



TP 8 Web Dynamique

Développer une application PHP en respectant le modèle MVC - Partie 2

Cette série est la suite de la série précédente. L'objectif est toujours le développement d'une application PHP en respectant le modèle **MVC**.

Nous allons reprendre le résultat de la série précédente pour le compléter et l'améliorer.

Dans cette série nous allons :

1. Améliorer la manipulation des formulaires à travers la révision des actions **ajouter** et **modifier** (voir dossier *step1/version6-bis*).
2. Améliorer la gestion des exceptions (voir dossier *step1/version7*);
3. Améliorer le routage (voir dossier *step1/version8* et *step1/version8_suite*) ;
4. Améliorer l'affichage (voir dossier *step1/version9*) ;
5. Externaliser la configuration (voir dossier *step1/version10*).

I- Manipulation des formulaires

Lorsque une action gère un formulaire (dans notre cas par exemple : ajouter ou modifier un étudiant), alors l'action doit gérer 3 cas possibles (voir **figure 1**) :

- i. Le formulaire est affiché pour la première fois (généralement avec la méthode **GET**) ;
 - ii. Le formulaire est renvoyé (avec la méthode **POST**), mais avec des erreurs (i.e. non valide) ;
 - iii. Le formulaire est renvoyé (avec la méthode **POST**) et sans erreur.
- 1- Inspirez-vous du code de la **figure 1** pour implémenter et améliorer l'action **ajouterAction()** et de la même manière l'action **modifierAction()**.
- 2- Inspirez-vous du code de la **figure 2** pour améliorer la vue qui affiche les formulaires pour ajouter et modifier un étudiant.
- 3- Si vous ne l'avez pas fait dans la séance précédente, modifiez le routeur pour prendre en considération les actions **ajouter** et **modifier**. Modifiez aussi le modèle en écrivant les fonctions responsables d'insérer et mettre à jour les étudiants.



```

1  <?php
2  function ajouterAction() {
3      $etudiant = array("Code"=>"", "Nom"=>"", "Prenom"=>"", "Filiere"=>"SMI", "Note"=>"")
4      $erreur = array("Code"=>"", "Nom"=>"", "Prenom"=>"", "Filiere"=>"", "Note"=>"");
5      if ($_SERVER["REQUEST_METHOD"]=="POST") {
6
7          $etudiant = $_POST;
8          //valider les champs du formulaire
9          //le code, le nom, le prénom et la filière doivent être non vides
10         //vous pouvez ajouter d'autres contraintes
11         if(empty($etudiant["Code"]))    $erreur["Code"]      ="Le code est vide !...."
12         if(empty($etudiant["Nom"]))     $erreur["Nom"]       ="Le Nom est vide !...."
13         if(empty($etudiant["Prenom"]))   $erreur["Prenom"]     ="Le prénom est vide !...."
14         if(empty($etudiant["Filiere"])) $erreur["Filiere"]   ="La filière est vide !...."
15         // la note doit être non vide, et il doit être un nombre entre 0 et 20
16         if(empty($etudiant["Note"])){
17             $erreur["Note"]="La Note est vide !....";
18         elseif(!is_Numeric($etudiant["Note"])){
19             $erreur["Note"]="La Note doit être un un nombre !....";
20         elseif($etudiant["Note"] < 0 or $etudiant["Note"] > 20)
21             $erreur["Note"]="La Note doit être entre 0 et 20 !....";
22
23         //si aucune erreur n'est détectée, on appelle le modèle pour insérer dans la B
24         //puis en reexécute l'action liste pour afficher la liste actualisée
25         if (!isset($erreur)) {
26             ajouterEtudiant($etudiant);
27             header ("location: index.php?action=liste");
28             //le reste du script ne sera pas exécuté, dans ce cas
29         }
30     }
31
32     /*si on arrive ici, ça veut dire que la méthode n'est pas "post", donc c'est
33     le premier affichage du formulaire,oubien,les données envoyées par post ne sont
34     pas valide (donc le tableau $erreur est rempli (isset($erreur)==true)
35     dans ce cas, on affiche, ou réaffiche le formulaire*/
36     $filieres= getListeFilières();
37     require ("Views/vformAjout.php");
38 }

```

Figure 1 : la fonction *AjouterAction()*



```
1 <?php include("haut.php"); ?>
2
3 <h1>Ajouter un étudiant</h1>
4 <hr />
5
6 <!-- l'attribut name permet d'accéder facilement au formulaire -->
7
8 <form name = "myForm" action = "" method = "post">
9
10 <pre>
11 <!-- chaque élément de formulaire à un attribut name -->
12 Entrez le code:
13 <input type="text" name="Code" value="<?= $etudiant["Code"] ?>" /> <span class="Err"><?= $erreur["Code"] ?> </span>
14
15
16 Entrez le nom:
17 <input type="text" name="Nom" value="<?= $etudiant["Nom"] ?>" /> <span class="Err"><?= $erreur["Nom"] ?> </span>
18
19 Entrez le prénom:
20 <input type="text" name="Prenom" value="<?= $etudiant["Prenom"] ?>" /> <span class="Err"><?= $erreur["Prenom"] ?> </span>
21
22 Entrez la Note
23 <input type="text" name = "Note" value="<?= $etudiant["Note"] ?>" /> <span class="Err"><?= $erreur["Note"] ?> </span>
24
25 <!-- Un seule nom pour la liste, chaque élément de la liste a une "Value" -->
26 <?php // $selected = isset($_POST["Filiere"])?$_POST["Filiere"]:"SMI" ?>
27
28 Filière:
29 <select name = "Filiere">
30
31 <?php
32
33     foreach ($filieres as $f) { ?>
34
35         <option value = "<?= $f['CodeF'] ?>"?
36
37             <?php if ($f['CodeF']==$etudiant["Filiere"]) echo "selected"; ?>
38
39             ><?= $f["IntituleF"] ?></option>
40
41
42     <?php } ?>
43
44     </select> <span class="Err"><?= $erreur["Filiere"] ?> </span>
45
46 <input type = 'submit' value = 'Envoyer' /> <input type = 'reset' value = 'Annuler' />
47 </pre>
48 </form>
49
50 <?php include("bas.php"); ?>
51
```

Figure 2 : la vue *vFormAjouter.php*

II- Améliorer la gestion des exceptions

Pour une gestion aisée des erreurs éventuelles lors de l'exécution de votre application, il faut opter pour la centralisation de la gestion des exceptions. Puisque le fichier ***index.php*** est le seul point d'entrée à l'application, il est ainsi très facile de mettre tout le code de notre application dans un bloc ***try {...}*** et d'attraper dans un bloc ***catch {...}*** toute exception éventuelle (qui survient n'importe où dans l'application). Pour cela :

- a. Inspirez-vous de la **figure 3** pour créer une nouvelle vue **vError.php** qui permettra d'afficher une page d'erreurs personnalisée.
 - b. Mettez tout le code du fichier **index.php** dans un bloc **try {...}** et créez un bloc **catch {...}** pour y gérer les exceptions (voir **figure 4**).
 - c. Pour générer une exception à un emplacement donnée, il suffit d'utiliser l'instruction :

throw new Exception ("message d'erreur à afficher");



Remplacez donc toutes les anciennes instructions *die ("message")* par cette nouvelle instruction. Ainsi, si une erreur est survenue n'importe où dans le code, le bloc *catch {...}* sera exécuté et la vue *vError.php* sera affichée avec le bon message d'erreur.

```

1 <?php include("haut.php"); ?>
2
3 <div style="color: red; background-color: yellow"><h1>Erreur!! </h1>
4 <hr /><br />
5
6 <b>Une erreur est survenue: <?= $message ?></b> <br /><br />
7 </div>
8 <hr /><br /><br />
9 <b><a href ="javascript: history.go(-1)">Retour</a></b>
10
11 <?php include("bas.php"); ?>
12

```

Figure 3 : la vue *Views/vError.php*

```

1 <?php
2 try{
3     require ("Models/EtudiantManager.php");
4     require ("Controllers/EtudiantController.php");
5
6     $action = isset($_GET["action"]) ? $_GET["action"] : "index";
7
8     if($action=="index")
9         indexAction();
10    elseif($action=="detail")
11        detailAction($_GET["CodeE"]);
12    elseif($action=="liste")
13        listeAction();
14    elseif($action=="ajouter")
15        ajouterAction();
16    elseif($action=="modifier")
17        modifierAction($_GET["CodeE"]);
18    elseif($action=="supprimer")
19        supprimerAction($_GET["CodeE"]);
20    else
21        //die("Cette action n'est pas autorisée");
22        throw new Exception("Cette action n'est pas autorisée");
23
24 }
25 catch(Exception $e) {
26
27     $message = $e->getMessage();
28     //die ("Une erreur est survenue ". $message);
29     require ("views/vError.php");
30
31 }
32

```

Figure 4 : le fichier *index.php* avec des bloc *try...catch*



III- Améliorer le routeur

Comme le montre la **figure 4**, notre routeur dans le fichier **index.php** teste (via des **if-elseif**) la valeur de la variable **\$_GET["action"]** pour savoir quelle est l'action à exécuter. Le nombre de tests **if-elseif** augmente autant que de nouvelles fonctionnalités sont ajoutées à l'application.

Nous pouvons améliorer notre code en éliminant carrément ces blocs **if-else-if** en adoptant une convention de nommage (des actions) adéquate. Inspirez-vous de la **figure 5** ci-dessous pour implémenter cette astuce :

```

1  <?php
2  try{
3      require ("Models/EtudiantManager.php");
4      require ("Controllers/EtudiantController.php");
5
6      //appel de l'action, C'est le vrai début de l'exécution
7
8      $action = isset($_GET["action"]) ? $_GET["action"] : "index";
9
10     $action= $action."Action";
11
12     if (is_callable($action)) {
13         $action($_REQUEST);
14     }
15
16     else
17         throw new Exception("Cette action n'est pas autorisée");
18
19 }
20 catch(Exception $e) {
21
22     $message = $e->getMessage();
23     //die ("Une erreur est survenue ". $message);
24     require ("views/vError.php");
25
26 }
```

Figure 5 : amélioration du routeur dans le fichier **index.php**

Dans cet exemple :

- 1- Les noms des actions que nous avons utilisé dans les liens sont « **index**, **liste**, **detail**, **ajouter**,... » et les noms des fonctions correspondantes dans le contrôleur sont « **indexAction()**, **listeAction()**, **detailAction()**, **ajouterAction()**,...), c'est-à-dire, en concatène le nom de l'action avec la chaîne constante « **Action** » pour trouver le nom de la fonction à exécuter (**Figure 5, ligne 10**). Remarquez que le nom de la fonction est stocké dans une variable qui est exécutée à la ligne 13 (**N.B.** PHP permet d'exécuter des fonctions passées dans des variables !).



- 2- La fonction ***is_callable ("nom de fonction")*** permet de vérifier que la fonction existe bien et est exécutable (pour éviter toute sorte d'erreur, par exemple : appeler une fonction inexistante si l'utilisateur entre l'adresse ***index.php?action=blablabla***).
- 3- A la ligne 13, nous avons passé le tableau ***\$_REQUEST*** à l'action (pour permettre de passer les variables de la requête aux actions qui en ont besoin (par exemple, les actions ***supprimer***, ***modifier*** et ***detail*** ont besoin de savoir le code de l'étudiant concerné). Modifiez donc toutes les actions du contrôleur pour qu'elles acceptent un tableau (***array***) en argument.

IV- Améliorer l'affichage

Toute action se termine d'habitude par afficher une vue. Pour afficher toutes les vues d'une manière unifiée et faire toutes les vérifications nécessaires dans un emplacement unique, nous allons confier l'affichage à une fonction nommée « ***afficherReponse()*** » (Normalement, cette fonction fait partie du contrôleur, mais dans l'exemple du cours, elle est mise dans le fichier ***Views/Reponse.php***). Inspirez-vous de la ***figure 6*** ci-dessous pour implémenter cette fonction :

Dans cet exemple :

1. La fonction ***afficherReponse()*** se charge de construire la page à afficher. Pour cela, on lui passe la vue à afficher avec le tableau de variables nécessaires à l'affichage de cette vue. Les éléments communs à toutes les pages sont regroupés dans un ***template*** (par exemple ; le haut et le bas des pages, les menus,...). Donc, Le template contient tout le code HTML commun avec une instruction de type ***<?= \$view ?>*** dans le bon emplacement où on veut afficher les différentes vues (voir ***figure 7***). On peut aussi créer plusieurs templates si on veut changer l'apparence de notre application ou son layout.
2. La ligne 6 de la figure 6 : extraire le tableau des variables pour pouvoir accéder aux différentes variables indépendamment.
3. La ligne 8 : vérifier que le fichier de la vue passée en paramètre existe bien dans le dossier Views.
4. La ligne 9: la fonction ***ob_start()*** signifie « ***output buffer start*** ». On demande à PHP d'envoyer, désormais, toute la sortie au buffer (au lieu d'afficher). La ligne 11 récupère le contenu du buffer dans une variable ***\$view***. Autrement dit, les lignes de 9 à 11 permettent de récupérer le contenu du fichier de la vue dans une variable ***\$view***.
5. La ligne 18 : la seule ligne qui permet l'affichage à l'écran. Il affiche le template choisi (le code de ce template contient l'affichage de la variable ***\$view***).
6. La ligne 15 génère une exception si le fichier n'existe pas.
7. Evidemment, vous devez modifier le code des anciennes vues pour supprimer les instructions ***include("haut.php");*** et ***include("bas.php");*** car elles sont remplacées maintenant par le template.



```

1  <?php
2  /***** générer la réponse --- Affichage *****/
3
4  function AfficherReponse ($vue , array $data=array()) {
5
6      extract ($data);
7      $file = "Views/" . $vue;
8      if (file_exists ($file)) {
9          ob_start ();
10         require ($file);
11         $view= ob_get_clean ();
12     }
13     else
14     {
15         throw new Exception ("Fichier Introuvable!....");
16     }
17
18     require ("Views/templates/template0.php");
19 //require ("Views/templates/template1.php");
20 //require ("Views/templates/template2.php");
21 //require ("Views/templates/template3.php");
22 //require ("Views/templates/template4.php");
23
24 }
```

Figure 6 : la vue *Views/vError.php*

```

28  <!DOCTYPE html>
29  <html>
30  <head>
31  <meta charset="utf-8" />
32  <link rel="stylesheet" type="text/css" href="public/style.css" />
33  </head>
34  <body>
35  <div class="top">
36  <img src='public/images/fsdm.jpg' border = '0' hspace = '20' vspace = '20' align = 'left' />
37  <span class="top">SMI6</span><br />
38  Facult&eacute; des Sciences Dhar Mehraz, F&egrave;s </div>
39  <h4> <?= afficherDate("AR") ?></h4><br /><br />
40  <!--*****-->
41  <!--*****-->
42
43  <?= $view ?>
44
45  <!--*****-->
46  <!--*****-->
47  <br /><br />
48  <a href ="index.php?action=index">Accueil</a> |
49  <a href ="index.php?action=liste">Liste de &egrave;tudiants</a> |
50  <a href ="index.php?action=ajouter">Ajouter un &egrave;tudiant</a>
51  <br /><br /><br />
52  <div class="bas">&copy; copyright: SMI6 2020<br />Facult&eacute; des Sciences Dhar Mehraz<br />smi6@fsd
53
54  </body>
55  </html>
```

Figure 7 : le template *Views/templates/template0.php* (Remarquez la ligne 43)



V- Ajouter un fichier de configuration

Dans la *figure 6* ci-dessus, l'utilisateur peut modifier l'apparence de toute l'application en choisissant un autre template (il peut activer/désactiver les lignes de 18 à 22). Mais normalement cette méthode n'est pas pratique. L'utilisateur doit avoir la possibilité d'accéder à un fichier de configuration externe à l'application et dans lequel il peut modifier toutes les options possibles de l'application (par exemple : choix du template, choix de la langue de l'interface, configuration de la BD : server, login, password).

Pour cela :

1. Créez le fichier de configuration *Config/config.php* représenté par la *figure 8* ci-dessous.
2. Ajoutez l'instruction `require("Config/config.php");` dans tout fichier susceptible d'utiliser cette configuration.

```

1 <?php
2
3 //langue de l'interface
4 $config["lang"]="AR";
5
6 //templates
7 $config["templates_path"]="Views/templates/";
8 $config["active_template"]="template2.php";
9
10 //database
11 $config["database_server"]="localhost";
12 $config["database_name"]="SMI2021";
13 $config["database_user"]="root";
14 $config["database_password"]="";
15

```

Figure 8 : le fichier de configuration *Config/config.php*