

FAQ

Symfony Recrutic



Introduction

Tâche à faire pendant ce stage :

- Migrer le site Recrutic sur Symfony
- Permettre la gestion de la base de données à partir du site
 - o Gestion des clients
 - o Gestion des annonces
 - o ...
-

Contenu

A.	SYMFONY	4
A.1	SON INSTALLATION	4
A.2	INTRODUCTION A SYMFONY	5
A.3	NAVIGATION ENTRE PAGES	6
A.4	CONNEXION A UNE BASE DE DONNEES	7
A.5	INSTALLATION DE COMPOSER	7
A.6	OUTILS UTILISES	8
B.	LES TACHES ADMINISTRATIVES.....	ERREUR ! SIGNET NON DEFINI.
B.1	EN QUELQUES MOTS, COMMENT PEUT-ON RESUMER LA COLLABORATION ?	ERREUR ! SIGNET NON DEFINI.
B.2	LA GESTION DES APPLES TELEPHONIQUES.....	ERREUR ! SIGNET NON DEFINI.
B.3	LA GESTION DU COURRIER ?	ERREUR ! SIGNET NON DEFINI.
B.4	QUELLES SONT MES OBLIGATIONS ?	ERREUR ! SIGNET NON DEFINI.
B.5	QUELLES SONT LES OBLIGATIONS DE BIZ-AND-COACH.....	ERREUR ! SIGNET NON DEFINI.
B.6	LES OBLIGATIONS MUTUELLES	ERREUR ! SIGNET NON DEFINI.
B.7	QUELS SONT LES PRODUITS COMMERCIALISES ?	ERREUR ! SIGNET NON DEFINI.

A. Symfony

A.1 Son installation

Avoir Wamp.

Vérifier que PHP est bien installé.

Dans l'invite de commande faire la commande « *php -v* ».

Si nous obtenons la version de php en réponse, php est bien installé.

Sinon, il faut aller dans les variables d'environnement et mettre le chemin de php dans path.

Panneau de configuration -> Système -> Paramètre système avancés -> Variable d'environnement -> Path -> Nouveau -> C:\wamp64\bin\php\php7.0.10

Télécharger symfony.phar.

Le mettre dans le fichier « *www* » de wamp.

Exécuter la commande « *php symfony.phar new symfony* ».

En cas d'erreur :

[GuzzleHttp\Exception\RequestException]

cURL error 60: SSL certificate problem: unable to get local issuer certificate

Télécharger le fichier « *cacert.pem* »

Changer le fichier php.ini dans le dossier php7.0.10.

curl.cainfo ="C:/ « *Chemin* » /cacert.pem"

Attention ! Ne pas oublier d'enlever le « ; » au début de la ligne.

Relancer wamp et « *php symfony.phar new symfony* ».

A.2 Introduction à Symfony

Création d'un Bundle -> `php bin/console generate:bundle`

Symfony se découpe en 3 parties :

- un fichier « route » contenant les redirections.
- Un fichier « controller » dans laquelle nous regrouperont nos fonctions.
- Un fichier « html.twig » contenant le html.

```

1  {# src/OC/PlatformBundle/Resources/views/layout.html.twig #}
2
3
4  <!DOCTYPE HTML>
5
6  <html>
7
8  <head>
9
10     <meta charset="utf-8">
11
12     <title>{% block title %}OC Plateforme{% endblock %}</title>
13     {% block stylesheets %}
14
15     {# On charge le CSS de bootstrap depuis le site directement #}
16
17     <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
18
19     {% endblock %}
20 </head>
21
22 <body>
23     <div id="menu">
24         {{ render(controller("OCPlatformBundle:Advert:menu")) }}
25     </div>
26
27
28     {% block body %}
29
30     {% endblock %}
31
32
33 </body>
34
35 </html>

```

Découpage du site avec un layout.

C'est la base de notre code. Les autres pages serviront à remplir cette page.

```

1  {% extends "OCPlatformBundle::layout.html.twig" %}
2
3  {% block title %}
4      Recrutic
5  {% endblock %}
6
7  {% block body %}
8      Page 3
9  {% endblock %}

```

Voici un exemple de remplissage d'une page.

A.3 Navigation entre pages

```
<div id="menu">
  <a href="{{ path('oc_platform_homepage') }}">Accueil</a>
  <a href="{{ path('oc_platform_offreEmploi') }}">Nos offre d'emploi</a>
  <a href="{{ path('oc_platform_deposerCV') }}">Déposer un CV</a>
</div>
```

Dans le layout, nous utiliserons la fonction « path » qui va permettre de naviguer entre les pages. Cette fonction prend en paramètre le nom de la route.

```
oc_platform_homepage:
  path:      /
  defaults: { _controller: OCPlatformBundle:Advert:index }

oc_platform_deposerCV:
  path:      /Deposer_un_cv/
  defaults: { _controller: OCPlatformBundle:Advert:DeposerCV }

oc_platform_offreEmploi:
  path:      /Nos_offres_d_emploi/
  defaults: { _controller: OCPlatformBundle:Advert:NosOffresDEmploi }
```

Nous devons aussi créer des fonctions pour ces routes.

```
public function indexAction()
{
    $content = $this
        ->get('templating')
        ->render('OCPlatformBundle:Advert:index.html.twig');

    return new Response($content);
}

public function DeposerCVAction()
{
    return $this->render('OCPlatformBundle:Advert:DeposerCV.html.twig');
}

public function NosOffresDEmploiAction()
{
    return $this->render('OCPlatformBundle:Advert:NosOffresDEmploi.html.twig');
}
```

A.4 Ajout des CSS et du JavaScript

```
{% block stylesheets %}
    <link href="{{ asset('css/style.css') }}" rel="stylesheet" />
    <link href="{{ asset('css/ceebox.css') }}" rel="stylesheet" />
    <link href="{{ asset('css/styles.css') }}" rel="stylesheet" />
    <link href="{{ asset('css/styles-rtl.css') }}" rel="stylesheet" />
{% endblock %}
```

Pour ajouter le CSS ou le JavaScript, il suffit d'assigner les liens dans le block leur correspondant. Nous mettrons ce bout de code dans le fichier « layout » pour qu'il soit affiché partout dans nos pages.

Il est possible de rassembler tous les liens CSS ou JavaScript dans une seule ligne avec le Bundle « Assetic ».

A.5 Connexion à une base de données

```
parameters:
    database_driver : pdo_mysql
    database_host: 192.168.0.51
    database_port: null
    database_name: maneom
    database_user: maneom
    database_password: 
    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: null
    mailer_password: null
    secret: df61e7482b49812a28777fa36c6c76f76eb112a1
```

Dans le cas, où nous aimerions nous connecter à une base de données, il suffit de changer les paramètres de connexion dans le fichier parameters.yml.

A.6 Installation de Composer

Composer ne fait pas partie de Symfony. Néanmoins, il est très utilisé par celui-ci, notamment dans le cas d'installation de Bundle.

Pour cela, il suffit de le récupérer avec commande

« `php -r "eval('?'>'.file_get_contents('http://getcomposer.org/installer'))";` »

Nous obtenons le fichier « composer.phar ».

Il faut ensuite lui demander de s'installer dans Symfony. Pour cela, la commande « `php composer.phar update` » est utilisée.

Attention, ce composant met tout à jours !

Cela peut comporter des erreurs notamment dans la version du Twig et celle de PHP. Si une erreur apparaît, il suffit d'aller dans « composer.json » et de changer le twig en "twig/twig": "^1.0|^1.3".

Une fois le composer mis en place, on peut installer des bundles.

A.7 Création d'une page de connexion utilisateurs / administrateurs

Pour faire une partie utilisateur et une partie administrateur, il faut que l'on puisse se loguer avec soit des identifiants d'utilisateurs soit d'administrateur. A partir de là, l'interface sera différentes selon notre statut.

A.7.1 Utilisation de Doctrine

Doctrine est un composant de Symfony qui va nous aider à gérer notre base de données.

Pour créer une base de données, (après avoir mis à jour le fichier « parameter.yml ») il suffit de faire la commande « `php bin/console doctrine:database:create` ». Cette touche nous permet de créer une base de données dans notre base.

Les données ici ne sont pas géré directement par l'administrateur. Doctrine ajoutera ou supprimera pour nous les données dans la base de données.

Pour cela, il faut créer des classes objets de nos données. Contrairement à Java, où nous devons faire cette classe avec des getters et des setters, Doctrine nous la génère. Il suffit de taper « `php bin/console doctrine:generate:entity` »

```
Welcome to the Doctrine2 entity generator

This command helps you generate Doctrine2 entities.

First, you need to give the entity name you want to generate.
You must use the shortcut notation like AcmeBlogBundle:Post.

The Entity shortcut name: User
The entity name isn't valid ("User" given, expecting something like AcmeBlogBundle:Blog/Post)
The Entity shortcut name: OCPlatformBundle:User
Entity "OCPlatformBundle:User" already exists.
The Entity shortcut name: OCPlatformBundle:Utilisateur

Determine the format to use for the mapping information.
Configuration format (yaml, xml, php, or annotation) [annotation]: annotation

Instead of starting with a blank entity, you can add some fields now.
Note that the primary key will be added automatically (named id).

Available types: array, simple_array, json_array, object,
boolean, integer, smallint, bigint, string, text, datetime, datetimetz,
date, time, decimal, float, binary, blob, guid.

New field name (press <return> to stop adding fields): Mail
Field type [string]:
```


Doctrine nous a généré automatiquement notre entité.

```
namespace OC\PlatformBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Utilisateur
 *
 * @ORM\Table(name="utilisateur")
 * @ORM\Entity(repositoryClass="OC\PlatformBundle\Repository\UtilisateurRepository")
 */
class Utilisateur
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="Mail", type="string", length=255, unique=true)
     */
    private $mail;

    /**
     * @var string
     *
     * @ORM\Column(name="MotDePasse", type="string", length=255)
     */
    private $motDePasse;

    /**
     * @var string
     *
     * @ORM\Column(name="Prenom", type="string", length=255)
     */
    private $prenom;

    /**
     * @var string
     *
     * @ORM\Column(name="Nom", type="string", length=255)
     */
}
```

Il faut ensuite créer cette entité dans la base de données. Cette entité correspondra à l'inscription des utilisateurs au site Recrutic.

Pour cela, nous ferons la commande « `php bin/console doctrine:schema:update --dump-sql` ». Celle-ci nous affichera les commandes nécessaires pour ajouter ou modifier nos entités à la base de données.

Cette requête n'a pas été exécutée. Pour l'exécuter, il suffit de faire « `php bin/console doctrine:schema:update --force` ».

Comme on peut le voir, la table a bien été ajoutée à RecruticSymfony.

recruticsymfony

utilisateur
Nouvelle table

```
SELECT *
FROM 'utilisateur'
LIMIT 0, 30
```

☐ Profilage
[\[En ligne\]](#)
[\[Modifier\]](#)

#	Colonne	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
<input type="checkbox"/>	1 id	int(11)			Non	Aucune	AUTO_INCREMENT	Modifier Supprimer plus ▼
<input type="checkbox"/>	2 Mail	varchar(255)	utf8_unicode_ci		Non	Aucune		Modifier Supprimer plus ▼
<input type="checkbox"/>	3 MotDePasse	varchar(255)	utf8_unicode_ci		Non	Aucune		Modifier Supprimer plus ▼
<input type="checkbox"/>	4 Prenom	varchar(255)	utf8_unicode_ci		Non	Aucune		Modifier Supprimer plus ▼
<input type="checkbox"/>	5 Nom	varchar(255)	utf8_unicode_ci		Non	Aucune		Modifier Supprimer plus ▼
<input type="checkbox"/>	6 Adresse	varchar(255)	utf8_unicode_ci		Non	Aucune		Modifier Supprimer plus ▼
<input type="checkbox"/>	7 Ville	varchar(255)	utf8_unicode_ci		Non	Aucune		Modifier Supprimer plus ▼
<input type="checkbox"/>	8 Pays	varchar(255)	utf8_unicode_ci		Non	Aucune		Modifier Supprimer plus ▼
<input type="checkbox"/>	9 CP	int(11)			Non	Aucune		Modifier Supprimer plus ▼

A.7.2 Création du formulaire

```
public function InscriptionAction(Request $request)
{
    $utilisateur = new Utilisateur();

    $form = $this->createForm(InscriptionType::class, $utilisateur);

    if ($request->isMethod('POST')) {
        if ($form->handleRequest($request)->isValid()) {
            $em = $this->getDoctrine()->getManager();
            $em->persist($utilisateur);
            $em->flush();

            $request->getSession()->getFlashBag()->add('notice', 'Inscription bien enregistrée.');
```

Dans le constructeur, il faut appeler notre entité et lui donner la forme que l'on souhaite.

Pour cela, nous avons créé un fichier InscriptionType.php.

```
class InscriptionType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('mail', EmailType::class)
            ->add('motDePasse', PasswordType::class)
            ->add('prenom', TextType::class)
            ->add('nom', TextType::class)
            ->add('adresse', TextType::class)
            ->add('ville', TextType::class)
            ->add('CP', IntegerType::class)
            ->add('pays', CountryType::class)
            ->add('telephone', IntegerType::class)
            ->add('confirmer', SubmitType::class)
        ;
    }
}
```

Dans celle-ci, nous ajoutant les attribues de notre entité que nous souhaitons faire afficher dans le formulaire. Nous précisions aussi l'état des champs (Texte, nombre,...)

A.8 Commandes utiles

php bin/console doctrine:generate:entity -> générer une entité entière
php bin/console doctrine:generate:entities « Nom du Bundle:Nom de l'entité »
-> générer ce qui manque (modification dans une entité)
php bin/console doctrine:schema:update --dump-sql -> générer une requête SQL
php bin/console doctrine:schema:update --force -> appliquer la requête à la BD

A.9 Outils utilisés

- Wamp
- PHP
- HTML/CSS
- GIT
- Symfony
- Netbeans