

## 1/ Le Design Pattern

On désigne par design pattern (ou *patron de conception*) une manière de résoudre un problème en suivant les bonnes pratiques de développement et d'organisation d'un code. On utilise généralement la programmation orientée objet (POO), mais pas toujours.

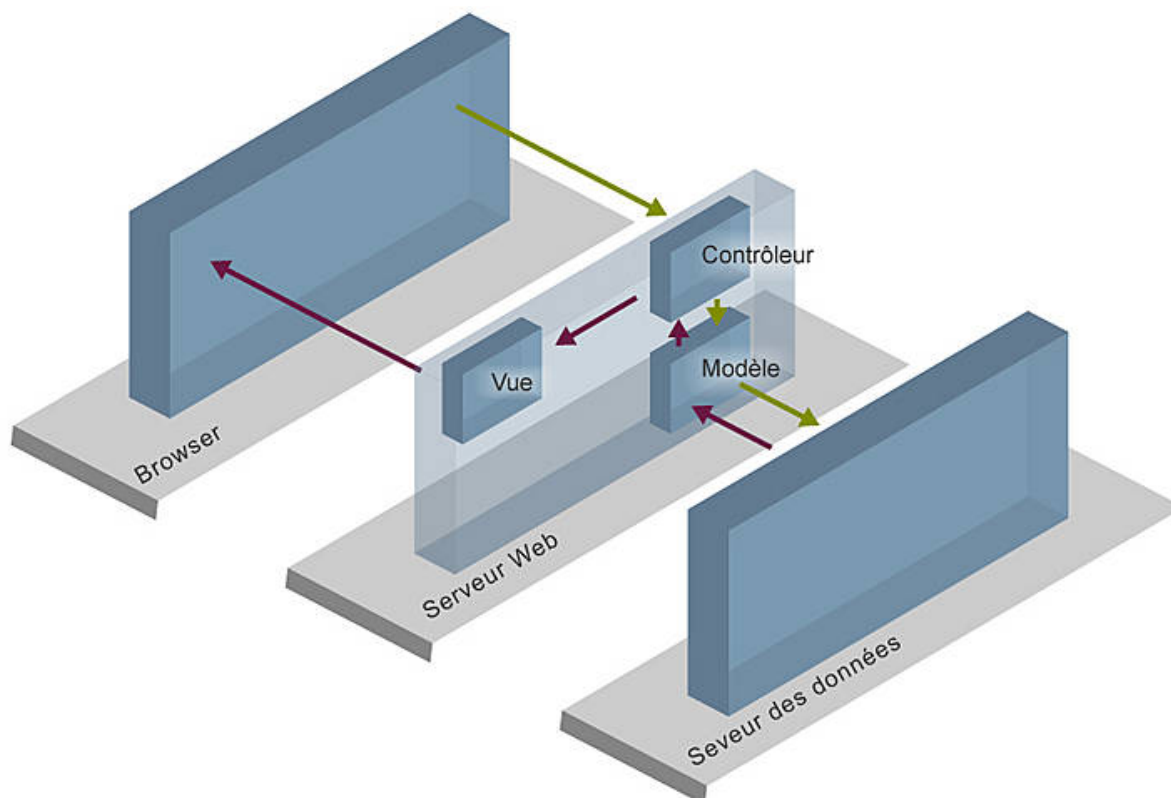
En développement informatique, il existe de nombreux design pattern, par exemple : **Singleton** (= classe statique unique instanciée/chargée depuis un trigger), **Strategy** (= séparation des algo), **Factory** (= classe d'objets), **Observer** (= écouter les objets, les tester, les logger), **DAO** Direct Acces Object (= mappage des bases de données via un CreateReadUpdateDelete), **ORM** Object-Relational Mapping (= accès automatisé des tables et besoins en requêtes côté programmation orientée objet. Plus besoin d'écrire la moindre ligne de SQL !).

Dans le cas d'une application PHP, le Design Pattern **Model View Controller** (**MVC**) est l'un d'entre eux.

## 2/ Le Design Pattern MVC et PHP

Dans un projet web, la démarche de génie logiciel pousse à définir l'architecture d'une application dans le respect des **design pattern**. L'architecture MVC cherche à séparer trois grandes pratiques :

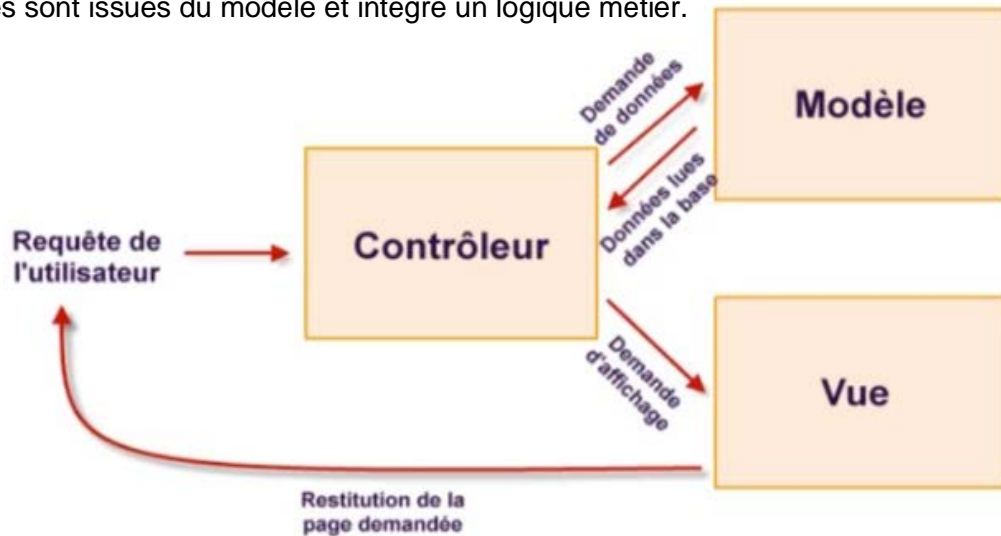
- La façon d'accéder aux données en fonction des besoins → le **Modèle**
- L'interface homme/machine: la visibilité, l'habillage, le design → la **Vue**
- Les traitements liés aux métiers/besoins de l'application → le **Contrôleur**



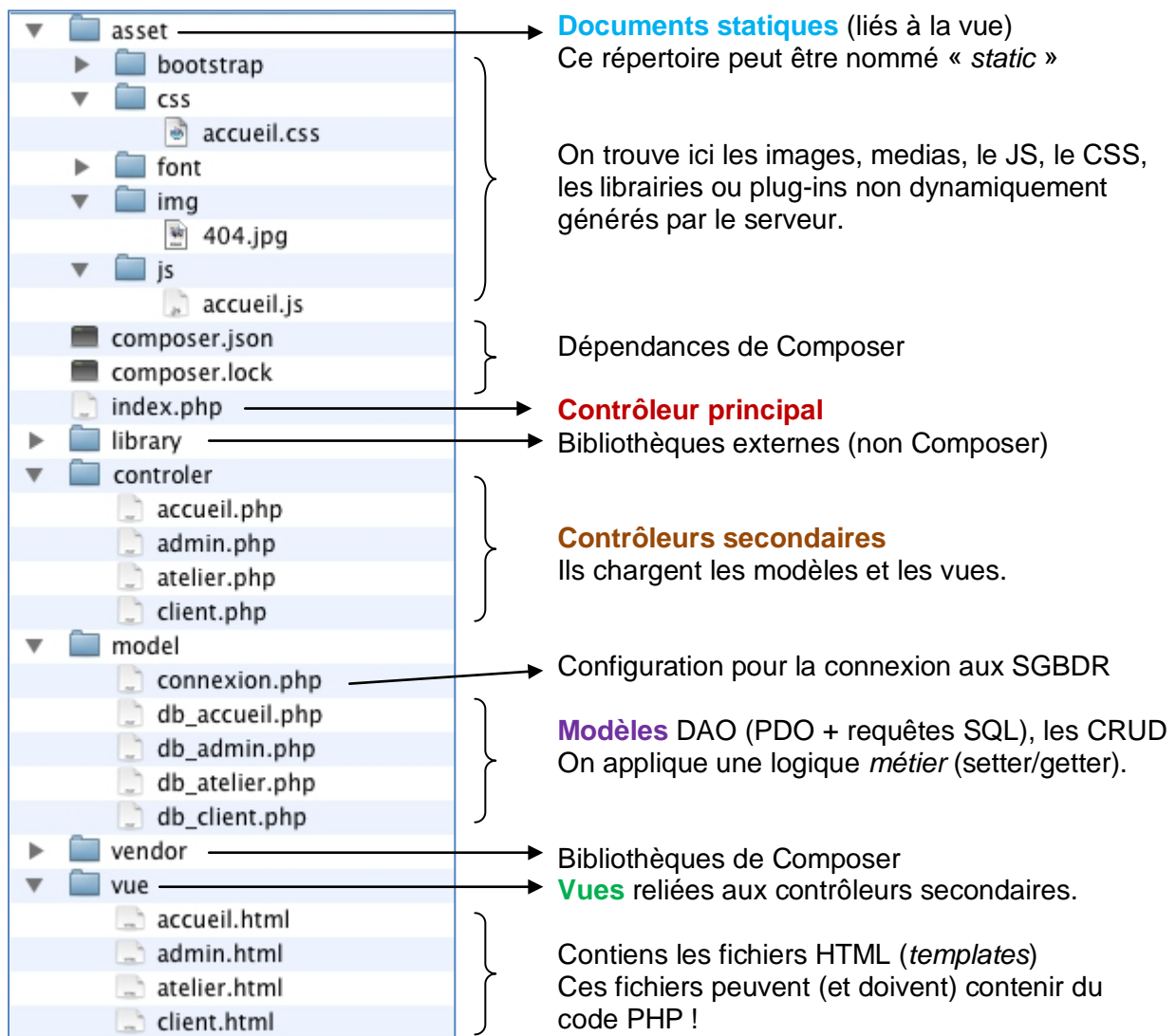
Sur PHP, le **contrôleur** orchestre au niveau ZÉRO (global) les besoins en **modèle** et en **vue**.

Soit le **Modèle**, les **Vues** et les **Contrôleurs** :

Les contrôleurs permettent de répondre aux actions de l'utilisateur. Chaque contrôle est associé à une vue : cette dernière permet de présenter l'information retournée à l'utilisateur. Les données sont issues du modèle et intègre un logique métier.



### 3/ Exemple non exhaustif d'une structure / arborescence MVC



## 4/ Exemple illustratif

Ici les exemples présentés ne sont pas à tester. Il s'agit avant tout de comprendre la structure d'un MVC.

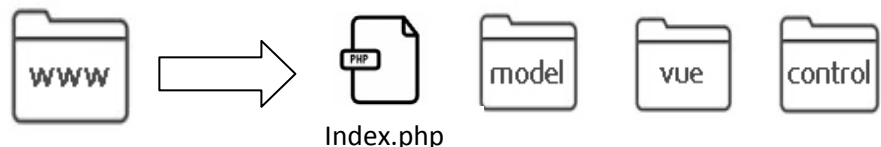
Pour commencer, l'expression la plus simple d'une page affichant un listing de prix pourrait être la suivante :

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      Exemple
    </title>
  </head>
  <body>
    <?php
      $db=mysql_connect($dbhost,$dbuser,$dbpass);
      mysql_select_db($dbname,$db);
      $query="SELECT product.name, product.price FROM product where product.type=42";
      $req=mysql_query($query)or die('Erreur SQL!<br>'. $sql.'<br>'.mysql_error());
      while ($row = mysql_fetch_array($req)) {
        echo "Nom :".$row[0]." Prix :".$row[1];
      }
      mysql_close();
    ?>
  </body>
</html>
```

C'est simple, c'est court... mais tout est mélangé ! Ce n'est donc pas une bonne pratique de développement. Ici il faudrait :

1. Savoir ce que l'on veut traiter par une requête de type HTTP → GET (**le contrôle**).
2. Se connecter à la base de données et collecter les besoins en requête SQL (**le modèle**).
3. Traiter et afficher le contenu du tableau reçu par la requête SQL dans l'HTML (**la vue**).

Selon l'arborescence suivante :



## Le Contrôleur

Pour cela il faut découper le code. D'abord, en construisant un aiguillage à requête : le **contrôleur principal**.

Exemple du point d'entrée unique : **index.php**

```
<?php
if(empty($_GET['do']))
{
    require ("vue/default.html");
}
else
{
    if($_GET['do']=="affichage")
    {
        require("control/class_example.php");
        new Example();
    }
}
?>
```

Ensuite, on isole le seul contrôleur secondaire :

Exemple du point d'entrée secondaire **/control/class\_example.php** :

```
<?php
require ("model/dao_example_dao.php");

Class Example {
    public function example() {
        $exampleDao= new ExampleDao();
        $tableau_data = $exampleDao->get_products();
        require("vue/affichage.php");
    }
}
?>
```

## Le Modèle

L'accès aux données est isolé dans une classe DAO (*Data Access Object*).

Exemple d'un fichier modèle : **/model/dao\_example\_dao.php** :

```
<?php
require ("model/connexion_bd.php");

class ExampleDao {
    public function get_products() {
        $sql = "SELECT product.name, product.price FROM product where product.type=42";
        $query = mysql_query($sql) or die('Erreur SQL !<br>'.mysql_error());
        $all = mysql_fetch_all($query);
        return $all;
    }
}
?>
```

Il convient d'implémenter la fonction *mysql\_fetch\_all()* depuis le fichier **/model/connexion\_db.php**

```
<?php

$db = mysql_connect('localhost', 'user', 'password');
mysql_select_db('test',$db);

function mysql_fetch_all($query, $kind = 'assoc') {
    $result = array();
    $kind = $kind === 'assoc' ? $kind : 'row';
    eval('while(@$r = mysql_fetch_'. $kind . '($query)) array_push($result, $r);');
    return $result;
}
?>
```

## La Vue

Dernier élément de notre architecture : la vue qui exploite l'IHM et le design de l'application.

Exemple d'une vue : `/vue/affichage.php`

```
<html>
<head>
  <title>
    Exemple
  </title>
</head>
<body>
  <?php foreach($tableau_data as $row) { ?>

    Nom : <?= $row["name"]; ?>

    Prix : <?= $row["price"]; ?>

    <br/>

  <?php } ?>

</body>
</html>
```

L'exemple présenté ici de l'architecture MVC se lance en passant le paramètre GET « **do** » dans l'url (un prototype de routage) → `index.php?do=affichage`

Évidemment, sur un exemple court, l'explosion du nombre de lignes du code n'est pas à l'avantage du pattern. Toutefois, sur le développement d'une application, le rangement opéré sur le code est nécessaire. En effet, le MVC apporte un niveau d'industrialisation du code.

Cette architecture est particulièrement préconisée dans la mise en œuvre d'une application Web pour de nombreuses raisons :

- Le point d'entrée unique permet la mise en place de traitement centralisé. Typiquement, le contrôle d'accès. Plutôt que de copier-coller (et oublier...) un test sur chaque page pour vérifier que l'utilisateur est identifié. On réalise ce test une fois et ce code est maintenu à un seul endroit.
- Le code est également beaucoup plus facile à maintenir et à partager en équipe (Designer, Front-End, Back-End)
- L'enchainement des écrans est défini par le point d'entrée unique, une modification se fera à cet endroit (et pas en partant à la recherche des 90 liens perdus dans une montagne de code).
- Si demain le produit n'est plus à chercher en base de données, mais dans un service web, il est facile de ne modifier que le DAO correspondant sans impacter la logique métier.
- Et si le design évolue, seuls les fichiers de la vue seront concernés.

Les bénéfices d'une bonne structuration d'un code sont nombreux. C'est pour cela que le MVC est devenu incontournable et qu'il est utilisé dans la quasi-totalité des frameworks PHP.

## 5/ Exemples concrets d'un MVC en PHP



Un projet utilisant le patron de conception **MVC à tester**.

Exemple du MVC avec un seul objet de connexion PDO, update avec AJAX.

➔ Il est fortement recommandé de lire les commentaires dans le code source.

- Une version simple sans AJAX :

- **NAS : RESSOURCES :** [mvc\\_form.zip](#)

- Une autre version avec AJAX :

- **NAS : RESSOURCES :** [mvc\\_ajax.zip](#)

### Installation :

Voir le fichier inclus : **INFO.txt**

Base et table générées automatiquement.