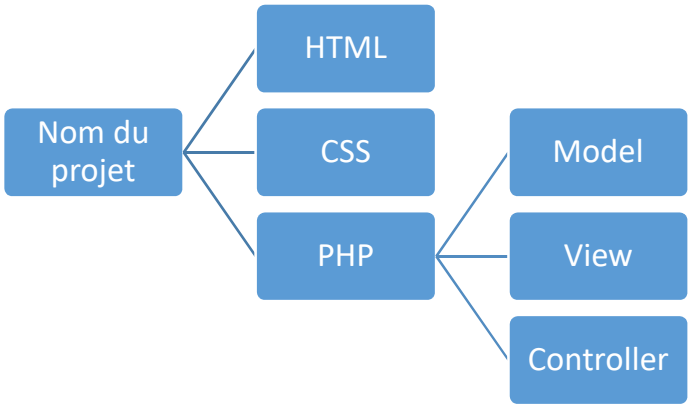
Programmation Orienté Objet
PHP –Web – Mysql

Table des matières

I. Organisation Projet

I.1. Arborescence

Pour chaque projet, on va créer une arborescence qui est standard et utilisé par tous y compris les framework.



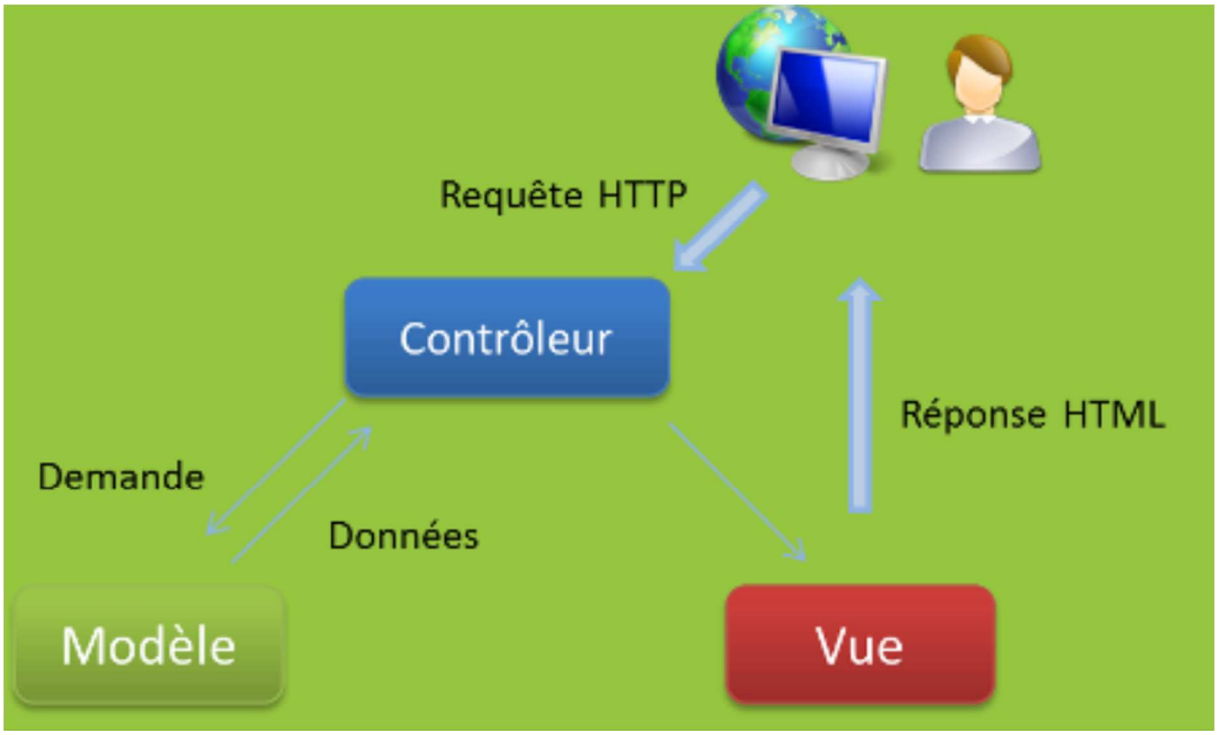
I.2. Création de Virtual Host

Afin de pouvoir ranger le code à un endroit propre et ne pas tout entasser dans le www du wamp, il est nécessaire de créer des virtuals hosts. Ainsi on peut créer un virtual host par projet.
Pour cela, dans le wampserver, dans apache, cliquer sur gestion des virtuals Hosts, ajouter le virtalhost sur le nouveau dossier.
Redémarrer les services

> Remarque : travail en réseau (à l'afpa vos droits ne sont pas assez fort pour faire cela, la configuration n'est pas à changer)
> Pour connecter un Virtual host sur un disque réseau soumis à autorisation, il faut changer les droits d'accès du service Apache.
> Services → WampApache → Connection → Ce compte → Emplacement : Tout l'annuaire, ad.afpanet.exchange.ad.afpanet + Nom : Exchange/matricule
> Renseigner le mot de passe et relancer le service

II. Le pattern MVC

Modèle Vue Contrôleur
Objectif : séparation des données, des traitements et de la présentation
Frameworks (tous langages) implémentent le MVC via la P.O.O.



Avantages

- Couches séparées (par exemple vue HTML/CSS séparée du code PHP)
- Projet mieux structuré
- Développeurs produisent un code similaire
- Equipes travaillent en parallèle sur les couches
- Développements plus rapides
- Maintenabilité, évolutions facilitées

Inconvénients

- Courbe d'apprentissage (Frameworks)
- 2 fichiers en plus !
- Migrations (évolutions des frameworks)

II.1. Le modèle

- Réceptionne les informations du contrôleur
- Données : SQL ou fichiers (XML, Json etc.)
- Connexion à la base de données
- Requêtes B.D.D
- Envoie le résultat des requêtes au contrôleur

II.2. La vue

- La sortie affichée à l'écran (HTML pour le web)
- Affiche les variables envoyées par le contrôleur
- La vue = le template (= layout)
- Moteurs de templates (Smarty, Twig)

II.3. Le contrôleur

- Lie le modèle et la vue (entre les deux)
- Gère les actions de l'application (C.R.U.D.)

- Intercepte l'action à exécuter via l'url (routing)
- Envoie des informations au modèle, les récupère
- Envoie des informations à la vue

Lisez la ressource MVC : 8 - Architecture MVC.pdf

III. Connection à la base de données

III.1. Qu'est-ce que PDO ?

PDO fournit une interface d'abstraction à l'accès aux données, ce qui signifie que vous utilisez les mêmes fonctions pour exécuter des requêtes ou récupérer les données quel que soit la base de données utilisée (MySQL, Oracle...).
PDO est une classe qui va nous permettre d'instancier un objet représentant la connexion à la base.

III.2. Connexion à un serveur MySql

Les connexions sont établies en créant des instances de la classe de base de PDO. Peu importe quel driver vous voulez utiliser ; vous utilisez toujours le nom de la classe PDO. Le constructeur accepte des paramètres pour spécifier le nom/l'adresse du serveur de bases de données (Data Source Name, en abrégé : DSN), le nom d'utilisateur et le mot de passe (s'il y en a un).

Exemple :

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=jarditou;charset=utf8', 'root', '');

```

Les paramètres requis sont les suivants :

- host : adresse du serveur hébergeant la base de données (localhost ou votre serveur web)
- dbname : nom de la base de données
- charset : jeu de caractères utilisé
- root : nom de l'utilisateur de la base de données, par exemple root
- '' : le dernier argument précise le mot de passe ; dans phpMyAdmin celui-ci n'est pas défini par défaut donc on met une chaîne vide.

L'appel au mot-clé new retourne un objet de la classe PDO que l'on stocke dans une variable ($db par exemple).

III.3. Gestion des erreurs de connexion

S'il y a des erreurs de connexion, un objet PDOException est lancé. Vous pouvez attraper cette exception si vous voulez gérer cette erreur (ou la laisser au gestionnaire global d'exception) :

```php
<?php
$base = 'jarditou';
$utilisateur = 'root';
$motdepasse = '';

try
{
   $db = new PDO('mysql:host='.$host.';charset=utf8;dbname='.$base, $utilisateur, $motdepasse);

   // Ajout d'une option PDO pour gérer les exceptions
   $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}
catch (Exception $e)
{
    echo 'Erreur : ' . $e->getMessage() . '<br />';
    echo 'N° : ' . $e->getCode();
    die('Fin du script');
}

```

Après cette connexion, la variable `$db` référence un objet connexion. Il va nous permettre d'utiliser toutes les méthodes décrites dans la classe PDO (`query()`, `prepare()`, `execute()`, etc.).

IV. Construction d'une requête SQL

IV.1. La requete

Une fois connectés à la base, nous allons en extraire l'article que nous voulons afficher.
On utilise pour cela une requête en langage SQL :

```php
$requete = "SELECT * FROM produits WHERE pro_id = 7";
$result = $db->query($requete);
$produit = $result->fetch(PDO::FETCH_OBJ);
$result->closeCursor();
```

- Ligne 1 : on a écrit une requête SQL dans une chaîne PHP.
- Ligne 2 : $db est l'objet retourné par l'appel à PDO, query() est une méthode de cet objet (c'est-à-dire, au sens programmation, une fonction de l'objet). La flèche -> permet d'accéder (appeler) une méthode ou une propriété (attribut) de l'objet.
  $db->query($requete) revient à appeler la fonction query() de l'objet $db en lui passant la requête SQL en argument. Le résultat $db->query() est stocké dans un objet $result.
- Ligne 3 : pour lire le contenu de ce résultat (qui pourrait contenir plusieurs lignes), PHP propose la méthode fetch() (= ramener). Ici, la méthode fetch(PDO::FETCH_OBJ) renvoie l'enregistrement sous la forme d'un objet (dont les propriétés contiennent les différents champs), ou FALSE s'il n'y a plus de lignes. Indirectement, cela signifie que vous ne pouvez accéder aux données que par leur nom de colonne et non par leur numéro.
  Plusieurs options de PDOStatement::fetch sont disponibles pour formater le type de retour : tableau associatif, objet, etc.
- Ligne 4 : la méthode closeCursor() sert à finir proprement une série de fetch(). En théorie, quand on exécute une requête (via query() ou execute()), puis qu'on récupère les données trouvées dans la base avec une série de fetch(), il convient de faire un closeCursor() avant de faire une autre exécution de requête (via query() ou execute()). En pratique, si on utilise MySql, ça ne sert à rien car MySql sait faire une nouvelle exécution

de requête sans qu'il ait eu de closeCursor() après la précédente exécution. Si on utilise autre chose que MySql, ou si on envisage de migrer vers autre chose que MySql un jour ou l'autre, ou si on tient à faire un code le plus portable possible, alors closeCursor() peut être utilisé, mais après chaque série de fetch().

IV.2. Affichage d'un résultat

Les données du produit sont en notre possession, il ne reste plus qu'à les afficher dans la partie <body> de notre page :

Cet affichage se déroule bien entendu dans le corps de notre page, sous la forme de trois commandes PHP :

```
<body>
 <?php echo $produit->pro_id; ?>
 <br>
 <?php echo $produit->pro_cat_id; ?>
 <br>
 <?php echo $produit->pro_ref; ?>
 <br>
 <?php echo $produit->pro_libelle; ?>
 <br>
 <?php echo $produit->pro_description; ?>
 <br>
 <?php echo $produit->pro_prix; ?>
 <br>
 <?php echo $produit->pro_stock; ?>
 <br>
 <?php echo $produit->pro_couleur; ?>
 <br>
 <?php echo $produit->pro_photo; ?>
 <br>
 <?php echo $produit->pro_d_ajout; ?>
 <br>
 <?php echo $produit->pro_d_modif; ?>
 <br>
 <?php echo $produit->pro_bloque; ?>
    </body>
  </html>
```

Nous l'avons dit plus haut, la variable $produit contient l'ensemble des informations d'un produit.
Pour isoler chacun de ces champs, il faut utiliser la construction objet->nomdelacolonne.
De cette façon, on affiche les différentes informations du produit, chacune des informations correspond à une colonne de la table.
Remarque : Il faut respecter la casse des noms de champs de la table !

IV.3. Affichage de plusieurs résultats

Le résultat d'une fonction query() est un jeu d'enregistrements. Dans l'exemple précédent, le critère sur la clé primaire pro_id permet d'être sûr que ce jeu sera constitué d'un seul enregistrement.

Mais dans le cas où plusieurs enregistrements sont remontés (par exemple avec une requête sans clause de restriction WHERE), la fonction de parcours des résultats (fetch) devra être incluse dans une boucle.

```php
while ($produit = $result->fetch(PDO::FETCH_OBJ))
{
    echo $produit->pro_id." - ".$produit->pro_libelle. "<br>";
}
```

IV.4. Une requête paramétrée

Jusqu'à présent, nous avons vu comment envoyer une requête pour remonter un enregistrement dont l'identifiant était codé en dur dans le programme.
Nous allons maintenant rendre notre page paramétrable pour afficher n'importe quel article de la table.
Il suffit pour cela de modifier la ligne de constitution de la requête

```php
$requete = "SELECT * FROM produits WHERE pro_id=".$_GET["pro_id"];
```

Nous demandons maintenant à MySQL de renvoyer le produit dont l'identifiant correspond au contenu d'une variable `$_GET["pro_id"]`.
D'où vient cette variable `$pro_id` ? Pour le moment, de l'URL, envoyée par le navigateur.
L'appel de notre script par le navigateur devra donc avoir la forme
`http://monserveur/monsite/testDb.php?pro_id=1`
Une fois notre page d'affichage de liens un peu plus esthétique, nous pouvons très bien envisager un sommaire contenant différents liens vers notre script PHP pour des valeurs de `pro_id` différentes.

IV.5. Une requête préparée

Si vous exécutez une requête une seule fois, la méthode query() convient parfaitement. Si vous avez besoin d'exécuter plusieurs fois la même requête (et c'est bien souvent le cas), il est fortement conseillé d'utiliser des requêtes préparées (méthodes: prepare() et execute()). Voici quelques exemples de leur utilisation

```php
<?php
/* Exécution de requêtes préparées */

// requête préparée avec marqueur nominatif
print "<h1>requête préparée avec marqueur nominatif</h1>";
$str_requete = "SELECT id,titre,webmaster FROM liens WHERE id>=:idMin and id<=:idMax";
// preparation de la requête
$pstmt = $db->prepare($str_requete);

// 1) Exécution de la requête en renseignant les marqueurs
print "<h2>1) Exécution de la requête en renseignant les marqueurs</h2>";
$pstmt->execute(array('idMin' => 1, 'idMax' => 15));
$lien = $pstmt->fetch(PDO::FETCH_OBJ);
print $lien->id . " -" . $lien->titre . "<br />";
$result->closeCursor();

$str_requete = "SELECT id,titre,webmaster FROM liens WHERE id>=:id and titre LIKE :titr
// preparation de la requête
$pstmt = $db->prepare($str_requete);

// 2) Exécution de la requête en liant les variables
print "<h2>2) Exécution de la requête en liant les variables</h2>";
$id = 4;
$titre = "%PHP%";
$pstmt->bindParam(':id', $id, PDO::PARAM_INT);
$pstmt->bindParam(':titre', $titre, PDO::PARAM_STR);
$pstmt->execute();
$lien = $pstmt->fetch(PDO::FETCH_OBJ);
print $lien->id . " -" . $lien->titre . "<br />";
$result->closeCursor();

// requête préparée avec marqueur interrogatif
print "<h1>requête préparée avec marqueur interrogatif</h1>";
$str_requete = "SELECT id,titre,webmaster FROM liens WHERE id>=?and id <=?";
// preparation de la requête$pstmt = $db->prepare($str_requete);

// 3) Exécution de la requête en renseignant les marqueurs
print "<h2>1) Exécution de la requête en renseignant les marqueurs</h2>";
$pstmt->execute(array(1, 15));
while ($lien = $pstmt->fetch(PDO::FETCH_OBJ))
{
    print $lien->id . " -" . $lien->titre . "<br />";
}
$result->closeCursor();

```

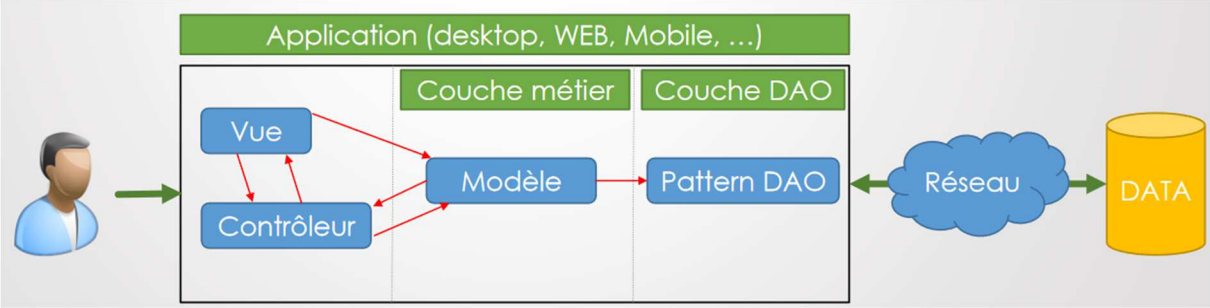
Elles sont plus rapides à s'exécuter et de plus elles protègent naturellement le code de l'injection de SQL. Même si du SQL est envoyé dans les marqueurs, la requête étant déjà compilée, il sera vu comme une chaîne de caractères.

Différence entre BindValue et BindParam

V. Le Pattern DAO

Le modèle DAO propose de regrouper les accès aux données persistantes dans des classes à part, plutôt que de les disperser. Il s'agit surtout de ne pas écrire ces accès dans les classes "métier", qui ne seront modifiées que si les règles de gestion métier changent.

L'utilisation de DAO permet de s'abstraire de la façon dont les données sont stockées au niveau des objets métier. Ainsi, le changement du mode de stockage ne remet pas en cause le reste de l'application.



V.1. Mise en place du Pattern DAO

1. Créer une classe de connexion à la base de données
2. Créer les classes métiers en rapport avec les tables de la base de données
3. Pour chaque classe nécessitant un accès aux données, créer une classe permettant de réaliser les opérations CRUD
    - Créer (Create)
    - Lire (Read)
    - Modifier (Update)
    - Supprimer (Delete)

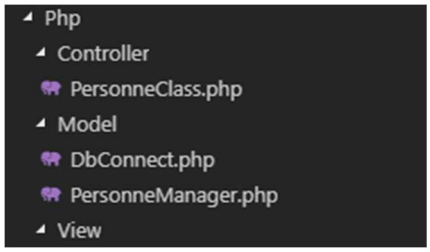
V.2. Aller plus loin

Il est possible de créer des Factory qui permettent de sélectionner une base de données ou d'accéder à différents supports comme les fichiers JSON, XML, … ou autres base de données.

La composition d'un Dal permet de standardiser l'accès aux tables en viabilisant les méthodes. Ainsi un DAL générique permet de ne plus avoir à se soucier de de l'écriture des couches basses du DAO.

V.3. Mise en place

Architecture :



- Dans Controller, on trouve l'objet métier

- Dans Model, les classes d'accès aux données. DbConnect pour la connection et PersonneManager pour le CRUD sur la table personne
- Dans View, il n'y a rien pour cette partie

PersonneClass.php contient la définition de la classe, les attributs, getters, setters et constructeurs

```php
<?php

class Personne{
    private $_id;
    private $_nom;
    private $_prenom;

    // Getter / Setter
    public function getId() {
        return $this->_id;
    }
    public function getNom() {
        return $this->_nom;
    }
    public function getPrenom() {
        return $this->_prenom;
    }
    public function setId($_id) {
        $this->_id = $_id;
    }
    public function setNom($_nom) {
        $this->_nom = $_nom;
    }
    public function setPrenom($_prenom) {
        $this->_prenom = $_prenom;
    }
    // Constructeur
    public function __construct(array $options = [])
    {
        if (!empty($options))
        {
            $this->hydrate($options);
        }
    }
    public function hydrate($data)
    {
        foreach ($data as $key => $value)
        {
            $method = 'set'.ucfirst($key);

            if (is_callable([$this, $method]))
            {
                $this->$method($value);
            }
        }
    }
}
```

DbConnect.php contient la connexion à la base de données. La méthode init sera invoqué en début de projet et la méthode getDB permettra de récupérer la connection.

```php
<?php

// Ce fichier sera inclus ? chaque fois que l'on aura besoin d'acceder ? la base de donn?es.
// Il permet d'ouvrir la connection ? la base de donn?es
class DbConnect {
    private static $db;

    public static function getDb() {
        return DbConnect::$db;
    }

    public static function init() {
        try {
            // On se connecte ? MySQL
            self::$db= new PDO ( 'mysql:host=localhost;dbname=PhpClasses;charset=utf8', 'root', '' );
        } catch ( Exception $e ) {
            // En cas d'erreur, on affiche un message et on arr?te tout
            die ( 'Erreur : ' . $e->getMessage () );
        }

    }
}
```

PersonneManager.php contient le crud de gestion de la table.
Présentation ici d'un crud avec méthode static et passage de paramètre.

```php
<?php

class PersonneManager
{

    static public function add(Personne $perso)
    {
        $db = DbConnect::getDb(); // Instance de PDO.
        // Pr?paration de la requ?te d'insertion.
        $q = $db->prepare('INSERT INTO personnes(nom, prenom) VALUES(:nom, :prenom)');

        // Assignation des valeurs pour le nom, le pr?nom.
        $q->bindValue(':nom', $perso->getNom());
        $q->bindValue(':prenom', $perso->getPrenom());

        // Ex?cution de la requ?te.
        $q->execute();

    }

    static public function delete(Personne $perso)
    {
        $db = DbConnect::getDb(); // Instance de PDO.
        // Ex?cute une requ?te de type DELETE.
        $db->exec('DELETE FROM personnes WHERE id = '.$perso->getId());
    }

    static public function get($id)
    {
        $db = DbConnect::getDb(); // Instance de PDO.
        // Ex?cute une requ?te de type SELECT avec une clause WHERE, et retourne un objet Personne.
        $id = (int) $id;

        $q = $db->query('SELECT id, nom, prenom FROM personnes WHERE id = '.$id);
        $donnees = $q->fetch(PDO::FETCH_ASSOC);

        return new Personne($donnees);
    }

```

```php
    static public function getList()
    {
        $db = DbConnect::getDb(); // Instance de PDO.
        // Retourne la liste de tous les personnes.
        $persos = [];

        $q = $db->query('SELECT id, nom, prenom FROM personnes ORDER BY nom');

        while ($donnees = $q->fetch(PDO::FETCH_ASSOC))
        {
            $persos[] = new Personne($donnees);
        }

        return $persos;
    }

    static public function update(Personne $perso)
    {
        $db = DbConnect::getDb(); // Instance de PDO.
        // Pr�pare une requ�te de type UPDATE.
        $q = $db->prepare('UPDATE personnes SET nom=:nom, prenom=:prenom WHERE id = :id');

        // Assignation des valeurs � la requ�te.
        $q->bindValue(':nom', $perso->getNom());
        $q->bindValue(':prenom', $perso->getPrenom());
        $q->bindValue(':id', $perso->getId());

        // Ex�cution de la requ�te.
        $q->execute();
    }


}
```

VI. La gestion du CRUD

Schéma complet de l'application