

Exercice guidé Symfony : Salariés / Entreprises

Les objectifs de cet exercice sont les suivants :

- ✚ Création d'un projet Symfony en mode « pas-à-pas »
- ✚ Concepts de base du framework
- ✚ Installation du framework, lignes de commande principales
- ✚ Gestion des classes et des vues

A travers 2 entités majeures, **Salarié** et **Entreprise**, l'objectif sera de pouvoir afficher la liste des salariés (nom, prénom, adresse, entreprise actuelle et date d'embauche) ainsi que la liste des entreprises (raison sociale, SIRET, adresse). On pourra accéder aux détails de chacune des entités : informations supplémentaires du salarié (date de naissance par exemple) et la liste des salariés d'une entreprise.

I. Entités

Les 2 entités dont nous aurons besoin seront composées des attributs suivants :

- **Salarié** : nom, prénom, date de naissance, adresse, code postal, ville, date embauche, entreprise
- **Entreprise** : raison sociale, SIRET, adresse, code postal, ville, liste des salariés

Dans votre application, il faudra accéder aux 2 listes grâce à un menu de navigation.

Nous aborderons également la création de formulaires nous permettant d'ajouter et de modifier nos enregistrements.

II. Installation et configuration du projet

Grâce à la console (Windows + R ► cmd), rendez-vous dans le dossier « www » de WAMP (en général C:\wamp ou C:\wamp64) et tapez la commande suivante (prenez le soin de vérifier que Composer est installé sur votre poste. Dans le cas contraire : <https://getcomposer.org/download/> afin de créer votre projet :

```
composer create-project symfony/website-skeleton ExoSalaries
```

```
Microsoft Windows [version 10.0.17763.253]
(c) 2018 Microsoft Corporation. Tous droits réservés.

C:\wamp64\www>composer create-project symfony/website-skeleton ExoSalaries
Installing symfony/website-skeleton (v4.2.3.1)
- Installing symfony/website-skeleton (v4.2.3.1): Loading from cache
Created project in ExoSalaries
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
Package operations: 99 installs, 0 updates, 0 removals
- Installing ocranius/package-versions (1.3.0): Loading from cache
- Installing symfony/flex (v1.1.8): Loading from cache
- Installing symfony/polyfill-mbstring (v1.10.0): Loading from cache
- Installing symfony/contracts (v1.0.2): Loading from cache
- Installing doctrine/lexer (v1.0.1): Loading from cache
- Installing doctrine/annotations (v1.6.0): Loading from cache
```

L'installation de votre projet Symfony s'effectue en déployant tous les packages nécessaires (notez qu'une connexion internet est requise pour les packages principaux).

Exercice guidé Symfony : Salariés / Entreprises

Accédez à votre dossier crée précédemment :

```
cd ExoSalaries
```

Créez une base de données « exoSalaries » sur phpMyAdmin (<http://localhost/phpmyadmin/>) :



Ouvrez votre projet ExoSalaries dans l'IDE de votre choix afin de déployer / visualiser la structure de l'application. Dans l'arborescence, trouvez le fichier `.env` et configurez l'URL de la base de données de la façon suivante (on admettra que vous utilisez le compte root par défaut sans mot de passe, attention au port de votre moteur de base de données 3306, 3308 voir ① plus haut) :

```
###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=bf4f406870d264107046506128fb5dd9
#TRUSTED_PROXIES=127.0.0.1,127.0.0.2
#TRUSTED_HOSTS='^localhost|example\.com$'
###< symfony/framework-bundle ###

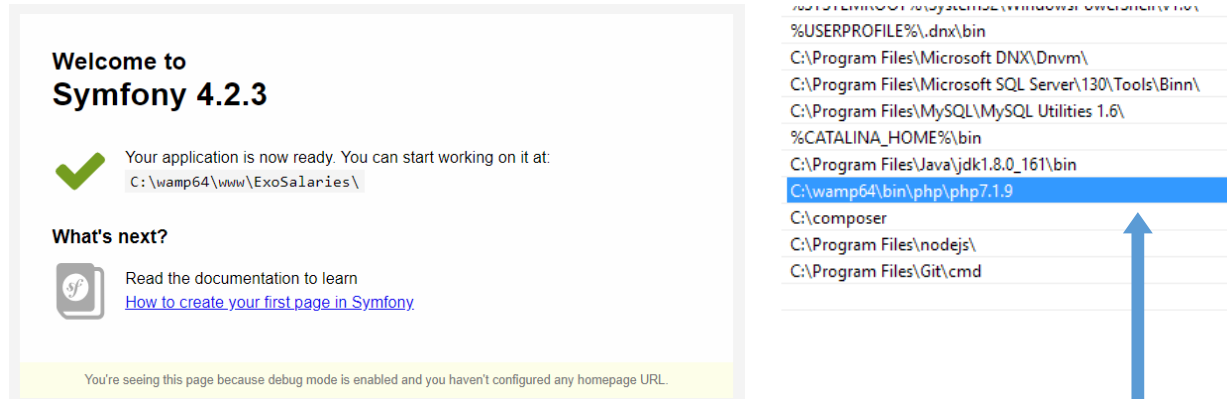
###> doctrine/doctrine-bundle ###
# Format described at http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html# configuring-the-database-connection
# For an SQLite database, use: "sqlite:///kernel.project_dir/var/data.db"
# Configure your db driver and server_version in config/packages/doctrine.yaml
DATABASE_URL=mysql://root@127.0.0.1:3308/exosalaries
###< doctrine/doctrine-bundle ###
```

Votre projet Symfony peut désormais communiquer avec votre base de données.

Exercice guidé Symfony : Salariés / Entreprises

Pour vérifier que votre projet Symfony s'exécute correctement, vous pouvez lancer la commande suivante. Vous devriez voir apparaître la page d'accueil de Symfony (<http://127.0.0.1:8000/> ou <http://localhost:8000/>) :

```
php bin/console server:run
```



The image shows two side-by-side screenshots. The left screenshot is the 'Welcome to Symfony 4.2.3' page, which includes a green checkmark icon and the text 'Your application is now ready. You can start working on it at: C:\wamp64\www\ExoSalaries\'. It also has a 'What's next?' section with a link to 'Read the documentation to learn How to create your first page in Symfony'. The right screenshot is a Windows command prompt showing the contents of the %PATH% environment variable. The path 'C:\wamp64\bin\php\php7.1.9' is highlighted in blue, and a blue arrow points from this path down to the text 'Attention de vérifier que la variable d'environnement PHP soit bien présente !'.

Attention de vérifier que la variable d'environnement PHP soit bien présente !

III. Création des entités, des contrôleurs et des repositories associés

En explorant l'arborescence de votre projet, vous remarquerez qu'il contient un dossier « src » dans lequel se trouvent les dossiers « **Entity** », « **Controller** » et « **Repository** ». Ce sont dans ces dossiers que se situe la logique métier de votre application.

1) Création des entités

Grâce à votre console, nous allons pouvoir commencer par créer les entités nécessaires à notre application. Pour ce faire, exécutez la commande suivante :

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

Nous définirons les entités **Salarié** et **Entreprise** comme suit :

Nom de l'entité : Salarie

Champ	Type	Taille	Nullable
nom	string	50	no
prenom	string	20	no
dateNaissance	date	/	no
adresse	string	50	yes
cp	string	10	yes
ville	string	30	yes
dateEmbauche	date	/	no

Nom de l'entité : Entreprise

Champ	Type	Taille	Nullable
raisonSociale	string	50	no
siret	string	20	no
adresse	string	50	yes
cp	string	10	yes
ville	string	30	yes
salaries	OneToMany ⁽¹⁾		no

(1) Concernant la relation « OneToMany » du champ « salaries », vous préciserez qu'elle concerne l'entité Salarie qu'elle n'est pas nullable et que vous acceptez que la relation soit créée également du côté de l'entité Salarie avec le nom « entreprise ». Cette relation a donc pour but de traduire qu'**une entreprise possède plusieurs salarié(e)s et qu'un(e) salarié(e) n'appartient qu'à une seule entreprise.**

Vous obtenez les entités correspondantes dans le dossier « **Entity** » (voir annexe). Vous remarquerez qu'un identifiant est généré automatiquement pour chaque entité.

Exercice guidé Symfony : Salariés / Entreprises

2) Création des contrôleurs

Afin de créer les contrôleurs nécessaires, vous exécuterez la commande suivante :

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

Il sera nécessaire de modifier les contrôleurs ainsi créés.

La méthode `index()` de `SalarieController` permettra d'afficher la liste de tous les salariés tandis que la méthode `show()` affichera le détail d'un seul salarié.

```
namespace App\Controller;

use App\Entity\Salarie;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Response;

/**
 * @Route("/salaries")
 */
class SalarieController extends AbstractController
{
    /**
     * @Route("/", name="salaries_index")
     */
    public function index()
    {
        $salaries = $this->getDoctrine()
            ->getRepository(Salarie::class)
            ->getAll();

        return $this->render('salarie/index.html.twig', [
            'salaries' => $salaries,
        ]);
    }

    /**
     * @Route("/{id}", name="salarie_show", methods="GET")
     */
    public function show(Salarie $salarie): Response {
        return $this->render('salarie/show.html.twig', ['salarie' => $salarie]);
    }
}
```

Les `@Route` dans les annotations permettent de définir la façon dont on appellera les différentes méthodes grâce à l'URL de notre navigateur. Ainsi pour lister les salariés ou pour accéder au détail d'un salariés, nos URL ressembleront à :

<http://localhost:8000/salaries> ou <http://localhost:8000/salaries/2>

(2 étant l'identifiant du salarié souhaité)

Exercice guidé Symfony : Salariés / Entreprises

Vous ferez en sorte de coder le contrôleur « EntrepriseController » de la même façon en remplaçant « Salarie » par « Entreprise ».

Nous avons jusqu'à présent codé les **Entity** et les **Controllers**. Il nous reste à implémenter les **Repositories**, à migrer notre base de données et à créer les vues (templates TWIG).

3) Modification des Repositories

En créant les **Entity** manuellement, Symfony a également généré les **Repositories** associés (dans le dossier « Repository »). Ceux-ci contiennent l'ensemble des méthodes permettant d'interagir avec notre base de données notamment grâce au langage DQL (Doctrine Query Language), Doctrine étant l'ORM (**O**bject-**R**elational **M**apping) utilisé par Symfony, nous allons bien manipuler des **objets** contrairement au langage SQL classique.

`SELECT s FROM App\Entity\Salarie s ORDER BY s.dateEmbauche DESC` affichera la liste de tous les salariés triés par date d'embauche de la plus récente à la plus ancienne.

Pour l'entité **Salarie**, nous modifierons **SalarieRepository** comme ci-dessous (même logique pour **EntrepriseRepository**). Notez-bien le « `use App\Entity\Salarie` » en début de classe sans lequel il nous sera impossible d'exploiter l'entité **Salarie**.

```
namespace App\Repository;

use App\Entity\Salarie;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Symfony\Bridge\Doctrine\RegistryInterface;

/**
 * @method Salarie|null find($id, $lockMode = null, $lockVersion = null)
 * @method Salarie|null findOneBy(array $criteria, array $orderBy = null)
 * @method Salarie[] findAll()
 * @method Salarie[] findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class SalarieRepository extends ServiceEntityRepository
{
    public function __construct(RegistryInterface $registry)
    {
        parent::__construct($registry, Salarie::class);
    }

    public function getAll(){
        $entityManager = $this->getEntityManager();
        $query = $entityManager->createQuery(
            'SELECT s
            FROM App\Entity\Salarie s
            ORDER BY s.dateEmbauche DESC'
        );
        return $query->execute();
    }
}
```

IV. Migration de la base de données

Maintenant que l'ensemble de la logique métier a été implémentée, nous pouvons passer à la migration de notre base de données grâce à deux lignes de commande :

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

Si la migration s'est bien passée, vous devriez retrouver dans votre base de données les 2 tables « entreprise » et « salarié » créées depuis vos entités.

Vous pouvez à présent remplir la base de données avec 2 entreprises et 4 salariés (via phpMyAdmin) :

Entreprises

- **ELAN FORMATION**
 - o SIRET : 123456
 - o 202 avenue de Colmar 67100 STRASBOURG
- **SEEDDEV**
 - o SIRET : 987654
 - o 10 route de Strasbourg, 67000 STRASBOURG

Salariés

- **DUSSAUCY Pascal**
 - o Né le 01/05/1970
 - o 5 rue de la Gare, 68000 COLMAR
 - o Embauché le 02/05/1990 chez ELAN FORMATION
- **GIBELLO Virgile**
 - o Né le 16/01/1984
 - o 1 bd de Lyon, 67000 STRASBOURG
 - o Embauché le 03/04/2012 chez ELAN FORMATION
- **SUDARO Mathieu**
 - o Né le 09/08/1996
 - o 20 rue Principale, 67000 STRASBOURG
 - o Embauché le 12/09/2016 chez SEEDDEV
- **ANDRES Mathilde**
 - o Née le 05/04/1994
 - o 5 route des Vignes, 67000 STRASBOURG
 - o Embauchée le 01/06/2017 chez ELAN FORMATION

V. Création des vues grâce à TWIG

Twig est un moteur de templates pour le langage de programmation PHP, utilisé par défaut par le Framework Symfony.

En déployant notre projet comme précédemment, Symfony a déjà créé les dossiers de vues en accord avec nos entités. Vous les trouverez dans le dossier « **templates** » à la racine du projet.

Par défaut, vous trouverez la vue « **base.html.twig** » qui contient le squelette de l'ensemble de vos pages. Toutes vos vues hériteront de cette « base » pour éviter d'avoir à recoder les parties communes à celles-ci (`{% extends 'base.html.twig' %}`)

Nous avons ajouté ici un block « **contenu** » grâce à `{% block contenu %} {% endblock %}`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>{% block title %}Welcome!{% endblock %}</title>
    {% block stylesheets %}{% endblock %}
  </head>
  <body>
    {% block body %}
      {% block contenu %}{% endblock %}
    {% endblock %}
    {% block javascripts %}{% endblock %}
  </body>
</html>
```

Si nous souhaitons afficher la liste des salariés (sous forme de tableau par exemple) :

```
{% extends 'base.html.twig' %}

{% block contenu %}
  <table>
    <thead>
      <tr>
        <th>SALARIE</th>
        <th>ADRESSE</th>
        <th>CP</th>
        <th>VILLE</th>
        <th>DATE_EMBAUCHE</th>
        <th>ENTREPRISE</th>
      </tr>
    </thead>
    <tbody>
      {% for salarie in salaries %}
        <tr>
          <td><a href="{{ path('salarie_show', {'id':salarie.id}) }}">{{ salarie.nom ~ ' ' ~ salarie.prenom }}</a></td>
          <td>{{ salarie.adresse }}</td>
          <td>{{ salarie.cp }}</td>
          <td>{{ salarie.ville }}</td>
          <td>{{ salarie.dateEmbauche | date('d/m/Y') }}</td>
          <td>{{ salarie.entreprise.raisonSociale }}</td>
        </tr>
      {% endfor %}
    </tbody>
  </table>
{% endblock %}
```

| `date('d/m/Y')` permet d'afficher la date d'embauche au format francophone.
Il existe beaucoup de filtres très pratiques dans la documentation de Twig

Exercice guidé Symfony : Salariés / Entreprises

Dans la boucle `for`, « salaries » fait bien référence au nom du paramètre du contrôleur « **SalarieController** » de la méthode `getAll()`.

```
return $this->render('salarie/index.html.twig', [
    'salaries' => $salaries,
]);
```

Concernant la vue qui affiche le détail d'un salarié, notre vue sera plus simple à écrire :

```
{% extends 'base.html.twig' %}

{% block contenu %}
    <h1>{{ salarie.nom ~ ' ' ~ salarie.prenom }}</h1>
{% endblock %}
```

Vous pouvez désormais coder les vues concernant la liste des entreprises et leurs détails selon le même principe.

Vous noterez dans la vue du listing des employés, que l'on accède à son détail en passant par un lien hypertexte construit avec `path`. Celui-ci fait référence au nom de la route de la méthode `show()` de **SalarieController**. Il est plus facile d'appeler une méthode par son nom. C'est l'annotation de la route qui permet cette simplification.

```
/**
 * @Route("/{id}", name="salarie_show", methods="GET")
 */
public function show(Salarie $salarie): Response {
```

Pour tester votre application, rendez-vous dans votre console.

A la racine de votre projet, vous pourrez exécuter la commande suivante :

```
php bin/console server:run
```

Dans votre navigateur, vous pourrez alors saisir l'URL suivante :

```
http://localhost:8000/salaries
```

VI. Ajout et modification des données

Dans cette partie de l'exercice, nous traiterons de la possibilité de créer un formulaire permettant à la fois l'ajout et la modification d'un enregistrement dans notre base de données.

Par exemple, pour l'ajout / modification d'un salarié, il faudra créer et modifier des fichiers existants du projet :

- **Ajout** : add_edit.html.twig
- **Modification** : SalarieController, salarie/index.html.twig

Salarie/index.html.twig

Ajoutez une colonne supplémentaire « Modifier » dans votre tableau et lien correspondant :

```
<a href="{{ path('salarie_edit', {'id':salarie.id}) }}">Modifier</a>
```

SalarieController

Nous allons ajouter une méthode `addSalarie()` dans notre contrôleur qui nous permettra à la fois d'ajouter et de modifier un salarié. Pour ce faire, nous nous servirons des annotations afin de construire les bonnes routes.

Attention à créer cette méthode avant les méthodes « index » et « show » pour éviter les conflits de routage !

Quelques précisions :

- Les 2 routes nous permettent de gérer l'ajout et la modification d'un salarié
- Le test conditionnel en début de méthode nous permet de vérifier si le salarié existe déjà (dans le cas d'une modification). Si ce n'est pas le cas, nous instancions un nouveau Salarié.
- Nous déléguons ici la création du formulaire via le contrôleur. Notez que ce n'est pas l'unique façon de faire. Vous remarquerez l'ensemble des « use » qui ont été ajoutés en début de classe.

```
use App\Entity\Salarie;
use App\Entity\Entreprise;

use Doctrine\Common\Persistence\ObjectManager;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;

use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
```

Exercice guidé Symfony : Salariés / Entreprises

- L'argument « `editMode` » en fin de méthode nous permettra de modifier certains éléments dans notre vue (notamment le titre de la page. Cf page 11).
- Afin de pouvoir modifier l'entreprise du salarié, nous sommes dans l'obligation de définir une liste déroulante de type « EntityType » (`'class' => Entreprise::class`) dont le contenu sera la raison sociale de l'entreprise (`'choice_label' => 'raisonSociale'`)

```
/**
 * @Route("/add", name="salarie_add")
 * @Route("/{id}/edit", name="salarie_edit")
 */
public function addSalarie(Salarie $salarie = null, Request $request, ObjectManager $manager){

    if(!$salarie){
        $salarie = new Salarie();
    }

    $form = $this->createFormBuilder($salarie)
        ->add('nom', TextType::class)
        ->add('prenom', TextType::class)
        ->add('datenaissance', DateType::class, [
            'years' => range(date('Y'), date('Y')-70),
            'label' => 'Date de naissance',
            'format' => 'ddMMMyyyy'
        ])
        ->add('adresse', TextType::class)
        ->add('cp', TextType::class)
        ->add('ville', TextType::class)
        ->add('dateEmbauche', DateType::class, [
            'years' => range(date('Y'), date('Y')-70),
            'label' => 'Date d\'embauche',
            'format' => 'ddMMMyyyy'
        ])
        ->add('Entreprise', EntityType::class, [
            'class' => Entreprise::class,
            'choice_label' => 'raisonSociale',
        ])
        ->add('Valider', SubmitType::class)
        ->getForm();

    $form->handleRequest($request);

    if($form->isSubmitted() && $form->isValid()){

        $manager->persist($salarie);
        $manager->flush();

        return $this->redirectToRoute('salaries_index');
    }

    return $this->render('salarie/add_edit.html.twig', [
        'form' => $form->createView(),
        'editMode' => $salarie->getId() != null
    ]);
}
```

Exercice guidé Symfony : Salariés / Entreprises

Salarie/add_edit.html.twig

Il ne nous reste plus qu'à créer la vue nous permettant d'accueillir le formulaire d'ajout / modification de salariés. Vous remarquerez que le test sur « `editMode` » permet bien de modifier le titre de la page en fonction de l'action souhaitée.

Vous pouvez désormais ajouter un bouton « Ajouter un salarié » dans la vue des salariés (qui mènera vers la route correspondante du contrôleur `SalarieController`).

```
add_edit.html.twig x SalarieController.php
1  {% extends 'base.html.twig' %}
2
3  {% block contenu %}
4      {% if editMode %}
5          <h1>Modification</h1>
6      {% else %}
7          <h1>Ajout</h1>
8      {% endif %}
9
10     {{ form(form) }}
11 {% endblock %}
```

VII. Bonus

- Ajoutez un lien « Supprimer » dans la liste des salariés. Ce lien permettra de supprimer définitivement un salarié de la base de données.
- Afin d'ajouter une surcouche de présentation Bootstrap à vos formulaires, ouvrez le fichier « **config/packages/twig.yaml** » et ajoutez la ligne suivante (**form_themes**) :

```
twig:
  default_path: '%kernel.project_dir%/templates'
  debug: '%kernel.debug%'
  strict_variables: '%kernel.debug%'
  form_themes: ['bootstrap_4_layout.html.twig']
```

- Afin que l'affichage des mois de l'année soient en français, vous pouvez définir la « locale » dans le fichier « **config/packages/translation.yaml** », comme suit :

```
framework:
  default_locale: 'fr'
  translator:
    default_path: '%kernel.project_dir%/translations'
    fallbacks:
      - '%locale%'
```

VIII. Annexes

Entité Entreprise

```
namespace App\Entity;

use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\EntrepriseRepository")
 */
class Entreprise
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=200)
     */
    private $raisonSociale;

    /**
     * @ORM\Column(type="string", length=100)
     */
    private $SIRET;

    /**
     * @ORM\Column(type="string", length=100, nullable=true)
     */
    private $adresse;

    /**
     * @ORM\Column(type="string", length=10, nullable=true)
     */
    private $cp;

    /**
     * @ORM\Column(type="string", length=100, nullable=true)
     */
    private $ville;

    /**
     * @ORM\OneToMany(targetEntity="App\Entity\Salarie", mappedBy="entreprise")
     */
    private $salaries;
```

Exercice guidé Symfony : Salariés / Entreprises

Entité Salarie

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\SalarieRepository")
 */
class Salarie
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=50)
     */
    private $nom;

    /**
     * @ORM\Column(type="string", length=50)
     */
    private $prenom;

    /**
     * @ORM\Column(type="date")
     */
    private $dateNaissance;

    /**
     * @ORM\Column(type="string", length=150, nullable=true)
     */
    private $adresse;

    /**
     * @ORM\Column(type="string", length=10, nullable=true)
     */
    private $cp;

    /**
     * @ORM\Column(type="string", length=100, nullable=true)
     */
    private $ville;

    /**
     * @ORM\Column(type="date")
     */
    private $dateEmbauche;

    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\Entreprise", inversedBy="salaries")
     */
    private $entreprise;
```

Exercice guidé Symfony : Salariés / Entreprises

Vues principales de l'application

Salariés Entreprises

SALARIE	ADRESSE	CP	VILLE	DATE EMBAUCHE	ENTREPRISE	MODIFIER	SUPPRIMER
SUDARO Mathieu	20 route des Champs	67000	STRASBOURG	07/05/2016	SEEDDEV	Modifier	Supprimer
GIBELLO Virgile	1 bld de Lyon	67300	SCHILTIGHEIM	03/02/2012	ELAN FORMATION	Modifier	Supprimer
MURMANN Mickael	10 rue de la Gare	67000	STRASBOURG	01/02/2010	ELAN FORMATION	Modifier	Supprimer

[Ajouter un salarié](#)

Ajout

Nom

Prenom

Date de naissance

Adresse

Cp

Ville

Date d'embauche

Entreprise

[Valider](#)

Modification

Nom

Prenom

Date de naissance

Adresse

Cp

Ville

Date d'embauche

Entreprise

[Valider](#)