



TEK-UP Ecole Supérieure Privée Technologie & Ingénierie

Ateliers Framework (Symfony 3)

Wisssem ELJAOUED
wisssem.eljaoued@ensi-uma.tn

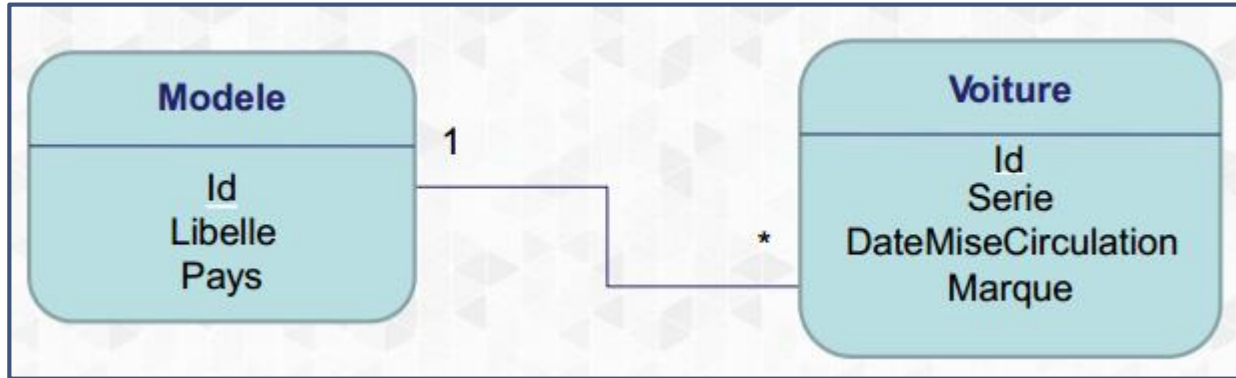
A.U. 2019-2020

Atelier 4

Doctrine

I.1. Etude de cas

- Nous allons travailler pendant ce TP avec l'étude de cas suivant:
- « Parc automobile » représentée avec son diagramme de classe:

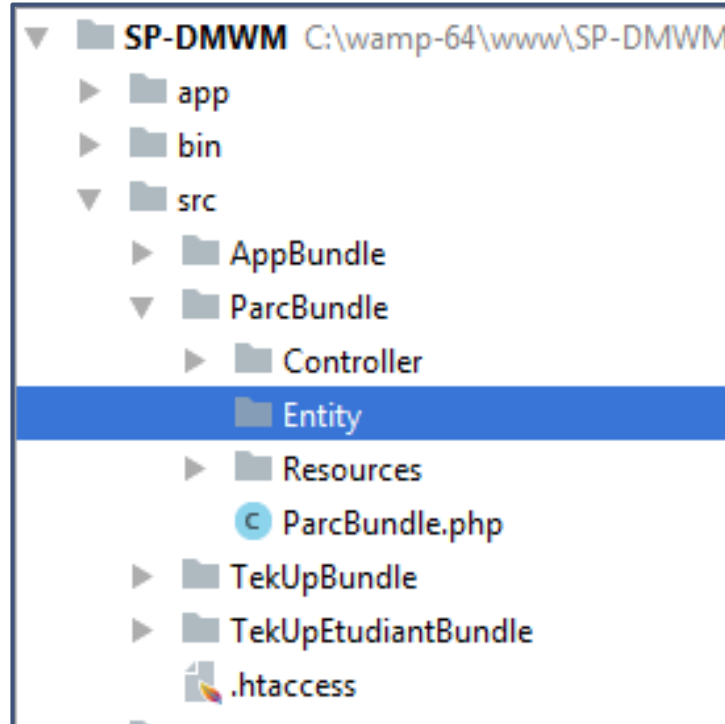


I.2. Création du Bundle

- Créer le Bundle « Parc »
 1. Saisir ensuite : `generate:bundle`
 2. Choisir le `nom du bundle`, le `format du fichier du routing` et le `répertoire source`:
 - Bundle Name: `ParcBundle`
 - Target directory: `src`
 - Configuration format : `yaml`

I.3. Création des entités

- Tout d'abord il faut créer un dossier **Entity** sous le répertoire de notre Bundle (ParcBundle dans notre cas)

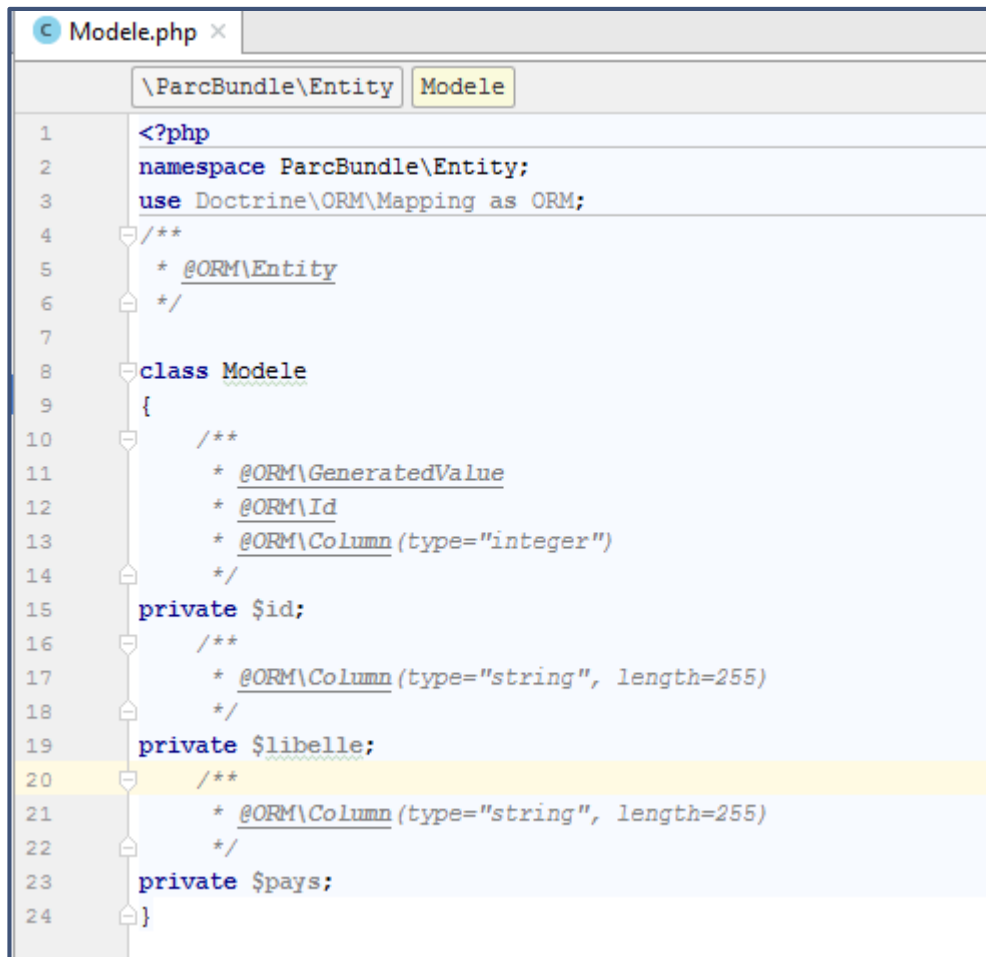


I.3. Création des entités

Maintenant nous devons créer nos **classes** sous le répertoire Entity

- Les attributs sont enrichis avec des **annotations** qui vont aider l'ORM à créer l'équivalent de la classe en table dans la base de données.
- Les annotations vont spécifier les types, les clés,...

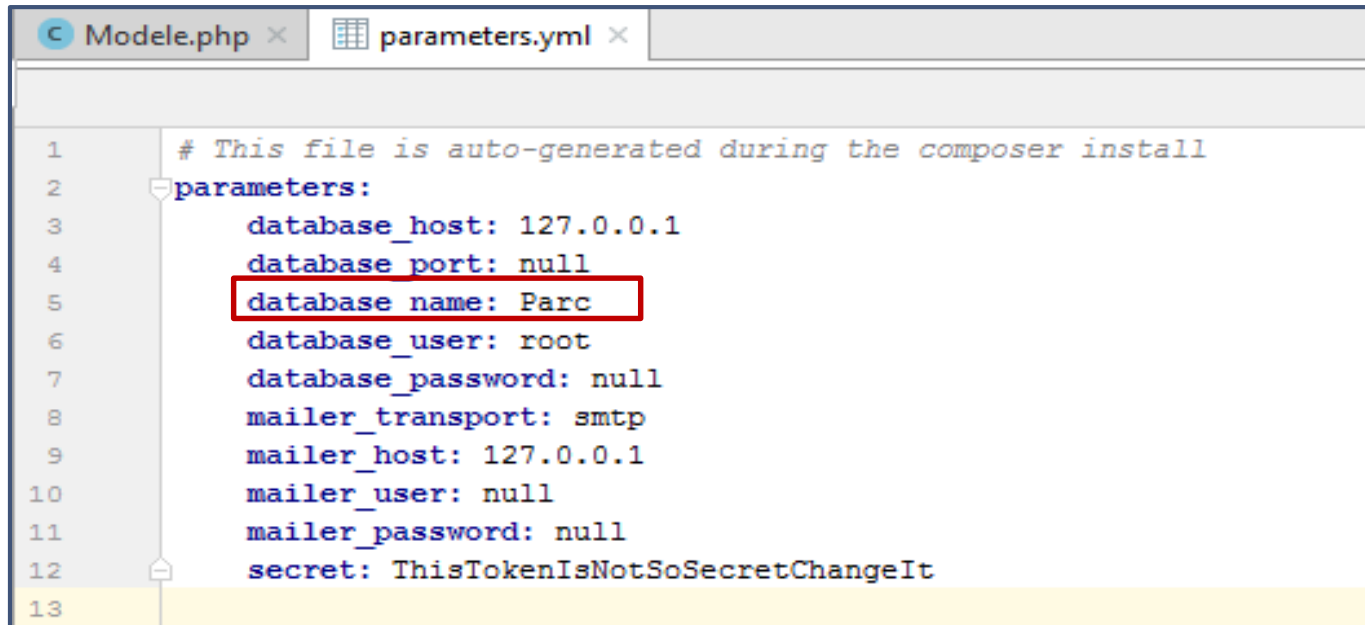
Il faut générer les **getters** et les **setters** de tout les attributs (Menu>Code>Generate...)



```
1 <?php
2 namespace ParcBundle\Entity;
3 use Doctrine\ORM\Mapping as ORM;
4 /**
5  * @ORM\Entity
6  */
7
8 class Modele
9 {
10     /**
11      * @ORM\GeneratedValue
12      * @ORM\Id
13      * @ORM\Column(type="integer")
14      */
15     private $id;
16     /**
17      * @ORM\Column(type="string", length=255)
18      */
19     private $libelle;
20     /**
21      * @ORM\Column(type="string", length=255)
22      */
23     private $pays;
24 }
```

I.4. Configuration BD

- Les accès à la base de données ainsi qu'au serveur de messagerie sont centralisés dans le fichier TpSymfony\app\config\parameters.yml



```
1  # This file is auto-generated during the composer install
2  parameters:
3      database_host: 127.0.0.1
4      database_port: null
5      database name: Parc
6      database_user: root
7      database_password: null
8      mailer_transport: smtp
9      mailer_host: 127.0.0.1
10     mailer_user: null
11     mailer_password: null
12     secret: ThisTokenIsNotSoSecretChangeIt
13
```

I.5. Génération de la BD

- Pour **créer la base de données**, il faut appeler avec l'invite des commander de PhpStorm la fonction suivante:

```
s doctrine:database:create
```

- Pour générer **le schéma de la base de données** il faut appler avec la console la fonction suivante:














```
s doctrine:schema:create
```

- Pour une probable **modification** on peut utiliser la commande :

```
s doctrine:schema:update --force
```


I.5. Génération de la BD

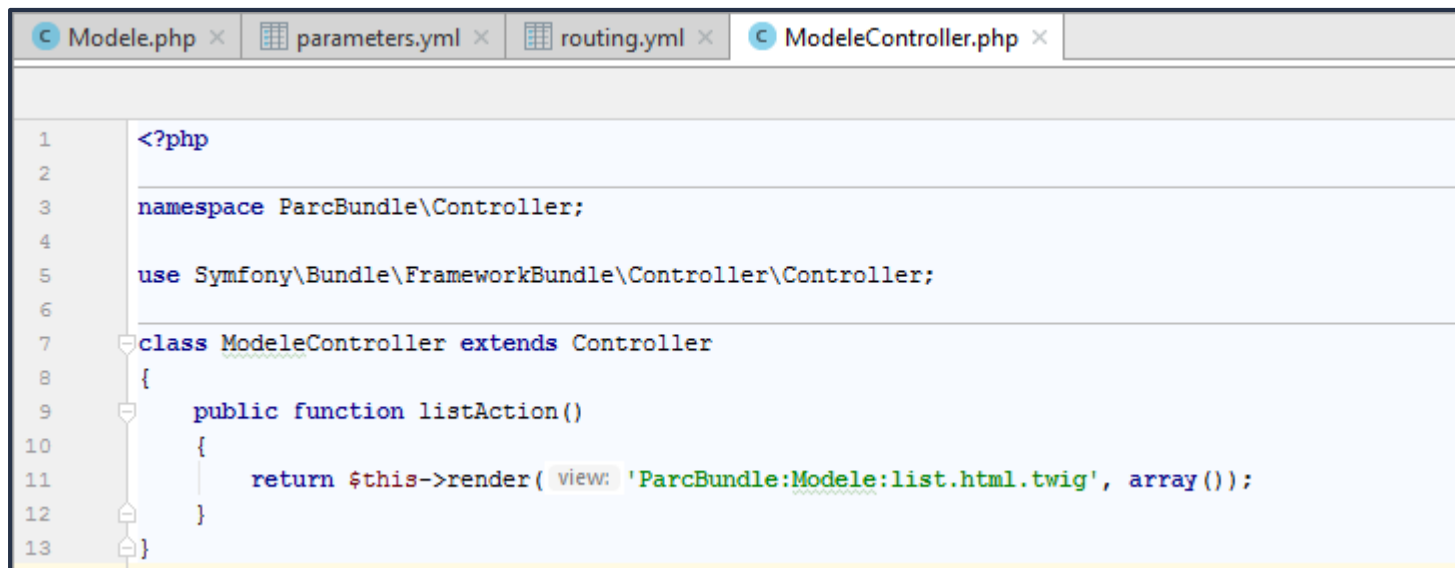
- Ajouter des modèles manuellement dans la table **Modele**

+ Options						
				▼		
				id	libelle	pays
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	1	Mercedes	Allemagne
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	2	Renault	France
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	3	BMW	Allemagne
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	4	Fiat	Italie

- Créer un nouvel élément dans le fichier routing, avec les informations suivantes

```
parc_affichage_modele:  
  path:      /listModele  
  defaults: { _controller: ParcBundle:Modele:list }
```

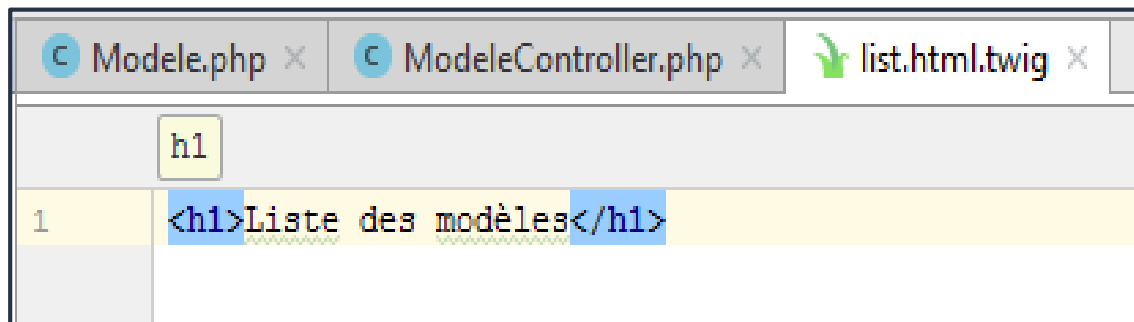
- Créer un contrôleur **ModelController**
- Ajouter l'action **list**



The screenshot shows a code editor with four tabs: 'Modele.php', 'parameters.yml', 'routing.yml', and 'ModelController.php'. The 'ModelController.php' tab is active, displaying the following PHP code:

```
1 <?php
2
3 namespace ParcBundle\Controller;
4
5 use Symfony\Bundle\FrameworkBundle\Controller\Controller;
6
7 class ModelController extends Controller
8 {
9     public function listAction()
10     {
11         return $this->render( view: 'ParcBundle:Modele:list.html.twig', array());
12     }
13 }
```

- Créer un dossier nommé **Modele** sous le répertoire ParcBundle\Ressources\views\
- Créer le fichier **list.html.twig** sous le répertoire Modele




The screenshot shows an IDE window with three tabs: 'Modele.php', 'ModeleController.php', and 'list.html.twig'. The 'list.html.twig' tab is active and displays the following code:

```
1 <h1>Liste des modèles</h1>
```

The code is a single line of Twig template syntax. The opening tag '<h1>' is highlighted in blue, the text 'Liste des modèles' is in yellow, and the closing tag '</h1>' is also highlighted in blue. A yellow box with the text 'h1' is positioned above the opening tag. The line number '1' is visible in the left margin.

- Il faut charger la liste des modèles de la base de données et ceci dans l'action `list`

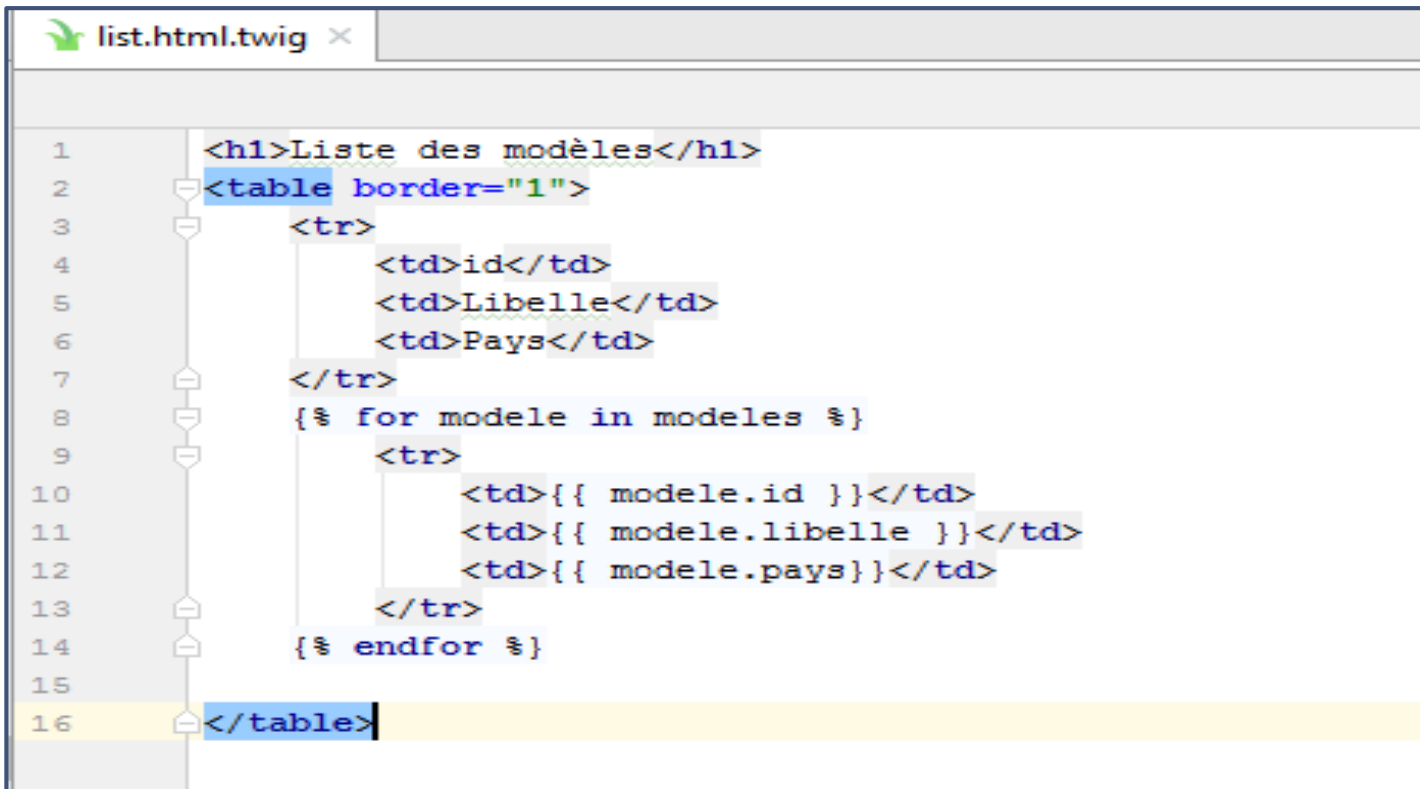


```
1 <?php
2
3 namespace ParcBundle\Controller;
4
5 use Symfony\Bundle\FrameworkBundle\Controller\Controller;
6
7 class ModelController extends Controller
8 {
9     public function listAction()
10     {
11         $em = $this->container->get('doctrine')->getEntityManager();
12         $modeles = $em->getRepository('ParcBundle:Modele')->findAll();
13
14         return $this->render( view: 'ParcBundle:Modele:list.html.twig',
15                             array(
16                                 'modeles'=>$modeles
17                             ));
18     }
19 }
20 }
```

Créer une instance de l'ORM

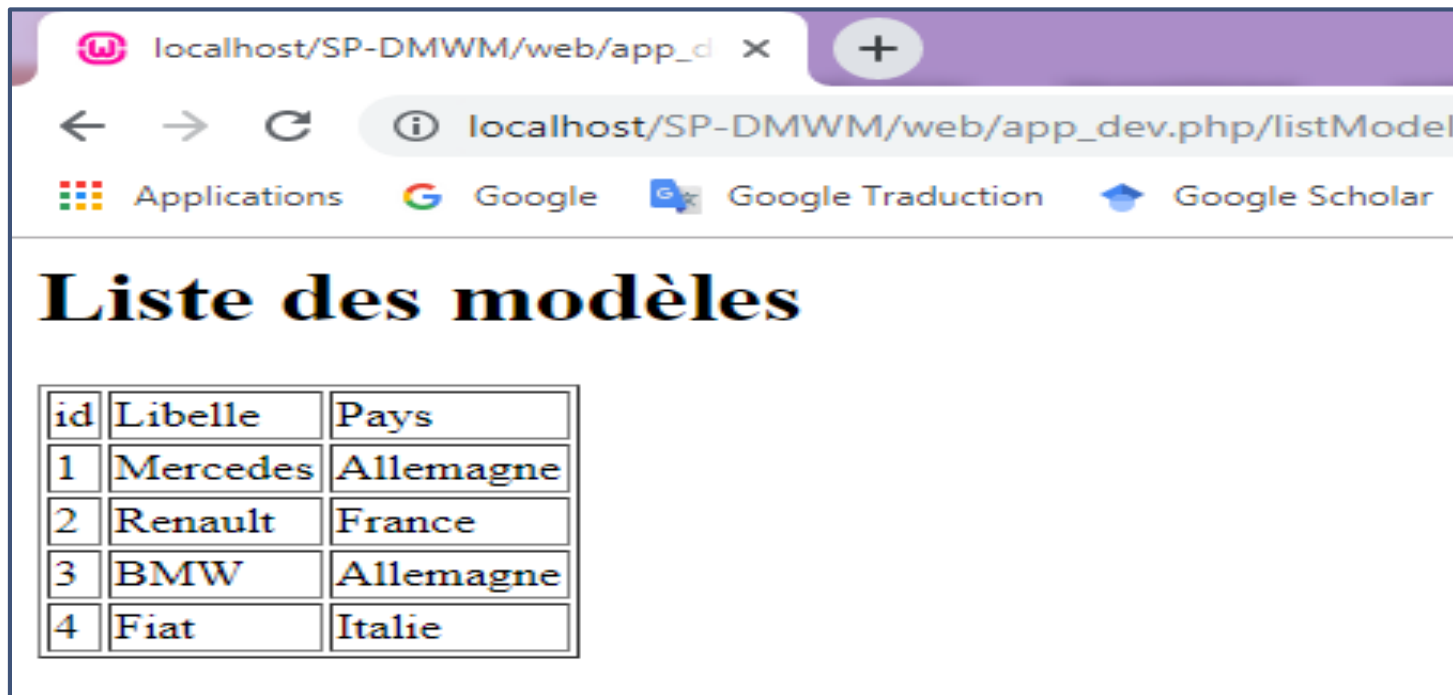
Récupérer la liste de tous les modèles

- Maintenant nous allons personnaliser l'affichage de la liste dans la vue



```
list.html.twig x
1 <h1>Liste des modèles</h1>
2 <table border="1">
3   <tr>
4     <td>id</td>
5     <td>Libelle</td>
6     <td>Pays</td>
7   </tr>
8   {% for modele in modeles %}
9     <tr>
10      <td>{{ modele.id }}</td>
11      <td>{{ modele.libelle }}</td>
12      <td>{{ modele.pays }}</td>
13    </tr>
14  {% endfor %}
15
16 </table>
```

- Affichez la nouvelle page via l'URL suivante :
http://localhost/ProjectName/web/app_dev.php/listModele



The screenshot shows a web browser window with the address bar displaying `localhost/SP-DMWM/web/app_dev.php/listModel`. The page content features a heading **Liste des modèles** and a table with the following data:

id	Libelle	Pays
1	Mercedes	Allemagne
2	Renault	France
3	BMW	Allemagne
4	Fiat	Italie

- Créer une entité Voiture
- Créer un Contrôleur Voiture
- Créer une page pour afficher la liste des voitures

Astuce

- Pour créer un attribut dans une entité qui représente une clé étrangère, il faut lui spécifier l'annotation ci-dessous:

```
/**  
 * @ORM\ManyToOne(targetEntity="Modele")  
 */
```

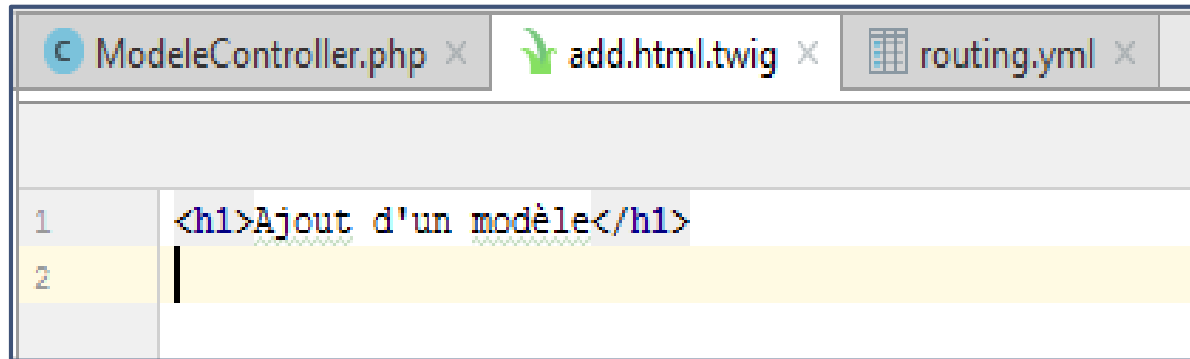

- Créer un nouvel élément dans le fichier routing, avec les informations suivantes:

```
parc_ajout_modele:  
  path:      /addModele  
  defaults: { _controller: ParcBundle:Modele:add }
```

- Créer l'action **add** dans le contrôleur **Modele**

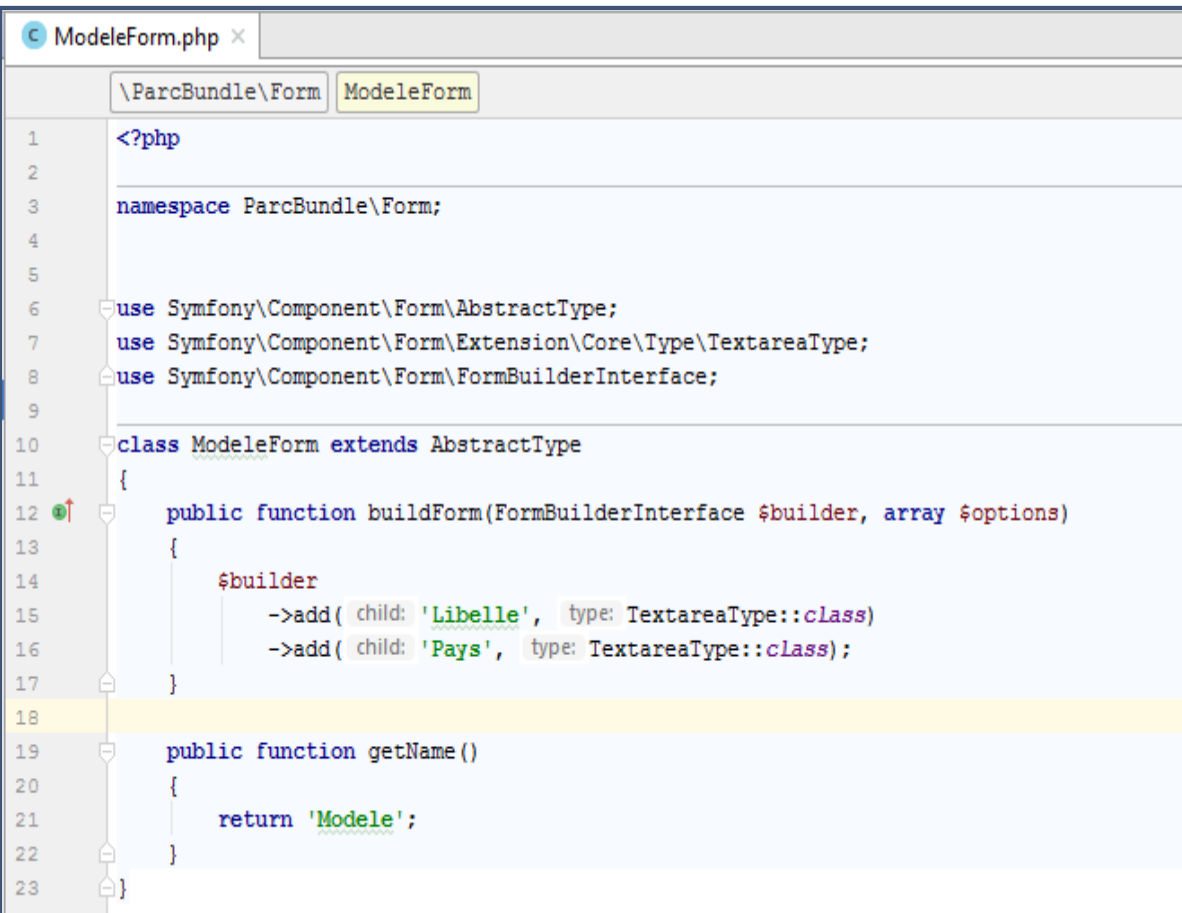
```
//Add modele
public function addAction()
{
    return $this->render( view: '@Parc/Modele/add.html.twig', array());
}
```

- Créer le fichier `add.html.twig` sous le répertoire `Modele`
- Ecrire le code ci-dessous



The screenshot shows a code editor with three tabs: 'ModeleController.php', 'add.html.twig', and 'routing.yml'. The 'add.html.twig' tab is active, displaying the following code:

```
1 <h1>Ajout d'un modèle</h1>
2 |
```



```
1 <?php
2
3 namespace ParcBundle\Form;
4
5
6 use Symfony\Component\Form\AbstractType;
7 use Symfony\Component\Form\Extension\Core\Type\TextareaType;
8 use Symfony\Component\Form\FormBuilderInterface;
9
10 class ModelForm extends AbstractType
11 {
12     public function buildForm(FormBuilderInterface $builder, array $options)
13     {
14         $builder
15             ->add( child: 'Libelle', type: TextareaType::class)
16             ->add( child: 'Pays', type: TextareaType::class);
17     }
18
19     public function getName()
20     {
21         return 'Modele';
22     }
23 }
```

Maintenant nous allons créer la classe formulaire, pour qu'il soit réutilisable la ou on veut l'instancier . Pour cela il faut :

- Créer un dossier Form sous ParcBundle
- Puis créer la classe **ModelForm**

Maintenant nous allons créer une instance du formulaire dans le contrôleur et l'afficher dans la vue

- Créer une instance dans l'action du **contrôleur**:

```
//Add modele
public function addAction()
{
    $Modele = new Modele();
    $form = $this->createForm( type: ModeleForm::class, $Modele);

    return $this->render( view: '@Parc/Modele/add.html.twig',
        array(
            'Form'=>$form->createView()
        ));
}
```

- Modifier la vue `add.html.twig` :

Retour à la même page
lors de la confirmation du
formulaire

```
1 <h1>Ajout d'un modèle</h1>
2
3 <form action="" method="POST">
4     {{ form_widget(Form) }}
5     <input type="submit"/>
6 </form>
```

Affichage du
formulaire

Bouton de confirmation du
formulaire

Maintenant il faut:

- récupérer les informations envoyées par l'utilisateur
- et les insérer dans la base de données.

Pour faire ceci, il faut récupérer les données après le **POST**, créer une instance d'**entity Manager** et **persister** dans la base de données.

```
//Add modele
public function addAction(Request $request)
{
    $Modele = new Modele();
    $form = $this->createForm( type: ModeleForm::class, $Modele);

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid())
    {
        $em= $this->getDoctrine()->getManager();
        $em->persist($Modele);
        $em->flush();
        return $this->redirect($this->generateUrl( route: "parc_affichage_modele"));
    }

    return $this->render( view: '@Parc/Modele/add.html.twig',
        array(
            'Form'=>$form->createView()
        ));
}
```

La variable `$request` contient les valeurs entrées dans le formulaire

Vérifier si la requête viens suite à un submit les données sont valides

Persister les données à travers l'ORM

Rediriger la page vers la liste des modèles suite à l'ajout

- L'action de suppression n'a pas besoin d'une vue, il faut juste créer le routing ainsi que l'action à faire.
- Mais tout d'abord il faut créer le lien vers l'action de suppression d'une ligne particulière. Nous allons créer ce lien dans la vue de l'affichage.
- Créer le routing:

```
parc_supprimer_modele:  
  path:      /deleteModele/{id}  
  defaults: { _controller: ParcBundle:Modele:delete }
```

- Créer le lien vers l'action de suppression et ceci dans la vue de l'affichage

The screenshot shows a code editor with two tabs: 'ModeleController.php' and 'list.html.twig'. The 'list.html.twig' tab is active, displaying a Twig template for a list of models. The template includes a table with columns for 'id', 'Libelle', 'Pays', and 'Supprimer'. A loop iterates over 'modeles' to populate the table. The 'Supprimer' column contains a link to a delete action. Annotations highlight the route and the parameter in the href attribute.

```
1 <h1>Liste des modèles</h1>
2 <table border="1">
3   <tr>
4     <td>id</td>
5     <td>Libelle</td>
6     <td>Pays</td>
7     <td>Supprimer</td>
8   </tr>
9   {% for modele in modeles %}
10    <tr>
11      <td>{{ modele.id }}</td>
12      <td>{{ modele.libelle }}</td>
13      <td>{{ modele.pays }}</td>
14      <td><a href="{{ path('parc supprimer modele', {'id': modele.id}) }}" >Supprimer</a> </td>
15    </tr>
16  {% endfor %}
17
18 </table>
```

Le paramètre à passer

La valeur du paramètre à passer

Lien vers la page de suppression (nom de la route)

- Maintenant il ne reste plus que l'action. Il faut récupérer l'ID passé en paramètre, récupérer l'entité ayant cet ID puis supprimer cette entité.

```
//Delete modele  
public function deleteAction(Request $request, $id)
```

```
{
```

```
    $em= $this->getDoctrine()->getManager();
```

```
    $modele = $em->getRepository( className: 'ParcBundle:Modele')->find($id);
```

```
    if ($modele!==null)
```

```
    {
```

```
        $em->remove($modele);
```

```
        $em->flush();
```

```
    }
```

```
    else
```

```
    {
```

```
        throw new NotFoundException( message: "Le modele d'id".$id."n'existe pas");
```

```
    }
```

```
    return $this->redirectToRoute( route: "parc_affichage_modele");
```

```
}
```

Récupérer l'entité
avec l'id spécifié

Supprimer l'entité
récupérée et valider

Rediriger la page vers
la liste des modèles