## SUMÁRIO

[SUMÁRIO 0](#_Toc487106277)

[SOBRE TESTES AUTOMATIZADOS 1](#_Toc487106278)

[Testes END-TO-END (e2e) 1](#_Toc487106279)

[TECNOLOGIAS UTILIZADAS 2](#_Toc487106280)

[CONFIGURANDO O AMBIENTE (Windows) 3](#_Toc487106281)

[Proxy: 4](#_Toc487106282)

[Node 4](#_Toc487106283)

[Git 5](#_Toc487106284)

[Java 6](#_Toc487106285)

[Protractor 6](#_Toc487106286)

[BAIXANDO O PACOTE NO GIT 7](#_Toc487106287)

[Manter o pacote atualizado 8](#_Toc487106288)

[ARQUITETURA DOS TESTES 8](#_Toc487106289)

[EXECUTANDO OS TESTES 10](#_Toc487106290)

[RELATÓRIOS DOS TESTES 11](#_Toc487106291)

[CONFIGURAÇÕES AVANÇADAS 12](#_Toc487106292)

[Versionamento e execução 13](#_Toc487106293)

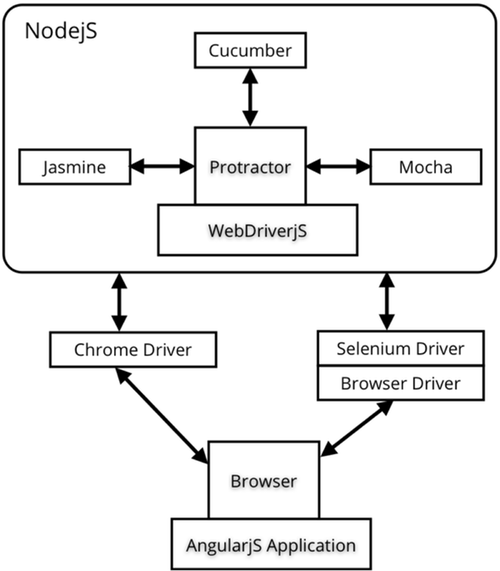
[Configurações dos testes 14](#_Toc487106294)

## SOBRE TESTES AUTOMATIZADOS

### **Testes END-TO-END (e2e)**

O teste de ponta a ponta (end-to-end) é uma metodologia utilizada para testar se o fluxo de um aplicativo está executando conforme especificado do início ao fim. O objetivo da realização de testes de ponta a ponta é identificar dependências do sistema e garantir que a informação correta é passada entre vários componentes e sistemas do sistema.

O teste end-to-end envolve garantir que os componentes integrados de um aplicativo funcionem como esperado. Todo o sistema é testado em um cenário real, como a comunicação com o banco de dados, rede, hardware e outros aplicativos.   
  
À medida que as aplicações crescem em tamanho e complexidade, torna-se impossível confiar em testes manuais para verificar a correção de novos recursos, capturar bugs e observar regressões. Os testes unitários são a primeira linha de defesa para a captura de bugs, mas às vezes problemas surgem com a integração entre os componentes que não podem ser capturados em um teste de unidade.   
Testes end-to-end são feitos para encontrar esses problemas.



## TECNOLOGIAS UTILIZADAS



O **Protractor** é uma estrutura de teste de ponta a ponta (end-to-end ou e2e) para aplicações **Angular** e **AngularJS** (Framework de desenvolvimento para Desktop e Mobile)

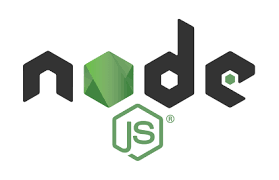


Protractor executa testes em seus aplicativos em execução em um navegador real, interagindo com ele como um usuário faria.

Visite Protractor: <http://www.protractortest.org/#/>  
Visite Angular: <https://angular.io/>



**Jasmine** é uma estrutura de teste de desenvolvimento orientada a comportamento para JavaScript. Ele não depende de navegadores, DOM ou qualquer framework JavaScript. Assim, é adequado para sites, projetos [Node.js](http://nodejs.org/) , ou em qualquer projeto com JavaScript.  
  
Visite Jasmine: [http://jasmine.github.io](http://jasmine.github.io/)  ou <http://jasmine.github.io/2.0/introduction.html>



**Node** é uma linguagem baseada no  [V8 JavaScript Engine](https://code.google.com/p/v8/), orientada a eventos e feita para ser facilmente escalável, é a tecnologia em que o Protractor foi escrito.  
Visite: https://nodejs.org/



**Selenium Webdriver:** Basicamente automatiza processos em navegadores. É ele quem obedece o Jasmine e clica, preenche e interage com a sua aplicação como se fosse um usuário.

Visite: http://www.seleniumhq.org/

## CONFIGURANDO O AMBIENTE (Windows)

### Proxy:

Antes de começar, verifique as configurações de proxy com o administrador de sua rede:

Setando o Proxy para NPM

Crie um arquivo **set-proxy.bat**Basta abrir um arquivo de texto, copiar as instruções abaixo e salvar com a extensão **.bat**Antes de salvar insira suas credenciais substituindo “usuario” e “senha”:

@ECHO OFF

ECHO Setando proxy para o NPM...

CALL npm config set proxy http://usuario:senha@proxy-br.lanet.accorservices.net:8080

CALL npm config set https-proxy http://usuario:senha@proxy-br.lanet.accorservices.net:8080

CALL npm set strict-ssl false

#### Removendo o Proxy

Após baixar as dependências, rode o arquivo **remove-proxy.bat** pra remover as configurações de proxy.

Crie o arquivo **remove-proxy.bat** com as seguintes instruções:

@ECHO OFF

ECHO Removendo proxy do NPM...

CALL npm config rm proxy

CALL npm config rm https-proxy

CALL npm config rm strict-ssl

ECHO Removendo proxy do GIT...

CALL git config --global --unset http.proxy

CALL git config --global --unset https.proxy

CALL git config --global --unset http.sslverify

**Nota Nota:**  Caso você não seja administrador de sua máquina solicite a instalação para o Suporte técnico para os pacotes abaixo:

### Node

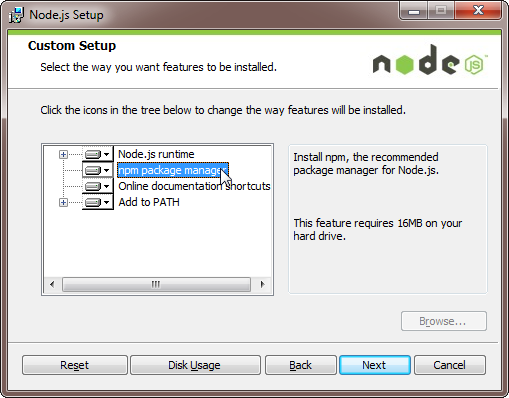
Para verificar a versão do Node.js instalada em seu computador, basta executar no terminal CMD o seguinte comando:

node –version

Deve aparecer algo como:  

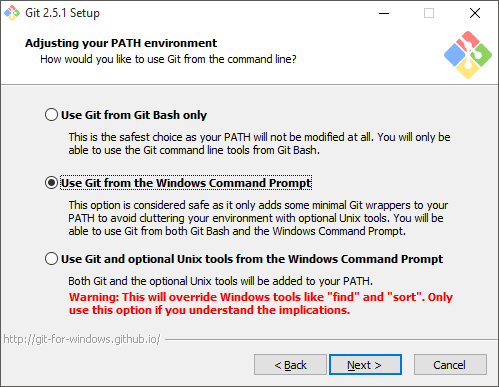

Caso o Node.js não esteja instalado, basta baixá-lo a partir do site oficial do Node: https://nodejs.org/en/.

Na instalação, deixe marcado a opção “NPM PACKAGE MANAGER”:



### Git

Usamos o GIT como repositório de código. Baixe o Git a partir do   
site oficial : <https://git-scm.com/>  
Instale, marcando a opção “USE GIT FROM THE WINDOWS COMMAND PROMPT”.



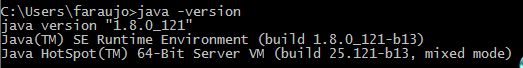
Caso já esteja habituado com GIT, ignore as configurações padrões.

### Java

Instalar a versão recente do JDK;

Para verificar a versão do Java está instalada e configurada, basta executar no terminal CMD o seguinte comando:

java -version

Deve aparecer algo como:  


Caso não apareça, baixe a versão atual a partir do site: <http://www.oracle.com/technetwork/pt/java/javase/downloads/>

### Protractor

Para instalar o Protractor, basta executar o seguinte comando no

terminal:

npm install -g protractor

Este comando instalará o Protractor globalmente em seu

computador. Após a instalação, você pode verificar a versão do

Protractor instalada executando o seguinte comando:

protractor --version

Deve aparecer algo como:  


Após a instalação do Protractor, você precisará atualizar o webdriver-manager .   
Utilize o seguinte comando:

webdriver-manager update

Você também pode instalar o Protractor e o webdrivermanager como dependências de desenvolvimento do seu projeto em vez de instalá-los globalmente.   
Para isso, basta executar os seguintes comandos no diretório do seu projeto:  
npm install protractor --save-dev

npm install webdriver-manager --save-dev

## BAIXANDO O PACOTE NO GIT

endereço do git

Com o Git já instalado abra o terminal ou o CMD;

Crie um diretório no **C:** chamado “automacao” (nome que quiser)  
mkdir automacao

Entre no diretório   
cd automacao

Dentro do diretório **repos**, digite o seguinte comando para clonar o projeto dentro de seu computador:

git init

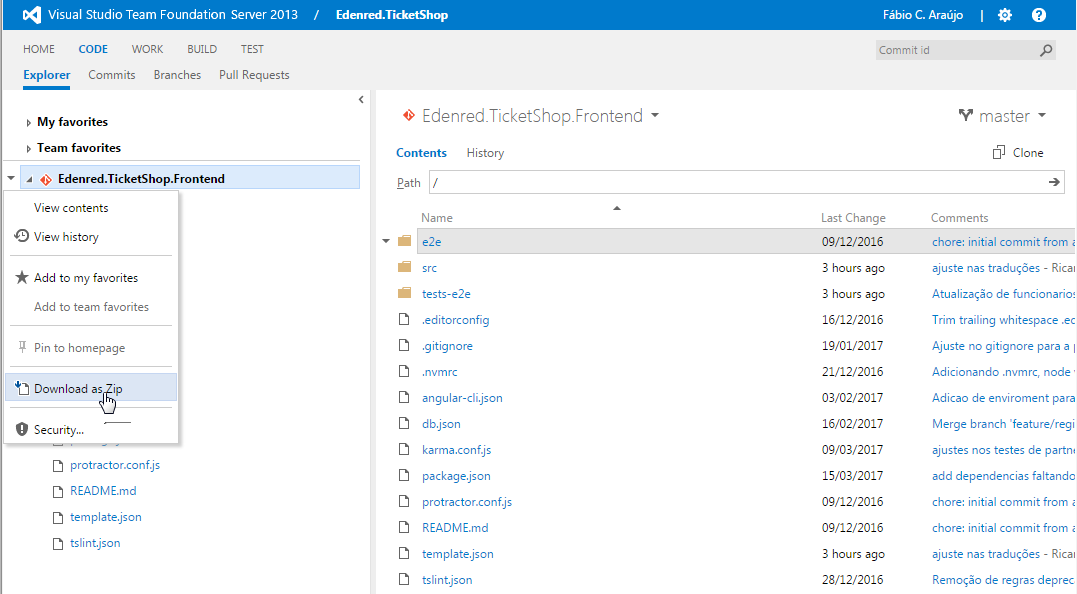
git clone “endereço do git”

Deve criar uma pasta chamada: C:\automacao\”nome do projeto no git”

Dentro deste diretório, entre no tests-e2e: **C:\**automacao**\nomeDoProjeto\tests-e2e**

Este diretório vai ser o padrão para rodar os testes:

Toda estrutura de Testes automatizados estão dentro deste diretório:

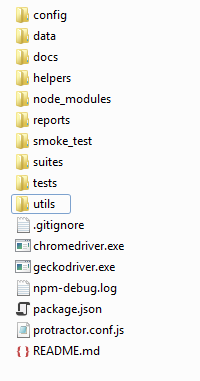
**Nota Nota:** A estrutura de diretório também pode ser criada manualmente. O pacote pode ser baixado direto do TFS, como mostrado na figura a seguir:  


### Manter o pacote atualizado

No terminal CMD, navegue até o diretório do teste:  
cd C:\automacao\nomeDoProjeto  
  
e digite o seguinte comando GIT:  
git pull origin master

## ARQUITETURA DOS TESTES

Após baixar o pacote, deve ser criado uma estrutura parecida com esta:



**config:** contem arquivos de configurações do navegador;  
**data**: arquivos de dados como CPF, CNPJ, Nasc. CEP, etc;  
**docs**: conteúdos de ajuda;  
**helpers**: implementação de ajuda para evitar duplicidade de código e facilitar a manutenção, além de ajudarem na legibilidade dos testes;  
**node modules**: arquivos baixado do NPM (Este diretório só é criado após a execução do comando “webdriver manager update” descrito no próximo capítulo);  
**reports**: arquivos de Relatórios gerados após os testes;  
**smoke\_test**: arquivos do Prétest: Specs e PO, do Smoke Test que rodam antes de todos os testes;  
**tests**: diretórios onde estão armazenados as sítes de testes juntamente com os PO (Page Objects) e Specs (especificações de testes);  
**utils**: alguns arquivos de configurações uteis para o projeto;  
**.gitignore**: arquivo de configuração do git , contem arquivos ignorados pelo git commit;  
**chromedriver.exe**: driver do navegador Chrome, importante para abrir o navegador;  
**geckodriver.exe**: driver do navegador I.E, importante para abrir o navegador;  
**package.json**: arquivo de configuração do pacote, define a versão dos teste versões dos pacotes de instalação;  
**protractor.conf.js**: Arquivos de configurações do Protractor. Define todas as configurações do testes como navegador, timeout, console, etc.  
**README.md:** arquivo de ajuda rápido, estão alguns comandos necessários para rodar o arquivo. Arquivo apenas de texto, não interfere nas configurações dos testes.

## EXECUTANDO OS TESTES

1 - Abra o terminal CMD e navegue até o diretório dos testes:

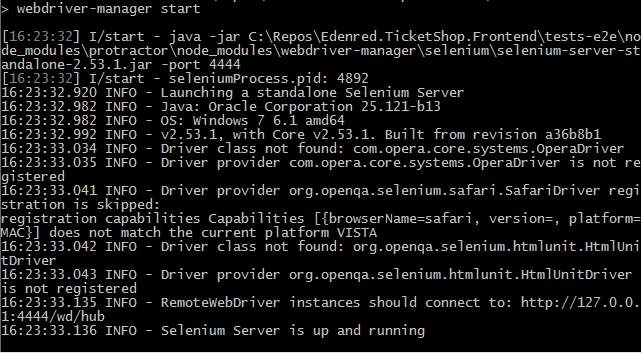
cd C:\automacao\nomeDoProjeto\tests-e2e

2 – Digite o seguinte comando para instalar as dependências ( verifique s as configurações do proxy )   
npm install

No diretório dos testes deve criar uma pasta chamada **node\_modules**

3 – Após a instalação, se não houver nenhum erro, digite o próximo comando no mesmo terminal , para iniciar o driver do navegador.   
npm start

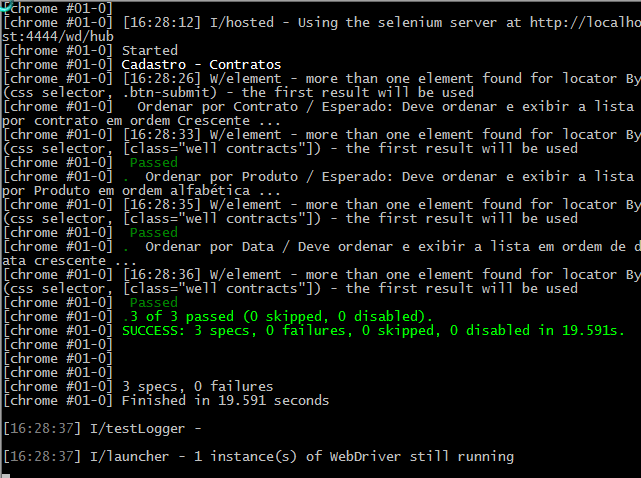
Este processo vai ficar rodando em background:

  
  
4 – Abra **outra janela do terminal**, no mesmo diretório de teste (Não feche a primeira janela):  
cd C:\automacao\nomeDoProjeto\tests-e2e

5- Para rodar os testes, digite o seguinte comando:  
npm test

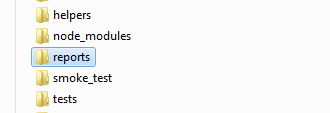
Neste momento inicia os testes end-to-end , abrindo o navegador. Agora é só aguardar até o término.

Em paralelo na segunda janela do terminal, exibe o log e status do testes, mostrando quais falharam e quais passaram:



## RELATÓRIOS DOS TESTES

Na pasta **report** , estão os relatórios de todos os testes executados em seu computador, ordenado por data e hora:



Abra o arquivo “report\_e2e.html” em seu navegador.

Deve abrir um relatório com os detalhes da última rodada de teste automatizados (end-to-end), com o link do print de cada cenário de teste.

Os testes falhados, marcados com X, tem o log do navegador informando o motivo da falha.

Os testes passados estão marcados com um V, apenas com o link do *print* e o tempo de cada teste em parênteses.

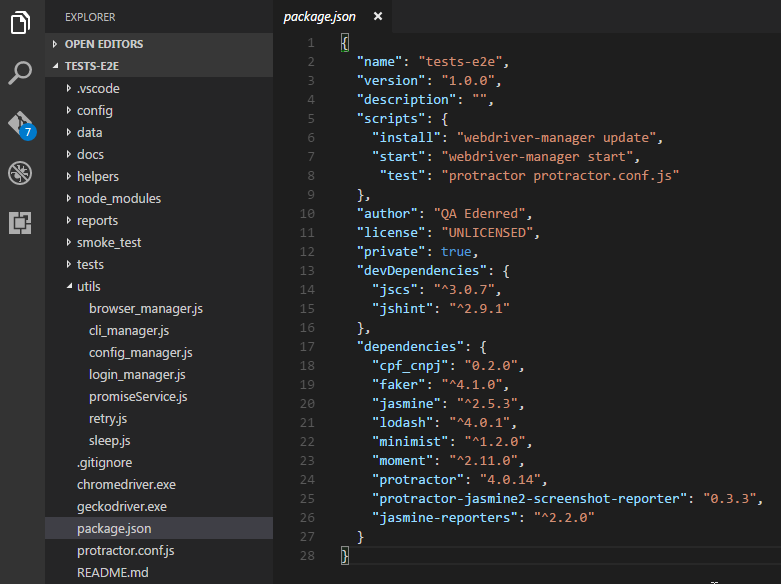


## CONFIGURAÇÕES AVANÇADAS

Segue algumas configurações avançadas contidas nos arquivos de configurações do pacote de automação:

### Versionamento e execução

Dentro do arquivo Package.json, contem a versão das dependências e os comandos de execução do console:



Segue o detalhamento de cada bloco:  
As primeiras 3 linhas, tem os detalhes do pacotes, tais como Nome, versão e descrição:

{  
 "name": "tests-e2e",  
 "version": "1.0.0",  
 "description": "Teste de regressão automatizados do Ticket Shop",  
  
Já no bloco “Script” contem os comandos do console CMD, abstraído em comandos com nomes curtos:

**-install**: Instala as dependências descritas no arquivo:  
Comando: $ npm install  
**-start:** Inicia o driver do Selenium.   
Comando: $ npm start  
**-prestest:** Este comando executa um pré teste antes de iniciar a bateria de testes regressivos. Neste pacote definimos a suíte “Smoke test” para rodar antes. Caso algum cenários do Smoke test falhe, o processo é abortado. Caso não queira o Smoke teste, oculte esta linha do arquivo.  
**-test:** Executa a as suítes de testes regressivos.  
Comando: $ npm test  
"scripts": {  
 "install": "webdriver-manager update",  
 "start": "webdriver-manager start",  
 "pretest": "protractor protractor.conf.js --suite smoke",  
 "test": "protractor protractor.conf.js"  
 },

Este bloco define as dependências do pacote e suas versões.

"author": "QA Edenred",

"license": "UNLICENSED",

"private": true,

"devDependencies": {

"jscs": "^3.0.7",

"jshint": "^2.9.1"

},

"dependencies": {

"cpf\_cnpj": "0.2.0",

"faker": "^4.1.0",

"jasmine": "^2.5.3",

"lodash": "^4.0.1",

"minimist": "^1.2.0",

"moment": "^2.11.0",

"protractor": "4.0.14",

"protractor-jasmine2-screenshot-reporter": "0.3.3",

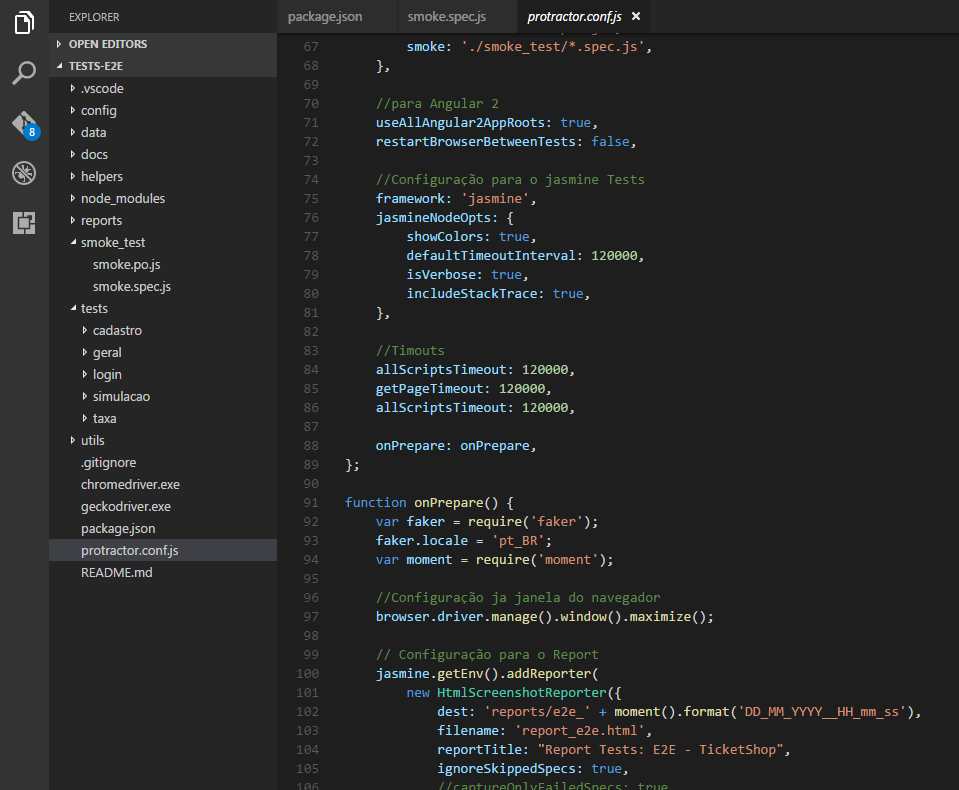
"jasmine-reporters": "^2.2.0"

}

}

### Configurações dos testes

O arquivo “protractor.conf.js” define as configurações dos testes:



Segue o detalhamento de cada bloco:

As primeiras linhas estão as dependências de arquivos externos ou bibliotecas:  
var HtmlScreenshotReporter = require('protractor-jasmine2-screenshot-reporter');

var jasmineReporters = require('jasmine-reporters');

var suites = require('./suites/suites');

**-multiCapabilities:** define quais navegadores e quais configurações serão usadas para cada navegador:

**-browserName:** Define qual o navegador a ser usado.

-**shardTestFiles** : Roda um conjunto de testes em paralelo. O default é false.

**-maxInstances:** Quantas janelas (instancias) do navegador executa ao mesmo tempo.

exports.config = {

multiCapabilities: [

{

'browserName': 'chrome',

shardTestFiles: 'true', maxInstances: 1   
 },

{

'browserName': 'firefox'

},

{

'browserName': 'phantomjs'

},

{

'browserName': 'ie'

},

{

],

**Maximizar o browser:**

browser.driver.manage().window().maximize();

**-seleniumAddress:** Define o endereço e porta do selenium

seleniumAddress: 'http://localhost:4444/wd/hub',

**-specs:** Define os “specs” a serem testados

specs: [

'./tests/\*\*/\*.spec.js',

'./smoke\_test/\*.spec.js'

],

**-suites:** Define quais suítes de testes serão testadas

suites: {

all: './tests/\*\*/\*.spec.js',

smoke: './smoke\_test/\*.spec.js',

},

**-Angular 2:** Necessário para rodar bem com Angular 2.0  
 useAllAngular2AppRoots: true,

restartBrowserBetweenTests: false,

**-Console Jasmine:** configurações para exibição do console CMD em tempo real:

framework: 'jasmine',

jasmineNodeOpts: {

showColors: true,

defaultTimeoutInterval: 120000,

isVerbose: true,

includeStackTrace: true,

},

**-Timouts:** Define o tempo máximo de cada teste. Se atingido o tempo limite, o teste falha e passa para o próximo teste:

allScriptsTimeout: 120000,

getPageTimeout: 120000,

allScriptsTimeout: 120000,

onPrepare: onPrepare,

};

-Configurações da biiblioteca faker e moment

function onPrepare() {

var faker = require('faker');

faker.locale = 'pt\_BR';

var moment = require('moment');

**-Jasmine Report:** Configurações para o Relatório de teste:

jasmine.getEnv().addReporter(

new HtmlScreenshotReporter({

dest: 'reports/e2e\_' + moment().format('DD\_MM\_YYYY\_\_HH\_mm\_ss'),

filename: 'report\_e2e.html',

reportTitle: "Report Tests: E2E - TicketShop",

ignoreSkippedSpecs: true,

//captureOnlyFailedSpecs: true

showSummary: true,

showQuickLinks: true,

showConfiguration: true

})

);

**-Jasmine report ( terminal):** Configurações para o log do terminal

jasmine.getEnv().addReporter(

new jasmineReporters.TerminalReporter({

color: true,

verbosity: 3,

showStack: false,

}));

}