

2025

CAHIER DES CHARGES

Knowledge Learning -

E-learning

Sommaire du projet Portfolio

1. Présentation du projet

2. Lien du repository GitHub

3. Identité Graphique

4. La conception et la réalisation

5. Model Physique de données

6. Tests unitaires (fonctionnels et de sécurité)

7. Recherches effectuées

8. Support de présentation de la conception

1. Présentation du projet

Knowledge Learning est une plateforme d'apprentissage en ligne permettant aux utilisateurs de consulter, acheter, et valider des leçons par thème, cursus et certification.

L'application est construite avec le framework Symfony.

2 Lien du repository GitHub

Liens : <https://github.com/kilianaBldr/Knowledge-Learning-CEF.git>

Le dépôt est public, entièrement commenté en anglais, et contient toutes les ressources nécessaires à l'exécution locale de l'application.

Prérequis techniques

Avant d'installer le projet assurez-vous d'avoir installé sur votre machine :

- PHP 8.1 ou supérieur
- Composer
- Symfony CLI (optionnel mais recommandé)
- MySQL
- XAMPP

Installation et lancement du projet

1. Cloner le dépôt :

```
git clone https://github.com/kilianaBldr/Knowledge-Learning-CEF.git
```

```
cd Knowledge-Learning-CEF
```

2. Installer les dépendances PHP :

```
composer install
```

3. Configurer l'environnement :

Dupliquer le fichier .env et adapter les accès à la base de données

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/
Knowledge_learning_cef?
serverVersion=8.0.32&charset=utf8mb4"
```

4. Créer la base de données :

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

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

```
php bin/console doctrine:fixtures:load
```

5. Lancer le serveur Symfony :

```
symfony server:start
```

6. Démarrer Maildev pour l'email de confirmation (pré-requis : Node.js) :

```
npm install -g maildev
```

```
maildev
```

7. Accéder au projet :

Ouvrir <http://127.0.0.1:8000> dans votre navigateur.

3. Identité graphique

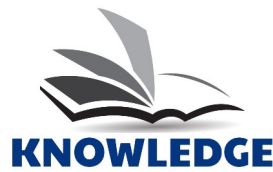
1. Police typographique

La police utilisée pour l'ensemble des supports est : **Comic Sans MS**

Cette police sera appliquée à tous les éléments textuels de l'application (Titre, Paragraphes, Formulaire).

2. Logo

Le logo officiel a été conçu spécialement pour l'occasion et doit figurer sur l'ensemble des pages principales du site (header).









3. Favicon

Le favicon ci-dessous est à utiliser pour représenter l'application dans l'onglet du navigateur :



4. Palette de couleurs

L'application utilise la palette suivante :

couleurs	Code HEX	Utilisation
	#f1f8fc	Arrière-plans, sections neutres
	#0074c7	Boutons principaux, liens
	#00497c	Boutons
	#384050	Textes
	#cd2c2e	Alertes, erreurs (avec parcimonie)
	#82b864	Validation, succès (avec parcimonie)

4. La conception et la réalisation

Le front-end

L'aspect graphique doit rester minimaliste. Le focus est mis sur la fonctionnalité et la navigation fluide :

- Présence de tous les liens fonctionnels entre routes
- Affichage correct des formulaires
- Simulations de pages avec titre et contenu (lorem ipsum)

L'objectif principal reste la démonstration de la logique applicative et non du design.

Le back-end

Le projet doit suivre une architecture orientée objet claire, en favorisant :

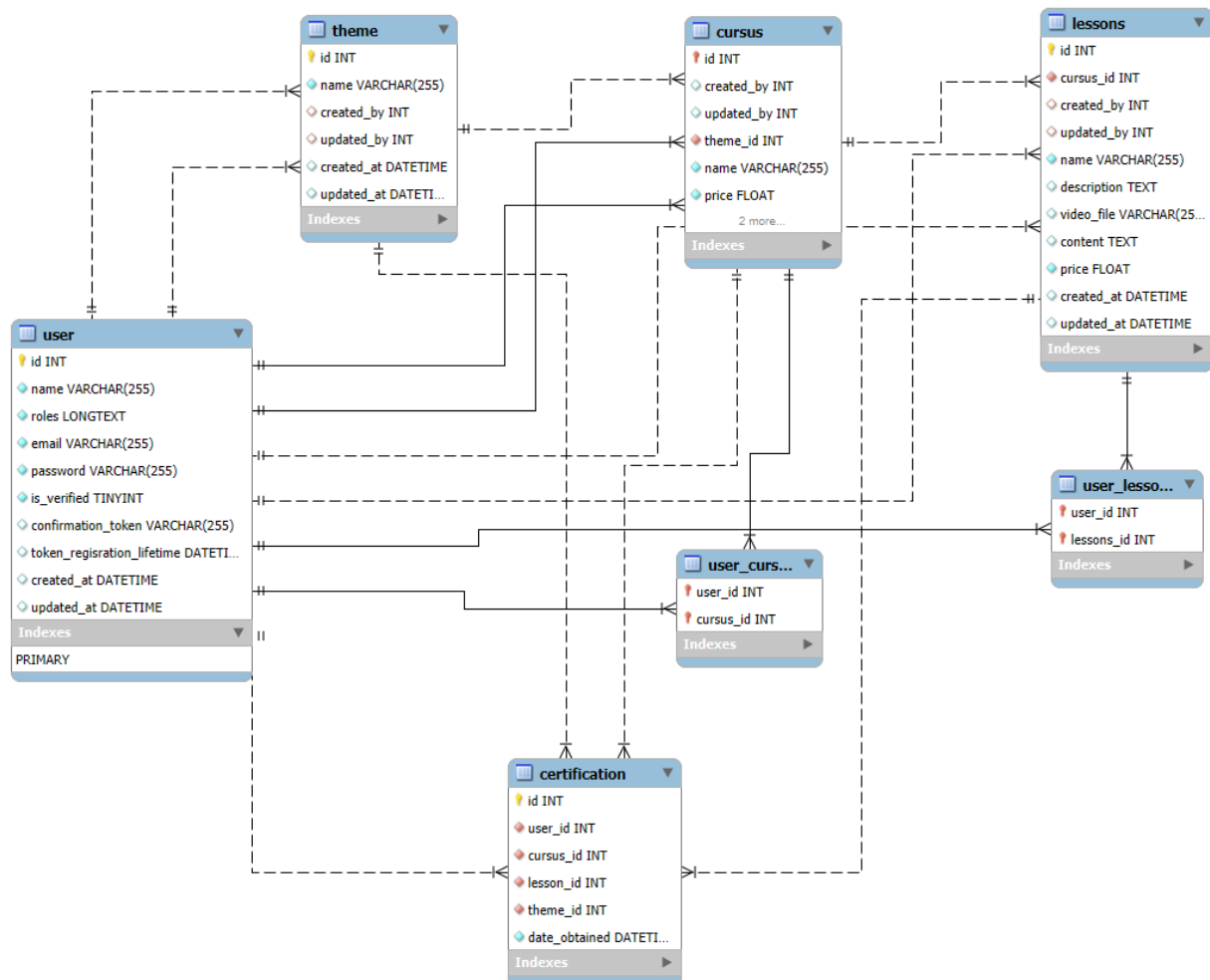
- La séparation des responsabilités (Controllers, Services, Repositories, etc.)
- L'utilisation de Design Patterns pour améliorer la maintenabilité
- L'organisation cohérente du code source (dossiers, namespaces, injections de dépendances)

Les tests

Les fonctionnalités suivantes doivent impérativement être testées unitairement :

- L'inscription d'un utilisateur
- La validation de l'e-mail via lien de confirmation
- La connexion d'un utilisateur
- L'achat de leçons ou de cursus
- Les composants d'accès aux données (tests fonctionnels et de sécurité sur les repositories)

5. Modèle physique de données



Objectif :

Représenter les relations clés entre les entités fonctionnelles de l'application .

. **Entités (avec attributs principaux) :**

. **1. Utilisateur**

- id (PK)
- name
- roles
- email
- password
- is_verified (TINYINT)

. **2. Thème**

- id (PK)
- name
- created_at (DATETIME)
- updated_at (DATETIME)
- created_by (FK → Utilisateur)
- updated_by (FK → Utilisateur)

3. Coursus

- id (PK)
- name
- price (FLOAT)
- theme_id (FK → Thème)
- created_by (FK → Utilisateur)
- updated_by (FK → Utilisateur)

4. Lessons

- id (PK)
- name
- description (TEXT)
- content (TEXT)
- video_file (VARCHAR255)
- price (FLOAT)
- cursus_id (FK → Coursus)
- created_by (FK → Utilisateur)
- updated_by (FK → Utilisateur)

7. Certification

- Id (PK)
- user_id (FK → Utilisateur)
- theme_id (FK → Thème)
- cursus_id (FK → Coursus, optionnel)
- lesson_id (FK → Leçon, optionnel)
- date_obtained (DATETIME)

8. User Lesson

- user_id (FK → Utilisateur)
- lessons_id (FK → Leçon)

9. User Coursus

- user_id (FK → Utilisateur)
- cursus_id (FK → Coursus)

Documentation Technique (code commenté)

Toutes les entités, services, contrôleurs et tests sont commenté selon une syntaxe inspirée de JSDoc en anglais, avec des blocs comme :

```
/**
 * Delete a user after CSRF token validation.
 *
 * @Route("/{id}", name="admin_user_delete", methods={"POST"})
 * @param Request $request
 * @param User $user
 * @param EntityManagerInterface $entityManager
 * @return Response
 */
```

6. Test unitaires (fonctionnels et de sécurité)

Des tests ont été écrits avec PHPUnit :

- Chaque test est commenté selon la norme JSDoc, en anglais.
- UserCreationTest.php - Vérifie la création d'un utilisateur (le hashing, rôles, etc)

```
class UserCreationTest extends KernelTestCase
{
    /**
     * @test
     * @description Verifies that a user object is created with valid data, roles are set,
     * and the password is securely hashed (not stored in plain text).
     */
    public function testUserIsCreatedCorrectly(): void
    {
        // Boot the Symfony kernel and get container
        self::bootKernel();
        $container = static::getContainer();

        // Get the password hasher from the container
        /** @var UserPasswordHasherInterface $hasher */
        $hasher = $container->get(UserPasswordHasherInterface::class);

        // Create and set up user data
        $user = new User();
        $user->setEmail('test@user.com');
        $user->setName('Test Unitaire');
        $user->setRoles(['ROLE_USER']);

        // Hash the password and assign it
        $hashedPassword = $hasher->hashPassword($user, 'testpassword');
        $user->setPassword($hashedPassword);

        // Functional assertions
        $this->assertSame('test@user.com', $user->getEmail(), 'Email should match the expected value.');
```

Vérifie la création d'un utilisateur avec e-mail, nom, rôle et mot de passe hashé. Assure que le mot de passe n'est pas stocké en clair.

- UserLoginTest.php – Teste l'authentification réussie et échouée

```
/**
 * @test
 * @description Tests login with correct username and password.
 * Expects a successful response and username to be displayed after login.
 */
public function testUserLogin(): void
{
    $crawler = $this->client->request('GET', '/login');

    $form = $crawler->selectButton('Se connecter')->form([
        'name' => 'login',
        'password' => 'password123',
    ]);

    $this->client->submit($form);
    $this->client->followRedirect();

    $this->assertResponseIsSuccessful();
    $this->assertSelectorTextContains('body', 'login');
}
```

Vérifie la connexion avec des identifiants valides. Le test s'assure que l'utilisateur est bien redirigé après la connexion et que la page contient son nom.

- PaymentTest.php – Simule l'achat et vérifie l'enregistrement des leçons

```
/**
 * @test
 * @description Verifies that a verified user can successfully purchase a lesson.
 * This simulates a real purchase process by creating related entities (theme, cursus, lesson),
 * attaching the lesson to the user, and confirming the lesson is saved correctly.
 */
public function testLessonPurchase(): void
{
    // Create user
    $user = new User();
    $user->setName('testbuyer');
    $user->setEmail('testbuyer@test.com');
    $user->setPassword('password'); // This would usually be hashed
    $user->setIsVerified(true); // Required for purchase

    // Create theme
    $theme = new Theme();
    $theme->setName('Test Theme');

    // Create cursus
    $cursus = new Cursus();
    $cursus->setName('Test Cursus')
        ->setPrice(0)
        ->setTheme($theme);

    // Create Lesson
    $lesson = new Lessons();
    $lesson->setName('Test Lesson')
        ->setPrice(10)
        ->setDescription('Test description')
        ->setVideoFile('videos/test.mp4')
        ->setContent('Mock content for testing')
        ->setCursus($cursus);
}
```

```
        ->setCursus($cursus);

    // Persist data
    $this->entityManager->persist($theme);
    $this->entityManager->persist($cursus);
    $this->entityManager->persist($lesson);
    $this->entityManager->persist($user);
    $this->entityManager->flush();

    // Simulate purchase
    $user->addPurchasedLesson($lesson);
    $this->entityManager->persist($user);
    $this->entityManager->flush();

    // Assert the lesson was purchased
    $this->assertTrue(
        $user->getPurchasedLessons()->contains($lesson),
        'The user should have the lesson in their purchased lessons list.'
    );
}
```

Simule un achat de leçon par un utilisateur vérifié. Ce test vérifie que les entités nécessaires (themes, cursus, leçons) sont créés et liés correctement, et que la leçon peut être achetée et associée à l'utilisateur .

Et vérification que celle-ci est bien enregistrée parmi les leçons achetées de l'utilisateur.

- UserRepositoryTest.php – Teste les accès aux données (requêtes spécifiques)

```
/**
 * @test
 * @description Ensures a user can be persisted and retrieved by name or email.
 */
public function testFindUserByEmail(): void
{
    // Create user
    $user = new User();
    $user->setEmail('repo@test.com');
    $user->setName('RepositoryTestUser');
    $user->setPassword('hashed-password');
    $user->setIsVerified(true);

    // Persist user
    $em = static::getContainer()->get('doctrine')->getManager();
    $em->persist($user);
    $em->flush();

    // Retrieve user by email
    $fetchedReader = $this->userRepository->findOneBy(['email' => 'repo@test.com']);

    // Assert user is correctly fetched
    $this->assertNotNull($fetchedReader, 'User should be found by email.');
```

```
$this->assertSame('repo@test.com', $fetchedReader->getEmail());
$this->assertSame('RepositoryTestUser', $fetchedReader->getName());

// Cleanup
$em->remove($fetchedReader);
$em->flush();
}
```

Vérifie que l'on peut correctement enregistrer un utilisateur en base et le retrouver à l'aide de son e-mail via userRepository.

```
Testing C:\Users\kilia\Knowledge-Learning-CEF\tests
..... 5 / 5 (100%)

Time: 00:35.411, Memory: 40.00 MB

OK (5 tests, 16 assertions)
PS C:\Users\kilia\Knowledge-Learning-CEF> 
```

✓ Tous les tests validés avec succès :

Ce résultat console confirme que les 5 tests unitaires ont été exécutés avec succès, totalisant 16 assertions validées.

Cela garantit le bon fonctionnement des fonctionnalités critiques de l'application (création utilisateur, connexion, paiement, accès aux données).

7. Recherches effectuées

- Sécurité avec Symfony :

Étude de la documentation officielle :

<https://symfony.com/doc/current/security.html>

- Vérification e-mail avec Maildev

Documentation lue sur <https://github.com/maildev/maildev>

Test locaux sur <http://localhost:1080> pour l'émulation des mails de confirmation

- Tests automatisés avec PHPUnit :

Consultation de tutoriels sur SymfonyCasts :

<https://symfonycasts.com/screencast/phpunit>

- Stripe et paiements :

Documentation officielle : <https://stripe.com/docs>

- Composants Symfony :

Documentation Symfony + Tutos vidéos Youtube (chaîne : Grafikart, Code With Dary)

8. Support de présentation de la conception

Un diaporama PDF (Slides) est fourni et contient :

- La justification de la base de données via Merise
- Mise en place : Des commandes Doctrine, fixtures, migrations
- Accès aux données : Repository, Entity Manager, injection
- Composants e-commerce : Panier, Stripe, session, paiement
- Intégrations des tests unitaires avec PHPUnit
- Recherches effectuées : Stripe, sécurité Symfony, gestion des fixtures et tests