

## Travail dirigé

**IDE imposée : Netbeans**  
**Utilisation de xdebug**

Objectifs du TD :

- ✓ Lire une documentation technique
- ✓ Mettre en œuvre une architecture MVC entièrement avec des classes
- ✓ Utiliser composer
- ✓ **Découvrir twig**

Prérequis : quelques notions de POO.

Avoir fait le TD MonPetitMVC\_partie1

**Ce TP sera effectué sur une version 7.4x de PHP**



**Vous aurez à**

- ✓  **typer tous les paramètres de fonction (et méthodes)**
- ✓  **typer les retours de fonctions (et méthodes)**
- ✓  **travailler en `strict_types=1`**

**Dans le code fourni, il se peut que vous ayez à corriger certaines parties pour satisfaire ces conditions.**

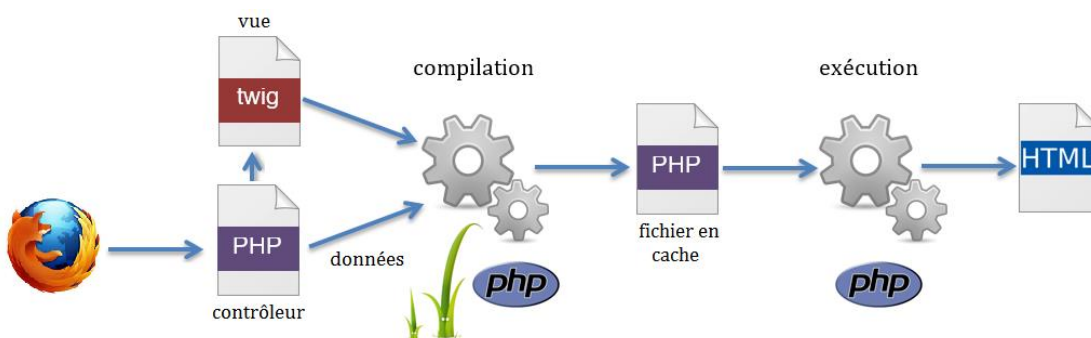
### Partie 1 : INSTALLATION DE TWIG

Twig est un composant qui fait partie de la famille des moteurs de template.

Il permet de dissocier le langage de programmation de l'affichage des données.

On affichera donc des "*vues*" ou "*templates*". Les vues constituent la partie V de la structure MVC, C'est-à-dire que ces templates sont dédiés à l'affichage.

De fait, grâce aux moteurs de template, on sépare la partie code de la partie graphisme. Les développeurs et les graphistes n'ont qu'à s'entendre sur les noms des variables à afficher.



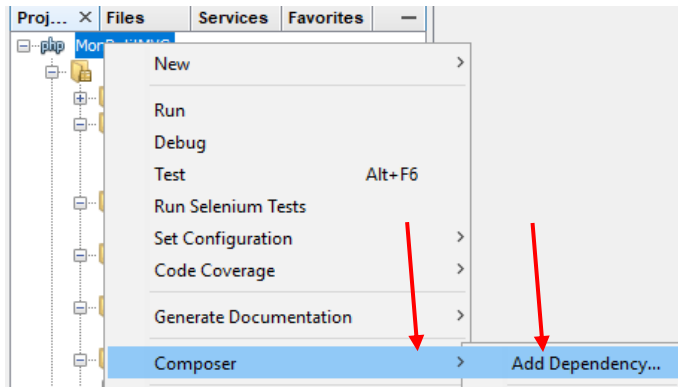
Dans cette partie, vous allez créer et utiliser des templates twig.

Vous utiliserez composer pour charger la bibliothèque Twig.

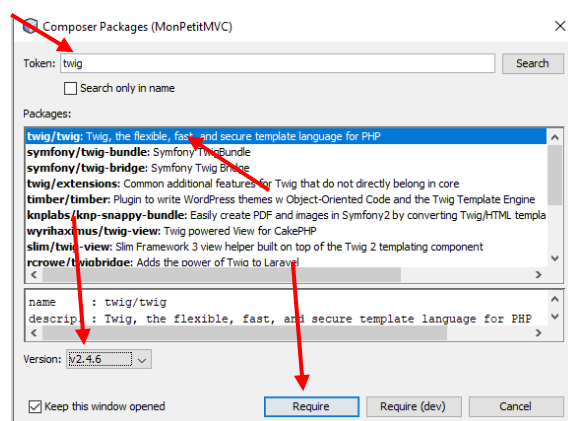
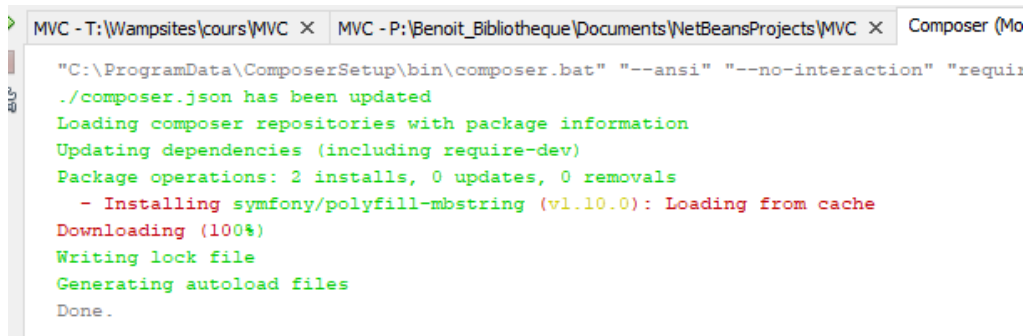
#### 1. Installation de twig

Il vous est proposé d'utiliser la console netbeans pour charger les dépendances, mais vous êtes libres d'utiliser la console powershell ou cmd pour arriver au même résultat.

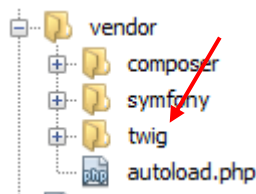
Vous lancerez donc la console composer de netbeans :



puis vous rechercherez twig :



twig s'est installé dans le dossier vendor :



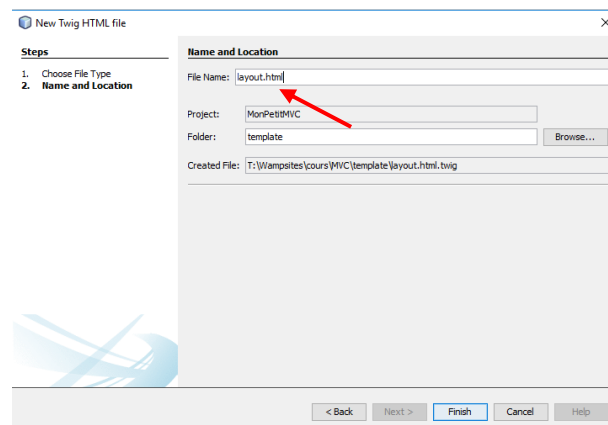
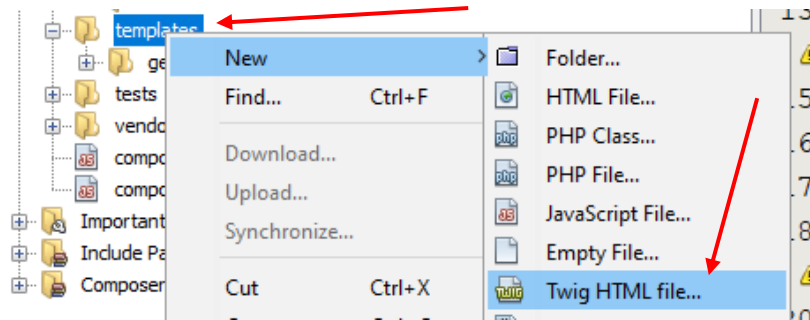
## 2. Création des vues

Vous commencerez par supprimer les fichiers footer.html et header.html du dossier template

Vous allez maintenant créer les vues en utilisant le langage de twig.

L'idée est de créer un squelette de page dans laquelle on créera des blocs twig correspondant à des emplacements dont le contenu sera fourni par la vue. Ce squelette (template ou layout) sera disposé à la racine du dossier template car il servira à toutes les pages de l'application.

Nous appellerons ce squelette layout.html.twig .



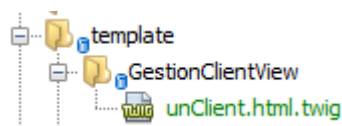
Le contenu sera le suivant :

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ title }}</title>
  </head>
  <body>
    {% block contenu %}{% endblock %}
  </body>
  <footer>
    {% block footer %} <p> Mentions légales </p> {% endblock %}
  </footer>
</html>
```

Ce bloc contiendra le contenu de la page.

Ce bloc contiendra le pied de page.

Vous définirez ensuite le contenu qui correspondra à l'affichage d'un employé :  
Vous créerez le fichier unClient.html.twig dans le dossier templates/gestionClientView :



Et vous y mettrez le contenu suivant :

```
{% extends "layout.html.twig" %}
{% block contenu %}
    <p> <h2> Informations du client numéro {{ unClient.getNoCli() }}</h2></p>
    <p>{{ unClient.getPrenomCli() }} {{ unClient.getNomCli() }}</p>
    <p> {{ unClient.getAdresseRue1Cli() }}
    {% if unClient.getAdresseRue2Cli() %}
        <p> {{ unClient.getAdresseRue2Cli() }}</p>
    {% endif %}
    <p>{{ unClient.getCpCli() }} {{ unClient.getVilleCli() }}</p>
    <p>{{ unClient.getTelCli() }}</p>
{% endblock %}
{% block footer %}
    {{ parent() }}
    Fin de l'affichage
{% endblock %}
```



Vous répondrez aux questions suivantes :

- ✓ Que fait l'instruction suivante :  `{{ unClient.getNoCli() }}`
- ✓ Que fait l'instruction suivante : `{% extends "layout.html.twig" %}`
- ✓ Que fait l'instruction suivante :

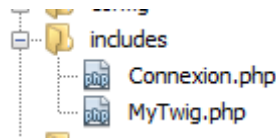
```
    {% if unClient.getAdresseRue2Cli() %}
    <p> {{ unClient.getAdresseRue2Cli() }}</p>
```
- ✓ Que fait la série d'instructions suivante :

```
{% block footer %}
    {{ parent() }}
    Fin de l'affichage
{% endblock %}
```

Pour que twig puisse fonctionner, il faut charger tout son environnement, c'est-à-dire instancier la classe *Twig\_Loader\_Filesystem*. Cette classe fait partie de la bibliothèque twig chargée par composer. Elle permettra de faire le lien entre la vue et les variables envoyées par PHP et de créer le fichier html.

Travaillant en programmation objet, vous allez créer une classe MyTwig qui ne sera pas instanciable et qui contiendra une méthode statique publique d'affichage des vues en plus de la méthode statique qui renverra l'environnement.

Cette classe sera dans le dossier includes :



Cette classe sera définie obligatoirement dans l'espace de noms Tools.

Vous créerez 2 méthodes :

- ✓ La méthode privée `getLoader()` qui aura pour mission de renvoyer l'environnement twig
- ✓ La méthode publique `afficheVue($vue, $param)` qui aura pour mission d'appeler la vue passée en paramètre avec les données passées dans le tableau de paramètre.

Le code de votre classe sera le suivant :

```
<?php
namespace Tools;

abstract class MyTwig {

    private static function getLoader() {

        $loader = new \Twig_Loader_Filesystem(PATH_VIEW); // Dossier contenant les templates
        // pas de cache en mode debug
        return new \Twig_Environment($loader, array(
            'cache' => false
        ));
    }

    public static function afficheVue($vue, $params) {
        $twig = self::getLoader();

        $template = $twig->loadTemplate($vue);
        echo $template->render($params);
    }
}
```



Vous répondrez aux questions suivantes :

- ✓ Que fait l'instruction suivante :

```
$loader = new \Twig_Loader_Filesystem(PATH_VIEWS);
```

- ✓ A quoi correspond cette option :

```
'cache' => false
```

- ✓ Justifiez la présence de l'antislash suivant :

```
new \Twig_Environment
```

- ✓ Que fait l'instruction suivante :

```
echo $template->render($params);
```

Il ne nous reste plus qu'à modifier le code de la méthode `GestionClientControleur::chercheUn`

Les modifications à apporter sont relativement simples, elles consisteront à appeler la méthode `MyTwig::afficheVue` avec les bons paramètres :

```

$unClient = $modele->find($id);
if ($unClient) {
    $r = new ReflectionClass($this);
    $vue = str_replace('Controller', 'View', $r->getShortName()) . "/unClient.html.twig";
    MyTwig::afficheVue($vue, array('unClient' => $unClient));
}

```



Vous répondrez à la question suivante :

✓ Pourquoi le test suivant : `if ($unClient) {`

Essayez l'URL suivante : <http://monpetitmvc/?c=GestionClient&a=chercheUn&id=1>

Vous devriez obtenir ceci :



Essayez maintenant avec les URL :

<http://monpetitmvc/?c=GestionClient&a=chercheUn&id=2>

<http://monpetitmvc/?c=GestionClient&a=chercheUn&id=1234>

## Partie 2 : TRAVAIL A FAIRE

### 1. Affichage de tous les clients

En vous inspirant de ce que vous venez de faire, vous afficherez la liste des clients grâce à l'URL suivante : <http://monpetitmvc/?c=GestionClient&a=chercheTous>

Vous aurez à créer

- ✓ la méthode `chercheTous()` de la classe `GestionClientController`
- ✓ la méthode `findAll()` de la classe `GestionClientModel`
- ✓ la vue twig `tousClients.html.twig`

monpetitmvc/?c=GestionClient&a= X +

monpetitmvc/?c=GestionClient&a=chercheTous

## Liste des clients

num	Nom	Prenom	adresse	complement	cp	ville	tel
	Tienun	Jean	112, rue du Départ		13000	Marseille	0404040404
	Terrature	Julie	Résidence Mermoz	1234 Boulevard des Aviateurs	14000	Caen	0202020202
	Cerf	Paulette	343 Avenue Henri Barbusse		33000	Bordeaux	0550505050
	Morizet	Patricia	Hameau de Pau	23 Boulevard du Lycée	33000	Bordeaux	0250505052
	Nolapin	Jean	12 quai des Brumes		83000	Toulon	0404505050
	Entete	Martel	Résidence du Faron	30 rue du téléphérique	83000	Toulon	0250505050
	Entete	Martel	12 Avenue de Lille		59140	Dunkerque	0250905057
	DUMANS	Henriette	Corniche des Bolides	Villa Ferrari	49000	Angers	0250765357

Mentions légales

Nombre de clients : 8



Vous ne calculerez pas le nombre de clients ... vous l'afficherez !!!

## 2. Création d'un client

Vous ferez un formulaire de création d'un client qui s'affichera avec l'url :

<http://monpetitmvc/?c=GestionClient&a=creerClient>

monpetitmvc/?c=GestionClient&a= X +

monpetitmvc/?c=GestionClient&a=creerClient

**Nouveau client**

Titre :  Prénom :  Adresse :  Complément d'adresse :  Code postal :  Ville :  Ville :

Mentions légales

Vous enregistrerez le client saisi dans la BD.

Vous aurez donc à créer :

- ✓ la méthode `creerClient($params)` de la classe `GestionClientController` qui affichera le formulaire ci-dessus
- ✓ la méthode `enregistreClient($client)` de la classe `GestionClientModel` qui se charge de faire l'insertion d'un client dans la BD à partir de l'objet de la classe `Client` passé en paramètre.
- ✓ la vue `creerClient.html.twig` qui permet de saisir un client (formulaire ci-dessus)
- ✓ la méthode `enregistreClient($params)` de la classe `GestionClientController` qui se charge de créer un objet de la classe `client` à partir des paramètres reçus (`$params`) et qui appelle la méthode `enregistreClient($client)` de la classe `GestionClientModel`

## 3. Solutions

a) Méthode creerClient(\$params) de la classe GestionClientController

```
public function creerClient($params) {
    $vue = "GestionClientView\\creerClient.html.twig";
    MyTwig::afficheVue($vue, array());
}
```

b) Méthode enregistreClient(\$client) de la classe GestionClientModel

```
public function enregistreClient($client) {
    $unObjetPdo = Connexion::getConnexion();
    $sql = "insert into client(titreCli, nomCli, prenomCli, adresseRue1Cli, adresseRue2Cli, cpCli, villeCli, telCli) "
        . "values (:titreCli, :nomCli, :prenomCli, :adresseRue1Cli, :adresseRue2Cli, :cpCli, :villeCli, :telCli)";
    $s = $unObjetPdo->prepare($sql);
    $s->bindValue(':titreCli', $client->getTitreCli(), PDO::PARAM_STR);
    $s->bindValue(':nomCli', $client->getNomCli(), PDO::PARAM_STR);
    $s->bindValue(':prenomCli', $client->getPrenomCli(), PDO::PARAM_STR);
    $s->bindValue(':adresseRue1Cli', $client->getAdresseRue1Cli(), PDO::PARAM_STR);
    $s->bindValue(':adresseRue2Cli', ($client->getAdresseRue2Cli() == "") ? (null) : ($client->getAdresseRue2Cli()), PDO::PARAM_STR);
    $s->bindValue(':cpCli', $client->getCpCli(), PDO::PARAM_STR);
    $s->bindValue(':villeCli', $client->getVilleCli(), PDO::PARAM_STR);
    $s->bindValue(':telCli', $client->getTelCli(), PDO::PARAM_STR);
    $s->execute();
    //return Connexion::insereTable("Client2", $client);
}
```

c) Formulaire de saisie d'un client : creerClient.html.twig :

```
{% extends "layout.html.twig" %}
{% block contenu %}

<div class="Container">
    <h2>Nouveau client</h2>
    <form action="index.php?c=GestionClientsa=creerClient" method="post">
        <div class="form-group">
            <label for="titre">Titre</label>
            <select class="form-control" id="titre" name="titreCli">
                <option>Choisir...</option>
                <option>Monsieur</option>
                <option>Madame</option>
                <option>Mademoiselle</option>
            </select>
        </div>
        <div class="form-group">
            <label for="nom">Nom</label>
            <input type="text" class="form-control" id="nom" placeholder="Nom :" name="nomCli" value="Roche">
            <label for="prenom">Prénom</label>
            <input type="text" class="form-control" id="prenom" placeholder="Prénom" name="prenomCli" value="Benoit">
            <label for="adresse1">Adresse</label>
            <input type="text" class="form-control" id="adresse1" placeholder="Adresse" name="adresseRue1Cli" value="Les Lauriers">
            <label for="adresse2">Complément d'adresse</label>
            <a href="creerClient.html.twig"></a>
            <input type="text" class="form-control" id="adresse2" placeholder="Complément d'adresse (facultatif)" name="adresseRue2Cli" value="Avenue W. Churchill">
            <label for="CP">Code postal</label>
            <input type="text" class="form-control" id="cp" placeholder="Code postal" name="cpCli" value="83000">
            <label for="ville">Ville</label>
            <input type="text" class="form-control" id="ville" placeholder="Ville" name="villeCli" value="Toulon">
            <label for="tel">Téléphone</label>
            <input type="tel" class="form-control" id="tel" placeholder="tel" name="telCli" value="0404040404">
        </div>
        <button type="submit" class="btn btn-default">Créer</button>
    </form>
</div>

{% endblock %}
{% block footer %}
    {{ parent() }}
{% endblock %}
```

d) méthode enregistreClient(\$params) de la classe GestionClientController



```

public function enregistreClient($params) {
    // création de l'objet client
    $client = new Client($params);
    $modele = new GestionClientModel();
    $modele->enregistreClient($client);
}

```

### Partie 3 : UTILISATION DE BOOTSTRAP

Dans cette partie, vous allez améliorer sensiblement les IHM en utilisant bootstrap en mode CDN.



Vous répondrez à la question suivante :

- ✓ Que signifie "en mode CDN" ?

L'idée est d'arriver à ce type d'interface :

Vous allez commencer à modifier votre fichier layout.html.twig pour rajouter les lignes suivantes :

```

meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeR1dGMjJkqO9UAHJgRCz3cgU4W7XRnKU6q4mc18p2ztr5WZVqFtS42" crossorigin="anonymous">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-T8CCHvNmYNiNOK7p7Df/YO3+v8hp69dFt/qwGZLLQ9NTSG68DP48I2656vrMB3pw" crossorigin="anonymous"></script>

```

Vous pourrez en faire un copier/coller ici :

[https://www.w3schools.com/bootstrap/tryit.asp?filename=trybs\\_navbar\\_inverse&stacked=h](https://www.w3schools.com/bootstrap/tryit.asp?filename=trybs_navbar_inverse&stacked=h)

Et vous ajouterez un nouveau bloc de navigation. Vous pouvez le récupérer et le transformer à partir du lien ci-dessus.



Tous les éléments HTML que vous trouverez sur cette partie sont issus du site

<https://www.w3schools.com/bootstrap/>

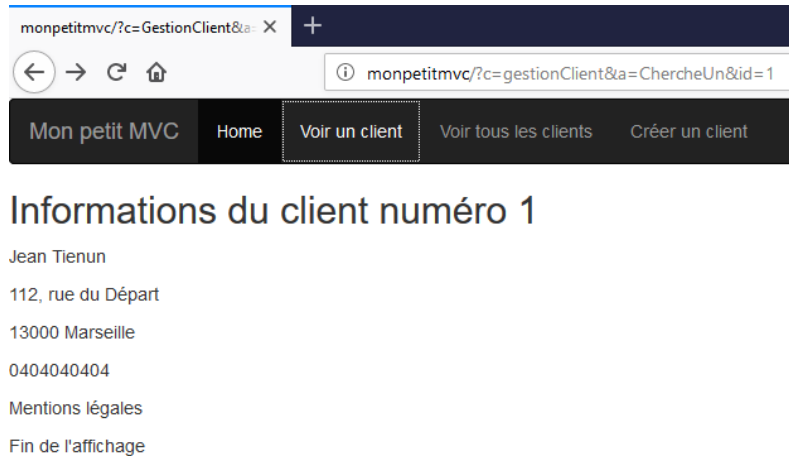
```

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-T8CCHvNmYNiNOK7p7Df/YO3+v8hp69dFt/qwGZLLQ9NTSG68DP48I2656vrMB3pw" crossorigin="anonymous"></script>
{% block navigation %}
<nav class="navbar navbar-inverse">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand" href="#">Mon petit MVC</a>
    </div>
    <ul class="nav navbar-nav">
      <li class="active"><a href="#">Home</a></li>
      <li><a href="/?c=gestionClient&a=ChercheUn">Voir un client</a></li>
      <li><a href="/?c=gestionClient&a=ChercheTous">Voir tous les clients</a></li>
      <li><a href="/?c=gestionClient&a=creerClient">Créer un client</a></li>
    </ul>
  </div>
</nav>
{% endblock %}
</head>

```

Testez : <http://monpetitmvc/?c=gestionClient&a=ChercheUn&id=1>

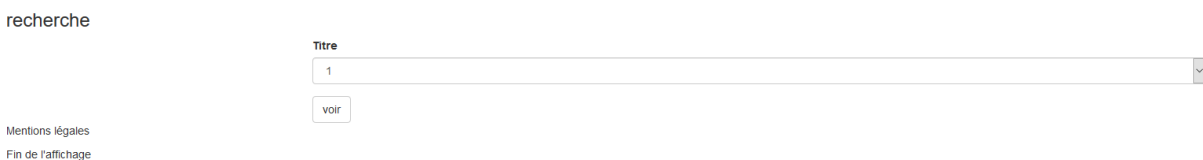
Vous devriez obtenir ceci :



On va faire beaucoup mieux ....

Si on ne précise pas l'id cherché dans l'URL, on va afficher une liste déroulante de tous les ID pour que l'utilisateur puisse sélectionner l'id désiré :

<http://monpetitmvc/?c=gestionClient&a=ChercheUn>



Pour réaliser ceci, il faudra :

- ✓ Créer une méthode findIds dans la classe GestionClientModel
- ✓ Modifier la méthode chercheUn du contrôleur GestionClientController
- ✓ Modifier la vue unClient.html.twig

e) Méthode findIds dans la classe GestionClientModel

Cette méthode récupérera les Id des clients de la table Client et les retournera sous la forme d'un tableau d'entiers

```
public function findIds() {
    $unObjetPdo = Connexion::getConnection();
    $sql = "select id from CLIENT";
    $lignes = $unObjetPdo->query($sql);
    // on va configurer le mode objet pour la lisibilité du code
    if ($lignes->rowCount() > 0) {
        // $lignes->setFetchMode();
        $t = $lignes->fetchAll(PDO::FETCH_ASSOC);
        return $t;
    } else {
        throw new Exception('Aucun client trouvé');
    }
}
```

f) Méthode chercheUn du contrôleur GestionClientController

La modification va consister à

- ✓ Récupérer dans tous les cas la liste des Ids des clients pour remplir la liste déroulante qui

sera toujours affichée

- ✓ Si l'id est passé dans l'URL à aller chercher le client voulu

```
public function chercheUn($params) {
    $modele = new GestionClientModel();
    // dans tous les cas on récupère les Ids des clients
    $sids = $modele->findIds();
    // on place ces Ids dans le tableau de paramètres que l'on va envoyer à la vue
    $params['lesId']=$sids;
    // on teste si l'id du client à chercher a été passé dans l'URL
    if (array_key_exists('id', $params)) {
        $id = filter_var(intval($params['id']), FILTER_VALIDATE_INT);
        $unClient = $modele->find($id);
        // on place le client trouvé dans le tableau de paramètres que l'on va envoyer à la vue
        $params['unClient']=$unClient;
    }
    $r = new ReflectionClass($this);
    $vue = str_replace('Controller', 'View', $r->getShortName()) . "/unClient.html.twig";
    MyTwig::afficheVue($vue, $params);
    } else {
        throw new Exception("Client " . $id . " inconnu");
    }
}
```

g) vue unClient.html.twig

La modification va consister à

- ✓ Créer un formulaire qui servira à afficher la liste des Ids des clients et un bouton voir pour lancer la recherche.

Id du client

Ce formulaire sera placé au début du bloc contenu:

```
{% block contenu %}
    {% block recherche %}
        <h3>recherche</h3>
        <div class="container">
            <form action="?c=GestionClient&a=chercheUn" method="post">
                <div class="form-group">
                    <label for="Numéro du client">Id du client</label>
                    <select class="form-control" id="titre" name="id">
                        {% for id in lesId %}
                            <option>{{ id['id'] }}</option>
                        {% endfor %}
                    </select>
                </div>
                <input type="submit" value="Voir" class="btn btn-default"/>
            </form>
        </div>
    {% endblock %}
{% endblock %}
```

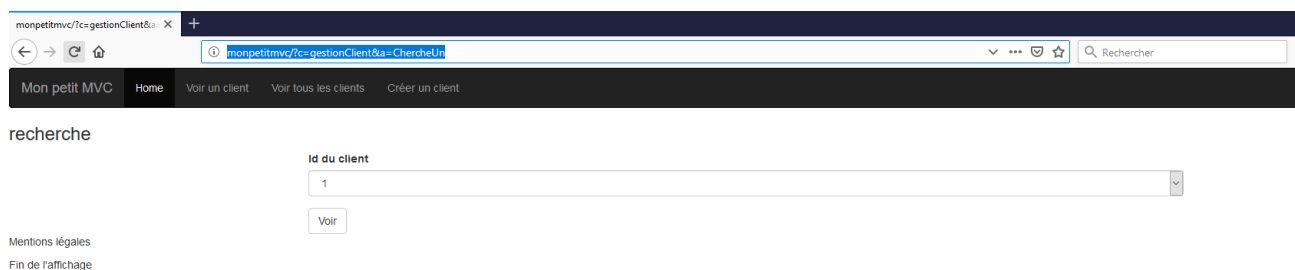
A la suite de ce block, vous allez créer le block affiche qui ne s'affichera que si un client a été

```
        </form>
    </div>
{% endblock %}
{% if unClient %}
    {% block affiche %}
        <p> <h2> Informations du client numéro !!! {{ unClient.getId() }} </h2></p>
        <p>{{ unClient.getPrenomCli() }} {{ unClient.getNomCli() }}</p>
        <p> {{ unClient.getAdresseRue1Cli() }}
        {% if unClient.getAdresseRue2Cli() %}
            <p> {{ unClient.getAdresseRue2Cli() }}</p>
        {% endif %}
        <p>{{ unClient.getCpCli() }} {{ unClient.getVilleCli() }}</p>
        <p>{{ unClient.getTelCli() }}</p>
    {% endblock %}
{% endif %}
{% endblock %}
{% block footer %}
```

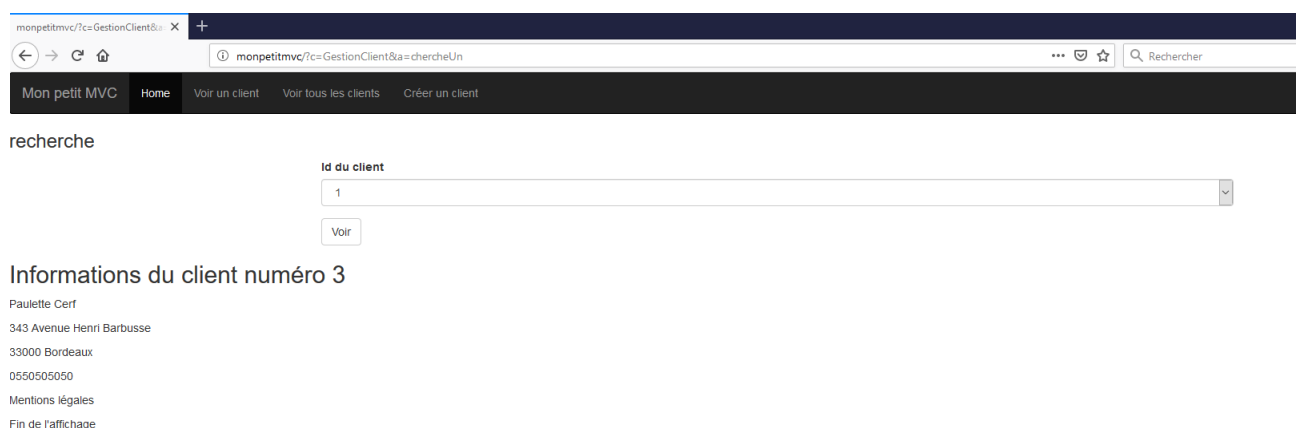
passé dans les paramètres de la vue.

Il ne vous reste plus qu'à tester l'URL : <http://monpetitmvc/?c=gestionClient&a=ChercheUn>

Vous obtiendrez :



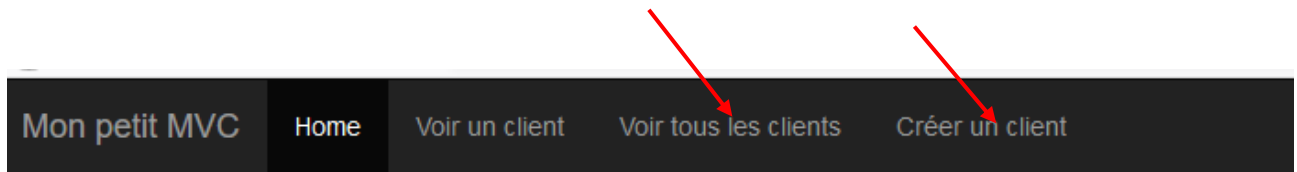
Il ne vous reste plus qu'à choisir un Id et à cliquer sur le bouton *voir*



Vous remarquerez que l'Id du client recherché a disparu de l'Id.  
Pourquoi ?  
Comment peut-on toutefois afficher le bon client ?

#### 4. Travail à faire

Testez vos liens



#### Partie 4 : LA SUITE ....

---

Nous respectons maintenant bien l'architecture MVC, mais on peut encore améliorer bien des choses....

- ✓ Améliorer la navigation
- ✓ On risque de recopier souvent dans les couches Model les méthodes find et findAll

C'est ce que vous améliorerez dans le prochain TP

