Posa é uma micro framework rápida, simples e eficiente para o desenvolvimento de aplicações web.

Nota do autor

Posa é uma framework que desenvolvi quando senti a necessidade de criar uma Api em uma pequena quantia de tempo. Conversando com vários colegas, percebi que existem grandes frameworks, e que de fato são incríveis, mas que de forma geral, contém diversas funcionalidades e as vezes nem usamos, o que acaba gerando um peso em nossas aplicações. O Posa foi uma criação para projetos pessoais, mas que com muita solidariedade a comunidade de desenvolvedores Web, disponho da framework para utilização em projetos pessoais e acadêmicos, dada a credibilidade ao criador.

Sumário

Instalação

Ao baixar o Posa no **gitHub(BRSystem64/posa)**, ele já acompanha um conjunto de diretórios, e arquivos com as configurações mínimas para o desenvolvimento, a única recomendação, é que o desenvolvedor verifique se na raiz do projeto, tem a pasta app e dentro dela os subdiretórios Controllers, Models e Views. Caso não tenha, o desenvolvedor deve criar o diretório app(minusculo), e dentro dele, os três respectivos diretórios(1° letra maiscula e no plural).

Documentação

A documentação do Posa é simples, no geral, Posa trabalha com uma arquitetura similar a outras frameworks PHP. Caso já tenha usado Slim Framework, ou Laravel, não terá dificuldade alguma de se adaptar ao Posa, e caso não tenha utilizado nenhum framework, também não é um problema, o Posa é extremamente amigável aos novos desenvolvedores.

Como descrito na instalação, o posa já contém um conjunto de diretórios e arquivos primordiais para seu funcionamento. Portanto, o desenvolvedor deve estar ciente da arquitetura utilizada no Posa(descrito da sessão abaixo), a fim de que não haja desentendimento do desenvolvedor com a framework.

Arquitetura / Diretórios

A raiz do Posa é composta por 4 diretórios, são eles: *app, Posa, public, vendor*; e 1 arquivo *composer.json*. Abaixo, segue a definição de cada diretório e arquivo.

app: O Posa utiliza o padrão MVC para o desenvolvimento, nesse diretório há 3 sub diretórios que representam o padrão, são eles: Controllers, Models, Views.

OBS: Caso não tenha o diretório app em seu projeto, crie ele na raiz do projeto e dentro do mesmo crie os outros três diretórios: *Controllers, Models, Views*.

Posa: O diretório Posa é o Core de sua aplicação, aqui se pode encontrar diversos arquivos, com as responsabilidades de gerenciar as rotas, os requests e reponses, as middlewares e outros recursos utilizáveis pela framework.

public: O diretório public é onde está seu arquivo index, que inicia sua aplicação, nele também está localizado o arquivo *htaccess*.

vendor: O diretório vendor é onde ficam as dependências do seu projeto, por padrão o Posa utiliza o PSR-4, para definição dos autoloads.

composer.json: O arquivo composer.json é o gerenciador de dependências, aos novos desenvolvedores Web, ele não é exclusividade do Posa. Portanto é recomendável um compreendimento sobre o composer e vendor, caso o desenvolvedor queira gerenciar suas dependências no Posa.

Primeira Aplicação

Em seu arquivo principal(index.php), é necessário que a primeira tarefa seja importar os dois arquivos necessários para execução do Posa. São eles, autoload.php e o App.php, o 1° se encontra na pasta vendor, e o 2° na pasta Posa. Caso você tenha mantido o seu index.php dentro do public, o seu código ficar estar como a figura abaixo:

```php
<?php

require_once __DIR__."/../vendor/autoload.php";
require_once __DIR__."/../Posa/App.php";



?>
```

Fonte: autor (2019)

Agora que importamos o autoload e o App, precisamos instanciar a classe App, para utilizarmos os recursos do Posa. Utilizando o seguinte comando:

$app = new Posa\App;

Agora podemos definir nossa primeira rota. Para definir as rotas, utilizamos a variável $app, mais o verbo Http que desejarmos. Então ficaria algo como:

$app->get(param1, param2);

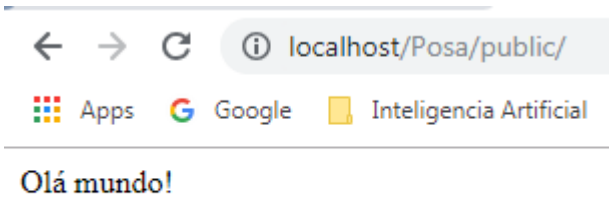
Observe que o nosso método requer que seja passado parâmetros para sua execução, o 1° parâmetros é a Url que existe em nosso sistema, e o segundo parâmetros pode ser uma classe e um método, ou apenas uma função anônima. Para exemplos práticos, vamos criar uma rota que exibirá a frase tão famosa 'hello world' ao nosso usuário. Para melhor compreensão, vamos utilizar a função anônima para essa tarefa, então se código deve ficar algo como o código abaixo:

```php
<?php

require_once __DIR__."/../vendor/autoload.php";
require_once __DIR__."/../Posa/App.php";

$app = new Posa\App;

$app->get('/Posa/public/', function($req, $res){

    echo "Olá Mundo!";

});

$app->run();

?>
```

Fonte: autor (2019)

Observe que a nossa função anônima recebe dois parâmetros $req, $res; que respectivamente são, request e response. Para desenvolvedores mais experientes, pode ser interessante a manipulação dessas variáveis, mas para início, nos contentaremos apenas com o básico. Prosseguindo a explicação, note que o método utilizado foi get, o primeiro parâmetros foi a rota**/Posa/public/,** ou seja, se você acessar o **localhost/Posa/public/** ele deve executar o que foi passado como segundo parâmetros, no caso, nossa função anônima. É importante ressaltar também, que no final é imprescindível ter o **$app->run().**

Após abrir o seu navegador, e acessar o localhost/Posa/public deve aparecer uma mensagem como a imagem abaixo:



Fonte: autor (2019)

Variáveis

Na versão Atual do Posa, é possível utilizar os quatros verbos principais do Http, são eles: Post, Get, Put, Delete.

Em nosso primeiro exemplo, utilizamos o GET para apresentar um Hello World, mas há casos em que precisamos passar alguma variável pela Url, como por exemplo, o nome de uma pessoa para exibir uma mensagem de boas-vindas.

No Posa, uma variável pode ser declarada na Url utilizando o **:variável**. O que resultaria em algo como:

$app->get('Posa/public/:variavel', param2);

É importante lembrar, que o Posa suportar passar N variaveis, ou seja, o desenvolvedor pode querer fazer algo como:

$app->get('Posa/public/:nome/:idade', param2);

Para exemplificar a teoria com a prática, vamos utilizar o arquivo do exemplo passado e apenas adicionarmos uma nova URL. O seu código deve parecer conforme a imagem abaixo:

```php
<?php

require_once __DIR__."/../vendor/autoload.php";
require_once __DIR__."/../Posa/App.php";

$app = new Posa\App;

$app->get('/Posa/public/', function($req, $res){

    echo "Olá Mundo!";

});

$app->get('/Posa/public/pessoa/:nome', function($req, $res){
    echo $req->params['nome']. ", bem vindo!";
});

$app->run();

?>
```

Fonte: autor (2019)

Note que criamos uma nova rota chamada /Posa/public/pessoa/:nome, e que ao entrar nessa rota, a nossa função é chamada, o nosso request ($req), passou um valor como parâmetro, que foi o nome da pessoa. Para acessar esse valor, utilizamos a variável $req junto com o atributo param e a chave.

OBS: A chave sempre vai ser o nome da variável(No caso da nossa URL a variável chama nome).

Nos exemplos acima, só utilizamos o método GET, mas caso a requisição tenha sido feita via POST ou PUT, o acesso a variável se dá na mesma forma, utilizando o $req->params['nome da variavel'].

Callable

Em nossos exemplos acima, nós utilizamos uma função anônima para execução de correspondência da nossa URL. Entretanto, por vezes, há um interesse em realizar a execução de algum método de uma determinada classe. Para que isso se torne possível, é necessário que as classes na qual definimos como classes chamáveis, estejam no diretório app/Controllers.

O Posa segue a ideia de que todas suas classes que possam vir ser instanciadas pelo acesso a uma URL, estejam efetivamente armazenadas em app/Controllers, conforme descreve a filosofia do MVC.

A chamada de um método de uma classe, é definido da seguinte forma:

$app->get('Posa/public/', 'Home@index');

Sendo o 1° parâmetro a nossa Url, conforme vimos anteriormente, e nosso segundo parâmetro o nome da classe(localizada no app/Controllers) e do método, ambos separados por um @.

Podemos então definir que há dois modos de se executar algo após a chamada de uma URL, são eles: função anônima e métodos de classe. E Suas respectivas implementação são:

```
$app->get('Posa/public/', function($req, $res){
    // seu código aqui
});
```

```
$app->get('Posa/public/', 'Classe@método');
```

Para melhor compreendermos o funcionamento de uma callable por método de uma classe, vamos criar uma classe no app/Controllers chamada HomeController com a namespace App\Controllers. É importante lembrar que a definição da namespace é algo imprescindível, já que estamos utilizando o autoload para carregas os arquivos.
O seu código deve estar como o código abaixo:

```php
<?php
namespace App\Controllers;

class HomeController{

    function index($req, $res){

        echo "Pagina index";
    }
}

?>
```

Fonte: autor (2019)

Há alguns pontos importantes a notarmos, o 1° é a definição da namespace, caso você não defina a namespace conforme o exemplo acima, o autload não carregará seu arquivo. O 2° ponto é que todo método que definirmos como correspondência das nossas rotas, devem obrigatoriamente existir em nossas classes.

No nosso exemplo criamos o método index na classe HomeController. Para utilizarmos esse método e classe na chamada de uma URL, voltamos ao index e definimos o método.

```
$app->get('Posa/public/home', 'HomeController@index');
```

Após utilizar essa URL no navegador, deve aparecer no seu browser a mensagem Pagina index.

Views

Até o momento, só trabalhamos com o controller, sem utilizar nenhum recurso gráfico, como o Css ou as propriedades do próprio HTML. O Posa não trabalha apenas com APIs, é possível responder mais do que apenas um JSON.

Para trabalharmos com as Views, a primeira regra que devemos saber, é que, todas nossas páginas, ficam armazenadas em app/Views. Dentro dessa pasta, você pode organiza-lo conforme for sua vontade, ou seja, você pode criar subdiretórios para organizar o seu site de acordo com sua vontade.

Vamos utilizar a nossa rota que acessa o index do HomeController para desenvolver nossa primeira view.

O controller que trabalhar com view, é necessário fazer a herança de uma classe chamada BaseController, que pertence ao namespace Posa. O primeiro passo então é extendermos a nossa classe. Sua classe HomeController, agora deve estar assim:

```
class HomeController extends \Posa\BaseController{
        // codigos aqui
}
```

Pronto, agora a sua classe já pode chamar qualquer arquivo que esteja na pasta view. Para isso, vá até a pasta View e crie um arquivo chamado home.html, e cole o texto abaixo:

```
<!DOCTYPE html>
<html>
<body>
<h1>Olá mundo!</h1>
</body>
</html>
```

Agora, em nosso HomeController, no método index, apageremos aquele echo e usaremos o código a seguir:

```
return $this->with($req, $res)->render("home.html");
```

Basicamente, o que estamos dizendo é retorne com nosso request e response a exibição da página home.html

Execute em seu navegador e deverá aparecer a mensagem Olá mundo!

As vezes, desejamos ter um layout padrão e dentro desse layout, diversas páginas sejam carregadas, como uma barra de menu, que independente da página, desejamos que ela esteja lá.

O Posa permite que de modo fácil e prático essa tarefa possa ser realizada, a apenas duas tarefas distintas da definição anterior. O primeiro passos é passar como parâmetro o layout, por exemplo:

```
return $this->with($req, $res)->render("home.html", "layout.html");
```

Caso o seu layout esteja em outro diretório, você pode definir da seguinte forma:

```
return $this->with($req, $res)->render("home.html","diretorio/layout.html");
```

O segundo passo, é irmos no arquivo de layout, e onde desejarmos que a outra pagina(home.html) apareça, definirmos o seguinte código:

```
$this->content();
```

O **with**, permite que os dados da requisição não sejam perdidos, por exemplo, caso o usuário tenha passado algum tipo de parâmetro, e esses deve ser exibidos na página, você pode utilizar a recuperação dos dados conforme já vimos acima, com o código:

```
$req->params['nome da variavel'];
```

Middleware

O Posa permite implementar middlewares de modo rápido e prático. As middlewares no Posa também podem ser chamadas de dois modos, igualmente vimos na sessão callable, utilizando função anônima ou um método de uma classe.

A definição de uma nova middleware para uma determinada URL, é simpels, basta utilizar o método **midd** antes do verbo. Conforme segue o exemplo abaixo:

```
$app->midd('parametro')get('Posa/public/home','HomeController@index');
```

Conforme foi descrito, o parâmetro do **midd** pode ser uma função anônima ou um método de uma classe. Lembrando que todo método do midd deve retornar algo, já que é possível criar diversas middlewares para uma única url e o fato do Posa cria uma Stack de Middlewares, e de fato elas só são executadas quando o a URL desejada é aquela URL em análise. Segue abaixo alguns exemplos de implementação de middlewares:

Exemplo1: Função Anônima

```
$app->midd(function($res, $req){
        echo "passando pela middleware 1";
            return this;
})get('Posa/public/home','HomeController@index');
```

Exemplo2: Método de uma Classe

```
$app->midd("MiddController@login")get('Posa/public/home','HomeController@index');
```

Exemplo3: Varias Middlewares

```
$app->midd("MidController@login")->("MidController@boasVindas")->
get('Posa/public/home','HomeController@index');
```