Smaine Milianni Follow

Jul 1, 2021 · 3 min read · Listen

Save

Mock the Symfony HttpClient

👋 Hey, let's continue our series about how to test services and classes that uses external API, this article is the second, in the first one we saw how to test the Symfony Http Client with behat

In this article back to the basics with PHPUnit 🔥

In our use case, we want to expose an endpoint that says if a user exists or not **BUT** we

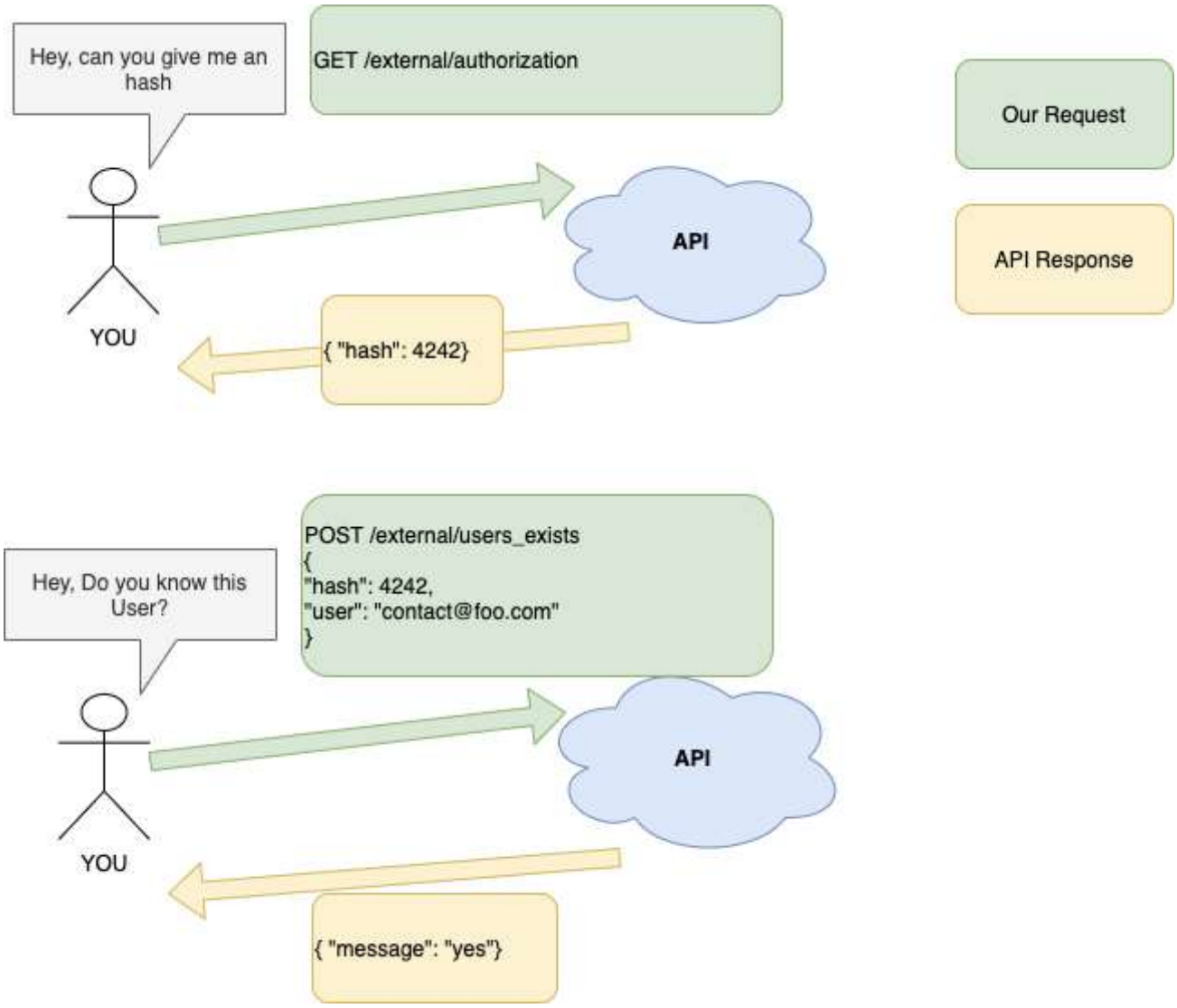
pass the hash and the email of the user we want to check in a **POST** request. At the end, the API tells us if the user exists and then we return this information as the Response of our endpoint.

The workflow is:

1️⃣ Asking for a hash (**GET** request at **/external/authorization**)

2️⃣ Use the hash with the email to verify (**POST** request with hash and email at **/external/user_exists**)

I draw this for better comprehension of the workflow



```php
final class UserExistService
{
    private HttpClientInterface $httpClient;

    public function __construct(HttpClientInterface $httpClient)
    {
        $this->httpClient = $httpClient;
    }

    // The function that will be used is our controller
    public function userExist(string $email): bool
    {
       $hash = $this->getHash();

       return $this->checkUserExistInApi($hash, $email);
    }

    // Here is the 1st call to the API to get the hash.
    private function getHash(): string
    {
        $response = $this->httpClient->request(
            "GET",
            "/external/authorization"
        );

        if (200 !== $response->getCodeStatus()) {
            // 👀 If request is not a success we throw an custom Error that will be catch by t
            // 💡 you can log as well
            throw new UnableToVerifiyUserException(
                $response->getContent(false)
            );
        }

        return $response->toArray(false)["hash"] ?? '';
    }

    // Here is the second call to check if the user exists
    private function checkUserExistInApi(string $hash, string $emailToVerify): bool
    {
        $response = $this->httpClient->request(
            "POST",
            "/external/user_exists",
```

```
            ]
        );

        if (200 !== $response->getCodeStatus()) {
            throw new UnableToVerifiyUserException(
                $response->getContent(false)
            );
        }

        $message = $response->toArray(false)["message"] ?? null;

        return "yes" === $message;
    }
}
```

UserExistService.php hosted with ❤ by GitHub view raw

Now we've created our service, let's use it in our controller:

```php
<?php

final class UserExistAction
{
  private UserExistService $userExistService;

  public function __construct(UserExistService $userExistService)
  {
    $this->userExistService = $userExistService;
  }

  /**
  * @Route("admin/{email}")
  */
  public function __invoke(string $email): JsonResponse
  {
    try {
     $userExist = $this->userExistService->userExist($email);
      // Remeber if we don't get a 200 an exception will be thrown in the service
      // and we catch here to return false (user not exits and cannot access)
    } catch (UnableToVerifiyUserException $e) {
```

UserExistAction.php hosted with ❤ by GitHub view raw

Thanks to the Symfony autowiring and the default configuration, we don't have to declare anything all services will be resolved automatically ✨ 🚀

🍿 The interesting part

🔍 What should we test?

We should test our endpoint `/admin/{email}` and expect that our code does the job, that means when we call `/admin/{email}` and the **external API return "yes" we expect to have a response:**

```
{"user_exists": true}
```

So tests can be like:

```php
<?php

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class ApiTest extends WebTestCase
{
  public function testItReturnsTrueIfUserExistInApi(): void
    {
        $client = static::createClient();
        //💡 We don't care about the email ("smaone@foo.com")
        // We will explicity tells to the API, return "yes" in this test.
        $client->request('GET', '/admin/smaone@foo.com');

        $this->assertResponseIsSuccessful();
        $response = json_decode($client->getResponse()->getContent(), true);

        // We expect to have `TRUE`
        $this->assertEquals(['user_exists' => true], $response);
    }
}
```

```
{"user_exists": false}
```

```php
<?php

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class ApiTest extends WebTestCase
{
  // other test

  public function testItReturnsFalseIfUserNotExistInApi(): void
    {
        $client = static::createClient();

        //💡Same here, We don't care about the email ("smaone@foo.com")
        // We will explicity tells to the API, return "NO" in this test.
        $client->request('GET', '/admin/smaone@foo.com');

        $this->assertResponseIsSuccessful();
        $response = json_decode($client->getResponse()->getContent(), true);

        // We expect to have "FALSE"
        $this->assertEquals(['user_exists' => false], $response);
    }
}
```

testUserExistFalse.php hosted with ❤️ by GitHub view raw

When the API have a code status different than 200 we expect this response

```
{"user_exists": false}
```

```php
<?php

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
```

```
    {
        $client = static::createClient();

        //💡Same here, We don't care about the email ("smaone@foo.com")
        // We will explicity tells to the API, return "NO" in this test.
        $client->request('GET', '/admin/smaone@foo.com');

        $this->assertResponseIsSuccessful();
        $response = json_decode($client->getResponse()->getContent(), true);

        // We expect to have "FALSE"
        $this->assertEquals(['user_exists' => false], $response);
    }
}
```

testUserExistFalseIfErrorOccurs.php hosted with ❤️ by GitHub view raw

💡 To be fair all of these tests are simple and should be written before, to let them drive our development #TDD

📚 How to test this stuff?

In a test environment, you don't care about infrastructure and external services such as API. So we have to mock the API Response and see if our code works as well.

Before each call to our endpoint, we will set the response returned by the request to the API. In order to do this, we will create a `FakeHttpClient` and we inject it in our service `UserServiceExist` . Then before each test we fetch this fake from the container and set it the Response that we want.

1- Create the FakeHttpClient

```
<?php

// this client will be used in our test
final class FakeHttpClient implements HttpClientInterface
{
  private array $responses;

  /**
```

```php
    */
    public function __construct(array $responses = [])
    {
        $this->responses = $responses;
    }


     public function request(string $method, string $url, array $options = []): ResponseInterface
     {
          // Get the reponse by accessing to the "key"
          $response = $this->responses[$url] ?? null;

          if (null === $response) {
              throw new \LogicException(\Safe\sprintf('There is no response for url: %s', $url));
          }

          return (new MockHttpClient($response, 'https://user_service_api.fake'))->request($metho
     }

    public function stream($responses, float $timeout = null): ResponseStreamInterface
      {
          throw new \LogicException(sprintf('%s() is not implemented', __METHOD__));
      }

      public function withOptions(): self
      {
          return $this;
      }
  }
```

mock_http_client_service.php hosted with ❤ by GitHub view raw

2- Inject it in the service ONLY in a test environment

```yaml
# service_test.yaml

imports:
    - 'services.yaml'

    // We override the original http client injected with App\FakeHttpClient
```

3- Use it!

1 Simulate that the external API returns "YES"

```php
<?php

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
use Symfony\Component\HttpClient\Response\MockResponse;
use App\FakeHttpClient;

class ApiTest extends WebTestCase
{
  public function testItReturnsTrueIfUserExistInApi(): void
    {
        $client = static::createClient();

        // create each response that matches an URL
        $responses = [
            '/external/authorization' => new MockResponse(json_encode(['hash' => '4242'])),
            '/external/user_exists' => new MockResponse(json_encode(['user_exists' => 'yes']))
        ];

        // Inject our HttpClient with our Responses in the contaoner
        // in order to be injected in our service `UserExistService`
        self::$container->set(FakeHttpClient::class, new FakeHttpClient($responses));

        $client->request('GET', '/admin/smaone@foo.com');

        $this->assertResponseIsSuccessful();
        $response = json_decode($client->getResponse()->getContent(), true);

        // We expect to have `TRUE`
        $this->assertEquals(['user_exists' => true], $response);
    }
}
```

2 Simulate that the external API returns "NO"

```php
<?php

```

```php

    public function testItReturnsFalseIfUserNotExistInApi(): void
      {
          $client = static::createClient();

          // create each response that matches an URL
          $responses = [
              '/external/authorization' => new MockResponse(json_encode(['hash' => '4242'])),
              '/external/user_exists' => new MockResponse(json_encode(['user_exists' => 'no']))
          ];

          // Inject our HttpClient with our Responses in the contaoner
          // in order to be injected in our service `UserExistService`
          self::$container->set(FakeHttpClient::class, new FakeHttpClient($responses));

          // 💡Same here, We don't care about the email ("smaone@foo.com")
          // We will explicity tells to the API, return "NO" in this test.
          $client->request('GET', '/admin/smaone@foo.com');

          $this->assertResponseIsSuccessful();
          $response = json_decode($client->getResponse()->getContent(), true);

          // We expect to have "FALSE"
          $this->assertEquals(['user_exists' => false], $response);
      }
}
```

Simulate that the external API returns a 400 error

```php
<?php

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class ApiTest extends WebTestCase
{
  // other test

  public function testItReturnsFalseIfUserNotExistInApi(): void
    {
        $client = static::createClient();

```

```
        ];

        // Inject our HttpClient with our Responses in the contaoner
        // in order to be injected in our service `UserExistService`
        self::$container->set(FakeHttpClient::class, new FakeHttpClient($responses));

        //💡Same here, We don't care about the email ("smaone@foo.com")
        // We will explicity tells to the API, return "NO" in this test.
        $client->request('GET', '/admin/smaone@foo.com');

        $this->assertResponseIsSuccessful();
        $response = json_decode($client->getResponse()->getContent(), true);

        // We expect to have "FALSE"
        $this->assertEquals(['user_exists' => false], $response);
    }
}
```

That's All 🎉,

I hope you liked it, don't forget to clap 👏 and share it 🔥