**Protractor**

Protractor is an end to end test framework for Angular and Angular JS applications.

WebDriver + Angular support = Protractor.

**Why Protractor:** The only stable framework to Automate Angular apps. It is developed and maintained by Angular team.

**Support language:** Protractor is a Node.JS program built on top of WebDriver JS. So it uses Java script as core language to develop tests.

**What are the tools/languages we came across in this Protractor learning?**

Node.js, Jasmine, JavaScript, Protractor API

Difference between JavaScript and TypeScript?

* Type script is not the new language
* It is superset of JavaScript
* JavaScript + Few Additional features(OOPS) = Type Script
* No matter you use JS or TS, Protractor API remains same in syntaxes and it is originally built in JS.
* Difference arises in Syntaxes when building the Test Framework for Protractor.
* One is JavaScript (Jasmine) and another is TypeScript(Cucumber)

**Installation:**

Download node.js and install it

Set the NODE\_HOME C:\Program Files\nodejs in system env variables

Check the node.js installed correctly or not by running the command in the command prompt “node -v”

To download the node from protractor package we will use npm(node pack manager). It automatically downloads when the node.js installed. Check the version using “npm -v”

Install protractor using npm install -g protractor

Whenever npm installs anything it will be stored in default path C:\Users\1024812\AppData\Roaming\npm

This will install two command line tools, protractor and webdriver-manager. Try running protractor --version to make sure it's working.

The webdriver-manager is a helper tool to easily get an instance of a Selenium Server running. Use it to download the necessary binaries with:

webdriver-manager update

Now start up a server with:

webdriver-manager start

This will start up a Selenium Server and will output a bunch of info logs. Your Protractor test will send requests to this server to control a local browser. You can see information about the status of the server at <http://localhost:4444/wd/hub>.

Install Eclipse IDE For JavaScript Developers

Jasmine

Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests.

**Describe:** defines a test suite

**It:** stands for testcase

The describe and it syntax is from the Jasmine framework

These two blocks are mandatory to write test case which is called spec(in jasmine terminology)

**Spec(Jasmine terminology)** - Test File (in java it is called class)

Every Spec should have describe and it blocks.

**Configuration file:** driver file. We will include all spec file names and run. Its like TestNG xml file.

Sample code:

describe('Protractor baby steps',**function**(){

it('basic program',**function**(){

//Protractor code

})

it('close program',**function**(){

//Protactor code

})

})

// first parameter - Test suite name (Protractor baby steps)

//second Parameter - function(){} will have all it blocks

//\*\*\*\*\*\*\*\*\*\*\*\*\*Parameters for IT block\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//first parameter - Test case name

//second parametr - function(){} will have protractor code for that test case

**Configuration**

Now create the configuration file. Copy the following into conf.js:

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

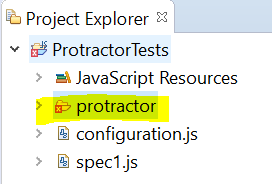
specs: ['todo-spec.js']

};

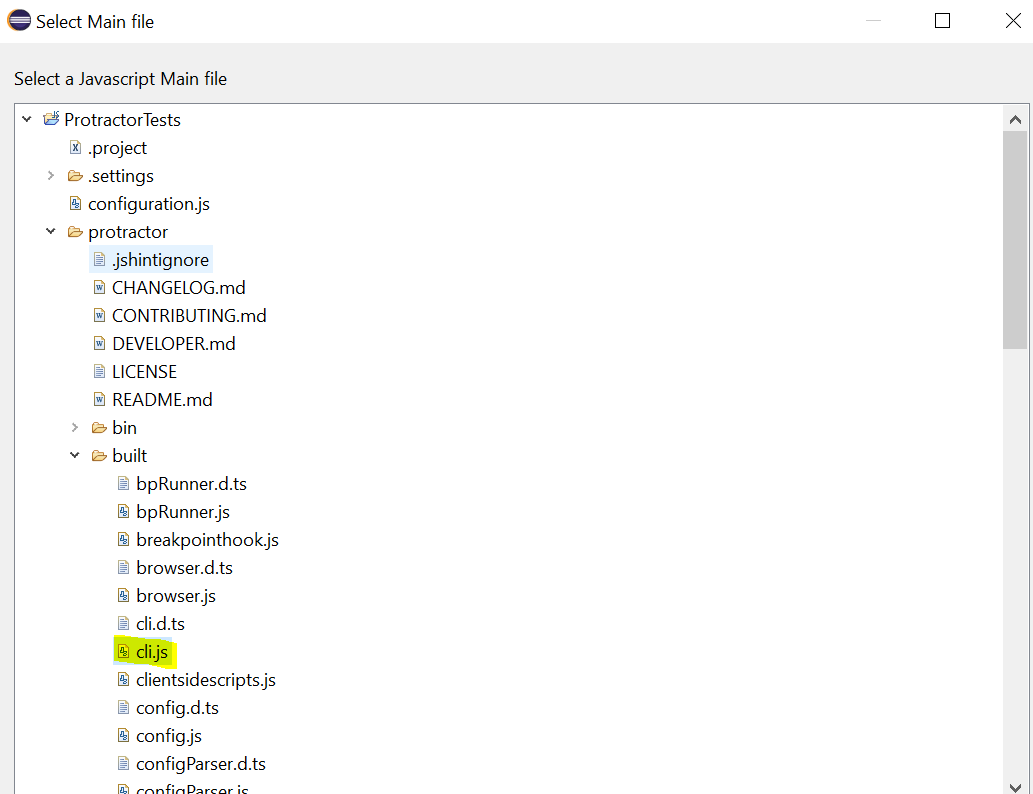
This configuration tells Protractor where your test files (specs) are, and where to talk to your Selenium Server (seleniumAddress). It will use the defaults for all other configuration. Chrome is the default browser.

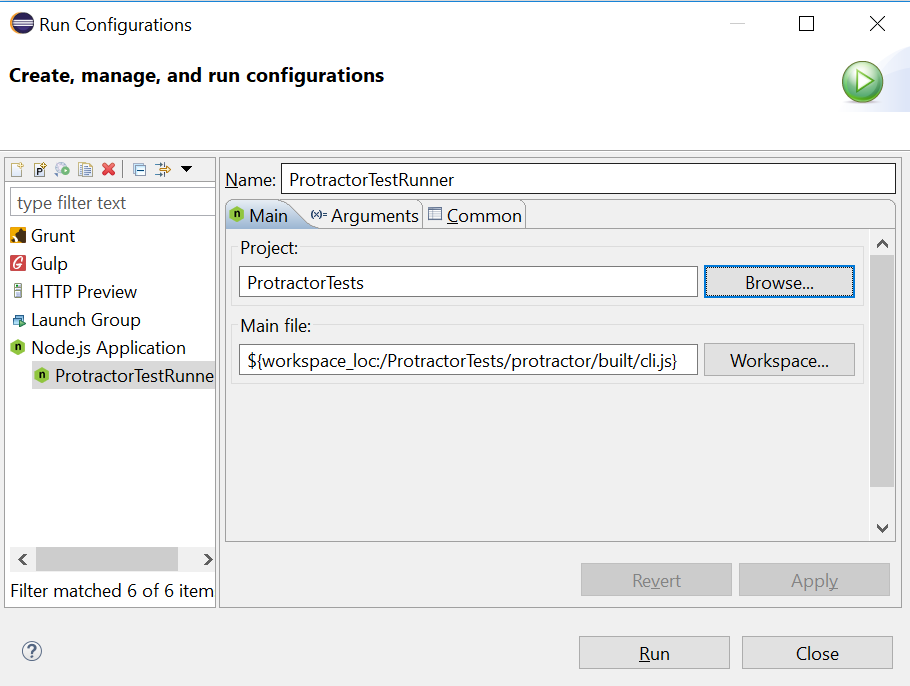
Running the Protractor Test:

1. Navigate to C:\Users\1024812\AppData\Roaming\npm\node\_modules and copy the protractor folder and paste into eclipse project.

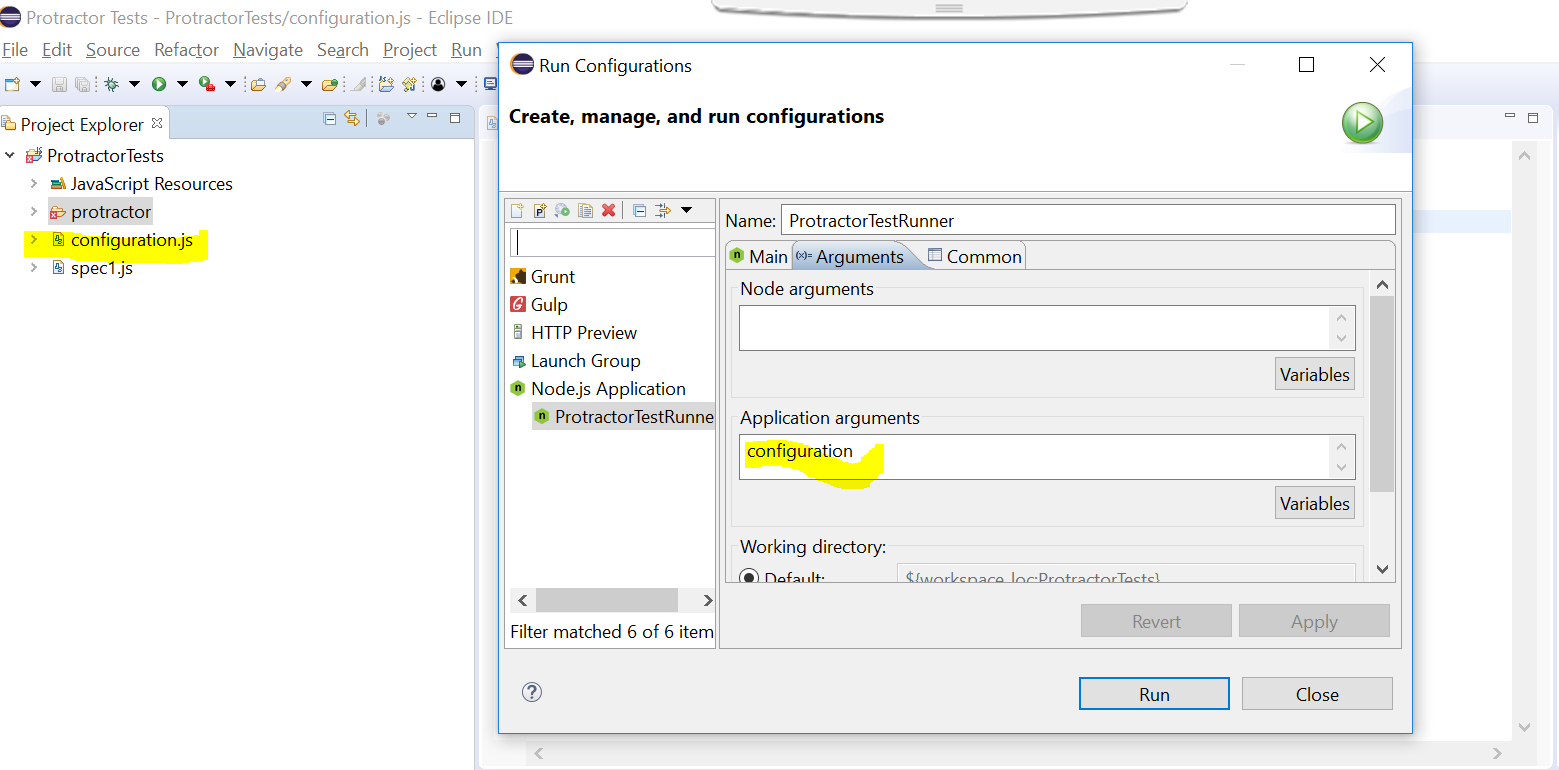


1. Right click on the project 🡪select Run as 🡪 Run configurations🡪Double click on Node.js application
2. Enter the Name, Select the current project in the project field and click on workspace and select the cli.js from the Project🡪protractor🡪built🡪cli.js





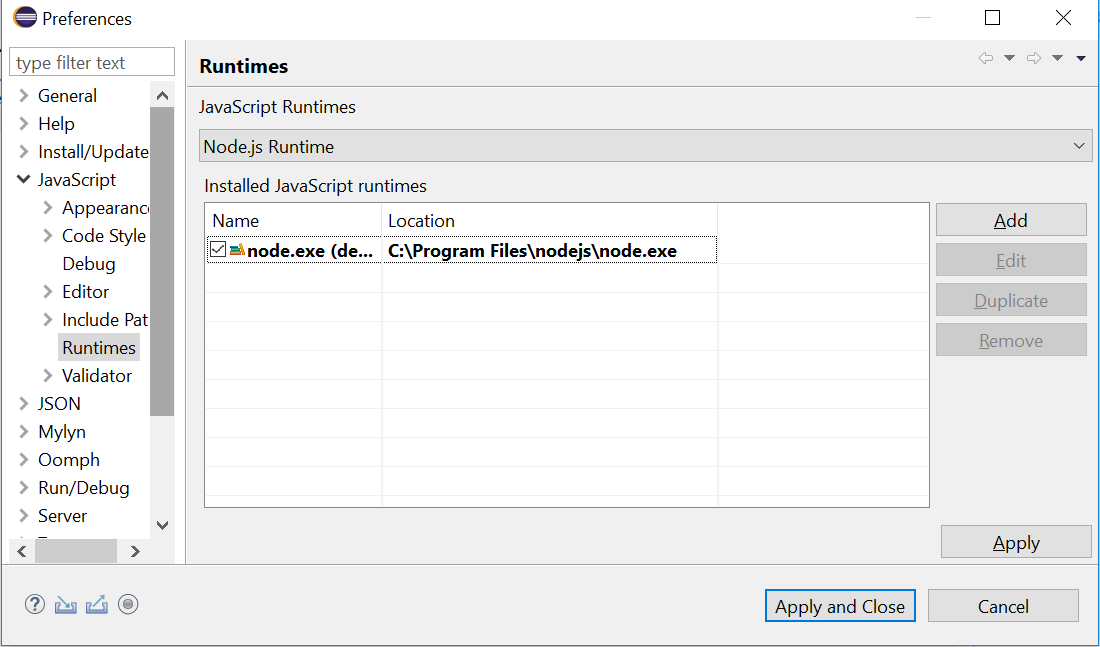
1. Move to arguments tab and enter the configuration file name in the application arguments and click on apply close.



1. Open Run configuration and select the test from the Node.js application and click on Run.
2. If you are receiving error like this “Unable to run Node.js program. Verify you have a configured Node.js runtime environment in your system and try again.”

Do the following.

1. Window 🡪 Preferences 🡪JavaScript 🡪 Run times 🡪 Select Node.js Runtime 🡪 select the node.exe from the folder and click on apply/close



**Install Angular JS intern plugin :** open eclipse market place🡪search angularjs and select the AngularSJ eclipse 1.2.0 and restart.

After restart right click on project🡪configure🡪convert to tern project(designed to help javascript project)🡪select Protractor and click on apply and close

Now you can see suggestions while you writing the script.

**JavaScript asynchronous and its problem with Protractor:**

JS is asynchronous i.e. the code does not run in sequential. To overcome this , we have option called Promise

Everything JS step returns Promise. Promise is nothing but a result of your step or current state of your step. Promise has 3 stages.

Pending --

Resolved – if it completes the step it is resolved state.

Rejected

In synchronous (Java, Python) you will move to next step only after promise for current step is either resolved or rejected.

Asynchronous: JS moves to next step even if promise is pending.

90% (all the actions perform on the browser) of our protractor API/methods (send keys, element etc. methods) will not move to next step until the Promise is resolved.

If you want to retrieve anything from browser (like get text, get title) – then that particular functionality related methods have not support for promise resolving.

We have to explicitly to run the steps in sequential by using the keyword “then” with the below

**Useful links:**

https://spin.atomicobject.com/2014/12/17/asynchronous-testing-protractor-angular/

https://www.sohamkamani.com/blog/2016/03/14/wrapping-your-head-around-async-programming/

<https://bridge360blog.com/2015/05/05/improving-protractor-tests-using-shared-functions-and-promises/>

Locators in Protractor

// Find an element using a css selector.

by.css('.myclass')

// Find an element with the given id.

by.id('myid')

// Find an element using an input name selector.

by.name('field\_name')

// Find an element with a certain ng-model.

// Note that at the moment, this is only supported for AngularJS apps.

by.model('name')

// Find an element bound to the given variable.

// Note that at the moment, this is only supported for AngularJS apps.

by.binding('bindingname')

CSS selector:

element(by.css("h2[class='ng-binding']"))

Important points:

1)real output will display only when you resolve promise

2)protractor for 90% of methods will resolve promise automatically.

3)there could be 2 potential problems a) sequence is not guaranteed.

Example: if you want to get any data i.e. text/title from the browser you need to resolve promise

browser.get("http://juliemr.github.io/protractor-demo/");

element(by.model("first")).sendKeys("3");

element(by.model("second")).sendKeys("5");

element(by.id("gobutton")).click();

element(by.css("h2[class='ng-binding']")).getText().then(**function**(textoutput){

console.log(textoutput);

})

**\*\*\*\*\*\*\*\*\*\*\*Assertions in Protractor\*\*\*\*\*\*\*\*\*\*\***

Assertions in Jasmine takes care of promise resolve

Sytax: expect(true).toBe(true);

Example: expect(element(by.css("h2[class='ng-binding']")).getText()).toBe("10");

describe('Element Locators',**function**(){

it('First Element Locator',**function**(){

browser.get("http://juliemr.github.io/protractor-demo/");

element(by.model("first")).sendKeys("3");

element(by.model("second")).sendKeys("5");

element(by.id("gobutton")).click();

//jasmine takes care of promise resolve

expect(element(by.css("h2[class='ng-binding']")).getText()).toBe("10");

element(by.css("h2[class='ng-binding']")).getText().then(**function**(textoutput){

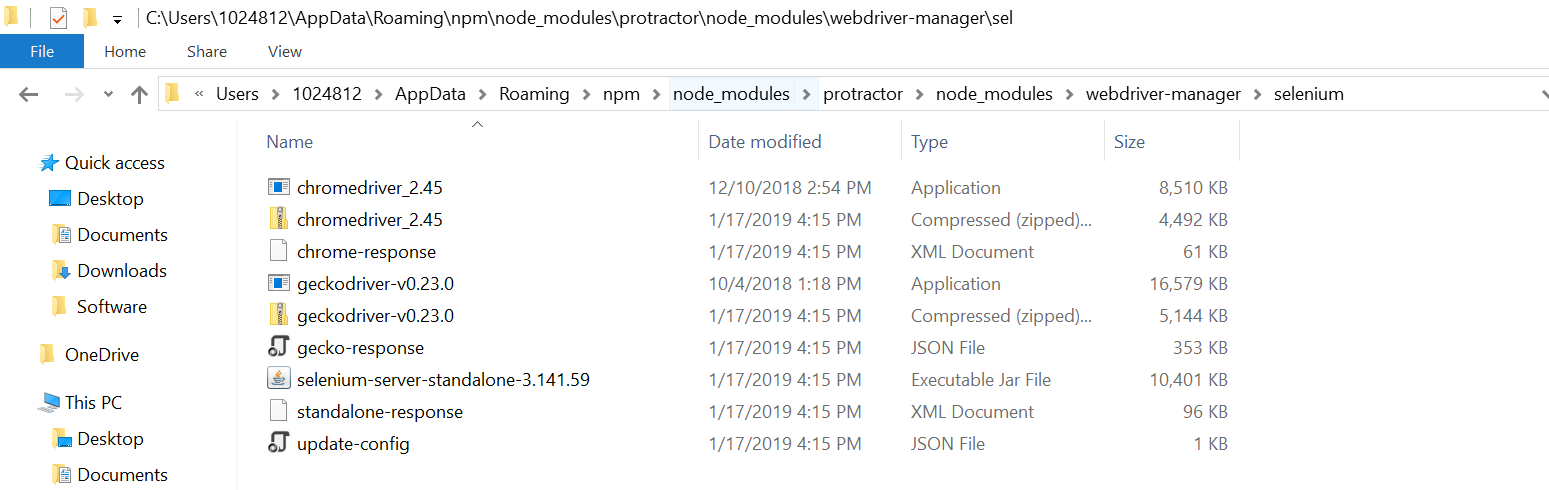
console.log("the additon is " + textoutput);

})

})

**Running tests on Firefox and internet explorer:**

**When we run the command webdriver-manager update it automatically downloads the Chrome driver and Firefox in the folder C:\Users\1024812\AppData\Roaming\npm\node\_modules\protractor\node\_modules\webdriver-manager\selenium**



By using the capabilities in the configuration file

exports.config = {

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

specs: ['ElementBasics.js'],

**capabilities: {**

**'browserName': 'firefox'**

**}**

}

TO work with IE:

1)stop the server if it is running and run the command **webdriver-manager update –ie**

2)it downloads the IE driver in the folder C:\Users\1024812\AppData\Roaming\npm\node\_modules\protractor\node\_modules\webdriver-manager\selenium

3)Start the server

4)Enter the below in configuration file

exports.config = {

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

specs: ['ElementBasics.js'],

capabilities: {

'browserName': 'internet explorer'

}

}

Importance of chain locators with Example:

3 important locators we have

1. Repeater
2. Chained locator – if we have same attributes
3. CSS for identical tags

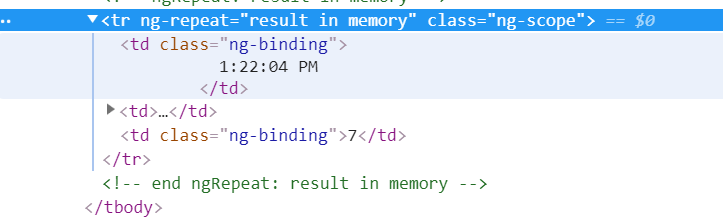
Repeater: if the tag has ng-repeat, then we can use repeater tag.

element(by.repeater("result in memory"));

Chained Locator:

element(by.repeater("result in memory")).element(by.css("td:nth-child(3)")).getText().then(function(text){})

Explanation: element(by.repeater("result in memory")) – this is the parent element and from the parent element we are accessing the child element by using element(by.css("td:nth-child(3)")). Here nth-child means we are accessing the 3rd child element i.e. 3rd td tag from the parent element.



**All method: used to get the list of elements**

element.all(by.repeater("result in memory")).count().then(**function**(text){

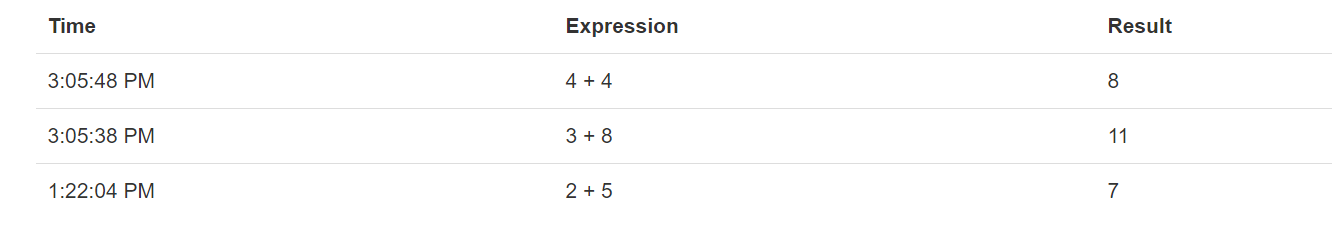
console.log(text);

})

In the above example we have 3 elements with the same name “result in memory”. By using this “all” method we can access all the 3 elements.

**“Each”: each is used to resolve each promise if we have multiple elements.Below is the example to retrieve web table data.Here we have 3 rows and we are printing the last column data of all rows. So we have used "each" instead of "then". If we use it will resolve only first one.**

**Output is: 8,11,7**



**Writing a function:**

Functions should be written in describe block and not in it block.

describe('Chained Locators',**function**(){

**function** Add(a,b){

element(by.model("first")).sendKeys(a);

element(by.model("second")).sendKeys(b);

element(by.id("gobutton")).click();

}

it('Chained Locator Example',**function**(){

browser.get("http://juliemr.github.io/protractor-demo/");

Add(2,5);

Add(4,5);

Add(6,5);

element.all(by.repeater("result in memory")).count().then(**function**(text){

console.log(text);

})

element.all(by.repeater("result in memory")).each(**function**(item) {

item.element(by.css("td:nth-child(3)")).getText().then(**function**(text)

{

console.log(text);

})

})

})

})

**Handling dropdowns:**

element.all(by.tagName("option")).each(**function**(item)

{

item.getAttribute("value").then(**function**(values)

{

**if**(values==c)

{

item.click();

}

})

})

Handling Auto Dropdowns:

element(by.model("userInputQuery")).sendKeys("river");

//Handling auto complete drop down

//Moving the mouse to edit box and then entering the value "London"

browser.actions().mouseMove(element(by.model("locationQuery")).sendKeys("London")).perform()

browser.actions().sendKeys(protractor.Key.ARROW\_DOWN).perform();

browser.actions().sendKeys(protractor.Key.ENTER).perform().then(**function**(){

//To get the count of the result

expect(element.all(by.css("div[ng-mousehover\*='onSearchResultOver']")).count()).toEqual(7)

})

**Handling Child windows with Protractor/Handling Windows opened in another tab:**

browser.getAllWindowHandles().then(**function**(handles){

//swithching to another window/tab by using index

browser.switchTo().window(handles[1]);

//To validate whether we switch successfully or not

browser.getTitle().then(**function**(childwindowtitle){

console.log("title of the child page after swithcing"+childwindowtitle)

})

**Handling Java alert Example:**

describe('Java Alert Demo',**function**(){

it('Handling Non JS Website Alerts',**function**(){

//if the website is non angular then you have write the below line

browser.waitForAngularEnabled(**false**);

browser.get("http://qaclickacademy.com/practice.php");

element(by.id("confirmbtn")).click();

browser.switchTo().alert().accept().then(**function**(){

browser.sleep(5000);

})

})

})

Handling Frames:

describe('Frames Demo',**function**(){

it('Frames',**function**(){

//if the website is non angular then you have write the below line

browser.waitForAngularEnabled(**false**);

browser.get("http://qaclickacademy.com/practice.php");

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

browser.switchTo().frame("iframe-name");

//browser.switchTo().alert().accept();

element(by.css("a[href\*='sign\_in']")).getText.then(**function**(text){

browser.sleep(5000);

console.log(text);

});

})

})

Synchronization: used for non angular apps

<https://www.protractortest.org/#/api?view=ProtractorExpectedConditions>

**Developing Protractor Framework**

1. Understand Objects in JavaScript
2. Page Object Mechanism in Protractor
3. Setup and Tear down method for tests
4. Importance of on Prepare Function in Framework
5. Understand Package.json
6. Remove the dependency of local system dependencies
7. Data Driven testing with Protractor
8. Report generation for test results
9. Understanding controlling test cases execution from Config file
10. Build npm commands to trigger the test
11. Importance of Continuous integration
12. Integrate Protractor with Jenkins

JavaScript Objects:

JavaScript is an obejct-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don’t create class to get the object. But, we direct create objects.

A JavaScript object is an entity having state and behaviour.

Function Car(){

This.colour=”red”;

This.engine=”turbo”;

This.model=”Mercedes”;

This.getModel=function(){

Console.log(“2020 model”);

};

};

**Java Syntax:**

Class Car {

String colour=”red”;

String engine=”turbo”

String brand=”Mercedes”

Public void getModel()

{

Sysout(“2020 Model”);

}

}

**Accessing the Java Objects from another JS file:**

1)Accessing from the same file

Create a object for the method and call it

Below is the function

**function** Car()

{

**this**.color="Red";

**this**.engine="turbo";

**this**.model="BMW";

**this**.getModel=**function**()

{

console.log("this is 2020 Model")

};

};

//code to access

**var** a= **new** Car();

a.getModel();

console.log(a.color);

How to Run: Right click🡪Run as 🡪 Node.js

From another JS file:

To expose the Object globally we will use

module.exports=new Car(); - by using this we can access the method anywhere in the protractor project.

// require - is used to inherit the properties from the child class. In java we called it as inheritance

Write the below code in another js file

**var** obj=require("./JavaScriptObjectsDemo.js");

obj.getModel();

console.log(obj.color)

Example:

JavaScriptObjectsDemo.js

**function** Car()

{

**this**.color="Red";

**this**.engine="turbo";

**this**.model="BMW";

**this**.getModel=**function**()

{

console.log("this is 2020 Model")

};

};

//code to access from the same file

//var a= new Car();

//a.getModel();

//console.log(a.color);

//Code to access from different file

module.exports=**new** Car();

JavaScriptObjectsDemo2.js

// require - is used to inherit the properties from the child class. In java we called it as inheritance

**var** obj=require("./JavaScriptObjectsDemo.js");

obj.getModel();

console.log(obj.color)

Page Object Mechanism for Protractor tests

**CalculatorPageObject.js**

**function** CalPageObject()

{

**this**.firstInput=element(by.model("first"));

**this**.secondInput=element(by.model("second"));

**this**.goButton=element(by.id("gobutton"));

**this**.result=element(by.css("h2[class='ng-binding']"));

}

module.exports=**new** CalPageObject();

**ElementBasicsWithPageObject.js**

describe('Elements with Page locators',**function**(){

**var** obj=require("./CalculatorPageObject.js");

it('Using Page Locator',**function**(){

browser.get("http://juliemr.github.io/protractor-demo/");

obj.firstInput.sendKeys("3");

obj.secondInput.sendKeys("5");

obj.goButton.click();

//jasmine takes care of promise resolve

expect(obj.result.getText()).toBe("10");

obj.result.getText().then(**function**(textoutput){

console.log("the additon is " + textoutput);

})

})

})

**beforeEach():** it is a pre-requistie for one test case.

beforeEach(**function**(){

obj.getURL();

})

**afterEach() :** It is a post-requistie for one test case.

afterEach(**function**(){

console.log("Test is completed")

});

**onPrepare() : It is a common global pre-requisite for all test cases. if the pre-requisite is common for all test cases then we should write onPrepare() method in Configuration file.**

onPrepare : **function**(){

//used for initializing the reports

//maximize the browser

}

}

**Understanding the importance of Package.json**

Package.json is a build tool available for npm packages, it can install the required packages to run a particular system, It will install npm packages only if the mentioned packages are not present in system  
  
Package.json tracks all your installations of npm packages, so when you want to install in different machine rather than using the individual commands you can use the package.son file to install them all together. Package.json not only tracks the installation it also tracks the dependencies for your installations  
  
**if you ask, is that all ?**  
Obviously no, using he package.json file we can also run our script, the major advantage will come when you want to run your script in remote machine, there you cannot install packages one by one, so those cases you just need to fire one single command to install all the packages from package.json and to one command to run the code.

How to create Package.JSON file?

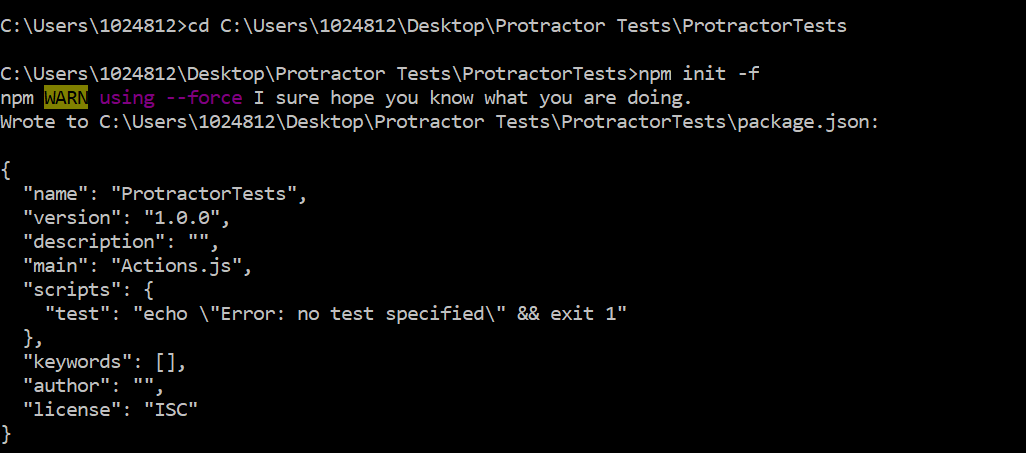
Open the command prompt and go to the project folder

cd C:\Users\1024812\Desktop\Protractor Tests\ProtractorTests

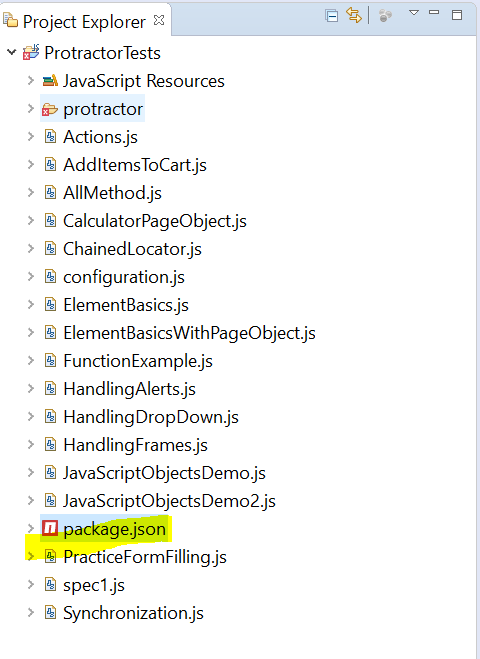
Run the below command

npm init - used to initialize the Package.json

if you want to initialize Package.JSON with default values then “npm init -f”



Referesh the project in Eclipse and see the Package.JSON added.



Edit the JSON file and Save

{

"name": *"package.json"*,

"dependencies": {

"protractor": *"^5.4.2"*

},

*"scripts"*: {

"test": *"echo \"Error: no test specified\" && exit 1"*

},

"author": *"Narayana"*,

"license": *"ISC"*

}

Run the command : npm install - this will install all the dependencies present in the Package.json

Refresh the project and see the node\_modules folder is added.

Go to Run🡪Run configurations and change the Main file(cli.js) location

Browse🡪select project 🡪node\_modules/protractor/built/cli.js

**How to add dependencies in Package.json file?**

Open the Package.json file and enter the below lines

"dependencies": {

"protractor": *"^5.4.2"*

},

This will install the npm\_modules folder in the project

**How to add npm commands in the Package.json file?**

"scripts": {

"test": *"echo \"Error: no test specified\" && exit 1"*,

"webdriver-update":*"./node\_modules/.bin/webdriver-manager update"*,

"webdriver-start":*"./node\_modules/.bin/webdriver-manager start"*,

"protractor":*"./node\_modules/.bin/protractor configuration.js"*,

"start":*"npm run webdriver-update && npm run webdriver-start"*

},

Explanation: Basically, here when we give

"webdriver-update":*"./node\_modules/.bin/webdriver-manager update"*,

i.e. here we are creating one variable with “webdriver-update”(it can be any name) and assigning the value i.e. location of the webdriver manager to it.it will first run the command webdriver-manager update from the location mentioned ./node\_modules/.bin

"webdriver-start":*"./node\_modules/.bin/webdriver-manager start"*,

i.e. here we are creating one variable with “webdriver-start” (it can be any name) and assigning the value i.e. location of the webdriver manager to it.

"protractor":*"./node\_modules/.bin/protractor configuration.js"*,

i.e. here we are creating one variable with “protractor” (it can be any name) and providing the configuration file name(configuration.js).

"start":*"npm run webdriver-update && npm run webdriver-start && npm run*

All the above 3 commands I am storing it in “start” variable.

To run the above command:

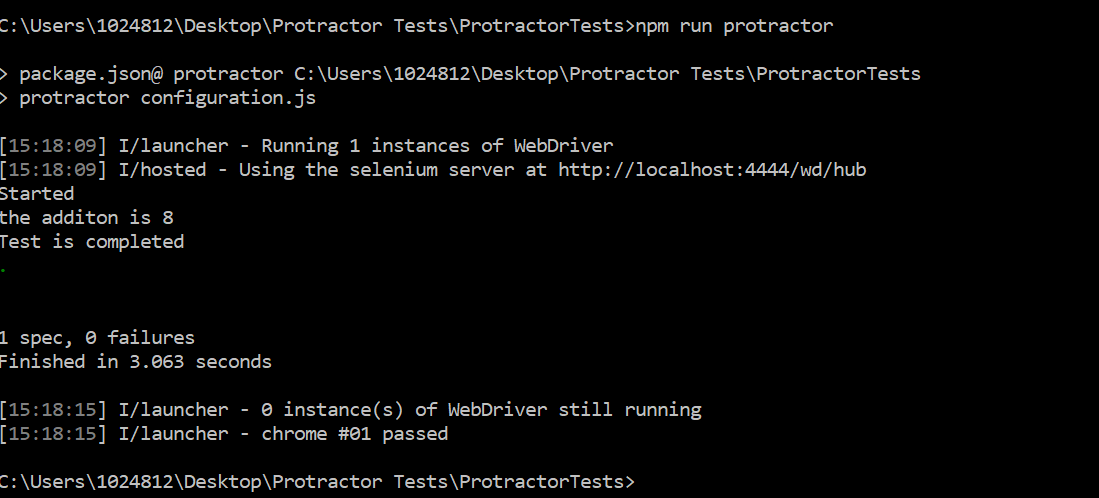
Open the command prompt and navigate to project folder 🡪 run the below commands

npm run start

it will run both the commands webdriver-manager update and webdriver-start.

Npm run protractor

This will execute the configuration file



**Data Driven Testing with Protractor**

Using **Jasmine-Data-provider dependency** we can perform Data Driven Testing

Add the dependency in the Package.json file

"dependencies": {

"protractor": *"^5.4.2"*,

"jasmine-data-provider": *"^2.2.0"*

},

Run the command “npm install” and it will download the dependency.

Refresh the project and see the jasmine-data-provider” folder added or not under npm\_modulues

Create the below JS file(TestDataProvider.js) to create multiple test data

module.exports={

DataDriven:

{

FirstData:

{

firstInput: "5",

secondInput:"6",

Result :"11"

},

SecondData:

{

firstInput: "5",

secondInput:"10",

Result :"15"

},

ThirdData:

{

firstInput: "5",

secondInput:"15",

Result :"20"

}

}

}

To use the above test data into test case,

describe('Elements with Page locators',**function**(){

**//To import Jasmine-Data Provider use the below command**

**var using=require("jasmine-data-provider");**

**//To import Page Objects from CalculatorPageObject JS file**

**var** obj=require("./CalculatorPageObject.js");

**//To import Test data defined in TestDataProvider JS file**

var testdata=require("./TestDataProvider.js");

beforeEach(**function**(){

obj.getURL();

});

**//by using this we can call test data from other file**

**//here plusProvider is the Parent object name in the TestDataProvider JS file**

**//data stores sub object name i.e. example for TestDataProvider.js is FirstData values (firstinput**

**//description - for every iteration one data is picked and it is stored i.e. FisrtData name**

**Syntax:**

**// using(plusProvider, function (data,description) {**

**//**

**// });**

using(testdata.DataDriven, **function** (data,description) {

it('Using Page Locator',**function**(){

//browser.get("http://juliemr.github.io/protractor-demo/");

obj.firstInput.**sendKeys(data.firstInput);**

obj.secondInput.sendKeys(**data.secondInput**);

obj.goButton.click();

//jasmine takes care of promise resolve

expect(obj.result.getText()).toBe(data.Result);

obj.result.getText().then(**function**(textoutput){

console.log("the additon is " + textoutput);

})

})

});

afterEach(**function**(){

console.log("Test is completed");

});

})

Generate HTML Reports in Protractor:

# By using protractor-jasmine2-html-reporter plugin

HTML reporter for Jasmine2 and Protractor that will include screenshots of each test if you want.

Add the dependenct in the Package.json file

{

"name": *"package.json"*,

"dependencies": {

"protractor": *"^5.4.2"*,

"jasmine-data-provider": *"^2.2.0"*,

"protractor-jasmine2-html-reporter":*"^0.0.7"*

},

Run the command npm install and see the folder “protractor-jasmine2-html-reporter” added in the node\_modules

Add the below highlighted code in the configuration file

**var Jasmine2HtmlReporter = require('protractor-jasmine2-html-reporter');**

exports.config = {

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

specs: ['PassingTestDataFromTestDataProvider.js'],

// capabilities: {

// 'browserName': 'internet explorer'

// }

onPrepare : **function**(){

//maximize the browser

//used for initializing the reports

**jasmine.getEnv().addReporter(**

**new Jasmine2HtmlReporter({**

**savePath: 'target/screenshots'**

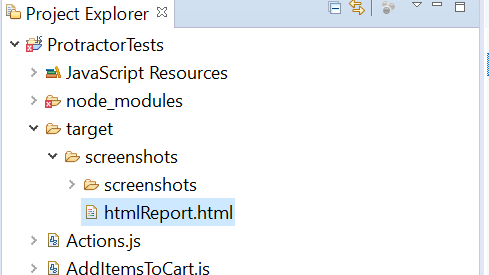
**})**

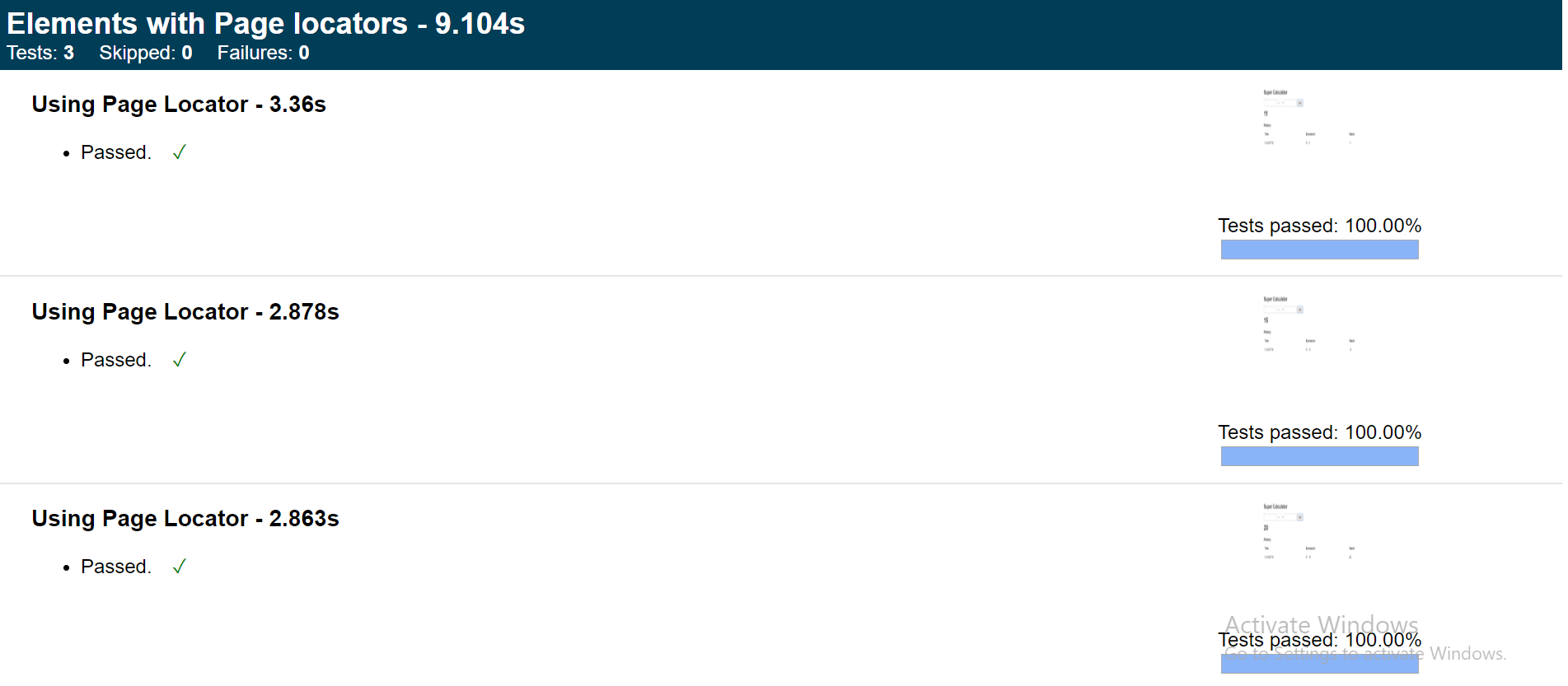
**);**

}

}

Once you added the above code in configuration file run the test case and refresh the project and see the folder “target/screenshots” added in the project





**Tags Configuration to control Test execution:**

If we want to execute only regression test cases or smoke test cases, add the below in the configuration file

suites:

{

Smoke : ['Actions.js','ChainedLocator.js'],

Regression : ['PassingTestDataFromTestDataProvider.js']

},

In Package.json file

"SmokeSuite": "./node\_modules/.bin/protractor configuration.js --suite Smoke",

{

"name": *"package.json"*,

"dependencies": {

"protractor": *"^5.4.2"*,

"jasmine-data-provider": *"^2.2.0"*,

"protractor-jasmine2-html-reporter": *"^0.0.7"*

},

"scripts": {

"test": *"echo \"Error: no test specified\" && exit 1"*,

"webdriver-update": *"./node\_modules/.bin/webdriver-manager update"*,

"webdriver-start": *"./node\_modules/.bin/webdriver-manager start"*,

"protractor": *"./node\_modules/.bin/protractor configuration.js"*,

**"SmokeSuite": *"./node\_modules/.bin/protractor configuration.js --suite Smoke"*,**

"start": *"npm run webdriver-update && npm run webdriver-start"*

},

"author": *"Narayana"*,

"license": *"ISC"*

}

To execute, in the command prompt

Run the command: npm run SmokeSuite

**For new project do the set up**

**1)Create the project in eclipse**

**2)open the command prompt and navigate to the project folder and run the command npm init. This will create package.json file in the project folder**

**3)once the package.json file created, open it and enter the dependencies(highlighted in yellow color) you required**

**It should be like this**

{

"name": *"protractorjasmineframework"*,

"version": *"1.0.0"*,

"description": *""*,

"main": *"index.js"*,

**"dependencies": {**

**"protractor": *"^5.4.2"*,**

**"jasmine-data-provider": *"^2.2.0"*,**

**"protractor-jasmine2-html-reporter": *"^0.0.7"***

**},**

"scripts": {

"test": *"echo \"Error: no test specified\" && exit 1"*

},

"author": *""*,

"license": *"ISC"*

}

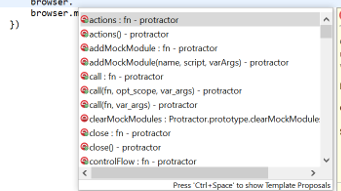
**4)open the command prompt and navigate to the project folder and run the command npm install. This will install protractor folder(npm\_modules folder) under the project directory.**

**5)Install Angular JS intern Plugin and convert the project into tern project.**

**Install Angular JS intern plugin :** open eclipse market place🡪search angularjs and select the AngularSJ eclipse 1.2.0 and restart.

After restart right click on project🡪configure🡪convert to tern project(designed to help javascript project)🡪select Protractor and click on apply and close

This plugin will give you suggestions when you write the command. For example, if you write browser. and press control+space it will give you the suggestions like get…



**6)Create one spec file and write the test case code and save it**

**6)Create one configuration file , write the below code and save it.**

exports.config = {

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

specs: ['BrowserCommandsScript.js']

// capabilities: {

// 'browserName': 'internet explorer'

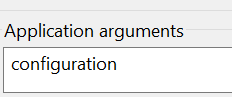
// }

}

**7)** Right click on the project 🡪select Run as 🡪 Run configurations🡪Double click on Node.js application

Enter the Name, Select the current project in the project field and click on workspace and select the cli.js from the project🡪node\_modules🡪protractor🡪built🡪cli.js

Navigate to arguments tab🡪enter the configuration file name in the below field



8)To run the spec file from eclipse, select run🡪run configuration🡪seelct the created test runner from step 7 and click on Run.

Protactor Browser Commands

To open WebPage: browser.get('http://www.way2automation.com/angularjs-protractor/banking/#/login');

Maximize the browser: browser.manage().window().maximize();

To set the window size : browser.manage().window().setSize(200, 300);

To get the window size:

browser.manage().window().getSize().then(**function**(windowsize){

console.log("Browser width : " + windowsize.width);

console.log("Browser Height: " + windowsize.height);

})

Set window position: browser.manage().window().setPosition(200, 300);

Get window Position:

browser.manage().window().getPosition().then(function(windowLocation){

console.log("Browser x postion : " +windowLocation.x)

console.log("Browser y postion : " +windowLocation.y)

})

To get the current URL of the page:

browser.getCurrentUrl().then(**function**(getCurrentURL){

console.log("The current url is: " + getCurrentURL)

})

To get the title of the webpage:

browser.getTitle().then(function(title){

console.log("The title of the webpage is: " + title);

})

To get the page source of a webpage

browser.getPageSource().then(**function**(pagesource){

console.log("The page source is: " + pagesource);

})

To refresh the webpage: browser.navigate().refresh();

To back to the webpage: browser.navigate().back();

To forward to the webpage: browser.navigate().forward();

To move to specifif URL: browser.navigate().to(“url”);

**Locators in Protractor:**

* ***binding :****Find an element by text binding, binding locator finds the element based on partial match of****ng-bind attribute****, if the attribute has some value matching with given locator then this element will be found by our locator-*

//HTML code

<span ng-bind="person.email.id"></span>

// usage of locator

element(by.binding('person.email'));

* ***exactBinding :****Finds the element with exact value, if ng-bind is equal the given locator value then the elemnet will be found. It will not check for partial matches.-*

//HTML code

<span ng-bind="person.email.id"></span> // this will not be found as it is not exact value

<span ng-bind="person.email"></span> // this element will be found as it is exach match

// usage of locator

element(by.binding('person.email'));

***model :*** Model locator checks whether any element has ng-model values matching with given locator, if so that element will be returned-

//HTML code

<span ng-model="person.name"></span>

// usage of locator

element(by.model('person.name'));

**buttonText :** buttonText locator will try to match with an element which as same text as the given locator, or its sub element inside the button tag-

//HTML code

<button>Save</button>

// usage of locator

element(by.buttonText('Save'));

//HTML code

<button>

<label>Save As //matches

</label>

</button>

// usage of locator

element(by.buttonText('Save As'));

**partialButtonText : partialButtonText** locator tries to find element with partial match present in the button element's text.-

//HTML code

<button>Save As Text </button> //matches

// usage of locator

element(by.buttonText('Save'));

//HTML code

<button>

<label>Save As File //matches

</label>

</button>

// usage of locator

element(by.buttonText('Save As'));

**repeater :** Find elements inside an ng-repeat. -

<div ng-repeat="cat in pets">

<span>{{cat.name}}</span>

<span>{{cat.age}}</span>

</div>

<div class="book-img" ng-repeat-start="book in library">

<span>{{$index}}</span>

</div>

<div class="book-info" ng-repeat-end>

<h4>{{book.name}}</h4>

<p>{{book.blurb}}</p>

</div>

// Returns the DIV for the second cat.

var secondCat = element(by.repeater('cat in pets').row(1));

// Returns the SPAN for the first cat's name.

var firstCatName = element(by.repeater('cat in pets').

row(0).column('cat.name'));

// Returns a promise that resolves to an array of WebElements from a column

var ages = element.all(

by.repeater('cat in pets').column('cat.age'));

// Returns a promise that resolves to an array of WebElements containing

// all top level elements repeated by the repeater. For 2 pets rows

// resolves to an array of 2 elements.

var rows = element.all(by.repeater('cat in pets'));

// Returns a promise that resolves to an array of WebElements containing

// all the elements with a binding to the book's name.

var divs = element.all(by.repeater('book in library').column('book.name'));

// Returns a promise that resolves to an array of WebElements containing

// the DIVs for the second book.

var bookInfo = element.all(by.repeater('book in library').row(1));

// Returns the H4 for the first book's name.

var firstBookName = element(by.repeater('book in library').

row(0).column('book.name'));

// Returns a promise that resolves to an array of WebElements containing

// all top level elements repeated by the repeater. For 2 books divs

// resolves to an array of 4 elements.

var divs = element.all(by.repeater('book in library'));

**exactRepeater :** exactRepeater locators looks for an element which has exact same attibute value-

cssContainingText : cssContainingText locator tries to find elements by CSS which contain a certain string.-

<ul>

<li class="pet">Dog</li> //matches

<li class="pet">Cat</li>

</ul>

// Returns the li for the dog, but not cat.

element(by.cssContainingText('.pet', 'Dog'))

options : -

<select ng-model="color" ng-options="c for c in colors">

<option value="0" selected="selected">red</option>

<option value="1">green</option>

</select>

var allOptions = element.all(by.options('c for c in colors'));

expect(allOptions.count()).toEqual(2);

var firstOption = allOptions.first();

expect(firstOption.getText()).toEqual('red');

**deepCss :** Find an element by css selector within the Shadow DOM.-

<div>

<span id="outerspan">

<"shadow tree">

<span id="span1"></span>

<"shadow tree">

<span id="span2"></span>

</>

</>

</div>

var spans = element.all(by.deepCss('span'));

expect(spans.count()).toEqual(3);

**Inherited locators**

Below locators are available in protractor as inherited locators, these locators are inherited from selenium webdriver

id-

name-

className-

tagName-

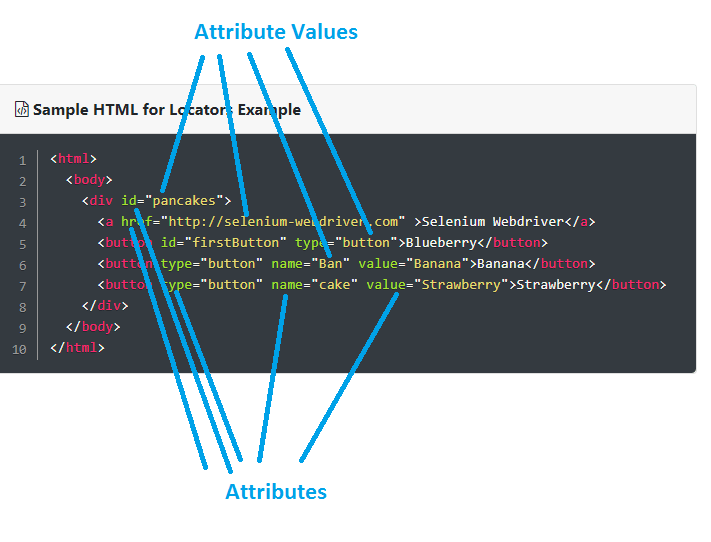
partialLinkText -

linkText-

css-

js-

xpath



element(by.id("firstButton")

element(by.name("Ban")

element(by.className("banana")

element(by.tagname("button")

element(by.linkText("Selenium Webdriver")

element(by.partialLinkText("Selenium")

element(by.css("#firstButton")

element(by.xpath("//button[@name="Ban"]")

Xpath: refer below website for xpath

<https://chercher.tech/protractor/xpath-protractor>

css : <https://chercher.tech/protractor/css-selector-protractor>

**How to skip a test case in Protractor:**

When we run our test cases sometimes we might need to skip the tests present in the spec file

Use x prefix to a test to skip a test in the jasmine

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true;

// below test will be skipped as it has prefix as 'x'

**xit**('click operation', function(end) {

setTimeout(function(){

end();

}, 5000)

console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

});

});

**How to run a particular test in Jasmine :**

Sometimes you might have 20 it blocks in your spec file, but if you want to only a particular it block. Do you think it is feasible for prefixing all the 19 it blocks with x ?? Very Big NO.

Run any particular 'it block' by prefixing it with f

describe('Protractor Typescript Demo', function() {

**fit**('click operation', function() {

console.log("Iam the only one who will run");

});

it('ignore test 1', function() {

console.log("they have not prefixed me with f");

});

it('ignore test 2', function() {

console.log("they have not prefixed me with f");

});

});

BeforeEach block: BeforeEach block will contains all the steps which are pre-requisite and common for all the test cases like opening a browser and navigating to the URL.

beforeEach method will get executed before executing an it block

beforeEach function is applicable for all the it blocks present inside a [describe](https://chercher.tech/protractor/jasmine-protractor#describe)block.

describe('Protractor Typescript Demo', function() {

beforeEach(function(){

console.log("Before Each block")

});

it('aaa', function() {

console.log("test One");

});

it('bbb', function() {

console.log("test two");

});

it('ccc', function() {

console.log("test three");

});

})

**afterEach block:**

afterEach block will be executed after completing a it block inside a [describe](https://chercher.tech/protractor/jasmine-protractor#describe)block.

afterEach block is applicable for all the it blockspresent in a [describe](https://chercher.tech/protractor/jasmine-protractor#describe)block

afterEach block will be executed after completing a it block inside a [describe](https://chercher.tech/protractor/jasmine-protractor#describe)block.

afterEach block is applicable for all the it blockspresent in a [describe](https://chercher.tech/protractor/jasmine-protractor#describe)block

describe('Protractor Typescript Demo', function() {

afterEach(function(){

console.log("After Each block")

});

it('aaa', function() {

console.log("test One");

});

it('bbb', function() {

console.log("test two");

});

it('ccc', function() {

console.log("test three");

});

})

**Describe block:**

describe block is nothing but the test Suite, a [describe](https://chercher.tech/protractor/jasmine-protractor#describe)block can contains multiple it blocks.  
  
Like it block, describe block also will have two parameters in the syntax, 1. Description of the suite, 2. function.

**Assertions in Jasmine Framework**

toBe() : passed if the actual value is of the same type and value as that of the expected value. It compares with === operator

toEqual() : works for simple literals and variables; should work for objects too

toMatch() : to check whether a value matches a string or a regular expression

toBeDefined() : to ensure that a property or a value is defined

toBeUndefined() : to ensure that a property or a value is undefined

toBeNull() : to ensure that a property or a value is null.

toBeTruthy() : to ensure that a property or a value is true

toBeFalsy() : to ensure that a property or a value is false

toContain() : to check whether a string or array contains a substring or an item.

toBeLessThan() : for mathematical comparisons of less than

toBeGreaterThan() : for mathematical comparisons of greater than

toBeCloseTo() : for precision math comparison

toThrow() : for testing if a function throws an exception

toThrowError() : for testing a specific thrown exception

not : we can invert above expectations using not keyword

Configuration file:

Conf file is the trigger for the protractor package execution.

The execution of protractor starts from the conf file, some people also call it as config file or protractor.config file.  
  
Conf file a javascript file, which will invoke the total framework for execution.  
  
We can set differemt properties in conf file like Remote machine ips, hubs and nodes. Local machine configurations .

Conf file helps the user to set pre and post completion actions for the test suites and for launches

**Protractor test related parameters in config file**

**Specs:**

specs parameter is an Arrays, which holds the tests file location for running the files.

You can also provide regular expression inside the specs file to find the test files.

We can provides the locations relative to this conf.js file or full locations like below.

specs: ['D:\\Protractor Demo\\specs\\test.js']

specs: ['D:\\Protractor Demo\\specs\\\*.js'] // will run all the files with js extension

If you donot set any value then, protractor will say no specs found.

**exclude :**

We can exclude test files using exclude parameter in conf.js file. Exclude parameter will be useful when you do not want to run only few tests or a folder.

Exclude also accepts Array as parameter values, also responds to regular expression.

exclude: ['D:\\Protractor Demo\\specs\\dummytest.js']

**Suites :**

We can suites instead of Specs when we want to categorize the test. When run without a command line parameter, all suites will run.

If run with --suite=smoke or --suite=smoke,full only the patterns matched by the specified suites will run.

suites: {

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

sanity: 'spec/sanitytests/\*.js',

full: 'spec/\*.js'

}

**suite :**

If you would like protractor to use a specific suite by default instead of all suites, you can put that in the config file as well.

To use suite parameter, configuring suites parameter is very important.

suite:sanity,

**Hooks in Protractor file**

**beforeLaunch :**

A callback function called once configs are read but before any environment setup. This will only run once, and before onPrepare.

You can specify a file containing code to run by setting beforeLaunch to the filename string.

At this point, global variable 'protractor' object will NOT be set up, and globals from the test framework will NOT be available. The main purpose of this function should be to bring up test dependencies.

**onPrepare :**

A callback function called once protractor is ready and available, and before the specs are executed.

If multiple capabilities are being run, this will run once per capability.

You can specify a file containing code to run by setting onPrepare to the filename string. onPrepare can optionally return a promise, which Protractor will wait for before continuing execution.

This can be used if the preparation involves any asynchronous calls, e.g. interacting with the browser. Otherwise Protractor cannot guarantee order of execution and may start the tests before preparation finishes.

At this point, global variable 'protractor' object will be set up, and globals from the test framework will be available.

**onComplete :**

A callback function called once tests are finished. onComplete can optionally return a promise, which Protractor will wait for before shutting down webdriver.

At this point, tests will be done but global objects will still be available.

**onCleanUp :**

A callback function called once the tests have finished running and the WebDriver instance has been shut down. It is passed the exit code (0 if the tests passed). This is called once per capability.

**afterLaunch :**

A callback function called once all tests have finished running and the WebDriver instance has been shut down. It is passed the exit code (0 if the tests passed).

afterLaunch must return a promise if you want asynchronous code to be executed before the program exits.

This is called only once before the program exits (after onCleanUp).

**Params in Protractor config.js file**

The params object will be passed directly to the Protractor instance, and can be accessed from your test as browser.params. It is an arbitrary object and can contain anything you may need in your test.

This can be changed via the command line as: --params.login.user "Joe"

params: {

login: {

user: 'Jane',

password: '1234'

}

}

*WAITS in Protractor*

Implictit Wait:

implicitly wait in protractor, sets maximum time that we are going to wait for the element to be available in the Website.

Implicit wait affects only element and element.all methods in protractor

If we have given implicit wait of 30,000 milli-seconds then, Implicit wait tries to find the element in first go, if element is not present implicit wait tries to find the element after 500ms of first polling, if element is not available on second time also then implicit wait tries third time after 500 ms of second try and it goes on till the time reaches the 30,000 milli-seconds

// implicitly wait

browser.manage().timeouts().implicitlyWait(30000)

If element is found before implicitly wait time, selenium moves to next commands in the program without waiting to complete the implicitly wait time, this wait is also called dynamic wait.

Implicit wait is set for the entire duration of your webdriver and is set at the start of your program. Most of the automation testers write the implicit wait in the conf file.

Implicitly wait is one of the way to request protractor not throw any exception until provided time. Default wait time of the selenium is 500 milli-seconds, implicitly wait overrides the default wait time of the selenium webdriver.

Setting the wait timeout to 0 (its default value), disables implicit waiting, but not recommended

Implicitly wait method accepts int parameter, which specifies the amount of time the protractor should wait when searching for an element if it is not immediately present.

When searching for a single element, the protractor should poll the page until the element has been found, or this timeout expires before failing with a NO\_SUCH\_ELEMENT Error

When searching for multiple elements, the protractor should poll the page until at least one element has been found or this timeout has expired.

import { browser, element, by} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

browser.manage().timeouts().implicitlyWait(50000)

it('sendkeys operation', function() {

browser.get('https://google.com/');

browser.sleep(1000)

element(by.name('noSuchNameIsThere')).sendKeys("hello")

});

});

Above example will wait for 50 seconds, till that it polls again and again to find the element but we have given an 'name' locator which is not present in the google page.

Video Tutorial for Implicit Wait in protractor

Subscribe to my youtube channel :

Click Element with Javascript Executor in Protractor

Scope of Implicitly Wait in Protractor

Once set, the implicit wait is set for the life of the protractor object instance. You can re-assign the implicit wait time anywhere you want.

When does Implicitly Wait is applicable for element.all method. :

Implicit wait in protractor is applicable for all element statements, but applicability of implicit wait to the element.all is Limited.

Implicitly wait is applicable on element.all only when there is no elements present, protractor moves to next command the moment it finds a single element.

If there is Zero matching elements protractor waits till the time it reaches provided implicitly wait

Scenario 1 : Consider a page having 100 checkboxes and applied implicit wait is 30 Seconds

When the page is loading protractor tries to find all the matching elements, but at that moment there no matching elements .

After 10 seconds of wait 1 check box is loaded, now protractor finds that one checkbox and concludes that there is only one element, so it ends the wait and moves to next command.

protractor doesnot wait till 100 check boxes loaded into page, as protractor doesnot know how many elements going to be present on the page.

Scenario 2 : Consider a page having 0 checkboxes and applied implicit wait is 30 Seconds

Protractor tries to find all the matching elements in DOM, but so far no check box is loaded.

Now there is zero match so protractor waits till 30 seconds, and if doesnot find element after 30 seconds protractor throws NoSuchElement Error

Explicit wait in Protractor

The explicit wait tells the protractor to wait for certain conditions or the maximum time limit before throwing an "Exception".

ExplicitWait is a dynamic wait, which means it will wait only till the condition is not met, the moment condition is met then protractor start executing next line of code.

For example: if you are waiting for a condition with 60 seconds, if the condition is meth at 4th second then Explicit wait will not wait till 60 seconds.

ExplicitWait does not have any effect on element and element.all, ExplicitWait also called as WebdriverWait.

ExplicitWait by default calls the ExpectedCondition every 500 milliseconds until it returns successfully.

browser.wait(conditions)

Most of the time the conditions are used from ExpctedCoditions class in protractor.

Expected Conditions in Protractor

ExpectedConditions is a class represents a library of canned expected conditions that are useful for protractor, especially when dealing with non-angular apps.

Each condition returns a function that evaluates to a promise

alertIsPresent

elementToBeClickable

textToBePresentInElement

textToBePresentInElementValue

titleContains

titleIs

urlContains

urlIs

presenceOf

stalenessOf

visibilityOf

invisibilityOf

elementToBeSelected

not

and

or

alertIsPresent

alertIsPresent method wait checks whether an alert is present or not, the value is returned in promise, we have to resolve it to get the value.

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

browser.sleep(1000)

element(by.id('alert')).click();

browser.wait(ExpectedConditions.alertIsPresent(), 30000)

});

});

Sometimes instead of writing ExpectedConditions full form all the we can store it in a variable and use a short name like below.

Below code wait for an alert to be present within 30 seconds, if not the browser.wait() will throw an Timeout Error.

let EC = ExpectedConditions;

browser.wait(EC.alertIsPresent(), 30000)

elementToBeClickable

elementToBeClickable() method will wait till, a given element becomes clickable. If element is not clickable after reaching the given time limit then protractor will throw Timeout Error.

In below example if you click button 'Enable after 10 seconds' and wait for the target button to become clickable, max in 30 seconds.

We can store the total condition in a variable.

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

browser.sleep(7000)

element(by.id('enable-button')).click();

let EC = ExpectedConditions;

let condition = EC.elementToBeClickable(element(by.id("disable")))

browser.wait(condition, 30000)

});

});

textToBePresentInElement

textToBePresentInElement() methods waits till a given element's text becomes some value that we pass.

We have to pass target and element and expected text to this message.

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

browser.sleep(7000)

element(by.id('populate-text')).click();

let EC = ExpectedConditions;

let condition = EC.textToBePresentInElement(element(by.class("target-text")), "Selenium Webdriver")

browser.wait(condition, 30000)

});

});

textToBePresentInElementValue

textToBePresentInElementValue method is almost like textToBePresentInElement method, but the change is; in textToBePresentInElementValue we are looking for matching case sensitive substring the text values.

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

browser.sleep(7000)

element(by.id('populate-text')).click();

let EC = ExpectedConditions;

let condition = EC.textToBePresentInElementValue(element(by.class("target-text")), "Webdriver")

browser.wait(condition, 30000)

});

});

titleContains

titleContains() wait for an webpage title contains a given substring.

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

let EC = ExpectedConditions;

let condition = EC.titleContains("Practice")

browser.wait(condition, 30000)

});

});

titleIs

titleIs() method wait till a page title becomes required string, but if the title is not given string then protractor throws exception

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

let EC = ExpectedConditions;

let condition = EC.titleIs("ExplicitlyWait Practice page in Selenium Webdriver")

browser.wait(condition, 30000)

});

});

urlContains

urlContains() method waits till a page url becomes to contain expected substring or till the timeout reaches

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

let EC = ExpectedConditions;

let condition = EC.urlContains("explicit")

browser.wait(condition, 30000)

});

});

urlIs

urlIs() waits till the page url becomes expected string or till max timeout

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

let EC = ExpectedConditions;

let condition = EC.urlContains("explicit-wait-sample-selenium-webdriver")

browser.wait(condition, 30000)

});

});

presenceOf

presenceOf() methods waits will a particular element is present with given max timeout. If element is not present in webpage with given time then it will throw an TimeoutError.

let EC = ExpectedConditions;

let condition = EC.presenceOf(element(by.id("something")))

browser.wait(condition, 30000)

stalenessOf

stalenessOf() method waits for an element to be refreshed.

let EC = ExpectedConditions;

let condition = EC.stalenessOf(element(by.id("something")))

browser.wait(condition, 30000)

visibilityOf

An expectation for checking that an element is present on the DOM of a page and visible.

Visibility means that the element is not only displayed but also has a height and width that is greater than 0.

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

element(by.id("display-other-button")).click()

let EC = ExpectedConditions;

let condition = EC.visibilityOf(element(by.id("hidden")))

browser.wait(condition, 30000)

});

});

invisibilityOf

This is opposite of visibilityOf() method, it wait for an element to disappear within given time.

elementToBeSelected

elementToBeSelected() this method waits till Checkbox, Radio button, or dropdown option is selected with given time limit.

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

element(by.id("checkbox")).click()

let EC = ExpectedConditions;

let condition = EC.elementToBeSelected(element(by.id("ch")))

browser.wait(condition, 30000)

});

});

not

not() method in expected condition invert the given expected condition.

Below code waits for element to be unchecked with a specific time

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

element(by.id("checkbox")).click()

let EC = ExpectedConditions;

let condition = EC.not(EC.elementToBeSelected(element(by.id("ch"))))

browser.wait(condition, 30000)

});

});

and

We can group the expected condition using and method in ExpectedConditions class.

This will throw Error if any of the given condition is not met within given time.

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

element(by.id("checkbox")).click()

element(by.id("populate-text")).click()

let EC = ExpectedConditions;

let condition = EC.elementToBeSelected(element(by.id("ch")))

let condition2 = EC.textToBePresentInElement(element(by.class("target-text")), "Selenium Webdriver")

browser.wait(EC.and(condition,condition), 30000)

});

});

or

or() method combines all the Expected conditions, and it will wait for atleast one conditions to be met.

or() throws Error if non of the conditions are met.

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

element(by.id("checkbox")).click()

element(by.id("populate-text")).click()

let EC = ExpectedConditions;

let condition = EC.elementToBeSelected(element(by.id("ch")))

let condition2 = EC.textToBePresentInElement(element(by.class("target-text")), "Selenium Webdriver")

browser.wait(EC.or(condition,condition), 30000)

});

});

pageLoadTimeout in Protractor

pageLoadTimeout() method in protractor waits for an page to load within given time limit when we use get() or navigate().to() method to load a webpage.

If page is not loaded within given time then protractor throws an Timeout Error, this method is not applicable for a page which loaded by click or someother method.

I guess you donot have doubt, whether to use it before using get() method or after using get() method.

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.manage().timeouts().pageLoadTimeout(45000)

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

});

});

browser.sleep() in Protractor

sleep() method is more useful method when you have nightmare scenarios in your application.

sleep() is static wait, which mean it will wait till given time no matter what happens or no matter who comes.

sleep() method will be useful when you have a condition where element is refresh in the last step of code, but try to avoid sleep method.

sleep() accepts a parameter which is in milli-seconds, below program waits for 5 seconds

import { browser, element, by, ExpectedConditions} from 'protractor'

describe('Protractor Typescript Demo', function() {

browser.ignoreSynchronization = true; // for non-angular websites

it('sendkeys operation', function() {

browser.get('https://chercher.tech/practice/explicit-wait-sample-selenium-webdriver.php');

browser.sleep(5000)

element(by.id("checkbox")).click()

element(by.id("populate-text")).click()

});

});