

Inf340 Systèmes d'information

Deuxième site

Objectifs

Objectif :

- Reprendre le site précédent en utilisant le patron de conception Modèle Vue Contrôleur.
- Comprendre l'apport d'un framework basic
- Le site permet de gérer un carnet d'adresse composé d'une liste de noms et de numéros de téléphone, un nom peut posséder plusieurs numéros de téléphones.

Bilan du premier site

Nous avons un site :

- Sans sécurité
- Difficilement déployable
- Impossible à maintenir :
 - Redondance
 - Mélange des concepts et des technologies dans un même script.

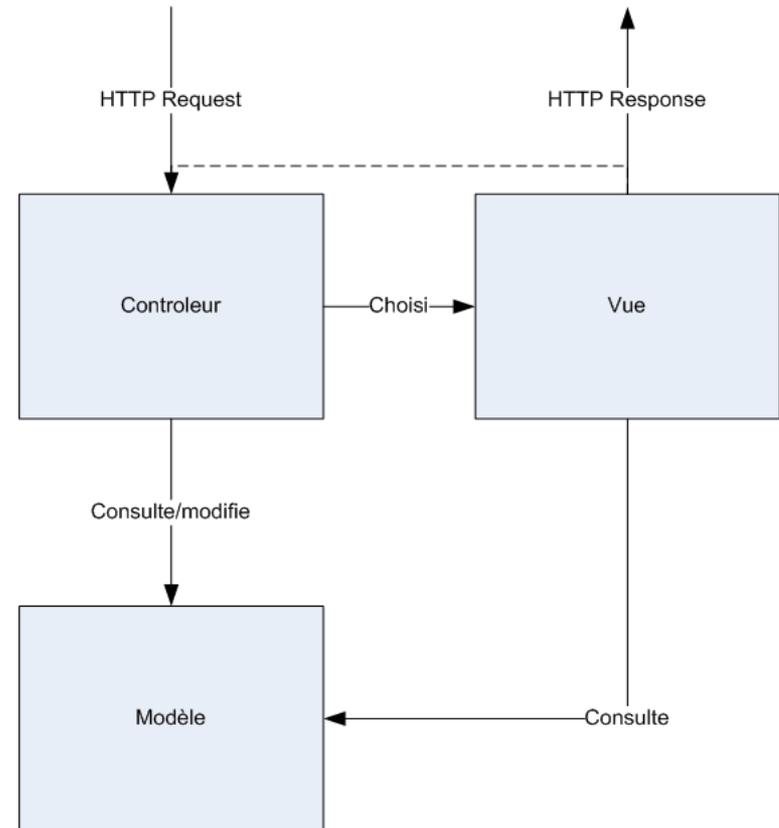
Les patrons de conceptions

Les patrons de conceptions ou Design patterns sont des solutions pour des problèmes récurrents notamment liés au paradigme objet :

- MVC
- Singleton
- Proxy
- ...

MVC appliqué au Web

- Le MVC souvent utilisé dans les interfaces graphiques s'applique sous une forme restreinte au Web.

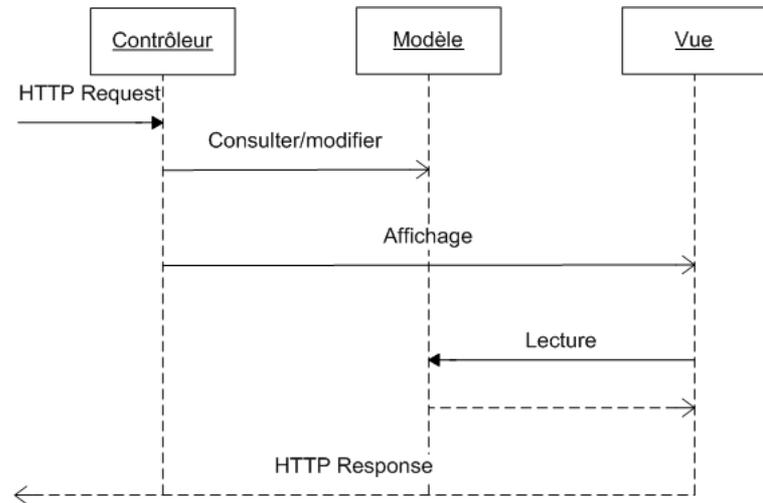


Modèle

- Le modèle qui définit le comportement de l'application et le traitement des données. Les données ne sont associées à aucune présentation. Le modèle assure la cohérence des données et offre des méthodes de manipulation de ces dernières. Dans une architecture non Web, le modèle peut signaler ses changements à la vue.
- SQL + PHP

Contrôleur

- Le contrôleur gère les événements en modifiant le modèle et en invoquant les vues



- PHP

Vue

- La vue est l'interface utilisateur, elle présente le modèle, elle reçoit les événements pour en avertir le contrôleur. Elle ne modifie pas le modèle et elle n'effectue aucun traitement.
- PHP + HTML + JS + Flash + ...

Notre mini-framework

Nous utiliserons un contrôleur principal, seul à apparaître dans l'URL.

Il est là pour charger les constantes (l'URL et les informations de connexion vers la base de données) puis aiguiller vers les contrôleurs de pages

Dans notre exemple, il recherche le nom 'module' en GET ou en POST

Le contrôleur principal

```
define('FCPATH', realpath(dirname(__FILE__)).DIRECTORY_SEPARATOR);  
//définir la constante APPPATH vers le répertoire application  
define('APPPATH', FCPATH.'application'.DIRECTORY_SEPARATOR);  
//inclure en utilisant APPPATH le fichier constants.php dans config  
require_once APPPATH.'config'.DIRECTORY_SEPARATOR.'constants.php';  
//inclure en utilisant CONTROLLERPATH le controleur de page utilisateur.php  
if (isset($_GET['module']))  
$module = $_GET['module'];  
else  
if (isset($_POST['module']))  
$module = $_POST['module'];  
else $module='personne';  
  
require_once CONTROLLERSPATH.$module.'.php';
```

Le contrôleur principal

C'est une mauvaise approche de sécurité que de permettre sans filtre d'inclure un fichier en fonction de données externes (GET, POST)

Un contrôleur de page

Il permet de gérer les actions d'une page

```
if (isset($_GET['action']))
```

```
    $action = $_GET['action'];
```

```
else
```

```
if (isset($_POST['action']))
```

```
    $action = $_POST['action'];
```

```
else
```

```
    $action='read';
```

```
//si action=une_valeur est envoyé au script nous avons récupéré dans $action la valeur et read sinon
```

```
switch ($action) {
```

```
    case 'create':
```

```
        break;
```

```
    case 'read':
```

```
        break;
```

```
    case 'update':
```

```
        break;
```

```
    case 'updateOk':
```

```
        break;
```

```
    case 'delete':
```

```
        break;
```

```
    default:
```

```
        break;
```

```
}
```

Action read

- Pas de lecture sur GET et POST
- Sollicitation du modèle
- Affichage d'une vue

Action read

```
case 'read':
```

```
    require_once
```

```
        MODELSPATH.'personne_utils.php';
```

```
    $personnes = personne\getAll();
```

```
    require_once TEMPLATESPATH.'header.php';
```

```
    require_once VIEWSPATH.'accueil_view.php';
```

```
    require_once TEMPLATESPATH.'footer.php';
```

```
    break;
```

Action create

- Lit en POST les données d'un formulaire
- Sollicite le modèle
- Effectue une redirection pour produire un rafraichissement.

Action create

```
case ('create'):
```

```
$nom = $_POST['nom'];
```

```
$prenom = $_POST['prenom'];
```

```
require_once MODELSPATH.'personne_utils.php';
```

```
personne\create($nom, $prenom);
```

```
header('Location: '.URL.'index.php?action=read');
```

```
break;
```

Action delete

- lit en GET les données d'une ancre
- Sollicite le modèle
- Effectue une redirection

Action delete

```
case 'delete':
```

```
$idpersonne = $_GET['idpersonne'];
```

```
require_once
```

```
    MODELSPATH.'personne_utils.php';
```

```
$personnes = personne\delete($idpersonne);
```

```
header('Location:
```

```
    '.URL.'index.php?action=read');
```

```
break;
```

La modification

- La modification est en deux phases
 - L’affichage du formulaire de modification : l’action update
 - Le traitement de l’information du formulaire de modification : l’action updateOk

L'action update

- Lire en GET depuis une ancre l'identifiant de l'enregistrement à modifier.
- Solliciter le modèle pour récupérer l'enregistrement depuis l'identifiant
- Afficher la vue de modification

L'action update

```
case 'update':  
$idpersonne = $_GET['idpersonne'];  
require_once MODELSPATH.'personne_utils.php';  
$personne = personne\getOne($idpersonne);  
//require_once  
    MODELSPATH.'numero_telephone_utils.php';  
//$numeros =  
    numero_telephone\getByldpersonne($idpersonne);  
require_once TEMPLATESPATH.'header.php';  
require_once VIEWSPATH.'consulter_modifier_view.php';  
require_once TEMPLATESPATH.'footer.php';  
break;
```

L'action updateOk

- Attend en POST depuis un formulaire
- Sollicite le modèle
- Actualise le modèle

L'action updateOk

```
case 'updateOk':
```

```
$idpersonne = $_POST['idpersonne'];
```

```
$nom = $_POST['nom'];
```

```
$prenom = $_POST['prenom'];
```

```
require_once MODELSPATH.'personne_utils.php';
```

```
personne\update($idpersonne, $nom, $prenom);
```

```
header('Location: '.URL.'index.php?action=read');
```

```
break;
```

Le modèle

Pour le moment non objet et directement lié à MySQL

Exemple récupérer tous les enregistrements :

```
function getAll() {  
connect();  
$sql = 'SELECT idpersonne, nom, prenom FROM personne';  
$req = mysql_query($sql) or die('Erreur SQL !<br>' . $sql . '<br>' . mysql_error());  
$res= array();  
while ($data = mysql_fetch_assoc($req)){  
$res[$data['idpersonne']] = array('idpersonne'=>$data['idpersonne'],  
    'nom'=>$data['nom'], 'prenom'=>$data['prenom']);  
};  
disconnect();  
return $res;  
}
```

Le modèle

```
function create($nom, $prenom){  
connect();  
$sql = 'INSERT INTO personne (nom, prenom)  
VALUES (\".$nom.\",\".$prenom.\")';  
  
$req = mysql_query($sql) or die('Erreur SQL !<br>' .  
    $sql . '<br>' . mysql_error());  
disconnect();  
}
```

Les vues

- Pour le moment pas de langage de templates.
- Elles sont invoquées par les contrôleurs
- Elles utilisent les données du modèle fournies par le contrôleur.

accueil_view.php

```
<?php foreach ($personnes as $personne):?>
<tr>

<td> <?php echo $personne['idpersonne']; ?></td>
<td> <?php echo $personne['nom']; ?></td>
<td> <?php echo $personne['prenom']; ?></td>
<td> <a href="<?php echo
    URL.'index.php?action=update&idpersonne='.$personne['idp
    ersonne'];?>"> modifier </a></td>
<td> <a href="<?php echo
    URL.'index.php?action=delete&idpersonne='.$personne['idp
    ersonne'];?>"> supprimer </a></td>
</tr>
<?php endforeach;?>
```

accueil_view.php

```
<form action='<?php echo
  URL.'index.php?action=create' ?>'
  method='post' >
<p>
Nom <input type="text" name="nom"/>
Prénom <input type="text" name="prenom"/>
<input type="submit" value=""
  class="mesSubmits"/>
</p>
</form>
```

Conclusion

- Une approche de conception plus propre MVC
- Un mini-framework qui permet de travailler en équipe
- Toujours pas de sécurité ou d'indépendance vis-à-vis de la base de donnée.

=>

Utiliser un vrai framework

La suite

- Présentation d'un framework léger
CodeIgniter
- Présentation de l'ORM doctrine