Docker, PHP Built-In Server, Composer, Geradores de Documentação e Testes

Docker

Docker é uma plataforma de virtualização open source que permite a criação, implantação e execução de aplicações usando containers. Os containers permitem que você embale seu software junto com todas as partes de que ele depende, incluindo bibliotecas, sistemas operacionais e outras dependências. Isso significa que você pode ter certeza de que sua aplicação funcionará exatamente da mesma maneira, independentemente do ambiente onde ela está sendo executada.

Para mais informações ou para acessar o link de download verifique o site oficial do Docker https://www.docker.com/.

Utilizando Docker no VS Code

- Instale a extenção Dev Container no VS Code
- Crie uma pasta .devcontainer contendo os arquivos Dockerfile e devcontainer.json
- No menu lateral do VS Code está o icone da extensão Dev Container onde é possíve abrir gerênciar (criar, pausar, abrir) containers e diretórios em um container. Opcionalmente, ao abrir um diretório de um projeto com a pasta .devcontainer previamente configurada, o VS Code irá sugerir que esta seja reaberta no container.
- Acesse os arquivos via terminal do VS Code

OBS.: No primeiro acesso ao container, será feito um processo de instalação e configuração deste que poderá levar alguns minutos.

OBS.: Utilize git fora do container. Apesar de ser possível utilizar no container, será necessário configurar chaves ssh, mas como o projeto está versionado e pode ser clonado em diversas estações de trabalho, não é recomendável salvar as credenciais do git.

Arquivos de configuração do Docker

- O arquivo devcontainer.json configura o container dentro do VS Code, habilita/desabilita o uso de outras extenções e o envelopamento do diretório dentro do volume virtualizado no container

```
{
    "name": "PHP",
    "build": {
        "dockerfile": "Dockerfile",
        "args": {
            "VARIANT": "8.1",
            "NODE_VERSION": "none"
        }
    },
    "customizations": {
        "vscode": {
            "settings": {
                "php.validate.executablePath": "/usr/local/bin/php"
```

```
            },
            "extensions": [
                "xdebug.php-debug",
                "bmewburn.vscode-intelephense-client",
                "mrmlnc.vscode-apache"
            ]
        }
    },
    "forwardPorts": [8080],
    "remoteUser": "vscode"
}
```

- O arquivo Dockerfile contém as especificações do container e pode ser utilizado em qualquer ambiente configurado com docker, inclusive no servidor em produção.

```
ARG VARIANT="8.1-apache-bullseye"
FROM mcr.microsoft.com/vscode/devcontainers/php:0-${VARIANT}

ARG NODE_VERSION="none"
RUN if [ "${NODE_VERSION}" != "none" ]; then su vscode -c "umask 0002 && .
/usr/local/share/nvm/nvm.sh && nvm install ${NODE_VERSION} 2>&1"; fi

RUN apt-get update && export DEBIAN_FRONTEND=noninteractive \
     && apt-get -y install --no-install-recommends wget graphviz \
     && wget https://phpdoc.org/phpDocumentor.phar \
     && chmod +x phpDocumentor.phar \
     && mv phpDocumentor.phar /usr/local/bin/phpDocumentor \
     && echo xdebug.mode=coverage >> /usr/local/etc/php/conf.d/xdebug.ini
```

OBS.: O container acima é fornecido pela Microsoft e consiste de um ambiente linux com instações de PHP e Node, além de configurações adicionais feitas através de comandos unix para instalação de dependências a nível de SO e configurações do PHP Documentor e Xdebug em modo de cobertura.

PHP Built-In Server

O PHP Built-in Server é um servidor web embutido em uma versão específica do PHP, a partir da versão 5.4. Ele permite que desenvolvedores iniciem rapidamente um servidor de desenvolvimento local, sem a necessidade de instalar e configurar outros servidores como Apache ou Nginx. O servidor embutido só deve ser usado para fins de desenvolvimento e não para ambientes de produção.

Para iniciar o servidor acesse o diretório público via terminal e digite o seguinte comando:

```
php -S localhost:8888
```

Após acesse a url no navegador.

OBS.: Utilize qualquer porta alta, não é obrigatório utilizar a 8888 e evite utilizar a 8080 pois o VS Code a utiliza para tunelamento.

Disponibilizando acesso ao Built-In Server com Ngrok

Ngrok é uma ferramenta que permite criar um servidor seguro para testar seus aplicativos e serviços hospedados localmente. Ele cria um túnel que expõe seu servidor local à Internet pública, permitindo que seja acessível de qualquer lugar, enquanto mantém seu IP e a porta local privados. Ele é usado para fins de desenvolvimento, testes, demonstrações e outras aplicações.

Cadastre-se no site https://ngrok.com/ e após ligar seu Built-In Server abra outro terminal e digite o seguinte comando:

```
ngrok http 8888
```

OBS.: O terminal ficará travado na apliação do Ngrok que fornecerá um link público para disponibilização do acesso ao seu ambiente de desenvolvimento.

Composer

O que é?

O Composer é um gerenciador de dependências para o PHP. Ele é usado para gerenciar as dependências do seu projeto, como bibliotecas, frameworks e outras ferramentas. O Composer permite que você crie seu projeto de forma fácil e rápida, mantendo o seu código sempre atualizado e acessível.

Acesso a documentação

https://getcomposer.org/doc/

Instalação

https://getcomposer.org/download/

Onde encontrar pacotes PHP instaláveis via Composer

- Site oficial para download de repositórios - https://packagist.org/
- Repositório no GitHub que mantem lista das melhores e mais utilizadas bibliotecas em várias áreas - https://gist.github.com/llbbl/7607016

Como iniciar um projeto com Composer?

```
composer init
```

Em seguida responda as perguntas e configure seu composer.json.

Você também pode criar o arquivo composer.json manualmente.

Principais comandos

- init - Inicializa um projeto com o composer
- install ou i - Instala todas as dependências
- update ou u - Atualiza todas as dependências
- clear-cache - Limpa o cache do Composer
- dumpautoload -o - Atualiza o classpath do autoload PSR-0 e PSR-4

Principais propriedades presentes no composer.json

- name: Utilizando o formato [vendor/name], especifica o nome do distribuidor e o nome do projeto
- description: Descrição do projeto
- type: Tipo do projeto podendo ser library, project ou metapackage
- version: Versão do projeto, utilizando formato numérico comumente com grandes e pequenas versões
- autoload: Especifica um classpath para carregamento de classes ou arquivos utilizando apenas o autoload do composer
- require: Lista de todas as dependências do projeto
- scripts: Lista de funções, atalhos e scripts de uso geral utilizados via composer

OBS.: O autoload e o require possuem propriedades semelhantes com o sufixo dev, onde ficam armazenadas definições válidas apenas para o ambiente de desenvolvimento. Para instalar as dependências sem carregar os recursos de desenvolvimento utilize o comando a seguir:

```
composer install --no-dev
```

Exemplo de arquivo composer.json completo

```
{
    "name": "senaces/sig-integration",
    "type": "project",
    "description": "SENAC | ES",
    "authors": [
        {
            "name": "Felipe Gaspar"
        }
    ],
    "autoload": {
        "psr-4": {
            "Senac\\Cron\\": "scripts/",
            "Senac\\Services\\": "src/integration/",
            "WebApp\\Platform\\Core\\Utils\\": "src/utils",
            "WebApp\\Platform\\Core\\Database\\": "src/database",
            "WebApp\\Platform\\OpenApi\\Senac\\": "src/api/"
        }
    },
```

```
    "autoload-dev": {
        "psr-4": {
            "Test\\UnitTest\\": "tests/unit"
        }
    },
    "require": {
        "php": "^8",
        "symfony/yaml": "^5.3",
        "guzzlehttp/guzzle": "^7.0",
        "monolog/monolog": "^3.2",
        "symfony/mailer": "^6.1",
        "symfony/mime": "^6.1",
        "phuml/phuml": "^6.1",
        "code-simplify/ciesta-project": "^0.0.1"
    },
    "require-dev": {
        "phpunit/phpunit": "^9",
        "codeception/codeception": "^5.0",
        "codeception/module-asserts": "^3.0",
        "phpmetrics/phpmetrics": "^2.8",
        "phpunit/php-code-coverage": "^9",
        "rregeer/phpunit-coverage-check": "^0.3.1"
    },
    "scripts": {
        "clear": [
            "rm -f composer.lock",
            "@composer clear-cache"
        ],
        "code-coverage": "codecept run --coverage",
        "code-coverage-details": [
            "codecept run --coverage --coverage-html",
            "mv reports/coverage docs"
        ],
        "unit-test": "codecept run unit",
        "test-application": [
            "@code-coverage"
        ],
        "build-application": [
            "@clear",
            "@composer install",
            "@test-application"
        ],
        "doc": [
            "rm -rf docs",
            "mkdir docs",
            "mkdir docs/metrics",
            "mkdir docs/code",
            "mkdir docs/class",
            "mkdir docs/coverage",
            "vendor/bin/phpmetrics --report-html=docs/metrics src",
            "phpdoc -d src/ -t docs/code",
            "vendor/bin/phuml phuml:diagram -r -a -i -o -e php -p dot src/
docs/class/diagram.png",
            "@code-coverage-details"
```

```
            ]
        },
        "scripts-descriptions": {
            "clear": "Limpa o cache de dependências e o arquivo composer.lock",
            "code-coverage": "Executa o teste de cobertura de código com saida
pelo console",
            "code-coverage-details": "Gera um relatório completo do teste de
cobertura de código em formato HTML",
            "code-coverage-test": "Verifica o sucesso da cobertura de código
com base em um parâmetro numérico especificado",
            "unit-test": "Executa os testes unitários utilizando o framework
PHPUnit",
            "test-application": "Executa todas as rotinas de testes
configuradas. OBS.: Habilite apenas as rotinas implementadas",
            "build-application": "Limpa o cache e instala as dependências, e em
seguida executa todas as rotinas de testes configuradas",
            "doc": "Gera a documentação completa do projeto"
        }
    }
```

composer.lock e vendor

O arquivo composer lock contém a referência para todas as instalações efetuadas no ambiente, incluindo a informação das versões instaladas de cada biblioteca agindo como uma referência para o composer preservar as mesmas versões especificadas.

Já o diretório vendor contém os códigos-fonte de todas as bibliotecas instaladas, o classpath do autoload e scripts binários. É a base para o carregamento de projetos que utilizam composer. Para utilizá-lo basta carregar o arquivo /vendor/autoload.php em qualquer arquivo de seu projeto.

É bem comum não guardar ambos os arquivos durante o versionamento. Para fácilitar este gerênciamento, utilize o arquivo .gitignore especificando os paths da raiz do projeto até todo e qualquer arquivo e diretório que queira manter fora do controle de versão.

Criando scripts para execução com composer

Neste exemplo trabalharemos com um script para limpeza do cache e dos arquivos do composer

1. Crie a seção script como no exemplo abaixo:

```
"scripts": {

},
```

2. Adicione a chave "clear" e atribua no formato json um array vazio
3. Adicione o comando unix "rm -f composer.lock" utilizando as aspas e ao final coloque a virgula. (O composer aceita quaisquer comandos unix e comandos php através da flag -r. Ex.: php -r "echo 'teste';"😉

4. Adicione o comando "@composer clear-cache". Para utilizar comandos composer nos seus scripts basta referenciá-los com o @ antes da palavra reservada composer
5. Teste o seu script com o comando a seguir:

```
composer clear
```

Atividade

Agora crie um comando com o nome "build-app" que limpe seu ambiente e instale todas as dependências

Criando sua primeira biblioteca com composer

1. Crie seu composer.json utilizando o composer init
2. Crie um arquivo dentro da pasta /src com o nome Conexao.php
3. Copie o código abaixo dentro deste arquivo:

```php
<?php

namespace Treinamento\Teste;

final class Conexao {
    private array $data = [];
    public function __construct() {
        $this->data = [
            (object)[
                'titulo' => 'Exemplo 01',
                'autor' => 'Fulano',
                'corpo' => 'Foo'
            ],
            (object)[
                'titulo' => 'Exemplo 02',
                'autor' => 'Beltrano',
                'corpo' => 'Bar'
            ],
            (object)[
                'titulo' => 'Exemplo 03',
                'autor' => 'Cicrano',
                'corpo' => 'Baz'
            ]
        ];
    }

    public function lerNoticiasDoBanco() : array {
        return $this->data;
    }
}
```

4. Crie o autoload PSR-4 para a classe conexão com as seguintes especificações:

```
"autoload": {
    "psr-4": {
        "Treinamento\\Teste\\": "src/"
    }
}
```

5. Crie o repositório público no seu github. (Altere o vendor name para o nome da sua conta no github)
6. Faça o upload do código para o seu github
7. Crie uma release e uma tag 1.0 para o seu projeto
8. Acesse e faça login no site https://packagist.org/
9. Publique seu repositório clicando no menu submit e fornecendo o link para o seu repositório no github
10. Com o repositório publicado no packagist você poderá inseri-lo como dependência em seu projeto. Para isso, feche a pasta atual e crie um projeto novo utilizando o .devcontainer e copie no packagist o comando de instalação da sua biblioteca.

Geradores de Documentação

PHUML

PHUML (Processing and Handling of UML Models) é uma ferramenta de modelagem baseada em UML (Unified Modeling Language) que permite aos usuários criar modelos de software e sistemas. O PHUML possibilita a visualização, análise e manipulação desses modelos, além de oferecer recursos para a geração automática de código a partir dos modelos criados.

Em seu novo projeto criado na atividade passada, acrescente a dependência de desenvolvimento abaixo:

```
composer require --dev phuml/phuml ^6.1
```

Utilize o comando a seguir para gerar o diagrama de classes.

```
vendor/bin/phuml phuml:diagram -r -a -o -e php -p dot examples/
docs/class/diagram.png
```

Você poderá trocar o diretório de origem e destino caso precise. Para mais informações sobre a configuração do PHUML acesse https://montealegreluis.com/phuml/docs/class-diagram.html. É possível gerar versões da documentação com apenas métodos, ou apenas atributos, com ou sem associações e escolher a ferramenta de desenho entre as disponíveis, dot e neato. O comando sugerido acima já traz a versão mais completa do gerador do phuml utilizando o graphviz dot que já está instalado no .devcontainer.

Atividade

Acrescente uma nova classe para categoria nas notícias e atualize o diagrama de classes

PHP Documentor

O PHP Documentor é uma ferramenta de documentação de código aberto que gera documentação estruturada e legível para códigos PHP. O PHP Documentor é usado para aumentar a produtividade e facilitar a manutenção, pois ajuda a criar documentos ricos em conteúdo, estruturados e atualizados. O PHP Documentor permite aos desenvolvedores criar documentos HTML, PDF, DocBook e man pages a partir do código-fonte.

Para gerar a documentação basta utilizar o comando abaixo:

```
phpDocumentor -d examples -t docs
```

Repare que a documentação está incompleta e dispondo apenas das informações básicas das classes e arquivos.

Utiliza-se docblocks, que são um conjunto de propriedades que agregam na definição de arquivos, classes, funções e variáveis, para melhorar a qualidade da documentação gerada, não somente pelos arquivos gerados mas pela explicitação em código de comentários.

DocBlocks

1. Arquivo

É comum documentar arquivos especificando o pacote (namespace) e a finalidade deste.

```
/**
 * Breve descrição sobre o seu arquivo
 * @package Exemplo\Treinamento
 * @author Felipe Gaspar <felipesouzalimagaspar@gmail.com>
 */
```

2. Classe ou interface

Documenta-se a classe com uma breve descrição acompanhada das docblocks @final ou @abstract caso estas sejam abstrações ou classes finais

```
/**
 * Classe de exemplo Foo que não serve para nada
 * @abstract
 */
abstract class Foo {
```

3. Funções

Na documentação de funções, é comum utilizar o @access para definir o nível de privacidade da função, sua lista de parâmetros através do uso do @param, o retorno @return e caso ela possa disparar exceptions

previsiveis, utiliza-se o @throws

```
/**
 * Função de demonstração de docblocks
 * @access public
 * @param string $foo Parâmetro de exemplo
 * @param array $bar Outro parâmetro
 * @throws TypeError Minha função pode disparar exceptions da classe
TypeError
 * @return int Minha função sempre retornará um inteiro
 */
public function minhaFuncao(string $foo, array $bar) : int {
```

4. Atributos

No caso de atributos, utiliza-se a docblock @var acompanhada do tipo e da descrição

```
/**
 * @access private
 * @var bool Minha váriavel que não serve para nada além de demonstração
 */
private bool $exemplo;
```

Caso você tenha uma variável, parâmetro ou retorno que pode ser de múltiplos tipos ou você não sabe o tipo, utilize o mixed. O mixed é o tipo básico do PHP que desde as primeiras versões permite que uma váriavel mude de tipo.

Acesse o site do PHP Documentor para ver a lista completa de docblocks disponíveis e como utilizá-los.

https://docs.phpdoc.org/3.0/guide/guides/docblocks.html#more-on-docblocks

Atividade

Agora que você já conhece as docblocks e o PHP Documentor, acrescente as docblocks nas classes do diretório exemplos e gere novamente a documentação

Atividade

Utilizando os conhecimentos em scripts do composer e o PHP Documentor e PHUML, crie um script "composer doc" que crie uma pasta docs sem conteúdo e gere a documentação utilizando as duas ferramentas estudadas.

Tipos de testes

Testes unitários

Testes unitários são testes que validam unidades individuais de código. Eles são projetados para verificar se uma unidade específica de código está funcionando corretamente. Esses testes são executados em uma

única unidade de código ao mesmo tempo, o que significa que eles são executados isoladamente do resto do código.

Testes de integração

Testes de integração são usados para verificar se diferentes unidades de código interagem corretamente. Estes testes são usados para verificar se as unidades de código trabalham bem juntas e se eles são capazes de produzir resultados esperados.

Testes de aceitação

Testes de aceitação são usados para verificar se um sistema satisfaz os requisitos de um usuário. Esses testes são usados para verificar se o sistema está funcionando como o usuário espera.

Cobertura de código

Cobertura de código é uma métrica usada para medir a quantidade de código que foi testado. A cobertura de código é usada para verificar se todas as partes do código foram testadas adequadamente.

Asserções

Asserções em testes são declarações que afirmam se um resultado obtido em um teste é ou não o esperado. Elas são usadas para verificar se o código está executando corretamente e produzindo os resultados esperados. Asserções são geralmente representadas por frases como "O resultado deve ser igual a ..." ou "O resultado deve estar entre ...".

PHPUnit

PHPUnit é um framework de testes de unidade para a linguagem de programação PHP. Ele fornece ferramentas para testar aplicativos PHP para garantir sua qualidade e confiabilidade. O PHPUnit permite que os desenvolvedores escrevam e execute testes unitários automatizados para aplicativos e bibliotecas de código PHP. O PHPUnit é construído com base no xUnit e cumpre os princípios do desenvolvimento ágil. É o mais popular framework de testes de unidade para PHP.

Para instalar o PHP Unit basta acrescentar no seu composer.json a seguinte dependenca

```
composer require --dev phpunit/phpunit ^9
```

Configurando a suíte de testes

A suíte de testes consiste de uma coletânea de testes unitários executados em um ambiente com configurações pré-definidas.

1. Crie um arquivo na raiz do projeto com o nome de phpunix.xml e acrescente as linhas a seguir:

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:noNamespaceSchemaLocation="./src/vendor/phpunit/phpunit/phpunit.xsd"
         bootstrap="./tests/bootstrap.php"
         colors="true"
>
<testsuites>
    <testsuite name="Unit Test Suite">
        <directory> ./tests/unit/</directory>
    </testsuite>
</testsuites>
</phpunit>
```

2. Crie um diretório tests e nele um arquivo bootstrap.php com o seguinte código:

```
<?php
    require_once __DIR__ . '/../vendor/autoload.php';
    /**
    * @todo acrescente aqui outras definições base para execução dos testes
    */
```

3. Acrescente em seu composer.json a seguinte seção:

```
"autoload-dev": {
    "psr-4": {
        "Test\\UnitTest\\": "tests/unit"
    }
},
```

4. Crie dentro da pasta tests o diretório unit e nele começe a criar seus testes.

Criando testes unitários com o PHPUnit

Neste exemplo, trabalharemos com operações matemáticas básicas para conhecer os recursos do PHPUnit

Todos os testes feitos com PHPUnit devem ter o nome do arquivo com o sufixo Test e o nome dos métodos de teste com o prefixo test

1. Crie um arquivo com o nome MathTest.php e acrescente nele o código a seguir:

```
<?php
    namespace Test\UnitTest;

    final class MathTest extends \PHPUnit\Framework\TestCase {
        public function testTaFuncionandoOPHPUnit(): void {
            $this->assertTrue(true);
        }
    }
```

2. Crie seu primeiro teste. Para isso, acrescente uma função com o nome testSoma. Teste se a soma 2 + 2 é igual a 4. Para isso, utilize a asserção assetEquals que recebe dois parâmetros e os compara.

3. Vamos alterar o teste criado anteriormente para torná-lo dinâmico. Para isso, troque os dois números 2 e o número quatro por váriaves $n1, $n2 e $n3 nos parâmetros da função.

Antes de rodar o teste, crie uma função com o nome de NumerosProvider com o seguinte código:

```
public function NumerosProvider() : array {
    return [
        '2+2' => [2,2,4],
        '3+3' => [3,3,6],
        '2+3' => [2,3,5]
    ];
}
```

Acrescente antes da função testSoma o docblock a seguir:

```
/**
 * @dataProvider NumerosProvider
 */
```

DataProviders são um conjunto de instâncias de teste que podem ser utilizados para ampliar a precisão de seus testes informando múltiplos parâmetros e respostas esperadas.

4. Acrescente em sua classe as funções a seguir:

```
public static function setUpBeforeClass(): void {
    echo "Imagine que isso seja muito importante e tenha de ser executado
antes dos testes";
}
public static function tearDownAfterClass(): void {
    echo "Imagine que isso seja muito importante e tenha de ser executado
após os testes";
}
```

Atividade

Pesquise e implemente testes utilizando algumas das asserções disponíveis na doc oficial do PHPUnit https://docs.phpunit.de/en/9.Para isso, utilize operações básicas do PHP.

Atividade

Carregue as classes do diretório examples com o autoload do composer e faça testes unitários para cada classe.

Atividade

Carregue a biblioteca que você criou e faça testes unitários para a classe Conexão.

Atividade

Implemente um teste de integração, utilizando as classes Conexão, Autor e Notícia.

Codeception

Codeception é um framework de teste automatizado para aplicativos web PHP. Ele provê uma série de ferramentas para ajudar a testar aplicativos web, desde testes unitários até testes de aceitação do usuário. Ele também possui um conjunto de bibliotecas e recursos para ajudar a automatizar a execução de testes.

Criando testes de cobertura

1. Acrescente em seu composer.json as segunintes dependências:

```
"codeception/codeception": "^5.0",
"codeception/module-asserts": "^3.0",
"phpunit/php-code-coverage": "^9"
```

2. Crie um arquivo na raíz do seu projeto com o nome codeception.yml e cole o código a seguir:

```
paths:
    tests: tests
    output: reports
    data: reports
    support: tests/.codeception
    envs: tests/_envs
bootstrap: bootstrap.php
settings:
    suite_class: \PHPUnit_Framework_TestSuite
    memory_limit: 1024M
    log: true
    colors: true
actor_suffix: Tester
extensions:
    enabled:
        - Codeception\Extension\RunFailed
coverage:
    enabled: true
    include:
        - examples/*
    exclude:
        - examples/*log
    low_limit: 50
    high_limit: 90
    show_only_summary: false
    show_uncovered: true
```

3. Extraia o arquivo codeception.zip para dentro da pasta tests
4. Execute o comando a seguir para rodar o teste de cobertura:

```
vendor/bin/codecept run --coverage
```

Caso prefira gerar o relatório em modo gráfico, utilize o comando:

```
vendor/bin/codecept run --coverage --coverage-html
```

Criando testes de aceitação

1. Utilizando o PHP Built-In Server, disponibilize na porta 8888 o conteúdo da pasta view.
2. Crie uma pasta dentro de /tests com o nome acceptance
3. Instale a dependência a seguir:

```
"codeception/module-phpbrowser": "^3.0"
```

4. Copie o código a seguir dentro do arquivo FirstCest.php, que você deverá criar na pasta acceptance

```php
<?php
class FirstCest
{
    public function frontpageWorks(AcceptanceTester $I)
    {
        $I->amOnPage('/');
        $I->see('Exemplo');
    }

    public function viewpageWorks(AcceptanceTester $I)
    {
        $I->amOnPage('/view.php?url=exemplo-01');
        $I->see('Foo');
    }
}
```

Atividade

Implemente um teste para formulários utilizando um formulário online qualquer e os métodos da documentação disponíveis em https://codeception.com/docs/AcceptanceTests