Introdução ao Slim Framework

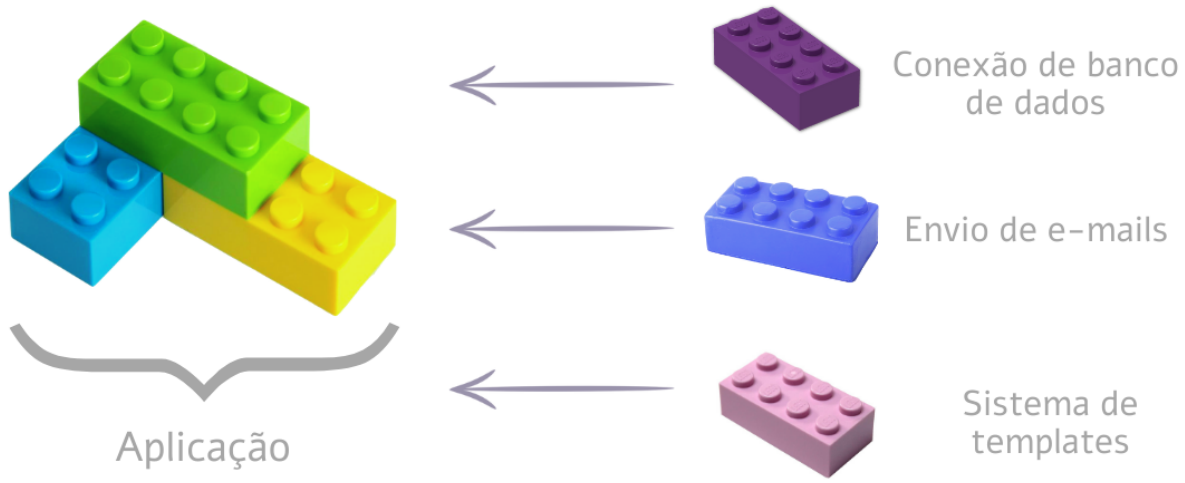
por Jefferson Chaves

*O Slim Framework é um **micro-framework**, escrito em PHP para criação de APIs. Por sua baixa complexidade de uso e por ser essencialmente modular, tem se destacado como uma boa opção para o desenvolvimento de APIs, pequenas aplicações ou websites.*

*Como um de seus principais recursos, o Slim implementa um **sistema de gerenciamento de rotas**, permitindo que funções e classes PHP possam ser chamadas por meio de requisições HTTP, tais como chamadas get, post, put e delete.*

Como principais características, podemos citar:

- *Baixa necessidade de configuração;*
- *Por não possuir um padrão de desenvolvimento definido, há liberdade para definir uma estrutura personalizada para o projeto;*
- *Facilidade na integração de aplicações externas ao projeto;*
- *Possui suporte nativo a rotas HTTP;*
- *Possui suporte a injeção de dependências;*
- *Possui uma comunidade ativa, entre outros.*



Micro-Frameworks são Frameworks modularizados que possuem uma estrutura inicial muito mais simples quando comparado a um Framework convencional.

Configurando seu ambiente

O Slim embora simples, é um conjunto de dezenas de classes e arquivos de códigos com funcionalidades já implementadas. Baixar e configurar o Slim manualmente é uma tarefa que não é tão simples. Por essa razão, esse processo pode ser realizado por meio do Composer, um Gerenciador de Dependências e configurações para projetos PHP. Uma curiosidade é que todas as linguagens possuem gerenciadores de dependência, tais como o PIP do Python e NPM do JavaScript.
Além disso, todos os frameworks usados no mercado, usam composer para seu gerenciamento, de forma que é fundamental compreender seu uso.

A instalação do Composer é simples e pode ser encontrada no link abaixo:

Windows

https://www.youtube.com/watch?v=URsHXA664KA

Linux

https://www.youtube.com/watch?v=h7dpIUUCf1k

Configuração e Instalação do Slim

A instalação do Slim deve ser realizada por meio do gerenciador de dependências do PHP, o Composer. Com o Composer devidamente instalado em seu computador, acesse a pasta do projeto e digite o seguinte comando:

```
composer require slim/slim:3.*
```

Após a execução do comando, a pasta do projeto deve se parecer com o esquema abaixo:

```
--ProjetoIFPR/
---Vendor/
------autoload.php
------Slim/
------PSR/
------Composer/
------Pimple/
------Nikic/
---composer.json
---composer.lock
```

O Composer é um gerenciador de dependências e configurações focado no PHP. É como se fosse uma "loja" de aplicativos para o PHP. Mas não só isso. As dependências são gerenciadas pelo composer. ***Saiba mais sobre o Composer clicando aqui.***

Gerenciamento de Rotas

Assim como mencionado anteriormente, uma das grandes razões para se usar o Slim é devido ao seu sistema de gerenciamento de rotas. Vamos configurar a primeira rota da API, que corresponderá à raiz da API.

Assegurado que estamos dentro da pasta do projeto, ***criaremos um arquivo chamado index.php****. O conteúdo desse arquivo de seguir o código abaixo:*

```
<?php
require __DIR__ . '/vendor/autoload.php';


use Slim\Http\Request as Request;
use Slim\Http\Response as Response;
use Slim\App;

$app = new App();


$app->run();
```

Disponibilizando o website no localhost

No ***Terminal*** *acesse a pasta do projeto e execute o comando:*

```
 php -S localhost:8080
```

Agora, no seu browser, acesse localhost:8080. Nesse momento uma mensagem de erro - ***HTTP ERROR 500.*** *Faça uma breve pesquisa: o que significa* ***error 500****?*
Melhore as mensagens de erros adicionando o seguinte trecho de código:

```
$configuration = [
    'settings' => [
       'displayErrorDetails' => true,
    ],
];

$container = new Container($configuration);
$app = new App($container);
```

Configurando nossas rotas

Pronto, instalamos o Slim e o Composer! Estamos prontos para desenvolver as rotas da nossa aplicação. Para isto cole o seguinte código no seu arquivo ***index.php****:*

```
<?php
require __DIR__ . '/vendor/autoload.php';


use Slim\Http\Request as Request;
use Slim\Http\Response as Response;
use Slim\App;

$configuration = [
    'settings' => [
       'displayErrorDetails' => true,
    ],
];

$container = new Container($configuration);
$app = new App($container);

$app->get('/', function (Request $request, Response $response, array
$args) {
    $response->getBody()->write("Hello Slim Framework!");
    return $response;
});

$app->run();
```

No ***Terminal,*** *novamente, execute o comando* `php -S localhost:8080`, *acesse* ***localhost:8080 e veja a página funcionando!***

PARE AQUI!!!

Recebendo dados pela URL

Conforme visto na teoria é possível passar informações (dados, parâmetros) pelas URLS. O slim permite o acesso a parâmetros que foram enviados como:

i) Route Params (ou Path Params): os dados fazem parte da URL

ii) Query Params: os dados vem depois do sinal de interrogação

Criando rotas

A primeira rota de um sistema é a rota raiz ou "/". Essa é a rota que será chamada quando não houver uma rota especificada na URL;

```
$app→get('/', function ($request, $response, $params) {
      return return $response→getBody()→write("Olá Slim!");
});
```

Ainda é possível determinar uma rota para um determinado recurso;

```
$app→get('/estudantes, function ($request, $response, $params) {
      return return $response→getBody()→write("Página de alunos!);
});
```

Obtendo um parâmetro da Rota (*Route Params****)***

Existem diversas formas de se enviar e receber parâmetros das rotas. É possível que esses parâmetros sejam opcionais, obrigatórios, finitos ou infinitos.

Para se obter um parâmetro da rota, são necessárias duas configurações. A primeira delas é adicionar um caractere coringa à rota, que será variável. O próximo passo é obter esses dados por meio do variável ***$parameters****.*

```
$app->get('/estudantes/{nome}', function (Request $request, Response
$response, array $parameters) {

    $name = $parameters['nome'];
    $response->getBody()->write("Olá estudante: {$name}");
```

```
    return $response;
});
$app->get('/estudantes[/{nome}]', function (Request $request,
Response $response, array $parameters) {
    $name = $parameters['nome'] ? $parameters['nome'] : "visitante";
    $response->getBody()->write("Olá estudante: {$name}");
    return $response;
});
```

*É possível ainda receber Query Params, os filtros da URL. Considere a rota localhost:8080/rota**?name=Jefferson**. Como obter o query param da URL? Com Slim é simples:*

```
$app->get('/rota', function ($request, $response, $params) {
        $name = $request->getQueryParam('name', Convidado);
       $response->getBody()->write("Olá estudante: {$name}");
});
```

Nesse algoritmo `'name'` *representa o dado vindo da URL. Quando não existir esse dado, o valor padrão será atribuído para a variável* `$name`.

Instalando um gerenciador de templates

Somente exibir mensagens na tela não parece tão interessante. Para isso, o Slim possui uma biblioteca simples, mas completa para gerenciamento de telas, a ***PHP View****. Sua instalação é simples.*

```
composer require slim/php-view
```

Após instalar a biblioteca, é necessário realizar sua importação:

```
use Slim\Views\PhpRenderer;
```

Realizada a importação, é necessário a criação de uma pasta em nosso projetos, chamada de "templates". Todas as telas devem estar dentro desta pasta. O código para um método que usa um template deve se parecer com:

```
$app->get('/hello/{name}', function ($request, $response, $args) {
    $renderer = new PhpRenderer('caminho/para/templates');
    return $renderer->render($response, "hello.php", $args);
})
```