



**TEK-UP Ecole Supérieure Privée Technologie &
Ingénierie**

Ateliers Framework (**Symfony 5**)

Wisssem ELJAOUED
wisssem.eljaoued@ensi-uma.tn

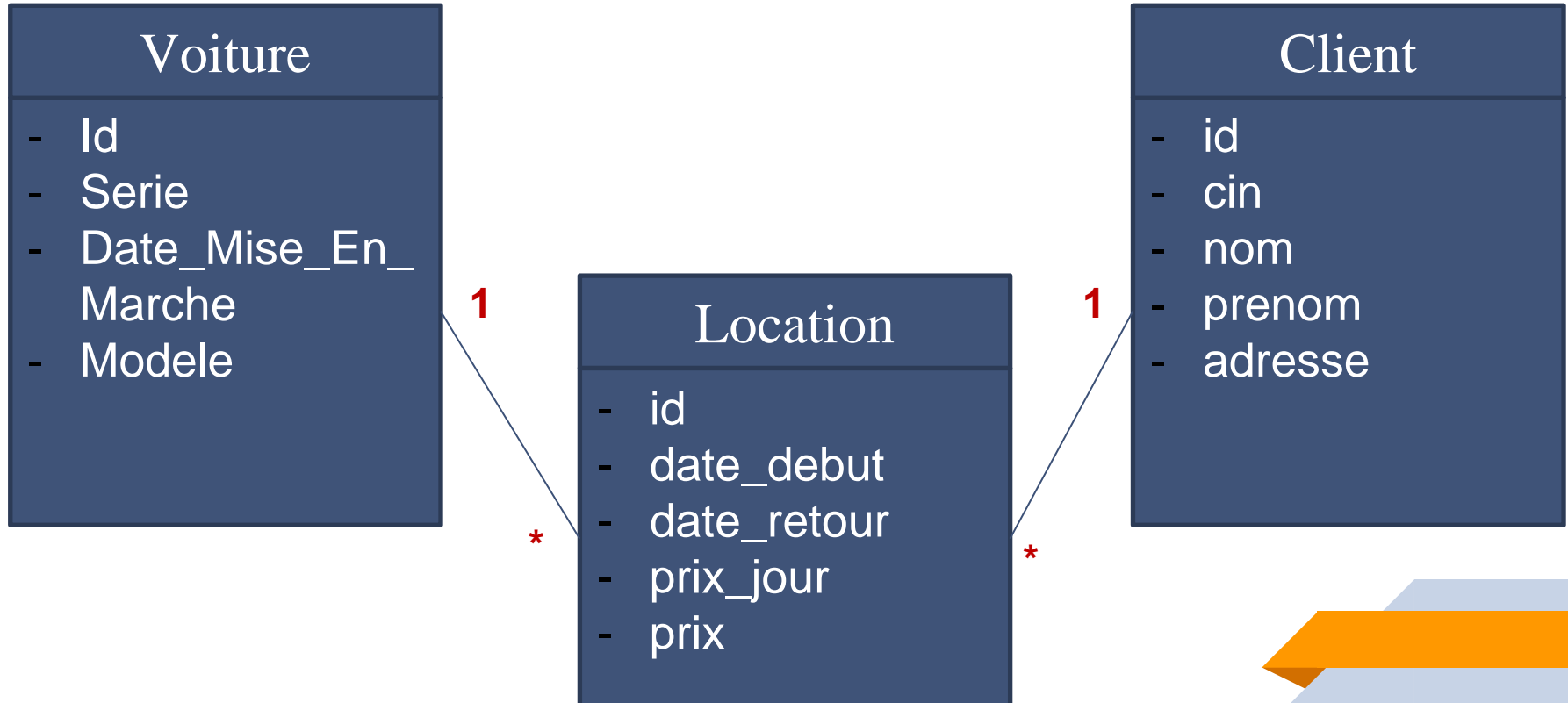
A.U. 2021-2022

Atelier 4

Doctrine

I.1. Etude de cas

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



I.2. 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...)

Deux méthodes de création des entités:

- Manuellement (new PHP Class)
- La commande: **make:entity**

I.3. Configuration BD

- Les accès à la base de données ainsi qu'au serveur de messagerie sont centralisés dans le fichier App\env

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/workshop1?serverVersion=5.7"
```

Serveur de BD

DB user

DB name

I.4. Génération de la BD

- Pour faire le mapping entre les entités et la base de données, nous devons créer des migrations à l'aide de la commande suivante:

`make:migration`

- Deuxièmement, exécuter ces migrations :

`doctrine:migrations:migrate`

En cas d'erreur, changer l'URL de la base de données dans le fichier `.env` :

- en éliminant la version du serveur comme suit:

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/workshop1"
```

- Ou préciser la version exacte de votre serveur:

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/workshop1?serverVersion=mariadb-10.4.14"
```

I.4. Génération de la BD

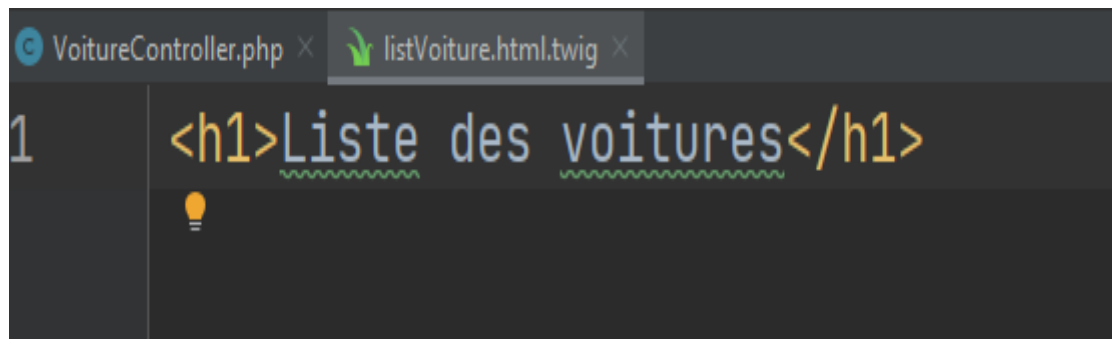
- Ajouter des voitures manuellement dans la table **Voiture**

 ▼				id	serie	date_mise_circulation	modele
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	1	1234	2021-08-10	BMW
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	2	5678	2021-07-12	FORD
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	3	9123	2020-12-08	AUDI

- Créer un contrôleur **VoitureController**
- Ajouter l'action **listVoiture**

```
/**
 * @Route("/voiture", name="voiture")
 */
public function listVoiture(): Response
{
    return $this->render(view: 'voiture/listVoiture.html.twig', [
    ]);
}
```


- Créer un dossier nommé **voiture** sous le répertoire **Templates**
- Créer le fichier **listVoiture.html.twig** sous le répertoire **voiture**



The screenshot shows a code editor with two tabs: 'VoitureController.php' and 'listVoiture.html.twig'. The 'listVoiture.html.twig' tab is active, displaying the following HTML code:

```
1 <h1>Liste des voitures</h1>
```

The code is syntax-highlighted, with the opening and closing tags in blue and the text in orange. A lightbulb icon is visible below the code, indicating a suggestion or tip.

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

```
public function listVoiture(): Response
{
    $em= $this->getDoctrine()->getManager();
    $voitures = $em->getRepository( className: "App\Entity\Voiture")->findAll();

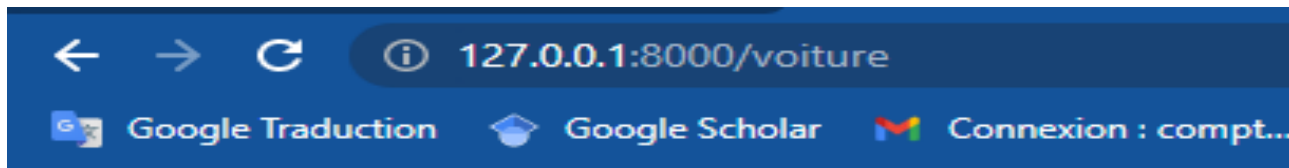
    return $this->render( view: 'voiture/listVoiture.html.twig', [
        "listeVoiture"=>$voitures
    ]);
}
```

Créer une
instance
de l'ORM

Récupérer
la liste de
tous les
voitures

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

```
<h1>Liste des voitures</h1>
<table border="1">
  <tr>
    <td>Serie</td>
    <td>Date M C</td>
    <td>Modele</td>
  </tr>
  {% for v in listeVoiture %}
    <tr>
      <td>{{ v.serie }}</td>
      <td>{{ v.dateMiseCirculation | date }}</td>
      <td>{{ v.modele }}</td>
    </tr>
  {% endfor %}
</table>
```



Liste des voitures

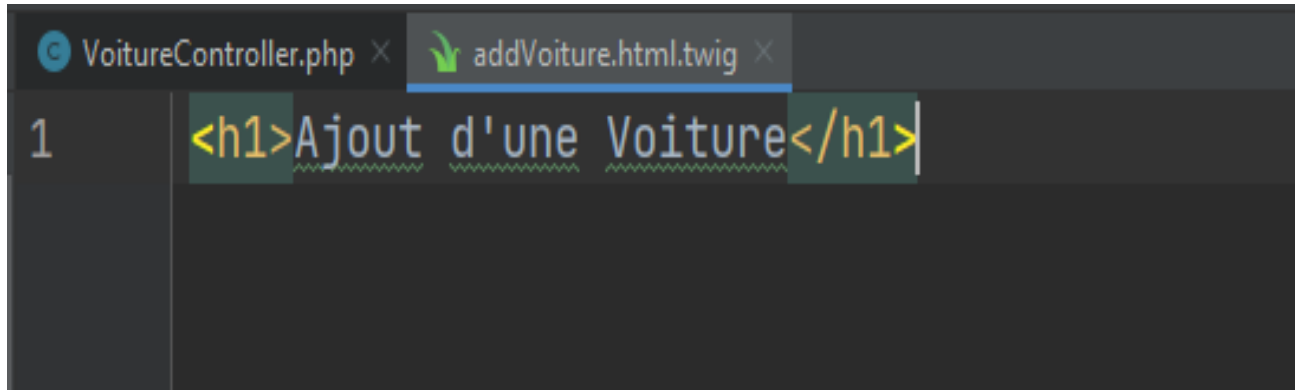
Serie	Date M C	Modele
1234	August 10, 2021 00:00	BMW
5678	July 12, 2021 00:00	FORD
9123	December 8, 2020 00:00	AUDI

- Ajouter les relations entre les entités:
 - Location ManyToOne Voiture
 - Voiture OneToMany Location
 - Location ManyToOne Client
 - Client OneToMany Location

- Créer l'action `addVoiture` dans le contrôleur `Voiture`

```
/**
 * @Route("/addVoiture", name="add_voiture")
 */
public function addVoiture(): Response
{
    return $this->render(view: 'voiture/addVoiture.html.twig', [
    ]);
}
```

- Créer le fichier `addVoiture.html.twig` sous le répertoire `voiture`
- Ecrire le code ci-dessous



The screenshot shows a code editor with two tabs: 'VoitureController.php' and 'addVoiture.html.twig'. The 'addVoiture.html.twig' tab is active and shows a single line of code: `<h1>Ajout d'une Voiture</h1>`. The line number '1' is visible on the left side of the editor.

```
1 <h1>Ajout d'une Voiture</h1>
```

```
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;

class VoitureForm extends AbstractType
{

    public function buildForm(FormBuilderInterface $builder, array $options){
        $builder->add( child: 'serie', type: TextType::class)
            ->add( child: 'date_mise_circulation', type: DateType::class)
            ->add( child: 'modele', type: TextType::class);
    }

    public function getName(){
        return "Voiture";
    }
}
```

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 /src
- Puis créer la classe **VoitureForm**

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**:

```
/**
 * @Route("/addVoiture", name="add_voiture")
 */
public function addVoiture(): Response
{
    $voiture = new Voiture();
    $form = $this->createForm( type: VoitureForm::class, $voiture);

    return $this->render( view: 'voiture/addVoiture.html.twig', [
        'formVoiture'=>$form->createView()
    ]);
}
```

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

Retour à la même page
lors de la confirmation du
formulaire

```
VoitureController.php x addVoiture.html.twig x VoitureForm.php x Voiture.php x
1 <h1>Ajout d'une Voiture</h1>
2
3 <form action="" method="Post">
4     {{ form_widget(formVoiture) }}
5     <input type="submit" value="Add Voiture">
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.

```
/**
 * @Route("/addVoiture", name="add_voiture")
 */
public function addVoiture(Request $request): Response
{
    $voiture = new Voiture();
    $form = $this->createForm( type: VoitureForm::class, $voiture);

    $form->handleRequest($request);

    if($form->isSubmitted() and $form->isValid()){
        $em= $this->getDoctrine()->getManager();
        $em->persist($voiture);
        $em->flush();

        return $this->redirectToRoute( route: 'voiture');
    }

    return $this->render( view: 'voiture/addVoiture.html.twig', [
        'formVoiture'=>$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 voitures suite à l'ajout

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

```
<h1>Liste des voitures</h1>
<table border="1">
  <tr>
    <td>Serie</td>
    <td>Date M C</td>
    <td>Modele</td>
  </tr>
  {% for v in listeVoiture %}
    <tr>
      <td>{{ v.serie }}</td>
      <td>{{ v.dateMiseCirculation | date }}</td>
      <td>{{ v.modele }}</td>
      <td><a href="{% path('voitureDelete', {'id': v.id}) %}">Supprimer</a></td>
    </tr>
  {% endfor %}
</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.

```
/**
 * @Route("/deleteVoiture/{id}", name="voitureDelete")
 */
public function deleteVoiture($id): Response
{
    $em= $this->getDoctrine()->getManager();
    $voiture = $em->getRepository( className: "App\Entity\Voiture")->find($id);

    if($voiture!== null){
        $em->remove($voiture);
        $em->flush();
    }else{
        throw new NotFoundHttpException( message: "La voiture d'id ".$id." n'existe pas");
    }

    return $this->redirectToRoute( route: 'voiture');
}
```

Récupérer l'entité
avec l'id spécifié

Supprimer l'entité
récupérée et valider

Rediriger la page vers
la liste des voitures

- Pour la partie mise à jour elle est identique à celle de l'ajout, sauf qu'il faut seulement charger l'entité sélectionnée lors de l'appel de l'action update.
- Nous allons commencer par créer le lien vers le formulaire de mise à jour dans la view `listVoiture.html.twig`

```
<h1>Liste des voitures</h1>
<table border="1">
  <tr>
    <td>Serie</td>
    <td>Date M C</td>
    <td>Modele</td>
  </tr>
  {% for v in listeVoiture %}
    <tr>
      <td>{{ v.serie }}</td>
      <td>{{ v.dateMiseCirculation | date }}</td>
      <td>{{ v.modele }}</td>
      <td><a href="{{ path('voitureDelete', {'id': v.id}) }}">Supprimer</a></td>
      <td><a href="{{ path('voitureUpdate', {'id': v.id}) }}">MAJ</a></td>
    </tr>
  {% endfor %}
</table>
```



```
/**
 * @Route("/updateVoiture/{id}", name="voitureUpdate")
 */
public function updateVoiture(Request $request, $id): Response
{
    $em= $this->getDoctrine()->getManager();
    $voiture = $em->getRepository( className: "App\Entity\Voiture")->find($id);

    $editform = $this->createForm( type: VoitureForm::class, $voiture);

    $editform->handleRequest($request);

    if($editform->isSubmitted() and $editform->isValid()){

        $em->persist($voiture);
        $em->flush();

        return $this->redirectToRoute( route: 'voiture');
    }

    return $this->render( view: 'voiture/updateVoiture.html.twig', [
        'editFormVoiture'=>$editform->createView()
    ]);
}
```

Charger l'instance
qui correspond à
l'ID en paramètre

```
<h1>MAJ d'une Voiture</h1>

<form action="" method="Post">
  {{ form_widget(editFormVoiture) }}
  <input type="submit" value="MAJ Voiture">
</form>
```