

TP3 :

Partie 1 : L'ORM Doctrine

- ✓ Les Frameworks Web PHP, en particulier Symfony, n'utilisent pas ces extensions **directement** pour l'accès aux BDR mais utilise une autre technique appelée ORM (Object-Relational Mapping).
- ✓ L'ORM utilisé par Symfony s'appelle Doctrine.
- ✓ Les données seront toujours sauvegardées dans des tables dans une BDR.
- ✓ Mais l'application manipulera ces données (en ajout, suppression, sélection, modification,...) à travers la vue objet(classes entités) correspondante à cette BDR offerte par l'ORM.

Démarche à suivre :

1. Configuration de l'application

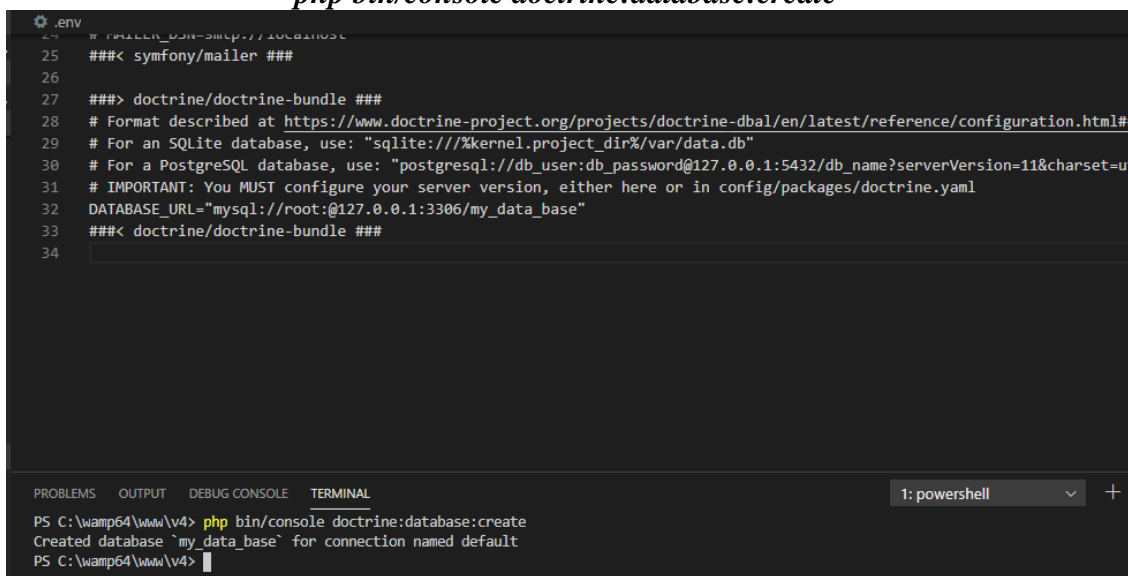
Fichier **.env**, on définit dans la variable **DATABASE_URL** les paramètres d'accès à la BDR: nom de la BD, nom utilisateur et son mot de passe:

DATABASE_URL=mysql://db_user:db_password@127.0.0.1/db_name

2. Création de la BD

On peut créer la BD en exécutant la commande :

php bin/console doctrine:database:create



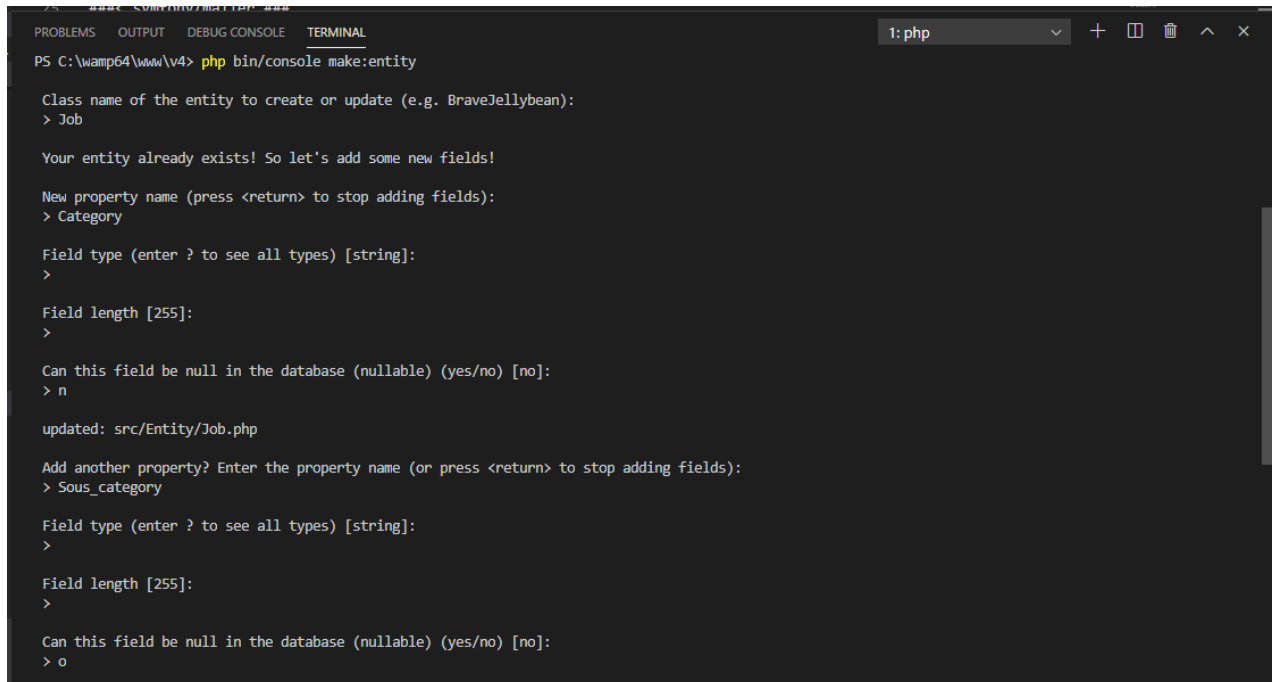
```
env
25  ###< symfony/mailer ###
26
27  ###> doctrine/doctrine-bundle ###
28  # Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#
29  # For an SQLite database, use: "sqlite:///kernel.project_dir%/var/data.db"
30  # For a PostgreSQL database, use: "postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=11&charset=utf8"
31  # IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
32  DATABASE_URL="mysql://root:@127.0.0.1:3306/my_data_base"
33  ###< doctrine/doctrine-bundle ###
34

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\wamp64\www\w4> php bin/console doctrine:database:create
Created database 'my_data_base' for connection named default
PS C:\wamp64\www\w4>
```

3. Définition de l'entité :

Pour créer une nouvelle entité, il faut taper la commande :

php bin/console make:entity



```
PS C:\wamp64\www\v4> php bin/console make:entity

Class name of the entity to create or update (e.g. BraveJellybean):
> Job

Your entity already exists! So let's add some new fields!

New property name (press <return> to stop adding fields):
> Category

Field type (enter ? to see all types) [string]:
>

Field length [255]:
>

Can this field be null in the database (nullable) (yes/no) [no]:
> n

updated: src/Entity/Job.php

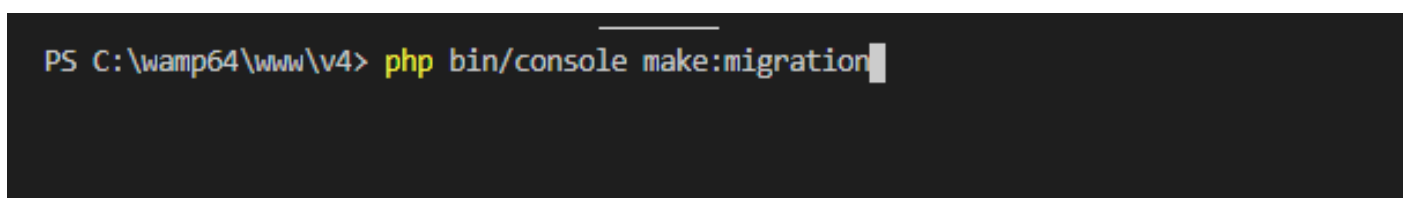
Add another property? Enter the property name (or press <return> to stop adding fields):
> Sous_category

Field type (enter ? to see all types) [string]:
>

Field length [255]:
>

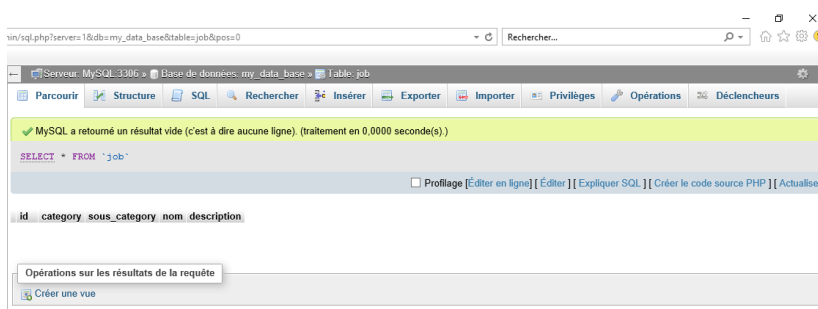
Can this field be null in the database (nullable) (yes/no) [no]:
> o
```

Doctrine permet de lancer une commande qui affiche la ou les requêtes SQL **qui vont générer la ou les tables correspondante à la définition de ou des Entités définis.**



```
PS C:\wamp64\www\v4> php bin/console make:migration
```

Et voilà notre table a été créée.



Pour migrer vers une nouvelle version de la BDR à partir de définition des nouvelles entités, il suffit d'exécuter la commande :

```
PS C:\wamp64\www\v4> php bin/console doctrine:migrations:migrate
```

Les services de l'ORM :

✓ Le service Doctrine :

\$doctrine = \$this->getDoctrine() : On fait appel à ce service pour pouvoir accéder à un autre service : celui de l'EntityManager.

Pour insérer des nouvelles instances, on définit par exemple une action save() :

```
/**
 * @Route("/save")
 */
public function save(){

    $entityManager = $this->getDoctrine()->getManager();
    $job = new Job();
    $job->setCategory('informatique');
    $job->setSousCategory('Développeur_web');
    $job->setNom('Développeur web');
    $job->setDescription('Développer votre site web dans une semaine');
    $entityManager->persist($job);
    $entityManager->flush();
    return new Response ('save this job'. $job->getId());
}
```

✓ Le service EntityManager :

Permet de gérer les entités à savoir :

- ✓ Créer une entité
- ✓ Modifier une entité
- ✓ Supprimer une entité...

Persist(\$job) : Permet de persister l'instance d'une entité en argument. Ceci a pour conséquence la création d'un nouvel enregistrement dans la table correspondante à l'entité.

flush() : synchronise l'état des instances de l'entité avec celui de la table dans la BDR.

✓ Le service Repository :

Ce service utilise le service Doctrine pour **récupérer** les entités depuis la base de données.

Pour utiliser ce service il suffit de déclarer comme argument dans une action, une instance de la classe Repository:

```
Public function Index(JobRepository $repository){}
```

Et par la suite on utilise les méthodes du service repository tel que :

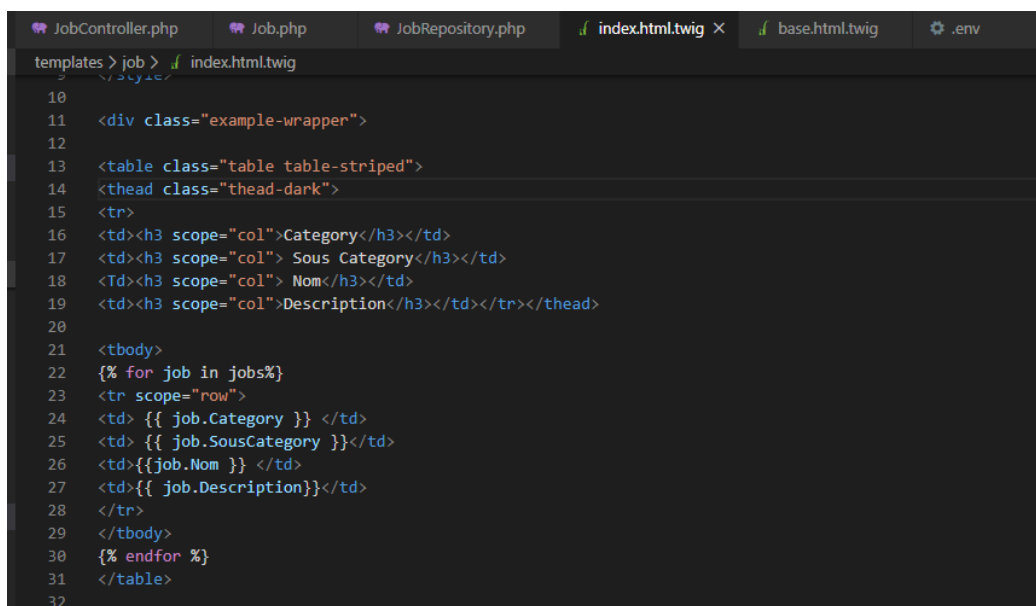
- **FindAll()** : pour la recherche de toutes les entités.



```
Controller.php • JobRepository.php index.html.twig .env home.html.twig
Controller > JobController.php

/**
 * @Route("/job", name="joblist")
 */
public function index(JobRepository $repository)
{
    $jobs= $repository->findAll();
    return $this->render('job/index.html.twig', ['jobs' => $jobs]);
}
```

index.html.twig :



```
JobController.php Job.php JobRepository.php index.html.twig × base.html.twig .env
templates > job > index.html.twig

10
11 <div class="example-wrapper">
12
13 <table class="table table-striped">
14 <thead class="thead-dark">
15 <tr>
16 <td><h3 scope="col">Category</h3></td>
17 <td><h3 scope="col"> Sous Category</h3></td>
18 <td><h3 scope="col"> Nom</h3></td>
19 <td><h3 scope="col">Description</h3></td></tr></thead>
20
21 <tbody>
22 {% for job in jobs%}
23 <tr scope="row">
24 <td>{{ job.Category }} </td>
25 <td>{{ job.SousCategory }}</td>
26 <td>{{ job.Nom }} </td>
27 <td>{{ job.Description}}</td>
28 </tr>
29 </tbody>
30 {% endfor %}
31 </table>
32
```

Find(\$id) : qui permet la recherche d'une entité par son id :

```
49      /**
50       * @Route("/job/{id}", name="JobById")
51       */
52      public function byCategory($id){
53          $j = $this->getDoctrine()->getRepository(job::class)->find($id);
54          return $this->render('job/category.html.twig', ['job'=>$j]);
55      }
```

ById.html.twig :

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

{% block title %}Job by id{% endblock %}

{% block body %}

<h1>{{ job.id }}</h1>
{% endblock %}
```

Partie2 : Construire un formulaire :

Installer le package form :

Composer require form

Dans une action d'un contrôleur :

Créer une instance de l'entité job

- Construire un formulaire en spécifiant la classe du formulaire et l'instance de l'entité job.
- Renvoi une vue en lui transmettant le vue crée du formulaire.

```

Debug Terminal Help JobController.php - v4 - Visual Studio Code [Administrator]

JobController.php x new.html.twig category.html.twig Job.php JobRepository.php

src > Controller > JobController.php
53     $j = $this->getDoctrine()->getRepository(job::class)->find($id);
54     return $this->render('job/category.html.twig', ['job'=>$j]);
55 }
56 /**
57  * @Route("/new", name="new_job")
58  * Method( { "POST"})
59  */
60     public function new(Request $request){
61         $job = new Job();
62         $form = $this->createFormBuilder($job)
63             ->add('Category', TextType ::class, ['attr'=>['class'=>'form-control']])
64             ->add('SousCategory', TextType ::class, ['attr'=>['class'=>'form-control']])
65             ->add('Nom', TextType ::class, ['attr'=>['class'=>'form-control']])
66             ->add('Description', TextareaType::class, array(
67                 'attr'=>array('class'=>'form-control') ))
68             ->add('save', SubmitType::class,array(
69                 'label'=>'Insert',
70                 'attr'=>array('class'=>'btn btn-primary') )
71             )
72             ->getForm();
73     return $this->render('job/new.html.twig', ['form'=>$form->createView()]);
74
75 }

```

Donc c'est la même action qui a généré le formulaire qui va traiter sa soumission.

`$form->handleRequest($request)` : Affecter les attributs de l'objet sur lequel le formulaire est créé par les valeurs issues de la requête.

Mais Avant de persister l'objet, on doit vérifier si le formulaire est bien soumis et validé : **`if($form->isSubmitted() && $form->isValid())`**

```

Debug Terminal Help JobController.php - v4 - Visual Studio Code [Administrator]
JobController.php X new.html.twig category.html.twig Job.php JobRepository.php
src > Controller > JobController.php
56  /**
57   * @Route("/new", name="new_job")
58   * Method( { "POST"})
59   */
60   public function new(Request $request){
61       $job = new Job();
62       $form = $this->createFormBuilder($job)
63           ->add('Category', TextType::class, ['attr'=>['class'=>'form-control']])
64           ->add('SousCategory', TextType::class, ['attr'=>['class'=>'form-control']])
65           ->add('Nom', TextType::class, ['attr'=>['class'=>'form-control']])
66           ->add('Description', TextareaType::class, array(
67               'attr'=>array('class'=>'form-control') ))
68           ->add('save', SubmitType::class, array(
69               'label'=>'Insert',
70               'attr'=>array('class'=>'btn btn-primary') )
71       )
72       ->getForm();
73   return $this->render('job/new.html.twig', ['form'=>$form->createView()]);
74
75
76       $form->handleRequest($request);
77       if($form->isSubmitted() && $form->isValid()){
78           $article = $form->getData();
79           $entityManager = $this->getDoctrine()->getManager();
80           $entityManager->persist($article);
81           $entityManager->flush();
82           return $this->redirectToRoute("joblist");
83       }
84   }

```

Job/new.html.twig :

```

Debug Terminal Help new.html.twig - v4 - Visual Studio Code
JobController.php new.html.twig X Job.php JobRepository.php
templates > job > new.html.twig
1  {% extends 'base.html.twig' %}
2
3  {% block title %}New Article{% endblock %}
4
5  {% block body %}
6      {{ form_start(form) }}
7      {{ form_widget(form) }}
8      {{ form_end(form) }}
9  {% endblock %}

```