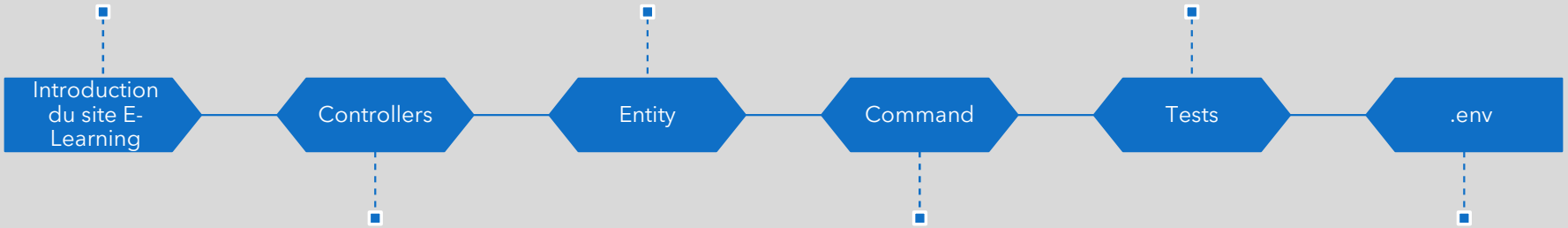

KNOWLEDGE
E-Learning

Sommaire


Introduction du site E-Learning
Controllers
Entity
Command
Tests
.env

INTRODUCTION DU SITE E-LEARNING

- D'abord vous devez ajouter ceci au fichier .env car github n'accepte pas les clé API:###> symfony/mailer ###
- MAILER_DSN=sendgrid://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@default
- Les lignes de commandes que j'ai eu a exécuter :
- composer create-project symfony/skeleton e-learning (pour la création du site)
- composer require symfony/maker-bundle --dev
- composer require doctrine/orm
- composer require security
- composer require symfony/form
- php bin/console make:entity(pour créer des entité)
- php bin/console make:migration
- composer require symfony/asset
- composer require validator twig-bundle
- php bin/console doctrine:fixtures:load --env=test
- composer require form orm security
- symfony console make:registration (pour créer un compte utilisateurs)
- composer require symfonycasts/verify-email-bundle
- composer require symfony/mailer
- symfony console make:auth (pour la connexion)
- php bin/console doctrine:fixtures:load (pour stocker des composants dans ma base)
- php bin/console doctrine:migrations:migrate --env=test
- php bin/console app:create-admin(pour créer un compte administrateur)
- php bin/console doctrine:database:create --env=test
- php vendor/bin/phpunit tests/Controller/RegistrationControllerTest.php(pour tester la logique d'inscription)
- php vendor/bin/phpunit tests/Controller/SecurityControllerTest.php(pour tester la logique de connexion)
- php vendor/bin/phpunit tests/Controller/AcheterControllerTest.php(pour tester la logique pour l'achat d'une leçon)

APP FIXTURES

- Dans cette classe AppFixtures, la méthode load est utilisée pour charger les données.
- Elle crée des données de thème (Theme), qui ont des images et des icônes associées.
- Pour chaque thème, elle crée des cursus (Cursus) avec des leçons (Lesson) associées.
- Chaque leçon a un prix, une description Lorem Ipsum (générée) et une URL vidéo YouTube associée.
- Ces données simulées représentent des offres de cours dans divers domaines tels que la musique, l'informatique, le jardinage et la cuisine, avec des informations factices pour les prix, les descriptions et les vidéos associées.
- La méthode generateLoremIpsum est utilisée pour générer un texte Lorem Ipsum en fonction du nombre de paragraphes spécifié.

CONTROLLER

- **1. RegistrationController**
- Ce contrôleur gère l'inscription des utilisateurs.

- **register**:
  - Crée un nouveau formulaire d'inscription pour un utilisateur.
  - Si le formulaire est soumis et valide, il hache le mot de passe de l'utilisateur, assigne un rôle, et persiste l'utilisateur dans la base de données.
  - Envoie un email de confirmation pour vérifier l'adresse email.
  - Redirige vers la page d'accueil après l'inscription.
- **verifyUserEmail**:
  - Vérifie le lien de confirmation de l'email.
  - Marque l'utilisateur comme vérifié si le lien est valide, sinon affiche un message d'erreur.
  - Redirige vers la page d'inscription après vérification.

CONTROLLER

- **2. SecurityController**
- Ce contrôleur gère la sécurité de l'application, principalement l'authentification.
- **login**:
  - Affiche la page de connexion.
  - Récupère et affiche les erreurs de connexion.
- **logout**:
  - Gère la déconnexion des utilisateurs. Cette méthode est interceptée par le pare-feu de sécurité.
- **3. PurchaseController**
- Ce contrôleur gère les achats des utilisateurs.
- **show**:
  - Affiche les achats effectués par l'utilisateur connecté.

CONTROLLER

- **4. LessonController**
- Ce contrôleur gère la validation des leçons et la création de certifications.
- **validateLesson**:
  - Valide une leçon spécifique.
  - Vérifie si toutes les leçons du cursus sont validées et, si oui, crée une certification pour l'utilisateur.

- **5. HomeController**
- Ce contrôleur gère la page d'accueil.
- **index**:
  - Affiche tous les thèmes disponibles sur la page d'accueil.

CONTROLLER

- **6. DefaultController**
- Ce contrôleur gère les pages par défaut comme la découverte des cursus.
- **discoverCursus**:
  - Affiche les détails d'un cursus spécifique selon le thème choisi.
- **7. CurcusController**
- Ce contrôleur gère les opérations liées à l'achat d'un cursus y compris les paiements via Stripe.
- **index**:
  - Affiche la page de paiement Stripe pour un cursus spécifique.
  - Redirige les utilisateurs non connectés ou non vérifiés vers des pages spécifiés.
- **createCharge**:
  - Gère la création de charges pour les paiements Stripe.
  - Vérifie si l'utilisateur a déjà acheté le cursus.
  - Enregistre l'achat dans la base de données.

CONTROLLER

- **8. CertificationController**
- Ce contrôleur gère l'affichage des certifications des utilisateurs.
- **index**:
  - Affiche toutes les certifications obtenues par l'utilisateur connecté.
- **9. ActivationController**
- Ce contrôleur gère l'activation des comptes utilisateurs et contient les routes pour d'autres pages tel que le contoller pour l'affichage d'un cursus ou d'une leçon.
- **curcusPage**:
  - Affiche les détails d'un cursus spécifique.
  - Redirige si le cursus ou le thème associé n'est pas trouvé.
- **lessonPage**:
  - Affiche les détails d'une leçon spécifique.
- **activationPage**:
  - Affiche la page d'activation du compte.
- **optionPage**:
  - Affiche la page des options de connexion.

CONTROLLER

- **10. AcheterController**
- Ce contrôleur gère l'achat de leçons.

- **index**:
  - Affiche la page de paiement Stripe pour une leçon spécifique.
  - Redirige les utilisateurs non connectés ou non vérifiés.
- **createCharge**:
  - Gère la création de charges pour les paiements Stripe.
  - Vérifie si l'utilisateur a déjà acheté le cursus ou la leçon spécifique.
  - Enregistre l'achat dans la base de données.

ENTITY

1. **Certification.php:**

- La classe Certification représente une certification attribuée à un utilisateur pour un cursus spécifique. Elle est définie par les propriétés suivantes :

- id: Identifiant unique de la certification.
- UpdatedAt: Date de la dernière mise à jour.
- createdAt: Date de création.
- createdBy: Utilisateur ayant créé la certification.
- updatedBy: Utilisateur ayant mis à jour la certification.
- user: Utilisateur associé à la certification.
- cursus: Cursus auquel est associée la certification.
- Méthodes :
- getId(), getUser(), setUser(), getCursus(), setCursus(), getCreatedAt(), setCreatedAt(), getUpdatedAt(), setUpdatedAt(), getCreatedBy(), setCreatedBy(), getUpdatedBy(), setUpdatedBy(): méthodes getter et setter pour les propriétés.

ENTITY

2. Cursus.php:

- La classe Cursus représente un cursus d'apprentissage. Elle comprend :
- id: Identifiant unique du cursus.
- name: Nom du cursus.
- price: Prix du cursus.
- createdAt: Date de création.
- UpdatedAt: Date de la dernière mise à jour.
- createdBy: Utilisateur ayant créé le cursus.
- updatedBy: Utilisateur ayant mis à jour le cursus.
- theme: Thème associé au cursus.
- lessons, purchases, certifications: Collections de leçons, achats, et certifications associées au cursus.
- Méthodes :
- getId(), getName(), setName(), getPrice(), setPrice(), getCreatedAt(), setCreatedAt(), getUpdatedAt(), setUpdatedAt(), getCreatedBy(), setCreatedBy(), getUpdatedBy(), setUpdatedBy(), getTheme(), setTheme(), getLessons(), addLesson(), removeLesson(), getPurchases(), addPurchase(), removePurchase(), getCertifications(), addCertification(), removeCertification(): méthodes getter et setter pour les propriétés.

ENTITY

3. **Lesson.php:**

- La classe Lesson représente une leçon d'un cursus. Elle est définie par :
- id: Identifiant unique de la leçon.
- name: Nom de la leçon.
- price: Prix de la leçon.
- youtubeUrl: URL de la vidéo YouTube de la leçon.
- description: Description de la leçon.
- isValidated: Statut de validation de la leçon.
- createdAt: Date de création.
- UpdatedAt: Date de la dernière mise à jour.
- createdBy: Utilisateur ayant créé la leçon.
- updatedBy: Utilisateur ayant mis à jour la leçon.
- cursus: Cursus auquel appartient la leçon.
- purchases: Collection d'achats associés à la leçon.
- Méthodes :
- getId(), getName(), setName(), getPrice(), setPrice(), getCreatedAt(), setCreatedAt(), getUpdatedAt(), setUpdatedAt(), getCreatedBy(), setCreatedBy(), getUpdatedBy(), setUpdatedBy(), getCursus(), setCursus(), getYoutubeUrl(), setYoutubeUrl(), getDescription(), setDescription(), getIsValidated(), setIsValidated(), getPurchases(), addPurchase(), removePurchase(): méthodes getter et setter pour les propriétés.

ENTITY

4. **Purchase.php:**

- La classe Purchase représente un achat effectué par un utilisateur pour une leçon ou un cursus. Elle inclut :

- id: Identifiant unique de l'achat.
- createdAt: Date de création de l'achat.
- UpdatedAt: Date de la dernière mise à jour.
- createdBy: Utilisateur ayant créé l'achat.
- updatedBy: Utilisateur ayant mis à jour l'achat.
- user: Utilisateur ayant effectué l'achat.
- lesson: Leçon associée à l'achat.
- cursus: Cursus associé à l'achat.
- Méthodes :

- getId(), getUser(), setUser(), getLesson(), setLesson(), getCursus(), setCursus(), getCreatedAt(), setCreatedAt(), getUpdatedAt(), setUpdatedAt(), getCreatedBy(), setCreatedBy(), getUpdatedBy(), setUpdatedBy(): méthodes getter et setter pour les propriétés.

ENTITY

5. **Theme.php**:

- La classe Theme représente le thème d'un cursus. Elle est caractérisée par :
- id: Identifiant unique du thème.
- name: Nom du thème.
- image: Image associée au thème.
- icon: Icône associée au thème.
- createdAt: Date de création.
- UpdatedAt: Date de la dernière mise à jour.
- createdBy: Utilisateur ayant créé le thème.
- updatedBy: Utilisateur ayant mis à jour le thème.
- cursuses: Collection de cursuses associés au thème.
- Méthodes :
- getId(), getName(), setName(), getImage(), setImage(), getIcon(), setIcon(), getCreatedAt(), setCreatedAt(), getUpdatedAt(), setUpdatedAt(), getCreatedBy(), setCreatedBy(), getUpdatedBy(), setUpdatedBy(), getCursuses(), addCursus(), removeCursus(): méthodes getter et setter pour les propriétés.

ENTITY

6. **User.php:**

- La classe User représente un utilisateur de l'application. Elle comprend :
- id: Identifiant unique de l'utilisateur.
- username: Nom d'utilisateur.
- roles: Rôles de l'utilisateur.
- password: Mot de passe (haché).
- email: Adresse email.
- resetToken: Jeton de réinitialisation du mot de passe.
- isVerified: Statut de vérification de l'utilisateur.
- createdBy: Utilisateur ayant créé cet utilisateur.
- updatedBy: Utilisateur ayant mis à jour cet utilisateur.
- createdAt: Date de création.
- UpdatedAt: Date de la dernière mise à jour.
- purchases: Collection d'achats effectués par l'utilisateur.
- certifications: Collection de certifications obtenues par l'utilisateur.
- Méthodes :
- getId(), getUsername(), setUsername(), getEmail(), setEmail(), getCreatedAt(), setCreatedAt(), getUpdatedAt(), setUpdatedAt(), getCreatedBy(), setCreatedBy(), getUpdatedBy(), setUpdatedBy(), getUserIdentifier(), getRoles(), setRoles(): méthodes getter et setter pour les propriétés.

COMMAND

- La classe CreateAdminCommand étend la classe Command de Symfony, ce qui permet de créer une nouvelle commande CLI.
- Trois dépendances sont injectées dans le constructeur de la classe : EntityManagerInterface pour interagir avec la base de données, UserPasswordHasherInterface pour hasher les mots de passe utilisateur, et EmailVerifier pour vérifier les adresses e-mail.
- La méthode configure() configure la description de la commande.
- La méthode execute() est exécutée lorsqu'on lance la commande. Elle récupère les informations de l'utilisateur (nom d'utilisateur, e-mail et mot de passe) à l'aide de la classe Question de Symfony.
- Un nouvel utilisateur est créé avec les informations fournies, et il est persisté dans la base de données.
- Le mot de passe de l'utilisateur est hashé avant d'être stocké dans la base de données.
- Un e-mail de confirmation est envoyé à l'utilisateur nouvellement créé à l'aide du service EmailVerifier.
- Un message de succès est affiché à la sortie de la commande.

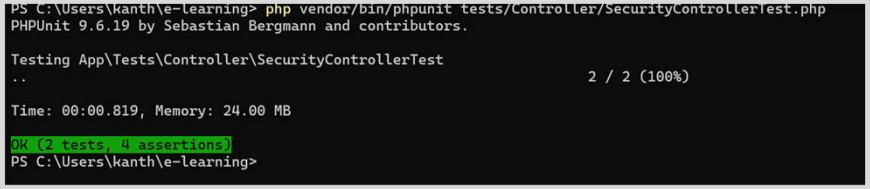
TESTS

```
PS C:\Users\kanth\e-learning> php vendor/bin/phpunit tests/Controller/RegistrationControllerTest.php
PHPUnit 9.6.19 by Sebastian Bergmann and contributors.

Testing App\Tests\Controller\RegistrationControllerTest
..                                                                  2 / 2 (100%)

Time: 00:00.659, Memory: 30.00 MB

OK (2 tests, 4 assertions)
PS C:\Users\kanth\e-learning>
```

- testRegisterPage(): Ce test vérifie si la page d'inscription est accessible en envoyant une requête GET à l'URL /register. Ensuite, il vérifie si la réponse est réussie (code HTTP 2xx) et si le titre de la page contient le texte "S'inscrire".

- testRegisterUser(): Ce test simule l'inscription d'un utilisateur en envoyant une requête GET à l'URL /register pour récupérer le formulaire d'inscription. Ensuite, il remplit le formulaire avec des données d'utilisateur simulées, soumet le formulaire et vérifie que la réponse a un code de statut 200 (OK).

TESTS



- testLoginPageIsAccessible(): Ce test vérifie si la page de connexion est accessible en envoyant une requête GET à l'URL /login. Ensuite, il vérifie si la réponse est réussie (code HTTP 2xx) et si le titre de la page contient le texte "Se connecter".

- testLoginWithInvalidCredentials(): Ce test simule une tentative de connexion avec des identifiants invalides. Il envoie d'abord une requête GET à l'URL /login pour récupérer le formulaire de connexion. Ensuite, il remplit le formulaire avec des identifiants invalides, soumet le formulaire et vérifie que la réponse redirige vers la même page de connexion. Enfin, il suit la redirection.

TESTS POUR ACHETER UN CURSUS

- testIndex(): Ce test vérifie si la route /curcus redirige correctement en vérifiant si la réponse a un code de statut HTTP 302 (redirection). Cela peut être utilisé pour s'assurer que la redirection vers la page d'accueil ou une autre page appropriée se produit comme prévu.

- testCreateChargeSuccess(): Ce test simule la création d'une charge (paiement) réussie en effectuant une requête POST vers la route /cursus/create-charge avec des données de paiement valides. Avant cela, il crée un utilisateur et un cursus enregistrés dans la base de données, puis il envoie les données de paiement. Après cela, il vérifie si la réponse a un code de statut HTTP 302 (redirection) et s'assure également que l'achat a été enregistré dans la base de données en vérifiant si une entrée correspondante existe pour l'utilisateur et le cursus dans la table des achats.

.ENV

- Structure des fichiers .env: Le fichier décrit la structure des fichiers .env utilisés dans différentes situations. Il explique que plusieurs fichiers peuvent être chargés selon l'environnement de l'application.
- Variables d'environnement:
- APP_ENV: Indique l'environnement de l'application. Dans cet exemple, il est défini sur "dev" pour le développement.
- APP_SECRET: Un secret utilisé par Symfony pour la sécurité et la cryptographie.
- Configuration de la base de données:versionnés.
- DATABASE_URL: il s'agit d'une base de données MySQL.
- Configuration du service de messagerie:
- MAILER_DSN: La configuration de connexion au service de messagerie, dans ce cas, SendGrid.
- Configuration de Stripe:
- STRIPE_KEY et STRIPE_SECRET: Les clés d'API de Stripe pour l'intégration de paiement.
- En résumé, ce fichier définit diverses variables d'environnement utilisées par l'application Symfony, notamment pour la configuration de la base de données, du service de messagerie et de l'intégration de Stripe. Il fournit également des informations sur la gestion des fichiers .env dans différents environnements et met en garde contre la définition de secrets de production dans des fichiers