Architectures et technologies WEB

Introduction aux technologies WEB

V01.00 17/08/2016 - MAURICE

Déroulement du cours

- *Chapitre 1 : Préambule*
- *Chapitre 1 : Les technologies WEB*
- *Chapitre 2 : La Programmation cliente*
- *Chapitre 3 : Le modèle MVC / Les templates*
- Chapitre 4 : La sécurité
- **Chapitre 5 : Les bases de PHP / Le Contrôleur**
- Chapitre 6 : PHP 2ème partie / ORM
- Chapitre 7 : SOA / ROA
- Chapitre 8 : Le référencement

Plan de la séance

- Chap 5 : PHP
  - Présentation – Introduction PHP
  - Le Controleur

Introduction rapide à PHP

PHP

- Créé en 1994 par Rasmus Lerdorf
- PHP signifiait Personnal Home Page.
- Dérivé du C et du Perl
- Extensible grâce à de nombreux modules
- Code source ouvert.
- Très répandu sur le web
- En 1997, php est réécrit par Zeev Suraski et Andi Gutmans pour donner la version 3
- Il s'appelle désormais PHP : Hypertext Preprocessor

Présentation

- La version actuelle est la version 7.
- Précédemment la v5 sortie le 13 juillet 2004 à introduit un véritable modèle objet, une gestion des erreurs fondée sur le modèle des exceptions, ainsi que des fonctionnalités de gestion pour les entreprises.
- PHP 5 apporte beaucoup de nouveautés, telles que :
  - le support de SQLite, qui est un système légé de gestion de bases de données embarquées,
  - le moyen de manipuler des fichiers et des structures XML
  - La gestion des exceptions

Intégration d'un script

```
<?php … ?>

Exemple:
<html>
<body>
<?php
echo “Bonjour”;
?>
</body>
</html>
```

Commentaires

Un script php se commente comme en C :

```
// commentaire de fin de ligne
/* commentaire
sur plusieurs
lignes */
# commentaire de fin de ligne comme en Shell
```

Les variables (1/6)

- Typage des variables implicite
- Pas de déclaration au préalable.
- Les identificateurs de variable sont précédés du symbole « $ »
- Les variables peuvent être de type
  - entier (integer),
  - réel (double),
  - chaîne de caractères (string),
  - tableau (array),
  - objet (object),
  - booléen (boolean).
- Possibilité de convertir une variable par un cast explicite Exemple :
  - `$str = "12";`
  - `$nbr = (int)$str;`

Les variables (2/6)

- Quelques fonctions :
  - empty($var) : renvoie vrai si la variable est vide
  - isset($var) : renvoie vrai si la variable existe
  - unset($var) : détruit une variable
  - gettype($var) : retourne le type de la variable
  - settype($var, "type") : convertit la variable en type type (cast)
  - is_long(), is_double(), is_string(), is_array(), is_object(), is_bool(), is_float(), is_numeric(), is_integer(), is_int()…

Les variables (3/6)

- La portée d'une variable est limitée au bloc dans lequel elle a été créée :
- Une variable locale à une fonction n'est pas connue dans le reste du programme.
- Tout comme une variable du programme n'est pas connue dans une fonction.
- Une variable créée dans un bloc n'est pas connue dans les autres blocs, mêmes supérieurs.

Les Variables (4/6)

- Les Variables pre definies
  - `$_SERVER` est une variable auto_globale de PHP, qui contient toutes les informations relatives au serveur web.
  - Exemple : Afficher le contenu d'une variable (élément de tableau)
  - `<?php echo $_SERVER["HTTP_USER_AGENT"]; ?>`
  - Un résultat possible du script pourra alors être :
  - Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)

- Variables Apache
  - Elles sont spécifiques et créées par le serveur Apache.
  - `SERVER_NAME` : nom du serveur hôte
  - `SERVER_PROTOCOL`: Nom et révision du protocole de communication
  - `REQUEST_METHOD` : Méthode de requête utilisée pour accéder à la page
  - `QUERY_STRING` : La chaîne de requête,
  - `DOCUMENT_ROOT` : La racine sous laquelle le script courant est exécuté

Les Variables (5/6)

- Variables PHP
  - Elles sont créées par PHP
  - `argv` : Tableau des arguments passées au script.
  - `argc` : Contient le nombre de paramètres de la ligne de commande passés au script
  - `PHP_SELF` : Le nom du fichier du script en cour d'éxécution,
  - `HTTP_COOKIE_VARS` : Un tableau associatif
  - `HTTP_GET_VARS` : Un tableau associatif
  - `HTTP_POST_VARS` : Un tableau associatif
- Variables d'environnement
  - Ces variables sont importées depuis l'environnement sous lequel PHP fonctionne.
  - EX : `PATH`

Les Variables (6/6)

- `$_POST` variable auto_globale qui contient toutes les données envoyées par la méthode POST.
- `$_GET`, une autre auto-globale pour les données envoyées par la méthode GET.
- `$_REQUEST`, si vous ne souhaitez pas vous embarrasser de la méthode utilisée. Elle contient un mélange des données de `GET`, `POST`, `COOKIE` et `FILE`.

```
<?
foreach ($_SERVER as $key => $value) echo ($key . " : " . $value . "<br>");
foreach ($_COOKIE as $key => $value) echo ($key . " : " . $value . "<br>");
foreach ($_POST as $key => $value) echo ($key . " : " . $value . "<br>");
foreach ($_GET as $key => $value) echo ($key . " : " . $value . "<br>");
foreach ($_REQUEST as $key => $value) echo ($key . " : " . $value . "<br>");
?>
```

Les opérateurs

- Opérateurs arithmétiques :

+ (addition), - (soustraction), * (multiplié), / (divisé), % (modulo), ++ (incrément), --(décrément). Ces deux derniers peuvent être pré ou post fixés

- Opérateurs d'assignement :

= (affectation), *= ($x*=$y équivalent à $x=$x*$y), /=, +=, -=, %=

- Opérateurs logiques :

and, && (et), or, || (ou), xor (ou exclusif), ! (non)

- Opérateurs de comparaison :

== (égalité), < (inférieur strict), <= (inférieur large), >, >=, != (différence)

Constantes

- Les constantes ne portent pas le symbole $ (dollars)
- Elles ne sont pas modifiables.

```
define(“var”,valeur)
```

- Exemple 1 :

```
define(“author”,”Foobar”);
echo author;
```

Références

- La référence à une variable s'effectue avec l'opérateur &
- Exemple :

```
$toto = 100;
$foobar = &$toto;
$toto++;
echo $foobar;
```

Structures de contrôle

- if/ if .. else / if .. elseif …
- while / do .. while
- for / foreach
- switch
- break / continue

Bloc d'instructions

- Est délimité par des accolades { } : les instructions à l'intérieur du bloc sont à exécuter séquentiellement

```
{
instruction_1
instruction_2
...
instruction_n
}
```

- les n instructions sont simples ou des blocs ou des instructions if-else, for-do, do-while .....
- L'exécution du bloc d'instructions consiste en l'exécution séquentielle (l'une après l'autre, dans l'ordre) des n instructions.
- Un bloc d'instructions peut apparaitre partout où une instruction apparaît

Quelques fct Mathématiques

- `abs($x)` : valeur absolue
- `ceil($x)` : arrondi supérieur
- `floor($x)` : arrondi inférieur
- `pow($x,$y)` : x exposant y
- `round($x,$i)` : arrondi de x à la ième décimale
- `max($a, $b, $c …)` : retourne l'argument de valeur maximum
- `pi()` : retourne la valeur de Pi

- Et aussi : `cos, sin, tan, exp, log, min, pi, sqrt…`

- Quelques constantes :
  - `M_PI` : valeur de pi (3.14159265358979323846)
  - `M_E` : valeur de e (2.7182818284590452354)

Booléens

- Les variables booléennes prennent pour valeurs `TRUE` (vrai) et `FALSE` (faux).
- Les valeurs considérée comme false sont :
  - Une valeur entière nulle
  - Toute chaîne de caractères vide `""`.
  - Ou encore comme les chaînes `"0"` et `'0'`
  - l'entier 0

Chaînes de caractères

- Une variable chaîne de caractères n'est pas limitée en nombre de caractères.
- Elle est toujours délimitée par des simples quotes ou des doubles quotes.
- Exemples :
  - `$nom = “HUGO”;`
  - `$prenom = 'Victor';`
- Les doubles quotes permettent l'évaluation des variables et caractères spéciaux contenus dans la chaîne.

Chaînes de caractères

- Opérateur de concaténation de chaînes : . (point)
- Exemple :

```
$foo = "Hello";
$bar = "Word";
echo $foo.$bar;
```

Chaînes de caractères

- Affichage d'une chaîne avec echo:
- Exemples:

```
echo 'Hello Word.';
echo "Hello ${name}\n";
echo "Nom : ", $name;
echo("Bonjour");
```

Chaînes de caractères

- Quelques fonctions:
  - `strlen($str)` : retourne le nombre de caractères d'une chaîne
  - `strtolower($str)` : conversion en minuscules
  - `strtoupper($str)` : conversion en majuscules
  - `trim($str)` : suppression des espaces de début et de fin de chaîne
  - `substr($str,$i,$j)` : retourne une sous chaîne de $str de taille $j et débutant à la position $i
  - `strnatcmp($str1,$str2)` : comparaison de 2 chaînes
  - `addslashes($str)` : déspécialise les caractères spéciaux (', ", \)
  - `ord($char)` : retourne la valeur ASCII du caractère $char

Chaînes de caractères

- On peut délimiter les chaînes de caractères avec la syntaxe Here-doc.
- Exemple :

```
$essai = <<<EOD
Ma chaîne "essai"
sur plusieurs lignes.
EOD;
echo $essai;
```

Affichage

- Les fonctions d'affichage :
- `echo()` : écriture dans le navigateur
- `print()` : écriture dans le navigateur
- `printf([$format, $arg1, $arg2])` : écriture formatée

Tableaux

- Une variable tableau est de type array.
- Les éléments d'un tableau peuvent être de types différents et sont séparés d'une virgule.
- Un tableau peut être initialisé avec la syntaxe array.
- Exemple :
  - `$tab_colors = array('red', 'yellow', 'blue', 'white');`
  - `$tab = array('foobar', 2002, 20.5, $name);`

Tableaux

Parcours d'un tableau :

```
$tab = array('Hugo', 'Jean', 'Mario');
```

Exemple :

```
$i=0;
while($i < count($tab)) {
      echo $tab[$i].'\n';
   $i++;
}

foreach($tab as $elem) {
  echo $elem."\n";
}
```

Tableaux

- Quelques fonctions:
  - `count($tab), sizeof` : retournent le nombre d'éléments du tableau
  - `in_array($var,$tab)` : dit si la valeur de $var existe dans le tableau $tab
  - `list($var1,$var2…)` : transforme une liste de variables en tableau
  - `sort($tab)` : trie alphanumérique les éléments du tableau
  - `rsort($tab)` : trie alphanumérique inverse les éléments du tableau

Tableaux associatifs

- Un tableau associatif est appelé aussi dictionnaire ou hashtable.
- On associe à chacun de ses éléments une clé dont la valeur est de type chaîne de caractères.

Exemple :

```
$personne = array("Nom" => "César", "Prénom" => "Jules");
$personne["Nom"] = "César";
$personne["Prénom"] = "Jules";
```

Tableaux associatifs

- Parcours d'un tableau associatif.

```
$personne = array("Nom" => "César", "Prénom" =>
"Jules");
```

- Exemple :

```
foreach($personne as $elem) {
  echo $elem;
}
```

Tableaux associatifs

- Quelques fonctions :
  - `array_count_values($tab)` : retourne un tableau contenant les valeurs du tableau $tab comme clés et leurs fréquence comme valeur
  - `array_keys($tab)` : retourne un tableau contenant les clés du tableau associatif $tab
  - `array_values($tab)` : retourne un tableau contenant les valeurs du tableau associatif $tab
  - `array_search($val,$tab)` : retourne la clé associée à la valeur $val

Expression Régulières(1/6)

```
int preg_match ( string $pattern , string $subject [, array &$matches] )
```

Test de chaines par rapport à un pattern :

```
preg_match('boy', 'cowboys');
preg_match('^boy', 'cowboys');
preg_match('^cow', 'cowboys');
preg_match('cow$', 'cowboys');
preg_match('boys$', 'cowboys');
```

true
false
true
false
true

Expression Régulières(2/6)

- *pattern* : Le masque à chercher, sous la forme d'une chaine
- *subject* : La chaîne d'entrée.
- *matches* : Si *matches* est fourni, contient les résultats de la recherche.
  - *$matches[0]* contiendra le texte qui satisfait le masque complet,
  - *$matches[1]* contiendra le texte qui satisfait la première parenthèse capturante, etc.

Expression Régulières(4/6)

```
preg_match('^[a-z]*$', 'cowboy');
preg_match('^[a-z]*$', 'cowboy123');
preg_match('^[a-z]*[0-9]*$', 'cowboy123');
preg_match('^[a-z]{6}$', 'cowboy');
preg_match('^[a-z]{4}$', 'cowboy');
preg_match('^[bcowy]*$', 'cowboy');
```

true
false
true
true
false
true

Expression Régulières(5/6)

```
$email = “minoura@eecs.orst.edu”
preg_match('(.*)@(.*)', $email, $arr);

echo “User: $arr[0]\n”;
echo “Host: $arr[1]\n”;



User: minoura
Host: eecs.orst.edu
```

Expression Régulières(6/6)

```
^   - start of line, or negation of
      characters
$   - end of line
.   - any character
[] - range, or set of characters
() - grouping
*   - any number of times
+   - at least once
?   - exactly once
{n, m} - match n through m occurrences
```

Fonctions (1/4)

- Par def Les paramètres sont passés par copie
- les résultats sont retournés par copie, .
- Même sans paramètre, un entête de fonction doit porter des parenthèses ( ).
- Les différents arguments sont séparés par une virgule délimité par des accolades { }.

- Exemple :

```
function afficher($str1, $str2) {   // passage de deux paramètres
    echo "$str1, $str2";
}
```

Fonctions (2/4)

- L'appel à une fonction peut ne pas respecter son prototypage (nombre de paramètres).
- Les identificateurs de fonctions sont insensibles à la casse.
- Exemple :

```
function mafonction($toto) {
	$toto += 15;
	echo "Salut !";
	return ($toto+10);
}
$nbr = MaFonction(15.1);
```

Fonctions (3/4)

- static permet de conserver la valeur d'une variable locale à une fonction.
- Exemple :

```
function change() {
    static $var;
    $var++;
}
```

Fonctions (4/4)

- On peut donner une valeur par défaut aux arguments lors de la déclaration de la fonction.
- Exemple :

```
function Set_Color($color="black") {
        global $car;
        $car["color"] = $color;
}
```

- Forcer le passage de paramètre par référence
- Exemple :

```
function change(&$var) { // force le passage systématique par ref
        $var += 100;    // incrémentation de +100
}
$toto = 12;         // $toto vaut 12
change($toto); // passage par valeur mais la fonction la prend en  référence
echo $toto;        // $toto vaut 112
```

Fonctions dynamiques

- Il est possible de créer dynamiquement des appels de fonctions.
- Pour les déclarer, on affecte à une variable chaîne de caractères le nom de la fonction à dupliquer.
- Puis on passe en argument à cette variable les paramètres normaux de la fonction de départ.

- Exemple :

```
function divers($toto) {
      echo $toto;
}
$mafonction = “divers”;
$mafonction(“bonjour !”);  // affiche 'bonjour !' par appel de divers()
```

Inclusions

- La fonction `include()` inclut et exécute le fichier spécifié en argument.
- `require()` est identique à `include`() mise à part le fait que lorsqu'une erreur survient, il produit une erreur fatale de type E_ERROR. En d'autres termes, il stoppera le script alors que `include()` n'émettra qu'une alerte de type `E_WARNING`, ce qui permet au script de continuer.

- Exemple :

```
require("fichier.php");
```

- Exemple :

```
include("fichier.php");
```

Arrêt prématuré

die arrête un script et affiche un message d'erreur dans le navigateur.

- Exemple :

```
if(mysql_query($requette) == false)
     die("Erreur de base de données à la
requête : <br />$requet");


exit l'arrête aussi mais sans afficher de
message d'erreur.
Exemple :
function foobar() {
     exit();
}
```

Entêtes HTTP

- Gérer l'envoie des entêted HTTP
- La création des entêtes HTTP doit s'effectuer avant l'envoie de la réponse
- Syntaxe : header($str);
- Exemples :
  - `header("Content-type: image/gif");` // spécifie le type d'image gif
  - `header("Location: ailleurs.php");` // redirection vers une autre page
  - `header("Last-Modified: ".date("D, d M Y H:i:s")." GMT");`
  - `headers_sent()` : Retourne TRUE si les entêtes ont déjà été envoyées, FALSE sinon.

Entêtes HTTP

Exemple :

```
<?php
header("Location: home2.php");
exit();
?>
```

Ce script effectue une redirection vers une autre page.

Évaluation d'une portion de code PHP

- La fonction `eval($str)` évalue la chaîne de caractères `$str` comme du code php.
- Le code de la chaîne `$str` doit respecter les mêmes contraintes que le code normal. Notamment :
  - point virgule après chaque instruction
  - respect des séquences d'échappement
  - etc…

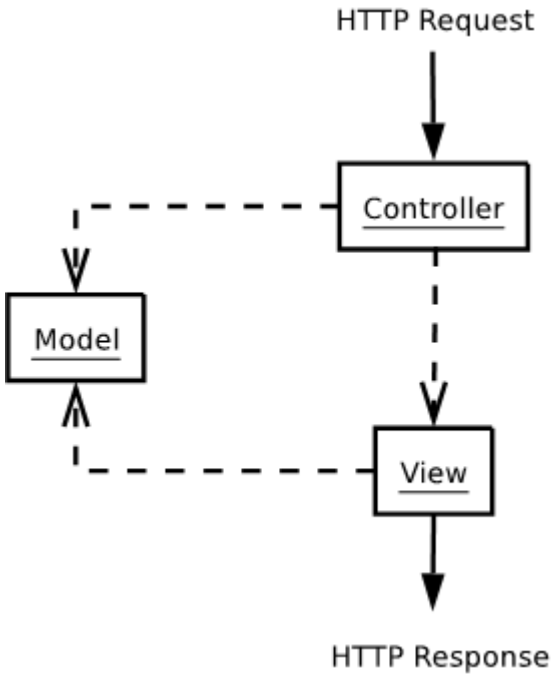
Architecture – Le pattern MVC

Le contrôleur

Le contrôleur

- Le contrôleur est la partie de l'application qui réceptionne les demandes, traite les infos et retourne les réponses.
- Le contrôleur fait le lien entre le modèle et la vue. Il s'occupe de traiter les données reçues du modèle et d'indiquer à la vue ce qu'elle doit afficher.
- Le contrôleur regroupe les traitements de gestion de l'application, c'est à dire les événements déclenchés par l'utilisateur ou par le système.
- Le contrôleur est le premier point d’entrée pour toutes les requêtes HTTP reçues dans une application Web.

Le contrôleur



Le contrôleur

- Le rôle du front contrôleur :
  - Analyser la requête, extraire les informations dans l'URL
  - Définir la route
  - Définir l'action a executer
  - Définir les paramètres

```
index.php?module=detail&action=afficher&id=1
```

Le contrôleur

- C'est le motif le plus clair et le plus facile à mettre en œuvre, la POO n'étant pas nécessaire

```
$route  = $_GET['module'];



switch ($route) {
    case "detail":
        require("detail.php");
        break;
    case "catalogue":
        require("catalogue.php");
        break;
    default:
        require("home.php");
    break;
}
```

Le contrôleur

- Le contoleur va traiter l'action et les paramêtres asscociés

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


switch ($action) {
    case "afficher":
       $id=$_GET ["id"];
       echo (afficherDetail ($id));

    default:
        echo (afficherErreur());
    break;
}
```

Le contrôleur

- Un contrôleur en POO nécessite l'implémentation de ces principales classes :
  - FrontController
  - Request
  - Response
  - ActionController

Le contrôleur

- Le **`FrontController`** est destiné à prendre en charge l'analyse de la requête client.
- Il est le seul et unique point d'entrée de l'application
- L'url contiendra le module, l'action et d'éventuels paramètres.

```
abstract class FrontController {
	public static function dispatch() {
		$request = new Request();
		$response = new Response();
		ActionController::process($request, $response)-
>display();
	}
}
```

Le contrôleur

```
http://mondomaine/monappli/index.php?module=catalogue&action=list
```

- La classe **`Request`** gère les informations communiquée par la vue à destination du contrôleur.

```
class Request {
public function route()
{
        if (isset($_GET['module'])){
                $this->module = $this->getParam('module');
        }
        if (isset($_GET['action']))
        {
                $this->action = $this->getParam('action');
        }
}
```

Le contrôleur

- La classe **`Response`** produit le flux HTTP généré par la vue a destination du client.
- Elle met en relation les données résultant du traitement associé au module et le template a utiliser pour afficher le résultat spécifique à l'action de l'utilisateur .

Le contrôleur

- L'**`ActionControleur`** permet de déclencher l'exécution du module fonctionnel associé à l'action effectuée par l'utilisateur sur l'IHM

```
class ActionController
{
    protected $_request;
    protected $_response;

    public static function process(Request $request, Response $response)
    {
        if (!file_exists($path = 'controllers/' . $request->getParam('module') .'.php')){
            throw new ControleurIntrouvableException ('contrôleur introuvable');
        }
        require_once($path);
        $class = $request->getParam('module') . 'Controller';
        $controller = new $class($request, $response);
        return $controller->launch();
    }
```

Le contrôleur

- La méthode launch de l'**`ActionController`** va déclencher l'action associer à l'action de l'utilisateur

Elle prend en charge :

  - la relation avec le modèle
  - La génération du flux spécifique a destination de la vue

Architecture – Le pattern MVC

SLIM

SLIM

http://www.slimframework.com/

SLIM est un Framework simple pour PHP5 permettant d'implémenter un controleur

SLIM

- Installation

```
composer require slim/slim "^3.0"
```

SLIM

- Un contrôleur doit définir :
  - Une méthode HTTP (GET / POST / ..)
  - Une route
  - Des Paramètres

```
get('/accueil');
post('/catalogue');:
get('/detail/{id}');:
```

SLIM

- Un contrôleur doit définir :
  - Une méthode HTTP (GET / POST / ..)
  - Une route
  - Des paramètres
  - Une méthode pour le traitement de l'action

```
get('/accueil',accueil);
post('/catalogue',catalogue);:
get('/detail/{id}',detail);:
```

SLIM

- La méthode admet 3 paramêtres :
  - Resquest : accès au flux envoyé par le client dans la requete HTTP. Récupérer l'URI et les paramêtres envoyer dans la querystring
  - Response : flux de la réponse HTTP envoyé dans la réponse HTTP

```
$app->get('/detail/{id}',
      function ($resquest,$response,$args) {
      $id = $args['id'];
      $resquest
      return $response->write("<h1>Bonjour</h1>"});
```

SLIM

- L’objet resquest :
  - Récupérer l’URI et les paramètres
  - Récupérer les Headers
  - Récupérer les données envoyées au serveur

```
$uri = $request->getUri();
$uri = $request->getUri()->getQuery();
$method = $request->getMethod();

$headers = $request->getHeaders()
$headerValueArray = $request->getHeader('Accept')
$parsedBody = $request->getParsedBody();
$files = $request->getUploadedFiles();
```

SLIM

- L’objet response :
  - Changer le statut de la réponse
  - Changer les Headers
  - Envoyer les données au client

```
$headers = $response>getHeaders();
$headerValueArray = $response->getHeader('Vary');

$newResponse = $response->withStatus(302);
$body = $response->getBody()->write(‘Hello’);
```

SLIM

Exemple controleur

```
<?php
require 'vendor/autoload.php';
$app = new \Slim\App;

$app->get('/accueil',
      function ($resquest,$response,$args) {
      return $response->write("<h1>Bonjour</h1>"});

$app->run();

?>
```

SLIM & TWIG

- Interfacer SLIM et TWIG pour permettre aux contrôleur de générer des vues sur la base de templates HTML

```
$app->get('/detail/{id}', detail);
function detail ($request, $response, $args) {
      global $twig;
      $params = array("titre" => "Bonjour");
      $template = $twig ->loadTemplate ('detail.twig');
      return $response->write( $template ->render
($params)); }
```

Architecture – Le pattern MVC

Le rewriting

Rewriting d'URL

- **L' URL Rewriting - APACHE mod_rewrite**
- MOD_Rewrite du serveur Apache permet de réécrire les URL à la volée (appelé URL Rewriting).
- Ce module utilise des expressions régulières.
- Ouvrir le fichier **httpd.conf** dans le dossier Apache et de décommenter les 2 lignes suivantes en enlevant le # :

```
LoadModule rewrite_module modules/mod_rewrite.so
AddModule mod_rewrite.c
```

Rewriting d'URL

- Créez un fichier .htaccess dans les répertoires ciblés par le rewriting d'URL
- **`RewriteEngine on`**
  Cette instruction active la réécriture d'URL. Elle devra toujours être mise dans le fichier htaccess.

Rewriting d'URL

- **Réécrire des URL dynamiques**
- La réécriture des URL permet de présenter à l'internaute une url plus mnémotechnique, facilitant dans la foulée son indexation par les moteurs de recherche qui ne laisseront plus sur le côté des pages dynamiques avec de multiples paramètres.

RewriteRule ^index\.php$ [QSA,L]

```
RewriteRule pattern target [Flag1,Flag2,Flag3]
```

Rewriting d'URL

- http://www.monsite.fr/detail_livre1.html
- Deviendra : http://www.monsite.fr/index.php?module=detail&param=livre1

```
RewriteEngine on
RewriteRule ([a-z]+)_([a-z]+)\.html$ /index.php?module=$1&ref=$2
```

- http://www.monsite.fr/index.php?module=catalogue&action=12&param=1
  deviendra : http://www.monsite.fr/catalogue/12/1

```
RewriteEngine on
RewriteRule ([a-z]+)/([0-9]+)/([0-9]+).\html$
/index.php?module=$1&action=$2param=$3
```

Rewriting d'URL

- [L] Ce (drapeau) signifie que la règle est la dernière à être appliquée et que le module ne doit plus tenter de réécrire cette chaîne.
- [QSA] : une requête

```
/index/123?one=two
```

sera réécrite en

```
/index.php?page=123&one=two
```

Sans ce drapeau la requête sera réécrite en :

```
/index.php?page=123
```

la query string existante sera supprimée.

Rewriting d'URL

- Simplifier les URL SLIM

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [QSA,L]
```