Cheik-Siramakan Keita

cheiksiramakankeita@gmail.com

iramakan Keita

cheiksiramakankeita@gmail.com

iramakan Keita

cheiksiramakankeita@gmail.com

iramakan Keita

cheiksiramakankeita@gmail.com

Application web

## Documentation technique

Projet E4

PHP / Symfony

Application web

## Documentation technique

Projet E4

Application Java / Java FX

Application web

## Documentation technique

Projet E4

Application Java / Java FX

Application web

## Documentation technique

Projet E4

Application Java / Java FX

Table des matières

[Cahier des charges 2](#_Toc6841940)

[Contexte 2](#_Toc6841941)

[Prestation attendu 2](#_Toc6841942)

[Outils utilisé 3](#_Toc6841943)

[Base de données 4](#_Toc6841944)

[Liste des tables 4](#_Toc6841945)

[MLD 4](#_Toc6841946)

[Connexion à la base de données 5](#_Toc6841947)

[Création de la base de données 5](#_Toc6841948)

[Entités 5](#_Toc6841949)

[Création des entités 5](#_Toc6841950)

[Controllers 5](#_Toc6841951)

[Templates 6](#_Toc6841952)

[Formulaires 6](#_Toc6841953)

[Créer un formulaire 6](#_Toc6841954)

[Gérer les formulaires 6](#_Toc6841955)

[Modification 8](#_Toc6841956)

[Requêtes SQL 9](#_Toc6841957)

[Ecrire une requête 9](#_Toc6841958)

[Téléchargement 11](#_Toc6841959)

[Gérer le téléchargement 11](#_Toc6841960)

[Statistiques 12](#_Toc6841961)

[Gérer les statistiques 12](#_Toc6841962)

[Géolocalisation 14](#_Toc6841963)

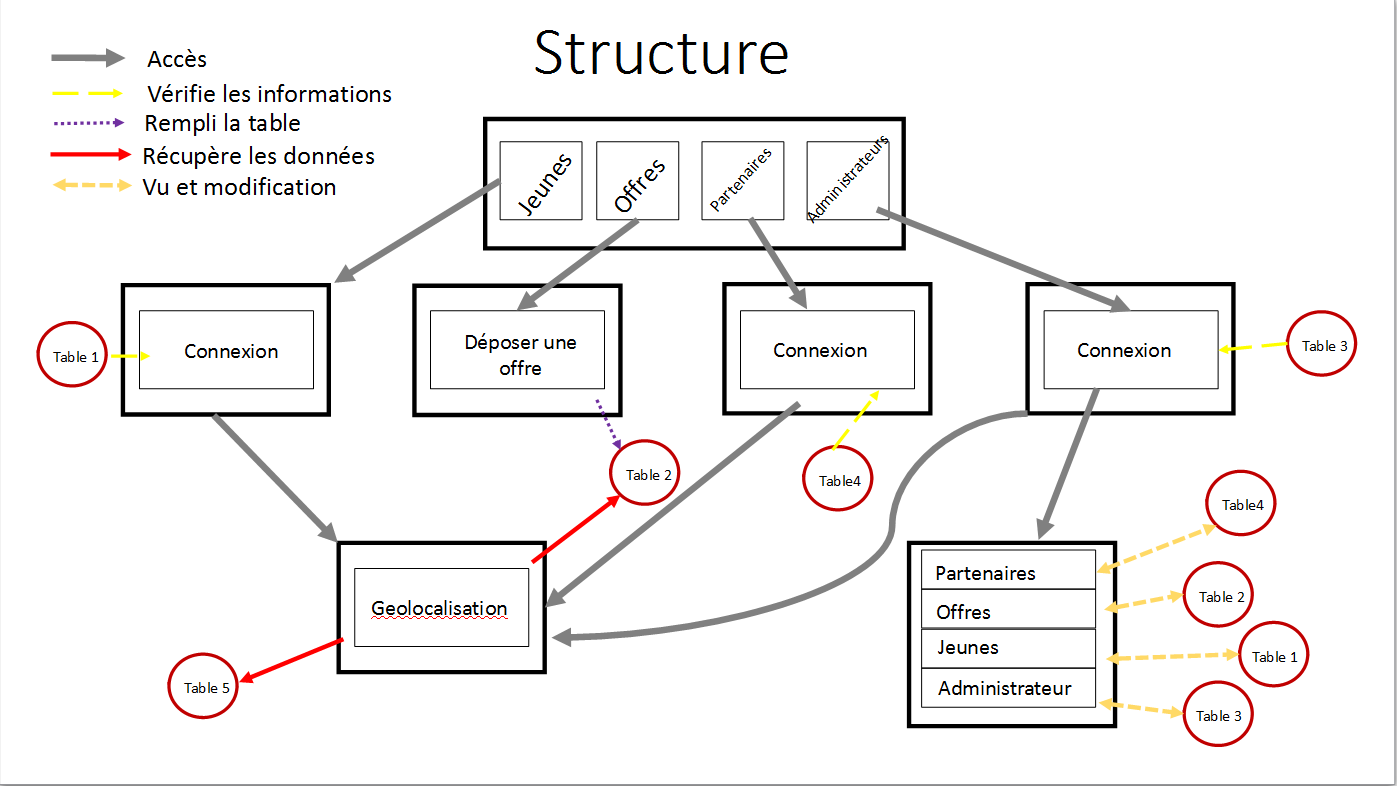
[Gérer la Géolocalisation 14](#_Toc6841964)

## 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

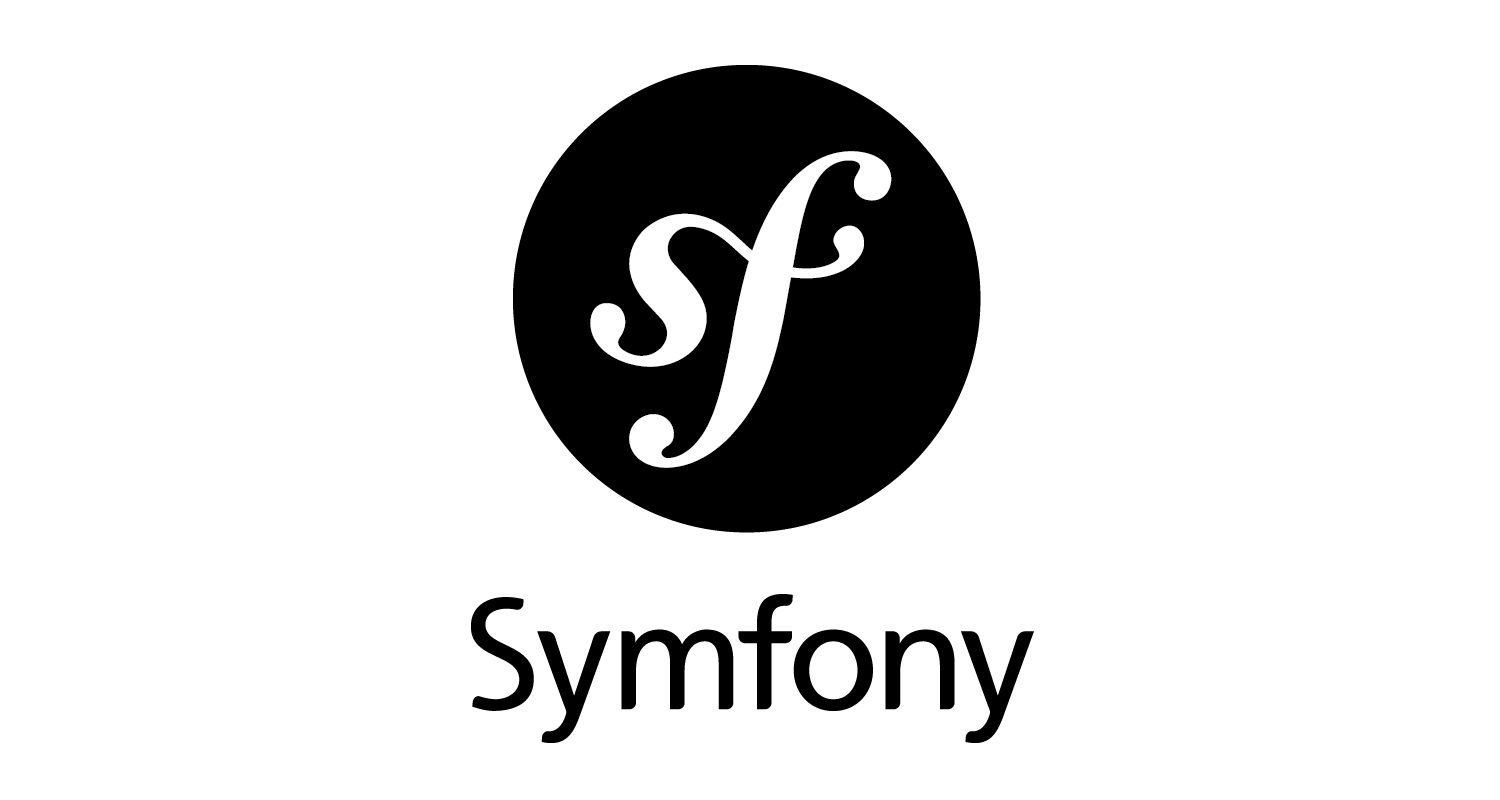
Pour accéder à l'administration, il faudra déjà qu'il se connecte, la connexion de l'administrateur se fera par un nom d’utilisateur et un mot de passe. Une fois connecté l'administrateur pourra créer des comptes jeunes/partenaire/administrateur, gérer les offres(supprimer) ainsi que supprimer les comptes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

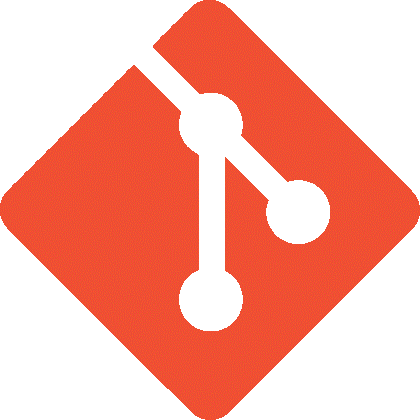
## Outils utilisé

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.

[](https://www.sublimetext.com/)

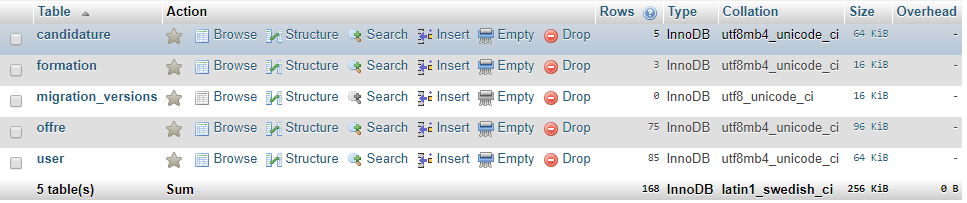
[](https://www.mysql.com/)

[](https://symfony.com/)

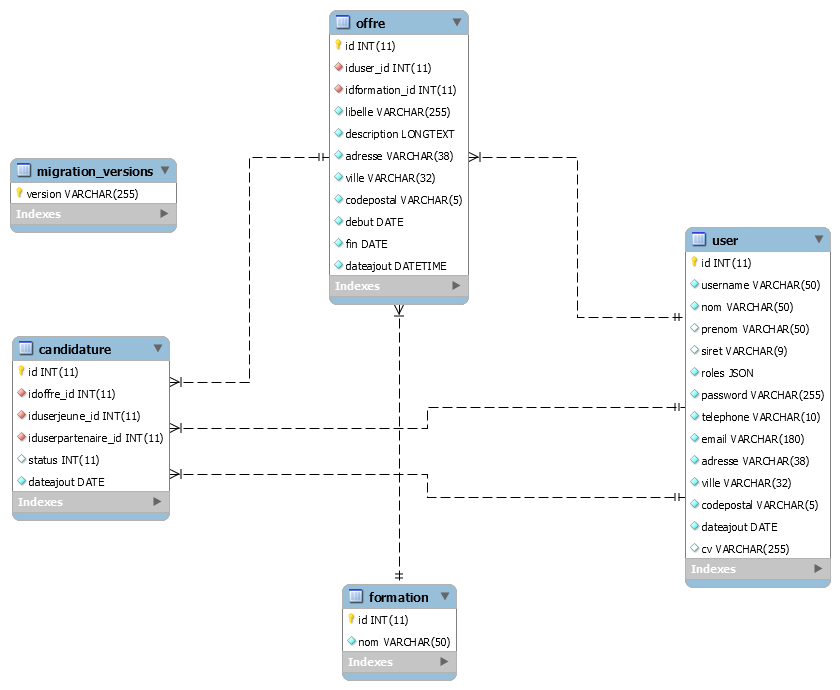
[](https://git-scm.com/)

## Base de données

### Liste des tables



### MLD



## Connexion à la base de données

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]



### Création de la base de données

La base de données est ensuite créée via cette commande :

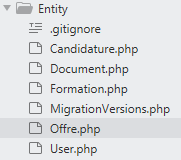


Pour générer les tables dans la base de données :



## Entités

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.



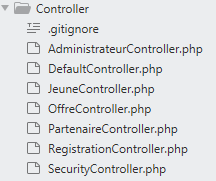
### Création des entités

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



## Controllers

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



Pour créer des controllers il faut utiliser la commande suivante :



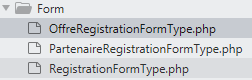
### Templates

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.



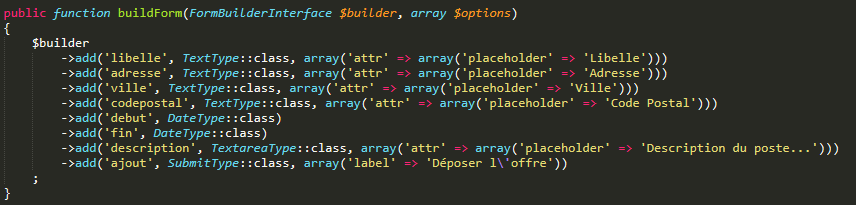
### Créer un formulaire

Dans la console il faut entrer cette commande :



### Gérer 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.



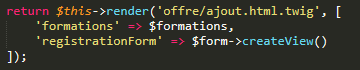
Dans le fichier « src/Controller/OffreController.php » se trouve la fonction de la capture d’écran en dessous qui montre comment l’ajout d’un formulaire se passe dans un controller, ce dernier fait appel à la classe dont elle a besoin qui se trouve dans src/Form. Pour cela il faut faire appel à ce fichier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



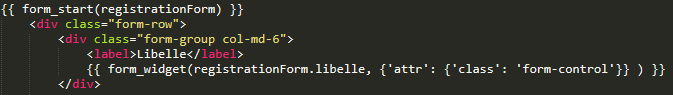
C’est dans une des fonctions d’un controller qu’il faudra faire appel au code générateur de formulaire comme le montre la capture d’écran en-dessous.



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() ».

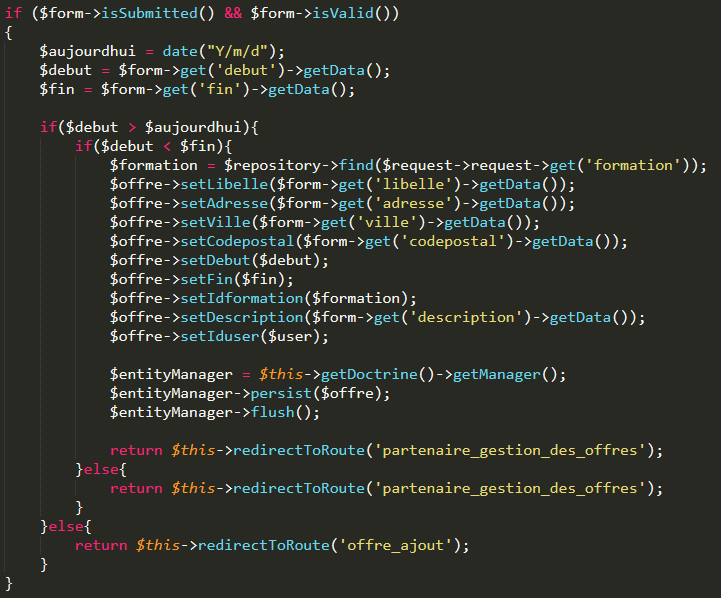


Dans la template qui se trouve dans « templates/offre/ajout.html.twig » nous mettons la syntaxe nécessaire pour faire appel au formulaire obtenu depuis le controller. « {{ form\_start(registrationForm) }} » signale où le formulaire devra être généré dans le code HTML quant à « {{ form\_end(registrationForm)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





Pour insérer ces informations dans la base de données une fois le formulaire remplit 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]) ».

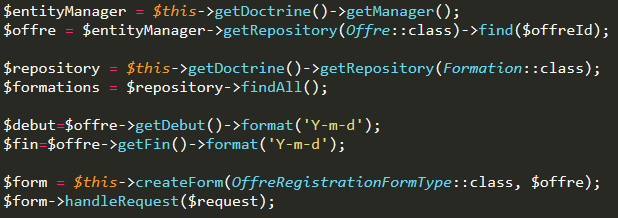


Le résultat de ces manipulations donne ceci.

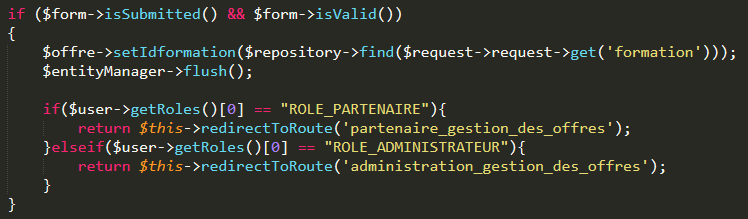


### Modification

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 controller 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 récupéré seront placé dans une instance toute seule.



Une fois les données récupéré 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.

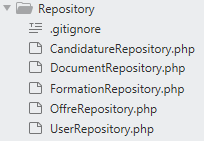


Le résultat de ces manipulations donne ceci.



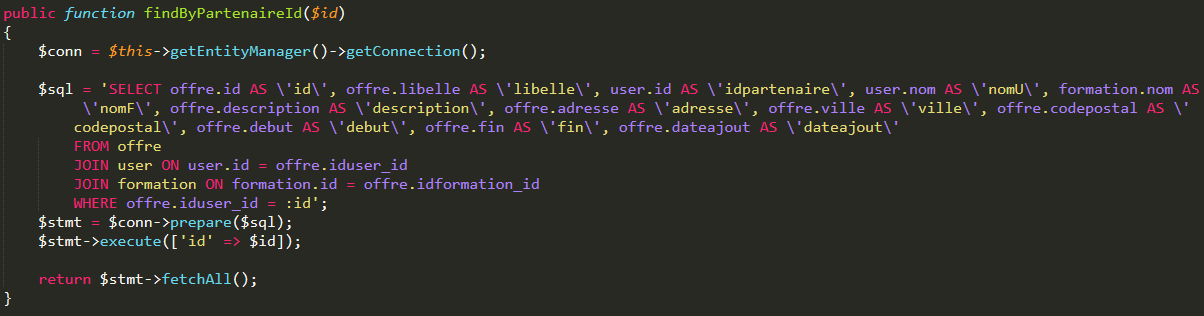
## Requêtes SQL

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 requête

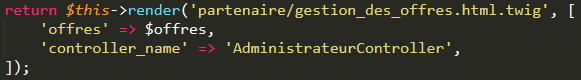
Dans le répertoire « src/Repository » se trouve les fichiers où seront écrite les requêtes, chaque fichier est lié à une entité. La requête qui se trouve dans la capture d’écran en dessous permet de récupérer les informations des offres et du partenaire en utilisant l’id du partenaire dans la clause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



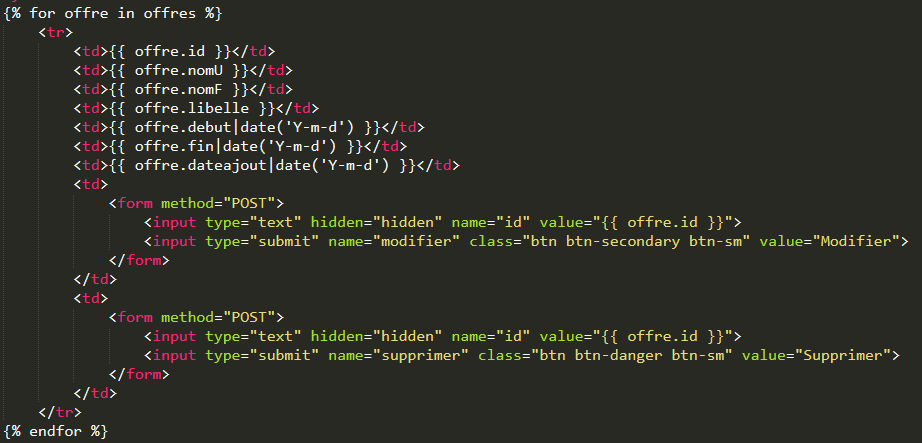
Dans le controller il faut simplement faire appel à la fonction de la classe Repository de l’entité dont vous avez besoin, cette fonction retournera des instances de l’entité lié à la classe Repository.



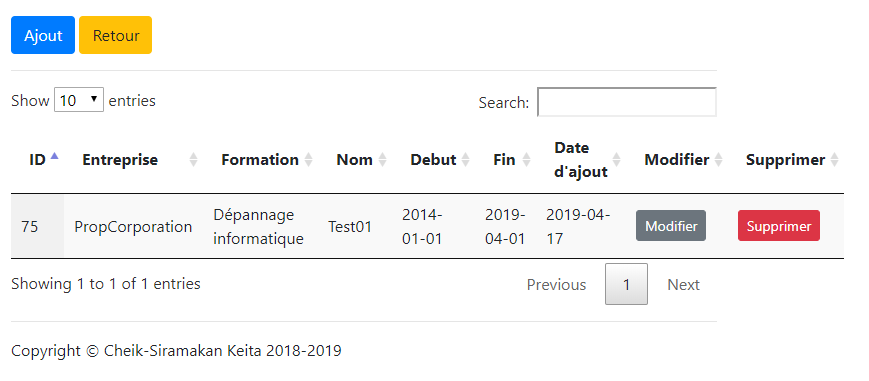
Tout ce qui reste à faire dans le controller est de faire passer les données récupérées par la requête vers la template via la fonction « render » comme ceci « ‘offres’ => $offres ».



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.



Le résultat de ces manipulations donne ceci.

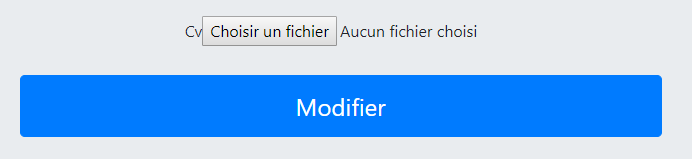


## Téléchargement

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.

### Gérer le téléchargement

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 :



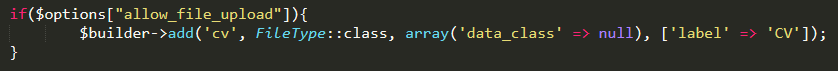
Dans le controller qui gère cette template il faut tout d’abord permettre de faire appel au code nécessaire pour le téléchargement en ajoutant les « use » en haut du fichier contenant la classe controller.



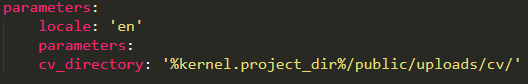
Dans la fonction qui gère la page où s’effectuera le téléchargement je fais appel à la fonction qui génère le formulaire, je permets le téléchargement de fichier avec « [‘allow\_file\_upload’ => true] ».



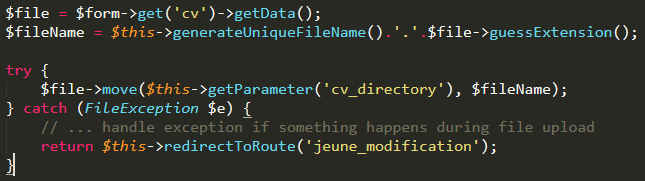
Dans le la classe « RegistrationiFormType » où se trouve la fonction qui génère le formulaire qui se trouve dans le fichier « src/Form/RegistrationFormType.php » j’ai ajouté une condition pour ajouter un champ de téléchargement.



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



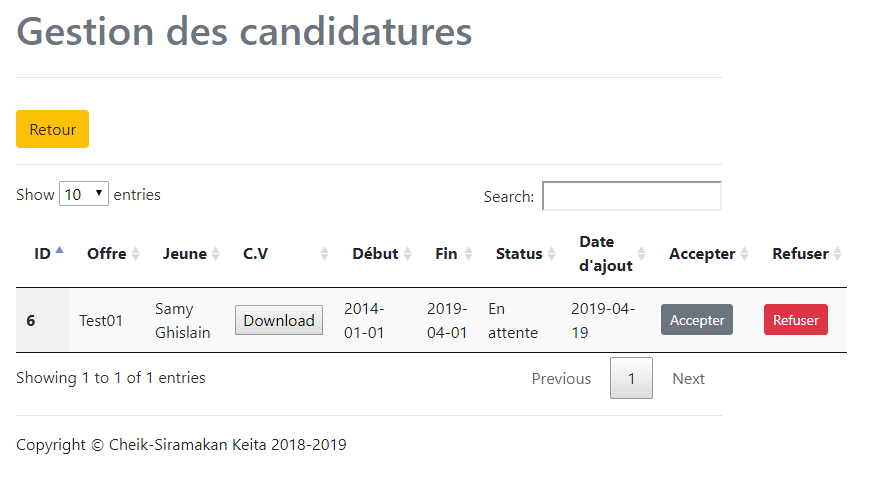
Une fois le formulaire envoyé nous récupérons le fichier puis nous le plaçons dans le répertoire que nous avons spécifié dans « config/services.yaml ». Ensuite nous enregistrons le nom du fichier dans la base de données.



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



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

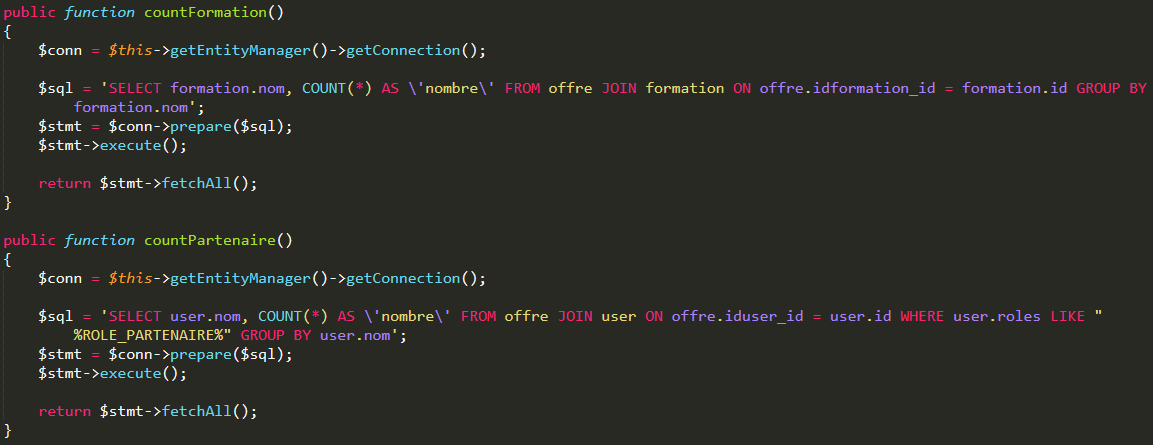


## Statistiques

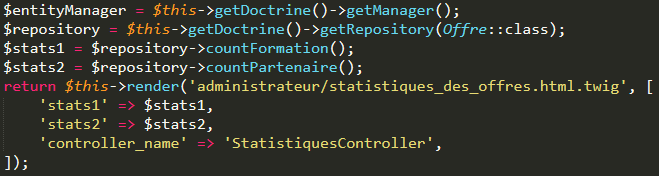
Les statistiques affichés sont le pourcentage de formations par rapport aux offres le pourcentage des offres offert par les entreprises par rapport à l’ensemble des offres.

### Gérer les statistiques

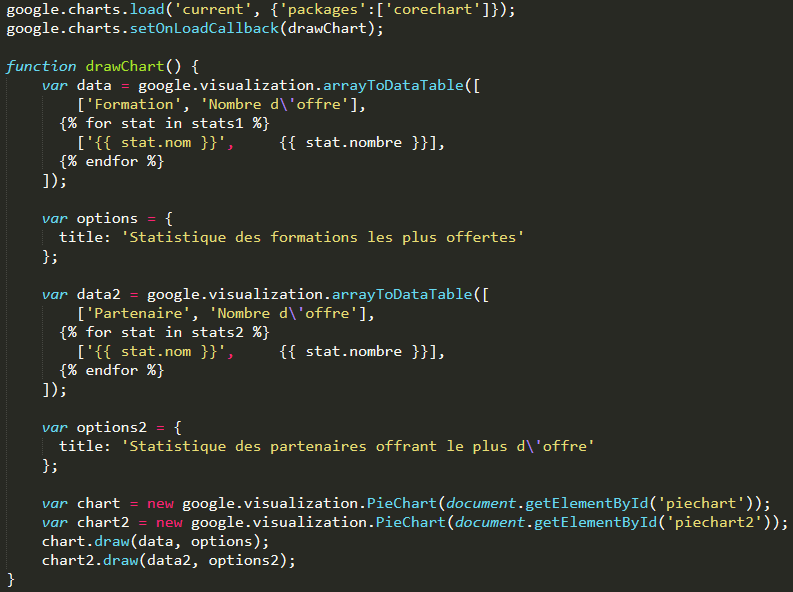
Dans la classe « OffreRepository » se trouvent les fonctions que j’utilise pour effectuer les requêtes dont j’ai besoin pour récupérer les statistiques sur les offres. La classe se trouve dans le fichier « src/Repository/OffreRepository.php ».



Dans le controller « 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.



Dans la template j’utilise Google Chart pour générer des camemberts à partir des données que j’ai passé via le controller. 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.



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



## Géolocalisation

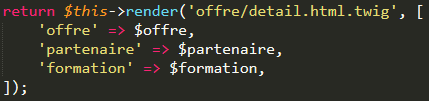
La géolocalisation est utilisée pour afficher où se passe la formation, l’adresse à laquelle se situe la formation est facilement visible grâce à un marqueur.

### Gérer la Géolocalisation

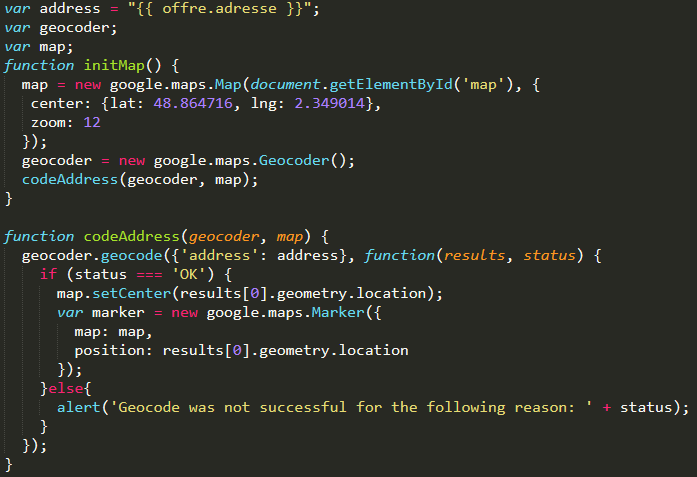
Dans le controller « OffreController », dans la fonction qui s’occupe de gérer la page de détail d’une offre qui a été sélectionné et dont l’id est enregistré dans la session je récupère les informations de cette offre avec la fonction « find() ».



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



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 }}.



J’affiche la carte en utilisant « div ».



Le résultat de ces manipulations donne ceci.

