Building an API with Slim 3

Rob Allen ~ @akrabat
June 2016

Let's start with Slim 3

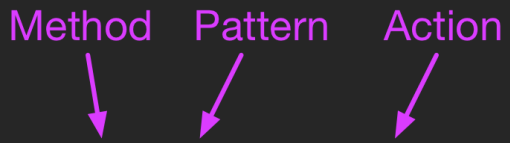
Hello world

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


$app->get('/ping', function ($request, $response) {
    $body = json_encode(['ack' => time()]);
    $response->write($body);
    $response = $response->withHeader(
        'Content-Type', 'application/json');
    return $response;
});

$app->run();
```

Hello world




```
$app->get('/ping', function ($request, $response) {
    $body = json_encode(['ack' => time()]);
    $response->write($body);
    $response = $response->withHeader(
        'Content-Type', 'application/json');
    return $response;
});
```

Hello world

```
$ http --json http://localhost:8888/ping
HTTP/1.1 200 OK
Connection: close
Content-Length: 18
Content-Type: application/json
Host: localhost:8888
X-Powered-By: PHP/5.6.14

{
    "ack": 1445111794
}
```

Slim 3 is a PSR-7 microframework

It's all about HTTP

Request:

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

Message body
```

Response:

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

Message body
```

PSR 7: HTTP messaging

OO interfaces to model HTTP

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

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 APIs

What makes a good API?

A good API

- HTTP method negotiation
- Content-type handling
- Honour the Accept header
- Error handling
- Versions

HTTP method negotiation

HTTP method negotiation

```
$ http --json PUT http://localhost:8888/ping
HTTP/1.1 405 Method Not Allowed
Allow: GET
Connection: close
Content-Length: 53
Content-type: application/json
Host: localhost:8888
X-Powered-By: PHP/5.6.14

{
    "message": "Method not allowed. Must be one of: GET"
}
```

HTTP method routing

```
$app->get('/author', function($req, $res) {});
$app->post('/author', function($req, $res) {});

$app->get('/author/{id}', function($req, $res) {});
$app->put('/author/{id}', function($req, $res) {});
$app->patch('/author/{id}', function($req, $res) {});
$app->delete('/author/{id}', function($req, $res) {});

$app->any('/author', function($req, $res) {});
$app->map(['GET', 'POST'], '/author', /* … */);
```

Dynamic routes

```
 1 $app->get('/author/{id}',
 2   function($request, $response, $args) {
 3     $id = $args['id'];
 4     $author = $this->authors->loadById($id);
 5
 6     $body = json_encode(['author' => $author]);
 7     $response->getBody()->write($body);
 8
 9     $response = $response->withHeader(
10           'Content-Type', 'application/json');
11
12     return $response;
13 });
```

It's just Regex

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

// optional segments
$app->get('/author[/{id:\d+}]', $callable);
$app->get('/news[/{y:\d{4}}[/{m:\d{2}}]]', $callable);
```

Content-type handling

Content-type handling

The *Content-type* header specifies the format of the incoming data

```
$ curl -X "POST" "http://localhost:8888/author" \
  -H "Content-Type: application/json" \
  -d '{ "name":"Terry Pratchett" }'
```

Read with getBody()

```
1 $app->post('/author',
2   function ($request, $response, $args) {
3     $data = $request->getBody();
4
5     return $response->write(print_r($data, true));
6   }
7 );
```

Output:

```
'{ "name":"Terry Pratchett" }'
```

Read with getParsedBody()

```
1 $app->post('/author',
2   function ($request, $response, $args) {
3     $data = $request->getParsedBody();
4
5     return $response->write(print_r($data, true));
6   }
7 );
```

Output:

```
Array
(
    [name] => Terry Pratchett
)
```

This also works with XML

```
curl -X "POST" "http://localhost:8888/author" \
  -H "Content-Type: application/xml" \
  -d "<author><name>Terry Pratchett</name></author>"
```

Output:

```
Array
(
    [name] => Terry Pratchett
)
```

And form data

```
curl -X "POST" "http://localhost:8888/author" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  --data-urlencode "name=Terry Pratchett"
```

Output:

```
Array
(
    [name] => Terry Pratchett
)
```

Other formats? e.g. CSV

```
name,dob
Terry Pratchett,1948-04-28
Andy Weir,1972-06-17
```

as curl:

```
curl -X "POST" "http://localhost:8888/author" \
  -H "Content-Type: text/csv" \
  -d $'name,dob
Terry Pratchett,1948-04-28
Andy Weir,1972-06-17'
```

Register media type

```php
$request->registerMediaTypeParser(
  'text/csv',
  function ($input) {
    $data = str_getcsv($input, "\n");
    $keys = str_getcsv(array_shift($data));

    foreach ($data as &$row) {
      $row = str_getcsv($row);
      $row = array_combine($keys, $row);
    }

    return $data;
  }
);
```

Result

```
Array
(
    [0] => Array
        (
            [name] => Terry Pratchett
            [dob] => 1948-04-28
        )

    [1] => Array
        (
            [name] => Andy Weir
            [dob] => 1972-06-17
        )
)
```

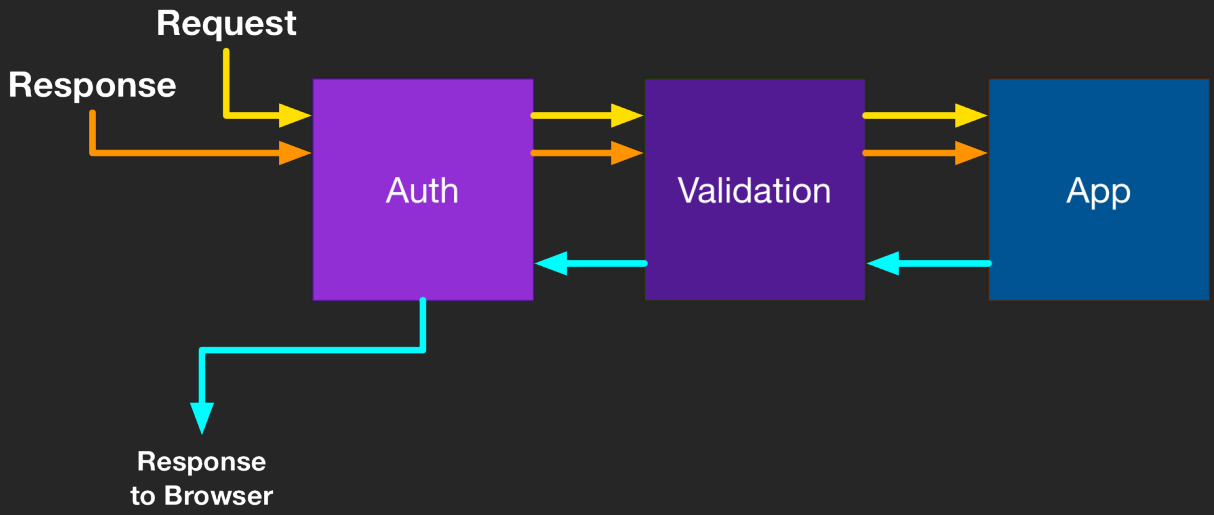
Middleware

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

Take a request, return a response



Middleware

```
 1 function ($request, $response, callable $next = null)
 2 {
 3     /* do something with $request before */
 4
 5     /* call through to next middleware */
 6     $response = $next($request, $response);
 7
 8     /* do something with $response after */
 9
10     return $response;
11 }
```

Media type middleware

```
1 $app->add(function ($request, $response, $next) {
2
3   $request->registerMediaTypeParser(
4     'text/csv',
5     function ($input) {
6       /* same csv parsing code as before */
7     }
8   );
9
10  /* call through to next middleware & return response */
11  return $next($request, $response);
12 });
```

Honour the Accept header

Honour the Accept header

Return data in the format the client expects

```
curl -X "POST" "http://localhost:8888/author" \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -d '{ "name":"Terry Pratchett" }'
```

Returning JSON

It's built-in: use `withJson()`

```
 1 $app->post(
 2   '/author',
 3   function ($request, $response, $args) {
 4     $author = new Author($request->getParsedBody());
 5     $this->authors->save($author);
 6
 7     $response = $response->withJson($author->asArray());
 8     $response = $response->withStatus(201);
 9     return $response;
10   }
11 );
```

Returning JSON

```
HTTP/1.1 201 Created
Content-type: application/json
Content-Length: 106

{
  "id":"2ff815ad-491d-4db8-a025-363516e7c27e",
  "name":"Terry Pratchett",
  "biography":null
}
```

Returning XML

```
curl -X "POST" "http://localhost:8888/author" \
  -H "Accept: application/xml"                \
  -H "Content-Type: application/json"         \
  -d '{ "name":"Terry Pratchett" }'
```

Determine media type

```
$ composer require willdurand/negotiation
```

```php
/* find preferred format from Accept header */
function determineMediaType($acceptHeader)
{
  $negotiator = new \Negotiation\Negotiator();
  $known = ['application/json', 'application/xml'];

  $mediaType = $negotiator->getBest($acceptHeader, $known);
  if ($mediaType) {
    return $mediaType->getValue();
  }
  return false;
}
```

Format output

```
 1 $acceptHeader = $request->getHeaderLine('Accept')
 2 $mediaType = determineMediaType($acceptHeader);
 3
 4 switch ($mediaType) {
 5   case 'application/xml':
 6     $response->getBody()->write(arrayToXml($data)); break;
 7
 8   case 'application/json':
 9     $response->getBody()->write(json_encode($data)); break;
10
11   default:
12     return $response->withStatus(406);
13 }
14
15 return $response->withHeader("Content-Type", $mediaType);
```

Accept: application/xml

```
HTTP/1.1 201 Created
Content-type: application/xml
Content-Length: 131

<?xml version="1.0"?>
<root>
  <id>98c22fa3-bf97-48c8-accd-025470c34b46</id>
  <name>Terry Pratchett</name>
  <biography/>
</root>
```

Error handling

Error handling

- Method not allowed
- Not found
- Generic error

Method not allowed

```
curl -X "PUT" "http://localhost:8888/ping"

HTTP/1.1 405 Method Not Allowed
Content-type: text/html;charset=UTF-8
Allow: GET

<html>
    <body>
        <h1>Method not allowed</h1>
        <p>Method not allowed. Must be one of:
           <strong>GET</strong></p>
    </body>
</html>
```

Not found

```
curl -X "GET" "http://localhost:8888/foo" \
     -H "Accept: application/xml"

HTTP/1.1 404 Not Found
Content-Type: application/xml
Allow: GET

<root><message>Not found</message></root>
```

Raise your own

```
 1 $app->get(
 2   '/author/{id}',
 3   function ($request, $response, $args) {
 4     $author = $this->authors->loadById($args['id']);
 5     if (!$author) {
 6
 7       return $this->notFoundHandler($request, $response);
 8
 9     }
10
11     // continue with $author
12   }
13 );
```

Generic error

```
1 $app->get('/error',
2   function ($request, $response, $args) {
3     throw new \Exception("Something has gone wrong!");
4   }
5 );
```

```
curl -H "Accept: application/json" "http://localhost:8888/error"

HTTP/1.1 500 Internal Server Error
Content-type: application/json
Content-Length: 43

{
    "message": "Slim Application Error"
}
```

Exception information

```
1 $settings = [
2   'settings' => [
3     'displayErrorDetails' => true,
4   ]
5 ];
6
7 $app = new Slim\App($settings);
```

Exception information

```
HTTP/1.1 500 Internal Server Error
Content-type: application/json

{
  "message": "Slim Application Error",
  "exception": [
    {
      "type": "Exception",
      "code": 0,
      "message": "Something has gone wrong!",
      "file": "/dev/an-api/app/routes.php",
      "line": 8,
      "trace": [
          "#0 [internal function]: Closure->{closure} …
          "#2 /dev/an-api/vendor/slim/slim/Slim/Route.php(…
```

Use an error handler for warnings

```php
/* convert errors into exceptions */
function exception_error_handler($level, $message, $file, $line) {
    if (!(error_reporting() & $level)) {
        return;
    }

    throw new ErrorException($message, 0, $level, $file, $line);
}

set_error_handler("exception_error_handler");
```

Versions

Versions

Two choices:

- Segment within URL: `http://api.example.com/v1/author`
- Media type: `Accept: application/vnd.rka.author.v1+json`

URL segment

Use route groups:

```
 1 $app->group('/v1', function () {
 2
 3   // http://api.example.com/v1/author
 4   $this->get('/author',
 5     function ($request, $response) { /*…*/ }
 6   );
 7
 8   // http://api.example.com/v1/author/123
 9   $this->get('/author/{id}',
10     function ($request, $response, $args) { /*…*/ }
11   );
12
13 });
```

Segue: Dependency injection in Slim

The container

Register services with the DIC

```
 1 $app = new Slim\App($settings);
 2 $container = app->getContainer();
 3
 4 $container['pdo'] = function ($c) {
 5     return new PDO($c['settings']['dsn']);
 6 };
 7
 8 $container['authors'] = function ($c) {
 9   return new Bibliotheque\AuthorMapper($c['pdo']);
10 };
```

Controller classes

Register your controller with the container

```
1 $container['AuthorController'] = function ($c) {
2   $renderer = $c->get('renderer');
3   $authors = $c->get('authors');
4   return new App\AuthorController($renderer, $authors);
5 }
```

Use when defining your route

```
1 $app->get('/author', 'AuthorController:listAll');
```

Author controller

```
 1 class AuthorController
 2 {
 3   public function __construct($renderer, $authors) {/**/}
 4
 5   public function listAll($req, $res)
 6   {
 7     $authors = $this->authors->fetchAll();
 8     $data = ['authors' => $authors];
 9     return $this->renderer->render($req, $res, $data);
10   }
11 }
```

Media type versioning

Select controller based on Accept header

```
 1 $container['AuthorController'] = function ($c) {
 2
 3   $request = $c->get('request');
 4   $acceptHeader = $request->getHeaderLine('Accept');
 5
 6   if (strpos($acceptHeader,
 7     'application/vnd.rka.author.v2') !== false) {
 8     return new App\V2\AuthorController(/*…*/);
 9   }
10
11   return new App\V1\AuthorController(/*…*/);
12 };
```

To sum up

Summary

A good API deals with:

- HTTP method negotiation
- Content-type handling
- Honour the Accept header
- Error handling
- Versions

Resources

- http://phptherightway.com

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

Questions?

https://joind.in/talk/7f408

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

Thank you!

https://joind.in/talk/7f408

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