

# 01.Introduction aux frameworks PHP

11 octobre 2018

## Développement web dlm3

### Introduction aux frameworks PHP

HE-Arc 2016-18 DGR et YBL

---

### Framework<sup>1</sup>

- Fonctionnalités similaires pour de nombreuses applis
- Composants de haut-niveau réutilisables (faible couplage)
- Règles de codage et d'architecture
- Code sûr et efficace
- Facilite les tests et la gestion de projets complexes
- Utilisation de Design Patterns dès que possible
- Comportement par défaut
- Extensible
- Principe d'inversion de contrôle

Différences entre framework et library sur Stack Overflow<sup>2</sup> ou artima developer<sup>3</sup>.

### Design Patterns et webdev

- Inversion de contrôle (IoC<sup>4</sup>)
- Model View Controller

---

<sup>1</sup>[http://en.wikipedia.org/wiki/Software\\_framework](http://en.wikipedia.org/wiki/Software_framework)

<sup>2</sup><http://stackoverflow.com/questions/148747/what-is-the-difference-between-a-framework-and-a-library>

<sup>3</sup><http://www.artima.com/forums/flat.jsp?forum=106&thread=152104>

<sup>4</sup><http://martinfowler.com/bliki/InversionOfControl.html>

- M : Accès aux données, logique métier
- V : Templates des pages à générer
- C : Orchestration, transfert des infos
- Front Controller
  - Traitement et dispatch des requêtes
  - (bootstrap, ré-écriture des URL, ...)
- Object Relational Mapping<sup>5</sup>
  - Active Record, Table Data Gateway, Data Mapper, ...
- UI Patterns<sup>6</sup>

## MVC for webdev

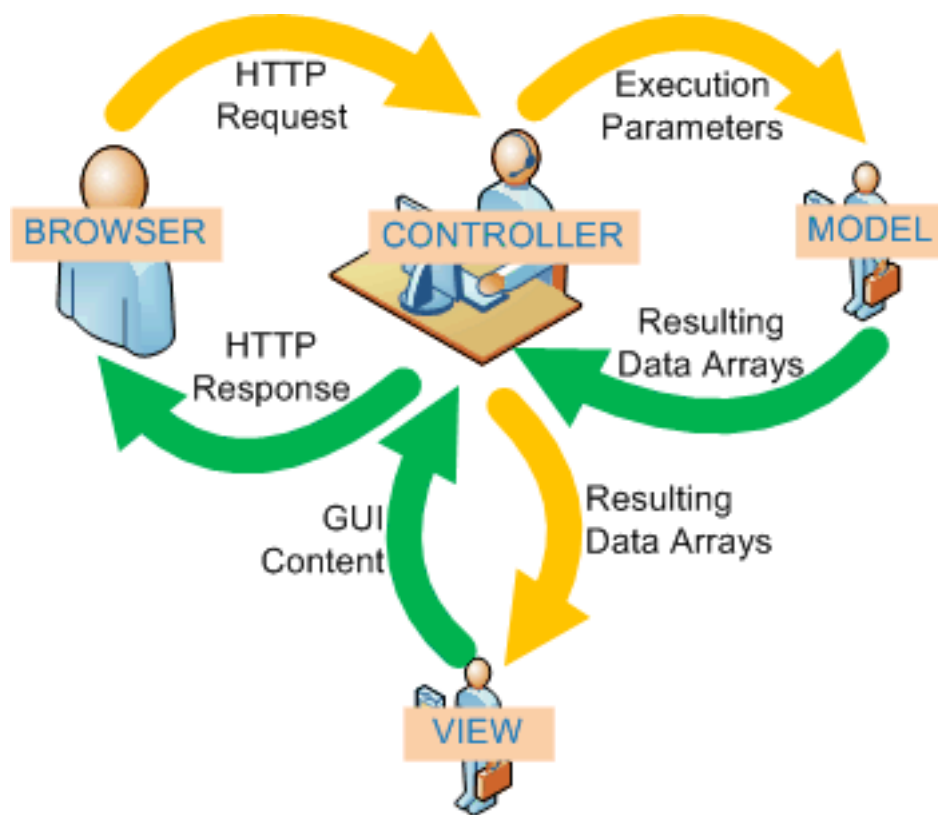


FIG. 1 : “MVC”

<sup>5</sup><https://web.archive.org/web/20160316065751/http://blog.mazenod.fr/2010/01/design-pattern-mvc-zoom-sur-la-couche-modele-dal-dao-orm-crud/>

<sup>6</sup><http://ui-patterns.com/>

## Conventions

- Nommage
  - Classes
  - Base de données
  - Fichiers et dossiers
- ROUTES : `http://app.host.tld/controller/action[/key/val]`
- Arborescence :
  - Imposée ou libre selon frameworks
  - Pas de code (minimum) sous la racine web
- Conventions obligatoires ou non, mais RECOMMANDEES dans tous les cas

## Bonnes pratiques

- Heavy Model, Light Controller
- Don't Repeat Yourself
- You Ain't Gonna Need It
- Convention Over Configuration
- Keep It Simple and Stupid
- 12 factor app<sup>7</sup> - fr<sup>8</sup>

## Pretty ( | smart | clean | formatted) URL

- Les URL doivent être explicites :
  - Manipulées par l'utilisateur
  - Utilisées pour le référencement
- Cohérence avec l'implémentation MVC :

`http://app.host.tld/controller/action[/key/val]`


- Le routage (routing)
  - Le Front Controller reçoit toutes les requêtes (URL rewriting)
  - Il les dispatche vers les contrôleurs

---

<sup>7</sup><https://12factor.net/>


<sup>8</sup><https://12factor.net/fr/>

## Smart URL & SEO

 **seomoz.org**  
Read SEOMoz. Rank Better

### SEO Cheat Sheet: Anatomy of A URL

**1**  
SEO-FRIENDLY URL

**1** **2** **3** **4** **5** **6** **7**  
`http://store.example.com/topics/subtopic/descriptive-product-name#top`

**1** Protocol  
**2** Subdomain  
**3** Domain  
**4** Top-Level Domain  
**5** Folders / Paths  
**6** Page  
**7** Named Anchor


**Keyword Priority<sup>1</sup>**  
Observed Google priority of keyword placement:  
(1) Domain  
(2) Subdomain  
(3) Folder  
(4) Path/Page

**SEO Tips for URLs**

- Use **subdomains** carefully. They may be treated as separate entities, splitting domain authority.
- Separate **path** & **page** keywords with hyphens ("-").
- **Anchors** may help engines understand page structure.
- Keyword effectiveness in URLs decreases as URL length and keyword position increases.<sup>1</sup>

<sup>1</sup> SEOMoz correlational data (2009)

**2**  
OLD DYNAMIC URL

**1** **2** **3** **4** **5** **6** **7** **7** **7**  
`http://www.example.com/index.php?product=1234&sort=price&print=1`

**1** Protocol  
**2** Subdomain  
**3** Domain  
**4** Top-Level Domain  
**5** Page / File Name  
**6** File Extension  
**7** CGI Parameters

**Popular TLDs<sup>2</sup>**

- .com** - commercial
- .net** - infrastructure
- .org** - non-profit
- .edu** - schools
- .info** - informational
- .biz** - small business
- .name** - personal sites

**Popular ccTLDs\***

- .cn** - China
- .de** - Germany
- .uk** - United Kingdom
- .nl** - Netherlands
- .eu** - European Union
- .ru** - Russian Federation
- .ar** - Argentina

**Popular Extensions**

- .htm** - Static HTML
- .html** - Static HTML
- .php** - PHP code
- .asp** - ASP code
- .aspx** - ASP.NET
- .cfm** - ColdFusion
- .jsp** - Java Code

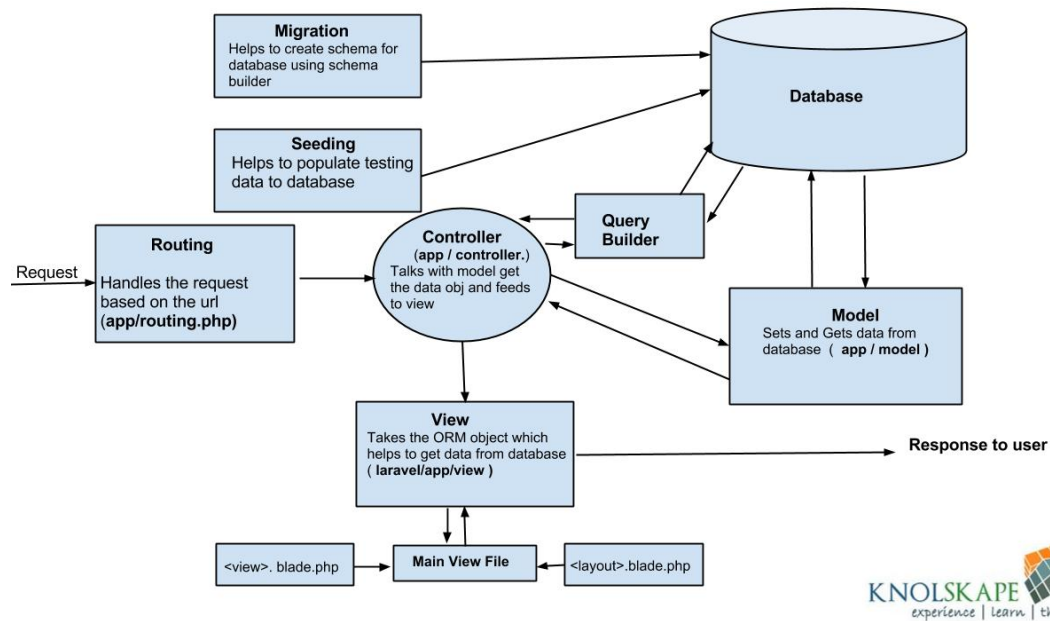
<sup>2</sup> Verisign domain report (2009)      \* ccTLD = Country Code TLD

© 2009 SEOMoz · [www.seomoz.org](http://www.seomoz.org) · Read SEOMoz. Rank Better.

## Autres Services

- Migrations : Evolutions de la structure de la BDD
- Tests
- Génération, validation et traitement de formulaires
- Authentification, Sessions, Permissions, Roles, ACL
- Pagination
- I18n
- Génération de code
- Mail
- Connecteurs aux webservices
- Captchas
- Loggers
- ...

## Exemple d'architecture : Laravel



## Performance


- Un framework web est lent :
  - Rendu d'une page nécessite de traverser tout le code
  - Pour chaque requête toute l'appli est chargée
  - Plus de code qu'une appli standalone
  - Plus de requêtes
- Solutions
  - Cache de pages, d'opcode
  - Jointures ORM, vues, procédures stockées
  - Outils d'optimisation : YSlow, page speed, mytop

## Frameworks PHP

- Lesquels connaissez-vous ?
- Lesquels avez-vous utilisé ?
- Pourquoi y en a-t-il tant ?

L'explication donnée par Joe Gregorio pour le langage Python<sup>9</sup> est : « parce que c'est facile. » Dans les faits, cela montre également une maturité de la plateforme.

---

*There are people who actually like programming. I don't understand why they like programming.* Rasmus Lerdorf <sup>10</sup>

---

- PHP-FI *Forms Interpreter*
- PHP 3, réécrit en C++
- PHP 4 *Zend Engine*, fausse POO
- PHP 5, vraie POO
- PHP 5.1, PDO
- PHP 5.2, JSON
- PHP 5.3, goto et namespace
- PHP 5.4, [] et trait
- PHP 5.5, yield
- PHP 6, Unicode 🍌, 🍊, 🐧
- PHP 7, que du rêve !

Il y a plus de vingt ans, Rasmus Lerdorf bricola un outil pour savoir qui consultait son CV.

Zend, c'est à dire *ZEev* et *aNDi*, ont réécrit PHP et qui allait devenir PHP 3 le précurseur du langage de prédilection pour créer sur le web.

PHP a évolué depuis pour devenir ce qu'il est aujourd'hui. Sa popularité est liée au fait qu'il est simple à mettre en œuvre, gratuit et libre. Tout un tas de modules est fourni avec pour faire de l'imagerie, des bases de données, du XML, etc.

Et plus encore sur la page History of PHP<sup>11</sup> et Wikipedia : PHP<sup>12</sup>.

Les différentes moutures de PHP 7 offrent ceci, entre autres.

- PHP 7, performances
- PHP 7.1, void
- PHP 7.2, sodium

---

L'évolution de PHP a fait que les usagers du langage, créateur de *frameworks*, d'outils (comme *Composer*<sup>13</sup>), ont senti le besoin d'émettre des recommandations afin d'aller vers un plus inter-opérable.

---

<sup>9</sup>[http://bitworking.org/news/Why\\_so\\_many\\_Python\\_web\\_frameworks](http://bitworking.org/news/Why_so_many_Python_web_frameworks)

<sup>10</sup>[https://en.wikiquote.org/wiki/Rasmus\\_Lerdorf](https://en.wikiquote.org/wiki/Rasmus_Lerdorf)

<sup>11</sup><http://php.net/manual/en/history.php.php>

<sup>12</sup><https://en.wikipedia.org/wiki/PHP>

<sup>13</sup><http://getcomposer.org/>

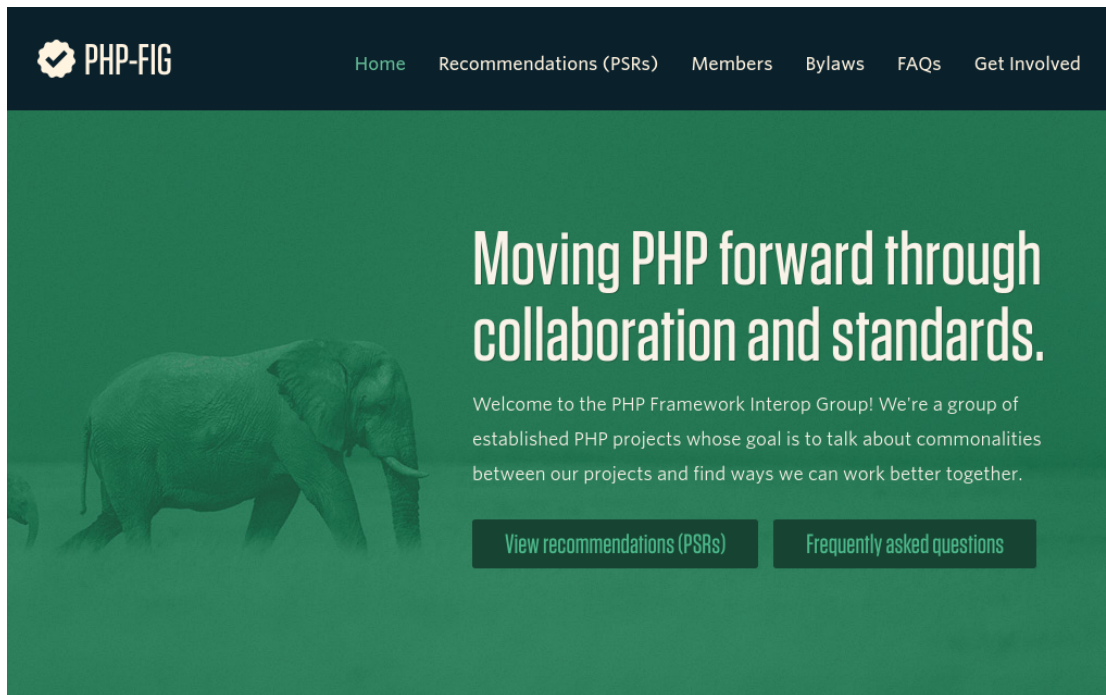


FIG. 2 : PHP Framework Interop Group

Durant ce cours, nous allons vous embêter avec PSR-1, PSR-2 et PSR-4.

---

## Quiz

Qui est qui ?

---

oOops, ceci n'a rien à voir avec le cours.

---





FIG. 3 : source<sup>14</sup>



(1)



Donc, ce ne sont pas Gandalf (sans sa barbe) et Saruman mais bien Sir Tim Berners-Lee et Vinton Cerf, responsables du (World Wide) Web et de l'Internet.

---

### Qu'est-ce qu'Internet<sup>15</sup> ?

- un réseau IP
- 

### Qu'est-ce que le World Wide Web<sup>16</sup> ?

- URI/URL, des identifiants uniques
  - HTML, un langage de publication
  - HTTP, un protocole d'échange de texte (ou *HyperText*)
- 

## Préparatifs

<https://github.com/HE-Arc/php-intro-framework/><sup>17</sup>

```
$ sudo systemctl start httpd
$ cd /var/www/html
$ git clone \
> https://github.com/\
> HE-Arc/php-intro-framework

$ cd php-intro-framework
$ open http://localhost/php-intro-framework
```

Les exemples suivant travaillent sur le code disponible dans le dépôt HE-Arc/php-intro-framework<sup>18</sup>.

---

```
$ curl -v "http://he-arc.ch/?id=25"
> GET /?id=25 HTTP/1.1
```

---

<sup>15</sup><https://www.youtube.com/watch?v=iDbyYGrswtg>

<sup>16</sup><http://line-mode.cern.ch/www/hypertext/WWW/TheProject.html%5E>

<sup>17</sup><https://github.com/HE-Arc/php-intro-framework>

<sup>18</sup><https://github.com/HE-Arc/php-intro-framework>

```

> Host: he-arc.ch
>
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=utf-8
<

<!DOCTYPE html>
<title>HE-Arc</title>
<p>Hello

```

HTTP est un protocole texte plutôt simple, jugez plutôt :

Ce que nous voyons est une connexion TCP/IP au serveur `he-arc.ch`. Une fois la connexion établie, il envoie en texte ASCII les entêtes HTTP puis deux retours à la ligne (ce qui correspond à une ligne vide). La requête HTTP commence toujours par la demande, ici `GET /index.php?page=equipe&id=25 HTTP/1.1` puis les entêtes, ici `Host: www.he-arc.ch`. La réponse du serveur est du même type, le code de réponse (`HTTP/1.1 200 OK`), les entêtes, une ligne vide puis le contenu.

La demande et les entêtes sont en US-ASCII mais le corps peut être encodé autrement, ici c'est dit dans l'entête `Content-Type: text/html; charset=utf-8`.

---

## Fait #1

PHP parle HTTP.

Le fichier `index.php` est le code PHP le plus simple qui soit. Simple au sens du niveau de compréhension de PHP et d'une forme de complexité.

```

<?php // 00-base

// Lecture de la query string `page=<XX>&id=<YY>`.
$page = $_GET["page"] ?? null;
$id = (int) ($_GET["id"] ?? 0);

// Connexion à la base de données.
$db = new PDO("sqlite:../users.db");

// Page HTML
?>
<!DOCTYPE html>
<meta charset=utf-8>
<title>HE-Arc</title>
<?php

```

```

// Contenu
if ("equipe" === $page):
    $query = $db->query("SELECT * FROM `personnes` WHERE `id` = :id;");
    $query->execute(compact('id'));

    $personne = $query->fetch(PDO::FETCH_OBJ);
?>
<p><a href="<?php echo $_SERVER["PHP_SELF"] ?>">retour</a>
<h1>Équipe</h1>
<h2>
    <?php echo $personne->prenom ?>
    <?php echo $personne->nom ?>
</h2>
<p>
    
<?php
else:
?>
    <h1>Accueil</h1>
    <ul>
        <li><a href="?page=equipe&id=1">Yoan Blanc</a>
        <li><a href="?page=equipe&id=2">Yoan Blanc</a>
    </ul>
<?php
endif

```

---

## Fait #2

PHP est un langage de template.

Pour preuve, il faut ouvrir une balise <?php pour commencer la partie code.

Avec la pratique, on a réalisé que mélanger la logique métier et celle d’affichage n’est pas optimal car difficile à lire et maintenir.

## Séparation métier/affichage

```
<?php // 01-includes/index.php

// ...

include "templates/entete.html";

if ("equipe" === $_GET["page"]) {
    // SELECT FROM u WHERE id=$_GET["id"]
    // ...
    include "templates/equipe.html";
} else {
    // ...
    include "templates/accueil.html";
}
```

---



Quel est le problème avec cette solution ?

(Source de l'image<sup>19</sup>)

---

## Sécurité des templates

- *Principle of Least Privilege* ( polp<sup>20</sup> )
- Intégration faite par un graphiste, société externe

Dans ce le cas présent rien ne nous empêche de mettre de la logique métier dans nos fichiers de *template*, car ils sont faits de PHP eux aussi.

---

```
{# 02-twig/templates/collaborateur.html #}
{%- extends "base.html" -%}

{% block corps -%}
<p><a href="?">retour</a>
<h1>Équipe</h1>
<h2>
  {{- personne.prenom -}}
  {{ personne.nom -}}
</h2>
<p>
{% endblock -%}
```

La page est réalisée avec Twig<sup>21</sup> <2.0. À partir de la version 2.0, il faut utiliser un *autoloader* externe, comme celui de composer (voir ci-dessous).

Le code est un poil plus propre du côté de nos *templates* qui ne peuvent plus exécuter de PHP sauf ce qu'on leur autorise, ici md5 et strtolower. Voir 02-twig/index.php<sup>22</sup>.

```
<?php // 02-twig

require_once 'Twig/lib/Twig/Autoloader.php';
Twig_Autoloader::register();
```

---

<sup>19</sup><https://raw.githubusercontent.com/cyrilmanuel/picbot/e6ff24a8bfd7ee9f0514a4fd8f49b1255ef26178/picbot/Images/meme10.jpg>

<sup>20</sup>[https://en.wikipedia.org/wiki/Principle\\_of\\_least\\_privilege](https://en.wikipedia.org/wiki/Principle_of_least_privilege)

<sup>21</sup><http://twig.sensiolabs.org/>

<sup>22</sup>02-twig/index.php

```
// ...

// Configuration de Twig
$loader = new Twig_Loader_FileSystem("templates");
$twig = new Twig_Environment($loader);

// Ajout des filtres md5 et strtolower qui sont les fonctions PHP du même nom.
$twig->addFilter(new Twig_SimpleFilter('strtolower', 'strtolower'));
$twig->addFilter(new Twig_SimpleFilter('md5', 'md5'));

// variable globale
$titre = "HE-Arc";

// Contenu
if ("equipe" === $page) {
    // ...
    $personne = // ...

    echo $twig->render("equipe.html", compact("titre", "personne"));
} else {
    $personnes = // ...

    echo $twig->render("accueil.html", compact("titre", "personnes"));
}
```



Problème d'injection SQL.

Effectuer des requêtes MySQL à la main ou devoir connaître tous les champs crée beaucoup de redondance et de failles de sécurité potentielles.

Une solution est d'ajouter une couche d'abstraction qui va cacher la structure réelle de notre base de données et offrir une interface orientée objet. Un *Object-Relational Mapping* ou ORM(3) dans le jargon.



---

```
<?php
// Ne dites plus
$query = $db->query(
    "SELECT * FROM `personnes` ".
    "WHERE `id` = :id;"
);
$query->execute(compact('id'));
$personne = $query->fetch(PDO::FETCH_OBJ);

// Mais dites plutôt

// RedBean
$personne = R::load('personnes', $id);
// ou Doctrine
$personne = $om->find('Personne', $id);
```

---

## *Object-Relational Mapping*

- RedBean<sup>23</sup>
- Doctrine<sup>24</sup> (ORM, ODM)
- Eloquent ORM<sup>25</sup>
- etc.<sup>26</sup>

Une bibliothèque qui va créer ce lien entre les mondes objet et relationnel ou document (généralement MongoDB). Il en existe toute une foule.

---

```
<?php // 03-redbean/index.php
require 'RedBean/rb.php';
R::setup("sqlite:../users.db");
// ...
if ("equipe" === $page) {
    $personne = R::load("personnes", $id);
    echo $twig->render(
        "equipe.html",
```

---

<sup>23</sup><http://www.redbeanphp.com/>

<sup>24</sup><http://www.doctrine-project.org/>

<sup>25</sup><http://laravel.com/docs/master/eloquent>

<sup>26</sup>[https://en.wikipedia.org/wiki/List\\_of\\_object-relational\\_mapping\\_software#PHP](https://en.wikipedia.org/wiki/List_of_object-relational_mapping_software#PHP)

```

        compact("titre", "personne")
    );
} else {
    $personnes = R::find("personnes");
    echo $twig->render(
        "accueil.html",
        compact("titre", "personnes")
    );
}

```

---

## URI as UI

Pensez à Wikipedia.

Les adresses des pages font partie de l'expérience utilisateur. Un utilisateur doit être capable d'imaginer le contenu de la page en lisant l'URI. Certainement, ce que vous faites avant de cliquer sur un lien.

---

## Comment humaniser ?

```
/index.php?page=equipe&id=42
```

La personne avec l'identifiant 42 aura également un *slug* unique créé à partir de son nom, ici jean-bon.

La solution à notre problème est de demander au serveur web de réécrire les URL pour nous.

---

## Réécriture d'URL

```

# 04-routes/.htaccess

# mod_rewrite
RewriteEngine on
RewriteBase /php-intro-framework/04-routes/

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L,QSA]

```

Apache le fait via `mod_rewrite`<sup>27</sup> et Nginx `try_files`<sup>28</sup>.

---

```
// 04-routes/index.php

$uri = $_SERVER['REQUEST_URI'],
$matches = [];

preg_match(
    "#^/(?P<page>[^/]+)/(?P<slug>[^/]+)/?#",
    $uri,
    $matches
) or die('Arrrrrrgh');

echo call_user_func_array(
    [$matches['page'],
    [$matches['slug']]
];
```

Le code complet va nettoyer l'URI et définir les fonction correspondant aux pages possibles.

---

## Routing

Lien entre les adresses (URI) et des actions dans le code.

a.k.a. the *Front Controller*.

En pratique, les actions ne sont pas des fonctions mises à plat mais sont encapsulées dans une classe qu'on nomme un contrôleur. Faire ainsi permet de regrouper logiquement les fonctions et éviter d'utiliser d'affreux éléments tel que `global`.

---

## Modèle - Vue - Contrôleur

- Modèle : l'ORM qui s'occupe de notre base de données
- Vue : les templates qui affiche les données

---

<sup>27</sup>[https://httpd.apache.org/docs/current/mod/mod\\_rewrite.html](https://httpd.apache.org/docs/current/mod/mod_rewrite.html)

<sup>28</sup>[http://nginx.org/en/docs/http/nginx\\_http\\_core\\_module.html#try\\_files](http://nginx.org/en/docs/http/nginx_http_core_module.html#try_files)

- Contrôleur : une classe qui définit quoi faire en fonction des entrées utilisateur (URI, formulaire, etc.)

MVC(4) vient des applications bureau et ne représente pas toujours le fonctionnement dans le monde du web. Par exemple, Django, un framework Python, se décrit comme étant *Modèle - Template - Vue*(5).

Les frameworks web en PHP (ou d'autres langages) reposent majoritairement sur ce paradigme.

---

## Composer

Gestionnaire de paquets pour PHP : [getcomposer.org](http://getcomposer.org)<sup>29</sup>

Maintenir notre répertoire de *vendor* ainsi que les *require* est peu pratique. Voici qu'entre en scène Composer<sup>30</sup>, le gestionnaire de paquet pour PHP. Packagist<sup>31</sup> est le dépôt en ligne de paquets public et utilisé par défaut.

---

### composer.json

```
{
  "require": {
    "twig/twig": "^2.0",
    "gabordemooij/redbean": "^4.3",
  }
}
```

Nos dépendances sont ainsi matérialisées dans le projet et peuvent être installée, ou mises à jour simplement.

En principe les numéros de version respectent le SemVer<sup>32</sup> (*Semantic Versioning*) et les différents signes permettent de sélectionner une ou plusieurs versions (voir [Version and constraints][<https://getcomposer.org/doc/articles/versions.md>]).

---

---

<sup>29</sup><http://getcomposer.org/>

<sup>30</sup><http://getcomposer.org/>

<sup>31</sup><https://packagist.org/>

<sup>32</sup><http://semver.org/lang/fr/>

```
$ composer install

puis

<?php // 05-composer/index.php

require 'vendor/autoload.php';

use RedBeanPHP\Facade as R;
```

Enfin, nous pouvons réduire le nombre de `require` et `include` à un seul, en laissant soin à l'*auto-loader* de charger le bon fichier à la demande. Tout ceci est spécifié dans PSR-4<sup>33</sup>. Ainsi, les définitions de Twig sont présentes et il nous suffit d'obtenir la classe `R` depuis `RedBean`<sup>34</sup>.

---

## Front-Controller

Utilisation de `FastRoute`<sup>35</sup> (voir 06-fastroute/index.php<sup>36</sup>).

```
$ composer require nikic/fast-route
```

`FastRoute` repose sur un système proche de celui que nous avons utilisé jusqu'ici. D'autres systèmes, tels que `Aura.Router` pour ne citer que lui, reposent sur la spécification PSR-7<sup>37</sup>. Cette dernière décrit l'interface objet d'un message HTTP, tant au niveau de la requête que de la réponse.

Si ça ajoute, une bonne couche de complexité, l'énorme avantage offert par cette idée là est de déléguer le rendu d'une page, ni echo, ni header, Donc il est envisageable de pouvoir tester (au sens de test unitaire), notre *FrontController*.

D'autre part, le `call_user_func_array` d'avant n'était pas très solide,

---

```
<?php // 06-fastroute/index.php
// ...
use function FastRoute\simpleDispatcher;
use FastRouter\Dispatcher;

$dispatcher = simpleDispatcher(function($r)
```

---

<sup>33</sup><http://www.php-fig.org/psr/psr-4/>

<sup>34</sup><http://www.redbeanphp.com/>

<sup>35</sup><https://github.com/nikic/FastRoute>

<sup>36</sup><https://github.com/HE-Arc/php-intro-framework/blob/master/06-fastroute/index.php>

<sup>37</sup><http://www.php-fig.org/psr/psr-7/>

```

{
    $r->addRoute('GET', '/', 'accueil');

    $r->addRoute(
        'GET',
        '/equipe/{slug}',
        'equipe'
    );
});

```

---

```

<?php // 06-fastroute/index.php (suite)

$httpMethod = $_SERVER["REQUEST_METHOD"];
$uri = $_SERVER["REQUEST_URI"];

// nettoyage de $uri
// - prefix
// - query string
// - caractères spéciaux (e.g. %20)

$routeInfo = $dispatcher->dispatch(
    $httpMethod,
    $uri
);

```

---

```

<?php // 06-fastroute (suite)

switch($routeInfo[0]) {
    case Dispatcher::NOT_FOUND:
    case Dispatcher::METHOD_NOT_ALLOWED:
        /* ... */break;
    case Dispatcher::FOUND:
        try {
            echo call_user_func_array(
                $routeInfo[1],
                $routeInfo[2]
            );
        } catch (Exception $e){
            echo server_error($e);
        }
        break;
}

```

---

## Framework PHP

Une collection de bibliothèques avec un peu de glue.

Un framework web vous propose une structure de base pour construire selon une méthode jugée bonne par ses concepteurs. Il est possible de remplacer un composant par un autre, par le sien. Et même de créer sa *glue* ou même ses outils propres.

---

### Liens avec Laravel

- Modèle MVC
- Templates utilisant *blade*.
- ORM nommé *Eloquent*.
- *Front-Controller* (Illuminate\Routing)
- Bibliothèques ... (Illuminate\\*)
- Composer<sup>38</sup>

Je vous invite à aller lire le code généré pour vous par Laravel. Vous allez retrouver ces éléments. Symfony, CakePHP, etc. auront les mêmes idées.

---

### Exercice

- Refaites les différentes étapes à partir de 00-base.
  - Tel quel ou en utilisant d'autres bibliothèques : Smarty<sup>39</sup>, Doctrine<sup>40</sup>, Aura.Router<sup>41</sup>
- 

## Fin

Questions ?

---

<sup>38</sup><http://getcomposer.org/>

<sup>39</sup><https://github.com/smarty-php/smarty>

<sup>40</sup><https://www.doctrine-project.org/projects/doctrine-orm/en/current/tutorials/getting-started.html>

<sup>41</sup><https://github.com/auraphp/Aura.Router>



## Sources

1. W3C. W3C 20 Anniversary Symposium. [en ligne]. 2014. [Consulté le 7 février 2017]. Disponible à l'adresse : <https://www.w3.org/20/Overview.html>
2. MUNROE, Randall. Exploits of a mom. [en ligne]. 2007. [Consulté le 7 février 2017]. Disponible à l'adresse : <https://xkcd.com/327/>
3. WIKIPEDIA. *Mapping objet-relationnel* [en ligne]. [Consulté le 7 février 2017]. Disponible à l'adresse : [https://fr.wikipedia.org/wiki/Mapping\\_objet-relationnel](https://fr.wikipedia.org/wiki/Mapping_objet-relationnel)
4. WIKIPEDIA. Modèle-Vue-Contrôleur. [en ligne]. [Consulté le 7 février 2017]. Disponible à l'adresse : <https://fr.wikipedia.org/wiki/Modèle-vue-contrôleur>
5. DJANGO PROJECT. Django appears to be a MVC framework, but you call the Controller the « view », and the View the « template ». How come you don't use the standard names? *FAQ : General* [en ligne]. [Consulté le 7 février 2017]. Disponible à l'adresse : <https://docs.djangoproject.com/en/1.11/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names>