

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

Prérequis : maîtrise de la POO.

Code fourni :

- ✓ Classe Client
- ✓ Classe Commande
- ✓ Classe Personnel
- ✓ Classe Connexion

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 : MISE EN PLACE DU PROJET

1. Vous allez commencer à créer le projet MonPetitMVC

A screenshot of the NetBeans Project Wizard dialog. The 'Project Name' field contains 'MonPetitMVC'. The 'Sources Folder' field contains 'T:\Wampsites\cours\MVC' and has a 'Browse...' button next to it.

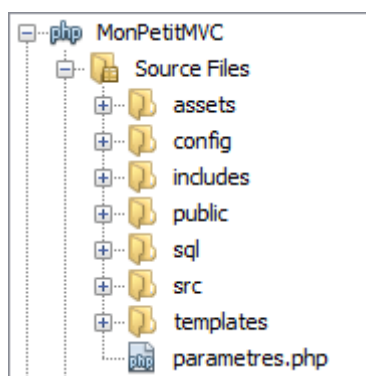
L'URL qui fera marcher l'application sera <http://monpetitmvc>

A screenshot of the NetBeans Run Configuration dialog. The 'Run As' dropdown menu is set to 'Local Web Site (running on local web server)'. The 'Project URL' field contains 'http://monpetitmvc'.

2. Architecture de l'application

Vous allez commencer par créer les dossiers suivants dans votre projet.

Explication des dossiers :



Voyons en détails :

	<p>assets : Tous les fichiers interagissant sur le front de votre appli. Vous aurez donc les dossier js et css notamment</p>
	<p>Config : Tous vos fichiers de configuration. Pour l'instant on gardera les fichiers conf.php et define.php</p>
	<p>includes : tous les fichiers externes à l'application, dont l'application a toutefois besoin. On y placera donc le fichier Connexion.php qui contient la classe Connexion.</p>
	<p>public : la partie publique de votre application. On y mettra le moins de fichiers possible. Pour l'instant on n'y trouvera que le fichier index.php (point d'entrée routeur de l'application). On devra donc modifier le virtualHost précédemment créé.</p>
	<p>Sql : facultatif, permet de stocker les scripts sql. Vous y placerez donc le script de création de la base de données.</p>
	<p>Src : les sources de votre application. Vous aurez donc les dossiers</p> <ul style="list-style-type: none"> ✓ Controller ✓ entities ✓ Model ✓ Repository
	<p>Templates : Contendra toutes les vues stockées logiquement. Errors : si une erreur se produit dans un contrôleur, on déroutera vers une page pour afficher le message. Solution temporaire</p>
Parametres.php	Fichier contenant la déclaration des paramètres de nécessaires à l'application

3. Création du virtualHost du document root de l'application

Vous créez le virtualHost monpetitmvc :

Nom du Virtual Host Pas de caractères diacritiques (éçëñ) - Pas d'espace - Pas de tiret bas ()

monpetitmvc

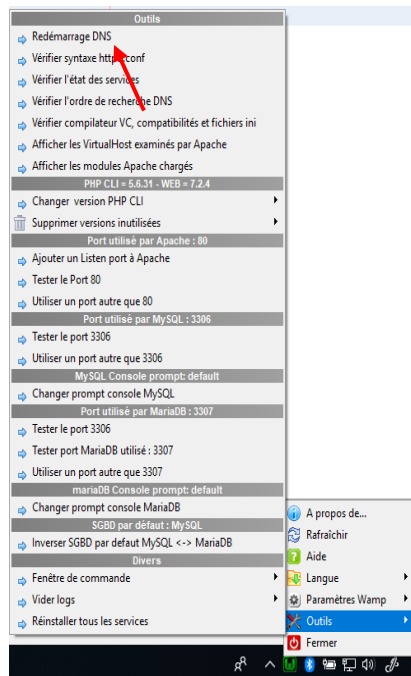
Chemin complet absolu du dossier VirtualHost - Exemples : C:/wamp/www/projet/ ou E:/w

T:\Wampsites\cours\MVC\public\



Attention, il pointe sur le dossier Public situé à la racine du site

Vous n'oublierez pas de redémarrer le DNS de votre serveur web. Sous Wamp



Si vous essayez de relancer votre application, vous devriez ouvrir une page qui contient des erreurs liées à l'autochargement des classes.



Warning: require(\\.\vendor\autoload.php): failed to open stream: No such file or directory in T:\Wampsites\cours\MVC\public\index.php on line 5

Call Stack				
#	Time	Memory	Function	Location
1	0.0020		400120 {main}()	...index.php:0

Fatal error: require(): Failed opening required '\\.\vendor\autoload.php' (include_path='.:C:\php\pear') in T:\Wampsites\cours\MVC\public\index.php on line 5

Call Stack				
#	Time	Memory	Function	Location
1	0.0020		400120 {main}()	...index.php:0



C'est normal, on n'a pas encore travaillé avec composer

4. URL's de l'application

Toutes les URL's seront constituées ainsi :

<http://monpetitmvc/?c=xxxxx&a=yyyyyy>

où :

- ✓ le paramètre **c** représentera le nom du contrôleur à appeler
- ✓ le paramètre **a** représente la méthode du contrôleur à exécuter.

Ainsi l'URL :

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

appellera la méthode publique *chercheTous* de la classe *GestionClientController*

ou encore :

<http://monpetitmvc/?c=GestionClient&a=chercheUn&id=1>

appellera la méthode publique *chercheUn* de la classe *GestionClientController* qui accepte un paramètre nommé id.

Cette méthode recherchera donc le client d'id 1

5. Mise en place de composer

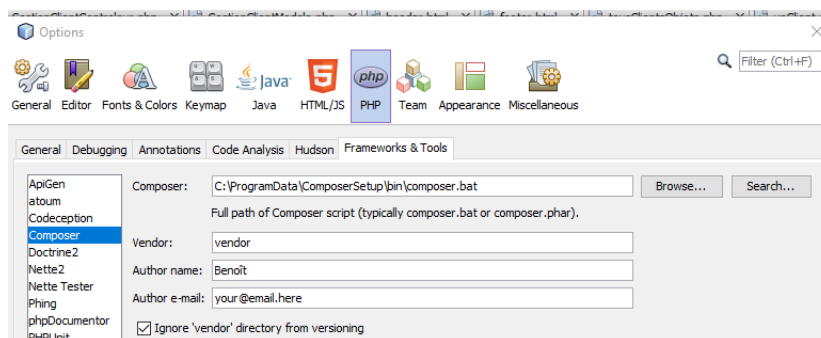
Assurez-vous que composer est installé sur votre poste de travail.

Par exemple en entrant composer dans une fenêtre de commande :

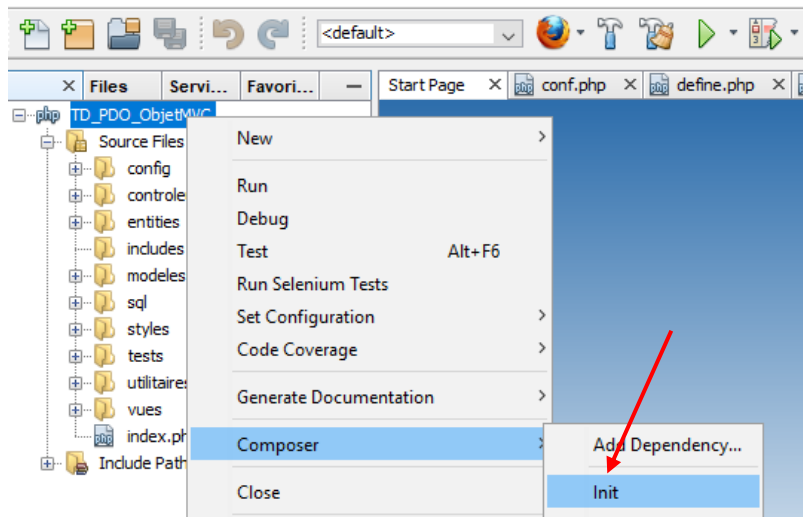
```
C:\Users\Benoit>composer
Composer
Composer version 1.6.3 2018-01-31 16:28:17
```

S'il n'est pas installé, allez voir ici : <https://getcomposer.org/> et installez composer.

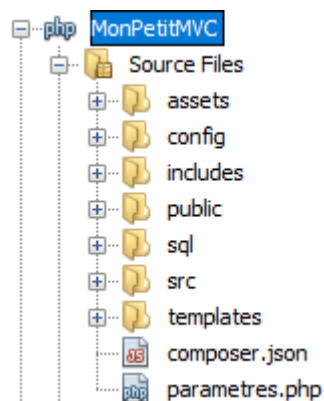
Vérifiez qu'il est configuré dans Netbeans :



Vous allez maintenant initialiser composer dans votre projet :



Vous constaterez la création du fichier composer.json à la racine du projet :



Vous rajouterez le code suivant



```
"require": {},
"autoload": {
    "psr-4": {
        "APP\\": "src",
        "Tools\\": "includes"
    }
}
```

Ce code sera invariant d'un projet à l'autre

Pour que votre autochargement de classes puisse marcher avec composer, il faut charger l'autoloader.

Vous le chargerez en mode console.

Ouvrez une fenêtre console et placez-vous à la racine de votre projet :

```
Windows PowerShell
PS T:\wampsites\cours\MVC>
```

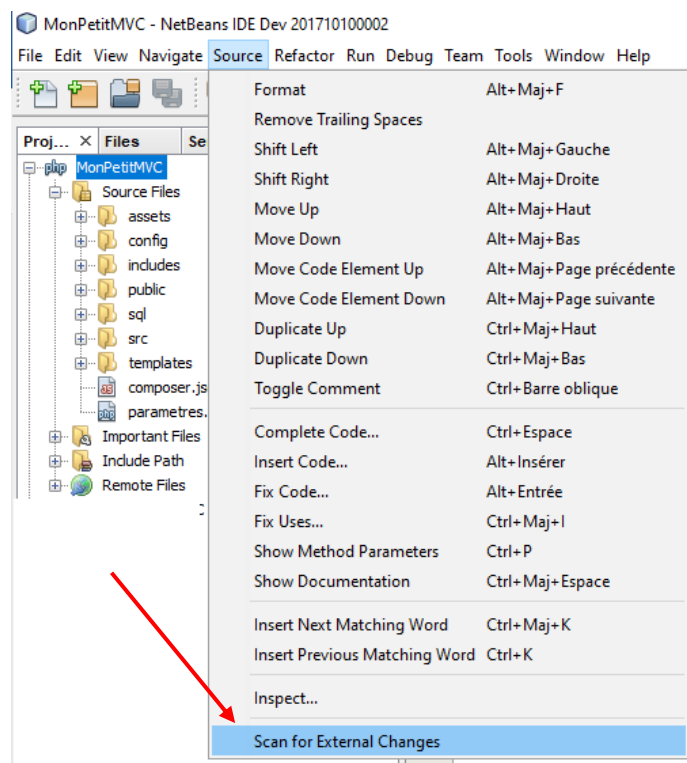
Et entrez la commande suivante :

Composer dumpautoload -o

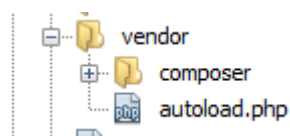
```
Windows PowerShell
PS T:\wampsites\cours\MVC> Composer dumpautoload -o
Generating optimized autoload files
PS T:\wampsites\cours\MVC>
```

Rafraîchissez votre explorateur de solution dans netbeans :

Source/Scan For External Changes



Et vous verrez qu'un nouveau dossier a été créé. Il s'agit du dossier vendor qui contient le fichier autoload.php qui va servir à lancer l'autoloader de classes.



Vous pouvez l'ouvrirpar curiosité !

Vous n'oubliez pas d'exécuter la commande



Composer dumpautoload -o

```
// autoload.php @generated by Composer

require_once __DIR__ . '/composer/autoload_real.php';

return ComposerAutoloaderInit5248b78c5a952c2717f39c194ae53c8e::getLoader();
```

chaque fois que vous modifierez les paramètres d'autoload dans le fichier composer.json

Partie 2 : MISE EN PLACE DE LA BASE DE DONNEES

Dans cette partie, vous allez :

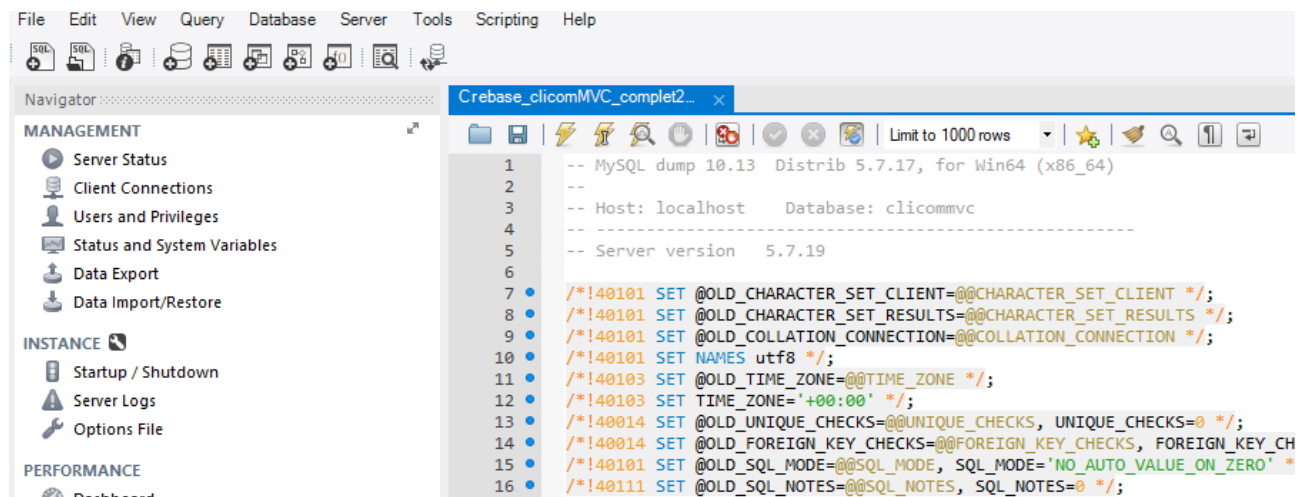
- ✓ Créer la base de données clicomMVC sous Mysql
- ✓ Créer un utilisateur votrenom/votrenom
- ✓ Donner les droits de cet utilisateur sur la base

Vous pouvez faire ces manipulations sur phpmyadmin ou mysql workbench.

1. Créer la base de données clicomMVC sous Mysql

En vous connectant avec le compte root/ vous exécutez le script

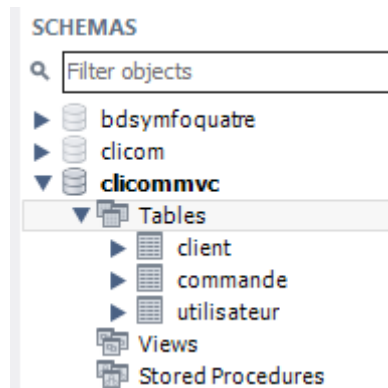
Crebase_clicomMVC_complet2019.sql



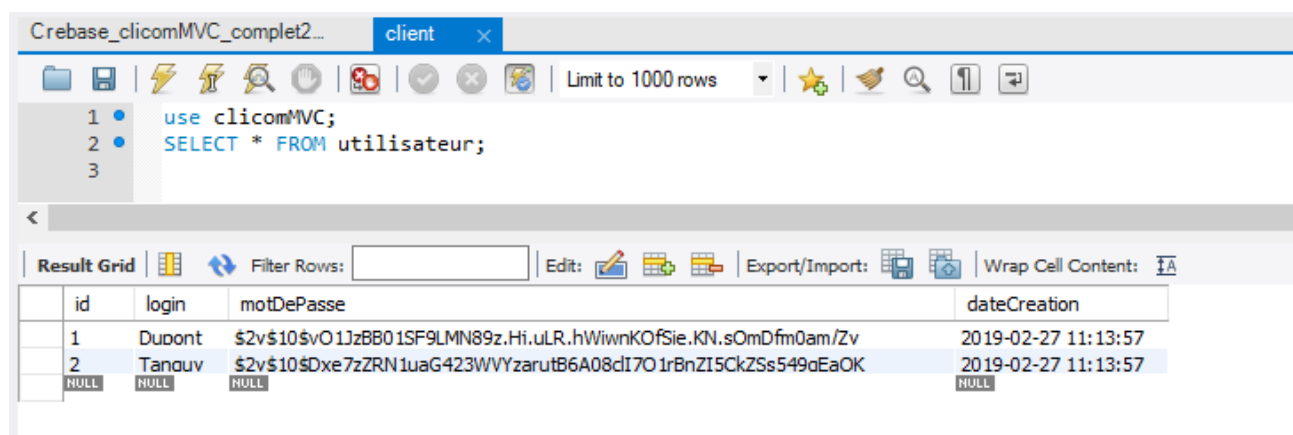
Vérifiez qu'il n'y a pas eu d'erreur dans l'exécution du script : tous les voyants sont verts :

#	Time	Action
36	13:26:59	LOCK TABLES `utilisateur` WRITE
37	13:26:59	/*!40000 ALTER TABLE `utilisateur` DISABLE KEYS */
38	13:26:59	INSERT INTO `utilisateur` VALUES (1,'Dupont','\$2y\$10\$V01JzBB01SF9LMN89z.Hj.uLR.hWiwnKQfSie.KN.sQmDfm0am/Zy','201
39	13:26:59	/*!40000 ALTER TABLE `utilisateur` ENABLE KEYS */
40	13:26:59	UNLOCK TABLES
41	13:26:59	/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */

Et que la base est bien créée :

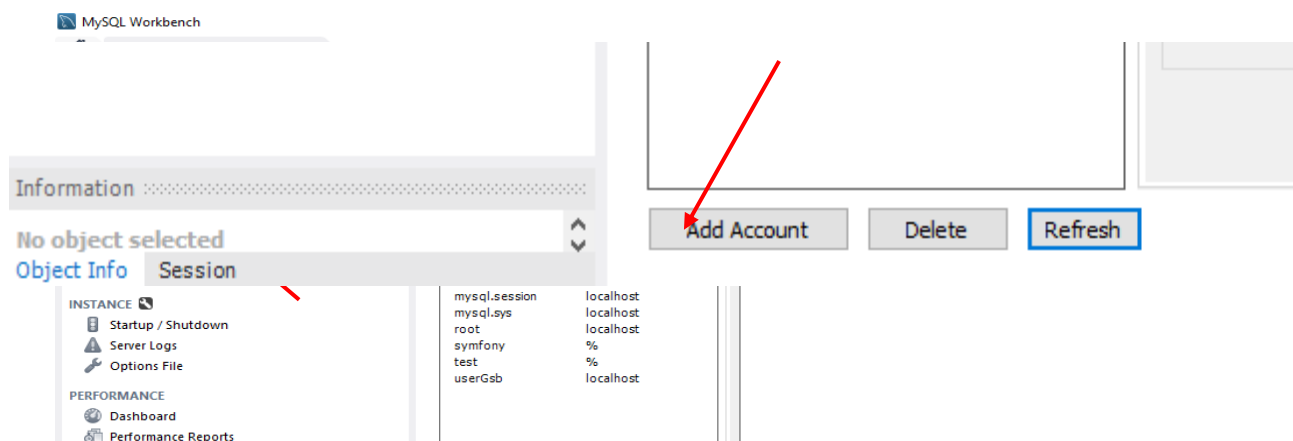


Voyez aussi que deux utilisateurs ont bien été créés :



6. Créer un utilisateur votrenom/votrenom et droits

Vous allez éviter de vous connecter sur le compte Root, vous allez créer un utilisateur :



Ajout de l'utilisateur



Vous pourrez rentrer le nom et le mot de passe de votre choix !

Details for account newuser@%

Login | Account Limits | Administrative Roles | Schema Privileges

Login Name: You may create multiple accounts with the same name to connect from different hosts.

Authentication Type: For the standard password and/or host based authentication select 'Standard'.

Limit to Hosts Matching: % and _ wildcards may be used

Password: Type a password to reset it.

Confirm Password: Enter password again to confirm.

Weak password.

Expire Password

Login | Account Limits | Administrative Roles | Schema Privileges

Role	Description
<input type="checkbox"/> DBA	grants the rights to perform all tasks
<input type="checkbox"/> MaintenanceAdmin	grants rights needed to maintain server
<input type="checkbox"/> ProcessAdmin	rights needed to assess, monitor, and kill any user proce...
<input type="checkbox"/> UserAdmin	grants rights to create users logins and reset passwords
<input type="checkbox"/> SecurityAdmin	rights to manage logins and grant and revoke server an...
<input type="checkbox"/> MonitorAdmin	minimum set of rights needed to monitor server
<input type="checkbox"/> DBManager	grants full rights on all databases
<input type="checkbox"/> DBDesigner	rights to create and reverse engineer any database sche...
<input type="checkbox"/> ReplicationAdmin	rights needed to setup and manage replication
<input type="checkbox"/> BackupAdmin	minimal rights needed to backup any database

Vous ne donnerez aucun droit d'administration à cet utilisateur
 Vous lui donnerez des droits sur le schéma que vous venez de créer (Add Entry) :

User Accounts | Details for account newuser@%

Login | Account Limits | Administrative Roles | Schema Privileges

Schema Privileges

Schema and Host fields may use % and _ wildcards.
 The server will match specific entries before wildcarded ones.

Revoke All Privileges | Delete Entry | Add Entry...

New Schema Privilege Definition

Select the Schema for which the user 'newuser' will have the privileges you want to define.

Schema

☐ All Schema (%)

☐ Schemas matching pattern:

☒ Selected schema:

This rule will apply to an

This rule will apply to sd

You may use _ and % a

Escape these character:

Select a specific schema

Schema and host fields may use % and _ wildcards.

The server will match specific entries before wildcards ones.

The user 'newuser'@'%' will have the following access rights to the schema 'clicommmvc':

Object Rights

☒ SELECT

☒ INSERT

☒ UPDATE

☒ DELETE

☐ EXECUTE

☐ SHOW VIEW

DDL Rights

☐ CREATE

☐ ALTER

☐ REFERENCES

☐ INDEX

☐ CREATE VIEW

☐ CREATE ROUTINE

☐ ALTER ROUTINE

☐ EVENT

☐ DROP

☐ TRIGGER

Other Rights

☐ GRANT OPTION

☐ CREATE TEMPORARY TABLES

☐ LOCK TABLES

Unselect All

Select "ALL"

Revert

Apply

Vérification

User Accounts

User	From Host
benoit	%
dbuser	localhost
areta	%

Details for account benoit@%

Login	Account Limits	Administrative Roles	Schema Privileges
Schema			Privileges
clicommmvc			DELETE, INSERT, SELECT, UPDATE



Répondez à la question suivante :

Quelle différence faites-vous entre l'utilisateur benoit et l'utilisateur Dupont qui apparait dans le script sql fourni ?

Partie 3 : ECRITURE DE L'APPLICATION

1. Fichiers de configuration :

Contenu du fichier define.php :

```
<?php
```

```
// Define path
define('PATH_CSS', RACINE. "assets".DS."css".DS);
define('PATH_JS', RACINE. "assets".DS."js".DS);
define('PATH_VENDOR', RACINE. "vendor".DS);
define('PATH_VIEW', RACINE . "template".DS);
```

Contenu du fichier conf.php

```
<?php
```

```
include_once(RACINE. "config".DS."define.php");
include_once(RACINE. "parametres.php");
```

2. Examen du fichier index.php

```
<?php
```

```
define('DS', DIRECTORY_SEPARATOR);
define('RACINE', new DirectoryIterator(dirname(__FILE__)) . DS . ".." . DS);
include_once(RACINE . DS . 'config/conf.php');
require PATH_VENDOR . "autoload.php";
$BaseController = filter_input(INPUT_GET, 'c', FILTER_SANITIZE_STRING);
$action = filter_input(INPUT_GET, 'a', FILTER_SANITIZE_STRING);
try {
    if (empty($BaseController)) {
        $BaseController = 'Identification';
        $action = 'login';
    }
    $controller = "APP\Controller\\" . $BaseController . 'Controller';

    $c = new $controller();
    $params=array(array_slice($_REQUEST, 2));
    call_user_func_array(array($c, $action), $params);
} catch (Exception $ex) {
    echo $ex->getMessage();
    //$vue=
    //$params= array(...)
    //include PATH_VIEW . '$BaseController'errors/ . 'View' . DS . 'unClient.php';
}
```



Vous allez répondre aux questions suivantes :

✓ Que va faire ce test :

```
if (empty($BaseController)) {
    $BaseController = 'Identification';
    $action = 'login';
}
```

✓ Que va faire cette `$c = new $controller();` instruction ?

✓ Que font ces instructions ?

```
// new controller //  
$params=array(array_slice($_REQUEST, 2));  
call_user_func_array(array($c, $action), $params);  
// fin de la fonction //
```

✓ Pourquoi utilise-t-on le tableau `$_REQUEST` plutôt que `$_GET` ou `$_POST` ?

7. Règles pour les espaces de noms

On va maintenant adopter une stratégie simple et efficace en matière d'espace de noms.

Vous allez associer à l'espace de noms APP le dossier src, puis,

- ✓ les classes *controller* auront comme espace de nom App\Controller
- ✓ les classes *entites* auront comme espace de nom App\Entity
- ✓ les classes *modeles* auront comme espace de nom App\Model



Chaque sous espace de nom correspond au dossier dans lesquels les classes sont stockées !!

Vous serez donc en conformité avec votre fichier composer.json :

```
-----  
"autoload": {  
    "psr-4": {  
        "APP\\": "src",  
        "Tools\\": "includes"  
    }  
}
```

Et vous attribuerez à chaque classe le bon espace de noms.



Exemple

```
<?php  
namespace APP\Entity;  
  
class Client {  
    // ...  
}
```

```
<?php  
namespace APP\Controller;  
  
class IdentificationController {  
    // ...  
}
```

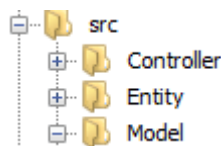
```
<?php  
namespace APP\Modele;  
  
class GestionClientModel {  
    // ...  
}
```



Vous remarquerez que l'on respecte la norme PSR-4 pour le nommage des classes et espaces de noms.

- ✓ src\Entity
- ✓ src\Controller
- ✓ src\Model

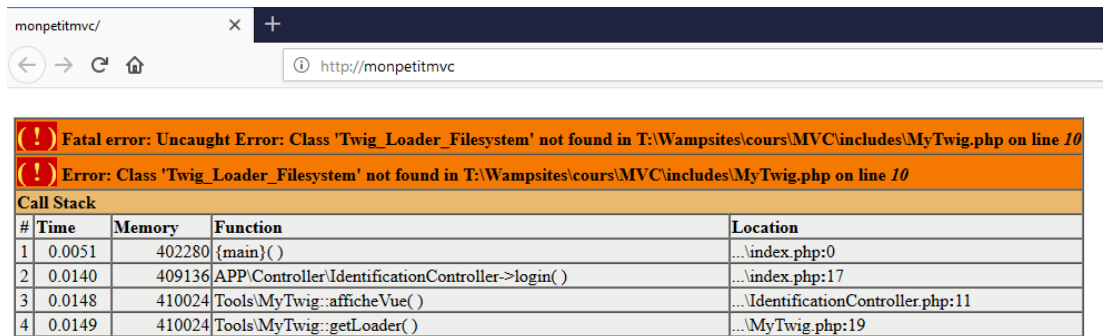
pour :



Vous pouvez maintenant tester avec l'url :

<http://monpetitmvc/>

Vous obtiendrez une erreur, mais vous constaterez que la classe *IdentificationController* a bien été trouvée :



The screenshot shows a web browser window with the address bar displaying 'http://monpetitmvc'. Below the address bar, there are two error messages in orange boxes:

- Fatal error: Uncaught Error: Class 'Twig_Loader_Filesystem' not found in T:\Wampsites\cours\MVC\includes\MyTwig.php on line 10
- Error: Class 'Twig_Loader_Filesystem' not found in T:\Wampsites\cours\MVC\includes\MyTwig.php on line 10

Below the error messages is a 'Call Stack' table:

#	Time	Memory	Function	Location
1	0.0051	402280	{main}()	...index.php:0
2	0.0140	409136	APP\Controller\IdentificationController->login()	...index.php:17
3	0.0148	410024	Tools\MyTwig::afficheVue()	...IdentificationController.php:11
4	0.0149	410024	Tools\MyTwig::getLoader()	...MyTwig.php:19

8. Classes GestionClientController et GestionClientModel

Vous allez créer la classe *GestionClientController* qui va stocker toutes les méthodes concernant la gestion des clients. Cette classe sera stockée dans le dossier *src\Controller*



Cette classe sera dans l'espace de noms *App\Controller*

```
<?php

namespace APP\Controller;

class GestionClientController {

}
```

Puis vous créerez la classe *GestionClientModel* qui va implémenter les méthodes d'accès à la BD qui concernent les clients. Cette classe sera stockée dans le dossier *src\Model*

```
<?php

namespace APP\Model;

class GestionClientModel {

}
```

Partie 4 : OPERATIONS SUR LES CLASSES CONTROLLER ET MODEL

1. Fichier des paramètres

Les paramètres de la base de données dont vous aurez besoin pour vous connecter sont stockés dans le fichier `parametres.php` :

```
<?php
define('DATABASE_URL', "localhost" );
define('DATABASE_USER', "benoit");
define('DATABASE_PWD', "benoit");
define('DATABASE_NAME', "clicom");
define('CNSTRING', "mysql:host=" . DATABASE_URL . ";dbname=" . DATABASE_NAME . ";charset=UTF8" );
```

2. Récupération d'un client

On va appeler la méthode **chercheUn** de la classe `GestionClientController` qui va se charger :

- ✓ D'appeler la méthode `find($id)` de la classe `GestionClientModel` qui va se charger de récupérer le client voulu et qui va le retourner au contrôleur sous la forme d'un objet de la classe `Client`.
- ✓ D'afficher le client dans une vue

Vous allez donc rajouter la méthode **chercheUn(\$params)** dans la classe `GestionClientController` :

```
namespace APP\Controller;

use APP\Model\GestionClientModel;
use ReflectionClass;
use \Exception;

class GestionClientController {

    public function chercheUn($params) {
        // appel de la méthode find($id) de la classe Model adequate
        $modele = new GestionClientModel();
        $id = filter_var(intval($params["id"]), FILTER_VALIDATE_INT);
        $unClient = $modele->find($id);
        if ($unClient) {
            $r = new ReflectionClass($this);
            include_once PATH_VIEW . str_replace('Controller', 'View', $r->getShortName()) . "/unClient.php";
        } else {
            throw new Exception("Client " . $id . " inconnu");
        }
    }
}
```



Avez-vous vérifié dans le fichier `define.php` que la constante `PATH_VIEWS` a été correctement déclarée ?



Vous allez répondre aux questions suivantes :


Pourquoi ce test ?

```
if ($unClient) {
```

Que fait cette instruction ?

```
$r = new ReflectionClass($this);
```

Vous allez maintenant écrire le code pour la méthode find(\$id) de la classe GestionClientModele



```
namespace APP\Model;

use \PDO;
use APP\Entity\Client;
use Tools\Connexion;

class GestionClientModel {

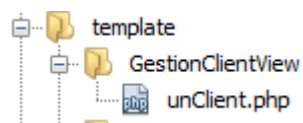
    public function find($id) {
        $unObjetPdo = Connexion::getConnexion();
        $sql = "select * from CLIENT where id=:id";
        $ligne = $unObjetPdo->prepare($sql);
        $ligne->bindValue(':id', $id, PDO::PARAM_INT);
        $ligne->execute();
        return $ligne->fetchObject(Client::class);
    }
}
```

Vous répondrez à la question suivante

Que fait cette instruction, ?

```
return $ligne->fetchObject(Client::class);
```

Vous allez maintenant écrire le code de la vue à afficher .
Créez le fichier unClient.php à cet endroit :

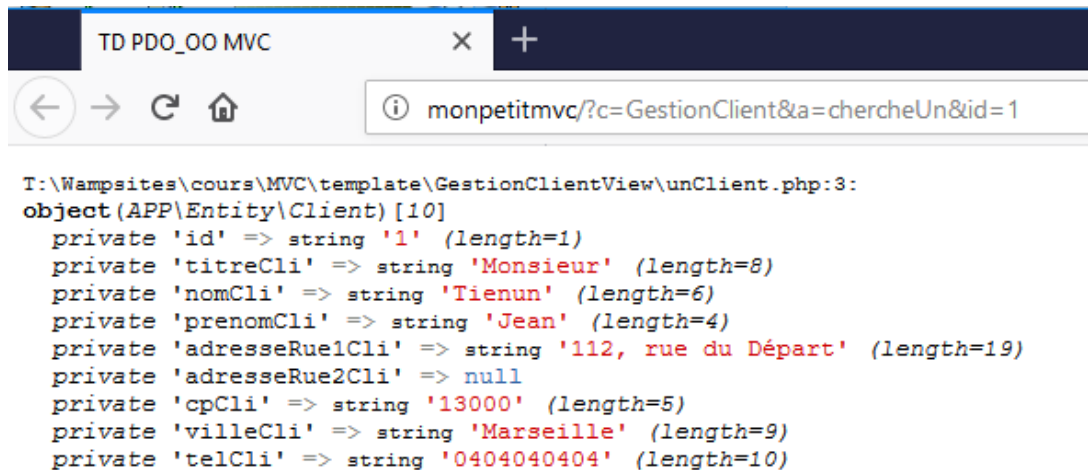


Code de la vue :

```
<?php
include_once PATH_VIEW . "header.html";
var_dump($unClient);
echo "Nom du client : " . $unClient->getNomCli();
include_once PATH_VIEW . "footer.html";
```

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

Vous devriez obtenir ceci :



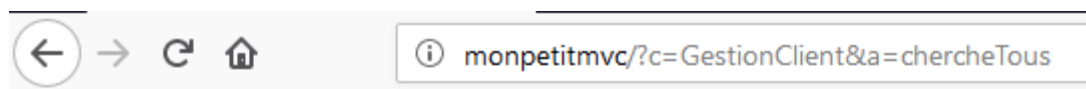
```
T:\Wampsites\cours\MVC\template\GestionClientView\unClient.php:3:
object(APP\Entity\Client) [10]
  private 'id' => string '1' (length=1)
  private 'titreCli' => string 'Monsieur' (length=8)
  private 'nomCli' => string 'Tienun' (length=6)
  private 'prenomCli' => string 'Jean' (length=4)
  private 'adresseRue1Cli' => string '112, rue du Départ' (length=19)
  private 'adresseRue2Cli' => null
  private 'cpCli' => string '13000' (length=5)
  private 'villeCli' => string 'Marseille' (length=9)
  private 'telCli' => string '0404040404' (length=10)
```

Nom du client : Tienun

3. Affichage de tous les clients

On vous demande maintenant, en vous inspirant de ce que vous venez de faire d'afficher tous les clients.

Votre URL : <http://monpetitmvc/?c=GestionClient&a=chercheTous>



Nombre de clients trouvés : 8

- 1 Monsieur Jean Tienun
- 2 Madame Julie Terrature
- 3 Madame Paulette Cerf
- 4 Mademoiselle Patricia Morizet
- 5 Monsieur Jean Nolapin
- 6 Monsieur Martel Entete
- 7 Monsieur Martel Entete
- 8 Madame Henriette DUMANS



Solution proposée :

Modification de la classe GestionClientController : ajout de la méthode chercheTous


```

class GestionClientController {

    public function chercheUn($params) {...12 lines }
    public function chercheTous() {
        // appel de la méthode findAll() de la classe Model adequate
        $modele = new GestionClientModel();
        $clients = $modele->findAll();
        if ($clients) {
            $r = new ReflectionClass($this);
            include_once PATH_VIEW . str_replace('Controller', 'View', $r->getShortName()) . "/plusieursClients.php";
        } else {
            throw new Exception("Aucun Client à afficher");
        }
    }
}

```

Modification de la classe GestionClientModel : ajout de la méthode findAll()

```

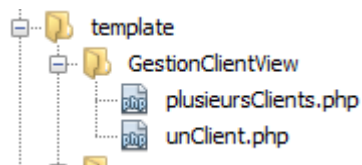
class GestionClientModel {

    public function find($id) {...8 lines }

    public function findAll() {
        $unObjetPdo = Connexion::getConnexion();
        $sql = "select * from CLIENT";
        $lignes = $unObjetPdo->query($sql);
        return $lignes->fetchAll(PDO::FETCH_CLASS, Client::class);
    }
}

```

Création de la vue destinée à afficher plusieurs clients :



```

<?php
include_once PATH_VIEW . "header.html";
echo "<p>Nombre de clients trouvés : ".count($clients) . "</p>";

foreach ($clients as $client){
    echo $client->getId() . " " . $client->getTitreCli() . " " . $client->getPrenomCli() . " " . $client->getNomCli() . "<br>";
}
include_once PATH_VIEW . "footer.html";

```

4. Exercice supplémentaire : affichage de une commande

Ecrire le code permettant avec cette URL

<http://monpetitmvc/?c=GestionCommande&a=chercheUne&id=98762>

D'afficher ceci :

```
TD PDO_OO MVC
monpetitmvc/?c=GestionCommande&a=chercheUne&id=98762

T:\Wampsites\cours\MVC\template\GestionCommandeView\uneCommande.php:3:
object(APP\Entity\Commande) [10]
  private 'id' => string '98762' (length=5)
  private 'dateCde' => string '2014-09-07' (length=10)
  private 'noFacture' => string '123454' (length=6)
  private 'idClient' => string '1' (length=1)

id du client : 1
```

5. Exercice supplémentaire : affichage de toutes les commandes

Ecrire le code permettant avec cette URL

<http://monpetitmvc/?c=GestionCommande&a=chercheToutes>

D'afficher ceci :

```
TD PDO_OO MVC
monpetitmvc/?c=GestionCommande&a=chercheToutes

Nombre de commandes trouvées : 7

98762 - 2014-09-07 - 123454 - 1
98763 - 2014-09-08 - 123455 - 2
98764 - 2014-09-10 - 123487 - 4
98765 - 2014-10-01 - 123475 - 2
98766 - 2014-10-01 - Non facturée - 4
98767 - 2014-10-01 - 123489 - 5
98768 - 2014-10-01 - Non facturée - 6
```

