Inscription à un cours

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

Table des matières

[1 Analyse préliminaire 4](#_Toc72845109)

[1.1 Introduction 4](#_Toc72845110)

[1.2 Objectifs 4](#_Toc72845111)

[1.3 Méthodologie de gestion de projet 4](#_Toc72845112)

[2 Analyse et Conception 5](#_Toc72845113)

[2.1 Planification initial 5](#_Toc72845114)

[2.2 Symfony 5](#_Toc72845115)

[2.3 Bootstrap 5](#_Toc72845116)

[2.4 Twig 6](#_Toc72845117)

[2.5 Mockup 6](#_Toc72845118)

[2.6 MCD et MLD 7](#_Toc72845119)

[2.7 Doctrine ORM 8](#_Toc72845120)

[2.8 Vue smartphone 9](#_Toc72845121)

[2.9 Politique des mots de passe 9](#_Toc72845122)

[3 Réalisation 10](#_Toc72845123)

[3.1 Routes 10](#_Toc72845124)

[3.2 Entités 11](#_Toc72845125)

[3.2.1 Environnement 11](#_Toc72845126)

[3.2.2 Création d’une entité 11](#_Toc72845127)

[3.2.3 Migration 14](#_Toc72845128)

[3.2.4 Les requêtes 14](#_Toc72845129)

[3.3 Le paquet « Security » 16](#_Toc72845130)

[3.3.1 L’entité « User » 16](#_Toc72845131)

[3.3.2 Formulaire de connexion 17](#_Toc72845132)

[3.3.3 Gestion des accès 17](#_Toc72845133)

[3.3.4 Le fichier « security.yaml » 18](#_Toc72845134)

[3.4 Les formulaires 21](#_Toc72845135)

[3.4.1 Création d’un formulaire 21](#_Toc72845136)

[3.4.2 Configuration du formulaire 21](#_Toc72845137)

[3.4.3 Validation des champs 21](#_Toc72845138)

[3.5 Le mot de passe 22](#_Toc72845139)

[3.5.1 Génération d’un nouveau mot de passe 22](#_Toc72845140)

[3.5.2 Envoi d’un mail 22](#_Toc72845141)

[3.6 La pagination 22](#_Toc72845142)

[4 Tests 23](#_Toc72845143)

[4.1 Page d’accueil 23](#_Toc72845144)

[4.2 Connexion d’un administrateur 23](#_Toc72845145)

[5 Conclusion 24](#_Toc72845146)

[6 Glossaire 24](#_Toc72845147)

[7 Sources et références 25](#_Toc72845148)

[8 Annexes 25](#_Toc72845149)

[8.1 Journal de travail 25](#_Toc72845150)

# 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

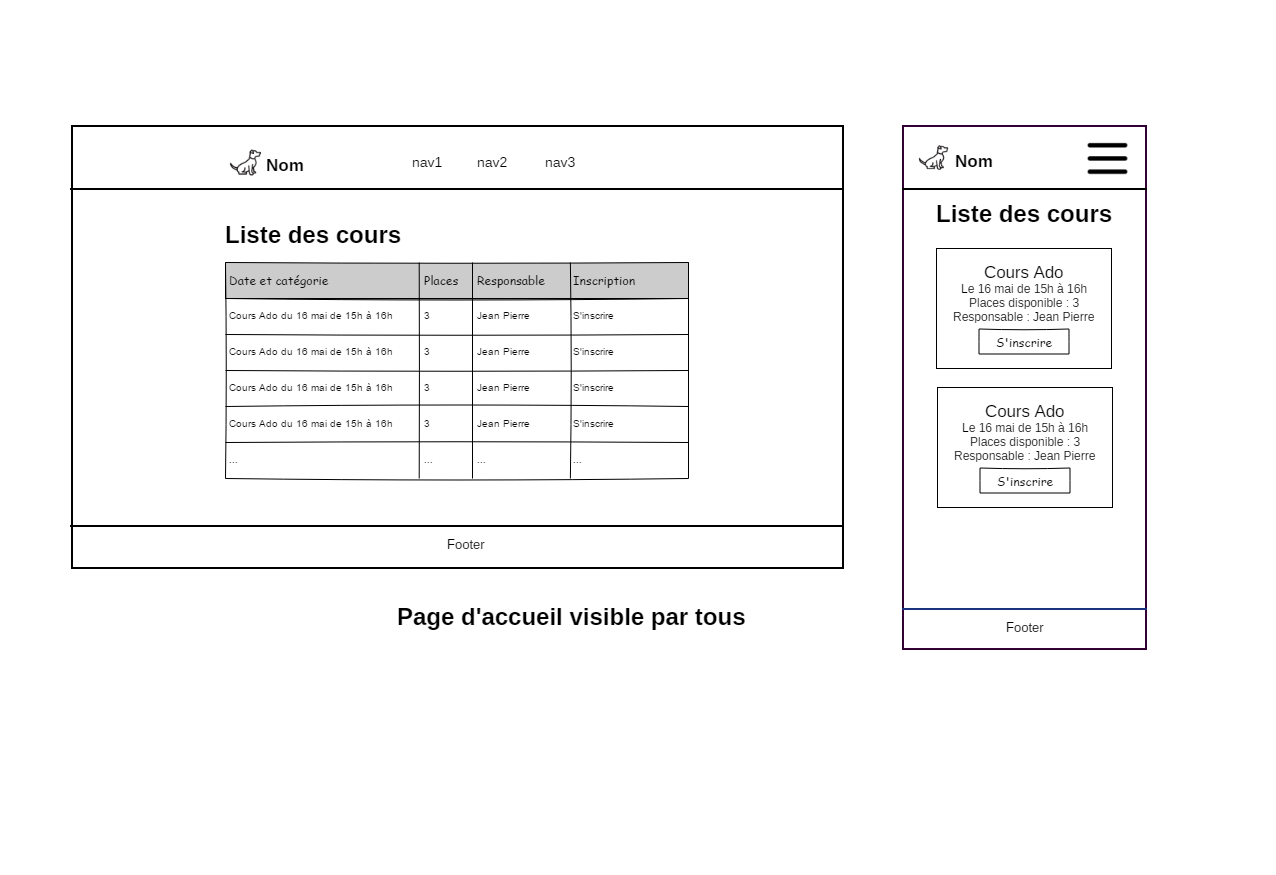
Une mockup est une maquette permettant de représenter le site web de manière simpliste.

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.

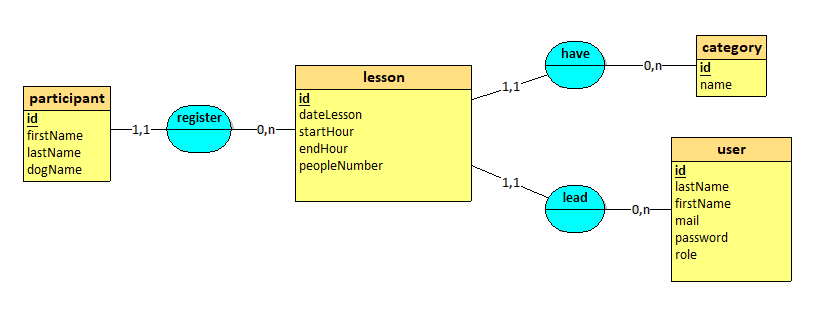
* Non redondance : chaque information n’apparaît qu’une fois
* Atomicité : une seule information par champ
* Unicité : chaque champ est identifiable de manière unique

Figure 3 MCD de la base de données

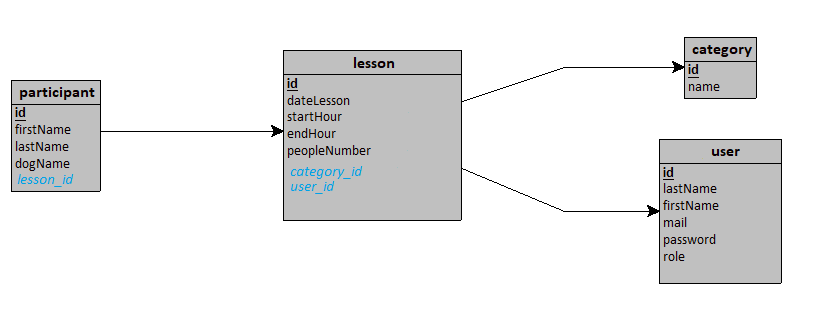
Dans le CDC, une liste des caractéristiques d’un cours est donnée. Cependant, créer un type d’entité directement d’après cette liste ne respecterait pas les règles d’élaboration d’une BD. L’une des caractéristiques est qu’un cours possède « Une catégorie de cours (ex : Ado) ». Ajouter, sur le type d’entité « 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

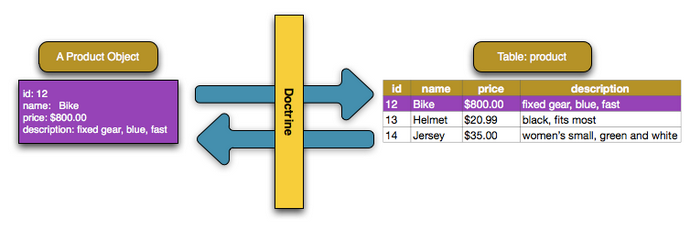
Doctrine est l’ORM (object-relational mapping) utilisé par Symfony. Un ORM est un couche d’abstraction qui synchronise les objets PHP avec une base de données relationnelle. Son but est de créer un ensemble de classe permettant de manipuler les tables et données de la BD. Il permet de transformer les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

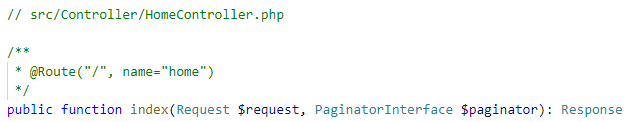
Il est possible de définir les routes de plusieurs façons, : via des attributs ou des annotations, dans des fichiers YAML, XML ou PHP. Selon les [bonnes pratiques](https://symfony.com/doc/current/best_practices.html#use-attributes-or-annotations-to-configure-routing-caching-and-security)[[2]](#footnote-2)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

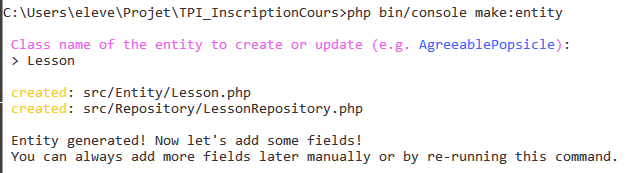
La création d’une entité se fait via une ligne de commande.

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.

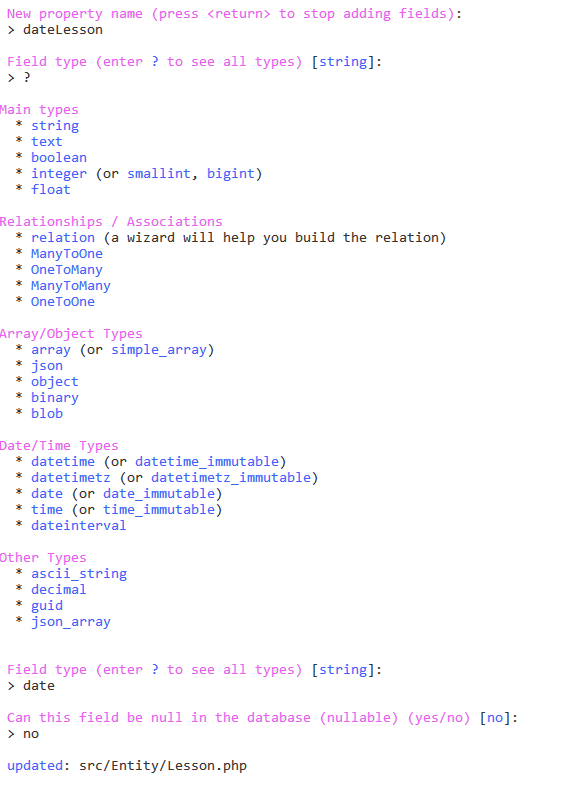
L’invite de commande proposera ensuite d’ajouter des propriétés à l’entités créées.

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.

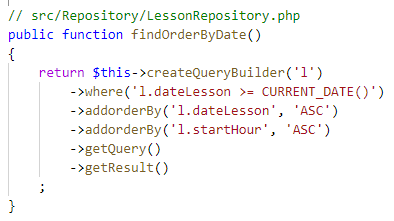
Cependant, tout cela n’est pas suffisant pour ce projet. Les utilisateurs ont besoin de voir tous les cours ne s’étant pas encore déroulés et pour des raisons de lisibilités dans un ordre précis. Doctrine permet via le « QueryBuilder » d’écrire des requêtes en utilisant du PHP.

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 »

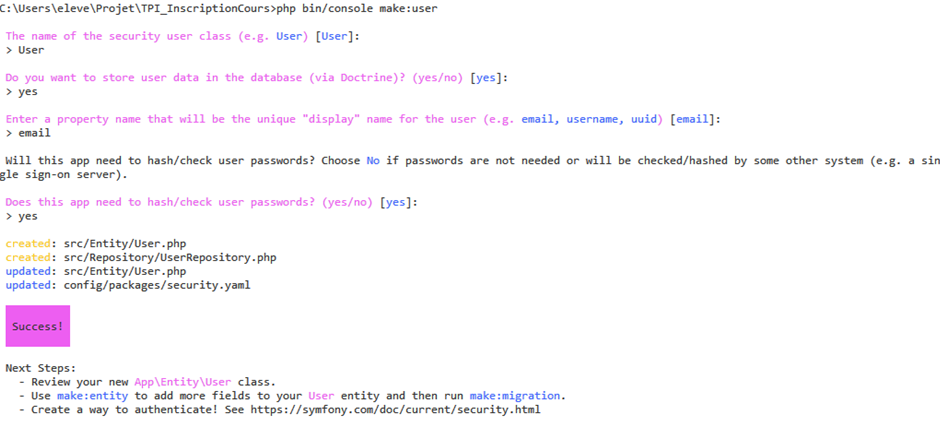
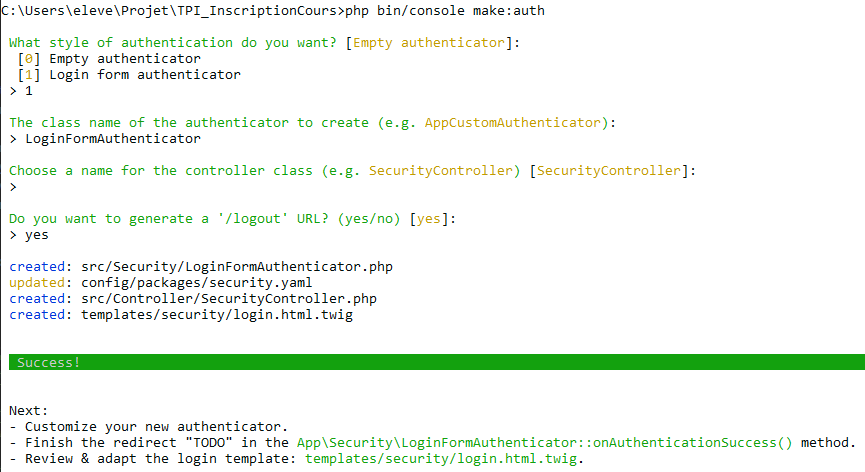
Les informations des utilisateurs seront stockées dans l’entité « User ». Elle doit obligatoirement implémenter « Symfony\Component\Security\Core\User\UserInterface » pour être utilisée. Pour générer facilement cette entité spéciale, une commande est disponible.

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

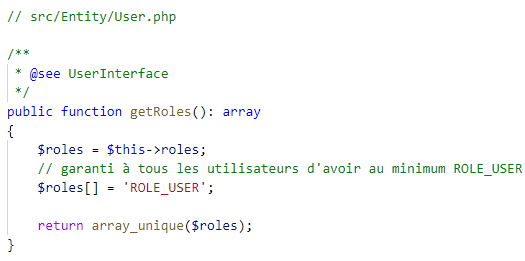
Les utilisateurs peuvent maintenant se connecter et se déconnecter. Cependant, cela ne donne aucun accès aux utilisateurs et n’en restreint aucun, il faut alors les configurer. Par défaut, la class User possède une propriété utilisée pour la gestion des accès. Il s’agit de la propriété « roles » qui 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()

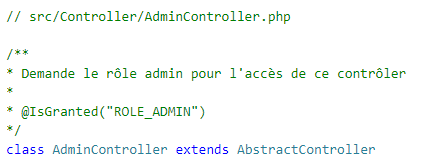
Il est possible d’attribuer d’autres rôles aux utilisateurs, mais ils doivent respecter une règle : commencer par « ROLE \_ » comme par exemple « ROLE\_ ADMIN ». Il est possible de configurer les différentes restrictions d’accès dans le fichier security.yaml ou dans le contrôleur (ou dans d’autre partie 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 »

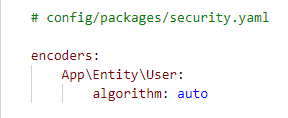
Plusieurs fois lors des étapes précédentes, l’invite de commande indiquait que le fichier « security.yaml » avait été mis à jour. Il contient la configuration du paquet « Security » et permet son fonctionnement. Il est séparé en plusieurs options.

Figure 17 L'option « encoders »

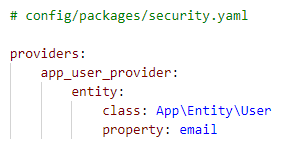
La première option est « encoders » elle permet de déterminer l’algorithme utilisé pour le hachage. Le nom fait plus penser à de l’encodage que du hachage, mais il s’agit bien de hachage ce nom est présent pour des raisons historiques. La classe sur la quel s’applique cette algorithme est à préciser 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 »

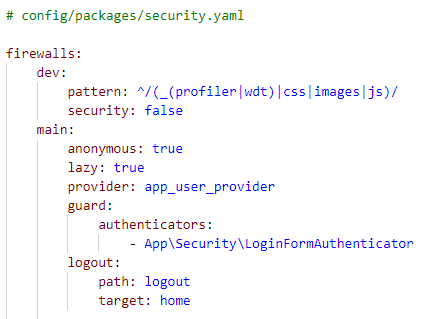
L’option suivante est « providers ». Il permet d’indiquer quelle entité représente les utilisateurs et quelle propriété est à utiliser pour l’authentification. Ces paramètres ont été générés automatiquement à la création de la classe « User ». La classe « User Providers » permet également de l’actualisation 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

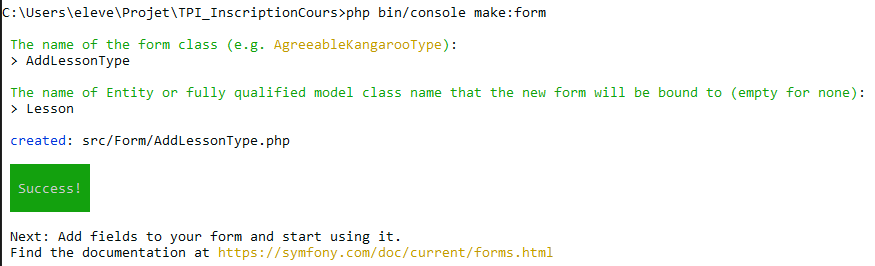
Une commande permet de générer automatiquement un formulaire.

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 le formulaire est ensuite généré automatiquement.

### Configuration du formulaire

Une fois le formulaire généré certaine modification peuvent être à faire pour le rendre plus pertinent.

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

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 divers autres options permettent de modifier l’affichage ou le fonctionnement d’un champ.

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.

### Affichage et envoie des données

### Validation des champs

## 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.

|  |  |  |
| --- | --- | --- |
| Tests | Résultat | Commentaires |
| Tous les cours s’affichent sauf ceux passés | Valide | Ils s’affichent triés par jour puis par heure. |
| Le bouton disparaît si le cours a lieu le même jour. | Valide |  |
| Le texte « cours complet » s’affichent pour les cours complet | Valide | Le texte s’affiche et le bouton d’inscription disparaît. |
| Il n’est pas possible d’accéder au bouton disparu via le F12. | Valide | La balise <li> contenant normalement le bouton est vide. |
| La pagination est automatique. | Valide | Le nombre de page est automatiquement générés en fonction du nombre de cours disponible. |
| La page est responsive. | Valide | 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.

|  |  |  |
| --- | --- | --- |
| Tests | Résultat | Commentaires |
| Le formulaire de connexion est accessible via l’URL /admin. | Valide |  |
| Un administrateur inscrit dans la base de données peut se connecter | Valide | L’administrateur de test [admin@gmail.com](mailto:admin@gmail.com) peut se connecter. |
| Un message d’erreur s’affiche si les données rentrées ne correspondent pas à la BD | Valide | Le message « Identifiants invalides. » s’affichent. Il n’est pas possible de soumettre le formulaire si un des champs est vide.  Enlever le « requied » via le F12 permet de soumettre le formulaire avec un champ vide, mais le message « Identifiants invalides. » apparaitra. |

# Conclusion

# Glossaire

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 |

# 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)