PROGRAMMATION WEB

PHP MVC (2)

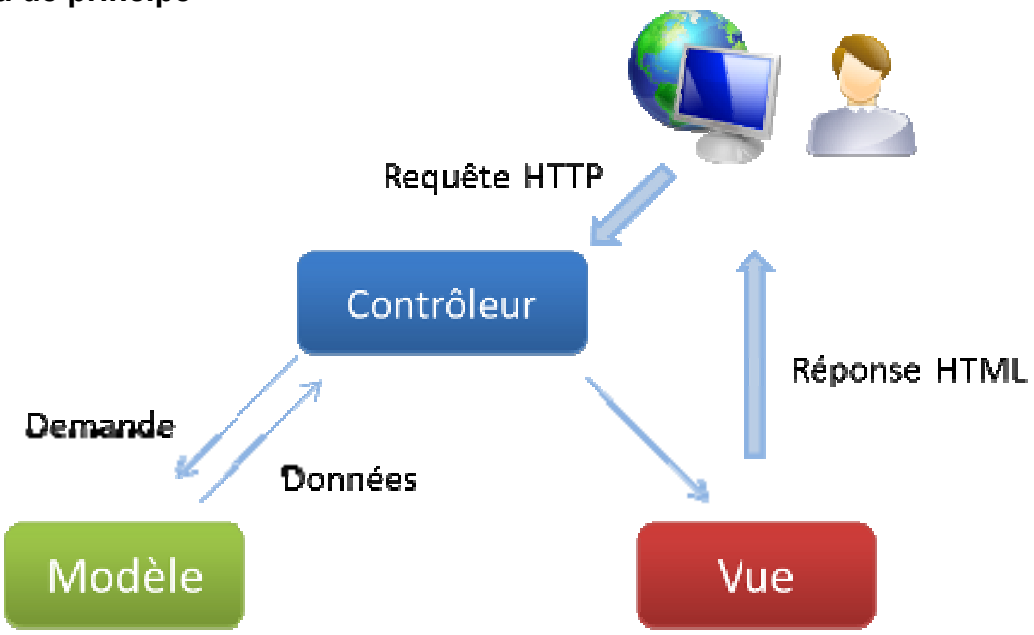
1. Rappel MVC

Plus le site grandit, en pages, en fonctionnalités ➔ plus le nombre de fichiers augmente et plus le nombre de lignes de code augmente.
➔ il faut organiser et structurer son code, ses fichiers, ses dossiers.

Signification MVC

M = Modèle
V = Vue
C = Contrôleur

Schéma de principe



Explication des différentes composantes

La couche Contrôleur :

- elle reçoit la requête http de l'internaute
- elle traite la demande :
- elle récupère les données nécessaires : elle s'adresse à la couche Modèle

La couche Modèle contient les classes, et en particulier les outils nécessaires à interagir avec la bdd

- elle récupère les données en bdd
- elle les retourne au contrôleur qui en a besoin

Le Contrôleur réceptionne les données :

- réalise les traitements éventuels (calculs…)
- fournit les données à afficher à la couche Vue

La vue

- réceptionne les données
- fabrique le code html (+ css + js)
- retourne le résultat à l'internaute (html + css + js)

MISE EN ŒUVRE

Deux façons:
1) "Manuellement" : on code tout, selon bien sûr une organisation correspondant au modèle MVC
2) Utilisation d'un framework qui fournit une cadre de travail (ensemble de classes)

Même "manuellement" deux approches :
1) organisation simplifiée (cf poly MVC Introduction)
2) organisation dite "avancée"

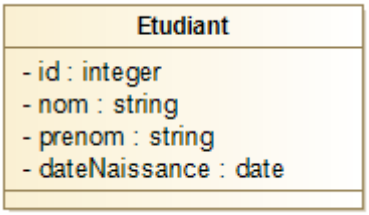
1. Mise en œuvre manuelle d'un MVC avancé

(le site de base est celui utilisé dans MVC - Introduction)
Nous allons améliorer le codage de l'architecture MVC créée pour le site de gestion des étudiants..

1.1. La couche model

La couche Model mise en place s'appuie sur plusieurs class :

- une class technique pour la connexion à la base de données.
- pour chaque class métier (ici Etudiant seulement) :
  - une class métier propre (cf diagramme de class ci-dessous)
  - une class technique permettant de faire le lien (on parle de mapping) entre la class métier (côté application) et la table correspondante de la base de données (côté SGBD).



- La classe **`EtudiantManager`** qui contiendra les méthodes :
  - **`getLesEtudiants( )`** qui retourne un tableau d'objets de type Etudiant qui correspond à tous les étudiants présents dans la bdd.
  - **`getUnEtudiantById(int $id)`** qui retourne un objet de type Etudiant qui correspond à l'étudiant dont l'id est passé en paramètre.

Rappel: ces méthodes sont statiques et seront donc appelées par la syntaxe `<class>::<method>`
Pas de changement dans notre couche Model

1.2. La couche Vue

Rappel
La vue n'effectue pas de traitements (pas de calculs). Elle se contente de préparer le code qui affiche la page : principalement du html.
C'est tout de même elle qui référence les fichiers css et js à envoyer.
Elle utilise quelques fonctions php pour intégrer au html les données récupérées par le controller. Exemple : données issues de traitements, de la base de données etc…
Pas de changement dans notre couche View

1.3. Le controller

Rappel
Le controller est un fichier en php qui gère les traitements :

- *réception et vérification des éventuelles paramètres, données de formulaires etc…*
- *récupération en base de données des données nécessaires*
- *calculs divers*

Et puis il appellera la vue pour rendre l'affichage.

Suivant l'importance du site (ou d'autres critères comme les habitudes de programmation et d'organisation du code) il est possible d'envisager plusieurs niveaux de controller :

- *un controller unique pour tout le site (uniquement adapté aux petits sites)*
- *un controller pour chaque item de model (ici un controller pour **`etudiant`** qui prendra en charge toutes les actions concernant les étudiants)*
- *et si le code est vraiment imposant, on peut même envisager un controller pour chaque action de chaque model (exemple: un controller spécialisé dans l'affichage des listes d'étudiants…)*

Implémentons ici une solution communément répandue : un controller par model.
Un controller est une class.

Deux aspects sont à prendre en considération :

- *dans le dossier controller il n'y aura pas que **`EtudiantController.php`** => il y aura aussi un controller pour les autres items du model : les utilisateurs par exemple (**`UserController`**) et bien d'autres items. Il nous faut donc un moyen de savoir quel controller est concerné par la requête http de l'internaute.*
- *chaque controller devra savoir gérer différentes actions, par exemple pour **`EtudiantController`**, il doit gérer les listes d'étudiants, les formulaires de saisie ou de modification, la fiche détails d'un étudiant etc… lorsqu'on se sera adressé au bon controller, il faudra lui préciser quelle action est requise.*

Nota:
On appelle "action" une fonction du controller. Une action correspond généralement à une page web (mais pas forcément).

*Notre controller prend maintenant la forme d'une classe, exemple avec **`EtudiantController.php`** :*

```
class EtudiantController {
    public static function readAll( ) {
        $lesEtudiants = EtudiantManager::getLesEtudiants( ); // appel du model
        require_once ('../view/etudiant/list.php); // appel de la vue
    }

    // autre action
    public static function…

}
```

Voici une liste d'actions type que nous allons bientôt ajouter au EtudiantController :

- *afficher tous les étudiants : action **`readAll`***
- *afficher les détails d'un étudiant : action **`read`***
- *afficher le formulaire de création d'un étudiant : action **`create`***
- *créer un étudiant dans la base de données et afficher un message de confirmation : action **`created`***
- *supprimer un étudiant et afficher un message de confirmation : action **`delete`***

Pas de changement non plus sur ces principes.

2. Appeler la bonne page

C'est ici que commence notre travail.
La requête http reçue de l'internaute devra avoir la forme :
http://127.0.0.1/1sio-php-mvc-2/index.php?controller=Etudiant&action=readAll&filtre_section=SLAM

Il faut donc récupérer :

- la valeur du controller
- la valeur de l'action
- les noms et valeurs des éventuels autres paramètres

En vous servant des extraits de code ci-dessous, mettez au point le décodage de l'url

```php
define('DEFAULT_CONTROLLER', 'accueil');
define('DEFAULT_ACTION', 'index');

// récupère les paramètres de l'url

if(isset($_GET) && !empty($_GET)){
    // extrait les valeurs du tableau $_GET
    extract($_GET);
} else {
    $controller = DEFAULT_CONTROLLER;
    $action = DEFAULT_ACTION;
}

$params = array();
echo 'Boucle $_GET'.'<br />';
foreach($_GET as $key => $value){
    if(($key != 'controller') && ($key != 'action')){
        $params[$key] = $value;
    }
}

// teste la bonne lecture
print_r($controller);
print_r($action);
foreach($params as $key => $value){
    print_r('<br />'.$key.' => '.$value);
}
```

2.1. Le routage

On appelle routage le traitement qui consiste à décoder l'url, et à invoquer le bon controller et la bonne action.
Dans certains exemples didactiques ce fichier se nomme routeur.php, mais pourquoi ne pas l'inclure tout simplement dans index.php (si celui-ci ne s'en trouve pas surchargé, illisible) puisque ce sera le point d'accès de l'internaute (toute requête http passera par ce fichier).

Ce fichier s'occupera des traitements :

- réception et décodage de l'url requêtée
- détermination et vérification du controller désiré
- détermination et vérification de l'action désirée
- traitement éventuel des erreurs

EXERCICE
Compléter le code du fichier index.php

```php
<?php

    /**
    * /index.php
    * Page d'accueil
    *
    *
    * @auteur T. Savary
    * @date 04/2022
    */

    // autochargement des class
    require_once __DIR__.'/autoload.php';

    // enregistrement de la racine du site
    define('ROOT', __DIR__);
    define('DEFAULT_CONTROLLER', 'accueil');
    define('DEFAULT_ACTION', 'index');

    // récupère les paramètres de l'url
    if(isset($_GET) && !empty($_GET)){
        // extrait les valeurs du tableau $_GET
        extract($_GET);
    } else {
        // s'il n' en a pas alors c'est la page par défaut
        $controller = DEFAULT_CONTROLLER;
        $action = DEFAULT_ACTION;
    }

    // s'il y a des paramètres en + de controller et action :
    // ils sont stockés dans un array nommé $params
```

```
    $params = array();
    foreach($_GET as $key => $value){
        if(($key != 'controller') && ($key != 'action')){
            $params[$key] = $value;
        }
    }

    // teste la bonne lecture des paramètres de $_GET
    print_r($controller);
    print_r($action);
    foreach($params as $key => $value){
        print_r('<br />'.$key.' => '.$value);
    }

    // route vers le controller et l\'action
    // vérifie que le controller requêté existe
    // sinon page d'erreur
    // idem pour l'action
    $controller .= 'Controller'; // la variable controller ne contenait qu'une part
ie du nom du fichier, on le complète
    $filename = ROOT.'/controller/'.$controller.'.php';
    print_r('filename = '.$filename);
    if(file_exists($filename)){
        // le fichier du controller existe
        // inclut le fichier de class du controller
        require_once ROOT.'/controller/'.$controller.'.php';
        if(method_exists($controller, $action)){
            print_r('Le controller et l\'action existe');
            // appelle la méthode correspondant à l\'action
            // si dans l\'url, en plus des paramètres controller et action, il y a
d\'autres paramètres alors ils dovent être passés au contrôleur
            $controller::$action($params);
        } else {
            // la méthode correspondant à l\'action n\'existe pas
            print_r('L\'action n\'existe pas');
        }
    } else {
        // le fichier du controller n\'existe pas
        print_r('Le controller n\'existe pas');
    }

?>
```

2.2. Notion de réécriture d'url

https://www.webrankinfo.com/dossiers/techniques/tutoriel-url-rewriting
Pour améliorer son référencement naturel, rien de tel que la réécriture d'urls.
Il n'est d'ailleurs pas très pratique, ni très sécuritaire de laisser des urls de la forme
http://www.mon-site.fr/controller=accueil&action=index&id=5
Ces urls peuvent être normalisées sous la forme :
http://www.mon-site.fr/accueil/index/5

Du coup il faut "traduire" de l'une à l'autre : on parle tout simplement de réécrire les urls.
Ceci est réalisé par un fichier **.htaccess** placé à la racine du site.

Les directives possibles dans un fichier **.htaccess** sont nombreuses. Un fichier **.htaccess** peut servir à bien d'autres choses que de réécrire des urls.
https://www.mauricelargeron.com/parametrer-les-acces-a-son-serveur/

Voici le code et les explications du fichier .htaccess à créer pour notre site :
.htaccess

```
RewriteEngine on
RewriteRule ^(.*)/(.*)/(.*)$  index.php?controller=$1&action=$2&params=$3 [QSA,L]
```

RewriteEngine on active le module de réécriture
RewriteRule définit 1 règle de réécriture. Plusieurs règles peuvent se succéder dans le même fichier **.htaccess**
La syntaxe générale est **Rewrite Rule url_reçue url_réécrite**
La syntaxe ne fait pas partie de nos objectifs: si besoin il est relativement facile de trouver les bonnes infos.
Les directives entre crochets sont optionnelles [QSA, L]. Cherchez leur signification sur Internet.

Réaliser un petit projet de test (uniquement un fichier **index.php** et un fichier **.htaccess**
Exemple de fichier test **index.php**

```php
<?php
    print_r('Test réécriture URL');
    $url = $_SERVER['REQUEST_URI'];
    print_r('<br />'.$url);

    extract($_GET);
    print_r('<br />controller = '.$controller);
    print_r('<br />action = '.$action);
    print_r('<br />params = '.$params);

?>
```

3. Une classe Controller

- Codons une classe technique **Controller** pour mutualiser des fonctionnalités propres à tous nos controllers (**EtudiantController** etc…).

Pour faire un exemple simple, nous ne coderons qu'une méthode nommée **render( )** et qui permettra à notre contrôleur d'invoquer la vue adéquate.

Controller.php

```php
<?php

/**
 * /controller/Controller.php
 *
 * class technique pour définir les membres communs aux controllers
 *
 * @auteur T. Savary
 * @date 04/2022
 */

    class Controller {
        public static function render($params){
            print_r('<br />'.$params);
        }
    }
?>
```

Définir que notre **EtudiantController** hérite de la class **Controller**

```php
class EtudiantController extends Controller
```

Remplacer l'appel de la vue **require_once**… en utilisant la méthode **render( )** dans **EtudiantController** (bien sûr dans le **require_once** est toujours présent, mais cette fois dans la méthode **render( )**)

EXEMPLE DE CODE FINAL

Class Controller

```php
<?php

/**
 * /controller/Controller.php
 *
 * class technique pour définir les membres communs aux controllers
 *
 * @auteur T. Savary
 * @date 04/2022
 */

    class Controller {
        public static function render($view, $params){
            extract($params);
            require_once $view;
        }
    }
?>
```

Class EtudiantController

```php
<?php

/**
 * /controller/EtudiantController.php
 *
 * Contrôleur pour l'entité Etudiant
 *
 * @author T. Savary
 * @date 04/2022
 */

    class EtudiantController extends Controller {

        /**
         * Action qui affiche la table de tous les étudiants
         * params : tableau des paramètres
         * filtre_section permet de choisir entre Tous, SISR ou SLAM
         */
        public static function readAll($params){

                /**
```

```
                * Teste le filtre section
                */
                if(empty($params['filtre_section'])){
                    $filtre = 'tous';
                }else{
                    $filtre = $params['filtre_section'];
                }

                /**
                * récupère les étudiants de la bdd
                */

                switch($filtre)
                {
                    case 'tous':
                        $lesEtudiants = EtudiantManager::getLesEtudiants();
                        break;
                    case 'sisr':
                        $uneSection = new Section();
                        $uneSection->setLibelleSection('1ère année');
                        $lesEtudiants = EtudiantManager::getLesEtudiantsBySection($
uneSection);
                        break;
                    case 'slam':
                        $uneSection = new Section();
                        $uneSection->setLibelleSection('2ème année');
                        $lesEtudiants = EtudiantManager::getLesEtudiantsBySection($
uneSection);
                        break;
                    default :
                        $lesEtudiants = EtudiantManager::getLesEtudiants();
                }

            // appelle la vue
            $view = ROOT.'/view/etudiant/list.php';
            $params = array();
            $params['filtre'] = $filtre;
            $params['etudiants'] = $lesEtudiants;
            self::render($view, $params);
        }
    }
?>
```