Symfony et Doctrine

Notre serveur sera responsable de traiter les données de l'application. Les données seront stockées dans une BD MySQL accessible par l'entremise de l'ORM **Doctrine**.

ORM veut dire Object Relational Mapping. Un ORM sert d'intermédiaire (Mapping) entre le monde SQL (Relational) de MySQL et le monde orienté objet de PHP (Object). Il rend transparent pour votre code les concepts SQL (table, colonne, query, etc) pour que vous puissiez vous consacrer à votre modèle de classes, sans vous soucier des aspects de la conversion de vos objets en données relationnelles et vice versa.

L'entité (entity) est le concept central de l'ORM-Doctrine. Une entité est une classe PHP à laquelle on ajoute des annotations qui font la correspondance entre le monde OO et le monde SQL. Plusieurs classes métiers d'un projet (parfois toutes) seront des « entities ».

1- Configurer la connexion de la BD dans le fichier …/chomeQuiPeut/**.env**
Insérez une ligne définissant la **DATABASE_URL** pour la BD de dév :

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/chomeQuiPeut?serverVersion=8.0.32&charset=utf8mb4"
```

Note : Le fichier **.env** est stratégique : en plus de configurer la connexion de la BD il configure aussi le type d'exécution de l'app, dev ou prod. Nous y reviendrons…

2- Nous utiliserons les fonctionnalités de l'ORM, donc les tables de la BD ne seront pas accédées directement. Nous passerons par les « entités ». Créons notre première entité : **OffreEmploi**
Dans une ICW:

```
php bin/console make:entity OffreEmploi
(ou symfony console make:entity OffreEmploi)
```

La console vous demandera de fournir les noms et les types des attributs de l'entité. Chomeur aura les attributs suivants et aucun ne peut être null **:**
titre (string 100)
description(string 255)
salaireAnnuel (integer)
datePublication(datetime)

Note : on ne fournit pas de id, c'est Doctrine qui s'en occupe

3- À la racine du projet créez un fichier **init.bat** et mettez-y ce contenu :

```
php bin\console doctrine:database:drop --force
```

```
del var\cache\*.* /s /q >scrap
php bin\console doctrine:database:create
php bin\console doctrine:schema:create
```

4- Dans une ICW, à la racine du projet (./chomeQuiPeut) exécutez la commande :

```
Init
```

ATTENTION : ce script est radical car il supprime la bd configurée dans votre fichier **.env** **N'exécutez jamais ce script en production** et assurez-vous d'avoir une sauvegarde de vos données avant de le rouler.

5- Voyez ensuite le résultat en BD par phpMyAdmin :
La BD est créée ainsi que la table **offre_emploi** dont chaque colonne correspond à un attribut de l'entité **OffreEmploi.** Remarquez que la colonne **id** a été créée et que c'est une clé primaire auto-increment.

6- Voici un extrait du PHP de l'entité ./chomeQuiPeut/src/Entity/**OffreEmploi.php :**

```
<?php

namespace App\Entity;

use App\Repository\OffreEmploiRepository;
use Doctrine\DBAL\Types\Types;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: OffreEmploiRepository::class)]
class OffreEmploi
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 100)]
    private ?string $titre = null;

    #[ORM\Column(length: 255)]
    private ?string $description = null;
```

7- L'entité est une simple classe PHP qui a la particularité d'être précédée d'une annotation. Les lignes suivantes :

```
#[ORM\Entity(repositoryClass: OffreEmploiRepository::class)]
class OffreEmploi
{
```

Indiquent que la classe OffreEmploi est une entité (#[ORM\Entity(…) donc que Doctrine doit créer une table pour stocker les instances de cette classe et les récupérer au besoin par l'entremise d'une requête au repository des offres d'emplois.

8- Chaque attribut sera aussi préfixé d'une annotation (#[ORM\Column…). Par exemple :

```
#[ORM\Column(length: 100)]
 private ?string $titre = null;
```

9- Les annotations sont des directives pour orienter Doctrine. Parfois il y a des attributs qu'on ne souhaite pas conserver dans une colonne en BD, dans ce cas vous ne mettrez aucune annotation et l'attribut sera ignoré par l'ORM.

Note : Doctrine a ses propres règles pour traduire un nom d'entité en nom de table et un nom d'attribut en nom de colonne. Par exemple l'attribut salaireAnnuel sera converti en colonne salaire_annuel. Si ces règles ne vous conviennent pas il est possible d'imposer vos propres noms de tables et de colonnes.

10- Dans phpmyadmin, choisissez la BD chomeQuiPeut et sa table **offre_emploi** et par l'onglet **Insérer,** insérez trois offres d'emplois.

11- Prochain défi: récupérer les trois offres d'emplois et les afficher à l'accueil de notre site. Dans ./chomeQuiPeut/src/Controller/BaseController.php nous:

- Ajoutons un **use Doctrine\Persistence\ManagerRegistry;** pour accéder aux fonctionnalités de Doctrine.
- Ajoutons un **use App\Entity\OffreEmploi;** pour avoir la définition de l'entité OffreEmploi.
- Ajoutons le paramètre **$doctrine** à la méthode ->index() de la route 'accueilCQP'.
- Utilisons la méthode **findAll()** du repository des entités **OffreEmploi,** obtenu par le manager des entités du paramètre **$doctrine**. Le findAll() retournera toutes les entités enregistrées dans la BD dans la table **offre_emploi**.
- Mettons le retour de **findAll()** dans le tableau **$offresEmplois** et passons ce tableau à **accueil.html.twig :**

```
use Doctrine\DBAL\Connection;
```

```
class BaseController extends AbstractController
{
    #[Route('/', name: 'accueilCQP')]
    public function index(ManagerRegistry $doctrine): Response
    {
        $offresEmplois = $doctrine->getManager()
                                  ->getRepository(OffreEmploi::class)
                                  ->findAll();
        return $this->render('accueil.html.twig',
                             ['tabOffresEmplois' => $offresEmplois ]);
    }
}
```

12- Modifions le fichier ./chomeQuiPeut/templates/accueil.html.twig pour traiter le tableau des offres d'emplois reçu:

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

{% block stylesheets %}
   <link rel="stylesheet" href="/css/chomeQuiPeut.css">
{% endblock %}

{% block body %}
   <div class="row">
    <section class="col-3 sectionGauche">
        <h4>Travailleurs inscrits</h4>
        <ul>
        </ul>
     </section>
     <section class="col-9 sectionDroite">
        <h4>Emplois disponibles</h4>
        <ul>
            {% for offre in tabOffresEmplois %}
            <li>{{offre.titre}}</li>
            {% endfor %}
        </ul>
     </section>
   </div>
 {% endblock %}
```

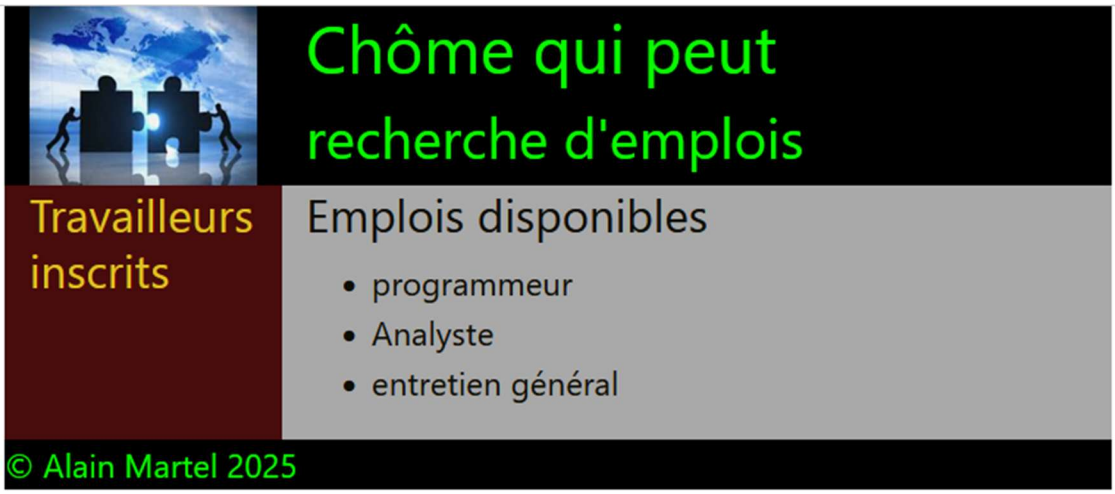
13- Remarquez l'utilisation de la boucle **or** de Twig.

- Les commandes en twig : {% %} {%end%}
- Les interpolations {{ }}
- Les commentaires {# #}

14- Testez cette route dans votre fureteur avec cette URL :

https://localhost:8000/

Vous devriez obtenir vos vraies données!



Pour aller plus loin

À vous de jouer maintenant.

1- Créez une entité pour les chômeurs qui aura les attributs suivants :
   - nom (string 50)
   - courriel (string 100)
   - telephone (string 10)
   - dateInscription (datetime)
   - dateNaissance (date)
2- Créez un fichier maj.bat et mettez-y ce code :

```
del var\cache\*.* /s /q >scrap
php bin\console doctrine:schema:update --dump-sql
php bin\console doctrine:schema:update --force
```

Exécutez-le, et vérifiez que la table des chômeurs a été créée avec les bonnes colonnes.

Note : Maj.bat est un fichier du même genre que init.bat, mais moins violent, il ne détruit pas les données. Il met à jour la BD pour qu'elle reflète l'état des entités définies dans le projet

3- Insérez trois chômeurs dans la BD
4- Modifiez le contrôleur pour qu'il récupère vos trois chômeurs et les passe à twig
5- Modifiez le twig pour qu'il affiche vos chômeurs