Dossier de Projet - Knowledge Learning

"Plateforme E-Learning & E-Commerce"

💼 SOMMAIRE

1. PRÉSENTATION DU PROJET

🎓 Knowledge Learning

Plateforme E-Learning avec Paiement Intégré

Projet académique - Développement Backend Orienté Objet
Cadre : Projet d'évaluation - Conception d'une application e-commerce/e-learning

Contexte

L'entreprise fictive **Knowledge**, éditeur de livres de formation, souhaite étendre son offre en ligne via une plateforme e-learning permettant à ses clients d'étudier en autonomie depuis chez eux.

Mission

Concevoir et développer le **backend complet** de la plateforme :

- Base de données relationnelle optimisée
- Composants d'accès aux données (Repositories)
- Système de paiement intégré (Stripe)
- Architecture orientée objet maintenable

2. RÉSUMÉ DU PROJET

Knowledge Learning est une plateforme e-learning développée avec **Symfony 7.4** permettant aux utilisateurs d'acheter et suivre des formations en ligne. L'application propose une architecture hiérarchique Thème → Cursus → Leçons, avec un système d'authentification sécurisé nécessitant une activation par email. Les utilisateurs peuvent acheter des cursus complets ou des leçons individuelles via **Stripe** en mode sandbox. Un système de validation progressive des leçons permet l'obtention automatique de certifications « Knowledge Learning » après complétion d'un cursus. L'interface d'administration, développée avec **EasyAdminBundle**, offre une gestion complète des utilisateurs, contenus et transactions. Le projet respecte les principes SOLID avec une architecture 3-tiers (MVC + Service Layer), intègre des tests unitaires et fonctionnels (PHPUnit), et utilise des design patterns modernes (Action Controller, Repository, DTO). La sécurité est assurée par un hashage bcrypt des mots de passe, une protection CSRF native Symfony, et une gestion fine des autorisations avec les rôles ROLE_USER et ROLE_ADMIN.

3. LIEN REPOSITORY GITHUB

⚯ Accès au Code Source

Repository principal :
https://github.com/MathP-dev/knowledge_learning

Structure du Repository

```
knowledge_learning/
├── src/
│   ├── Controller/      # Action Controllers (MVC)
│   ├── Entity/          # Modèles Doctrine ORM
│   ├── Repository/      # Composants d'accès aux données
│   ├── Service/         # Logique métier (Service Layer)
│   ├── DTO/             # Data Transfer Objects
│   ├── Security/        # Authenticator personnalisé
│   └── DataFixtures/    # Données de test
├── tests/
│   ├── Unit/            # Tests unitaires
│   └── Functional/      # Tests fonctionnels
├── templates/           # Vues Twig
├── migrations/          # Migrations Doctrine
├── config/              # Configuration Symfony
├── README.md            # Documentation d'installation
├── CONFIG_TEST.md       # Guide de configuration des tests
└── TESTS_ANALYSIS.md    # Analyse de la couverture de tests
```

Fichiers README Disponibles

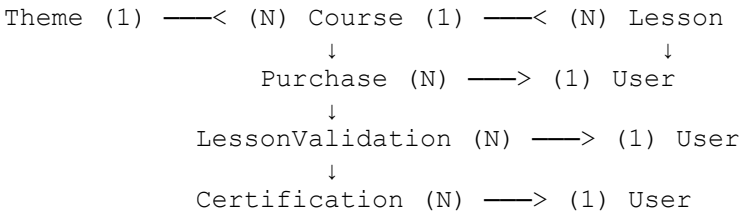
1. **README.md** : Instructions complètes d'installation et de lancement
2. **CONFIG_TEST.md** : Configuration de l'environnement de test
3. **TESTS_ANALYSIS.md** : Analyse détaillée des tests unitaires

4. DOCUMENTATION TECHNIQUE

Architecture de l'Application

4.1 Modèle Conceptuel de Données

Schéma Relationnel



Tables Principales

User

- `id` (PK, INT)
- `email` (VARCHAR, UNIQUE)
- `password` (VARCHAR, hashed bcrypt)
- `first_name`, `last_name` (VARCHAR)
- `roles` (JSON)
- `is_verified` (BOOLEAN)
- `verification_token` (VARCHAR, nullable)
- `created_at` (DATETIME IMMUTABLE)

Theme

- `id` (PK, INT)
- `name`, `slug` (VARCHAR, UNIQUE)
- `description` (TEXT)

Course

- `id` (PK, INT)
- `theme_id` (FK → Theme)
- `title`, `slug` (VARCHAR, UNIQUE)
- `description` (TEXT)
- `price` (DECIMAL 10,2)
- `created_at` (DATETIME IMMUTABLE)

Lesson

- `id` (PK, INT)
- `course_id` (FK → Course)
- `title`, `slug` (VARCHAR, UNIQUE)
- `content` (TEXT)
- `video_url` (VARCHAR, nullable)
- `price` (DECIMAL 10,2)
- `position` (INT)
- `created_at` (DATETIME IMMUTABLE)

Purchase

- `id` (PK, INT)
- `user_id` (FK → User)
- `course_id` (FK → Course, nullable)
- `lesson_id` (FK → Lesson, nullable)
- `amount` (DECIMAL 10,2)
- `stripe_payment_intent_id` (VARCHAR, UNIQUE)
- `status` (VARCHAR: pending, completed, failed)
- `purchased_at` (DATETIME IMMUTABLE)

LessonValidation

- `id` (PK, INT)
- `user_id` (FK → User)
- `lesson_id` (FK → Lesson)
- `validated_at` (DATETIME IMMUTABLE)
- **UNIQUE KEY** (`user_id`, `lesson_id`)

Certification

- `id` (PK, INT)
- `user_id` (FK → User)
- `course_id` (FK → Course)
- `certificate_number` (VARCHAR, UNIQUE)
- `obtained_at` (DATETIME IMMUTABLE)
- **UNIQUE KEY** (`user_id`, `course_id`)

4.2 Architecture du Code

Patterns Utilisés

1. **Action Controller Pattern**
    - Un contrôleur = une action (`__invoke()`)
    - Injection de dépendances via le constructeur
    - Exemple :

```
#[Route('/connexion', name: 'app_login')]
class LoginController extends AbstractController
{
    public function __construct(
        private AuthenticationUtils $authenticationUtils
    ) {}

    public function __invoke(): Response
    {
        // Logique unique
    }
}
```

2. **Service Layer Pattern**
    - Services readonly (immutables)
    - Logique métier isolée des contrôleurs
    - Exemple :

```
readonly class CourseService
{
    public function __construct(
        private CourseRepository $courseRepository,
        private ThemeRepository  $themeRepository
    ) {}

    public function getAllThemes(): array
    {
        return $this->themeRepository->findAllThemes();
    }
}
```

3. **Repository Pattern (Doctrine ORM)**
    - Séparation de la couche d'accès aux données
    - Requêtes personnalisées optimisées

```
class CourseRepository extends ServiceEntityRepository
{
    public function findBySlug(string $slug): ?Course
    {
        return $this->createQueryBuilder('c')
            ->leftJoin('c.theme', 't')
            ->addSelect('t')
            ->leftJoin('c.lessons', 'l')
            ->addSelect('l')
            ->where('c.slug = :slug')
            ->setParameter('slug', $slug)
            ->getQuery()
            ->getOneOrNullResult();}}
```

4. **DTO Pattern (Data Transfer Object)**
    - Validation des données entrantes
    - Séparation entités/formulaires

```
class RegistrationDTO
{
    #[Assert\NotBlank]
    #[Assert\Email]
    public ?string $email = null;

    #[Assert\Length(min: 8)]
    #[Assert\Regex(
        pattern: '/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/',
        message: 'Le mot de passe doit contenir majuscule, minuscule et
chiffre.'
    )]
    public ?string $password = null;
}
```

4.3 Composants E-Commerce

Intégration Stripe

Service de Paiement (`StripeService.php`)

```
readonly class StripeService
{
    public function __construct(
        private ParameterBagInterface $params,
        private UrlGeneratorInterface $urlGenerator,
        private PurchaseRepository    $purchaseRepository
    ) {
        Stripe::setApiKey($this->params->get('stripe.secret_key'));
    }

    public function createCheckoutSession(
        User $user,
        ?Course $course = null,
        ?Lesson $lesson = null
    ): Session {
        // Création de la session Stripe Checkout
        return Session::create([
            'payment_method_types' => ['card'],
            'line_items' => [/* ... */],
            'mode' => 'payment',
            'success_url' => $successUrl,
            'cancel_url' => $cancelUrl,
            'metadata' => $metadata,
        ]);
    }
}
```

Gestion des Webhooks (`WebhookHandlerService.php`)

- Vérification de la signature Stripe
- Traitement événement `checkout.session.completed`
- Création automatique des `Purchase` en base

4.4 Tests Unitaires et Fonctionnels

Couverture des Tests

Composant	Type	Fichier	Statut
Inscription	Unitaire	`RegistrationServiceTest.php`	✅
Vérification email	Unitaire	`VerificationServiceTest.php`	✅
Connexion	Fonctionnel	`LoginTest.php`	✅
Achats	Unitaire + Fonctionnel	`PurchaseServiceTest.php`, `PurchaseFlowTest.php`	✅
Repositories	Unitaire	`UserRepositoryTest.php`, `CourseRepositoryTest.php`, `PurchaseRepositoryTest.php`	✅

Exemple de Test Unitaire

```
class PurchaseServiceTest extends TestCase
{
    public function testCreatePurchaseForCourse(): void
    {
        // Arrange
        $user = $this->createUser();
        $course = $this->createCourse();

        // Act
        $purchase = $this->purchaseService->createPurchase(
            $user, 'pi_test_123', '50.00', $course, null
        );

        // Assert
        $this->assertInstanceOf(Purchase::class, $purchase);
        $this->assertEquals('50.00', $purchase->getAmount());
        $this->assertEquals('completed', $purchase->getStatus());
    }
}
```

Lancement des Tests

```
# Tous les tests
php bin/phpunit

# Tests unitaires uniquement
php bin/phpunit tests/Unit

# Tests fonctionnels uniquement
php bin/phpunit tests/Functional

# Avec rapport de couverture
php bin/phpunit --coverage-html coverage/
```

4.5 Sécurité

Mesures Implémentées

1. **Protection CSRF**
    - Token CSRF automatique sur tous les formulaires (Symfony)
    - Validation côté serveur
2. **Hashage des Mots de Passe**
    - Algorithme **bcrypt** (Symfony PasswordHasher)
    - Salage automatique
3. **Validation des Mots de Passe**
    - Minimum 8 caractères
    - Au moins 1 majuscule, 1 minuscule, 1 chiffre
    - Validation via Assert\Regex

```
#[Assert\Regex(
    pattern: '/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/',
    message: 'Le mot de passe doit contenir majuscule, minuscule et
chiffre.'
)]
public ?string $password = null;
```

4. **Gestion des Rôles**
    - `ROLE_USER` : accès utilisateur standard
    - `ROLE_ADMIN` : accès backoffice EasyAdmin
    - Hiérarchie : `ROLE_ADMIN` hérite de `ROLE_USER`

```
# config/packages/security.yaml
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/mon-compte, roles: ROLE_USER }
    - { path: ^/cursus/.*/acheter, roles: ROLE_USER }
```

5. **Activation par Email**
    - Token UUID v4 généré à l'inscription
    - Lien d'activation unique et temporaire

5. ANNEXES

5.1 Instructions d'Installation

Voir **README.md** du repository pour :

- Prérequis (PHP 8.2, Composer, MySQL, Node.js)
- Installation des dépendances
- Configuration de l'environnement (`.env.local`)
- Création de la base de données
- Chargement des fixtures
- Configuration Stripe
- Lancement de l'application

5.2 Comptes de Test

Créés automatiquement par les fixtures (`doctrine:fixtures:load`) :

Rôle	Email	Mot de passe
Administrateur	`admin@knowledge-learning.com`	`Admin123!`
Utilisateur	`jean.dupont@example.com`	`User123!`

5.3 Stack Technique

Backend

- **PHP 8.2+**
- **Symfony 7.4**
- **Doctrine ORM** (gestion BDD)
- **MySQL 8.0** / MariaDB
- **PHPUnit 9** (tests)

Frontend

- **Twig** (templates)
- **Bootstrap 5.3**
- **Stimulus** (interactions JavaScript)
- **Webpack Encore** (bundler)

E-Commerce

- **Stripe API** (paiements)
- **Stripe CLI** (webhooks locaux)

Outils

- **Composer** (dépendances PHP)
- **Yarn** (dépendances JS)
- **Mailpit** (emails de développement)

5.4 Périmètre Fonctionnel Réalisé

✅ Inscription avec activation email
✅ Connexion/Déconnexion
✅ Achat de cursus complets
✅ Achat de leçons individuelles
✅ Gestion des droits d'accès aux contenus
✅ Validation progressive des leçons
✅ Attribution automatique de certifications
✅ Backoffice administrateur (EasyAdmin)
✅ Dashboard utilisateur
✅ Gestion des achats et certifications

5.5 Points d'Amélioration Futurs

- Système de notation/commentaires sur les cours
- Lecteur vidéo intégré (Vimeo/YouTube)
- Export PDF des certifications
- Statistiques de progression avancées
- Mode "prévisualisation" des leçons gratuites
- Système de panier multi-produit

📧 Contact : mathieu.paquier85@gmail.com
🔗 GitHub : https://github.com/MathP-dev/knowledge_learning