Dossier technique
Site web de Pierre LeFournier

Projet de Licence 3 d'informatique

Lê–Grigy Matthieu et Leclair Valentin

université de TOURS

9 décembre 2023

Table des matières

1 Projet

1.1 Contexte du projet

Pour ce projet, nous avons choisi de poursuivre le sujet initialement commencé dans le module de conception ergonomique d'interface.

Ce projet à pour objectif de répondre aux attentes d'un client fictif. Un artisan boulanger-pâtissier nommé "Pierre LeFournier" nous à contacté afin de lui développer un site capable de guider ses clients vers ses nouveaux services tout en mettant en avant son expertise. Le site devra donc présenter la boulangerie "Douce Saveur" ainsi que l'expérience de Pierre LeFournier. Il devra également permettre aussi aux utilisateurs de candidater à des offres de stage mais aussi d'envoyer des devis au service de traiteur de l'artisan.

Le scénario :

Avant d'être une marque, Pierre LeFournier est avant tout un artisan passionné par le domaine de la boulangerie pâtisserie. En tant que grand défenseur des traditions françaises, M. LeFournier s'est rapidement mis à l'œuvre afin de créer sa propre boulangerie pâtisserie joliment intitulée "Douce saveur".

Aujourd'hui, fort du succès de cette dernière, l'artisan parisien souhaite proposer d'autres services sous la garantie de son nom. Afin de faire connaître ces nouvelles activités, il nous à demander de lui concevoir un site web lui permettant d'exposer l'identité de sa nouvelle marque et ainsi fidéliser ses clients à ses nouveaux services.

1.2 Clients cibles

Le site web que nous développons s'adresse à trois types d'utilisateurs :

- **Les visiteurs :** regroupe l'ensemble des utilisateurs n'ayant pas besoin de s'authentifier.
- **Les gestionnaires :** personnel de l'artisan ayant la possibilité de se connecter au *back office* pour pouvoir gérer les demandes de devis, les offres de stage et d'éventuelles candidatures.
- **Les administrateurs :** sont des gestionnaires qui ont également la possibilité de gérer l'ensemble des utilisateurs ayant accès au *back office*.

1.3 Cahier des charges

A l'issue de ce développement, le site devra contenir ces différents éléments fonctionnels :

- **Site fonctionnel** avec un minimum de cohérence graphie afin de susciter de l'intérêt chez les clients et nouveaux clients de Pierre LeFournier.
- **Une page d'offre de stage :** offrant la possibilité à tous visiteurs de consulter et de candidater à une offre de stage.
- **Une page devis :** afin que chaque visiteur puisse demander un devis concernant le nouveau service de traiteur.
- **Une authentification :** permettant aux utilisateur d'accéder au *back office*.
- **Un *back office* :** partie du site dont l'accès est réservé aux utilisateurs de manière sécurisée. Elle permettra la gestion des devis, des offres de stage, des candidatures et des utilisateurs dépendamment du rôle de l'utilisateur.

Tout cela en respectant la limite de livraison du projet, prévu pour le 11 décembre 2023.

2 Environnement de développement

2.1 Présentation du Framework Symfony

Un framework est une architecture applicative fournissant un ensemble d'outils et de composants logiciels réutilisables, tels qu'une bibliothèque de codes, qui sont conçus pour faciliter et standardiser le développement de logiciels. Il intègre des pratiques et normes de développement pour guider les développeurs. En agissant comme un squelette sur lequel les applications peuvent être construites, un framework simplifie le processus de développement en offrant une structure de base, réduisant ainsi le temps et l'effort nécessaires pour développer de nouvelles applications.

Symfony est le framework utilisé pour réaliser ce projet, il possède plusieurs avantages :

- Architecture MVC : Symfony utilise le modèle MVC (Modèle-Vue-Contrôleur), facilitant l'organisation du code et la séparation des préoccupations.
- Composants Réutilisables : Il est constitué de composants individuels qui peuvent être réutilisés dans différents projets, augmentant l'efficacité et la modularité.

— Flexibilité et Extensibilité : Symfony est très flexible, permettant aux développeurs de choisir les composants qu'ils souhaitent utiliser. Il est facilement extensible avec des bundles (paquets).
— Performances Optimisées : Conçu pour des performances élevées, Symfony est adapté pour des applications web complexes et à fort trafic.
— Communauté et Ressources : Il bénéficie d'une grande communauté de développeurs et d'un vaste éventail de ressources, de tutoriels et de documentation.
— Compatibilité avec les Bases de Données : Symfony est compatible avec de nombreuses bases de données comme MySQL, PostgreSQL, Oracle, et SQLite.
— Outils de Développement : Intègre des outils de débogage et de développement web, comme Symfony Profiler et Web Debug Toolbar, pour simplifier le développement.
— Sécurité : Offre des fonctionnalités de sécurité, comme la protection contre les attaques et autres vulnérabilités web courantes.
— Tests Unitaires et Fonctionnels : Facilite la réalisation de tests unitaires et fonctionnels avec des outils comme PHPUnit.
— Internationalisation : Prise en charge de l'internationalisation et de la localisation pour créer des applications multilingues.

2.2 Description des différentes technologies utilisées

Voici l'ensemble des technologies que nous utilisons :

— MySQL - version 5.7.39
— Apache - version 2.4.54
— PHP - version 8.1
— Symfony - version 6.1
— Sass - version 1.69.5
— Webpack - version 5.74.0

2.3 Procédure d'installation

Voici l'ensemble des pré-requis pour pouvoir installer et lancer notre projet :

— Un serveur local tel que XAMPP (sous Windows) :https://www.apachefriends.org/
ou Mamp (sous MacOS) : https://www.mamp.info/en/mac/
— L'outil de *versioning* Git : https://git-scm.com/
— NodeJS avec le gestionnaire de package npm ou yarn : https://nodejs.org/
— Php : https://www.php.net/downloads
— le gestionnaire de dépendances Composer : https://getcomposer.org/
— Installation du Framework Symfony comme expliqué dans la documentation officiel : https://symfony.com/download

Concernant notre projet, pour l'importer et le lancer vous devez suivre les étapes décrite ci-dessous que vous pouvez retrouver dans le `readme.md` du répertoire GitHub https://github.com/valentinlym/PierreLeFournier_website

```
git clone git@github.com:valentinlym/PierreLeFournier_website.git
```

Modifier la ligne 22 du fichier .env avec l'identifiant et le mot de passe de votre base de données.

```
composer install
symfony console doctrine:database:create
symfony console doctrine:migration:migrate
yarn install
yarn run build
```

Vous devez importer des données de test dans votre base de données avec le fichier `data.sql` ainsi vous pourrez vous connecter au *back office* avec l'identifiant : admin@exemple.com et le mot de passe : admin. Parfait, maintenant vous pouvez lancer le projet via `symfony serve`, celui ci se lancera en mode production. Il faut modifier la ligne 18 du fichier .env avec `APP_ENV=dev` pour avoir le mode de développemnt de Symfony.

3 Architecture et conception

3.1 Modèle de données

Notre modèle de données est structuré autour de quatre tables principales. La première table, "offre_de_stage", regroupe toutes les informations relatives aux offres de stage. La deuxième, "candidature", fait le lien entre les candidats et les offres de stage, en référençant l'ID de l'offre concernée. La troisième table, "utilisateur", contient les informations d'identification des utilisateurs, y compris leurs rôles. Enfin, la quatrième table, "devis", est dédiée au stockage des informations des devis complétés via le formulaire correspondant sur le site.

Modèle Logique de Données (MLD) :

- offre_de_stage(id : int PK, titre : varchar(255), salaire : varchar(50), duree : varchar(50), description : varchar(500), brouillon : tinyint(1))
- candidature(id : int PK, id_offre_de_stage_id : int FK(offre_de_stage.id), nom : varchar(180), prenom : varchar(180), email : varchar(180))
- utilisateur(id : int PK, email : varchar(180), roles : longtext, mot_de_passe : varchar(255))
- devis(id : int PK, nom : varchar(50), prenom : varchar(50), objet : varchar(255), description : varchar(500), email : varchar(180))

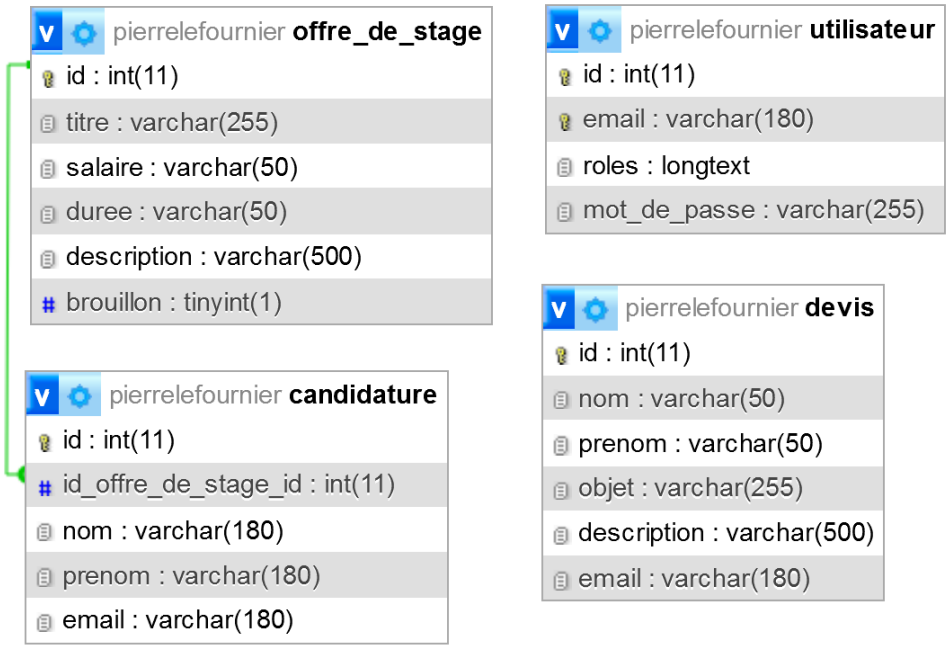

FIGURE 1 – MCD de la base de donnée

3.2 Découpage des *controllers*

Concernant les contrôleurs, nous les avons catégorisé en deux parties (segmentation par défaut de Symfony). Dans le répertoire "/src/Controller/" on retrouve quatre contrôleurs :

- Deux contrôleurs, `OffreDeStageController` et `DevisController` respectivement pour les pages dynamiques offre de stage et traiteur.
- Le contrôleur, `StaticController` qui s'occupe de l'ensemble des pages statiques du site.
- Le contrôleur, `AuthentificationController` générer par la commande `symfony console make:auth` pour gérer la connexion au *back office*

Puis dans le répertoire '/src/Controller/Admin/' nous retrouvons l'ensemble des contrôleurs pour la partie *back office* géré par easyadmin.

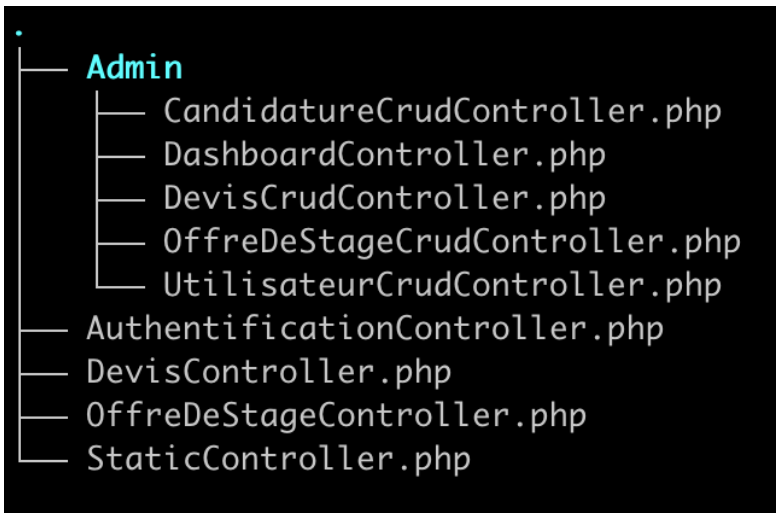

FIGURE 2 – Organisation des contrôleurs

3.3 Présentation du *templating*

Dans le dossier templates situé après le dossier src du projet Il y a 5 dossiers répartissant les différents fichiers twig en fonction de leurs rôles :

- base : qui contient les composants réutilisables tel que *footer* et le *header*
- bundles : qui stocke et remplace *(overwrite)* les différentes pages d'erreur, initialement proposées par symfony.
- dynamic : les pages dynamiques du site
- security : la page de *login*
- static : les pages statiques du site

Ainsi que deux fichiers `base.html.twig` et `base-light.html.twig` qui constituent tout deux la base `html` de chaque pages avec comme différence pour le dernier de ne pas importer les composants *header* et *footer*, par exemple la page de connexion n'a pas besoin de ces deux composants.

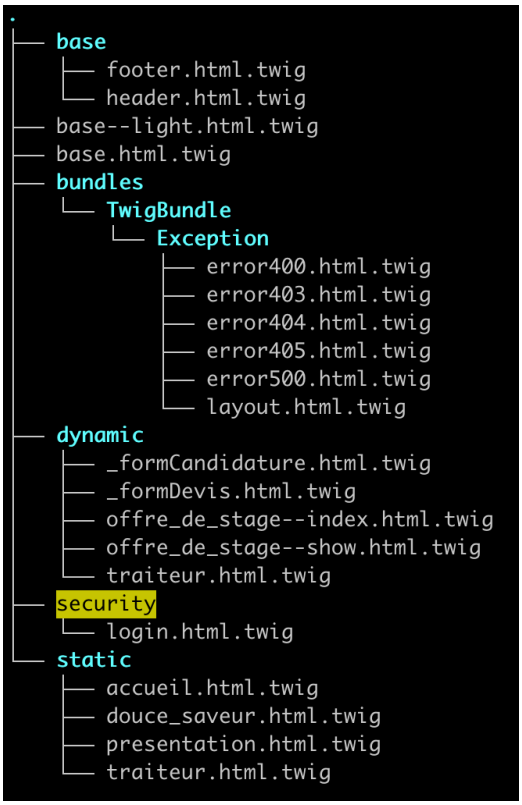

FIGURE 3 – Organisation des fichiers Twig

3.4 Gestion de la sécurité et des rôles utilisateurs

Pour le développement de ce projet nous avons mis en place un **back office** qui a nécessité l'ajout de sécurité de rôle à notre projet. Pour la partie sécurité nous avons utilisé les outils proposés par Symfony à savoir la commande `composer require security` et la commande `symfony console make:user`. Afin de restreindre l'accès à la partie d'administration de notre site nous avons ajouté cette ligne : `- { path: ^/admin, roles: ROLE_USER}` au fichier `security.yaml` dans la partie `access_control:`

3.5 Solution d'upload des fichiers

Au sein de notre projet, nous nous sommes attelés à la gestion de document pouvant être transmis lors d'une candidature de stage. Pour cela nous avons utilisé VichUploader en l'important dans notre projet via la commande `composer require vich/uploader-bundle` puis nous avons fait la configuration du répertoire à utiliser pour stocker les CV ainsi que l'ajout d'un champ spécifique dans le formulaire concerné.

3.6 Configuration des librairies communautaires

Voici un bref aperçu des librairies communautaires utilisées :
— Sass : préprocesseur css
— Webpack encore : pour toute la partie *front* du site (compilation des fichiers `.scss` et `.js`)
— CopieFile : permet de copier les ressources graphiques présentes dans le répertoire `assets` pour les placer dans le 'public/build/'
— EasyAdmin : qui gère toute la partie *back office* de manière sécurisée

3.7 Structure des fichiers de style

Pour gérer le style des pages, nous avons utilisé le langage SASS (.scss). C'est un préprocesseur CSS, pour améliorer et simplifier l'écriture des styles. Grâce à ses fonctionnalités comme les variables, l'imbrication, SASS nous a permis de créer des feuilles de style plus structurées, et réutilisables.

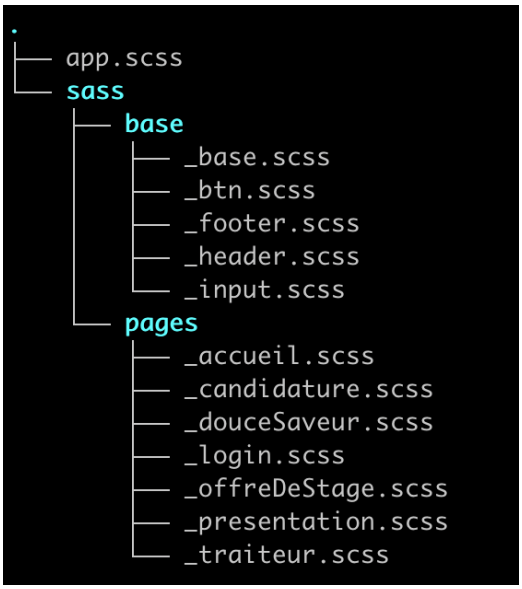

FIGURE 4 – Organisation des fichiers Scss

3.8 Structure des fichiers JS

Notre projet utilise très peu de JavaScipt, mais voici tout de même comment nous avons organisé nos quelques fichiers. Nous les avons placé dans un répertoire nommé "js" puis nous les avons importés dans le fichier `app.js`. Ce dernier est ensuite compilé dans le répertoire "/public/build/" par *webpack encore* via la commande `yarn run build`.

FIGURE 5 – Organisation des fichiers JavaScript