SlimPHP

New PHP Framework? Again? *Not so New*

But first, who am I?

Justinas Rakašis

- TV Star
- Tech Lead of Tesonet
- Contributor of SlimPHP
- Business oriented geek
- PetrolHead
- Just random guy

Why I’m talking about SlimPHP?

Andrew Smith liked

Rob Allen @akrabat · 19 May 2016

Thanks **@hope_for_dope** for working on @slimphp's unit tests! It's not glamorous work, but very much appreciated.

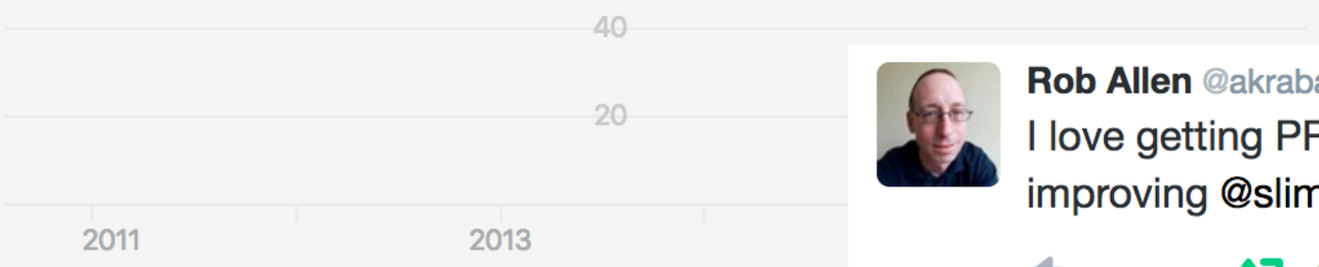
1 5

dopesong commented on 27 Nov 2016 Contributor

No description provided.

dopesong #10

29 commits / 25,841 ++ / 25,130 --



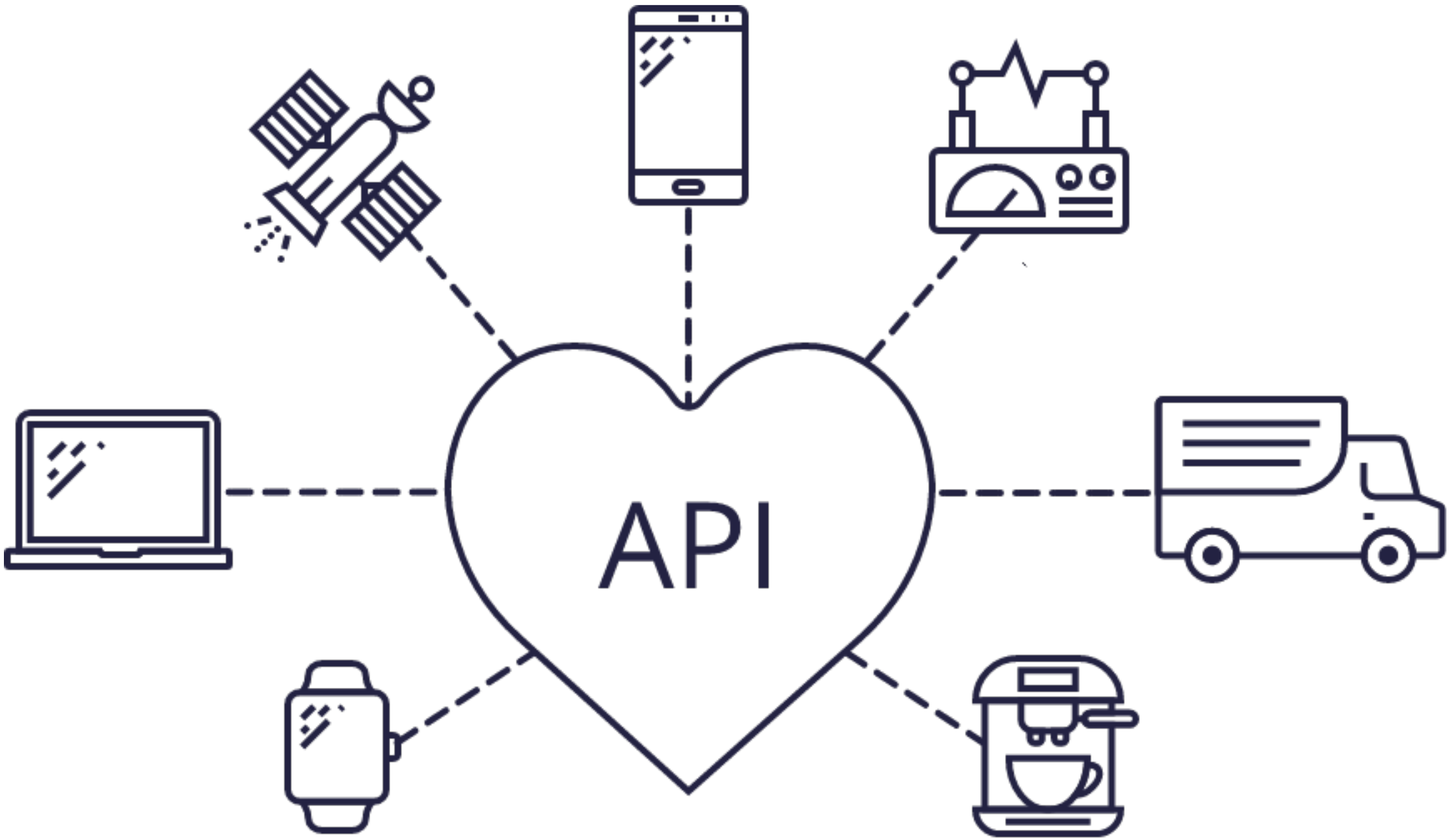
Rob Allen @akrabat · 5 Jul 2016

I love getting PRs that add unit tests. Thanks @hope_for_dope for improving @slimphp's test coverage!

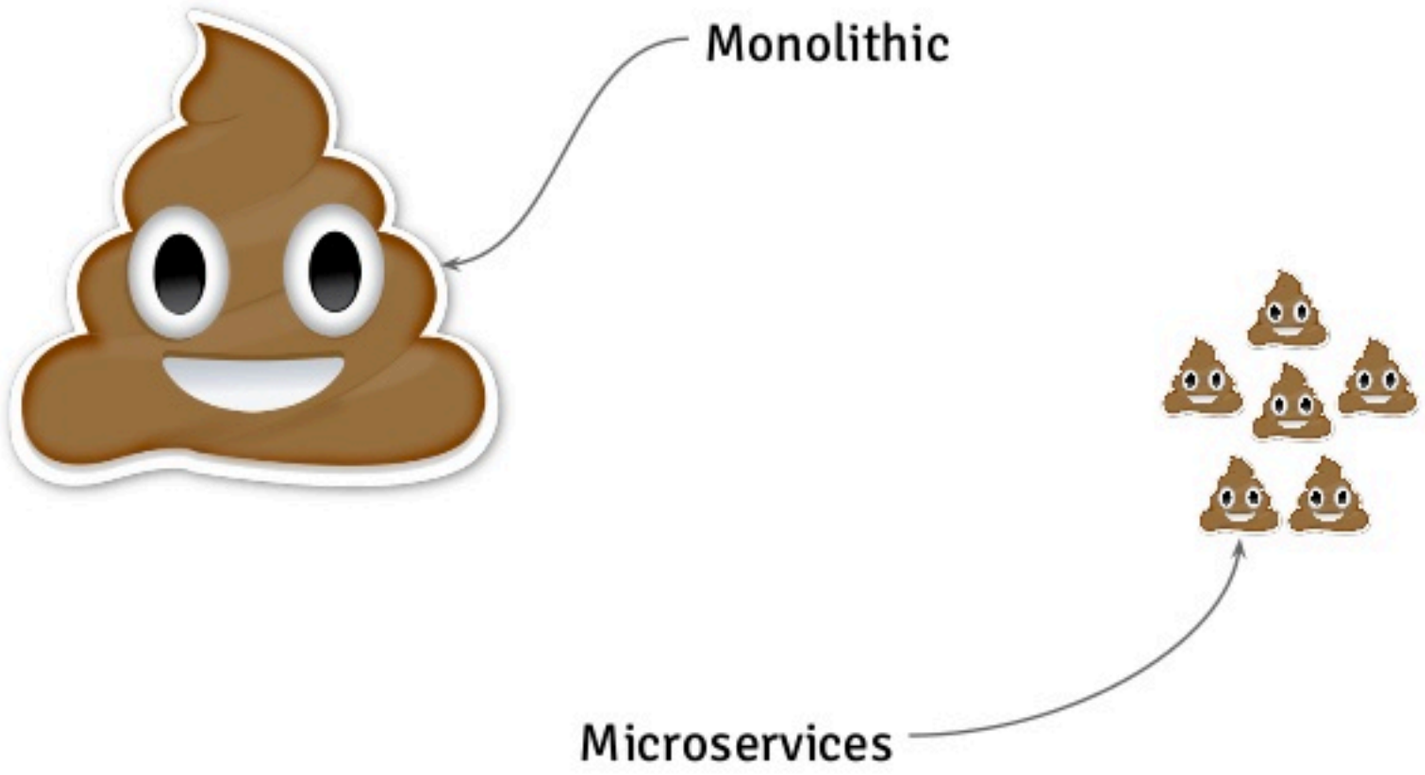
2 14

Where I met Slim?

It’s all about love…


API

Monolithic vs Microservices

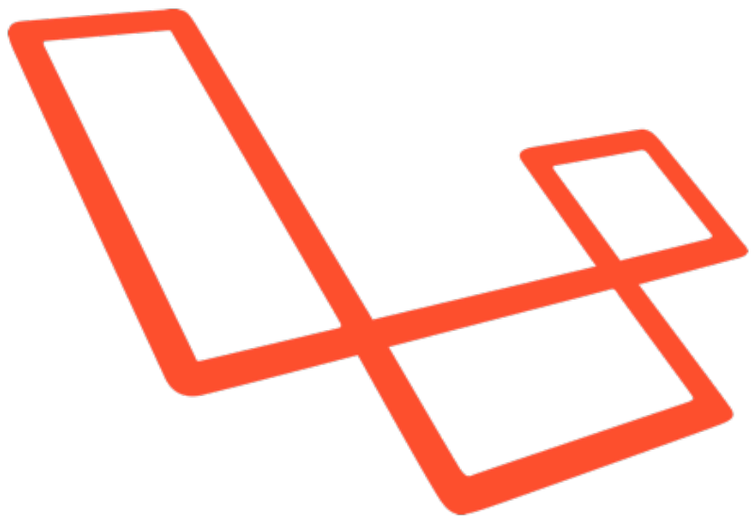


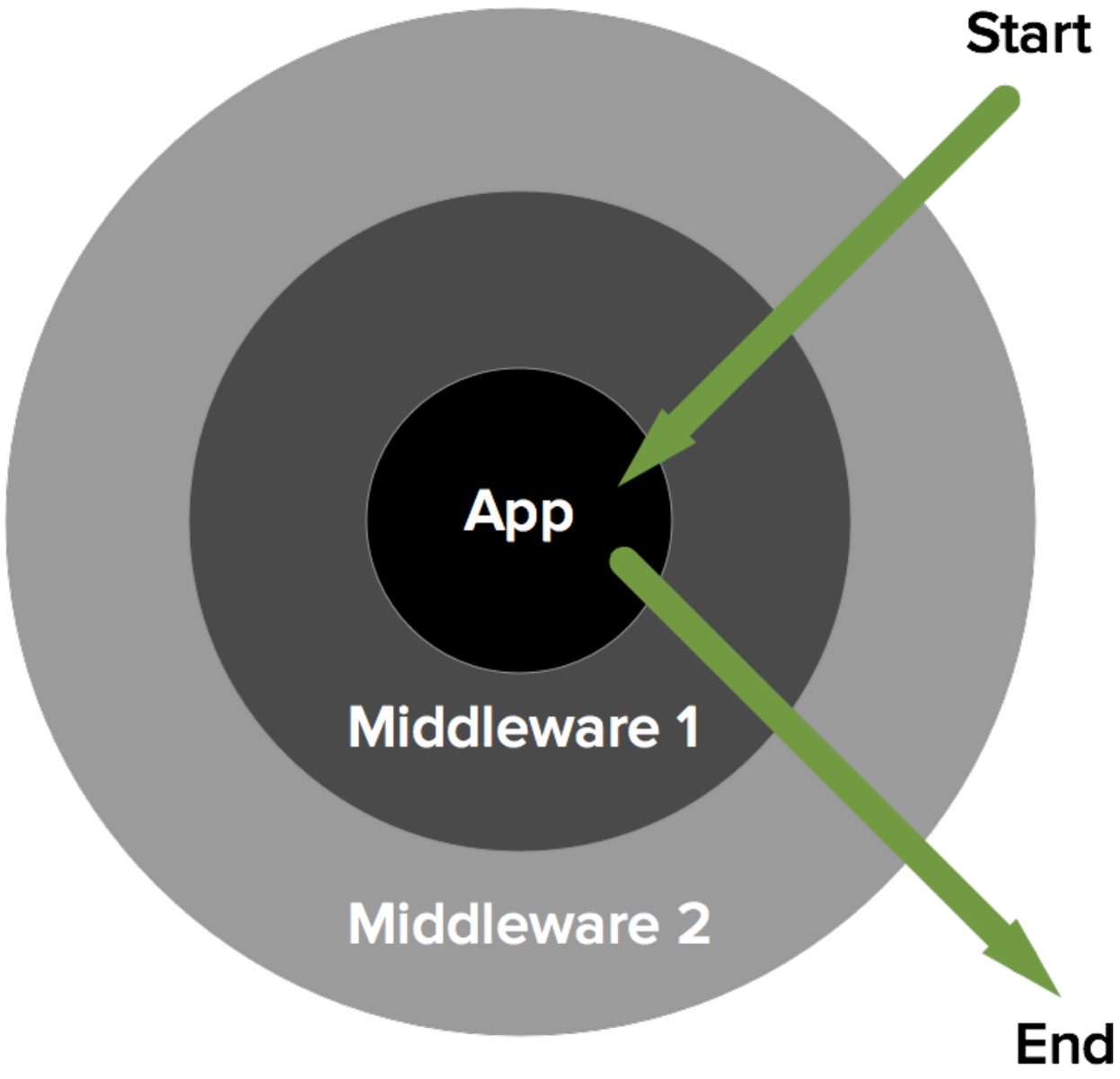
@alvaro_sanchez

odobo

?


CakePHP

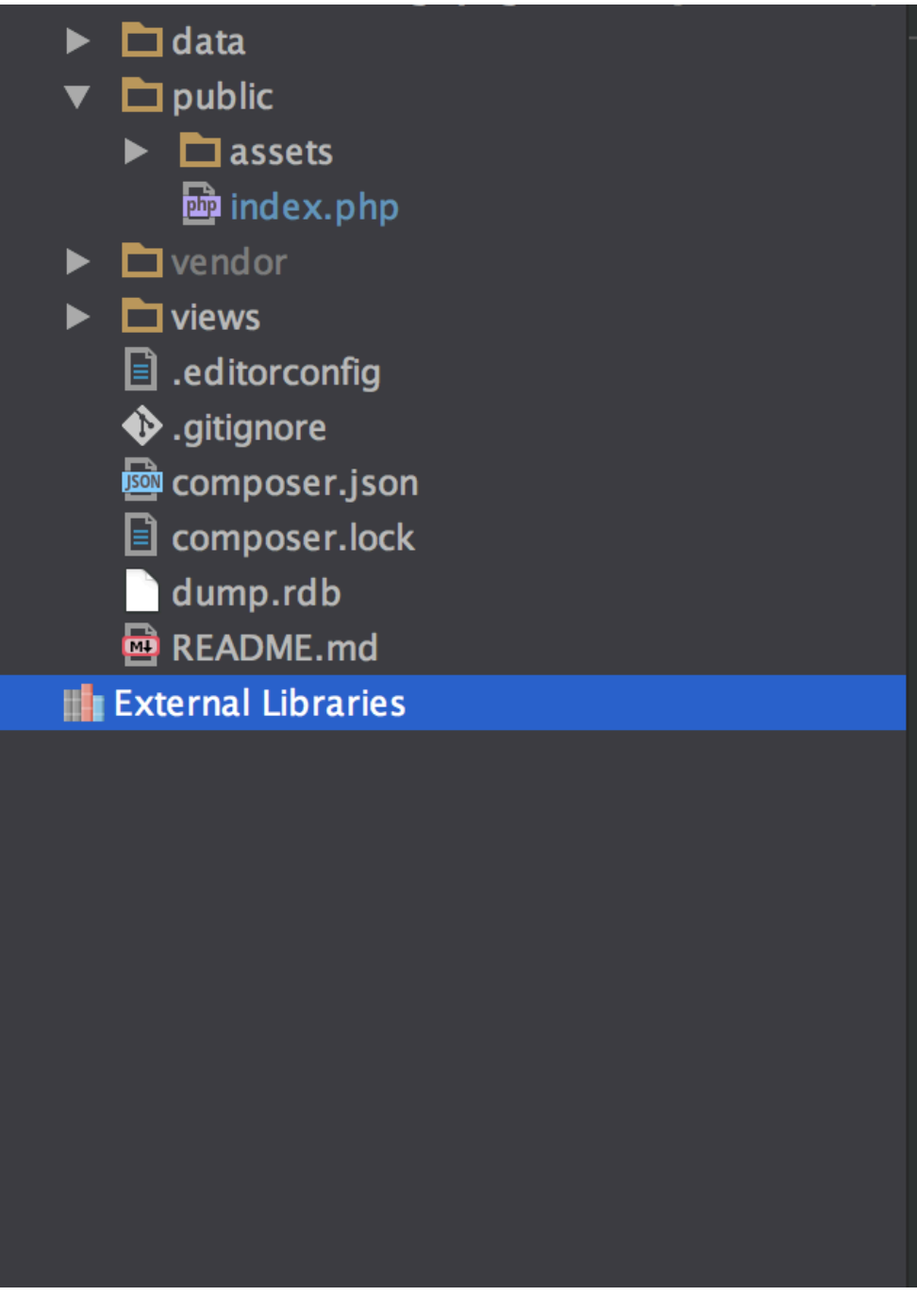



Start
App
Middleware 1
Middleware 2
End

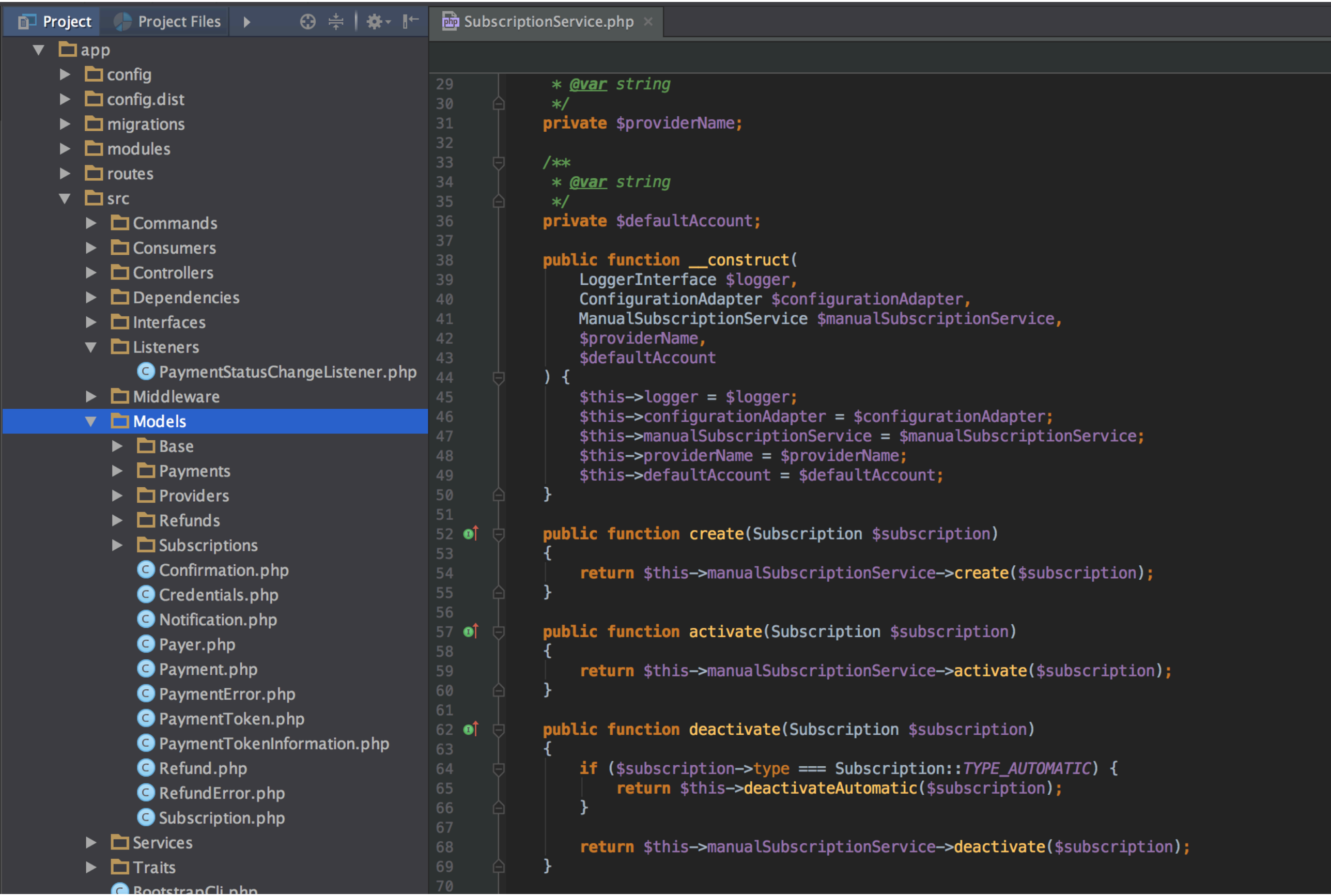
What is Slim?

Slim is a PHP micro framework that helps you quickly write simple yet powerful web applications and APIs.

- Dispatcher
- Move, manipulate, expose data
- Application development
- Rapid prototype development




```php
<?php
require_once "../vendor/autoload.php";

use Slim\App;
use Slim\Views\PhpRenderer;
use GuzzleHttp\Client as GuzzleClient;
use Predis\Client as RedisClient;
use GeoIp2\Database\Reader as GeoIpReader;
use Slim\Http\Request;
use Slim\Http\Response;

// Create and configure Slim app
$config = ['settings' => [
    'addContentLengthHeader' => false,
    'displayErrorDetails' => true,
]];

$app = new App($config);

$app->add(new RKA\Middleware\IpAddress());

$container = $app->getContainer();
$container['renderer'] = new PhpRenderer("../views");

// Define app routes
$app->get('/', function (Request $request, Response $response) {
    return $this->renderer->render($response, "/index.php");
});
```


Project
Project Files
app
config
config.dist
migrations
modules
routes
src
Commands
Consumers
Controllers
Dependencies
Interfaces
Listeners
PaymentStatusChangeListener.php
Middleware
Models
Base
Payments
Providers
Refunds
Subscriptions
Confirmation.php
Credentials.php
Notification.php
Payer.php
Payment.php
PaymentError.php
PaymentToken.php
PaymentTokenInformation.php
Refund.php
RefundError.php
Subscription.php
Services
Traits
SubscriptionService.php
29 * @var string
30 */
31 private $providerName;
32
33 /**
34 * @var string
35 */
36 private $defaultAccount;
37
38 public function __construct(
39 LoggerInterface $logger,
40 ConfigurationAdapter $configurationAdapter,
41 ManualSubscriptionService $manualSubscriptionService,
42 $providerName,
43 $defaultAccount
44) {
45 $this->logger = $logger;
46 $this->configurationAdapter = $configurationAdapter;
47 $this->manualSubscriptionService = $manualSubscriptionService;
48 $this->providerName = $providerName;
49 $this->defaultAccount = $defaultAccount;
50 }
51
52 public function create(Subscription $subscription)
53 {
54 return $this->manualSubscriptionService->create($subscription);
55 }
56
57 public function activate(Subscription $subscription)
58 {
59 return $this->manualSubscriptionService->activate($subscription);
60 }
61
62 public function deactivate(Subscription $subscription)
63 {
64 if ($subscription->type === Subscription::TYPE_AUTOMATIC) {
65 return $this->deactivateAutomatic($subscription);
66 }
67
68 return $this->manualSubscriptionService->deactivate($subscription);
69 }
70

The Bus Factor, Solved

- Slim is a GitHub organization with multiple owners who care about Slim's future.
- No single person controls the code.
- Code is open source with MIT Public License.

Josh Lockhart

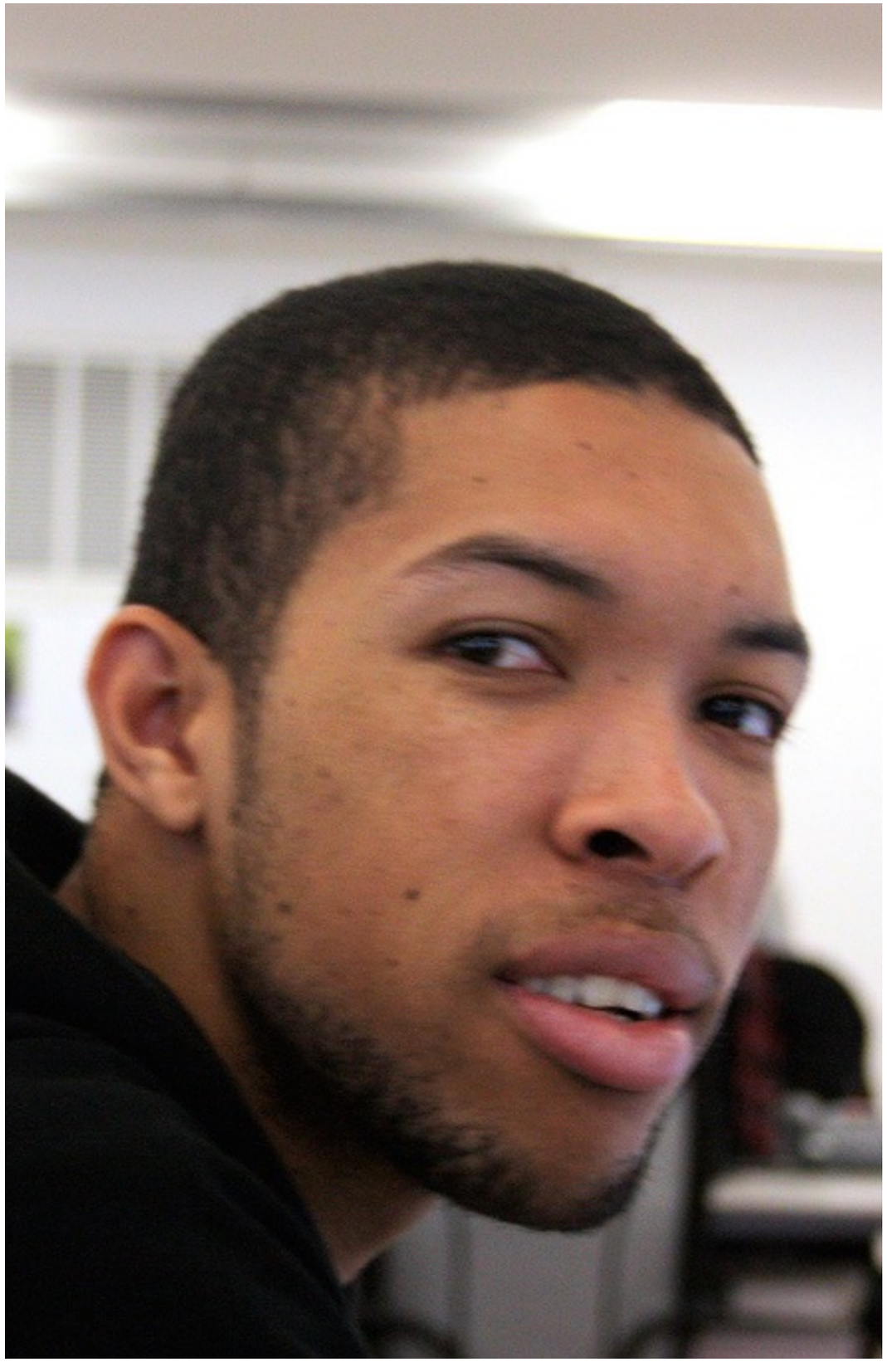
@codeguy

- Author of *Modern PHP* (O'Reilly)
- PHP The Right Way
- Slim Framework
- PHP-FIG
- New Media Campaigns

Andrew Smith

@silentworks

- http://silentworks.co.uk/
- Web developer
- Jamaica (Caribbean) born/raised
- “Working in open source have grown me as a developer"



Rob Allen

@akrabat

- Author of *Zend Framework in Action* (Manning)
- Zend Framework contributor and certification author
- 19ft.com

160+ Contributors

https://github.com/slimphp/Slim/graphs/contributors

History

Version 1

- Introduced in 2011
- Singleton architecture (don't do this kids 😬)
- Inspired by Sinatra
- Mentioned on *The Changelog* podcast
- Codeguys playground

Version 2

- Introduced in 2012
- OOP architecture (better!)
- Inspired by Rack

Version 3

- Introduced in 2015
- DIC, Middleware, HTTP architecture
- Maturity
- FastRoute router
- PSR-7 support
- Streamlined middleware architecture

Installation & Autoloading

Install with Composer

```
composer require slim/slim:~3.0
```

Semantic Versioning

Slim respects Semantic Versioning. All breaking changes will create a new major version. It is safe to rely on these Composer version constraints:

```
^3.0.0
~3.0
```

PSR-4 Autoloader

```
<?php
require ‘vendor/autoload.php’;

$app = new \Slim\App;
// Define routes
$app->run();
```

Objects

Slim has four primary objects

Request

- Method
- Uri
- Headers
- Body (JSON, XML, Form URL Encoding)
- Query parameters

Response

- Status code
- Headers
- Body

Router

- Define routes (method, pattern, callable)
- Create groups (route prefixes)
- Dispatch requests
- Generate URLs for named routes

Environment

- Decouple Slim from global environment
- Mock requests for unit tests

Re-organization

- `\Slim\App` class simplified in 3.0
- Request methods are on the Request object
- Response methods are on the Response object
- HTTP caching moved to external component

Routing

Built with FastRoute

What is a route?

- A combination of method(s), URI pattern, and callable.
- Slim provides proxy methods on **`\Slim\App`** for major HTTP verbs: GET, POST, PUT, DELETE, OPTIONS, PATCH.

```
<?php
$app->get('/users/{id}', function ($req, $res, $args) {
    // Callable here
});
```

Route URI Pattern

- Built on FastRoute by Nikita Popov
- https://github.com/nikic/FastRoute

- Provides regular expression placeholders

```
<?php
$app->get('/users/{id:[0-9]+}', function ($req, $res, $args) {
   echo 'User ID is: ' . $args['id'];
});
```

Route Callable

- Anything that is callable and accepts three arguments: request, response, URI arguments.
- “Controller:action” callable syntax delegates to container and invokes action on resultant object.

```
<?php
$app->get(‘/users/{id}’, function ($req, $res, $args) {
    // Callable here
});


$app->put(‘/users/{id}’, Users::class . ’:update’);


$app->put(‘/users/{id}’, ‘Users:update’);
```

Route Groups

- Quickly prefix related routes with groups

```
<?php
$app->group('/admin', function () use ($app) {
    $app->get('/users/{id}', function ($req, $res, $args) {
        // Callable here
    });
});
```

Route Return Value

- Routes SHOULD return a PSR-7 response object
- Routes MAY echo a string (sorry, @GeeH 😘)

```
<?php
$app->get('/users/{id}', function ($req, $res, $args) {
    $res->getBody()->write('Hello');

    return $res;
});
```

Multiple Methods

- Quickly assign multiple HTTP methods to a single pattern and callable

```
<?php
$app->map(
    ['GET', 'PUT'],
    '/users/{id}',
    function ($req, $res, $args) {
        if ($req->isPut()) {
            // Do something
        }
    }
);
```

Dependency Injection

Or how to bring your own stuff

Container-Interop

- Supports Container-Interop interface
- Provides default container based on Pimple
- Goal: How do we have least complexity with most flexibility?

Constructor Injection

- Container is injected via constructor
- Third-party container MUST provide expected objects (see `\Slim\Container`)
- You may also inject an array of application settings into constructor (legacy)

Container Objects

Container must have these objects:

- request (facepalm)
- response (facepalm)
- router
- settings
- environment (facepalm)
- notFoundHandler
- notAllowedHandler
- errorHandler

Container Usage

Instantiate container externally and inject:

```
<?php
$app = new \Slim\App(new \My\Interop\Container);
```

Get container internally:

```
$app->get('/user/{id}', function ($req, $res, $args) {
    // Explicit
    $container = $this->getContainer();

    // Implicit via __get() or __isset()
    $request = $this->request;
});
```

Autocomplete container usage

```php
<?php
namespace App\Dependencies\Controllers;

use Pimple\Container;

class PaymentControllerProvider
{
}
```

PDO Integration

- https://php.net/manual/book.pdo.php

Prepare Container

Create a new factory method on the container that prepares and returns a PDO instance.

```
<?php
$c = new \Slim\Container;
$c['db'] = function ($c) {
    return new \PDO(
        sprintf(
            'mysql:host=%s;port=%s;dbname=%s',
            $c['settings']['db_host'],
            $c['settings']['db_port'],
            $c['settings']['db_name']
        ),
        $c['settings']['db_user'],
        $c['settings']['db_pass']
    );
};
```

Use Container

Fetch PDO instance via implicit call to container.

```
<?php
$app = new \Slim\App($c);
$app->get('/users', function ($req, $res, $args) {
    $result = $this->pdo->query('SELECT name, email FROM users');
    // Render results…
});
$app->run();
```

Twig Integration

- http://twig.sensiolabs.org/

Install Component

The `slim/twig-view` component is a Pimple service provider. It registers a Twig instance with the container using the “view” key.

```
$ composer require slim/twig-view
```

Register Service Provider

```
<?php
$c = new \Slim\Container;
$c->register(new \Slim\Views\Twig('path/to/templates', [
    'cache' => 'path/to/cache'
]);
```

Use Service Provider

Render a template using the Twig service via implicit call to container.

```
$app = new \Slim\App($c);
$app->get(‘/users/{name}’, function ($req, $res, $args) {
    return $this->view->render($res, 'profile.html', [
        'name' => $args[‘name']
    ]);
});
$app->run();
```

PSR-7

It was accepted!😂

What is PSR-7?

The purpose of this PSR is to provide a set of common interfaces for HTTP messages as described in RFC 7230 and RFC 7231

https://github.com/php-fig/http-message

Built-in PSR-7 Classes

Slim supports PSR-7, and it provides default implementations for these objects:

- Request
- Response
- Body
- Uri
- UploadedFile

Value Objects

- Each Request and Response object is immutable.
- Changes require a new clone with the modified attributes.
- Minimal overhead (1000s required to affect perf)
- Improves confidence, predictability
- Use the `with*()` PSR-7 interface methods

Value Objects

Each Request and Response instance is immutable. However, you can *clone* an object that has the desired property changes.

```
<?php
$res1 = new \Slim\Http\Response;
$res2 = $res1->withStatus(301);
$res3 = $res2->withHeader('Location', '/foo');
```

External PSR-7 Classes

- Slim lets you inject alternative PSR-7 implementations, too, via the container.

- I hope that we will extract Slim's default PSR-7 classes into their own "slim/http" component.

Request URI Parsing

- Simplified and more predictable in 3.0.
- Occurs in `\Slim\Http\Uri::createFromEnvironment()`
- Derived from Environment's **`SCRIPT_NAME`** and **`REQUEST_URI`**.
- Detects URL rewriting automatically.

Body Parsing

Slim automatically parses JSON, XML, and application/x-www-form-urlencoded HTTP request bodies.

```
POST / HTTP/1.1
Host: example.com
Content-Type: application/json
Content-Length: 14

{"foo": "bar"}
```

Body Parsing

You can fetch the parsed body with the PSR-7 Request object's **`getParsedBody()`** method.

```
<?php
$app->get(‘/‘, function ($req, $res, $args) {
    $data = $req->getParsedBody();
    // [“foo” => “bar”]
});
```

Middleware

Inspect or modify the request and response
around your application.

What is Middleware?

- Code that surrounds your application and/or route in concentric (onion) layers.

- Opportunity to inspect and manipulate the HTTP request and response objects before or after the application runs.

- Responsible for calling (or skipping) the next downstream middleware.

- Lets us expand Slim's feature set without bloating the core repository.

Interface

- Both the application and its routes support the same middleware interface.
- Callable that accepts three arguments: request, response, next middleware:
    - \Psr\Http\Message\RequestInterface
    - \Psr\Http\Message\ResponseInterface
    - callable
- Callable MUST return a PSR-7 response object

The Basics

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

Add Middleware

Slim organizes middleware layers as linked stack.

```
<?php
$app = new \Slim\App;

// App middleware
$app->add($middlewareCallable);

// Route middleware
$app->get('/foo', function ($req, $res, $args) {
    // Route callable
})->add($middlewareCallable);
```

Example Uses

- Authentication
- HTTP caching
- CSRF protection
- Flash messaging

Available on GitHub

- https://github.com/slimphp/Twig-View
- https://github.com/slimphp/Slim-Csrf
- https://github.com/slimphp/Slim-HttpCache
- https://github.com/slimphp/Slim-Flash

Error Handling

You have complete control of the response
when things go wrong.

5xx Server Error

- Callable to handle otherwise uncaught exceptions or errors.
- Container key “errorHandler”
- Accepts three arguments:
    - \Psr\Http\Message\RequestInterface
    - \Psr\Http\Message\ResponseInterface
    - \Exception
- Callable MUST return a PSR-7 response object

5xx Server Error

```
<?php
$c = new \Slim\Container;
$c[‘errorHandler’] = function () {
    return function ($req, $res, $e) {
        return $res->withStatus(500);
    };
};
$app = new \Slim\App($c);
```

404 Not Found

- Callable invoked when matching route not found.
- Container key “notFoundHandler”
- Accepts two arguments:
  - \Psr\Http\Message\RequestInterface
  - \Psr\Http\Message\ResponseInterface
- Callable MUST return a PSR-7 response object

404 Not Found

```
<?php
$c = new \Slim\Container;
$c[‘notFoundHandler’] = function () {
    return function ($req, $res) {
        return $res->withStatus(404)
                   ->write(“Not Found”);
    };
};
$app = new \Slim\App($c);
```

405 Not Allowed

- Callable invoked when route matches request URI but not request method.
- Container key “notAllowedHandler”
- Accepts three arguments:
  - \Psr\Http\Message\RequestInterface
  - \Psr\Http\Message\ResponseInterface
  - Array of allowed HTTP methods
- Callable MUST return a PSR-7 response object

405 Not Allowed

```
<?php
$c = new \Slim\Container;
$c[‘notAllowedHandler’] = function () {
    return function ($req, $res, $methods) {
        return $response
            ->withStatus(405)
            ->withHeader(
                ‘Allow',
                implode(', ', $methods)
            );
    };
};
$app = new \Slim\App($c);
```

Or you can use Whoops wrapper

```
composer require dopesong/slim-whoops:~2.0
```

Usage

```php
use Dopesong\Slim\Error\Whoops as WhoopsError;

require_once "vendor/autoload.php";

$app = new Slim\App();
$container = $app->getContainer();

$container['phpErrorHandler'] = $container['errorHandler'] = function($c) {
    return new WhoopsError($c->get('settings')['displayErrorDetails']);
};

$app->run();
```

Testing

Overview

- We use PHPUnit.
- Best way to learn is to read the existing tests in the `tests/` directory.
- `Environment::mock()` is your friend!
- Tests always need improvement if you don't know how else to help.

Run Tests

Navigate to the project root directory and execute the `phpunit` command.

```
> phpunit
```

What is the Environment?

- The `Environment` class decouples the Slim application from the **`$_SERVER`** super global.
- Mimics the **`$_SERVER`** super global.
- The `SCRIPT_NAME` and `REQUEST_URI` keys are required.
- Create custom environment instances with the `Environment::mock()` static method.

Custom Environment

```
// Create mock environment
$env = \Slim\Http\Environment::mock([
    'SCRIPT_NAME' => '/index.php', // <— REQUIRED
    'REQUEST_URI' => '/users/2', // <—REQUIRED
    'METHOD' => 'POST'
]);
```

```
// Derive PSR-7 URI
$uri = \Slim\Http\Uri::createFromEnvironment($env);

// Derive headers
$headers = \Slim\Http\Headers::createFromEnvironment($env);

// Derive cookies
$cookies = \Slim\Http\Cookies::parseHeader($headers->get('Cookie', []));

// Derive server parameters
$serverParams = $env->all();
```

Branching Strategy

NOT Git Flow

Slim version 1 and 2 (unsuccessfully) used the Git Flow branching strategy. This created more confusion than anything else. It wasn't a good fit for the Slim project.

Currently

The current `2.x` branch contains code for version v2.
Only accepts security fixes.

The current `3.x` branch contains code for version v3.
This is under active development.

The current `4.x` branch contains code for version v4.
This is under active development.

Slim Ecosystem

Hint: It’s components

- Following PHP ecosystem trend toward distributed, standalone components.

- The Slim core repository will remain lightweight.

- Additional functionality will be introduced as separate PHP components in the form of Slim middleware.

- See Slim-Csrf, Slim-HttpCache, Slim-Flash

Contributing

https://github.com/slimphp/

How can you help?

- Read the GitHub Issue Tracker.
- Ask for feedback before working on larger tasks.
- Leave a note on the issues you intend to work on to avoid duplicate work.
- We always need help improving tests.
- We always need help improving documentation.

Pull Requests

- All pull requests should be submitted against the appropriate 2.x, 3.x or 4.x branch.
- Must use PSR-2 code style.
- Must have accompanying unit tests.

What’s next?

- Version 4 should be released this summer
- Mostly to tidy up things that are wrong in v3
- Error handlers will be moved to middleware
- Router will be moved to middleware
- DIC will be optional
- PSR-7 implementation will moved to separate repo
- Response, Request, Environment will be removed from DIC

Stay Informed

Documentation

Slim's documentation is available on GitHub and can be improved by anyone. Please send PRs if you see something missing or wrong.

- https://www.slimframework.com/docs/
- https://github.com/slimphp/Slim-Website

Website

Slim's website is available on GitHub and can be improved by anyone. Please send PRs if you see something missing or wrong.

- https://www.slimframework.com/
- https://github.com/slimphp/Slim-Website
- https://coveralls.io/r/slimphp/Slim

Support

Visit Slim's support forum to ask questions and help your fellow Slim users.

- http://discourse.slimframework.com/

Resources

Here are more resources where you can follow Slim news and updates:

- http://www.slimframework.com/blog/
- https://twitter.com/slimphp
- https://slimphp.slack.com
- Freenode IRC #slimphp channel

Thank You!

Swag!


CAUTION
LOW MOVING
VEHICLE
WARNING
THIS VEHICLE
TURNS INTO A
GIANT
ROBOT
CAUTION
VALVOLINE
HOLES