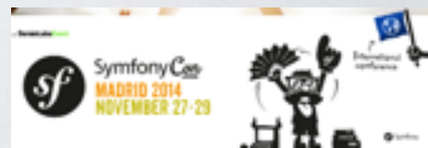
PHP OBJECT MOCKING FRAMEWORK WORLD

LET'S COMPARE PROPHECY AND PHPUNIT

Sarah Khalil - @saro0h



TODAY, HOPEFULLY, WE LEARN NEW THINGS

1. Terminology about objects doubling
2. PHPUnit implementation
3. Prophecy implementation
4. The differences between the two philosophies

TERMINOLOGY

DUMMIES

Dummies are objects that are passed around but never used. They are usually used to fill a list of parameters.

STUB

Stubs are objects that implement the same methods than the real object.

These methods do nothing and are configured to return a specific value.

MOCK

Mocks are pre-programmed with **expectations** which form a specification of the calls they are **expected** to receive. They can throw an exception if they receive a call they don't expect and are checked during verification to **ensure they got all the calls they were expecting**.

PHPUNIT MOCKING LIBRARY

INSTALL IT

4.3.5 reference: 2dab9d5

2014-11-11 10:11 UTC
BSD-3-Clause

require: `"phpunit/phpunit": "4.3.5"`

Author

- Sebastian Bergmann <sebastian@phpunit.de>

Requires

- php: >=5.3.3
- phpunit/php-file-iterator: ~1.3.2
- phpunit/php-text-template: ~1.2
- phpunit/php-code-coverage: ~2.0
- phpunit/php-timer: ~1.0.2
- phpunit/phpunit-mock-objects: ~2.3
- symfony/yaml: ~2.0
- sebastian/comparator: ~1.0
- sebastian/diff: ~1.1
- sebastian/environment: ~1.0
- sebastian/exporter: ~1.0
- sebastian/version: ~1.0
- ext-dom: *
- ext-json: *
- ext-pcre: *
- ext-reflection: *
- ext-spl: *

Requires (Dev)

None

Suggests

- phpunit/php-invoker: ~1.1

Provides

None

Conflicts

None

Replaces

None

FUNDAMENTALS

Your test class must extend

`PHPUnit_Framework_TestCase`

The method you need to know

```
$this->getMock()
```

- Generates an object
- All methods return NULL
- You can describe the expected behavior of your object

DUMMY

```
$dummy = $this->getMock('Namespace');
```

STUB

```
$event = $this->getMock(‘Symfony\Component\HttpKernel\Event
\GetResponseEvent’);
```

```
$event
    ->method('getRequest')
    ->will($this->returnValue($this->request))
;
```

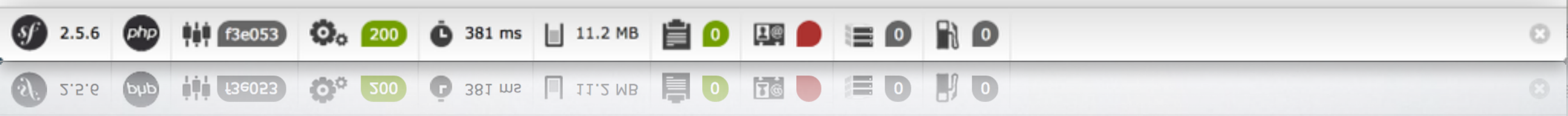
MOCK

```
$dispatcher = $this->getMock(‘Symfony\Component
\EventDispatcher\EventDispatcherInterface');

$dispatcher
        ->expects($this->once())
        ->method(‘dispatch')
        ->with(
            $this->equalTo(SecurityEvents::INTERACTIVE_LOGIN),
            $this->equalTo($loginEvent)
        )
;
```

REAL LIFE EXAMPLE

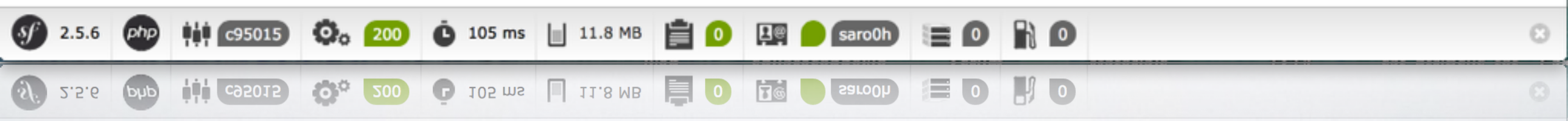
127.0.0.1:8000

Connect with Github



127.0.0.1:8000/admin/github

User authenticated currently : **Sarah KHALIL**



Who knows **exactly** how the Oauth dance of Github works?

Don’t worry, we don’t care!

THE ORIGINAL CODE

```php
namespace PoleDev\AppBundle\Security;

use Guzzle\Service\Client;
use Psr\Log\LoggerInterface;
use Symfony\[…]\Router;
use Symfony\[…]\Response;
use Symfony\[…]\Request;
use Symfony\[…]\SimplePreAuthenticatorInterface;
use Symfony\[…]\AuthenticationFailureHandlerInterface;
use Symfony\[…]\TokenInterface;
use Symfony\[…]\UserProviderInterface;
use Symfony\[…]\AuthenticationException;
use Symfony\[…]\UrlGeneratorInterface;
use Symfony\[…]\HttpException;
use Symfony\[…]\PreAuthenticatedToken;

class GithubAuthenticator implements
SimplePreAuthenticatorInterface,
AuthenticationFailureHandlerInterface
{
   // Some code…
}
```

```php
    private $client;
    private $router;
    private $logger;

    public function __construct(
        Client $client,
        Router $router,
        LoggerInterface $logger,
        $clientId,
        $clientSecret
    )
    {
        $this->client = $client;
        $this->router = $router;
        $this->logger = $logger;
        $this->clientId = $clientId;
        $this->clientSecret = $clientSecret;
    }
```

```
function createToken(Request $request, $providerKey)
{
        $request = $this->client->post(…);
        $response = $request->send();
        $data = $response->json();

        if (isset($data['error'])) {
            $message = ‘An error occured…’;
            $this->logger->notice($message);
            throw new HttpException(401, $message);
        }

        return new PreAuthenticatedToken(
            ‘anon.',
            $data[‘access_token'],
            $providerKey
        );
}
```

ZOOM IN

STEP 1: GET ACCESS TOKEN

```
$url = $this->router->generate(‘admin’,[], true);
$endpoint = ‘/login/oauth/access_token’;

$request = $this->client->post($endpoint,[], [
    'client_id' => $this->clientId,
    'client_secret' => $this->clientSecret,
    'code' => $request->get('code'),
    'redirect_uri' => $url
]);

$response = $request->send();
$data     = $response->json();
```

STEP 2: IF ERROR FROM GITHUB, EXCEPTION

```
if (isset($data['error'])) {
      $message = 'An error occured during authentication with Github.';
      $this->logger->notice($message, [
          'HTTP_CODE_STATUS' => 401,
          ‘error' => $data['error'],
          'error_description'  =>  $data['error_description'],
       ]);

     throw new HttpException(401, $message);
}
```

STEP 3: CREATE TOKEN

```
return new PreAuthenticatedToken(
    'anon.',
    $data['access_token'],
    $providerKey
);
```

LET’S TEST IT!

WHAT TO TEST?

```php
public function createToken(Request $request, $providerKey)
{
        $request = $this->client->post('/login/oauth/access_token', array(), array (
            'client_id' => $this->clientId,
            'client_secret' => $this->clientSecret,
            'code' => $request->get('code'),
            'redirect_uri' => $this->router->generate('admin', array(), UrlGeneratorInterface::ABSOLUTE_URL)
        ));

        $response = $request->send();

        $data = $response->json();

        if (isset($data['error'])) {
            $message = sprintf('An error occured during authentication with Github: (%s)',
$data['error_description']);
            $this->logger->notice(
                $message,
                array(
                    'HTTP_CODE_STATUS' => Response::HTTP_UNAUTHORIZED,
                    'error'  => $data['error'],
                    'error_description'  =>  $data['error_description'],
                )
            );

            throw new HttpException(Response::HTTP_UNAUTHORIZED, $message);
        }

        return new PreAuthenticatedToken(
                'anon.',
                $data['access_token'],
                $providerKey
        );
}
```

We need to test the result of this method

CREATE THE TEST CLASS

```php
namespace PoleDev\AppBundle\Tests\Security;

class GithubAuthenticatorTest extends \PHPUnit_Framework_TestCase
{
    public function testCreateToken()
    {
    }
}
```

LET'S CALL THE CODE WE NEED TO TEST

LET'S GET OUR DUMMIES AND CALL OUR METHOD TO TEST

```
public function testCreateToken()
{
    $githubAuthenticator = new GithubAuthenticator(
        $client,
        $router,
        $logger,
        '',
        ''
    );

    $token = $githubAuthenticator
      ->createToken($request, ‘secure_area')
    ;
}
```

To construct the `GithubAuthenticator`, we need:

- `Guzzle\Service\Client`
- `Symfony\Bundle\FrameworkBundle\Routing\Router`
- `Psr\Log\LoggerInterface`
- `$clientId = ‘’`
- `$clientSecret = ‘’`

To call the `createToken()` method, we need

- `Symfony\Component\HttpFoundation\Request`
- `$providerKey = 'secure_area'`

```
$client = $this->getMock(‘Guzzle\Service
\Client’);
```

This a dummy

```
$router = $this->getMock('Symfony\Bundle
\FrameworkBundle\Routing\Router');

$logger = $this->getMock('Psr\Log
\LoggerInterface');

$request = $this->getMock('Symfony\Component
\HttpFoundation\Request');
```

```
E

Time: 140 ms, Memory: 4.50Mb

There was 1 error:

1) PoleDev\AppBundle\Tests\Security\GithubAuthenticatorTest::testCreateToken
Argument 1 passed to Symfony\Bundle\FrameworkBundle\Routing\Router::__construct() m
ust implement interface Symfony\Component\DependencyInjection\ContainerInterface, n
one given, called in /Users/saro0h/.composer/vendor/phpunit/phpunit-mock-objects/sr
c/Framework/MockObject/Generator.php on line 288 and defined

/Users/saro0h/Documents/talks/symfonycon-madrid-2014/code-exemple/vendor/symfony/sy
mfony/src/Symfony/Bundle/FrameworkBundle/Routing/Router.php:39
/Users/saro0h/Documents/talks/symfonycon-madrid-2014/code-exemple/src/PoleDev/AppBu
ndle/Tests/Security/GithubAuthenticatorTest.php:14

FAILURES!
Tests: 1, Assertions: 0, Errors: 1.
```

We must **disable** the use of the original Router **constructor** as we don't actually care.

```
$router = $this->getMockBuilder('Symfony
\Bundle\FrameworkBundle\Routing\Router')
        ->disableOriginalConstructor()
        ->getMock()
;
```

```
MacBook-Pro-de-Sarah:~/Documents/talks/
symfonycon-madrid-2014/code-exemple$ phpunit -
c app/ src/PoleDev/AppBundle/Tests/Security/
GithubAuthenticatorTest.php

PHPUnit 4.3.5 by Sebastian Bergmann.
Configuration read from /Users/saro0h/
Documents/talks/symfonycon-madrid-2014/code-
exemple/app/phpunit.xml.dist

PHP Fatal error:  Call to a member function
send() on null in /Users/saro0h/Documents/
talks/symfonycon-madrid-2014/code-exemple/src/
PoleDev/AppBundle/Security/
GithubAuthenticator.php on line 43
```

STEP 1: GET ACCESS TOKEN

```
$url = $this->router->generate('admin',[], true);
$endpoint = '/login/oauth/access_token';

$request = $this->client->post($endpoint,[], [
    'client_id' => $this->clientId,
    'client_secret' => $this->clientSecret,
    'code' => $request->get('code'),
    'redirect_uri' => $url
]);

$response = $request->send();
$data     = $response->json();
```

1/ We need to stub the call to the method

`$router->generate()`

it needs to return an url

```
$router->method('generate')
    ->with('admin',[], true)
    ->willReturn(‘http://domain.name')
;
```

2/ We need to create a dummy `Guzzle\Http\Message\EntityEnclosingRequest` `$guzzleRequest`

```
$guzzleRequest = $this
    ->getMockBuilder('Guzzle\Http
\Message\EntityEnclosingRequest')
    ->disableOriginalConstructor()
    ->getMock()
;
```

3/ We need to stub the call to the method `$client->post()`, it needs to return a `$guzzleRequest`

```
$endpoint = '/login/oauth/access_token';

$client->method('post')
    ->with($endpoint, [], [
        'client_id' => '',
        'client_secret' => '',
        'code' => '',
        'redirect_uri' => 'http://domain.name'
    ])
    ->willReturn($guzzleRequest)
;
```

ORIGINAL CODE (STEP 1: GET ACCESS TOKEN)

```
$url = $this->router->generate(‘admin’,[], true);
$endpoint = ‘/login/oauth/access_token’;

$request = $this->client->post($endpoint,[], [
    'client_id' => $this->clientId,
    'client_secret' => $this->clientSecret,
    'code' => $request->get('code'),
    'redirect_uri' => $url
]);

$response = $request->send();
$data     = $response->json();
```

Create a stub for **`$request->send()`**

This method must return a:

`Guzzle\Http\Message\Response $response`

Let's go for it!

```
$guzzleResponse = $this->getMockBuilder('Guzzle\Http
\Message\Response')
    ->disableOriginalConstructor()
    ->getMock()
;

$guzzleRequest
    ->method('send')
  ->willReturn($guzzleResponse)
;
```

```
.

Time: 46 ms, Memory: 4.00Mb

OK (1 test, 0 assertions)
```

Hurray! The original code is running with our **dummies** and **stubs**.

But we do not test anything.

A LITTLE REMINDER

```php
public function createToken(Request $request, $providerKey)
{
        $request = $this->client->post('/login/oauth/access_token', array(), array (
            'client_id' => $this->clientId,
            'client_secret' => $this->clientSecret,
            'code' => $request->get('code'),
            'redirect_uri' => $this->router->generate('admin', array(), UrlGeneratorInterface::ABSOLUTE_URL)
        ));

        $response = $request->send();

        $data = $response->json();

        if (isset($data['error'])) {
            $message = sprintf('An error occured during authentication with Github: (%s)',
$data['error_description']);
            $this->logger->notice(
                $message,
                array(
                    'HTTP_CODE_STATUS' => Response::HTTP_UNAUTHORIZED,
                    'error'  => $data['error'],
                    'error_description'  =>  $data['error_description'],
                )
            );

            throw new HttpException(Response::HTTP_UNAUTHORIZED, $message);
        }

        return new PreAuthenticatedToken(
                'anon.',
                $data['access_token'],
                $providerKey
        );
}
```

We need to test the result of this method

TEST THAT THE **TOKEN** IS WHAT WE NEED TO BE

```
$token = $githubAuthenticator->createToken($request, ‘secure_area’);
$this->assertSame('a_fake_access_token', $token->getCredentials());
$this->assertSame('secure_area', $token->getProviderKey());
$this->assertSame('anon.', $token->getUser());
$this->assertEmpty($token->getRoles());
$this->assertFalse($token->isAuthenticated());
$this->assertEmpty($token->getAttributes());
```

```
F

Time: 163 ms, Memory: 6.00Mb

There was 1 failure:

1) PoleDev\AppBundle\Tests\Security\GithubAuthenticatorTest::testCreateToken
Failed asserting that null is identical to 'a_fake_access_token'.

/Users/saro0h/Documents/talks/symfonycon-madrid-2014/code-exemple/src/PoleDev
s/Security/GithubAuthenticatorTest.php:50

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

Duck! Our token has its **credentials at null**.
You need to **provide it** as the **real code would have done it**.

Github returns an access token at that point.

```
$guzzleResponse
        ->method('json')
        ->willReturn([
            'access_token' =>
            'a_fake_access_token'
        ])
;
```

```
.

Time: 63 ms, Memory: 6.00Mb

OK (1 test, 6 assertions)
```

Hurray! The original code is running with our **dummies** and **stubs**.

And it is tested !

USING A MOCK THIS TIME

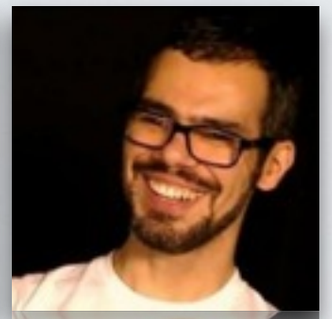
```
$guzzleResponse
    ->expects($this->once())
    ->method('json')
    ->willReturn([
        'access_token' => 'a_fake_access_token'
    ])
;
```

Expectation create a new assertion.

```
.

Time: 160 ms, Memory: 6.50Mb

OK (1 test, 7 assertions)
```

PROPHECY

INSTALL IT

v1.3.1 reference: 9ca5232

2014-11-17 16:23 UTC
MIT

require: `"phpspec/prophecy": "1.3.1"`

Authors

- Konstantin Kudryashov <ever.zet@gmail.com>
- Marcello Duarte <marcello.duarte@gmail.com>

Requires

- phpdocumentor/reflection-docblock: ~2.0
- doctrine/instantiator: ~1.0,>=1.0.2

Requires (Dev)

- phpspec/phpspec: ~2.0

Suggests

None

Provides

None

Conflicts

None

Replaces

None

FUNDAMENTALS

PROPHET

PROPHET

```
$prophet = new \Prophecy\Prophet;
```

DUMMY

DUMMY

```
$routerObjectProphecy = $prophet
   ->prophesize('Symfony\Bundle\FrameworkBundle\Routing\Router')
;

$router = $routerObjectProphecy->reveal();
```

dummy

PROPHECY

PROPHECY

- Object used to **describe the future of your objects**.

```
$prophecy = $prophet->prophesize('YourClass');
```

Note that the prophet won't use the original constructor.

OBJECT DOUBLE

The goal is to **get** the object double

```
$prophecy->reveal();
```

STUB 1/2

- Get a **stub** out from the Router of Symfony.
- `$router->generate()` must return http://www.google.com

STUB 2/2

```
$prophecy
    ->generate('route_name')
    ->willReturn('http://www.google.com')
;
```

PROMISE

- `willReturn() =>` is actually a promise.

A promise is **a piece of code** allowing that a **method call** with a certain argument (if there is one), **returns always the same value**.

PROMISE - API

`$prophecy->willReturn('my value');`	Returns the value 'my value'.
`$prophecy->willReturnArgument();`	Returns the first method argument.
`$prophecy->willThrow('ExceptionClass');`	Throws an exception.

`$prophecy->will($callback)`	`$prophecy->will(new Prophecy\Promise\ReturnPromise(array('my value')));`
	`===`
	`$prophecy->willReturn('my value');`

Details about the implementation:

https://github.com/phpspec/prophecy/blob/master/src/Prophecy/Prophecy/MethodProphecy.php

NOT ENOUGH?

Implement the

`Prophecy\Promise\PromiseInterface`

ARGUMENT

HARD CODED ARGUMENT

```
$prophecy
    ->generate(‘route_name’)
    ->willReturn(‘http://www.google.com’)
;
```

NO HARD CODE

- Prophecy offers you plenty of methods to **« wildcard » the arguments**
- Any argument is ok for the method you are « stubbing »

```
Prophecy\Argument::any()
```

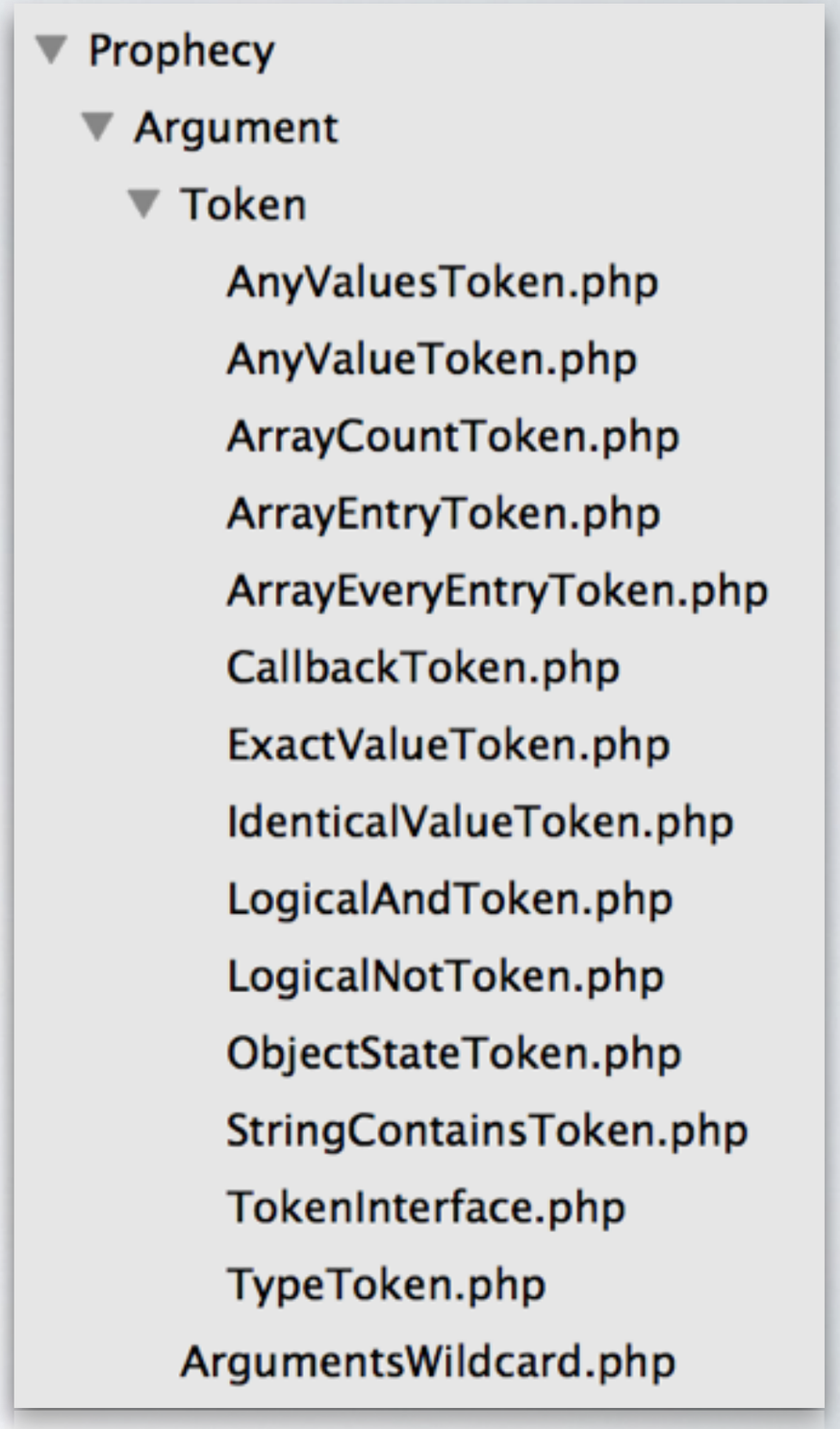
```
$prophecy
    ->myMethod(Prophecy\Argument::any())
;
```

ARGUMENT THE API

- Pretty complete
- To go further with this

https://github.com/phpspec/prophecy#arguments-wildcarding

=>



MOCK

ADD EXPECTATIONS

```
$prophecy
    ->generate('route_name')
    ->willReturn('http://www.google.com')
   ->shouldBeCalled()
;
```

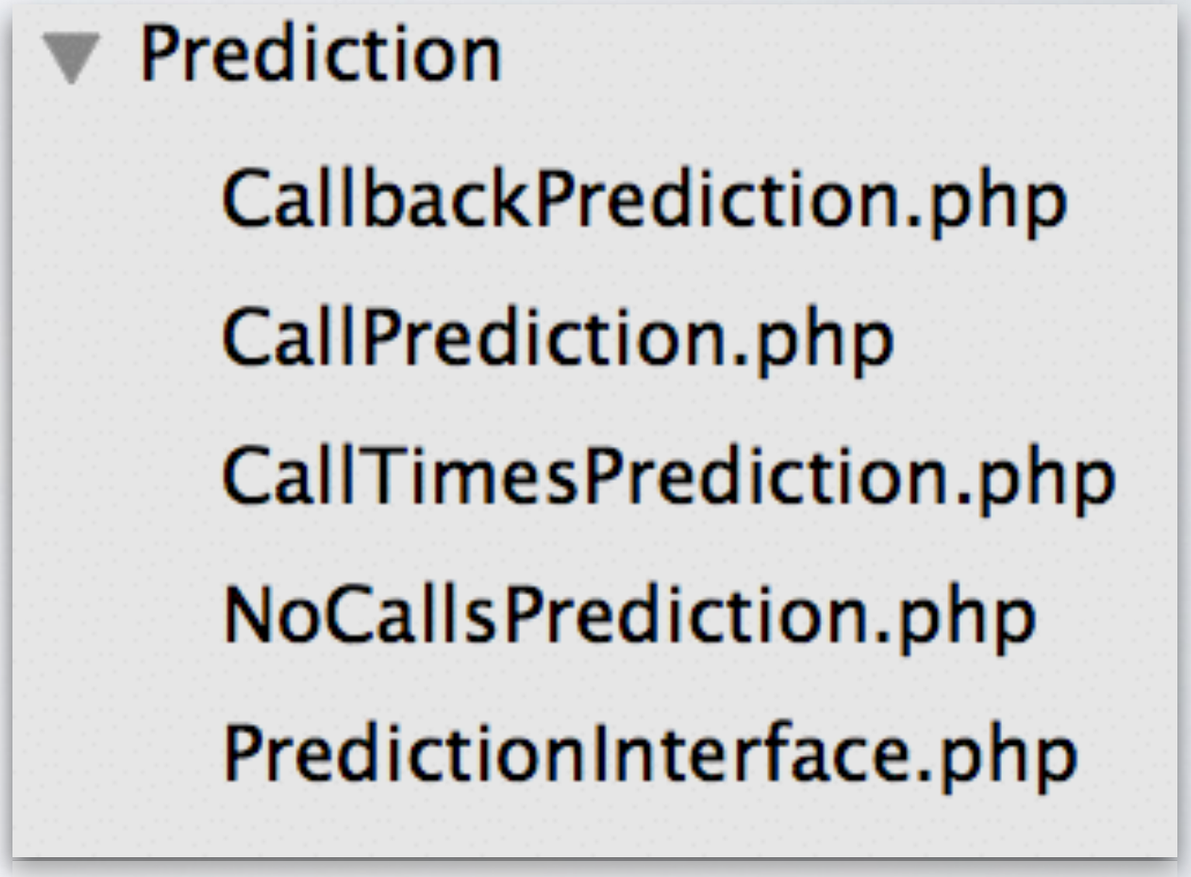
We expect to have that method `generate()` called at least one time.

How we call it in real life ?

Predictions!

PREDICTIONS API

ShouldBeCalled()
shouldNotBeCalled()
shouldBeCalledTimes(2)



NOT ENOUGH?

Implement the

`Prediction\PredictionInterface`

```
$prophet->checkPredictions();
```

(*in your* `tearDown()` *method to check for all tests*)

If the prediction **fails**, it **throws an exception.**

SPIES

Verifies that a method has been called during the execution

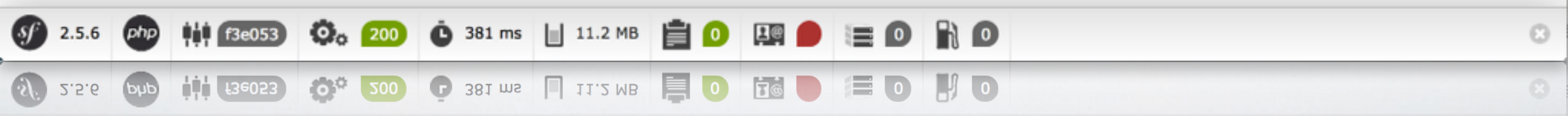
```
$em = $prophet->prophesize('Doctrine\ORM\EntityManager');

$controller->createUser($em->reveal());

$em->flush()->shouldHaveBeenCalled();
```

Exemple taken from the official prophecy repository

REAL LIFE EXAMPLE

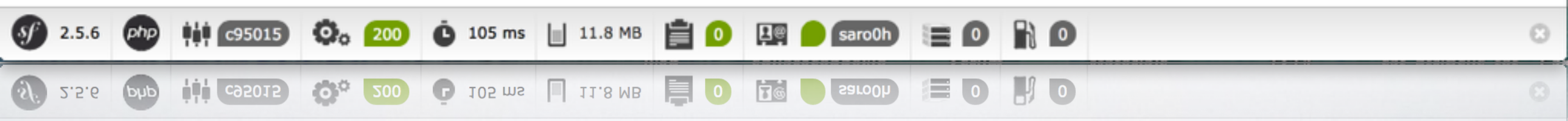
127.0.0.1:8000

Connect with Github



127.0.0.1:8000/admin/github

User authenticated currently : **Sarah KHALIL**



THE ORIGINAL CODE

You don’t remember right…

```php
namespace PoleDev\AppBundle\Security;

use Guzzle\Service\Client;
use Psr\Log\LoggerInterface;
use Symfony\[…]\Router;
use Symfony\[…]\Response;
use Symfony\[…]\Request;
use Symfony\[…]\SimplePreAuthenticatorInterface;
use Symfony\[…]\AuthenticationFailureHandlerInterface;
use Symfony\[…]\TokenInterface;
use Symfony\[…]\UserProviderInterface;
use Symfony\[…]\AuthenticationException;
use Symfony\[…]\UrlGeneratorInterface;
use Symfony\[…]\HttpException;
use Symfony\[…]\PreAuthenticatedToken;

class GithubAuthenticator implements
SimplePreAuthenticatorInterface,
AuthenticationFailureHandlerInterface
{
   // Some code…
}
```

```php
    private $client;
    private $router;
    private $logger;

    public function __construct(
        Client $client,
        Router $router,
        LoggerInterface $logger,
        $clientId,
        $clientSecret
    )
    {
        $this->client = $client;
        $this->router = $router;
        $this->logger = $logger;
        $this->clientId = $clientId;
        $this->clientSecret = $clientSecret;
    }
```

```
function createToken(Request $request, $providerKey)
{
        $request = $this->client->post(…);
        $response = $request->send();
        $data = $response->json();

        if (isset($data['error'])) {
            $message = ‘An error occured…’;
            $this->logger->notice($message);
            throw new HttpException(401, $message);
        }

        return new PreAuthenticatedToken(
            ‘anon.',
            $data[‘access_token'],
            $providerKey
        );
}
```

FOCUS

STEP 1: GET ACCESS TOKEN

```
$url = $this->router->generate(‘admin’,[], true);
$endpoint = ‘/login/oauth/access_token’;

$request = $this->client->post($endpoint,[], [
    'client_id' => $this->clientId,
    'client_secret' => $this->clientSecret,
    'code' => $request->get('code'),
    'redirect_uri' => $url
]);

$response = $request->send();
$data     = $response->json();
```

STEP 2: IF ERROR FROM GITHUB, EXCEPTION

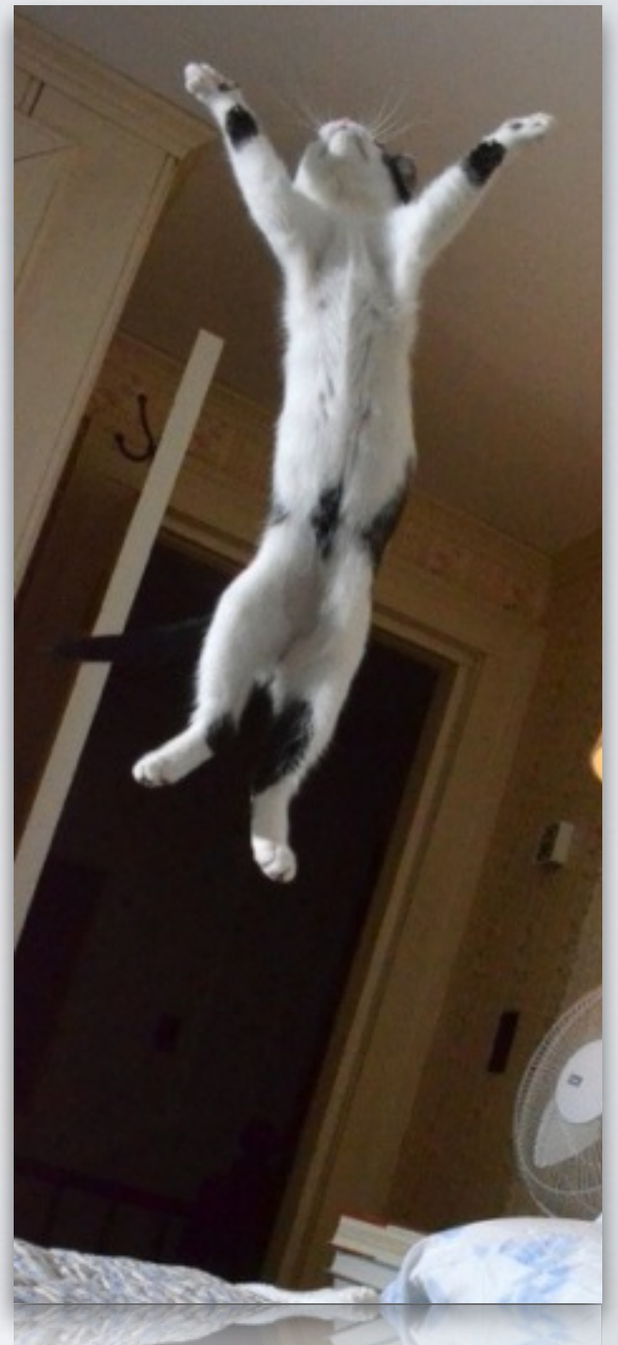
```
if (isset($data['error'])) {
      $message = 'An error occured during authentication with Github.';
      $this->logger->notice($message, [
          'HTTP_CODE_STATUS' => 401,
          ‘error' => $data['error'],
          'error_description'  =>  $data['error_description'],
       ]);

     throw new HttpException(401, $message);
}
```

STEP 3: CREATE TOKEN

```
return new PreAuthenticatedToken(
    'anon.',
    $data['access_token'],
    $providerKey
);
```

LET'S TEST IT!

AGAIN !



WHAT TO TEST?

```php
public function createToken(Request $request, $providerKey)
{
        $request = $this->client->post('/login/oauth/access_token', array(), array (
            'client_id' => $this->clientId,
            'client_secret' => $this->clientSecret,
            'code' => $request->get('code'),
            'redirect_uri' => $this->router->generate('admin', array(), UrlGeneratorInterface::ABSOLUTE_URL)
        ));

        $response = $request->send();

        $data = $response->json();

        if (isset($data['error'])) {
            $message = sprintf('An error occured during authentication with Github: (%s)',
$data['error_description']);
            $this->logger->notice(
                $message,
                array(
                    'HTTP_CODE_STATUS' => Response::HTTP_UNAUTHORIZED,
                    'error'  => $data['error'],
                    'error_description'  =>  $data['error_description'],
                )
            );

            throw new HttpException(Response::HTTP_UNAUTHORIZED, $message);
        }

        return new PreAuthenticatedToken(
                'anon.',
                $data['access_token'],
                $providerKey
        );
}
```

We need to test the result of this method

CREATE THE TEST CLASS

```php
namespace PoleDev\AppBundle\Tests\Security;

class GithubAuthenticatorTest extends \PHPUnit_Framework_TestCase
{
    public function testCreateToken()
    {
    }
}
```

FIRST, GET THE PROPHET

```php
namespace PoleDev\AppBundle\Tests\Security;

class GithubAuthenticatorTest extends \PHPUnit_Framework_TestCase
{
    private $prophet;

    public function testCreateToken()
    {
    }

    public function setUp()
    {
        $this->prophet = new \Prophecy\Prophet;
    }

    public function tearDown()
    {
        $this->prophet = null;
    }
}
```

LET'S GET OUR DUMMIES AND CALL OUR METHOD TO TEST

```
public function testCreateToken()
{
    $githubAuthenticator = new GithubAuthenticator(
        $client,
        $router,
        $logger,
        '',
        ''
    );

    $token = $githubAuthenticator
      ->createToken($request, ‘secure_area')
    ;
}
```

To construct the `GithubAuthenticator`, we need

- `Guzzle\Service\Client`
- `Symfony\Bundle\FrameworkBundle\Routing\Router`
- `Psr\Log\LoggerInterface`
- `$clientId = ‘’`
- `$clientSecret = ‘’`

To call the `createToken()` method, we need

- `Symfony\Component\HttpFoundation\Request`
- `$providerKey = 'secure_area'`

LET'S GET OUR DUMMIES AND CALL OUR METHOD TO TEST

```php
public function testCreateToken()
{
    $clientObjectProphecy = $this->prophet->prophesize('Guzzle\Service\Client');
    $client = $clientObjectProphecy->reveal();

    // …
}
```

This a prophecy

This a dummy

```
$routerObjectProphecy = $this->prophet
    ->prophesize('Symfony\Bundle\FrameworkBundle
\Routing\Router');
$router = $routerObjectProphecy->reveal();

$loggerObjectProphecy = $this->prophet
    ->prophesize('Psr\Log\LoggerInterface');
$logger = $loggerObjectProphecy->reveal();

$requestObjectProphecy = $this->prophet
    ->prophesize('Symfony\Component\HttpFoundation
\Request');
$request = $requestObjectProphecy->reveal();
```

```
MacBook-Pro-de-Sarah:~/Documents/talks/
symfonycon-madrid-2014/code-exemple$ phpunit -
c app/ src/PoleDev/AppBundle/Tests/Security/
GithubAuthenticatorTest.php

PHPUnit 4.3.5 by Sebastian Bergmann.
Configuration read from /Users/saro0h/
Documents/talks/symfonycon-madrid-2014/code-
exemple/app/phpunit.xml.dist

PHP Fatal error:  Call to a member function
send() on null in /Users/saro0h/Documents/
talks/symfonycon-madrid-2014/code-exemple/src/
PoleDev/AppBundle/Security/
GithubAuthenticator.php on line 43
```

The null object expected is a:

```
Guzzle\Http\Message\EntityEnclosingRequest $request
```

Let's provide it!

```
$guzzleRequestObjectProphecy = $this
   ->prophet
   ->prophesize(‘Guzzle\Http\Message\EntityEnclosingRequest')
;

$guzzleRequest = $guzzleRequestObjectProphecy->reveal();
```

ORIGINAL CODE (STEP 1: GET ACCESS TOKEN)

```
$url = $this->router->generate(‘admin’,[], true);
$endpoint = ‘/login/oauth/access_token’;

$request = $this->client->post($endpoint,[], [
    'client_id' => $this->clientId,
    'client_secret' => $this->clientSecret,
    'code' => $request->get('code'),
    'redirect_uri' => $url
]);

$response = $request->send();
$data     = $response->json();
```

```
$clientObjectProphecy = $this->prophet
->prophesize('Guzzle\Service\Client');

$clientObjectProphecy
    ->post('/login/oauth/access_token', [],
[
            'client_id' => ' ',
            'client_secret' => ' ',
            'code' => ' ',
            'redirect_uri' => ' '
    ])
    ->willReturn($guzzleRequest)
;

$client = $clientObjectProphecy->reveal();
```

ORIGINAL CODE (STEP 1: GET ACCESS TOKEN)

```
$url = $this->router->generate(‘admin’,[], true);
$endpoint = ‘/login/oauth/access_token’;

$request = $this->client->post($endpoint,[], [
    'client_id' => $this->clientId,
    'client_secret' => $this->clientSecret,
    'code' => $request->get('code'),
    'redirect_uri' => $url
]);

$response = $request->send();
$data     = $response->json();
```

Create a stub for **`$request->send()`**

This method must return a:

`Guzzle\Http\Message\Response $response`

Let's go for it!

```
$guzzleResponseObjectProphecy = $this->prophet
    ->prophesize('Guzzle\Http\Message\Response');
$guzzleResponse = $guzzleResponseObjectProphecy->reveal();

$guzzleRequestObjectProphecy = $this
    ->prophet
    ->prophesize(‘Guzzle\Http\Message\EntityEnclosingRequest')
;

$guzzleRequestObjectProphecy
    ->send()
    ->willReturn($guzzleResponse)
;

$guzzleRequest = $guzzleRequestObjectProphecy->reveal();
```

```
.

Time: 46 ms, Memory: 4.00Mb

OK (1 test, 0 assertions)
```

Hurray! The original code is running with our **dummies** and **stubs**.

But we do not test anything.

WHAT WE NEED TO TEST AGAIN?

```php
public function createToken(Request $request, $providerKey)
{
        $request = $this->client->post('/login/oauth/access_token', array(), array (
            'client_id' => $this->clientId,
            'client_secret' => $this->clientSecret,
            'code' => $request->get('code'),
            'redirect_uri' => $this->router->generate('admin', array(), UrlGeneratorInterface::ABSOLUTE_URL)
        ));

        $response = $request->send();

        $data = $response->json();

        if (isset($data['error'])) {
            $message = sprintf('An error occured during authentication with Github: (%s)',
$data['error_description']);
            $this->logger->notice(
                $message,
                array(
                    'HTTP_CODE_STATUS' => Response::HTTP_UNAUTHORIZED,
                    'error'  => $data['error'],
                    'error_description'  =>  $data['error_description'],
                )
            );

            throw new HttpException(Response::HTTP_UNAUTHORIZED, $message);
        }

        return new PreAuthenticatedToken(
                'anon.',
                $data['access_token'],
                $providerKey
        );
}
```

We need to test the result of this method

TEST THAT THE **TOKEN** IS WHAT WE NEED TO BE

```
$token = $githubAuthenticator->createToken($request, ‘secure_area’);
$this->assertSame('a_fake_access_token', $token->getCredentials());
$this->assertSame('secure_area', $token->getProviderKey());
$this->assertSame('anon.', $token->getUser());
$this->assertEmpty($token->getRoles());
$this->assertFalse($token->isAuthenticated());
$this->assertEmpty($token->getAttributes());
```

```
F

Time: 163 ms, Memory: 6.00Mb

There was 1 failure:

1) PoleDev\AppBundle\Tests\Security\GithubAuthenticatorTest::testCreateToken
Failed asserting that null is identical to 'a_fake_access_token'.

/Users/saro0h/Documents/talks/symfonycon-madrid-2014/code-exemple/src/PoleDev
s/Security/GithubAuthenticatorTest.php:50

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

```
object(Symfony\Component\Security\Core\Authentication\Token\PreAuthenticatedToken)#298 (6) {
  ["credentials":"Symfony\Component\Security\Core\Authentication\Token\PreAuthenticatedToken":private]=>
  NULL
  ["providerKey":"Symfony\Component\Security\Core\Authentication\Token\PreAuthenticatedToken":private]=>
  string(11) "secured_area"
  ["user":"Symfony\Component\Security\Core\Authentication\Token\AbstractToken":private]=>
  string(5) "anon."
  ["roles":"Symfony\Component\Security\Core\Authentication\Token\AbstractToken":private]=>
  array(0) {
  }
  ["authenticated":"Symfony\Component\Security\Core\Authentication\Token\AbstractToken":private]=>
  bool(false)
  ["attributes":"Symfony\Component\Security\Core\Authentication\Token\AbstractToken":private]=>
  array(0) {
  }
}
```

Remember that output…

Github returns an access token at that point.

```
$guzzleResponseObjectProphecy = $this->prophet-
>prophesize('Guzzle\Http\Message\Response');

$guzzleResponseObjectProphecy
    ->json()
    ->willReturn(['access_token' => 'a_fake_access_token'])
;

$guzzleResponse = $guzzleResponseObjectProphecy->reveal();
```

```
.

Time: 63 ms, Memory: 6.00Mb

OK (1 test, 6 assertions)
```

Hurray! The original code is running with our **dummies** and **stubs**.

And it is tested !

USING A MOCK THIS TIME

```
$guzzleResponseObjectProphecy
    ->json()
    ->willReturn(array(‘access_token' => 'a_fake_access_token'))
    ->shouldBeCalledTimes(1)
;
```

Don’t expect to get a new assertion, as in PHPUnit

DON'T FORGET ABOUT THE CHECK

```
public function tearDown()
{
    $this->prophet->checkPredictions();
    $this->prophet = null;
}
```

Mandatory

WRONG EXPECTATION

```
$guzzleResponseObjectProphecy
    ->json()
    ->willReturn(array(‘access_token' => 'a_fake_access_token'))
    ->shouldBeCalledTimes(10)
;
```

But if the expectation is not right, you’ll get an **exception.**

```
E

Time: 154 ms, Memory: 6.25Mb

There was 1 error:

1) PoleDev\AppBundle\Tests\Security\GithubAuthenticatorTest::testCreateToken
Prophecy\Exception\Prediction\AggregateException: Some predictions failed:
  Double\Guzzle\Http\Message\Response\P1:
    Expected exactly 10 calls that match:
      Double\Guzzle\Http\Message\Response\P1->json()
    but 1 were made:
      - json() @ src/PoleDev/AppBundle/Security/GithubAuthenticator.php:45

/Users/saro0h/Documents/talks/symfonycon-madrid-2014/code-exemple/vendor/phpspec/prophecy/src/Prophecy/P
rophet.php:121
/Users/saro0h/Documents/talks/symfonycon-madrid-2014/code-exemple/src/PoleDev/AppBundle/Tests/Security/G
ithubAuthenticatorTest.php:70

FAILURES!
Tests: 1, Assertions: 6, Errors: 1.
```

TO SUM UP ALL OF THIS

Prophecy

The Prophecy mock library philosophy is around the description of the future of an object double through a prophecy.

Prophecy

A prophecy must be revealed to get a dummy, a stub or a mock.

With PHPUnit, all is around the getMock() method.

The first step is to get a mock, then describe the future of the double of the object.

PHPUnit

```
$dummy = $this->getMock('Foo\Bar');
```

```
$prophet = new \Prophecy\Prophet;
$prophecy = $prophet->prophesize('Foo\Bar');
$dummy = $prophecy->reveal();
```

Prophecy

Prophecy

```
$prophecy
    ->send()
    ->willReturn($valueToReturn)
;
```

```
$dummy
    ->method('send')
    ->willReturn($valueToReturn)
;
```

PHPUnit

Prophecy

Extensible.

Not extensible.

PHPUnit

AWSOMNESS!

Sebastian Bergmann
@s_bergmann

Following

The support for Prophecy in PHPUnit 4.5 is now documented at phpunit.de/manual/4.5/en/ …

Sebastian Bergmann @s_bergmann · 5m
@kjarli I'll probably recommend Prophecy over PHPUnit's own implementation at some point, yes.

Resources

- https://github.com/phpspec/prophecy
- https://phpunit.de/manual/4.5/en/test-doubles.html#test-doubles.prophecy
- all the code is here: http://bit.ly/1Ipnp2I

joind.in/talk/view/12957

sarah-khalil.com/talks

@saro0h

saro0h