



Programmation orientée objet en PHP

[Accueil](#) ► [Mes cours](#) ► [Développement logiciel](#) ► [POO PHP](#) ► [Petite application version 2](#) ► [Le projet](#)

Le projet

Dans ce TP nous allons reprendre une architecture en couche comme dans le module précédent. Nous retrouverons donc les couches dao, controller et view.

Nous allons ajouter à cette architecture un front controller, qui servira de point d'entrée unique à l'application. Lors la réception d'une requête HTTP, la requête sera redirigée vers [le front controller](#), puis [le front controller](#) sollicitera à son tour un controller adéquate.

L'un des intérêts de cette architecture est qu'elle permet une conception orientée objet de l'application. Les controllers pourront donc être implémentés sous forme de classes.

Des classes utilitaires pourront venir compléter cette architecture, telles que des classes "routers" gérant les "routes" (c'est-à-dire les correspondances entre les URL(s) et [les contrôleurs](#)), des classes gérant les formulaires et leurs validations, etc...

Pour ce projet nous allons utiliser le serveur web interne de PHP. Pour démarrer notre serveur, dans un terminal nous nous plaçons dans le répertoire web de notre projet, puis nous exécuterons la commande `php -S localhost:8080 router.php`. En passant en paramètre le fichier router.php contenu dans le répertoire web, nous demanderons à notre serveur interne de passer tout d'abord par ce script quelque soit l'URL demandée. Si l'URL demandée correspond à une ressource (image, css...) alors le script router.php renverra null et l'URL sera traitée normalement. Si la ressource ne correspond pas à une ressource alors [le front controller](#) entrera en action. Le script router.php jouera ici le rôle du fichier .htaccess du serveur HTTP Apache.

```
1 <?php
2
3 // web/router.php
4 if (preg_match('/\.(png|jpg|jpeg|gif|css)$/ ', $_SERVER["REQUEST_URI"])) {
5     return false;
6 } else {
7     include 'index.php';
8 }
```

Le projet sera constitué de trois vues :

- une vue "accueil" avec deux liens => <http://localhost:8080/>
- une vue de consultation des personnes => <http://localhost:8080/personnes>
- une vue d'ajout d'une personne => <http://localhost:8080/personnes/ajouter>
- une vue d'erreur

L'accueil

Accueil !

[Consulter les personnes](#)
[Ajouter une personne](#)

La consultation des personnes :

	Id	Nom	Prénom
1	Sparrow	Jack	
2	Wayne	Bruce	
3	Parker	Peter	

4 Kirk James T.
[Ajouter une personne](#)

L'ajout d'une personne :

Nom :
 Prénom :

La page d'erreur :

Oups...

Nous allons créer un projet nommé `petite_application`. Dans ce projet nous allons créer un répertoire pour chaque couche applicative (dao, controller, view), un répertoire config qui contiendra le fichier de configuration ainsi que le fichier de routing, un répertoire entity qui contiendra les entités du modèle, un répertoire web qui contiendra toutes les ressources publiques de notre application (images, css...) mais aussi [le front controller](#) (index.php), un répertoire util qui contiendra les classes utilitaires de routing, et enfin le répertoire test qui contiendra tous les tests de l'application. Nous aurons besoin de bibliothèques de test (PHPUnit, DBUnit et PHPUnit-selenium), que nous installerons via composer. L'archive `composer.phar` sera à la racine de notre application ainsi que le fichier `composer.json`.

```

1 {
2     "require-dev": {
3         "phpunit/phpunit": "4.*",
4         "phpunit/dbunit": "1.*",
5         "phpunit/phpunit-selenium": "1.*"
6     }
7 }
```

Une fois les dépendances installées via composer, l'arborescence du projet sera la suivante :

```

petite_application
├── Source Files
│   ├── config
│   ├── controller
│   ├── dao
│   ├── entity
│   ├── test
│   ├── util
│   ├── vendor
│   ├── view
│   ├── web
│   ├── composer.json
│   ├── composer.lock
│   ├── log.txt
│   └── setup.php
└── Include Path
```

Nous créerons pour ce TP deux bases de données MySQL : une base de production nommée `petite_application` et une base de test nommée `test_petite_application`.

Fin

NAVIGATION

[Accueil](#)

■ [Ma page](#)



Pages du site

Mon profil

Cours actuel

POO PHP

Participants

Généralités

La programmation orientée objet : premiers pas

L'héritage

Les interfaces

Le typage

Les namespaces

Les exceptions

Les bases de données avec PDO

Les tests avec PHPUnit

Petite application version 2

 **Le projet**

 conf.ini et setup.php

 Les entités et la dao

 Test de la DAO

 La gestion des routes

 Test des routes

 Les contrôleurs

 Les vues

 Le front controller

 Test de l'ajout d'une personne

 T.P. mettre à jour une personne

Petite application version 3

[Mes cours](#)

ADMINISTRATION



Administration du cours

Réglages de mon profil

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

[Accueil](#) ▶ [Mes cours](#) ▶ [Développement logiciel](#) ▶ [POO PHP](#) ▶ [Petite application version 2](#) ▶ [conf.ini et setup.php](#)

conf.ini et setup.php

Dans le répertoire config de l'application nous trouverons le fichier conf.ini qui contiendra les configurations de base de notre application. Dans ce fichier nous trouverons trois sections. La section prod qui contient les configurations liées à l'environnement de production, la section dev qui contient celles liées à l'environnement de développement et de test. La section settings contiendra les paramètres de l'application, dont l'environnement choisi.

config/conf.ini

```
1 [prod]
2 host = localhost
3 db_name = petite_application
4 user = root
5 password = root
6
7 [dev]
8 host = localhost
9 db_name = test_petite_application
10 user = root
11 password = root
12
13 [settings]
14 env = dev
```

Dans le fichier de setup, à la racine de notre application, nous allons définir toutes les constantes utiles à notre application et déclarer une fonction d'autoload. Cette fonction nous permettra de "charger" les classes au moment de leurs instanciations via un include_once. Cette fonction inclut la classe à partir de son namespace : dans cette application le namespace correspond au répertoire de la classe. Cette fonction est une fonction anonyme qui est passée directement en argument d'une autre fonction : spl_autoload_register(). Cette dernière fonction permet l'enregistrement et l'activation d'une fonction d'autoload.

```
1 <?php
2
3 // setup.php
4 define('SITE_URL', 'http://localhost:8080');
5 define('ROOT', realpath(__DIR__));
6 define('CONTROLLER', ROOT . '/controller');
7 define('ENTITY', ROOT . '/entity');
8 define('VIEW', ROOT . '/view');
9 define('DAO', ROOT . '/dao');
10 define('CONFIG', ROOT . '/config');
11 define('INI_FILE', CONFIG . '/conf.ini');
12
13 spl_autoload_register(function($class) {
14     list($dir, $file) = explode('\\', $class);
15     include_once ROOT . "/" . $dir . $file . ".php";
16 });
```

Fin

NAVIGATION



Accueil

■ [Ma page](#)[Pages du site](#)[Mon profil](#)[Cours actuel](#)[POO PHP](#)[Participants](#)[Généralités](#)[La programmation orientée objet : premiers pas](#)[L'héritage](#)[Les interfaces](#)[Le typage](#)[Les namespaces](#)[Les exceptions](#)[Les bases de données avec PDO](#)[Les tests avec PHPUnit](#)[Petite application version 2](#) [Le projet](#) [conf.ini et setup.php](#) [Les entités et la dao](#) [Test de la DAO](#) [La gestion des routes](#) [Test des routes](#) [Les contrôleurs](#) [Les vues](#) [Le front controller](#) [Test de l'ajout d'une personne](#) [T.P. mettre à jour une personne](#)[Petite application version 3](#)[Mes cours](#)

ADMINISTRATION

[Administration du cours](#)[Réglages de mon profil](#)Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))[POO PHP](#)



Programmation orientée objet en PHP

[Accueil](#) ► [Mes cours](#) ► [Développement logiciel](#) ► [POO PHP](#) ► [Petite application version 2](#) ► [Les entités et la dao](#)

Les entités et la dao

La couche entity sera constituée de la classe Personne, liée au namespace entity, qui correspond à son répertoire :

```
1 <?php
2
3 // entity/Personne.php
4
5 namespace entity;
6
7 class Personne {
8
9     protected $id;
10    protected $nom;
11    protected $prenom;
12
13    function __construct($nom, $prenom, $id = null) {
14        $this->nom = $nom;
15        $this->prenom = $prenom;
16        $this->id = $id;
17    }
18
19    public function getId() {
20        return $this->id;
21    }
22
23    public function getNom() {
24        return $this->nom;
25    }
26
27    public function getPrenom() {
28        return $this->prenom;
29    }
30
31    public function setNom($nom) {
32        $this->nom = $nom;
33    }
34
35    public function setPrenom($prenom) {
36        $this->prenom = $prenom;
37    }
38
39 }
```

La classe dao MySqlConnection sera dans l'espace de nom dao. Elle possédera en attribut d'instance une connexion PDO (\$conn). Cette connexion sera instanciée dans le constructeur et utilisée dans les méthodes. Pour créer la connexion, la classe utilisera les configurations du fichier conf.ini dont le chemin est représenté par la constante INI_FILE.

```
1 <?php
2
3 // dao/MysqlDao.php
4
```

```

5 namespace dao;
6
7 use PDO;
8 use entity\Personne;
9
10 class MysqlDao {
11
12     private $datasource;
13     private $user;
14     private $password;
15     private $conn;
16
17     public function __construct() {
18         $conf = parse_ini_file(INI_FILE, true);
19         $settings = $conf['settings']['env'];
20         $host = $conf[$settings]['host'];
21         $db_name = $conf[$settings]['db_name'];
22         $this->user = $conf[$settings]['user'];
23         $this->password = $conf[$settings]['password'];
24         $this->datasource = "mysql:host=$host;dbname=$db_name";
25         $this->conn = new PDO($this->datasource, $this->user, $this->password
26             , [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]);
27     }
28
29     public function getAllPersonnes() {
30         $statement = 'SELECT * FROM personne';
31         $stmt = $this->conn->prepare($statement);
32         $stmt->execute();
33         $result = [];
34         while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
35             $pers = new Personne($row['nom'], $row['prenom'], $row['id']);
36             $result[] = $pers;
37         }
38         return $result;
39     }
40
41     public function addPersonne(Personne $personne) {
42         $statement = 'INSERT INTO personne (nom,prenom) values (:nom, :prenom)';
43         $stmt = $this->conn->prepare($statement);
44         $stmt->bindValue(':nom', $personne->getNom());
45         $stmt->bindValue(':prenom', $personne->getPrenom());
46         $result = $stmt->execute();
47         return $result;
48     }
49
50 }

```

Les exceptions qui pourraient être renvoyées dans ces méthodes seront propagées aux méthodes appelantes.

Fin

NAVIGATION



Accueil

■ Ma page

Pages du site

Mon profil

Cours actuel

POO PHP












Participants

Généralités

La programmation orientée objet : premiers pas

L'héritage

Les interfaces

- Le typage
- Les namespaces
- Les exceptions
- Les bases de données avec PDO
- Les tests avec PHPUnit
- Petite application version 2
 -  [Le projet](#)
 -  [conf.ini et setup.php](#)
 -  **[Les entités et la dao](#)**
 -  [Test de la DAO](#)
 -  [La gestion des routes](#)
 -  [Test des routes](#)
 -  [Les contrôleurs](#)
 -  [Les vues](#)
 -  [Le front controller](#)
 -  [Test de l'ajout d'une personne](#)
 -  [T.P. mettre à jour une personne](#)
- Petite application version 3

[Mes cours](#)

ADMINISTRATION

[Administration du cours](#)[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

[Accueil](#) ► [Mes cours](#) ► [Développement logiciel](#) ► [POO PHP](#) ► [Petite application version 2](#) ► [Test de la DAO](#)

Test de la DAO

Nous allons effectuer un [premier test](#) d'intégration pour vérifier le bon fonctionnement de notre DAO. En testant [la classe MysqlDao](#) nous testerons aussi sa dépendance, c'est-à-dire la classe PDO qui fournit la connexion à la base donnée. On ne peut donc pas parler ici de test unitaire, mais plutôt de test d'intégration. Pour un premier exemple, ce niveau de granularité nous suffit.

Pour effectuer le test de [la classe MysqlDao](#) nous allons donc créer une classe `test\dao\MysqlDaoTest` héritant de `PHPUnit_Extensions_Database_TestCase`.

Nous placerons dans le répertoire `test/dataset` les deux fichiers qui nous serviront de dataset. À savoir `personne_base.yml` :

```
1 # test/dataset/personne_base.yml
2
3 personne:
4   -
5     id: 1
6     nom: "Sparrow"
7     prenom: "Jack"
8   -
9     id: 2
10    nom: "Wayne"
11    prenom: "Bruce"
```

et `add_personne.yml` :

```
1 # test/dataset/add_personne.yml
2
3 personne:
4   -
5     id: 1
6     nom: "Sparrow"
7     prenom: "Jack"
8   -
9     id: 2
10    nom: "Wayne"
11    prenom: "Bruce"
12   -
13     id: 3
14     nom: "Kent"
15     prenom: "Clark"
```

L'arborescence de notre répertoire de test sera donc la suivante :

```
▼ test
  ▼ dao
    MysqlDaoTest.php
  ▼ dataset
    add_personne.yml
    personne_base.yml
    phpunit.xml
```

La classe `MysqlDaoTest` sera quasiment identique à celle du module précédent, à ceci près que cette fois la méthode `addPersonne()` de `MysqlDao` prendra en argument un objet de type `Personne` et non deux chaînes de caractères. De plus nous irons cette fois chercher la configuration de connexion dans le fichier `conf.ini`. Les

"include" et les namespaces utilisés seront aussi différents. Notez que nous incluons le fichier setup.php car il contient la définition des constantes utilisées dans l'application.

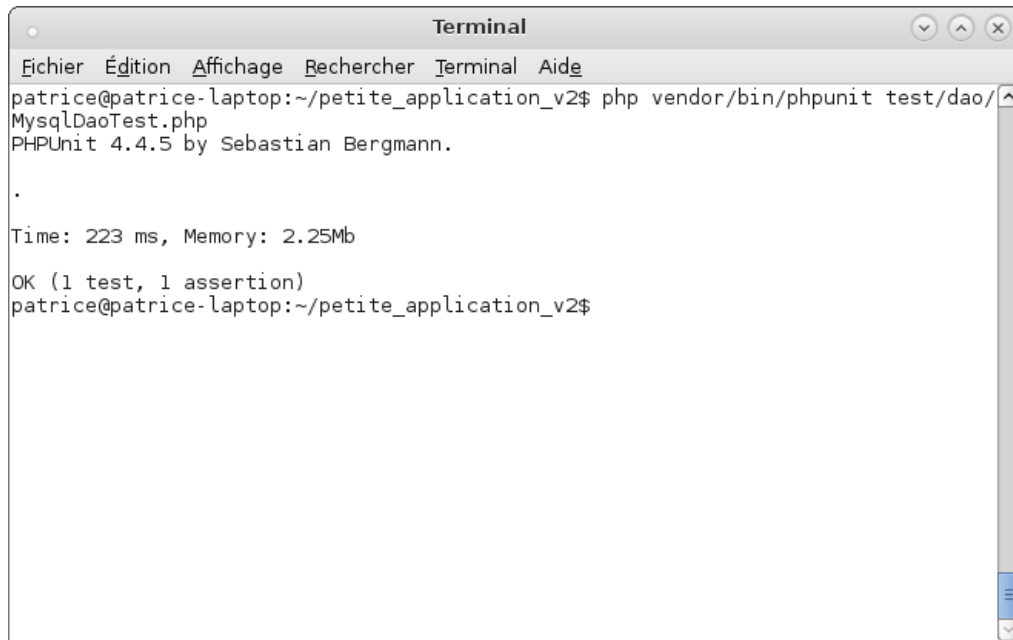
```

1  <?php
2
3  // test/dao/MysqlDaoTest.php
4
5  include_once './setup.php';
6  include_once './entity/Personne.php';
7  include_once './dao/MysqlDao.php';
8
9  use entity\Personne;
10 use dao\MysqlDao;
11
12 class MysqlDaoTest extends PHPUnit_Extensions_Database_TestCase {
13
14     protected $connection;
15     protected $dao;
16
17     protected function getConnection() {
18         if ($this->connection == null) {
19             $conf = parse_ini_file(INI_FILE, true);
20             $settings = $conf['settings']['env'];
21             $host = $conf[$settings]['host'];
22             $db_name = $conf[$settings]['db_name'];
23             $this->user = $conf[$settings]['user'];
24             $this->password = $conf[$settings]['password'];
25             $this->datasource = "mysql:host=$host;dbname=$db_name";
26             $conn = new PDO($this->datasource, $this->user, $this->password
27                 , [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]);
28             $this->connection = $this->createDefaultDBConnection($conn);
29         }
30         return $this->connection;
31     }
32
33     protected function getDataSet() {
34         return new PHPUnit_Extensions_Database_DataSet_YamlDataSet(
35             './test/dataset/personne_base.yml');
36     }
37
38     protected function setUp() {
39         $conn = $this->getConnection();
40         // désactivation des contraintes de clés étrangères pour permettre
41         // le chargement des données via le dataset
42         $conn->getConnection()->query('set foreign_key_checks=0');
43         parent::setUp();
44         // activation des contraintes de clés étrangères
45         $conn->getConnection()->query('set foreign_key_checks=1');
46         $this->dao = new MysqlDao();
47     }
48
49     public function testAddPersonne() {
50         $pers = new Personne("Kent", "Clark");
51         $this->dao->addPersonne($pers);
52         // création d'un objet dataset à partir de la connexion
53         $actualDataset = new PHPUnit_Extensions_Database_DataSet_QueryDataSet(
54             $this->getConnection());
55         // ajout à ce dataset des enregistrements de la table personne
56         // de notre base de données de test
57         $actualDataset->addTable('personne');
58         // récupération d'un dataset à partir de notre fichier
59         $expectedDataset = new PHPUnit_Extensions_Database_DataSet_YamlDataSet(
60             './test/dataset/add_personne.yml');
61         // comparaison des deux datasets
62         $this->assertDataSetsEqual($expectedDataset, $actualDataset);
63     }
64

```

```
65 }
```

Nous effectuons le test :



```
Terminal
Fichier Édition Affichage Rechercher Terminal Aide
patrice@patrice-laptop:~/petite_application_v2$ php vendor/bin/phpunit test/dao/
MysqlDaoTest.php
PHPUnit 4.4.5 by Sebastian Bergmann.

.

Time: 223 ms, Memory: 2.25Mb

OK (1 test, 1 assertion)
patrice@patrice-laptop:~/petite_application_v2$
```

Il nous faudra ainsi tester toutes les méthodes de [la classe MysqlDao](#).

Fin

NAVIGATION

Accueil

■ Ma page

Pages du site

Mon profil

Cours actuel

POO PHP

Participants

Généralités

La programmation orientée objet : premiers pas

L'héritage

Les interfaces

Le typage

Les namespaces

Les exceptions

Les bases de données avec PDO

Les tests avec PHPUnit

Petite application version 2

Le projet

conf.ini et setup.php

Les entités et la dao

Test de la DAO

La gestion des routes

Test des routes

Les contrôleurs

Les vues

Le front controller

Test de l'ajout d'une personne

T.P. mettre à jour une personne

Petite application version 3

Mes cours

ADMINISTRATION[Administration du cours](#)[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

[Accueil](#) ► [Mes cours](#) ► [Développement logiciel](#) ► [POO PHP](#) ► [Petite application version 2](#) ► [La gestion des routes](#)

La gestion des routes

Pour gérer les "routes" de notre application, c'est-à-dire les liens entre les URL et [les contrôleurs](#), nous allons utiliser un fichier de configuration et deux classes.

Le fichier routing.json du répertoire config sera notre fichier de configuration. Il contiendra un tableau d'objets au format json possédant les propriétés name, path, controller et action. Le path correspond à la partie de l'URL qui suit le nom de domaine.

config/routing.json

```
1  [  
2    {  
3      "name": "index",  
4      "path": "/",  
5      "controller": "IndexController",  
6      "action": "indexAction"  
7    },  
8    {  
9      "name": "afficher_personne",  
10     "path": "/personnes",  
11     "controller": "PersonneController",  
12     "action": "getAllPersonnes"  
13   },  
14   {  
15     "name": "ajouter_personne",  
16     "path": "/personnes/ajouter",  
17     "controller": "PersonneController",  
18     "action": "addPersonne"  
19   }  
20 ]
```

D'autres formats auraient pu être utilisés pour structurer ce fichier de configuration : ini, XML, YAML...

Dans notre fichier setup.php nous allons ajouter une constante ROUTING_FILE représentant le fichier de routing :

```
1  <?php  
2  
3  // setup.php  
4  define('SITE_URL', 'http://localhost:8080');  
5  define('ROOT', realpath(__DIR__));  
6  define('CONTROLLER', ROOT . '/controller');  
7  define('ENTITY', ROOT . '/entity');  
8  define('VIEW', ROOT . '/view');  
9  define('DAO', ROOT . '/dao');  
10 define('CONFIG', ROOT . '/config');  
11 define('INI_FILE', CONFIG . '/conf.ini');  
12 define('ROUTING_FILE', CONFIG . '/routing.json');  
13  
14 spl_autoload_register(function($class) {  
15     list($dir, $file) = explode('\\', $class);  
16     include_once ROOT . "/" . $dir . $file . ".php";  
17 });
```

La classe Route du namespace util permettra d'encapsuler les données d'une route et d'y accéder en lecture :

```

1  <?php
2
3  // util/Route.php
4
5  namespace util;
6
7  class Route {
8
9      private $name;
10     private $path;
11     private $controller;
12     private $action;
13
14     function __construct($name, $path, $controller, $action) {
15         $this->name = $name;
16         $this->path = $path;
17         $this->controller = $controller;
18         $this->action = $action;
19     }
20
21     public function getName() {
22         return $this->name;
23     }
24
25     public function getPath() {
26         return $this->path;
27     }
28
29     public function getController() {
30         return $this->controller;
31     }
32
33     public function getAction() {
34         return $this->action;
35     }
36
37 }
```

La classe Router du namespace util possédera une méthode de classe `getRoute()`. Cette méthodeinstanciera et renverra un objet Route en fonction du chemin donné en argument. Pour cela, cette fonction parcourra le fichier de configuration `routing.json`. Si aucun objet json ne correspond au path donné dans l'URL, alors la méthode renverra null.

```

1  <?php
2
3  // util/Router.php
4
5  namespace util;
6
7  use util\Route;
8
9  class Router {
10
11     public static function getRoute($path) {
12         $routing_file_content = file_get_contents(ROUTING_FILE);
13         $routing = json_decode($routing_file_content);
14         $route = null;
15         foreach ($routing as $item) {
16             if ($item->path == $path) {
17                 $route = new Route($item->name, $item->path, $item->controller, $item-
18 >action);
19                 break;
20             }
21         }
22     }
23 }
```

```
21     }
22     return $route;
23 }
24 }
```

[Fin](#)

NAVIGATION





Accueil

■ Ma page

[Pages du site](#)[Mon profil](#)[Cours actuel](#)

POO PHP

[Participants](#)[Généralités](#)[La programmation orientée objet : premiers pas](#)[L'héritage](#)[Les interfaces](#)[Le typage](#)[Les namespaces](#)[Les exceptions](#)[Les bases de données avec PDO](#)[Les tests avec PHPUnit](#)[Petite application version 2](#) [Le projet](#) [conf.ini et setup.php](#) [Les entités et la dao](#) [Test de la DAO](#) [La gestion des routes](#) [Test des routes](#) [Les contrôleurs](#) [Les vues](#) [Le front controller](#) [Test de l'ajout d'une personne](#) [T.P. mettre à jour une personne](#)[Petite application version 3](#)[Mes cours](#)

ADMINISTRATION

[Administration du cours](#)[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

[Accueil](#) ▶ [Mes cours](#) ▶ [Développement logiciel](#) ▶ [POO PHP](#) ▶ [Petite application version 2](#) ▶ [Test des routes](#)

Test des routes

Nous allons tester le comportement de la classe Router, notamment celui de la méthode de classe `getRoute()`. Pour tester de cette méthode, nous allons récupérer un objet de type Route renvoyé par la méthode, puis comparer cet objet Route à un autre objet Route initialisé aux valeurs attendues. Pour créer un objet Route, la méthode `getRoute()` a besoin du fichier de configuration `routing.json` qui contient les valeurs de la route.

Dans le répertoire `test/util` nous créons une classe RouterTest :

```
1 <?php
2
3 // test/util/RouterTest.php
4
5 include_once './setup.php';
6 include_once './util/Route.php';
7 include_once './util/Router.php';
8
9 use util\Route;
10 use util\Router;
11
12 class RouterTest extends PHPUnit_Framework_TestCase {
13
14     public function testGetRoute() {
15         $name = "ajouter_personne";
16         $path = "/personnes/ajouter";
17         $controller = "PersonneController";
18         $action = "addPersonne";
19         $expectedRoute = new Route($name, $path, $controller, $action);
20         $route = Router::getRoute($path);
21         $this->assertEquals($expectedRoute, $route);
22     }
23
24 }
```

Nous exécutons le test en console :

```
Terminal
Fichier Édition Affichage Rechercher Terminal Aide
patrice@patrice-laptop:~/petite_application$ vendor/bin/phpunit test/util/RouterTest.php
PHPUnit 4.3.4 by Sebastian Bergmann.
.
Time: 34 ms, Memory: 1.75Mb
OK (1 test, 1 assertion)
patrice@patrice-laptop:~/petite_application$
```


[Fin](#)

NAVIGATION



Accueil

■ [Ma page](#)

[Pages du site](#)[Mon profil](#)[Cours actuel](#)

[POO PHP](#)

[Participants](#)[Généralités](#)[La programmation orientée objet : premiers pas](#)[L'héritage](#)[Les interfaces](#)[Le typage](#)[Les namespaces](#)[Les exceptions](#)[Les bases de données avec PDO](#)[Les tests avec PHPUnit](#)[Petite application version 2](#) [Le projet](#) [conf.ini et setup.php](#) [Les entités et la dao](#) [Test de la DAO](#) [La gestion des routes](#) [Test des routes](#) [Les contrôleurs](#) [Les vues](#) [Le front controller](#) [Test de l'ajout d'une personne](#) [T.P. mettre à jour une personne](#)[Petite application version 3](#)[Mes cours](#)

ADMINISTRATION

[Administration du cours](#)[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

[Accueil](#) ► [Mes cours](#) ► [Développement logiciel](#) ► [POO PHP](#) ► [Petite application version 2](#) ► [Les contrôleurs](#)

Les contrôleurs

[Les contrôleurs](#) porteront le namespace controller et seront placés dans le répertoire controller.

La classe IndexController possédera une méthode indexAction() dont le rôle sera simplement d'inclure la vue d'accueil.

```
1 <?php
2
3 // controller/IndexController.php
4
5 namespace controller;
6
7
8 class IndexController {
9
10     public function indexAction() {
11         include VIEW . '/index.php';
12     }
13
14 }
```

La classe PersonneController possédera un attribut d'instance dao qui sera initialisé avec un objet de type MysqlDao dans le constructeur. Le controller possédera deux méthodes : getAllPersonnes() et addPersonne(). La méthode getAllPersonnes() récupérera l'ensemble des personnes via la méthode getAllPersonnes() du dao et inclura la vue d'affichage des personnes.

La fonction addPersonne() inclura la vue d'ajout de personne (le formulaire) si la requête est en GET, si la requête est en POST, la méthode ajoutera la personne via la méthode addPersonne() de MysqlDao et appellera la méthode getAllPersonnes() du controller.

Les exceptions qui pourraient être déclenchées dans ces méthodes seront propagées aux méthodes appelantes.

```
1 <?php
2
3 // controller/PersonneController.php
4
5 namespace controller;
6
7 use entity\Personne;
8 use dao\MysqlDao;
9
10 class PersonneController {
11
12     private $dao;
13
14     public function __construct() {
15         $this->dao = new MysqlDao();
16     }
17
18     public function addPersonne() {
19         if ($_SERVER['REQUEST_METHOD'] == 'POST') {
20             $nom = filter_input(INPUT_POST, 'nom', FILTER_VALIDATE_REGEXP,
21                               ['options' => ['regexp' => '/^[A-Za-z- ]{1,20}$/']]);
22             $prenom = filter_input(INPUT_POST, 'prenom', FILTER_VALIDATE_REGEXP,
23                                   ['options' => ['regexp' => '/^[A-Za-z0-9]{4,10}$/']]);
24         }
25     }
26 }
```

```
24         if ($nom && $prenom) {
25             $personne = new Personne($nom, $prenom);
26             $this->dao->addPersonne($personne);
27             $this->getAllPersonnes();
28         } else {
29             $msg = 'Champs incorrects';
30             include VIEW . '/formulaire.php';
31         }
32     } else {
33         include VIEW . '/formulaire.php';
34     }
35 }
36
37 public function getAllPersonnes() {
38     $result = $this->dao->getAllPersonnes();
39     include VIEW . '/display_personne.php';
40 }
41
42 }
```

[Fin](#)

NAVIGATION



Accueil

■ Ma page

[Pages du site](#)[Mon profil](#)[Cours actuel](#)

POO PHP

[Participants](#)[Généralités](#)[La programmation orientée objet : premiers pas](#)[L'héritage](#)[Les interfaces](#)[Le typage](#)[Les namespaces](#)[Les exceptions](#)[Les bases de données avec PDO](#)[Les tests avec PHPUnit](#)[Petite application version 2](#) [Le projet](#) [conf.ini et setup.php](#) [Les entités et la dao](#) [Test de la DAO](#) [La gestion des routes](#) [Test des routes](#) [Les contrôleurs](#) [Les vues](#) [Le front controller](#) [Test de l'ajout d'une personne](#) [T.P. mettre à jour une personne](#)[Petite application version 3](#)[Mes cours](#)

ADMINISTRATION

[Administration du cours](#)[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

[Accueil](#) ► [Mes cours](#) ► [Développement logiciel](#) ► [POO PHP](#) ► [Petite application version 2](#) ► [Les vues](#)

Les vues

Les vues seront des pages html contenant du PHP. Elles seront dans le répertoire view.

```
1 <?php
2 // view/index.php
3 ?>
4 <!DOCTYPE html>
5 <html>
6   <head>
7     <meta charset="UTF-8">
8     <title></title>
9   </head>
10  <body>
11    <h1>Accueil !</h1>
12    <a href="<?php echo SITE_URL . "/personnes"; ?>">
13      Consulter les personnes
14    </a><br>
15    <a href="<?php echo SITE_URL . "/personnes/ajouter"; ?>">
16      Ajouter une personne
17    </a>
18  </body>
19 </html>
```

```
1 <?php
2 // view/display_personne.php
3 ?>
4 <!DOCTYPE html>
5 <html>
6   <head>
7     <meta charset="UTF-8">
8     <title>Les personnes</title>
9   </head>
10  <body>
11    <div id="content">
12      <table>
13        <tr>
14          <th>Id</th>
15          <th>Nom</th>
16          <th>Prénom</th>
17        </tr>
18
19        <?php foreach ($result as $personne) : ?>
20          <tr>
21            <td><?php echo $personne->getID() ?></td>
22            <td><?php echo $personne->getNom() ?></td>
23            <td><?php echo $personne->getPrenom() ?></td>
24          </tr>
25        <?php endforeach; ?>
26      </table>
27      <a href="<?php echo SITE_URL ?>/personnes/ajouter">Ajouter une personne</a>
28    </div>
29  </body>
30 </html>
```

```

1  <?php
2  // view/formulaire.php
3  ?>
4  <!DOCTYPE html>
5  <html>
6      <head>
7          <meta charset="UTF-8">
8          <title>Ajout d'une personne</title>
9      </head>
10     <body>
11         <?php if (isset($msg)) : ?>
12             <h3><?php echo $msg ?></h3>
13         <?php endif; ?>
14         <form action="<?php echo SITE_URL ?>/personnes/ajouter" method="post">
15             <label>Nom : <input type="text" id="nom" name="nom"></label><br>
16             <label>Prénom : <input type="text" id="prenom" name="prenom"></label><br>
17             <input type="submit" value="Valider">
18         </form>
19     </body>
20 </html>

```

errorPage.html

```

1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>TODO supply a title</title>
5          <meta charset="UTF-8">
6          <meta name="viewport" content="width=device-width">
7      </head>
8      <body>
9          <div>Oops...</div>
10     </body>
11 </html>

```

Fin

NAVIGATION



Accueil

■ Ma page

Pages du site

Mon profil

Cours actuel

POO PHP

Participants

Généralités

La programmation orientée objet : premiers pas

L'héritage

Les interfaces

Le typage

Les namespaces

Les exceptions

Les bases de données avec PDO

Les tests avec PHPUnit

Petite application version 2







Le projet

conf.ini et setup.php

Les entités et la dao

Test de la DAO

La gestion des routes

-  [Test des routes](#)
-  [Les contrôleurs](#)
-  **[Les vues](#)**
-  [Le front controller](#)
-  [Test de l'ajout d'une personne](#)
-  [T.P. mettre à jour une personne](#)

Petite application version 3

[Mes cours](#)

ADMINISTRATION



[Administration du cours](#)

[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

[Accueil](#) ▶ [Mes cours](#) ▶ [Développement logiciel](#) ▶ [POO PHP](#) ▶ [Petite application version 2](#) ▶ [Le front controller](#)

Le front controller

Le script index.php du répertoire public constituera notre front controller, c'est-à-dire le point d'entrée de notre application. Ce front controller aura pour rôle de déterminer à partir de l'URL quel controller doit être instancié et quelle méthode de ce controller doit être appelée. Si une exception survient, le front controller captera cette exception dans un bloc try/catch et inclura la vue d'erreur.

```
1 <?php
2
3 // public/index.php
4 include '../setup.php';
5
6 use util\Router;
7
8 $request_uri = $_SERVER['REQUEST_URI'];
9 $route = Router::getRoute($request_uri);
10 if ($route == null) {
11     include VIEW . '/index.php';
12 } else {
13     try {
14         $controller_name = $route->getController();
15         $action_name = $route->getAction();
16         $class_name = "controller\\" . $controller_name;
17         $class = new ReflectionClass($class_name);
18         $controller = $class->newInstance();
19         $method = $class->getMethod($action_name);
20         $method->invoke($controller);
21     } catch (Exception $ex) {
22         include VIEW . '/errorPage.html';
23     }
24 }
```

La partie de l'URL qui servira de chemin est passée en argument de la méthode de classe Router::getRoute(). Cette méthode renvoie un objet de type Route ou null. L'objet Route permet de récupérer le nom qualifié du controller à instancier. Cette instanciation se fera via des méthodes dites d'introspection ou de "reflection". La classe ReflectionClass va nous permettre de récupérer un objet représentant la classe dont le nom est passé en argument (ici ce sera le nom qualifié de notre controller). Grâce à cette instance de ReflectionClass, représentant la classe de notre controller, nous allons pouvoir instancier un controller via la méthode newInstance() de ReflectionClass. Cette méthode newInstance() va donc renvoyer une instance de la classe du controller dont le nom qualifié a été passé en argument du constructeur de ReflectionClass.

La méthode getMethod() de la classe ReflectionClass nous permet, elle, de récupérer un objet qui représente une méthode. À partir de cet objet nous allons pouvoir invoquer cette méthode sur une instance du controller.

Par exemple, si l'URL saisie par l'utilisateur est http://localhost:8080/personnes, alors la valeur de \$request_uri sera "/personnes". La méthode Router::getRoute() nous enverra donc un objet Route correspondant au path "/personnes". Cet objet Route aura donc en attribut controller la valeur "PersonneController" et en attribut action la valeur "getAllPersonnes".

Le nom qualifié de la classe du controller sera donc "controller\PersonneController" et sera assigné à la variable class_name. Nous pourrons alors récupérer un objet de type ReflectionClass qui représentera la classe controller\PersonneController et instancier une classe de type controller\PersonneController grâce à la méthode newInstance() de ReflectionClass.

L'attribut action de l'objet Route et la méthode getMethod de ReflectionClass nous permettront de récupérer un

objet représentant la méthode `getAllPersonnes()`. Nous pourrions ensuite invoquer cette méthode sur l'instance du contrôleur que nous avons précédemment créée.

L'`index.php` représente notre point d'entrée, il est donc nécessaire que les requêtes soient redirigées vers ce front controller. Le script `router.php` du répertoire web se chargera de cette redirection pour nous, mais en production il sera nécessaire de créer un fichier `.htaccess` afin d'utiliser la ré-écriture d'URL et/ou de créer un hôte virtuel.

Fin

NAVIGATION



Accueil

■ Ma page

Pages du site

Mon profil

Cours actuel

POO PHP

Participants

Généralités

La programmation orientée objet : premiers pas

L'héritage

Les interfaces

Le typage

Les namespaces

Les exceptions

Les bases de données avec PDO

Les tests avec PHPUnit

Petite application version 2

Le projet

`conf.ini` et `setup.php`

Les entités et la dao

Test de la DAO

La gestion des routes

Test des routes

Les contrôleurs

Les vues

Le front controller

Test de l'ajout d'une personne

T.P. mettre à jour une personne

Petite application version 3

Mes cours

ADMINISTRATION



Administration du cours

Réglages de mon profil

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
POO PHP



Programmation orientée objet en PHP

[Accueil](#) ▶ [Mes cours](#) ▶ [Développement logiciel](#) ▶ [POO PHP](#) ▶ [Petite application version 2](#) ▶ [Test de l'ajout d'une personne](#)

Test de l'ajout d'une personne

Nous allons à présent tester la fonctionnalité d'ajout d'une personne. Cette fonctionnalité mettra en jeu plusieurs composants de notre application. Certains de ces composants ont déjà été testés, comme [la dao](#) et le routing, d'autres non, comme le controller `PersonneController` et [le front controller](#). Ce test fonctionnel va donc nous permettre de vérifier que les composants non testés pour l'instant fonctionnent correctement.

Notre controller étant très lié à la couche présentation et au serveur web, nous allons utiliser Selenium RC pour effectuer des tests à partir des IHM. Selenium va nous permettre de décrire les sollicitations effectuées sur une IHM (une vue). Nous pourrions ainsi décrire, dans un cas de test, les valeurs à injecter dans les champs de formulaire, ou encore des actions à effectuer sur les boutons et autres composants graphiques. Selenium permettra, en fait, de simuler les actions d'un utilisateur, et de vérifier que ces actions amènent au bon résultat.

Pour effectuer des tests via un navigateur internet nous allons avoir besoin de Selenium RC (Remote Control). Selenium RC (ou Selenium Server) est un outil d'automatisation de test qui va nous permettre d'effectuer des actions sur les IHM via JavaScript. Selenium est écrit en java et se présente sous la forme d'une archive java exécutable (par exemple `selenium-server_standalone-2.43.1.jar`). Selenium RC est téléchargeable sur le site de [Selenium](#). Dans le cadre de ce TP, vous pourrez placer le jar dans le répertoire test de l'application. Pour lancer le serveur, il suffit, dans un terminal, de se placer dans le répertoire de l'archive et d'exécuter la commande `java -jar selenium-server_standalone-2.43.1.jar` :

```
Terminal
Fichier Édition Affichage Rechercher Terminal Aide
patrice@patrice-laptop:~/petite_application$ java -jar test/selenium-server-stan
dalone-2.43.1.jar
10:45:56.349 INFO - Launching a standalone server
10:45:56.394 INFO - Java: Oracle Corporation 24.65-b04
10:45:56.394 INFO - OS: Linux 3.2.0-4-686-pae i386
10:45:56.412 INFO - v2.43.1, with Core v2.43.1. Built from revision 5163bce
10:45:56.523 INFO - Default driver org.openqa.selenium.ie.InternetExplorerDriver
registration is skipped: registration capabilities Capabilities [{platform=WIND
OWS, ensureCleanSession=true, browserName=internet explorer, version=}] does not
match with current platform: LINUX
10:45:56.567 INFO - RemoteWebDriver instances should connect to: http://127.0.0.
1:4444/wd/hub
10:45:56.568 INFO - Version Jetty/5.1.x
10:45:56.569 INFO - Started HttpContext[/selenium-server/driver,/selenium-server
/driver]
10:45:56.570 INFO - Started HttpContext[/selenium-server,/selenium-server]
10:45:56.570 INFO - Started HttpContext[/,/]
10:45:56.594 INFO - Started org.openqa.jetty.jetty.servlet.ServletHandler@2f40f4
10:45:56.594 INFO - Started HttpContext[/wd,/wd]
10:45:56.599 INFO - Started SocketListener on 0.0.0.0:4444
10:45:56.599 INFO - Started org.openqa.jetty.jetty.Server@ca66b4
```

Nous allons créer une classe `PersonneControllerTest` dans le répertoire `test/controller` de notre application. Cette classe de test héritera de la classe `PHPUnit_Extensions_SeleniumTestCase`.

La classe `PHPUnit_Extensions_SeleniumTestCase` possède de nombreuses méthodes qui vont nous permettre d'effectuer des actions sur les IHM, parmi elles :

- `setBrowser($url)` : prend en argument le nom du navigateur de test
- `setBrowserUrl($url)` : prend en argument l'URL qui sera ouverte dans le navigateur de test
- `open($url)` : permet d'ouvrir une URL passée en argument
- `click($locator)` : permet d'effectuer un click sur un composant, prend en argument une chaîne locator
- `type($locator, $value)` : permet d'injecter une valeur dans un champ texte, prend en argument un locator et une valeur
- `submit($locator)` : permet de soumettre un formulaire, prend en argument le locator du formulaire

Après une action qui entraîne un chargement de page (clique sur un lien ou validation d'un formulaire par

exemple), il est nécessaire d'appeler la méthode `waitForPageToLoad()`.

Il existe de nombreuses autres méthodes décrites dans [le document de référence](#).

La chaîne locator permet la sélection d'éléments et est de la forme "typeLocator=valeur". Parmi les types de locator nous trouvons les locators :

- css : dont la valeur sera le sélecteur css correspondant à l'élément que l'on souhaite sélectionner (ex : "css=#nom")
- id : dont la valeur sera l'id du composant (ex : "id=nom")
- name : dont la valeur sera le name du composant
- link : dont la valeur sera le contenu texte de l'élément a

La classe `PHPUnit_Extensions_SeleniumTestCase` possède aussi des méthodes de vérification sur le modèle `assert*****()`. Par exemple :

- `assertTitle($title)` : vérifie le titre de la page
- `assertElementContainsText($locator, $value)` : vérifie si l'élément correspondant au locator contient la valeur \$value
- `assertChecked($locator)` : vérifie si l'élément correspondant au locator est sélectionné
- `assertTable($locator, $value)` : vérifie si le tableau et la cellule correspondant au locator contient la valeur \$value

Pour tester la méthode `getAllPersonnes()` du controller nous allons créer une méthode de test dans laquelle nous ouvrirons l'URL 'http://petite_application/personnes' dans la navigateur, puis nous vérifierons le contenu des cellules de la deuxième ligne de notre tableau (la première étant la ligne contenant les en-têtes). Pour accéder à ces cellules du tableau nous utiliserons un sélecteur du type `css=table.numCol.numLigne`, par exemple 'css=table.1.1' pour la deuxième cellule de la deuxième ligne.

La méthode `setUp()` permettra de choisir le navigateur et son URL.

```

1  <?php
2
3  // test/controller/PersonneControllerTest.php
4
5  class PersonneControllerTest extends PHPUnit_Extensions_SeleniumTestCase {
6
7      protected function setUp() {
8          $this->setBrowser('firefox');
9          $this->setBrowserUrl('http://localhost:8080/');
10     }
11
12     public function testGetAllPersonnes() {
13         $this->open('http://localhost:8080/personnes');
14         $this->waitForPageToLoad();
15         $this->assertTable('css=table.1.0', '1');
16         $this->assertTable('css=table.1.1', 'Sparrow');
17         $this->assertTable('css=table.1.2', 'Jack');
18     }
19
20 }
```

Pour tester l'ajout de la personne nous ouvrirons l'URL "http://petite_application/personnes/ajouter", puis nous "injecterons" de la donnée dans les champs nom et prénom du formulaire, puis nous soumettrons le formulaire. Nous vérifierons dans la page suivante si le champ désiré a bien été ajouté à la fin de notre tableau. Avant de lancer ce test il faut bien sûr avoir une base de données "propres" (vous pouvez pour cela importer le dump [personne.sql](#)).

```

1  <?php
2
3  // test/controller/PersonneControllerTest.php
4
5  class PersonneControllerTest extends PHPUnit_Extensions_SeleniumTestCase {
6
7      protected function setUp() {
8          $this->setBrowser('firefox');
9          $this->setBrowserUrl('http://localhost:8080/');
```

```

10     }
11
12     public function testGetAllPersonnes () {
13         $this->open('http://localhost:8080/personnes');
14         $this->waitForPageToLoad();
15         $this->assertTable('css=table.1.0', '1');
16         $this->assertTable('css=table.1.1', 'Sparrow');
17         $this->assertTable('css=table.1.2', 'Jack');
18     }
19
20     public function testAddPersonne () {
21         $nom = 'Stark';
22         $prenom = 'Tony';
23         $this->open('http://localhost:8080/personnes/ajouter');
24         $this->waitForPageToLoad();
25         $this->type('id=nom', $nom);
26         $this->type('id=prenom', $prenom);
27         $this->submit('css=form');
28         $this->waitForPageToLoad();
29         $this->assertTable('css=table.5.0', '5');
30         $this->assertTable('css=table.5.1', $nom);
31         $this->assertTable('css=table.5.2', $prenom);
32     }
33
34 }

```

Fin

NAVIGATION



Accueil

■ Ma page

Pages du site

Mon profil

Cours actuel

POO PHP

Participants

Généralités

La programmation orientée objet : premiers pas

L'héritage

Les interfaces

Le typage

Les namespaces

Les exceptions

Les bases de données avec PDO

Les tests avec PHPUnit

Petite application version 2

Le projet

conf.ini et setup.php

Les entités et la dao

Test de la DAO

La gestion des routes

Test des routes

Les contrôleurs

Les vues

Le front controller

Test de l'ajout d'une personne

T.P. mettre à jour une personne

Petite application version 3

Mes cours

ADMINISTRATION[Administration du cours](#)[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)



Programmation orientée objet en PHP

[Accueil](#) ► [Mes cours](#) ► [Développement logiciel](#) ► [POO PHP](#) ► [Petite application version 2](#) ► [T.P. mettre à jour une personne](#)

T.P. mettre à jour une personne

Ajouter à l'application une fonctionnalité de mise à jour d'une personne en respectant l'architecture.

Les fichiers seront à remettre dans une archive dont le nom sera de la forme "maj_personne_*nom_prenom*.zip".

État du travail remis

Numéro de tentative	Ceci est la tentative 1.
Statut des travaux remis	Aucune tentative
Statut de l'évaluation	Pas évalué
Dernière modification	vendredi 27 mars 2015, 16:02

Ajouter un travail

Modifier votre travail remis

NAVIGATION



[Accueil](#)

■ [Ma page](#)

[Pages du site](#)

[Mon profil](#)

[Cours actuel](#)

[POO PHP](#)

[Participants](#)

[Généralités](#)

[La programmation orientée objet : premiers pas](#)

[L'héritage](#)

[Les interfaces](#)

[Le typage](#)

[Les namespaces](#)

[Les exceptions](#)

[Les bases de données avec PDO](#)

[Les tests avec PHPUnit](#)

[Petite application version 2](#)

[Le projet](#)

[conf.ini et setup.php](#)

[Les entités et la dao](#)

[Test de la DAO](#)

[La gestion des routes](#)

[Test des routes](#)

[Les contrôleurs](#)

[Les vues](#)

[Le front controller](#)

[Test de l'ajout d'une personne](#)[T.P. mettre à jour une personne](#)

Petite application version 3

[Mes cours](#)

ADMINISTRATION

[Administration du cours](#)[Réglages de mon profil](#)

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))
[POO PHP](#)