**KnowLedge**

1. **Présentation de l’entreprise.**
2. **Mise en place de méthode pour la conception de la base de données.**
3. **Mise en place de la base de données.**
4. **Présentation des composants.**
5. **Présentation des tests.**
6. **Méthode de recherche menant à bien le projet**
7. ***Présentation de l’entreprise***

**Knowledge** est une société éditrice réputée pour la vente de ses supports de formations vendus en librairie.

Cette société propose également des formations sous forme de livres, ou de kits dans le domaine de la **musique** de **l’informatique**, du **jardinage**, et de la **cuisine,** maintenant ils souhaitent développer leur société en proposant une plateforme d’E-learning, proposants des formations en ligne, c’est le projet **Knowledge Learning**.

La plateforme d’E-learning permettra également de se former sur ces thèmes pour la bêta du projet, tout ça à distance.

Pour chacun de ses thèmes, il existe au minimum un cursus d’initiation, et deux leçons par thèmes, allant de 16€ à 60€

***I.2 Cahier des charges.***

Pour réaliser ce projet nous avons dû mettre en place une stratégie de développement et de réalisation du projet.

Pour réaliser la plateforme partie back-end, un **diagramme** **de** **classe** à étais réaliser au départ du projet afin d’aider à la **réalisation** des **composants** et de leurs **fonctions**.

Nous avons éplucher chaque **fonctions** et **composants** à réaliser pour désigner leur **rôle** et les **méthodes** **à** **adopter** pour les réaliser.

Par exemple : **L’Utilisateur** de la plateforme doit avoir **vérifié** son adresse **Email** afin d’accéder à nos services,

Il peut-être soit un **utilisateur** **classique**, **vérifié** mais **non** **étudiant**, mais il peut également être un **utilisateur** **étudiant** en **informatique** par exemple, pour savoir le **domaine** que **l’étudiant** à choisis, nous avons mis en place un **système** **de** **rôle**, cela consiste à attribuer un **rôle** **spécifique** à un **domaine** **spécifique**, par exemple pour le **domaine** de la **cuisine**, si **l’utilisateur** décide **d’acheter** cette **formation**, **l’utilisateur** ce verra recevoir le **rôle** de **Cuisinier**, il sera donc **autorisé** à accéder à **l’espace** **étudiant** lui permettant d’accéder à son apprentissage, une fois qu’il à **fini** **sa** **formation** **intégralement**, il se verra recevoir la **certification** correspondante.

***I.3 Environnement humain.***

Le développeur choisis pour ce projet à pris en compte les attentes des utilisateurs sur cette plateforme, il s’est penché sur le back-end de ce projet en gardant un espace front-end plutôt simple.

Les utilisateurs ont besoin d’un site simple d’accès et d’un espace dédié à l’apprentissage afin de ne pas se perdre lors de l’utilisation du site.

***I.4 Environnement technique.***

La société à demander au développeur de choisir le langage de programmation php.

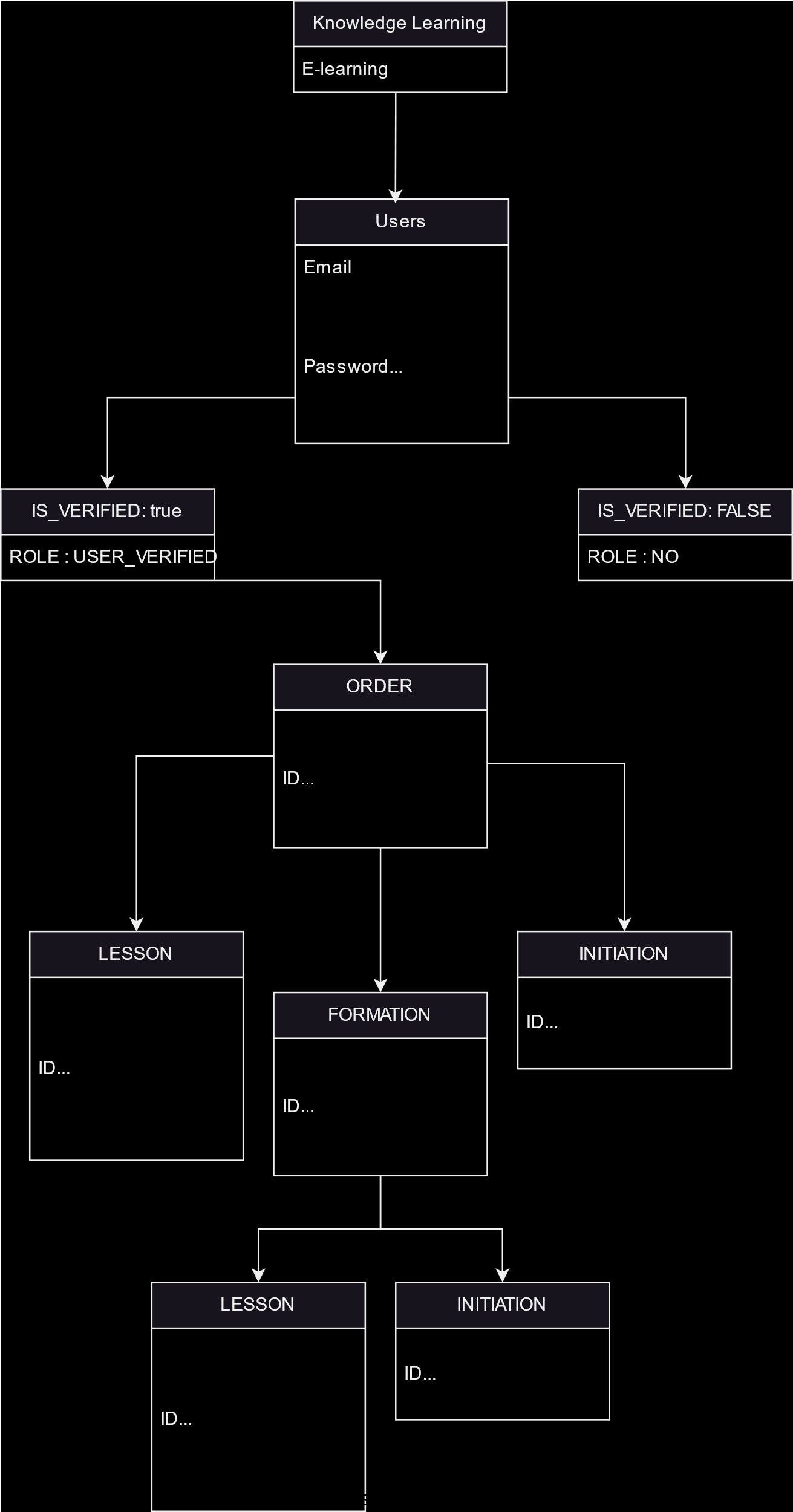
Le développeur à choisis le framework Symfony pour réaliser ce projet, simple d’utilisation, avec une documentation complète, et également des modules qui simplifie le développement.

Pour la conception de la base de données il à choisis une base de données MySQL en innoDB, avec un interclassement UTF8MB3\_UNICODE\_CI afin de limiter les erreurs qui peuvent-être liées à la base de données.

1. ***Méthode de conception de la base de données.***

Pour la conception de la base de données un diagramme à étais réaliser en amont afin de faciliter le développement de la base de données au développeur.

Voici le diagramme :



1. ***Mise en place de la base de données.***

Pour la mise en place de la base de données le développeur s’est appuyé sur le diagramme réalisé, afin de voir comment il devra s’y prendre pour qu’il y est les bonnes fonctions dans les bonnes tables, et leurs rôles.

1. ***Présentation des composants***

Une fois la réalisation de la base de données faites, le développeur à du réaliser les composants adaptés pour les utilisateurs, pour ce faire il à du créer des Entity (entité) correspondant aux noms attribués dans la base de données, avec les mêmes colonnes.

*(Exemple : Si la table User comporte la colonne ID, il faut que cette donnée soit envoyer par le projet depuis l’entité)*

Puis le développeur réalise des Controller (Contrôleur) afin de contrôlé les templates (Vue), mais également de contrôler les entités dans les templates et leurs fonction.

*(Exemple : Si l’entité contient une formation une id avec un titre, le contrôleur peut choisir d’afficher le titre correspondant à la formation grâce à l’id qui est automatiquement reliée au titre)*

1. ***Présentation des tests***

Afin de garder un développement stable des tests ont étais réaliser.

Une image contenant texte, Police, capture d’écran, ligne

Le contenu généré par l’IA peut être incorrect.

Tous les tests réaliser passe, j’ai réaliser tous les tests pour les entités, et le test pour s’inscrire sur le site web.

Les tests pour les Entité servent à tester que toutes les fonctionnalités conçues pour l’entité testé, j’ai fait en sorte que ce qui peut-être testé, soit testé et commenté.

Le test de sécurité sert à essayer l’inscription dans les deux cas possibles :

Si le test est réalisé avec des données valide, l’inscription sera réalisée.

Si le test est réalisé avec de mauvaise données, l’inscription sera refusée.

Entités :

FormationTest.php

1. **Initialisation des valeurs** : Les propriétés de la classe Formation (comme id, title, desc, image, createdAt, etc.) sont testées pour s'assurer qu'elles prennent les bonnes valeurs par défaut.
2. **Accesseurs pour title** : Le getter et le setter pour le titre (title) sont testés.
3. **Accesseurs pour desc** : Le getter et le setter pour la description (desc) sont testés.
4. **Gestion de l'image** : Le getter et le setter pour l'image (image) sont testés, y compris le cas où l'image peut être définie à null.
5. **Gestion des leçons (Lesson)** : Les méthodes d'ajout et de suppression des leçons (addLesson et removeLesson) sont testées, ainsi que l'intégrité de la collection des leçons.
6. **Gestion de updatedAt** : La méthode setUpdatedAt() est testée pour vérifier que la propriété updatedAt est bien définie.
7. **Méthode getRoleForUser** : Cette méthode est testée pour s'assurer que le rôle retourné dépend de l'id de la formation.

InitiationTest.php

* 1. **testGetTitle** : Le getter pour le titre (title) est testé pour s'assurer qu'il retourne null par défaut.
  2. **testGetType** : Le getter pour le type (type) est testé pour s'assurer qu'il retourne null par défaut.
  3. **testGetPrice** : Le getter pour le prix (price) est testé pour s'assurer qu'il retourne null par défaut.
  4. **testIdIsNullInitially** : Le getter pour l'identifiant (id) est testé pour s'assurer qu'il est null au départ.
  5. **tearDown** : Après chaque test, l'objet initiation est réinitialisé à null pour préparer l'environnement pour le test suivant.

LessonTest.php

1. **testInitialValues** : Vérifie que les propriétés de la classe Lesson sont initialisées à null par défaut, à l'exception de createdAt qui doit être un objet de type DateTimeImmutable.
2. **testTitleAccessors** : Teste le getter et setter pour la propriété title, en s'assurant que la valeur de title peut être définie et récupérée correctement.
3. **testTypeAccessors** : Teste le getter et setter pour la propriété type. Vérifie que la valeur de type peut être modifiée et que null est un état valide pour cette propriété.
4. **testImageAccessors** : Teste le getter et setter pour la propriété image, en s'assurant que l'image est correctement définie et récupérée.
5. **testPriceAccessors** : Teste le getter et setter pour la propriété price, en vérifiant que la valeur du prix est bien définie et récupérée.

6. **testFormationAccessors** : Teste le getter et setter pour la relation avec Formation. Vérifie que l'objet Formation peut être correctement associé à la leçon, et que la valeur null est acceptable.

7. **testUpdatedAtSetter** : Teste le setter pour la propriété updatedAt, en s'assurant que cette propriété est bien mise à jour et que le format de l'objet est un DateTime. Il vérifie aussi que le timestamp de updatedAt est proche du moment actuel (moins de 2 secondes de différence).

**8. testFluentInterfaces** : Teste si les méthodes setFormation() et setType() retournent correctement l'objet Lesson pour permettre un usage fluide (méthodes chaînées).

OrderTest.php

1. **testInitialValues** :

* Vérifie que les propriétés id et title sont initialement null.
* Vérifie que la propriété purchaseDate est une instance de DateTimeImmutable et qu'elle n'est pas null.
* Vérifie que la méthode getPurcharser() retourne une instance de Collection vide.

1. **testTitleAccessors** :

* Teste le getter et le setter de la propriété title. Il modifie la propriété via la réflexion et vérifie ensuite que la valeur de title est correctement définie et récupérée.

1. **testArticleProperty** :

* Teste la propriété article. Il utilise également la réflexion pour définir une valeur d'article (id), puis vérifie que la valeur est correctement assignée et récupérée.

1. **testPurchaserProperty** :

* Teste la propriété Purchaser. Le test utilise la réflexion pour définir le nom de l'acheteur comme étant "John Doe", puis vérifie que la valeur est correctement définie et récupérée.

1. **testUserCollection** :

* Teste la collection d'acheteurs (propriété purcharser). Ce test vérifie que la collection est bien de type Collection, puis ajoute un objet User fictif et vérifie qu'il est bien ajouté à la collection.

1. **testPurchaseDateImmutability** :

* Vérifie l'immuabilité de la propriété purchaseDate. Il prend la date d'achat originale, attend brièvement, puis s'assure que la date n'a pas changé (car elle devrait être immuable).

UserTest.php

* 1. **testInitialValues** :
* Vérifie les valeurs initiales de la classe User :
  + L'ID est null.
  + L'email, le nom d'utilisateur, le mot de passe, et la date de mise à jour sont null.
  + L'utilisateur n'est pas vérifié (isVerified() retourne false).
  + La date de création est bien définie et est une instance de \DateTimeImmutable.
  + Le rôle par défaut de l'utilisateur est ROLE\_USER, et la méthode getRoles() retourne un tableau avec un seul rôle.
  1. **testEmailAccessors** :
* Teste le getter et le setter pour l'email. L'email est défini via setEmail(), et le test vérifie que la valeur est correctement récupérée via getEmail() et également via getUserIdentifier().
  1. T**estUsernameAccessors** :
* Teste le getter et le setter pour le nom d'utilisateur. Le nom d'utilisateur est défini via setUsername() et le test vérifie que la valeur est correctement récupérée via getUsername().
  1. **testPasswordAccessors** :
* Teste le getter et le setter pour le mot de passe. Le mot de passe est défini via setPassword() et le test vérifie que la valeur est correctement récupérée via getPassword().
  1. **testRolesAccessors** :
* Vérifie la gestion des rôles :
  + Par défaut, l'utilisateur a le rôle ROLE\_USER.
  + Il teste l'ajout de rôles (ROLE\_ADMIN, ROLE\_MANAGER) et vérifie que le tableau de rôles retourne bien ROLE\_USER en plus des nouveaux rôles.
  + Teste également que les rôles en double sont supprimés, en passant les rôles ['ROLE\_ADMIN', 'ROLE\_USER', 'ROLE\_ADMIN'], et vérifie que la liste retournée est unique (['ROLE\_ADMIN', 'ROLE\_USER']).
  1. **testIsVerifiedAccessors** :
* Teste l'état de vérification de l'utilisateur :
  + Par défaut, l'utilisateur n'est pas vérifié.
  + L'état de vérification est modifié via setIsVerified() et vérifie que l'utilisateur devient vérifié ou non vérifié selon la valeur définie.
  1. **testUpdatedAtSetter** :
* Vérifie la gestion de la date de mise à jour. Par défaut, getUpdatedAt() est null. Après avoir appelé setUpdatedAt(), la date de mise à jour doit être définie, et le test vérifie qu'il s'agit d'une instance de \DateTime.
  1. **testEraseCredentials** :
* Teste la méthode eraseCredentials(). Le test appelle cette méthode et vérifie simplement qu'il ne génère pas d'erreur (la méthode ne retourne rien, donc l'assertion est triviale).
  1. **testUserInterface** :
* Vérifie que la classe User implémente les interfaces nécessaires :
  + Symfony\Component\Security\Core\User\UserInterface pour la gestion des utilisateurs dans Symfony.
  + Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface pour la gestion de l'authentification avec mot de passe.

En général, les tests pour les entités, servent juste à tester que les composant ont bien les propriétés qui sont testé.

**Test des données valides (testSubmitValidData) :**

* **Création de données valides** : On crée un tableau avec des données valides pour un utilisateur, incluant un email, un nom d'utilisateur, et deux champs de mot de passe identiques.
* **Soumission du formulaire** : On soumet ces données au formulaire de création de compte.
* **Validation du formulaire** : On vérifie que le formulaire est valide après la soumission.
* **Vérification des données** : On vérifie que les données soumises (email, nom d'utilisateur) sont bien enregistrées dans l'objet User et que le mot de passe n'est pas nul après la soumission.

**2. Test des données invalides (testSubmitInvalidData) :**

* **Création de données invalides** : On crée un tableau avec des données incorrectes (les mots de passe ne correspondent pas).
* **Soumission des mauvaises données** : On soumet ces données au formulaire.
* **Validation du formulaire** : On vérifie que le formulaire n'est pas valide à cause des mots de passe différents.
* **Vérification des données de l'utilisateur** : On s'assure que le mot de passe de l'utilisateur reste nul, car la soumission est invalide.

Les tests manquant font la preuve d’un manque d’expérience en test de ma part, je n’ai pas su les réaliser.

1. ***Méthodes de recherche mise en place pour réaliser ce projet.***

Pour réaliser ce projet, j’ai dû effectuer des recherches.

Pour commencer le projet, je me suis poser la question de si je n’avais pas un projet à peu près similaire afin d’avoir une base pour ce projet, puis j’ai fait le rapprochement avec un ancien projet « Stubborn » comportant quasiment les mêmes caractéristiques, j’ai donc repris ce projet, et modifier aux goûts de Knowledge.

Puis je me suis servi d’une feuille pour dessiner la méthode d’approche à adopter pour créer des entités.

Je me suis rendu compte lors de mon croquis que les entités avait toutes des caractéristiques spécifique comme : un type, un prix….

J’ai donc regardé sur la documentation de symfony pour en

savoir plus sur les champs qu’on pouvait mettre sur

les entitées (boolean, String, Integer…) afin de mieux orienté mes

données.

Lien vers la documentation Entity de symfony : <https://symfony.com/doc/current/reference/forms/types/entity.html>

Puis après je me suis rendu compte qu’il fallait que lors de l’inscription l’application envoie un mail afin de confirmer son adresse email.

J’ai donc fais la même choses qu’avec les champs pour Symfony, j’ai d’abord regardé dans la documentation et j’ai trouvé ce qu’il me fallait,

(<https://symfony.com/doc/current/mailer.html#creating-sending-messages>)

J’ai donc mis en place ce système avec Mailtrap afin de faciliter le déploiement, puis j’ai modifié le SMTP avec celui de mon mail google.

Cependant pour créer le système SMTP avec Google j’ai rencontré quelques problèmes, j’ai donc mener quelques recherches jusqu’à tomber sur un forum expliquant comment faire :

<https://www.gmass.co/blog/gmail-smtp/#gmailsmtp>

J’ai donc réussi à créer mon lien SMTP, que l’on peux retrouver dans le .env du projet.

Puis je me suis renseignée afin de créer une relation entre les entitées

Afin de réalisée une relation formation/leçon

<https://www.youtube.com/watch?v=4FgqOSBuem8>

Pour réaliser les options demandé dans la page back-office, je me suis servis de ce que j’ai appris pour retranscrire les informations de base de données sur la page template, sauf pour les paiements stripe ou je n’ai trouvé aucune documentation sur cela, j’ai du faire appel à l’IA dont je ne me sers quasiment jamais, alors j’ai eu un peu de mal à comprendre comment formulé ma phrase, mais surtout à faire fonctionner le code qu’elle me renvoyais, alors j’ai pris la base de son code et je l’ai modifier avec les informations que j’avais à disposition afin de faire marcher le code.

Pour réaliser les tests, je me suis servis de plusieurs vidéos et d’une documentation.

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

https://www.youtube.com/watch?v=UW9UoMcHLbc