**PROTRACTOR DEPLOY**

1. In the project folder run initiation script: npm init. It creates environment and fundament for NodeJS testing project.

*npm init*

1. After that, we need to install all necessary libraries, such as mocha and chai. We need to run two scripts:

*npm i mocha*

*npm i chai*

1. Next step is to install protractor. From the memories of the deep past, we must avoid to install protractor globally, so we have to use such script as:

*Npm I protractor*

1. Then we need to create two folders in project root. One folder will take name “**config**”, the other one for storage tests there – “**tests**”.
2. In **CONFIG** folder we create file **conf.js**. There will be a configuration for keeping the settings for using protractor. In this file we put such information as:

|  |
| --- |
| exports.config = {      directConnect: true,        capabilities: {        'browserName': 'chrome',      },        framework: 'mocha',        specs: ['../tests/\*.js'],        mochaOpts: {        timeout: 120000      }    }; |

1. In **package.json** file we also need to specify scripts part, where we setting an options what file will be executed if we run “test” command. As well, we also can realize there any command we like or need, for example **Npm run <something>.** For that, we need to add or to change “scripts” part of the **package.json** file:

|  |
| --- |
| "scripts": {  "postinstall": "webdriver-manager update --versions.standalone=3.14.0 --gecko=false",  "test": "protractor config/conf.js"  }, |

So, if we run command – ***npm run test***, protractor runs his configuration file **conf.js** which stored into **CONFIG** folder.

1. In configuration file **conf.js** we also need to point protractor where and how it can found and run tests. By some unknown reason, tests in protractor community called – **specs**. So this part in this file:

*specs: ['../tests/\*.js']* means that tests are placed in folder TESTS and it can be any file with JS extension.

1. Now we need to create initial smoke test. In folder TESTS create a file with name, for example, **initiateTest.spec.js.** Put there this code:

|  |
| --- |
| const {expect} = require("chai");  describe('opens the main landing page of the resource', function () {  beforeEach(()=>{  browser.waitForAngularEnabled(false);  browser.driver.manage().window().setSize(1680, 840);  });  it('lands to exchange page', async function () {  browser.get('https://us.tamrieltradecentre.com/pc/Trade');  expect(true).to.be.equal(true);  });  afterEach(async ()=>{  return browser.driver.manage().deleteAllCookies();  });  }); |

1. After that, run this script in console:

*node node\_modules/protractor/bin/webdriver-manager update*

1. After that, you can run your test scenario using this nice command:

*Npm run test*

Concluding that chain of running, we can see that starting process happens by this vector:

PACKAGE.JSON -> CONF.JS -> CUSTOM TEST PARAMETERS PATH

So, first environment puts a **package.json** file and when we running our command *npm run test*, it looks for script what has being linked for test command. There we can see a path to protractor configuration file **conf.js**, and please NB – protractor was not installed globally. In conf.js file we can see what files protractor will run as test ones – it assigned in the ***specs: ['../tests/\*.js'],*** string, it says – launch all files with JS extraction in folder *tests*.

**RUNNIN WITH PARAMETERS**

For example, we have two versions of the one server – one, for example, deployed on NA zone, and another one – in EU zone. We need to run our vector from time to time on the different versions of it. For that case, there are some parameters can be used.

There are kind of variables in NodeJS what called – environment variables. It can be raised, for example, this way – let the name of this variable will be KIM\_KILO.

This variable can be used such as: first, place it to those point where it needs to be called, for example:

*browser.get(`https://.${process.env.****KIM\_KILO****}.tamrieltradecentre.com/pc/Trade`);*

Then, we need to pass there some parameters. We will do it by passing it into command line or into terminal command line. But! If we using protractor, we will call it directly also as well using *npx* instead of *npm*.

*KIM\_KILO=eu npx protractor config/conf.js*

So in this case **KIM\_KILO** variable will receive “eu” value, and then protractor will run via it config file **conf.js**

**SETTING CHROME OPTIONS**

chromeOptions: {

args: [

// disable chrome's wakiness

'--disable-infobars',

'--window-size=1800,900',

'--disable-extensions',

'verbose',

'log-path=/tmp/chromedriver.log',

'--disable-web-security',

'--allow-running-insecure-content',

"--headless",

'--allow-cross-origin-auth-prompt',

'--no-sandbox',

],

prefs: {

// disable chrome's annoying password manager

'profile.password\_manager\_enabled': false,

credentials\_enable\_service: false,

password\_manager\_enabled: false,

},

},

},

So to better side, all capabilities and options will look like this:

capabilities: {

       browserName: 'chrome',

       acceptInsecureCerts: true,

       shardTestFiles: false,

       chromeOptions: {

      args: [

        // disable chrome's wakiness

        '--disable-infobars',

        '--window-size=1800,900',

        '--disable-extensions',

        'verbose',

        'log-path=/tmp/chromedriver.log',

        '--disable-web-security',

        '--allow-running-insecure-content',

        "--headless",

        '--allow-cross-origin-auth-prompt',

        '--no-sandbox',

      ],

      prefs: {

        // disable chrome's annoying password manager

        'profile.password\_manager\_enabled': false,

        credentials\_enable\_service: false,

        password\_manager\_enabled: false,

      },

    },

    },

DEPLOYING PROJECT ON JENKINS

To deploy our project in Jenkins, you need to create some very important steps. Deploying job on NodeJS is a bit different then making the same process in Java, so you need to know some details, without that it wouldn’t run successfully. So, our first step will be – create a Jenkins job – pipeline type.

1. Create pipeline
2. Create choice parameter – for our example, we can pass there two values. For that we need to put a flag into checkbox – This project is parameterized

Choose a name – for example, SERVERS\_LOCATION\_JEN. It will take one of two parameters – EU or US.

1. Then we need to use a Pipeline settings. We need to use a pipeline because before the launch main job it is necessary to install an environment.

* So, in *Definition* parameter we should choose – Pipeline script from SCM

Then, enter your repository URL from github.

* And, in the Script path put a name of your Jenkins file – jenkinsfile

1. Then, we need to create a script for jenkinsfile. It will looks like:

|  |
| --- |
| pipeline{  agent any    options{  disableConcurrentBuilds()  }  stages{  stage("INSTALL NPM AND DEPLOY ENVIRONMENT"){  steps{  echo "========executing NPM INSTALL COMMAND========"  bat "npm install"  echo "========executing WEBDRIVER MANAGER UPDATE COMMAND========"  bat "npx webdriver-manager update --versions.chrome 76.0.3809.68"    }  }  stage("EXECUTING APPLICATION"){  steps{  echo "======== LAUNCHING TEST ========"  bat "set SERVERS\_LOCATION = %SERVERS\_LOCATION\_JEN% npx protractor config/conf.js"  }  }  }  } |

Two important moments :

1. Use BAT command instead of SH (if you are working on Windows)
2. Use SET command to assign a value for variable SERVERS\_LOCATION (it will be environment variable in nodeJS code, looks like *browser.get(`http://${process.env.SERVERS\_LOCATION}.exchange.com*`); )

**ADDING ES LINT (traversy media)**

Install ESLint plugin and Prettier plugin into VisualCode.

Run ***npm i -D eslint prettier eslint-plugin-prettier eslint-config-prettier eslint-plugin-node eslint-config-node***

Installing air-bnb-style guide: ***npx install-peerdeps --dev eslint-config-airbnb***

Install rules for prettier by creating file - .prettierrc

It’s a json config file where prettier settings can be stored.

More options can be taken from here - *https://prettier.io/docs/en/options.html*

Installing ESLint: ***npm I eslint***

Generate ESLint config file by: ***eslint –init***

After questions have been answered, a eslint config file will appears.

Then we need to correct a config dile by deleting unnecessary info and add our prettier plugin – it should look like this:

|  |
| --- |
| {    "extends": ["airbnb", "prettier", "plugin:node/recommended"],    "plugins": ["prettier"],    "rules": {      "prettier/prettier": "error",      "no-unused-vars": "warn",      "no-console": "off",      "func-names": "off",      "no-process-exit": "off",      "object-shorthand": "off",      "class-methods-use-this": "off"    }  } |

**RUN WITH PARAMETERS (ANOTHER OPTION)**

There are one more option to run with parameters. For example, if you need to use in your application call to two different resources (for example, DEV.creation.com and DEV.track.com), and you need to change DEV environment to QA, UAT or some other environment, there are another way how to do it.

First, you need too install a lib named – **cross-env.**

***npm install --save-dev cross-env***

After that, in your package.json file a record will be created: "cross-env": "^6.0.3",

Then,

You need to determine an environment variable in your code:

*${process.env.NODE\_ENV}*

It looks like:

*This.url = `https://${process.env.NODE\_ENV}.creation.com`*

After that in package.json file under SCRIPTS path you need to create a new script to launch this parameterized call:

**“test:dev”:”cross-env NODE\_ENV=DEV npx protractor config/conf.js”**

Where:

**“test:dev” –** name of the script what has been called from CLI

**cross-env NODE\_ENV=DEV –** a point where we assign value DEV to environment variable NODE\_ENV using cross-env lib

**npx protractor config/conf.js –** launch protractor via its configuration file **conf.js**

**config/conf.js –** path to protractor configuration file