Inscription à un cours

Laetitia Guidetti  
Chef de projet : Cindy Hardegger  
ETML mai-juin 2021

Table des matières

[1 Analyse préliminaire 4](#_Toc73023167)

[1.1 Introduction 4](#_Toc73023168)

[1.2 Objectifs 4](#_Toc73023169)

[1.3 Méthodologie de gestion de projet 4](#_Toc73023170)

[2 Analyse et Conception 5](#_Toc73023171)

[2.1 Planification initial 5](#_Toc73023172)

[2.2 Symfony 5](#_Toc73023173)

[2.3 Bootstrap 5](#_Toc73023174)

[2.4 Twig 6](#_Toc73023175)

[2.5 Mockup 6](#_Toc73023176)

[2.6 MCD et MLD 7](#_Toc73023177)

[2.7 Doctrine ORM 8](#_Toc73023178)

[2.8 Vue smartphone 9](#_Toc73023179)

[2.9 Politique de mots de passe 9](#_Toc73023180)

[3 Réalisation 10](#_Toc73023181)

[3.1 Routes 10](#_Toc73023182)

[3.2 Entités 11](#_Toc73023183)

[3.2.1 Environnement 11](#_Toc73023184)

[3.2.2 Création d’une entité 11](#_Toc73023185)

[3.2.3 Migration 14](#_Toc73023186)

[3.2.4 Les requêtes 14](#_Toc73023187)

[3.3 Le paquet « Security » 16](#_Toc73023188)

[3.3.1 L’entité « User » 16](#_Toc73023189)

[3.3.2 Formulaire de connexion 17](#_Toc73023190)

[3.3.3 Gestion des accès 17](#_Toc73023191)

[3.3.4 Le fichier « security.yaml » 18](#_Toc73023192)

[3.4 Les formulaires 21](#_Toc73023193)

[3.4.1 Création d’un formulaire 21](#_Toc73023194)

[3.4.2 Configuration du formulaire 22](#_Toc73023195)

[3.4.3 Création et envoie des données 23](#_Toc73023196)

[3.4.4 Affichage du formulaire 24](#_Toc73023197)

[3.4.5 Validation des champs 25](#_Toc73023198)

[3.5 Le mot de passe 26](#_Toc73023199)

[3.5.1 Génération d’un nouveau mot de passe 26](#_Toc73023200)

[3.5.2 Envoi d’un mail 26](#_Toc73023201)

[3.6 La pagination 26](#_Toc73023202)

[4 Tests 27](#_Toc73023203)

[4.1 Page d’accueil 27](#_Toc73023204)

[4.2 Connexion d’un administrateur 28](#_Toc73023205)

[4.3 Gestion des cours 28](#_Toc73023206)

[4.4 Inscription à un cours 30](#_Toc73023207)

[4.5 Gestion du profil 32](#_Toc73023208)

[5 Conclusion 34](#_Toc73023209)

[5.1 Bilan personnel 34](#_Toc73023210)

[6 Glossaire 35](#_Toc73023211)

[7 Sources et références 35](#_Toc73023212)

[8 Annexes 35](#_Toc73023213)

[8.1 Journal de travail 35](#_Toc73023214)

# Analyse préliminaire

## Introduction

## Objectifs

## Méthodologie de gestion de projet

Pour ce projet le choix de la méthode de gestion de projet c’est porté sur le modèle en cascade. Il est généralement séparé en 5 phases qui doivent se réaliser les unes après les autres.

1. Analyse
2. Conception
3. Implémentation
4. Test
5. Exploitation

Ce modèle permet d’avoir une structure simple et claire. Comme chaque étape se déroule l’une après l’autre, le moindre retard est directement remarquable. Le défaut principal de cette méthodologie est qu’elle ne supporte pas le changement. Si le client change d’avis, il sera difficile d’apporter ses modifications à un projet utilisant la méthodologie en cascade. Dans le cadre de ce TPI, cela n’est pas un problème car le CDC et les objectifs ne peuvent pas être modifier au cours de la réalisation du projet.

Chaque fonctionnalité sera analysée puis conceptualisée. Elle sera ensuite implémentée dans le code, après cela des tests permettront de vérifier que la réalisation est conforme au attente. L’exploitation se fera à la toute fin du TPI avec la mise en service du site sur les serveurs de l’ETML.

# Analyse et Conception

## Planification initial

## Symfony

Dans ce projet utiliser un framework PHP n’est pas imposé. Cependant, je souhaite utiliser Symfony et une explication s’impose.

Lors de la création d’un site Web certaines fonctionnalités reviennent très souvent : La gestion de l’authentification des utilisateurs, un formulaire de contact ou encore la validation des données. Il s’agit de tâches redondantes, ce serait une perte de temps de les recréer à chaque nouveau site. Un framework PHP permet alors de fournir différent composants permettant d’implémenter directement ses fonctionnalités sans avoir besoin de les recréer. Ses composants ont chacun déjà été utilisé par des milliers des sites, par conséquent un grand nombre de bugs et de faille de sécurités ont déjà été trouvés et corrigés.

Un framework permet une fois un minimum maitriser de gagner beaucoup de temps tout en améliorant la sécurité globale du site. Le défaut principal est qu’au début, il est assez dur d’apprendre un framework PHP, car il nécessite de comprendre son fonctionnement.

Parmi tous les framework PHP, le choix s’est porté sur Symfony. Il s’agit de l’un des plus utilisé à travers le monde et il est le premier dont j’ai entendu parlé. Il intègre également l’architecture MVC nécessaire dans ce projet.

## Bootstrap

Le choix du framework CSS c’est porté sur Bootstrap. Il est OpenSource et a été développer pour le mobile-first. Cela implique qu’il est facile à adapter pour les différentes tailles d’écran. J’ai également déjà eu l’occasion de l’utiliser durant des projets de développement Web à l’ETML, ce qui me permet de l’utiliser avec plus d’aisance qu’un autre framework CSS.

Son utilisation ne demande aucun téléchargement, il suffit d’insérer les liens vers le CSS et les fichiers JavaScript correspondent pour le faire fonctionner.

La version de Bootstrap utilisée durant ce projet est la 4.6. La version 5.0 est sortie le 5 mai dernier. Elle n’est cependant pas utilisée, car elle n’a pas encore été implémentée dans les thèmes de Symfony. Plusieurs fonctionnalités de ce dernier donne la possibilité d’utiliser un thème précis pour modifier l’affichage. Par exemple, lorsqu’un formulaire est créé, il est possible de lui choisir le thème Bootstrap 4. Par conséquent, si l’application utilise la version 5, l’apparence du formulaire sera mal gérée car certaines classes possèdent des noms différents d’une version à l’autre.

## Twig

Twig est un moteur de modèle PHP et il est par défaut utilisé par Symfony. Il est recommandé d’utiliser snake case pour les noms de fichiers et de dossier. Cette convention indique que chaque espace est remplacé par un tiret bas.

Il possède 3 syntaxes de base :

{{ … }} permet d’appeler une expression (variable, fonction, etc.)

{% … %} exécute une commande

{# … #} n’est pas exécuté, il s’agit des commentaires

## Mockup

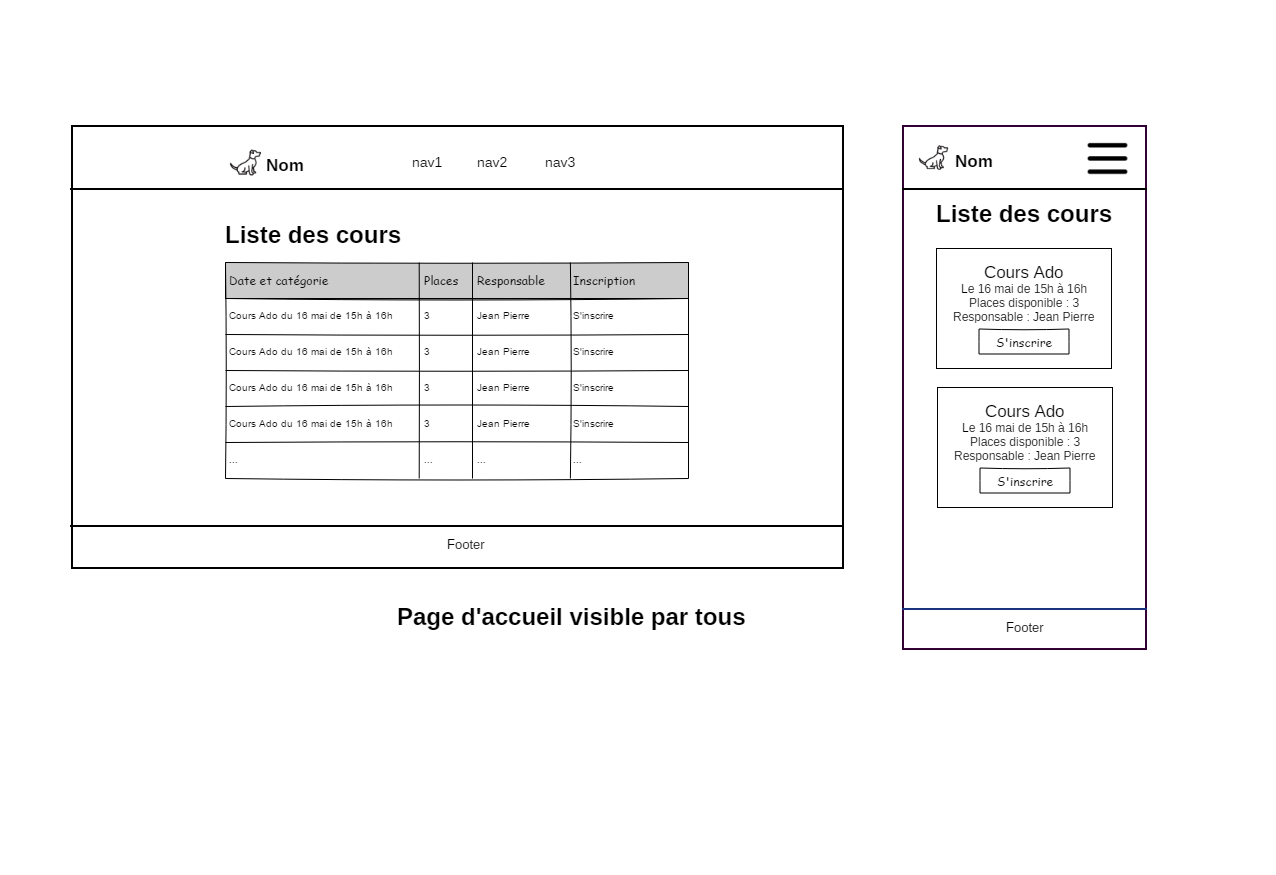
Une mockup est une maquette permettant de représenter le site web de manière simpliste.

Figure 1 Mockup de la page d'accueil

La page d’accueil du site sert à afficher les différents cours et permet aux utilisateurs de s’y inscrire. Par conséquent, l’aspect graphique doit rester assez simple afin de ne pas rendre la tâche difficile aux utilisateurs. L’affichage sur téléphone est modifié car la largeur de ces appareils ne permet pas d’afficher les grands tableaux.

Le formulaire d’ajout d’un cours ne contient que les informations utiles pour l’administrateur tout en restant intuitif. Ce mockup sert de base à toutes les pages du site utilisant des formulaires comme pour l’inscription d’un utilisateur à un cours.

Figure 2 Mockup du formulaire d'ajout de cours

## MCD et MLD

La modélisation d’une base de données ce faite en 3 étapes.

1. Le modèle conceptuel des données (MCD)
2. Le modèle logique de données (MLD)
3. Le modèle physique de données (MPD)

Pour réaliser une base de données plusieurs règles sont à respecter.

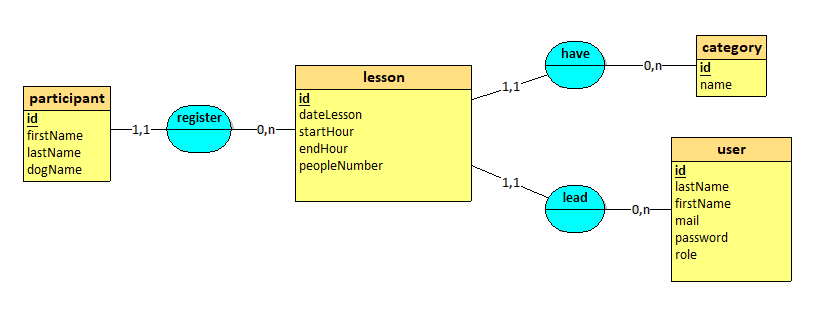
1. Non redondance : chaque information n’apparaît qu’une fois
2. Atomicité : une seule information par champ
3. Unicité : chaque champ est identifiable de manière unique

Figure 3 MCD de la base de données

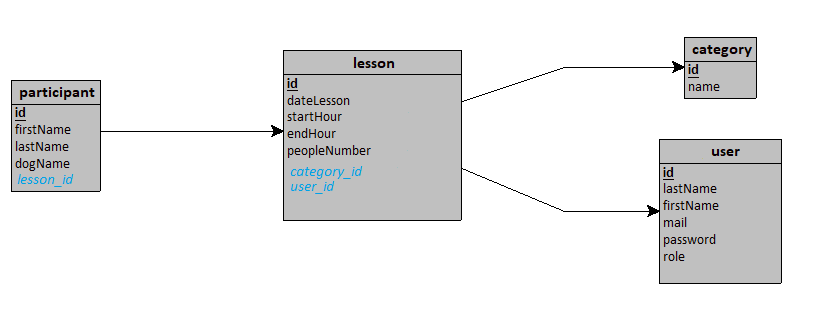
Dans le CDC, une liste des caractéristiques d’un cours est donnée. Cependant, créer un type d’entité directement d’après cette liste ne respecterait pas les règles d’élaboration d’une BD. L’une des caractéristiques est qu’un cours possède « Une catégorie de cours (ex : Ado) ». Ajouter, sur le type d’entité « lesson », l’attribut « category » serait contraire au principe de la non redondance. En effet, plusieurs cours pourraient avoir comme catégorie « Ado », il y aurait alors une redondance s’il s’agissait d’un attribut de « lesson ». Un nouveau type d’entité est donc utilisé afin de respecter cette règle. On remarque également que pour le nom des participants, 2 attributs sont prévus l’un pour le nom de famille et l’autre pour le prénom, afin de respecter la règle d’atomicité.

Figure 4 MLD de la base de données

Le MLD est basé sur le MCD et représente les futurs tables et colonnes de la base de données.

## Doctrine ORM

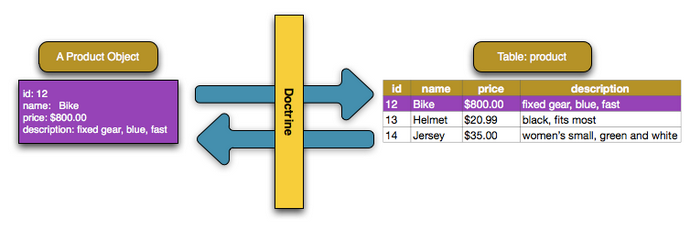
Doctrine est l’ORM (object-relational mapping) utilisé par Symfony. Un ORM est un couche d’abstraction qui synchronise les objets PHP avec une base de données relationnelle. Son but est de créer un ensemble de classe permettant de manipuler les tables et données de la BD. Il permet de transformer les objets PHP en écriture SQL ce qui permet notamment de générer des requêtes SQL.

Figure 5 Fonctionnement de Doctrine, source : https://symfony.com/doc/current/doctrine.html

C’est donc via Doctrine que la base de donnée va être créée ainsi que ses différentes tables en se basant sur le MLD. Il va permettre de créer des classes appelées entités. A partir de ses dernières, une base de données correspondante pourra être générée automatiquement.

## Vue smartphone

Explication du système de bootstrap, choix des listes au lieu du tableau

## Politique de mots de passe

Explication sur l’importance d’un bon mot de passe et de son utilité

Politique de mots de passe :

* Longueur minimal de 12 caractères
* Minimum une lettre minuscule
* Minimum une lettre majuscule
* Minimum un chiffre
* Minimum un caractère spécial

Cette politique de mots de passe permet d’éviter un grand nombre de mots de passe générique. Elle est notamment recommandée par le [NCSC](https://www.ncsc.admin.ch/ncsc/fr/home/infos-fuer/infos-private/aktuelle-themen/schuetzen-sie-ihre-konten.html)[[1]](#footnote-1) (centre national pour la cyber sécurité).

Cette dernière est plus stricte que celle de nombreux site web. Généralement, le nombre minimal de caractères demandé est de 8 et aucun caractère spécial n’est exigé. Cependant, dans le cadre de ce projet, les personnes touchées par cette politique sont les administrateurs. Ils ont un accès directement aux données personnelles des utilisateurs. Pour cette raison, une politique très stricte est utilisée. Si dans le futur des utilisateurs non administrateurs pouvaient s’inscrire sur le site, la politique de mots de passes de ses derniers pourrait être différentes de celle des administrateurs.

# Réalisation

## Routes

Quand un utilisateur clique sur un lien ou écrit une URL, la page correspondante doit s’afficher. Le routage permet de définir quelle action doit être effectuer en fonction de la requête reçue par l’application Web. L’action renverra ensuite les différentes informations à afficher.

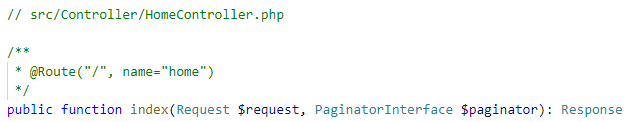
Il est possible de définir les routes de plusieurs façons, : via des attributs ou des annotations, dans des fichiers YAML, XML ou PHP. Selon les [bonnes pratiques](https://symfony.com/doc/current/best_practices.html#use-attributes-or-annotations-to-configure-routing-caching-and-security)[[2]](#footnote-2) de Symfony, il est recommandé de les définir via des attributs ou des annotations.

Figure 6 Route définie par une annotation

Dans ce cas, quand l’utilisateur va sur l’adresse « / », la méthode index est appelée. Elle se charge de répondre à la demande en affichant la page correspondante.

## Entités

### Environnement

Pour créer et utiliser des entités, il faut que l’environnement soit correctement configuré. Cela demande une base de données vide et les informations pour s’y connecter. Elles doivent être définie dans le fichier .env à la racine du projet sous la variable d’environnement « DATABASE\_URL ». Cette ligne contient en premier le nom et le mot de passe de l’utilisateur qui accède à la BD. Puis l’adresse de l’hôte et le nom de la base de données sont renseignés.

Figure 7 Information de la base de données dans le fichier .env

C:\Users\laeguidetti\Desktop\Screen\Database\CreateDB.PNGUne base de données vide correspondant à ses informations peut être alors généré via une commande. Une erreur sera soulevée si une BD portant son nom existe déjà.

Figure 8 Commande créant la base de données

Il est maintenant possible de créer les entités qui permettront de générer les tables dans la base de données.

### Création d’une entité

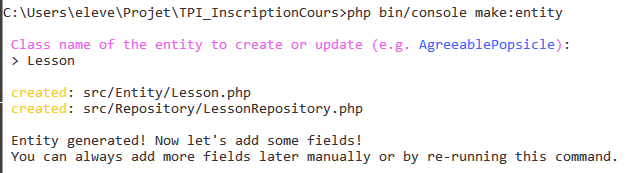
La création d’une entité se fait via une ligne de commande.

Figure 9 Création d'une entité

Après avoir donné le nom de l’entité, 2 fichiers sont automatiquement créés. Le premier « Lesson.php » contient la classe « Lesson ». Le deuxième est le fichier « LessonRepository.php », il permet d’exécuter des requêtes de base et de créer des requêtes précises liées.

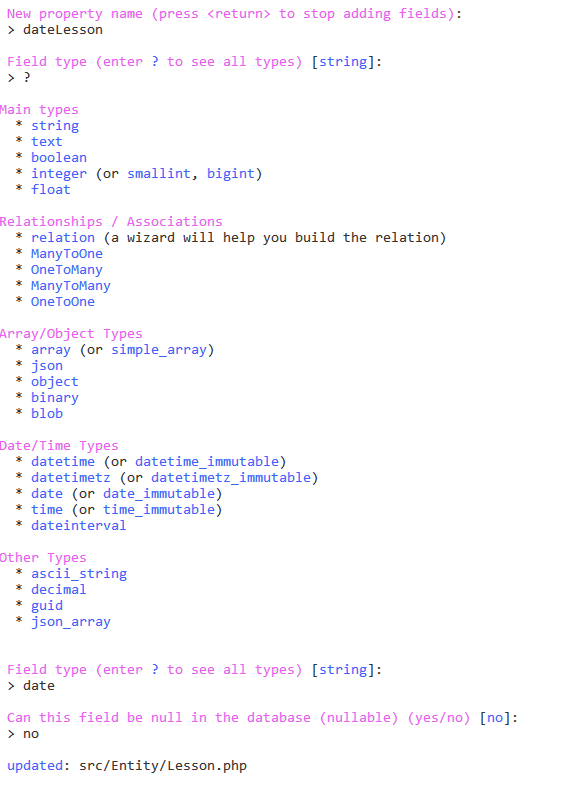
L’invite de commande proposera ensuite d’ajouter des propriétés à l’entités créées.

Figure 10 Ajouter une propriété à une entité

Plusieurs questions seront alors posées pour configurer la propriété correctement. Le fichier « Lesson.php » est alors mis à jour en conséquence. Il est possible dans ce dernier de configurer de manière plus précisée les propriétés comme ajouter une valeur par défaut. Ce fichier génère également des getter et setter pour chaque propriété afin de faciliter leur utilisation.

Pour chaque entité une propriété « id » est automatiquement créée.

Après avoir créé toutes les entités, il manque les relations entre elles.

Figure 11 Création d'une relation

Pour modifier une entité déjà créée, il suffit lors d’un « make:entity » d’indiquer son nom. La console reconnait automatiquement que l’entité existe déjà et permet de lui ajouter de nouvelles propriétés. Comme son nom l’indique le type « relation » permet de définir les relations. Il suffit ensuite de préciser avec quel entité la relation se fait et son type. Cela mettra à jour les 2 entités afin que la relation soit valide.

Toutes ses opérations sont également réalisables en modifiant à la main les divers fichier d’entités.

### Migration

Toutes les entités et leurs relations sont créées, cependant la base de données ne contient toujours aucunes tables. Il faut alors migrer toutes les informations présentent dans les classes entités dans cette dernière. Une commande permet de générer automatiquement un fichier contenant des requêtes SQL correspondant aux entités.

Figure 12 Migration

Une deuxième commande permet d’exécuter tous les fichiers de migrations qui n’ont pas déjà été opéré. La base de données est alors modifiée en conséquence. Si plus tard des modifications sur une entité surviennent, il suffit de recréer un fichier de migration et de l’exécuter.

Il existe d’autres commandes liées aux migrations. Il est possible de supprimer la dernière migration, d’obtenir la liste de toutes les migrations ou encore la version actuelle. Ses cas spécifiques ne sont pas abordé ici, mais ils sont présent dans la [documentation](https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html)[[3]](#footnote-3) de Symfony.

### Les requêtes

Certaines informations présentent dans la base de données doivent être affichée aux utilisateurs. Dans le cadre de ce projet, il s’agirait d’afficher les différents cours. C’est à ce moment que le fichier « LessonRepository.php » généré plutôt devient utile. Doctriner permet via ce dernier de générer des requête SQL à partir d’objet PHP. Certaines requêtes basiques sont même déjà utilisables. Dans un contrôler via une instance de la classe « LessonRepository », il est possible d’appeler la méthode « findAll() ». Cette dernière va alors retourner toutes les entrées présentent dans la table « Lesson ». D’autres méthodes pré-écrites du même genre existent comme « find($id) » qui retourne l’entrée possèdent l’id spécifié.

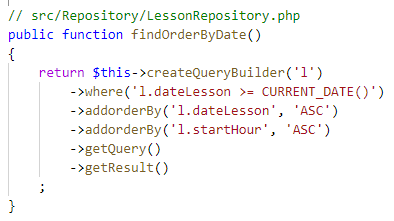
Cependant, tout cela n’est pas suffisant pour ce projet. Les utilisateurs ont besoin de voir tous les cours ne s’étant pas encore déroulés et pour des raisons de lisibilités dans un ordre précis. Doctrine permet via le « QueryBuilder » d’écrire des requêtes en utilisant du PHP.

Figure 13 méthode retournant les cours qui ne se sont pas encore déroulés

La méthode « createQueryBuilder » défnini automatiquement le SELECT et le FORM, l’argument fourni est généralement la première lettre de l’entité du répertoire. Dans ce cas, l’entité est « Lesson », l’argument utilisé est donc un « l ».

La classe « [QueryBuilder](https://www.doctrine-project.org/projects/doctrine-orm/en/2.8/reference/query-builder.html)[[4]](#footnote-4) » possède de nombreuses méthodes permettent de construire des requêtes complexes. Les noms de ses dernières sont souvent très similaire au SQL se qui facilite leur utilisation. La requête est alors construite étape par étape, il est donc facilement possibile d’y ajouter des conditions.

Doctrine se charge ensuite de générer le SQL corréspondent et de l’executer.

## Le paquet « Security »

Security est un paquet utiliser pour l’authentification et la gestion des accès de manière sécurisée.

### L’entité « User »

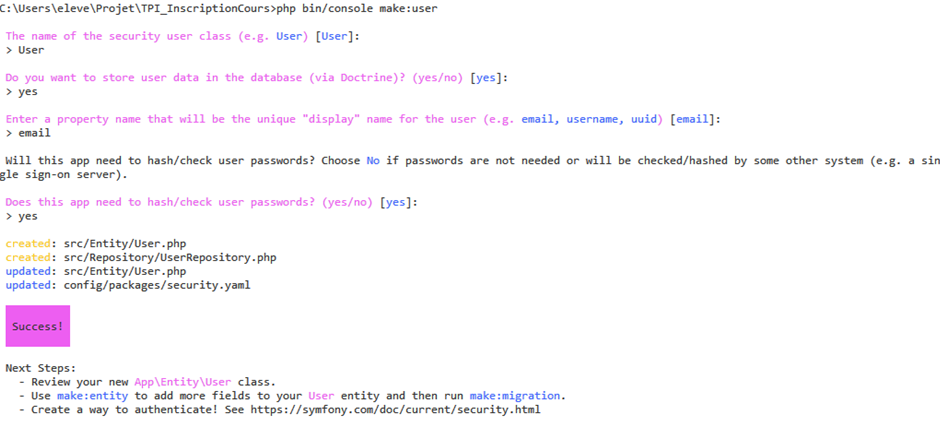
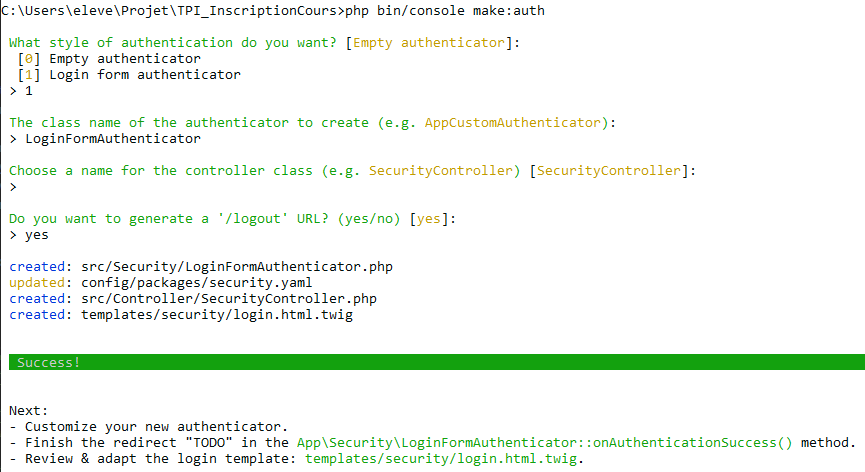
Les informations des utilisateurs seront stockées dans l’entité « User ». Elle doit obligatoirement implémenter « Symfony\Component\Security\Core\User\UserInterface » pour être utilisée. Pour générer facilement cette entité spéciale, une commande est disponible.

Figure 14 Création de l'entité « User »

Comme durant la création des entités classique, 2 fichiers sont générés. Ils sont similaires aux précédents à quelques exceptions près. Le classe « User » implémente « UserInterface », elle possède l’attribut « roles » qui permettra la gestion des accès ainsi que l’attribut « password » qui permettra de stocker le mot de passe haché. Le fichier de configuration du paquet « security.yaml » est également mis à jour, il est détaillé au point 3.3.4 Le fichier « security.yaml ».

### Formulaire de connexion

Des administrateurs peuvent être créer dans la table « user ». Il faut maintenant qu’ils puissent se connecter pour administrer le site. Le paquet « Security » permet de générer un formulaire sécurisé via une commande.

Il génère le fichier « templates/security/login.html.twig », il contient le formulaire de connexion. Il est à modifier car tous les termes y sont en anglais et son aspect graphique n’est pas le plus agréable. Le fichier « src/Controller/SecurityController.php » est également créé. Il contient un contrôleur chargé de gérer les routes de connexion et de déconnexion. Le dernier fichier généré « src/security/LoginFormAuthenticator.php » est le plus complexe. Il se charge de toute la procédure d’authentification.

### Gestion des accès

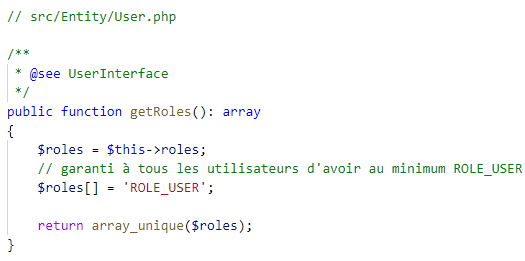
Les utilisateurs peuvent maintenant se connecter et se déconnecter. Cependant, cela ne donne aucun accès aux utilisateurs et n’en restreint aucun, il faut alors les configurer. Par défaut, la class User possède une propriété utilisée pour la gestion des accès. Il s’agit de la propriété « roles » qui est de type JSON. C’est la méthode getRoles() qui est ensuite utilisée pour déterminer quelles rôles sont possédés par un utilisateur. Dans tous les cas pour les utilisateurs connectés, la méthode renverra au minimum le rôle : « ROLE\_USER ».

Figure 15 La méthode getRoles()

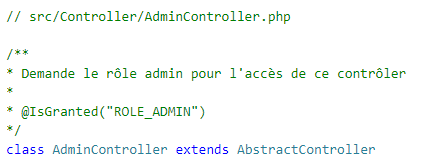
Il est possible d’attribuer d’autres rôles aux utilisateurs, mais ils doivent respecter une règle : commencer par « ROLE \_ » comme par exemple « ROLE\_ ADMIN ». Il est possible de configurer les différentes restrictions d’accès dans le fichier security.yaml ou dans le contrôleur (ou dans d’autre partie du code comme les fichiers twig). Cependant, il n’est pas recommandé par les [bonnes pratiques](https://symfony.com/doc/current/best_practices.html#use-attributes-or-annotations-to-configure-routing-caching-and-security)[[5]](#footnote-5) de symfony de le faire dans le fichier security.yaml.

Figure 16 Annotation empêchant l'accès à tous les utilisateurs ne possèdent pas le rôle "ROLE\_ADMIN"

Avant une méthode de controller, l’annotation « IsGranted » permet de restreindre la méthode aux utilisateurs possèdenant le rôle indiqué. Si aucun utilisateur n’est connecté, il sera redirigé vers la page de connextion. Si l’utilisateur est connecté mais ne possède pas le rôle, il aura le message d’erreur 403 accès refusé. L’annotation IsGranted peut également être utilisée avant la déclaration du controller pour restreindre l’entiéreté des méthodes du controller. Cette annotation demande de faire référence à l’espace de nom Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted pour fonctionner.

Il est également possible de faire une restriction sur un templates Twig. Par exemple, si l’on souhaite que les bouttons « Connexion » et « Deconnexion » s’affichent en fonction de si l’utilisateur est connecté ou non. Il est possible alors d’utiliser la fonction is\_granted(role) pour appliquer des restricitions. A noter qu’il est possibile d’utiliser certains attributs à la place des rôles. Par exemple l’attribut « IS\_ANONYMOUS » marche uniquement avec les utilisateurs non connecté.

### Le fichier « security.yaml »

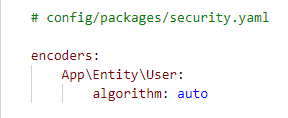
Plusieurs fois lors des étapes précédentes, l’invite de commande indiquait que le fichier « security.yaml » avait été mis à jour. Il contient la configuration du paquet « Security » et permet son fonctionnement. Il est séparé en plusieurs options.

Figure 17 L'option « encoders »

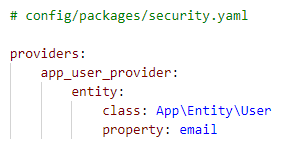
La première option est « encoders » elle permet de déterminer l’algorithme utilisé pour le hachage. Le nom fait plus penser à de l’encodage que du hachage, mais il s’agit bien de hachage ce nom est présent pour des raisons historiques. La classe sur la quel s’applique cette algorithme est à préciser car il est possible d’avoir plusieurs classes avec des algorithmes différents. Choisir « auto » permet de laisser symfony choisir le meilleur algorithme de hachage disponible selon lui. Actuellement, il s’agit de l’Argon2 qui hache sur 96 caractères. S’il n’est pas disponible, c’est l’algorithme bcrypt qui est utilisé. Symfony se charge de réaliser un salage différent à chaque nouveau mot de passe haché. L’algorithme de hachage utilisé est voué à être changé dans le futur lorsqu’un meilleur algorithme sera trouvé, par conséquent il est recommandé de prévoir assez d’espace pour sauvegarder les mots de passe haché.

Figure 18 L'option « providers »

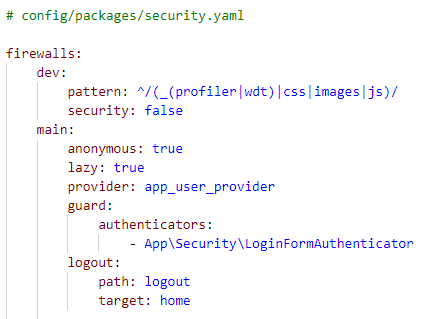
L’option suivante est « providers ». Il permet d’indiquer quelle entité représente les utilisateurs et quelle propriété est à utiliser pour l’authentification. Ces paramètres ont été générés automatiquement à la création de la classe « User ». La classe « User Providers » permet également de l’actualisation de l’utilisateur courant dans les variables de session et rend possible l’ajout de plusieurs fonctionnalités comme le « se souvenir de moi ».

Figure 19 L'option « firewalls »

Une des autres options est « firewalls », elle est considérée comme la plus importante. Elle gère comment les utilisateurs vont être authentifier. Il peut y avoir plusieurs firewalls, chacun opère comme un système de sécurité différent. Dans ce cas, il y en a deux « dev » et « main ». Le premier comme son nom l’indique est présent pour le développement. Il gère les accès au « profiler » un outil de développement permettent d’avoir des informations détaillées sur chaque requête. Le deuxième « main » permet de gérer l’authentification des utilisateurs du site. Chaque firewall peut ensuite être configurés via plusieurs option. L’option « provider » permet de déterminer quelle entité définie plus tôt comprend les utilisateurs. La classe chargée de l’authentification est indiquée grâce à l’option « guard ». Il est également possible de définir la procédure de déconnexion. Sous « logout », « path » indique quelle route gère la déconnexion et « target » indique la route sur laquelle l’utilisateur doit être redirigé après cette opération.

La majorité des informations présentes dans ce fichier sont générées automatiquement lors de la création d’un système d’authentification via « Security ». Si ce système est modifié, il faut impérativement modifier ce fichier. Par exemple, si à la place du mail, il est souhaité d’utiliser un pseudo lors de la connexion. La valeur de l’option « property » du « provider » doit être modifié pour indiquer la nouvelle propriété.

## Les formulaires

Sur un site web, il est commun pour les utilisateurs de remplir des formulaires. Cela renseigner une base de données dans les cas d’inscriptions ou cela peut être d’envoyer un mail dans le cas du formulaire de contact.

Cependant, la création de ses derniers est relativement répétitive et longue à réaliser. Il faut renseigner tous les champs, faire une validation de données, s’assurer de la conformité de tous les champs et de leur bonne intégration dans la BD. Mais, Symfony possède une fonctionnalité qui simplifie grandement toutes ses opérations, il s’agit de l’outil « form ».

### Création d’un formulaire

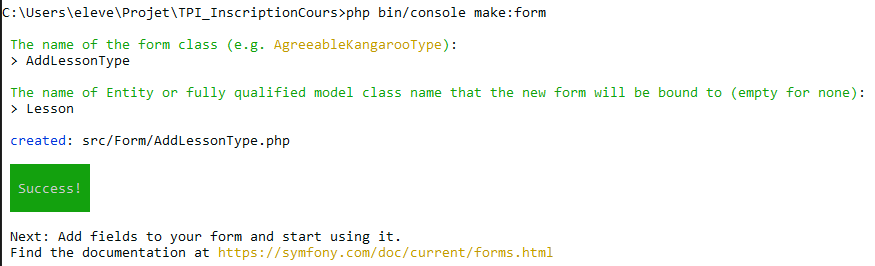
Une commande permet de générer automatiquement un formulaire.

Figure 20 Création du formulaire « AddLessonType »

L’invite de commande demande en premier le nom du formulaire qui doit finir par « type ». Puis, il demande s’il se refaire à une entité existante. Cela permet de générer automatiquement les champs correspondent aux différentes propriétés de l’entité. Un fichier contenant la classe du formulaire est ensuite généré automatiquement.

### Configuration du formulaire

Une fois la classe du formulaire généré certaines modifications peuvent être à faire pour le rendre plus pertinent.

La méthode « buildForm » permet d’ajouter les différents champs du formulaire via la méthode « add » provenant de l’interface « FormBuilderInterface ». Pour chaque champ, il est possible de configurer plusieurs paramètres. Par exemple, à l’affichage d’un champ, le nom de ce dernier est utilisé comme label. Cependant, lorsque le formulaire provient d’une entité, le nom de chaque champ se refaire à une des propriétés de l’entité. Cependant, pour l’utilisateur il n’est pas pratique d’avoir le nom d’un paramètre comme label. Il est possible de régler ce type de problème via l’option « label » qui permet de redéfinir le label affiché. Il existe diverses autres options permettent de modifier l’affichage ou le fonctionnement d’un champ.

Figure 21 Classe « AddCategoryType » permettant l'ajout d'une catégorie

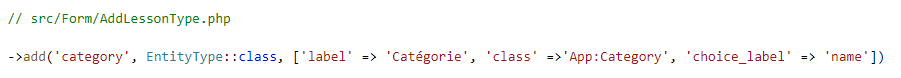
La deuxième méthode « configureOptions » permet d’indiquer la classe utilisée par le formulaire. Elle permet également de modifier certaines options comme la méthode http utilisée qui est par défaut du POST. Il est également possible de modifier l’action qui par défaut redirige sur la même URL que celle qui a affiché le formulaire.

Figure 22 Champ « category »

Cependant, certains champs doivent plus détaillés pour ne pas générer d’erreur. C’est le cas des champs provenant d’une relation entre 2 tables. Un exemple est dans la classe « AddLessonType » qui permet d’ajouter un cours. Chaque cours doit appartenir à une catégorie renseignée dans la classe « Category ». Pour cela le champs « category » est utilisé. Cependant, le formulaire ne sait pas ce qu’il doit afficher à partir de cette objet. Il faut donc lui donner des informations détaillées. Il s’agit d’une entité « EntityType » et plus précisément de la classe « Category ». Il ne sait alors toujours pas quoi afficher. Le problème est régler grâce à « choice\_label » qui permet d’indiquer la propriété à afficher.

### Création et envoie des données

Une fois la classe du formulaire configurée, il faut l’afficher dans la vue pour que l’utilisateur puisse le remplir et permettre alors de récupérer les données pour les enregistrer dans la base de données.

Dans le cas de l’ajout d’une catégorie, il faut tout d’abord créer un objet « Category » qui permettra de contenir les informations rentrées. Il faut ensuite créer le formulaire grâce à la méthode « createForm » héritée de la classe « AbstractController » en paramètre, il faut lui donner le formulaire a utilisé ainsi que l’objet créé plutôt. La représentation visuelle du formulaire est ensuite envoyée à la vue via la méthode « createView ».

Cependant, il manque encore plusieurs points pour que ce formulaire soit utilisable. Le formulaire n’est pas traité, les informations fournies ne sont pas sauvegardées et l’affichage n’est pas gérer.

Pour régler ses problèmes, il faut commencer par utiliser la méthode « handleRequest » qui permet de reconnaitre si le formulaire est soumis et permet l’inscription des données dans l’objet « $newCategory ». Ensuite, la méthode « isValid » détecte si les informations remplissent les critères donnés (cf 3.4.5 Validation des champs). Il reste alors à ajouter toutes ses informations dans la base de donnée. Pour cela, l’objet « entity manager » est utilisé, il a été au préalable instancié dans la variable « $om » dans le constructeur de ce contrôleur. L’ajout des données dans la base de données se fait en 2 méthodes « persiste($newCategory) et flush(). La première indique à Doctrine qu’il doit gérer l’objet « $newCategory ». La deuxième demande à Doctrine de conserver dans la base de données tous les objets qu’il gère. Cela signifie qu’il va automatiquement générer et exécuter des requêtes correspondent aux informations indiquée pour les sauvegarder. Doctrine sait déterminer si les objets sont à ajouter ou à mettre à jour.

### Affichage du formulaire

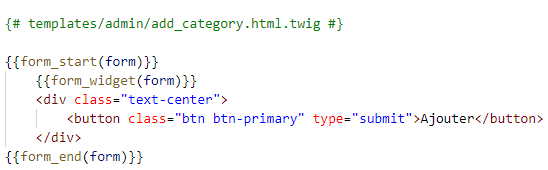
Au point précédent, le formulaire a été envoyé à la vue. Mais, il n’est alors pas affiché et donc pas utilisable.

Figure 23 Code permettant l'affichage du formulaire d'ajout de catégorie

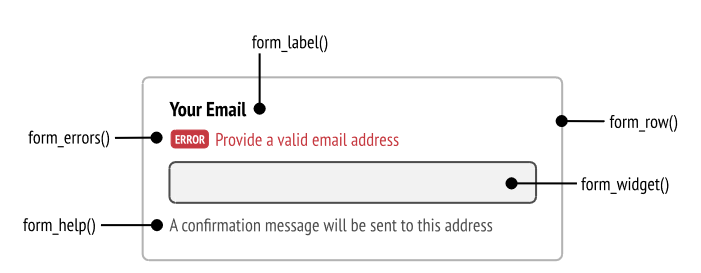
Il est possible d’afficher l’entière du formulaire en écrivant : « {{ form(form) }} ». Cela affiche tous les champs et gérer automatiquement les tags <form> de l’HTML. Mais cela ne permet aucune personnalisation. Twig possède plusieurs fonctions pour permettre la personnalisation. Il est possible d’utiliser les fonctions « form\_start() » et « form\_end() » pour gérer les tags du début et de la fin du formulaire.

Figure 24 Les fonctions twig permettant la personnalisation de chaque champ  
Source : https://symfony.com/doc/current/form/form\_customization.html

Ses fonctions permettent un affichage très précis et personnalisé pour chaque champ.

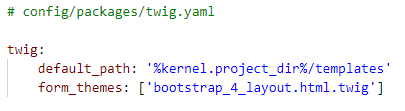
Twig permet également l’utilisation de thèmes gérant l’affichage.

Figure 25 Configuration du thème pour les formulaires

Il est possible dans créer un ou d’en utiliser un déjà existant. L’un des thème existant est celui de Bootstrap 4. Le choix du thème utilisé est à définir dans le fichier « twig.yaml », il sera alors automatiquement utilisé pour tout les formulaires. Si l’on souhaite un thème différent pour un formulaire particulier, il est possible de l’indiquer grâce au tag « form\_theme » de twig.

### Validation des champs

En fonction du contexte, chaque champ peut être soumis à des règles précises à respecter. Par exemple, un cours de peut pas avoir un nombre négatif de participant ou il n’est pas supposé être possible de créer un cours à une date antérieure à la date du jour. Pour empêcher l’utilisateur de remplir des données incohérente, une validation doit être effectuée sur chaque champ.

## Le mot de passe

### Génération d’un nouveau mot de passe

Comment générer un mot de passe

### Envoi d’un mail

Comment envoyer un mail

## La pagination

Le bundle Knp\_pafinator

# Tests

## Page d’accueil

Tests réalisés le 19.05.2021.

|  |  |
| --- | --- |
| 1 | Affichage des cours |
| But | Vérifier que tous les cours s’affichent sauf ceux passés sur la page d’accueil. |
| Action | Comparer les cours afficher sur la page d’accueil avec les entrées présentes dans la BD. |
| Résultat | Les cours s’affichent triés par jour puis par heure. Toutes les entrées de la table « lesson » dont la date est égal ou supérieur au 19.05.2021 s’affichent. |

|  |  |
| --- | --- |
| 2 | Affichage du bouton d’inscription |
| But | Vérifier que le bouton d’inscription ne s’affiche pas quand le jour actuel. |
| Action | Vérifier que pour tous les cours du 19.05.2021 le bouton « s’inscrire » ne soit pas affiché sur la page d’accueil. |
| Résultat | Il n’est pas affiché pour les cours du 19.05.2021 |

|  |  |
| --- | --- |
| 3 | Cours complet |
| But | Affichage de « Cours complet » lorsqu’un cours est complet |
| Action | Regarder dans la BD tous les cours ayant autant d’inscrit que de place disponible. Puis, sur la page d’accueil vérifier que le texte « Cours complet » s’affiche à la place du nombre de places disponible. |
| Résultat | Le texte s’affiche pour les cours complet et le bouton d’inscription disparait. |
|  | |

|  |  |
| --- | --- |
| 4 | Pagination |
| But | La pagination est automatique |
| Action | Vérifier que la pagination est automatique en ajoutant des cours à la BD. Cela devrait automatiquement ajouter une page. |
| Résultat | Le nombre de page est passée de 2 à 3. La pagination est toujours fonctionnelles. |

|  |  |
| --- | --- |
| 5 | Responsivité |
| But | La page doit être visuellement correcte sur mobile. |
| Action | Grâce à la vue adaptive (Ctrl + shift + m sur Firefox), passer la largeur de l’écran à 300 pixel et vérifier comment les éléments s’affichent. |
| Résultat | La barre de navigation devient un menu hamburger. Les cours ne s’affichent plus sous la forme d’un tableau, mais sous forme de blocs les uns après les autres |

## Connexion d’un administrateur

Tests réalisés le 19.05.2021.

|  |  |
| --- | --- |
| 6 | Accessibilité à la page de connexion de l’administrateur |
| But | Accéder au formulaire de connexion de l’administrateur. |
| Action | Être sur la page d’accueil et ajouter « /admin » dans l’URL. |
| Résultat | Une page proposant de se connecter s’affiche. |

|  |  |
| --- | --- |
| 7 | Connexion |
| But | Se connecter en tant qu’administrateur. |
| Action | Remplir le formulaire avec comme mail « admin@gmail.com » et comme mot de passe « 123456 ». Il s’agit d’un utilisateur de test qui doit être présent dans la table « user » avec le rôle « ROLE\_ADMIN ». |
| Résultat | La connexion réussi et l’utilisateur est redirigé vers la page /admin/home. |
|  | |

|  |  |
| --- | --- |
| 8 | Vérification des erreurs |
| But | Impossible de se connecter avec des identifiant incorrecte. |
| Action | Remplir le formulaire avec des identifiants non présent dans la BD |
| Résultat | Le texte « Identifiants invalides » s’affichent si les informations ne sont pas présentes. Il n’est pas possible de soumettre le formulaire si un champ est vide. |
|  | |

## Gestion des cours

Tests réalisés le 26.05.2021 sur la page possédant l’URL /admin/home.

|  |  |
| --- | --- |
| 9 | Accessibilité de la page |
| But | Vérifier que seul un admin peut accéder à la page gestion des cours |
| Action | Taper l’URL /admin/home sans être connecté. |
| Résultat | L’utilisateur est automatiquement redirigé vers la page /admin qui demande la connexion. Dans la barre de navigation l’onglet « Gestion des cours » n’est pas visible pour un utilisateur déconnecté. |

|  |  |
| --- | --- |
| 10 | Affichage des participants |
| But | Vérifier que l’administrateur puisse voir les participants à un cours |
| Action | Appuyer sur le bouton « Les participants » de l’un des cours. |
| Résultat | La liste de tous les participants du cours s’affiche en appuyant sur le bouton. Appuyer à nouveau dessus fait disparaitre la liste. Le message « Aucun participant » est affiché à la place du bouton si le cours ne contient pas encore de participant. |
|  | |

|  |  |
| --- | --- |
| 11 | Ajout d’un cours |
| But | Il est possible d’ajouter un cours. |
| Action | Appuyer sur le bouton « Ajouter un cours » sur la page /admin/home et tenter d’ajouter un cours. |
| Résultat | Le bouton redirige vers l’URL /admin/addLesson contenant un formulaire. Une fois rempli correctement et soumis. Le cours ajouté est visible sur la page /admin/home et un message confirmant l’ajout du cours s’affiche. |
|  | |

|  |  |
| --- | --- |
| 12 | Vérification des erreurs |
| But | Il n’est pas possible d’ajouter des cours contenants des options impossible |
| Action | Inscrire un jour passé, un nombre de participant négatif et une heure de fin avant l’heure de début dans le formulaire de l’URL /admin/addLesson. |
| Résultat | Des erreurs apparaissent pour chacun des cas. |
|  | |

## Inscription à un cours

Tests réalisés le 26.05.2021.

|  |  |
| --- | --- |
| 13 | Inscription à un cours |
| But | Un utilisateur peut s’inscrire à un cours disponible |
| Action | Appuyer sur le bouton « S’inscrire » d’un des cours disponibles sur la page d’accueil et tenter de s’inscrire au cours. Le test se déroule sur le cours possédant l’id 18. |
| Résultat | L’utilisateur est redirigé vers l’URL /register:18 contenant un formulaire. Les informations concernant le cours sont également présentées. Une fois le formulaire envoyé en appuyant sur « S’inscrire » un message confirment l’inscription s’affiche. |
| Après avoir appuyé sur « S’inscrire » | |

|  |  |
| --- | --- |
| 14 | Tenter de s’inscrire à un cours indisponible |
| But | Vérifier qu’il n’est pas possible de s’inscrire à un cours indisponible (cours complet ou passé) |
| Action | Lors de l’inscription à un cours dans l’URL l’id du cours est écrite. Rentrer l’URL /register:7 et /register:11 pour accéder aux cours qui possèdent l’id 7 et l’id 4. Le cours possèdent l’id 7 s’est déroulé le 20 mai 2021 et le cours possédant l’id 11 est complet. |
| Résultat | Dans les 2 cas, la page affiche le message « Il n’est pas possible de s’inscrire à ce cours ». |
|  | |

|  |  |
| --- | --- |
| 15 | Vérification des erreurs |
| But | Vérifier que seul un nom et un prénom valide peuvent être entrés. |
| Action | Sur le formulaire d’inscription un prénom invalide est rentré (contenant des chiffres ou caractères spéciaux) et un nom valide est rentré (contenant des lettres, espaces, apostrophe ou tirait). Prénom invalide : Jéré45mia435h: Nom valide : Steiner Du'Commun-Dit-Veron |
| Résultat | Pour le prénom, le message d’erreur « Veuillez entrer un prénom valide » apparaît. Pour le nom, aucun message d’erreur apparaît. |
|  | |

## Gestion du profil

Tests réalisés le 26.05.2021 sur la page possédant l’URL /admin/profil.

|  |  |
| --- | --- |
| 16 | Modification d’information personnelles |
| But | Modifier ses informations personnelles |
| Action | Rentrer un mail, prénom et nom dans le formulaire « Modifier mes informations personnelles » et le soumettre. |
| Résultat | Les informations sont correctement modifiée et le message « Modification d’informations effectuées ». |
|  | |

|  |  |
| --- | --- |
| 17 | Informations invalides |
| But | Vérifier que seul des informations valides peuvent être entrées |
| Action | Rentrer un texte qui n’est pas un mail et rentrer des caractères spéciaux ou des nombres pour le prénom et le nom. Mail : pas un mail, Prénom : test42, Nom : .\_wqe |
| Résultat | Pour chaque champ, une erreur indique que l’entrée n’est pas valide. |
|  | |

|  |  |
| --- | --- |
| 18 | Modifier son mot de passe |
| But | Modifier son mot de passe |
| Action | Entrer 2 fois le mot de passe « 3c2F.erUZ.5+ » dans le formulaire « Modifier le mot de passe » et le soumettre. Puis se déconnecter et tenter de se reconnecter. |
| Résultat | Le message « Mot de passe modifié » apparaît. Une fois déconnecté, il est possible de se connecter avec le nouveau mot de passe, mais pas l’ancien. |
|  | |

|  |  |
| --- | --- |
| 18 | Mot de passe invalide |
| But | Inscrire un mot de passe invalide |
| Action | Entrer les mots de passe suivants :   1. 2 mots de passe différents : hE5T3|k8?Ns( et +^ah&Y4r22YJ 2. Moins de 12 caractères : t5Uk[7 3. Pas de minuscules : ;Y7:5U8\_;DW7 4. Pas de majuscules : h=<)29@r7d3d 5. Pas de chiffres : MQuXb^^pfC|( 6. Pas de caractères spéciaux : B27hv3KShhF2 |
| Résultat | Dans tous les cas un message d’erreur apparaît. |
| Cas 1 |  |
| Cas 2 |  |
| Cas 3 à 6 |  |

# Conclusion

## Bilan personnel

# Glossaire

|  |  |
| --- | --- |
| Mot | Définition |
| BD |  |
| CSS |  |
| Framework |  |
| MVC |  |
| ORM |  |
| Responsive |  |
| Route |  |
| SQL |  |
|  |  |

Alphabétiquement trié

# Sources et références

|  |  |
| --- | --- |
| Information | Lien |
| Mockup | https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203293-mockup-definition-traduction/ |
| Doctrine ORM | https://symfony.com/doc/current/doctrine.html |
| Information sur Symfony | https://symfony.com/why-use-a-framework |
| Bonnes pratiques Symfony | https://symfony.com/doc/current/best\_practices.html |
| Migration Symfony | https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html |
| Informations sur Twig | https://fr.wikipedia.org/wiki/Twig |
| Information sur le modèle en cascade | <https://www.ionos.fr/digitalguide/sites-internet/developpement-web/modele-en-cascade/> |
| Gestion des accès | https://symfony.com/doc/current/security.html#denying-access-roles-and-other-authorization |
| Security.yaml | https://symfony.com/doc/current/reference/configuration/security.html |
| Formulaire Symfony | https://symfony.com/doc/current/forms.html |
| Validation Symfony | https://symfony.com/doc/current/validation.html |

# Annexes

## Journal de travail

1. https://www.ncsc.admin.ch/ncsc/fr/home/infos-fuer/infos-private/aktuelle-themen/schuetzen-sie-ihre-konten.html [↑](#footnote-ref-1)
2. https://symfony.com/doc/current/best\_practices.html#use-attributes-or-annotations-to-configure-routing-caching-and-security [↑](#footnote-ref-2)
3. https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html [↑](#footnote-ref-3)
4. https://www.doctrine-project.org/projects/doctrine-orm/en/2.8/reference/query-builder.html [↑](#footnote-ref-4)
5. https://symfony.com/doc/current/best\_practices.html#use-attributes-or-annotations-to-configure-routing-caching-and-security [↑](#footnote-ref-5)