Partie 1 : Utiliser un micro-framework Php

- un micro-framework, c'est quoi ?
- un exemple : Slim

- **Un framework** : un ensemble cohérent de composants (classes) organisant la structure d'une application et proposant des services génériques
- un framework est ***actif*** : le déroulement des opérations de l'application est dirigé par le framework ; les modules construits par les développeurs sont ***appelés*** par le framework

un framework MVC ?

- un **framework MVC** = 1 framework basé sur le patron d'architecture MVC
- un framework MVC contraint l'architecture d'une application à suivre ce schéma :
  - des modèles, basés sur 1 orm,
  - des contrôleurs/dispatchers
  - des vues et des templates
  - des utilitaires : forms, auth, mail …
- PHP : symfony, zend, laravel, cakePHP …
- ruby : rails
- java : spring, struts2 ...

les plus, les moins

- + beaucoup de services
- + architecture MVC

- - complexes : difficiles à apprendre et maitriser
- - lourds : performances dégradés
- - gros : tout n'est utile dans tous les projets

micro-framework mvc

- même principe mais plus légers et agiles
  - moins de services
  - uniquement des services indispensables dans tous les cas : routages d'urls, gestion des requêtes/réponses/erreurs
  - bien adaptés pour la réalisation d'api
  - plus faciles et rapides à apprendre, à maitriser
- PHP : lumen, **slim**
- ruby : sinatra, camping
- java : spark, micro
- nodejs : express.js

le framework **Slim 3**

- micro-framework php léger et rapide, permettant de réaliser rapidement des applis et api web

- routage d'urls
- gestion des requêtes/réponses
- extensible

- http://www.slimframework.com/
- PHP 5.5 minimum
- **`composer require slim/slim "3.*"`**

Slim Hello World

```php
<?php
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require 'vendor/autoload.php';

$app = new \Slim\App;

$app->get('/hello/{name}',
      function (Request $req, Response $resp, $args) {
        $name = $args['name'];
        $resp->getBody()->write("Hello, $name");
        return $resp;
      }
);

$app->run();
```

principes de Slim

Slim est un ***dispatcher*** : sert à écrire le programme principal qui traite les requêtes et appelle des méthodes/fonctions.

- **principe** : associer
  - une url et une méthode http à ...
  - … une fonction/méthode appelée lorsque la requête reçue correspond au couple url/méthode
- la fonction appelée
  - reçoit la **requête** reçue et la **réponse** en cours de construction
  - **retourne** une réponse complétée

définition des routes

```php
<?php

$app = new \Slim\App;

        post
$app->  get  ('U/R/I' , callback() ) ;
        put
        delete

$app->run();
```

Méthode http concernée

URI concernée

Action à réaliser

définition des routes

```php
<?php

$app = new \Slim\App;

$app->get('/games/{id}',
     function (Request $req, Response $resp, $args) {
        ...
     });
$app->post('/games/{id}',
     function (Request $req, Response $resp, $args) {
        ...
     });
$app->put('/games/{id}',
     function (Request $req, Response $resp, $args) {
        ...
     });

$app->run();
```

définition des routes

- les **méthodes http** : `get`, `post`, `put`, `delete`, `options`, `patch`
- le **callback** : un objet *callable*
  - une fonction anonyme :

    ```
    function($req, $resp, $args){ ... }
    ```

  - un nom de fonction

    ```
    'processRequest'
    ```

  - une méthode dans une classe :

    ```
    '\app\controller\HomeController:display'
    ```

  - une méthode dans une classe : `\app\controller\HomeController::class. ':display'`

fonctionnement du routage

GET /index.php**/hello/michel** **route activée**
GET /index.php**/allo/michel** **erreur**

```php
<?php

$app->get('/hello/{name}',
                function () { } );

$app->run();
```

index.php

La première route coïncidant avec la requête est activée

simplifier les urls : utiliser mod_rewrite

GET /hello/michel route activée

```
RewriteEngine on

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

.htaccess

```
<?php

$app->get('/hello/{name}',
            function () { } );

$app->run();
```

index.php

routes paramétrées

- paramètres : { … }
- éléments optionnels : […]

```php
$app = new \Slim\App;

$app->get('/hello/{name}', function ($rq,$rs,$args) {
        echo $args['name'] ;
    });

$app->get('/hello/{n}/{id}',function ($rq,$rs,$args){
        echo $args['n'] . $args['id'] ;
    });

$app->put('/allo[/{name}]', function ($rq,$rs,$args) {
        // active pour `/allo` et `/allo/michel`
    });
```

la requête http

- objet injecté par le framework dans le callback au déclenchement d'une route
- conforme recommandation PSR-7

```php
<?php
use \Psr\Http\Message\ServerRequestInterface as Request;

$app = new \Slim\App;

$app->get('/hello/{name}',
     function (Request $rq, $rs, $args) {
        ...
        return $resp;
     }
);
```

```php
function (Request $rq, $rs, $args) {
  // récupérer la méthode http
  $method = $rq->getMethod() ;
  // l'uri sous forme d'objet
  $uri    = $rq->getUri() ; $uri->getHost() ;

  // les headers
  $headers = $rq->getHeaders() ;
  $accept  = $rq->getHeader('Accept') ;
  $ctype   = $rq->getContentType() ;
  $clength = $rq->getContentLength() ;

  // les paramètes dans l'url
  $queryParams  = $rq->getQueryParams() ;
  $queryId      = $rq->getQueryParam( 'id' ) ;
  // les cookies
  $cookieParams = $rq->getCookieParams() ;
  $cookieId     = $rq->getCookieParam( 'id' ) ;
}
```

Accès au body de la requête

- Slim utilise le Content-Type pour décoder le contenu de la requête
  - xml → SimpleXMLElement
  - json, url-encoded → php array

```php
function (Request $rq, $rs, $args) {

   $parsedBody = $rq->getParsedBody() ;

    // url-encoded uniquement :

    $name = $rq->getParsedBodyParam('name') ;

}
```

Construction de la réponse

- un objet **`response`** initial est injecté dans le callback
- le callback ***doit*** retourner un objet **response**
- **les objets `response` sont non modifiables : les méthodes créent un nouvel objet à chaque appel**

```php
use \Psr\Http\Message\ResponseInterface as Response ;

function ( $rq, Response $rs, $args) {

   $rs = $rs->withStatus( 201 ) ;

    return $rs ;
}
```

construction de la réponse

- ajouter, retirer des headers :

```
function ( $rq, Response $rs, $args) {

  // ajoute ou remplace
  $rs = $rs->withHeader( 'Content-Length', 567 ) ;

  // ajoute 1 valeur à 1 header existant
  $rs = $rs->withAddedHeader( 'Allow', 'PUT' ) ;

  // retire un header
  $rs = $rs->withoutHeader( 'Allow' ) ;


   return $rs ;
}
```

construction de la réponse : le body

- le body est un objet stream dans lequel on peut écrire
- on peut aussi remplacer le body par un stream connecté à un fichier

```
function ( $rq, Response $rs, $args) {

   $rs = $rs->withStatus( 201 ) ;

   $rs->getBody()->write( json_encode( $o ) ) ;

   return $rs ;
}
```

streamer un fichier

- remplacer le body par 1 nouveau Stream
- withBody() crée une nouvelle Response

```
use \Psr\Http\Message\ResponseInterface as Response ;
use \Slim\Http\Stream ;

function ( $rq, Response $rs, $args) {

  $f = new Stream( fopen(medianet.png', 'r') ) ;

   $rs = $rs->withStatus(200)
         ->withHeader('Content-Type', 'image/png')
         ->withBody( $f );

    return $rs ;
}
```

configuration

- Slim utilise un ***conteneur de dépendances*** pour stocker sa configuration :
  - variables diverses, services

```php
$configuration = [
    'settings' => [
        'displayErrorDetails' => true,
        'dbconf' => '/conf/db.conf.ini' ]
];
$c = new \Slim\Container($configuration);
$app = new \Slim\App($c);


// récupérer le conteneur :
$container = $app->getContainer() ;
$prod = $container['settings']['dbconf'] ;
```

conteneur & routes

- le conteneur est disponible dans la fonction anonyme associée à une route dans **`$this`** *`("closure binding")`*

```
$configuration = ['settings' => [ … ] ] ;

$c = new \Slim\Container($configuration);
$app = new \Slim\App($c);

$app->get('/hello/{name}',function ($rq,$rs, $args) {
        ...
        $p = $this[ 'settings' ]['dbconf'] ;
    }
);
```

routes nommées

- Slim permet de nommer les routes définies afin de générer une url à partir de ce nom
- **à utiliser systématiquement pour générer les urls dans une application / api**

```
$app = new \Slim\App;

$app->get('/hello/{name}',function ($rq,$rs, $args) {
        ...
        return $resp;
    }
)->setName('hello');
```

génération d'url

- fait par le service 'router' enregistré dans le conteneur

```
$app = new \Slim\App;

$app->get('/hello/{name}',
          function ($rq,$rs, $args) { … }
)->setName('hello');
```

```
$url = $app->getContainer()['router']
            ->pathFor('hello', [ 'name'=>'bob'] ) ;

$app->get('/allo', function ($rq,$rs,$args) {
            $url = $this['router']
                ->pathFor('hello',['name'=>'bob']) ;
    }
)
```

Utiliser des contrôleurs

- **Bonne pratique** : utiliser des contrôleurs, créés dans les callbacks

```php
$app = new \Slim\App;

$app->get('/games/{id}',function ($rq,$rs, $args) {

    // injecter le conteneur dans le constructeur
    $c = new GameController( $this ) ;

    // injecter les requêtes, réponses et arguments
    // dans l'action
    return $c->getGame($rq, $rs, $args ) ;

}
);
```

autres notations pour des contrôleurs

```
$app = new \Slim\App;

$app->get('/games/{id}',
          "\app\controls\GameController:getGame"
)
```

```
$app = new \Slim\App;

$app->get('/games/{id}',
     \app\controls\GameController::class . ':getGame'
);
```

- le container est injecté dans le controller (constructeur)
- la requête, la réponse et les args sont passés en paramètre de la méthode