

Getting started with Slim 3

Rob Allen - June 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

PSR 7: HTTP messaging

OO interfaces to model HTTP

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

PSR 7: Example

```
1 /* Body implements Psr\Http\Message\StreamInterface */
2 $body = new Body(fopen('php://temp', 'r+'));
3 $body->write('Hello World');
4
5 /* Response implements Psr\Http\Message\ResponseInterface
6 $response = new Response();
7 $response = $response->withStatus(200)
8             ->withHeader('Content-Type', 'text/html')
9             ->withBody($body);
10
11
12 /* Note: with Slim's Response: */
13 $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

```
[rob@swiftsure /www/dev/slim3/hello-world $ composer require slim/slim
Using version ^3.4 for slim/slim
./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
- Installing container-interop/container-interop (1.1.0)
  Loading from cache

- Installing nikic/fast-route (v1.0.0)
  Loading from cache

- Installing psr/http-message (1.0)
  Loading from cache

- Installing pimple/pimple (v3.0.2)
  Loading from cache

- Installing slim/slim (3.4.2)
  Loading from cache

Writing lock file
Generating autoload files
```


index.php

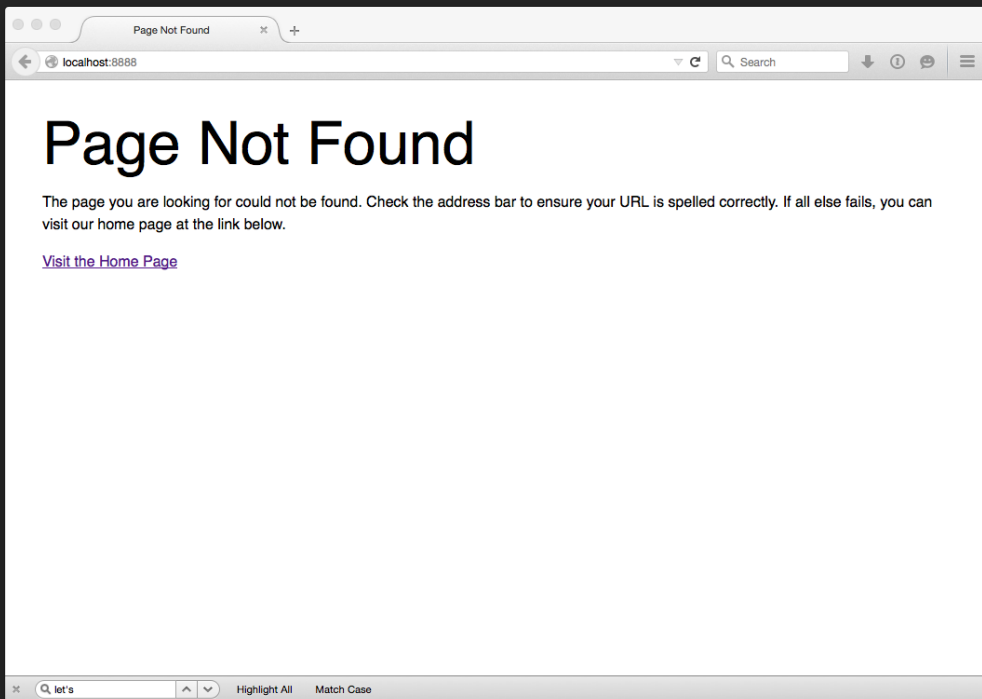
```
<?php
// Setup autoloader
require __DIR__ . '/../vendor/autoload.php';

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

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

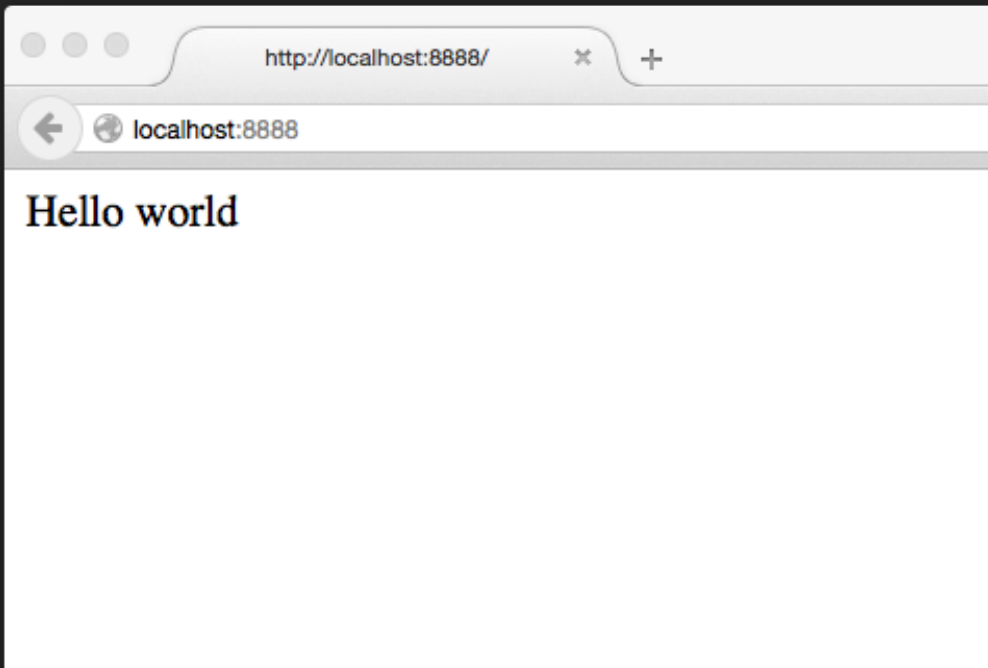
$app->run();
```

Routes

Method Pattern Action



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



Method

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

Multiple methods:

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

Dynamic routes

```
$app->get(
    '/hello/{name}',
    function ($request, $response, $args) {
        $name = $args['name'];
        $name = htmlspecialchars($name);
        return $response->write("Hello $name");
    });
```

It's just Regex

```
$app->get('/user/{id:0-9+}', $callable);  
$app->get('/hello/{name:[\w]+}', $callable);  
  
$app->get('/hello[/]{name:[\w]+}', $callable);
```

Optional segments

Use `getAttribute()`:

```
$app->get(
  '/hello[/{name:[\w]+}]',
  function ($request, $response, $args) {
    $name = $request->getAttribute('name', 'world');
    $name = htmlspecialchars($name);
    return $response->write("Hello $name");
  }
);
```

Route groups

```
$app->group('/books', function () use ($app) {
    $app->get('', function ($req, $res) {
        // Return list of books
    });
    $app->post('', function ($req, $res) {
        // Create a new book
    });
    $app->get('/{id:\d+}', function ($req, $res, $args) {
        // Return a single book
    });
    $app->put('/{id:\d+}', function ($req, $res, $args) {
        // Update a book
    });
});
```

Route groups

```
$app->group('/api', function () use ($app) {  
    $app->group('/books', function () use ($app) {  
        // routes for /api/books here  
    });  
    $app->group('/authors', function () use ($app) {  
        // routes for /api/authors here  
    });  
});
```

Named routes

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

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

creates: /hello/Rob

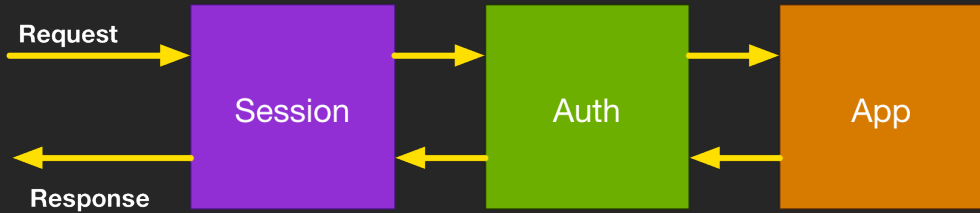
Middleware

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



Application middleware

```
$timer = function ($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);
```

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:dev-master
```

Register with \$app:

```
session_start();
```

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

```
$container = $app->getContainer();
```

```
$container->register(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

```
$app->get('/blog/list', function ($req, $res) {  
  
    // Get messages  
    $messages = $this->flash->getMessages();  
  
    // render  
    return $response->write($messages['result'][0]);  
});
```


Twig views

Installation

```
[rob@swiftsure /www/dev/slim3/hello-world $ composer require slim/twig-view
Using version ^2.1 for slim/twig-view
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
- Installing twig/twig (v1.24.1)
  Downloading: 100%

- Installing slim/twig-view (2.1.1)
  Loading from cache

Writing lock file
Generating autoload files
```

Configure the view

```
// 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']
    );

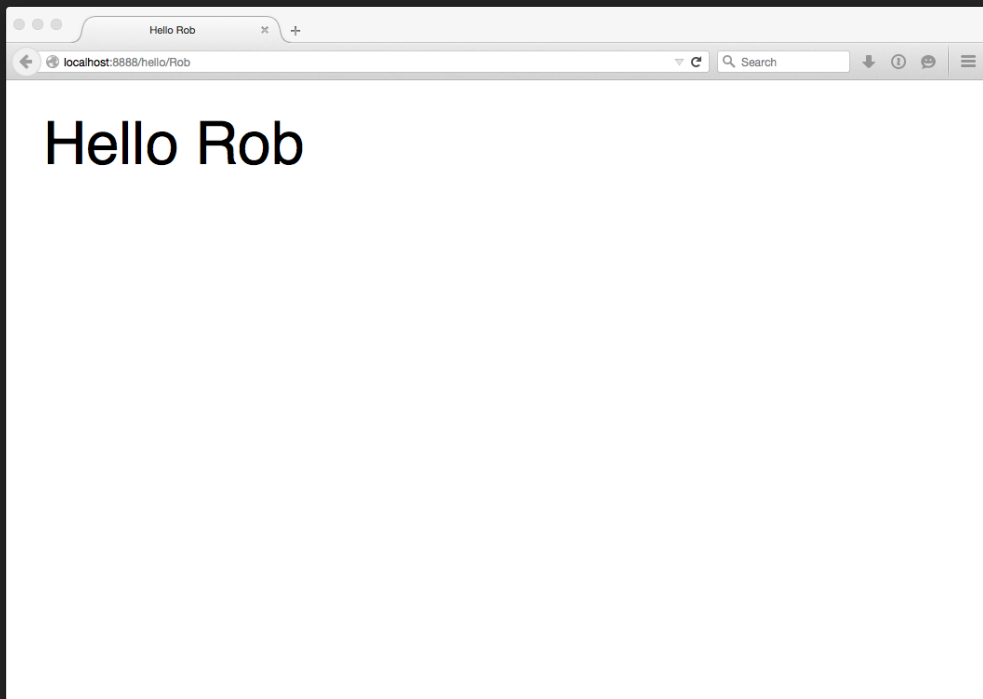
    $view->addExtension(new Slim\Views\TwigExtension(
        $c['router'], $c['request']->getUri()));
    return $view;
};
```

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

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

```
// 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
return [
    // app specific
    'flickr' => [
    ],

    'db' => [
    ],

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

DI is your friend

```
// 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
$app->get('/', function ($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);
});
```

Use the DIC within routes

```
// 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('/photos', 'Photos\PhotosController:list')
        ->setName('list-photos');
    $app->post('/upload', 'Photos\PhotosController:upload')
        ->setName('upload-photo');
});
```


Controller

```
namespace Photos;
```

```
final class PhotosController
```

```
{
```

```
    private $flickr;
```

```
    private $view;
```

```
    public function __construct($flickr, $view)
```

```
    {
```

```
        $this->flickr = $flickr;
```

```
        $this->view = $view;
```

```
    }
```

Controller (cont)

```
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>
- <http://www.slimframework.com/docs>
- <https://github.com/slimphp/Slim> (develop branch)
- <http://akrabat.com/category/slim-framework/>
- IRC: #slimphp on Freenode
- Forum: <http://discourse.slimframework.com>
- Slack: <https://slimphp-slack-invite.herokuapp.com>

Questions?

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

Thank you!

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