

A thick dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date. Below the arrow, several thin, curved lines in dark blue and light grey sweep upwards from the bottom left corner.

28/07/2022

PHP EXERCICE MVC

Le Vidéo Club MVC

Rudy Lesur

<u>I</u>	<u>Objet de l'exercice</u>	<u>3</u>
<u>II</u>	<u>Choix d'implémentation</u>	<u>3</u>
1/	Choix d'implémentation pour les Contrôleurs :	3
2/	Choix d'implémentation pour les Vues :	4
3/	Choix d'implémentation pour les Modèles :	4
<u>III</u>	<u>Préliminaires</u>	<u>4</u>
<u>IV</u>	<u>Première étape : écran d'accueil.....</u>	<u>5</u>
1/	Contrôleur	5
2/	Vue	5
<u>V</u>	<u>Deuxième étape : réservation de film - page 1</u>	<u>6</u>
1/	Contrôleur	6
2/	Model : La classe DAO	7
3/	Vue : Présentation	8
<u>VI</u>	<u>Troisième étape : réservation de film - page 2</u>	<u>9</u>
1/	Contrôleur	9
2/	Model : La classe DAO	10
3/	Vue : Présentation	11
<u>VII</u>	<u>Quatrième étape : réservation de film - page 3</u>	<u>12</u>
1/	Contrôleur	12
2/	Vue : Présentation	13

<u>VIII Cinquième étape : réservation de film - page 4</u>	14
1/ Contrôleur	14
2/ Model : La classe DAO	16
3/ Vue : Présentation	17
<u>IX Sixième étape : accès au module administration</u>	19
<u>X Septième étape : saisie d'un nouveau film.....</u>	19
1/ Contrôleur	20
2/ Model : La classe DAO	20
3/ Vue : Présentation	20
<u>XI Huitième étape : Enregistrement d'un nouveau film ..</u>	21
Contrôleur	21
2/ Model : La classe Métier Mfilm	22
3/ Model : La classe DAO	23
4/ Vue : Présentation	24
<u>XII Bilan de l'exercice</u>	25
1/ Fondamentaux.....	25
2/ En PHP/MySQL	25
Contrôleurs.....	25
Vues/Présentation	25
Métier.....	26
DAO	26

I Objet de l'exercice

Il s'agit dans cet exercice de *transformer* une application Web opérationnelle, depuis une version écrite en code PHP procédural « classique » (code « *spaghetti* »), vers du code PHP « *en couches* » et « *orienté objet* » selon l'architecture MVC, sans apporter aucune modification aux fonctionnalités. Reprenez votre projet réalisé précédemment.

II Choix d'implémentation

Le langage PHP en version 5 n'offre pas encore toutes les fonctionnalités attendues d'un langage orienté objet (comme celles qui caractérisent C# ou Java).

De plus, pour une application Web, instancier des objets et, surtout, mémoriser des objets et collections d'objets tout au long d'une *transaction* est rapidement très coûteux en ressources serveur ; aussi on privilégiera ici des *accès ponctuels*, page par page, à la Base de Données, à partir de *données transmises de page en page* plutôt que la mémorisation de variables, d'objets ou collections d'objets en *variables de session*.

1/ Choix d'implémentation pour les Contrôleurs :

Par définition, un **Contrôleur** contrôle le déroulement des opérations. Il doit être concis de manière à *faire ressortir la logique applicative*.

Les Contrôleurs seront appelés par les différents liens entre pages Web (< a href...> ou action de formulaires). On définira donc un script Contrôleur pour chaque page Web.

Un Contrôleur sera développé en *langage PHP procédural classique* (non objet). Il fera appel à des fonctions/méthodes externes de haut niveau (implémentées dans les autres couches). *Il ne contiendra que du code PHP.*

La logique générale d'un Contrôleur sera :

- *inclure tous les scripts externes nécessaires (autres couches, modules génériques...)*
- *recupérer éventuellement les données transmises par la page précédente (en méthode HTTP get ou post)*
- *préparer l'accès à la base de données et appeler les fonctions/méthodes nécessaires de la couche DAO*
- *traiter ces données*
- *appeler les fonctions/méthodes de la couche présentation de manière à générer la page Web suivante en passant éventuellement en paramètres les données à afficher*

2/ Choix d'implémentation pour les Vues :

On définira un script de Présentation pour chaque page Web.

Un script de la **couche Présentation** sera développé en *code « PHP spaghetti » procédural* (mêlant langages PHP, HTML, JavaScript et CSS) et exposera des *fonctions paramétrées* permettant de générer des portions de page ou même des pages complètes personnalisées en fonction de données reçues en paramètres.

Notons que toutes les pages du site ont la même structure (bandeau supérieur de titre, menu d'actions à gauche, contenu principal au centre) définie par un tableau HTML ; chaque partie fait déjà l'objet de scripts élémentaires à fusionner dans les différentes pages du site. Les pages d'administration se distinguent par une variante du bandeau et une couleur de fond différente.

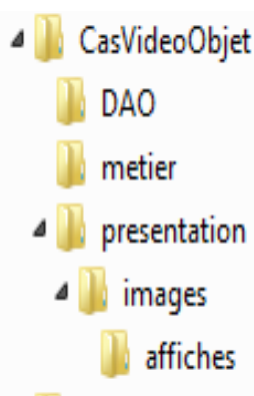
3/ Choix d'implémentation pour les Modèles :

Les **objets Métier** pourront être développés en *PHP orienté objet* de manière à bénéficier des possibilités de contrôle et/ou transformation assurés par les *accesseurs aux attributs privés*. On privilégiera les *tableaux associatifs (non objet)* pour représenter les *collections d'objets Métier*, en particulier lors des affichages de listes, de manière à économiser les ressources serveur.

La **couche DAO** sera implémentée en *PHP orienté objet* sous forme de classes exposant des *méthodes statiques* (de manière à économiser des instanciations d'objets coûteuses en ressources serveur). Afin de rendre les couches Métier et Contrôleur indépendantes du SGBD utilisé par la couche DAO, les méthodes de sélection d'enregistrements en tables retourneront des *objets Métier élémentaires* ou de *simples tableaux associatifs*.

III Préliminaires

Créer tout d'abord une arborescence de dossiers de manière à stocker les différents scripts et images constituant le site.



Le dossier « CasVideoObjet » correspondant au site ; ce dossier contient lui-même les sous-dossiers « DAO » pour la couche Accès aux données, « metier » pour la couche Métier et « presentation » pour la couche Vue ; les Contrôleurs seront stockés directement dans le dossier « CasVideoObjet ».

Récupérer les composants actuels du site développé en PHP classique ; recopier les images dans les dossiers adéquats du nouveau site.

IV Première étape : écran d'accueil

1/ Contrôleur

Créer le premier Contrôleur, **VCIAccueil.php**, stocké directement dans le dossier **CasVideoObjet**, permettant d'afficher la page d'accueil. La logique applicative se limite à un affichage non paramétré :

```
<?php // ***** code PHP objet en couches *****
// script contrôleur page accueil site Video-club
// module présentation
require("presentation/VCIAccueil.vue.php");

// afficher page
AfficheEcranAccueil();
?>
```

Par convention, tous les scripts de Présentation prendront le nom du Contrôleur correspondant avec une double extension.**vue.php**

2/ Vue

Créer le script de vue correspondante **VCIAccueil.vue.php**, stocké dans le sous-dossier **presentation**, à partir de la page **VCIAccueil.php**; il s'agit simplement de transformer le code PHP procédural en une fonction PHP (copier/coller...) :

```
<?php // ***** code PHP objet en couches *****
// script affichage page accueil site Video-club
function AfficheEcranAccueil(){
?>
<html>
<head>
<title>Bienvenue au Vidéo-Club !</title>
```

Copier ici le code de votre page **VCIAccueil.php**.

Dans cette page, nous avons inclus une page contenant le titre (**VCITitre.php**) et une page contenant la colonne menu (**VCIMenu.htm**).

Copiez ces 2 fichiers dans le dossier "presentation" et changez le chemin du fichier dans votre code.

```
</html><?php
}
?>
```

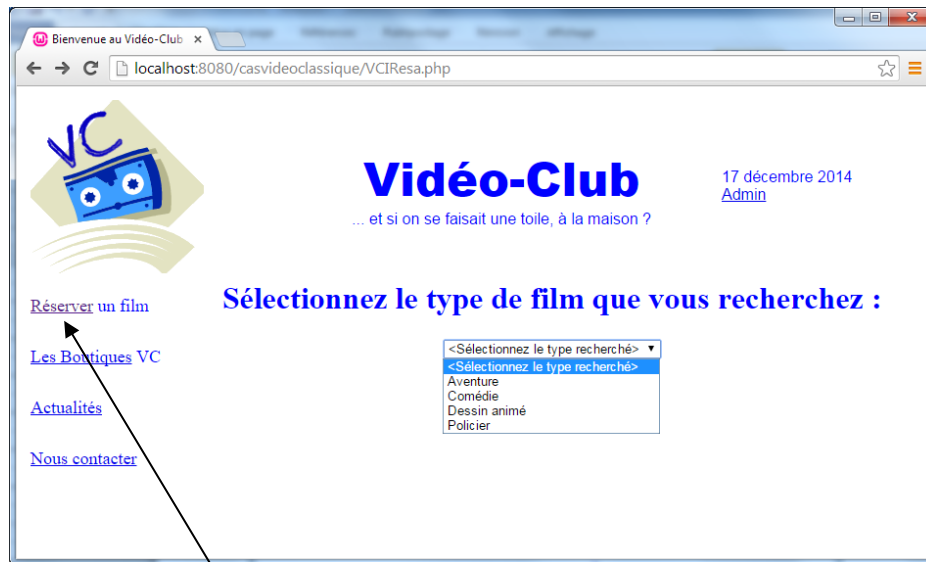
Tester l'affichage.

Les liens du menu de gauche sont définis dans le script de présentation **VCIMenu.html** ; la page en construction est définie par un Contrôleur qui reste à écrire sur le même modèle que le Contrôleur **VCIAccueil.php**. A vous de jouer...

Tester et mettre au point la page d'accueil.

V Deuxième étape : réservation de film - page 1

1/ Contrôleur



Sur l'écran d'accueil, le choix **Réserver** doit permettre à l'utilisateur de choisir le type de film désiré. Le Contrôleur doit donc accéder à la Base de Données pour récupérer le contenu de la table **typefilm** et adresser cette liste au script de Présentation afin que celui-ci génère les balises HTML **<option> </option>** nécessaires.

Comme il s'agit d'afficher une liste, sans mise à jour directe, la méthode **ListeTypefilms()** de la couche DAO retournera un simple tableau associatif qui sera transmis au script de Présentation.

Ecrire le Contrôleur **VCIResa.php** :

<?php // script contrôleur pour lister les types de films

require("presentation/VCIResa.vue.php"); // module présentation

require("DAO/Video.DAO.PHP"); // module DAO BDD Video

// pour se connecter a la BDD

\$user = "root"; // ou autre indentifiant

\$password = ""; // le mot de passe s'il est défini

//charger la liste des types de films en array()

\$data = **array()** ;

\$data = **VideoDAO::ListeTypesFilms(\$user, \$password)**; // appel de la méthode statique de la classe VideoDAO

// présentation

AfficheListeFilmsParType(\$data);

?>

C'est au Contrôleur de déterminer le profil utilisateur qui accèdera à la Base de Données.

2/ Model : La classe DAO

La classe DAO **VideoDAO** contiendra ici toutes les méthodes nécessaires pour accéder à l'ensemble des tables de la Base de Données **Video** (on aurait aussi bien pu répartir ces méthodes dans différents scripts). Cette classe contient de plus deux méthodes privées permettant la connexion/déconnexion à la Base De Données :

```
<?php // classe DAO BDD Video
```

```
class VideoDAO{
```

```
    private static $cn;
```

La classe DAO détermine serveur et BDD

```
    // fonction de connexion à la BDD video
```

```
    private static function ConnectVideo($user, $password) {
```

```
        $dns = "mysql:host=localhost;dbname=video";
```

```
        VideoDAO::$cn = new PDO($dns, $user, $password)
```

```
        or die("Erreur de connexion au serveur MySQL");
```

```
    }
```

```
    // fonction de déconnexion de la BDD video
```

```
    private static function DisconnectVideo($result){
```

```
        $result->closeCursor();
```

```
        unset($result);
```

```
        VideoDAO::$cn=null;
```

```
    }
```

Les clauses **or die** pourraient être avantageusement remplacées par des structures **try/catch...**

Seule la couche DAO « sait » le type de Base de Données, les noms des tables et les instructions spécifiques à utiliser. Le Contrôleur doit simplement communiquer nom et mot de passe nécessaires à la connexion puis récupérer un simple tableau associatif, d'où la méthode **ListeTypesFilms()** :

```
// retourne un tableau des données types de films issues de la table TypeFilm
```

```
public static function ListeTypesFilms($user, $password){
```

```
    // connexion BDD
```

```
    VideoDAO::ConnectVideo($user, $password); // syntaxe pour appeler une méthode static
```

```
    // récupère liste types films depuis table typefilm
```

```
    $sql = "select CODE_TYPE_FILM, LIB_TYPE_FILM from typefilm order by LIB_TYPE_FILM";
```

```
    // echo $sql; // pour mise au point
```

```
    $result = VideoDAO::$cn->query($sql) or die ("Requete SQL invalide");
```

```
    // transformer recordset en tableau associatif
```

```
    $data = array();
```

```
    while($row = $result->fetch()) {
```

```
        $data[] = $row;
```

```
    }
```

```
    //facultatif, pour faire propre
```

```
    VideoDAO::DisconnectVideo($result);
```

```
    return $data;
```

```
}
```


3/ Vue : Présentation

Définir le script **VCIResa.vue.php** à partir du script procédural écrit précédemment **VCIResa.php** :

- Transformer l’affichage en une fonction PHP paramétrée
- Enlever la connexion à la base de données ainsi que la requête de recherche des types de film
- Pour la répétitive permettant de générer les balises HTML **<option></option>**, adapter le code d’affichage des données issues de la Base de Données, fourni en code spécifique MySQL, pour exploiter maintenant le simple tableau associatif reçu en paramètre (\$data) :

```
<?php // script affichage liste des types de films
```

```
// affiche boite select des types de films
```

```
function AfficheListeFilmsParType($data){
```

```
?>
```

```
<html>
```

```
<head>
```

```
<title>Bienvenue au Vidéo-Club !</title>
```

```
// Votre code pour l’affichage des infos provenant de votre exercice video club
```

```
</body>
```

```
</html>
```

```
<?php
```

```
}
```

```
?>
```

NB : le formulaire reste associé au script **VCIResa2.php** mais ce sera bientôt un script Contrôleur.

Tester et mettre au point.

VI Troisième étape : réservation de film - page 2

1/ Contrôleur

Ecrire le script Contrôleur **VCIResa2.php** qui doit assurer les traitements suivants :

- Inclure les scripts Présentation et DAO nécessaires
- Récupérer l'identifiant du type de film choisi dans le formulaire et transmis en méthode HTTP GET ou POST
- Demander à la couche DAO les informations concernant ce type de film ainsi que la liste des films de ce type pour demander à la couche Présentation d'afficher la page suivante contenant la liste des films du type reçu



```
<?php // script contrôleur lister les films d'un type particulier
// appelé par VCIResa.php
// recherche des films du type demandé en VCIResa.php
// et affichage en tableau avec liens a href personnalisés vers VCIResa3.php
```

```
require("presentation/VCIResa2.vue.php"); // module présentation
require("DAO/Video.DAO.PHP"); // module DAO BDD Video
```

```
// pour se connecter à la BDD
```

```
$user = "root";
```

```
$password = "";
```

```
// récupère données du type de film voulu (titre du type)
```

```
$rowTypeFilm = VideoDAO::RetourneTypeFilm($user, $password,$_GET["typef"] );
ou $_POST["typef"]
```

```
// récupère liste des films du type voulu
```

```
$dataFilms = array();
```

```
$dataFilms = VideoDAO::ListeFilmsParType($user, $password,$_GET["typef"]);
ou $_POST["typef"]
```

```
// présentation
```

```
AfficheListeFilms($rowTypeFilm, $dataFilms);
```

```
?>
```

NB : remarquer comme ces appels de fonctions de haut niveau font bien ressortir la logique applicative en rendant le code du Contrôleur clair et concis.

NB : la fonction de Présentation nécessite ici la passation de nombreuses informations (données complètes concernant le type de film, liste des films du type choisi) ce qui multiplie les paramètres passés à la fonction ; on pourrait très bien, dans une logique purement orientée Objet, encapsuler l'ensemble de ces données dans une classe afin de ne transmettre à la fonction que la référence d'un objet instancié. Il est fait ici le choix de multiplier les paramètres afin de limiter la charge de traitement serveur (car le code incluant une classe serait nécessairement plus volumineux, le langage PHP reste interprété, et une instanciation d'objet reste coûteuse en ressources).

2/ Model : La classe DAO

Ajouter à la classe **VideoDAO.php** le code de **RetourneTypeFilm()** et **ListeFilmsParType()** :

// retourne les données d'un type de film issu de la table TypeFilm

public static function RetourneTypeFilm(\$user, \$password, \$typevoulu){

// connexion BDD

VideoDAO::ConnectVideo(\$user, \$password);

// récupère 1 type film depuis la table TypeFilm

\$sql = "select CODE_TYPE_FILM, LIB_TYPE_FILM from typefilm where CODE_TYPE_FILM=" . \$typevoulu . " " ;

//echo \$sql; // pour test

\$result = **VideoDAO::\$cn->query**(\$sql) or **die** ("Requête SQL select typefilm invalide");

\$data = **\$result->fetch**();

//facultatif, pour faire propre

VideoDAO::DisconnectVideo(\$result);

return \$data;

}

// retourne la liste des films d'un type voulu

public static function ListeFilmsParType(\$user, \$password, \$typevoulu){

VideoDAO::ConnectVideo(\$user, \$password);

// requête des films du type demande (avec réalisateur)

// - projection

\$sql = "select ID_FILM, TITRE_FILM, ANNEE_FILM, ID_REALIS, REF_IMAGE, RESUME, NOM_STAR, PRENOM_STAR " ;

// - jointure

\$sql = **\$sql** . " from film, star ";

// - restriction concordance

\$sql = **\$sql** . " where film.ID_REALIS=star.ID_STAR ";

// - restriction utilisateur

\$sql = **\$sql** . " and CODE_TYPE_FILM=' " . \$typevoulu . " ' " ;

// - tri

\$sql = **\$sql** . " order by TITRE_FILM; " ;

//echo \$sql ; // pour mise au point

\$result = **VideoDAO::\$cn->query**(\$sql) or **die** ("Requete SQL select film,star invalide");

//transformer recordset en tableau

\$dataFilms = **array**();

while(\$rowFilm = **\$result->fetch**()) {

\$dataFilms[] = **\$rowFilm**;

}

//facultatif, pour faire propre

VideoDAO::DisconnectVideo(\$result);

return \$dataFilms;

}

3/ Vue : Présentation

Transformer le script **VCIResa2.php**, écrit précédemment, de manière à ne conserver que le code de présentation sous forme d'une fonction PHP ; il faut aussi adapter le code concernant l'affichage des données issues de la Base de Données car elles sont maintenant reçues par la fonction sous forme de simples tableaux associatifs :

```
<?php // script affichage liste des films d'un type
```

```
function AfficheListeFilms($rowTypeFilm, $dataFilms){
```

```
?>
<html>
<head>
<!--contenu du head de vos pages -->
</head>
<body>
<table width="100%" border="0" > <!-- tableau général de mise en page sans frame -->
```

```
.....
<!-- contenu principal de la page -->
<h1 align="center">
<font color ="blue" name="Arial">Liste des films du type
<?php echo $rowTypeFilm["LIB_TYPE_FILM"] ;?></font>
</h1>
```

```
<?php // si aucun film pour le type : pas de tableau
```

```
if (count($dataFilms)==0){
```

```
?>
<p align="center">
<font color ="blue" name="Arial"><b><i>Désolé, aucun film disponible pour le type
<?php echo $rowTypeFilm["LIB_TYPE_FILM"] ;?>.</font></i></b><br>
</p>
```

Pour une meilleure lisibilité du code, on aurait pu décomposer tout cela en différentes fonctions PHP

```
<?php //il y a des films pour le type demande
```

```
} else {
```

```
?>
<h2>Sélectionnez le film que vous désirez réserver : </h2>
```

```
<table border="1" align="center" width="80%" cellpadding="2" cellspacing="4"
bgcolor="#99FFFF" bordercolor="#0000FF">
```

```
.....
```

```
</body>
```

```
</html>
```

```
<?php
```

```
}
?>
```

Tester et mettre au point.

VII Quatrième étape : réservation de film - page 3

1/ Contrôleur



Afin de minimiser les accès à la Base de Données, il a été choisi ici de transmettre à cette page toutes les informations nécessaires à son affichage depuis la page précédente **VCIResa2.php**.

Le contrôleur **VCIResa3.php** doit donc assurer un traitement simple :

- Récupérer les données reçues en méthode HTTP GET ou en POST
- Demander leur affichage par la couche Présentation.

Ecrire le code du Contrôleur **VCIResa3.php** :

```
<?php // script contrôleur saisie coordonnées adhérent pour réservation
// page appelée par VCIResa2.php
```

```
require("presentation/VCIResa3.vue.php"); // module présentation
```

```
// récupération des données du film choisi et mise en tableau associatif pour passer à la couche
Présentation
```

```
$dataFilm["filmchoisi"] = $_GET["filmchoisi"] ;
$dataFilm["libfilmchoisi"] = $_GET["libfilmchoisi"] ;
$dataFilm["affiche"] = $_GET["affiche"] ;
$dataFilm["anfilmchoisi"] = $_GET["anfilmchoisi"] ;
$dataFilm["reafilmchoisi"] = $_GET["reafilmchoisi"] ;
```

```
// présentation
```

```
AfficheFormSaisieAdherent($dataFilm);
```

```
?>
```

Pour faciliter la passation de paramètres à la fonction de Présentation, ici encore il est fait le choix de regrouper toutes les données nécessaires dans un simple tableau associatif ; on aurait pu les encapsuler dans une classe.

2/ Vue : Présentation

Adapter le code de **VCIResa3.php** pour en faire la vue exposant une fonction paramétrée dans le fichier VCIResa3.vue.php dans le répertoire presentation :

```
<?php // script affichage form de saisie coordonnées adhérent
```

```
function AfficheFormSaisieAdherent($dataFilm){
```

```
?>
```

```
<html>
```

```
<head>
```

```
<title>Bienvenue au Video-Club !</title>
```

```
// copier/coller de VCIResa3.php
```

```
</html>
```

```
<?php
```

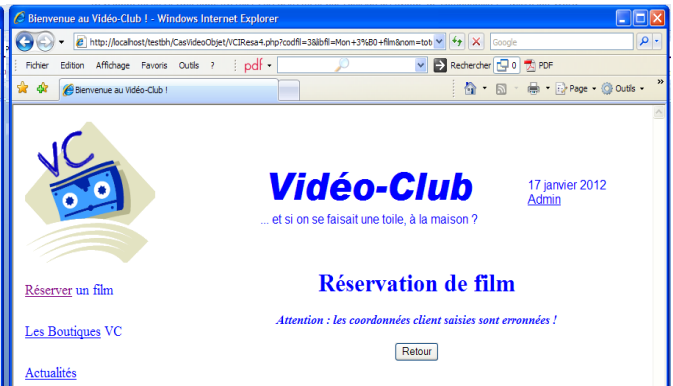
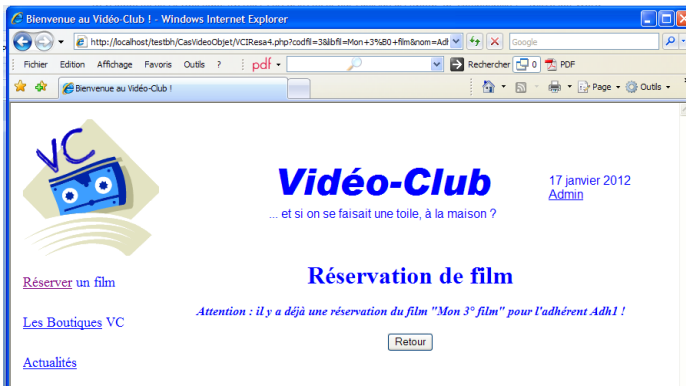
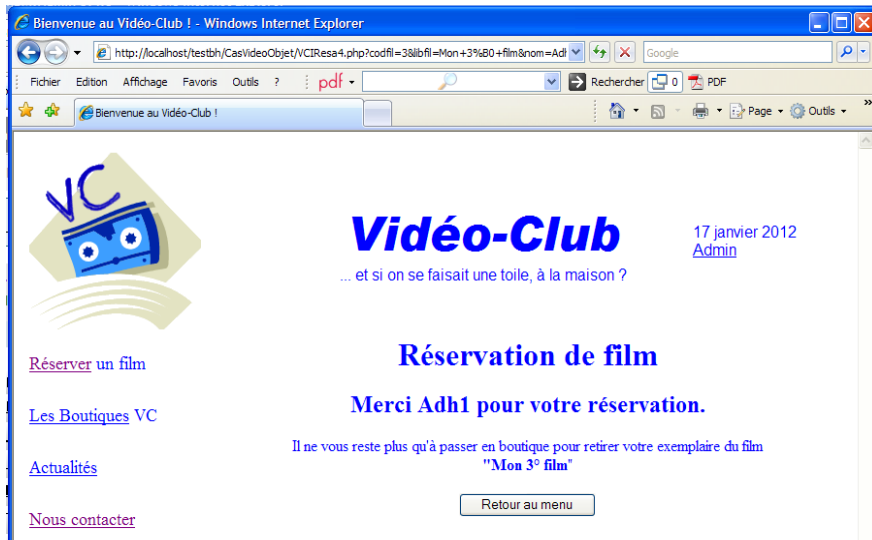
```
}
```

```
?>
```

Tester et mettre au point.

VIII Cinquième étape : réservation de film - page 4

1/ Contrôleur



Le Contrôleur **VCIResa4.php** va effectuer les contrôles nécessaires pour, si tout va bien, enregistrer la réservation ; il faudra afficher des variantes de la page selon les résultats de ces contrôles. Pour ce faire, et afin de mieux structurer le script fourni **VCIResa4.php**, la présentation sera décomposée en plusieurs fonctions affichant chacune une partie spécifique de la page.

Ecrire le Contrôleur **VCIResa4.php** ; comme toujours, il est plus facile de se laisser guider par la logique applicative et de concevoir au fil de l'eau les signatures de fonctions/méthodes nécessaires :

```
<?php // script contrôleur enregistrement effectif de la réservation
// page appelée par VCIResa3.php
// contrôles et insertion de la commande en bdd, affichage accuse de réception
// et retour à la page VCIAccueil.php
```

```
require("presentation/VCIResa4.vue.php");
require("DAO/Video.DAO.PHP");
```

```
$sejour = getdate();
$libsejour = $sejour["year"] . "-" . $sejour["mon"] . "-" . $sejour["mday"] ;
// echo $libsejour . "<br />" ; // pour test
```

```
// récupération données réservation et mise en tableau associatif
```

```
$dataResa["numadherent"] = $_GET["numadherent"];
$dataResa["nom"] = $_GET["nom"];
$dataResa["codfil"] = $_GET["codfil"];
$dataResa["libfil"] = $_GET["libfil"] ;
$dataResa["libsejour"] = $libsejour ;
```

```
// pour se connecter a la BDD
```

```
$user = "root";
$password = "";
```

```
// afficher début de page
```

```
AfficheDebutPage();
```

```
// si adhèrent inconnu afficher erreur adhèrent
```

```
if (!(VideoDAO::ExisteAdherent($user, $password, $dataResa)))
{
```

```
    AfficheErreurAdherentInconnu();
```

```
}
```

```
else
```

```
{
```

```
// s'il existe déjà une réservation de ce film, afficher erreur
```

```
    if(VideoDAO::ExisteResaPourCeClient($user, $password, $dataResa))
```

```
    {
```

```
        AfficheErreurResa($dataResa);
```

```
    }
```

```
    else // par d'erreur, enregistrer la réservation et afficher accusé réception
```

```
    {
```

```
        VideoDAO::InsereResa($user, $password, $dataResa);
```

```
        AfficheOKResa($dataResa);
```

```
    }
```

```
}
```

```
?>
```

A nouveau, on constate que l'appel à des fonctions/méthodes de haut niveau fait ressortir la logique applicative et rend le code plus concis et plus clair. Reste à écrire le contenu de ces fonctions/méthodes...

2/ Model : La classe DAO

Ajouter les 3 méthodes static nécessaires dans la classe **VideoDAO.php** :

```
// teste l'existence d'un adhérent dans la table Adherent
public static function ExisteAdherent($user, $password, $dataResa){
// connexion BDD
VideoDAO::ConnectVideo($user, $password);
// requête SQL
...
//facultatif, pour faire propre
VideoDAO::DisconnectVideo($result);

// retour booléen indiquant si trouve
return ...
}

// teste l'existence d'une réservation pour un client dans la table Location
public static function ExisteResaPourCeClient($user, $password,$dataResa){
// connection BDD
VideoDAO::ConnectVideo($user, $password);
// requête SQL
...
//facultatif, pour faire propre
VideoDAO::DisconnectVideo($result);

// retour booléen indiquant si trouve
return ...
}

// enregistre dans la table location
public static function InsereResa($user, $password, $dataResa){
// connection BDD
VideoDAO::ConnectVideo($user, $password);
// requete SQL
...
//facultatif, pour faire proper
VideoDAO::DisconnectVideo($result);

}
```

3/ Vue : Présentation

Adapter le script fourni **VCIResa4.php** afin de le décomposer en 4 fonctions d'affichage paramétrées. Ces 4 fonctions seront intégrées dans le fichier **VCIRes4.vue.php** du dossier **presentation**.

Début de page :

```
<?php // script affichage réservation (cas d'erreurs et réservation OK)

// début de page commun à tous les cas
function AfficheDebutPage() {
?>
<html>
<head>
...
</head>
<body >
<table width="100%" border="0" > <!-- tableau général de mise en page sans frame -->
    <tr>
        <td colspan="2"><!-- 1° ligne de titre -->
            <?php require("presentation/VCITitre.php") ; ?>
        </td>
    </tr>
    <tr><!-- 2° ligne : colonne menu et colonne contenu principal -->
        <td>
            <?php require("presentation/VCIMenu.html") ; ?>
        </td>
        <td align="left" valign="top">
            <!-- contenu principal de la page -->
            <h1 align="center">Réservation de film </h1>
            <?php
        }
    ?>
```

Cas d'erreur adhérent inconnu :

```
<?php // erreur : le client demande n'est pas trouvé
function AfficheErreurAdherentInconnu() {
?>
<div align="center">
<p><b><i>Attention : les coordonnées client saisies sont erronées !</i></b></p>
<form>
<input type="button" value="Retour" onClick="javascript:history.go(-1)">
</form>
</div>
<?php // bas de page
AfficheFinPage();
}
?>
```

On en profite pour restructurer le code PHP procédural et s'éloigner du « code spaghetti » indigeste...

Cas d'erreur réservation en doublon :

```
<?php // Erreur : il existe déjà une réservation de ce film pour ce client
function AfficheErreurResa($dataResa){
?>
<div align="center">
<p><b><i>Attention : il y a déjà une réservation du film "<?php echo $dataResa["libfil"]; ?>" pour
l'adhérent <?php echo $dataResa["nom"]; ?> !</i></b></p>
<form>
<input type="button" value="Retour" onClick="javascript:history.go(-1)">
</form>
</div>
<?php
// bas de page
AfficheFinPage();
}
?>
```

Cas correct, affichage de l'accusé de réservation :

```
<?php // réservation OK : affichage accusé
function AfficheOKResa($dataResa) {
?>
<div align="center">
<h2>Merci <?php echo $dataResa["nom"]; ?> pour votre réservation. </h2>
<p align="center">Il ne vous reste plus qu'à passer en boutique pour retirer votre exemplaire du
film<br> <b>"<?php echo $dataResa["libfil"]; ?>"</b></p>
<form>
<input type="button" value="Retour au menu" onClick="javascript:window.location.replace('VCIAccueil.php')">
</form>
</div>
<?php
// base de page
AfficheFinPage();
}
?>
```

Sous-fonction commune aux différents cas :

```
<?php
function AfficheFinPage(){
?>
<!-- fin de contenu principal de la page -->
        </td>
    </tr>
</table>
</body>
</html>
<?php
}
?>
```

Tester et mettre au point.

Avec une construction Objet de la couche Présentation, cette fonction deviendrait une *méthode privée* de la classe, assurant ainsi une meilleure sécurité dans la programmation.

IX Sixième étape : accès au module administration

L'accès au module d'administration du site se fait par le lien **Admin** du bandeau de titre. Ce lien affiche en surimpression (grâce à CSS) un formulaire de login. Actuellement, les données saisies ne sont pas contrôlées en Base de Données et le clic sur Go ! dirige systématiquement vers la page **VCIAdmin.php**.

Les transformations nécessaires pour reproduire à l'identique en restructurant selon l'architecture MVC ressemblent fort à ce qui a été fait en tout début de cet exercice :

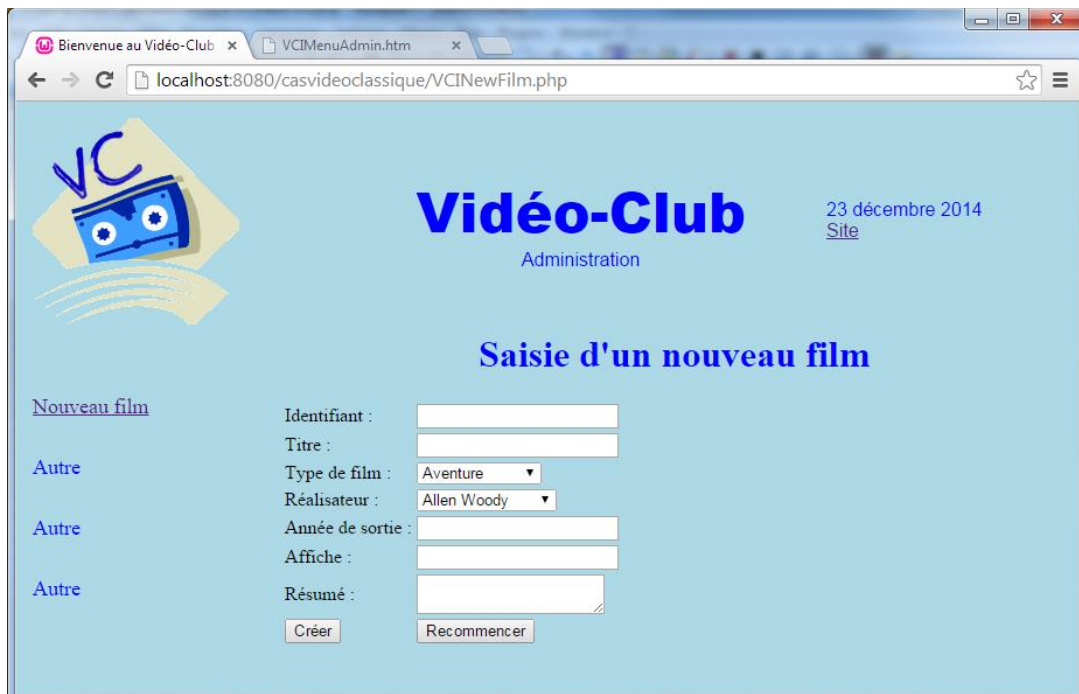
- Créer un script Contrôleur qui appellera une fonction de présentation écrite dans un script de présentation
- Adapter en fonction PHP le script fourni.

Réaliser les transformations nécessaires sur les scripts fournis.

Tester et mettre au point.

NB : le contrôle en Base de Données des informations de login serait bien entendu indispensable « dans la vraie vie » mais ce n'est pas l'objet de cet exercice.

X Septième étape : saisie d'un nouveau film



Bienvenue au Vidéo-Club x VCIMenuAdmin.htm x

localhost:8080/casvideoclassique/VCINewFilm.php

Vidéo-Club 23 décembre 2014
Administration [Site](#)

Saisie d'un nouveau film

[Nouveau film](#)

[Autre](#)

[Autre](#)

[Autre](#)

Identifiant :

Titre :

Type de film :

Réalisateur :

Année de sortie :

Affiche :

Résumé :

1/ Contrôleur

Dans la partie administration du site, le lien **Nouveau film** permet d'afficher un formulaire de saisie. La plupart des informations seront saisies dans des boîtes de texte mais, puisqu'un film est relié à un type de film et à un réalisateur existant, ceux-ci devront être choisis dans des boîtes de listes qui seront, au préalable, alimentée par le Contrôleur via la couche DAO.

Ecrire le script Contrôleur **VCINewFilm.php** appelé par le bandeau de menu des pages d'administration :

```
<?php // script controleur saisie d'un nouveau film
// module presentation
require("presentation/VCINewFilm.vue.php");
// module DAO BDD Video
require("DAO/Video.DAO.PHP");
// pour se connecter a la BDD pensez à créer un utilisateur en MySQL qui a des droits en écriture
$user = "utilweb";
$password = "utilweb";

//charger la liste des types de films en array()
$dataType = array();
$dataType = VideoDAO::ListeTypesFilms($user, $password);
// charger la liste des realisateurs
$dataReal = array();
$dataReal = VideoDAO::ListeRealisateurs($user, $password);
// afficher
AfficheFormNewFilm($dataType, $dataReal);
?>
```

2/ Model : La classe DAO

Le classe DAO expose déjà une méthode **ListeTypesFilms()** permettant d'obtenir un tableau PHP contenant toutes informations utiles sur les types de films existants dans la Base de Données (voir script **VCIResa.php**). Cette méthode est réutilisable dans l'état pour alimenter la boîte HTML **<select>** de ce nouveau formulaire. Comme cette architecture MVC sépare totalement l'extraction de données de son usage, notez que la méthode existante serait tout aussi réutilisable pour un autre mode d'affichage tel qu'une série de boutons-radio ou un tableau HTML.

Ecrire la nouvelle méthode static **ListeRealisateurs()** nécessaire ; à vous de jouer, elle ressemble fort à la liste des types de film... Ne pas oublier de trier les données suivant un ordre logique pour l'utilisateur et des retourner un simple tableau associatif PHP.

3/ Vue : Présentation

Il s'agit ici d'adapter le code procédural fourni **VCINewFilm.php** en une simple fonction de présentation ; rien de nouveau ; à vous de jouer...

XI Huitième étape : Enregistrement d'un nouveau film

Contrôleur

L'action associée au formulaire de la page **VCINewFilm.php** est bien l'appel d'un nouveau Contrôleur **VCINewFilm2.php** qui doit assurer :

- la récupération des données du formulaire reçues en méthode HTTP POST
- l'instanciation d'un objet Métier afin de bénéficier des avantages liés aux accesseurs **setXXX()** et autres méthodes comme **ToString()**
- l'appel de la méthode DAO permettant l'enregistrement du nouveau film en Base de Données
- la redirection vers la page d'accueil administration avec affichage du message confirmant la création du nouveau film.

Ecrire le script Contrôleur **VCINewFilm2.php** :

```
<?php // script contrôleur insertion de nouveau film
// classe metier
require("metier/MFilm.php");
// classe DAO
require("DAO/Video.DAO.php");

// instancie un objet Metier MFilm et le renseigne
$newFilm = new MFilm();
$newFilm->setID_FILM($_POST["ID_FILM"]);
$newFilm->setTITRE_FILM($_POST["TITRE_FILM"]);
$newFilm->setID_REALIS($_POST["ID_REALIS"]);
$newFilm->setANNEE_FILM($_POST["ANNEE_FILM"]);
$newFilm->setCODE_TYPE_FILM($_POST["CODE_TYPE_FILM"]);
$newFilm->setRESUME($_POST["RESUME"]);
$newFilm->setREF_IMAGE($_POST["REF_IMAGE"]);

//print_r($newFilm); // pour test
//echo $newFilm->ToString(); // pour test

// pour se connecter à la BDD
$user = "root";
$password = "";
// appel méthode DAO pour insert into
VideoDAO::InsererNewFilm($newFilm, $user, $password);

// redirige vers la liste des films avec message OK
header("Location: VCIAdmin.php?error=film " .
        $newFilm->getTITRE_FILM() . " insère avec succès");
exit();
?>
```

On instancie l'objet Métier grâce à son constructeur standard (unique en PHP) et on affecte les attributs privés grâce aux méthodes **setXXX()** qui assurent contrôles et/ou transformations

Ce script PHP de traitement du formulaire n'affiche pas de page spécifique mais redirige vers la page d'accueil d'administration avec un message de confirmation

2/ Model : La classe Métier Mfilm

Il est choisi ici de réaliser une classe Métier instanciable pour représenter les films :

- toutes les données du film sont stockées dans des attributs privés
- toutes les données du film sont accessibles en lecture par des méthodes publiques **getXXX()**
- toutes les données du film sont affectables par des méthodes publiques **setXXX()**
- un seul constructeur standard pour la classe
- une méthode **ToString()** qui permet de restituer « en clair » les données essentielles du film

Ecrire le script PHP de la classe **MFilm.php** et le stocker dans le dossier adéquat :

```
<?php
```

```
// classe Métier film
```

```
class MFilm {
```

```
    // attributs
```

```
    private $ID_FILM;
```

```
    private $CODE_TYPE_FILM;
```

```
    private $ID_REALIS;
```

```
    private $TITRE_FILM;
```

```
    private $ANNEE_FILM;
```

```
    private $REF_IMAGE;
```

```
    private $RESUME;
```

```
    // constructeur
```

```
    public function __construct(){} // même pas nécessaire...
```

```
    // getters
```

```
    public function getID_FILM(){return $this->ID_FILM;}
```

```
    public function getTITRE_FILM(){return $this->TITRE_FILM; }
```

```
    public function getCODE_TYPE_FILM(){return $this->CODE_TYPE_FILM;}
```

```
    public function getREF_IMAGE(){return $this->REF_IMAGE;}
```

```
    public function getID_REALIS(){return $this->ID_REALIS;}
```

```
    public function getRESUME(){return $this->RESUME;}
```

```
    public function getANNEE_FILM(){return $this->ANNEE_FILM;}
```

```
    // setters
```

```
    public function setID_FILM($value){ $this->ID_FILM = $value;}
```

```
    // titre converti en MAJ
```

```
    public function setTITRE_FILM($value) { $this->TITRE_FILM = strtoupper(trim($value)) ;}
```

```
    public function setCODE_TYPE_FILM($value) { $this->CODE_TYPE_FILM = $value;}
```

```
    public function setREF_IMAGE($value) { $this->REF_IMAGE = $value;}
```

```
    public function setID_REALIS($value) { $this->ID_REALIS = $value;}
```

```
    // capitale forcée au debut du résumé
```

```
    public function setRESUME($value) { $this->RESUME = ucfirst(trim($value)) ;}
```

```
    public function setANNEE_FILM($value) { $this->ANNEE_FILM = $value;}
```

```
    // méthode de restitution en clair
```

```
    public function ToString(){
```

```
        return "Film : " . $this->ID_FILM . " - " . $this->TITRE_FILM . " (" . $this->ANNEE_FILM . ")";
```

```
    }
```

```
?>
```

3/ Model : La classe DAO

Ajouter dans le script **VideoDAO.php** la fonction static **InsererNewFilm()** qui reçoit en paramètre la référence de l'objet Métier à insérer dans la Base de Données :

- Construire une requête SQL **insert into...** à partir des données récupérées de l'objet Métier grâce à ses accesseurs getxxx()
- Faire exécuter cette requête par MySQL

// insere une ligne dans la table Film

```
public static function InsererNewFilm($newFilm, $user, $password){
```

// connection BDD

```
VideoDAO::ConnectVideo($user, $password);
```

// requête insert SQL

```
$requete = "insert into film (ID_FILM, TITRE_FILM, ID_REALIS, CODE_TYPE_FILM, ANNEE_FILM, REF_IMAGE, RESUME) values (";
```

```
$requete .= $newFilm->getID_FILM() . ", ";
```

```
$requete .= " ' " . $newFilm->getTITRE_FILM() . " , ";
```

```
$requete .= " ' " . $newFilm->getID_REALIS() . " , ";
```

```
$requete .= " ' " . $newFilm->getCode_TYPE_FILM() . " , ";
```

```
$requete .= " ' " . $newFilm->getANNEE_FILM() . " , ";
```

```
$requete .= " ' " . $newFilm->getREF_IMAGE() . " , ";
```

```
$requete .= " ' " . $newFilm->getRESUME() . " ) ";
```

//echo \$requete; // pour test

```
VideoDAO::$cn->query($requete) or die ('Requête SQL insert invalide');
```

//facultatif, pour faire proper

```
VideoDAO::$cn=null;
```

```
}
```

NB : il serait judicieux de sécuriser tout cela en contrôlant au dernier moment la non-existence en Base de Données (en application Web, on est souvent nombreux en ligne...) et en utilisant une structure **try/catch** au lieu de la clause **or die** afin de personnaliser les éventuels messages d'erreur.

4/ Vue : Présentation

<?php // script affichage page administration site Video-club

```
function AfficheEcranAccueil(){  
    if(isset($_GET["error"]))  
        $erreur=$_GET["error"];  
    else  
        $erreur="";
```

?>

<html>

.....

<!-- contenu principal de la page -->

<h1 align="center">

Bienvenue sur le site Administration du Vidéo-Club !

</h1>

<?php

// label erreur après création nouveau film

```
if($erreur != "") {
```

```
echo "<h3 align=\"center\"><font color =\"blue\" name=\"Arial\"> ". $erreur . "</h3></font>";
```

```
}
```

?>

<!-- fin de contenu principal de la page -->

Tester et mettre au point.

XII Bilan de l'exercice

1/ Fondamentaux

L'application initiale a été entièrement restructurée sans rien changer à ses fonctionnalités. Les deux constructions sont donc tout aussi valables *fonctionnellement*.

Le découpage en couches est caractérisé par la répartition des rôles (=des préoccupations, des traitements) dans des classes ou scripts distincts.

Les principaux avantages du découpage en couches sont :

- une **meilleure lisibilité** du code par une meilleure modularisation
- d'où une **meilleure maintenabilité** (il est plus facile de pointer là où il faut modifier un code existant)
- et une **meilleure évolutivité** (une page Web peut aussi bien présenter une donnée dans un `<input type='text'.../>` que dans une boîte `<select>` ou un `<input type ='radio'... />` sans pour autant changer la logique de l'application et donc sans aucune incidence sur les scripts Contrôleur, DAO ou Métier
- une **meilleure réutilisabilité** des classes et scripts (par exemple les classes Métier ou DAO sont indépendantes de leur usage dans des pages Web particulières).

C'est bien ce qui a motivé l'élaboration de ce modèle général de conception (=design pattern) « MVC ».

2/ En PHP/MySQL

Contrôleurs

- Correspondent à l'URL appelée par l'utilisateur ou par la page Web précédente
- Code 100% PHP procédural
- 1 Contrôleur par page Web
- Appelés par les liens entre les pages (`<a href...>` ou action des form)
- Fusionnent le code des autres couches nécessaires (instruction `require` ou `require_once`)
- Code concis qui suit l'algorithme du traitement particulier à une page Web
- Appellent des fonctions/méthodes de haut niveau exposées par les autres couches
- Se terminent par la génération de l'affichage de la page Web suivante

Vues/Présentation

- Code spaghetti HTML/CSS/JavaScript/PHP
- Organisé sous forme de fonctions ou méthodes assurant la génération de (portions de) pages Web
- Les fonctions/méthodes reçoivent en paramètres les références des objets/array nécessaires pour personnaliser le code HTML/CSS/JavaScript final
- Les fonctions/méthodes sont appelées par les Contrôleurs ou d'autres fonctions/méthodes de Présentation

Métier

- Code 100% PHP
- Conception pure orientée Objet possible (constructeur, héritage, attributs accesseurs, méthodes, levée d'Exception et même collections d'objets...)
- Les objets Métier contiennent des attributs privés, accesseurs publics aux attributs privés et des méthodes privés nécessaires aux besoins de service
- Les objets Métier sont instanciés par la couche DAO ou par les Contrôleurs et sont transmis en paramètres entre les couches
- Les collections d'objets peuvent être représentées par de simples tableaux associatifs

DAO

- Code 100% PHP procédural ou objet
- Expose des fonctions/méthodes qui assurent les services de récupération de données (select...), mise à jour de données (insert... update... delete...), ou tests d'existence de données dans les bases de données
- Masquent aux autres couches les modalités d'accès aux données sur disque
- Retournent des structures de données « génériques » comme des *booléens* ou des tableaux associatifs

On gardera aussi en mémoire que :

- Encapsuler des données dans une classe nécessite toujours *plus de code* PHP que de les structurer dans un simple tableau associatif
- Le code source PHP nécessaire à une page Web est toujours *fusionné complètement* sur le serveur Web (*require*), d'où la nécessité de bien soigner la décomposition en scripts assez élémentaires pour limiter la charge serveur
- Le code source PHP complet nécessaire à une page Web est toujours *interprété* par le serveur Web, ce qui est plus consommateur de ressources que du langage compilé
- La *montée en charge* d'un serveur Web (nombre d'utilisateurs simultanés) peut être parfois spectaculaire et dans le domaine Web le 'dénî de service' ou les temps de réponse trop longs sont dramatiques pour l'image de marque et la fidélisation des usagers

Il s'agit donc toujours de faire des *choix d'architecture et d'implémentation* de chaque application Web selon ses propres caractéristiques.