



APPLICATION WEB

---

**DOCUMENTATION TECHNIQUE**

Projet E4

PHP / Symfony

Cheik-Siramakan Keita

[cheiksiramakankeita@gmail.com](mailto:cheiksiramakankeita@gmail.com)



## Table des matières

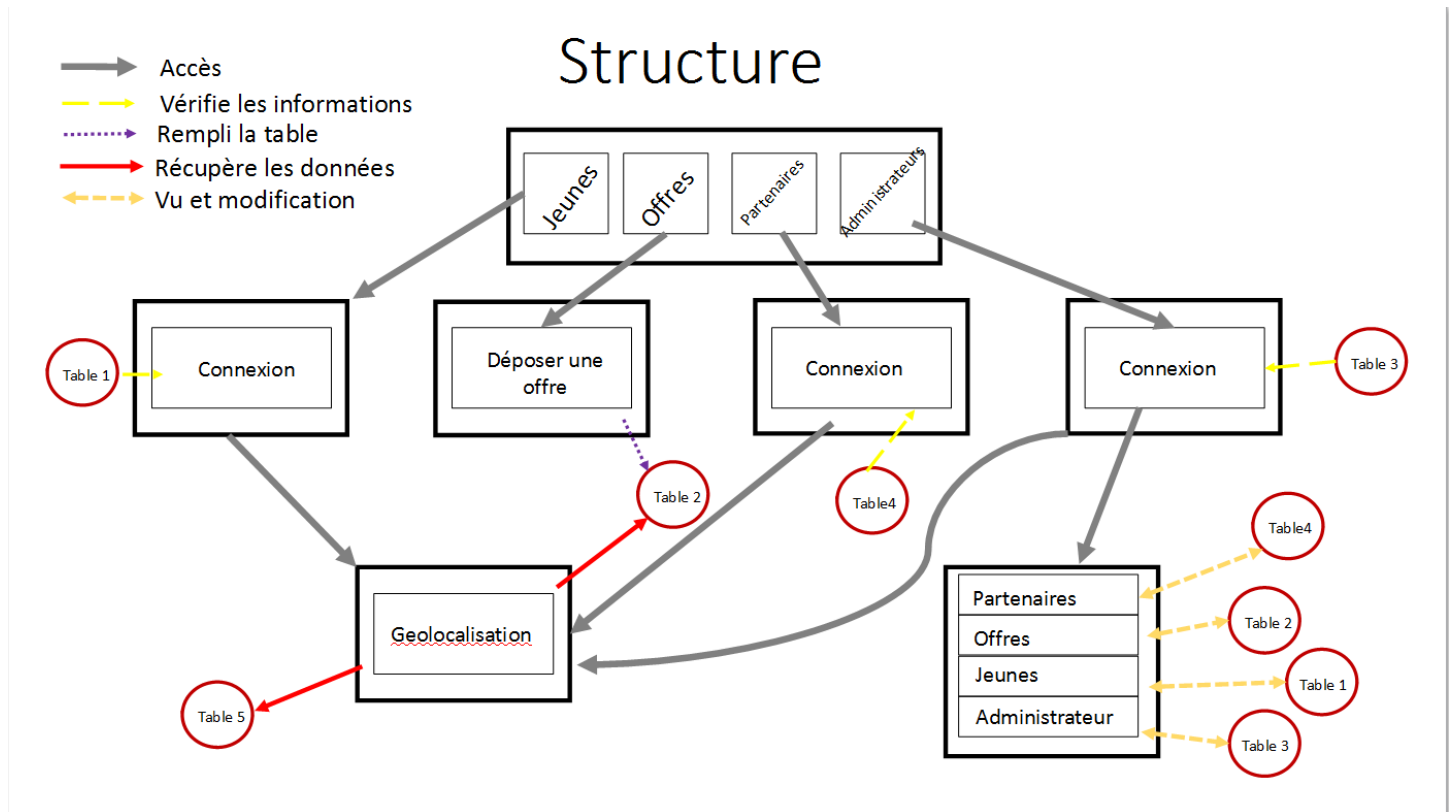
Cahier des charges .....	2
Contexte .....	2
Prestation attendu.....	2
Outils utilisé .....	3
Base de données .....	4
Liste des tables .....	4
MLD .....	4
Connexion à la base de données.....	5
Création de la base de données.....	5
Entités .....	5
Création des entités .....	5
Controllers.....	5
Templates .....	6
Formulaires .....	6
Créer un formulaire.....	6
Gérer les formulaires.....	6
Modification .....	8
Requêtes SQL .....	9
Ecrire une requête.....	9
Téléchargement.....	11
Gérer le téléchargement .....	11
Statistiques.....	12
Gérer les statistiques.....	12
Géolocalisation.....	14
Gérer la Géolocalisation.....	14

## CAHIER DES CHARGES

### CONTEXTE

Le Lycée du Parc de Vilegénis a fait appel à la société ZenMedia, une SSII afin de lui développer une application web de mise en relation pour son organisation. Cette application web devra permettre de mettre en relation des jeunes avec des entreprises qui déposeront des offres d'emploi.

### PRESTATION ATTENDU



Dans un premier temps l'utilisateur arrivera sur la page d'accueil, sur cette page l'utilisateur pourra accéder à 4 modules grâce à des boutons, module jeune, module déposer une offre, module partenaire, module administration. Il devra donc y avoir 4 boutons d'accès aux modules au centre de la page avec une image en fond (des étudiants).

#### Premier module : Jeune

C'est l'administrateur qui fera l'inscription des jeunes, les jeunes ne pourront pas s'inscrire mais seulement se connecter pour accéder à la partie géolocalisation. Sur cette page on leur demandera un login qui sera composé de la première lettre de leur prénom ainsi que leur nom et un mot de passe. Une fois connecté ils pourront accéder à la page de géolocalisation qui reprendra les offres des entreprises, ils pourront alors faire des recherches par ville ainsi que par départements avec une partie Google Map, et une description des offres. Aussi chaque étudiant connecté sur la partie géolocalisation devra voir affiché un message du style : bienvenue « nom » « prénom » ainsi qu'un bouton déconnexion.

#### Deuxième module : Déposer une offre

On demandera sur cette page la raison sociale, adresse/ville/code-postal du lieu de l'offre, type de formation (3 choix : réseau/développement/dépannage informatique), descriptif de l'offre ou ils pourront parler de l'offre, donner leur tel et adresse mail ainsi que la durée de la formation (tout ça dans le descriptif de l'offre), et pour finir que la date du jour soit automatiquement marquée. Une fois l'offre enregistrée, elle sera indiquée sur la page géolocalisation.

#### Troisième module : Partenaire

C'est encore une fois l'administrateur qui fera l'inscription des partenaires, une fois connectés les partenaires pourrons accéder à la partie géolocalisation. La connexion du partenaire se fera par son numéro de SIRET et son mot de passe.

#### Quatrième module : Administrateur

Pour accéder à l'administration, il faudra déjà qu'il se connecte, la connexion de l'administrateur se fera par un nom d'utilisateur et un mot de passe. Une fois connecté l'administrateur pourra créer des comptes jeunes/partenaire/administrateur, gérer les offres(supprimer) ainsi que supprimer les comptes jeune/partenaire ou alors accéder à la partie géolocalisation.

---

### OUTILS UTILISE

Pour ce projet j'ai utilisé Sublime Text comme éditeur de texte, MySQL pour la base de données, Symfony comme Framework pour développer l'application et Git comme gestionnaire de version pour l'application.

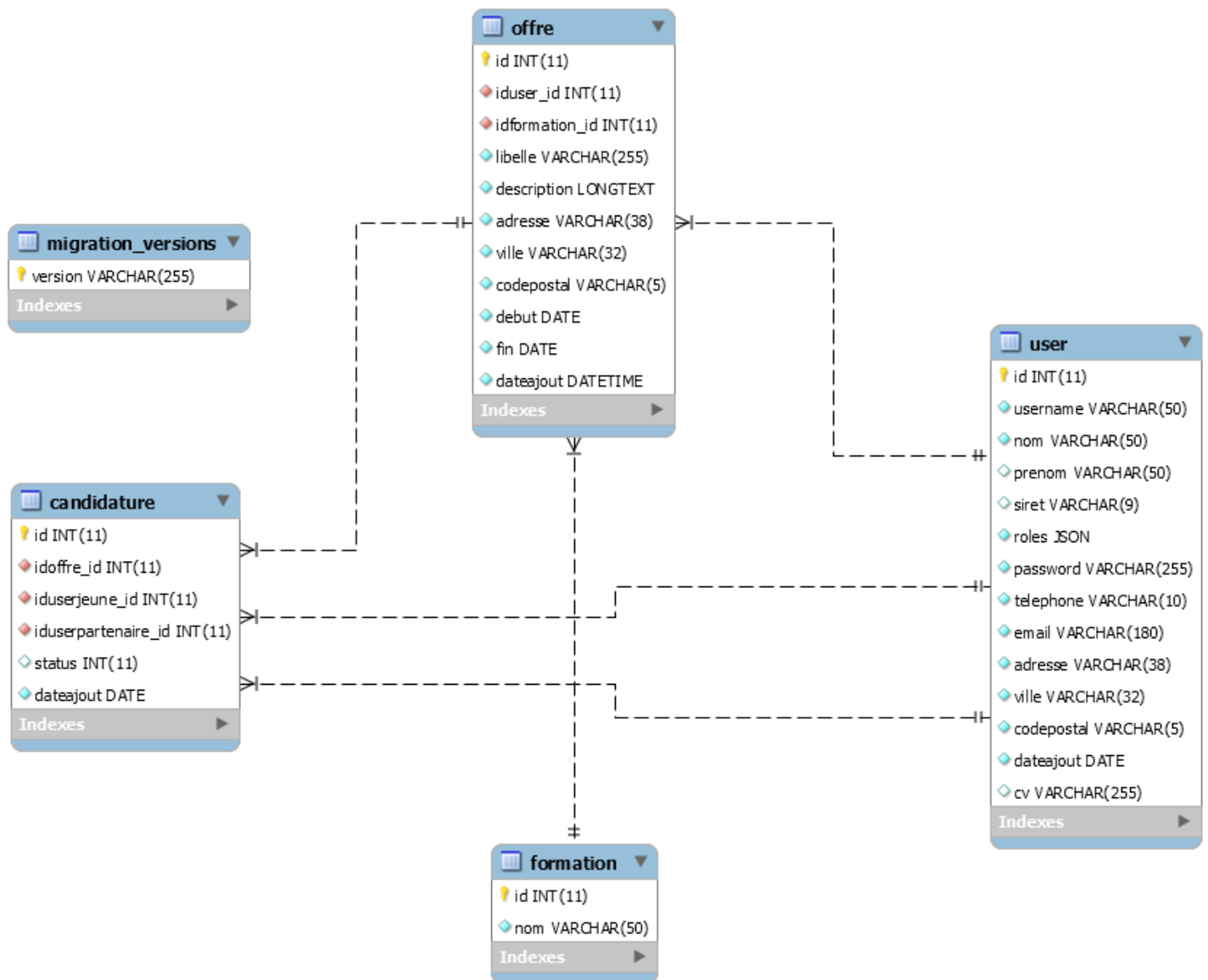


## BASE DE DONNEES

### LISTE DES TABLES

Table	Action	Rows	Type	Collation	Size	Overhead
candidature		5	InnoDB	utf8mb4_unicode_ci	64 KiB	-
formation		3	InnoDB	utf8mb4_unicode_ci	16 KiB	-
migration_versions		0	InnoDB	utf8_unicode_ci	16 KiB	-
offre		75	InnoDB	utf8mb4_unicode_ci	96 KiB	-
user		85	InnoDB	utf8mb4_unicode_ci	64 KiB	-
5 table(s)	Sum	168	InnoDB	latin1_swedish_ci	256 KiB	0 B

### MLD



---

## CONNEXION A LA BASE DE DONNEES

La connexion à la base de données se fait à l'aide du fichier `.env` qui se trouve à la base du dossier. La structure de cette ligne est comme ceci :

`DATABASE_URL=mysql://[nom de l'utilisateur]:[mot de passe]@[nom d'hôte]:[port]/[nom de la base de données]`

```
DATABASE_URL=mysql://ppe:rk-mSx4f$+w*kVFL@192.168.1.15:3306/ppe
```

## CREATION DE LA BASE DE DONNEES

La base de données est ensuite créée via cette commande :

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

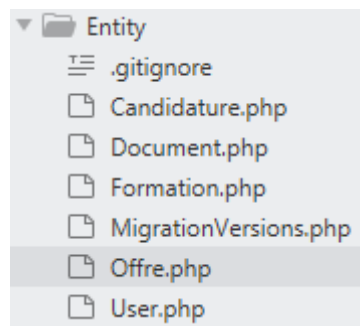
Pour générer les tables dans la base de données :

```
php bin/console doctrine:schema:update --force
```

---

## ENTITES

Les entités sont les classes métiers utilisé par l'application, ces fichiers contiennent aussi des annotations Doctrine pour lier les données aux champs de la base de données. La création des entités est nécessaire pour la création des tables et des champs de la base de données.



## CREATION DES ENTITES

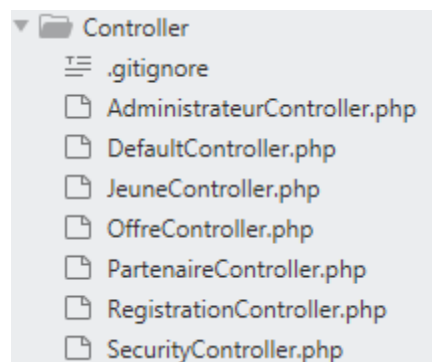
Pour créer les entités il faut utiliser la commande suivante dans la console :

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

---

## CONTROLLERS

Les controllers sont les classes qui gère les templates et les informations qui seront transmises.



Pour créer des controllers il faut utiliser la commande suivante :

```
php bin/console make:controller
```

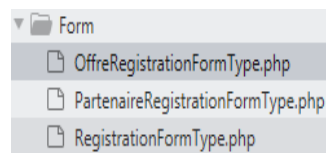
## TEMPLATES

Les templates se trouvent dans le répertoire templates à la base du projet, dedans les fichiers sont séparés selon le controller et sont appelés par ces derniers. Les templates sont des fichiers contenant du code HTML et Twig car c'est le moteur de template utilisé par Symfony, Twig est très utile pour faire passer des données dans les templates.



## FORMULAIRES

Les formulaires sont les l'une des choses les plus importante du projet car la majorité des insertions dans la base de données s'effectuent via celles-ci. Dans Symfony il est possible de séparer le code de création de formulaire afin de ne pas gonfler le code dans les controllers donc dans src/Forms se trouvent le code qui génère les formulaires dans les templates. Ce code est appelé dans le controllers lorsqu'il est nécessaire de générer un formulaire.



## CREER UN FORMULAIRE

Dans la console il faut entrer cette commande :

```
php bin/console make:form
```

## GERER LES FORMULAIRES

Dans le fichier « src/Form/OffreRegistrationFormType.php » se trouve le code de création d'un formulaire pour gérer les offres chaque ligne contenant « ->add » est un nouveau champ de ce formulaire. Il est possible d'indiquer de quel type de champ il s'agit, quel est son nom ou encore ses attributs.

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('libelle', TextType::class, array('attr' => array('placeholder' => 'Libelle')))
        ->add('adresse', TextType::class, array('attr' => array('placeholder' => 'Adresse')))
        ->add('ville', TextType::class, array('attr' => array('placeholder' => 'Ville')))
        ->add('codepostal', TextType::class, array('attr' => array('placeholder' => 'Code Postal')))
        ->add('debut', DateType::class)
        ->add('fin', DateType::class)
        ->add('description', TextareaType::class, array('attr' => array('placeholder' => 'Description du poste...')))
        ->add('ajout', SubmitType::class, array('label' => 'Déposer l\'offre'))
    ;
}
```

Dans le fichier « src/Controller/OffreController.php » se trouve la fonction de la capture d'écran en dessous qui montre comment l'ajout d'un formulaire se passe dans un controller, ce dernier fait appel à la classe dont elle a besoin qui se trouve dans src/Form. Pour cela il faut faire appel à ce fichier de cette façon car ce projet utilise des « namespace ».

```
use App\Form\OffreRegistrationFormType;
```

C'est dans une des fonctions d'un controller qu'il faudra faire appel au code générateur de formulaire comme le montre la capture d'écran en-dessous.

```
$form = $this->createForm(OffreRegistrationFormType::class, $offre);  
$form->handleRequest($request);
```

Dans cette fonction lorsque l'on fait appel à la template qui contient le formulaire nous faisons passer ce qui a été généré et le faisons passer dans la template via « 'registrationForm' => \$form->createView() ».

```
return $this->render('offre/ajout.html.twig', [  
    'formations' => $formations,  
    'registrationForm' => $form->createView()  
]);
```

Dans la template qui se trouve dans « templates/offre/ajout.html.twig » nous mettons la syntaxe nécessaire pour faire appel au formulaire obtenu depuis le controller. « {{ form\_start(registrationForm) }} » signale où le formulaire devra être généré dans le code HTML quant à « {{ form\_end(registrationForm) }} » il signale où le formulaire fini. Les « {{ form\_widget(registrationForm.[nom du champ], {'attr': {'class': 'form-control'}}) }} » signalent où les champs seront positionnés dans le formulaire.

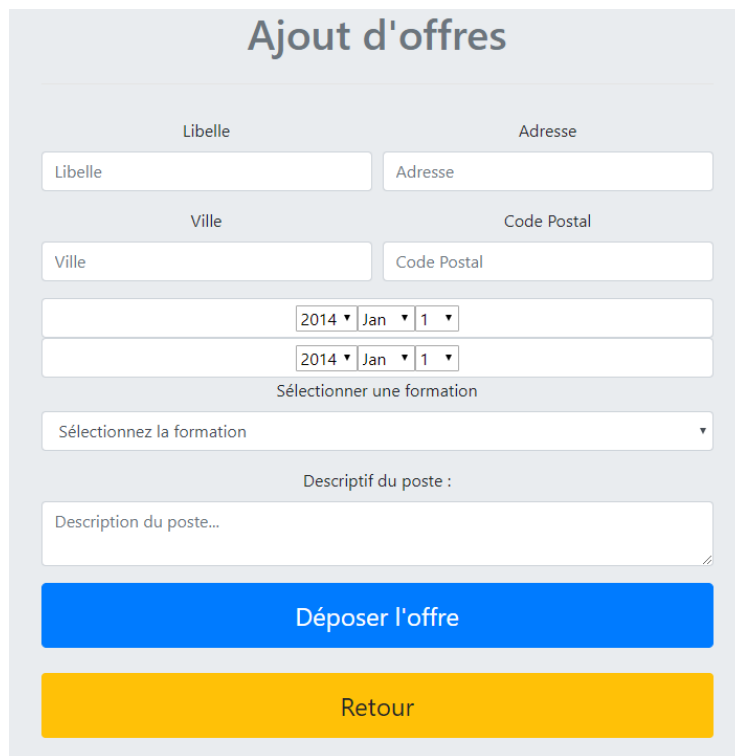
```
{{ form_start(registrationForm) }}  
    <div class="form-row">  
        <div class="form-group col-md-6">  
            <label>Libelle</label>  
            {{ form_widget(registrationForm.libelle, {'attr': {'class': 'form-control'}}) }}  
        </div>  
  
        {{ form_widget(registrationForm.ajout, {'attr': {'class': 'btn btn-primary btn-lg btn-block'}}) }}  
        <br><a href="{{ path('partenaire_gestion_des_offres') }}" class="btn btn-warning btn-block">Retour</a>  
    {{ form_end(registrationForm) }}
```

Pour insérer ces informations dans la base de données une fois le formulaire rempli et envoyé, dans la fonction qui fait appel à la template en question nous vérifions les données soumises puis insérons les données en utilisant « \$entityManager->persist([\$offre dans cette situation]) ».

```
if ($form->isSubmitted() && $form->isValid())  
{  
    $aujourd'hui = date("Y/m/d");  
    $debut = $form->get('debut')->getData();  
    $fin = $form->get('fin')->getData();  
  
    if($debut > $aujourd'hui){  
        if($debut < $fin){  
            $formation = $repository->find($request->request->get('formation'));  
            $offre->setLibelle($form->get('libelle')->getData());  
            $offre->setAdresse($form->get('adresse')->getData());  
            $offre->setVille($form->get('ville')->getData());  
            $offre->setCodepostal($form->get('codepostal')->getData());  
            $offre->setDebut($debut);  
            $offre->setFin($fin);  
            $offre->setIdformation($formation);  
            $offre->setDescription($form->get('description')->getData());  
            $offre->setIduser($user);  
  
            $entityManager = $this->getDoctrine()->getManager();  
            $entityManager->persist($offre);  
            $entityManager->flush();  
  
            return $this->redirectToRoute('partenaire_gestion_des_offres');  
        }else{  
            return $this->redirectToRoute('partenaire_gestion_des_offres');  
        }  
    }else{  
        return $this->redirectToRoute('offre_ajout');  
    }  
}
```



Le résultat de ces manipulations donne ceci.



## MODIFICATION

Modifier les informations d'une entité se fait via un formulaire donc la façon dont le formulaire est généré est la même que celle vu dans la section « Formulaire » mis à part que dans le contrôleur certaines choses changent. A la place de générer le formulaire avec une instance d'une entité vide nous donnons à la fonction pour second paramètre une instance contenant des informations. Dans la situation de la capture d'écran nous récupérons une offre depuis la base de données en utilisant son id pour la retrouver via la fonction « find() ». « getRepository(Offre :class) » permet de spécifier de quelle entité il s'agit grâce à cela les informations récupérées seront placées dans une instance toute seule.

```
$entityManager = $this->getDoctrine()->getManager();
$offre = $entityManager->getRepository(Offre::class)->find($offreId);

$repository = $this->getDoctrine()->getRepository(Formation::class);
$formations = $repository->findAll();

$debut=$offre->getDebut()->format('Y-m-d');
$fin=$offre->getFin()->format('Y-m-d');

$form = $this->createForm(OffreRegistrationFormType::class, $offre);
$form->handleRequest($request);
```

Une fois les données récupérées cette fois nous n'utilisons pas la fonction persist comme pour insérer de nouvelles données. Doctrine est capable de savoir si cette information existe déjà dans la base de données et de la mettre à jour.

```
if ($form->isSubmitted() && $form->isValid())
{
    $offre->setIdformation($repository->find($request->request->get('formation')));
    $entityManager->flush();

    if($user->getRoles()[0] == "ROLE_PARTENAIRE"){
        return $this->redirectToRoute('partenaire_gestion_des_offres');
    }elseif($user->getRoles()[0] == "ROLE_ADMINISTRATEUR"){
        return $this->redirectToRoute('administration_gestion_des_offres');
    }
}
```

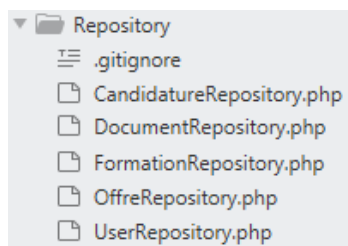
Le résultat de ces manipulations donne ceci.

## Modifiez les informations de cette offre

Libelle	Adresse
<input type="text" value="Test01"/>	<input type="text" value="57 Boulevard de l'Yerres"/>
Ville	Code Postal
<input type="text" value="Evry"/>	<input type="text" value="91000"/>
<input type="text" value="2014"/> <input type="text" value="Jan"/> <input type="text" value="1"/>	
<input type="text" value="2019"/> <input type="text" value="Apr"/> <input type="text" value="1"/>	
<input type="text" value="Dépannage informatique"/>	
Descriptif du poste :	
<input type="text" value="Hello world!"/>	
<input type="button" value="Modifier"/>	
<input type="button" value="Retour"/>	

## REQUETES SQL

Pour effectuer des requêtes SQL un peu plus complexe que ce que Doctrine offre par défaut il est possible d'en écrire et d'y faire appel dans les controllers. Ces requêtes sont dans les fonctions des classes dans les fichiers qui se trouvent dans le répertoire « src/Repository ».



## ECRIRE UNE REQUETE

Dans le répertoire « src/Repository » se trouve les fichiers où seront écrites les requêtes, chaque fichier est lié à une entité. La requête qui se trouve dans la capture d'écran en dessous permet de récupérer les informations des offres et du partenaire en utilisant l'id du partenaire dans la clause pour ne récupérer que ses offres.

```
public function findByPartenaireId($id)
{
    $conn = $this->getEntityManager()->getConnection();

    $sql = 'SELECT offre.id AS \'id\' , offre.libelle AS \'libelle\' , user.id AS \'idpartenaire\' , user.nom AS \'nomU\' , formation.nom AS
    \'nomF\' , offre.description AS \'description\' , offre.adresse AS \'adresse\' , offre.ville AS \'ville\' , offre.codepostal AS \'
    codepostal\' , offre.debut AS \'debut\' , offre.fin AS \'fin\' , offre.dateajout AS \'dateajout\'
    FROM offre
    JOIN user ON user.id = offre.iduser_id
    JOIN formation ON formation.id = offre.idformation_id
    WHERE offre.iduser_id = :id';
    $stmt = $conn->prepare($sql);
    $stmt->execute(['id' => $id]);

    return $stmt->fetchAll();
}
```

Dans le controller il faut simplement faire appel à la fonction de la classe Repository de l'entité dont vous avez besoin, cette fonction retournera des instances de l'entité lié à la classe Repository.

```
$repository = $this->getDoctrine()->getRepository(Offre::class);  
$offres = $repository->findByPartenaireId($user->getId());
```

Tout ce qui reste à faire dans le controller est de faire passer les données récupérées par la requête vers la template via la fonction « render » comme ceci « offres' => \$offres ».

```
return $this->render('partenaire/gestion_des_offres.html.twig', [  
    'offres' => $offres,  
    'controller_name' => 'AdministrateurController',  
]);
```

Dans la template en utilisant la syntaxe Twig nous utilisons une boucle For pour traverser le tableau offres et pour chaque élément je place les informations dont j'ai besoin dans le tableau HTML.

```
{% for offre in offres %}  
    <tr>  
        <td>{{ offre.id }}</td>  
        <td>{{ offre.nomU }}</td>  
        <td>{{ offre.nomF }}</td>  
        <td>{{ offre.libelle }}</td>  
        <td>{{ offre.debut|date('Y-m-d') }}</td>  
        <td>{{ offre.fin|date('Y-m-d') }}</td>  
        <td>{{ offre.dateajout|date('Y-m-d') }}</td>  
        <td>  
            <form method="POST">  
                <input type="text" hidden="hidden" name="id" value="{{ offre.id }}">  
                <input type="submit" name="modifier" class="btn btn-secondary btn-sm" value="Modifier">  
            </form>  
        </td>  
        <td>  
            <form method="POST">  
                <input type="text" hidden="hidden" name="id" value="{{ offre.id }}">  
                <input type="submit" name="supprimer" class="btn btn-danger btn-sm" value="Supprimer">  
            </form>  
        </td>  
    </tr>  
{% endfor %}
```

Le résultat de ces manipulations donne ceci.

[Ajout](#) [Retour](#)

Show 10 entries Search:

ID ▲	Entreprise	Formation	Nom	Debut	Fin	Date d'ajout	Modifier	Supprimer
75	PropCorporation	Dépannage informatique	Test01	2014-01-01	2019-04-01	2019-04-17	<a href="#">Modifier</a>	<a href="#">Supprimer</a>

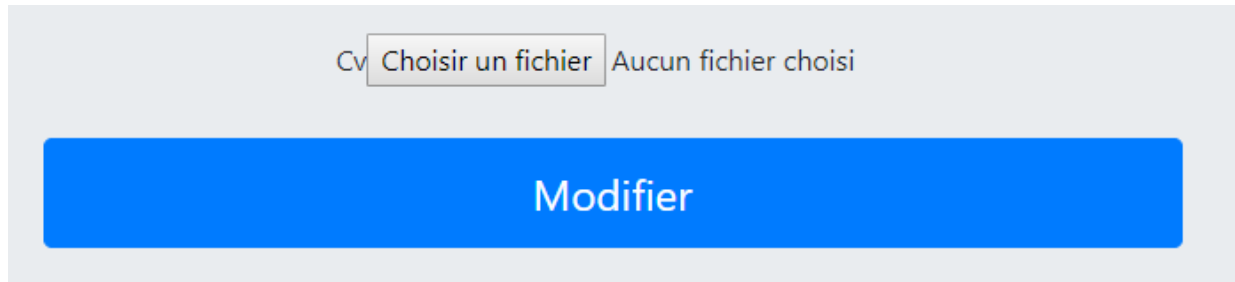
Showing 1 to 1 of 1 entries Previous 1 Next

## TELECHARGEMENT

Le téléchargement de fichiers est utilisé pour les jeunes car ils le mettront en ligne et les partenaires pourront les télécharger et pour les lire.

### GERER LE TELECHARGEMENT

Pour permettre le téléchargement de fichier par exemple lors de la mise en ligne d'un curriculum vitae lors de la modification de ses informations par un jeune :



Dans le contrôleur qui gère cette template il faut tout d'abord permettre de faire appel au code nécessaire pour le téléchargement en ajoutant les « use » en haut du fichier contenant la classe contrôleur.

```
use Symfony\Component\HttpFoundation\File\MimeType\ExtensionGuesser;
use Symfony\Component\HttpFoundation\File\File;
```

Dans la fonction qui gère la page où s'effectuera le téléchargement je fais appel à la fonction qui génère le formulaire, je permets le téléchargement de fichier avec « ['allow\_file\_upload' => true] ».

```
$form = $this->createForm(RegistrationFormType::class, $jeune, ['allow_file_upload' => true]);
$form->handleRequest($request);
```

Dans la classe « RegistrationFormType » où se trouve la fonction qui génère le formulaire qui se trouve dans le fichier « src/Form/RegistrationFormType.php » j'ai ajouté une condition pour ajouter un champ de téléchargement.

```
if($options["allow_file_upload"]){
    $builder->add('cv', FileType::class, array('data_class' => null), ['label' => 'CV']);
}
```

Dans le fichier « services.yaml » qui se trouve dans le répertoire « config » il faut spécifier le chemin vers le répertoire où seront stockés les fichiers téléchargés.

```
parameters:
  locale: 'en'
parameters:
  cv_directory: '%kernel.project_dir%/public/uploads/cv/'
```

Une fois le formulaire envoyé nous récupérons le fichier puis nous le plaçons dans le répertoire que nous avons spécifié dans « config/services.yaml ». Ensuite nous enregistrons le nom du fichier dans la base de données.

```
$file = $form->get('cv')->getData();
$fileName = $this->generateUniqueFileName().'.'.$file->guessExtension();

try {
    $file->move($this->getParameter('cv_directory'), $fileName);
} catch (FileNotFoundException $e) {
    // ... handle exception if something happens during file upload
    return $this->redirectToRoute('jeune_modification');
}
```

Lorsque l'on veut récupérer un fichier comme pour un partenaire voulant télécharger un curriculum vitae il faut faire appel à la fonction « file ».

```
return $this->file($this->getParameter('cv_directory') . $fileName);
```

Grâce à ce code il est possible de récupérer le fichier, voici ce que ça donne.

## Gestion des candidatures

[Retour](#)

Show  entries Search:

ID ▲	Offre ▲	Jeune ▲	C.V	Début ▲	Fin ▲	Status ▲	Date d'ajout ▲	Accepter ▲	Refuser ▲
6	Test01	Samy Ghislain	<a href="#">Download</a>	2014-01-01	2019-04-01	En attente	2019-04-19	<a href="#">Accepter</a>	<a href="#">Refuser</a>

Showing 1 to 1 of 1 entries Previous 1 Next

Copyright © Cheik-Siramakan Keita 2018-2019

## STATISTIQUES

Les statistiques affichés sont le pourcentage de formations par rapport aux offres le pourcentage des offres offert par les entreprises par rapport à l'ensemble des offres.

## GERER LES STATISTIQUES

Dans la classe « OffreRepository » se trouvent les fonctions que j'utilise pour effectuer les requêtes dont j'ai besoin pour récupérer les statistiques sur les offres. La classe se trouve dans le fichier « src/Repository/OffreRepository.php ».

```
public function countFormation()
{
    $conn = $this->getEntityManager()->getConnection();

    $sql = 'SELECT formation.nom, COUNT(*) AS \'nombre\' FROM offre JOIN formation ON offre.idformation_id = formation.id GROUP BY formation.nom';
    $stmt = $conn->prepare($sql);
    $stmt->execute();

    return $stmt->fetchAll();
}

public function countPartenaire()
{
    $conn = $this->getEntityManager()->getConnection();

    $sql = 'SELECT user.nom, COUNT(*) AS \'nombre\' FROM offre JOIN user ON offre.iduser_id = user.id WHERE user.roles LIKE "%ROLE_PARTENAIRE%" GROUP BY user.nom';
    $stmt = $conn->prepare($sql);
    $stmt->execute();

    return $stmt->fetchAll();
}
```

Dans le contrôleur « OffreController », dans la fonction qui s'occupe de gérer la page des statistiques je fais appel aux requêtes SQL. Une fois que j'ai récupéré les données de ces requêtes je les passe dans la template où je dois utiliser ces informations pour générer des camemberts.

```
$entityManager = $this->getDoctrine()->getManager();
$repository = $this->getDoctrine()->getRepository(Offre::class);
$stats1 = $repository->countFormation();
$stats2 = $repository->countPartenaire();
return $this->render('administrateur/statistiques_des_offres.html.twig', [
    'stats1' => $stats1,
    'stats2' => $stats2,
    'controller_name' => 'StatistiquesController',
]);
```

Dans la template j'utilise Google Chart pour générer des camemberts à partir des données que j'ai passé via le contrôleur. Avec la syntaxe Twig j'utilise une boucle For pour traverser les tableaux « stats » et insérer les données dans le code Javascript pour chaque élément.

```
google.charts.load('current', {'packages':['corechart']});
google.charts.setOnLoadCallback(drawChart);

function drawChart() {
    var data = google.visualization.arrayToDataTable([
        ['Formation', 'Nombre d\'offre'],
        {% for stat in stats1 %}
        [{ stat.nom }, {{ stat.nombre }}],
        {% endfor %}
    ]);

    var options = {
        title: 'Statistique des formations les plus offertes'
    };

    var data2 = google.visualization.arrayToDataTable([
        ['Partenaire', 'Nombre d\'offre'],
        {% for stat in stats2 %}
        [{ stat.nom }, {{ stat.nombre }}],
        {% endfor %}
    ]);

    var options2 = {
        title: 'Statistique des partenaires offrant le plus d\'offre'
    };

    var chart = new google.visualization.PieChart(document.getElementById('piechart'));
    var chart2 = new google.visualization.PieChart(document.getElementById('piechart2'));
    chart.draw(data, options);
    chart2.draw(data2, options2);
}
```

Plus bas dans la template je place les camemberts avec des « div ».

```
<div id="piechart" style="width: 640px; height: 500px;"></div><br>
<div id="piechart2" style="width: 640px; height: 500px;"></div>
```

## GEOLOCALISATION

La géolocalisation est utilisée pour afficher où se passe la formation, l'adresse à laquelle se situe la formation est facilement visible grâce à un marqueur.

### GERER LA GEOLOCALISATION

Dans le controller « OffreController », dans la fonction qui s'occupe de gérer la page de détail d'une offre qui a été sélectionné et dont l'id est enregistré dans la session je récupère les informations de cette offre avec la fonction « find() ».

```
$offreId = $this->container->get('session')->get('offreId');  
$offre = $this->getDoctrine()->getRepository(Offre::class)->find($offreId);
```

Une fois les informations récupérées je les passe dans la template avec la fonction « render() ».

```
return $this->render('offre/detail.html.twig', [  
    'offre' => $offre,  
    'partenaire' => $partenaire,  
    'formation' => $formation,  
]);
```

Dans la template « templates/offre/detail.html.twig » j'utilise Google Maps pour la carte et Geocode pour placer un marqueur sur la carte en utilisant l'adresse de l'offre. La variable « address » contient l'adresse que je passe à la template avec le controller dans « {{ offre.adresse }}».

```
var address = "{{ offre.adresse }}";  
var geocoder;  
var map;  
function initMap() {  
    map = new google.maps.Map(document.getElementById('map'), {  
        center: {lat: 48.864716, lng: 2.349014},  
        zoom: 12  
    });  
    geocoder = new google.maps.Geocoder();  
    codeAddress(geocoder, map);  
}  
  
function codeAddress(geocoder, map) {  
    geocoder.geocode({'address': address}, function(results, status) {  
        if (status === 'OK') {  
            map.setCenter(results[0].geometry.location);  
            var marker = new google.maps.Marker({  
                map: map,  
                position: results[0].geometry.location  
            });  
        }else{  
            alert('Geocode was not successful for the following reason: ' + status);  
        }  
    });  
}
```

J'affiche la carte en utilisant « div ».

```
<div id="map"></div>
```



Le résultat de ces manipulations donne ceci.

