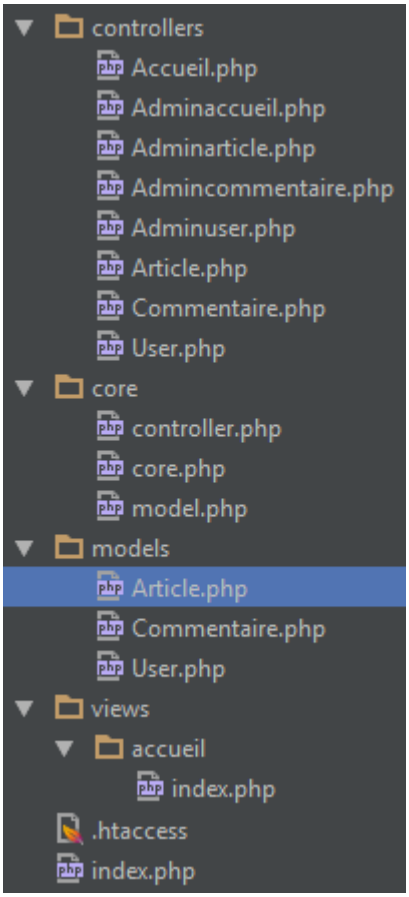
MVC

Arborescence



.htaccess

La réécriture d'URL nous permettra d'avoir des URL plus propres.

Le paramètre QSA (Query String Append) permet de garder les données $_GET après la réécriture, par exemple :

http://www.monsite.com/controller/action?id=8 (avec QSA)

http://www.monsite.com/controller/action (sans QSA, on perd id=8)

On va faire pointer toutes les requêtes sur notre index.php qui sera notre dispatcher, il se chargera d'appeler les controller et méthodes nécessaires.

```
RewriteEngine On
RewriteRule ^([a-zA-Z0-9\-\_\/]*)$ index.php?p=$1 [L,QSA]
```

Index.php

```
<?php
//on définit les constantes qui nous permettront d'inclure nos fichiers
```

```php
n'importe où
define('WEBROOT', str_replace('index.php', '', $_SERVER['SCRIPT_NAME']));
define('ROOT', str_replace('index.php', '', $_SERVER['SCRIPT_FILENAME']));
//on inclu le coeur de notre MVC
require(ROOT . 'core/core.php');
require(ROOT . 'core/controller.php');
require(ROOT . 'core/model.php');
//Si on a une URL du type www.monsite.com/controller/action on récupère le
controller et l'action
if(isset($_GET['p']) && !empty($_GET['p'])){
   $param = explode('/', $_GET['p']);
   $controller = $param[0];
   $action = $param[1];
}else{
   //sinon on défini un controller et une action par défaut
   $controller = 'Accueil';
   $action = 'index';
}
$controller .= 'Controller';
//on instancie notre controller, $controller est une variable nommée qui vaudra
par exemple AccueilControlelr, ça reviendrait donc à faire new
AccueilController
$cont = new $controller();
if(method_exists($controller, $action)){
   //comme pour le contrôleur, l'action est une variable nommée, ça équivaut à
faire $cont->index(); par exemple
   $cont->$action();
}else{
   echo '404 - method not found';
   die();
}
```

Core : core.php

Le fichier /core/core.php va nous permettre de stocker les constantes et configurations (accès BDD, débug, autoload)

```php
<?php
define('DEBUG', true);
if(DEBUG){
   error_reporting(-1);
   ini_set('display_errors', '1');
}
//DATABASE
define('DB_HOST', 'localhost');
define('DB_USER', 'world');
define('DB_PASS', 'miwworld');
define('DB_DATABASE', 'miw_world');
//include automatique des classes
//cette fonction sera appelée à chaque new MaClasse si MaClasse n'a pas déjà
été incluse
spl_autoload_register(function ($class) {
   if(preg_match("/[a-zA-Z0-9\_]+Controller$/", $class)){
      //remove the "controller" part of the class name
      $controller = ucfirst(substr($class, 0, -10));
      if(file_exists(ROOT.'controllers/' . $controller . '.php'))
         require ROOT.'controllers/' . $controller . '.php';
      else{
         echo '404 - Controller ' . $class . ' not found';
```

```
            die();
        }
    }else if(preg_match("/[a-zA-Z0-9\_]+Model$/", $class)){
        //remove the "controller" part of the class name
        $model = substr($class, 0, -5);
        if(file_exists(ROOT.'models/' . $model . '.php'))
            require ROOT.'models/' . $model . '.php';
        else{
            echo '404 - Model ' . $class . ' not found';
            die();
        }
    }
});
function p($data = null){
    echo '<pre>';
    var_dump($data);
    echo '</pre>';
}
function d($data = null){
    p($data);
    die();
}
```

Core : controller.php

Ce fichier définit le fonctionnement de tous les controller que l'on va utiliser. Nous pourrons séparer le traitement (dans les controller) de l'affichage (dans les vues).

```
<?php
class Controller{
    public $vars = array('menu'=>array('adminarticle'=>'Articles',
'admincommentaire'=>'Commentaires', 'adminuser'=>'Users'));
    function set($d){
        $this->vars = array_merge($this->vars, $d);
    }
    function render($view){
        extract($this->vars);
        //remove the "controller" part of the class name
        $controller = strtolower(substr(get_class($this), 0, -10));
        if(file_exists(ROOT.'views/'.$controller.'/'.$view.'.php')){
            require(ROOT.'views/'.$controller.'/'.$view.'.php');
        }else{
            echo '404 - view not found';
            die();
        }
    }
}
```

On définit chaque variable passée à la vue en utilisant la fonction « set » qui se contente de remplir un tableau de variable.

Lors du « render », la fonction extract récupère tous les éléments du tableau $vars et rend chaque élément accessible à la vue sous forme de variable.

Par exemple :

```
//si après utilisation de "set($d)"
$vars = array(
    'variable1'=>'Contenu de ma variable',
```

```php
    'autrevar'=>array(0=>'elem 1', 1=>'elem 2')
);
//alors dans la vue je pourais faire
echo $variable1;
echo $autrevar[0];
echo $autrevar[1];
```

Core : model.php

La class Model est ce qu'on appel un singleton, on a défini une propriété statique $instance et une méthode statique getInstace() .

Lorsque j'ai besoin d'un accès à la base, au lieu de faire new Model() à chaque fois, je n'ai qu'à appeler Model::getInstance() et, si une instance existe la fonction me la retourne, sinon elle créée une nouvelle instance et me la retourne.

```php
<?php

class Model{
    //Instance de la classe PDO
    public $bdd = null;
    //Instance de la classe Model
    private static $instance = null;
    //Constante: nom d'utilisateur de la bdd
    const DEFAULT_SQL_USER = DB_USER;
    //Constante: hôte de la bdd
    const DEFAULT_SQL_HOST = DB_HOST;
    //Constante: hôte de la bdd
    const DEFAULT_SQL_PASS = DB_PASS;
    //Constante: nom de la bdd
    const DEFAULT_SQL_DTB = DB_DATABASE;
    public function __construct(){
        if(DEBUG)
            $this->bdd = new PDO('mysql:dbname='.self::DEFAULT_SQL_DTB.';host='.
self::DEFAULT_SQL_HOST
                ,self::DEFAULT_SQL_USER,self::DEFAULT_SQL_PASS
                , array(PDO::ATTR_ERRMODE=>PDO::ERRMODE_WARNING));
        else
            $this->bdd = new PDO('mysql:dbname='.self::DEFAULT_SQL_DTB.';host='.
self::DEFAULT_SQL_HOST
                ,self::DEFAULT_SQL_USER,self::DEFAULT_SQL_PASS);
    }
    //Crée et retourne l'objet Model
    public static function getInstance(){
        if(is_null(self::$instance))
            self::$instance = new Model();
        return self::$instance;
    }
}
```

Controller : Accueil.php

Tous les controller seront construits pareil :

- Ils doivent tous être dans le dossier controller
- Le premier caractère du fichier doit être en majuscule

- Le nom du fichier ne doit pas porter le suffixe « Controller »
- Le nom de la classe doit être le nom du fichier suivi du suffixe « Controller »
- Chaque controller doit étendre la classe Controller (pour pouvoir utiliser set, render et donc les vue)
- AUCUN echo, affichage, ou HTML de quelque sorte dans les controllers

```php
<?php
class AccueilController extends Controller{
   public function index(){
      //Je défini les variables auxquels j'aurai accès dans ma vue
      $this->set(array('nom'=>'Mon nom', 'prenom'=>'Mon prénom'));
      //J'appelle ma vue (essayer au possible d'avoir des vues qui portent le même nom que la méthode
      $this->render('index');
   }
}
```

Vue : accueil/index.php

Les vues sont comme leur nom l'indique les fichiers où l'on va procéder à l'affichage. Aucun traitement PHP n'y est autorisé, aucune fonction PHP ne doit y être déclarée, uniquement du echo, des if et des boucles

```php
<h1>Accueil - index</h1>
Je m'appelle <?php echo $prenom ?> <?php echo $nom ?>.
```

Model : article.php

Pour chaque table en bdd son modèle, les modèles nous permettent d'interagir avec la base de façon simple et en objet grâce aux fonctions de CRUD.

Tous les modèles sont construits pareil :

- Ils doivent tous être dans le dossier model
- Le premier caractère du fichier doit être en majuscule
- Le nom du fichier ne doit pas porter le suffixe «Model»
- Le nom de la classe doit être le nom du fichier suivi du suffixe «Model»
- Chaque modèle doit étendre la classe Model (pour avoir accès à la connexion bdd)
- AUCUN echo, affichage, ou HTML de quelque sorte dans les modèles

```php
<?php
Class ArticleModel extends Model{
   public $id;
   public $titre;
   public $contenu;
   public $id_user;
   public $datetime;
   public function __construct($id=null) {
      parent::__construct();
      //Le reste ne change pas
   }
```

```
   //Dans les fonction CRUD on utilisera $this->bdd (défini dans Model)au lieu
de $bdd
   //Pour une fonction static, on récupère la connexion via le design pattern
singleton, en utilisant la méthode getInstance
   public static function getAll($user_id = null){
      $model = self::getInstance();
      if(!is_null($user_id))
         $req = $model->bdd->prepare('SELECT id FROM article  WHERE id_user =
'.$user_id);
      else
         $req = $model->bdd->prepare('SELECT id FROM article');
      $req->execute();
      $articles = array();
      while($row = $req->fetch()){
         $article = new ArticleModel($row['id']);
         $articles[] = $article;
      }
      return $articles;
   }
}
```