Système d'information

Développement d'une application Web

Jean-François Berdjugin
Vincent Lestideau

Système d'information: Développement d'une application Web

par Jean-François Berdjugin et Vincent Lestideau

Publié le 28 Septembre 2009

Résumé

Nous allons construire, en utilisant un forum comme exemple, un système d'information. Ce système repose sur une architecture trois tiers[1] ; la couche présentation utilise XHTML (eXtensible HyperText Markup Language) ; la couche métier utilise PHP (Hypertext Preprocessor) et la couche d'accès aux données utilise un (SGBD) relationnel. Nous n'utiliserons pas dans cet exemple JScript (JavaScript) et AJAX (Asynchronous JavaScript and XML (Extensible Markup Language)) ce langage sera vu ultérieurement. Les ORM (Object-Relational Mapping) et les moteurs de template (gabarit ou patrons de mise en page) seront vus en cours mais ne seront pas utilisés ici. Nous continuerons à utiliser SQL (Structured Query Language) au travers d'une couche d'accès aux données.

[1] Un tiers est une couche.

Table des matières

Liste des illustrations

Liste des tableaux

Liste des exemples

Chapitre 1. Présentation du projet

1. Séances et évaluations

Nous avons trois séances de TD (Travaux Dirigés) de trois heures suivies de trois séances de TP (Travaux Pratiques) de trois heures. Nous aurons deux évaluations le première à la fin des séances de TP, sur machine et d'une durée de une heure trente ; la seconde sous la forme d'un devoir à la maison personnel dont le sujet doit être choisi avec l'enseignant à la fin des séances de TD.

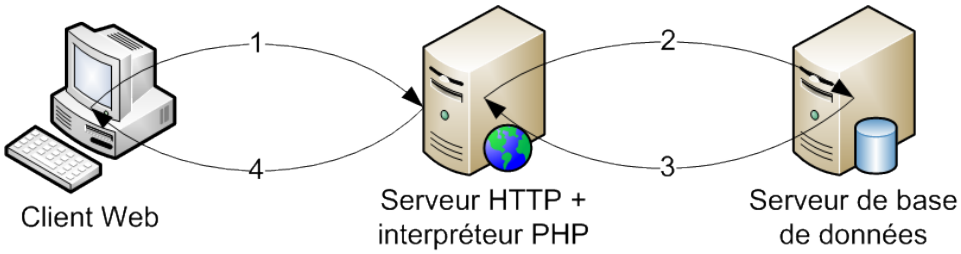
Chapitre 2. Présentation de l'architecture et des outils

Le développement Web repose sur une architecture distribuées, nous n'avons plus un seul programme sur une seule machine mais un ensemble réparti sur plusieurs machines communicantes.

1. Présentation

Nous allons utiliser une architecture Web composée d'un serveur HTTP (HyperText Transfer Protocol), d'un SGBD (Système de Gestion de Base de Données), d'un client Web et enfin de PHP. PHP est un langage de script libre permettant de produire des pages Web dynamiques[1]. Nous avons un client Web, un serveur Web, une application PHP et un SGBD qui contient les bases de données.

Figure 2.1. Architecture



L'obtention d'une page dynamique suit les étapes suivantes :

1. Tout commence par le client Web qui émet une requête vers un serveur HTTP.

2. Le serveur analyse la requête HTTP et en fonction de la ressource demandée renvoie cette ressource ou soustraite la gestion de la requête à une autre application. Pour ce qui nous intéresse, toutes les ressources en `.php` sont envoyées vers l'interpréteur PHP celui-ci va gérer la réponse soit directement soit en faisant appel à un SGBD.

3. Le SGBD fournit les enregistrements à l'interpréteur PHP.

4. L'interpréteur PHP génère la page Web, la fournit au serveur HTTP et enfin le serveur HTTP renvoie la réponse au client.

Pour l'ensemble de nos serveurs nous avons choisi une solution intégrée : WAMP (Windows, Apache, MySQL, PHP). WAMP est disponible à l'URL (Uniform Resource Locator) suivante : `http://www.wampserver.com/`. Il existe d'autres solutions intégrées aussi nommées plate-forme de développement Web, comme easyphp, XAMPP (X Apache MySQL Perl PHP), LAMP (Linux, Apache, MySQL, PHP), ...

Important

Malgré tous les efforts pour rendre le code indépendant de l'architecture, il ne l'est jamais totalement, il vous faut donc lors de vos projets tutorés connaître votre hébergeur et sa configuration.

Chaque serveur dispose de ses fichiers de configuration, avec un hébergement mutualisé, vous ne pourrez y avoir accès. Voici les serveurs que le nous allons mettre en oeuvre.

Apache
: Le serveur Web a pour principal fichier de configuration `httpd.conf`, vous le verrez en détail pendant les TP de réseaux. Nous ne l'utiliserons par directement mais nous utiliserons l'interface "graphique" de WAMP et des fichiers `.htaccess`. Les fichiers `.htaccess` sont des fichiers de configuration des serveurs Web Apache

[1] PHP est un langage interprété, impératif et objet depuis la version 1.5. Il peut être utilisé en local sans serveur Web.

utilisables sans droits d'administration et qui permettent de redéfinir des directives de `httpd.conf`. Les fichiers `.htaccess` sont généralement le seul moyen de redéfinir une configuration d'apache offert par votre hébergeur.

PHP
: Le principal fichier de configuration est `php.ini`, ce fichier permet au travers de directives de définir le comportement de PHP en spécifiant les extensions, les répertoires les chemins d'accès aux fichiers, ... Vous ne pourrez accéder au `php.ini` d'un fournisseur aussi certaines directives sont utilisables dans des fichiers `.htaccess`.

MySQL
: Le principal fichier de configuration est `my.ini`, nous ne l'utiliserons pas et nous administrerons le SGBD en utilisant une application PHP : *phpMyAdmin*.

Nous utiliserons comme l'année précédente eclipse mais avec une version préconfigurée pour le développement Web : *Eclipse for PHP Developers*.

2. Première mise en oeuvre

Nous allons créer et afficher notre première page Web dynamique puis nous créerons un compte et une base de données au sein du SGBD.

2.1. Apache et eclipse

Démarrer WAMP serveur (il lance l'ensemble des services) puis consulter l'URL : `http://localhost/`. Votre serveur apache fonctionne, mais comment pointer vers un répertoire de publication Web: en utilisant un alias.

1. Créer sous le disque "public" (`e:`) un répertoire nommé `m3.23.4` qui lui même contiendra le répertoire workspace (`e:\m3.23.4\workspace`).

2. Lancer eclipse et choisir le répertoire précédent comme workspace.

3. Créer un nouveau projet PHP nommé *td-tp-1*.

4. Créer un répertoire *www* et un sous-répertoire *tests* (`z:\m3.23.4\workspace\td-tp-1\www\tests`)

5. Créer un fichier PHP nommé `test.php` en utilisant un fichier PHP simple. Le contenu du fichier est le suivant :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="fr" xml:lang="fr">
  <head>
    <title>XHTML 1.0 Strict Test</title>
  </head>
  <body>
    <p>Pas interprété
    <?php phpinfo();?>
    </p>
  </body>
 </html>
```

6. Créer dans apache un alias nommé td-tp-1 pointant vers le répertoire physique `e:\m3.23.4\workspace\td-tp-1`. Ainsi apache pourra accéder aux ressources de votre projet. *Dans la pratique, pour des raisons de sécurité, seul les scripts devant apparaître dans l'URL doivent être dans le répertoire de publication et donc directement accessibles par les clients.*

7. Tester depuis eclipse.

PHP est un préprocesseur appelé par le serveur Web lorsqu'il rencontre une page en `.php`. Dans la page en `.php` le préprocesseur traite tout ce qui est dans la balise

```
<?php ?>
```

. `phpinfo();` est une fonction qui donne des informations sur la configuration courante de PHP.

Important

Avant la fin de chaque séance de TP ou de TD, il vous faudra exporter votre projet :

1. Se placer sur le projet
2. Exporter
3. Général
4. Archive file

Pour l'import il vous faudra importer un projet existant dans le workspace courant.

2.2. PHP

Nos serveurs sont des serveurs de développement, nous allons pouvoir les modifier en ce sens pour :

- afficher toutes ou parties des erreurs
- permettre le débogage sous eclipse

Commençons par l'affichage des erreurs :

1. Introduire une erreur dans `test.php`, par exemple

   ```
   echo 1/0;
   ```

   l'affichage du résultat d'une division par zéro.

2. Visualiser votre page
3. Dans le `php.ini`[2] (accessible via WamServer -> PHP) modifier

   ```
   error_reporting  =  E_ALL ;toutes les erreurs affichées
   ```

   en

   ```
   error_reporting  =  E_PARSE  ; uniquement les erreurs d'interprétation
   ```

4. Redémarrer apache (WampServer -> apache -> service -> redémarrer le service ou WampServer -> redémarrer tous les services)
5. Visualiser votre page, vous ne devriez pas voir d'erreur car

   ```
   echo 1/0;
   ```

   est syntaxiquement correcte.

6. Remettre `test.php` et `php.ini` dans leur état initial puis redémarrer *apache*. Nous souhaitons, le temps du développement, afficher nos erreurs.

Pour développer, il est pratique de disposer d'un débogueur :

1. Copier la dll (Dynamic Link Library) `ZendDebugger.dll`, fournie avec le sujet, dans `C:\wamp\bin\php\php5.2.9-2\ext\` (Windows dispose d'un nouveau programme)
2. Rajouter à la fin du `php.ini` les directives suivantes

   ```
   ; a rajouter après avoir copie la dll pour utiliser le débogueur
   zend_extension_ts=C:\wamp\bin\php\php5.2.9-2\ext\ZendDebugger.dll
   zend_debugger.allow_hosts=127.0.0.1/32,192.168.0.0/16
   zend_debugger.expose_remotely=allowed_hosts
   ```

ainsi PHP sait comment utiliser la dll.

3. redémarrer le serveur apache et enfin observer avec `phpinfo();` la présence de l'extension *zenddebugger*.

Dans un développement normal; trois ensembles de serveurs sont utilisés :

les serveurs de développement
: les serveurs de développement peuvent être personnel, ce qui est notre cas ou mutualisés. Ils sont dédiés au développement, souvent l'utilisateur dispose de droits étendus sur ces serveurs.

les serveurs de test
: ce sont les serveurs qui permettent d'anticiper les problèmes de déploiement, ils permettent une validation par l'équipe et par les clients. Ils peu

  vent être les dépôts d'un logiciel de gestion de versions.

les serveurs de productions
: Ce sont les serveurs finaux, aucune modification ne devrait normalement y être faite.

Nous nous contenterons des serveurs de développement, mais pour vos projets tutorés, il ne vous faudra pas oublier la partie production sur laquelle il est déconseillé de faire des modifications. Les modifications en production ne sont pas compatibles avec un déploiement.

2.3. MySQL

Pour administrer MySQL nous allons utiliser une application PHP : PHPMyAdmin.

1. Accéder à *PHPMyAdmin* avec l'URL : `http://localhost/phpmyadmin.`

2. Exécuter le script suivant qui va créer la base *user_db*, l'utilisateur *user* avec tous les droits sur cette base et ayant comme mot de passe *password*.

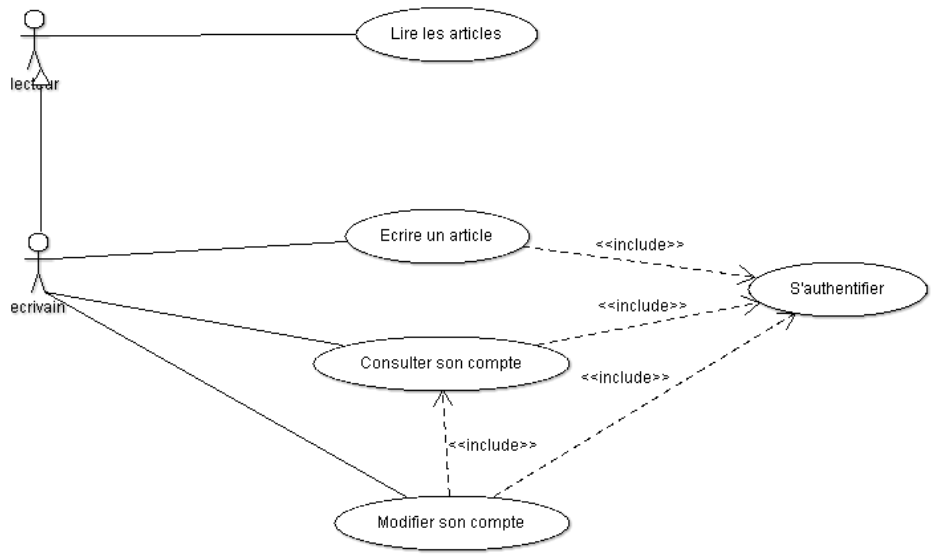
```
CREATE USER 'user'@'%' IDENTIFIED BY 'password';
GRANT USAGE ON * . * TO 'user'@'%' IDENTIFIED BY 'password'
       WITH MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0
       MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0 ;
CREATE DATABASE IF NOT EXISTS `user_db` ;
GRANT ALL PRIVILEGES ON `user_db` . * TO 'user'@'%';
```

Lors de la suite du développement vous pourrez utiliser PHPMyAdmin pour vérifier le résultat de vos scripts. *Nous avons choisi de ne pas utiliser le compte root de MySQL, pour des raisons de sécurité et pour rester conforme à l'approche d'un hébergement mutualisé où chaque utilisateur dispose de sa propre base.*

Chapitre 3. Développement

Nous souhaitons avoir un site où des écrivains authentifiés postent des articles que viennent consulter des lecteurs anonymes. Le diagramme des cas d'utilisations du front office pourrait-être :

Figure 3.1. Front office



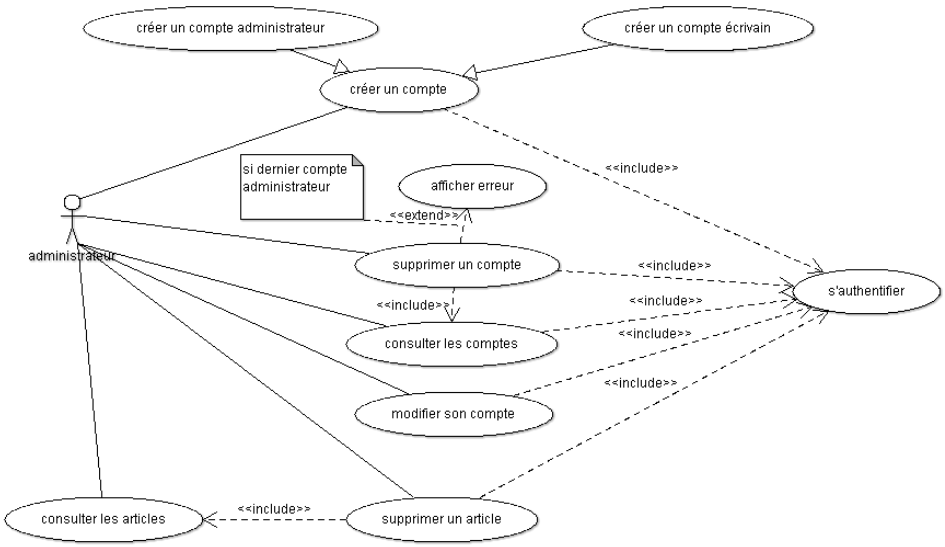
Un tel choix n'a pas de sens sans back office. Nous allons donc nous concentrer sur le back office.

Plutôt que de développer proprement, en séparent les différentes couches et donc les différents métiers, nous allons commencer avec une approche pragmatique ou tout sera confondu mais qui produira un résultat utilisable. Puis, nous verrons évoluer cette approche vers un code plus réutilisable mais aussi plus abstrait.

1. Présentation du sujet

Les acteurs de notre système sont des administrateurs qui vont pouvoir supprimer des articles et gérer les comptes écrivains (consultation, création, suppression). Nous allons nous baser sur le diagramme de cas d'utilisation suivant :

Figure 3.2. Back office



2. Présentation rapide du langage

Nous n'allons pas faire un cours détaillé de PHP mais simplement introduire les éléments qui nous seront utiles. Pour le moment nous savons qu'un script est compris dans la balise

```
<?php ?>
```

, il nous reste a trouver quoi y mettre. Les fonctions et les objets seront introduits plus finement ultérieurement.

2.1. Les types et variables

2.1.1. Présentation

PHP est un langage faiblement typé, les variables ont un type défini par leur contenu, il n'est pas nécessaire de spécifier un type. Le type est défini par le contexte d'utilisation mais des transtypages sont possibles. PHP dispose de huit types :

4 scalaires

boolean

```
True, False
```

integer
: entier relatifs, avec 0 comme préfix pour l'octal et 0x pour l'hexadécimal

float
: 0.1, 1.1e2, 4E-10

string
: 'une chaine', "une chaine avec évaluation des variables" ou encore avec la notation heredoc

```
public $bar = <<<EOT
bar
EOT;
```

Voici un petit exemple :

Exemple 3.1. Variables

```
<?php
$a;

var_dump($a); //NULL
$a=0;
var_dump($a); //int(0)
$a=$a+5;
var_dump($a); //int(5);
$a = $a + "un 10 zero";
var_dump($a); //int(5);

$a = $a+1.2;
var_dump($a); //float(6.2)

$A="toto";
var_dump($A); //string(4) "toto"
$A ='toto$a';
var_dump($A); //string(6) "toto$a"
$A = "toto$a";
var_dump($A); //string(7) "toto6.2"
//. est la concatenation
$A=$A."concat";
var_dump($A); //string(13) "toto6.2concat"


$b;
//isset permet de savoir si une variable est définie
var_dump(isset($b)); //bool(false)
$b = 1;
var_dump(isset($b)); //bool(true)

$c=true;
var_dump($c); //bool(true)
$c=True;
var_dump($c); //bool(true)
?>
```

2 composés

Les types composés comportent :

les tableaux

qui sont des tableaux associatifs de taille variable introduits par le "constructeur" array :

```
array( clé => valeur , ... )
```

Voici un exemple d'utilisation de tableau :

Exemple 3.2. Tableaux

```
<?php
$t = array(); //cree un tableau vide
$t[1]='un'; //associe à la clef 1 la valeur 1
$t['four']='quatre'; //associe à la clef 4 la valeur 4
$t['five']=5;
$t['clef']='valeur';
var_dump($t);
/*array(4)
 *  { [1]=>  string(2) "un"
 *  ["four"]=>  string(6) "quatre"
 *  ["five"]=>  int(5)
 *  ["clef"]=>  string(6) "valeur" }*/
echo count($t); //4

?>
```

les classes et les objets
: les principes sont les mêmes que pour le langage java, mais PHP reste plus limité, notamment pour la surcharge et la redéfinition. Voici l'exemple de la classe Point et son utilisation :

Exemple 3.3. Classe Point

```
class Point
{
 /**
  * @var unknown_type
  */
 private $x;
 private $y;

 public function  __construct($x=0, $y=0){
  $this->x= (int) $x;
  $this->y= (int) $y;
 }

 public function  __toString(){
  return "($this->x,$this->y)";
 }

 public function move($x,$y)
 {
  $this->x = (int) $x;
  $this->y= (int) $y;
 }
}

$p = new Point();

echo $p; //(0,0)
print_r ($p); //Point Object ( [x:private] => 0 [y:private] => 0 )
var_dump($p); //object(Point)#1 (2) { ["x:private"]=>  int(0) ["y:private"]=>  int(0) }

$p->move(2,4);
print_r($p); //Point Object ( [x:private] => 2 [y:private] => 4 )
$p2=$p;
$p2->move(0,0);
print_r($p); //Point Object ( [x:private] => 0 [y:private] => 0 )

$p3=clone $p;
$p3->move(2,2);
print_r($p); //Point Object ( [x:private] => 0 [y:private] => 0 )
?>
```

Vous noterez au passage le constructeur (__conbstruct), il existe de même les méthodes __destruct(), __clone(), ...

2 spéciaux

Ressource
: Une ressource est une variable spéciale, contenant une référence vers une ressource externe.

NULL
: En PHP une variable qui n'a pas encore reçue de valeur ou qui a été détruite(unset) est NULL.

La portée d'une variable dépend du contexte, dans un script c'est le script, dans une fonction c'est la fonction à moins que la variable ne soit définie comme étant globale. Les constantes peuvent aussi être statiques.

2.1.2. Réalisations

2.1.2.1. Exercice 1 (variables)

Créer dans le répertoire `test` un fichier `test1.php` qui doit traduire l'algorithme suivant :

```
var a
a <- "il faut beau"
b <- "tous les jours"
c <- "$a $b"
c <- c + 1
c <- c
```

Pour visualiser le résultat de votre algorithme (a, b restent inchangés, c est un entier qui vaut 1) vous pouvez utiliser var_dump() ou le déboguer en plaçant un point d'arrêt sur la dernière ligne.

2.1.2.2. Exercice 2 (tableaux indicés)

Nous allons commencer par des tableaux indicés, dans un tableau indicé, la clef est un entier. Pour tout ce qui est PHP vous pouvez vous référer à l'URL : `http://www.php.net`. Pour les tableaux la page suivante peut vous aider : `http://fr2.php.net/manual/fr/language.types.array.php`.

Reproduire dans le fichier `test2.php` le tableau et vérifier son contenu avec le débugger.

Tableau 3.1. $tab

-1	'moins un'
0	'zéro'
1	'humour'

2.1.2.3. Exercice 3 (tableaux associatifs)

Un tableau associatif est un tableau qui n'est plus accédé par un indice mais par une clef.

Exemple 3.4. Tableau associatif

```
$map = array( 'version'    => 4,
              'OS'         => 'Linux',
              'lang'       => 'english',
              'short_tags' => true
            );
```

Dans l'exemple précédent `$map['version']` vaut 4, `$map['OS']` vaut 'Linux'.

Dans un fichier nommé `test3.php` créer un tableau de trois lignes qui a un email associe un mot de passe.

2.1.2.4. Exercice 4 (tableaux multidimensionnels)

Voici un extrait de la documentation PHP,

```
$fruits = array ( "fruits"  => array ( "a" => "orange",
                                       "b" => "banana",
                                       "c" => "apple"
                                     ),
                  "numbers" => array ( 1,
                                       2,
                                       3,
                                       4,
                                       5,
                                       6
                                     ),
                  "holes"   => array (      "first",
                                       5 => "second",
                                            "third"
                                     )
                );
```

correspondant au tableau :

Tableau 3.2. Exemple de tableau multidimensionnel

"fruits" =>	"a" => "orange"

	b" => "banana"
	"c" => "apple"
numbers =>	0 => 1
	1 => 2
	2 => 3
	3 => 4
	4 => 5
	5 => 6
"hole" =>	0 => "first"
	5 => "second"
	6 => "third"

En vous inspirant de l'exemple précédent, coder dans le fichier `test4.php` le tableau `$user_fileds` :

Tableau 3.3. Tableau multidimensionnel $user_fileds à coder

"email" =>	'type' => 'text'
	'length' => 20
	'notnull' => true
"password" =>	'type' => 'text'
	'length' => 20
	'notnull' => true
"level"=>	'type' => 'integer'
	'unsigned' => true
	'notnull' => true
	'autoincrement' => true

2.2. Les constantes

Les constantes en PHP sont introduites avec la fonction `define(nom, valeur)` ou avec le mot clef **const**.

Exemple 3.5. Définition et utilisation d'une constante

```
const CONSTANT = 'Hello World';
define('CONSTANTE' = "Bonjour le monde";

echo CONSTANT; //noter la disparition du $
echo CONSTANTE;
```

Il existe aussi des variables magiques qui renseignent sur l'état de l'interpréteur : `__FILE__` (le fichier courant), `__LINE__` (la ligne courante), ...

2.3. Les structures de contrôles

Sans faire de cours sur les expressions et sur les opérateurs de comparaison vous devez savoir que :

$a == $b
: est vraie si la valeur de `$a` et égale à la valeur de `$b`,

$a === $b
: est vraie si la valeur de `$a` et égale à la valeur de `$b` et si `$a` et `$b` sont de même type.

Vous pouvez utiliser les mêmes opérateurs logiques qu'en java : **&&** (AND), || (OR), **!** (Not).

2.3.1. Présentation

A l'exception des inclusions et du **foreach**, vous connaissez déjà les structures qui vont suivre, nous allons en donner un rappel sous forme de syntaxe et d'exemple.

2.3.1.1. if, if else, if elseif

En PHP vous retrouverez à peut les mêmes structure qu'en *java* , il existe cependant des écritures alternatives comme par exemple pour la conditionnelle :

```
(cond) ? commande1 : commande2 )
```

Je vous conseil de ne pas utiliser les écritures alternatives.

if (expression) commandes

Exemple 3.6. Exemple d'utilisation du if

```
if($a > $b)
    echo $a." est plus grand que ".$b;
```

if (expression) commandes else commandes

Exemple 3.7. Exemple d'utilisation du if else

```
if($a > $b):
    echo $a." est plus grand que ".$b;
else
    echo $a. "est plus petit ou egale à".$b;
```

if (expression) commandes elseif (expression) commandes

Exemple 3.8. Exemple d'utilisation du elseif

```
if ($a > $b) {
    echo "a est plus grand que b";
} elseif ($a == $b) {
    echo "a est égal à b";
} else {
    echo "a est plus petit que b";
}
```

2.3.1.2. switch

Le switch est une structure pratique qui comme son nom l'indique permet un aiguillage sans un grand nombre de if.

Exemple 3.9. Exemple d'utilisation du switch

```
<?php
switch ($i){
    case 0:
        echo "i égal 0";
        break;
    case 1:
        echo "i égal 1";
        break;
    case 2:
        echo "i égal 2";
        break;
    default:
        echo "i n'est ni égal à 2, ni à 1, ni à 0";
}
?>
```

2.3.1.3. while, do while

Nous utiliserons les syntaxes **while (expression)** commandes et **do commandes while (expression);**, d'autres syntaxes existent en PHP.

Exemple 3.10. Exemple de boucle while

```
$i = 1;
while ($i <= 10) {
    echo $i++;  /* La valeur affiche est $i avant l'incrémentation
                   (post-incrémentation)  */
}
```

Exemple 3.11. Exemple de boucle do while

```
$i = 0;
do {
    echo $i;
} while ($i > 0);
```

2.3.1.4. for, foreach

for (expr1; expr2; expr3) commandes

Exemple 3.12. Exemple de boucle for

```
for ($i = 1; $i <= 10; $i++) {
    echo $i;
```

La boucle **for** n'est pas simplement utilisable pour parcourir un tableau associatif, la boucle **foreach** répond à ce problème, la syntaxe est **foreach (array_expression as $value) commandes** ou **foreach (array_expression as $key => $value) commandes**.

Exemple 3.13. Exemple de boucle foreach

```
$arr = array("un", "deux", "trois");
foreach ($arr as $key => $value) {
    echo "Clé : $key; Valeur : $value<br />\n";
}
/*
Clé : 0; Valeur : un
Clé : 1; Valeur : deux
Clé : 2; Valeur : trois
 */

foreach ($arr as $value) {
    echo "Valeur : $value<br />\n";
}
/*
Valeur : un
Valeur : deux
Valeur : trois
 */

$arr = array('velo'=>'rouge', 'voiture'=>'bleu');
foreach ($arr as $key => $value) {
    echo "Clé : $key; Valeur : $value<br />\n";
}
/*
Clé : velo; Valeur : rouge
Clé : voiture; Valeur : bleu
 */
```

2.3.1.5. require, include, require_once, include_once

Un script peut faire appel à d'autres scripts, **require** et **include** permettent de réaliser ces appels. **include** contrairement à **require** ne conduit pas à une erreur en cas de non chargement du script. D'inclusion en inclusion, une

boucle peut se produire et des doubles définitions apparaitre, **require_once** et **include_once**, ne conduisent pas à une erreur en cas de double définition.

2.3.2. Exercices

2.3.2.1. Exercice 5 (if, switch

Ecrire un script `test5.php` qui si la variable `$action` vaut "C" affiche "Create", si elle vaut "R" affiche "Read", si elle vaut "U" affiche "Update", si elle vaut "D" affiche "Delete". Vous pouvez utiliser la structure de contrôle de votre choix. La méthode echo permet une sortie vers la page Web, ne sachant pas encore lire depuis un formulaire, vous affectez des valeurs à $action pour tester.

2.3.2.2. Exercice 6 (for, foreach, require)

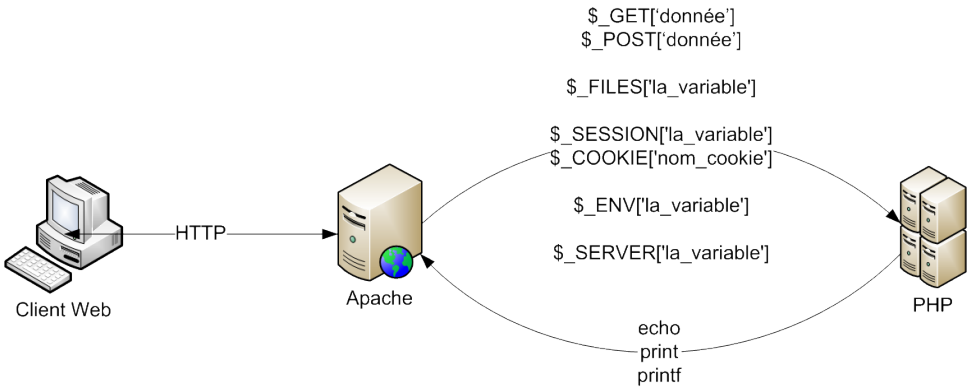
Ecrire un script `test6.php` qui en utilisant **require**, **for** ou **foreach** affiche le contenu du tableau de la question 2.

2.3.2.3. Exercice 7 (foreach et require_once)

Ecrire un script `test7.php` qui en utilisant **require**, **for** ou **foreach** affiche le contenu du tableau de la question 3.

3. La communication entre le serveur Web et PHP

Figure 3.3. Communication entre le serveur Web et PHP



3.1. Présentation

Le serveur Web et PHP communique, PHP reçoit des informations du serveur Web qui lui même en tient une partie du client. Pour ce qui nous intéresse ces informations sont dans les tableaux associatifs : $_GET, $_POST et $_REQUEST (un tableau associatif qui contient par défaut le contenu des variables $_GET, $_POST et $_COOKIE). L'écriture est réalisée avec **echo** ou **print** ou **printf**.

3.2. Réalisation (exercice 8)

En utilisant le formulaire suivant stocké dans le fichier `login_formulaire.html` faire, dans `test8.php`, afficher les paramètres passés dans le formulaire, le script doit aussi marcher pour la méthode POST. Vous pouvez utiliser la méthode **isset($var)** qui permet de savoir si une variable à une valeur.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.d
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Insert title here</title>
</head>
<body>
```

```
<form action="test8.php" method="get">
 <p>
 login :
 <input name="login" type="text" value="toto@dom"/>
 password :
 <input name="password" type="password" value="passtoto"/>
 <br />
 <input type="submit" value="login"/>

 </p>
</form>
</body>
</html>
```

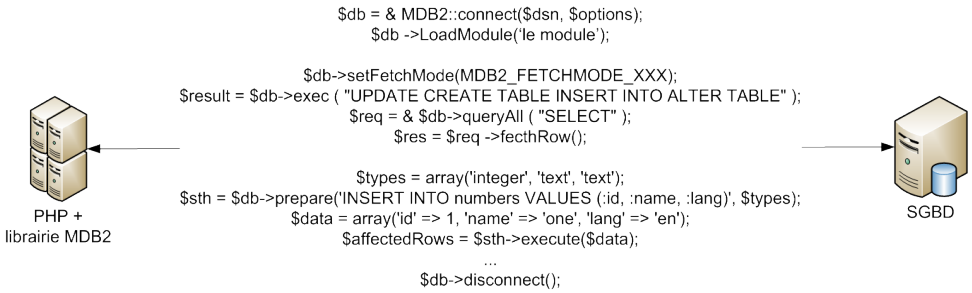
4. La communication entre PHP et le SGBD

PHP peut créer ses pages en sollicitant un SGBD, il existe pour ce faire deux approches :

Communication directe
: utiliser des commandes spécifiques au SGBD comme **mysql_query, pg_query**, ... Le problème est que le code est fortement lié au SGBD.

Utiliser une couche d'abstraction
: la couche d'abstraction permet de s'affranchir du SGBD, nous allons utiliser PEAR:MDB2 l'une d'entre elle. Il en existe d'autre comme PDO, ou EZPDO.

Figure 3.4. Communication entre PHP avec la bibliothèque MDB2 et un SGBD



4.1. Présentation de MDB2

La classe MDB2 de PEAR est une couche d'abstraction de bases de données. Elle est fortement supportée, vous pourrez trouver la documentation à l'URL suivante : `http://pear.php.net/package/MDB2`. La bibliothèque est riche et permet des extensions sous formes de modules, nous allons en présenter une utilisation simple.

4.1.1. La connexion

Pour pouvoir utiliser le SGBD, il faut se connecter[1] :

```
//inclusion d'un fichier de constante et redefinition du php.ini
 require_once "../../../config/inc.config.php";

 //inclusion de mdb2
 require PEAR_PATH."mdb2.php";

 //connexion
 //le sgbd://user:password@serveur/base_de_donnees
 $dsn="mysql://user:password@localhost/user_db";

$options = array(
    'debug'       => 2,
    'portability' => MDB2_PORTABILITY_ALL
```

[1]Le code qui suit est un exemple, il vous faudra attendre d'avoir importer un nouveau projet avec les bibliohèques pour pouvoir l'utiliser.

```
);

 $mdb2 = & MDB2::connect($dsn, $options);
```

Il nous a fallu, indiquer où se connecter (`$dsn`), avec quelles options (`$options`) et récupérer une référence (&) vers une connexion (`$mdb2`). Le

```
&
```

que vous avez vu apparaître permet d'obtenir une référence vers la connexion. Les références permettent les alias et sont possibles en PHP sur tous les types.

Nous allons maintenant utiliser notre connexion pour préparer et exécuter une requête.

4.1.2. La préparation de la requête et son exécution

Nous allons utiliser des requêtes préparées à savoir de requêtes à trous qui attendent des paramètres. Les requêtes préparées nous soulagent de la gestion de l'encodage et améliorent la sécurité.

```
/utilisation d'une requête préparée
    $type_passe=array('text'); //les types des arguments passes en parametres
    $type_retourne=array('text','text'); //les types retour de la requête
    //la requête
    $sth = $mdb2->prepare('SELECT email,password FROM user where email=:emailParametre',
           $type_passe,$type_retourne);
 //un jeu de paramètre
    $data = array('emailParametre'=>'jfb@ujf-grenoble.fr');

 $res = $sth->execute($data); //execution de la requête avec les paramètres choisi
 if (PEAR::isError($res))
     die($res->getMessage());
```

Ici, nous n'avons qu'un paramètre *:email*, son type est *text* (`$type_passe`). Nous attendons deux colonnes de type texte (`$type_retourne`). La requête est **SELECT email, password FROM user where email=:emailParametre** avec *:email* encore inconnu. Nous associons au paramètre *:email* la valeur 'jfb@ujf-grenoble.fr'. Tous les paramètres sont fournis, la requête est prête pour l'exécution (**$res = $sth->execute($data)**). Les documentations PHP utilisées sont :

mixed &prepare(string $query, [mixed $types = null], [mixed $result_types = null], [mixed $lobs = array()])
: `prepare` renvoie une référence dont le type peut varier (mixed), la méthode prend obligatoirement une requête (`$query`), optionnellement, elle peut prendre les types fournis, ceux des paramètres (`$types`), les types renvoyés (`$result_types`) et enfin un dernier paramètre optionnel qui contient des associations clef valeur ($lobs). L'utilisation du

  ```
  =
  ```

  au niveau des paramètres permet de définir une valeur par défaut.

mixed &execute([array $values = null], [mixed $result_class = true], [mixed $result_wrap_class = false])
: execute permet d'exécuter la requête et d'obtenir suivant la requête un tableau, le nombre de lignes modifiées. Nous n'utiliserons que le premier paramètre qui contient les associations entre le nom des paramètres de la requête et leur valeur.

Une requête de sélection renvoie un tableau, nous allons le parcourir pour l'afficher.

4.1.3. Le traitement de la requête

Pour parcourir l'ensemble des lignes de résultat il nous faut une boucle. **fetchrow()** permet de se placer sur la ligne suivante (enregistrement) si elle existe et si elle n'existe pas de renvoyer faux. Le mode associatif(`MDB2_FETCHMODE_ASSOC`) permet de récupérer les valeurs dans un tableau associatif.

```
while (($row = $res->fetchRow(MDB2_FETCHMODE_ASSOC)))
    {
     echo $row['email'] . "\n";
     echo $row['password']."\n";
```

```
    }
```

$row['email'] correspond à la colonne email de l'enregistrement courant.

4.1.4. La déconnexion

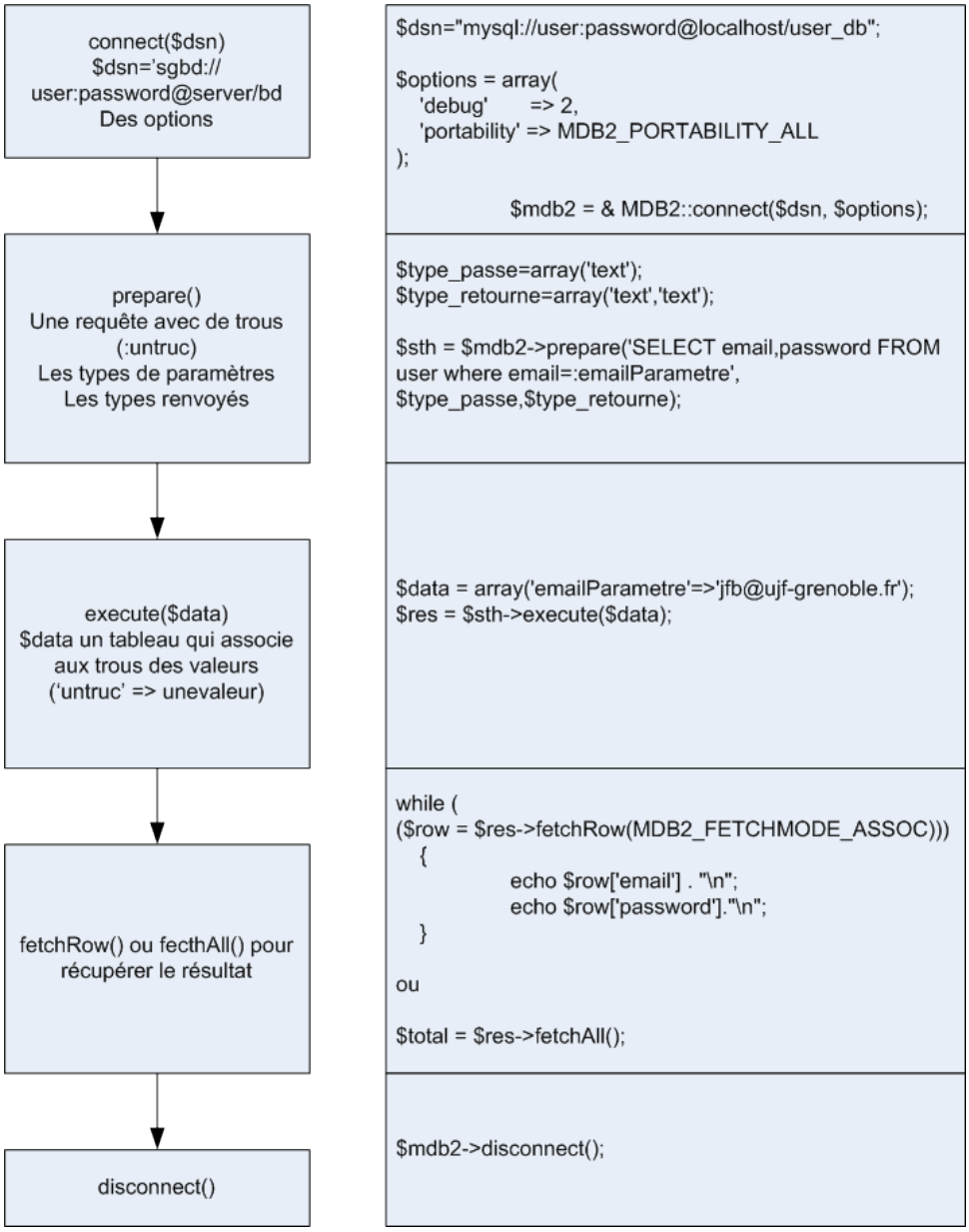
Une fois une ressource utilisée, il faut la libérer :

```
//deconnexion
 $mdb2->disconnect();
```

4.1.5. En résumé

En résumé nous avons les étapes suivantes :

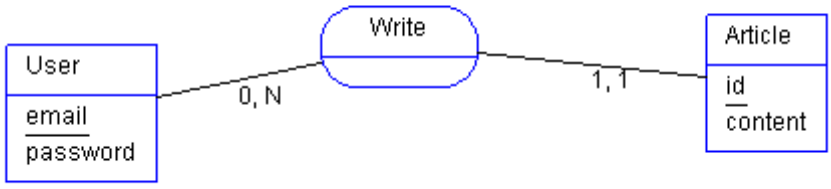
Figure 3.5. Accès au SGBD



4.2. Réalisations

Nous allons utiliser le MCD (Modèle Conceptuel de données) suivant :

Figure 3.6. MCD



Ce qui nous donne le MLD (Modèle Logique de données) :

Figure 3.7. MLD

Nous pourrions à partir de ce MLD créer nos tables dans MDB2, ce qui ne constitue pas l'approche, la plus simple. Nous allons donc utiliser *PHPMyAdmin* pour créer les tables avec le SQL suivant :

```
CREATE TABLE user (email varchar(20) NOT NULL, password varchar(20));

CREATE TABLE article (id int NOT NULL, content varchar(2048), email varchar(20));

ALTER TABLE user ADD CONSTRAINT PK_user PRIMARY KEY (email);

ALTER TABLE article ADD CONSTRAINT PK_article PRIMARY KEY (id);

ALTER TABLE article ADD CONSTRAINT FK_article_email
         FOREIGN KEY (email) REFERENCES user (email);
```

Dans le même esprit nous allons remplir nos tables avec des jeux de test :

```
INSERT INTO `user` (`email`, `password`) VALUES
('vl@ujf-grenoble.fr', 'passvl'),
('jfb@ujf-grenoble.fr', 'passjfb');

INSERT INTO `user_db`.`article` (
`id` ,
`content` ,
`email`
)
VALUES (
'1', 'etre directeur des études', 'vl@ujf-grenoble.fr'
), (
'2', 'etre informaticien au 22ème siecle', 'vl@ujf-grenoble.fr'
);
```

MDB2 repose sur des bibliothèques, elles sont fournies en standard avec WAMP mais ne sont pas forcement installées chez votre hébergeur. Nous avons préparé un projet, contenant les bibliothèques qu'il ne vous reste plus qu'a importer : `td-tp-2`.

4.2.1. Sans fonction

Vous placerez vos script dans le répertoire www/tests/scripts, vous pouvez inclure dans vos scripts, le fichier config/config.inc.php qui contient des constantes permettant d'utiliser la bibliothèque PEAR:MDB2.

4.2.1.1. Exercice 9 Afficher un utilisateur

Écrire le script `test9.php` qui permet d'afficher l'utilisateur d'email "vl@ujf-grenoble.fr", le code SQL est **SELECT * FROM user where email='vl@ujf-grenoble.fr';** Le code donné en exemple devrait vous aider, 'vl@ujf-grenoble.fr' est un paramètre de la requête.

4.2.1.2. Exercice 10 Afficher tous les utilisateurs

Écrire le script `test10.php` qui permet d'afficher tous les utilisateurs, le code SQL est **SELECT * FROM user;** Comme vous pouvez le constater, il n'y a pas de paramètres donc pas de type de retour :

- la méthode `prepare` de `$mdb2` prend deux paramètres : le SQL et les types renvoyés.
- la méthode `execute` de la requête `$sth` ne prend pas de paramètre.

Bien que cela soit contraire à la séparation des couches, si vous voulez un affichage plus lisible, vous pouvez remplacer :

```
while (($row = $res->fetchRow(MDB2_FETCHMODE_ASSOC)))
    {
     echo $row['email'] . "\n";
     echo $row['password']."\n";
    }
```

par

```
echo '<table>';
    while (($row = $res->fetchRow(MDB2_FETCHMODE_ASSOC)))
    {
     echo '<tr>';
     echo '<td>'. $row['email'] . '</td>';
     echo '<td>'. $row['password'] .'</td>';
     echo '</tr>';
    }
 echo '</table>';
```

4.2.1.3. Exercice 11 Insérer un utilisateur

Écrire le script `test11.php` qui permet d'insérer l'utilisateur *toto@dom* ayant comme mot de passe *passtoto*. Vous pouvez vous inspirer du SQL du jeu de test donné précédemment. Votre requête doit prendre deux paramètres (*:email*, *:password* par exemple) de type texte. Vous pouvez utiliser *PHPMyAdmin* ou le script `test10.php` pour tester. *L'email étant une clef primaire, une deuxième exécution du script doit lever une contrainte d'intégrité.*

4.2.1.4. Exercice 12 Supprimer un utilisateur

Écrire le script `test12.php` qui permet de supprimer l'utilisateur *toto@dom*. L'email doit être un paramètre de la requête. SQL la suppression se traduit comme suit : **DELETE FROM table WHERE contrainte**.

4.2.1.5. Exercice 13 Modifier un utilisateur

Écrire le script `test13.php` qui permet de changer le mot de passe en *totopass*. Pour finir, nous allons réaliser une mise à jour, un UPDATE, ainsi nous aurons fait le tour. Le code SQL d'un Update est : **UPDATE table SET attribut=valeur WHERE contrainte**.

4.2.2. Avec des fonctions

Comme vous vous en êtes rendus compte, le code se répète et il serait heureux de pouvoir en réutiliser des fragments avec des paramètres.

4.2.2.1. Présentation

En PHP, une fonction doit-être déclarée avant d'être appelée. Lors de l'appel des paramètres peuvent êtres fournis et des valeurs renvoyées. La portée des variables introduites dans une fonction est la fonction sauf si elles sont explicitement rendues globales (**global**) . Il est possible d'avoir des variables statiques (**static**). Les fonctions ne peuvent-être surchargées[2].

4.2.2.1.1. Déclaration

La déclaration est réalisée en utilisant le mot clef function

```
<?php

function nomDeMaFonction( $argument1, $argument2 ) {
  // code
}

?>
```

4.2.2.1.2. Appels

L'appel est réalisé par le nom de la fonction, si il n'y pas de valeur par défaut tous les arguments doivent-être fournis.

```
<?php
$unAutreTruc = 22;
nomDeMaFonction( 'unTruc', $unAutreTruc );
?>
```

4.2.2.1.3. Arguments

Les arguments peuvent être passés par valeur (une recopie) ou par référence. Le passage par référence est choisi en uitiisant

```
&
```

.

```
<?php
function valeur($val) {
  $val = 'par valeur ' . $val;
}

$truc = 'un truc';
valeur($truc);

echo $truc; //un truc

function reference(&$val) {
$val = 'par reference ' . $val;
}

reference($truc);
echo $truc; // par reference un truc
?>
```

Les paramètres optionnels sont possibles en utilisant des valeurs par défaut :

```
<?php
function moyenne( $arg1 = 0, $arg2=1) {
  echo ($arg1 + $arg2)/2;
}

moyenne(10,2); //6
moyenne(); //0.5
moyenne(10)//5.5 le premier argument à pris sa valeur par défaut
?>
```

[2]La surcharge via __call peut-être utilisée au niveau des méthodes et autre particularité du langage il est possible d'assigner une variable à une fonction.

4.2.2.1.4. Retour

Une fonction peut renvoyer une valeur en utilisant le mot clef **return**.

```
<?php
function moyenne( $arg1 = 0, $arg2=1) {
  return ($arg1 + $arg2)/2;
  echo 'jamais';
}

echo moyenne(10,2); //6

?>
```

4.2.2.2. Réalisations

Nous n'allons pas réaliser l'ensemble des fonctions mais seulement la connexion (`connect`) et la récupération de tous les utilisateurs (`getAllUsers`). Les deux fonctions seront placées dans le fichier `fonctions_user.php` du répertoire `fonctions`. Il est recommandé pour des raisons de sécurité que les bibliothèques soient non accessibles via un navigateur mais uniquement depuis un autre script. Le scripts qui testera nos deux fonctions se nomme `test_fonction_user` et est placé dans `www/tests/scripts`. Les inclusions seront pour le test :

```
require_once('../../../config/inc.config.php');
require_once('../../../functions/functions_user.php');
```

et pour les fonctions :

```
require_once(PEAR_PATH.'mdb2.php');
```

4.2.2.2.1. Exercice 14 (connexion)

La méthode de connexion `&connect($host='localhost', $database, $user, $password)` doit renvoyer une référence vers une connexion. Il ne vous faut pas oublier d'inclure PEAR::MDB2.

4.2.2.2.2. Exercice 15 (getAllUsers)

La méthode permettant de récupérer tous les utilisateurs `$array() getAllUser($mdb2)` prend en paramètre une connexion et renvoie le tableau des utilisateurs. Vous pouvez utiliser `fetchAll()` sur le résultat d'une requête pour récupérer un tableau contenant les enregistrements (

```
$res->fetchAll();
```

).

5. Les classes et les objets

Nos pages vont manipuler des utilisateurs et des articles, il est possible de stocker les informations dans des tableaux mais nous pouvons faire mieux en utilisant des objets.

5.1. Présentation

PHP permet une programmation orienté objet avec la notion d'héritage, de surcharge, de classe abstraite et d'interface. Contrairement au type classique, il est possible d'imposer un typage objet aux paramètres.

5.1.1. Syntaxe de base

Nous n'allons pas refaire un cours sur la programmation objet mais simplement énoncer les mots clefs, les spécificités de PHP et étudier un exemple.

5.1.1.1. Mots clef

Les mots clef sont très proches de ceux que vous avez utilisés en java nous avons :

class
: permet de définir une classe

  ```
  class MaClasse{
  ```

```
}
```

extends
: permet de définir une classe comme étant une classe fille ("une sorte de")

```
class MaFille extends LaMere{
}
```

L'héritage multiple n'est pas possible, une classe fille n'a qu'une mère et elle dispose de tout ce dont disposé la classe mère plus ces spécificités : ajout et redéfinitions

function
: permet de définir une méthode.

abstract
: Permet de définir une classe ou une méthode comme étant abstraite, le prototype est connu mais pas le code. Une classe qui contient au moins une méthode abstraite est abstraite et une classe abstraite ne peut-être instanciée, il faut d'abord en hériter et définir les méthodes qui ne l'étaient pas.

```
abstract class MaClassAbtraite{
  abstract function uneMéthodeAbstraite
}
```

interface
: Une interface permet de créer un modèle que les classes qui l'implémentent doivent respecter. Une interface est une classe abstraite pure.

```
interface MonInterface {
}
```

implements
: Permet d'implémenter une ou plusieurs interfaces

```
class UneClasse implements MonInterface{
}
```

les méthodes de l'interface doivent-être implémentées.

public, protected, private
: Les modificateurs de visibilité public pour tous, protected pour les classes filles et private pour la classe elle-même.

```
private $connexion;
public function connect()
{
}
```

const
: Pour définir des constantes de classe

```
const CONST_TRUC = 'Un truc constant';
```

static
: Pour définir un attribut statique (partagé par toutes les instances) ou pour définir une méthode de classe.

self, parent
: permettent de résoudre avec l'opérateur de résolution (::) la classe elle-même et la classe mère.

```
class Etudiant
{
    protected function mange()
    {
        return 'Je mange mal';
    }
}

class Etudiant_Doctorant extends Etudiant
{
    protected function mange()
```

```
    {
        return 'Je mange bien';
    }

    public function identifierParent()
    {
        return parent::mange(); //je mange mal
    }

    public function identifierSelf()
    {
        return self::mange(); //je mange bien
    }
}
```

this
: l'objet courant

new
: permet de créer une nouvelle instance. La méthode __construct de la classe de l'instance est utilisée.

5.1.1.2. Spécificité

L'opérateur de résolution de portée est ::, il permet aussi d'accéder aux attributs statiques. L'accès à une méthode de classe est réalisé avec ->. Le constructeur se nomme __construct, __toString permet un affichage personnalisé. Pour information d'autres méthodes magiques existent, nous ne les utiliserons pas, en voici un extrait :

__destruct
: le destructeur

__set(), __get()
: déclenchées lors des accès en écriture ou en lecture à une propriété inexistante de l'objet ;

__call()
: déclenché lors de l'appel à une méthode inexistante (permet la surcharge)

__callstatic()
: déclenchée lors de l'appel à une méthode statique

__isset(), unset()
: déclenchées sur l'utilisation de isset() et unset() sur un attribut

__clone()
: déclenchée lorsque l'on essaye de cloner un objet

5.1.1.3. Exemple

Nous allons traiter la classe User et son utilisation.

```
/**
 * @author jub
 * La classe User représente un utilisateur défini par son email et son mot de passe
 */
class User {
 /**
  * @var string l'email
  */
 private $email;
 /**
  * @var string le mot de passe
  */
 private $password;

 /**
  * @param $email l'email
  * @param $password le mot de passe si il est disponible, la chaine vide sinon
  * @return rien
```

```
 */
function __construct($email, $password='')
{
 $this->email = (string) $email; //un transtypage est utilisé pour avoir une chaine
 $this->password = (string) $password;
}

/**
 * @return string l'email
 */
function getEmail()
{
 return $this->email;
}

/**
 * @return string le mot de passe
 */
function getPassword()
{
 return $this->password;
}


/**
 * @param $email le nouvel email
 * @return rien
 */
function setEmail($email)
{
 $this->email = (string) $email;
}

/**
 * @param $password le nouveau mot de passe
 * @return rien
 */
function setPassword($password)
{
 $this->password = (string) $password;
}

/**
 * @return string une représentation de l'utilisateur
 */
function __toString()
{
 return "email : $this->email password : $this->password";
 }
}
```

Voici un script de teste et son résultat :

```
$u1 = new User('vl');
echo ''.$u1; //par utilisation de __toString email : vl password :
$u2 = $u1;
$u2->setPassword('pass');
echo $u2->getPassword();  //'pass'
echo $u1->getPassword();  //''
```

5.2. Réalisation

Nous allons commencer par écrire un autre programme de test de la classe User, puis nous coderons la classe article. Vous pouvez importer le projet td-tp-3 et réaliser un alias.

5.2.1. Exercice 16 utilisation d'une classe

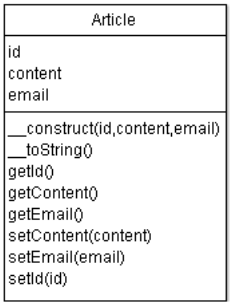
En utilisant la classe User créer un utilisateur toto avec comme mot de passe passtoto, puis créer titi sans mot de passe puis modifier titi en lui donnant un mot de passe : passtiti. Enfin faite afficher les deux utilisateurs. Vous

placerait User.php dans le répertoire classe et test16.php dans www/tests. Il ne vous faut pas oublier de d'inclure la classe User.php dans le script test.php.

5.2.2. Exercice 17 création d'une classe

Coder la classe correspondant au diagramme de classe UML suivant :

Figure 3.8. Diagramme de classe Article



Dans le constructeur $content aura comme valeur par défaut la chaîne vide.

6. Les classes métiers

Nous avons des objets (article et user) qui représentent notre système. Il nous faut maintenant les lier au SGBD. Le lien est réalisé par des classes et des objets métiers. Un objet métier est un concept ou une abstraction ayant un sens pour des acteurs. L'objet métier permet de décrire les entités manipulées par les acteurs dans le cadre de la description du métier. Pour nous les objets métiers sont article et user. Pour faciliter leur manipulation, nous allons utiliser deux classes : GestionBDUser et GestionBDArticle. Vous pouvez importer le projet td-tp-4 et créer un alias. Vous remarquerez une nouvelle structuration, nous avons un répertoire class qui contient un répertoire model (avec User_classe.php et Article_classe.php) et enfin un répertoire GestionBD qui contiendra nos classe de gestion de la base de données.

6.1. Présentation

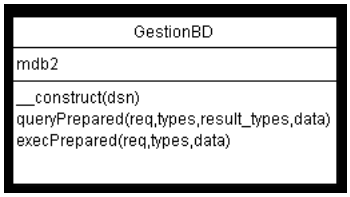
La classe GestionBD vous est fournie :

```
<?php

/**
 * @author jub
 *Classe de gestion d'une base de données
 *Elle facilite l'utilisation des requêtes préparées
 */
class GestionBD {

    /**
     * @var unknown_type la connexion à la base de données
     */
    protected $mdb2 = null;

    /**
     * @param $dsn l'URL de connexion
     */
    public function __construct($dsn) {

        $options = array(
            'debug' => 2,
            'result_buffering' => true,
        );
        $mdb2 = & MDB2::singleton($dsn,$options);

        if (MDB2::isError($mdb2)) {
            die($mdb2->getDebugInfo() & $mdb2->getMessage());
```

```
        }
        $this->mdb2 = $mdb2;
    }

    /**
     * Permet des requête péparées de type SELECT
     * @param $req la requête SQL avec ses paramètres
     * @param $types les types des paramètres
     * @param $result_types les types renvoyés
     * @param $data le lien entre les paramètres et leurs valeures
     * @return unknown_type un resultset un tableau qui contient les enregistrements
     */
    public function queryPrepared($req,$types,$result_types,$data) {
       /*echo 'requête '.$req. '<br/>';
        echo 'types '; print_r($types); echo '<br/>';
        echo 'result_types'; print_r($result_types); echo'<br/>';
        echo 'data '; print_r($data); echo '<br/>';
        */
     $statement = $this->mdb2->prepare($req,$types, $result_types);

        $resultset = $statement->execute($data);
        //echo $this->mdb2->last_query;
        if (MDB2::isError($resultset)) {
         die($resultset->getMessage());
        }
        return $resultset;
    }

    /**
     * Permet des requêtes de type UPDATE, INSERT, DELETE, CREATE, DROP
     * @param $req la requête SQL avec ses paramètres
     * @param $types les types des paramètres
     * @param $data le lien entre les paramètres et leurs valeures
     * @return unknown_type généralement le nombre de lignes modifiées
     */
    public function execPrepared($req,$types,$data) {
     /*echo 'requête '.$req. '<br/>';
        echo 'types '; print_r($types); echo '<br/>';
        echo 'data '; print_r($data); echo '<br/>';
        */
        $statement = $this->mdb2->prepare($req,$types, MDB2_PREPARE_MANIP);
        $affectedRows = $statement->execute($data);
        //echo $this->mdb2->last_query;
        if (MDB2::isError($affectedRows)) {
            die($affectedRows->getMessage());
        }
        return $affectedRows;
    }

    public function getMdb2()
    {
     return $this->mdb2;
    }
}
?>
```

La classe GestionBD n'est qu'une reformulation objet des requêtes préparées, la classe n'a qu'un attribut `$mdb2` la connexion et dispose de deux méthodes `execPrepared()` pour les requêtes de mise à jour et `queryPrepared()` pour les requêtes de sélection.

Figure 3.9. Classe GestionBD

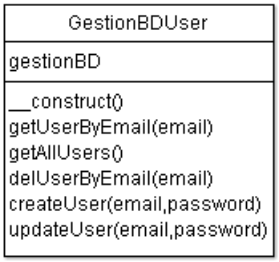


Pour gérer nos utilisateurs nous souhaitons avoir une classe disposant de méthode comme getUserByEmail, getAllUser, ... Cette classe reposant sur `GestionBD` vous est donnée :

```
<?php

//require_once '../../../config/inc.config.php';
require_once(PEAR_PATH."MDB2.php");
require_once('GestionBD_classe.php');
require_once(MODEL_PATH.'User_classe.php');

class GestionBDUser {
 private $gestionBD;

 function __construct()
 {
  $this->gestionBD = new GestionBD(DSN);
 }

 function getUserByEmail($email)
 {
  $req= 'select email, password from User where email=:email';
  $types = array('text');
  $result_types = array('text', 'text');
  $data = array('email' => $email);

  $resultSet = $this->gestionBD->queryPrepared($req,$types, $result_types, $data);

  $user = null;
  if ($resultSet->numRows()==1) {
   $row = $resultSet->fetchRow(MDB2_FETCHMODE_ASSOC);
   $user =  new User($row['email'],$row['password']);
  }

  return $user;
 }

 function getAllUsers()
 {
  $req= 'select email, password from user';
  $types=array();
  $result_types = array('text', 'text');
  $data = array();

  $resultSet = $this->gestionBD->queryPrepared($req,$types, $result_types, $data);

  $users = array();
  $rows = $resultSet->fetchAll(MDB2_FETCHMODE_ASSOC);
  foreach ($rows as $row)
  {
   $user =  new User($row['email'],$row['password']);
   array_push($users, $user);
  }

  return $users;
 }

 function delUserByEmail($email)
 {
  $req= 'delete from User where email=:email';
  $types = array('text');
  $data = array('email' => $email);
  $res = $this ->gestionBD ->execPrepared($req, $types, $data);
  return $res;
 }

 function createUser($email, $password)
 {
  $req = 'insert into User (email,password) values (:email, :password)';
  $types = array('text','text');
  $data = array('email' => $email, 'password' => $password);
```

```
  $res = $this ->gestionBD ->execPrepared($req, $types, $data);
  return $res;
 }

 function updateUser($email, $password)
 {
  $req = 'update User set email=:email, password=:password where email=:email';
  $types = array('text','text');
  $data = array('email' => $email, 'password' => $password);
  $res = $this ->gestionBD ->execPrepared($req, $types, $data);
  return $res;
 }
}


?>
```

Figure 3.10. Classe GestionBDUser



6.2. Réalisation

Nous allons, dans un premier temps, tester la classe `GestionBDUser` puis nous créerons et testerons la classe `GestionBDArticle`.

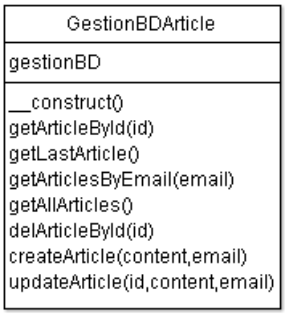
6.2.1. Exercice 17 Tests de la classe GestionBDUser

Ecrire un script (`www/tests/test_GestionBDUser.php`) permettant de tester la classe `GestionBDUser`.

6.2.2. Exercice 18 Création et tests de la classe GestionBDArticle

Ecrire la classe `GestionBDArticle` puis la tester.

Figure 3.11. Classe GestionBDArticle



3

7. Le MVC (Modèle Vue Contrôleur)

Nous avons travaillé sur le modèle, avec nos classes `Article`, `User`, `GestionBDArticle`, `GestionBDUser`, il est maintenant complet. Nous allons maintenant mettre en places nos contrôleurs, les contrôleurs sont responsables

[3]Dans un soucis de simplicité nous n'avons pas utlisé de champ auto-incrémentant pour l'attribut Id bien que PEAR MDB2 le permet.

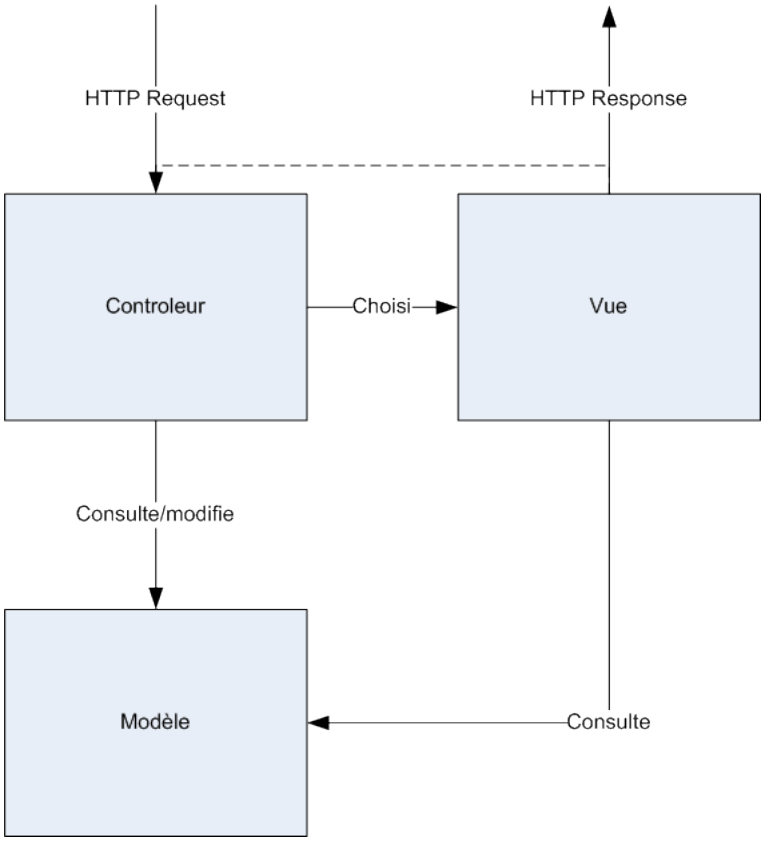
de la logique de contrôle. Le contrôleur analyse la requête du client, accède aux données et invoque la vue qui réalise la présentation.

7.1. Présentation

Nous allons bièvement présenter le patron de conception MVC et les bénéfices qu'il peut nous apporter pour nous détaillerons l'approche choisie.

7.1.1. Le patron de conception MVC

Figure 3.12. MVC en PHP



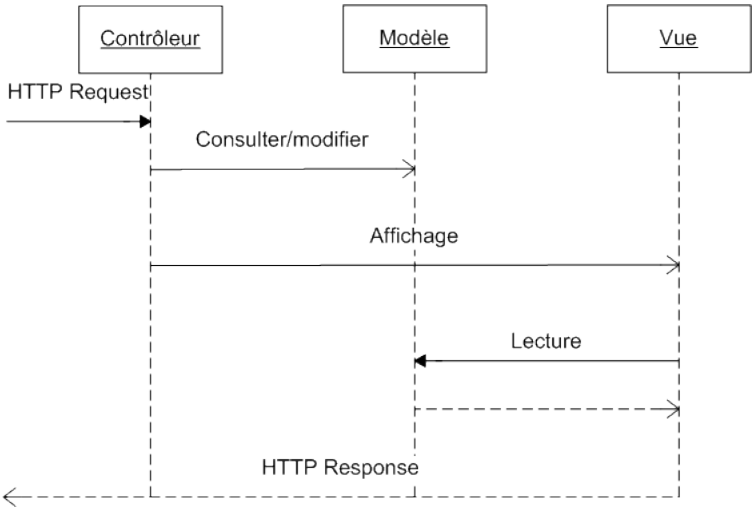
Séparer les concepts permets de séparer les métiers et faciliter la réutilisation. Dans ce patron de conception nous avons :

la vue
: qui est l'interface utilisateur, elle présente le modèle, elle reçoit les événements pour en avertir le contrôleur. Elle ne modifie pas le modèle et elle n'effectue aucun traitement.

le modèle
: qui défini le comportement de l'application et le traitement des données. Les données ne sont associées à aucune présentation. Le modèle assure la cohérence des données et offre des méthodesde manipulation de ces dernières. Dans une architecture non Web, le modèle peut changer ses changements à la vue.

le contrôleur
: qui gère les événements en modifiant le modèle et en invoquant les vues.

Figure 3.13. MVC diagramme de séquence



Dans les architectures Web PHP, nous retrouvons deux types de contrôleurs :

le contrôleur frontal
: il reçoit les requêtes et sollicite les contrôleurs de page, c'est le seul point d'entrée du système, tout passe par lui. C'est un singleton généralement nommé `index.php`. Le choix du contrôleur de page, de l'action et des paramètres est réalisé grâce à l'URL.

le contrôleur de page
: il gère sous commande du contrôleur frontal une partie du modèle

7.1.2. L'approche que nous avons choisie

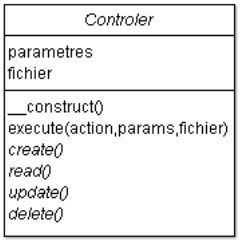
Nous avons choisi un front contrôleur qui traite des URL au format suivant : `index.php?module=unModule&action=uneAction&para1=unParametre&para2=unpParametre`. Ainsi l'URL `index.php?module=User&action=create&email=toto&password=passtoto`, va déclencher l'appel du contrôleur de page de `User` avec ordre de création et comme paramètres *`toto`* et *`password`*.

```
require_once("../config/inc.config.php");
if (isset($_GET['module']) && isset($_GET['action'])) {
 $moduleName = $_GET['module']; //ici $moduleName='User'
 $action = $_GET['action']; //ici action='create'
 $parametres = $_GET; //ici parametres['email']=toto et parametres['password']=passtoto
 $fichier=null;


}
if(isset($_POST['module']) && isset($_POST['action'])) {
 $moduleName = $_POST['module'];
 $action = $_POST['action'];
 // Cas des fichiers en upload
 $fichier = $_FILES;
 $parametres = $_POST;
}


if (isset($moduleName))
{
 $controleurName = "Controler".$moduleName; //Ici le controleur de page controleurName = 'ControlerUser'

 require_once(BO_CONTROLER_PATH.$controleurName."_classe.php");
 $controleur = new $controleurName(); //Ici $controleur = new ControleurUser();
 $controleur->execute($action,$parametres,$fichier); //demande de l'execution de l'action de création
}
```

Notre conttôleur principal nous permet donc de créer un contrôleur de page et de lancer la méthode exécute de ce dernier. Pour fixer le comportement de nos contrôleurs de page, nous avons défini une classe abstraite qu'ils devront implémenter.

```
<?php
 abstract class Controler {

  protected $parametres;
  protected $fichier;

  public function __construct() {
   $this->parametres = array();
   $this->fichier = array();
  }

  public function execute($action,$params,$fichier) {
   $this->parametres = $params;
   $this->fichier = $fichier;
   $this->$action();
  }


  abstract protected function create();
  abstract protected function read();
  abstract protected function update();
  abstract protected function delete();

 }

?>
```

Figure 3.14. Diagramme de classe de Controler



Comme nous pouvons le voir, tous nos contôleurs de page sont munis des méthodes create, read, update et delete. Ce qui nous donne par exemple pour `ControlerUser` :

```
class ControlerUser extends Controler
 {
  private $gestionBDUser;

  public function __construct()
  {
   parent::__construct();
   $this->gestionBDUser = new GestionBDUser();
  }

  public function create()
  {
   if (isset($this->parametres['email']) && isset($this->parametres['password']))
   {
   $email = $this->parametres['email'];
   $password = $this->parametres['password'];
   $this->gestionBDUser->createUser($email,$password);
   }
  }

  public function read()
```

```
 {
  if (isset($this->parametres['email']))
  {
   $email = $this->parametres['email'];
   $user=$this->gestionBDUser->getUserByEmail($email);
   echo $user;
  }
  else
  {
   $users=$this->gestionBDUser->getAllUsers();
   print_r($users);
  }
 }

 public function update()
 {
 if (isset($this->parametres['email']) && isset($this->parametres['password']))
  {
  $email = $this->parametres['email'];
  $password = $this->parametres['password'];
  $this->gestionBDUser->updateUser($email,$password);
  }
 }
 public function delete()
 {
 if (isset($this->parametres['email']))
  {
   $email = $this->parametres['email'];
   $this->gestionBDUser->delUserByEmail($email);
  }
 }
}L
```

Notre contrôleur de page `ControlerUser` lit et met à jour la base de donnée en utilisant la classe `GestionBDUser` qui fait partie de notre modèle.

7.2. Réalisation

Il vous faut en premier lieu importer le projet td-tp-5.

7.2.1. Exercice 19 Tests du contrôleur principal et tests du contrôleur utilisateur

En utilisant le contrôleur frontale, tester les quatres fonctionnalité du contrôleur de page utilisateurs.

7.2.2. Exercice 20 Création et tests de la classe ControlerArticle

En vous inspirant du contôleur de page des utilisateurs, créer et tester celui des articles

8. Les moteurs de template

Nous avons un modèle et un contrôleur, pour le moment notre contrôleur produit des affichages avec des `echo` et reçoit ses informations via des url saisies dans le navigateur. Nous allons réaliser la saisie et l'affichage dans des vues. Nous pourrions avoir des vues en PHP mais pour séparer les métiers nous allons utiliser un générateur de template. Nous produirons de l'XHTML à "trous", les trous étant ensuite remplis dynamiquement par du PHP. Cela permet designers XHTML de ne pas bricoler les code XHTML mais impose l'apprentissage d'un nouveau language celui du moteur de template bien plus spécialisé et plus simple que PHP.

8.1. Présentation

Le moteur de template que nous allons utiliser cette année est smarty : `http://smarty.net/`. Une documentation en français est disponible sur le même site : `http://smarty.net/manual/fr/index.php`.

8.1.1. Introduction

Commençons, par un exemple, nous avons deux fichiers écrits Hello.tpl le template, TestHello.php qui remplis le template et nous avons un fichier généré qui ne contient que du texte.

Exemple 3.14. Mon premier template

Le fichier PHP qui va remplir les trous TestHello.php :

```
require_once(SMARTY_DIR . 'Smarty.class.php');
$smarty = new Smarty();

$smarty->assign('trou','un truc'); //assignation des valeurs aux trous

$smarty->display('Hello.tpl'); //affichage
```

Le patron Hello.tpl :

```
Hello, {$trou} !
```

Le fichier texte résultant de la compilation :

```
Hello, un truc !
```

Pour fonctionner `Smarty` a besoin d'au moins deux répertoires :

templates
: le répertoire qui contient les templates

templates_c
: le répertoire qui contient les fichiers compilés

Pour spécifier ces répertoires nous avons défini une classe `MonSmarty` qui hérite de `Smarty` :

```
<?php
//  require_once("../../inc.config.php");
    require_once(SMARTY_PATH."Smarty.class.php");

    class MonSmarty extends Smarty {

        public function monSmarty () {
            $this->Smarty();
            $this->template_dir = TEMPLATES_PATH.'templates';
            $this->compile_dir = TEMPLATES_PATH.'templates_c';
            $this->config_dir = TEMPLATES_PATH.'configs';
            $this->cache_dir = TEMPLATES_PATH.'cache';

            $this->caching = 0;
            $this->debugging = false;
            $this->assign('app_name','TD-TP');
        }
    }
?>
```

8.1.2. Variables

Il existe trois types de variables celles assignées depuis PHP (assign(nom, valeur)), celles chargées depuis un fichier de configuration et enfin celle réservée.

Comme nous l'avons vu en php nous n'avons besoin que de assign et display. Par contre dans la template selon la nature de la variable (simple, tableau, objet), la syntaxe d'utilisation est différente, voici un exemple de issu de la documentation de Smarty. :

```
{$foo}        <-- affiche une variable simple (qui n'estpas un tableau ou un objet)
{$foo[4]}     <-- affiche le 5ème élément d'un tableau indexé
{$foo.bar}    <-- affiche la clé "bar" d'un tableau, identique à $foo['bar'] en PHP
{$foo.$bar}   <-- affiche la valeur de la clé d'un tableau, identique à $foo[$bar] en PHP
{$foo->bar}   <-- affiche la propriété "bar" de l'objet
```

```
{$foo->bar()} <-- affiche la valeur retournée de la méthode "bar" de l'objet
{#foo#}       <-- affiche la variable du fichier de configuration "foo"
{$smarty.config.foo} <-- synonyme pour {#foo#}
{$foo[bar]}   <-- synthaxe uniquement valide dans une section de boucle, voir {section}
{assign var=foo value='baa'}{$foo} <--  affiche "baa", voir {assign}

Plusieurs autres combinaisons sont autorisées

{$foo.bar.baz}
{$foo.$bar.$baz}
{$foo[4].baz}
{$foo[4].$baz}
{$foo.bar.baz[4]}
{$foo->bar($baz,2,$bar)} <-- passage de paramètres
{"foo"}       <-- les valeurs statiques sont autorisées

{* affiche la variable serveur "SERVER_NAME" ($_SERVER['SERVER_NAME'])*}
{$smarty.server.SERVER_NAME}
```

8.1.3. Modificateurs de variables

Les modificateurs de variables permettent de modifier la valeur affichée d'une variable. Les modificateurs de variable sont utilisés avec | et peuvent-être cascadés.

Exemple 3.15. Modificateur de variable

```
{$titre|upper}
{$titreArticle|upper|spacify}
```

Sur le premier exemple le titre est mis en majuscule, sur le deuxième exemple, il est mis en majuscule puis complété avec des espaces.

Voici la liste des modificateurs de variable : capitalize cat count_characters count_paragraphs count_sentences count_words date_format default escape indent lower nl2br regex_replace replace spacify string_format strip strip_tags truncate upper wordwrap.

8.1.4. Fonctions natives

Les fonctions natives sont fournies avec sont moteur et ne peuvent-être redéfinies, nous avons : {capture} {config_load} {foreach},{foreachelse} {if},{elseif},{else} {include} {include_php} {insert} {ldelim},{rdelim} {literal} {php} {section},{sectionelse} {strip}.

{foreach},{foreachelse}, {section},{sectionelse} permettent d'itérer sur des tableaux

{if},{elseif},{else} permettent d'exprimer la conditionnelle

{include}, {insert} permetttent d'inclure d'autres patron, ils sont pratiques par exemple pour inclure l'entête et le pied de page.

8.1.5. Fonctions utilisateurs

Smarty est livré avec un ensemble de fonctions utilisateurs, qu'il est possible d'étendre avec plugin : {assign} {counter} {cycle} {debug} {eval} {fetch} {html_checkboxes} {html_image} {html_options} {html_radios} {html_select_date} {html_select_time} {html_table} {mailto} {math} {popup} {popup_init} {textformat}.

A vous de lire la documentation, mais {debug} est interressant car il vous permet de lancer la console de débogage dans le navigateur. {html_table} vous permet aussi de répondre au problème comment afficher un tableau.

8.2. Réalisation

Récuperer le fichier td-tp-6.zip, la partie utilisateur est réalisée à vous de faire celle d'article. Le contrôleur des articles existe déjà, vous n'avez plus qu'a réaliser les templates, bien entendu, il vous faut regarder le contrôleur des articles pour connaître les assignations.

9. Synthèse

Le projet est quasiment fini, il ne nous reste plus qu'à rajouter l'authentification. Cette étape peut être réalisée en utlisant par exemple PEAR AUTH et en modifiant la table User pour qu'elle ait un niveau et le mot de passe chiffré. Les principales modifications portent sur le modèle.

Nous n'allons pas continuer a faire évoluer notre code, à ce stade nous avons la notion de vue (les templates), de modèle et de controleur, autant réutiliser le travail d'autrui et utiliser un framework.

Chapitre 4. Utilisation d'un framework