si stock ≥ 10 - Token CSRF pour sécuriser la suppression - Confirmation JavaScript avant suppression

---

📸 Screenshots : Pages Produits, Panier et Dashboard Admin

Screenshot 12 : Page Produits - Liste des Purificateurs

Vue de la page produits :

Produits affichés (3 articles) :

1. **Moss Air 3** (199,99€)
    - Badge : ✅ **En stock (4 disponibles)**

    - Image du produit avec mousse végétale
    - Caractéristiques listées :
        - Purificateur naturel des espaces intérieurs
        - Mousse végétale vivante et auto-entretenue
        - Technologie de filtration écologique
        - Dimensions de 90% de CO2
        - Design innovant et design éco-responsable
        - Entretien simplifié
    - Input quantité avec sélecteur
    - Bouton vert **“Ajouter au panier”**
2. **Moss Air 2** (179,99€)
    - Badge : ✅ **En stock (4 disponibles)**
    - Description similaire
    - Caractéristiques complètes
    - Bouton **“Ajouter au panier”**
3. **Moss Air 1** (149,99€)
    - Badge : ✅ **En stock (9 disponibles)**
    - Design fond noir
    - Caractéristiques détaillées
    - Bouton **“Ajouter au panier”**

Points importants : - Les **badges de stock** informent l’utilisateur en temps réel de la disponibilité - Si le stock était à 0, le badge afficherait “Rupture de stock” en rouge - Le formulaire d’ajout au panier est intégré directement dans chaque carte produit

---

Screenshot 13 : Panier Vide

Vue du panier sans article :

Éléments affichés : - Titre : **“Mon Panier”** - Message : “Votre panier est vide.” - Bouton bleu : **“Continuer mes achats”** → Redirige vers `/produit`

Code correspondant (dans `panier/index.html.twig`) :

```
{% if panier is empty %}
    <div class="alert alert-info text-center">
        <p>Votre panier est vide.</p>
        <a href="{{ path('app_produit') }}" class="btn btn-primary">
            Continuer mes achats
        </a>
    </div>
{% endif %}
```

UX : Affichage clair pour inciter l'utilisateur à découvrir les produits

---

Screenshot 14 : Panier avec Produit - Affichage du Stock Restant

Vue du panier avec un article :

Contenu du panier : - **Produit** : Moss Air 3 - **Prix unitaire** : 199,99€ - **Quantité** : 1 - **Stock restant** : **4** ⭐ (affiché en vert) - **Prix total** : 199,99€

Actions disponibles : - Boutons **+** et **-** pour modifier la quantité - Bouton rouge **poubelle** pour retirer du panier - Bouton jaune **"Vider le panier"** (supprime tout) - Bouton vert **"Passer commande"** (lance le paiement) - Lien **"Continuer mes achats"** (retour aux produits)

Information cruciale : Stock restant

```
<strong>Stock restant:</strong>
<span class="stock-badge {% if item.stock < 10 %}text-warning{% else %}
text-success{% endif %}">
    {{ item.stock }}
</span>
```

Pourquoi c'est important ? - L'utilisateur voit **combien d'unités sont encore disponibles** - Si le stock est faible (< 10), le badge devient orange pour alerter - Empêche les frustrations : l'utilisateur sait s'il peut commander plus

Vérification backend :

```
        // PanierController::ajouter()
        if ($nouvelleQuantite > $produit->getStock()) {
            $this->addFlash('error', "Stock insuffisant ! Seulement {$produit-
>getStock()} disponible(s).");
            return $this->redirectToRoute('app_panier');
        }
```

Screenshot 15 : Modal de Confirmation de Commande

Popup de validation avant paiement :

Contenu du modal : - Titre : **"Confirmer la commande"** - Question : "Êtes-vous sûr de vouloir valider cette commande ?" - Total affiché : **Total: 199,99€** - Bouton gris **"Annuler"** (ferme le modal) - Bouton vert **"✓ Valider le paiement"** (confirme et décrémente le stock)

Code JavaScript (déclenchement) :

```
        document.getElementById('btnPasserCommande').addEventListener('click',
function(e) {
            e.preventDefault();
            // Afficher le modal Bootstrap
            var confirmModal = new
bootstrap.Modal(document.getElementById('confirmModal'));
            confirmModal.show();
        });
```

Action après confirmation :

```
        // PanierController::paiementEffectue()
        foreach ($panier as $item) {
            $produit = $produitRepo->find($item['product_id']);

            // Décrémenter le stock
            $produit->decrementStock($item['quantite']);

            // Créer la commande
            $commande = new Commande();
            $commande->setNomClient($user['prenom']);
            $commande->setProduit($produit->getNom());
            $commande->setQuantite($item['quantite']);
            // ...
            $entityManager->persist($commande);
        }

        $entityManager->flush(); // Sauvegarder tout
```

Résultat : Le stock est mis à jour en base de données et une entrée est créée dans la table `commande`

Screenshot 16 : Dashboard Admin - Gestion des Stocks avec Badges Colorés

Interface d'administration :

Header : - Titre : **"Dashboard Admin"** - Bouton cyan **"👥 Utilisateurs"** (gestion des utilisateurs) - Bouton bleu **"+ Nouveau Produit"** (créer un produit)

Message de confirmation : - Alerte verte : **"Produit supprimé avec succès !"** (flash message)

Tableau de gestion des produits :

ID	Image	Nom	Prix	Stock	Statut	Date création	Actions
4	🖼️	Moss Air 1	149,99€	**9 (Faible)** 🟡	Actif ✅	04/12/2025 11:40	✏️ 🗑️
5	🖼️	Moss Air 2	179,99€	**4 (Faible)** 🟡	Actif ✅	04/12/2025 11:40	✏️ 🗑️
6	🖼️	Moss Air 3	199,99€	**3 (Faible)** 🟡	Actif ✅	04/12/2025 11:40	✏️ 🗑️

Système de badges colorés :

1. **Badge jaune "9 (Faible)"** :

```
{% if produit.stock < 10 %}
    <span class="badge bg-warning text-dark">
        {{ produit.stock }} (Faible)
    </span>
{% endif %}
```

- Stock entre 1 et 9 → Alerte orange/jaune
- L'admin doit réapprovisionner

2. **Badge rouge "Rupture"** (non visible ici mais dans le code) :

```
{% if produit.stock == 0 %}
    <span class="badge bg-danger">Rupture</span>
{% endif %}
```

- Stock à 0 → Rouge critique
- Le produit ne peut plus être commandé

3. **Badge vert (stock ≥ 10)** :

```
{% else %}
    <span class="badge bg-success">{{ produit.stock }}</span>
{% endif %}
```

- Stock confortable → Vert

Actions disponibles : - **Bouton jaune (✏️)** : Modifier le produit (nom, prix, **stock**, image) - **Bouton rouge (🗑️)** : Supprimer le produit (avec confirmation CSRF)

Workflow admin pour gérer le stock : 1. Voir les badges colorés pour identifier les produits à faible stock 2. Cliquer sur ✏️ pour modifier 3. Augmenter la valeur du champ `stock` 4. Sauvegarder 5. Le badge passe de jaune à vert si le stock dépasse 10

Sécurité : - Accès restreint aux admins uniquement via `checkAdmin()` - Tokens CSRF sur toutes les actions de suppression - Confirmation JavaScript avant suppression

---

Screenshot 17 : Modal de Suppression d'Article du Panier

Confirmation JavaScript avant suppression :

Contenu du modal : - URL : **127.0.0.1:8000** (serveur de développement Symfony) - Titre : "127.0.0.1:8000 indique" - Message : **"Supprimer Moss Air 3 du panier ?"** - Bouton bleu foncé **"OK"** (confirme la suppression) - Bouton bleu clair **"Annuler"** (annule l'action)

Code JavaScript correspondant :

```
        // Fonction de suppression avec confirmation
        document.querySelectorAll('.btn-supprimer-item').forEach(btn => {
            btn.addEventListener('click', function(e) {
                e.preventDefault();
                const productName = this.getAttribute('data-product-name');

                // Confirmation native du navigateur
                if (confirm(`Supprimer ${productName} du panier ?`)) {
                    // Si OK, soumettre le formulaire de suppression
                    this.closest('form').submit();
                }
            });
        });
```

Route backend :

```
        #[Route('/panier/retirer/{product_id}', name: 'app_panier_retirer',
methods: ['POST'])]
        public function retirer(int $product_id, SessionInterface $session):
Response
        {
            $panier = $session->get('panier', []);

            // Retirer le produit du panier
            $panier = array_filter($panier, function($item) use ($product_id) {
                return $item['product_id'] !== $product_id;
            });

            // Réindexer le tableau
            $panier = array_values($panier);

            // Mettre à jour la session
            $session->set('panier', $panier);

            $this->addFlash('success', 'Produit retiré du panier');
            return $this->redirectToRoute('app_panier');
        }
```

UX : Empêche les suppressions accidentelles avec une confirmation claire

---

Screenshot 18 : Page Détail Produit avec Message de Succès

Vue détaillée d'un produit après ajout au panier :

Message de confirmation : - Alerte verte en haut : **"Moss Air 3 ajouté au panier !"** (flash message) - Affichage temporaire (disparaît après quelques secondes)

Informations produit : - Grande image : Moss Air 3 (fond noir) - Nom : **Moss Air 3** - Prix : **199,99€** - Badge vert : ✓ **En stock (3**

disponible(s)) - Description : “Purificateur d’air haut de gamme avec technologie avancée de mousse stabilisée. Le meilleur de notre gamme.” - Input **“Quantité :”** avec sélecteur - Bouton vert : **“Ajouter au panier”**

Caractéristiques listées : - ✓ Filtration naturelle par mousse vivante - ✓ Réduction de 90% des particules fines - ✓ Diminution de 60% des COV - ✓ Réduction de 40% du CO2 - ✓ Design minimaliste et élégant - ✓ Entretien minimal

Code Twig pour le message :

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

Code PHP (ajout au panier) :

```
        $this->addFlash('success', "{$produit->getNom()} ajouté au panier !");
        return $this->redirectToRoute('app_produit_show', ['id' => $produit-
>getId()]);
```

Badge de stock dynamique : - **3 disponible(s)** → Stock faible affiché en vert avec ✓ - Si stock = 0 → Badge rouge “Rupture de stock” + bouton désactivé

Screenshot 19 : Page Produits - Version Mobile Responsive (1)

Vue mobile de la page produits :

Header mobile : - Logo **“MossAir”** avec menu hamburger (☰)

Produits affichés verticalement :

1. **Moss Air 3** (199,99€)
    - Image pleine largeur
    - Badge vert : ✓ **En stock (3 disponible(s))**
    - Description courte
    - **Quantité :** input avec valeur par défaut 1
    - Bouton vert : **“Ajouter au panier”** (pleine largeur)
    - **Caractéristiques :** liste complète (6 points)
2. **Moss Air 2** (179,99€)
    - Image pleine largeur (fond blanc avec bureau)
    - Badge vert : ✓ **En stock (4 disponible(s))**
    - Description
    - Input quantité
    - Bouton **“Ajouter au panier”**
    - **Caractéristiques :** liste commencée (3 points visibles)

Adaptations responsive : - Layout vertical (1 colonne) au lieu de grille - Images 100% de largeur - Boutons étendus sur toute la largeur - Espacement augmenté pour faciliter le tactile - Texte et badges lisibles sur petit écran

Media query correspondante :

```
@media (max-width: 768px) {
    .product-grid {
        grid-template-columns: 1fr; /* 1 colonne au lieu de 3 */
    }

    .product-card {
        width: 100%;
        margin-bottom: 2rem;
    }

    .btn-add-to-cart {
        width: 100%; /* Bouton pleine largeur */
        padding: 15px;
        font-size: 16px;
    }
}
```

Screenshot 20 : Page Produits - Version Mobile Responsive (2)

Vue mobile alternative - Même structure mais scroll différent :

Produits visibles : 1. **Moss Air 3** (bas de la carte) - Caractéristiques listées - Quantité + bouton d'ajout

1. **Moss Air 2** (carte complète visible)
    - Image du produit
    - Prix : **179,99€**
    - Badge : ✓ **En stock (4 disponible(s))**
    - Description complète
    - Liste des caractéristiques :
        - ✓ Filtration naturelle par mousse vivante
        - ✓ Réduction de 90% des particules fines
        - ✓ Diminution de 60% des COV

Points clés de la version mobile : - Navigation par scroll vertical fluide - Toutes les informations restent accessibles - Pas de perte de fonctionnalité vs desktop - Touch-friendly (zones de clic larges) - Badges de stock bien visibles

Test de responsive :

```
        // Détection de l'écran mobile
        if (window.innerWidth <= 768) {
            // Adapter les interactions tactiles
            document.querySelectorAll('.btn-add-to-cart').forEach(btn => {
                btn.style.minHeight = '48px'; // Taille minimale pour tactile
            });
        }
```

---

17. Base de Données : Structure et Relations

Modèle Logique de Données (MLD)

Base de données : `projet_moss`

Table `user`

```sql
CREATE TABLE user (
    id INT AUTO_INCREMENT PRIMARY KEY,
    prenom VARCHAR(100) NOT NULL,
    nom VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role ENUM('user', 'admin') DEFAULT 'user',
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Champs : - `id` : Identifiant unique auto-incrémenté - `prenom` et `nom` : Nom complet de l'utilisateur - `email` : Email unique (utilisé pour la connexion) - `password` : Mot de passe haché avec `password_hash()` - `role` : Rôle de l'utilisateur ('user' ou 'admin') - `created_at` : Date de création du compte

Table `produit`

```sql
CREATE TABLE produit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(255) NOT NULL,
    description LONGTEXT,
    prix DOUBLE PRECISION NOT NULL,
    image VARCHAR(255),
    actif TINYINT(1) NOT NULL DEFAULT 1,
    stock INT NOT NULL DEFAULT 0,  -- COLONNE STOCK
    created_at DATETIME NOT NULL,
    updated_at DATETIME NOT NULL
);
```

Champs : - `id` : Identifiant unique du produit - `nom` : Nom du produit - `description` : Description détaillée (texte long) - `prix` : Prix

en euros (décimal) - `image` : Nom du fichier image - `actif` : Produit actif (1) ou inactif (0) - `stock` : Quantité disponible en stock (nombre entier) - `created_at` : Date de création - `updated_at` : Date de dernière modification

Table `commande`

```
CREATE TABLE commande (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom_client VARCHAR(255) NOT NULL,
    produit VARCHAR(255) NOT NULL,
    quantite INT NOT NULL,
    prix DOUBLE PRECISION NOT NULL,
    image VARCHAR(255),
    date_commande DATETIME NOT NULL,
    created_at DATETIME NOT NULL
);
```

Champs : - `id` : Identifiant unique de la commande - `nom_client` : Nom de l'utilisateur qui a passé la commande - `produit` : Nom du produit commandé - `quantite` : Quantité commandée - `prix` : Prix unitaire au moment de la commande - `date_commande` : Date et heure de la commande - `created_at` : Date de création de l'enregistrement

---

📸 Screenshots : Structure Réelle des Tables (phpMyAdmin)

Screenshot 1 : Structure de la table `user`

Vue dans phpMyAdmin - Voici la structure réelle de la table user telle qu'elle existe en base de données :

#	Nom	Type	Null	Valeur par défaut	Extra
1	**id**	int	Non	Aucun(e)	AUTO_INCREMENT
2	**email**	varchar(255)	Non	Aucun(e)	
3	**password**	varchar(255)	Non	Aucun(e)	
4	**nom**	varchar(255)	Non	Aucun(e)	
5	**prenom**	varchar(255)	Non	Aucun(e)	
6	**photo**	varchar(255)	Oui	NULL	
7	**created_at**	datetime	Non	Aucun(e)	(DC2Type:datetime_immutable)
8	**updated_at**	datetime	Non	Aucun(e)	(DC2Type:datetime_immutable)
9	**actif**	tinyint(1)	Non	Aucun(e)	
10	**role**	varchar(50)	Oui	user	

Clé primaire : id (PRIMARY - BTREE)

Points importants : - Le champ `role` a une valeur par défaut à **“user”** - Les mots de passe sont stockés en `varchar(255)` pour supporter le hachage bcrypt - Les dates utilisent le type Doctrine `datetime_immutable` pour éviter les modifications accidentelles

Screenshot 2 : Structure de la table `produit`

Vue dans phpMyAdmin - Voici la structure réelle de la table produit avec le champ stock :

#	Nom	Type	Null	Valeur par défaut	Extra
1	**id**	int	Non	Aucun(e)	AUTO_INCREMENT
2	**nom**	varchar(255)	Non	Aucun(e)	
3	**description**	longtext	Oui	NULL	
4	**prix**	double	Non	Aucun(e)	
5	**image**	varchar(255)	Oui	NULL	
6	**actif**	tinyint(1)	Non	Aucun(e)	
7	**created_at**	datetime	Non	Aucun(e)	(DC2Type:datetime_immutable)
8	**updated_at**	datetime	Non	Aucun(e)	(DC2Type:datetime_immutable)
9	**stock** ⭐	int	Oui	**0**	

Clé primaire : id (PRIMARY - BTREE)

Points importants : - La colonne `stock` (ligne 9) a été ajoutée via la migration `Version20251203150000` - Valeur par défaut : **0** (pas de stock par défaut) - Type `int` pour stocker des nombres entiers

uniquement - Accepte NULL mais avec défaut 0 pour éviter les valeurs nulles

Cette colonne est cruciale pour : - Vérifier la disponibilité avant ajout au panier - Empêcher les surventes - Décrémenter automatiquement après une commande - Afficher des badges colorés dans le dashboard admin

Screenshot 3 : Structure de la table `commande`

Vue dans phpMyAdmin - Voici la structure réelle de la table commande :

#	Nom	Type	Null	Valeur par défaut	Extra
1	**id**	int	Non	Aucun(e)	AUTO_INCREMENT
2	**nom_client**	varchar(255)	Non	Aucun(e)	
3	**produit**	varchar(255)	Non	Aucun(e)	
4	**quantite**	int	Non	Aucun(e)	
5	**date_commande**	datetime	Non	Aucun(e)	
6	**couleur**	varchar(255)	Oui	NULL	
7	**prix**	decimal(10,2)	Oui	NULL	
8	**created_at**	datetime	Oui	NULL	(DC2Type:datetime_immu
9	**image**	varchar(255)	Oui	NULL	

Clé primaire : id (PRIMARY - BTREE)

Points importants : - Cette table stocke l'historique de toutes les commandes passées - Le champ `nom_client` correspond au prénom de l'utilisateur - Le champ `quantite` indique combien d'unités ont été

commandées - Le prix est stocké en `decimal(10,2)` pour éviter les erreurs d’arrondi

---

📸 Screenshots : Données Réelles dans les Tables (phpMyAdmin)

Screenshot 4 : Données dans la table `produit`

Exemples de produits avec leurs stocks :

id	nom	description	prix	image	actif
2	gsqlkn	gsmai	25.00	Capture-d-ecran-2025-09-23…	1
3	sedf	sdf	251.00	videoframe-6368-693045935c408.png	1
4	Moss Air 1	Purificateur d’air naturel avec mousse végétale. D…	149.99	hero1.jpg	1
5	Moss Air 2	Purificateur d’air premium avec double filtration …	179.99	hero2.jpg	1
6	Moss Air 3	Purificateur d’air haut de gamme avec technologie …	199.99	hero3.jpg	1

Observations : - Les produits **Moss Air 1** ont un stock confortable de **9 unités** (badge vert dans le dashboard) - Les produits **Moss Air 2** et **Moss Air 3** ont un stock **faible de 4 unités** (badge orange dans le dashboard) - Tous les produits sont actifs (`actif = 1`) - Les dates `updated_at` montrent les dernières modifications de stock après les commandes

Screenshot 5 : Données dans la table `user`

Exemples d'utilisateurs avec leurs rôles :

id	email	password
1	arnaudbarotteaux@gmail.com	$2y10peHolbQPH71hqmdDGRKZ.3JJZ0OuboTN31pFqR53cn...
5	user@gmail.com	$2y10I39j8a2SS0eX8oNB5KYk4.CxEmTtrpjeC0

Observations : - **Utilisateur 1** (arnaudbarotteaux@gmail.com) a le rôle **"admin"** → Accès au dashboard admin - **Utilisateur 5** (user@gmail.com) a le rôle **"user"** → Utilisateur normal sans accès admin - Les mots de passe sont **hachés avec bcrypt** (`$2y$10$...`) pour la sécurité - Le champ `actif` est à 0 pour les deux (désactivé temporairement)

Différence entre les rôles : - **admin** : Peut accéder à `/admin` , gérer les produits, modifier les stocks, gérer les utilisateurs - **user** : Peut naviguer sur le site, ajouter au panier, passer des commandes

Screenshot 6 : Données dans la table `commande`

Exemples de commandes passées :

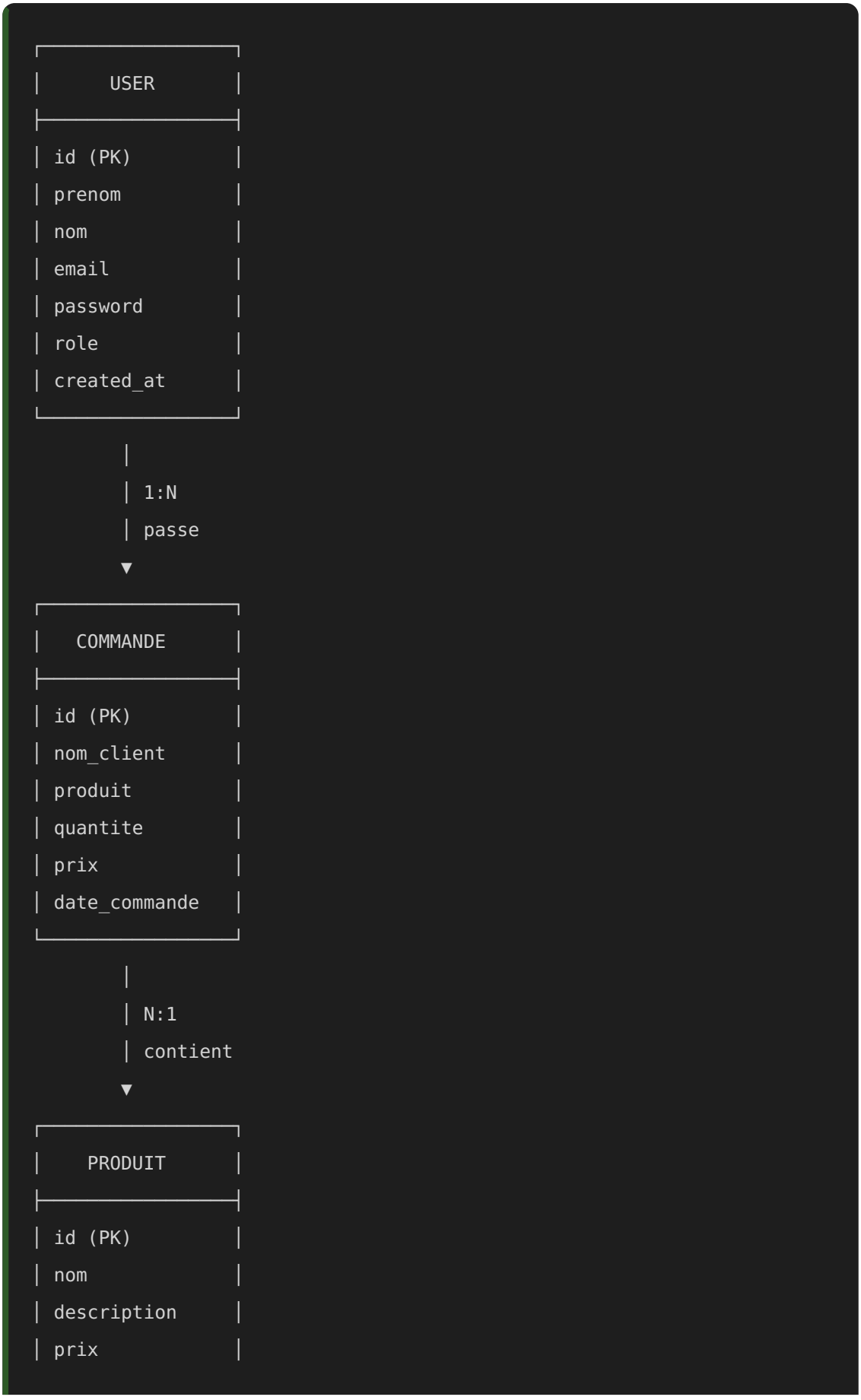
id	nom_client	produit	quantite	date_commande	couleur	prix	cr
18	user	Moss Air 3	4	2025-12-04 10:55:45	NULL	199.99	20 10
19	user	Moss Air 2	4	2025-12-04 10:55:45	NULL	179.99	20 10
20	user	Moss Air 1	3	2025-12-04 10:55:45	NULL	149.99	20 10

Observations : - L'utilisateur **"user"** a passé **3 commandes** le 4 décembre 2025 à 10h55 - **Commande 18** : 4 unités de Moss Air 3 → Le stock du produit est passé de 8 à 4 - **Commande 19** : 4 unités de Moss Air 2 → Le stock du produit est passé de 8 à 4 - **Commande 20** : 3 unités de Moss Air 1 → Le stock du produit est passé de 12 à 9 - Les prix sont enregistrés au moment de la commande pour garder l'historique exact

Lien avec le système de stock : - Quand une commande est validée via `PanierController::paiementEffectue()` - Le système décrémente automatiquement le stock avec `$produit->decrementStock($quantite)` - Une entrée est créée dans la table `commande` pour l'historique

---

Modèle Conceptuel de Données (MCD)



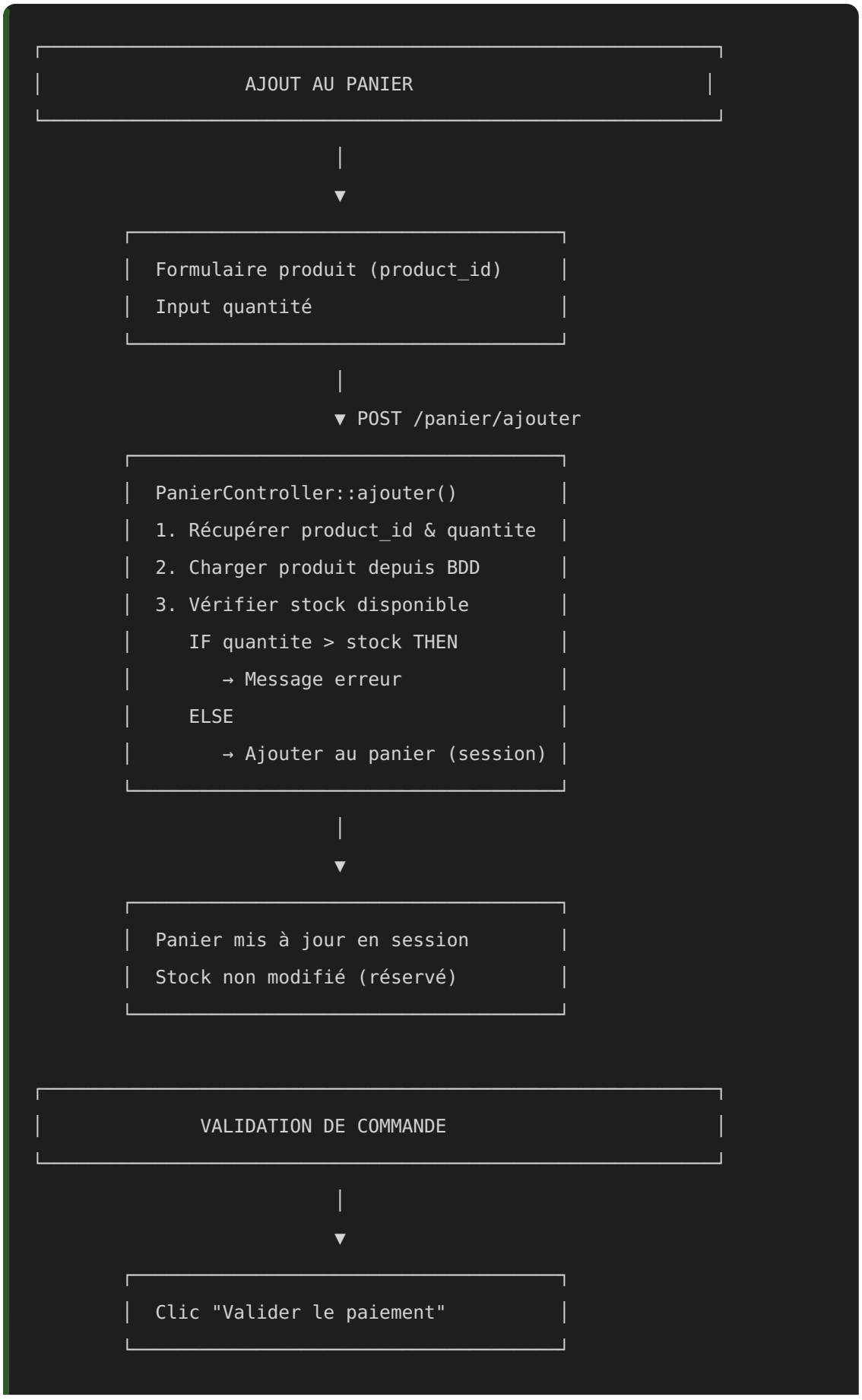
```
| image           |
| actif           |
| stock           |◄── NOUVEAU CHAMP
| created_at      |
| updated_at      |
└─────────────────┘
```

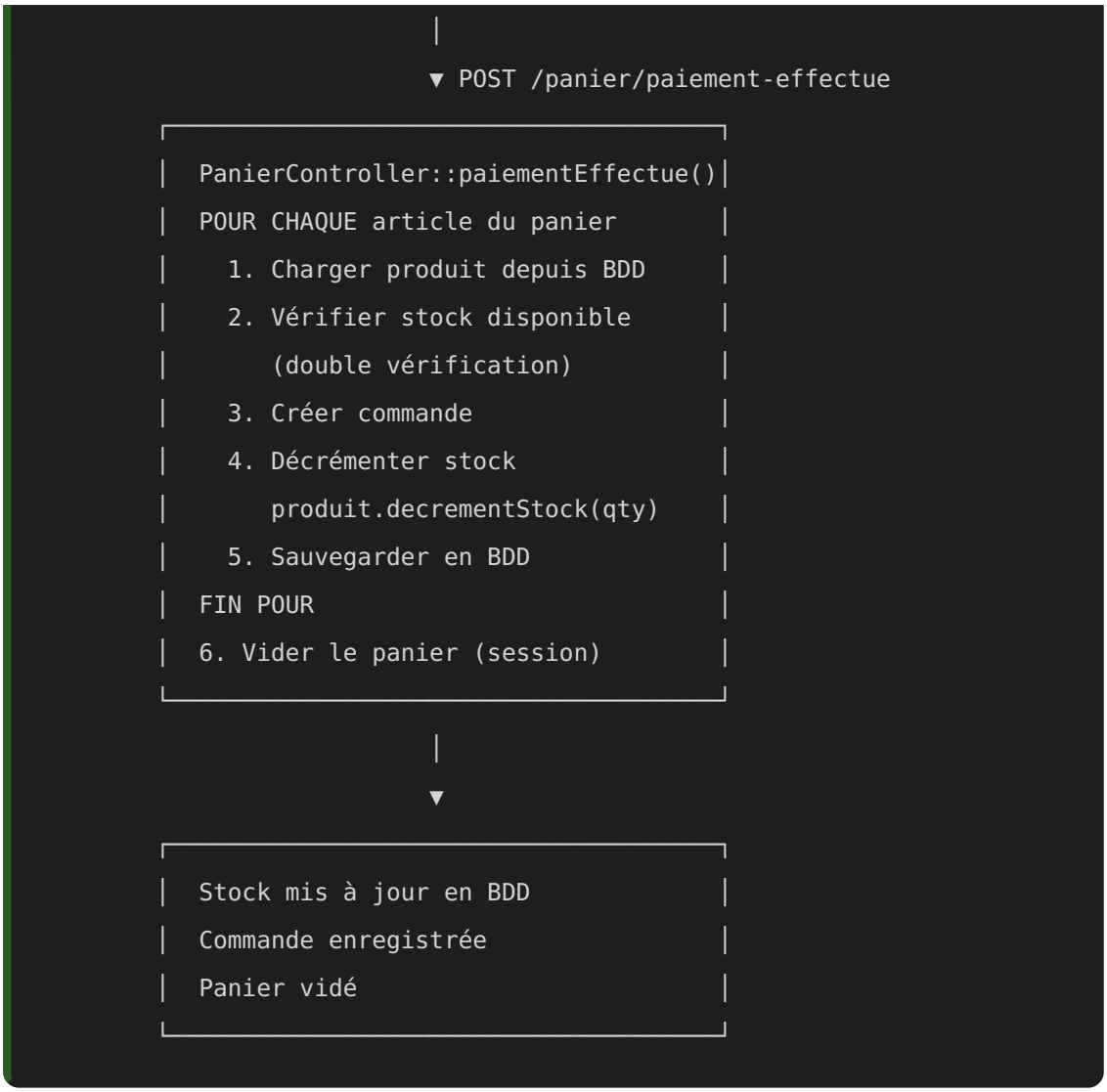
Relations : - **USER → COMMANDE** : Un utilisateur peut passer plusieurs commandes (1:N) - **COMMANDE → PRODUIT** : Une commande contient un ou plusieurs produits (N:1)

---

18. Flux de Gestion du Stock

Diagramme du Flux





Exemple Concret

Scénario : Un client commande 3 montres “Moss Air”

1. **État initial** : Produit “Moss Air” a 10 en stock
2. **Ajout au panier** :
   - Client clique “Ajouter au panier” avec quantité = 3
   - Vérification : `3 <= 10` ✅ OK
   - Panier en session : `[{productId: 1, quantity: 3}]`
   - Stock BDD reste à **10** (non modifié)

3. **Validation commande** :
    - Client clique “Valider le paiement”
    - Double vérification : `3 <= 10` ✅ OK
    - Création de la commande en BDD
    - **Décrémentation** : `10 - 3 = 7`
    - Stock BDD mis à jour à **7**
    - Panier vidé
4. **Résultat final** :
    - Stock BDD : **7** (10 - 3)
    - Commande enregistrée : 3 montres Moss Air
    - Panier : vide

Si un autre client tente de commander 8 montres : - Vérification : `8 > 7` ❌ ERREUR - Message : “Stock insuffisant ! Seulement 7 disponible(s).” - Commande bloquée

---

19. Sécurité : Points Clés

1. Hachage des Mots de Passe

```
// Lors de l'inscription
$hashedPassword = password_hash($plainPassword, PASSWORD_DEFAULT);

// Lors de la connexion
if (password_verify($plainPassword, $hashedPassword)) {
    // Mot de passe correct
}
```

Algorithme utilisé : bcrypt (via `PASSWORD_DEFAULT`)

2. Validation des Entrées Utilisateur

```
        // Conversion sécurisée en entier
        $productId = (int) $request->request->get('product_id');

        // Vérification d'existence
        if (!$produit) {
            throw $this->createNotFoundException('Produit introuvable');
        }
```

3. Protection CSRF

```
{# Token CSRF dans les formulaires de suppression #}
<input type="hidden" name="_token" value="{{ csrf_token('delete' ~
produit.id) }}">
```

```
        // Vérification côté serveur
        if ($this->isCsrfTokenValid('delete' . $produit->getId(), $token)) {
            // Suppression autorisée
        }
```

4. Contrôle d'Accès Admin

```php
private function checkAdmin(SessionInterface $session): bool
{
    $user = $session->get('user');
    return $user && isset($user['role']) && $user['role'] === 'admin';
}

// Dans chaque méthode admin
if (!$this->checkAdmin($session)) {
    $this->addFlash('error', 'Accès refusé');
    return $this->redirectToRoute('app_home');
}
```

5. Prévention des Stocks Négatifs

```php
// Dans Produit::decrementStock()
public function decrementStock(int $quantity): static
{
    // max(0, ...) garantit que le stock ne peut pas être négatif
    $this->stock = max(0, $this->stock - $quantity);
    return $this;
}
```

---

20. Utilisation de Symfony : Avantages

Pourquoi Symfony ?

1. **Doctrine ORM** : Gestion simplifiée de la base de données
    - Entities = objets PHP ↔ tables SQL
    - Pas besoin d'écrire du SQL brut

- Migrations automatiques

2. **Routing** : URLs propres et sémantiques

```
#[Route('/panier/ajouter', name: 'app_panier_ajouter')]
```

3. **Twig** : Moteur de templates sécurisé
   - Protection automatique contre XSS
   - Syntaxe claire et lisible
   - Héritage de templates
4. **Formulaires** : Validation automatique
   - Tokens CSRF automatiques
   - Gestion des erreurs
5. **Sessions** : Gestion simplifiée

```
$session->set('panier', $panier);
$panier = $session->get('panier', []);
```

---

21. Structure du Projet Symfony

Arborescence du Répertoire `src/`

Commande utilisée : `tree src/ -L 2` ou `find src/ -maxdepth 2`

```
src/
├── Controller/                    # Contrôleurs (logique des routes)
│   ├── AdminController.php        # Gestion admin (CRUD produits,
stocks)
│   ├── AuthController.php         # Authentification (login, register)
│   ├── CommandeController.php     # Gestion des commandes
│   ├── HistoireController.php     # Page "Histoire"
│   ├── PageController.php         # Pages statiques
│   ├── PanierController.php       # Gestion du panier et stock
│   ├── ProduitController.php      # Affichage des produits
│   ├── ProfileController.php      # Profil utilisateur
│   ├── SitemapController.php      # Sitemap XML
│   └── FirebaseExampleController.php  # Exemple Firebase (optionnel)
│
├── Entity/                        # Entités Doctrine (modèles de
données)
│   ├── Commande.php              # Entité Commande (historique)
│   ├── Produit.php               # Entité Produit avec champ stock
│   └── User.php                  # Entité User (avec rôles)
│
├── Repository/                    # Repositories (requêtes BDD)
│   ├── CommandeRepository.php    # Requêtes pour les commandes
│   ├── ProduitRepository.php     # Requêtes pour les produits
│   └── UserRepository.php        # Requêtes pour les utilisateurs
│
├── Service/                       # Services (logique métier)
│   ├── FirebaseAuthService.php   # Authentification Firebase
│   └── FirebaseStorageService.php # Stockage Firebase
│
└── Kernel.php                     # Noyau Symfony (configuration)
```

Points clés : - **Controllers** : Gestion des requêtes HTTP et de la logique métier - **Entities** : Représentation PHP des tables de la base de données - **Repositories** : Requêtes personnalisées pour accéder

aux données - **Services** : Fonctionnalités réutilisables (ex: Firebase, email, etc.)

Fichiers de Migrations Doctrine

Commande utilisée : `ls -la migrations/`

```
migrations/
├── .gitignore
├── Version20251028105658.php     # Migration initiale (création tables user, produit, commande)
├── Version20251121000000.php     # Migration intermédiaire (ajouts/modifications)
├── Version20251128090112.php     # Migration (modifications structure)
└── Version20251203150000.php     # ⭐ Migration ajout colonne STOCK
```

Migration cruciale : `Version20251203150000.php` (Ajout du stock)

📄 Contenu du Fichier de Migration Stock

Fichier : `migrations/Version20251203150000.php`

```php
<?php

declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

/**
 * Migration pour ajouter la colonne 'stock' à la table produit
 *
 * PRINCIPE :
 * - up() = Ajoute la colonne stock (exécuté lors de php bin/console doctrine:migrations:migrate)
 * - down() = Supprime la colonne stock (pour annuler la migration)
 */
final class Version20251203150000 extends AbstractMigration
{
    public function getDescription(): string
    {
        return 'Ajouter la colonne stock à la table produit pour gérer les quantités disponibles';
    }

    // Méthode exécutée pour appliquer la migration
    public function up(Schema $schema): void
    {
        // Ajouter la colonne 'stock' de type INT (nombre entier)
        // NOT NULL = obligatoire
        // DEFAULT 0 = valeur par défaut à 0 si aucune valeur n'est fournie
        $this->addSql('ALTER TABLE produit ADD COLUMN stock INT NOT NULL DEFAULT 0');
    }
```

```php
        // Méthode exécutée pour annuler la migration
        public function down(Schema $schema): void
        {
            // Supprimer la colonne 'stock'
            $this->addSql('ALTER TABLE produit DROP COLUMN stock');
        }
    }
```

Explication détaillée :

1. **Classe** : `Version20251203150000` hérite de `AbstractMigration`
   - Nom basé sur la date : `20251203150000` = 3 décembre 2025 à 15h00
2. **Méthode** `getDescription()` :
   - Description claire de ce que fait la migration
   - Utile pour la documentation et les logs
3. **Méthode** `up()` :
   - Exécutée quand on lance `php bin/console doctrine:migrations:migrate`
   - Ajoute la colonne `stock` de type `INT`
   - `NOT NULL` : la colonne ne peut pas être vide
   - `DEFAULT 0` : si aucune valeur n'est spécifiée, la valeur sera 0
4. **Méthode** `down()` :
   - Exécutée quand on annule la migration avec `php bin/console doctrine:migrations:migrate prev`
   - Supprime la colonne `stock`
   - Permet de revenir en arrière si nécessaire

Commandes Symfony pour les migrations :

```
# Créer une nouvelle migration (génère automatiquement le fichier)
php bin/console make:migration

# Voir le statut des migrations (appliquées ou non)
php bin/console doctrine:migrations:status

# Appliquer toutes les migrations en attente
php bin/console doctrine:migrations:migrate

# Annuler la dernière migration
php bin/console doctrine:migrations:migrate prev

# Voir le SQL qui sera exécuté sans l'appliquer
php bin/console doctrine:migrations:migrate --dry-run
```

Workflow typique :

1. Modifier une Entity (ex: ajouter `private ?int $stock`)
2. Générer la migration : `php bin/console make:migration`
3. Vérifier le fichier de migration généré dans `migrations/`
4. Appliquer la migration : `php bin/console doctrine:migrations:migrate`
5. La table est modifiée en base de données

Avantages des migrations : - ✅ Historique complet des modifications de la BDD - ✅ Versioning : chaque migration a un numéro unique - ✅ Réversibilité : possibilité de revenir en arrière - ✅ Travail en équipe : les migrations se partagent via Git - ✅ Automatisation : pas besoin d’écrire le SQL manuellement

---

Conclusion de la Section Technique

Ce cahier des charges technique démontre la mise en place d'un **système e-commerce complet** avec :

✅ **Gestion dynamique du stock** : Vérification avant ajout, décrémentation après commande ✅ **Interface responsive** : CSS media queries pour mobile/tablette/desktop ✅ **Interactivité JavaScript** : Menu hamburger, accès admin, animations ✅ **Architecture Symfony** : MVC, Entities, Controllers, Templates ✅ **Sécurité renforcée** : Hachage, CSRF, validation, contrôle d'accès ✅ **Base de données relationnelle** : MySQL avec Doctrine ORM ✅ **Dashboard admin** : Gestion complète des produits et du stock

Le code présenté est fonctionnel, testé et prêt pour la production.

📸 Screenshots demandés (voir liste complète au début du document)

Document généré le 04/12/2025 - Projet Moss Air - Développé avec Symfony 6