NOVEI FORMATION

LA MANU

L'ÉCOLE DES MÉTIERS DU NUMÉRIQUE
Manufacture de compétences

SUPPORT APPRENANT

PDO

CONFIDENTIEL

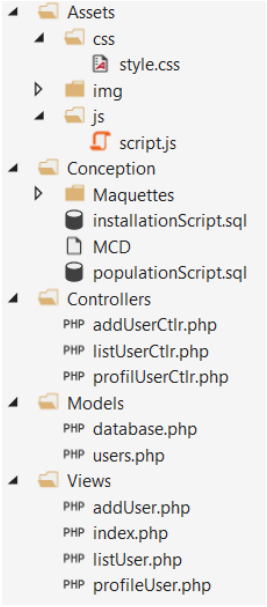
PDO

Ce que l'on a vu…

PDO est l'acronyme de **P**HP **D**ata **O**bject. Il s'agit d'une extension apparue avec la version 5.1 de PHP et qui permet de mettre en relation un Système de Gestion de Bases de Données (MySQL, SQL Server, Oracle,PostgreSQL,SQLite…) avec PHP. L'objet PDO permet de se connecter à la base de données et de communiquer avec elle. L'interface d'abstraction fournit par PDO permet d'utiliser les mêmes fonctions pour exécuter des requêtes quelque soit la base de données utilisée. Il faut simplement activer le pilote PDO de la base de données utilisée, ce qui a été fait automatiquement lors de l'installation du serveur LAMP. PDO est une extension de PHP orientée objet codée avec le langage C, ce qui la rend rapide, efficace et qui facilite sa manipulation.
L'utilisation de PDO va souvent de paire avec l'emploi de l'architecture MVC. Si on utilise correctement PDO le code sera plus sécurisé qu'en utilisant simplement PHP.

Architecture MVC

MVC est l'acronyme de **M**odèle – **V**ue – **C**ontrôleur, qui correspond à une organisation de son code en 3 parties principales. Découper son code afin de l'organiser a plusieurs avantages, cela permet de faciliter : la réutilisation de parties de code à différents endroits du projet, le travail d'équipe sur un même projet, la maintenance pour l'ajout/suppression/modification d'éléments sur le projet…
Avec une architecture MVC on ne découpe pas son code n'importe comment. Le découpage se fait selon une certaine logique et chaque partie a son rôle. Le code est réparti dans 3 dossiers : un dossier Models, un dossier Views et un dossier Controllers. On pourra également trouver d'autres dossiers utiles pour l'application tels que le dossier Assets ou encore un dossier Conception par exemple, qui contiendrait les éléments utiles à la conception de l'application mais qui ne sont pas du code (maquettes, MCD, script d'installation, script de population…).
Exemple d'arborescence d'une application web :



Modèle

Le modèle sert à traiter les données de la base de données. C'est lui qui fait le lien entre la base de données et l'application. On retrouve dans ce fichier des fonctions, qu'on appelle des **méthodes**, qui

contiennent des requêtes permettant notamment d'insérer des données dans la base, d'afficher des éléments présents dans la base de données, modifier ou encore supprimer des données.
Pour communiquer avec la base de données il faut d'abord faire une connexion à celle-ci. La connexion à la base de données se fait dans la méthode magique **__construct()** de la classe de la base de données grâce à l'instanciation d'un objet PDO de la manière suivante :

```
$this->nomAttributDeLObjetPDO = new PDO(DSN, 'nomUtilisateur', 'motDePasseUtilisateur');
```

Le DSN qui signifie **D**ata **S**ource **N**ame correspond au nom de la source de données, c'est-à-dire la base de données à laquelle la connexion se fait. Lorsqu'on utilise MySQL le DSN est de ce type : **'mysql:host=localhost;dbname=*nomBaseDeDonnées*;charset=utf8'**. L'attribut de l'objet PDO doit être déclaré au préalable en « protected » pour que les classes héritières puissent y avoir accès.
Lors de la connexion à la base de données, il faut penser à gérer les erreurs de connexion si celle-ci échouait, avec un try/catch par exemple. Pour s'aider dans la phase de conception de l'application web, on peut également afficher les erreurs SQL avec $this->*nomAttributDeLObjetPDO*->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);.

La classe de la base de données constitue la classe parent de toutes les autres classes qui représentent les tables de la base de données. Il y a donc une classe par table de la base de données et chaque classe se fait dans un fichier indépendant dans le dossier Models. Chaque classe qui représente une table est une classe qui hérite de la classe de la base de données et donc de ses méthodes magiques **__construct()** et **__destruct()**. Dans le __destruct() de la classe parent se trouve la déconnexion à la base de données. Celle-ci se réalise en affectant simplement la valeur null à l'attribut de l'objet PDO instancié dans le __construct().

Pour communiquer avec la base de données, il faut déclarer une méthode dans la classe concernée. Dans cette méthode, on commence par préparer la requête que l'on veut exécuter. Pour cela on crée un objet PDOStatement avec une méthode query() ou preprare(). On peut ensuite utiliser différentes méthodes sur cet objet PDOStatement.
Si on utilise une requête permettant d'afficher des données de la base de données (SELECT), on peut utiliser la méthode query() avec la méthode fetchAll() par exemple. La méthode fetchAll() peut prendre le paramètre PDO::FETCH_OBJ qui permet de retourner les résultats sous forme d'objet et de tableau d'objets avec un fetchAll(). Des vérifications telles que la méthode is_object() peuvent toujours être ajoutées pour plus de sécurité.

Exemple :

```
public function nomFonction()
{
    $nomVariable = 'requêteSQL' ;
    $nomObjetPDOStatement = $this->nomAttributDeLObjetPDO->query($nomVariable) ;
    if (is_object($nomObjetPDOStatement)) {
        $nomVariableResultat = $nomObjetPDOStatement->fetchAll(PDO::FETCH_OBJ) ;
    }
    return $nomVariableResultat ;
}
```

Lorsque certains éléments de la requête SQL à exécuter dépendent de l'extérieur et notamment de l'utilisateur, on utilise la méthode preprare() avec des bindValue(). En effet, faire une requête préparée à l'aide de la méthode prepare() permet de se protéger des injections SQL qu'un utilisateur malveillant pourrait vouloir tenter. Les éléments de la requête SQL provenant de l'utilisateur sont remplacés par des marqueurs nominatifs auxquels on attribue une valeur grâce à la méthode bindValue(). Ensuite la

requête peut être exécutée avec la méthode execute(). Puis, si nécessaire d'autres méhtodes telles que fetch() ou fetchAll() pourront être appliquées. Comme avec la méthode query() des vérifications peuvent être effectuées, notamment pour vérifier que la méthode execute() s'exécute correctement (dans ce cas elle renvoie true, sinon false) ou que l'on manipule bien des objets avec la méthode is_object().

Le modèle ne contient que du code PHP, il n'est donc pas nécessaire de mettre la balise fermante **?>** à la fin de ce fichier.
Lorsqu'on code une nouvelle fonctionnalité en utilisant une architecture MVC, on commence généralement par écrire le modèle avec notament la requête SQL dont on aura besoin dans notre méthode.

Vue

La vue sert à afficher les informations dans une page HTML. Il y a plusieurs vues dans un projet. Certaines comporteront uniquement de l'affichage d'éléments écrits à la main, d'autres afficheront des informations de la base de données, et d'autres encore permettront de demander des informations à l'utilisateur. Les vues sont majoritairement constituées de code HTML, mais contiennent également du PHP. On y retrouve notamment des variables PHP, des conditions ou des boucles. Lorsqu'on affiche un élément en PHP avec un echo, il est possible de simplifier l'écriture en utilisant un « short open tag » (**<?=** *élémentAAfficher* **?>**) pour faciliter la lisibilité du code.
C'est dans la vue que les éventuels messages d'erreur seront affichés si des erreurs sont faites par l'utilisateur dans un formulaire ou si un problème est survenu, empêchant l'exécution de la requête par exemple. Mais il faut égalemment prévoir d'afficher des messages de succès pour prévenir l'utilisateur lorsque tout s'est bien passé. Dans tous les cas il ne faut pas laisser l'utilisateur dans le doute.

Contrôleur

On fait un fichier contrôleur par vue. Pour une meilleure compréhension du code on nomme ce fichier du nom de la vue auquel il se rapporte, suivi du mot Controller ou de son abréviation Ctlr.
Le contrôleur sert à faire le lien entre la vue et le modèle. Ainsi, c'est dans ce fichier que l'on va instancier des objets avec les classes déclarées dans les modèles et que l'on va faire appel aux méthodes déclarées dans les modèles. Cela permettra d'afficher des données de la base de données dans la vue ou bien de récupérer des données depuis la vue pour les ajouter dans la base de données.

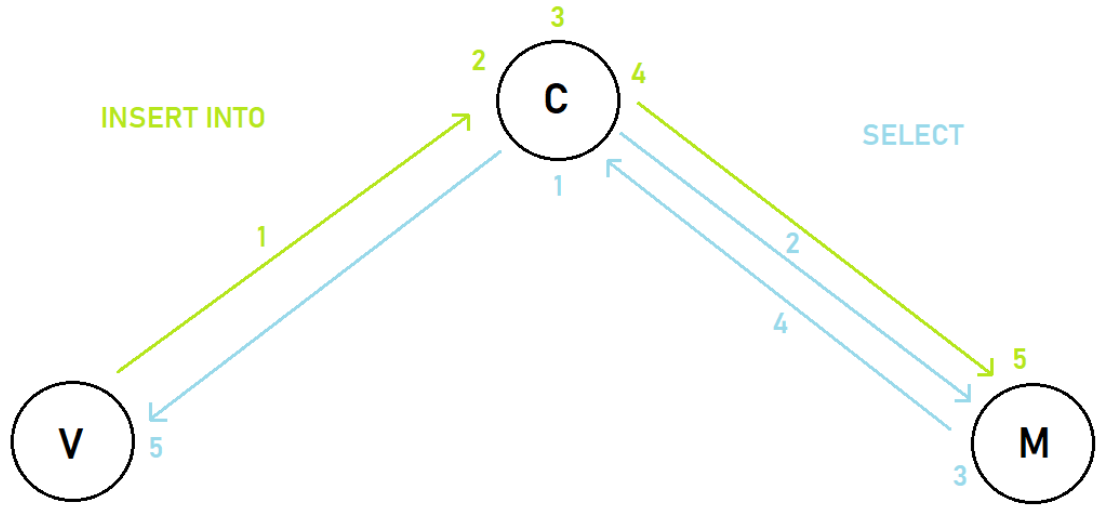
C'est aussi dans le contrôleur que tous les contrôles sont faits. Lorsqu'il y a un formulaire sur l'application il faut vérifier que les valeurs saisies par l'utilisateur correspondent à ce qui est attendu, notamment à l'aide de conditions ou de regex. Il ne faut pas oublier de renvoyer des messages d'erreur clairs à l'utilisateur lorsqu'une erreur est survenue. On veille également à utiliser la méthode htmlspecialchars() sur les valeurs avant de les attribuer aux attributs de notre objet. C'est une sécurité qui va permettre de transformer les caractères spéciaux qui pourraient être interprétés comme du code HTML en chaîne de caractères. Par exemple, les chevrons **<** et **>** seront transformés en **<** et **>** et deviennent ainsi innoffensifs.

Le contrôleur, comme le modèle ne contient que du PHP, il n'est donc pas nécessaire de fermer la balise PHP à la fin de ces fichiers car cela risquerait d'envoyer par erreur du code HTML sous forme d'espaces blancs à la vue.

Pour que les différents fichiers puissent communiquer entre eux, il faut veiller à les inclure là où ils sont nécessaires en utilisant par exemple des require, require_once ou include.

Exemples du cheminement entre les différents fichiers (Modèle, Vue, Contrôleur) pour 2 commandes couramment utilisées dans la réalisation d'une application web :

- **INSERT INTO**
  1. Envoie des informations saisies par l'utilisateur dans le formulaire de la vue au contrôleur.
  2. Instanciation d'un nouvel objet avec la classe relative à la table de la base de données que l'on souhaite remplir.
  3. Vérification des valeurs renseignées par l'utilisateur dans la vue, ajout d'un message d'erreur au tableau d'erreurs lorsqu'il y en a une, et attribution des valeurs contrôlées aux attributs de l'objet précédemment instancié.
  4. Appel de la méthode du modèle permettant l'ajout des éléments dans la base de données quand il n'y a aucune erreur (sinon les messages d'erreur présents dans le tableau d'erreurs sont affichés dans la vue).
  5. Exécution de la méthode appelée et insertion des données dans la base de données. Si le contrôleur détermine que l'exécution de la méthode s'est déroulée correctement un message de succès pourra être affiché dans la vue.



- **SELECT**
  1. Instanciation d'un objet dans le contrôleur.
  2. Appel de la méthode du modèle dans le contrôleur.
  3. Exécution de la méthode et récupération des données depuis la base de données.
  4. Attribution des données récupérées à l'objet précédemment instancié dans le contrôleur.
  5. Affichage des données dans la vue.

Liens utiles

https://www.php.net/
https://www.grafikart.fr/tutoriels/pdo-php-111
https://www.bestcours.com/programmation-web/php/590-php-avance-gerer-une-db-avec-pdo-pdf.html
https://www.php.net/manual/fr/book.pdo.php
https://openclassrooms.com/en/courses/4670706-adoptez-une-architecture-mvc-en-php
https://www.alsacreations.com/article/lire/254-le-point-sur-la-fonction-include-php.html
https://pear.php.net/manual/en/standards.php

Bonnes pratiques

Rappel des principales bonnes pratiques	
☐	Utiliser une architecture MVC.
☐	Organisation du dossier de projet : dossier Controllers, dossier Models, dossier Views
☐	Nommage des fichiers : • **Controller** : 1 controller par page, le fichier prend le nom de la page + controller ex : addPatientsCtrl.php ou addPatientsController.php • **Models** : 1 model par table, le fichier prend le nom de la table ex : patients.php • **Views** : Le fichier prend le nom de l'action de la page en anglais addPatient.php
☐	Ne pas mettre la balise fermante PHP ?> dans les fichiers ne contenant que du PHP.
☐	Utilisation de prepare() et bindValue() avec des marqueurs nominatifs quand des éléments de requête sont modifiable par l'utilisateur (protection contre injection SQL).
☐	L'utilisation de l'anglais est obligatoire pour les noms de variables, des class, des id…
☐	Pour une meilleure compréhension du code, il faut indenter son code.
☐	Utiliser les commentaires.
☐	Utilisation des simples quotes.
☐	Utilisation des && et || plutôt que AND et OR.
☐	Stocker les valeurs des $_POST['nameInput'] dans des variables et utiliser ces variables plutôt que d'utiliser plusieurs fois $_POST['nameInput'].
☐	Un seul return par fonction.
☐	Vocabulaire spécifique : • Model • Class • Attributs • Objet • Hydratation • Requêtes préparées • Try catch • Transaction • Héritage