Université Toulouse Jean Jaurès

UFR d'Histoire, Arts et Archéologie

Département Documentation, Archives, Médiathèque et Édition

D'une programmation maison en PHP orienté objet à Symfony.

Retour d'expérience de la prise en main d'un framework.

Elodie Vianai

Mémoire de stage présenté pour l'obtention du Master 2 Ingénierie de l'Information Numérique

Sous la direction de M. Yaël CHAMPCLAUX

Septembre 2017

Sommaire

Résumé 3

Mots-clefs 4

Remerciements 5

Glossaire 6

Notes 9

Introduction 10

Première partie : De PHP orienté objet à Symfony 13

Chapitre 1. PHP orienté objet : les bonnes pratiques 14

La programmation orientée objet 14

Les recommandations PSR 17

La méthode ORM 18

Chapitre 2. L'architecture web 19

L'architecture MVC 20

Les moteurs de templates 21

L'architecture de Symfony 26

Deuxième partie : Prise en main de Symfony 33

Chapitre 3. Analyse des choix 34

La gestion des utilisateurs 34

L'espace d'administration 39

Chapitre 4. Perspectives d'améliorations 41

Vers un *back-office* plus performant 42

Vers une application *responsive* et personnalisée 44

Troisième partie : Synthèse 46

Conclusion 50

Sitographie 51

Table des matières 54

Annexe 1 : Liste des technologies du projet Portfolio 56

Annexe 2 : Capture d'écran du fichier *security.yml* 57

Résumé

Dans le cadre de mon stage de fin d'études, j'ai intégré Web-atrio, une entreprise de services numériques. L'objectif de ces six mois était de monter en compétence dans le domaine du développement, et plus particulièrement sur un framework.

Après une remise à niveau en programmation orientée objet, et plus précisément en PHP, j'ai découvert et appris à prendre en main Symfony 3. Leader sur le marché des frameworks PHP, le produit de l'entreprise Sensio Labs permet de découvrir et d'aborder un certain nombre d'éléments généraux reconnus dans l'univers de PHP : les recommandations PSR, la méthode ORM mais aussi l'architecture MVC ou encore les moteurs de templates. Tous ces éléments se réunissent sous le nom de Symfony de façon à ne former qu'un framework puissant, fiable et reconnu.

A travers ma prise en main de Symfony sur le projet de mon portfolio, les atouts comme les inconvénients de ce framework sont présentés.

Mots-clefs

Français	Anglais
Infrastructure de développement	Framework
Développement web	Web development
Symfony	
Comparaison	Comparison
Portfolio	
Avantages	Advantages
Inconvénients	disadvantages

Remerciements

Avant toute chose je tenais à remercier Kevin PHILIPPE, mon tuteur de stage, ainsi qu'Eva ROUX, ma marraine au sein de l'entreprise Web-atrio. Je les remercie pour leur gentillesse, leur professionnalisme, leur suivi ainsi que leurs apports tout au long de ces six mois de stage. Je voulais aussi remercier Jennifer DA SILVA pour la relecture et la correction de ce mémoire. Mais l'équipe de Web-atrio est l'équivalent d'une grande famille où tout le monde y ajoute son empreinte, c'est donc à l'ensemble de l'équipe que j'adresse mes remerciements. La disponibilité, l'entraide, le dynamisme, les échanges et l'accueil de tous m'ont permis de trouver ma place au sein de cette entreprise, de gagner en compétences et de vivre une expérience riche tant sur le plan professionnel qu'humain. Web-atrio mérite grandement sa place dans les palmarès de « Great Place To Work »[1] et « Happy At Work 2017 »[2].

Je tenais aussi à remercier Yaël CHAMPCLAUX pour sa disponibilité et son évaluation en qualité de tuteur. Je souhaiterai également remercier Taoufiq DKAKI pour ses enseignements tout au long de l'année et son évaluation pour ce travail.

Pour cette année universitaire je remercie les membres de l'équipe pédagogique, et plus particulièrement Valérie Ziegler pour son soutien et sa motivation, ainsi que mes camarades.

Et pour terminer, je ne peux oublier de citer mes parents et même plus largement ma famille, mes amies Margaux BOISSONNADE et Eloddie MEDAR ainsi que tous mes proches qui n'ont cessé de m'inspirer, me soutenir, me porter et m'encourager durant mes études.

[1] Web-atrio termine 4e au Palmarès «Great Place To Work » de 2016 pour les entreprises françaises de moins de 50 salariés. Pour voir l'ensemble du palmarès : http://www.greatplacetowork.fr/meilleures-entreprises/best-workplaces-en-france-moins-de-50-salaries

[2] Web-atrio a obtenu le label « Happy at work 2017 » : http://meilleures-entreprises.com/recrutement/web-atrio/

Glossaire

API ou ***Application Programming Interface*** (« interface de programmation applicative » en français) : solution informatique permettant à des applications de communiquer entre elles et de s'échanger mutuellement des services. Dans le développement web, elle permet au développeur de pouvoir utiliser un programme sans avoir à se soucier du fonctionnement complexe d'une application.

Back-end (arrière-plan) : correspond à la couche métier, c'est-à-dire tout ce qui se passe côté serveur. En opposition avec *front-end*.

Back-office (arrière-guichet[3]) : interface qui n'est visible et accessible qu'aux administrateurs d'un site internet. En opposition avec *front-office*.

Bundle (paquet ou lot en français) : ensemble de fichiers et de répertoires implémentant une ou des fonctionnalités. En règle générale, un bundle correspond à une fonctionnalité générale de l'application (ex : la gestion des utilisateurs).

CMS (*Content Management System* en anglais) ou système de gestion de contenu en français : logiciels de création et de mise à jour dynamique de sites web.

CSS (acronyme de *Cascading Style Sheet)* ou « feuille de style en cascade » en français. Ce sont des fichiers mettant en forme les pages HTML. Pour cela chaque élément d'une page, un comportement lui est défini.

Design pattern (« modèle ou patron de conception » en français) : modèle de référence qui sert de source d'inspiration lors de la conception de l'architecture d'un système ou d'un logiciel informatique en sous-éléments plus simples, tels que des modules. Cette solution répond à un problème récurrent d'organisations de classes objet par le biais d'une conception empirique. Cette forme d'organisation rend ensuite possible la transposabilité des éléments dans une autre application.

[3] Source : « *Vocabulaire de l'informatique et de l'internet (liste de termes, expressions et définitions adoptés)*. JORF n°0214 du 16 septembre 2014 page 15186 texte n° 78 », LégiFrance. https://www.legifrance.gouv.fr/affichTexte.do;jsessionid=?cidTexte=JORFTEXT000029461191&dateTexte=&oldAction=dernierJO&categorieLien=id (consulté le 20 août 2017).

DQL (*Doctrine Query Language*) : langage de requête orienté objet propre à l'ORM Doctrine

Framework (« structure logicielle » en français) : ensemble d'outils et de composants logiciels à la base d'un logiciel ou d'une application qui établit ses fondations ou son squelette.

Front-end (frontal): correspond au design et à ce qui est affiché côté client, c'est-à-dire visible par l'utilisateur. En opposition avec *back-end*.

Front-office (guichet[4]) : interface permettant d'accéder aux services en ligne proposés par une entreprise ou une organisation. En opposition avec *back-office*.

Interopérabilité : capacité de matériels, de logiciels ou de protocoles différents à fonctionner ensemble et à partager des informations.

JavaScript : langage de script orienté objet qui permet entre autre d'apporter des animations sur une page ou encore d'exécuter du code sans être obligé de recharger la page.

Micro-framework : framework minimaliste dont l'objectif est de servir de base pour une application. L'avantage face à un framework est qu'il laisse le choix d'installer les composants et est moins contraignant lors des migrations ou évolutions. Voir *Framework*.

Moteur de *templates* : système permettant de rassembler le code de présentation (HTML, CSS…) et le code d'application (PHP, SQL…).

ORM (*Object-Relational Mapping*) : technique permettant de faire le lien entre une base de données et la programmation orientée objet, en transformant une table en objet manipulable via ses attributs et inversement.

PDO (*PHP Data Object*) : interface objet qui permet d'accéder à une base de données.

[4] Source : « *Vocabulaire de l'informatique et de l'internet (liste de termes, expressions et définitions adoptés)*. JORF n°0214 du 16 septembre 2014 page 15186 texte n° 78 », LégiFrance. https://www.legifrance.gouv.fr/affichTexte.do;jsessionid=?cidTexte=JORFTEXT000029461191&dateTexte=&oldAction=dernierJO&categorieLien=id (consulté le 20 août 2017).

PHP (*PHP Hypertext Preprocessor)* : langage de scripts généraliste et Open Source, spécialement conçu pour le développement d'applications web.

Plugin ou ***plug-in*** (« brancher » en français) : module d'extension apportant de nouvelles fonctionnalités à un programme ou logiciel.

PSR ou ***PHP Standards Recommendations*** : recommandations validées par le *PHP Framework Interoperability Goup* (PHP fig), dont l'objectif est l'amélioration de l'interopérabilité entre les développeurs PHP.

SQL ou ***Structured Query Language*** (« langage de requête structurée » en français) : langage permettant aux applications de communiquer avec une base de données.

Template (« thème » en français) : enveloppe graphique d'un site internet indépendante du contenu (aussi appelé *layout*).

Notes

Ce mémoire de stage, écrit dans le cadre d'une deuxième année de master Ingénierie de l'Information Numérique, puise ses constats et ses exemples dans mon expérience en entreprise en tant que stagiaire. J'ai choisi de m'adresser à un lecteur novice en informatique et plus précisément en développement web afin que cet écrit soit accessible à tous.

Pour le confort de la lecture, les citations sont écrites entre guillemets. De même que les termes techniques ou anglophones, écrits en italiques, sont explicités lors de leur première mention ainsi que définis dans le glossaire.

Le projet du Portfolio, mentionné en exemple tout au long de ce mémoire, est disponible en téléchargement libre à l'adresse suivante : https://github.com/elodie-vianai/portfolio (code).

Introduction

Web-atrio est une entreprise de type SARL spécialisée dans la réalisation de solutions logicielles et applications web. Créée en 2012, elle compte aujourd'hui 80 employés répartis sur deux agences situées à Paris et Toulouse. En seulement cinq ans d'existence, Web-atrio recense de nombreux clients de renom tel qu'Airbus, EDF, l'Autorité de Sûreté Nucléaire (ASN), PréviFrance, Eres, Darty, Body'minute ou encore Universal Music Group.

Web-atrio se démarque cependant de ses concurrents puisqu'elle est ce que l'on appelle « une entreprise libérée ». Cette nouvelle société est née de la volonté de créer une entreprise où le côté humain prime sur le côté commercial. Forte des expériences de chacun de ses fondateurs, Steve FERRERO, Rémi GAUBERT et Wojtek JAMKA, Web-atrio a été créée pour permettre à chacun de travailler dans une société où le contact humain et la bonne entente entre les employés sont privilégiés par rapport aux intérêts commerciaux. Ce système permet aussi à chaque employé de travailler en priorité dans son domaine de compétences pour mettre en avant la qualité du travail par rapport à la quantité. Cela conduit à la satisfaction de chacun, du client jusqu'à chaque employé.

Lors de mon arrivée à Web-atrio, mes compétences étaient en deçà des exigences professionnelles. Afin de me former à quelques technologies supplémentaires, il m'a été proposé de réaliser mon propre portfolio parmi une liste de technologies[5]. Ce projet de portfolio, qui deviendra par la suite un projet de formation généralisé, représente un support pour les commerciaux. En effet, lorsque ceux-ci souhaitent placer un développeur chez un client, ils pourront s'appuyer sur cette réalisation pour non seulement réaliser son CV mais aussi pour montrer les compétences de ce dernier : le code source est stocké sur une plateforme publique (Github[6]).

Le portfolio, qui possède un document générique de spécifications, doit posséder une partie publique présentant la formation, les expériences et les projets réalisés par le développeur. Ce dernier devra pouvoir ajouter, modifier et supprimer des éléments, ce qui implique une partie d'administration. De même qu'une gestion des authentifications est essentielle afin de pouvoir gérer les authentifications

[5] Vous trouverez la liste des technologies utilisées en Annexe 1.
[6] Lien vers mon Github : https://github.com/elodie-vianai/portfolio

tant pour l'administrateur que pour les visiteurs souhaitant accéder à l'interface de programmation applicative (API)7 de Spotify. Le service de streaming musical Spotify devait être accessible uniquement pour les personnes authentifiées de façon à présenter la ou les playlist(s) de travail du développeur. Bien entendu, cette application se devra d'être évolutive. En ce qui concerne l'ergonomie, le design et le référencement, aucune exigence n'a été définie.

Si les principales technologies étaient au choix, seuls le *framework*[8] Bootstrap pour la partie *front-end*[9] et le langage SQL (*Structured Query Language*)[10] pour la communication avec la base de données ont été imposés. En ce qui me concerne, j'ai choisi la technologie PHP pour le *back-end*[11] (côté serveur). La version PHP utilisée est la 7.0, et donc en orienté objet afin de mieux s'accorder aux demandes actuelles. De plus, ce projet est également un moyen d'acquérir les bases d'un framework puisqu'elle doit obligatoirement en inclure un. Pour ma part, j'ai opté pour le plus connu et le plus utilisé actuellement : Symfony. Lors de mon arrivée à Web-atrio, la version 3.3 était la plus récente donc c'est celle-ci qui a été installée.

Avant de poursuivre, il convient de définir précisément ce qu'est un framework. Un framework est un outil, souvent basé sur un langage, qui respecte des normes, des bonnes pratiques et une architecture établies par des experts. Aujourd'hui le développement dit « maison », c'est-à-dire que le développeur code entièrement son application seul, n'est plus réellement utilisé au profit de l'utilisation de frameworks, qui se multiplient : Laravel, Symfony ou Zend sont les plus connus pour le PHP, Django ou encore Turbogear pour le Python, Rails pour du Ruby, Angular ou React.js sont les plus connus pour le Javascript, Bootstrap pour le front… En tant que jeune développeur, il est utile de se demander pourquoi un tel engouement. Qu'est-ce qu'apporte l'utilisation d'un framework par rapport au développement dit « maison » ?

Symfony est un framework français PHP lancé en 2005 par l'agence web Sensio Labs, et plus précisément par Fabien Potencier. Cet outil serait une évolution d'un framework interne à l'agence qui a ensuite été

[7] Interface Application Programming (API), signifie « interface de programmation applicative » en français.
[8] Un framework, que l'on peut traduire par « structure logicielle », est un ensemble d'outils à la base d'une application qui établit ses fondations ou son squelette.
[9] Front-end : correspond au design et à ce qui est affiché côté client et qui est donc visible par l'utilisateur.
[10] SQL (*Structured Query Language*), signifie « langage de requête structurée » en français. Il s'agit d'un langage permettant aux applications de communiquer avec une base de données.
[11] Back-end : correspond à la couche métier, c'est-à-dire tout ce qui se passe côté serveur.

mis en Open Source, c'est-à-dire que son utilisation a été rendue accessible à tous gratuitement. Symfony appartient à la catégorie des frameworks, dont les objectifs premiers sont l'amélioration de la rapidité et de la qualité du développement.

La plus importante acquisition durant mon stage a été l'apprentissage du framework Symfony. Par conséquent cette comparaison se basera uniquement sur l'outil utilisé. Nous verrons ensuite l'évolution depuis le développement « maison » jusqu'à l'utilisation du framework, de façon à mettre en lumière les principaux points communs. Puis, à travers mon expérience sur le projet du portfolio, je présenterai certains avantages apportés par Symfony. Enfin, je terminerai par une synthèse de façon à reprendre l'ensemble des éléments précédents et à en dégager des avantages et des inconvénients généraux.

Première partie : De PHP orienté objet à Symfony

En tant que projet PHP, Symfony reprend une grande partie des éléments du langage mais aussi de ses méthodes de travail. Après l'étude et l'utilisation du framework, il m'a été possible de dégager quelques points communs entre les deux méthodes de développement.

Chapitre 1. PHP orienté objet : les bonnes pratiques

Symfony est un framework qui utilise la version 5 et ultérieure de PHP, c'est-à-dire la version orientée objet. Utilisant PHP orienté objet comme langage de référence, le framework répond par conséquent aux mêmes standards. Le développement s'est également doté de "bonnes pratiques", des méthodes d'organisation des projets et/ou d'éléments permettant d'optimiser son code, qu'il est conseillé d'adopter pour des raisons de compréhensibilité et de performances.

0La programmation orientée objet

Contrairement à la programmation procédurale, qui sépare le traitement des données des données elles-mêmes, c'est-à-dire une suite de données et de leurs traitements, l'orienté objet transforme un site internet en une multitude d'objets qui interagissent entre eux. Steve Jobs, dans une interview[12], explique la programmation orientée objet de la façon suivante : « *les objets sont comme les gens. Ce sont des choses qui vivent, respirent, ont des connaissances intrinsèques sur comment faire les choses et ont une mémoire à l'intérieur d'eux qui leur permet de se souvenir des choses. Et plutôt que d'interagir avec eux à un très bas niveau, on interagit avec eux à un très haut niveau d'abstraction comme nous le faisons présentement* ». Et de façon plus concrète, il prend l'exemple d'une blanchisserie : « *Voici un exemple : si je suis votre objet de blanchisserie, vous pouvez me donner vos vêtements sales et m'envoyer un message qui dit "Pouvez-vous faire laver mes vêtements, s'il vous plaît ?". Je sais où se trouve la meilleure blanchisserie. Je parle anglais et j'ai des dollars dans mes poches. Donc, je sors et j'appelle un taxi et dis au chauffeur de me conduire à cet endroit à San Francisco. Je vais faire laver vos vêtements, je saute à nouveau dans un taxi et je reviens ici. Je vous donne vos vêtements propres, "Voici vos vêtements propres"* [...] *Vous n'avez aucune idée de comment je l'ai fait. Vous n'avez aucune connaissance de*

[12] Olivier FAMIEN, « Comment pourriez-vous expliquer l'orienté objet ? », 27/10/2015, Développez.com. https://www.developpez.com/actu/91705/Comment-pourriez-vous-expliquer-l-oriente-objet-Steve-Jobs-a-essaye-d-expliquer-ce-concept/

l'endroit où se trouve la blanchisserie. Peut-être que vous parlez français, et vous n'êtes même pas en mesure d'appeler un taxi. Pourtant, je savais comment faire tout cela. Et vous n'avez pas besoin de savoir tout ça. Toute cette complexité était cachée à l'intérieur de moi et nous avons pu interagir à un niveau d'abstraction très élevé. C'est ce que sont les objets. ».

Pour ce qui est du portfolio, celui-ci est composé de six objets :

- *User* (utilisateur) qui correspond aux utilisateurs externes qui s'enregistreront sur l'application,
- *Department* (département) qui contient l'ensemble des départements français,
- *Experience* (expérience) pour toutes mes expériences professionnelles,
- *Project* (projet) pour tous les projets réalisés lors de chaque expérience professionnelle,
- *Training* (formation) qui correspond à toutes mes formations, scolaires comme professionnelles,
- *Skill* (compétence) dans laquelle sont enregistrées toutes mes compétences techniques (langages informatiques, langues parlées…) et qui sont en lien avec les projets que j'ai réalisé

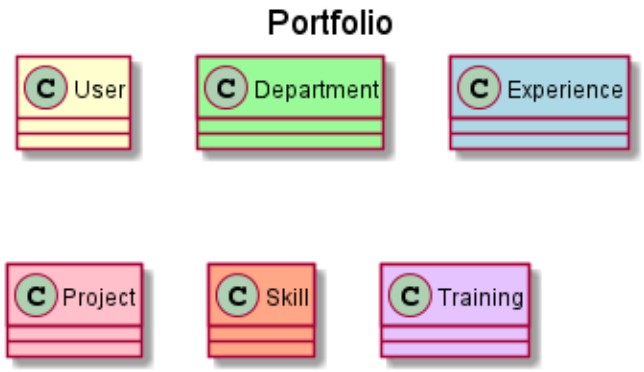


Image 1 : Les 6 objets du projet Portfolio

Chaque objet contient ensuite une classe, qui permet de décrire ses caractéristiques telles qu'un identifiant unique, un nom, etc. Le schéma UML suivant présente ainsi les éléments de base d'une classe :

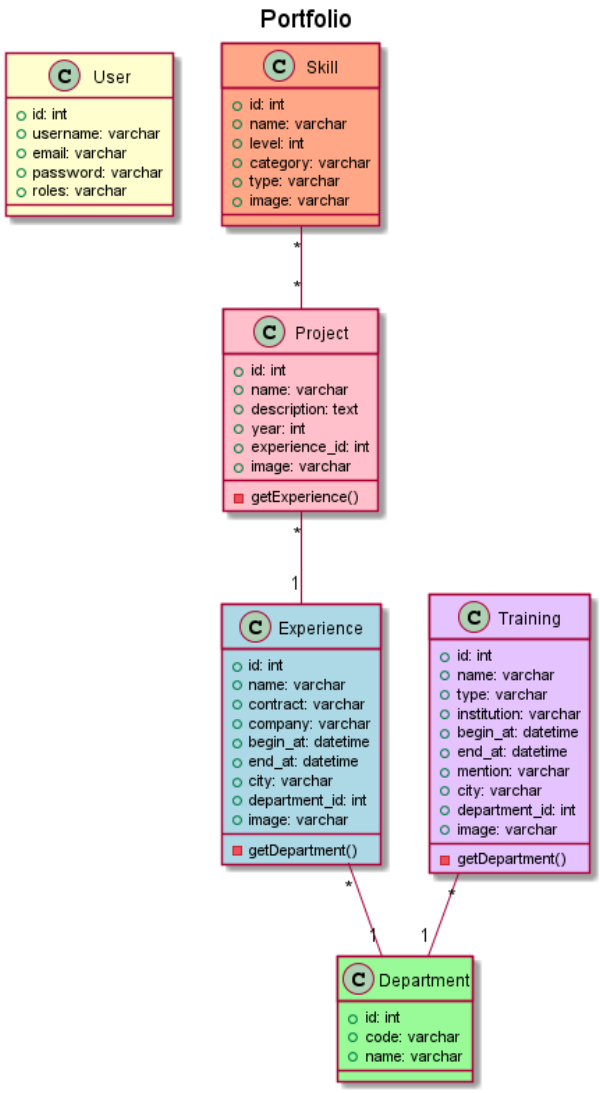


Image 2 : UML du projet Portfolio

En termes d'utilisation et de praticité, la programmation objet permet un code plus modulable mais implique une logique différente du PHP procédural, qui est basé sur une conception linéaire. En séparant les divers éléments, le développeur réalise l'interdépendance des différents éléments de son programme. Cette séparation apporte alors une meilleure lisibilité du code. La compréhensibilité d'un code informatique est essentielle pour la maintenance de l'application. Cependant, le PHP objet demande d'acquérir de nouvelles notions par rapport au procédural et la transition entre les deux types de programmation peut être complexe.

Pour ce qui est de mon expérience personnelle, le passage du PHP procédural au PHP orienté objet n'a pas été simple puisqu'il m'a fallu presque un mois et demi pour acquérir les bases. Mais une fois acquises, j'ai pu facilement me familiariser avec Slim, un micro-framework[13] PHP utilisé pour la version 2 du portfolio, puis avec Symfony.

Les recommandations PSR

Les recommandations PSR, qui signifient « *PHP Standard Recommandations* », ont été définies par le groupe de travail PHP-FIG (*PHP Framework Interoperability Group*, anciennement *PHP Standards Group*). Il s'agit de règles affectant différents niveaux du développement afin de le normaliser.

Il existe à ce jour 9 types de recommandations validées mais les 4 plus importantes sont les suivantes :

- « PSR-1 : *Basic Coding Standards* » fixe les conventions minimales de codage, comme les tags (*<?php ?>*), l'encodage, le fait qu'un fichier ne doit contenir qu'une seule classe, la manière d'écrire un *namespace*, etc.
- « PSR-2 : *Coding Guide Style* », qui étend le PSR-1, définit le style et l'organisation du code tels que l'indentation, le nombre d'espaces entre certains éléments, la limite de 120 caractères maximums par ligne, les retours à la ligne, la façon dont les méthodes doivent être déclarées ("abstract", "static"…) ainsi que leur ordre.
- « PSR-3 : *Logger Interface* » a pour objectif de permettre l'interopérabilité de l'interface des *loggers*.
- « PSR-4 : *Autoloading Standards* » reprend le PSR-0, aujourd'hui déprécié, et doit permettre l'auto-chargement des classes depuis leur chemin en les définissant.

Même si elles sont très utilisées dans le milieu du développement PHP, ces recommandations font encore débat. Malgré tout, elles sont très utilisées et sont intégrées à de nombreux logiciels d'environnement de développement intégrés (IDE), et tendent donc à devenir un standard.

[13] Un micro-framework est un framework minimaliste dont l'objectif est de servir de base pour une application.

L'avantage de l'IDE *PhpStorm*, que j'utilise, est qu'il possède un ou plusieurs *plugins*[14] prenant en compte les recommandations PSR de façon à signaler une erreur dans ma production.

En suivant ces recommandations PSR, le développeur est ainsi certain de produire un code clair, pouvant être compris par tous.

La méthode ORM

Le mapping objet-relationnel *(Object-Relational Mapping* ou *ORM* en anglais) est une méthode informatique qui permet de créer l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle. Pour se faire, des correspondances entre la base de données et les objets sont définis via ses attributs. En d'autres termes, cette technique associe une ou plusieurs classes avec une table puis chaque attribut de la classe avec un champ de la table. En PHP il existe plusieurs ORM différents tels que PdoMap, Propel ou encore Doctrine.

Doctrine est un logiciel libre sous licence GNU LGPL (Licence publique générale limitée GNU) qui est le plus connu de sa catégorie et qui est utilisé, par défaut, par Symfony. Doctrine emporte avec lui différents éléments tels que Doctrine DBAL (*DataBase Abstraction Layer* pour Couche d'abstraction de base de données) qui s'appuie lui-même sur PDO (*PHP Data Objects*) qui est l'interface objet qui permet d'accéder à une base de données. Pour communiquer, Doctrine utilise son propre langage de requête : le DQL (*Doctrine Query Language*). A l'inverse du langage SQL (*Structured Query Language*), il permet notamment de créer des requêtes d'accession et de manipulation des données. Un des avantages de Doctrine et du DQL est la sécurité accrue par rapport au langage SQL grâce à un envoi certifié de données propres. Cela s'explique notamment par la décomposition des requêtes :

```
$query = $entityManager→createQuery(SELECT name FROM
project WHERE p.year = :year');

$query→setParameter('year', '2017');

$projects = $query→getResult();
```

En SQL, on aurait écrit :

[14] *Un plugin* ou *plug-in* est un module d'extension apportant de nouvelles fonctionnalités à un programme.

```
$projects[] = 'SELECT name FROM project WHERE project.year
= 2017';
```

L'un des derniers avantages de Doctrine est qu'il n'y a aucun lien direct entre la base de données et l'application. Il est donc possible de changer de base de données entre la phase de développement et la phase de production sans avoir à modifier son code.

Installé par défaut avec Symfony, l'ORM Doctrine n'est pas indispensable au fonctionnement de l'application, mais conseillé.

La normalisation à venir du PSR devrait permettre de standardiser et de fixer les bonnes pratiques de PHP. Ainsi, le développement dans ce langage pourrait devenir relativement homogène et rendrait la compréhensibilité du code plus simple. Symfony, en se basant sur ces recommandations et en utilisant un ORM par défaut est donc un acteur de cette standardisation. Cependant le développement web ne se résume pas à la simple production de lignes de codes mais implique également une logique d'organisation qui se trouve être essentielle pour sa maintenabilité.

Chapitre 2. L'architecture web

La première règle pour produire une application compréhensible et maintenable est de soigner son architecture. Elle permet notamment d'avoir un aperçu de son système avant le développement et surtout de faciliter sa réutilisation. Il existe différents standards de *design pattern*[15] mais les trois principaux sont :

- le *MVC* (modèle-vue-contrôleur) décompose l'application en trois parties : le modèle pour les données, la vue pour l'affichage et le contrôleur pour les actions.

[15] Design pattern : modèle de conception de référence qui sert de source d'inspiration lors de la conception de l'architecture d'un système ou d'un logiciel informatique en sous-éléments plus simples, tels que des modules.

- le *MVP* (modèle-vue-présentation) est une architecture qui dérive du MVC, à l'exception de l'interaction entre le modèle et la vue qui est supprimée.
- le *MVVM* (modèle-vue et vue modèle) a été crée par Windows. Cette méthode sépare l'affichage (vue) des données. Pour communiquer, elle utilise les principes de binding (technique visant faire la liaison entre deux langages informatique) et d'événements (lorsqu'une action est effectuée, comme un mouvement de souris).

Symfony utilise le modèle MVC, qui est le plus utilisé dans le développement web, et fournit une solution performante pour l'affichage des données : un moteur de template.

L'architecture MVC

L'architecture MVC, acronyme pour Modèle - Vue - Contrôleur, est un *design pattern* séparant un système en trois modules distincts mais interdépendants. Originellement conçu pour les projets lourds, il a depuis été adopté par un grand nombre de développeurs.

Le modèle correspond à la partie « métier » de l'application, c'est-à-dire qu'il traite les données et les interactions avec la base de données. C'est lui qui définit les objets de notre application par le biais de classes. Par exemple, pour le portfolio, nous avons définis précédemment six objets donc il y a le même nombre de classes. De plus, le PSR-1 recommande qu'il y ait une classe par fichier : il y a donc sept fichiers. C'est dans ces fichiers que l'on définit les attributs de chaque objet mais aussi nos fonctions contenant les requêtes pour communiquer avec la base de données.

La vue correspond quant à elle à la représentation des données venant du modèle. Cette partie contient tous les fichiers destinés à l'affichage (pages HTML, XML, photos, vidéos…). Mais ses fonctions ne se cantonnent pas à un simple affichage : la vue est également le moyen d'interaction entre l'utilisateur et le modèle.

Le contrôleur est une partie qui gère les requêtes des utilisateurs. C'est lui qui retourne une réponse en se servant des modèles et de la vue. Lorsque l'utilisateur soumet une requête, le contrôleur vérifie la validité et les règles d'autorisation de cette demande, délègue le traitement et la récupération des données au modèle puis sélectionne la vue correspondant à l'attente du client et enfin délègue l'affichage à la vue.

Pour faciliter la compréhension, voici un schéma :

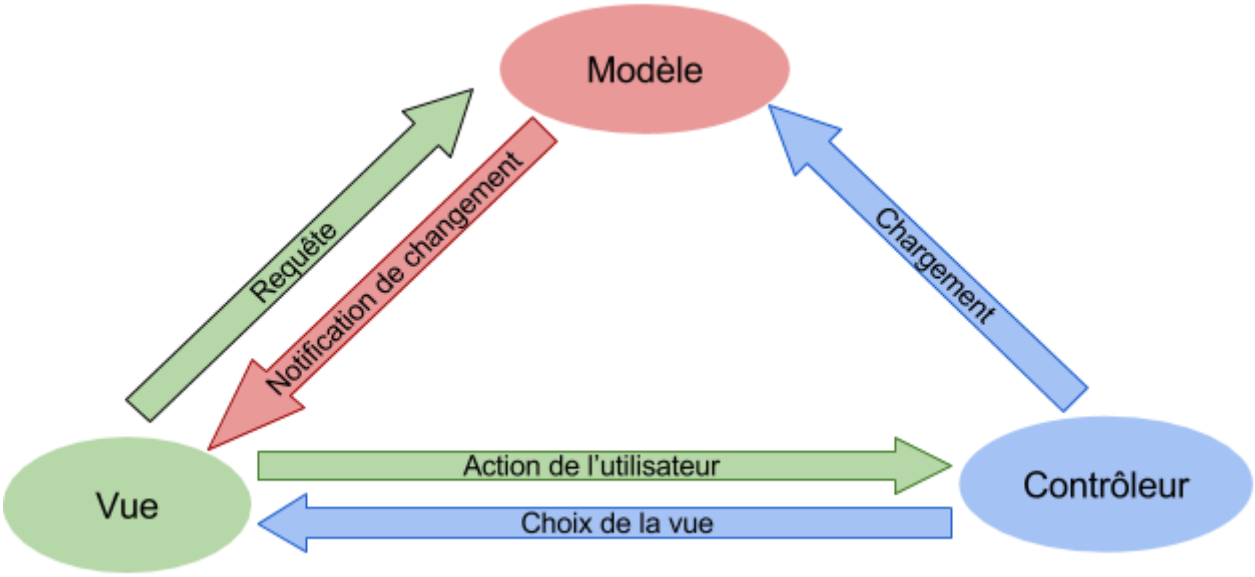


Schéma 1 : fonctionnement d'un modèle MVC

La méthode MVC apporte de nombreux avantages sur un projet. La clarté et l'efficacité de la conception ainsi que la simplification de la maintenance et l'évolution de l'application ont déjà été abordés. Mais elle donne également la possibilité aux développeurs de travailler en simultané sur un seul et même projet sans créer de conflits majeurs : un développeur peut ainsi travailler sur le modèle pendant qu'un autre s'occupe du webdesign. Enfin cette solution permet aussi aux pages HTML de contenir le moins de code serveur possible, ce qui favorise la performance et la sécurité de l'application.

A noter cependant que le choix d'une telle architecture, impliquant la création de plusieurs fichiers minimaux (trois pour être exacte), peut se révéler contraignant. Cela est souvent le cas lors de la réalisation de petits projets. Dans ces cas là, il est préférable d'opter pour une solution moins excessive.

En plus d'une architecture MVC, le développeur utilise bien souvent divers outils l'aidant à produire un travail toujours plus lisible et logique. Par exemple, celui-ci peut très bien avoir recours à ce qu'on appelle un moteur de template.

Les moteurs de templates

Un moteur de template est une technique de programmation qui permet de séparer l'interface graphique du reste de l'application. Cela revient à écrire un fichier de présentation (HTML pour la structure, CSS et images pour le design et enfin JavaScript pour les animations) d'un côté et l'autre partie du code dans un ou plusieurs fichier(s) séparé(s).

Les moteurs de templates fonctionnent par compilation de fichiers : dans un premier temps le fichier PHP est lu puis vient le tour du template. Ensuite les deux codes sont compilés de façon à obtenir un script PHP. Et enfin le code généré est exécuté.

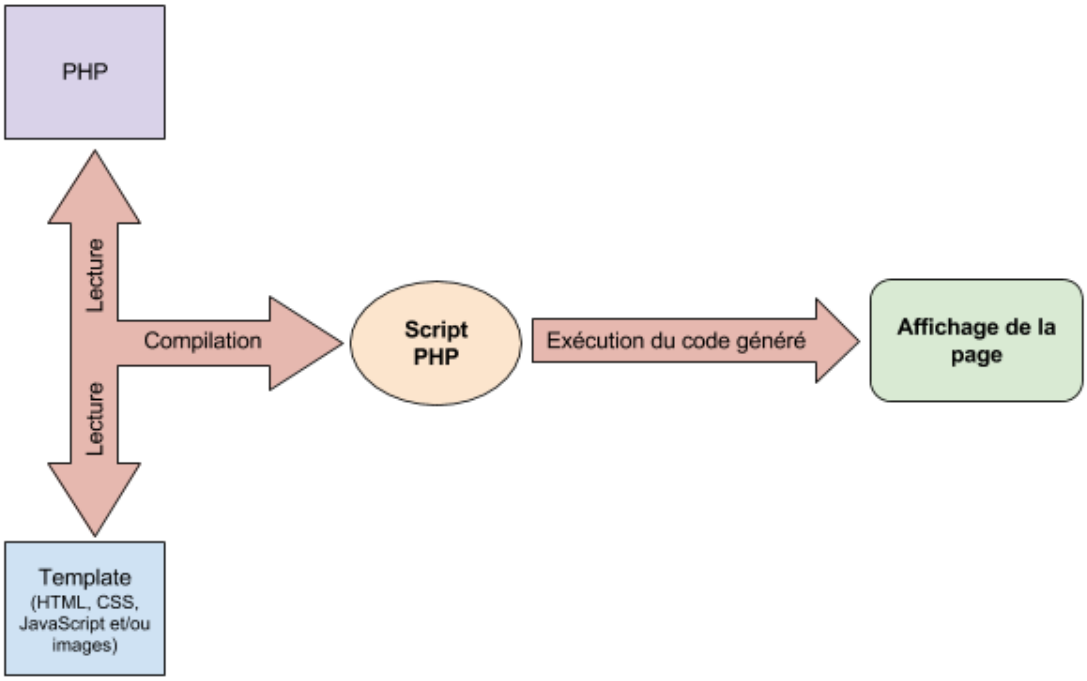


Schéma 2 : schéma du fonctionnement d'un moteur de templates

L'un des premiers avantages de cette séparation est, comme dit précédemment, de pouvoir travailler à plusieurs sur une seule et même application. Le webdesigner peut ainsi travailler en même temps que le développeur back-end. Le second consiste dans le fait qu'aucune requête n'est directement écrite dans les pages HTML. Ainsi, une requête qui est utilisée plusieurs fois sur plusieurs

pages web ne devra être modifiée qu'à un seul endroit : son fichier PHP voire sa fonction. En évitant de placer directement une variable ou une requête, cela raccourcit le fichier et améliore donc sa compréhension. Dans une vision à long terme, il peut également faciliter la révision du code : si l'on modifie le PHP, cela n'impactera pas ou peu le fichier HTML. Et enfin un des grands avantages des moteurs de templates est la mise en cache. Proposée nativement, elle permet d'économiser les ressources des serveurs si on le configure correctement. A l'inverse, l'utilisation d'un tel outil retarde le chargement d'une page en raison du travail supplémentaire que doit fournir le serveur : lire deux fichiers, les compiler et générer un fichier script. De même que la lecture des erreurs reste assez compliquée. Et enfin, pour terminer, le dernier inconvénient est l'apprentissage du langage du template.

En PHP il existe différents moteurs de templates : Smarty, le plus connu, ou encore Twig. Ce dernier est d'ailleurs nativement inclus dans Symfony.

Avec Twig, l'utilisation des données est assez simple puisqu'elles sont introduites dans les pages HTML à l'aide de doubles accolades : `{{ 'mon texte brut' }}` ou encore `{{ maVariable }}`. Dans le développement web il est souvent question de boucles ou de conditions. Twig apporte une alternative au PHP en proposant sa propre syntaxe. L'exemple ci-dessous fait une boucle sur un tableau contenant l'ensemble de mes expériences. Pour chaque entrée, il extrait et affiche le nom de cette expérience (attribut) sous forme de liste à puces :

```
<ul>
    {% for experience in experiences %}
        <li>{{ experience.name }}</li>
    {% endfor %}
</ul>
```

Image 3 : exemple d'une boucle for dans Twig

L'exemple ci-dessous est une condition : s'il y a une erreur dans mon tableau d'erreurs, alors il l'affiche :

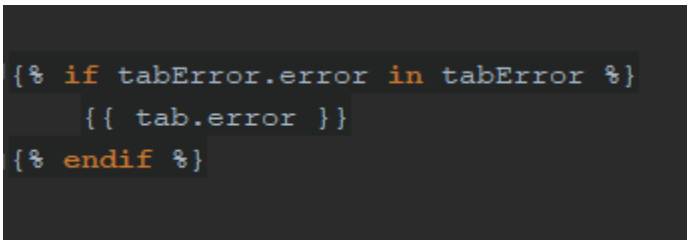


Image 4 : exemple d'une condition dans Twig

Avec les deux exemples ci-dessus, on remarque que le langage de Twig est très proche de celui de PHP. L'adaptation peut donc être relativement simple dès lors que l'on connaît les bases (« `{% … %}` » pour une action et « `{{ … }}` » pour afficher). Afin de faciliter le travail, le moteur de template possède également de nombreuses fonctionnalités telles que les filtres (« `{{ maVariable | lower }}` » permet par exemple de forcer le texte de ma variable à s'afficher en minuscule...).

La possibilité d'héritage est une des fonctionnalités les plus utilisées : il est désormais possible de créer une page HTML (exemple nommée « layout.html.twig ») qui serait commune à toutes les autres pages. Ce fichier contiendrait les métadonnées (*head*), le code pour l'en-tête des pages (*header*) et le pied-de-page (*footer*). Pour chaque autre page du site, ce fichier de base serait hérité à l'aide de la fonction Twig « `{% extends 'nom de mon fichier de base' %}` ». Le corps (*content* par exemple), qui est spécifique à chaque page ou chaque élément spécifique, seraient mentionnés à l'aide de fonctions Twig vides nommées « block » dans la page de base puis seraient également mentionnées dans chaque page en encapsulant le code HTML spécifique.

```twig
<!DOCTYPE html>

<html lang="{{ app.request.locale }}">
    <head>
        <meta charset="UTF-8"/>
        <meta name="author" content="Elodie VIANAI;Web-atrio;">
        <title>{% block title %}<!-- Title of each page here -->{% endblock title %}</title>
        <link rel="stylesheet" type="text/css" href="lib/bootstrap/css/bootstrap.min.css">
    </head>

    <body>
        <header...>

        {% block content %}<!-- Content of each page here -->{% endblock content %}

        <footer...>

        <script src="lib/bootstrap/js/bootstrap.min.js"></script>
    </body>
</html>
```

Image 5 : principe d'héritage des vues dans Twig (layout)

```twig
{% extends 'Layout/index.html.twig' %}

{% block title %}
    Elodie's Portfolio - accueil
{% endblock %}

{% block content %}

    <h1>Bienvenue sur mon portfolio !</h1>

    <p>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed cursus urna ut cursus condimentum. In facilisis
        sapien eu commodo sodales. Fusce ut neque nec ante bibendum cursus nec id nulla. Lorem ipsum dolor sit amet,
        consectetur adipiscing elit. Maecenas tempor aliquam turpis. Vivamus ultricies lacus velit, sed scelerisque
        libero molestie id. Phasellus pellentesque accumsan magna, vel feugiat nisl lobortis at. Praesent imperdiet
        eget tortor id dignissim. Suspendisse non libero purus. Duis vitae lectus eros. Integer ullamcorper quis justo
        interdum euismod. Aliquam tempor ullamcorper mollis. Sed erat tellus, pulvinar id sapien sed, tristique auctor
        nulla.

        Aliquam dignissim, lorem non fermentum commodo, mi nibh condimentum augue, ut ornare est metus ac lacus.
        Aliquam id magna sit amet elit consequat tincidunt id vel diam. Nulla non tincidunt purus. Quisque mauris elit,
        volutpat luctus gravida et, accumsan facilisis lectus. Vestibulum ante ipsum primis in faucibus orci luctus et
        ultrices posuere cubilia Curae; Mauris in ex tincidunt dui aliquet elementum. Integer non orci lacus. Vivamus
        feugiat bibendum enim, sed lacinia lacus tristique ac. Fusce sed tincidunt elit, non iaculis libero. Quisque
        ullamcorper scelerisque felis, ut fermentum est semper suscipit. Nullam semper eros id orci accumsan aliquet.
        Vestibulum rutrum lorem eget metus laoreet sollicitudin.
    </p>
{% endblock content %}
```

Image 6 : principe d'héritage des vues dans Twig (page spécifique)

Ce principe d'héritage est utilisable à l'infini mais une page ne peut hériter que d'une seule autre page. Au-delà, il faut utiliser la fonction « `{% include 'nom de fichier' %}` ».

Le ralentissement de chargement d'une page était mentionné plus haut. L'avantage de Twig est la mise en cache : la page est chargée une première fois puis est mise en cache. De ce fait à chaque fois que l'utilisateur fera appel à cette page, le navigateur n'aura pas à la recharger. Pour ce qui est de la sécurité, le développeur devait autrefois échapper chaque variable à la main. Avec ce moteur de *templates*, l'échappement est activé par défaut.

Les fonctionnalités citées précédemment ne sont qu'un aperçu de Twig et il ne serait pas pertinent ici de toutes les mentionner. De par mon expérience, je trouve Twig relativement simple à prendre en main et pratique pour le développement web : la syntaxe est facile à mémoriser, aucune remarque de ralentissement dans le chargement des pages et un gain de temps lors des besoins de modifications ou de projets à plusieurs.

L'architecture de Symfony

Afin de pouvoir se servir du framework Symfony, il est essentiel de comprendre comment celui-ci est organisé.

L'ensemble des éléments précédents sont indépendants. Cependant ils se rejoignent sur un point : ils ont été choisis par *Sensio Labs* pour composer le framework Symfony. Symfony, comme nous l'avons vu, est un projet PHP qui utilise divers composants de ce langage de façon à ce qu'ils fonctionnent en harmonie : c'est l'interopérabilité. L'ensemble de ces composants sont intégrés sous la forme de modules. Ceux qui sont inclus par défaut avec Symfony sont téléchargés directement dans le dossier « vendor ».

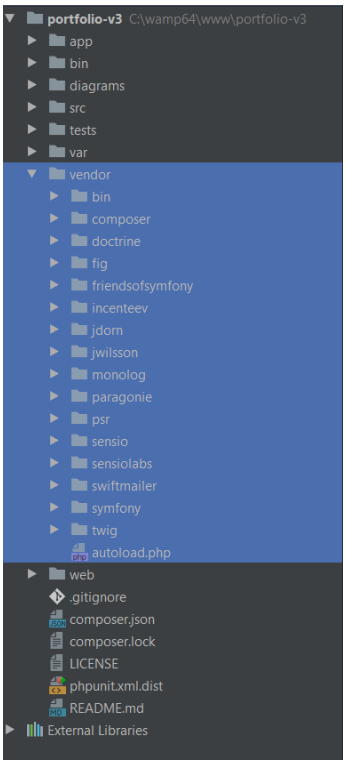


Image 7 : le dossier vendor et ses sous-dossiers dans l'architecture de Symfony

C'est également dans ce dossier que seront stockés les modules externes téléchargés via Composer, un outil de gestion de dépendances PHP qui gère la liste des modules et bibliothèques nécessaires au projet. A l'aide de la ligne commande, on peut ainsi lui demander de télécharger une bibliothèque particulière.

Les autres librairies, qui seront téléchargées manuellement pourront soit être installées dans le dossier « vendor », soit dans un sous-dossier de « web ». Ce dernier est un dossier relativement particulier puisqu'il est le seul élément de l'application accessible publiquement. C'est d'ailleurs la raison pour laquelle il contient les fichiers CSS16 (Cascading Style Sheet pour feuille de style en cascade), les fichiers JavaScript17ou encore les images.

[16] *CSS*, acronyme de *Cascading Style Sheet*, peut être traduit par « feuille de style en cascade » en français. Ce sont des fichiers mettant en forme les pages HTML.
[17] *JavaScript* est un langage de script orienté objet qui permet entre autre d'apporter des animations sur une page ou encore d'exécuter du code sans être obligé de recharger la page.

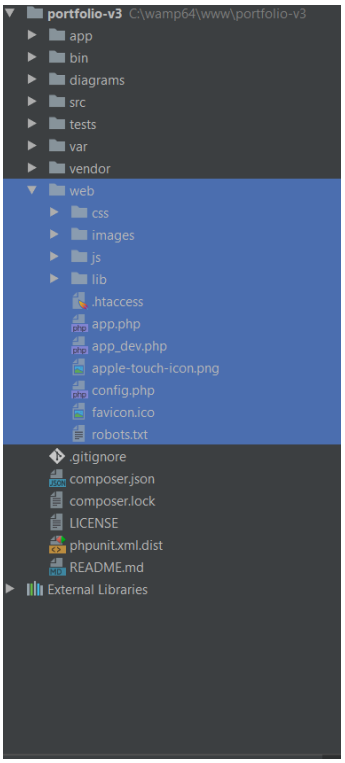


Image 8 : le dossier *web* et ses sous-dossiers dans l'architecture de Symfony

L'ensemble des fichiers du dossier cité précédemment sont appelés depuis « src », qui signifie « source » car il représente la source de notre application. C'est ici que sont stockés les bundles créés par le développeur. Depuis Symfony 3 la majorité des fonctionnalités principales d'une application sont enregistrées dans le bundle[18] principal, nommé AppBundle. A l'intérieur de ce bundle on retrouve le principe de l'architecture MVC sous les appellations Entity pour le modèle, Controller pour le contrôleur et pour terminer une petite exception : la vue se trouve dans le dossier Ressources (qui peut également regrouper les routes, élément qui seront définis ultérieurement) sous le nom de View. Mais ce AppBundle contient également les dossiers DataFixtures, dont les fichiers permettent de générer des données fictives pour alimenter la base de données durant le développement afin de pouvoir tester l'application, *Form* pour la création automatique des formulaires ou encore *Repository* dont les fichiers, un par entité et donc objet, regroupent les requêtes DQL particulières et écrites par le développeur.

[18] Un *bundle* (paquet ou lot en français) est un ensemble de fichiers et de répertoires implémentant une ou des fonctionnalités.

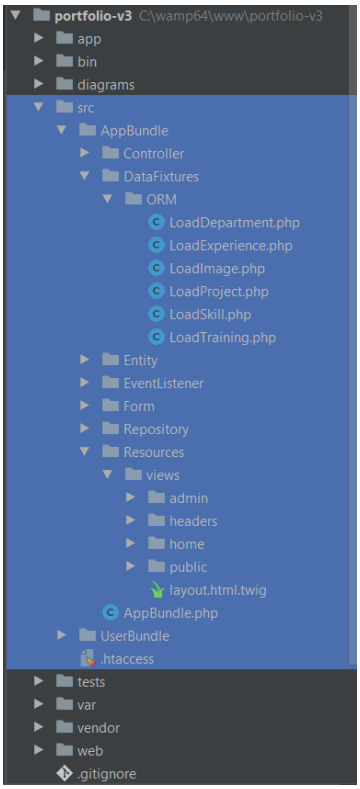


Image 8 : le dossier src, ses sous-dossiers et une partie de ses fichiers dans l'architecture de Symfony

Les éléments cités ici forment une liste qui n'est cependant pas exhaustive, puisque ce dossier correspond au code source de l'application.

Le dossier « app » (à ne pas confondre avec le *AppBundle*) est essentiel au bon fonctionnement de l'application puisqu'il contient sa configuration. Tout d'abord il y a le fichier *AppKernel.php*. Ce fichier sert à enregistrer les différents modules utilisés par l'application afin que le Kernel, c'est-à-dire le noyau de l'application chargé de faire communiquer l'ensemble des éléments entre eux, sache à qui s'adresser. C'est ici que l'on doit enregistrer chaque bibliothèque. Le sous-dossier « Config » regroupe quant à lui l'ensemble des fichiers de configuration, à savoir :

- *config.yml* sert à configurer chaque bibliothèque associée au projet, y compris Twig.

- *config_dev.yml*, *config_prod.yml* et *config_test.yml* servent quant à eux pour la configuration de chaque mode.
- *parameters.yml* est le fichier contenant les informations pour se connecter à la base de données : l'adresse et le port de connexion, le nom de la base de données, le nom d'utilisateur et le mot de passe pour y accéder…
- *routing.yml* enregistre les routes de l'application, c'est-à-dire ses url.
- *security.yml* est un fichier qui traite, comme son nom l'indique, de la sécurité. C'est ici que sont définis l'encodage, les pare-feux et les droits d'accès en fonction du type d'utilisateur.
- *services.yml* permet quant à lui d'enregistrer les services de l'application. Dans Symfony un service est tout simplement un objet qui remplit une fonction précise avec une configuration donnée mais qui fait appel à différentes classes. Cela peut être le cas par exemple pour l'envoi de mails ou encore pour la génération d'un PDF. L'autre particularité est que l'on peut réutiliser ce service à différents endroits de l'application.

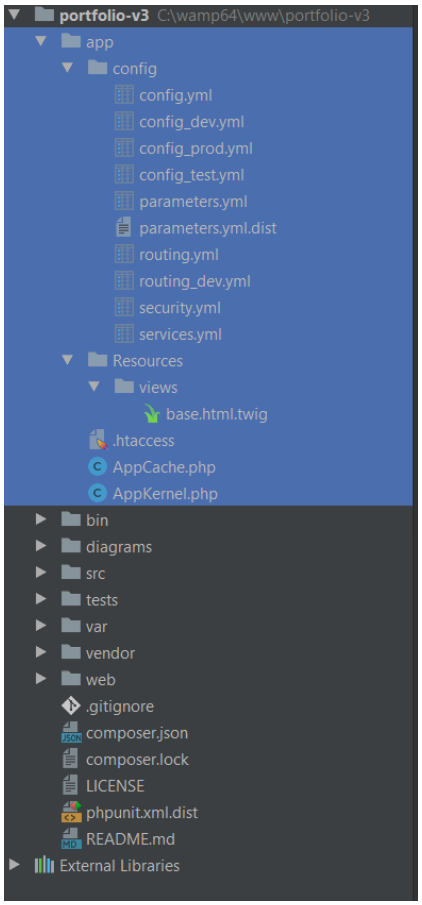


Image 9 : le dossier app, ses sous-dossiers et ses fichiers dans l'architecture de Symfony

L'ensemble de ces fichiers de configuration ne sont pas écrits en PHP mais en YAML. Le YAML, pour *Yet Another Language* dans la version 1.0 puis pour l'acronym récursif *Ain't Markup Language* à partir de sa version 1.1, est un format de représentation des données surtout utilisé pour les fichiers de configuration.

Enfin viennent les dossiers « bin », « tests » et « var ». Le premier rassemble tout simplement des fichiers executables, qui seront lus par la ligne de commande. Le deuxième contiendra des méthodes destinées à la procédure des tests unitaires qui consistent à tester un module ou une partie du code. Ce fichier, s'il est écrit, permet ainsi l'automatisation de ces tests. Et enfin le dernier dossier contient quant à lui les logs, des fichiers recensant les événements de l'application dans un ordre chronologique, ainsi que les fichiers de cache qui enregistrent diverses informations tels que les templates Twig. Ces fichiers

doivent être régulièrement détruits lors de la phase de développement, et plus particulièrement lorsque l'on travaille sur l'interface graphique. Avec Symfony il est d'ailleurs possible de « vider le cache » à l'aide de la ligne de commande (« `php bin/console cache:warmup` » pour Symfony 3).

L'architecture de Symfony peut donc paraître complexe. En effet elle présente de nombreux dossiers et sous-dossiers. Par conséquent il est courant d'avoir des difficultés pour se repérer ou pour retrouver des fichiers. Mais l'avantage de cette architecture est qu'elle est commune à l'ensemble des projets utilisant ce framework. Un développeur peut donc, s'il maîtrise la technologie, facilement intégrer ou reprendre un tel projet utilisant cet outil.

Le framework Symfony est un projet PHP utilisant le langage PHP orienté objet (version 5 et supérieure) qui transforme le codage linéaire en une multitude d'objets interdépendants. Depuis quelques années, le groupe *PHP-Framework Interop Group* s'accorde sur un certain nombre de recommandations, dans l'objectif d'obtenir un consensus pour les transformer en standards. Cela permettrait alors de parvenir à une harmonisation du code PHP. La programmation orientée objet a amené avec elle de nouvelles perspectives dans la programmation, particulièrement en terme d'interopérabilité. Les ORM en sont un exemple, car ils créent une illusion de base de données adaptée à la programmation orientée objet, tout en se servant d'une base de données relationnelle réelle. L'architecture sur laquelle se base Symfony est le concept de Modèle-Vue-Contrôleur, ou MVC, qui sépare la logique métier (les données) de l'affichage (vue). Les avantages de cette méthode sont nombreux tels que l'amélioration de la maintenance d'une application ou encore la possibilité de travailler à plusieurs sur un même projet. Le choix d'une telle architecture peut, et l'est souvent, être couplé avec un moteur de template afin de séparer davantage les données de l'affichage. C'est le cas par exemple de Twig qui est non seulement relativement simple à prendre en main avec l'avantage d'accroître la lisibilité du code mais qui est aussi un des plus rapide de sa catégorie.

Tous ces éléments sont repris par Symfony sous la forme de composants, de façon à offrir un framework robuste et puissant. Cependant ces éléments sont disponibles en-dehors du framework : ils ne lui sont pas propres. Qu'apporte ou n'apporte donc pas Symfony au développement d'une application ?

Deuxième partie : Prise en main de Symfony

Peu après mon arrivée chez Web-atrio, j'ai été placée en formation sur le framework Symfony. Entre le début de ma formation, sur tutoriel, et la fin du projet Portfolio il m'a été nécessaire presque un mois de travail pour terminer ces deux projets. L'application de mes compétences, acquises à partir du tutoriel, sur le projet Portfolio m'ont permis d'aller plus en profondeur du framework. Sans solution préétablie, j'ai alors été obligée de mener une réflexion sur cette application afin de trouver des solutions à mes interrogations mais aussi de faire des choix qui paraissent judicieux à l'instant présent.

Chapitre 3. Analyse des choix

L'application Portfolio avait des spécifications fonctionnelles précises. Celle-ci devait contenir une partie publique (*front-office*) et une partie d'administration (*back-office*). Et pour faire le lien entre les deux il a, par conséquent, été logique de mettre en place un système de gestion des utilisateurs afin de sécuriser la dernière partie. Pour réaliser ces fonctionnalités et au vu des différentes possibilités de réalisations qui s'offraient à moi, j'ai par conséquent eu à faire une sélection. Je vais donc détailler, pour certaines fonctionnalités essentielles à mes yeux, les choix qui m'étaient proposés, les raison pour lesquelles j'ai choisi une option au détriment d'une autre ainsi que la façon dont j'ai procédé pour réaliser ces éléments.

La gestion des utilisateurs

La gestion des utilisateurs est une fonctionnalité primordiale dans une application. En effet il s'agit là de la possibilité de créer un compte, pouvoir modifier les informations qui lui sont associées, pouvoir le supprimer mais aussi de pouvoir se connecter et se déconnecter. Un élément tout aussi important, voire essentiel, est la gestion des droits. Ces derniers représentent une partie de la sécurité d'une application puisqu'ils déterminent qui a le droit d'accéder à tel contenu ou pas. Les choix pour ce domaine doivent donc être judicieusement pensés.

La gestion d'un utilisateur peut être relativement simple puisqu'elle demande simplement un formulaire pour la création du compte qui pourra aussi être réutilisé pour la modification. Lui seront alors associés des fonctions (ou méthodes) permettant d'enregistrer une première fois les informations de l'utilisateur dans la base de données et une seconde qui les modifiera. Une troisième fonction devra également être créée pour la suppression du compte. Tout ceci représente par conséquent un travail

important de création. Avec le système de bundles de Symfony, cette charge de travail peut être réduite en utilisant un bundle déjà existant.

Pour ma part j'ai utilisé *FOS UserBundle*, créé par un groupe nommé *Friends Of Symfony* (« les amis de Symfony » en français). Il s'agit d'un des bundles les plus utilisés qui permet de créer un système de gestion des utilisateurs. Ce bundle est utile dans le sens où tout est pensé : les systèmes de création de compte, de modification, de suppression, de récupération de mot de passe en cas de perte, d'envoi d'emails de confirmation… Un des autres avantages est qu'il se configure facilement avec Doctrine mais aussi d'autres ORM et qu'il s'associe bien avec le pare-feu de Symfony. De plus son utilisation est relativement simple puisqu'il économise au développeur une grande partie de la tâche de création de cette fonctionnalité. En effet pour une utilisation minimaliste je n'ai finalement eu qu'à suivre les étapes suivantes :

- initialiser le bundle (téléchargement et ajout d'une ligne dans le fichier *AppKernel.php*),

```
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = [
            new Symfony\Bundle\FrameworkBundle\FrameworkBundle(),
            new Symfony\Bundle\SecurityBundle\SecurityBundle(),
            new Symfony\Bundle\TwigBundle\TwigBundle(),
            new Symfony\Bundle\MonologBundle\MonologBundle(),
            new Symfony\Bundle\SwiftmailerBundle\SwiftmailerBundle(),
            new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
            new Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle(),
            new AppBundle\AppBundle(),
            new \FOS\UserBundle\FOSUserBundle(),
            new UserBundle\UserBundle(),
        ];
```

Image 10 : initialisation de FOSUserBundle dans le fichier AppKernel.yml de Symfony (projet Portfolio)

- créer une entité *User* (utilisateur en français), composée tout simplement d'un identifiant unique, qui hérite de la classe *BaseUser* provenant de *FOSUserBundle* (à savoir que l'on peut surcharger notre classe si l'on veut rajouter des attributs),

```php
<?php

namespace UserBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use FOS\UserBundle\Model\User as BaseUser;

/**
 * User
 *
 * @ORM\Table(name="user")
 * @ORM\Entity(repositoryClass="UserBundle\Repository\UserRepository")
 */
class User extends BaseUser
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;
}
```

Image 11 : création de l'entité User qui hérite (« extends ») de la classe BaseUser de FOSUserBundle (projet Portfolio)

- intégrer FOSUser dans la configuration du pare-feu (*firewall*) de Symfony qui se trouve dans le fichier *security.yml* (ce point sera approfondi ultérieurement)

```yaml
    # https://symfony.com/doc/current/security.html#b-configuring-how-users-are-loaded
    providers:
        fos_userbundle:
            id: fos_user.user_provider.username

    firewalls:
        # disables authentication for assets and the profiler, adapt it according to your needs
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            pattern: ^/
            form_login:
                provider: fos_userbundle
            logout:
                path: fos_user_security_logout
                target: /
            anonymous: true
```

Image 12 : configuration du firewall avec FOSUserBundle dans le fichier security.yml de Symfony (projet Portfolio)

- intégrer *FOS User* dans la configuration de Symfony (fichier *config.yml*) en renseignant entre autre le nom du pare-feu et le chemin vers l'entité Utilisateur, la façon dont l'utilisateur peut s'enregistrer (type et nom du formulaire, si Symfony doit envoyer une demande de confirmation), etc.

```yaml
# FOSUserBundle Configuration
fos_user:
    db_driver: orm
    firewall_name: main
    user_class: UserBundle\Entity\User
    service:
        mailer: fos_user.mailer.twig_swift
    registration:
        form:
            type: FOS\UserBundle\Form\Type\RegistrationFormType
            name: fos_user_registration_form
            validation_groups: [Registration, Default]
        confirmation:
            enabled: false
#            template: '@FOSUser/Registration/email.txt.twig'
    profile:
        form:
            type: FOS\UserBundle\Form\Type\ProfileFormType
            #validation_groups: [Profile, Default]
    resetting:
        email:
            template: '@FOSUser/Resetting/email.txt.twig'
    from_email:
        address: elodie.vianai@web-atrio.com
        sender_name: user
```

Image 13 : configuration de FOSUserBundle dans le fichier config.yml de Symfony (projet Portfolio)

- configurer les routes vers les pages de *FOS UserBundle* dans le fichier *routing.yml* en rajoutant ces deux lignes :

```yaml
fos_user:
    resource: "@FOSUserBundle/Resources/config/routing/all.xml"
```

Image 14 : écriture de la route vers les pages de FOSUserBundle dans le fichier routing.yml de Symfony

- mettre à jour la base de données, depuis la console, avec cette commande :

```
Elodie Vianai@DELL-1FSD062 MINGW64 /c/wamp64/www/portfolio-v3 (integration)
$ php bin/console doctrine:schema:update --force
```

Image 15 : commande pour mettre à jour la base de données (projet Portfolio)

Une fois l'ensemble de ces étapes effectuées, le développeur dispose désormais d'une fonctionnalité de gestion des utilisateurs complète et opérationnelle. Sans ce bundle, développer un tel système aurait pris plusieurs jours, hors ici cette tâche prend au maximum une journée selon le niveau de modification apporté aux éléments de base de *FOS UserBundle*.

Il a été question précédemment de configurer le pare-feu de Symfony afin que *FOS Userbundle* soit pris en compte par le framework. Le pare-feu, appelé « firewall » en anglais, est un système de sécurité. C'est lui qui détermine si l'utilisateur est authentifié (membre du site) ou anonyme. S'il est anonyme, l'utilisateur est dans ce cas redirigé vers la page d'authentification. C'est à partir de cette authentification (ou non) que les autorisations vont être définies. Ce processus, géré par l'*access control* de Symfony vérifie donc si le rôle de l'utilisateur est autorisé à accéder à tel ou tel contenu du site. Par exemple un simple membre ne sera pas autorisé à entrer dans l'espace d'administration s'il n'a pas les droits. Si la théorie a l'air simple, en pratique cela l'est un peu moins et le fichier *security.yml*[19] le prouve.

La sécurisation est un point essentiel du développement web. Si une application n'est pas suffisamment sécurisée, ses données peuvent en être altérées. Il est donc évident que l'apprentissage et la compréhension de ce point est primordial. Même si Symfony offre des paramètres de sécurité par défaut performants, il ne faut jamais perdre de vue qu'une application est toujours vulnérable. Pour ma part il m'a fallu une journée pour comprendre correctement cette fonctionnalité et mettre en place le système de sécurité.

Le développement d'un système de gestion des utilisateurs est une tâche récurrente dans le travail d'un développeur. La mise en place du processus de gestion est souvent chronophage et la sécurisation peut être relativement compliquée à mettre en place. Au final, la réalisation de cet

[19] Une capture d'écran de ce fichier est disponible en annexe 2.

ensemble aurait pu me prendre plusieurs jours voire une semaine entière. L'utilisation du bundle FOS UserBundle a représenté pour moi un gain temps considérable sur la création d'éléments basiques tels qu'une base de données, des fonctions d'enregistrement, d'édition et de suppression d'utilisateurs..., ce qui m'a permis d'accorder plus de temps au volet sécuritaire de mon application portfolio. La documentation de Symfony a d'ailleurs été un support essentiel dans la compréhension de la configuration de la sécurité.

L'espace d'administration

Après avoir appréhendé la partie sensible et complexe qu'est la gestion des utilisateurs et de la sécurité, j'ai pu ensuite me concentrer sur l'espace d'administration de mon application. Selon les spécifications fonctionnelles, cette zone doit permettre d'ajouter, modifier et supprimer l'ensemble des éléments du portfolio.

Pour chaque objet, il m'a été incombé de créer un CRUD, acronyme pour *Create, Read, Update, Delete* (« création, lecture, mise à jour, suppression » en français). Généralement un CRUD est représenté sous la forme d'un tableau listant les entrées et dont chaque ligne comporte un ou plusieurs liens menant à une des quatre actions citées précédemment. Pour ce cas précis, il fallait donc récupérer l'ensemble des données d'un objet, prenons par exemple les formations. Symfony et son ORM Doctrine proposent des requêtes prédéfinies que l'on appelle à l'aide de fonctions : *findAll()* pour sélectionner toutes les données d'une table, *findOne()* pour ne sélectionner qu'un seul élément de la table à l'aide de l'identifiant (id) ou encore *findBy()* pour sélectionner un ou plusieurs éléments dans la table avec une ou plusieurs conditions (ex : récupérer les formations en fonction de la date et ranger les entrées par ordre décroissant). Pour le CRUD, le choix s'est porté sur cette dernière fonction afin d'ordonner les résultats par ordre chronologiques. Ainsi, ma requête est la suivante :

```
$trainings = $this->getDoctrine()->getManager()->getRepository( className: 'AppBundle:Training')->findBy(
    array(),
    array('endAt' => 'desc'),
    null,
    0
);
```

Image 16 : fonction findBy() pour faire une requête de sélection sur les formations (projet Portfolio)

Dans l'exemple ci-dessus la requête sélectionne toutes les formations, sans aucun critère particulier, mais les range par ordre décroissant en se basant sur la date de fin de la formation. Je n'ai indiqué ici aucune limite de résultats et je commence la sélection à partir de la toute première entrée. Ces requêtes, basiques, sont utiles car elles évitent au développeur de réécrire sans cesse ces petits bouts de codes. Le résultat, stocké dans ma variable « $trainings », est un tableau d'objets que je peux ensuite manipuler à mon aise. Ces objets sont, à la fin de la fonction, envoyé au moteur de templates qui se charge de les afficher. Pour cela je fais appel à la fonction *render()* qui retourne une réponse avec une vue (fichier Twig) et les objets passés en paramètres.

```
        return $this->render( view: '@App/admin/trainings/crud', array(
            'trainings' => $trainings,
        ));
```

Image 17 : fonction render() pour retourner une vue et ses objets afin de les afficher à l'aide du moteur de template (projet Portfolio)

Dans le fichier Twig mentionné dans le *render*, un tableau a été créé en HTML et mis en forme grâce au CSS et à Bootstrap, la célèbre librairie pour le design d'applications de sites web. L'affichage de chacune des formations, stockées dans la variable *$trainings*, sont quant à elle toutes affichées à l'aide d'une boucle *for*. Il n'y a plus qu'à utiliser le mot clef auquel on associe une valeur ou un objet (« *'trainings' => $trainings* ») en tant que variable Twig. Cette dernière, qui rappelons le est un objet, se comporte comme un objet normal :

```
{% for training in trainings %}
    <td>
        {{ trainin.name }}
    </td>
{% endfor %}
```

Image 18 : affichage des formations dans le fichier Twig (projet Portfolio)

L'espace d'administration du portfolio a, au final, été la plus longue à mettre en place, même si l'utilisation de Bootstrap facilite une partie du travail. Pour l'interface graphique j'ai choisi Bootstrap 4,

qui venait tout juste de sortir mais que je trouvais intéressant de découvrir. Au final peu d'éléments diffèrent de la version précédente. Puis il m'a fallu écrire les méthodes nécessaires au CRUD pour chaque élément. Cependant, grâce à Symfony, j'ai pu reproduire mon code de façon à n'avoir qu'à l'adapter pour chaque entité. De ce fait, avec cette réduction de la charge de travail, j'ai pu m'attarder sur l'aspect métier et non technique du développement. En ce qui concerne l'utilisation d'un CRUD, il me paraît assez évident qu'il permet une meilleure visibilité des éléments ainsi qu'une meilleure ergonomie pour l'utilisateur.

Les choix ont été pris selon plusieurs critères. Dans le cas de la gestion des utilisateurs, l'installation de *FOS Userbundle* a été motivée par la renommée du bundle mais aussi par le confort qu'il apportait dans le développement. En effet coder l'ensemble de ce système de gestion aurait été fastidieux et surtout chronophage, alors que cela ne m'a pris qu'une petite demi-journée au lieu de plusieurs jours. Dans le cas de l'espace d'administration, mes décisions ont aussi été attirées par la réduction de la charge de travail ainsi que par l'ergonomie.

Cependant il faut rester conscient qu'un choix à un instant donné peut paraître judicieux mais qu'il le paraîtra moins à un autre. En prenant du recul sur mon travail aujourd'hui, je peux déjà proposer des alternatives, comme par exemple pour le CRUD, ou de nouvelles améliorations.

Chapitre 4. Perspectives d'améliorations

Une application et tout projet web en général ont un cycle de vie. Par conséquent, ils sont voués à évoluer au grès des envies, des besoins et des technologies. C'est pour cela que le code doit être compréhensible et que le développeur doit y penser dès sa création. En ce qui concerne mon portfolio, seulement quelques semaines après l'avoir terminé, je pense déjà à des perspectives d'améliorations, notamment de par mes découvertes et de par l'acquisition de nouvelles compétences. Je vais ici développer deux éventuelles améliorations de mon projet : la première est d'ajouter de nouveaux bundles et la seconde consiste à le rendre plus *responsive* et adaptable.

Vers un *back-office* plus performant

Après mon projet de formation qu'est le portfolio, j'ai intégré un nouveau projet pour un client dans le domaine du conseil et de gestion. Sur le développement de leur application, j'ai pu découvrir de nouveaux outils qui pourraient améliorer mon application de portfolio. Leurs manipulations m'ont alors inspirées d'éventuelles modifications et de nouvelles fonctionnalités : la modification de la partie administration à l'aide de bundles et plugins ainsi que l'ajout d'une possibilité de télécharger mon CV en format imprimable depuis l'espace membre grâce à un bundle supplémentaire.

Le projet qui a suivi celui du portfolio requierait une fonctionnalité CRUD afin que des conseillers puissent gérer la liste de leurs clients ainsi que leurs contrats. Pour cela nous avons utilisé le plugin DataTables, qui fonctionne avec JQuery. DataTables a l'avantage de gérer l'interaction dans un tableau HTML. Il se charge dynamiquement de la pagination, de la réordonnance des colonnes ou encore de la recherche instantanée. Cette fonctionnalité peut être utile dans le cas où la liste de mes compétences ou encore de mes expériences professionnelles grandiraient. Cela permettrait notamment de retrouver plus facilement une compétence dans mon CRUD que je souhaite modifier : je n'aurais qu'à faire une recherche et non plus parcourir l'ensemble du tableau et/ou des pages. Cela peut paraître être un détail à première vue mais, dans le cas d'une longue liste, s'avère être un confort ergonomique. La seconde amélioration que je pourrais faire sur mon projet est la refonte globale de l'espace d'administration en utilisant un bundle dédié. A ce jour il en existe plusieurs, dont Sonata Admin ou encore EasyAdmin. Le premier est de loin le plus connu car le plus complet mais son installation reste néanmoins complexe. Le second est quant à lui beaucoup plus simple à mettre en place et a été créé par Sensio Labs, l'entreprise fondatrice de Symfony. En raison de la faible importance du portfolio il serait donc plus judicieux d'implémenter EasyAdmin, dont l'interface est très proche de son gros concurrent. Sa configuration s'établit via un fichier en YAML et peut être écrite directement dans le fichier *config.yml*. Avec ce bundle, l'espace d'administration est alors développée en seulement quelques minutes.

La seconde amélioration que je pourrais développer pour mon portfolio serait la génération de mon CV en format imprimable directement depuis le PHP. Il existe de nombreux bundle dédiés à la génération de documents. Cependant cette fonctionnalité requiert une utilisation limitée du langage CSS (versions CSS 1 et quelques rares éléments du CSS 2) pour mettre en page les éléments. La tâche de mise

en page d'un PDF est donc ardue et demande souvent du temps. Malgré cet inconvénient, non négligeable, l'avantage est que le rendu du HTML est identique en PDF.

Après avoir testé plusieurs bundles de génération de PDF tels que *DOMPDF*, *WKHTMLTOPDF*, *Knp-Snappy-Bundle* ou encore *TCPDF*. Finalement, sur le projet de gestion de clients et de contrats, nous avons choisi le premier qui semble un peu mieux supporter le CSS 2. Là encore l'installation est relativement simple, tout comme son utilisation. Le travail le plus complexe d'après mon expérience sera alors de mettre en page l'ensemble de mon CV. La solution que je pourrais adopter pour la génération de ce CV pourrait être de créer un service Symfony. Un service Symfony est souvent utilisé lorsqu'une ou plusieurs fonction(s) nécessitants plusieurs objets doivent être appelée(s) depuis n'importe quelle partie du code. Mais il faut bien garder à l'esprit qu'un service correspond à une seule fonctionnalité et qu'il indépendant de l'application de façon à pouvoir être réutilisé. C'est ce qu'on appelle une architecture orientée service. Par exemple dans le cas éventuel de l'écriture d'articles et de leurs publications sur mon application, je pourrais ensuite réutiliser ce service pour générer le ou les articles en PDF. Un service est tout simplement une classe donc pour toute personne qui maîtrise Symfony, sa création est simple.

L'interopérabilité apporte de nombreuses possibilités. Il est aujourd'hui possible d'ajouter une infinité de fonctionnalités à une application, sans pour autant passer plusieurs heures voire plusieurs jours à coder. Bien entendu la compréhension et la prise en main d'un framework, d'une librairie, d'un bundle ou d'un plugin peut parfois prendre du temps mais une fois maîtrisé, l'outil représente un véritable avantage. Dans le cas de l'application de ces possibles améliorations, la réalisation de telles fonctionnalités « à la main » m'auraient pris plusieurs semaines voire mois. L'ensemble des bundles cité tout au long de ce chapitre sont faciles à trouver puisqu'il existe divers sites les recensant : *knpBundles*[20] et *Packagist*[21] pour les plus connus.

[20] La bibliothèque KnpBundles est disponible à l'adresse suivante : http://knpbundles.com/
[21] La bibliothèque Packagist est disponible à l'adresse suivante : https://packagist.org/

Vers une application *responsive* et personnalisée

Avec la multiplication des formats d'écrans, il peut être utile de penser à une application dite « *responsive* », c'est-à-dire qui s'adapte à la taille de son support. On peut aussi prévoir une sorte de personnalisation telle que la personnalisation des pages d'erreurs ou encore la possibilité de traduire l'ensemble du contenu en anglais. De cette manière, l'application pourra à la fois être consultable sur divers types d'écrans (téléphones, tablettes, ordinateurs) mais aussi par des utilisateurs étrangers.

Rendre son application responsive est simple. La facilité est même encore plus poussée lorsque l'on utilise le framework Bootstrap. Le principe consiste en une adaptation de l'organisation du contenu en fonction de la taille de l'écran. C'est donc le CSS (ici la version 3) qui, à l'aide de *Media Queries* (« requêtes média »), détecte la dimension de l'écran. Ensuite selon ce que le développeur a codé, la mise en page correspondante est appliquée. Cependant il faut noter que cela représente 25% de temps de travail supplémentaire. Mais transformer mon application portfolio native, c'est-à-dire conçue au départ uniquement pour l'ordinateur, en une application responsive pourrait être une plus-value par rapport aux portfolios déjà présents sur le web.

En cas d'erreur, comme par exemple si l'utilisateur n'a pas les autorisations nécessaires pour l'accès à une page ou encore s'il y a une erreur, Symfony envoie ce que l'on appelle des « exceptions », autrement dit des pages d'erreurs. Il existe un template par type d'erreur (400, 403, 404, 500...). Or avec Symfony, il est possible de personnaliser ces pages en « *overridant* les templates originaux », c'est-à-dire en les surchargeant. Cela consiste en réalité à créer mes templates d'erreurs en reproduisant leur chemin dans mon dossier « app » (exemple : app/Resources/TwigBundle/views/Exceptions/ error404.html.twig). Il n'y a donc plus qu'à créer ma page personnalisée et cela permet de garder une certaine cohérence avec l'ensemble de mon application. Pour cette amélioration, je pourrais créer les pages « *error* » qui est une page générique, « 403 » pour un accès non autorisé, « 404 » qui correspond à une page non trouvée et « 500 » en cas d'erreur du serveur.

Enfin il pourrait être intéressant de traduire l'application en anglais, afin éventuellement d'envisager une exportation de mes compétences à l'international. Symfony intègre par défaut un composant de traduction : The Translation Component. Il faut utiliser la classe Translator, qui nécessite un seul argument : la locale (pays et langue par défaut). Pour traduire, le Translator va alors comparer la chaîne de caractères donnée par rapport à un catalogue. La problématique ici est que le développeur

doit rédiger lui-même le catalogue, ce qui peut-être contraignant. Pour que Twig sache quels éléments sont à traduire, il n'y a qu'à utiliser son filtre « trans ». On peut l'utiliser de manières différentes : soit directement dans la variable (« `{{ 'ma chaîne de caractères ou ma variable' | trans }}` ») ou alors dans une balise bloc (« `{% trans %} Mon texte {% endtrans %}` »). On peut également faire appel directement au service de traduction du composant, utilisé par Twig, afin de rendre la traduction totalement indépendante.

Vous l'aurez compris, la tendance est aujourd'hui à l'adaptabilité et à la personnalisation. Le but est donc que mon application s'ajuste à l'utilisateur tant en terme de support que de langue. Bien entendu traduire l'application dans toutes les langues représenterait un travail fastidieux, se contenter de l'anglais est donc un choix logique. L'adaptation des supports, quant à elle, pourrait être intéressante étant donné la tendance qui tend vers une consultation sur smartphone.

Le plus gros travail dans ce projet de portfolio a été la réalisation du *back-office* puisqu'il fallait à la fois créer une gestion des utilisateurs mais aussi une interface d'administration avec les actions qui lui incombent. Gérer des utilisateurs doit donc forcément impliquer un système de sécurité qui joue sur des rôles biens définis. Ce *back-office* pourrait aujourd'hui être optimisé avec notamment l'adoption d'un bundle dédié tel qu'*EasyAdmin*. Le portfolio est l'équivalent d'un CV et doit donc montrer l'ensemble des compétences d'un développeur. Il pourrait être judicieux de mettre à disposition des utilisateurs, authentifiés bien sûr, un bouton permettant de télécharger mon CV généré directement depuis le code PHP grâce au bundle *DOMPDF*. En enfin pour terminer, dans l'éventualité d'une vente de mes compétences à l'international, il pourrait être intéressant de traduire l'application en anglais à l'aide du composant *The Translation Component*.

Troisième partie : Synthèse

Symfony, créé par l'entreprise Sensio Labs, est le framework PHP le plus connu et le plus utilisé. De nombreuses entreprises ont opté pour cette solution lors de la création de leur application : Dailymotion (service français d'hébergement de vidéos), le leader du covoiturage Blablacar, Total (un des leaders des entreprises pétrolières et gazières), le prestigieux magazine de mode américain Vogue ou encore la Fédération Française de Rugby. Mais d'autres applications, parfois même concurrentes, reposent sur ce framework : les CMS[22] *Drupal* et *Joomla*, les solutions d'e-commerce *Magento* et *Prestashop*, le framework *Laravel* ou encore le micro-framework *Silex*. Ces utilisateurs prestigieux démontrent la robustesse et l'efficacité de ce projet PHP dans le milieu du développement web. Faisant partie des leader de son marché et étant souvent demandé, l'un des enjeux de mon stage a donc été de le prendre en main. Au cours de ces six derniers mois j'ai pu réaliser divers projets sous Symfony 3, ce qui m'a permis de constater ses points forts et ses points faibles.

Malgré un apprentissage qui peut être long et parfois complexe, Symfony offre une installation et une configuration des projets simples. En quelques lignes de code, mon portfolio était paramétré et affichait une page de bienvenue. Un des points forts du framework est qu'il propose une sécurité minimale, notamment contre les failles. Ainsi il n'est plus obligatoire de penser à sécuriser chaque requête. La partie technique du développement étant moins conséquente, le développeur peut désormais se focaliser sur l'aspect métier. Il dispose donc de plus de temps pour se consacrer à la création des tests : les tests unitaires, qui consistent à vérifier la bonne exécution d'un composant ou d'une petite partie (fonctions, etc.) de l'application, ou encore les tests globaux de l'application. D'ailleurs, l'architecture propose dès sa création un dossier de tests afin d'en faciliter leur développement.

La phase de tests permet entre autre de vérifier la performance de l'outil. De manière générale, Symfony a un temps de chargement des pages réduits, notamment grâce à un code optimisé et à la fonction cache HTTP qui est native. Les *templates* et tous les médias (feuilles de styles, images, vidéos…) ne sont alors chargés qu'une seule fois puis gardés en mémoire par le navigateur. La relation entre le serveur et le client (navigateur de l'utilisateur) est de ce fait moins fréquente.

[22] *CMS* (*Content Management System* en anglais) ou système de gestion de contenu en français : logiciels de création et de mise à jour dynamique de sites web.

Basé sur les recommandations PSR, qui tendent à devenir des standards de PHP, le code du framework est analysé de façon à l'harmoniser. De même que l'adoption d'une architecture MVC (Modèle-Vue-Contrôleur) apporte une logique commune. La maintenabilité de l'application est simplifiée et n'importe quel nouvel arrivant sur le projet est à même s'intégrer facilement.

Les nouveaux arrivants sur Symfony ont à leur disposition une documentation très complète, qui détaille la méthodologie. Si le développeur ne trouve toujours pas ses réponses, il peut toujours trouver de l'aide auprès de la communauté, très active. C'est cette même communauté qui met à disposition, sur de nombreuses plateformes, des bundles. Ces librairies sont très utiles et offrent de nombreuses possibilités en termes de fonctionnalités. De plus, il est possible de les réutiliser sur divers projets. Cependant la configuration de certaines librairies peuvent parfois être compliquées, et la documentation floue, ou celles-ci non mises à jours. Il est donc important de choisir un bundle dont toutes les informations en font un élément compatible. Le nombre de téléchargements est également un critère de choix à prendre en compte.

Les mises à jour importantes peuvent aussi apporter certaines problématiques telles que le suivi ou encore la compatibilité entre deux versions. Par exemple avec le framework de Sensio Labs, le passage de Symfony 1 à Symfony 2 a nécessité de revoir le code des applications avec notamment un changement drastique dans l'architecture.

Enfin pour terminer il est important de garder à l'esprit que Symfony est un framework très complet et donc forcément lourd. En effet lorsque l'on installe Symfony, on télécharge une trentaine de composants. L'utilisation de ce framework n'est donc pas recommandé pour tous les projets. Par exemple un site vitrine n'utilisera pas le quart des fonctionnalités natives et aura plutôt besoin d'un développement maison voire d'un site créateur de site en ligne ou un *CMS*. Ce dernier est d'ailleurs aussi conseillé dans le cadre de la réalisation d'un blog. Une alternative aux gros frameworks est également envisageable : les micro-frameworks, qui, contrairement à leur grand-frère, ne sont composés que des composants minimalistes.

Un projet web est donc un projet qui se réfléchit et dont le choix des outils est primordial tant en terme de développement, de maintenance que de performances. Symfony comporte de nombreux avantages mais n'est pas forcément adapté à tous les projets. Malgré tout il reste aujourd'hui un des leaders du marché du développement web en PHP puisque qu'il a dépassé la barre du milliard de téléchargements et que de nombreux projets reposent sur ce framework.

Conclusion

La création de mon portfolio, dans le cadre d'une formation interne, a été un projet très intéressant, car personnel, et très formateur en tous points. Il m'a permis d'avoir une meilleure compréhension de Symfony mais aussi de la programmation orientée objet, dont PHP. A mes yeux ce framework représente d'immenses possibilités de développement. La préoccupation technique étant amoindrie, j'ai davantage pu m'attarder sur des aspects plus complexes. La majorité des choix que j'ai pu prendre ont été influencés par les conseils de mon tuteur de stage, par la popularité des bundles, par la facilité que cela représentait (compétences acquise et/ou documentation riche) mais aussi par le défi que cela pouvait représenter. Mais Symfony n'a pas été simple à prendre en main et a d'ailleurs représenté un challenge à lui seul.

A l'heure actuelle, Symfony en est à sa version 3.3. S'il avait fallu attendre quatre ans entre Symfony 2 et Symfony 3, ce ne sera que deux ans entre la version 4 et sa précédente. En effet en mars 2017 lors du *SymfonyLive*, la conférence de la marque, son créateur Fabien POTENCIER a annoncé que les mises à jours mineures sortiraient tous les six mois et les majeures tous les deux ans. La sortie d'un Symfony 4 est d'ores et déjà prévue pour la fin de l'année. Cette nouvelle version du framework proposera une nouvelle façon d'installer les éléments de base de Symfony puisque l'on ne chargera plus autant de composants qu'avant, mais uniquement les essentiels. Les dépendances nécessaires au développeur seront donc à télécharger de sa propre initiative. L'installation des bundles sera également facilitée puisqu'il ne sera plus nécessaire de faire autant d'étapes qu'avant pour l'enregistrer : Symfony s'en chargera tout seul. Ce Symfony 4 s'annonce donc plus léger et continue d'essayer de simplifier le travail du développeur.

Sitographie

- Bootstrap. *Bootstrap the most popular HTML, CSS and JS library* [en ligne]. Disponible à l'adresse : https://getbootstrap.com/docs/4.0/getting-started/introduction/ [consulté le 01/09/2017]
- BOYER Jonathan. *Grafikart* [en ligne]. Disponible à l'adresse : https://www.grafikart.fr/ [consulté le 01/09/2017]
- Developpez LLC. *Developpez.com, le club des développeurs et IT pro* [en ligne]. Disponible à l'adresse : https://www.developpez.com/ [Consulté le 01/09/2017]
- Doctrine. *Doctrine project* [en ligne]. Disponible à l'adresse : http://www.doctrine-project.org/ [Consulté le 01/09/2017]
- LEULLIETTE Baptiste. "*Les changements majeurs à venir sur Symfony 4*", *100% web par Kaliop* [en ligne]. Publié le 18/05/2017. Disponible à l'adresse : http://blog.kaliop.com/blog/2017/05/18/les-changements-majeurs-a-venir-sur-symfony-4/ [consulté le 01/09/2017]
- OpenClassrooms. *OpenClassrooms* [en ligne]. Disponible à l'adresse : https://openclassrooms.com/ [Consulté le 01/09/2017]
- OpenClassrooms, NEBRA Mathieu . *Tutoriel « Concevez votre site web avec PHP et MySQL »* [en ligne]. Mise à jour le 14/08/2017. Disponible à l'adresse : https://openclassrooms.com/courses/concevez-votre-site-web-avec-php-et-mysql [Consulté le 01/09/2017]
- OpenClassrooms, THUILLIER Victor. *Tutoriel « Programmez en orienté objet en PHP »* [en ligne]. Mise à jour le 2/08/2017. Disponible à l'adresse : https://openclassrooms.com/courses/programmez-en-oriente-objet-en-php [Consulté le 01/09/2017]
- OpenClassrooms, SANCHEZ Luc, PESQUET Baptiste. *Tutoriel « Évoluez vers une architecture PHP professionnelle »* [en ligne]. Mise à jour le 25/07/2017. Disponible à l'adresse : https://openclassrooms.com/courses/evoluez-vers-une-architecture-php-professionnelle [Consulté le 01/09/2017]
- OpenClassrooms, DUPRET Robin. *Tutoriel « Utilisation de Twig, un moteur de templates ! »* [en ligne]. Mise à jour le 4/04/2017. Disponible à l'adresse

https://openclassrooms.com/courses/utilisation-de-twig-un-moteur-de-templates [Consulté le 01/09/2017]

- OpenClassrooms, TAFANI-DEREEPER Christophe. *Tutoriel « Utilisation d'un ORM : les bases de Doctrine* » [en ligne]. Mise à jour le 08/01/2013. Disponible à l'adresse : https://openclassrooms.com/courses/utilisation-d-un-orm-les-bases-de-doctrine [Consulté le 01/09/2017]
- OpenClassrooms. *Tutoriel « Apprendre à utiliser Doctrine* » [en ligne]. Mise à jour le 01/06/2015. Disponible à l'adresse : https://openclassrooms.com/courses/apprendre-a-utiliser-doctrine [Consulté le 01/09/2017]
- OpenClassrooms, BACCO Alexandre. *Tutoriel « Développez votre site web avec le framework Symfony* » [en ligne]. Mise à jour le 28/08/2017. Disponible à l'adresse : https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony [Consulté le 01/09/2017]
- PESQUET Baptiste. "*Le modèle MVC*", *prof.bpesquet.fr* [en ligne]. Disponible à l'adresse : http://prof.bpesquet.fr/cours/modele-mvc/ [consulté le 01/09/2017]
- Sensio Labs. *Symfony* [en ligne]. Disponible à l'adresse : https://symfony.com/ [Consulté le 01/09/2017]
- Sensio Labs. *Twig : the flexible, fast, and secure template engine for PHP* [en ligne]. Disponible à l'adresse : https://twig.symfony.com/ [Consulté le 01/09/2017]
- Stack Overflow. *Stack Overflow - Where Developers learn, share, & build careers* [en ligne]. Disponible à l'adresse : https://stackoverflow.com/ [Consulté le 01/09/2017]
- SUPINFO. *Articles | SUPINFO, École Supérieure d'Informatique* [en ligne]. Disponible à l'adresse : https://www.supinfo.com/articles/ [consulté le 01/09/2017]
- SUPINFO, HENRI "*Jacques Adrien. Framework : Une boîte à outils très pratique*", *Articles | SUPINFO, École Supérieure d'Informatique* [en ligne]. Publié le 09/06/2016. Disponible à l'adresse : https://www.supinfo.com/articles/single/1782-framework-une-boite-outils-tres-pratique [consulté le 01/09/2017]
- The PHP Framework Interop Group (PHP-FIG). *Site internet du PHP-FIG* [en ligne]. Disponible à l'adresse : http://www.php-fig.org [Consulté le 01/09/2017]

- The PHP Group. *PHP : Hypertext Processor* [en ligne]. Disponible à l'adresse : http://php.net/ [Consulté le 01/09/2017]
- VOISIN Guillaume. "*L'Architecture MVC dans le développement d'un site internet*", *GuillaumeVoisin.fr*. Publié le 17 avril 2009. Disponible à l'adresse : http://www.guillaumevoisin.fr/internet/larchitecture-mvc-dans-le-developpement-dun-site-internet [consulté le 01/09/2017]
- W3Schoolsl. *W3Schools Online Web Tutorials* [en ligne]. Disponible à l'adresse : https://www.w3schools.com/ [consulté le 01/09/2017]

Table des matières

Résumé 3

Mots-clefs 4

Remerciements 5

Glossaire 6

Notes 9

Introduction **10**

Première partie : De PHP orienté objet à Symfony **13**

Chapitre 1. PHP orienté objet : les bonnes pratiques 14

La programmation orientée objet 14

Les recommandations PSR 17

La méthode ORM 18

Chapitre 2. L'architecture web 19

L'architecture MVC 20

Les moteurs de templates 21

L'architecture de Symfony 26

Deuxième partie : Prise en main de Symfony **33**

Chapitre 3. Analyse des choix 34

La gestion des utilisateurs 34

L'espace d'administration 39

Chapitre 4. Perspectives d'améliorations 41

Vers un *back-office* plus performant 42

Vers une application *responsive* et personnalisée 44

Troisième partie : Synthèse **46**

Conclusion **50**

Sitographie 51

Table des matières 54

Annexe 1 : Liste des technologies du projet Portfolio 56

Annexe 2 : Capture d'écran du fichier *security.yml* 57

Annexe 1 : Liste des technologies du projet Portfolio

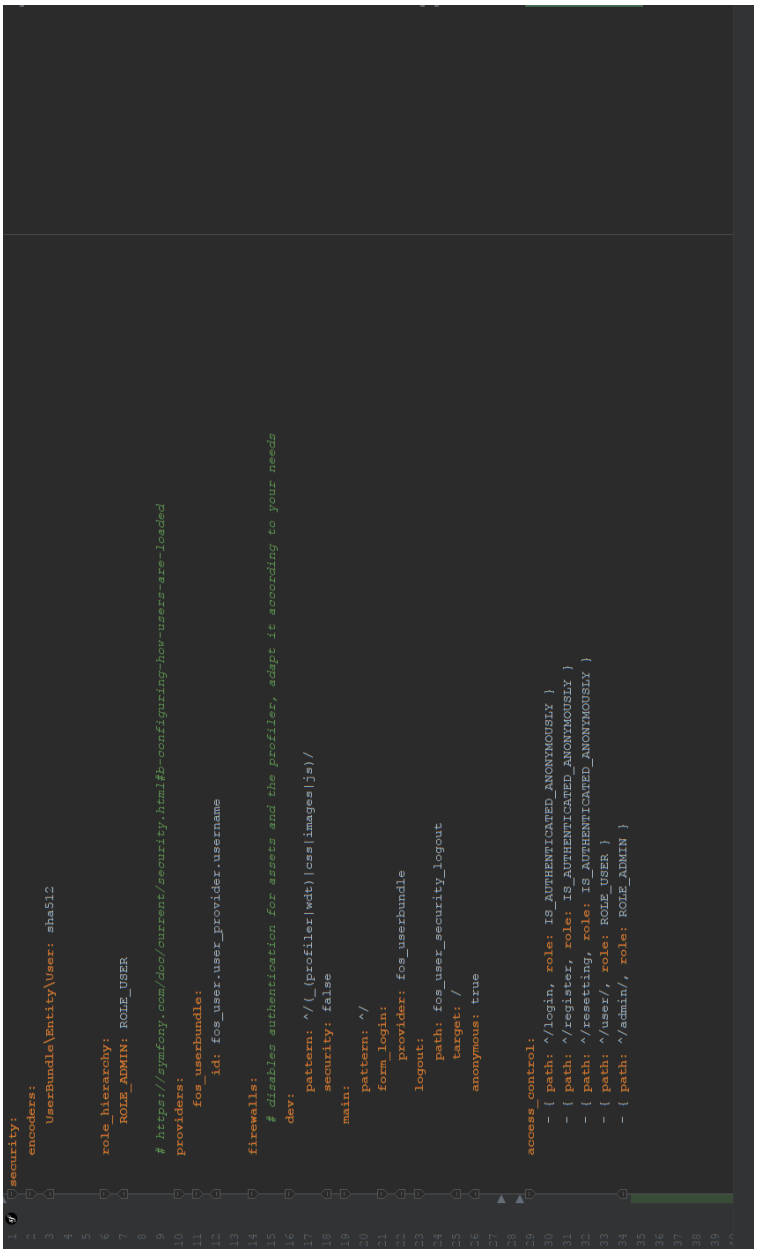
Back-end :

- PHP 7.0.10
- MySQL 5.0.12 (base de données)
- Symfony 3.3.6 (framework)
- Doctrine 2.5 (ORM)
- Apache 2.4.23 (serveur)
- Twig 2 (moteur de template PHP)

Front-end :

- HTML 5
- CSS 3
- JavaScript
- Bootstrap 4 (framework CSS et JavaScript)
- JQuery (framework JavaScript)

Annexe 2 : Capture d'écran du fichier *security.yml*



```
security:
    encoders:
        UserBundle\Entity\User: sha512

    role_hierarchy:
        ROLE_ADMIN: ROLE_USER

    # https://symfony.com/doc/current/security.html#b-configuring-how-users-are-loaded
    providers:
        fos_userbundle:
            id: fos_user.user_provider.username

    firewalls:
        # disables authentication for assets and the profiler, adapt it according to your needs
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            pattern: ^/
            form_login:
                provider: fos_userbundle
            logout:
                path: fos_user_security_logout
                target: /
            anonymous: true

    access_control:
        - { path: ^/login, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/register, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/resetting, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/user/, role: ROLE_USER }
        - { path: ^/admin/, role: ROLE_ADMIN }
```