Slim Framework

Un micro framework PHP

Définition

— Un routeur
— De l'injection de dépendance
— Un système de Middleware

Installation

```
composer require slim/slim
```

Arborescence

```
— app
    — vendor
    — public
        — index.php
    — src
    — composer.json
    — composer.lock
```

Le point d'entrée

```php
//index.php dans le dossier public

//Auto chargement des classes
require dirname(__DIR__)."/vendor/autoload.php";

use Slim\App;

//Instanciation du framework
$app = new App();

//Lancement du framwork
$app->run();
```

Le routage

```php
require dirname(__DIR__)."/vendor/autoload.php";
use Slim\App;
use Slim\Http\Request;
use Slim\Http\Response;

$app = new App();

//Définition d'une route
$app->get(
    "/hello",
    funtion(Request $request, Response $response){
        return $response->getBody()->write("Hello");
});

//Lancement du framwork
$app->run();
```

Test

```
# point d'entrée index.php
php -S localhost:8000 -t public

# autre point d'entrée
php -S localhost:8888 -t public public/app.php
```

Récupération de paramètres

```php
require dirname(__DIR__)."/vendor/autoload.php";
use Slim\App;
use Slim\Http\Request;
use Slim\Http\Response;

$app = new App();

//url : /hello?name=alfred
$app->get(
    "/hello",
    funtion(Request $request, Response $response){
        $name = $request->getParam("name") ?? "world";
        return $response->getBody()->write("Hello $name");
});

//Lancement du framwork
$app->run();
```

Paramètres dans la route

```php
require dirname(__DIR__)."/vendor/autoload.php";
use Slim\App;
use Slim\Http\Request;
use Slim\Http\Response;

$app = new App();

//url : /hello/alfred
//Les paramètres sont passés à la fonction dans le troisième argument
$app->get("/hello/{name}",
    function (Request $request, Response $response, array $args){
        $name = $args["name"] ?? "world";
        return $response->getBody()->write("Hello $name");
});

//Lancement du framwork
$app->run();
```

Données envoyée en POST

```php
$app->post("/form",
    function (Request $request, Response $response){
        $data = $request->getParsedBody();
        $name = filter_var("name", FILTER_SANITIZE_STRING);
        ...
});
```

Redirection

```
$app->post("/form",
    function (Request $request, Response $response){
        $return $response->withRedirect("/list");
});
```

JSON

```
$app->get("/products/all",
    function (Request $request, Response $response){
        $data = [
            ["code"=> "A5", "price": 12],
            ["code"=> "B8", "price": 19]
        ];

        return $response->withJson($data);
});
```

Routes nommées

```php
$app->get("/hello/{name}", function (Request $request, Response $response, array $args){
    $name = $args["name"] ?? "world";
    return $response->getBody()->write("Hello $name");
})->setName("hello");
```

Utilisation d'une route nommée

```
$app->get("/home",function (Request $request, Response $response){
    $url = $this->get("router")
        ->pathFor(
            "hello",
            ["name" => "Alfred"]
        );
    $link = "<a href=\"{$url}\">bonjour Alfred</a>";
    return $response->getBody()->write($link);
});
```

Groupes de routes

```php
// Les routes déclarées sont préfixées par /user

$app->group("/user", function () use ($app){
    $app->get('/list', function (Request $request, Response $response) {
        ...
    });
    $app->get('/details/{id}', function (Request $request, Response $response, $args) {
        ...
    });
});
```

Paramètres optionnels

```php
// Cette route réponds aux url /user et /user/4
$app->get('/users[/{id}]', function ($request, $response, $args) {
    ...
});
```

Expressions régulières

```php
// Le paramètre id ne peut contenir que des chiffres
$app->get('/users/{id:[0-9]+}', function ($request, $response, $args) {
    ...
});
```

L'injection de dépendances

Le conteneur

```php
use \Psr\Container\ContainerInterface;

$container = $app->getContainer();

$container["database"] = function ( ContainerInterface $container){
    return new \PDO(...);
};

$container["client.dao"] = function ( ContainerInterface $container){
    $pdo = $container->get("database");
    return new ClientDAO($pdo);
};
```

Utilisation

```
$app->get("/client/list", function (Request $request, Response $response){
    $dao = $this->get("client.dao");
    return $response->withJson($dao->findAll());
});
```

Configuration

Le conteneur peut également servir à enregistrer un tableau de configurations

```
use \Psr\Container\ContainerInterface;

$conf = [
    "db" => [
        "host" => "localhost",
        "dbName" => "boutique",
        "user" => "root"
        "pass" => ""
    ]
];

$container = $app->getContainer();

$container["config"] = $conf;
```

Récupération de la configuration

```php
$container["database"] = function ( ContainerInterface $container){
    $conf = $container->get("config");
    $host = $conf["db"]["host"];
    $dbName = $conf["db"]["dbName"];
    $dsn = "mysql:host={$host};dbname={$dbName}";

    $options = [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
                PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
    ];

    return new PDO($dsn,
                   $conf["db"]["user"],
                   $conf["db"]["pass"],
                   $options
        );
};
```

Middlewares

Un middleware est une fonction ou un classe qui prends une requête et retourne une réponse

Un peu comme une route

Un exemple

```php
/**
 * Un middleware capture la requête, effectue un traitement
 * généralement sur la réponse et exécute le prochain middleware
 * Ici le middleware s'applique à toutes les routes de l'application
 */
$app->add(function (Request $request, Response $response, Callable $next){
    $response->getBody()->write("Nous sommes le ". date("d/m/Y"));
    return $next($request, $response);
});
```

Un mode maintenance

Attention, le dernier middleware déclaré est le premier exécuté

```php
/**
 * Ce middleware ne renvoie pas la requête aux autres middlewares
 * Il interrompt donc la chaîne des middlewares
 */
$app->add(function (Request $request, Response $response, Callable $next){
    $message = "Le site est en maintenance, revenez plus tard";
    $response->getBody()->write($message);
    return $response;
});
```

Avec un activation dans la config

```php
$app->add(function (Request $request, Response $response, Callable $next){
    $maintenance = $this->get("config")["maintenance"]??false;
    if($maintenance){
        $message = "Le site est en maintenance, revenez plus tard";
        $response->getBody()->write($message);
    } else {
        $next($request, $response);
    }

    return $response;
});
```

Gestion d'une clef d'API

```php
$app->add(function (Request $request, Response $response, Callable $next){
    $apiKey = $this->get("config")["api_key"]??null;
    $requestKey = $request->getParam("API_KEY")??null;

    if( $requestKey == $apiKey){
        $next($request, $response);
    } else {
        $message = "Accès non autorisé";
        $response->withStatus(403)->getBody()->write($message);
    }

    return $response;
});
```

Middleware pour une route

```php
$middleware = function (Request $request, Response $response, Callable $next){
    $response = $next($request, $response, $next);
    $response->getBody()->write("Good Bye");
    return $response;
};

$app->get("/test/middleware", function (Request $request, Response $response){
    return $response->getBody()->write("test middleware");
})->add($middleware);
```

Middleware pour un groupe de routes

```php
$app->group("/api", function () use ($app){
    $app->get("/user", function (RequestInterface $request, ResponseInterface $response){
        return $response->getBody()->write("Hello user");
    });

    $app->get("/client", function (RequestInterface $request, ResponseInterface $response){
        return $response->getBody()->write("Hello client");
    });
})->add($middleware);
```

Passer des attributs à un middleware

```php
$middleware = function (Request $request, Response $response, Callable $next){
    $response = $next($request, $response, $next);
    $id = $request->getAttribute("id");
    $response->getBody()->write("Good Bye $id");
    return $response;
};

$mwAttribute = function (Request $request, Response $response, Callable $next) {
    $request = $request->withAttribute("id", 5);
    $response = $next($request, $response, $next);
    return $response;
};

$app->get("/test-mw","homeController:index")
    ->add($middleware)->add($mwAttribute);
```

Un classe Middleware

```php
namespace app\Middleware;

use Psr\Http\Message\RequestInterface;
use Psr\Http\Message\ResponseInterface;

class TestMiddleware
{
    public function __invoke(RequestInterface $request, ResponseInterface $response, Callable $next)
    {
        $response->getBody()->write('BEFORE');
        $response = $next($request, $response);
        $response->getBody()->write('AFTER');

        return $response;
    }
}
```

Utilisation de la classe Middleware

```
$app->get("/use-middleware",
    function($request, $response){
        ...
    }
)->add(new app\Middleware\TestMiddleware());
```

Un service Middleware

```php
//Le service
$container["middleware.test"] = function(){
    return new app\Middleware\TestMiddleware();
}

// La route
$app->get("/use-middleware",
    function($request, $response){
        ...
    }
)->add($container->get("middleware.test"));
```

Routage avancé

Classe contrôleur

```php
namespace app\Controller;

use Psr\Http\Message\RequestInterface;
use Psr\Http\Message\ResponseInterface;

class HomeController
{

    public function index(RequestInterface $request, ResponseInterface $response){
        return $response->getBody()->write("Home index");
    }

}
```

Utilisation du contrôleur

```
$app->get(
    "/home/index",
    \app\Controller\HomeController::class.":index"
);

//ou

$app->get(
    "/home/index",
    "HomeController:index"
);
```

Injection du conteneur de services

```php
namespace app\Controller;

class HomeController
{
    private $container;

    public function __construct(ContainerInterface $container)
    {
        $this->container = $container;
    }

    public function index(
        RequestInterface $request,
        ResponseInterface $response){

        $str = $this->container->get("hello")->sayHello();
        return $response->getBody()->write("$str Home index");
    }
}
```

Factory

Injection des dépendances et instanciation dans un service factory

```
$container['homeControllerService'] = function($c) {
    $hello = $c->get("hello");
    return new HomeController($hello);
};
```

Référence au service dans la route

```
$app->get(
    "/home/index",
    "homeControllerService:index"
);
```

La classe contrôleur

```php
class HomeController
{

    protected $helloService;

    public function __construct($hello)
    {
        $this->helloService = $hello;
    }


    public function index(RequestInterface $request, ResponseInterface $response){
        $str = $this->helloService->sayHello();
        return $response->getBody()->write("$str Home index");
    }

}
```