Slim

a micro framework for PHP

Rob Allen ~ @akrabat ~ November 2017

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
```

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);
```

Let's talk about Slim

Hello world

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


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


$app->run();
```

Let's look at that route

Method

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

Let's look at that route


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

Let's look at that route


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

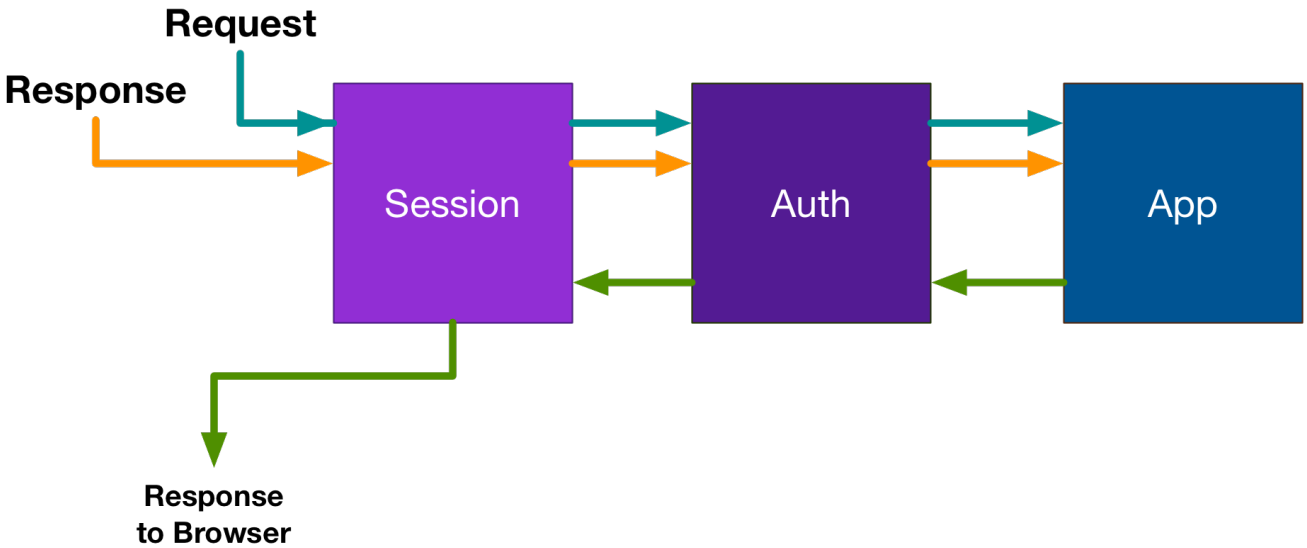
Middleware

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



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

```php
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

```php
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!

```
$app->get('/hi/{name}', function (...) {...})
    ->add(function ($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 template rendering
- `slim/php-view` - PHP view template rendering

Flash messages: Store

```php
$app->post('/blog/edit', function ($request, $response, $args) {

  // save data to database ...

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

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

Flash messages: Retrieve

```
$app->get('/blog/list', function ($request, $response) {

    // get $list of blogs ...

    // Get message
    $message = $this->flash->getFirstMessage('result');

    // Render page with Twig
    $html = $this->twig->fetch('blog.list.twig', [
        'message' => $message,
        'list' => $list,
      ]);
    return $response->write($html);
});
```

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();
```

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

```
// routes.php
$app->get('/list', 'Photos\PhotosController:list');
$app->post('/upload', 'Photos\PhotosController:upload');
$app->get('/{id:\d+}', 'Photos\PhotosController:view');
```

Register your controller with DIC

```php
// dependencies.php
$container = $app->getContainer();

$container['Photos\PhotosController'] = function ($c) {
    $flickr = $c['FlickrService'];
    $view   = $c['view'];
    return new Photos\PhotosController($flickr, $view);
};
```

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

Thank you!

https://joind.in/talk/4dbf6

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