

PHP : L'architecture MVC

L3 Informatique - UE Développement Web

David Lesaint
david.lesaint@univ-angers.fr



Janvier 2020

- ① L'architecture MVC
- ② Exemple de mise en oeuvre
 - Présentation de l'exemple
 - Mise en place d'une architecture MVC simple
 - Passage à une architecture orientée objet avec contrôleur frontal
 - Construction d'un framework MVC
- ③ La persistance

Se familiariser avec l'architecture MVC

- Comment est organisée l'architecture MVC ?
- Comment peut-t'on l'appliquer pour le Web ?

Illustration

Au travers d'un site jouet de type "blog" :

- Basé sur le cours de [B. Pesquet \(2015\)](#).
- [Codes sources](#).

L'architecture MVC

L'architecture MVC

Le modèle 1

Tous les traitements sont réalisés dans la même page.

Du modèle 1 au MVC

Une plus grande maîtrise du Développement Web requiert le passage du modèle 1 au modèle MVC.

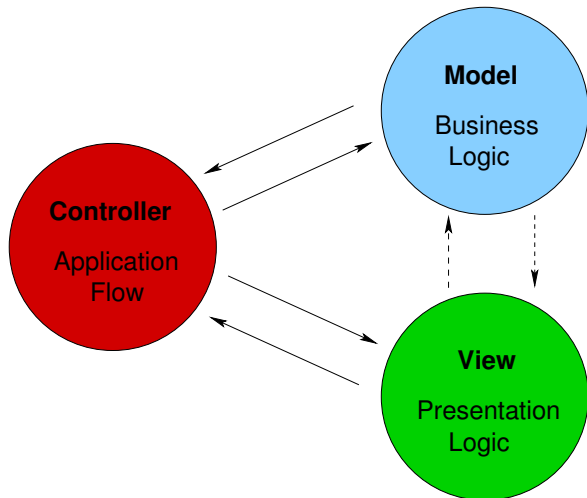
Patron de conception (alias design pattern)

- Décrit une solution standard suivant le paradigme objet pour répondre à un problème récurrent de conception logicielle.
- On encourage les développeurs à les mettre en oeuvre même si c'est parfois contraignant.

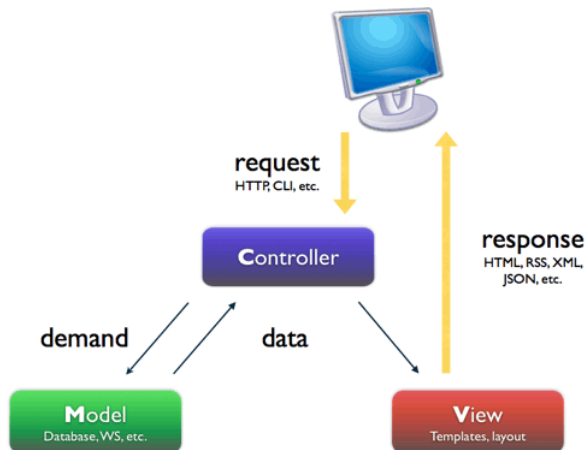
Model View Controller

- Patron de conception orienté objet.
 - Elaboré par Trygve Reenskaug en 1979 au Xerox PARC.
 - Initialement pour le langage Smalltalk.
 - Formalisé par Steve Burbeck.
 - Repose sur la séparation des préoccupations (separation of concerns) :
 - **Modèle** : L'accès aux données et la logique métier.
 - **Vue** : L'interaction avec l'utilisateur (présentation, saisie, validation).
 - **Contrôleur** : La dynamique de l'application.
- Différentes interprétations et implantations existent.

MVC schématique



MVC schématique (documentation Symfony)



Les trois parties du MVC

Le modèle

- Accède aux données : création, lecture, mise à jour, destruction (CRUD).
- Met en oeuvre la logique métier (business logic).

La vue

- Concerne l'interaction avec l'utilisateur : présentation des données du modèle, saisie, validation.
- Pour un ou plusieurs périphériques de sortie.

Le contrôleur

- Gère la dynamique de l'application en déterminant les traitements à réaliser en fonction des requêtes de l'utilisateur.
- Réécrit les URL.

Avantages du MVC et difficultés liées au Web

Apports du MVC

- Adaptée aux applications graphiques.
- Facilite l'ingénierie logicielle :
 - Conception modulaire.
 - Activités de développement naturellement distribuées.
 - Maintenance et évolution facilitées.

Difficultés liées au Développement Web

- Prendre en compte la persistance des données.
- Prendre en compte la distribution des objets (cas des applications distribuées).
- La “couche” Contrôleur doit-elle prendre en compte les traitements ?

Persistence et MVC

La “couche” persistence traite de l'échange de l'information avec les bases de données.

- Certains considèrent qu'elle fait partie du modèle.
- Il est préférable de l'extraire du modèle et de mettre en correspondance modèle relationnel et modèle objet \Rightarrow technique de l'ORM (**Object Relational Mapping**).

Distribution et MVC

Certaines applications Web sont distribuées.

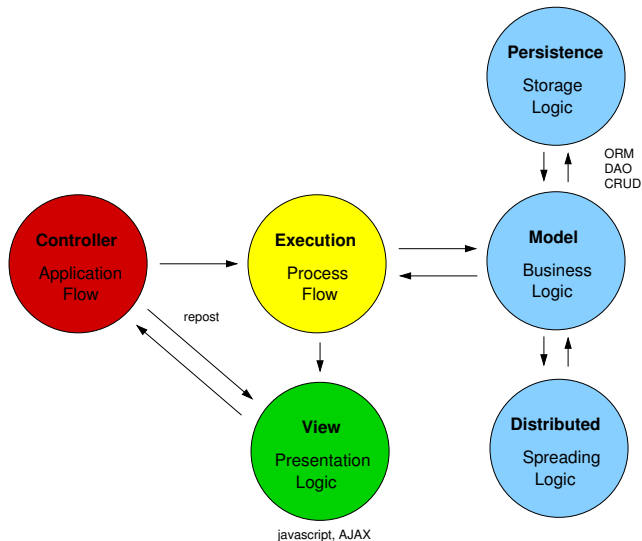
- Les classes du modèle ne sont pas toutes stockées sur le même serveur.

A prendre en compte lors de la conception et l'implantation.

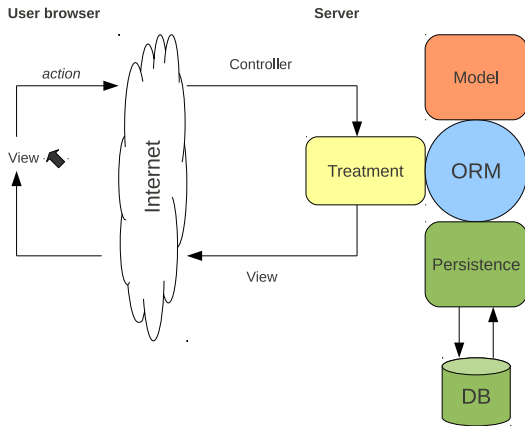
Le contrôleur doit-il réaliser les traitements ?

- C'est le cas pour les applications standards (non Web).
- On peut ajouter une nouvelle partie **Exécution** à l'architecture MVC qui se charge de réaliser les traitements.
- L'objectif est de simplifier l'implantation.
- Nouvelle architecture : MPD-V-CE.

M(PD)-V-C(E) schématique



L'architecture M(PD)-V-C(E) en action



Présentation de l'exemple





Présentation de l'exemple

Mise en oeuvre d'une page Web de type "blog"

- Articles (billets) et commentaires sur articles.

Base de données

- Une table pour stocker les articles.
- Une table pour stocker les commentaires.

monblog T_BILLET	
	BIL_ID : int(11)
	BIL_DATE : datetime
	BIL_TITRE : varchar(100)
	BIL_CONTENU : varchar(400)

monblog T_COMMENTAIRE	
	COM_ID : int(11)
	COM_DATE : datetime
	COM_AUTEUR : varchar(100)
	COM_CONTENU : varchar(200)
	BIL_ID : int(11)

index.php

- Ecrite en HTML5 : usage de la balise `article`, ...
- Affichage
 - Abrégé avec : `<?= ...?>`
 - Plutôt que `<php echo ...?>`
- Utilisation de `PDO` pour accès à la BD.

Limites de l'approche

- Mélange balises HTML et code PHP.
- Structure monolithique difficile à maintenir et faire évoluer.

Principe de responsabilité unique

Décomposer l'application en sous-parties ayant chacune un rôle et une responsabilité particulière :

- **Interactions utilisateur** : affichage, saisie et validation des données.
- **Données** : accès aux données manipulées et stockage.
- **Traitements** : opérations sur les données en lien avec les règles métiers.

Mise en place d'une architecture MVC simple

Isolation de l'affichage et de l'accès aux données

Méthode

- 1 Placer le code de présentation dans un fichier dédié `vueAccueil.php`.
- 2 Placer une fonction d'accès BDD aux billets dans un fichier dédié `Modele.php` :
 - `getBillets()`
- 3 Lier présentation et accès BDD dans le fichier `index.php` via
 - `require "Modele.php"` au début.
 - `require "vueAccueil.php"` à la fin.

Factorisation des éléments d'affichage communs

`gabarit.php`

Un gabarit (template)

- Contient les éléments communs aux pages constituant un site web : en-tête, pied de page, menu. ...
- Permet l'ajout d'éléments spécifiques à chaque vue.

`vueAccueil.php`

Définition des éléments spécifiques `$titre` et `$contenu`

- 1 Création et mise en tampon d'un flux HTML contenant la liste des articles avec `ob_start` et `ob_get_clean`.
- 2 Initialisation de `$contenu` avec ce flux puis inclusion ("instantiation") du gabarit.

Factorisation de la connexion à la BDD et gestion des erreurs

Ajout d'une fonction de connexion `getBdd()` à la BD dans `Modele.php`

- Instancie et renvoie l'objet PDO.
- Réutilisable par toute fonction exécutant des requêtes.

Gestion des erreurs par le contrôleur (`index.php`)

- 1 Signaler les erreurs de connexion dans `getBdd()`.
- 2 Ajouter l'affichage d'erreur à `index.php` en réutilisant le gabarit par import d'un fichier dédié `vueErreur.php`.
- 3 Ajouter un `try/catch` pour importer `vueAccueil.php` ou bien `vueErreur.php`.

Ajouter de nouvelles fonctionnalités

De manière méthodique

- 1 Ecriture des fonctions d'accès aux données dans le modèle.
- 2 Création d'une nouvelle vue utilisant le gabarit.
- 3 Ajout d'une page contrôleur pour lier modèle et vue.

Exemple : affichage des détails d'un billet

Cliquer sur le titre d'un billet doit afficher son contenu et les commentaires sur une nouvelle page

- 1 Ajouter à `Model.php` deux fonctions `getBillet($id)` et `getCommentaires($id)`.
- 2 Créer une vue `vueBillet.php` qui repose sur des variables fournies par le contrôleur.
- 3 Créer un contrôleur `billet.php` qui
 - Importe le modèle.
 - Reçoit l'identifiant du billet en paramètre (HTTP GET).
 - Récupère billet et commentaire par le biais du modèle importé.
 - Importe la vue.
- 4 Ajouter dans `vueAccueil.php` un lien vers `billet.php` paramétré avec l'id du billet pour chaque billet.

index.php

```

1 <?php
2
3 require 'Modele.php';
4
5 try {
6     $billets = getBillets();
7     require 'vueAccueil.php';
8 }
9 catch (Exception $e) {
10     $msgErreur = $e->getMessage();
11     require 'vueErreur.php';
12 }

```

Modele.php

```

1 <?php
2
3 // Renvoie la liste des billets du blog
4 function getBillets() {
5     $bdd = getBdd();
6     $billets = $bdd->query('select BIL_ID as id, BIL_DATE as date,
7         . ' BIL_TITRE as titre, BIL_CONTENU as contenu from T_BILLET'
8         . ' order by BIL_ID desc');
9     return $billets;
10 }
11
12 // Renvoie les informations sur un billet
13 function getBillet($idBillet) {}
14
15 // Renvoie la liste des commentaires associés à un billet
16 function getCommentaires($idBillet) {}
17
18 // Effectue la connexion à la BDD
19 // Instancie et renvoie l'objet PDO associé
20 function getBdd() {}

```

billet.php

```

1 <?php
2
3 require 'Modele.php';
4
5 try {
6     if (isset($_GET['id'])) {
7         // intval renvoie la valeur numérique du paramètre ou 0 en cas d'échec
8         $id = intval($_GET['id']);
9         if ($id != 0) {
10             $billet = getBillet($id);
11             $commentaires = getCommentaires($id);
12             require 'vueBillet.php';
13         }
14         else {
15             throw new Exception("Identifiant de billet incorrect");
16         }
17         else {
18             throw new Exception("Aucun identifiant de billet");
19         }
20     } catch (Exception $e) {
21         $msgErreur = $e->getMessage();
22         require 'vueErreur.php';
23     }
24 }

```

vueAccueil.php

```

1 <?php $titre = 'Mon Blog'; ?>
2
3 <?php ob_start(); ?>
4 <?php foreach ($billets as $billet): ?>
5     <article>
6     <headers>
7     <a href="<?php echo 'billet.php?id=' . $billet['id']; ?>">
8     <h1 class="titreBillet"><?php echo $billet['titre']; ?></h1>
9     </a>
10     <time><?php echo $billet['date']; ?></time>
11     </headers>
12     <p><?php echo $billet['contenu']; ?></p>
13     </article>
14     <hr />
15 <?php endforeach; ?>
16 <?php $contenu = ob_get_clean(); ?>
17
18 <?php require 'gabarit.php'; ?>

```

gabarit.php

```

1 <!doctype html>
2 <html lang="fr">
3 <head>
4     <meta charset="UTF-8" />
5     <link rel="stylesheet" href="style.css" />
6     <title><?php echo $titre; ?></title>
7 </head>
8 <body>
9     <div id="global">
10         <headers>
11         <a href="index.php"><h1 id="titreBlog">Mon Blog</h1></a>
12         <p>Je vous souhaite la bienvenue sur ce modeste blog.</p>
13         </headers>
14         <div id="contenu">
15             <?php echo $contenu; ?>
16         </div> <!-- #contenu -->
17         <footer id="piedBlog">
18             Blog réalisé avec PHP, HTML5 et CSS.
19         </footer>
20     </div> <!-- #global -->
21 </body>
22 </html>

```

vueBillet.php

```

1 <?php $titre = 'Mon Blog - ' . $billet['titre']; ?>
2
3 <?php ob_start(); ?>
4 <article>
5 <headers>
6 <h1 class="titreBillet"><?php echo $billet['titre']; ?></h1>
7 <time><?php echo $billet['date']; ?></time>
8 </headers>
9 <p><?php echo $billet['contenu']; ?></p>
10 </article>
11 <hr />
12 <headers>
13 <h1 id="titreReponses">Réponses à <?php echo $billet['titre']; ?></h1>
14 </headers>
15 <?php foreach ($commentaires as $commentaire): ?>
16     <p><?php echo $commentaire['auteur']; ?> dit :</p>
17     <p><?php echo $commentaire['contenu']; ?></p>
18 <?php endforeach; ?>
19 <?php $contenu = ob_get_clean(); ?>
20
21 <?php require 'gabarit.php'; ?>
22

```

Modèle MVC simple

- Sépare vue, modèle et contrôleur.
- Permet d'ajouter de nouvelles fonctionnalités de manière méthodique.

Limites de l'architecture

- Contrôleurs indépendants rendant difficile l'usage de politiques communes : authentification, sécurité ...
- Les noms de fichiers sont exposés dans les hyperliens.
- Le code reste procédural.

Etape suivante

- Introduire un vocabulaire d'actions et un contrôleur frontal.
- Passer à un modèle objet.

Passage à une architecture orientée objet avec contrôleur frontal

Mise en oeuvre d'un contrôleur frontal

Introduction d'un contrôleur frontal

- Constitue le point d'entrée unique du site.
- Centralise la gestion des requêtes de l'utilisateur.
- Utilise les autres contrôleurs pour réaliser l'action demandée et renvoyer la vue.

Rôles du contrôleur frontal

- 1 Analyse la requête entrante : demande d'affichage de la liste des billets ou demande d'un billet précis.
- 2 Vérifie les paramètres fournis.
- 3 Déclenche l'*action* à réaliser.
- 4 Signale l'erreur à l'utilisateur en cas d'incohérence.

Les actions

Objectif

Masquer la structure du site en utilisant un vocable d'actions désignant les différentes requêtes possibles.

- Seul le fichier du contrôleur frontal est visible dans les URL.
- Les URL pour utiliser le site ne changent pas et sont indépendantes de l'organisation en répertoires.

Mise en oeuvre

- Associer à chaque requête un nom d'action et, selon la requête, des paramètres.
- Modifier les hyperliens dans les fichiers de vue pour utiliser ces actions.
- Implémenter chaque action sous forme de méthode dans un contrôleur.

Passage à des contrôleurs orienté objet

Le contrôleur frontal

- Classe `Routeur` instanciée par `index.php`.
- Le `Routeur` construit et stocke les contrôleurs dédiés.
- Il analyse, vérifie, et aiguille la requête entrante vers le bon contrôleur en déclenchant la méthode appropriée.

Répartir les actions dans des contrôleurs dédiés

- Classe `ContrôleurAccueil` : gère la page d'accueil.
- Classe `ContrôleurBillet` : gère l'affichage d'un billet.

Chaque contrôleur

- Instancie la classe modèle requise pour récupérer les données.
- Instancie la classe `Vue` avec l'action (`string`) puis appelle la méthode `Vue::generer` avec les données.

Passage à un modèle orienté objet

Classes métiers et superclasse

- Créer des classes “métiers” modélisant les entités du domaine :
 - `Billet` (billet)
 - `Commentaire` (commentaires)
- Regrouper les services communs (connexion à la BD, exécution d'une requête SQL passée sous forme de chaîne de caractères) dans une superclasse commune `Modele`.

Remarques

- L'accès à la BD est masqué aux classes concrètes.
- La superclasse peut retarder l'instanciation de l'objet PDO à sa première utilisation (lazy loading).

Passage à une vue orientée objet

Classe de génération de vue

Objectif : centraliser la mise en tampon de(s) flux HTML.

- Créer une classe `Vue` qui générera chaque vue.
- L'affichage d'une vue se fera par instanciation de cette classe puis invocation de la méthode `Vue::generer($donnees)`.

Instanciation

- L'objet `Vue` est construit par le contrôleur qui lui communique l'action à réaliser.
- Il en déduit le fichier vue à utiliser.

Passage à une vue orientée objet

Méthode `Vue::generer($donnees)`

- Construit et stocke les variables communes (`$titre`, `$contenu`) dans un tableau associatif.
- Génère la partie spécifique de la vue puis le gabarit en appelant la méthode `generer_fichier()` avec le nom de fichier et le tableau de données.

Passage à une vue orientée objet

Méthode `Vue::generer_fichier($fichier,$donnees)`

- Vérifie l'existence du fichier.
- Extrait les données du tableau sous forme de variables par la fonction PHP `extract`.
- Met le flux HTML dans le tampon de sortie (`ob_start()`).
- Inclut le fichier.
- Renvoie le tampon de sortie (`ob_get_clean()`).

index.php 23

```

1 <?php
2
3 require 'Contrôleur/Routeur.php';
4
5 $routeur = new Routeur();
6 $routeur->routerRequete();

```

Routeur.php 23

```

1 <?php
2 require_once 'Contrôleur/ContrôleurAccueil.php';
3 require_once 'Contrôleur/ContrôleurBillet.php';
4 require_once 'Vue/Vue.php';
5
6 class Routeur {
7
8     private $ctrlAccueil;
9     private $ctrlBillet;
10
11     public function __construct() {
12         $this->ctrlAccueil = new ContrôleurAccueil();
13         $this->ctrlBillet = new ContrôleurBillet();
14     }
15
16     // Route une requête entrante : exécution l'action associée
17     public function routerRequete() {
18         try {
19             if (isset($_GET['action'])) {
20                 if ($_GET['action'] == 'billet') {
21                     $idBillet = intval($_GET['id']);
22                     if ($idBillet != 0) {
23                         $this->ctrlBillet->billet($idBillet);
24                     }
25                     else
26                         throw new Exception("Identifiant de billet non valide");
27                 }
28                 else if ($_GET['action'] == 'commenter') {
29                     $sauter = $_POST['sauter'];
30                     $contenu = $_POST['contenu'];
31                     $idBillet = $_POST['id'];
32                     $this->ctrlBillet->commenter($sauter, $contenu, $idBillet);
33                 }
34                 else
35                     throw new Exception("Action non valide");
36             }
37             else { // aucune action définie : affichage de l'accueil
38                 $this->ctrlAccueil->accueil();
39             }
40         }
41         catch (Exception $e) {
42             $this->erreur($e->getMessage());
43         }
44     }
45
46     // Affiche une erreur
47     private function erreur($msgErreur) {
48         $vue = new Vue("Erreur");
49         $vue->generer(array('msgErreur' => $msgErreur));
50     }
51
52     // Recherche un paramètre dans un tableau
53     private function getParametre($tableau, $non) {
54         if (isset($tableau[$non])) {
55             return $tableau[$non];
56         }
57         else
58             throw new Exception("Paramètre '$non' absent");
59     }
60 }

```

ContrôleurAccueil.php 23

```

1 <?php
2
3 require_once 'Modele/Billet.php';
4 require_once 'Vue/Vue.php';
5
6 class ContrôleurAccueil {
7
8     private $billet;
9
10     public function __construct() {
11         $this->billet = new Billet();
12     }
13
14     // Affiche la liste de tous les billets du blog
15     public function accueil() {
16         $billets = $this->billet->getBillets();
17         $vue = new Vue("Accueil");
18         $vue->generer(array('billets' => $billets));
19     }
20
21 }

```

ContrôleurBillet.php 23

```

1 <?php
2
3 require_once 'Modele/Billet.php';
4 require_once 'Modele/Commentaire.php';
5 require_once 'Vue/Vue.php';
6
7 class ContrôleurBillet {
8
9     private $billet;
10     private $commentaire;
11
12     public function __construct() {
13         $this->billet = new Billet();
14         $this->commentaire = new Commentaire();
15     }
16
17     // Affiche les détails sur un billet
18     public function billet($idBillet) {
19         $billet = $this->billet->getBillet($idBillet);
20         $commentaires = $this->commentaire->getCommentaires($idBillet);
21         $vue = new Vue("Billet");
22         $vue->generer(array('billet' => $billet, 'commentaires' => $commentaires));
23     }
24
25     // Ajoute un commentaire à un billet
26     public function commenter($sauter, $contenu, $idBillet) {
27         // Sauvegarde du commentaire
28         $this->commentaire->ajouterCommentaire($sauter, $contenu, $idBillet);
29         // Actualisation de l'affichage du billet
30         $this->billet($idBillet);
31     }
32
33 }
34
35

```

```

1 Modele.php 33
2
3 10@ abstract class Modele {
4
5     11
6     12     /** Objet PDO d'accès à la BD */
7     13     private $bdd;
8
9     14
10    16@ * Exécute une requête SQL éventuellement paramétrée
11    22@ protected function executerRequete($sql, $params = null) {
12    23     if ($params == null) {
13    24         $resultat = $this->getBdd()->query($sql); // exécution directe
14    25     }
15    26     else {
16    27         $resultat = $this->getBdd()->prepare($sql); // requête préparée
17    28         $resultat->execute($params);
18    29     }
19    30     return $resultat;
20    31 }
21
22    32
23    34@ * Renvoie un objet de connexion à la BD en initialisant la connexion au besoin
24    38@ private function getBdd() {
25    39     if ($this->bdd == null) {
26    40         // Création de la connexion
27    41         $this->bdd = new PDO('mysql:host=localhost;dbname=l3_crs_blog;charset=utf8',
28    42             'lesaint', 'lesaint',
29    43             array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
30    44     }
31    45     return $this->bdd;
32    46 }
33    47
34    48 }

```

```

1 Commentaire.php 33
2
3 1 <?php
4
5 2 require_once 'Modele/Modele.php';
6
7 4@ /**
8 5     * Fournit les services d'accès aux genres musicaux
9 6     *
10 7     * @author Baptiste Pesquet
11 8     */
12 9@ class Commentaire extends Modele {
13
14 10
15 11 // Renvoie la liste des commentaires associés à un billet
16 12@ public function getCommentaires($idBillet) {
17 13     $sql = 'select COM_ID as id, COM_DATE as date,
18 14         . ' COM_AUTEUR as auteur, COM_CONTENU as contenu from T_COMMENTAIRE'
19 15         . ' where BIL_ID=?';
20 16     $commentaires = $this->executerRequete($sql, array($idBillet));
21 17     return $commentaires;
22 18 }
23
24 19
25 20 // Ajoute un commentaire dans la base
26 21@ public function ajouterCommentaire($auteur, $contenu, $idBillet) {
27 22     $sql = 'insert into T_COMMENTAIRE(COM_DATE, COM_AUTEUR, COM_CONTENU, BIL_ID)'
28 23         . ' values(?, ?, ?, ?)';
29 24     $date = date('Y-m-d H:i:s'); //date(DATE_W3C); // Récupère la date courante
30 25     $this->executerRequete($sql, array($date, $auteur, $contenu, $idBillet));
31 26 }
32 27 }

```

```

1 Billet.php 33
2
3 1 <?php
4
5 2 require_once 'Modele/Modele.php';
6
7 4@ /**
8 5     * Fournit les services d'accès aux genres musicaux
9 6     *
10 7     * @author Baptiste Pesquet
11 8     */
12 9@ class Billet extends Modele {
13
14 10
15 11 // Renvoie la liste des billets du blog
16 12@ public function getBillets() {
17 13     $sql = 'select BIL_ID as id, BIL_DATE as date,
18 14         . ' BIL_TITRE as titre, BIL_CONTENU as contenu from T_BILLET'
19 15         . ' order by BIL_ID desc';
20 16     $billets = $this->executerRequete($sql);
21 17     return $billets;
22 18 }
23
24 19
25 20 // Renvoie les informations sur un billet
26 30@ public function getBillet($idBillet) {
27 31     $sql = 'select BIL_ID as id, BIL_DATE as date,
28 32         . ' BIL_TITRE as titre, BIL_CONTENU as contenu from T_BILLET'
29 33         . ' where BIL_ID=?';
30 34     $billet = $this->executerRequete($sql, array($idBillet));
31 35     if ($billet->rowCount() > 0)
32 36         return $billet->fetch(); // Accès à la première ligne de résultat
33 37     else
34 38         throw new Exception("Aucun billet ne correspond à l'identifiant '$idBillet'");
35 39 }
36 40
37 41 }

```

```

1 //Vue.php 23
2
3 <?php
4
5
6 // Nom du fichier associé à la vue
7 private $fichier;
8
9 // Titre de la vue (défini dans le fichier vue)
10 private $titre;
11
12 public function __construct($action)
13 {
14     // Détermination du nom du fichier vue à partir de l'action
15     $this->fichier = "Vue/vue". $action . ".php";
16 }
17
18 // Génère et affiche la vue
19 public function generer($donnees)
20 {
21     // Génération de la partie spécifique de la vue
22     $contenu = $this->genererFichier($this->fichier, $donnees);
23     // Génération du gabarit commun utilisant la partie spécifique
24     $vue = $this->genererFichier('Vue/gabarit.php', array(
25         'titre' => $this->titre,
26         'contenu' => $contenu
27     ));
28     // Renvoi de la vue au navigateur
29     echo $vue;
30 }
31
32 // Génère un fichier vue et renvoie le résultat produit
33 private function genererFichier($fichier, $donnees)
34 {
35     if (file_exists($fichier)) {
36         // Rend les éléments du tableau $donnees accessibles dans la vue
37         extract($donnees);
38         // Démarrage de la temporisation de sortie
39         ob_start();
40         // Inclut le fichier vue
41         // Son résultat est placé dans le tampon de sortie
42         require $fichier;
43         // Arrêt de la temporisation et renvoi du tampon de sortie
44         return ob_get_clean();
45     } else {
46         throw new Exception("Fichier '$fichier' introuvable");
47     }
48 }
49 }

```

```

1 <?php $this->titre = "Mon Blog - " . $billet['titre']; ?>
2
3 <article>
4     <header>
5         <h1 class="titreBillet"><?=$billet['titre'] ?></h1>
6         <time><?=$billet['date'] ?></time>
7     </header>
8     <p><?=$billet['contenu'] ?></p>
9 </article>
10 <hr />
11 <header>
12     <h1 id="titreReponses">Réponses à <?=$billet['titre'] ?></h1>
13 </header>
14 <?php foreach ($commentaires as $commentaire): ?>
15     <p><?=$commentaire['auteur'] ?> dit :</p>
16     <p><?=$commentaire['contenu'] ?></p>
17 <?php endforeach; ?>
18 <hr />
19 <form method="post" action="index.php?action=commenter">
20     <input id="auteur" name="auteur" type="text" placeholder="Votre pseudo"
21         required /><br />
22     <textarea id="txtCommentaire" name="contenu" rows="4"
23         placeholder="Votre commentaire" required></textarea><br />
24     <input type="hidden" name="id" value="<?=$billet['id'] ?>" />
25     <input type="submit" value="Commenter" />
26 </form>

```

```

1 <!doctype html>
2 <html lang="fr">
3 <head>
4 <meta charset="UTF-8" />
5 <link rel="stylesheet" href="Contenu/style.css" />
6 <title><? $titre ?></title>
7 </head>
8 <body>
9 <div id="global">
10 <header>
11 <a href="index.php">sh1_id="titreBlog">Mon Blog</a>
12 <? Je vous souhaite la bienvenue sur ce modeste blog.</?>
13 </header>
14 <div id="contenu">
15 <? $contenu ?>
16 </div>
17 <!-- $contenu -->
18 <footer id="piedBlog"> Blog réalisé avec PHP, HTML5 et CSS. </fo
19 </div>
20 <!-- #global -->
21 </body>
22 </html>

```

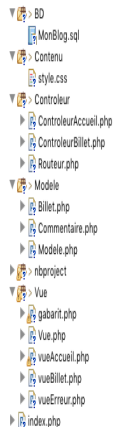
```
1 <?php $this->titre = "Mon Blog - Erreur !"; ?>
2
3 <?php $msgErreur ?></p>
4
```

```
vueAccueil.php
1 <?php $this->titre = "Mon Blog"; ?>
2
3 <?php foreach ($billets as $billet):
4     ?>
5     <article>
6     <header>
7     <a href="#"?="index.php?action=billet&id=" . $billet['id'] ?>">
8     <h1 class="titreBillet">?=" $billet['titre'] ?> </h1>
9
10    <time>?=" $billet['date'] ?> </time>
11    </header>
12    <p?=" $billet['contenu'] ?> </p>
13    </article>
14    <br />
15 <?php endforeach; ?>
16
```

Réorganisation des fichiers sources

Pour gagner en lisibilité, on regroupe les différents types de fichiers dans des répertoires dédiés

- `BD` : le script de création de la BD.
- `Contenu` : fichiers ressources “statiques” (feuilles CSS, images, ...).
- `Controleur` : contrôleurs.
- `Modele` : modèles.
- `Vue` : gabarit et vues.
- `index.php` : contrôleur frontal à la racine.



Par ajout ou spécialisation

- 1 De la classe modèle associée.
- 2 D'une vue utilisant le gabarit.
- 3 D'une classe contrôleur pour lier modèle et vue.

Exemple : ajout d'un commentaire sur un billet

- 1 Ajouter une méthode `ajouterCommentaire` à la classe `Commentaire` réalisant l'insertion SQL.
- 2 Ajouter le formulaire de saisie d'un commentaire à la vue `vueBillet.php` en créant l'action `action-commenter`.
- 3 Mettre à jour `style.css` (eg. pour dimensionner `textarea`).
- 4 Ajouter à la classe `ControlleurBillet` une méthode `commenter($auteur, $contenu, $billet)` associée à la nouvelle action qui stocke le commentaire et réactualise la page.
- 5 Mettre à jour la méthode de routage dans la classe `Routeur` pour intégrer cette action.

Construction d'un framework MVC

Les frameworks

Un framework de type MVC

- Fournit un ensemble de services de base sous la forme de classes prédéfinies
 - Routage des requêtes.
 - Sécurité.
 - Gestion du cache ...
- L'utilisateur du framework (développeur) peut se focaliser sur les classes métiers de son application.

Frameworks PHP

Laravel, Symfony, Zend Framework, CodeIgniter, CakePHP ...

Apport d'un framework à notre architecture MVC

Limites de l'architecture précédente

- Non-configurabilité des paramètres d'accès à la BD.
- Multiplication des objets de connexion PDO (un par classe modèle).
- Illisibilité des URL

`monsite.fr/index.php?action=yyy&id=zzz`

vs

`monsite.fr/action/id`

- Routage "manuel" des requêtes par le routeur.
- Aucun filtrage sur les paramètres des requêtes.
- Aucun nettoyage des données insérées dans les vues (risques de failles XSS).

Classe abstraite `Modele`

- On externalise les paramètres d'accès à la BD à l'aide d'une classe statique `Configuration.php` :
 - Lit un fichier de configuration `dev.ini` ou `prod.ini`
 - Avec la fonction `parse_ini($file)` : renvoie un tableau associatif des propriétés.
- On fait de l'objet `PDO` un attribut statique.

Automatisation du routage de la requête

Classe `Requete`

Contient les paramètres de la requête (auxquels on peut rajouter en-têtes HTTP, session, ...).

Instanciée en début de routage pour modéliser la requête.

Ajout d'un paramètre "contrôleur" aux URL en plus de l'action et paramètres d'actions :

- `monsite.fr/index.php?contrôleur=xxx&action=yyy&id=zzz`

Permet au `Routeur` de déduire de la requête quels

- Contrôleur instancier : `creerContrôleur($requete)`.
- Méthode invoquer : `creerAction($requete)`.

Puis d'invoquer : `$contrôleur->executerAction($action)`.

Usage de variable dynamique nommant la classe à instancier.

Classe abstraite `Contrôleur`

- Contient l'action à réaliser et la requête.
- La méthode `executerAction($action)` utilise la réflexion pour déduire quelle méthode de classe fille invoquer.
- La méthode `genererVue($donnees)` détermine le nom du fichier Vue à importer sur la base du nom de contrôleur utilisé.

Contraintes de nommage

Correspondances strictes à établir entre

- Nom d'action, nom de classe contrôleur, nom de fichier de classe.
- Nom de contrôleur et nom de fichier vue.

Mise en place d'URL génériques

Format standard d'URL

Pour une meilleure lisibilité et faciliter le référencement, on utilise un format "standard" d'URL pour les pages du site :

- [monsite.fr/controleur/action/id](#) remplace
[monsite.fr/index.php?controleur=xxx&action=yyy&id=zzz](#)
- Exemple : [monsite.fr/billet/index/2](#)

Réécriture d'URL

Le module `mod_rewrite` de Apache et le fichier de configuration `.htaccess` placé à la racine du site permettent de réécrire les URL avant chargement des scripts.

- Inclure l'élément `<base href="racineDuSite"/>` dans toutes les vues pour résoudre les liens relatifs.

Requete.php

```

1 <?php
2
3 @/**
4  * Classe modélisant une requête HTTP entrante
5  *
6  * @version 1.0
7  * @author Baptiste Pesquet
8  */
9 class Requete {
10
11     /** Tableau des paramètres de la requête */
12     private $parametres;
13
14     * Constructeur
15     public function __construct($parametres) {
16         $this->parametres = $parametres;
17     }
18
19     * Renvoie vrai si le paramètre existe dans la requête
20     public function existeParametre($nom) {}
21
22     * Renvoie la valeur du paramètre demandé
23     public function getParametre($nom) {}
24
25 }

```

Configuration.php

```

1 <?php
2
3 @/**
4  * Classe de gestion des paramètres de configuration
5  */
6 class Configuration {
7
8     /** Tableau des paramètres de configuration */
9     private static $parametres;
10
11     * Renvoie la valeur d'un paramètre de configuration
12     public static function get($nom, $valeurParDefaut = null) {}
13
14     * Renvoie le tableau des paramètres en le chargeant au besoin
15     private static function getParametres() {}
16
17 }

```

Modele.php

```

1 <?php
2
3 require_once 'Configuration.php';
4
5 @/**
6  * Classe abstraite Modèle
7  */
8 abstract class Modele {
9
10     /** Objet PDO d'accès à la BD */
11     private static $bdd;
12
13     * Exécute une requête SQL
14     protected function executerRequete($sql, $parametres = null) {}
15
16     * Renvoie un objet de connexion à la BDD en initialisant la connexion
17     private static function getBdd() {}
18
19 }

```

Contrroleur.php

```

1 <?php
2
3 require_once 'Requete.php';
4 require_once 'Vue.php';
5
6 @/**
7  * Classe abstraite Contrôleur
8  */
9 abstract class Contrôleur {
10
11     /** Action à réaliser */
12     private $action;
13
14     /** Requête entrante */
15     protected $requete;
16
17     * Définit la requête entrante
18     public function setRequete(Requete $requete) {}
19
20     * Exécute l'action à réaliser
21     public function executerAction($action) {}
22
23     * Méthode abstraite correspondant à l'action par défaut
24     public abstract function index();
25
26     * Génère la vue associée au contrôleur courant
27     protected function genererVue($donneesVue = array()) {}
28
29 }

```

Routeur.php

```

1 <?php
2
3 require_once 'Contrôleur.php';
4 require_once 'Requete.php';
5 require_once 'Vue.php';
6
7 @/**
8  * Classe de routage des requêtes entrantes
9  */
10 class Routeur {
11
12     * Méthode principale appelée par le contrôleur frontal
13     public function routerRequete() {}
14
15     * Instancie le contrôleur approprié en fonction de la requête reçue
16     private function creerContrôleur(Requete $requete) {}
17
18     * Détermine l'action à exécuter en fonction de la requête reçue
19     private function creerAction(Requete $requete) {}
20
21     * Gère une erreur d'exécution (exception)
22     private function genererErreur(Exception $exception) {}
23
24 }

```

Vue.php

```

1 <?php
2
3 require_once 'Configuration.php';
4
5 @/**
6  * Classe modélisant une vue
7  */
8 class Vue {
9
10     /** Nom du fichier associé à la vue */
11     private $fichier;
12
13     /** Titre de la vue (défini dans le fichier vue) */
14     private $titre;
15
16     * Constructeur
17     public function __construct($action, $contrôleur = "") {}
18
19     * Génère et affiche la vue
20     public function generer($donnees) {}
21
22     * Génère un fichier vue et renvoie le résultat produit
23     private function genererFichier($fichier, $donnees) {}
24
25     * Nettoie une valeur insérée dans une page HTML
26     private function nettoyer($valeur) {}
27
28 }

```

Sécurisation des données reçues et affichées

- Utiliser des requêtes SQL paramétrées.
- Nettoyer les valeurs PHP insérées dans les vues à l'aide de `htmlspecialchars()`.

Autres suggestions

- Authentification avec controlleur dédié pour mode “connecté”.
- Mise en place d'espaces de noms.
- Utiliser l'autochargement de classes.
- Ajouter des mécanismes de validation des données entrantes.
- Intégrer la journalisation d'évènements (logs).
- Utiliser un moteur de template (eg. Twig).
- Utiliser des composants de frameworks.

La Persistance

La couche Persistance (Persistence layer)

Définition (Persistance)

Chargée de gérer les objets persistants, ie. gère les interactions avec une base de données.

Technologies associées

- ORM (Object Relational Mapping).
- DAO (Data Access Object).
- CRUD (Create Retrieve Update Delete).

ORM, CRUD, DAO

Définition (ORM - Object Relational Mapping)

Technique dédiée à la mise en relation entre les attributs de la classe avec les champs des tables de la base de données.

Définition (CRUD - Create Retrieve Update Delete)

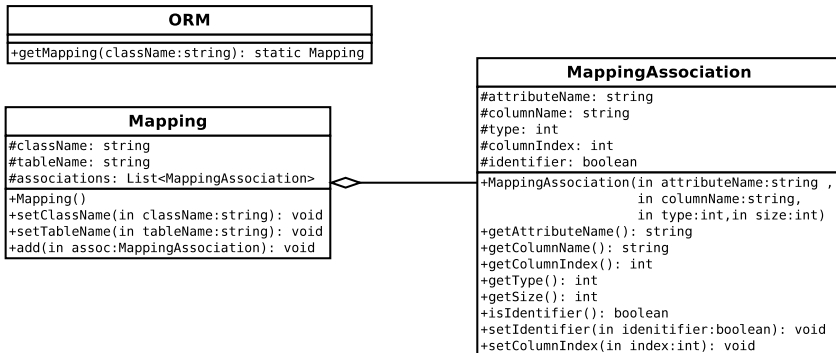
Opérations de base à implanter pour gérer l'échange d'informations entre objets et tables de la base de données.

Le CRUD peut être vu comme une interface.

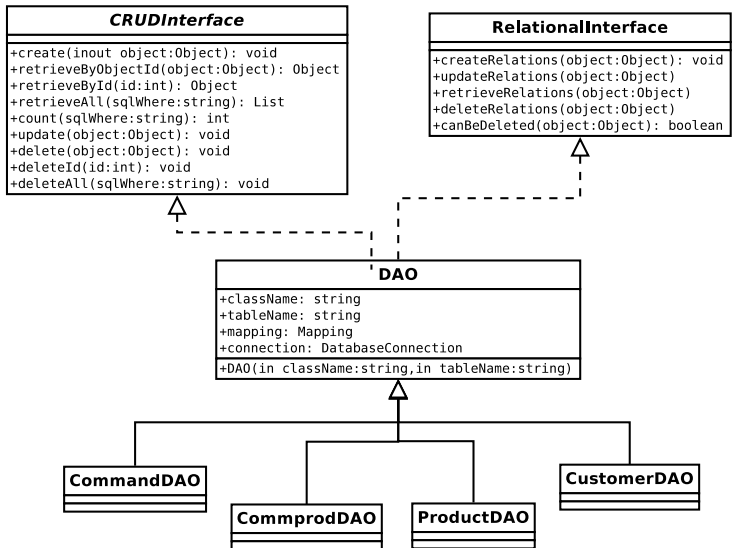
Définition (DAO - Data Access Object)

Implante le CRUD en gérant l'accès à la base de données.

ORM



CRUD et DAO



Fin