**Protractor** is end to end (E2E) automated testing framework for Angular and Angular js applications ( to check - Go to inspect element, press esc and enter angular in console, it should return something). It is intended not only to test AngularJS application but also for writing automating regression tests for normal Web Applications as well.

**AngularJS applications** are Web Applications which uses extended HTML's syntax to express web application components. It is mainly used for dynamic web applications. These applications use less and flexible code compared with normal Web Applications.

AngularJS is easy to learn yet powerful enough to help you develop complex single-page web applications. (SPA).

“**single-page application” (or SPA)** is usually used to describe applications that were built for the web. These applications are accessed via a web browser like other websites, but offer more dynamic interactions resembling native mobile and desktop apps.

It simply points to the fact that navigation between different screens of the website is achieved without loading a different webpage in the browser.

The most notable difference between a regular website and an SPA is the reduced amount of page refreshes. the process of rendering pages happens mostly on the client-side.

Protractor is a [Node.js](http://nodejs.org/) program built on top of [WebDriverJS](https://github.com/SeleniumHQ/selenium/wiki/WebDriverJs). Protractor runs tests against your application running in a real browser, interacting with it as a user would.

In Inspect element, there should be tags like ng-app, ng-model, ng-bindunder div tag in HTML**.** (these locators are not identified by selenium code so, protractor on the top of Selenium can handle and controls those attributes in Web Applications.)

By default, Protractor uses the [Jasmine](http://jasmine.github.io/) test framework for its testing interface. Jasmine is a behavior-driven development (BDD) framework for testing JavaScript code

Difference in Protractor from other automation tools is- It has in-build functionality to sync with elements, you do not have to wait for syncing like in webdriver. (so you don’t have to worry about waiting for your test and webpage to sync.)

**Protractor with Selenium-**

Your test scripts send commands to the Selenium Server, which in turn communicates with the browser driver

Protractor is a wrapper (built on the top) around Selenium WebDriver, so it contains every feature that is available in the Selenium WebDriver.

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code server-side.

NPM is package manager for Node.js packages containing repository with all softwares dependent on node.js (just like maven in selenium)

Headless browsers- PhantomJS, HTMLUnit, chromium- without GUI

**Installation-**

Install node.js from <https://nodejs.org/en/> using npm.

**Go to** *C:\Users\name\AppData\Roaming\npm ,* **In cmd as admin, run command-**

npm install –g protractor

If working behind proxy, set proxy first using these 2 commands then install-

npm config set proxy http://proxy.company.com:8080

npm config set https-proxy <http://proxy.company.com:8080>

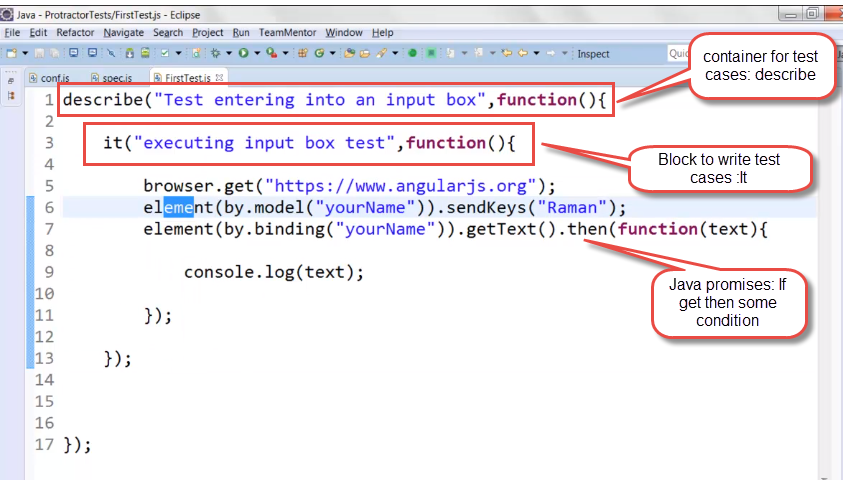
To check if installed, Check protractor version by-

Cmd- protractor –version

To update the selenium webdriver, cmd- *'webdriver-manager update'*

**Syntax-**

**Spec.js**



The describe and it syntax is from the Jasmine framework. browser is a global created by Protractor, which is used for browser-level commands

Describe- For test suite

It- for test case

Use word spec for test file in jasmine

**Docker** is a container technology for Linux that allows a developer to package up an application with all of the parts it needs.

You need-

**Test case spec file**- For test cases with extension .js

**Configuration file** - used to define some parameters which will be passed to Protractor to execute our spec files.

Required parameters in config file-

1. Selenium server details (required)
2. Location of our spec files (required)
3. Browser capabilities for spec files (required)
4. Jasmine node configuration options (required)
5. Jasmine reporter configuration options (optional)

**Sample Conf.js**

exports.config = {

//The address of a running selenium server.

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

//Capabilities to be passed to the webdriver instance.

capabilities: {

'browserName': 'chrome'

},

//Specify the name of the specs files.

specs: ['grid\_spec.js'],

//Options to be passed to Jasmine-node.

jasmineNodeOpts: {

onComplete: null,

isVerbose: false,

showColors: true,

includeStackTrace: true

}

}

Chrome is the default browser.

**Run with Eclipse**

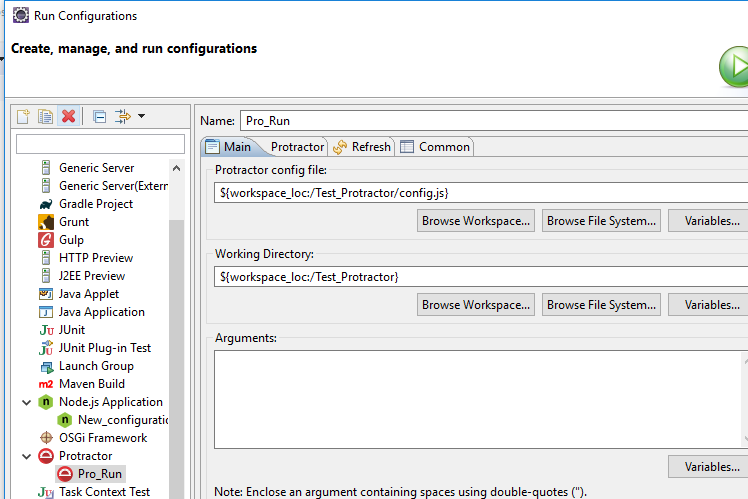
* Install- **Tern Eclipse IDE from Eclipse Marketplace**
* Create new javascript project and Click on convert to tern project (this for helping javascript projects)
* Properties window pop-up>> goto Modules
* Select Protractor and Jasmine and ok

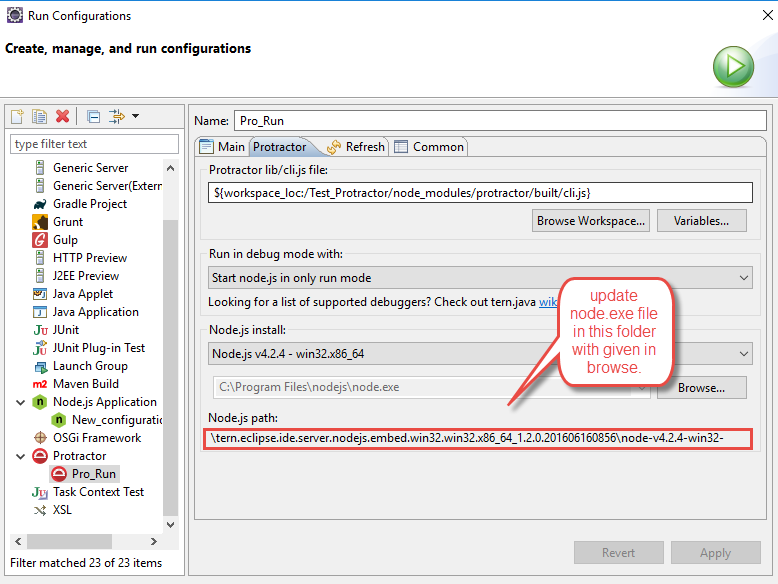
Now, you can write protractor scripts. And autosuggestion will come as u press ctrl + space.

create a bat file with content as "protractor conf.js" and save it under external tools configuration as runner file and run the same.

To run using eclipse- Install <http://oss.opensagres.fr/angularjs-eclipse/1.2.0/>

In Run configuration, update protractor setting to give Main and protractor settings-





For error like- exited with error code 1.., check node.js location in Run configuration.

To bypass **webdriver-manager update** command from cmd due to firewall,

Manually add geckodriver, chrome driver, selenium standalone server, standalone-response and update-config to folder- C:\Users\xxxx\AppData\Roaming\npm\node\_modules\protractor\node\_modules\webdriver-manager\selenium

And change location in update-config.json file to your user.

**Command to stop web driver manager-** Ctrl+C

**Webstorm** – Most popular Javascript IDE (tool for writing code) but its paid

Eclipse IDE for javascript (open source)- to be used for protractor

**Angular Js architecture-**

Components – methods containing property and behaviour

Routers- responsible for navigation

Directives- modify DOM or extend behaviour like autogrow

Services- Data access logging business logic configuration (making http call, logging)

Npm (node package manager)

Bower (browser package manager)- libraries required for client side code to run.

**Locators-**

* By.binding- works on 2 way binding of protractor in which data is bind to value given within <div>{{e.name}} </div>

We will be using element (by.binding(‘name’)) to get its value.

* By.model- works on angular ng-model.

<input ng-model=”name” />

By.model(‘name’);

* By.css- same as selenium.
* By.buttonText

<button>save</button>

By.buttonText(‘save’);

* By.repeater- Find all li that are created in repeater statement

<ul>

<li ng-repeat=”u in users”>{{u.name}}</li>

</ul>

by.repeater(‘u in users’);

* By.options
* element(by.buttonText('Submit')).click();

**Suites** in config.js to group different number of suites based on type of test, test areas and length of test.

To run, create different folders with different spec files and use as follows in config.js-

Suites:

{

Smoke:’./smoke/\*.spec.js,

Full:’./\*\*/\*.spec.js’

}

To run give command,

$ Protractor config.js --suite=smoke ( to run only smoke folder)

To run multiple files

Give path in config file as specs:[‘./\*\*/\*.spec.js] and give files names as create.event.spec.js, event.details.spec.js…

**Using Page Objects:-**

Create new file with js extension.

Identify elements, functions on those elements.

Import po file using

Var createPage=require(’./filename’);

and create object to use.

**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/>

**Javascript**

Javascript string methods- <https://www.w3schools.com/js/js_string_methods.asp>

Javascript is asynchronous

Every javascript step returns promise. Promise means it is promising that step will be executed but didn’t define time or sequence.

Promise resembles state of your step- can be pending, resolved or rejected.

Protractor has build its method (API) which wait for promise to resolve before moving further.

So syncing takes place on its own for 90% methods.

Promise resolving is available for actions to be performed on browser but to retrieve anything from browser, promises do not resolve (syncing do not take place on its own)

Resolve promise by using .then(**function**())-

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

console.log(result);

});

No datatypes require, only user var- javascript automatically identify type of datatype.

Arrays-

Var b=[“test”,”cool”,”4”]; //feeding data initially – string literal declaration

Or

Var c= new Array(); //feeding data at runtime

C[0]=”test”;

C[1]=”cool;

C[2]=”4”;

For string as well

Var name2=new String(“Rahul”); //string object declaration

*To convert String to Number-*

**var** intcount=Number(getcount);

**Writing Css-**

Tagname[attribute=’value’]

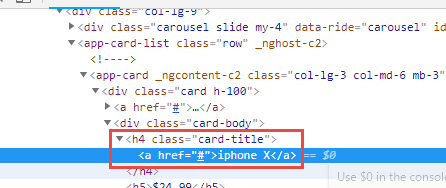
With wildcard-

By.css(“a[ng-href\*=’London/river Island’]”))

* Avoid using xpath in protractor

**CsscontainingText** for giving additional parameter in case identifiers are duplicate.

element(by.cssContainingText("[id='exampleFormControlSelect1'] option","Female")).click()



element(by.css("h4 a")).getText(); //will get iphone X –give parent tag space child tag if single present.

**Assertion**

We use jasmine method-

expect(true).toBe(true);

expect(false).not.toBe(true);

expect(a).not.toBe(null);

expect(a).toEqual(12);

expect(message).toMatch(/bar/);

Jasmine take care of resolving promise so no need to use promise.

Webdriver-manager update –ie (to add internet explorer files in selenium folder)

**Direct connection with browser** using ‘directConnect: true’ or not starting selenium server does not work on firefox and IE. It only work in chrome.

**Chain locators:-** Defining child within parent

expect(element(by.repeater('result in memory')).element(by.css("td:nth-child(3)")).getText()).toBe('10');

**All method:-** to get all values within locator

element(by.model('operator')).all(by.css("option")).getText().then(**function**(text)

{

console.log(text);

});

**Get value of all repeater function:-**

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);

})

// Number of elements.

element.all(locator).count();

// Get by index (starting at 0).

element.all(locator).get(index);

**Window handling-**

Browser.getAllwindowHandles().then(function(handles)

{

Browser.switchTo().window(handles[1]) //for child window ( [0] for parent window)

})

Alert-

browser.switchTo().alert().dismiss();

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

Frame-

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

maximise window-

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

Synchronization- using expectedConditions (method of protractor)

**var** ec=protractor.ExpectedConditions;

browser.wait(ec.invisibilityOf(element(by.css("img[id='loader']"))),8000);

To find visibility of any element-

expect(element.all(by.css("div[class\*='alert-danger']")).count()).toBe(1);

Javascript objects-

Javascript is object based language not object oriented language.

**Object oriented programming languages** follow all concepts belonging to **OOP**.**Object**-**based language** doesn't support all the features of OOPs like Polymorphism and Inheritance **Object**-**based language** has in-built **object** like JavaScript has window **object**. **Object**-**based languages** are JavaScript, VB etc.

**function** car() //like class in java

{

**this**.color='Red';

**this**.model=**function**() //like method in java

{

console.log('Print 2020 model');

}

var a=new car(); //like creating object in java

a.model();

To make it global so other class can access it—

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

For accessing in other class—

**var** obj=require('./Jsobject.js');

obj.model();

Setup and Teardown methods of Jasmine

beforeEach(function() { //same like @beforeTest in java

obj.getURL();

});

afterEach(function() { //same like @AfterTest in java

foo = 0;

});

Prerequisite for all test cases

Config file--

onPrepare :**function**()

{ ------ } ---it will execute before all spec files.

**Package.json**- to connect to npmjs and download all dependencies //same like POM.xml in Maven for downloading all dependencies.

Go to cmd- Go to project folder –**npm init –f**

Package.json gets added in your project folder.

Add dependency-

"dependencies": {

"protractor": *"^5.3.0"*},

Go to cmd, project- **npm install** >> node modules get install automatically in project. ( command to install dependencies)

Now for starting webdriver, updating, running through configurations—we give all commands in scripts in package.json.

"scripts": {

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

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

"run\_protractor": *"./node\_modules/.bin/protractor pack\_config.js"*,

"start":*"npm run webdriver\_update &&npm run webdriver\_start&&npm run run\_protractor"* //it will execute scripts you want with syntax- npm run scriptname

Go to cmd, project- **npm run scriptname (**command to run script of package.json)

***Data driven framework:***

Install jasmine-data-provider by including in package.json

Run command in cmd- npm install (in project folder)

Create a separate data.js file in which store multiple data objects under parent object and export it using module.exports like—

module.exports=

{

Datadriven:

{

firstobject:

{

firstinput:"2",

secondinput:"3",

result:"5"

},

secondobject:

{

firstinput:"3",

secondinput:"3",

result:"9"

}

In spec file, use it using—

**var** using=require('jasmine-data-provider'); //to use using keyword of Jasmine data provider

using(d.datadrive,**function**(data,description) //it will execute number of time sub data objects are present.

{

It(test case)

{…}

}

//data contains actual data, description contains sub object name

***Reporting:***

# Install protractor-jasmine2-html-reporter by including in package.json

Run command in cmd- npm install (in project folder)

Add reporting methods in config.js file—

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

exports.config = {

specs: ['datadrivespec.js'],

onPrepare :**function**()

{

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

jasmine.getEnv().addReporter(

**new** Jasmine2HtmlReporter({

savePath: 'target/screenshots'

})

);

} };

After execution, target folder will be created with screenshots and html report file.

HTML file only reports status of assertion, if there is no assertion it will report it as skipped.

***Suites:***

To run multiple spec files categorised in some keyword-

Give this in config.js

suites:

{

smoke:['run\_through\_pack.js'],

Regression:'datadrivespec.js'

}

Add in package.json-

"Smoke\_suite":*"./node\_modules/.bin/protractor pack\_config.js --suite smoke"*,

Run in command line as- *npm run Smoke\_suite*

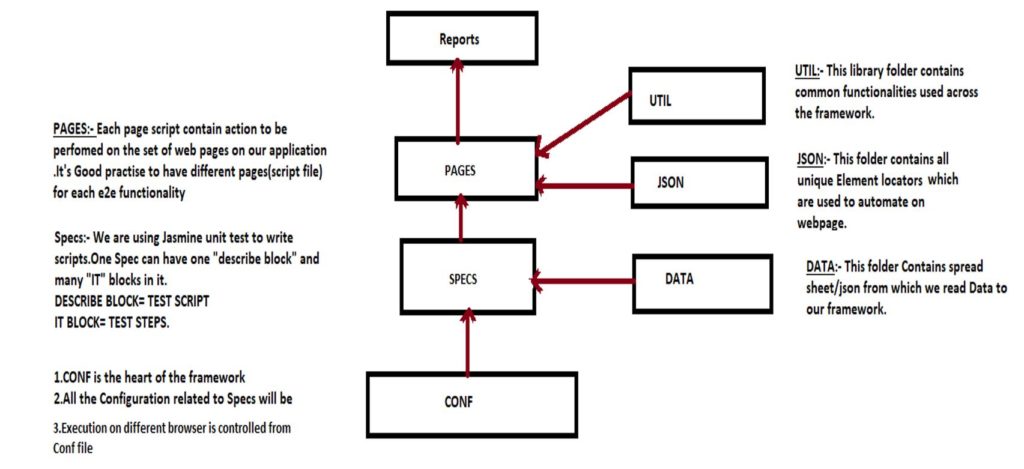
**Protractor project using Jenkins-**

Copy project to home directory folder- C:\Users\chaman.preet\.jenkins\

Create freestyle project and give this in custom workspace- ${JENKINS\_HOME}/Protractor\_package

Build using windows command line—npm install (to install all dependencies in other system)

And then Build using windows command line— npm run run\_protractor (run script)

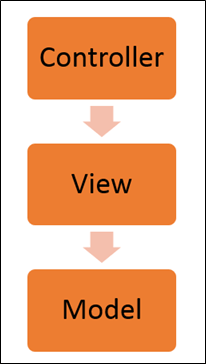


**Angular** is based on TypeScript while **AngularJS** is based on JavaScript.

**AngularJS** is an open source Model-View-Controller framework which is similar to the JavaScript framework

**Angular Js features-**

1. **MVC** – The framework is built on the famous concept of MVC (Model-View-Controller). This design pattern is based on splitting the business logic layer, the data layer, and presentation layer into separate sections. The division into different sections is done so that each one could be managed more easily.
2. **Data Model Binding** – You don't need to write special code to bind data to the HTML controls. This can be done by Angular by just adding a few snippets of code.
3. **Writing less code** – When carrying out DOM manipulation a lot of JavaScript was required to be written to design any application. But with Angular, you will be amazed with the lesser amount of code you need to write for DOM manipulation.
4. **Unit**[Testing](https://www.guru99.com/software-testing.html)ready – The designers at Google not only developed Angular but also developed a testing framework called "Karma" which helps in designing unit tests for AngularJS applications.

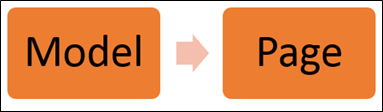
[](https://www.guru99.com/images/AngularJS/010416_0549_AngularJSIn1.png)

Angularjs Architecture Diagram

* The Controller represents the layer that has the business logic.
* Views are used to represent the presentation layer which is provided to the end users
* Models are used to represent your data

## AngularJS Advantages

* Two-way binding – Angular.js keeps the data and presentation layer in sync. Now you don't need to write additional JavaScript code to keep the data in your HTML code and your data later in sync. Angular.js will automatically do this for you. You just need to specify which control is bound to which part of your model.

[](https://www.guru99.com/images/AngularJS/010416_0549_AngularJSIn2.png)

* Routing – Angular can take care of routing which means moving from one view to another. This is the key fundamental of single page applications; wherein you can move to different functionalities in your web application based on user interaction but still stay on the same page.
* It extends HTML by providing its own elements called directives. These directives help in extending the functionality of existing HTML elements to give more power to your web application.

**Tips and tricks**- https://www.slideshare.net/bolshchikov/protractor-tips-tricks?next\_slideshow=1

Command in package.json to run in parallel or concurrently:-

"test1": "npm-run-all -p -r webdriver-start http-server protractor",

-p = Run commands in parallel.

-r = Kill all commands when one of them finishes with zero.

Running npm run test1 will start Selenium driver, start http server (to serve you files) and run protractor tests. Once all tests are finished, it will close the http server and the selenium driver.

Preinstall-

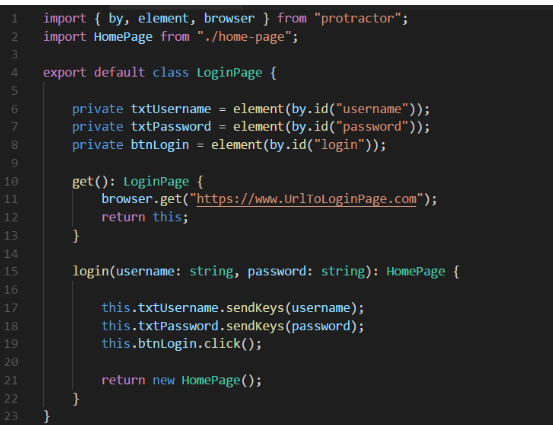
**Command- npm clear cache**, to remove the files from mode\_modules folder.

For Protractor framework-

<https://www.logigear.com/blog/test-automation/15-best-practices-for-building-an-awesome-protractor-framework/#section1>

* Use Page object model
* Place elements and methods in page classes.

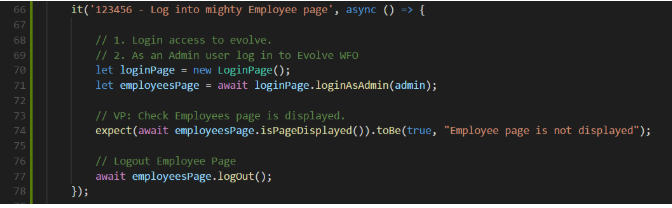
Page elements should be within methods with return statement.



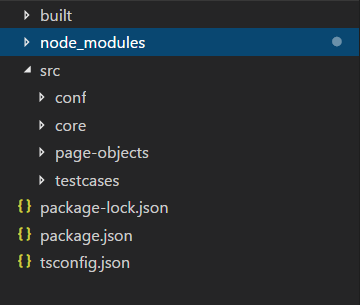
* Convention should be defined-
* File name should be identical to header of the AUT page, lower case separated by dash.

For example, according to the **Login** page from the AUT, the filename will be ***login-page.ts***.

* Name of the page object class should be in CamelCase with first letter in uppercase. For example, **LoginPage**.
* Page object’s methods should be in CamelCase with first letter in lowercase. Here are some examples: **login(), loginExpectingError(), gotoCustomerPage(), registerCustomer()**.
* Methods of page objects should return a page object.
* The access modifiers of all page objects’ elements should be private, on the other hand, in the base class or superclass, they should be protected**.**
* Test case should only call page object’s methods.

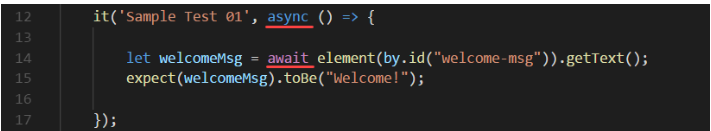


* Do not place assertions inside page object classes- Assertions should only be placed in test cases
* Project structure should be effective.



* Only push necessary things to code repository- the only files that we should push to the code repository are the [TypeScript](https://www.typescriptlang.org/docs/home.html" \t "_blank) files (inside the *src* folder) and all the project configuration files (.json). Thebuilt*and*node\_modules*folders are* ***not*** *necessary to push to the repository.*
* Use async and await for handling javascript asynchronous behaviour-

With await, use of async is mandatory. Await will wait for command to finish before moving to next command.



**Multicapabilities**- To run multiple browsers in parallel-

multiCapabilities: [

{'browserName': 'chrome'},

{'browserName': 'firefox'},

],

To pass multiple arguments to chrome options-

capabilities: {   'browserName': 'chrome',

'chromeOptions': {

'args': [ '--start-maximized', 'disable-infobars']   } },

Arguments can be-

--**start-maximized'** : - Starts the browser maximized, regardless of any previous settings

--**headless** :- Run chrome in headless mode, i.e., without a UI or display server dependencies.

--**ignore-certificate-errors :** - Ignore If there are any certification errors while running tests.

--**disable-infobars**- will not display any notification bar at top like- "Chrome is being controlled by automated test software".

**To execute your scripts on same browser with multiple instances**.-

-add two capabilities shardTestFiles and maxInstances in the Capabilities block

**shardTestFiles** - To enable sharing of tests at the spec level, we must configure shardTestFiles flag as true within the capabilities.

**maxInstances -** The maxInstances number represents the maximum number of browser windows that Protractor should create in parallel

If we set maxInstances to 2, a test suite of 20 tests would result in Protractor creating 2 Chrome instances with each instance running 10 tests.

This will help to drastically reduce the overall test execution time and speed up the continuous delivery process.

**Jasmine keywords-**

<https://jasmine.github.io/2.4/introduction.html>