10

Consolidando o seu conhecimento

Chegou a hora de você seguir todos os passos realizados por mim durante esta aula. Caso já tenha feito, excelente. Se ainda não, é importante que você execute o que foi visto nos vídeos para poder continuar com a próxima aula.

1) Na linha de comando, dentro do seu projeto, execute o comando abaixo para baixar o PHPUnit:

```
composer require --dev phpunit/phpunit
```

2) De volta no seu editor de código crie uma nova pasta chamada `tests`. Dentro dela adicione um novo arquivo `TestBuscadorDeCursos.php` com o conteúdo abaixo:

```
<?php

namespace Alura\BuscadorDeCursos\Tests;

use Alura\BuscadorDeCursos\Buscador;
use GuzzleHttp\ClientInterface;
use PHPUnit\Framework\TestCase;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\StreamInterface;
use Symfony\Component\DomCrawler\Crawler;

class TestBuscadorDeCursos extends TestCase
{
    private $httpClientMock;
    private $url = 'url-teste';

    protected function setUp(): void
    {
        $html = <<<FIM
        <html>
            <body>
                <span class="card-curso__nome">Curso Teste 1</span>
                <span class="card-curso__nome">Curso Teste 2</span>
                <span class="card-curso__nome">Curso Teste 3</span>
            </body>
        </html>
        FIM;



        $stream = $this->createMock(StreamInterface::class);
        $stream
            ->expects($this->once())
            ->method('__toString')
            ->willReturn($html);

        $response = $this->createMock(ResponseInterface::class);
        $response
            ->expects($this->once())
            ->method('getBody')
```

```
            ->willReturn($stream);

        $httpClient = $this
            ->createMock(ClientInterface::class);
        $httpClient
            ->expects($this->once())
            ->method('request')
            ->with('GET', $this->url)
            ->willReturn($response);

        $this->httpClientMock = $httpClient;
    }

    public function testBuscadorDeveRetornarCursos()
    {
        $crawler = new Crawler();
        $buscador = new Buscador($this->httpClientMock, $crawler);
        $cursos = $buscador->buscar($this->url);

        $this->assertCount(3, $cursos);
        $this->assertEquals('Curso Teste 1', $cursos[0]);
        $this->assertEquals('Curso Teste 2', $cursos[1]);
        $this->assertEquals('Curso Teste 3', $cursos[2]);
    }
}
```

4) Execute o teste na linha de comando:

```
vendor\bin\phpunit tests\TestBuscadorDeCursos.php
```

3) Agora vamos validar se nosso código segue alguns padrões definidos no PSR. Para tal, vamos usar o *PHP Codesniffer*. Na linha de comando execute:

```
composer require --dev squizlabs/php_codesniffer
composer require --dev phan/phan
```

4) Uma vez baixado verifique o padrão PSR12 executando na linha de comando:

```
vendor\bin\phpcs --standard=PSR12 src\Buscador.php
```

Tente corrigir os erros relacionado com a estrutura PHP e chame novamente `phpcs` .

5) Vamos testar a ferramenta *phan* para encontrar possíveis erros no código.

Primeiramente crie uma nova pasta `.phan` na raiz do seu projeto.

6) Na pasta `.phan` , crie um novo arquivo `config.php` com as configurações abaixo:

```
<?php

return [
```

```
        "target_php_version" => '7.3',
        'directory_list' => [
            'src',
            'vendor/symfony/dom-crawler',
            'vendor/guzzlehttp/guzzle',
            'vendor/psr/http-message'
        ],
        "exclude_analysis_directory_list" => [
            'vendor/'
        ],
        'plugins' => [
            'AlwaysReturnPlugin',
            'UnreachableCodePlugin',
            'DollarDollarPlugin',
            'DuplicateArrayKeyPlugin',
            'PregRegexCheckerPlugin',
            'PrintfCheckerPlugin',
        ],
    ];
```

6) Agora execute o *phan* na linha de comando:

```
vendor\bin\phan --allow-polyfill-parser
```