Slim

a micro framework for PHP

Rob Allen ~ @akrabat ~ October 2016

The C in MVC

Slim 3

- Created by Josh Lockhart (phptherightway.com)
- PSR-7 `Request` and `Response` objects
- Middleware architecture
- Built in DIC for configuration

HTTP Messages are the foundation

Request & Response

Request:

```
{METHOD} {URI} HTTP/1.1
Header: value1,value2
Another-Header: value

Message body
```

Response:

```
HTTP/1.1 {STATUS_CODE} {REASON_PHRASE}
Header: value
Some-Header: value

Message body
```

How do we do this in PHP?

Request:

- $_SERVER, $_GET, $_POST, $_COOKIE, $_FILES
- `apache_request_headers()`
- `php://input`

Response:

- `header()` / `http_response_code()`
- `header_list()` / `headers_sent()`
- `echo` (& `ob_*()` family)

PSR 7: HTTP messaging

OO interfaces to model HTTP

- `RequestInterface` (& `ServerRequestInterface`)
- `ResponseInterface`
- `UriInterface`
- `UploadedFileInterface`
- `StreamInterface`

PSR 7: Example

```php
/* Body implements Psr\Http\Message\StreamInterface */
$body = new Body(fopen('php://temp', 'r+'));
$body->write('Hello World');

/* Response implements Psr\Http\Message\ResponseInterface */
$response = new Response();
$response = $response->withStatus(200)
            ->withHeader('Content-Type', 'text/html')
            ->withBody($body);


/* Note: with Slim's Response: */
$response = $response->write("Hello world");
```

Key feature 1: Immutability

`Request`, `Response`, `Uri` & `UploadFile` are *immutable*

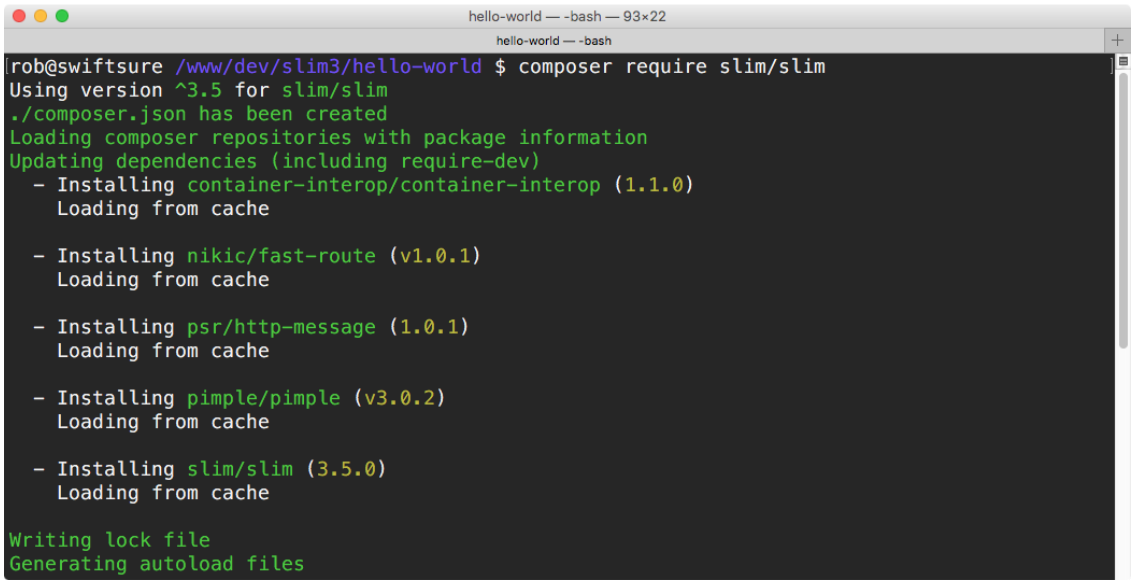
```
1 $uri = new Uri('https://api.joind.in/v2.1/events');
2 $uri2 = $uri->withQuery('?filter=upcoming');
3
4 $request = (new Request())
5     ->withMethod('GET')
6     ->withUri($uri2)
7     ->withHeader('Accept', 'application/json')
8     ->withHeader('Authorization', 'Bearer 0873418d');
```

Key feature 2: Streams

Message bodies are *streams*

```
1 $body = new Stream();
2 $body->write('<p>Hello');
3 $body->write('World</p>');
4
5 $response = (new Response())
6     ->withStatus(200, 'OK')
7     ->withHeader('Content-Type', 'application/header')
8     ->withBody($body);
```

Installation



index.php

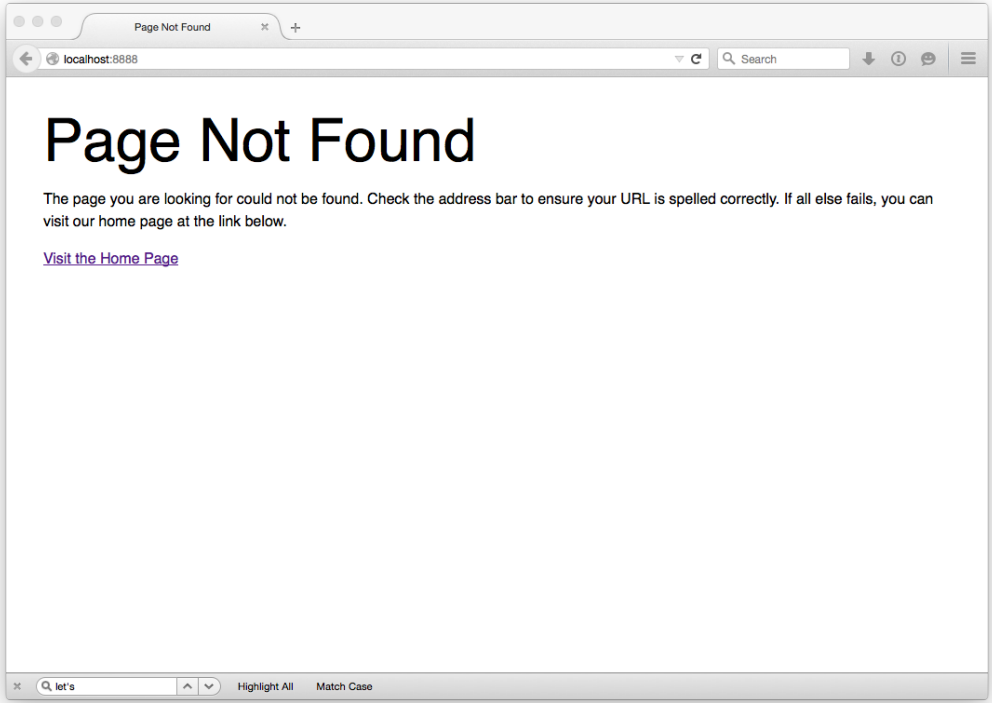
```php
<?php
// Setup autoloader
require __DIR__ . '/../vendor/autoload.php';

// Prepare app
$app = new \Slim\App();

// Run app
$app->run();
```

Run it

```
php -S localhost:8888
```


Page Not Found
localhost:8888
Search
Page Not Found
The page you are looking for could not be found. Check the address bar to ensure your URL is spelled correctly. If all else fails, you can visit our home page at the link below.
Visit the Home Page
let's
Highlight All
Match Case

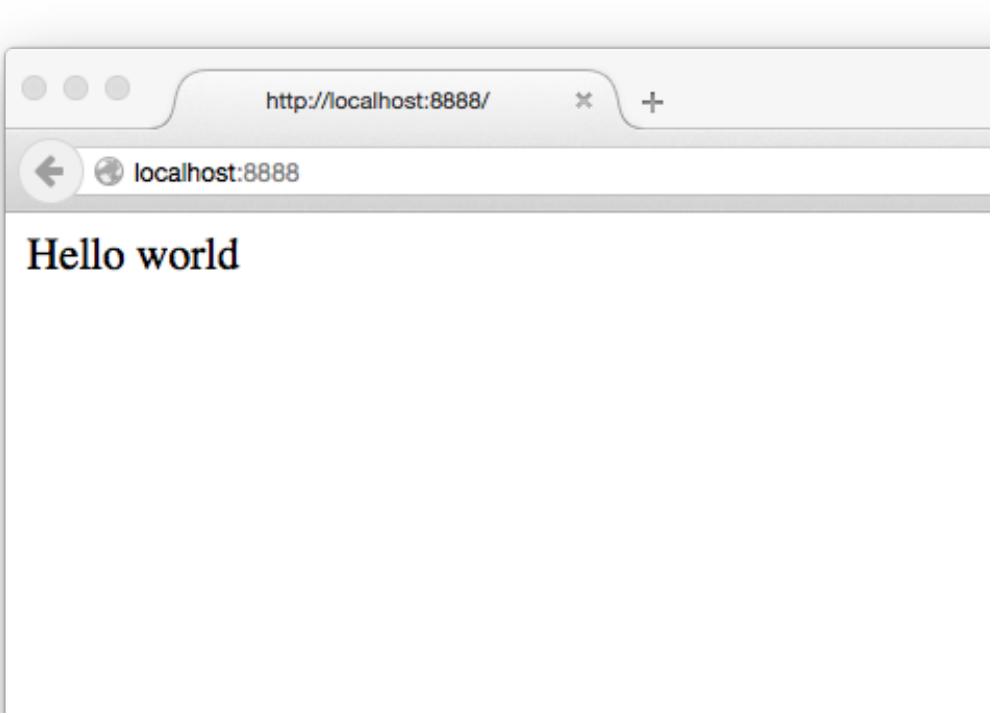
Routes

Routes

```php
<?php
require __DIR__ . '/../vendor/autoload.php';
$app = new \Slim\App();


$app->get('/hi', function ($request, $response) {
    $response->write("Hello world");
    return $response;
});


$app->run();
```


http://localhost:8888/
localhost:8888
Hello world

Let's look at that route

```php
<?php
require __DIR__ . '/../vendor/autoload.php';
$app = new \Slim\App();


$app->get('/hi', function ($request, $response) {
    $response->write("Hello world");
    return $response;
});


$app->run();
```

Let's look at that route

```
$app->get('/hi', function ($request, $response) {
    $response->write("Hello world");
    return $response;
});
```

Routes have a method


```
$app->get('/hi', function ($request, $response) {
    $response->write("Hello world");
    return $response;
});
```

HTTP Method

- `$app->get()`
- `$app->post()`
- `$app->put()`
- `$app->patch()`
- `$app->delete()`
- `$app->options()`

Multiple methods:

- `$app->map(['get', 'post'])`

Routes have a pattern


```
$app->get('/hi', function ($request, $response) {
    $response->write("Hello world");
    return $response;
});
```

URI pattern

- Literal string match

```
$app->get('/hello', $callable);
```

URI pattern

- Literal string match

```
$app->get('/hello', $callable);
```

- Placeholders are wrapped in *{ }*

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

URI pattern

- Literal string match

```
$app->get('/hello', $callable);
```

- Placeholders are wrapped in *{ }*

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

- Optional segments are wrapped with *[ ]*

```
$app->get('/news[/{year}[/{month}]]', $callable);
```

URI pattern

- Literal string match

```
$app->get('/hello', $callable);
```

- Placeholders are wrapped in *{ }*

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

- Optional segments are wrapped with *[ ]*

```
$app->get('/news[/{year}[/{month}]]', $callable);
```

- Constrain placeholders via Regex

```
$app->get('/user/{id:\d+}', $callable); // numbers only
```

Accessing placeholders

Use `$args`:

```
$app->get('/user/[{name}]', function ($request, $response, $args) {
        $name = $args['name'] ?? 'world';
        // do stuff
});
```

Accessing placeholders

Use `$args`:

```
$app->get('/user/[{name}]', function ($request, $response, $args) {
        $name = $args['name'] ?? 'world';
        // do stuff
});
```

or use PSR-7's `getAttribute()`:

```
$app->get('/user[/{name}]', function ($request, $response, $args) {
        $name = $request->getAttribute('name', 'world');
        // do stuff
});
```

Route groups

Organise routes into logical groups

```
$app->group('/user', function () {
    $this->get('/login', function ($req, $res) {});    //  /user/login
    $this->post('/auth', function ($req, $res) {});    //  /user/auth
    $this->get('/{name}', function ($req, $res) {});   //  /user/rob
});
```

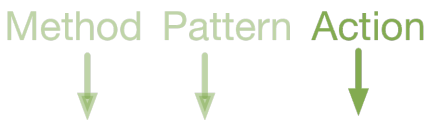
Named routes

```php
// Name the route
$app->get('/user/{name}', function (...) {...})
    ->setName('profile');


// build link:
$link = $this->router->urlFor('profile', ['name' => 'Rob']);
```

creates: `/user/Rob`

Routes have an action




```
$app->get('/hi', function ($request, $response) {
    $response->write("Hello world");
    return $response;
});
```

An action

- Is simply a PHP `callable`
- Takes three arguments:
  - PSR-7 `ServerRequest`
  - PSR-7 `Response`
  - array of URI arguments
- Must return a `Response`

Action signature

```php
use Slim\Http\Request;
use Slim\Http\Response;

function (Request $request, Response $response, $args) {

    // do stuff

    // return a Response
    return $response;
});
```

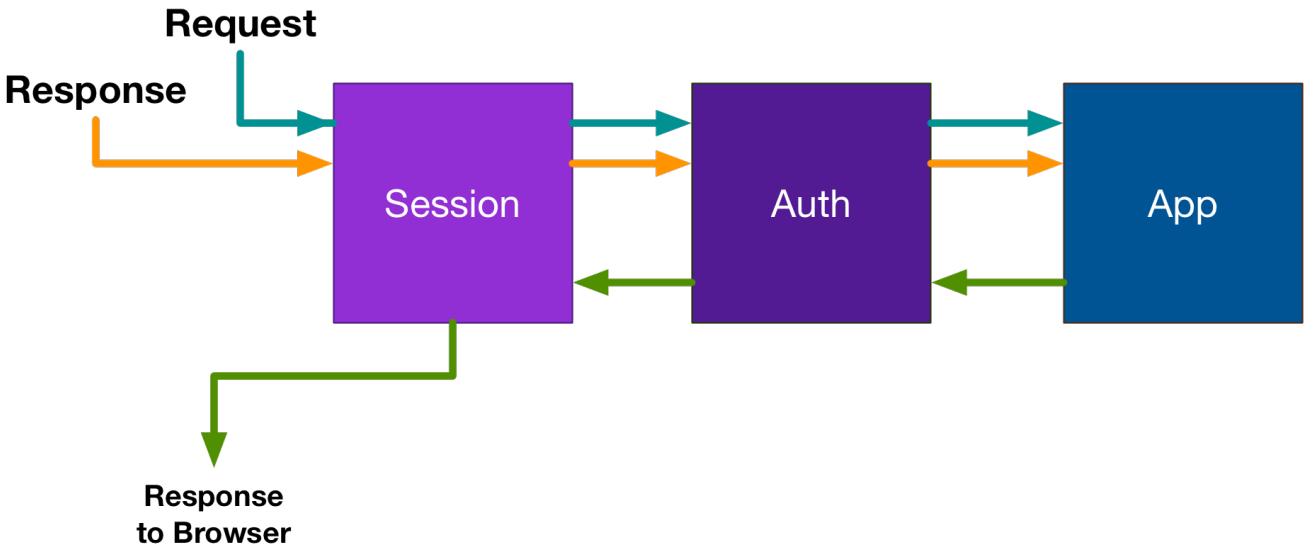
Middleware

Middleware

Middleware is code that exists between the request and response, and which can take the incoming request, perform actions based on it, and either complete the response or pass delegation on to the next middleware in the queue.

Matthew Weier O'Phinney

Middleware



What can you do with Middleware?

Session
Logging
Authentication
HTTP cache
CSP

CORS
Firewall
Throttling
Honeypot
etc...

CSRF
Debug bar
Error handling
Validation

Application middleware

```php
$timer = function ($request, $response, $next) {
  // before
  $start = microtime(true);

  // call next middleware
  $response = $next($request, $response);

  // after
  $taken = microtime(true) - $start;
  $response->write("<!-- Time taken: $taken -->");

  return $response;
}

$app->add($timer);
```

Data transfer between middleware

```
class IpAddressMiddleware
{
  public function __invoke($request, $response, $next)
  {
      ipAddress = $this->determineClientIpAddress($request);
      $request = $request->withAttribute('ip_address', ipAddress);

      return $next($request, $response);
  }

  private function determineClientIpAddress($request) {
    // ...
  }
}
```

Data transfer between middleware

```
class GateKeeperMiddleware
{
  public function __invoke($request, $response, $next)
  {
      $ipAddress = $request->getAttribute('ip_address');

        if (!in_array($ipAddress, $this->allowedIpAddresses)) {
            return $response->withStatus(403);
        }

        return $next($request, $response);
  }
}
```

Route middleware

Do stuff before or after this action!

```
$app->get('/hello/{name}', function (...) {...})
    ->add(function ($request, $response, $next) {

      // before: sanitise route parameter
      $name = strip_tags($request->getAttribute('name'));
      $request = $request->withAttribute('name', $name);

      return $next($request, $response);
    })
```

Leverage middleware

Application level:

- Authentication
- Navigation
- Session

Route level:

- Access control
- Validation

Slim Extras

Provided separately from Slim 3

Add via Composer

- `slim/slim-httpcache` - Cache-Control/Etag support
- `slim/slim-csrf` - CSRF protection
- `slim/slim-flash` - Transient messages
- `slim/twig-view` - Twig view layer

Flash messages

```
$ composer require slim/flash
```

Register with `$app`:

```php
// start PHP session
session_start();

$app = new Slim\App();

$container = $app->getContainer();
$container['flash'] = function () {
    return new \Slim\Flash\Messages();
};
```

Store message

```
$app->post('/blog/edit', function ($req, $res, $args) {

  ...

  // Set flash message for next request
  $this->flash->addMessage('result', 'Post updated');

  // Redirect
  return $res->withStatus(302)
             ->withHeader('Location', '/blog/list');
});
```

Retrieve message

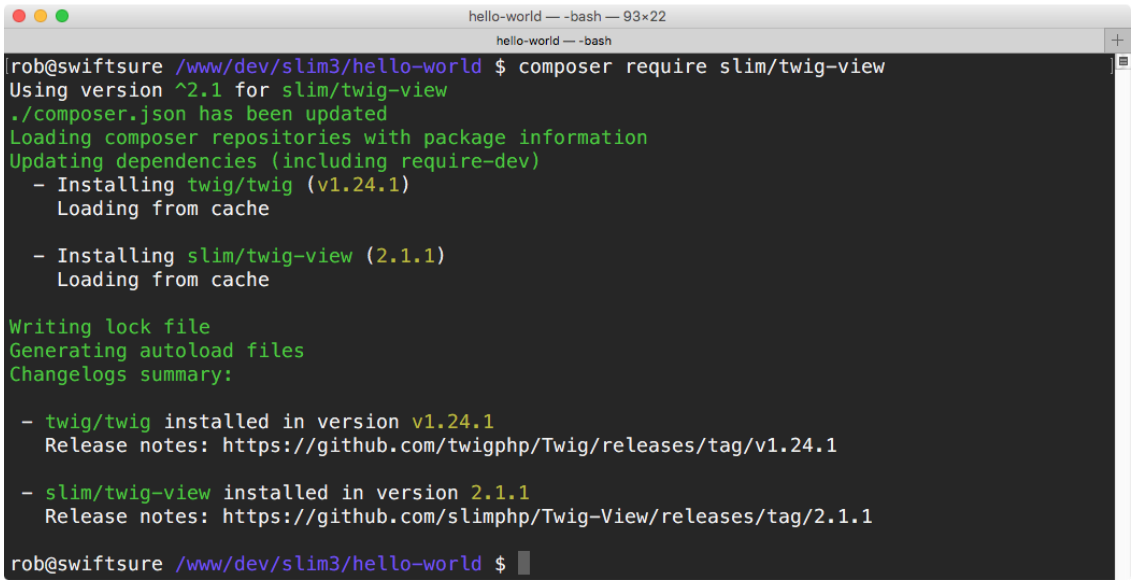
```php
$app->get('/blog/list', function ($req, $res) {

  // Get messages
  $messages = $this->flash->getMessages();

  // render
  return $response->write($messages['result'][0]);
});
```

Twig views

Installation



Configure the view

```php
// settings.php
return [
    // ...

    'view' => [
        'template_path' => 'app/templates',
        'twig_options' => [
            'cache' => 'cache/twig',
            'debug' => true,
            'auto_reload' => true,
        ],
    ],
];
```

Register the view

```
// Fetch DI Container
$container = $app->getContainer();

// Register Twig
$container['view'] = function ($c) {

  $view = new \Slim\Views\Twig(
    $c['settings']['view']['template_path'],
    $c['settings']['view']['twig_options']
  );

  $view->addExtension(new Slim\Views\TwigExtension(
    $c['router'], $c['request']->getUri()));
  return $view;
};
```

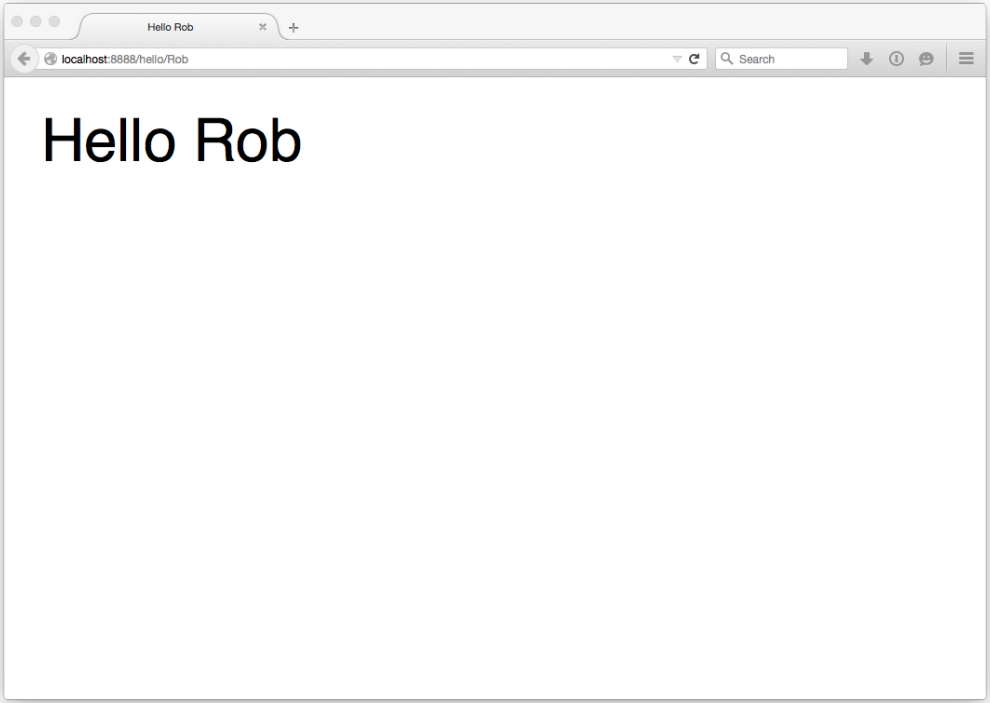
Template

```
<html>
  <head>
    <title>Hello {{ name }}</title>
    <link rel="stylesheet" href="/css/style.css">
  </head>
  <body>
    <h1>Hello {{ name }}</h1>
  </body>
</html>
```

Render

```
$app->get('/hello/{name}', function ($request, $response, $args) {
    $body = $this->view->fetch('hello.twig', [
      'name' => $args['name'],
    ]);

    return $response->write($body);
});
```


Hello Rob
localhost:8888/hello/Rob
Search
Hello Rob

Thoughts on organising your application

Directory layout

Choose your own file organisation. This is mine.

```
/
├── app/
├── cache/
├── public/
│   ├── css/
│   ├── js/
│   └── index.php
├── vendor/
├── composer.json
└── composer.lock
```

app holds my code

```
app/
├── src/
│   ├── App/
│   ├── Photos/
│   │   ├── FlickrService.php
│   │   └── Photo.php
├── templates/
│   ├── layout.twig
│   └── app/
│       └── home/
│           └── list.twig
├── dependencies.php
├── middleware.php
├── routes.php
└── settings.php
```

Keep index.php clean

```php
// Prepare app
$settings = require __DIR__ . '/../app/settings.php';
$app = new \Slim\App($settings);

// Register dependencies with the DIC
require __DIR__ . '/../app/src/dependencies.php';

// Register middleware
require __DIR__ . '/../app/src/middleware.php';

// Register routes
require __DIR__ . '/../app/src/routes.php';

// Run app
$app->run();
```

Autoload via composer

Add an `autoload` section to `composer.json`

```
"autoload": {
    "psr-4": {
        "App\\": "app/src/App",
        "Photos\\": "app/src/Photos"
    }
},
```

Generate:

```
$ composer dump-autoload
Generating autoload files
```

Configuration

```php
<?php
// settings.php
return [
    // app specific
    'flickr' => [
    ],

    'db' => [
    ],

    // view
    'view' => [
    ],
];
```

DI is your friend

```php
// dependencies.php

// Register FlickrService into DIC
$container = $app->getConatiner();
$container['FlickrService'] = function ($c) {

    $key    = $c['settings']['flickr']['key'];
    $secret = $c['settings']['flickr']['secret'];

    return new Photos\FlickrService($key, $secret);
};
```

All routes in a single file

```php
// routes.php
$app->get('/', function ($request, $response) {

    $flickr  = $this->FlickrService;
    $keyword = $request->getParam('keyword');

    $list = $flickr->search($keyword);

    $body = $app->view->fetch('list.twig', [
            'keyword' => $keyword,
            'list' => $list,
        ]);
    return $response->write($body);
});
```

Controller DI

```php
// dependencies.php
$container = $app->getContainer();
$container['Photos\PhotosController'] = function ($c) {
  $flickr = $c['FlickrService'];
  $view   = $c['view'];
  return new Photos\PhotosController($flickr, $view);
};
```

Controller DI

```php
// dependencies.php
$container = $app->getContainer();
$container['Photos\PhotosController'] = function ($c) {
  $flickr = $c['FlickrService'];
  $view   = $c['view'];
  return new Photos\PhotosController($flickr, $view);
};

// routes.php
$app->group('/photos', function () {
  $app->get('/list', 'Photos\PhotosController:list')
      ->setName('list-photos');
  $app->post('/upload', 'Photos\PhotosController:upload')
      ->setName('upload-photo');
});
```

Controller

```php
namespace Photos;

final class PhotosController
{
    private $flickr;
    private $view;

    public function __construct($flickr, $view)
    {
        $this->flickr = $flickr;
        $this->view = $view;
    }
```

Controller (cont)

```
    // action method
    public function list($request, $response)
    {
        $keyword = $request->getParam('keyword');
        $list = $this->flickr->search($keyword);

        $body = $this->view->fetch('list.twig', [
            'keyword' => $keyword,
            'list' => $list,
        ]);

        return $response->write($body);
    }
}
```

Resources

- http://www.slimframework.com/docs
- https://github.com/slimphp/Slim
- http://akrabat.com/category/slim-framework/

- Slack: https://slimphp-slack-invite.herokuapp.com
- Forum: http://discourse.slimframework.com
- IRC: #slimphp on Freenode

Questions?

https://joind.in/talk/d07b9

Rob Allen - http://akrabat.com - @akrabat

Thank you!

https://joind.in/talk/d07b9

Rob Allen - http://akrabat.com - @akrabat