HTTP & Middleware

Rob Allen

November 2018

Slim: The C in MVC

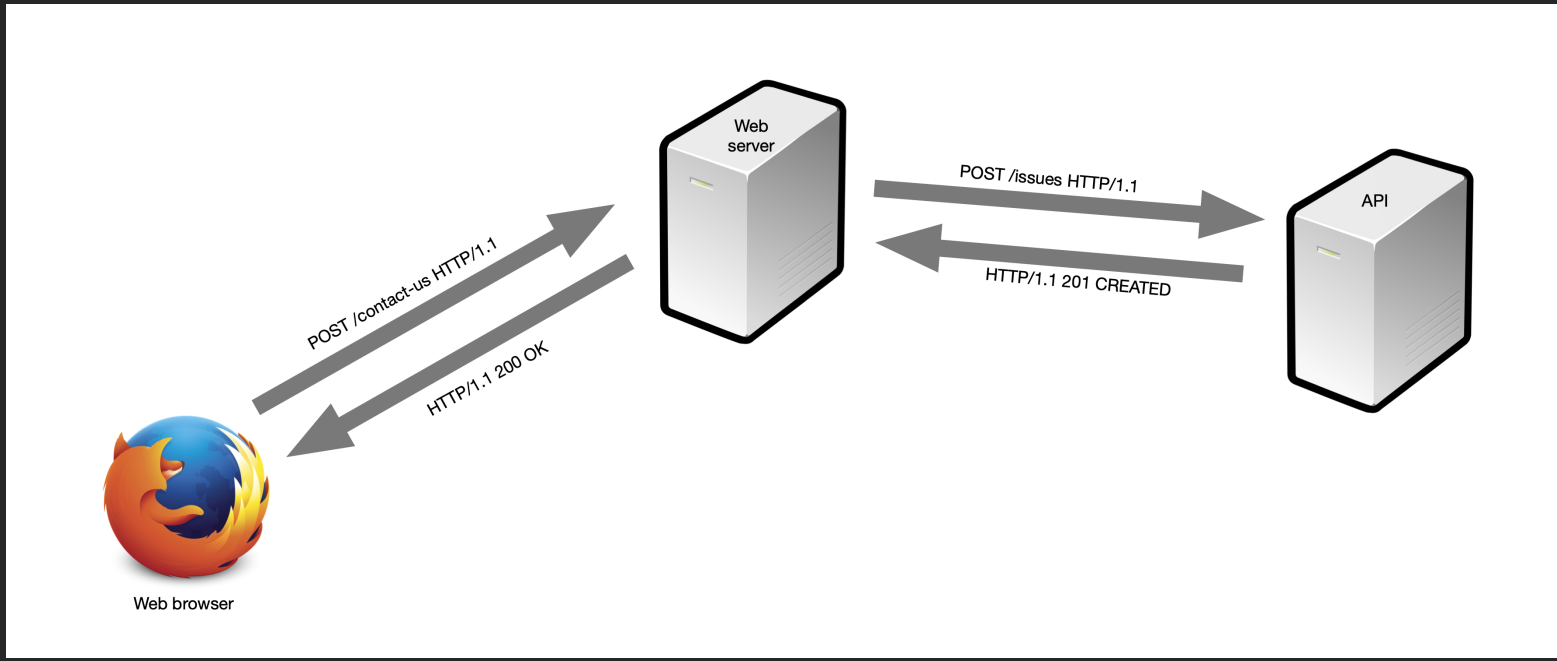
Slim

Created by Josh Lockhart (phptherightway.com)

- PSR-2 coding style
- PSR-4 autoloading
- PSR-7 Request and Response objects
- PSR-11 container support
- Middleware architecture (PSR-15 upcoming)

HTTP messages

HTTP messages



Request

```
GET / HTTP/1.1
Host: akrabat.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:63.0)
    Gecko/20100101 Firefox/63.0
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 Nov 2018 16:21:02 GMT
Cache-Control: max-age=0
```

Response

```
HTTP/1.1 200 OK
Server: nginx/1.7.6
Date: Mon, 04 Nov 2018 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()` / `getallheaders()`
- `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:

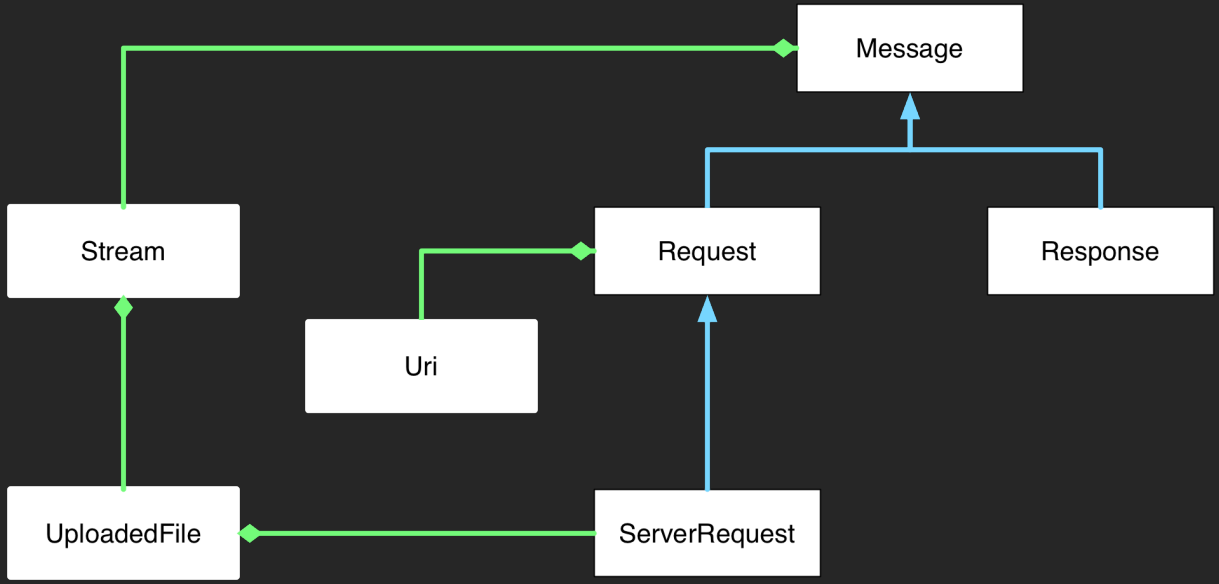
```
scheme://username:password@hostname:port/path?arg=value#anchor
```

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

PSR 7: HTTP messaging

- Provides for a uniform access to HTTP messages
- A set of interfaces for requests and responses
- Created by Framework Interoperability Group

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!');
```

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

ServerRequest Attributes

Allow passing of data from one component to the next

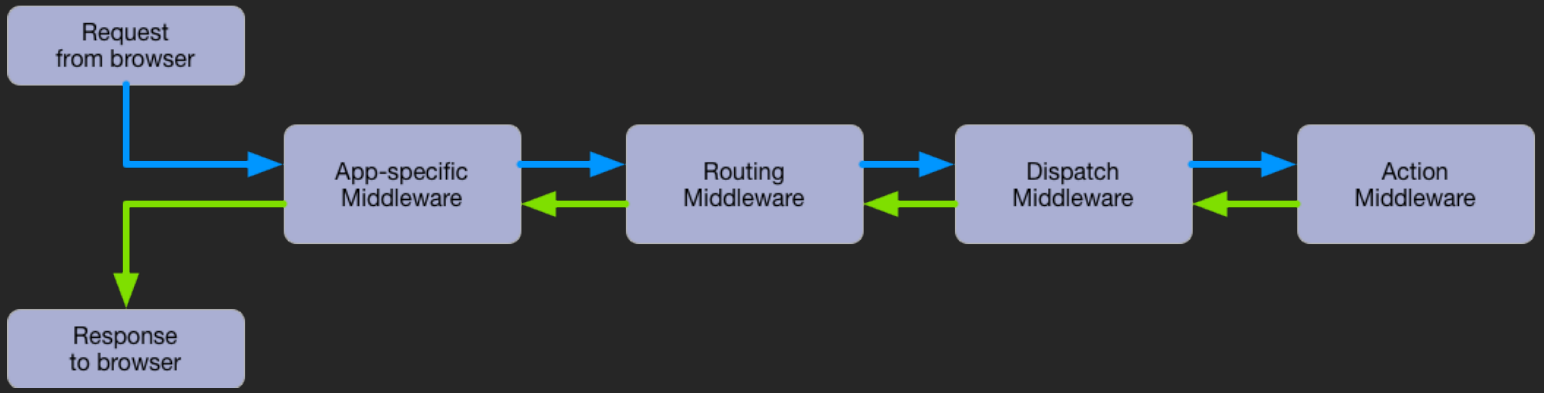
```
// component 1
function findCountry($request) {
    $country = $this->geoLocate($request);
    $request = $request->withAttribute('country', $country);
    return $request;
}

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

Middleware

Turns a request in to a response

Middleware


Request
from browser
App-specific
Middleware
Routing
Middleware
Dispatch
Middleware
Action
Middleware
Response
to browser

This is not new

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

PSR-7 Signature (by convention)

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

PSR-7 Signature (by convention)

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

PSR-7 Signature (by convention)

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

PSR-7 Signature (by convention)

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

PSR-7 Signature (by convention)

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

PSR-7 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;
}
```

PSR-7 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;
}
```

PSR-7 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;
}
```

PSR-7 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;
}
```

PSR-7 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

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

$app->add($mw1);
$app->add(SomeMiddleware::class);
$app->add('MoreMiddleware');
$app->add([$instance, 'methodName']);
$app->add(function ($request, $response, $next) {
    return $next($request, $response);
});

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

Data transfer between middleware

Attributes allow passing of data from one middleware to the next

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

    // full source: github.com/akrabat/ip-address-middleware
}
```

Data transfer between middleware

Attributes allow passing of data from one middleware to the next

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

    // full source: github.com/akrabat/ip-address-middleware
}
```

Data transfer between middleware

Attributes allow passing of data from one middleware to the next

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

    // full source: github.com/akrabat/ip-address-middleware
}
```

Data transfer between middleware

Attributes allow passing of data from one middleware to the next

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

    // full source: github.com/akrabat/ip-address-middleware
}
```

Data transfer between middleware

Attributes allow passing of data from one middleware to the next

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

    // full source: github.com/akrabat/ip-address-middleware
}
```

Data transfer between middleware

Read the attribute

```
class GuardMiddleware
{
    protected $allowedIps = [10.0.0.1, 192.168.0.1]

    public function __invoke($request, $response, $next)
    {
        $ipAddress = $request->getAttribute('ip_address');
        if (!in_array($ipAddress, $this->allowedIps)) {
            return $response->withStatus(401);
        }
        return $next($request, $response);
    }
}
```

Data transfer between middleware

Read the attribute

```
class GuardMiddleware
{
    protected $allowedIps = [10.0.0.1, 192.168.0.1]

    public function __invoke($request, $response, $next)
    {
        $ipAddress = $request->getAttribute('ip_address');
        if (!in_array($ipAddress, $this->allowedIps)) {
            return $response->withStatus(401);
        }
        return $next($request, $response);
    }
}
```

Data transfer between middleware

Read the attribute

```
class GuardMiddleware
{
    protected $allowedIps = [10.0.0.1, 192.168.0.1]

    public function __invoke($request, $response, $next)
    {
        $ipAddress = $request->getAttribute('ip_address');
        if (!in_array($ipAddress, $this->allowedIps)) {
            return $response->withStatus(401);
        }
        return $next($request, $response);
    }
}
```

Data transfer between middleware

Read the attribute

```
class GuardMiddleware
{
    protected $allowedIps = [10.0.0.1, 192.168.0.1]

    public function __invoke($request, $response, $next)
    {
        $ipAddress = $request->getAttribute('ip_address');
        if (!in_array($ipAddress, $this->allowedIps)) {
            return $response->withStatus(401);
        }
        return $next($request, $response);
    }
}
```

Data transfer between middleware

Read the attribute

```
class GuardMiddleware
{
    protected $allowedIps = [10.0.0.1, 192.168.0.1]

    public function __invoke($request, $response, $next)
    {
        $ipAddress = $request->getAttribute('ip_address');
        if (!in_array($ipAddress, $this->allowedIps)) {
            return $response->withStatus(401);
        }
        return $next($request, $response);
    }
}
```

PSR-15

Two Interfaces from the Framework Interoperability Group:

- `Psr\Http\Server\MiddlewareInterface`
- `Psr\Http\Server\RequestHandlerInterface`

PSR-15 MiddlewareInterface

```php
namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

interface MiddlewareInterface
{
    public function process(
        ServerRequestInterface $request,
        RequestHandlerInterface $handler
    ) : ResponseInterface;
}
```

PSR-15 MiddlewareInterface

```php
namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

interface MiddlewareInterface
{
    public function process(
        ServerRequestInterface $request,
        RequestHandlerInterface $handler
    ) : ResponseInterface;
}
```

PSR-15 MiddlewareInterface

```php
namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

interface MiddlewareInterface
{
    public function process(
        ServerRequestInterface $request,
        RequestHandlerInterface $handler
    ) : ResponseInterface;
}
```

PSR-15 MiddlewareInterface

```php
namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

interface MiddlewareInterface
{
    public function process(
        ServerRequestInterface $request,
        RequestHandlerInterface $handler
    ) : ResponseInterface;
}
```

PSR-15 MiddlewareInterface

```php
namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

interface MiddlewareInterface
{
    public function process(
        ServerRequestInterface $request,
        RequestHandlerInterface $handler
    ) : ResponseInterface;
}
```

Structure of PSR-15 middleware

```php
class TimerMiddleware implements MiddlewareInterface
{
    public function process($request, $handler)
    {
        $start = microtime(true);

        $response = $handler->handle($request);

        $taken = microtime(true) - $start;
        $response->getBody()->write("<!-- Time: $taken -->");

        return $response;
    }
}
```

Structure of PSR-15 middleware

```php
class TimerMiddleware implements MiddlewareInterface
{
    public function process($request, $handler)
    {
        $start = microtime(true);

        $response = $handler->handle($request);

        $taken = microtime(true) - $start;
        $response->getBody()->write("<!-- Time: $taken -->");

        return $response;
    }
}
```

Structure of PSR-15 middleware

```
class TimerMiddleware implements MiddlewareInterface
{
    public function process($request, $handler)
    {
        $start = microtime(true);

        $response = $handler->handle($request);

        $taken = microtime(true) - $start;
        $response->getBody()->write("<!-- Time: $taken -->");

        return $response;
    }
}
```

Structure of PSR-15 middleware

```
class TimerMiddleware implements MiddlewareInterface
{
    public function process($request, $handler)
    {
        $start = microtime(true);

        $response = $handler->handle($request);

        $taken = microtime(true) - $start;
        $response->getBody()->write("<!-- Time: $taken -->");

        return $response;
    }
}
```

Structure of PSR-15 middleware

```
class TimerMiddleware implements MiddlewareInterface
{
    public function process($request, $handler)
    {
        $start = microtime(true);

        $response = $handler->handle($request);

        $taken = microtime(true) - $start;
        $response->getBody()->write("<!-- Time: $taken -->");

        return $response;
    }
}
```

Structure of PSR-15 middleware

```
class TimerMiddleware implements MiddlewareInterface
{
    public function process($request, $handler)
    {
        $start = microtime(true);

        $response = $handler->handle($request);

        $taken = microtime(true) - $start;
        $response->getBody()->write("<!-- Time: $taken -->");

        return $response;
    }
}
```

PSR-15 RequestHandler

```php
namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

interface RequestHandlerInterface
{
    public function handle(
      ServerRequestInterface $request
      ): ResponseInterface;
}
```

PSR-15 RequestHandler

```php
namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

interface RequestHandlerInterface
{
    public function handle(
      ServerRequestInterface $request
      ): ResponseInterface;
}
```

PSR-15 RequestHandler

```php
namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

interface RequestHandlerInterface
{
    public function handle(
      ServerRequestInterface $request
      ): ResponseInterface;
}
```

A typical PSR-15 RequestHandler

```php
namespace App\Handler;

// (use statements here)

class HomePageHandler implements RequestHandlerInterface
{
    public function handle(ServerRequestInterface $request)
        : ResponseInterface
    {
        return new HtmlResponse(
            '<p>Hello World</p>'
        );
    }
}
```

A typical PSR-15 RequestHandler

```
namespace App\Handler;

// (use statements here)

class HomePageHandler implements RequestHandlerInterface
{
    public function handle(ServerRequestInterface $request)
        : ResponseInterface
    {
        return new HtmlResponse(
            '<p>Hello World</p>'
        );
    }
}
```

A typical PSR-15 RequestHandler

```php
namespace App\Handler;

// (use statements here)

class HomePageHandler implements RequestHandlerInterface
{
    public function handle(ServerRequestInterface $request)
        : ResponseInterface
    {
        return new HtmlResponse(
            '<p>Hello World</p>'
        );
    }
}
```

Frameworks

Zend Expressive: PSR-15

```php
$app = new Zend\Expressive\Application(/* ... */);

// add middleware
$app->pipe(ErrorHandler::class);
$app->pipe(RouteMiddleware::class);
$app->pipe(IpAddress::class);
$app->pipe(DispatchMiddleware::class);

// add routes
$app->get('/', App\Handler\HomePageHandler::class);
$app->get('/api/ping', App\Handler\PingHandler::class);

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

Slim Framework: PSR-7

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

// add middleware
$app->add(IpAddress::class);

// add routes
$app->get('/', App\Action\HomePageAction::class);
$app->get('/api/ping', 'App\Controller\ApiController:pingAction');

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

Middleware components

Search on Packagist

- https://packagist.org/search/?q=psr-7%20middleware
- https://packagist.org/search/?query=psr-15%20middleware

Ones I use:

- *akrabat:* ip-address-middleware & proxy-detection-middleware
- *tuupola:* cors-middleware, slim-jwt-auth & slim-basic-auth
- *franzl:* whoops-middleware
- *oscarotero:* psr7-middlewares & psr15-middlewares

Resources

- https://www.php-fig.org/psr/psr-7
- https://www.php-fig.org/psr/psr-15
- https://www.php-fig.org/psr/psr-17

- https://mwop.net/blog/2015-01-26-psr-7-by-example.html
- https://mwop.net/blog/2018-01-23-psr-15.html

Thank you!

Rob Allen

http://akrabat.com - @akrabat