08

Consolidando o seu conhecimento

Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.

1) Digite o seguinte comando para instalar a integração entre o Symfony e PHPUnit: `composer require --dev symfony/phpunit-bridge`.

2) Execute o comando `php bin/phpunit`. Caso você ainda não tenha o PHPUnit instalado no projeto, este comando o instalará.

3) Instale os componentes do Symfony para testes funcionais: `composer require symfony/browser-kit symfony/css-selector`.

4) (Opcional) No arquivo `config/bootstrap.php`, altere a linha que cria a classe `Dotenv` para que o construtor receba `false` como parâmetro `(new Dotenv(false))`.

5) Dentro da pasta `tests`, crie o arquivo `EspecialidadesControllerTest`, com o seguinte conteúdo:

```
<?php

namespace App\Tests;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase

class EspecialidadesControllerTest extends WebTestCase
{
    public function testGaranteQueRequisicaoFalhaSemAutenticacao()
    {
        $client = static::createClient();
        $client->request('GET', '/especialidades');

        self::assertEquals(401, $client->getResponse()->getStatusCode());
    }

    public function testGaranteQueEspecialidadesSaoListadas()
    {

        $client = static::createClient();
        $token = $this->login($client);
        $client->request('GET', '/especialidades', [], [], [
            'HTTP_AUTHORIZATION' => "Bearer $token",
        ]);

        $resposta = $client->getResponse()->getContent();
        self::assertTrue($resposta->success);
    }

    public function testInsereEspecialidade()
    {
        $client = static::createClient();
        $token = $this->login($client);
        $client->request(
            'POST',
            [],
```

```
                [],
                [
                    'CONTENT_TYPE' => 'application/json',
                    'HTTP_AUTHORIZATION' => "Bearer $token",
                ],
                json_encode([
                    'descricao' => 'Teste'
                ])
            );

            self::assertEquals(201, $client->getResponse()->getStatusCode());
        }

        private function login(KernelBrowswer $client): string
        {
            $client->request(
                'POST',
                '/login',
                [],
                [],
                [
                    'CONTENT_TYPE' => 'application/json'
                ],
                json_encode([
                    'username' => 'username', 'password' => 'password'
                ])
            );

            return json_decode($client->getResponse->getContent())->access_token;
        }
    }
```

6) No arquivo `.env.test`, defina a URL do banco de dados da seguinte forma:

```
DATABASE_URL=sqlite://%kernel.project_dir%/var/data/banco.sqlite
```

7) Crie a pasta `data` dentro da pasta `var` no projeto.

8) Execute o comando `php bin/console -e test doctrine:database:create`, para criar o arquivo do banco de dados.

9) Execute `php bin/console -e test doctrine:schema:create`, para criar as tabelas no banco recém-criado.

10) Execute agora `php bin/console -e test doctrine:fixtures:load`, para criar o usuário no banco.

11) Feito isso, execute `php bin/phpunit` e veja que todos os três testes passam, sem erros.