

# COURS

## PHP5 Avancé

## & MVC

---

DIFFUSION : Étudiants Master Langages et technologies du Web

---

### Table des matières

1	Pré requis.....	3
2	Objectif du cours .....	3
3	Le développement d'applications web, un développement complexe.....	4
4	Développement web .....	5
5	Modèle-Vue-Contrôleur (MVC) .....	6
6	MVC et PHP : implémentation .....	8
6.1	Architecture des dossiers.....	8
6.2	Contrôleur : .....	9
6.3	Modèle : .....	10
6.4	Vue : .....	10
7	Programmation orientée objet en PHP5.....	11
7.1	Rappel .....	11
7.2	Classe et instance.....	12
7.3	Copie, Héritage, redéfinition et subtilités. ....	14
7.3.1	Copie & référence.....	14
7.3.2	Héritage .....	15
7.3.3	Surcharge ou redéfinition.....	17
7.3.4	Subtilités .....	18
8	Base de données via PDO .....	23
8.1	Driver et installation .....	25
8.2	Connexion et gestion d'erreur de la connexion .....	26
8.2.1	Connexion .....	26
8.2.2	Erreurs de connexion.....	26
8.3	Requêtes et traitements.....	27

8.3.1	Requêtes.....	27
8.3.2	Traitements.....	28
9	Template et Vue .....	30
9.1	Principe du Template .....	30
9.2	Exemple « fait maison » :.....	31
9.2.1	Fichier de présentation .....	31
9.2.2	Script de données .....	31
9.2.3	Moteur du Template .....	31
9.2.4	Sortie HTML .....	32
9.3	Smarty : Gestionnaire de Template.....	33
10	Gestion des Erreurs .....	34
10.1	Gestion d'erreurs – Kit de survie.....	34
10.1.1	php.ini .....	35
10.1.2	Dans vos script .....	36
10.2	Les assertions .....	37
10.3	Les Exceptionsk .....	38
10.3.1	Principe .....	38
10.3.2	Usage .....	39
11	Développement et sécurité.....	40
11.1	Les points clés.....	40
11.2	Tout le monde ment ! .....	40
11.2.1	Ne fait pas confiance au utilisateur.....	40
11.2.2	Injection SQL.....	41
11.2.3	Session – Cookies et sécurité. ....	43
11.2.4	Vérifier vos logs .....	44

## Historique des révisions

Version	Auteur	Commentaires	Date
1	Frédéric Théron	Création du document	26/01/2012

Date d'édition :	20/02/2012		Page :	2 / 44

## 1 Pré requis

- Cours PHP de Jacques Bandet

[http://webu2.upmf-grenoble.fr/shs/master/wp-content/uploads/2011/11/04\\_php.pdf](http://webu2.upmf-grenoble.fr/shs/master/wp-content/uploads/2011/11/04_php.pdf)

-Cours HTML et XHTML de Jacques Bandet

[http://webu2.upmf-grenoble.fr/shs/master/wp-content/uploads/2011/10/01\\_xhtml.pdf](http://webu2.upmf-grenoble.fr/shs/master/wp-content/uploads/2011/10/01_xhtml.pdf)

## 2 Objectif du cours

L'objectif du cours est de vous donner une méthode de développement web (MVC), et de présenter des fonctions avancées en PHP5 pour une utilisation 'professionnelle'.

Date d'édition :	20/02/2012		Page :	3 / 44

### 3 Le développement d'applications web, un développement complexe

Le développement d'**applications** web est complexe, on **peut** travailler pour des cibles multiples :

- ⇒ Cibles machines (serveur, station client, Smartphone, Tablet ...)
- ⇒ Cibles browsers (IE 6et+, Chrome, Firefox, Safari, Opéra...)

Le développement d'**applications** web est complexe, on **doit** travailler avec de multiples langages :

- ⇒ Langages coté serveur web [ASP,PHP,Python,Perl ...]
- ⇒ Langages coté client [Javascript,Vbscript]
- ⇒ Langages de balisage [HTML, CSS]

Le développement d'**applications** web est complexe, on **doit** travailler pour de multiples publics:

- ⇒ Utilisateur débutant
- ⇒ Utilisateur confirmé
- ⇒ Gestionnaire des données
- ⇒ Administrateur de l'application

Le développement d'**applications** web est complexe, car on est soumis à des **contraintes** :

- ⇒ Contraintes de sécurités
- ⇒ Contraintes d'intégrité des données.
- ⇒ Contraintes légales

## 4 Développement web

Quand on débute en développement web, on mélange souvent les langages de script, de balisage, de CSS dans un seul fichier :

```
<html>
<style>
.Style1 {font-family: Verdana, Arial, Helvetica, sans-serif}
</style>
<div align="center" class="Style1">
<div align="center"><br>
    <br>
    <strong>BLABLABLA...</strong>
</div><br>
</center>
<table cellspacing="0"><!-- ligne de titre -->
    <tr> <td colspan="2" bgcolor="#336666"><h3><font color="#FFFFFF">
R&eacute;sum&eacute; de la demande</font></h3></td></tr>
</table>
<?php //d&eacute;but
function executer_recherche($requete)
{
    $retour=mysql_query($requete,$this->bdlien);
    if ($retour==false)
    {
        $this->erreurtexte=mysql_error();
        $this->erreurnbr= mysql_errno();
        $retour=" Erreur base de donnees : $this->erreurtexte, numero : --- $this->erreurnbr <br>";
    }else{
        $retour=mysql_fetch_array($retour);    }return $retour;
    }
}
?>
<form method="post" name="form" action="./U_renvoyer_dt.php" >
<table cols=2 cellspacing="0">
    <tr><td>Pr&eacute;nom :</td><td><?php echo $SESSION["prenom"]?></td></tr><tr>
    <td> Pr&eacute;nom :</td> <td><?php echo $SESSION["nom"]?></td></tr> <tr>
    <td>E-mail :</td> <td><?php echo $SESSION["mail"]?>@prb.com</td>
    </tr>
</table>
<?php
$num=$SESSION["service"];
$query="Select * From $tab_service Where $sch_valeur_service=$num";
$retour=$mabase->executer_recherche($query);
echo $retour[$sch_libelle_service];
?>
</table>    <input type=submit value="oui"> </form> </center>
</html>
```

On a dans ce bout de code, les erreurs majeures de programmation et de logique :

- ⇒ HTML, CSS, PHP dans un même document.
- ⇒ Pas de commentaires
- ⇒ Pas de bloc logique – d'indentation
- ⇒ Copier-coller incohérent
- ⇒ Formulaire pas compréhensible (Oui à quoi ?)

Q : Comment maintenir ce code ?

Date d'édition :	20/02/2012		Page :	5 / 44

## 5 Modèle-Vue-Contrôleur (MVC)

En développement logiciel, il existe des architectures de développement (**architectural pattern**) dont le but est d'isoler le domaine logique de la partie IHM.

### **Δ** Une architecture de développement ≠ Modèle conceptuel

Architecture de développement => Organisation logicielle de l'application. [Modèle programmeur]

Modèle conceptuel => Organisation des fonctionnalités de l'application. [Modèle utilisateur]

On trouve MVC dans des travaux datant de 1979 par un groupe de travail Xerox Parc.

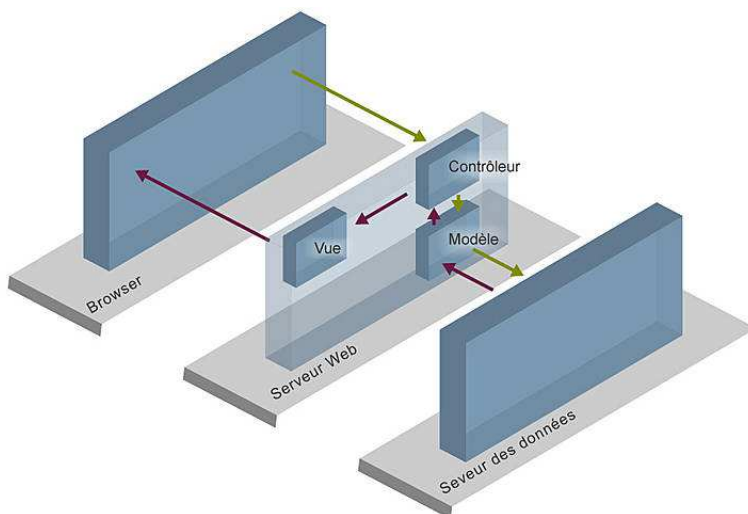
Les ingénieurs ont théorisé MVC pour le langage Smalltalk (langage objet, de haut **niveau d'abstraction**).

<http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>

Le but de MVC est la séparation de l'application en trois couches :

- ⇒ Le **Modèle** décrit les actions, les traitements métiers de l'application (connexion aux données, algo de résolution, vérification, sécurité...).
- ⇒ La **Vue** est la partie restitution des données aux utilisateurs de l'application (HTML, PDF, XML...)
- ⇒ Le **Contrôleur** est une interface entre la vue et le modèle. [Le mot interface ici ne doit pas être confondu avec l'interface utilisateur ou interface de classe]

Représentation MVC pour une application WEB.

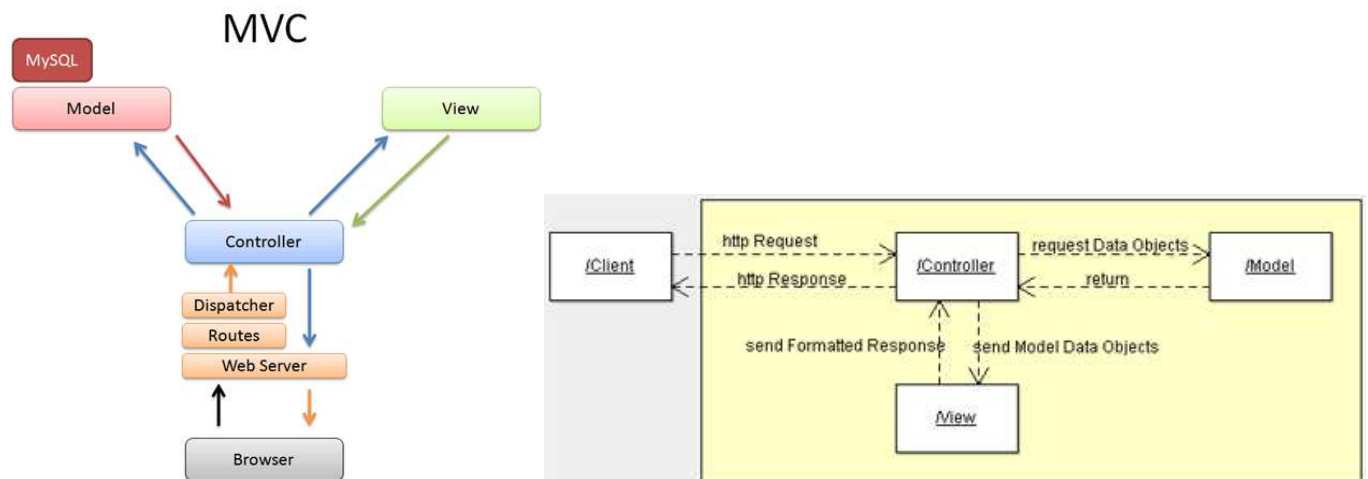


Date d'édition :	20/02/2012	Page :	6 / 44

## MVC et MVC2 :

Dans une architecture MVC, il peut exister plusieurs vues, plusieurs modèles mais aussi plusieurs contrôleurs.

Dans une architecture MVC2, il n'existe plus qu'un seul et unique contrôleur réceptionnant toutes les requêtes clientes.



Avantages et inconvénients de ce type de modèle.

Avantage :

- ⇒ Modularités.
- ⇒ Maintenance.
- ⇒ Gestion multi-développeurs.
- ⇒ L'isolation

Inconvénients :

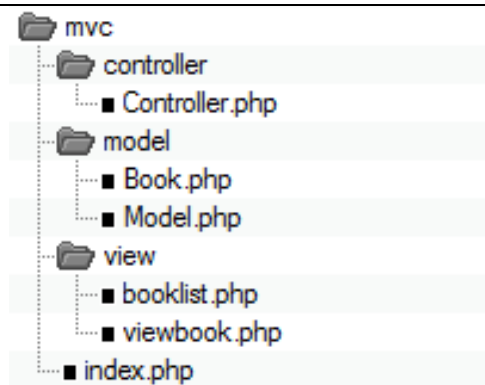
- ⇒ Complexité de communication entre composants.
- ⇒ Apprentissage notamment dans le cadre d'un Framework.

## 6 MVC et PHP : implémentation

L'implémentation dans une application web est simple dès lors que l'on a bien préparé l'analyse de l'application.

### 6.1 Architecture des dossiers

La mise en place d'une structure de dossier permet d'appréhender plus facilement l'implémentation du MVC.



On voit le principe d'une page de distribution/dispatch (index.php)

Pour utiliser proprement le modèle MVC, on peut utiliser un dispatcher, une page où toutes les requêtes arrivent et sont aiguillées.

Exemple :

<http://www.monsite.com/index.php?do=connexion> => Paramètre d'aiguillage dans l'URL (GET)

```
//DISPATCHER
if (!isset($_GET['do'])) {require ("default.php");}
else {
    switch ($_GET['do']){
        case "connexion":
            require ("action/connexion.php");break;
        case "crea_ad":
            require ("action/creation_adherent.php");break;
        case "inser_ad":
            require ("action/insert_ad.php");break;
    }
}
```

Sur une application importante, on risque d'avoir un énorme Switch ce qui n'est ni élégant, ni pertinent.

On peut travailler avec un dispatcher générique qui appelle directement le contrôleur.

On peut aussi avoir un appel à un unique contrôleur avec l'action en paramètre.



## 6.2 Contrôleur :

Le contrôleur reçoit les requêtes de l'utilisateur et fait l'intermédiaire entre le modèle et la vue pour répondre aux requêtes.

Scénario :

On a une demande de validation de connexion à une application avec Login et Password.

On va faire une demande au contrôleur pour gérer cette connexion.

Le contrôleur va appeler le modèle de gestion des utilisateurs et retourner une information indiquant si la personne peut ou ne pas se connecter. Il appelle la vue avec le retour indiqué par le modèle pour la présentation à l'utilisateur.

<p>Identifiant : <input type="text" value="toto"/></p> <p>Mot de passe : <input type="password" value="...."/></p> <p><input type="button" value="Envoyer"/></p>	<p>Page web</p> <p>Lors de la validation :</p> <p><a href="http://www.monsite.com/index.php?do=connexion">http://www.monsite.com/index.php?do=connexion</a></p>
--	---

<pre>require dirname(__FILE__)."\\controller\\controller.php"; if (!isset(\$_GET['do'])) {require ("index.php");} else { //on doit vérifier le do données. switch (\$_GET['do']){ case "connexion": connexion(); break;</pre>	<p>Page index.php (dispatcher)</p> <p>On inclut le contrôleur et selon la demande, on appelle la fonction du contrôleur.</p> <p><b>A Savoir :</b> Même si le contrôleur peut être de type fonction dans la majorité des cas, ce sera une classe.</p>
---	--

<pre>//CONTROLLER require dirname(__FILE__)."/../model/bd.php"; require dirname(__FILE__)."/../model/connexion.php"; require dirname(__FILE__)."/../vue/vue_acces.php";  function connexion(){  // Interface avec le modèle \$connect=new model_connexion(\$_POST['login'],\$_POST['pass']); \$connect-&gt;verif(); // Interface avec la vue display('acces',\$connect-&gt;autorisation); }</pre>	<p>Page controller.php</p> <p>On voit bien l'interface avec le modèle et avec la vue.</p> <p>Il y a bien <b>isolation</b> entre le modèle et la vue.</p>
---	--

Date d'édition :	20/02/2012	Page :	9 / 44
------------------	------------	--------	--------

--

### 6.3 Modèle :

On parle de Modèle pour la partie métier de l'application.

On a les scripts d'actions de l'application : creation\_data.php, verif\_utilisateur.php ...

On doit avoir dans le modèle, les classes, les fonctions, les fichiers de paramètres ...



On n'a pas et on ne doit pas avoir d'élément visuel dans le modèle sauf lors du développement, pour le debug.

### 6.4 Vue :

La vue est un ou plusieurs composants gérant la présentation. Ce peut être directement des fichiers HTML, mais aussi des fichiers textes, ou d'autres formats XML, PDF...

Le principe MVC est de dissocier les données et la présentation. La vue doit représenter un visuel (graphisme de l'application Web) et les données traitées par le modèle.

On a deux possibilités pour résoudre ce problème :

⇒ Créer des pages HTML avec du PHP.

```
<tr>
  <td>Prénom :</td>
  <td><?php echo $_SESSION["prenom"]?></td>
</tr>
```

Si on code avec ce système, on ne doit jamais utiliser de fonctions PHP pour créer la donnée. Les seules fonctions acceptables sont : **echo(); print();**

⇒ Utiliser un système de Template.

On verra le système de Template, plus longuement dans le chapitre 9

Quand on développe pour le web, on utilise souvent du JavaScript ou de l'AJAX. La question est de savoir ou mettre proprement les appels à AJAX ?

## 7 Programmation orientée objet en PHP5

### 7.1 Rappel

La programmation par objets ou par procédures classiques sont deux paradigmes du développement. La programmation orientée objet comporte les avantages suivants :

- ⇒ un code réutilisable.
- ⇒ un code modulaire.
- ⇒ un code compréhensible.
- ⇒ Un code conceptuel.

#### Qu'est-ce qu'une classe ?

Une classe est un modèle de définitions pour la création d'objets.

Une classe définit les attributs de l'objet (des champs) et des méthodes (des fonctions).

Par analogie : le plan d'une maison n'est pas une maison mais c'est le moyen de créer une ou plusieurs maison(s).

#### Qu'est-ce qu'un objet ?

Concrètement, un objet est une instance d'une classe.

Par analogie : La maison est une instance faite à partir du plan de la maison.



# CLASSE ≠ OBJET

## 7.2 Classe et instance.

Déclaration d'une classe :

```
Class nom_de_la_classe {  
    // attribut  
    Public $nom;  
    Protected $nom1;  
    Private $nom2;  
    //methode  
    function __construct(){}  
    function __destruct(){}  
}
```

Instanciation d'un objet :

```
$mon_objet=new nom_de_la_classe();
```

La sureté de l'application :

- public : L'accessibilité à la méthode ou à l'attribut est possible de toute l'application.  
**Par défaut** si aucun type d'accès n'est indiqué c'est un accès public.
- protected : L'accessibilité à la méthode ou à l'attribut est possible à la **classe** qui l'a défini mais aussi à ses classes héritées.
- private : L'accessibilité à la méthode ou à l'attribut est possible uniquement à la classe.

Constructeur et Destructeur :

PHP propose un moyen d'initialisation d'un objet au moment de sa création grâce à : `__construct (){}`   
Lors de l'instanciation par le mot clé **NEW** si dans la class la méthode `__construct` existe, elle va être **appelée**.

PHP propose un moyen de supprimer un objet grâce à `__destruct(){}` . Cette méthode est automatiquement appelée quand l'objet est détruit, soit par **delete()** ou **unset()**, soit à la fin du script.  
Un destructeur est pratique pour fermer les ressources ouvertes : fichiers, connexions vers des serveurs de bases de données, etc.

### Exemple

Définition de la classe :

```
class Voiture
{
    /* Déclaration des attributs */
    protected $niveau_carburant;
    public $nombre_portes;
    private $nombre_roues;
    /* Méthode */
    public function __construct(int $nb_carburant, int $nb_portes, int $nb_roues = 4)
    {
        $this->niveau_carburant = $nb_carburant;
        $this->nombre_portes = $nb_portes;
        $this->nombre_roues = $nb_roues;
    }
    public function __destruct()
    {
        echo 'L\'objet a été détruit';
    }
}
```

Usage de la classe :

```
<?php
$voiture = new Voiture(50, 3);
echo $voiture->nombre_portes;
//va afficher "3" car l'attribut est en accès public
echo $voiture->nombre_roues;
//Erreur, on ne peut pas y accéder car l'attribut est en accès privé !
```

### \$this: (comme en Java)

Il est souvent utile de pouvoir faire référence à l'objet en cours dans une méthode. C'est par exemple le cas pour accéder à un attribut ou lancer une autre méthode. La méta variable \$this est une référence permanente vers l'objet courant.

### Accès au méthode et attribut (≠ JAVA):

En java pour accéder à une méthode ou un attribut, on utilise **nom\_objet . nom\_attribut**  
En PHP, pour accéder à une méthode ou un attribut, on utilise **\$nom\_objet -> nom\_attribut**

**Δ** Il n'y a pas \$ devant nom\_attribut

## 7.3 Copie, Héritage, redéfinition et subtilités.

### 7.3.1 Copie & référence

Il faut définir la différence entre deux objets qui sont « identiques » et deux objets qui sont « les mêmes ».

Ex : 2 DVD du même film dans la même collection sont identiques mais ne sont pas les mêmes.

La classe

```
class compte {  
    /* Déclaration des attributs */  
    public $solde=0;  
    /* Méthode */  
    function virer($montant,$compte){  
        $this->solde-=$montant;  
        $compte->solde+=$montant;  
    }  
}
```

Sortie web :

50

150

---

0

150

200

Le script

```
$fred=new compte();  
$eric=new compte();  
$fred->solde=100;  
$eric->solde=100;  
$frederic=$eric; // Que fait-on ?  
$fred->virer(50,$frederic);  
echo $fred->solde;  
echo '<br>';  
echo $frederic->solde;  
echo "<hr>";  
$frederic= clone $eric; // Que fait-on ?  
$fred->virer(50,$frederic);  
echo $fred->solde;  
echo '<br>';  
echo $eric->solde;  
echo "<br>";  
echo $frederic->solde;
```

Par défaut en PHP5, l'assignation est une référence.

`$obj1 = $obj2` // on ne copie pas l'objet on crée une référence (le & est implicite)  
(appel contraire Jacques Bandet. PHP4-PHP5)

Si on veut réellement créer une copie de l'objet, on utilise la fonction **clone()** qui copie l'objet.

### 7.3.2 Héritage

L'héritage est la création de classe à partir d'une autre classe.

Ex : mon objet `ma_ferrari` provient de la class `voiture` qui est héritée de la class `véhicule_a_moteur`.

## PHP5 ne supporte pas l'héritage multiple (≠JAVA)

Prototype de l'héritage en PHP5:

```
Class nom_new_classe extends nom_classe_herité {  
}
```

L'héritage dans PHP5 est un héritage **strict**.

	Père	Fils
Paramètre obligatoire	w	x < w à condition de créer des valeurs par défaut.
Paramètre optionnel	y	z > y

Les méthodes de la classe fils doivent avoir des prototypes compatibles avec ceux de la classe père.

Il est possible d'ajouter des **paramètres supplémentaires**, à condition qu'ils soient **facultatifs**. Il est aussi possible de rendre facultatifs des paramètres en leur donnant une valeur par défaut.

Pour résumer, **le nombre de paramètres obligatoires de la méthode fille doit être inférieur ou égal au nombre de paramètres possibles de la méthode mère** ; le nombre de paramètres possibles de la méthode fille doit quant à lui être supérieur ou égal au nombre de paramètres possibles de la méthode mère.

Par exemple, une méthode qui a trois paramètres obligatoires sur cinq peut être remplacée par une méthode qui a un paramètre obligatoire (nombre inférieur) et cinq facultatifs (donc six au total, ce qui est supérieur aux cinq initiaux).

Seuls les constructeurs ne sont pas soumis à cette règle car la notion de constructeur générique n'a pas vraiment de sens.

<pre> class bourse extends compte {     /* Déclaration des attributs */     public \$nbr_actions=0;     public \$prix_actions;     /* Méthode */     function achat(\$nbr,\$valeur){         \$this-&gt;nbr_action+=\$nbr;         \$this-&gt;prix_action=\$valeur;         \$this-&gt;solde-=\$nbr*\$valeur;     }     function vendre(\$nbr,\$valeur){         \$this-&gt;nbr_action-=\$nbr;         \$this-&gt;prix_action=\$valeur;         \$this-&gt;solde+=\$nbr*\$valeur;     } } </pre>	<p>On créait une classe <b>héritée</b> de la <b>classe</b> compte <b>vue</b> précédemment.</p> <p>On créait <b>une</b> classe bourse qui étend les attributs et les méthodes de notre classe compte.</p>
--	--

On va utiliser cette classe :

```

$frederic =new bourse;
$frederic->solde = 1000;
echo "solde : ".$frederic->solde."<br>";
$frederic->achat(5,100);
echo "ACHAT : nbr action en portefeuille: ".$frederic->nbr_action.", prix de l'action : ".$frederic->prix_action."<br>";
echo "ACHAT SOLDE : ".$frederic->solde."<br>";
$frederic->vendre(4,150);
echo "VENTE : nbr action en portefeuille: ".$frederic->nbr_action.", prix de l'action : ".$frederic->prix_action."<br>";
echo "VENTE SOLDE : ".$frederic->solde."<br>";

```

La sortie web :

```

solde : 1000
ACHAT : nbr action en portefeuille: 5, prix de l'action :100
ACHAT SOLDE : 500
VENTE : nbr action en portefeuille : 1, prix de l'action :150
VENTE SOLDE : 1100

```

Date d'édition :	20/02/2012	Page :	16 / 44



### 7.3.3 Surcharge ou redéfinition

Dans la définition de l'héritage strict, on parle implicitement de la surcharge de méthode ou d'attribut.

On peut dans la classe fille, redéfinir une méthode ou un attribut. Dans ce cas, par défaut, c'est la dernière définition qui est utilisée.

Cependant, on peut vouloir accéder aux méthodes parentes, on utilise alors la notation statique:

Parent::methode()
-------------------

**A savoir :** L'accès statique est possible dans un script, attention si la méthode a besoin d'un objet (\$this) alors une erreur est déclenchée.

Class Parent

```
function virer($montant,$compte){  
    echo "function virer de la class compte<br>";  
    $this->solde-=$montant;  
    $compte->solde+=$montant;  
}
```

Class Fille

```
function virer($montant,$compte,$nbr_action){  
    echo "function virer de la class Bourse<br>";  
    parent::virer($montant,$compte);  
    $this->nbr_action+=$nbr_action;  
}
```

Script :

```
$fred=new bourse();  
$eric=new bourse();  
$fred->solde=100;  
$eric->solde=100;  
$frederic=$eric;  
$fred->virer(50,$frederic,10);  
echo "SOLDE :".$fred->solde."<br>";  
echo "NBR ACTION :".$fred->nbr_action."<br>";
```

Sortie Web :

```
function virer de la class Bourse  
function virer de la class compte  
SOLDE :50  
NBR ACTION :10
```

### 7.3.4 Subtilités

⇒ Héritage et redéfinition des contrôles d'accès.

Si vous redéfinissez une méthode ou un attribut, vous pouvez changer alors sa sécurité d'accès. Le principe est simple, vous ne pouvez restreindre un contrôle d'accès.

Tableau des comportements d'héritages :

	Classe père	Classe fils
Attributs ou méthodes	Public	Public
	Protected	Protected , public
	Private	<u>Private</u> , Protected , public

**Attention :**

On ne peut pas redéfinir une méthode privée. Si on le faisait, on créerait en réalité une nouvelle méthode. Si vous redéfinissez une méthode privée, PHP5 considérera qu'il a deux méthodes de même nom simultanément dans la classe. Si c'est une méthode de la classe mère qui y fait appel, elle accèdera à la méthode privée initiale. Si inversement c'est une méthode de la classe fille qui y fait appel, elle accèdera à la nouvelle implémentation.



**Il est déconseillé de redéfinir une méthode privée**

⇒ Classe abstraites et interfaces

**Une classe abstraite** c'est une implémentation minimale d'une classe.

Elle définit les méthodes qui devront être obligatoire pour la classe qui hérite de la classe abstraite.

**Les interfaces** (Attention : on parle ici d'interface au sens développement) permettent de créer du code qui spécifie quelles méthodes une classe doit implémenter. Toutes les méthodes déclarées dans une interface doivent être publiques.

Intérêt :

Les interfaces peuvent être vues comme des contrôles de qualité (vérifier que les objets correspondent bien aux spécifications) et les classes abstraites comme des implémentations incomplètes, à finir.

Déclaration class abstraite	Déclaration interface
<pre>abstract class compte_bancaire{     /* Méthode */     abstract function virer(\$montant,\$compte); }</pre>	<pre>interface compte_banque{     public function virer(\$montant,\$compte); }</pre>
Implémentation de classe abstraite	Implémentation d'une interface
<pre>class compte extends compte_bancaire</pre>	<pre>class compte implements compte_banque</pre>

#### A savoir

- Une classe peut implémenter plusieurs interfaces en même temps.

```
class compte implements compte_banque, ressource
```

- Une classe ne peut pas avoir d'héritage multiple classe abstraite ou pas.
- On ne peut pas surcharger une interface ou une méthode abstraite (intérêt même de ces systèmes)

**⚠ Il est déconseillé d'utiliser classe abstraite et interface. On utilise soit l'une ou l'autre des méthodes.**

#### ⇒ Fonction finales

Le concept de méthodes finales est simple, on indique à PHP5 qu'aucune classe dérivée n'a le droit de modifier l'implémentation d'une méthode.

Ex : On gère une collection de média, on a une classe media duquel on dérive pour créer la classe DVD, BD, CD ...

<pre>final function perdu(\$ref_media){     // Code pour indiquer     //que le media n°ref_media est perdu }</pre>	Si dans la class média, on créait une méthode <b>perdu(\$ref_media)</b> ; on ne veut pas pouvoir surcharger cette méthode.
--	--

On peut aussi déclarer une classe comme final, mais alors on ne pourra plus hériter d'elle. L'intérêt est là plus limité.

⇒ Fonctions particulières

Fonction de clonage :

```
function __clone(){  
    echo 'on clone un compte';  
}
```

On a vu le principe du clonage, on peut créer manuellement une fonction de clonage qui viendra s'ajouter au principe natif de PHP5

Fonction d'appel à une méthode (utile pour le debug) :

Cette méthode intercepte tous les appels à des **méthodes qui n'existent pas**.

```
function __call($nom,$val){  
    print "on a un appel pour la fonction $nom avec la valeur $val[0] <br>";  
}
```

Fonction d'information

```
string get_class($obj)
```

La fonction get\_class retourne une chaîne donnant le nom de la classe de l'objet.

Classe d'information

Il existe une classe ReflectionClass("Nom\_classe") ayant des méthodes d'information sur la classe.

Ex : Avec la classe bourse

```
$fct = new ReflectionClass( 'bourse' );  
echo 'La classe se nomme : ' . $fct->getName() . "<br>\n";  
echo 'Elle est définie dans le fichier ' . $fct->getFileName() . "<br>\n";  
echo 'entre les lignes ' . $fct->getStartLine();  
echo ' et ' . $fct->getEndLine();  
echo "<br>\n";
```

La sortie web donne

La classe se nomme : bourse

Elle est définie dans le fichier C:\wamp\www\cours\_php\class\_test.php

entre les lignes 26 et 47

Date d'édition :	20/02/2012		Page :	20 / 44

On peut aussi avoir la vue de la classe

```
echo '<pre>';
Reflection::export($fct);
echo '/<pre>';
```

On aura en sortie web :

```
Class [ class bourse extends compte ] {
  @@ C:\wamp\www\cours_php\class_test.php 26-47

  - Constants [0] {
  }

  - Static properties [0] {
  }

  - Static methods [0] {
  }

  - Properties [3] {
    Property [ public $nbr_actions ]
    Property [ public $prix_actions ]
    Property [ public $solde ]
  }

  - Methods [6] {
    Method [ public method achat ] {
      @@ C:\wamp\www\cours_php\class_test.php 31 - 35

      - Parameters [2] {
        Parameter #0 [ $nbr ]
        Parameter #1 [ $valeur ]
      }
    }

    Method [ public method vendre ] {
      @@ C:\wamp\www\cours_php\class_test.php 36 - 40

      - Parameters [2] {
        Parameter #0 [ $nbr ]
        Parameter #1 [ $valeur ]
      }
    }
  }
}
```

Date d'édition :	20/02/2012		Page :	21 / 44

⇒ Chargement de classe

Comme dans tout langage objet, on doit charger les classes que l'on va utiliser.

```
include ('fichier_definition_class.php');  
include_once ('fichier_definition_class.php');  
  
require ('fichier_definition_class.php');  
require_once ('fichier_definition_class.php');
```

Les fichiers sont inclus suivant le chemin du fichier fourni.

Si aucun n'est fourni, l'`include_path` sera vérifié.

Si le fichier n'est pas trouvé dans l'`include_path` alors `include()` vérifiera dans le dossier du script appelant et dans le dossier de travail courant avant d'échouer.

Si l'instruction est `_once`, PHP vérifie si le fichier a déjà été inclus et si c'est le cas, ne l'inclut pas une deuxième fois.

L'instruction **`include()`** enverra une erreur de type **warning** si elle ne peut trouver le fichier; ce comportement est différent de **`require()`**, qui enverra une erreur de type **fatal**.

Pour ne pas à chaque fois utiliser l'adresse exacte du fichier à inclure, on peut utiliser la variable magique `__FILE__`.

```
require dirname(__FILE__)."\nom_fichier.php");
```

**Δ** On utilise `require()` quand on gère des classes

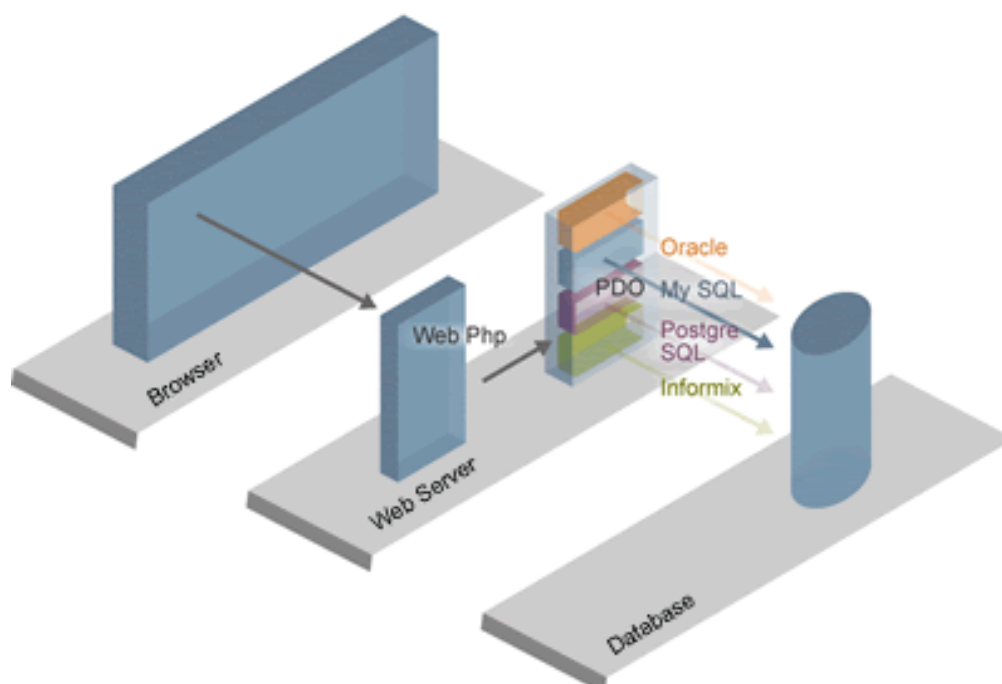
## 8 Base de données via PDO

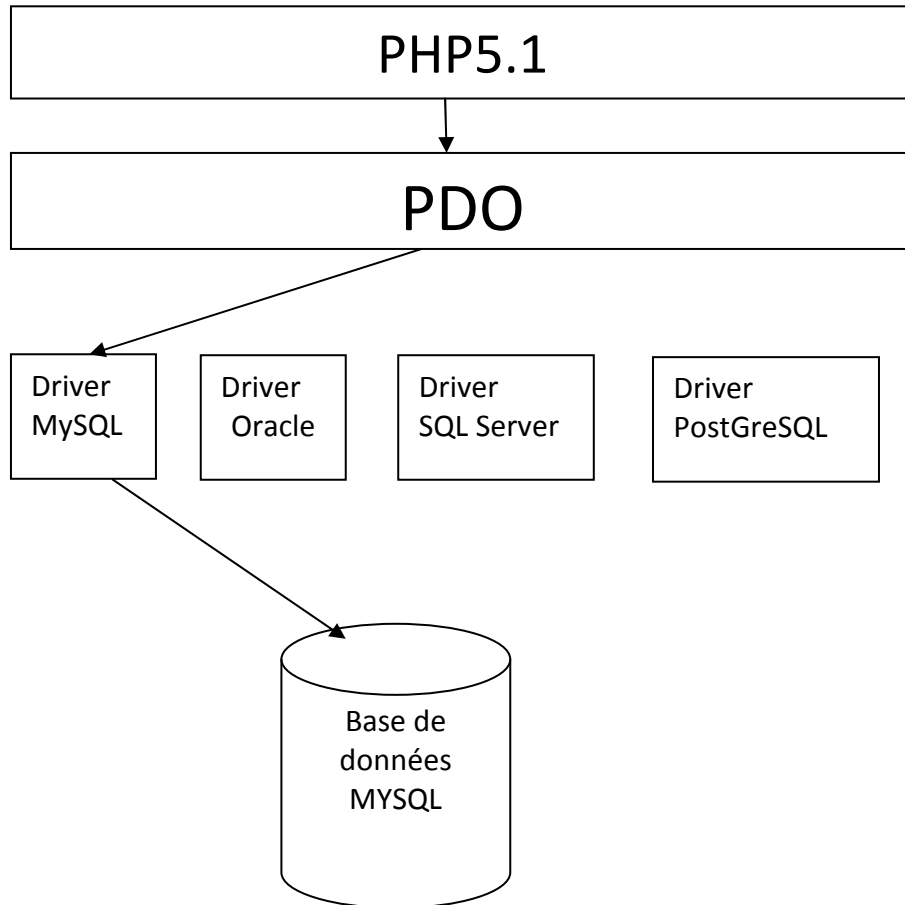
L'extension PHP Data Object (PDO) définit une interface pour accéder à une base de données depuis PHP.

PDO fournit une interface d'abstraction à l'accès de données, ce qui signifie que vous utilisez les mêmes fonctions pour exécuter des requêtes ou récupérer les données **quelle** que soit la base de données utilisée.

PDO ne fournit pas une abstraction de base de données : il ne réécrit pas le SQL, n'émule pas des fonctionnalités manquantes. Vous devriez utiliser une interface d'abstraction complète si vous avez besoin de cela.

PDO n'est disponible qu'avec PHP5 car PDO utilisent les notions OO de PHP5.







## 8.1 Driver et installation

Tableau des drivers et des bases de données (php.net) :

Nom du driver	Bases de données supportées
<a href="#">PDO_CUBRID</a>	Cubrid
<a href="#">PDO_DBLIB</a>	FreeTDS / Microsoft SQL Server / Sybase
<a href="#">PDO_FIREBIRD</a>	Firebird/Interbase 6
<a href="#">PDO_IBM</a>	IBM DB2
<a href="#">PDO_INFORMIX</a>	IBM Informix Dynamic Server
<a href="#">PDO_MYSQL</a>	MySQL 3.x/4.x/5.x
<a href="#">PDO_OCI</a>	<i>Oracle Call Interface</i>
<a href="#">PDO_ODBC</a>	ODBC v3 (IBM DB2, unixODBC et win32 ODBC)
<a href="#">PDO_PGSQL</a>	PostgreSQL
<a href="#">PDO_SQLITE</a>	SQLite 3 et SQLite 2
<a href="#">PDO_SQLSRV</a>	Microsoft SQL Server / SQL Azure
<a href="#">PDO_4D</a>	4D

Pour installer un driver :

On modifie le fichier php.ini (fichier de paramètre existant sous Windows et linux), ou on utilise la fonction dl() pour les charger au moment de l'exécution.

Sous windows:

```
;extension=php_pdo_mssql.dll
extension=php_pdo_mysql.dll
;extension=php_pdo_oci.dll
;extension=php_pdo_oci8.dll
extension=php_pdo_odbc.dll
;extension=php_pdo_pgsql.dll
extension=php_pdo_sqlite.dll
```

Sous linux les extensions sont des .so

Date d'édition :	20/02/2012	Page :	25 / 44

## 8.2 Connexion et gestion d'erreur de la connexion

### 8.2.1 Connexion

Pour se connecter à une base de données, il faut tout d'abord créer **une instance de la classe PDO**.

Le constructeur PDO à besoin de 3 paramètres :

- 1 - DSN (Data Source Name).
- 2- Le nom d'utilisateur.
- 3- Le mot de passe.

**Le DSN dépend du driver de la base de données.**

Le Data Source Name (DSN) de PDO\_MYSQL est composé des éléments suivants :

- **host** : adresse du serveur distant (nom ou adresse IP, « localhost » pour un serveur local) ;
- **dbname** : nom de la base de données à utiliser ;
- **port** : donnée facultative indiquant le port TCP/IP utilisé pour la connexion ;
- **unix\_socket** : donnée facultative indiquant l'adresse du socket unix pour la connexion locale.

```
$connexion = new PDO('mysql:host='.$PARAM_hote.';dbname='.$PARAM_nom_bd, $PARAM_utilisateur, $PARAM_mot_passe);
```

A Savoir :

Pour travailler proprement, on **crée** un script de paramétrage et de connexion.

```
//information de connexion pour la base de données
$PARAM_hote='localhost'; // le chemin vers le serveur
$PARAM_port='3306';
$PARAM_nom_bd='pwd'; // le nom de votre base de données
$PARAM_utilisateur='root'; // nom d'utilisateur pour se connecter
$PARAM_mot_passe=''; // mot de passe de l'utilisateur pour se connecter
```

### 8.2.2 Erreurs de connexion

Les erreurs de connexion peuvent provenir de plusieurs facteurs :

- ⇒ Le pilote n'a pas été chargé
- ⇒ La base de données n'a pas été trouvée
- ⇒ Le login et mot de passe ne sont pas corrects.
- ⇒ ...

Date d'édition :	20/02/2012		Page :	26 / 44

La classe PDO gère les exceptions et donc lèvera automatiquement une exception.

Comme en JAVA, la structure de gestion des exceptions est :

```
try {  
    //Code de connexion  
}  
catch(Exception $e){  
    //Code si une erreur a été levé  
}
```

Implémentation pour les erreurs de connexions :

```
try{  
    $connexion = new PDO('mysql:host='.$PARAM_hote.';dbname='.$PARAM_nom_bd, $PARAM_utilisateur, $PARAM_mot_passe);  
}  
catch(Exception $e){  
    echo 'Erreur : '.$e->getMessage().'\n';  
    echo 'N° : '.$e->getCode();  
    die();  
}
```

## 8.3 Requêtes et traitements

### 8.3.1 Requêtes

Pour envoyer des requêtes au serveur, on a deux méthodes de l'objet PDO :

- ⇒ Exec() → utiliser pour les requêtes INSERT, UPDATE, DELETE.  
Cette méthode renvoie le nombre de lignes modifiés.
- ⇒ Query() → utiliser pour les requêtes SELECT, SHOW, DESC, EXPLAIN.  
Cette méthode renvoie un objet PDOStatement comprenant les résultats de la requête.

Par Exemple :

```
$req_verif_identifiant=" SELECT count(*) as nombre
                           FROM comptes
                           WHERE cp_identifiant='$identifiant' and cpt_pass='$pass'";
$resultat=$connexion->query($req_verif_identifiant);
```

### 8.3.2 Traitements

La méthode query() renvoi un objet de type PDOStatement.

Il faut donc utiliser sur le résultat, des méthodes pour récupérer les données :

⇒ fetch() → Permet d'accéder aux résultats séquentiellement (on utilise while par exemple).

```
mixed PDOStatement::fetch ([ int $fetch_style [, int $cursor_orientation =
PDO::FETCH_ORI_NEXT [, int $cursor_offset = 0 ]]] )
```

⇒ fetchAll() → retourne l'ensemble des données dans un tableau.

```
array PDOStatement::fetchAll ([ int $fetch_style [, mixed $fetch_argument [, array $c
tor_args = array() ]]] )
```

Dans ces méthodes, il est important de qualifier la façon dont on veut le retour des données grâce à fetch\_style qui contrôle comment la prochaine ligne sera retournée à l'appelant.

Tableau de l'attribut fetch\_style (Il existe d'autres attributs cf doc) :

PDO::FETCH_ASSOC	retourne un tableau indexé par le nom de la colonne comme retourné dans le jeu de résultats
PDO::FETCH_BOTH	(défaut): retourne un tableau indexé par les noms de colonnes et aussi par les numéros de colonnes, commençant à l'index 0, comme retournés dans le jeu de résultats
PDO::FETCH_OBJ:	retourne un objet anonyme avec les noms de propriétés qui correspondent aux noms des colonnes retournés dans le jeu de résultats

```
$req="Select * from user";

$db=$connexion->query($req);
$res = $db->fetchAll(PDO::FETCH_OBJ);
print_r($res);

$db=$connexion->query($req);
$res = $db->fetchAll(PDO::FETCH_ASSOC);
print_r($res);
$db=$connexion->query($req);

$res = $db->fetchAll(PDO::FETCH_BOTH);
print_r($res);
```

```

Array
(
    [0] => stdClass Object
        (
            [user_id] => 1
            [user_nom] => Theron
            [user_prenom] => Frédéric
            [user_mail] => frederic.theron@gmail.com
        )
)
Array
(
    [0] => Array
        (
            [user_id] => 1
            [user_nom] => Theron
            [user_prenom] => Frédéric
            [user_mail] => frederic.theron@gmail.com
        )
)
Array
(
    [0] => Array
        (
            [user_id] => 1
            [0] => 1
            [user_nom] => Theron
            [1] => Theron
            [user_prenom] => Frédéric
            [2] => Frédéric
            [user_mail] => frederic.theron@gmail.com
            [3] => frederic.theron@gmail.com
        )
)

```

L'usage de `fetchAll` est souvent préféré à `fetch`, si dans la plupart des cas cela ne pose pas de problème cela peut avoir des incidences sur une requête ayant une lourde volumétrie car les tableaux en PHP ont des limites (notamment liées à la mémoire disponible).

100 000 d'enreg pour 10 ko=> 1000 000 Ko =>900Mo on ne peut pas utiliser un tableau

## 9 Template et Vue

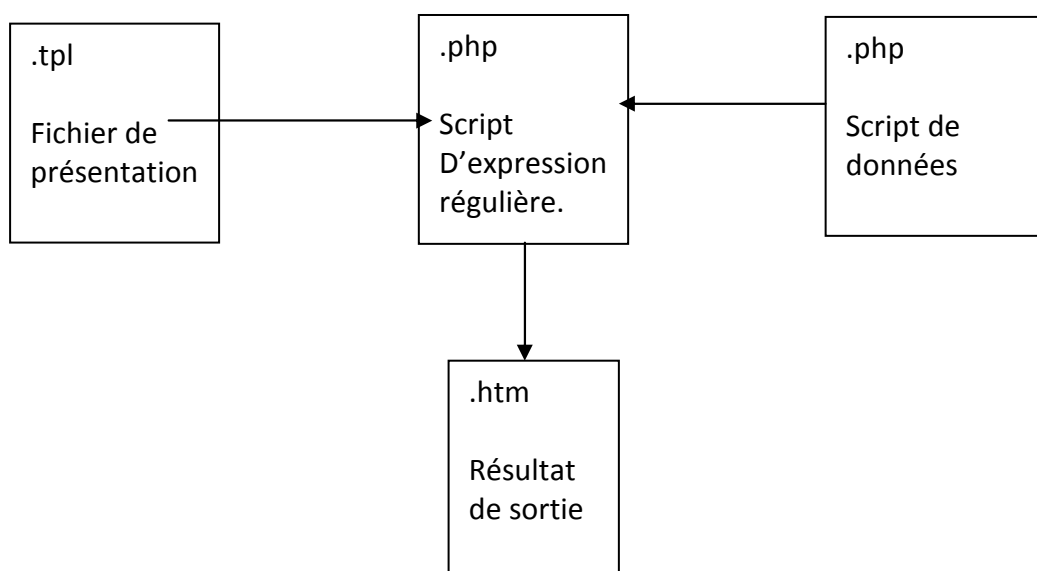
### 9.1 Principe du Template

Dans le point 6.3 du document, j'indique que pour gérer la vue de la partie MVC, il existe deux méthodologies dont l'une est le Template de présentation.

Le développeur pourrait s'affranchir de la partie représentation, il pourrait développer la partie Modèle, passer le résultat soit au contrôleur soit à la vue sans savoir comment les données vont être affichées (web, PDF, XML ...)

Le principe d'un système de Template est dans la séparation entre le contenu de l'information et la forme de sa représentation.

	Avantage	Inconvénient
Template de présentation	Clarté dans le développement  Facilité de développement à plusieurs (dev, graphiste)  Productivité de développement.	Apprentissage du système de Template.  Si pas de gestionnaire de cache, solution peu ralentir la création des pages web.



## 9.2 Exemple « fait maison » :

On veut créer une page web affichant le résultat d'une requête nous retournant qu'un enregistrement.

### 9.2.1 Fichier de présentation

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<div id="conteneur">
<h2>{{titre}}</h2>
<label for='nom'>Nom :</label>{#nom}
<br/>
<label for='prenom'>Prénom :</label>{#prenom}
<br/>
<label for='nom'>Nom :</label>{#mail}
</div>
</div>
</body>
```

Fichier Template :

Ce fichier comporte la page HTML avec des balises que l'on devra remplacer par les données provenant du résultat d'un script PHP.

On voit bien ici, que l'on peut donner cette page à créer à un webdesigner. Il n'y a pas de programmation.

### 9.2.2 Script de données

On utilisera la requête du script de la page 28.

```
$req= "select * from user limit 0,1";
```

### 9.2.3 Moteur du Template

Le but est d'analyser le fichier de Template et remplacer les balises par les données en provenance du script de données.

On peut utiliser plusieurs méthodes pour arriver à ce but :

- ⇒ Créer des tableaux : array(valeur\_chercher) , array(valeur\_replacement) et matcher
- ⇒ On peut utiliser des gestionnaires de chaînes.
- ⇒ On peut utiliser le remplacement via des fonctions d'expressions régulières
- ⇒

Pour l'exemple, on va utiliser les fonctions d'expressions régulières.

```

require ("test_pdo.php");      <- on appel le script de données
$sortie=file("template.tpl"); <-on lit le fichier template et on le met dans un tableau

$htm="";
//On récupère nos données
$nom =htmlentities($res[0]['user_nom']);
$prenom =htmlentities($res[0]['user_prenom']);
$mail =$res[0]['user_mail'];
$titre=htmlentities("à ne pas reproduire avec des millions d'utilisateurs");

//On travail par expression régulière
foreach ($sortie as $valeur){

    $valeur=preg_replace("/{#titre}/",$titre,$valeur);
    $valeur=preg_replace("/{#nom}/",$nom,$valeur);
    $valeur=preg_replace("/{#prenom}/",$prenom,$valeur);
    $valeur=preg_replace("/{#mail}/",$mail,$valeur);
    $htm.=$valeur;
}
print $htm;

```

**⚠** On n'utilise pas ce système avec un site à forte volumétrie.

#### 9.2.4 Sortie HTML.

**{#titre}**

Nom :{#nom}

Prénom :{#prenom}

Nom :{#mail}

Au final :

**à ne pas reproduire sur un site avec des millions d'utilisateurs**

Nom :Theron

Prénom :Frédéric

Nom frederic.theron@gmail.com

Date d'édition :	20/02/2012	Page :	32 / 44



### 9.3 Smarty : Gestionnaire de Template

Smarty est un gestionnaire de Template professionnel.

<http://www.smarty.net/>

Quelques caractéristiques de Smarty :

- ⇒ Il est efficace, le parser PHP s'occupe du sale travail.
- ⇒ Pas d'analyse de Template coûteuse, une seule compilation.
- ⇒ Il sait recompiler uniquement les fichiers de Template qui ont été modifiés.
- ⇒ Syntaxe des Templates configurable, vous pouvez utiliser {}, {{}}, <!--{}-->, etc. comme délimiteurs tag.
- ⇒ Les instructions if/elseif/else/endif sont passées au parser PHP.
- ⇒ Support de cache intégré.

Ce Template est OO.

Pour l'utiliser, on inclut le moteur de Template "setup\_smarty", on crée un objet et on utilise un pseudo langage pour les Templates.

Exemple de ce gestionnaire

Fichier de présentation

header.tpl

```
{* Smarty *}
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<!-- Balise meta -->
<meta name="title" content="{ $titre}" />
<meta name="description" content="{ $description}" />
<meta name="keywords" content="{ $mots_cles}" />
<!-- Indexer et suivre -->
<meta name="robots" content="index,follow" />
<!-- Relier une feuille CSS externe -->
<link rel="stylesheet" href="{ $fichier_css}" type="text/css" />
<!-- Relier un fichier JavaScript -->
<script type="text/javascript" src="{ $javascript}"></script>
<noscript>Attention votre navigateur n'autorise pas l'execution des scripts
<br>
Le Site ne peut pas fonctionner correctement
</noscript>
</head>
<header>
<!-- CSS + IMAGE -->

</header>
```

Error.tpl

```
{* Smarty *}
{include file="header.tpl"}
<body>
<div id="conteneur">
<h2>{ $titre_admin}</h2>
{ $texte_erreur}
</div>
</body>
```

Script d'usage :

```
require('/parametre/bd.php');
require('setup_smarty.php');
$smarty = new Smarty_pw();
$smarty->setCacheLifetime(5);
//données pour header
$smarty->assign('titre','Jeux&Manga');
$smarty->assign('description','Application de gestion');
$smarty->assign('mots_cles','Jeux,Manga');
$smarty->assign('fichier_css','../css/all.css');
$smarty->assign('javascript','');
$smarty->assign('image_header1','../images/header1.jpg');
```

## 10 Gestion des Erreurs

Lors d'un développement professionnel, on doit apporter un soin particulier à la gestion d'**erreurs**.

**Une erreur est tout comportement inattendu par rapport à ce que le développeur voulait avoir.**

Attention, on ne parle pas ici d'**erreurs** de conception.

On peut faire la distinction entre les erreurs lors du développement et les erreurs lors de la production de l'application.

Une erreur peut provenir :

- ⇒ Erreur de syntaxe.
- ⇒ Erreur de comportement.
- ⇒ Erreur de parcours.
- ⇒ Erreur du serveur ...

L'idée importante est de pouvoir repérer l'erreur, l'intercepter et/ou l'enregistrer. On veut un moyen pour connaître l'erreur afin d'y remédier (**ce** n'est pas toujours évident).

### 10.1 Gestion d'erreurs – Kit de survie

Date d'édition :	20/02/2012		Page :	34 / 44

### 10.1.1 php.ini

Le moteur de PHP sait gérer des erreurs mais encore faut-il qu'il soit bien paramétré.

⇒ Dans le fichier php.ini, on a une directive indiquant le niveau de remontée des erreurs par PHP.

Tableau niveaux d'erreurs

E_ALL	Toutes les erreurs et les warnings (E_ALL n'inclut pas tous les niveaux d'alerte)
E_PARSE	Erreur de syntaxe
E_ERROR	Erreur critique
E_STRICT	Suggestions de PHP pour assurer une meilleure interopérabilité et compatibilité du code.
E_DEPRECATED	Notice pour les éléments dépréciés dans le futur

Il existe plusieurs niveaux d'erreurs (+10) avec des combinaisons entre eux.

Les combinaisons sont écrites simplement :

`error_reporting= E_ALL & ~E_PARSE` [on reporte les erreurs E\_ALL et pas celles remontées par E\_PARSE ]

`error_reporting= E_ALL | E_STRICT` [on reporte les erreurs remontées par E\_ALL ou par E\_STRICT ]

Si vous ne pouvez pas modifier le fichier.ini, vous pouvez utiliser en début de script la fonction :

```
error_reporting( E_ERROR | E_WARNING | E_PARSE);
```

⇒ La directive d'affichage

La directive `display_errors` (On/Off) permet d'afficher ou pas les erreurs.

⇒ La directive de log

La directive `log_errors` (On/Off) permet d'indiquer si on peut logger les erreurs.

La directive `error_log = "c:/wamp/logs/php_error.log"` indique où.

Logger les erreurs pour pouvoir les analyser et très important notamment lorsqu'une application est en production car on ne peut pas afficher les erreurs PHP aux utilisateurs.

### 10.1.2 Dans vos script

- ⇒ Utiliser @ devant une fonction pour qu'elle n'affiche pas d'erreur ou warning à condition que vous gériez l'erreur.
- ⇒ Tester les retours des fonctions. Vous pouvez utiliser If/Else mais rend le code illisible.
- ⇒ Utiliser la méthode 'OR'. Intéressant pour substituer les If/Else
- ⇒ Déclencher une erreur avec trigger\_error(); Très Propre mais apprentissage.

Ex de script :

```
// $nombre=25;  
$nombre ="fred";  
  
$result=@is_int($nombre)  
or trigger_error ("ERREUR PAS UN INT",E_USER_ERROR);
```

(!) Fatal error: ERREUR PAS UN INT in C:\wamp\www\cours_php\test_erreur.php on line 7				
Call Stack				
#	Time	Memory	Function	Location
1	0.0007	671376	{main}()	..\test_erreur.php:0
2	0.0007	671864	trigger_error()	..\test_erreur.php:7

La fonction trigger\_error () :

```
bool trigger_error ( string $error_msg [, int $error_type = E_USER_NOTICE ] )
```

Les erreurs que l'on peut indiquer sont de 3 types :

E_USER_ERROR	= E_ERROR – Le script s'arrête
E_USER_WARNING	=E_WARNING – Le script continue
E_USER_NOTICE	=E_NOTICE – Le script continue

**Δ** L'utilisation de la fonction die ou exit() n'est pas un système de gestion d'erreur.

## 10.2 Les assertions

Les assertions sont des mécanismes pour tester la logique des données.

Ex : le développeur veut s'assurer qu'un nombre d'adhérents ne peut être négatif.

**Attention :** Les assertions ne doivent pas être utilisées pour gérer la logique de l'application. La logique applicative est faite par des structures de contrôle If/Else/Elseif

En règle générale, les assertions sont utilisées pendant le développement et on doit pouvoir supprimer toutes les assertions lors de la mise en production de l'application.

Pour utiliser les assertions, il faut les activer :

```
assert_options(ASSERT_ACTIVE, 1); // On active les assertions
```

On peut aussi les activer dans php.ini via la directive `assert.active`.

Utilisation des assertions :

Ex : On a une application qui doit gérer des prix. On récupère les prix via une requête, un calcul, ou dans un fichier par exemple. On sait que les prix des articles ne peuvent pas être  $\leq 0$ . On va utiliser une assertion pour vérifier cela lors du développement.

```
assert_options(ASSERT_ACTIVE, 1); // On active les assertions

$prix_recup = 0;
assert('$prix_recup > 0'); // Le prix ne peut être 0 ou -
```

 Warning: assert() [function.assert]: Assertion "\$prix\_recup > 0" failed in C:\wamp\www\cours\_php\test\_erreur.php on line 14

### Call Stack

#	Time	Memory	Function	Location
1	0.0014	674056	{main}()	..\test_erreur.php:0
2	0.0014	674856	assert()	..\test_erreur.php:14

On peut déduire que la donnée de prix récupérée n'est pas bonne, problème de requête, du calcul, de lecture du fichier...

L'assertion est un moyen rapide et simple de tester des informations.

### Astuce :

- ⇒ On peut dans mettre un commentaire dans le message d'assert(), ce qui permet de donner des indications lors du développement.
- ⇒ On peut aussi créer une fonction permettant de gérer l'assertion avec `assert_options(ASSERT_CALLBACK, 'nom_fct_gestion_assertion')`

Date d'édition :	20/02/2012		Page :	37 / 44

## 10.3 Les Exceptions

### 10.3.1 Principe

On a vu plus haut l'usage de la levée d'exceptions lors d'une connexion à la base de données, ce mécanisme est très utile, pour le développeur car c'est lui qui détermine les problèmes qu'il veut remonter.

Les Exceptions permettent la gestion logique de l'application.

Comme en JAVA, la structure de gestion des exceptions est →

```
try {  
    //Code de connexion  
}  
catch(Exception $e){  
    //Code si une erreur a été levé  
}
```

Quand vous développez, pensez à lever (throw) et à attraper (catch) les exceptions, autrement dit c'est au développeur de savoir ce qu'il estime être une Exception.

PHP a par défaut 2 classes pour les exceptions :

- ⇒ Classe Exception
- ⇒ Classe RuntimeException extends Exception

Avec ces classes on a des méthodes pour travailler sur l'exception levée :

- ⇒ getMessage() → Récupère le message de l'exception
- ⇒ getCode () → Récupère le code de l'exception
- ⇒ getFile() → Récupère le fichier dans lequel l'exception est survenue
- ⇒ getLine() → Récupère la ligne dans laquelle l'exception est survenue

#### A Savoir :

- ⇒ Le moteur PHP ne lève pas d'exception pour ses erreurs.  
Ainsi, si vous faites une erreur de syntaxe E\_PARSE n'envoie pas d'exception mais une erreur classique, c'est le cas pour tous les types d'erreurs PHP.

### 10.3.2 Usage

Ex : On imagine une application de calcul de prix TTC à partir du HT. On doit avoir un prix supérieur à 0 au niveau du prix HT sinon l'application n'a pas d'usage.

Script test\_exception :

```
function calcul_TTC($HT,$TVA){
    try{
        if($HT<=0){
            $message_erreur="Le prix n'est pas > 0 ";
            $exception=new Exception($message_erreur);
            throw $exception;
        }
    }
    catch (Exception $error){
        echo "Une Exception a surgi<br>";
        echo "Message d'erreur : ".$error->getMessage()."<br>";
        echo "Erreur ligne : ".$error->getLine()."<br>";
    }
}

$prix_HT = 0;
$res=calcul_TTC($prix_HT,"1.196");
```

Affichage :

Une Exception a surgi  
Message d'erreur : Le prix n'est pas > 0  
Erreur ligne : 8

Ce qui est important c'est l'usage de **throw** qui lève réellement l'exception.

On peut utiliser la fonction :

**error\_log** ( string \$message [, int \$message\_type = 0 [, string \$destination [, string \$extra\_headers ]]] )

pour logger l'erreur ce qui est **vivement conseillé**.

On peut créer sa propre classe d'exception en dérivant d'une classe exception.

Cela peut être utile notamment si vous développez des applications qui travaillent avec des ressources spécifiques (URL, Fichier, Base de données...)

Création d'une classe d'exception pour des ressources URL :

```
class Exception_URL extends Exception {
    public $URL;

    public function __construct($message,$url){
        $this->URL=$url;
        parent::__construct($message);
    }

    public function getURL(){
        return $this->URL;
    }
}
```

```
$url="http://google.fr";
try {
    if ($url!="http://google.fr"){
        $message_erreur="L'URL n'est pas disponible ";
        $exception=new Exception_URL($message_erreur,$url);
        throw $exception;
    }
} catch (Exception_URL $error){
    echo "Une Exception a surgi<br>";
    echo "Message d'erreur : ".$error->getMessage()."<br>";
    echo "Erreur ligne : ".$error->getLine()."<br>";
    echo "URL en erreur : ".$error->getURL()."<br>";
}
```

## 11 Développement et sécurité

### 11.1 Les points clés

Pour une application web, le développeur est en face d'au moins 4 problématiques :

- ⇒ Disponibilité de l'application – Le développeur doit s'assurer de la montée en charge.
- ⇒ Intégrité des données – Le développeur est le garant des données.
- ⇒ Intégrité du site – Le développeur doit s'assurer du détournement de l'application.
- ⇒ Sécurité des données – Le développeur doit veiller à la non divulgation des données.

**La sécurité est une préoccupation du développeur mais ne doit pas être une obsession.**

### 11.2 Tout le monde ment !

#### 11.2.1 Ne fait pas confiance au utilisateur.

Vérifier que la donnée est de la forme que vous attendez.

Pour la vérification, les expressions régulières et la fonction `preg_match()` sont redoutables.

Exemple :

Dans un formulaire, on demande le nom d'une personne.

On peut vouloir vérifier que le nom soit forcément entre 3 et 10 caractères alphabétiques majuscule ou minuscule.

Expression régulière : `/^[a-z]{3,10}$/i`

Informations sur les expressions régulières :

<http://www.siteduzero.com/tutoriel-3-14608-les-expressions-regulieres-partie-1-2.html>

<http://www.expreg.com/presentation.php>

```
$nom=htmlspecialchars($_POST['login']);  
  
if (preg_match("/^[a-z]{3,10}$/i", $_POST['login'])) {  
    echo "tout OK";  
} else {  
    echo "mauvaise data";  
}
```

La fonction `preg_match($_REGEX,$_DONNEE)` renvoie 0 si le REGEX n'a pas été trouvé ou 1, on peut donc l'utiliser pour valider des données.

Date d'édition :	20/02/2012		Page :	40 / 44



### 11.2.2 Injection SQL

L'injection SQL est un moyen de détourner l'utilisation des requêtes de l'application.

Pour faire de l'injection SQL, il faut connaître un minimum le langage SQL et PHP.

Ex : Dans de nombreuses applications, l'utilisateur doit s'authentifier pour accéder à son compte. Une fois les données saisies, on lance une requête pour vérifier que la personne a un compte et qu'il est valide.

<p>Identifiant : <input type="text" value="Votre identifiant"/></p> <p>Mot de passe : <input type="text" value="Votre Mot de Passe"/></p> <p><input type="button" value="Envoyer"/></p>	<p>Page securite.htm</p>
<pre>\$login=\$_POST['login']; \$pass=\$_POST['pass'];  \$sql="SELECT count(*) as nombre       FROM comptes       WHERE cp_identifiant='\$login' and cpt_pass='\$pass' ";</pre>	<p>Page securite.php</p> <p>On récupère le login et le password et on construit une requête SQL que l'on va ensuite passer à notre base de données.</p>

Sortie SQL et résultat :

Login =fred et le password jkklj

```
SELECT count(*) as nombre FROM comptes WHERE cp_identifiant='fred' and cpt_pass='jkklj'
```

**ERREUR ERREUR**

C'est Normal. Mais imaginons que je suis mal intentionné, et que je veux passer.

Login =theronf' –

Password ="vide"

```
SELECT count(*) as nombre FROM comptes WHERE cp_identifiant='theronf' -- ' and cpt_pass='vide'
```

**AUTORISATION ACCORDEE**

Pourquoi, le système m'authentifie ?

Mon injection SQL a fonctionné.

En SQL -- indique un commentaire, du coup la requête pour le système a été :

```
SELECT count(*) as nombre FROM comptes WHERE cp_identifiant='theronf'
```

Date d'édition :	20/02/2012		Page :	41 / 44

La requête écrite par le développeur n'a même pas été jouée.

Comment se prévenir de ce type de problème : **Tout le monde ment ! Donc vérifier les données. (11.2.1)**

**Avec PDO**, vous pouvez créer des requêtes préparées et utiliser la fonction PDO **quote()** pour gérer les problèmes de SQL.

**Sinon** utiliser une fonction spécifique à votre base de données :

⇒ pour MySQL : `mysql_real_escape_string()`,


⇒ Pour PostGres : `pg_escape_string`

**Ou** utiliser `addslashes()`. → Cette fonction n'est pas efficace sur tous les problèmes d'injection SQL

Avec PDO :

```
$login=$_POST['login'];
$pass=$_POST['pass'];
$login=$connexion->quote($login);
$pass=$connexion->quote($pass);
```

`SELECT count(*) as nombre FROM comptes WHERE cp_identifiant="theronf' -- " and cpt_pass="jhjhjk"`

 <b>Fatal error: Call to a member function fetch() on a non-object in C:\wamp\www\cours_php\securite.php on line 32</b>				
Call Stack				
#	Time	Memory	Function	Location
1	0.0014	684016	{main}()	..\securite.php:0

#### A Savoir :

Il existe dans `php.ini`, une variable `magic_quotes_gpc` qui fixe le mode `magic_quotes` pour les opérations GPC (Get/Post/Cookie). Lorsque `magic_quotes` est activé, tous les caractères ' (guillemets simples), " (guillemets doubles), \ (antislash) et NUL sont échappés avec un antislash.

Avertissement : Cette fonctionnalité est OBSOLETE depuis PHP 5.3.0.

Date d'édition :	20/02/2012		Page :	42 / 44

### 11.2.3 Session – Cookies et sécurité.

Le protocole HTTP est dit "**stateless**", ou autrement dit sans état. Pour un serveur, chaque requête qu'il reçoit est indépendante de la précédente par conséquent, il nous faut un mécanisme pour conserver les données de l'utilisateur, pour ce faire on peut utiliser les sessions et/ou les cookies.

#### Cookies :

Un cookie est généré et envoyée par un serveur HTTP à un **client http** qui le conserve (soit dans sa mémoire, soit dans un fichier texte), et ce dernier le retourne lors de chaque interrogation du même serveur HTTP.

Générer un cookie :

```
<?php  
setCookie('nom_du_cookie','data_cookie ou array()', Duree_cookie);
```

Le cookie doit être la première chose **créée** dans une page php.

Lire un cookie :

```
echo $_COOKIE['nom_cookie'];
```

#### Session :

La session est un mécanisme qui permet à PHP de garder "en mémoire" du **côté du serveur** un nombre illimité de valeurs entre plusieurs requêtes d'un même utilisateur. Une session est identifiée par un Id unique.

Créer une session :

```
session_start();
```

Dans chaque page de l'application où on doit utiliser les sessions, il faut appeler cette fonction qui initialise la session.

```
$_SESSION['nom_variable']="valeur";
```

Affecter une valeur à la session.

Du bon usage de la session et du cookie :

- ⇒ Ne pas utiliser les cookies pour l'authentification (mot de passe). **Fichier texte**
- ⇒ Stocker dans le cookie des informations non essentielles à l'usage du site. **Le cookie peut être supprimé.** ( ex : panier d'articles d'un site marchand)

Date d'édition :	20/02/2012	Page :	43 / 44

- ⇒ Lors de l'authentification dans la session, noter une information telle que **l'adresse IP** rend le vol de sessions plus compliqué, à condition de vérifier l'adresse IP dans le script.
- ⇒ Pour rendre le vol de session, plus compliqué encore lier **des données de la session avec des données du cookie** et vérifier les deux données à chaque script.
- ⇒ Utiliser des fonctions de cryptage pour des données sensibles. [md5(), mcrypt()]

#### 11.2.4 Vérifier vos logs

Dans une application, les développeurs négligent souvent l'analyse des logs de production, c'est une erreur.

Dans le fichier php.ini, on peut indiquer où sont les logs mais si on ne les analyse pas régulièrement, on se prive d'informations importantes, mais il faut aussi analyser les logs du serveur web aussi qui donne de bonnes informations.

Log d'accès Apache (Log automatique):

```
127.0.0.1 - - [19/Feb/2012:09:26:20 +0100] "GET /cours_php/session_cookie.php HTTP/1.1" 200 19
127.0.0.1 - - [19/Feb/2012:09:26:20 +0100] "GET /favicon.ico HTTP/1.1" 404 209
127.0.0.1 - - [19/Feb/2012:09:26:20 +0100] "GET /cours_php/session_cookie.php HTTP/1.1" 200 19
127.0.0.1 - - [19/Feb/2012:09:26:21 +0100] "GET /cours_php/session_cookie.php HTTP/1.1" 200 19
127.0.0.1 - - [19/Feb/2012:09:26:21 +0100] "GET /favicon.ico HTTP/1.1" 404 209
127.0.0.1 - - [19/Feb/2012:09:26:23 +0100] "GET /cours_php/session_cookie1.php HTTP/1.1" 200 19
127.0.0.1 - - [19/Feb/2012:09:26:23 +0100] "GET /favicon.ico HTTP/1.1" 404 209
127.0.0.1 - - [19/Feb/2012:10:05:17 +0100] "GET /phpmyadmin/sql.php?db=pwd&token=42cbdb3496f3f6f2ddf1280fa3059a0c&table=pwd&pos=0 HTTP/1.1" 200 82842
127.0.0.1 - - [19/Feb/2012:10:05:18 +0100] "GET /phpmyadmin/phpmyadmin.css.php?server=1&token=42cbdb3496f3f6f2ddf1280fa3059a0c&js_frame=right&nocache=52881 HTTP/1.1" 200 5478
127.0.0.1 - - [19/Feb/2012:10:05:18 +0100] "GET /phpmyadmin/js/messages.php?lang=fr&db=pwd&token=42cbdb3496f3f6f2ddf1280fa3059a0c HTTP/1.1" 200 5478
127.0.0.1 - - [19/Feb/2012:10:05:20 +0100] "GET /phpmyadmin/themes/pmahomme/img/b_relations.png HTTP/1.1" 200 217
```

Log d'erreur Apache (Log automatique):

```
[Sun Feb 19 09:25:45 2012] [error] [client 127.0.0.1] File does not exist: C:/wamp/www/favicon.ico
[Sun Feb 19 09:26:18 2012] [error] [client 127.0.0.1] File does not exist: C:/wamp/www/favicon.ico
[Sun Feb 19 09:26:20 2012] [error] [client 127.0.0.1] File does not exist: C:/wamp/www/favicon.ico
[Sun Feb 19 09:26:20 2012] [error] [client 127.0.0.1] File does not exist: C:/wamp/www/favicon.ico
```

Log d'Erreur PHP (Log créé cf. 10.3.2) :

```
Erreur PDO contact Serveur
```

Date d'édition :	20/02/2012	Page :	44 / 44