

PHP – MYSQL - MVC

<https://openclassrooms.com/courses/concevez-votre-site-web-avec-php-et-mysql/presentation-des-bases-de-donnees-2>
<http://php.net/manual/fr/langref.php>
Open Class Room

SOMMAIRE

Sommaire.....	1
PHP - MySQL	4
Installation des fichiers de test.....	4
Les exemples sont présentés dans un chapitre en vert.....	4
Les exercices à faire sont présentés dans un chapitre en jaune.....	4
Utilisation de PHPMyAdmin	4
Ouvrir phpMyAdmin.....	4
Créer une Base de données.....	4
Créer une table et les tuples.....	4
Mettre à jour la colonne des BD à gauche : onglet flèche circulaire verte.....	5
Affichez le contenu de la table utilisateur	5
Exécuter un SELECT	5
Exporter	5
Importer	5
Connexion à la BD et print_r du contenu d'une table.....	7
Utilisation de la base de donnée en PHP	7
Connexion à la BD : new PDO (exemple 1 – connexion).....	7
Code	7
Explications	7
Afficher le contenu d'une table -1 : query(), fetch(), print_r (exemple 2 – select).....	8
Exemple	8
Explications	9
Résultats	9
Chargez les exemples 1 et 2 et testez-les.....	10
Vocabulaire de Programmation orientée objet.....	10
Classe	10
Objet.....	10
Méthode	10
Exceptions.....	10
Guides de style	10
Gestion des requêtes.....	11
Technique de programmation – PDO et PDOStatement - \$bdd, \$requete, \$reqSQL.....	11
Terminologie : \$bdd - \$reqSQL - \$requete - \$ligne	11
\$bdd (PDO) – query - prepare	11
\$requete (PDOStatement) - execute - fetch - closeCurseur	11
Synthèse.....	11
Accès aux champs, gestion erreurs, order by, like, limit (exemple 3 – Select Where).....	12
Chargez la BD pour pouvoir tester les codes	12
Accéder aux champs	12
Select attribut1, attribut2, Where, Order by	12
Select ... like	12
Select ... Order by, limit.....	12
Requêtes variables : where realisateur = ? (exemple 4 – Select Variable).....	13

Solution basique à éviter : risque XSS	13
Solution avec requête préparée : where ? , prepare et execute	13
Remplacer les ? par des :alias	13
Déboguer : or die bdd->errorInfo()	14
Créer le \$bdd avec la gestion des erreurs (déjà vu)	14
Aternative : exécuter la requête (query ou execute) « or die ».....	14
Ajouter, modifier, supprimer des données dans une table (exemple 5 – insert-update-delete)	15
Via phpMyAdmin – Afficher	15
Via phpMyAdmin – SQL	15
Via php - INSERT	15
Via php – DELETE	15
Via php – UPDATE	16
Bons usages.....	16
TP 1 – Insert Update Delete et consultation	16
TP Site Artiste – non MVC	17
Exemple 6 : Etape 1 : uniquement les œuvres – TP 2 : Installez, testez et regardez bien le code	17
Contenu du dossier	17
Installation	17
Tester le code.....	17
TP3 : Etape 2 : œuvres, exposition et œuvres exposées – gestion basique dans le HTML	18
1) Quand on choisit « œuvres », on obtient ceci qui est la page d'accueil :	18
2) Quand on choisit « exposition » on obtient ceci :.....	18
3) Quand on clique sur une exposition, on obtient la liste des œuvres exposées :	19
TP 4 : Etape 3 – comme l'étape 2 mais avec séparation du PHP et du HTML	20
Problématique d'organisation de base : quels fichiers pour mes projets	21
Questions à se poser :	21
Schéma de synthèse à réaliser.....	21
MVC	22
Problématique : quels fichiers, quels dossiers pour mes projets	22
Organisation non-MVC	22
Principes d'organisation non-MVC	22
Défauts de l'organisation non-MVC	22
Présentation du MVC	23
MVC : Modèle - Vue - Contrôleur.	23
Le Modèle (SQL)	23
La Vue (HTML)	23
Le Contrôleur (PHP)	24
Fonctionnement global.....	24
MVC : design pattern tête la première	25
Organisation des répertoires et des fichiers dans le MVC	26
Organisation de répertoires MVC.....	26
Les fichiers « modèle »	26
Les fichiers « contrôleur »	26
Les fichiers « vue ».....	26
Le font contrôleur = contrôleur global = routeur = indexSwitch	27
Le problème.....	27
La solution : un « front contrôleur ».....	27
Le fichier indexSwitch.html : le routeur	27
Principes.....	27
Exemple	27
empty() ou isset().	28
Include entête et pied de page	28
Déconnexion	28
Debug.....	28
Variante sans switch : beaucoup plus courte !	28

Le fichier index.html	29
MVC – Exemple 7 : le Site Artiste V1-MVC - affichage des œuvres - Testez l'exemple.....	30
Organisation des dossiers et des fichiers.....	30
TP Site Artiste MVC	31
TP-MVC-V2 : Site Artiste V2 – MVC : affichage de toutes les tables.....	31
TP-MVC-V3 : Site Artiste V3 – MVC : administration des œuvres	31
La page d'administration des œuvres pourra ressembler à ceci :	31
Connexion comme administrateur	32
Vérifier le login.....	32
TP-MVC-V4 : Site Artiste V4 – MVC : administration des des expositions.....	34

Edition : mars 2019

PHP - MYSQL

PHP-MySQL :

<https://openclassrooms.com/courses/concevez-votre-site-web-avec-php-et-mysql/presentation-des-bases-de-donnees-2>

Manuel de Référence du PHP :

<http://php.net/manual/fr/langref.php>

Manuel de Référence du SQL :

<http://www.w3schools.com/sql/>

Installation des fichiers de test

Les exemples du cours sont dans un fichier zip fournis avec l'article du cours : 00-Exemples-PHP-MySQL.zip

Chargez ce fichier et mettez-le dans le dossier Partie_4 du répertoire web « www » du serveur WAMP.

Les exemples sont présentés dans un chapitre en vert.

Les exercices à faire sont présentés dans un chapitre en jaune.

Utilisation de PHPMyAdmin

Ouvrir phpMyAdmin

Icône Wamp / bouton droit / phpMyAdmin

Créer une Base de données

Onglet Base de données

Dans le champ « nom de base de donnée » saisissez : « BD_Utilisateur »

On vient de créer la BD_Utilisateur. Elle apparaît dans le navigateur à gauche.

Créer une table et les tuples

Allez dans l'onglet SQL.

On va créer la table suivante avec les tuples associés (fichier BD_Utilisateur.sql de l'exercice 1).

```
CREATE TABLE Utilisateur (
    id int(11) primary key AUTO_INCREMENT,
    prenomNom varchar(20) NOT NULL,
    adMail varchar(20) NOT NULL,
    motDePasse varchar(20) NOT NULL,
```

```

annee int(4) NOT NULL
) ENGINE=InnoDB;

Insert into Utilisateur values (NULL, 'Sia PEI',
'ji@gmail.com','jipei', 1995);
Insert into Utilisateur values (NULL, 'Yawei CAI',
'jawei@yahoo.com','yaweicai',1996);
Insert into Utilisateur values (NULL, 'Zikeng PENG',
'zikeng@china.com','zikeng',1994);
Insert into Utilisateur values (NULL, 'Jiawen LI',
'jiawen@orange.fr','jiawen',1995);
Insert into Utilisateur values (NULL, 'Xiaoyu LIU',
'xiowyu@gmail.fr','liu',1996);
Insert into Utilisateur values (NULL, 'Olivier
TRAN','tran@gmailcom','olivier',1997);

```

Collez le code dans la zone de saisie SQL et exécuter.

La table est créée avec les tuples.

Mettre à jour la colonne des BD à gauche : onglet flèche circulaire verte

Cliquez sur la flèche circulaire verte pour mettre à jour la colonne des BD.

Vous pouvez ouvrir la BD « BD_Utilisateur ».

Vous voyez apparaître la table « Utilisateur ».

Affichez le contenu de la table utilisateur

En cliquant sur la table dans la colonne des BD, on voit apparaître la liste des tuples.

Exécuter un SELECT

En allant dans l'onglet SQL, on peut saisir une requête. Par exemple :

```
SELECT * FROM `utilisateur` WHERE id = 3
```

Exporter

On peut exporter la BD dans différents formats :

SQL : Ca génère un script SQL qui permet de reconstituer la BD à l'identique, donc une sauvegarde.
Exportez la BD générée et regardez le script.

CSV : Ca génère un fichier texte qu'on pourra importer sur Excel.
Exportez la BD en format CSV et importez le fichier CSV dans Excel.

Importer

On commence par créer une BD. Ensuite, on peut importer le contenu de la BD dans différents formats :

SQL : on choisit le fichier à importer. C'est un fichier avec du code SQL (du DDL et du DML : des CREATE TABLE et des INSERT INTO). Par exemple, c'est un fichier exporté de sauvegarde.

CSV : on peut importer un fichier Excel qui aura été enregistré au format CSV. Il faut préciser le séparateur de colonne (plutôt un « ; » en CSV) et préciser si les colonnes ont une première ligne avec le nom de colonne. On peut dire de ne pas s'arrêter en cas d'erreur d'INSERT. Il faudra ensuite ajouter des contraintes d'intégrité dans la table. Particulièrement mettre les clés primaires et étrangères et donnez un nom à la table.

Connexion à la BD et print_r du contenu d'une table

Utilisation de la base de donnée en PHP

3 jeux de fonctions (API) permettent de se connecter à la BD et de l'utiliser : mysql, mysqli et PDO :
<http://php.net/manual/fr/mysqlinfo.api.choosing.php>

Le jeu mysql est le plus ancien : mieux vaut l'éviter.

On peut utiliser mysqli ou PDO (PHP Data Object), et particulièrement PDO_MYSQL.

L'intérêt du PDO est que c'est une interface d'abstraction permettant l'utilisation de n'importe quelle BD. De plus elle est « orienté objet ».

On utilisera plutôt PDO_MYSQL

<http://php.net/manual/en/ref pdo-mysql.php>

Connexion à la BD : new PDO (exemple 1 – connexion)

Code

```
<?php
function connexionBD($dbname) {
    // paramètres de la base de donnée
    $sgbdname='mysql';
    $host='localhost';
    $charset='utf8';
    // dsn : data source name
    $dsn = $sgbdname . ':host='.$host . ';dbname='.$dbname
        . ';charset='.$charset;
    // utilisateur connecté à la base de donnée
    $username = 'root';
    $password = 'root';

    // pour avoir des erreurs SQL plus claires
    $erreur = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);

    try {
        // connexion à la BD : new PDO
        $bdd = new PDO($dsn, $username, $password, $erreur);
        echo '<p>Connexion réussie</p>';
        return $bdd;
    } catch (PDOException $e) {
        echo 'Connexion échouée : ' . $e->getMessage();
        return NULL;
        // die ('Connexion échouée : ' . $e->getMessage() );
    }
}
?>
```

Explications

- **On fait un new PDO avec 4 paramètres**

PDO est une classe. On crée un nouvel objet de la classe qu'on appelle \$bdd.

<http://php.net/manual/fr/pdo.construct.php>

Le new PDO à 4 paramètres :

- \$dsn (data source name) : contient des infos sur le SGBD (mysql), le serveur (host, ici : localhost), le nom de la BD, le jeu de caractères utilisé (UTF8 pour que ce soit le plus générique).
- \$username : nom de l'utilisateur qui se connecte à la BD.
- \$password : password de l'utilisateur qui se connecte à la BD.
- \$erreur : pour gérer les messages d'erreur.

➤ ***On utilise des variables pour rendre le code générique***

On utilise des variables pour rendre le code plus générique :

- \$host : la machine du serveur de SGBD,
- \$sgbdname : le type de SGBD,
- \$username : le nom de l'utilisateur qui se connecte sur la BD,
- \$password : le mot de passe de cet utilisateur,
- \$dbname : le nom de la BD à laquelle on accède sur le SGBD.

➤ ***\$erreur : gestion des erreurs***

```
$erreur = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
```

En ajoutant le paramètre \$erreur tel qu'il est là dans le new PDO, on aura des messages d'erreurs du SGBD, par exemple si le SELECT est mal écrit.

Syntaxe :

PDO ::ATTR_ERRMODE et PDO::ERRMODE_EXCEPTION sont des constantes de classe définies dans la classe PDO.

On a la syntaxe : array (clé => valeur) qui permet de définir un couple clé-valeur dans le tableau.

PDO ::ATTR_ERRMODE : <http://php.net/manual/fr/pdo.constants.php>

PDO::ERRMODE_EXCEPTION : <http://php.net/manual/fr/pdo.error-handling.php>

➤ ***Gestion des erreurs avec try catch***

La structure « try » « catch » est une structure de programmation objet pour gérer les erreurs.

Le die permet d'arrêter proprement l'exécution de la page en cas d'erreur.

Afficher le contenu d'une table -1 : query(), fetch(), print_r (exemple 2 – select)

Exemple

```
// on écrit la requête
$reqSQL='SELECT * FROM utilisateur';

// on récupère le résultat
$requete=$bdd->query($reqSQL);
echo '<pre>'; print_r($requete); echo '</pre>';

// on affiche le résultat ligne par ligne
while($ligne=$requete->fetch()){
    echo '<pre>'; print_r($ligne); echo '</pre>';
    echo '<p> nombre d\'éléments de $ligne : ' .sizeof($ligne) .
'</p>';
}

// on libère les tables de la requête
$requete->closeCursor(); // pour finir le traitement
```

Explications

- On écrit le select dans \$reqSQL
- On utilise la méthode (fonction) query de \$bdd (->) en passant la \$reqSQL en paramètre.
- Le résultat est dans \$requete : c'est un objet complexe sur lequel on peut appliquer des méthodes (des fonctions) qui contient le tableau de données du select.
- On affiche ce qu'on peut de la réponse : en l'occurrence seule la requête s'affiche.
- On passe les lignes de la réponse en revue avec la méthode fetch().
- Chaque \$ligne retourné par le fetch est un tableau associatif : on peut faire un print_r de \$ligne.
- On constate alors qu'on a deux accès possible à chaque donnée : par le nom du champs (id par exemple) ou par un numéro (0 pour id) : il y a donc 2 fois plus d'éléments que prévu dans chaque ligne
- Quand on a fini de travailler, on fait un closeCursor, pour libérer les tables de la requête

Résultats

```
Connexion réussie
DEBUT
< pre > print_r($requete); < /pre >

PDOStatement Object
(
    [queryString] => SELECT * FROM utilisateur
)
while ($ligne=$requete->fetch()) {

    Array
    (
        [id] => 1
        [0] => 1
        [prenomNom] => Sia PEI
        [1] => Sia PEI
        [adMail] => ji@gmail.com
        [2] => ji@gmail.com
        [motDePasse] => jipei
        [3] => jipei
        [annee] => 1995
        [4] => 1995
    )
    nombre d'éléments de $ligne : 10

    Array
    (
        [id] => 2
        [0] => 2
        [prenomNom] => Yawei CAI
        [1] => Yawei CAI
        [adMail] => jawei@yahoo.com
        [2] => jawei@yahoo.com
        [motDePasse] => yaweicai
        [3] => yaweicai
        [annee] => 1996
        [4] => 1996
    )
    nombre d'éléments de $ligne : 10
```

Chargez les exemples 1 et 2 et testez-les.

Vocabulaire de Programmation orientée objet

Classe

Une classe, c'est un type, comme un entier, un réel, un caractère, une string ou un booléen.
En général, une classe correspond à l'équivalent d'un tableau associatif : elle contient plus couples de clé-valeur. Les différentes clés sont appelées « attribut ».
En plus, on associe des fonctions à une classe : on les appelle alors « méthode ».

Objet

Un objet c'est une variable de type Classe.
Quand on crée un objet avec des valeurs pour les couples clé-valeur (pour les attributs), on dit qu'on instancie un objet. Ca passe par la commande « new ».

Méthode

Les méthodes sont des fonctions qui sont attachées à une classe.
Elle ne sont utilisables que par les objets de la classe.
On écrit : objet->methode() pour appeler la méthode pour l'objet en question : c'est comme si on avait passé l'objet en paramètre de la méthode.

Exceptions

En cas d'erreur, en programmation objet on passe par des objets de classe Exception.
Ca se fait avec un « try » « catch »
try : on essaie d'exécuter une suite d'instruction
catch : si la suite d'instructions exécutée à générer une erreur sous la forme d'une exception, on passe dans le bloc catch
Le catch précise le nom de l'objet exception qu'on va traiter.
On peut alors accéder à des informations par la méthode getMessage() par exemple.

Guides de style

<https://fr.wikipedia.org/wiki/CamelCase>
https://eilgin.github.io/php-the-right-way/#code_style_guide

Gestion des requêtes

Technique de programmation – PDO et PDOStatement - \$bdd, \$requete, \$reqSQL

Pour manipuler la BD, on utilise principalement deux classes : PDO et PDOStatement

Terminologie : \$bdd - \$reqSQL - \$requete - \$ligne

- **\$bdd** : un objet de la classe PDO sera appelé \$bdd. C'est en quelque sorte l'objet qui permet l'accès concret à la base de données, pour un utilisateur et une base de donnée.
- **\$reqSQL** : le texte de la requête sera mis dans un \$reqSQL. C'est une simple chaîne de caractères. Il ne doit pas être confondu avec le résultat de la requête : \$requete.
- **\$requete** : un objet de la classe PDOStatement sera appelé \$requete (statement peut vouloir dire « requête »). Cette \$requete est un objet complexe qui contient à la fois le \$reqSQL et le résultat de la requête une fois celle-ci exécutée.
- **\$ligne** : le résultat d'un fetch() est une ligne : \$ligne = \$requete->fetch(). C'est un tuple de la table résultante de la requête SQL.

\$bdd (PDO) – query - prepare

<http://php.net/manual/fr/class.pdo.php>

PDO ne contient que des méthodes (il ne contient pas d'attributs). Notons particulièrement :

- **query** : renvoie un \$requete auquel est associé le \$reqSQL passé en paramètre et le résultat de la requête (le résultat du Select) prêt à être fetché (prêt à être parcouru).
- **prepare** : renvoie un \$requete auquel est associé le \$reqSQL passé en paramètre. La requête n'a pas été exécutée. Elle peut contenir des variables.

PDO contient aussi des méthodes propres à une BD comme la gestion des transactions : commit, rollback, etc., et d'autres choses.

\$requete (PDOStatement) - execute - fetch - closeCursor

<http://php.net/manual/fr/class.pdostatement.php>

PDOStatement contient un attribut : la valeur du \$reqSQL fourni en paramètre quand il a été créé. Il contient aussi des méthodes. Notons particulièrement :

- **execute** : permet d'exécuter une requête avec des variables. Il faut fournir en paramètre un tableau de valeurs pour les variables.
- **fetch** : permet de récupérer les lignes du résultat de la requête, une par une.
- **fetchAll** : permet de récupérer toutes les lignes du résultat de la requête, toutes dans un tableau.
- **closeCursor** : permet de refaire un execute.

Synthèse

\$bdd : PDO	\$requete : PDOStatement
-> query(\$reqSQL) : PDOStatement	-> fetch() : ligne
-> prepare(\$reqSQL) : PDOStatement	-> execute() : bool
	-> closeCursor() : bool
	-> fetchAll() : toutes les lignes

Accès aux champs, gestion erreurs, order by, like, limit (exemple 3 – Select Where)

Chargez la BD pour pouvoir tester les codes

Pour tester les codes, il faut charger la BD en plus d'installer le code dans le répertoire du serveur WEB.

Accéder aux champs

\$ligne est un tableau associatif. On peut donc écrire :

```
echo '<p><strong> prenom nom </strong></p>';
while($ligne=$requete->fetch()){
    echo '<p>' . $ligne[prenom] . ' ' . $ligne[nom] . '</p>';
}
```

Select attribut1, attribut2, Where, Order by

```
$reqSQL='
    Select realisateur, titre, annee from films
    where realisateur = \'King Vidor\'
    order by annee
';
// on precise 3 champs : realisateur, titre et annee
// attention au \
// on trie par annee
```

Select ... like

```
$reqSQL='
    SELECT * FROM films
    WHERE realisateur like \'%manki%\'
    order by realisateur, annee
';
// like % manki % : n\'importe quoi autour de manki
// order by realisateur, annee : plusieurs realisateurs possibles
// dans le resultat : j'ordonne le resultat
```

Select ... Order by, limit

```
$reqSQL='
    Select realisateur, titre, annee from films
    where annee = 1960
    order by annee
    limit 0, 10
';
// on prend les 10 premiers (de 1 à 10)
// limit 10, 10 pour les 10 suivants
// limite 20, 10 pour les 10 suivants, etc.
```

Requêtes variables : where réalisateur = ? (exemple 4 – Select Variable)

Objectif : mettre une variable dans une requête (par exemple, une information saisie par l'utilisateur)

Solution basique à éviter : risque XSS

On pourrait mettre un `$_GET` dans la `$reqSQL` :

```
$reqSQL='Select... where auteur=\'' . $_GET['realisateur'] . '\'');
```

A éviter !!! si le `$_GET` contient 'toto \' or \a\'=\\'a, le select renverra toute la table !

Solution avec requête préparée : where ?, prepare et execute

```
$reqSQL='Select... where réalisateur = ?;

$requete=$bdd->prepare($reqSQL)

$requete->execute(array(
    $_GET['réalisateur']
));
```

On sépare les arguments par des « , » dans le array

Remplacer les ? par des :alias

Cette solution est la plus lisible et celle qu'on va privilégier.

```
$reqSQL='Select... where réalisateur= :réalisateur;

$requete=$bdd->prepare($reqSQL)

$requete->execute(array(
    'réalisateur'=> $_GET['réalisateur']
));
```

Déboguer : or die bdd->errorInfo()

Créer le \$bdd avec la gestion des erreurs (déjà vu)

Pour afficher les détails d'une erreur, on crée un \$bdd avec la gestion des erreurs :

```
$erreur = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);  
$bdd = new PDO($dsn, $username, $password, $erreur);
```

C'est suffisant pour obtenir des messages d'erreur propres.

Alternative : exécuter la requête (query ou execute) « or die »

Si on n'utilise pas un \$erreur dans le new PDO, on peut ajouter un « or die » au query ou au execute pour afficher les détails d'une erreur. Mais c

```
$requete=$bdd->query($reqSQL) or die(print_r($bdd->errorInfo()));
```

ou

```
$requete->execute(array(...)) or die(print_r($bdd->errorInfo()));
```

Le « or die » est inutile avec une connexion PDO en ERRMODE. Le « or die » s'utilise quand on travaille avec le jeu « mysql » de fonctions et la fonction mysql_connect(...).

<http://php.net/manual/fr/pdo.errorinfo.php>

➤ *Les limites du « or die » : confidentialité et user friendly*

Le « or die » peut être pratique en phase de développement.

En production, il met au jour des informations qui peuvent être confidentielles et n'est pas user-friendly.

<http://www.alsacreations.com/tuto/lire/676-gestion-erreurs-mysql-php-or-die.html>

Ajouter, modifier, supprimer des données dans une table (exemple 5 – insert-update-delete)

Via phpMyAdmin – Afficher

Interface graphique

Via phpMyAdmin – SQL

On peut entrer les commandes SQL : INSERT, UPDATE et DELETE.

Le système propose un pré-remplissage des commandes.

Via php - INSERT

On peut entrer les commandes SQL : INSERT, UPDATE et DELETE.

```
$reqSQL='
    INSERT INTO films
        (titre, realisateur, annee) VALUES
        (:titre, :realisateur, :annee)
    ';

//exemple('L\'homme irrationnel','Woody Allen','2015');

$requete=$bdd->prepare($reqSQL)

$resultat=$req-> execute(array(
    'titre'=>$_GET['titre'],
    'realisateur'=>$_GET['realisateur'],
    'annee'=>$_GET['annee']
)); // or die(print_r($bdd->errorInfo())); ;

/* le or die est inutile avec la connexion en ERRMODE */
/* $resultat pour traiter les erreurs proprement, sans ERRMODE */
```

Via php – DELETE

```
$reqSQL='
    DELETE FROM films
    WHERE titre = :titre AND realisateur = :realisateur'
    ;

$requete=$bdd->prepare($reqSQL);
$resultat=$requete->execute(array(
    'titre'=>$_GET['titre'],
    'realisateur'=>$_GET['realisateur']
));

/* pour tester le résultat : 0 si pas de DELETE */
if($requete->rowCount() ){ // rowCount compte le nombre de delete
    echo '<br/>DELETE effectué ' . $requete->rowCount() . ' fois';
}
else {
    echo '<br/> Le DELETE a échoué';
}
```

rowCount permet de savoir combien de delete on été effectués.

<http://www.astuces-webmaster.ch/page/mysql-pdo>

➤ ***ATTENTION au DELETE !!***

Attention au delete : quand une donnée est supprimée, on ne peut pas la récupérer si on est en mode validation (autocommit) ce qui est le plus fréquent !

Il faut donc faire des vérifications, par exemple :

```
if (!isset($_GET['realisateur']) or !isset($_GET['titre'])
    or $_GET['realisateur']=='' or $_GET['titre']=='')
{
    echo '<br/> Vous n\'avez pas saisi tous les paramètres';
}
```

Via php – UPDATE

```
$reqSQL='
    UPDATE films
    SET duree=:duree
    WHERE titre = :titre AND realisateur = :realisateur
    ';
```

➤ ***ATTENTION à l'UPDATE!!***

Attention à l'UPDATE : quand une donnée est modifiée, on ne peut pas la récupérer si on est en mode validation (autocommit) ce qui est le plus fréquent !

Bons usages

A la place de :

```
'titre'=>$ GET['titre']
```

on aura

```
'titre'=>$titre
```

Les variables \$titre, \$realisateur, etc. seront récupérées via un \$_POST ou un \$_GET.

TP 1 – Insert Update Delete et consultation

On travaille avec la base CINEMA.

1. Changez volontairement les paramètres de connexion un par un pour voir le message d'erreur correspondant.
2. Ajoutez un film. Vérifiez qu'il est bien présent dans la BD en utilisant un script de consultation.
3. Supprimer le film que vous venez d'ajouter. Vérifiez qu'il est bien absent dans la BD en utilisant un script de consultation.
4. Rajoutez un film. Vérifiez qu'il est bien présent dans la BD en utilisant un script de consultation.
5. Modifiez l'année du film que vous venez de rajouter. Vérifiez que la modification a été faite dans la BD en utilisant un script de consultation.

TP Site Artiste – non MVC

Exemple 6 : Etape 1 : uniquement les œuvres – TP 2 : Installez, testez et regardez bien le code

Contenu du dossier

Le dossier de l'exemple 6 contient :

Le code de la BD : BD_ARTISTE.sql et un dossier d'images.

Le fichier de connexion à la BD.

Une page pour l'affichage des œuvres.

Installation

Installez la BD sur WAMP.

Parcourez le contenu de la BD pour bien la comprendre.

Regardez la page d'affichage des œuvres : indexOeuvres.php

Tester le code

On obtient l'affichage suivant :

DEBUT

[1] : titre1 - 2016 - 500 euros

huile sur toile : 40cm X 50cm



[2] : titre2 - 2016 - 500 euros

acrylique sur toile : 30cm X 60cm



TP3 : Etape 2 : œuvres, exposition et œuvres exposées – gestion basique dans le HTML

Ensuite on veut pouvoir afficher aussi les expositions et les œuvres exposées.

On considère que « Site Artiste » et le « nav » sont dans un header HTML.

1) Quand on choisit « œuvres », on obtient ceci qui est la page d'accueil :

Site ARTISTE

- [œuvres](#)
- [expositions](#)

[1] : titre1 - 2016 - 500 euros

huile sur toile : 40cm X 50cm



2) Quand on choisit « exposition » on obtient ceci :

Site ARTISTE

- [œuvres](#)
- [expositions](#)

[\[1\] : expo1 - centre 1 - rue machin](#)

Date de début : 2016-06-10. Date de fin : 2016-06-24. Date du vernissage : 2016-06-10 18:00:00

[\[2\] : expo2 - centre 2 - rue chose](#)

Date de début : 2017-09-03. Date de fin : 2016-09-24. Date du vernissage : 2016-09-05 19:00:00

[\[3\] : expo3 - centre 3 - rue true](#)

Date de début : 2018-02-25. Date de fin : 2018-03-10. Date du vernissage : 2016-02-10 18:00:00

3) Quand on clique sur une exposition, on obtient la liste des œuvres exposées :

Site ARTISTE

- [œuvres](#)
- [expositions](#)

Exposition n°[2] : expo2 - centre 2 - rue chose

Date de début : 2017-09-03. Date de fin : 2016-09-24. Date du vernissage : 2016-09-05 19:00:00

Oeuvre n°[2] : titre2 - 2016 - 550 euros

acrylique sur toile : 30cm X 60cm



TP 4 : Etape 3 – comme l’étape 2 mais avec séparation du PHP et du HTML

On va maintenant organiser le code en séparant au maximum le code PHP et le code HTML.

Pour ça, on va utiliser la fonction **fetchAll()** :

```
$lesOeuvres=$requete->fetchAll();
```

Ainsi, on récupère le tableau complet des œuvres issu du Select.

On peut ensuite parcourir ce tableau dans le PHP avec un classique « foreach » :

```
foreach($lesOeuvres as $ligne){  
    echo '<h3>' . $ligne['id']. ' : ' . $ligne['titre']. ' - '  
        . $ligne['annee']. ' - ' . $ligne['prix'].' euros</h3>';  
  
    etc.  
}
```

Problématique d'organisation de base : quels fichiers pour mes projets

Le but est d'afficher les œuvres, les expositions et les œuvres exposées.

Questions à se poser :

Quels fichier dois-je créer ?

Il y aura 1 fichier par page.

Comment circule-t-on de page en page ?

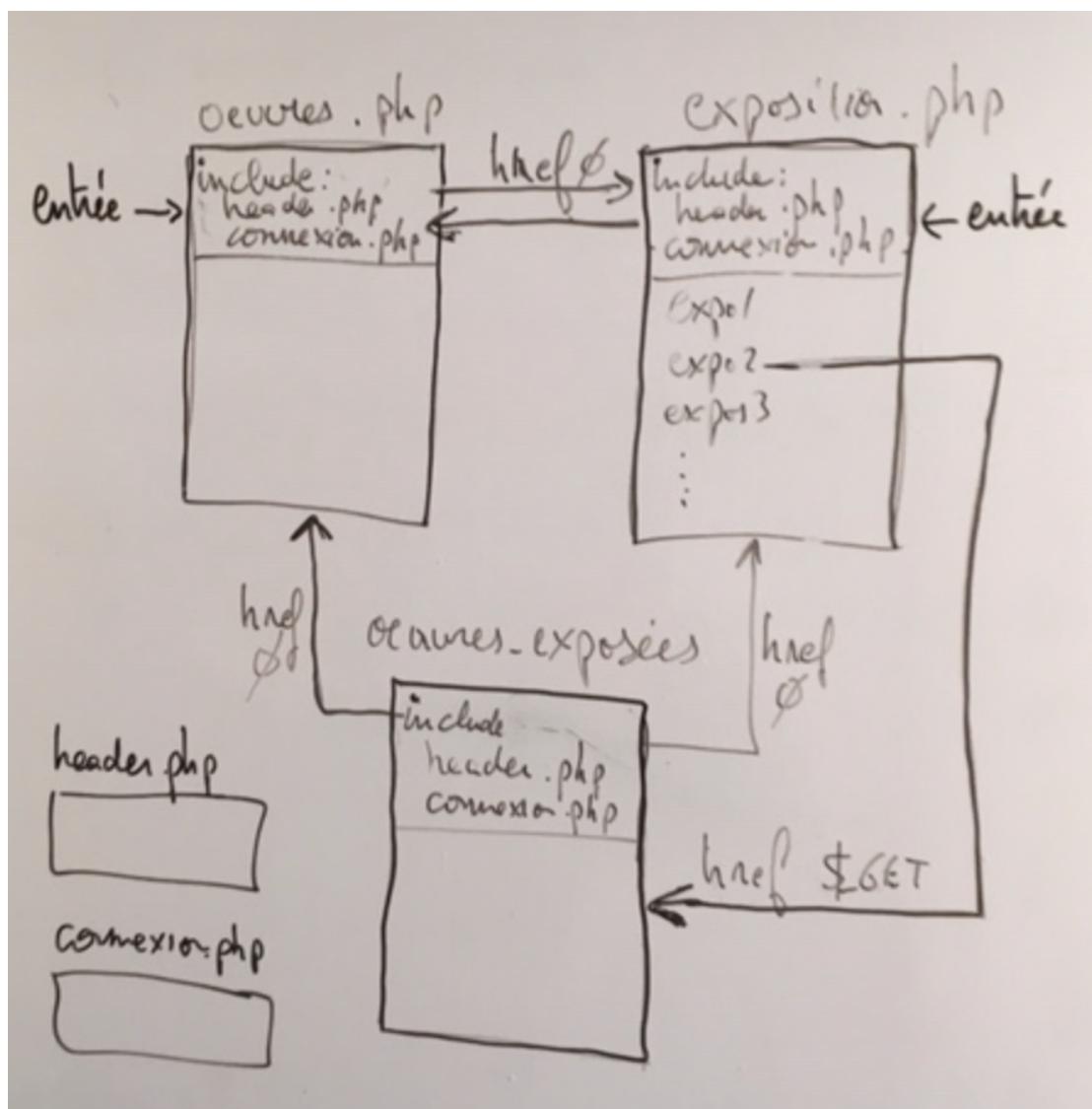
Y a-t-il des éléments commun côté HTML ?

Côté HTML, le header sera partagé.

Y a-t-il de éléments commun côté PHP ?

Côté PHP, la connexion à la BD sera partagée.

Schéma de synthèse à réaliser



MVC

<https://openclassrooms.com/courses/votre-site-php-presque-complet-architecture-mvc-et-bonnes-pratiques/avant-propos-comment-fonctionne-ce-tutoriel>
<https://openclassrooms.com/courses/adopter-un-style-de-programmation-clair-avec-le-modele-mvc>

Problématique : quels fichiers, quels dossiers pour mes projets

Le MVC s'intéresse à la problématique d'organisation des fichiers et des répertoires.

A noter particulièrement les problèmes suivants :

- Quels dossiers dois-je créer ?
- Comment dois-je organiser mes fichiers ?
- Comment passe-t-on d'un fichier à un autre ?
- Quels sont les inclusions à envisager ?
- Y a-t-il besoin d'un dossier admin ?

Organisation non-MVC

Principes d'organisation non-MVC

On va avoir une page php par page de site.

Une page c'est comme un « main » qui reçoit des paramètres (`$_GET`, `$_POST`). La liste des paramètres est variable selon l'entrée (un href, un autre href, un formulaire de saisie, un autre, un header). Il y a aussi des variables globales partagées par toutes les pages dans `$_SESSION`.

La page réagit différemment en fonction des paramètres qu'elle reçoit.

Selon les cas, elle fera en totalité ou en partie :

- Récupérer les contenus de `$_GET`, `$_POST`, `$_SESSION`
- Mettre à jour `$_SESSION`
- Inclure des fichiers : header, footer, connexion à la BD, etc.
- Gérer les données de la BD : `INSERT`, `UPDATE`, `DELETE` et des `SELECT` associés si nécessaire.
- La construction de la page HTML à afficher (et les `SELECT` associés si nécessaire)

Défauts de l'organisation non-MVC

- Tout est mélangé : le SQL, le PHP et le HTML, surtout si on ne sépare pas PHP et HTML.
- Les pages peuvent devenir très grosses.
- La maintenance n'est pas facile.
- Travail à plusieurs est rendu difficile.

Présentation du MVC

MVC : Modèle - Vue - Contrôleur.

L'architecture MVC sépare la logique du code en trois parties, trois ensembles de fichiers :

- le modèle (qui correspond au SQL)
- la vue (qui correspond au HTML)
- le contrôleur (qui correspond au PHP faisant le lien entre les deux précédent).

Cela rend le code plus facile à mettre à jour et permet d'organiser le travail en 3 parties et donc de travailler en parallèle.

L'architecture MVC est une bonne pratique de programmation.

La connaissance de l'architecture MVC rend capable de créer un site web de qualité et facile à maintenir.

En pratique, les architectures MVC mises en œuvre s'appuient sur la théorie mais l'adaptent de façon pragmatique. Il y a donc plusieurs façons de mettre en œuvre le MVC.

Les principaux framework sont développés en MVC : CodeIgniter, CakePHP, Symfony, Jelix, Zend Framework, etc.

Le Modèle (SQL)

Le modèle gère les données du site. Essentiellement les accès à la BD. Mais aussi la gestion de fichiers. Il propose des fonctions pour faire des Insert, des Update, des Delete, des Select. Ces fonctions peuvent renvoyer des tableaux de données. Les résultats seront exploités par le contrôleur mais aussi par le HTML.

C'est une page pur PHP.

L'idée générale est que dans une application, la base de données est centrale.

Si la BD est bien conçue, l'application sera facile à maintenir et à faire évoluer.

Si la BD est mal conçue, l'application sera complexe à maintenir.

La Vue (HTML)

La vue affiche la page HTML. Elles récupèrent des variables du Contrôleur et/ou du Modèle pour savoir ce qu'elles doivent afficher.

C'est une page HTML avec quelques boucles et conditions PHP très simples, pour afficher les

tableaux de données issus du Modèle.

La vue contient le **DOCTYPE**. Mais elle ne peut fonctionner qu'avec le contexte du contrôleur.

Le Contrôleur (PHP)

Le contrôleur est la page appelée (le véritable index, autrement dit, le « main »).

Il fonctionne en trois étapes :

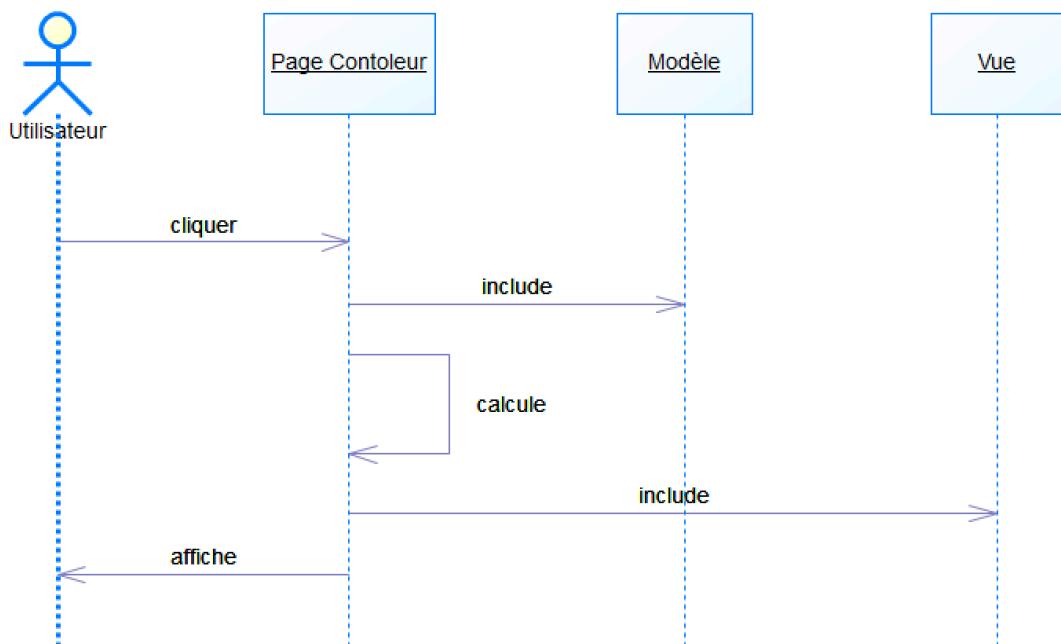
1. Il utilise les fonctions du Modèle (include et appels aux fonctions).
2. Il analyse et traite les données issues du Modèle mais aussi celles passées en paramètre à l'appel de la page (`$_GET`, `$_POST`, `$_SESSION`). Il détermine par exemple si le visiteur a le droit de voir la page ou non.
3. En fonction de ses calculs, il appelle la vue correspondante par un include.

C'est une page pur PHP.

Le contrôleur est le « chef d'orchestre » : il récupère la demande de l'utilisateur à travers la vue (la page HTML) par un href, un formulaire ou un header. Il échange des données avec le modèle, fait les calculs (qui peuvent être complexes) puis choisit une vue à afficher en lui fournissant les variables.

Le rôle du contrôleur peut se limiter à faire le lien entre le modèle et la vue : de la colle !

Fonctionnement global



Un utilisateur, à travers une vue, fait appel à une page : un contrôleur (par un href ou un formulaire). Le contrôleur « include » un modèle et utilise une des fonctions du modèle.

Il fait ensuite des calculs.

Selon les résultats, il inclut une vue ou une autre à afficher à l'utilisateur.

Et ainsi de suite.

MVC : design pattern tête la première

*Car si l'modèle - est essentiel
Et si la vue - est magnifique
J'suis p'têt feignant - oui mais c'est fou
Ces lignes de code - qui sont d'**la colle**
Et c'code n'fait rien - d'veraiment magique
Il n'fait qu'transmettre - que des valeurs*

La colle, c'est le contrôleur.

On dit que le MCV est un design pattern (DP). C'est en réalité un assemblage de DP élémentaires (un par lettre) : les DP « stratégie », « composite » et « observateur » (les DP ont des noms). Si on code réellement ces DP, alors on aura une mise à jour automatique des notifications (DP observateur).

Organisation des répertoires et des fichiers dans le MVC

Organisation de répertoires MVC

À la racine du site, on crée 3 répertoires :

- modele
- vue
- contrôleur

Dans chacun de ces répertoires, on crée un sous-répertoire par « module » du site (par exemple : forum, blog, chat, admin, etc.).

Les fichiers « modèle »

Chaque modèle est un fichier PHP contenant un appel à la BD et qui renvoie par exemple le tableau des résultats (fetchAll() plutôt que fetch() un par un).

Le fichier PHP n'aura pas de balise fermante (?>) : ça évite des problèmes !

L'objet bdd sera déclaré en global pour ne pas avoir à refaire la connexion à chaque opération. Dans l'idéal, il faut utiliser un DP singleton qui permet de ne recréer un objet que s'il n'a pas déjà été créé. Dans chaque sous-repertoire de module, on crée un fichier par fonction qui a le nom de la fonction. Ainsi on fait un include du fichier et un appel à la fonction.

Les fichiers « contrôleur »

Chaque contrôleur est aussi un fichier PHP.

Il inclut le modèle au début (include_once pour éviter de l'inclure plusieurs fois).

A la fin il inclura la vue. Autrement dit, en réalité il contient tout le code, mais on sépare le code en 3 parties (modèle, contrôle, vue) et chaque partie est dans son propre fichier.

Le contrôleur fait les calculs avant l'appel de la vue.

Quand on demande un fichier dans la barre d'adresse, c'est un contrôleur qu'on appelle.

Les fichiers « vue »

Chaque vue est un fichier HTML qui fera un simple affichage du jeu de données fourni par le contrôleur.

Il peut y avoir une boucle PHP pour parcourir les tableaux et des tests pour choisir ce qu'on veut afficher.

Il y a un fichier vue par page utilisateur.

Le font contrôleur = contrôleur global = routeur = indexSwitch

Le problème

Avec l'organisation précédente, on aura un « main » par page (par contrôleur).

Un problème majeur va être de faire le lien entre les pages : en effet, un href partira d'une page (un contrôleur) pour aller à un autre page (un autre contrôleur).

Le problème est que ça obligera à parcourir l'arborescence des fichiers avec des href du type :

Href= « ../admin/pageAdmin.php »

Si on a des href dans un fichier « includé », il en sera de même.

Avec une telle organisation, on risque souvent d'avoir des problèmes de « routage » (trouver la route pour une page à ouvrir) très complexe. Si on change un peu la structure des fichiers ou des répertoires, ou simplement si on change le nom d'un répertoire, il faudra retrouver tous les href qui sont référence à ce répertoire pour les mettre à jour : c'est long et pénible !

La solution : un « front contrôleur »

On se dotera d'un contrôleur global qui permet l'entrée dans le site et qui permet de choisir le contrôleur à appeler donc la page à afficher.

C'est le « front contrôleur » ou « routeur ». On retrouve le terme de « routeur » dans les frameworks. Les autres contrôleurs sont parfois appelés backController.

Le rôle du « front contrôleur » est de déterminer quel contrôleur appeler et de faire des initialisations générales (connexion à la BD, affichage d'en-tête ou de pied de page, etc.).

Le fichier index.html appellera le routeur avec une route particulière, c'est-à-dire un contrôleur particulier, donc une page particulière à afficher.

Ainsi, on va centraliser le problème du « routage » : si on change un nom de répertoire, il ne faudra faire des modifications que dans le « front-contrôleur ».

Le fichier indexSwitch.html : le routeur

Principes

IndexSwitch est le front contrôleur qui choisit le contrôleur à exécuter.

Ce front contrôleur, c'est ce qu'on appellera le « routeur » dans les framework php.

On regarde quel index est setté pour savoir quel contrôleur appeler.

On teste sur \$_GET et \$_POST car on peut venir d'un header, d'un href ou d'un formulaire.

Exemple

```
<?php
session_start(); //On démarre la session
include('modele/connexion_sql.php'); // connexion à la BD

include 'vues/entete.php'; //HTML de l'entête du site

// grand SWITCH d'accès aux pages // des elseif
if ( isset($_GET['indexOeuvres']) ) OR
    isset($_POST['indexOeuvres']) ) {
```

```

        include('controleur/public/' . indexOeuvres . '.php');
    }
    elseif ( isset($_GET['indexExpositions']) OR
        isset($_POST['indexExpositions']) ){
        include('controleur/public/' . indexExpositions . '.php');
    }
    //etc.

```

empty() ou isset()

C'est presque la même chose. empty() est vrai pour non défini, =0 ou =null. isset() est faux uniquement pour non défini.

<http://php.net/manual/fr/function.empty.php>

Include entête et pied de page

On peut faire des include d'entête et de pied de page dans le front contrôleur si ce sont toujours les même pour toutes les pages

Déconnexion

Pour libérer proprement la BD.

En PDO, la déconnexion n'est pas utile si on a fait des close cursor

```
include ('modele/deconnexion_sql.php'); // deconnection à la BD
```

Debug

On affiche \$_SESSION, POST et GET pour suivre ce qui se passe pour chaque page. On n'a plus besoin du nom du fichier puisque c'est toujours le même : celui du front contrôleur !

On peut mettre ça dans un test : ainsi, il suffit de passer debut à 0 pour arrêter l'affichage des superglobales.

```

$debug=1;
if ($debug==1) {
    print_r($_SESSION);echo'<br/>'; echo'POST : ';
    print_r($_POST);echo'<br/>'; echo'GET : ';
    print_r($_GET);echo'<br/>';
}

```

Variante sans switch : beaucoup plus courte !

Le \$_GET ou le \$_POST qui arrive dans le front contrôleur pourrait contenir le nom du contrôleur qu'on veut appeler (avec son chemin). Par exemple : « public/œuvres.php ». Ou encore « admin/œuvres.php ».

A partir de là, le front contrôleur vérifie que \$_GET ou \$_POST sont setté et que le fichier correspondant au contrôleur qu'on veut appeler existe (is_file()).

Si c'est le cas, on peut faire un include du contrôleur back. Sinon, on revient à la page d'accueil.

Ainsi on n'a plus besoin de swich !

```

...
//On inclut le contrôleur s'il existe et s'il est spécifié
if (!empty($_GET['page']) &&
    is_file('controleurs/'. $_GET['page'] . '.php'))

```

```
include 'controleurs/'.$_GET['page'].'.php';

else
    include 'controleurs/accueil.php'; // contrôleur accueil

...
```

Le fichier index.html

Le fichier index.html appelle avec une fonction header() le front contrôleur : ici le fichier indexSwitch.

On passe un paramètre à l'URL : par exemple, indexOeuvres : se sera la page d'accueil du site.

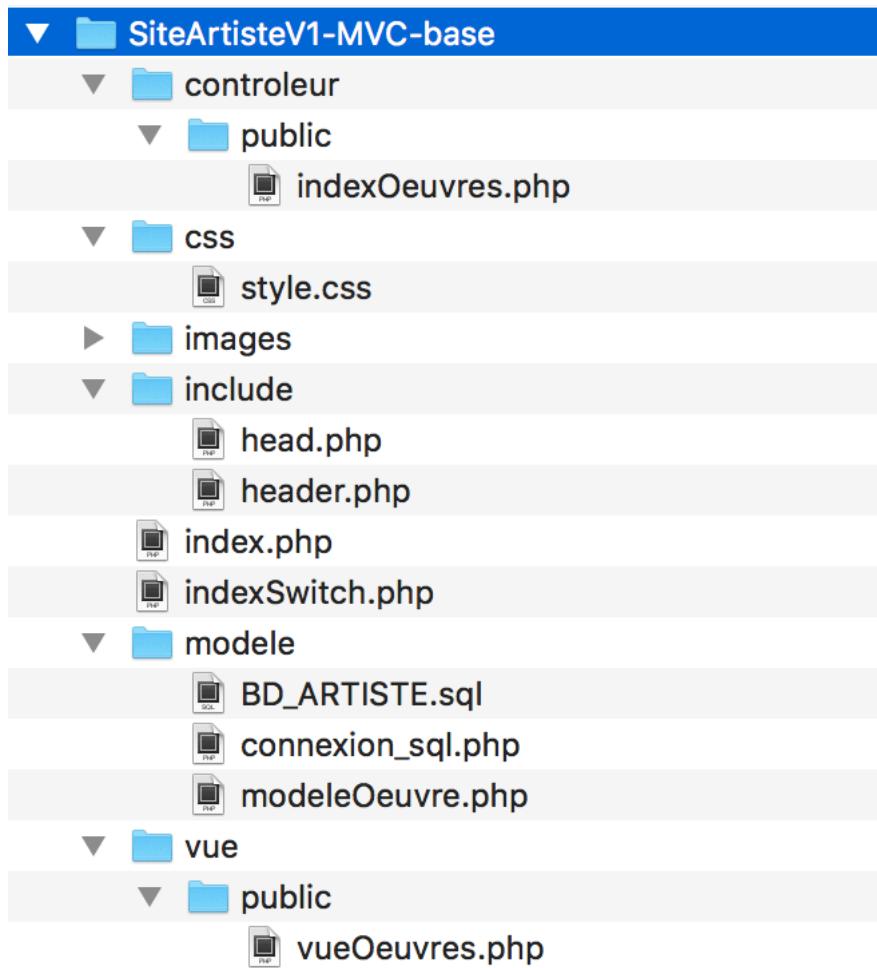
On donne n'importe quelle valeur au paramètre. On ne fera que tester s'il est setté.

```
<?php
// on appelle indexSwitch en settant indexArticle avec n'importe
quel valeur
header('location: indexSwitch.php?indexOeuvre=set');
?>
```

MVC – Exemple 7 : le Site Artiste V1-MVC - affichage des œuvres - **Testez l'exemple**

Organisation des dossiers et des fichiers

L'organisation MVC est la suivante :



TP Site Artiste MVC

TP-MVC-V2 : Site Artiste V2 – MVC : affichage de toutes les tables

A partir du SiteArtisteV1-MVC-base, faire une V2 qui prenne en compte la gestion des expositions et celle des œuvres exposées.

L'étape 2 MVC suit la même logique que l'étape 2 non-MVC.

TP-MVC-V3 : Site Artiste V3 – MVC : administration des œuvres

Ajoutez la possibilité de se connecter comme administrateur : entrée dans le back-office.

L'administrateur peut se déconnecter pour retourner au site public.

L'entrée dans l'administration se fera sur la page d'administration des œuvres.

Elle affichera la liste des œuvres dans un tableau et permettra pour chaque œuvre de la consulter en détail, de la modifier ou de la supprimer.

[La page d'administration des œuvres pourra ressembler à ceci :](#)

```
IndexSwitch.php - SESSION : Array ()  
POST : Array ([admin] => admin [password] => admin [indexAdminOeuvres] => )  
GET : Array ()
```

Administration du site ARTISTE

Gestion des oeuvres

gestion	id	titre	annee	prix	technique	support	largeur	hauteur	petite image	grande image
consulter modifier supprimer	1	titre1	2016	500	huile	toile	40	50		
consulter modifier supprimer	2	titre2	2016	500	acrylique	toile	30	60		
consulter modifier supprimer	3	titre3	2016	1000	huile	toile	100	100		

Pour charger des images, il faudra utiliser la superglobal `$FILE`.

Connexion comme administrateur

Pour se connecter comme administrateur, il faudra créer un formulaire par exemple comme ceci (dans chaque page de la partie publique) :

```
IndexSwitch.php - SESSION : Array ( [admin] => admin )
POST : Array ( [deconnexion] => deconnexion [indexOeuvres] => )
GET : Array ()
```

Site ARTISTE

oeuvres expositions

[1] : titre1 - 2016 - 500 euros

huile sur toile : 40cm X 50cm



Vérifier le login

➤ *Objectif dans un premier temps*

On vérifie de mot de passe.

En cas d'erreur, on affiche une page d'erreur.

En cas de réussite, on affiche la page d'administration des œuvres.

On gère la déconnexion.

➤ *Objectif dans un second temps*

On prendra aussi en compte la possibilité de modifier, consulter, supprimer les œuvres.

➤ *Vérification du login – mot-de-passe*

```
if(isset($_POST['admin']))
AND htmlspecialchars($_POST['admin'])== 'admin'
AND isset($_POST['password'])
AND htmlspecialchars($_POST['password'])=='admin' {
    $_SESSION['admin'] = $_POST['admin'];
}
```

Dans le contrôleur d'administration, on vérifie au début si on a bien reçu le bon login – motDePasse. On vérifie que les deux champs sont settés et qu'ils contiennent, par exemple « admin » et « admin ».

Si c'est le cas, on enregistre la valeur de `$_SESSION['admin']`. On commence donc par faire un `session_start()` en début de page.

➤ *Affichage de la bonne vue*

Selon que le `$_SESSION['admin']`) est setté ou pas, on choisit la vue à afficher.

```
// Si on n'a pas les droits d'administration
if (!isset($_SESSION['admin'])) {
    include('vue/admin/vueInterdiction.php');
}
else {
    include('vue/admin/vueAdminOeuvres.php');
}
```

➤ *Gestion de la déconnexion*

Quand on se déconnecte, on passera un `$_POST['deconnexion']` à la page de retour : en l'occurrence celle des œuvres.

De là, dans la page des œuvres, on teste si `$_POST['deconnexion']` est setté (on vient du bouton déconnexion). Si c'est le cas, on unset `$_SESSION['admin']`

➤ *Page d'erreur*

On crée une nouvelle vue pour gérer l'erreur si l'admin n'a pas le mot de passe.

La vue garde le header des utilisateurs publics : on peut retourner sur les œuvres et les expositions.

IndexSwitch.php - SESSION : Array ()
POST : Array ([admin] => admina [password] => [indexAdminOeuvres] =>)
GET : Array ()

The screenshot shows a web page with a teal header containing the text "Site ARTISTE". Below the header, there is a navigation bar with links for "oeuvres" and "expositions". A login form is present, featuring two input fields labeled "nom admin" and "password", and a button labeled "ok". At the bottom of the page, a large red banner displays the message "Vous n'avez pas l'autorisation d'accéder à l'administration".

Vous n'avez pas l'autorisation d'accéder à l'administration

TP-MVC-V4 : Site Artiste V4 – MVC : administration des des expositions

Une fois connecté en tant qu'administrateur, on peut aussi administrer les expositions : ajouter, supprimer ou modifier.

Quand on ajoute une exposition, il faut pouvoir préciser la liste des œuvres exposées.

Quand on supprime une exposition, il faut supprimer aussi toutes les œuvres exposées.

Quand on modifie une exposition, il faut pouvoir ajouter ou supprimer des œuvres et aussi modifier le prix des œuvres dans l'exposition.

Quand l'utilisateur sera connecté comme administrateur, on fera en sorte qu'il reste connecté où qu'il aille, sauf s'il se déconnecte.