Getting started with Slim 3

Rob Allen - May 2015

19

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

Expecting first beta early June 2015

PSR 7: HTTP messaging

- Provides for a uniform access to HTTP messages
- A set of interfaces for requests and responses
- Value objects are *immutable*
- Body is a stream

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");
```

Installation

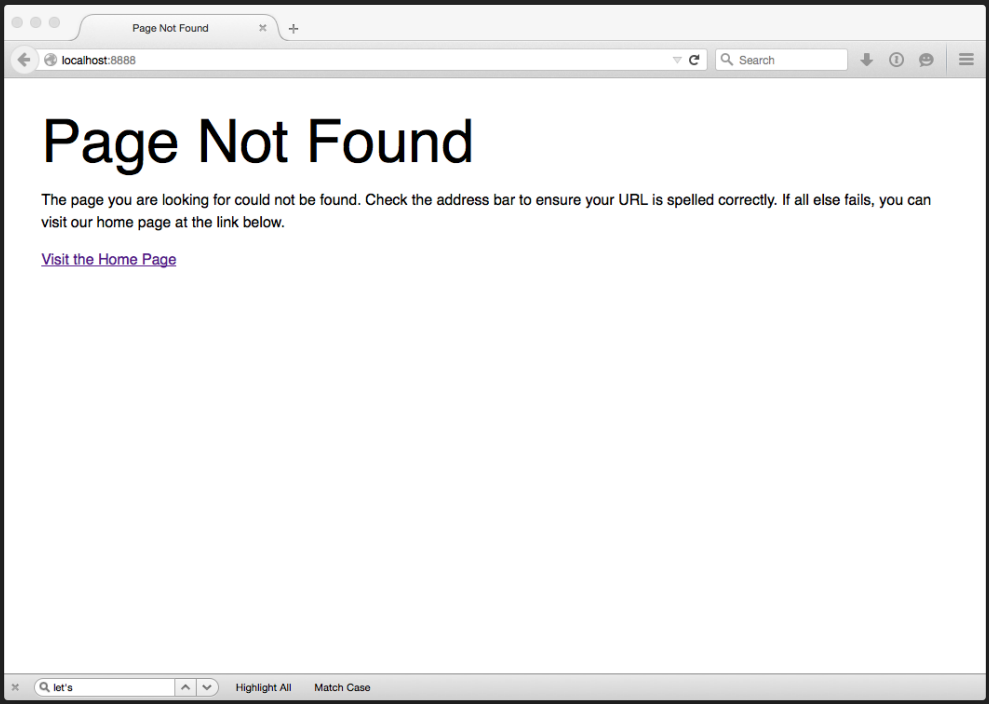
```
rob@swiftsure /www/dev/slim3/tmp $ composer require slim/slim:dev-develop
./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
  - Installing container-interop/container-interop (1.1.0)
    Loading from cache

  - Installing nikic/fast-route (v0.4.0)
    Loading from cache

  - Installing pimple/pimple (v3.0.0)
    Loading from cache

  - Installing psr/http-message (0.11.0)
    Loading from cache

  - Installing slim/slim (dev-develop e499ee4)
    Cloning e499ee4d3808485881a8f4a29dee04f70bc01d6f

Writing lock file
Generating autoload files
```

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
```

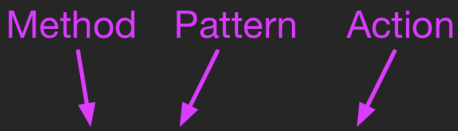

Page Not Found
localhost:8888
Search
Page Not Found
The page you are looking for could not be found. Check the address bar to ensure your URL is spelled correctly. If all else fails, you can visit our home page at the link below.
Visit the Home Page
let's
Highlight All
Match Case

Routes

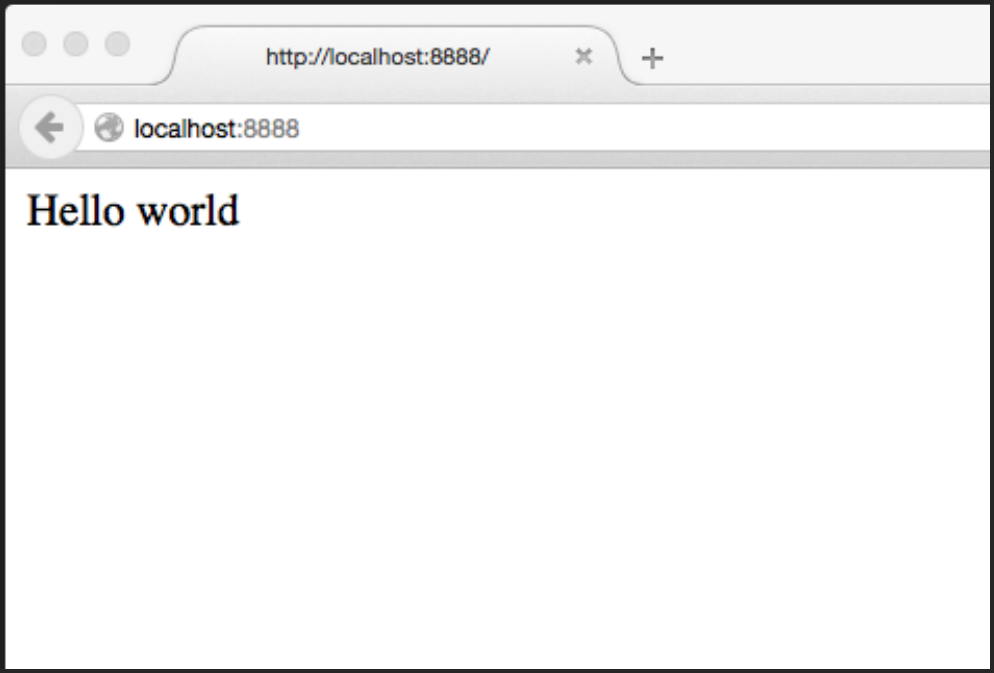
Routes

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


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


$app->run();
```

Routes




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


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

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:\d+}', $callable);

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

$app->get('/hello{a:/{0,1}}{name:[\w]*}', $callable);
```

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 = $app->router->urlFor('hi', ['name' => 'Rob']);
```

creates: `/hello/Rob`

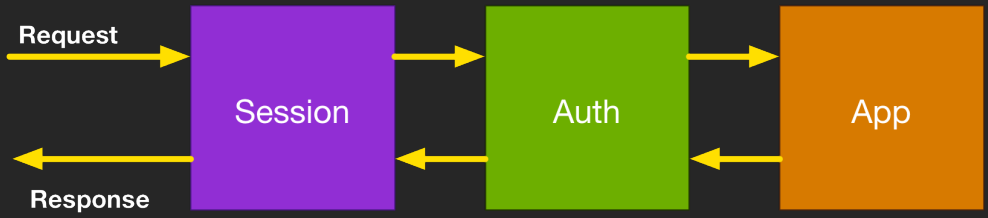
Middleware

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 your 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/nataero $ composer require slim/twig-view
Using version ~1.0 for slim/twig-view
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
  - Installing twig/twig (v1.18.1)
    Loading from cache

  - Installing slim/twig-view (1.0)
    Loading from cache

Writing lock file
Generating autoload files
rob@swiftsure /www/dev/slim3/nataero $
```

Configure the view

```php
<?php
return [
    // ...

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

Register the view

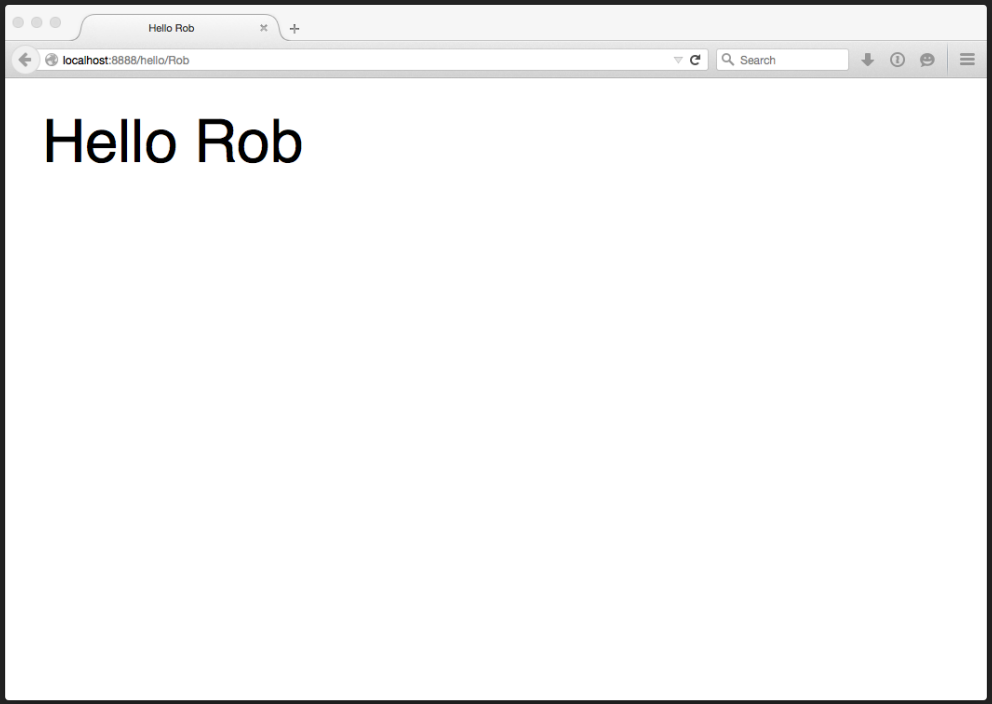
```
// Create the view object
$view = new \Slim\Views\Twig(
    $settings['view']['template_path'],
    $settings['twig']);

// add extensions
$twig = $view->getEnvironment();
$twig->addExtension(new Twig_Extension_Debug());

$app->register($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/
│   ├── Nataero/
│   │   ├── FlickrService.php
│   │   └── Photo.php
├── templates/
│   ├── layout.twig
│   └── 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": {
          "Nataero\\": "app/src/Nataero"
      }
  },
```

Generate:

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

Configuration

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

    'db' => [
    ],

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

DI is your friend

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

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

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

All routes in a single file

```php
<?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

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


// routes.php
$app->get('/', 'Nataero\PhotosController:listPhotos')
    ->setName('list-photos');
$app->post('/save', 'Nataero\PhotosController:saveSearch')
    ->setName('save-photos');
```

Controller

```
namespace Nataero;

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

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

Controller (cont)

```
    public function listPhotos($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://slimframework.com
- http://docs-new.slimframework.com (incomplete!)
- https://github.com/slimphp/Slim (develop branch)
- http://akrabat.com/category/slim-framework/