

DOSSIER DE SYNTHÈSE

Présenté par Nemanja ALABIC

Formation « Développeur Web et web mobile »
22. 10. 2018. – 05. 07. 2019.

Sommaire

INTRODUCTION.....	4
ORGANISME DE FORMATION.....	4
PRÉSENTATION	5
OBJECTIFS DU PROJET	5
RGPD.....	5
SPÉCIFICATIONS	6
LANGAGES UTILISÉS.....	6
<i>PHP</i>	6
<i>JavaScript</i>	6
<i>HTML</i>	6
<i>CSS</i>	6
FRAMEWORKS ET BIBLIOTHÈQUES	6
<i>Symfony</i>	6
<i>Twig</i>	6
<i>Bootstrap</i>	7
<i>jQuery</i>	7
<i>Leaflet</i>	7
OUTILS	7
<i>AJAX</i>	7
<i>Composer</i>	7
<i>Draw.io</i>	7
<i>WampServer</i>	8
<i>Jmerise</i>	8
<i>Visual code studio</i>	8
<i>MySQL</i>	8
<i>GitHub</i>	8
RÉALISATION.....	9
ORGANISATION DU TRAVAIL.....	9
INSTALLATION DE SYMFONY	10
<i>Liste des répertoires</i>	10
LA BASE DE DONNÉES.....	12
<i>MCD et MLD</i>	12
REQUÊTES HTTP PAR SYMFONY	13
COMMENT FONCTIONNE UNE ARCHITECTURE MVC ?	14
SWIFTMAILER.....	15
CSS3 MEDIA QUERIES.....	16
AJAX	18
SÉCURITÉ ET GESTION DES UTILISATEURS	20
L'AUTHENTIFICATION	20
L'AUTORISATION.....	20
<i>Exemples</i>	21
TWIG	25
<i>Avantages de Twig</i>	25
<i>Le langage Twig</i>	26
PHASE DE TEST	27

RECHERCHES ET VEILLE TECHNOLOGIQUE	31
VEILLE	31
RECHERCHES SUR UN SITE ANGLOPHONE.....	31
DIFFICULTÉS	35
LA GESTION DU PANIER	35
AXES D'AMÉLIORATIONS DE L'APPLICATION	35
<i>Jeton de confirmation de l'inscription.</i>	<i>36</i>
<i>Ajout de gestion de stock</i>	<i>36</i>
<i>Désinscription d'utilisateurs</i>	<i>36</i>
<i>Barre de recherche</i>	<i>36</i>
<i>Possibilité de répondre aux commentaires des utilisateurs.....</i>	<i>36</i>
CONCLUSION	37
ANNEXES	38
PRÉSENTATION DU MCD ET MLD	38

Introduction

Je m'appelle Nemanja ALABIC, j'ai 27 ans.

J'ai fait mes études de droit judiciaire entre 2010 et 2014 mais mon intérêt pour l'informatique et les encouragements de mon entourage m'ont orienté vers la programmation.

Grace à une motivation accrue j'ai eu l'opportunité d'effectuer la formation de « Développeur web et web mobile » à Elan Formation et aussi de pouvoir obtenir un titre RNCP III qui me permettra de continuer à approfondir mes connaissances en licence professionnelle.

Organisme de Formation



ELAN formation, c'est plus de 25 ans d'expérience dans les domaines de la Bureautique, la PAO, le Multimédia, d'internet, des Techniques de secrétariat. C'est des formations sur mesure et totalement individualisées. Possibilité de montrer un dossier en partenariat avec des OPCA (Organismes Paritaires Collecteurs Agréés) et optimiser ainsi local. A ce titre, il dispose de locaux sur Strasbourg, Sélestat, Haguenau, Saverne, Colmar, Mulhouse, Metz et Nancy

Présentation

Objectifs du projet

Pourquoi un site comme cela ? J'ai toujours cherché un site similaire à « IMDB » (Internet Movie Database) mais pour les livres. C'est le site « Goodreads » qui m'a inspiré. Actuellement, la vente des livres est possible en version numérique. Aussi mon projet permet aux individus de faire des recherches dans une base de données conséquente comprenant des livres, des annotations et des critiques littéraires. Les utilisateurs peuvent créer un compte pour consigner leurs impressions de lecture, ce qui génère des listes de suggestions personnalisées.

RGPD

Le règlement européen (RGPD ou GDPR) définit le régime de protection des données en Europe dans tous les secteurs d'activités, organismes privés ou publics :

- Les droits des personnes propriétaires des données,
- Obligation de désigner un DPO, interne ou externe,
- Obligation de tenue du registre des traitements,
- Apporter la preuve du respect des exigences,
- Responsabilisation des sous-traitants,
- Amendes jusqu'à 20 millions d'euros ou 4% du CA mondial,
- Analyses d'impact obligatoires,
- Notification des failles de sécurité

Les entreprises devront être en mesure de démontrer leur conformité avec la réglementation en cas de contrôle de la CNIL. Cette mesure se traduit par l'obligation de tenue d'un registre des traitements. Ce registre permettra de constituer une base de données des traitements, mais pourra aussi servir à centraliser et à suivre toutes les démarches de conformité mises en œuvre par l'entreprise.

Spécifications

Langages utilisés

PHP

PHP (HyperText Preprocessor) est un langage de programmation libre, principalement utilisé pour produire des pages Web dynamiques. PHP est un langage impératif orienté objet. PHP a permis de créer un grand nombre de sites Web célèbres, comme Facebook, YouTube, Wikipédia, Google, etc. Il est aujourd'hui considéré comme la base de la création des sites Internet dynamiques.

JavaScript

JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives

HTML

HTML est le format de données conçu pour représenter les pages Web. C'est un langage de balisage permettant d'écrire de l'hypertexte, d'où son nom. HTML permet de mettre en Forme le contenu des pages, d'inclure des ressources multimédias telles que des images, des sons, des vidéos, des formulaires de saisie, et des programmes informatiques. HTML5 désigne souvent un ensemble de technologies Web (HTML, CSS3 et JavaScript) Permettant notamment le développement d'applications.

CSS

CSS (les feuilles de style en cascade), Généralement appelées CSS, forment un langage informatique qui décrit la présentation des documents HTML. CSS devient couramment utilisé dans la conception de sites web et bien pris en charge par les navigateurs web. Des évolutions récentes de CSS permettent d'inclure des animations aux applications sans utiliser nécessairement de JavaScript.

Frameworks et bibliothèques

Symfony

Symfony est un ensemble de composants PHP ainsi qu'un framework MVC libre écrit en PHP. Il fournit des fonctionnalités modulables et adaptables qui permettent de faciliter et d'accélérer le développement d'un site web.

Twig

Twig est un moteur de template utilisé pour des pages HTML recevant du PHP (plus de détails page 26)

Bootstrap

Bootstrap est une collection d'outils utiles à la création du design (graphisme, animation et interactions avec la page dans le navigateur, etc.) de sites et d'applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option. C'est l'un des projets les plus populaires sur la plate-forme de gestion de développement GitHub.

jQuery

jQuery est une bibliothèque JavaScript libre et multiplateforme créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web. Le but de la bibliothèque étant le parcours et la modification du DOM (y compris le support des sélecteurs CSS 1 à 3 et un support basique de XPath), elle contient de nombreuses fonctionnalités : Des animations, la manipulation des feuilles de style en cascade (accessibilité des classes et attributs), la gestion des événements, etc. L'utilisation d'Ajax est facilitée et de nombreux plugins sont présents.

Leaflet

Leaflet est une bibliothèque JavaScript libre de cartographie en ligne. Elle est notamment utilisée par le projet de cartographie libre et ouverte OpenStreetMap.

Outils

AJAX

AJAX est l'acronyme d'Asynchronous JavaScript and XML, ce qui, transcrit en français, signifie « JavaScript et XML asynchrones ».

Derrière ce nom se cache un ensemble de technologies destinées à réaliser de rapides mises à jour du contenu d'une page Web, sans qu'elles nécessitent le moindre rechargement visible par l'utilisateur de la page Web.

Composer

Composer est un logiciel gestionnaire de dépendances libre écrit en PHP. Il permet à ses utilisateurs de déclarer et d'installer les bibliothèques dont le projet principal a besoin. Le développement a débuté en avril 2011 et a donné lieu à une première version sortie le 1er mars 2012. Le dépôt principal de Composer est le site web Packagist10, qui permet notamment la recherche de bibliothèques et leur entreposage centralisé.

Draw.io

Draw.io est une application gratuite en ligne, accessible directement depuis son navigateur qui permet de dessiner des diagrammes ou des organigrammes. Cet outil nous propose de concevoir toutes sortes de diagrammes, de dessins vectoriels, de les enregistrer au format XML puis de les exporter dans différents formats disponibles.

WampServer

WampServer est une plateforme de développement Web permettant de faire fonctionner localement (sans avoir à se connecter à un serveur externe) des scripts PHP. WampServer n'est pas en soi un logiciel, mais un environnement comprenant trois serveurs (Apache, MySQL et MariaDB), un interpréteur de script (PHP), ainsi que phpMyAdmin pour l'administration Web des bases MySQL.

Jmerise

Logiciel dédié à la modélisation des modèles conceptuels de données (MCD). Il permet la généralisation et la spécialisation des entités, la création des relations et des cardinalités ainsi que la généralisation des modèles logiques de données (MLD) et des script SQL.

Visual code studio

Visual code studio est un environnement de développement (integrated development environment en anglais), et un éditeur de texte destiné à la programmation.

MySQL

MySQL est un système de gestion de bases de données relationnelles (SGBDR). Il est distribué sous une double licence GPL et propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle, Informix et Microsoft SQL Server.

GitHub

GitHub est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. GitHub propose des comptes professionnels payants, ainsi que des comptes gratuits pour les projets de logiciels libres. Le site assure également un contrôle d'accès et des fonctionnalités destinées à la collaboration comme le suivi des bugs, les demandes de fonctionnalités, la gestion de tâches et un wiki pour chaque projet.

Réalisation

Organisation du travail

J'ai procédé en 4 étapes pour faire cette application. La première étape était la maquette d'interface graphique. Pour la réaliser, j'ai utilisé « draw.io ».



Dès le début, j'ai gardé ce plan, mais au fur et à mesure, je changeais d'avis lorsque j'avais nouvelles idées.

La deuxième étape consistait à créer une base de données. Créer le MCD (création des tables, des entités et des relations) et le MLD qui correspond aux dépendances entre les tables et la création des clés étrangères.

La troisième étape était la partie back-end.

Afin de respecter le design pattern MVC, l'utilisation d'un framework a été préféré. Un framework permet de créer les bases de la structure du projet à travers un environnement de travail cohérent en respectant le design pattern MVC, tout en évitant la répétition de tâches évidentes pour se concentrer sur le cœur de l'application.

La dernière étape était la partie front-end, notamment ajout des feuilles de style aux mes pages et la création des « CSS3 media queries » pour qu'elles s'adaptent complètement à la taille d'écran, quel que soit le point de rupture.

Installation de Symfony

Pour pouvoir installer Symfony je devais utiliser le Composer. Cet outil ne fait absolument pas partie de Symfony, mais son usage est tellement omniprésent dans la communauté Symfony, et même dans la communauté PHP dans son ensemble. Composer est un outil pour gérer les dépendances en PHP. Les dépendances, dans un projet, ce sont toutes les bibliothèques dont notre projet dépend pour fonctionner. Par exemple, si le projet utilise la bibliothèque SwiftMailer pour envoyer des e-mails, il « dépend » donc de SwiftMailer. SwiftMailer est une dépendance dans le projet.

Comme c'est un outil PHP, j'ai eu besoin avant toute chose de vérifier l'environnement PHP. En effet il faut que PHP soit bien configuré pour pouvoir s'exécuter via la console. Sur invite de commandes, j'ai exécuté la commande :

```
php -v
```

Avec la version 7.2.4 de PHP je pouvais procéder à installer le Composer. D'abord je l'ai téléchargé depuis ce lien :

```
getcomposer.org/Composer-Setup.exe.
```

Une fois qu'il était téléchargé et installé j'ai pu installer Symfony lui-même avec la commande :

```
composer create-project symfony/website-skeleton library
```

Il a tout d'abord créé un projet dans le répertoire Symfony, puis il a téléchargé toutes les dépendances de ce projet. Pour mettre à jour toutes les dépendances d'un projet, il faut exécuter la commande suivante :

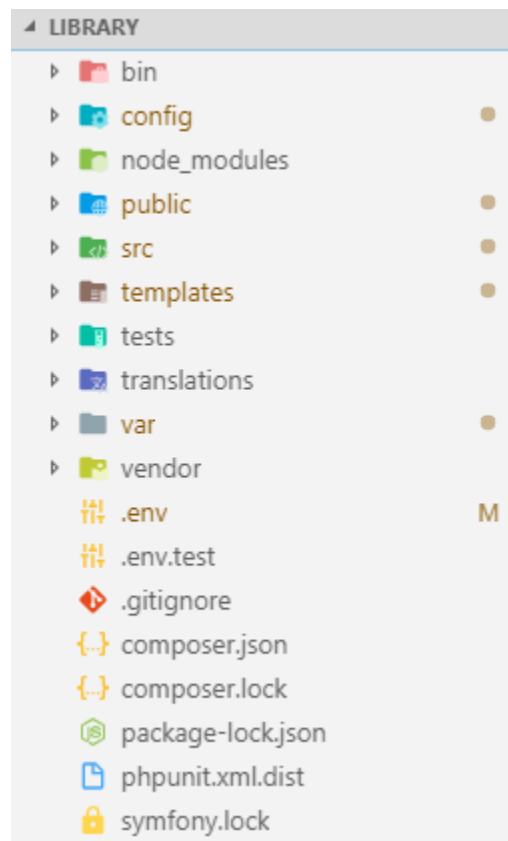
```
composer update
```

Liste des répertoires

Il n'y a pas beaucoup de fichiers ici, seulement des répertoires. En effet, tout est bien rangé dans chaque répertoire :

Le répertoire « /bin » contient tous les exécutable dont je vais me servir pendant le développement. Par exécutable, j'entends des commandes PHP, comme je l'ai fait avec Composer pour installer Symfony.

- Le répertoire « /config » contient toute la configuration du site. C'est ici que j'ai configuré Symfony lui-même, mais aussi les plugins (ou bundles) que j'ai installés par la suite. La configuration des différentes parties est éclatée dans différents répertoires et fichiers à l'intérieur de « /config ».
- Le répertoire « /public » contient tous les fichiers destinés aux visiteurs : images, fichiers CSS et JavaScript, etc. Il contient également le contrôleur frontal (index.php), dont je parlerai juste après. En fait, c'est le seul répertoire qui devrait être accessible aux visiteurs, d'où son nom : « public ». Les autres répertoires ne sont pas censés être accessibles (les fichiers de code source).



Dans ce répertoire nous trouvons le fichier « index.php », le contrôleur frontal (front controller, en anglais) qui est le point d'entrée de mon application. C'est le fichier par lequel passent toutes mes pages.

- Le répertoire « /src » dans lequel j'ai mis le code source ! C'est ici que j'ai passé le plus clair de mon temps. Sauf mes contrôleurs, entités, formulaires et repositories il contient aussi le noyau « Kernel ».
- Le répertoire « /templates » contient tous les templates de mon application. Dans le modèle MVC, cela correspond aux vues.
- Le répertoire « /var » contient tout ce que Symfony écrit durant son exécution : les logs, le cache, et d'autres fichiers nécessaires à son bon fonctionnement. Il ne faut jamais écrire dedans par soi-même.
- Le répertoire « /vendor » contient toutes les bibliothèques externes à l'application. Une bibliothèque est une sorte de boîte noire qui remplit une fonction bien précise, et dont nous pouvons se servir dans notre code. Par exemple, la bibliothèque SwiftMailer permet d'envoyer des e-mails. Nous ne savons pas comment elle fonctionne (principe de la boîte noire), mais nous savons comment s'en servir : nous pourrions donc envoyer des e-mails très facilement, juste en apprenant rapidement à utiliser la bibliothèque.

La base de données

MCD et MLD

Avant de créer ma base de données, j'ai créé un modèle conceptuel de données dans lequel j'ai créé les différentes tables dont j'avais besoin ainsi que les différentes relations qui les relie. Le modèle conceptuel des données a pour but d'écrire de façon formelle les données qui seront utilisées par le système d'information. Il s'agit donc d'une représentation des données, facilement compréhensible, permettant de décrire le système d'information à l'aide d'entités.

A la suite j'ai fait le modèle logique de données qui tient compte du niveau organisationnel des données. Il s'agit d'une vue logique en terme d'organisation de données nécessaire à un traitement. L'élaboration du MCD et du MLD a été réalisée sous JMerise. Les schémas sont illustrés sur les images en annexe. (Voir page 38)

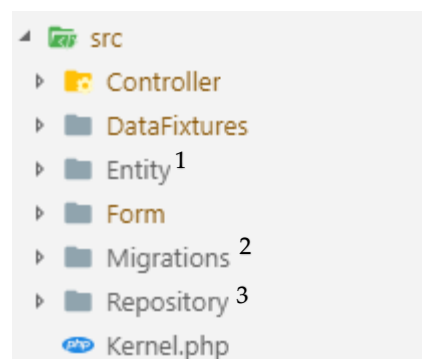
Le framework Symfony intègre par défaut Doctrine, un ORM (Object Relational Mapping ou mapping objet-relationnel en français) lui-même basé sur Doctrine « DBAL » (Couche d'abstraction de base de données ou DataBase Abstraction Layer) permettant de créer une base de données et de faire correspondre chacun de mes champs dans une table physique aux attributs d'un objet via un fichier de « mapping » (format YAML, XML ou annotations dans un fichier PHP). Ainsi un enregistrement correspondra à une instance (et vice versa) et une table correspond à une classe (ou entité). Doctrine reposant sur PDO, le bundle supporte les mêmes drivers.

J'ai commencé par créer la base de données grâce à la commande :

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

Suite donc à la génération de ma base de données, trois nouveaux dossiers étaient créés au sein de mon projet :

1. Un dossier « Entity » contenant mes objets nouvellement créés
2. Un dossier « Migrations » contenant les fichiers de migration
3. Un dossier « Repository » encapsulant les requêtes lancées à la base de données pour chaque objet



Puis j'ai généré mes entités toujours depuis la console :

```
php bin/console make:entity
```

Une fois celles-ci étaient créées j'ai généré le fichier de migration contenant les requêtes SQL :

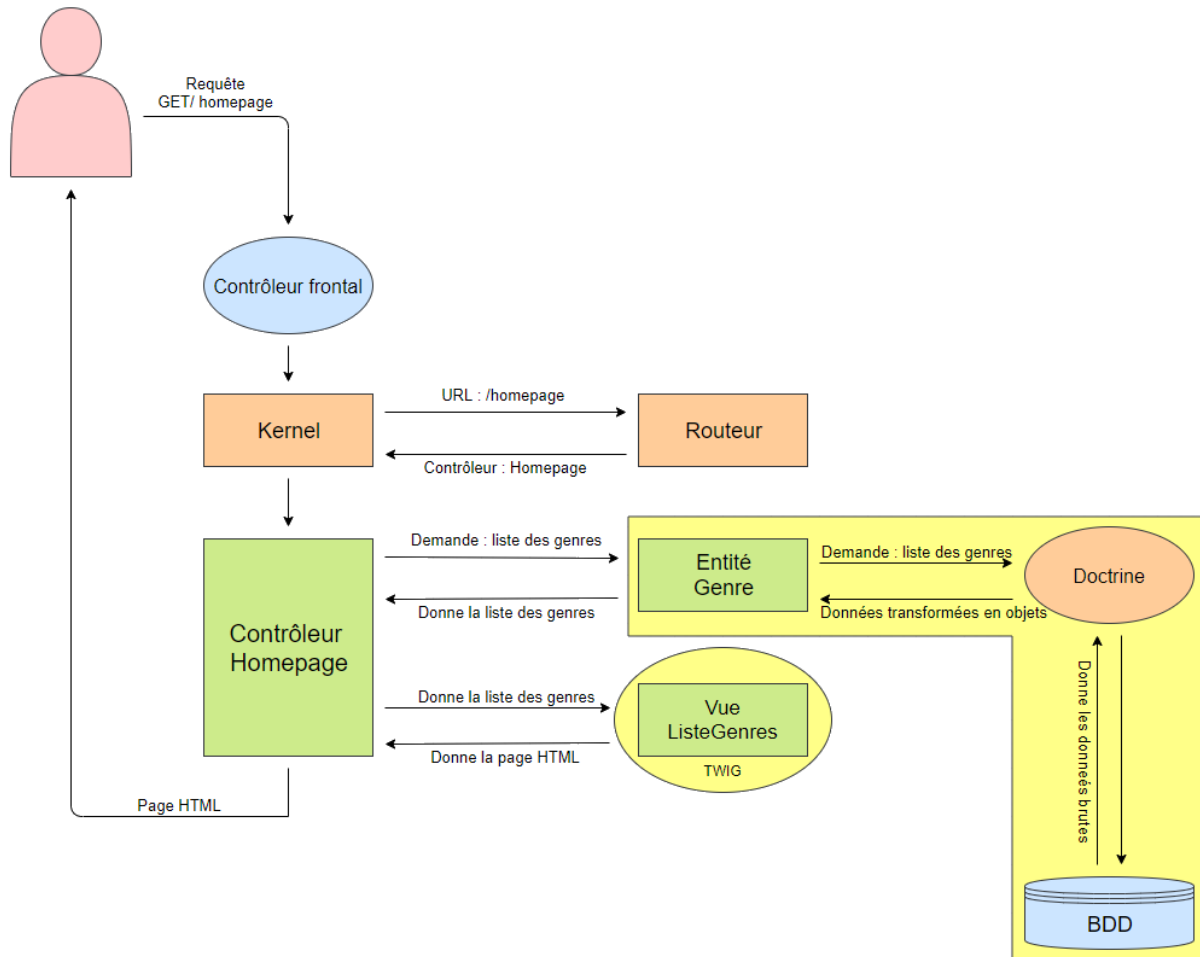
```
php bin/console make:migration
```

Puis j'ai exécuté ces requêtes :

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

La base de données est prête, les objets sont créés et hydratés.

Requêtes HTTP par Symfony



1. Le visiteur demande la page/homepage ;
2. Le contrôleur frontal reçoit la requête, charge le Kernel et la lui transmet.
3. Le Kernel demande au Routeur quel contrôleur exécuter pour l'URL/homepage. Ce Routeur est un composant Symfony qui fait la correspondance entre URL et contrôleurs. Le Routeur fait donc son travail, et dit au Kernel qu'il faut exécuter le contrôleur : Homepage ;
4. Le Kernel exécute donc ce contrôleur. Le contrôleur demande au modèle la liste des genres. Doctrine cherche dans la BDD les données. BDD renvoie les données brutes et le Doctrine les transforme en objets. Modèle donne la liste des genres au contrôleur et il la donne à la vue ListeGenres pour qu'elle construise la page HTML et la lui retourne. Une fois cela fini, le contrôleur envoie au visiteur la page HTML complète.

5. J'ai mis des couleurs pour distinguer les points où l'on intervient. En vert, les contrôleur, modèle et vue, c'est ce que nous devons développer nous-mêmes. En orange, le Kernel, le Doctrine et le Routeur, c'est ce que nous devons configurer. Nous ne toucherons pas au contrôleur frontal, en bleu.

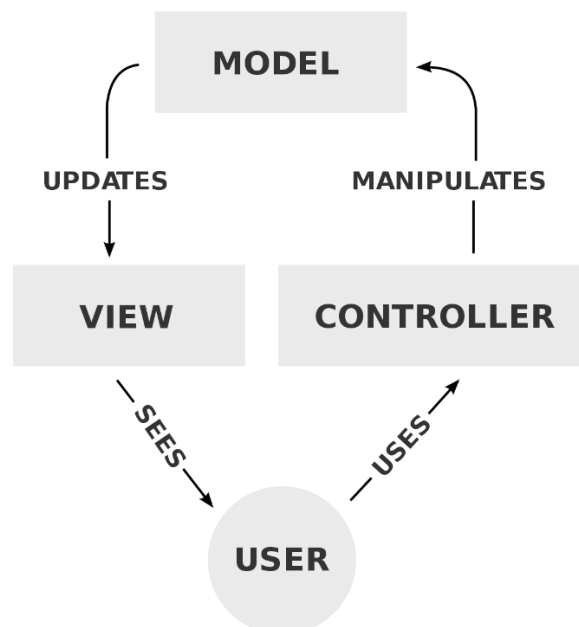
Comment fonctionne une architecture MVC ?

Design pattern est une bonne pratique de développement. Le pattern MVC permet de bien organiser notre code source et de justement séparer la logique du code en trois parties

Modèle : cette partie gère les données de notre site. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. C'est où on trouve les requêtes SQL.

Vue : cette partie se concentre sur l'affichage. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples, pour afficher par exemple une liste de messages.

Contrôleur : cette partie gère la logique du code qui prend des décisions. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. Le contrôleur contient exclusivement du PHP. C'est notamment lui qui détermine si le visiteur le droit de voir la page ou non (gestion des droits d'accès).



SwiftMailer

Comme je l'ai déjà mentionné, il existe dans Symfony un composant appelé Swiftmailer, qui permet d'envoyer des e-mails simplement. Il est présent par défaut dans Symfony, sous forme du service Mailer.

J'ai modifié le fichier « swiftmailer.yaml » :

```
{...} swiftmailer.yaml x
config › packages › {...} swiftmailer.yaml
1  swiftmailer:
2      transport:      gmail
3      username:       [REDACTED]@gmail.com
4      password:       [REDACTED]
5      host:           localhost
6      port:           465
7      encryption:     ssl
8      auth-mode:      login
9      spool: { type: 'memory' }
10     stream_options:
11         ssl:
12             allow_self_signed: true
13             verify_peer: false
14             verify_peer_name: false
```

Dès que le service est créé, et sa configuration est faite, il ne reste plus qu'à l'utiliser !

```
$url = $this->generateUrl('reset_password', array('token' => $token), UrlGeneratorInterface::ABSOLUTE_URL);

$message = (new \Swift_Message('Forgotten password'))
    ->setFrom('[REDACTED]@gmail.com')
    ->setTo($user->getEmail())
    ->setBody(
        "Token for change you password : " . $url,
        'text/html'
    );

$mailer->send($message);

$this->addFlash('notice', 'Mail sent');

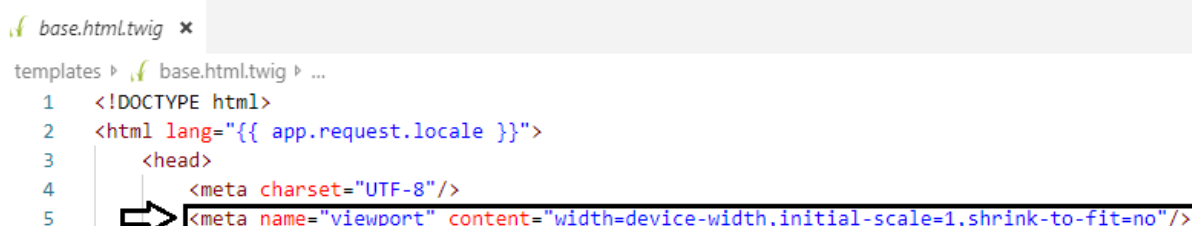
return $this->redirectToRoute('forgotten_password');
```


CSS3 Media Queries

Ce sont des règles à appliquer pour changer le design d'un site en fonction des caractéristiques de l'écran. Grâce à cette technique, je pouvais créer un design qui s'adaptait automatiquement à l'écran de chaque visiteur. Les « media queries » font partie des nouveautés de CSS3. Il ne s'agit pas de nouvelles propriétés mais de règles que nous pouvons appliquer dans certaines conditions. Concrètement, nous pourrions dire « Si la résolution de l'écran du visiteur est inférieure à tant, alors applique les propriétés CSS suivantes ». Cela nous permet de changer l'apparence du site dans certaines conditions.

Contrairement à ce que nous pourrions penser, les « media queries » ne concernent pas que les résolutions d'écran. Nous pouvons changer l'apparence du site en fonction d'autres critères comme le type d'écran (smartphone, télévision, projecteur...), le nombre de couleurs, l'orientation de l'écran (portrait ou paysage), etc. Les possibilités sont très nombreuses. Il y a deux façons de les utiliser :

- En chargeant une feuille de style « .css » différente en fonction de la règle (ex : « Si la résolution est inférieure à 1280px de large, charge le fichier `petite_resolution.css` ») ;
- En écrivant la règle directement dans le fichier « .css » habituel (ex : « Si la résolution est inférieure à 1280px de large, charge les propriétés CSS ci-dessous »).



```
base.html.twig x
templates ▸ base.html.twig ▸ ...
1 <!DOCTYPE html>
2 <html lang="{{ app.request.locale }}">
3   <head>
4     <meta charset="UTF-8"/>
5     <meta name="viewport" content="width=device-width,initial-scale=1,shrink-to-fit=no"/>
```

J'ai utilisé cette balise pour modifier la façon dont le contenu de ma page s'organise sur les mobiles. Pour obtenir un rendu facile à lire, sans zoom, j'ai demandé à ce que le « viewport » soit le même que la largeur de l'écran :

```
<!--links for CSS-->
<link rel="stylesheet" href="{{ asset('assets/css/home_style.css') }}">
<link rel="stylesheet" href="{{ asset('assets/css/role_style.css') }}">
<link rel="stylesheet" href="{{ asset('assets/css/cart_style.css') }}">
<!--media queries-->
<link rel="stylesheet" href="{{ asset('assets/css/media_queries/mobile.css') }}" media="screen and (max-width: 760px)">
<link rel="stylesheet" href="{{ asset('assets/css/media_queries/tablet.css') }}" media="screen and (max-width: 1020px)">
```

Ces balises permettent dans mon code HTML, de charger plusieurs fichiers « .css ». Et deux autres qui seront chargés en supplément uniquement si la règle correspondante s'applique. Les écrans des smartphones sont beaucoup moins larges que nos écrans habituels (seulement quelques centaines de pixels de large). Pour s'adapter, les navigateurs mobiles affichent le site en « dézoomant », ce qui permet d'avoir un aperçu de l'ensemble de la page. La zone d'affichage simulée est appelée le viewport : c'est la largeur de la fenêtre du navigateur sur le mobile.

Pour tous mes types d'écrans, si la largeur de la fenêtre ne dépasse pas 760 px, alors exécuter les règles CSS suivantes :

```
mobile.css x
public ▸ assets ▸ css ▸ media_queries ▸ mobile.css ▸ ...
1  @media screen and (max-width: 760px) {
2
3      #content {padding: 1em 5%;}
4
5      h1 {font-size: 8vw;}
6
7      h6, p, .nav__item-link {font-size: 4.5vw;}
8  }
```

Dans la ligne 5 pour mes polices j'ai utilisé la valeur 8vw. CSS3 a de nouvelles valeurs pour le dimensionnement des objets par rapport à la taille actuelle de la fenêtre : VW, VH, et vmin. Elles permettent entre autres d'avoir une taille de police et des colonnes variables selon la résolution de l'écran. Lors de la réduction du corps de la fenêtre le titre se réduira. L'unité de « vh » signifie hauteur de la fenêtre, « vw » largeur de la fenêtre, et vmin représente celui des vh ou vw est la plus courte longueur. Donc ici 8VW représente 8% de la fenêtre courante. Les unités de fenêtres sont dynamiques plutôt que statiques et si nous redimensionnons la fenêtre du navigateur, la valeur calculée des unités changera également.

La page est désormais complètement réorganisée lorsque la fenêtre fait 1020 px ou moins de largeur.



AJAX

L'AjAX est donc un ensemble de technologies visant à effectuer des transferts de données. Dans ce cas, il faut savoir structurer nos données. Il existe de nombreux formats pour transférer des données, je prends ici les quatre principaux :

- Le format texte est le plus simple, et pour cause : il ne possède aucune structure prédéfinie. Il sert essentiellement à transmettre une phrase à afficher à l'utilisateur, comme un message d'erreur ou autre. Bref, il s'agit d'une chaîne de caractères, rien de plus.
- Le HTML est aussi une manière de transférer facilement des données. Généralement, il a pour but d'acheminer des données qui sont déjà formatées par le serveur puis affichées directement dans la page sans aucun traitement préalable de la part du JavaScript.
- Un autre format de données proche du HTML est le XML, acronyme de « eXtensible Markup Language ». Il permet de stocker les données dans un langage de balisage semblable au HTML. Il est très pratique pour stocker de nombreuses données ayant besoin d'être formatées, tout en fournissant un moyen simple d'y accéder.
- Le plus courant est le JSON, acronyme de JavaScript Object Notation. Il a pour particularité de segmenter les données dans un objet JavaScript, il est très avantageux pour de petits transferts de données segmentées et est de plus en plus utilisé dans de très nombreux langages.

Il m'a servi pour afficher les citations aléatoires de plusieurs auteurs. D'abord je les ai récupérées de ma base de données et je les ai mélangées grâce à la fonction « **shuffle()** » .

```
private function randomQuote()
{
    // get all tasks
    $quotes = $this->getDoctrine()->getRepository(Quote::class)->findAll();
    // shuffle records
    shuffle($quotes);

    $quote = $quotes[0];

    $randomQuote = [
        "img" => array("src" => $quote->getAuthor()->getSrcImage(), "alt" => $quote->getAuthor()->getAltImage(), "title" => $quote->getAuthor()->getTitleImage()),
        "text" => $quote->getText(),
        "author" => $quote->getAuthor()->getName(). " " . $quote->getAuthor()->getSurname()
    ];

    return $randomQuote;
}

/**
 * @Route("/random", name="randquote")
 */
public function ajaxRandomQuote(){
    return new JsonResponse($this->randomQuote());
}
```

Ensuite, je les ai transformées en format JSON pour qu'elles puissent être utilisables exactement de la même manière dans JavaScript.

```

index.html.twig x
templates ▸ homepage ▸ index.html.twig ▸ script
40 {% block javascripts %}
41     {{ parent() }}
42     <!--script for Quotes and Leaflet-->
43     <script>
44
45         // quotes
46
47         $(document).ready(function(){
48
49             setInterval(function(){
50                 $.post(
51                     "{{ path('randquote') }}",
52                     function(data){
53                         $(".quotes").fadeOut(500, function(){
54                             $(".quotes__author-img").attr('src', '/assets/img/' + data.img.src)
55                             $(".quotes__author-img").attr('alt', data.img.alt)
56                             $(".quotes__author-img").attr('title', data.img.title)
57                             $(".quotes__content-txt em").html(data.text)
58                             $(".quotes__content-author strong").html(data.author)
59                             $(this).fadeIn();
60                         })
61                     }
62                 )
63                 }, $(".quotes__content-txt em").text().length * 75);
64     })

```

Une fois que le document est prêt, fonction JavaScript « function(data) » remplit les balises HTML avec les données du format JSON en intervalle (ligne 63). Cet intervalle est calculé avec l'aide d'une fonction JavaScript « length » qui compte le nombre de caractères de la citation et le multiplie par 75 millisecondes.

Lignes 54, 55 et 56 montrent que j'ai dû utiliser « attr(data) » pour remplir la balise au lieu du « html(data) » parce que la balise est une balise autofermante.

Sécurité et gestion des utilisateurs

La sécurité sous Symfony est très poussée, nous pouvons la contrôler très finement, mais surtout très facilement. Pour atteindre ce but, Symfony a bien séparé deux mécanismes différents : l'authentification et l'autorisation.

L'authentification

L'authentification est le processus qui va définir qui nous sommes, en tant que visiteur. L'enjeu est vraiment très simple : soit nous ne nous sommes pas identifiés sur le site et nous sommes un anonyme, soit nous nous sommes identifiés (via le formulaire d'identification ou via un cookie « Se souvenir de moi ») et nous sommes un membre du site. C'est ce que la procédure d'authentification va déterminer. Ce qui gère l'authentification dans Symfony s'appelle un firewall, ou un pare-feu en français.

Ainsi nous pourrions sécuriser des parties de notre site Internet juste en forçant le visiteur à être un membre authentifié. Si le visiteur l'est, le firewall va le laisser passer, sinon il le redirigera sur la page d'identification. Cela se fera donc dans les paramètres du firewall.

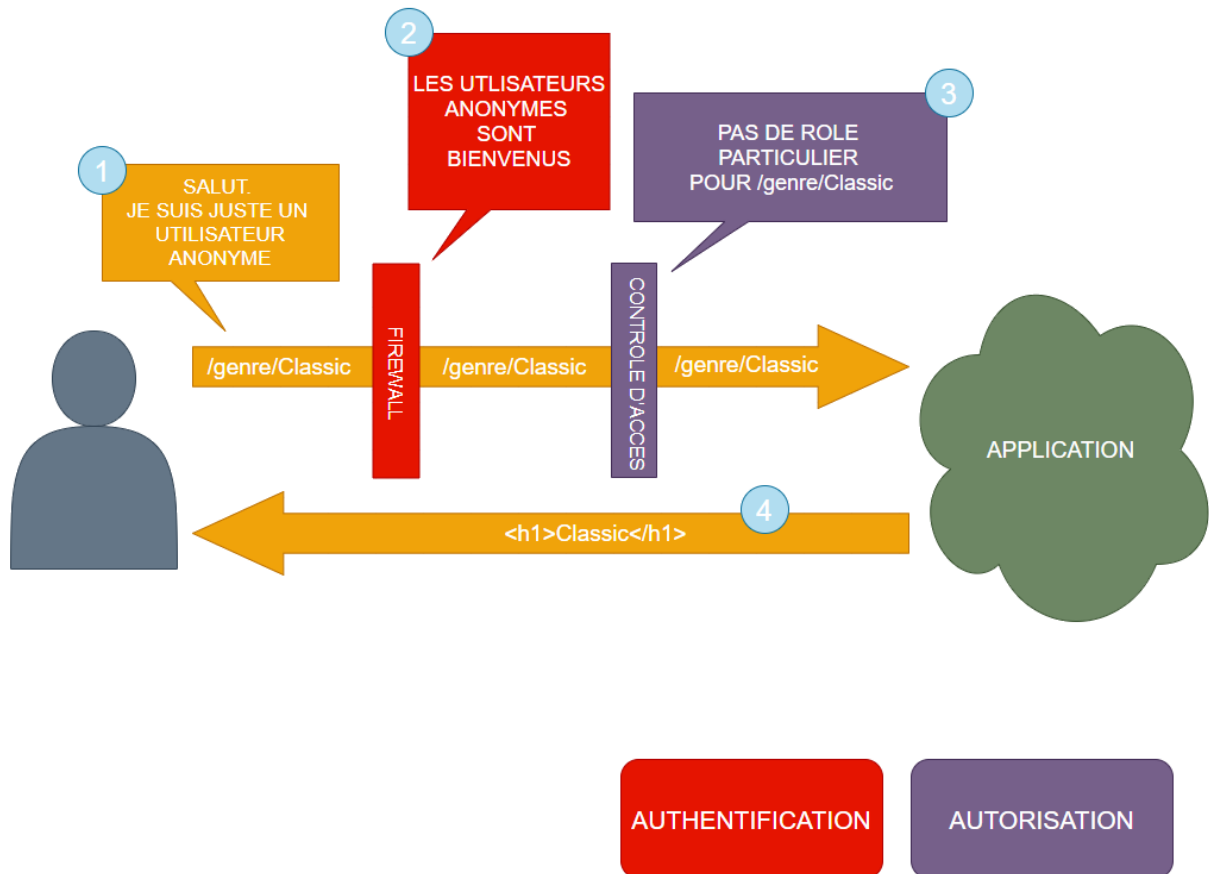
L'autorisation

L'autorisation est le processus qui va déterminer si nous avons le droit d'accéder à la ressource (la page) demandée. Il agit donc après le firewall. Ce qui gère l'autorisation dans Symfony s'appelle l'accès control, ou control d'accès en français. Par exemple, un membre identifié lambda aura accès à la liste de sujets d'un forum, mais ne peut pas supprimer de sujet. Seuls les membres disposant des droits d'administrateur le peuvent, c'est ce que l'accès control va vérifier.

Exemples

Je suis anonyme, et je veux accéder à la page/genre/Classic qui ne requiert pas de droits.

Dans cet exemple, un visiteur anonyme souhaite accéder à la page/genre/Classic. Cette page ne requiert pas de droits particuliers, donc tous ceux qui ont réussi à passer le firewall peuvent y avoir accès. Le schéma suivant montre le processus.



Le visiteur n'est pas identifié, il est anonyme, et tente d'accéder à la page/genre/Classic

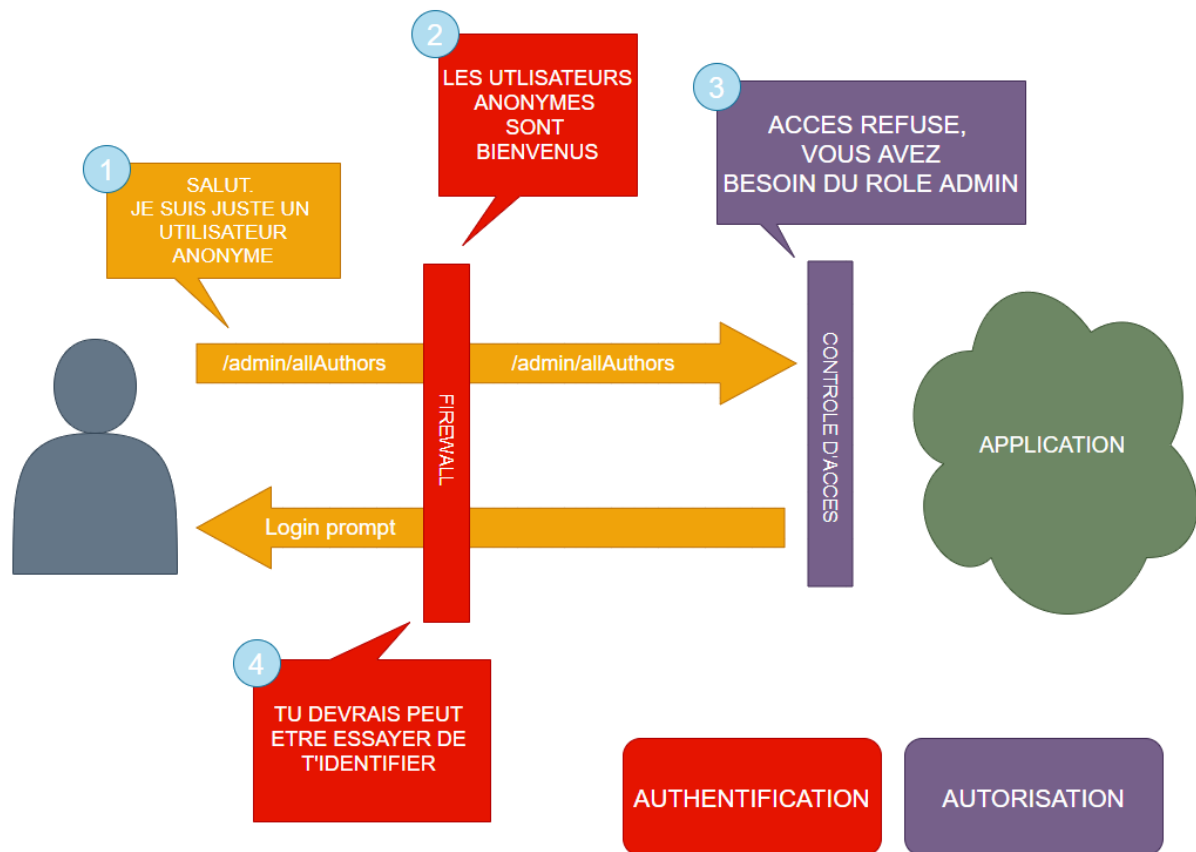
Le firewall est configuré de telle manière qu'il n'est pas nécessaire d'être identifié pour accéder à la page/genre/Classic. Il laisse donc passer le visiteur anonyme.

Le contrôle d'accès regarde si la page/genre/Classic requiert des droits d'accès : il n'y en a pas. Il laisse donc passer le visiteur, qui n'a aucun droit particulier.

Le visiteur a donc accès à la page/genre/Classic.

Je suis anonyme, et je veux accéder à la page/admin/allAuthors qui requiert certains droits

Dans cet exemple, c'est le même visiteur anonyme qui veut accéder à la page/admin/allAuthors. Mais cette fois, la page/admin/allAuthors requiert le rôle « ROLE_ADMIN ». Le visiteur va se faire refuser l'accès à la page, le schéma suivant montre comment.



Le visiteur n'est pas identifié, il est toujours anonyme, et tente d'accéder à la page/admin/allAuthors.

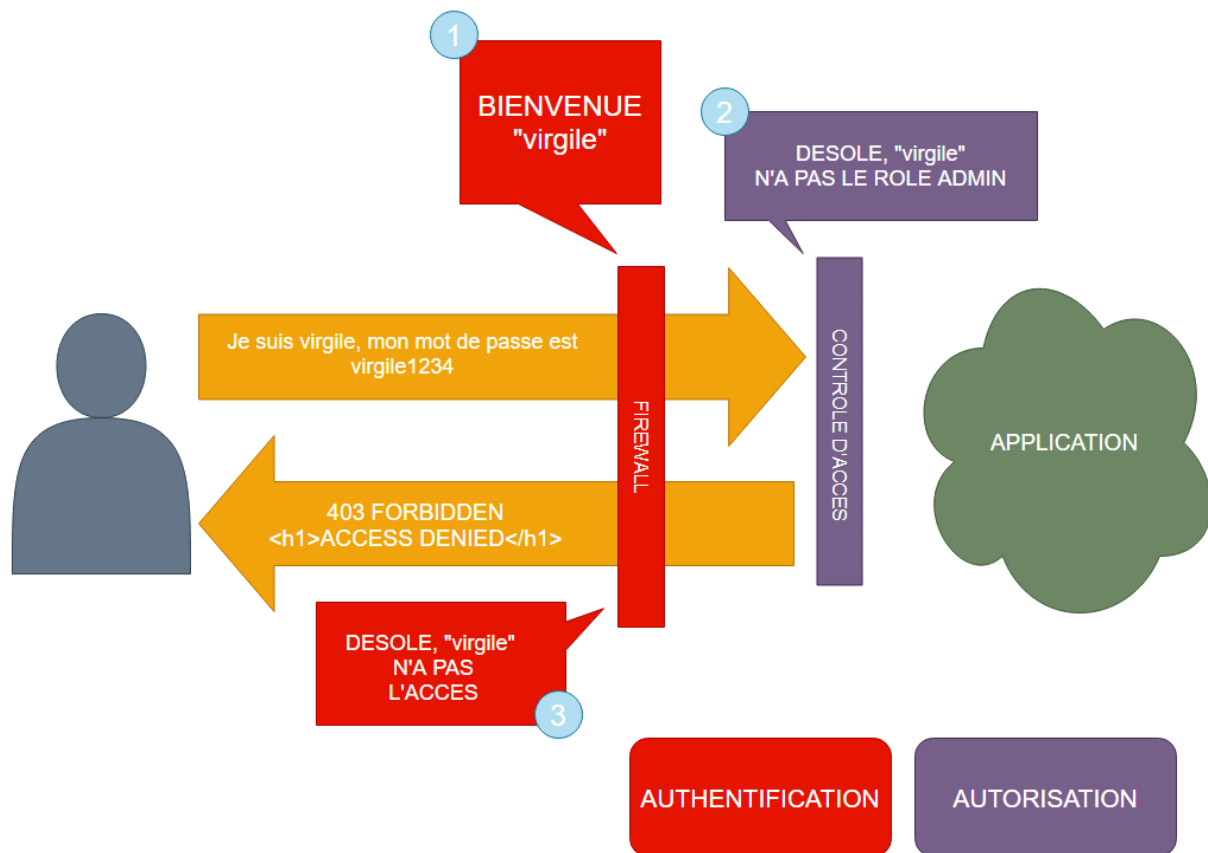
Le firewall est configuré de manière qu'il ne soit pas nécessaire d'être identifié pour accéder à la page/admin/allAuthors. Il laisse donc passer le visiteur.

Le contrôle d'accès regarde si la page/admin/allAuthors requiert des droits d'accès : oui, il faut le rôle « ROLE_ADMIN ». Le visiteur n'a pas ce rôle, donc le contrôle d'accès lui interdit l'accès à la page/admin/allAuthors.

Le visiteur n'a donc pas accès à la page/admin/allAuthors, et se fait rediriger sur la page d'identification.

Je suis identifié, et je veux accéder à la page/admin/allAuthors qui requiert certains droits.

Cet exemple est le même que précédemment, sauf que cette fois le visiteur est identifié, il s'appelle Virgile. Il n'est donc plus anonyme.



Virgile s'identifie et il tente d'accéder à la page/admin/allAuthors. D'abord, le firewall confirme l'authentification de Virgile (c'est son rôle !). Visiblement c'est bon, il laisse donc passer Virgile.

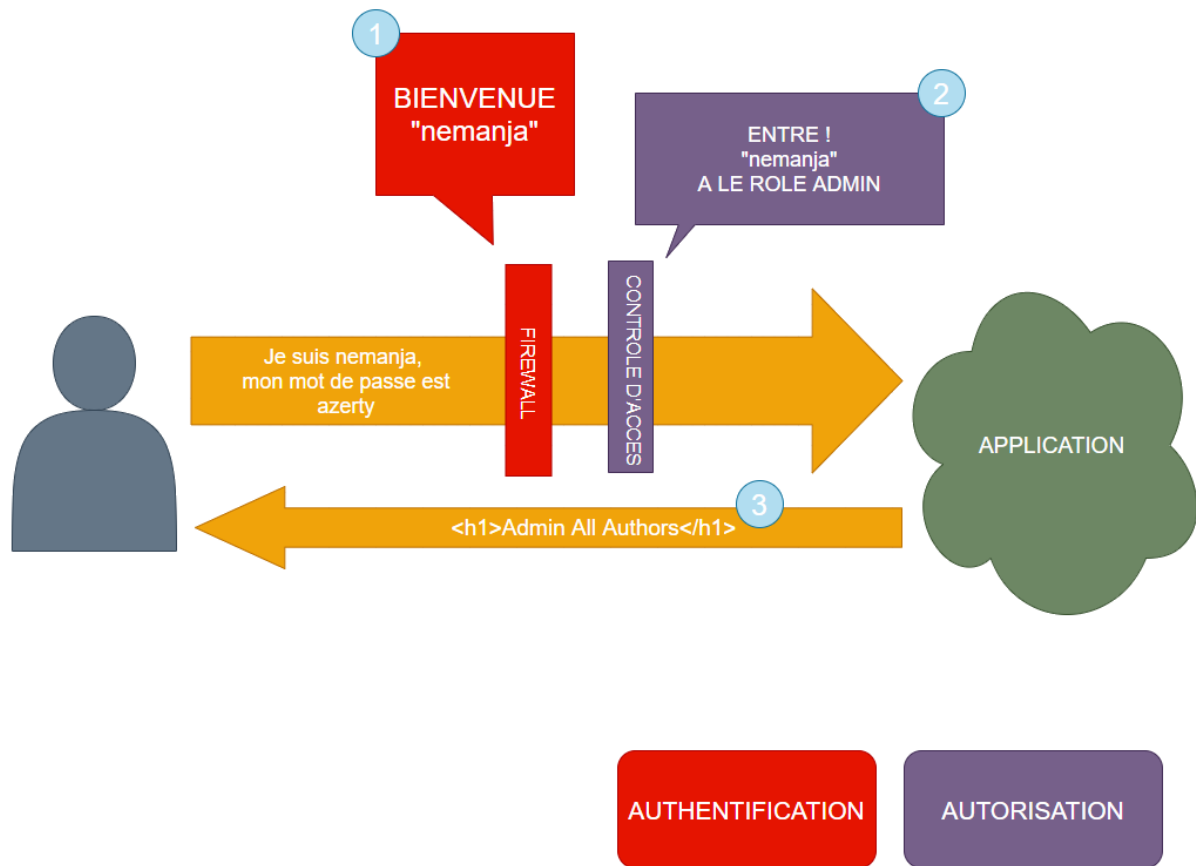
Le contrôle d'accès regarde si la page/admin/allAuthors requiert des droits d'accès : oui, il faut le rôle « ROLE_ADMIN », que Virgile n'a pas. Il interdit donc l'accès à la page/admin/allAuthors à Virgile.

Virgile n'a pas accès à la page/admin/allAuthors non pas parce qu'il ne s'est pas identifié, mais parce que son compte utilisateur n'a pas les droits suffisants. Le contrôle d'accès lui affiche donc une page d'erreur lui disant qu'il n'a pas les droits suffisants.

Je suis identifié, et je veux accéder à la page/admin/allAuthors qui requiert des droits que j'ai.

Ici, il est maintenant identifié en tant qu'administrateur, il a donc le rôle « ROLE_ADMIN » !

Du coup, il peut accéder à la page/admin/allAuthors, comme le montre le schéma suivant.



L'utilisateur Nemanja s'identifie, et il tente d'accéder à la page/admin/allAuthors. D'abord, le firewall confirme l'authentification d'admin. Ici aussi, c'est bon, il laisse donc passer admin.

Le contrôle d'accès regarde si la page/admin/allAuthors requiert des droits d'accès : oui, il faut le rôle « ROLE_ADMIN », que Nemanja a bien. Il laisse donc passer l'utilisateur.

L'utilisateur Nemanja a alors accès à la page/admin/allAuthors, car il est identifié et il dispose des droits nécessaires.

TWIG

Twig est un moteur de templates qui a été créé par SensioLabs, les créateurs de Symfony.

Avantages de Twig

- Twig permet de séparer la présentation des données du traitement

Bien souvent, afficher une donnée brute ne suffit pas. Nous voulons lui appliquer un traitement logique avant de l'afficher (une conversion, un calcul...). Twig permet de définir en dehors de la page web des filtres que nous pourrons appliquer à la donnée.

- Twig permet la personnalisation de page web.

Un block menu, un block recherche, un block contenu... le tout défini dans un template lui-même héritable.

- Twig permet de rendre les pages web plus lisibles, plus claires.

Nous avons souvent besoin de boucles, de conditions... à la construction d'une page web et pour cela, on utilise PHP. Notre page devient vite illisible, peu maintenable, sujette à de nombreuses erreurs.

TWIG [Sélectionnez](#)

```
{% for value in items if value.active %}  
    
{% endfor %}
```

PHP [Sélectionnez](#)

```
<?php foreach ($items as $value):  
  if ($value.active) {?  
      
  }  
<?php } endforeach;?>
```

- Twig par son langage est moins invasif que PHP et se substitue à celui-ci.

La conversion Twig/PHP ne se fait qu'une seule fois au premier appel, ensuite il est mis en cache pour les autres appels (et donc finalement, c'est aussi rapide qu'en PHP).

- Twig est rapide.

Twig apporte de nouvelles fonctionnalités : les macros, les filtres, les blocks, l'héritage de templates...

- Twig apporte de nouvelles fonctionnalités.

Twig apporte de nouvelles fonctionnalités : les macros, les filtres, les blocks, l'héritage de templates...

- Twig apporte plus de sécurité.

Avec Twig, plus de sécurité avec l'échappement des variables qui est activé par défaut.

Le langage Twig

{% %} : pour faire une action : un set de variable, une condition if, une boucle for ;

{{ }} : pour afficher quelque chose.

L'affichage des données et l'échappement :

<div>TWIG</div> <div><code>{{ 'coucou' }}</code></div>	<div>Sélectionnez</div>	COUCOU
<div>TWIG</div> <div><code>{% set data = '<h1>coucou</h1>' %} {{ data }}</code></div>	<div>Sélectionnez</div>	<h1>coucou</h1>

*Pour la sécurité, l'échappement est activé par défaut. Les balises ne sont pas interprétées. L'échappement convertit la donnée en code HTML avant l'affichage.

-Forcer l'interprétation des balises avec le filtre « raw » :

<div>TWIG</div> <div><code>{% set data = '<h1>coucou</h1>' %} {{ data raw }}</code></div>	<div>Sélectionnez</div>	COUCOU
---	-------------------------	--------

Une autre fonctionnalité qui m'a beaucoup plu est la possibilité d'héritage entre templates. Avec Twig, nous pouvons définir un template avec « header », « footer » et faire un héritage entre les deux templates (celui qui affiche le contenu et celui contenant le header et le footer) pour que nous n'ayons qu'à modifier un seul fichier si nous voulons changer la structure du site :

- Le mot « Extends » indique au moteur de template que ce template "étend" un autre template. Lorsque le système de template évalue ce template, il localise d'abord le parent. Cette balise est toujours la première balise du modèle.
- Le mot « Parent » est une fonction qui va nous permettre de récupérer le contenu du template du template parent de cette page.

Pour finir, Twig est facile à apprendre, les messages d'erreur sont clairs et précis et il est supporté par la plupart des éditeurs de code.

Phase de test

C'est normalement un des premiers réflexes à avoir lorsque nous demandons à l'utilisateur de remplir des informations : vérifier ce qu'il remplit ! Il faut toujours considérer que soit il ne sait pas remplir un formulaire, soit c'est un petit malin qui essaie de trouver la faille. Bref, ne jamais faire confiance à ce que l'utilisateur nous donne (« never trust user input » en anglais).

Premier jeu d'essai concernant la création d'un compte client, avec composants de formulaire : création d'une fonction fabriquant le formulaire en lui faisant passer les caractéristiques de ces champs (ex. le mot de passe sera un champ de type Password). Je ne veux pas que l'utilisateur puisse renseigner une adresse mail existante dans ma base de données et renseigner deux mots de passe différents. Pour ce deuxième point un champ de confirmation de mot de passe permet d'éviter ce point.

```
UserType.php •
src ▶ Form ▶ UserType.php ▶ ...
1  <?php
2
3  namespace App\Form;
4
5  use Symfony\Component\Form\AbstractType;
6  use Symfony\Component\Form\FormBuilderInterface;
7  use Symfony\Component\Form\Extension\Core\Type\TextType;
8  use Symfony\Component\Form\Extension\Core\Type\EmailType;
9  use Symfony\Component\Form\Extension\Core\Type\BirthdayType;
10 use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
11 use Symfony\Component\Form\Extension\Core\Type\RepeatedType;
12 use Symfony\Component\Form\Extension\Core\Type>PasswordType;
13 use Symfony\Component\Form\Extension\Core\Type\SubmitType;
14
15 class UserType extends AbstractType
16 {
17     public function buildForm(FormBuilderInterface $builder, array $options)
18     {
19         $builder
20             ->add('username', TextType::class)
21             ->add('email', EmailType::class)
22             ->add('plainPassword', RepeatedType::class, array(
23                 'type' => PasswordType::class,
24                 'first_options' => array('label' => 'Password'),
25                 'second_options' => array('label' => 'Confirm password'),
26             ))
27             ->add('birthdate', BirthdayType::class, [
28                 'widget' => 'single_text',
29             ])
30             ->add('gender', ChoiceType::class, array(
31                 'choices' => array(
32                     'Male' => 'Male',
33                     'Female' => 'Female',
34                 ),
35                 'expanded' => true
36             ))
37
38             ->add('submit', SubmitType::class, ['label' => 'Sign Up', 'attr' => ['class' => 'btn-primary btn-block']]);
39     }
40 }
```

- Définition de la fonction « register » (s'inscrire en français) dans le contrôleur :

Je passe un formulaire qui, s'il est soumis et bien valide, il va encoder le mot de passe à travers un algorithme de type **bcrypt**.

```
SecurityController.php
src ▶ Controller ▶ SecurityController.php ▶ PHP Intelephense ▶ SecurityController

27  /**
28   * @Route("/register")
29   */
30  public function register(Request $request, UserPasswordEncoderInterface $passwordEncoder)
31  {
32      // 1) build the form
33      $user = new User();
34      $form = $this->createForm(UserType::class, $user);
35      // 2) handle the submit (will only happen on POST)
36      $form->handleRequest($request);
37      if ($form->isSubmitted() && $form->isValid()) {
38          // 3) Encode the password
39          $password = $passwordEncoder->encodePassword($user, $user->getPlainPassword());
40          $user->setPassword($password);
41          $user->setIsActive(true);
42          $user->setRegistrationDate(new \DateTime());
43          $user->setSrcPhoto('img_user/user.png');
44          $user->setAltPhoto('user.png');
45          $user->setTitlePhoto('Default image');
46          // 4) save the User!
47          $entityManager = $this->getDoctrine()->getManager();
48          $entityManager->persist($user);
49          $entityManager->flush();
50          // ... do any other work - like sending them an email, etc
51          $this->addFlash('success', 'Your account has been saved.');
```

C'est une fonction de hachage créée par Niels Provos et David Mazières. En plus de l'utilisation d'un sel pour se protéger des attaques par table arc-en-ciel (rainbow table), bcrypt est une fonction adaptive, c'est-à-dire que nous pouvons augmenter le nombre d'itérations pour la rendre plus lente.

- Rendu Twig grâce à sa propre méthode de formulaire et essai réalisé :

```
register.html.twig
templates ▶ security ▶ register.html.twig ▶ ...

1  {% extends 'base.html.twig' %}
2  {% block title %}
3  {{title}}
4  {% endblock %}
5
6  {% block body %}
7
8      {% if title is defined %}
9      <h1 class="page__title">{{title}}</h1>
10     {% else %}
11     <h1>Header</h1>
12     {% endif %}
13
14     {{form(form)}}
15
16 {% endblock %}
```

Les tests sont concluants puisque des messages d'erreurs indiquent à l'utilisateur que les champs renseignés ne sont pas valides et le compte ne peut donc pas être créé.

The screenshot shows a 'Register' form with the following fields and values:

- Username:** Test
- Email:** nemanja@orange.fr (Error: **ERROR** This value is already used.)
- Password:** (Error: **ERROR** This value is not valid.)
- Confirm password:** (Empty)
- Birthdate:** 01/01/2000
- Gender:** ☐ Male, ☒ Female

A 'Sign Up' button is at the bottom.

Deuxième jeu d'essai concernant la connexion d'un client si celui-ci renseigne mal son mot de passe ou son adresse mail :

```
SecurityController.php
src > Controller > SecurityController.php > PHP Intelephense > SecurityController

63
64
65  /**
66   * @Route("/login", name="login")
67   */
68   public function login(Request $request, AuthenticationUtils $authenticationUtils)
69   {
70       // get the login error if there is one
71       $error = $authenticationUtils->getLastAuthenticationError();
72       // last username entered by the user
73       $lastUsername = $authenticationUtils->getLastUsername();
74       //
75       $form = $this->get('form.factory')
76         ->createNamedBuilder(null)
77         ->add('_username', null, ['label' => 'Email'])
78         ->add('_password', \Symfony\Component\Form\Extension\Core\Type\PasswordType::class, ['label' => 'Password'])
79         ->add('ok', \Symfony\Component\Form\Extension\Core\Type\SubmitType::class, ['label' => 'Log In', 'attr' => ['class' => 'btn-primary btn-block']])
80         ->getForm();
81
82       return $this->render('security/login.html.twig', [
83         'title' => 'Login',
84         'form' => $form->createView(),
85         'last_username' => $lastUsername,
86         'error' => $error,
87     ]);
88 }
```

Affichage dans Twig en incluant une condition si une erreur survient un message d'alerte apparait alors et l'utilisateur a de nouveau l'opportunité de renseigner ses identifiants pour tenter de se connecter.

```
login.html.twig
templates ▸ security ▸ login.html.twig ▸ ...
1  {% extends 'base.html.twig' %}
2  {% block title %}
3  |  {{title}}
4  {% endblock %}
5
6  {% block body %}
7  |  {% if title is defined %}
8  |  <h1 class="page__title">{{title}}</h1>
9  |  {% else %}
10 |  <h1>Header</h1>
11 |  {% endif %}
12 |  {% if error %}
13 |  |  <div class="alert alert-danger">
14 |  |  |  <ul class="list-unstyled mb-0">
15 |  |  |  |  <li>
16 |  |  |  |  |  <span class="initialism form-error-icon badge badge-danger">Invalid credentials</span>
17 |  |  |  |  |  <span class="form-error-message">{{ error.messageKey|trans(error.messageData, 'security') }}</span>
18 |  |  |  |  </li>
19 |  |  |  </ul>
20 |  |  </div>
21 |  {% endif %}
22 |  {{ form(form) }}
23 |  <div id="login_additional">
24 |  |  <div>
25 |  |  |  <input type="checkbox" id="remember_me" name="_remember_me"/>
26 |  |  |  <label for="remember_me">Remember me</label>
27 |  |  </div>
28 |  |  <a href="{{ path('forgotten_password') }}" class="links">Forgot your password?</a>
29 |  </div>
30 |  {% endblock %}
```

Login

ERROR Invalid credentials.

Email

nemanja@orange.fr

Password

Log In

☐ Remember me

[Forgot your password?](#)

Recherches et veille technologique

Veille

La veille effectuée a porté sur le composant sécurité de Symfony afin d'assurer l'accès à l'administration uniquement aux utilisateurs dont le rôle est Admin, l'accès au tableau de bord et aux pages d'information client aux utilisateurs dont le rôle est au minimum utilisateur : un visiteur ne peut pas avoir accès à des pages comme : « Settings » ou « Change password » par exemple. Si un visiteur tape l'URL en dur, il est redirigé vers la page de connexion.

A défaut le visiteur a accès à la route permettant de s'inscrire, ainsi que les autres pages du site, celles d'informations (les pages des livres, le panier).

Recherches sur un site anglophone

Les relations « ManyToMany » avec champs supplémentaires.

Source : SymfonyCasts.com

Texte en anglais :

Head back to /genus and click into one of our genres. Thanks to our hard work, we can link genres and users. So I know that Eda Farrell is a User that studies this Genus.

But, hmm, what if I need to store a little extra data on that relationship, like the number of years that each User has studied the Genus. Maybe Eda has studied this Genus for 10 years, but Marietta Schulist has studied it for only 5 years.

In the database, this means that we need our join table to have three fields now: genus_id, user_id, but also years_studied. How can we add that extra field to the join table?

The answer is simple, you can't! It's not possible. Whaaaaat?

You see, ManyToMany relationships only work when you have no extra fields on the relationship. But don't worry! That's by design! As soon as your join table need to have even one extra field on it, you need to build an entity class for it.

Creating the GenusScientist Join Entity

Create a new class: GenusScientist. Next, add some properties: id - we could technically avoid this, but I like to give every entity an id - genus, user, and yearsStudied:

`<code/>`

Oh, and notice! This generated a table name of genus_scientist: that's perfect! I want that to match our existing join table: we're going to migrate it to this new structure.

`<code/>`

Perfect! So how should we map the genus and user properties? Well, think about it: each is now a classic ManyToOne relationship. Every genus_scientist row should have a genus_id column and a user_id

column. So, above genus, say `ManyToOne` with `targetEntity` set to `Genus` Below that, add the optional `@JoinColumn` with `nullable=false`:

<code/>

Copy that and put the same thing above user, changing the `targetEntity` to `User`:

<code/>

Entity, done!

Updating the Existing Relationships

Now that the join table has an entity, we need to update the relationships in `Genus` and `User` to point to it. In `Genus`, find the `genusScientists` property. Guess what? This is not a `ManyToOne` to `User` anymore: it's now a `OneToMany` to `GenusScientist`. Yep, it's now the inverse side of the `ManyToOne` relationship we just added. That means we need to change `inversedBy` to `mappedBy` set to `genus`. And of course, `targetEntity` is `GenusScientist`:

<code/>

You can still keep the `fetch="EXTRA_LAZY"`: that works for any relationship that holds an array of items. But, we do need to remove the `JoinTable` annotation: both `JoinTable` and `JoinColumn` can only live on the owning side of a relationship.

There are more methods in this class - like `addGenusScientist()` that are now totally broken. But we'll fix them later. In `GenusScientist`, add `inversedBy` set to the `genusScientists` property on `Genus`:

<code/>

Finally, open `User`: we need to make the exact same changes here.

For `studiedGenuses`, the `targetEntity` is now `GenusScientist`, the relationship is `OneToMany`, and it's `mappedBy` the `user` property inside of `GenusScientist`:

<code/>

The `OrderBy` doesn't work anymore. Well, technically it does, but we can only order by a field on `GenusScientist`, not on `User`. Remove that for now.

Traduction :

Revenez à /genus et cliquez sur l'un de nos genres. Grâce à notre travail assidu, nous pouvons relier les genres et les utilisateurs. Je sais donc que Eda Farrell est un utilisateur qui étudie ce genre.

Mais, hmm, si je dois stocker un peu plus de données sur cette relation, comme le nombre d'années que chaque utilisateur a étudié le genre. Eda a peut-être étudié ce genre pendant 10 ans, mais Marietta Schulist ne l'a étudié que pendant 5 ans.

Cela veut dire que dans la base de données nous devons joindre notre table (de jointure) pour avoir trois champs : *genus_id*, *user_id*, mais aussi *years_studied*. Comment pouvons-nous ajouter ce champ supplémentaire à la table de jointure ?

La réponse est simple, vous ne pouvez pas ! Ce n'est pas possible. Quoi ?

Vous voyez, les relations « *ManyToMany* » ne fonctionnent que lorsque vous n'avez pas de champs supplémentaires dans la relation. Mais ne vous inquiétez pas ! C'est par conception ! Dès que votre table de jointure doit avoir même un champ supplémentaire, vous devez créer une classe d'entité pour celle-ci.

-Création de l'entité de jointure *GenusScientist*.

Créez une nouvelle classe : *GenusScientist*. Ensuite, ajoutez quelques propriétés : *id* - techniquement, nous pourrions éviter cela, mais j'aime bien donner à chaque entité un identifiant - genre, utilisateur et *years_studied* :

<code/>

Oh, et remarquez ! Cela a généré un nom de table de *genus_scientist* : c'est parfait ! Je veux que cela corresponde à notre table de jointure existante : nous allons la migrer vers cette nouvelle structure.

<code/>

Parfait ! Alors, comment devrions-nous mapper les propriétés de genre et d'utilisateur ? Eh bien, réfléchissez-y : chacun de ces propriétés est désormais une relation classique de *ManyToOne*. Chaque ligne *genus_scientist* doit avoir une colonne *genus_id* et une colonne *user_id*. Ainsi, au-dessus du genre, dites *ManyToOne* avec *targetEntity* défini sur *Genre*. Ajoutez ensuite l'option *@JoinColumn* facultative avec *nullable = false*

<code/>

Copiez-le et mettez la même chose au-dessus de l'utilisateur en remplaçant *targetEntity* par *User* :

<code/>

Entité faite !

-Mise à jour des relations existantes

Maintenant que la table de jointure a une entité, nous devons mettre à jour les relations entre *Genus* et *User* pour la souligner. Dans *Genus*, recherchez la propriété *genusScientists*. Devinez quoi ? Ce n'est plus un *ManyToMany* à *User* : c'est un *OneToMany* à *GenusScientist*. Oui, c'est maintenant l'inverse de la relation *ManyToOne* que nous venons d'ajouter. Cela signifie que nous devons changer *inversedBy* en *mappedBy* set to *genus*. Et bien sûr, *targetEntity* appartient à *GenusScientist* :

<code/>

Vous pouvez toujours garder le *fetch* = « *EXTRA_LAZY* » : cela fonctionne pour toute relation qui contient un tableau d'éléments. Cependant, nous devons supprimer *JoinTable* : annotation : *JoinTable* et *JoinColumn* ne peuvent vivre que du côté propriétaire d'une relation.

Il y a plus de méthodes dans cette classe - comme *addGenusScientist* () qui sont maintenant totalement cassées. Mais nous les réparerons plus tard. Dans *GenusScientist*, ajoutez le paramètre *inversedBy* à la propriété *genusScientists* sur le genre :

<code/>

Enfin, ouvrez *Utilisateur* : nous devons apporter exactement les mêmes modifications ici.

Pour *studiedGenuses*, *targetEntity* est désormais le *GenusScientist*, la relation est *OneToMany* et elle est mappée par la propriété utilisateur à l'intérieur du genre *GenusScientist* :

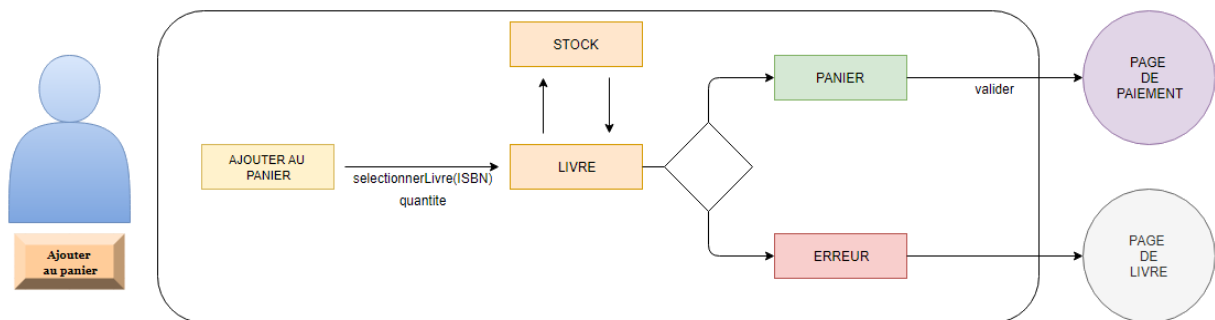
<code/>

OrderBy ne fonctionne plus. Techniquement, oui, mais nous ne pouvons commander que par un champ sur *GeniusScientist*, pas sur *User*. Enlevez cela pour l'instant.

Difficultés

La gestion du panier

Une des nombreuses difficultés que j'ai eues lorsque je développais cette application était la création du panier. C'est là que j'ai effectué le plus de recherches sur Internet. Pendant ma formation, lorsque nous faisons les exercices avec la « SESSION », nous utilisons la variable super globale \$_SESSION. Le problème était que je ne savais pas que dans Symfony, \$_SESSION était un objet. Avec l'aide de la documentation officielle de Symfony, j'ai réussi à la fin de créer le panier dont j'avais beaucoup besoin. Pour l'instant il est pas fini parce qu'il me manque plusieurs fonctionnalités notamment la quantité qui permettra aux utilisateurs de choisir le nombre des livres qu'ils veulent acheter. Dans le schéma suivant je vous montre comment j'ai imaginé qu'il fonctionnait.



Lorsque l'utilisateur clique sur le bouton « Ajouter au panier » il sélectionne un livre avec son ISBN (L'International Standard Book Number). Après vérification du livre en stock il y a deux possibilités :

- Si le livre est en stock, il est envoyé au panier où l'utilisateur décide combien livres il veut acheter. Ensuite il doit valider l'achat et il arrive sur la page de paiement.
- Si le livre n'est pas en stock, l'utilisateur reste sur la page de livre qui lui affiche l'erreur : « Rupture de stock ».

Etant donné que la Session est un objet et que ce panier est un tableau dans cet objet, j'ai utilisé les accesseurs(getters) à récupérer les valeurs de mes variables et les mutateurs(setters) pour les modifier.

Axes d'améliorations de l'application

En développant cette application j'ai appris que le développement d'un site web n'a pas de fin. Nous pouvons toujours ajouter des fonctionnalités qui pourraient s'intégrer facilement. Il y a plusieurs points que je n'ai pas eu le temps d'intégrer et voici quelques exemples :

Jeton de confirmation de l'inscription.

Lorsque l'utilisateur veut s'inscrire, il est redirigé sur la page d'enregistrement. Après avoir rempli le formulaire avec ses données personnelles (nom d'utilisateur, email, mot de passe, date de naissance, sexe) et l'avoir validé, son compte est créé. Cela signifie que l'utilisateur peut saisir une adresse fausse (azerty@qsdgfv.bv, qsdwxc@134679.yu, ...) et avoir un compte non sécurisé. Pour éviter cela, j'ai pensé à ajouter la possibilité de recevoir un mail qui contiendrait jeton de confirmation. Ce jeton servira à générer un lien qui lui redirigera vers la page de confirmation.

Ajout de gestion de stock

Actuellement, l'achat est possible qu'en version numérique. Ce serait bien que les utilisateurs achètent et commandent des livres en version papier aussi. Dans ma base de données j'ai une table « book » et une autre table « library », mais j'aurai surtout besoin d'une table « book_library » qui fera la liaison entre les deux. Cette table de liaison ne contiendra que trois colonnes : « book_id », « library_id » et « UGS (unité de gestion des stocks) ».

Après il faut adapter le panier, de sorte que les utilisateurs puissent sélectionner la quantité.

Désinscription d'utilisateurs

Tous les sites internet ont l'obligation de laisser à l'utilisateur la possibilité de supprimer son compte. Parfois, cette option n'est pas facilement accessible parce que les sites qui vendent des produits doivent conserver des données (nom, prénom, facture, ...) pour les raisons comptables. Il n'y pas malheureusement une démarche à le faire en ligne. Il faut envoyer un mail à l'équipe commerciale contenant la demande de résiliation.

Barre de recherche

Sur les sites qui contiennent beaucoup de contenus, surtout les sites d'e-commerce, la barre de recherche est souvent l'élément le plus fréquemment utilisé. Lorsque les utilisateurs rencontrent des sites relativement complexes, ils chercheront immédiatement une barre de recherche pour arriver à leur destination finale rapidement et facilement. La recherche est une activité fondamentale et un élément crucial dans la conception d'une application. Même un changement mineur comme une taille correcte pour le champ de saisie ou d'indiquer quels types d'information sont à entrer dans le champ peut augmenter significativement l'utilisabilité de la recherche, avec l'ensemble d'UX.

Possibilité de répondre aux commentaires des utilisateurs

Comme admin du site peut déjà supprimer les commentaires inappropriés, c'est normal qu'il pourra y répondre au cas où ils seraient adéquats aux règles de ce site. Peu importe si ce sont les avis positif ou négatif. Parce que en répondant aux avis, nous stimulons l'attention des utilisateurs et nous les valorisons, ce qui contribue à renforcer la fidélisation.

Conclusion

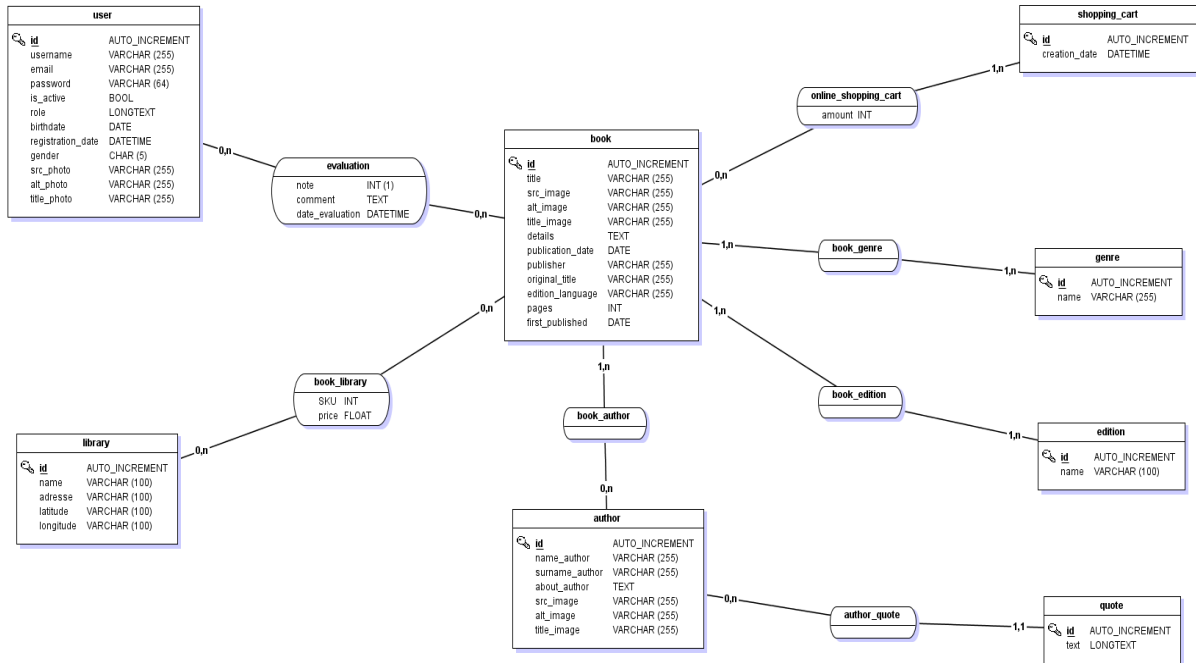
Au cours de la réalisation du projet, certaines difficultés ont été rencontrées. Notamment la création du panier et l'utilisation de la session avec Symfony. Le choix des technologies a également présenté certaines difficultés. En effet, avec du recul, je me rends compte qu'un travail de recherche plus approfondi est nécessaire afin de faire les bons choix.

Malgré les difficultés rencontrées, qui, je pense, font partie intégrante d'un quotidien de développeur, cela ne fait pas de doute pour moi que mon avenir est dans cette voie. Je suis très satisfait de cette formation et de la réussite du projet. J'ai découvert un milieu dans lequel je me plais, où j'ai envie de continuer à apprendre et découvrir. Mon objectif est de persévérer et je reste curieux de voir où je pourrai bien arriver un jour. Au final, je me suis vraiment éclaté à faire ce développement !

Annexes

Présentation du MCD et MLD

Schéma de représentation des entités à travers un modèle conceptuel de données (entités, attributs et relation) défini par JMerise.



Présentation des objets du MCD sous une forme compréhensible par un système de gestion de base de données, les objets formant des tables ainsi que les unissent.

C'est le modèle logique de données qui est l'étape précédent la création de la base de données via Doctrine sous Symfony.

