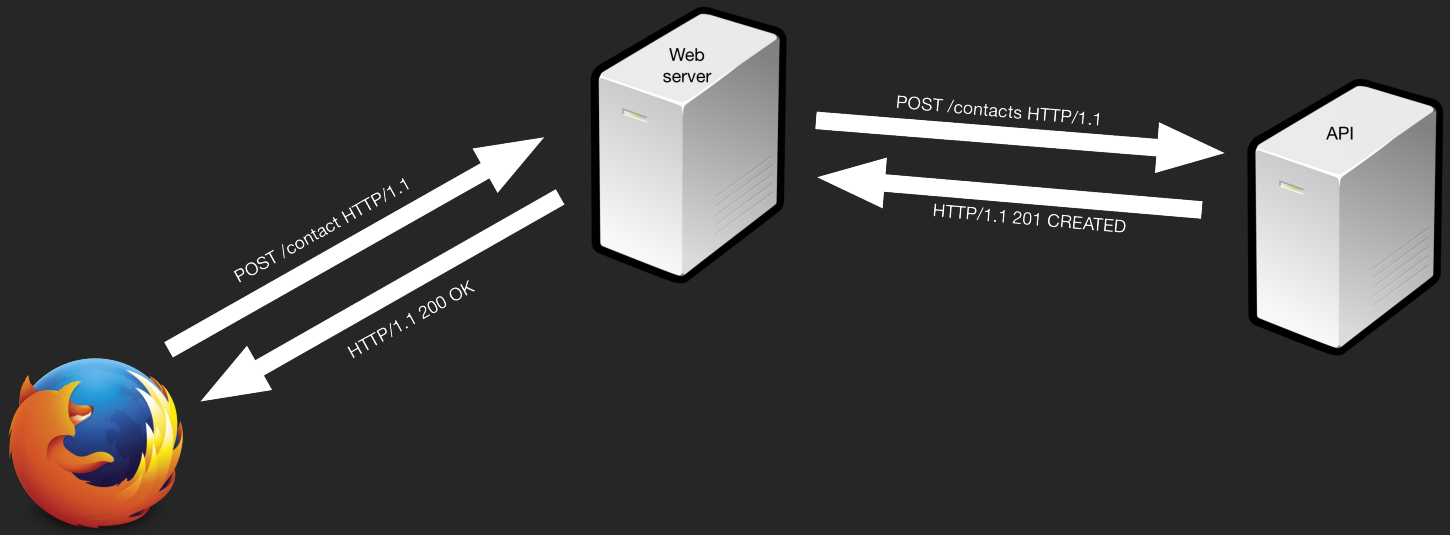
HTTP, PSR-7 and Middleware

Rob Allen ~ @akrabat ~ April 2016

HTTP Messages are the foundation

HTTP


Web
server
API
POST /contact HTTP/1.1
HTTP/1.1 200 OK
POST /contacts HTTP/1.1
HTTP/1.1 201 CREATED

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
```

Request

```
GET / HTTP/1.1
Host: akrabat.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:45.0)
    Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
If-Modified-Since: Mon, 04 Apr 2016 16:21:02 GMT
Cache-Control: max-age=0
```

Response

```
HTTP/1.1 200 OK
Server: nginx/1.7.6
Date: Mon, 04 Apr 2016 16:27:06 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 16091
Connection: keep-alive
Keep-Alive: timeout=65
Vary: Accept-Encoding, Cookie
Cache-Control: max-age=3, must-revalidate
Content-Encoding: gzip
Last-Modified: Mon, 04 Apr 2016 16:21:02 GMT
Strict-Transport-Security: max-age=15768000

<!DOCTYPE html>
<head>
```

How do we do this in PHP?

Request:

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

How do we do this in PHP?

Response:

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

How do we do this in PHP?

URI:

`scheme://username:password@hostname:port/path?arg=value#anchor`

- `parse_url()`
- `parse_str()`
- `http_build_query()`

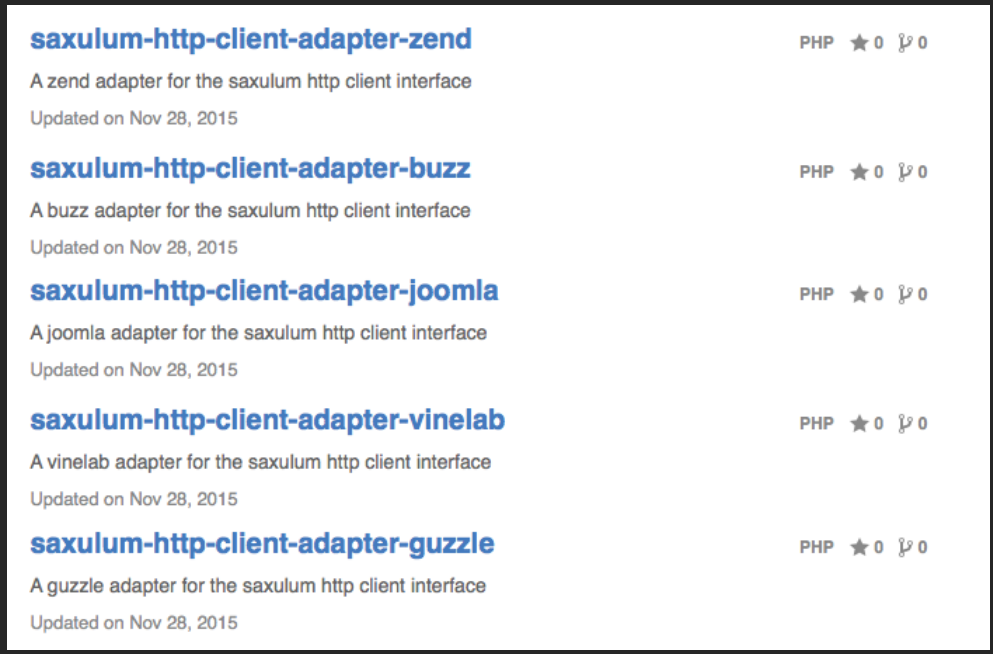
But we live in an OOP world!

OOP modelling of HTTP

- `Cake\Network\{Request,Response}`
- `CI_Input`, `CI_Output`
- `Nette\Http\{Request,Response}`
- `PHPixie\HTTP\{Request,Responses}`
- `Symfony\Component\HttpFoundation\{Request,Response}`
- `yii\web\{Request,Response}`
- `Zend\Http\{Request,Response}`

None of these work together!

Adapt all the things!



PSR-7

An HTTP Message interface
by the Framework Interoperability Group

It took a while!

- HTTP client mooted by Benjamin Eberlei on 24 March 2012

It took a while!

- HTTP client mooted by Benjamin Eberlei on 24 March 2012
- Chris Wilkinson suggested HTTP messages on 31 December 2012

It took a while!

- HTTP client mooted by Benjamin Eberlei on 24 March 2012
- Chris Wilkinson suggested HTTP messages on 31 December 2012
- Michael Dowling proposed PSR-7 draft on 28 January 2014

It took a while!

- HTTP client mooted by Benjamin Eberlei on 24 March 2012
- Chris Wilkinson suggested HTTP messages on 31 December 2012
- Michael Dowling proposed PSR-7 draft on 28 January 2014
- Michael gave up on 29th August 2014

It took a while!

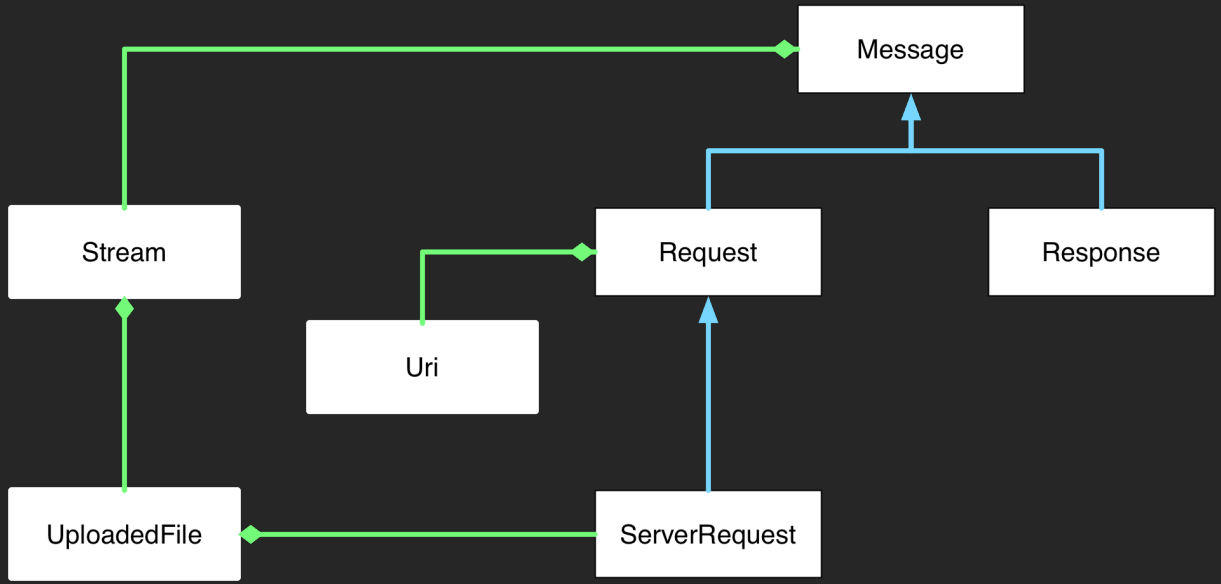
- HTTP client mooted by Benjamin Eberlei on 24 March 2012
- Chris Wilkinson suggested HTTP messages on 31 December 2012
- Michael Dowling proposed PSR-7 draft on 28 January 2014
- Michael gave up on 29th August 2014
- Matthew Weier O'Phinney re-proposed on 26 September 2014

It took a while!

- HTTP client mooted by Benjamin Eberlei on 24 March 2012
- Chris Wilkinson suggested HTTP messages on 31 December 2012
- Michael Dowling proposed PSR-7 draft on 28 January 2014
- Michael gave up on 29th August 2014
- Matthew Weier O'Phinney re-proposed on 26 September 2014
- PSR-7 accepted on 18 May 2015

So what is it?

It's just a set of interfaces



RequestInterface

- Protocol version
- HTTP method
- URI
- Headers
- Body

ServerRequestInterface

In addition to `RequestInterface`:

- Server parameters
- Query string arguments
- Deserialised body
- Uploaded Files
- Attributes

Response

- Protocol version
- Status code (& reason phrase)
- Headers
- Body

Two key things about PSR-7

Firstly: Messages and URIs are IMMUTABLE!

Immutability

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

```
$uri = new Uri('https://api.joind.in/v2.1/events');
$uri2 = $uri->withQuery('?filter=upcoming');
$uri3 = $uri->withQuery('?filter=cfp');
```

Immutability

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

```
$uri = new Uri('https://api.joind.in/v2.1/events');
$uri2 = $uri->withQuery('?filter=upcoming');
$uri3 = $uri->withQuery('?filter=cfp');
```

```
  public function withQuery($query)
  {
      $clone = clone $this;
      $clone->query = $this->filterQuery($query);
      return $clone;
  }
```

Common mistake

```
$response = new Response();
$response->withStatus(302);
$response->withHeader('Location', 'http://example.com');

return $response;
```

What you meant!

```php
$response = new Response();
$response = $response->withStatus(302);
$response = $response->withHeader('Location', 'http://example.com');

// or

$response = new Response();
$response = $response->withStatus(302);
            ->withHeader('Location', 'http://example.com');
```

Headers

Reading from the message:

```
$request->hasHeader('Accept');      // boolean
$request->getHeader('Accept');      // array
$request->getHeaderLine('Accept');  // string
```

Updating the message:

```
$response = $response->withHeader('Accept', 'application/xml');
$response = $response->withAddedHeader('Accept', 'text/xml');
$response = $response->withoutHeader('Location');
```

Secondly: Message bodies are STREAMS

Streams

Allows for very large message bodies in a memory efficient manner

```
$largeFile = __DIR__ . '/brand_guidelines.ppt';

return (new Response())
    ->withHeader('Content-Type', 'application/vnd.ms-powerpoint')
    ->withHeader('Content-Length', (string) filesize($largeFile))
    ->withBody(new Stream($largeFile));
```

Handling strings

For strings, use the `php://temp` stream:

```
$response = $response->withBody(new Stream('php://temp'));
$response->getBody()->write('Hello world!');
```

Note: Mutable

```
$body->write('Hello');
$body->write(' World!');

$response = (new Response())
  ->withStatus(200, 'OK')
  ->withHeader('Content-Type', 'application/html')
  ->withBody($body);
```

Mitigate:

Use read-only streams for server requests and client responses

Attributes

Allow passing of data from one component to the next

```
// component 1
function findCountry ($request) {
    $country = $this->geoLocate($request);
    $request = $request->withAttribute('country', $country);
    return $request;
}

// In an action method
function listAction($request, $response, $args) {
    $country = $request->getAttribute('country');
    // do something now that we know the country
}
```

Implementations

Implementations

- `Zend\Diactoros`
- `GuzzleHttp\Psr7`
- `Wandu\Http`
- `Asika\Http`
- `Slim\Http`

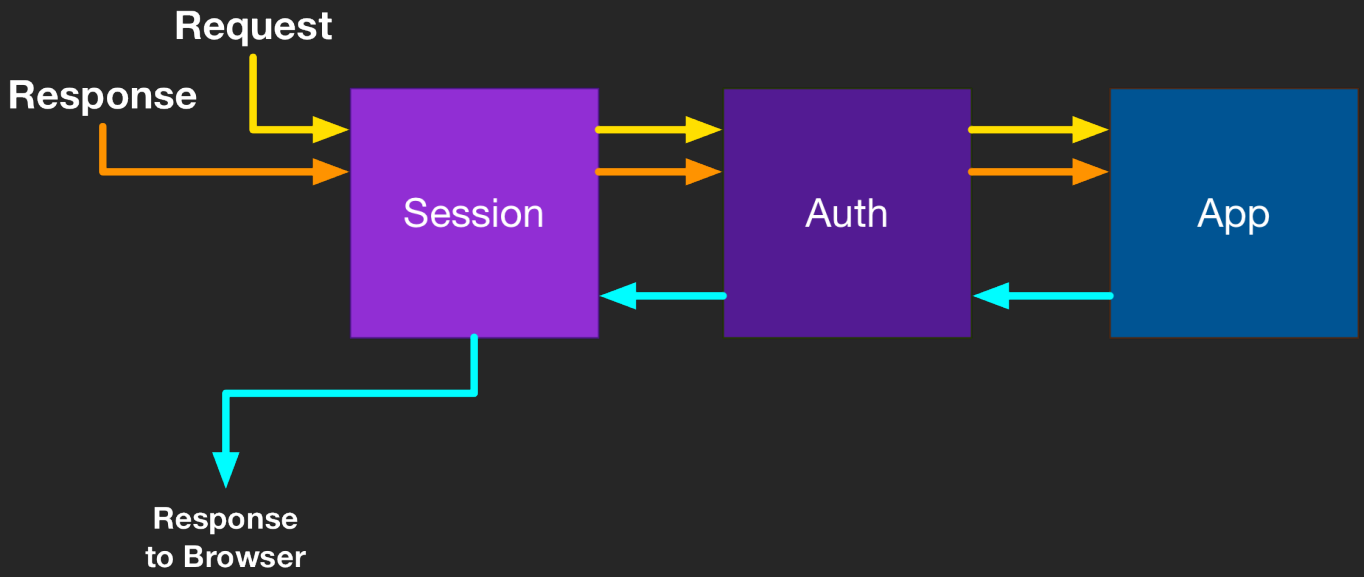
Bridges

- symfony-psr-http-message-bridge
- zendframework/zend-psr7bridge
- icicleio/psr7-bridge
- h4cc/stack-psr7-bridge
- CakePHP (coming in 3.3)

Middleware

Turns a request in to a response

Middleware



This is not new

- NodeJS: Connect
- Ruby: Rack
- Python: WSGI
- PHP: StackPHP, Slim

Signature

```
function (
    ServerRequestInterface $request,
    ResponseInterface $response,
    callable $next
) : ResponseInterface;
```

Middleware

```
function (ServerRequestInterface $request, ResponseInterface $response,
    callable $next) : ResponseInterface
{
    // do something before

    // call down the chain to the next middleware
    if ($next) {
        $response = $next($request, $response);
    }

    // do something with $response after

    return $response;
}
```

What can you do with Middleware?

Session
Logging
Authentication
HTTP cache
CSP

CORS
Firewall
Throttling
Honeypot
etc...

CSRF
Debug bar
Error handling
Validation

Middleware dispatching

```
$queue[] = $mw1;
$queue[] = [$instance, 'methodName'];
$queue[] = function ($request, $response, $next) {
    return $next($request, $response);
} );

$relay = (new RelayBuilder())->newInstance($queue);
$response = $relay($request, $response);
```

Middleware dispatching

```
$app->pipe($mw1);               // always evaluate
$app->pipe('/path', $mw2);      // only if path matches
$app->pipe('/different', $mw3);
$app->pipe($mw4);

$response = $app->run();
```

Data transfer between middleware

https://github.com/akrabat/rka-ip-address-middleware

```
class IpAddress
{
    public function __invoke($request, $response, $next)
    {
        if (!$next) {
            return $response;
        }
        $ipAddress = $this->determineClientIpAddress($request);
        $request = $request->withAttribute('ip_address', $ipAddress);
        return $response = $next($request, $response);
    }

    // ...
}
```

Data transfer between middleware

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

    if (!in_array($ipAddress, $this->allowedIpAddresses)) {
        return $response->withStatus(401);
    }

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

Interoperability with Guzzle

Both Slim 3 & Guzzle 6 implement PSR-7…

```
$app->get('/random', function ($request, $response, $args) {

    $choice = mt_rand(1, 15);
    $filename ='image_' . $choice . '.jpg';

    $guzzle = new \GuzzleHttp\Client();
    $apiResponse = $guzzle->get("https://i.19ft.com/$filename");

    return $apiResponse;
}
```

PSR-7 Middleware implementations

Middleware dispatchers

- Zend-Stratigility
- Relay

PSR-7 Middleware implementations

Middleware dispatchers

- Zend-Stratigility
- Relay

Frameworks

Slim Framework (v3)
Radar
Harmony
Spiral

Zend Expressive
Nimble
Gobline
Turbine

Expressive

```
$app = Zend\Expressive\AppFactory::create();

// add middleware
$app->pipe($middleware1);

// add routes
$app->get('/', function ($request, $response, $next) {
    $response->getBody()->write('Hello, world!');
    return $response;
});

// run
$app->pipeRoutingMiddleware();
$app->pipeDispatchMiddleware();
$app->run();
```

Slim Framework

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

// add middleware
$app->add($middleware1);

// add routes
$route = $app->get('/list', function($request, $response, $args) {
    $response->getBody()->write('Hello, world!');
    return $response;
});
$route->add($routeMiddleware);

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

Radar

```
$adr = (new Radar\Adr\Boot())->adr();

// add middleware
$adr->middle(new ResponseSender());
$adr->middle(new ExceptionHandler(new Response()));

// add routes
$adr->get('Hello', '/', function (array $input) {
    return (new Aura\Payload\Payload())
        ->setStatus(Aura\Payload_Interface\PayloadStatus::SUCCESS)
        ->setOutput(['phrase' => 'Hello world!']);
});

// run
$adr->run(Zend\Diactoros\ServerRequestFactory::fromGlobals(),
    new Zend\Diactoros\Response());
```

PSR-7 Middleware components

Too many to count!

- https://packagist.org/search/?q=psr-7%20middleware
- https://github.com/oscarotero/psr7-middlewares
- https://github.com/lalop/awesome-psr7#psr-7-middlewares

The future is interoperable!

Thank you!

https://joind.in/talk/88b94

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