3. Architecture MVC

Le développement de sites Web importants peut amener à écrire beaucoup de code et il est donc nécessaire de l'organiser correctement afin de faciliter sa maintenance/mise à jour. Pour cela, nous allons voir comment développer des sites Web en utilisant l'architecture MVC.

R *Ce chapitre est une introduction sur le développement d'un site Web en utilisant l'architecture MVC. Ce n'est pas un cours sur l'architecture MVC en général. De plus, l'implémentation proposée dans ce chapitre n'est pas la seule implémentation possible du MVC, ni la meilleure. Elle a pour objectif de présenter cette architecture de manière simple et proche de certaines implémentations utilisées par des frameworks PHP.*

3.1 Définition de l'architecture MVC

L'architecture MVC, pour Modèle-Vue-Contrôleur, est un modèle de conception (design pattern en anglais). Un design pattern est une solution issue de l'expérience des programmeurs, permettant de résoudre un problème récurrent de programmation. L'architecture MVC est un design pattern utilisé dans le cas de développement d'applications avec interface utilisateur. Il est très utilisé en Web, notamment par certains frameworks PHP (CakePHP, Laravel, Symphony, etc).

Le modèle MVC possède de nombreux avantages :
- il permet de structurer son code de façon claire, logique et efficace,
- son adoption par de nombreux programmeurs tend à uniformiser les codes des sites Web,
- il facilite la maintenance, le débuggage et la réutilisabilité du code,
- la compréhension de cette architecture MVC facilite l'apprentissage des frameworks.

En contrepartie, le modèle MVC demande un temps d'apprentissage et nécessite une grande rigueur lors du développement.

L'architecture MVC s'articule autour de 3 modules :
- Le *Modèle* s'occupe des données et de la partie métier. Il est donc indépendant du site Web et ne contient donc aucun code HTML. C'est le modèle qui va chercher dans la base de données les données nécessaires à l'affichage de la page Web ou fait les mises à jour dans la base de données. C'est dans ce module que se trouvent toutes les requêtes SQL.
- La *Vue* présente les données à l'utilisateur. C'est le seul module contenant du code HTML. Le code PHP est réduit au minimum et sert uniquement à l'affichage.
- Le *contrôleur* fait le lien entre le Modèle et la Vue. Concrètement, le contrôleur possède plusieurs actions. **Chaque action correspond à l'affichage d'une page Web**. Pour cela, le contrôleur :
  - récupère les paramètres nécessaires pour effectuer l'action (dans `$_GET` par exemple),
  - utilise le modèle pour mettre à jour certaines données dans le cas d'ajout, de modification ou de suppression dans la base de données,
  - récupère les données nécessaires pour l'affichage de la page Web grâce au modèle,
  - appelle la vue en lui transmettant les données pour obtenir une page Web.

  Seule la dernière étape est nécessaire, les 3 premières sont facultatives et dépendent de l'action.

Nous considérons également un *routeur* (appelé également contrôleur frontal, dispatcher ou router) dont le seul but est de recevoir les requêtes utilisateurs et de déterminer le contrôleur à appeler. Ce routeur sera joué par le fichier `index.php`.

Pour accéder aux différentes pages du site Web, les URLs devront appeler le fichier `index.php` et contenir deux paramètres `controller` et `action`. La valeur de `controller` indiquera le contrôleur à appeler et `action` indiquera quelle action ce contrôleur doit effectuer.

R *Il est possible de faire de la réécriture d'URL (en utilisant un fichier `.htaccess`) pour obtenir des URLs sans paramètre. On peut donc avoir des URLs de type `http://.../Site/C/A` pour appeler le fichier `index.php` avec les paramètres `controller=C` et `action=A`. Cela permet d'obtenir des URI respectant les recommandations de l'architecture REST.*

3.2 Traitement d'une requête PHP

Dans ce cours, nous utiliserons la programmation objet pour implémenter l'architecture MVC. Le modèle sera implémenté en utilisant la classe `Model` vue dans le cours précédent. Tous les contrôleurs seront des objets héritant de la classe abstraite `Controller`. Avant de donner les détails d'implémentation, nous présentons traitement d'une requête PHP. Cette requête est une URL demandant l'exécution du routeur `index.php`. Les différentes étapes pour traiter cette requête se déroulent dans l'ordre suivant :

1. Le routeur `index.php` est exécuté :
   (a) il détermine le contrôleur à appeler grâce au paramètre `controller` dans l'URL,
   (b) il inclut le fichier contenant la définition de la classe de ce contrôleur,
   (c) il crée une instance de cette classe.
2. Lors de l'appel du constructeur :
   (a) détermination de l'action à effectuer grâce au paramètre `action` dans l'URL,
   (b) appel de la méthode correspondant à l'action.
3. Lors de l'appel de la méthode :
   (a) contrôle des données et paramètres,
   (b) interaction avec le modèle pour modifier/interroger la base de données,
   (c) inclusion de la vue pour générer le code HTML de la page Web.

Les étapes 3a et 3b sont facultatives. Elles dépendent de la page à générer en réponse à la requête. Les autres étapes sont toutes nécessaires.

La figure 3.1 illustre les interactions entre les Modèles, Vues, Contrôleurs et Routeur.

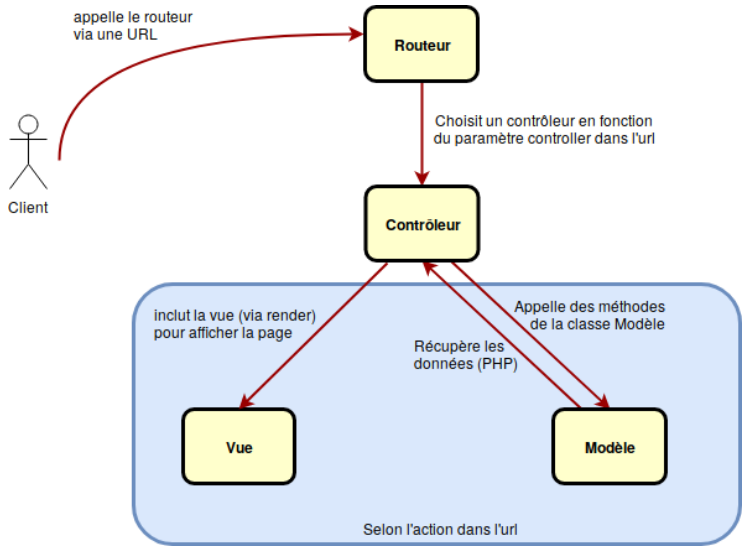

FIGURE 3.1 – Interactions Modèle/Vue/Contrôleur

3.3 Implémentation de l'architecture MVC

Nous expliquons ici une manière de coder respectant l'architecture MVC. Nous illustrons ceci en développant un site Web en exemple permettant d'afficher les personnages de la série les Simpsons. Ce site utilisera comme base de données la table `Personnages` contenant les noms et prénoms des personnages des Simpsons. Ce site aura une page d'accueil. Il comportera également une page listant tous les noms des personnages et une autre listant tous les membres d'une même famille.

Ce site utilisera deux contrôleurs :
- le contrôleur **home** dont la seule action (`home_page`) sera d'afficher la page d'accueil du site Web,
- le contrôleur **list** ayant deux actions :
  - `all` : affichera tous les personnages des Simpsons,
  - `family` : affichera tous les personnages d'une même famille.

3.3.1 Organisation du code

Le site Web sera organisé en plusieurs répertoires :
- le répertoire `Content` contiendra les images, fichiers javascript, css, les fonts, etc, utilisés pour le site Web (faire des sous-répertoires pour chaque type de contenu) ;
- le répertoire `Models` contiendra la classe `Model` définie dans le fichier `Model.php` ;
- le répertoire `Controllers` contiendra tous les contrôleurs du site ;
- le répertoire `Views` contiendra toutes les vues du site ;
- le répertoire `Utils` contiendra les autres fichiers de code PHP utilisés ;
- le fichier `index.php` sera le routeur.

Chaque contrôleur sera implémenté avec une classe dont le nom sera `Controller_` suivi du nom de la classe. Le fichier contenant la définition de la classe du contrôleur aura le même nom que la classe suivi de l'extension `.php`. Ainsi, le contrôleur **home** correspondra à la classe `Controller_home` définie dans le fichier `Controller_home.php`.

De manière similaire, chaque action du contrôleur correspondra à une méthode de la classe implémentant le contrôleur. Le nom de la méthode sera `action_` suivi du nom de l'action. Ainsi, la classe `Controller_list` contiendra deux méthodes : `action_all` et `action_family`.

Chaque vue sera stockée dans un fichier dont le nom sera `view_` suivi du nom de la vue suivi de l'extension `.php`. Par exemple, la vue `all` sera stockée dans le fichier `view_all.php`.

3.3.2 Les Vues

Chaque vue est un fichier contenant le code HTML d'une page Web du site ainsi que quelques instructions PHP nécessaires pour cet affichage. Ces instructions PHP sont essentiellement utilisées pour afficher des variables et parcourir des tableaux. Pour clarifier le code, on utilisera une syntaxe différente. On utilisera par exemple pour un `foreach` la syntaxe :

```
<!-- Code HTML -->
<?php foreach($tab as $v): ?>
    <!-- Code HTML -->
<?php endforeach ?>
<!-- Code HTML -->
```

Cette syntaxe alternative existe pour les autres structures de contrôle (`if` , `while` , etc). Pour afficher la valeur d'une variable `$var` , on utilisera l'instruction `<?= $var ?>` plutôt que l'instruction `<?php echo $var; ?>` .

La vue correspondant à l'affichage de la page listant tous les personnages des Simpsons contenus dans la base de données pourrait être :

```php
<!DOCTYPE html>
<html>
<head>
  <title> Les Simpsons</title>
</head>
<body>
<main>
<h1> Liste des personnages des Simpsons </h1>
<ul>
  <?php foreach($personnages as $p): ?>
    <li> <?= $p['Nom'] ?> <?= $p['Prenom'] ?> </li>
  <?php endforeach ?>
</ul>
</main>
</body>
</html>
```

On supposera par la suite que cette vue `all` est écrite dans le fichier `view_all.php` dans le répertoire `Views`.

R *Lors de l'écriture de la vue, on suppose que les variables contenant les informations à afficher (dans l'exemple `$personnages`) existent. Ce sera au contrôleur de créer ces variables avec les bonnes informations avant d'appeler la vue.*

R *Il est possible de découper une vue en plusieurs fichiers grâce aux inclusions de fichier (instruction `require`). On peut par exemple regrouper les premières lignes de code HTML dans un fichier et inclure ce fichier au début de chaque vue, pour que toutes les pages Web commencent par le même code HTML (en-têtes, menus, etc).*

! *Pour éviter des attaques XSS, il faut utiliser la fonction `htmlspecialchars` pour afficher les variables PHP. Pour faciliter l'écriture des vues, il est intéressant de définir la fonction (dans un fichier `functions.php` dans le répertoire `Utils` par exemple) :*

```php
function e($message){
  return htmlspecialchars($message, ENT_QUOTES);
}
```

L'affichage d'une variable `$var` dans une vue se fait alors en utilisant `<?= e($var) ?>`.

R *Pour des projets de grande envergure, il est possible d'utiliser des moteurs de templates pour définir les vues. Ces moteurs de template permettent d'écrire des templates et définissent des méthodes pour générer les vues en fonction des données et des templates. Ces moteurs de templates sont plus sophistiqués et permettent une écriture plus souple des vues. Les moteurs de template les plus connus sont Twig (utilisé par Symfony), smarty, Mustache, etc.*

3.3.3 Les modèles

Nous utiliserons comme modèle la classe `Model` introduite dans le chapitre précédent. Dès qu'un contrôleur aura besoin d'accéder à la base de données, il utilisera le modèle selon

```php
$m = Model::getModel();
$res = $m->getData(); // ou tout autre méthode de Model
```

Le fichier de définition de la classe `Model` sera inclus par défaut par le routeur selon : `require_once "Models/Model.php";`.

R *Pour des projets de grande envergure, il est possible d'utiliser des ORM (*Object Relation Manager*) tels que Doctrine (utilisé par Symfony), EZPDO ou Propel par exemple. Un ORM est une librairie qui permet d'interagir avec les bases de données via des objets. Elle permet de transformer une table en un objet facilement manipulable via ses attributs. Voici un apreçu de l'utilisation d'un ORM (doctrine) pour afficher tous les articles d'un utilisateur :*

```
$table = Doctrine_Core::getTable('Articles');

$articles = $table->findByUser($user);
foreach($articles as $article)
{
  echo $article->title;
}
```

L'ORM n'est pas la solution parfaite dans tous les cas. L'utilisation de l'ORM peut s'avérer assez complexe au niveau de l'installation et de la paramétrisation. De plus, l'utilisation d'un ORM représente un coût en termes de performance.

3.3.4 Les contrôleurs

Les contrôleurs sont des objets héritant de la classe abstraite `Controller`. Cette dernière contient trois méthodes qui seront utilisées par les contrôleurs enfants :

— le constructeur qui choisit l'action à effectuer,
— la méthode `render` qui appelle une vue pour l'affichage,
— la méthode `action_error` qui affiche une page d'erreur.

Cette classe est abstraite. Elle contient la méthode abstraite `action_default` qui doit être redéfinie dans chaque classe enfant. Cette méthode correspond à l'action qui doit être effectuée par défaut. Le squelette de la classe `Controller` est le suivant :

```
abstract class Controller
{
    // Force les classes filles à définir cette méthode
    abstract public function action_default();

    //Définition du constructeur et des méthodes render et action_error
    ...
}
```

Constructeur

La détermination de l'action se fait à la construction de l'objet. Ceci se fait selon le paramètre `action` dans l'URL. Si le paramètre `action` a la valeur `all` par exemple, l'action appelée par le constructeur, si elle existe, est `action_all()`. Si cette méthode n'existe pas ou s'il n'y a pas de paramètre de nom `action` dans l'URL, la méthode `action_default` est appelée. Le code du constructeur de la classe abstraite est donc le suivant.

```
public function __construct(){

  //On teste si un paramètre action existe et
  //s'il correspond à une action du contrôleur
  if(isset($_GET['action']) and
     method_exists($this, "action_" . $_GET["action"]) ){
    $action = "action_" . $_GET["action"];
    $this->$action(); //On appelle cette action
  }
  else
    $this->action_default(); //On appelle sinon l'action par défaut
}
```

R *Dans les contrôleurs enfants, on ne définira pas de constructeur. Le constructeur de la classe abstraite sera alors appelé. Ce dernier appellera directement l'action (spécifiée dans l'URL ou celle par défaut) du contrôleur enfant.*

Méthode `render`

La méthode `render` permet d'afficher une page HTML en insérant une vue. Elle prend en paramètre le nom de la vue à afficher ainsi qu'un tableau contenant les données (variables) à transmettre à la vue. Le code de la méthode `render` est le suivant.

```php
protected function render($vue, $data = []){

  //On extrait les données à afficher
  extract($data);

  //On teste si la vue existe
  $file_name = "Views/view_" . $vue . '.php';
  if(file_exists($file_name)){
    //Si oui, on l'affiche
    require $file_name;
  }
  else{
    //Sinon, on affiche la page d'erreur
    $this->action_error("La vue n'existe pas !");
  }
  die(); // On termine l'exécution du script une fois la vue affichée
}
```

R *La méthode utilise la fonction `extract($tab)` qui crée pour chaque case du tableau une variable dont le nom est la clé de la case et la valeur est la valeur associée à cette clé dans le tableau. Ainsi, `extract(['yes' => 'oui', 'nb' => 12])` crée les variables suivantes*

```php
$yes = 'oui';
$nb = 12;
```

La méthode `render` est utilisée dans les méthodes `action_*` des contrôleurs enfants. Par exemple, la méthode `action_all` du contrôleur `Controller_list` peut être codée de la manière suivante :

```php
public function action_all(){
  $m = Model::getModel();
  $data = [
    "personnages" => $m->getAll()
  ];
  $this->render("all", $data);
}
```

3.3.5 Le routeur

Le routeur doit déterminer, en fonction du paramètre `controller` dans l'URL, le contrôleur à appeler. Il teste si ce paramètre contient le nom d'un contrôleur existant. Si c'est le cas, il charge ce contrôleur et crée un objet de ce type. Le code du routeur pourrait être le suivant :

```php
//Pour avoir la fonction e()
require_once "Utils/functions.php";
require_once "Models/Model.php";
```

```php
require_once "Controllers/Controller.php";

$controllers = ["home", "list"]; //Liste des contrôleurs
$controller_default = "home"; //Nom du contrôleur par défaut

//On teste si le paramètre controller existe et correspond à un contrôleur
//de la liste $controllers
if(isset($_GET['controller']) and in_array($_GET['controller'],
↪  $controllers))
  $nom_controller = $_GET['controller'];
else
  $nom_controller = $controller_default;

//On détermine le nom de la classe du contrôleur
$nom_classe = 'Controller_' . $nom_controller;

//On détermine le nom du fichier contenant la définition du contrôleur
$nom_fichier = 'Controllers/' .  $nom_classe . '.php';

//Si le fichier existe et est accessible en lecture
if (is_readable($nom_fichier)) {

  //On l'inclut et on instancie un objet de cette classe
  require_once $nom_fichier;
  new $nom_classe();
}
else
  die("Error 404: not found!");
```

R *Pour des raisons de sécurité, les contrôleurs possibles à appeler sont stockés dans le tableau `$controllers`. De plus, un contrôleur par défaut est défini. Dans cet exemple, les contrôleurs pouvant être appelés sont `list` et `home`. Ce dernier est le contrôleur par défaut.*

R *Les seules lignes à modifier dans le routeur (suivant les différents sites Web) sont les lignes 7 et 8. Elles permettent de dire quels sont les contrôleurs (et le contrôleur par défaut) d'un site Web.*

Frameworks

Bien qu'un site Web puisse être développé directement en PHP ("from scratch"), il est préférable d'utiliser un framework lorsque le site Web à développer devient conséquent.

Un framework sert de squelette pour votre site Web et permet d'utiliser des modules/composants existants pour les tâches les plus courantes (moteur de template, ORM, authentification, envoi de mail, etc). Il incite également à adopter de bonnes pratiques de développement (architecture MVC par exemple).

Il existe plusieurs frameworks PHP. Les plus populaires sont : CakePHP, CodeIgniter, Laravel et Symphony. Le choix d'un framework doit se faire selon les fonctionnalités que l'on recherche, le temps d'apprentissage nécessaire pour maîtriser le framework, la communauté développant et utilisant ce framework, etc.