**Protractor Guidelines:**

***What is Protractor?***

It is an automation suite mainly used for automating angular based applications. It is built on top of Selenium WebDriver.

It can automate both Angular as well as Non Angular applications as it is a wrapper embedded on selenium WebDriver.

**WebDriver + Angular Support = Protractor**

Developed by Google – Angular Team. It is the most stable framework to automate angular applications.

Protractor opens up chrome browser by default

***Language Used for Protractor?***

Protractor is a **node.js program** built on top of WebDriver.js and not Java. Hence it uses JavaScript as core language to develop tests.

Other supporting languages: TypeScript, Python etc.

JavaScript is asynchronous

***Installation of Protractor?***

**Step 1: Download node.js**

**Step 2: Set node.js path in system variables**

**Step 3: Verify whether node.js is installed correctly by using**

**>>node –v in cmd prompt**

Importance of **NPM (Node Package Manager)** – To download the protractor package from node repository. It acts as a medium to download all necessary packages from node repository for the user to make use of it.

Eg. MAVEN – uses dependencies to download all associated packages from maven repository.

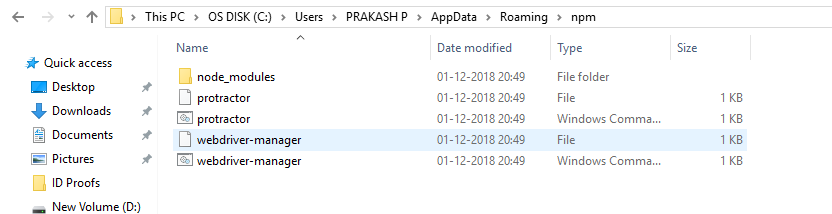
**Step 4: When node.js is downloaded, NPM comes in built with it.**

Open cmd prompt and type

**>>npm –v**

**If error occurred, go to node\_modules\npm\bin and copy the path. Add it to the system variables.**

Generally, npm will be installed **in c:\users\ComputerName\ AppData\Roaming\npm\**



**Step 5: Open cmd prompt and type**

**>>npm install –g protractor**

where -g represents installing suitable packages globally

You may get an error like:

“unexpected end of json input while parsing near”

If so, type the below command in cmd prompt ---

**>>npm cache clean –force**

After successful installation go to the path and check for the protractor package being installed.

**Step 6: Verify the version number**

**>>Protractor - - version**

**Step 7: If we want to update the webdriver manager, then type**

**>>webdriver-manager update (only if needed)**

If you get an error, try the below command,

**>>npm install – g webdriver-manager** (Only if webdriver-manager is not installed properly)

In **case** of proxy setup, try with:

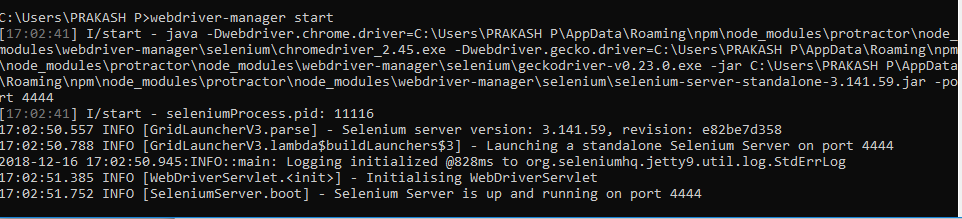
**>>webdriver-manager --proxy update**

**Step 8: To start selenium server**

This server helps you to connect the selenium commands to the browser. It a proxy layer to get communicated.

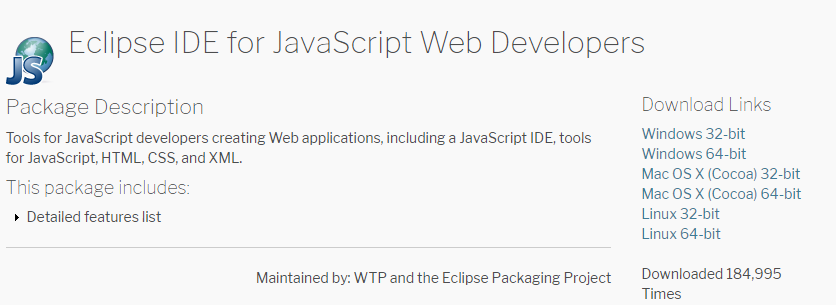
Your Protractor test will send requests to this server to control a local browser.

**>>webdriver-manager start**



***IDE for Protractor Test***

Install **eclipse IDE for JavaScript Web developers**



There are other tools in the market and eclipse is widely used among them.

**Other Tools: WebStrom, Visual Studio etc**

***Introduction to Jasmine Framework***

Documentation can be found here:

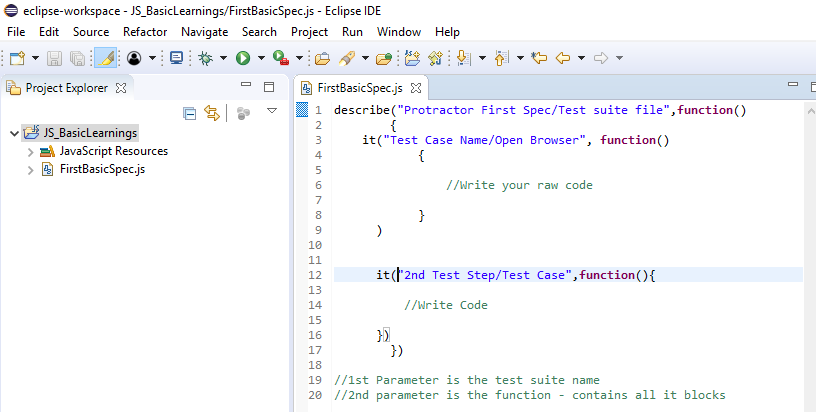
<https://jasmine.github.io/tutorials/your_first_suite>

It is a JavaScript Framework which is mainly used for JavaScript based development.

Important Terminologies:

1. **Describe keyword** - It is a test suite 🡪 collection of test cases
2. **It block –>** represents Test case/Test Steps
3. **Spec (Jasmine Terminology) –>** Test file ( eg. Java File/Class File)
4. **Configuration File –>** Execution happens only from configuration file level (eg. Testng.xml file)
5. \*\*Every spec file must have describe and it block

**Sample Syntax of describe and it block:**



**Importance of Configuration file:**

\*\*We need to pass selenium server address. Protractor should know where to send the request and hence we need to define the selenium address

e.g

exports.config = {

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

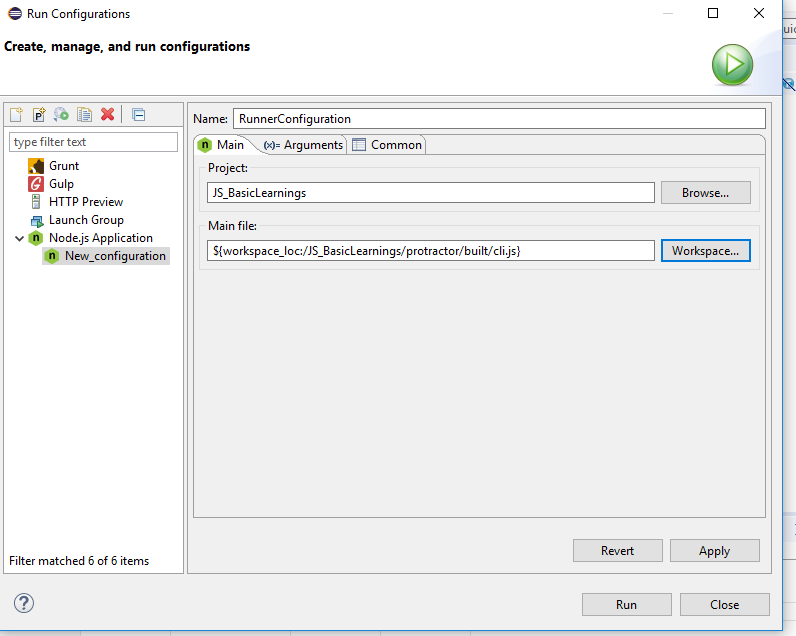
specs: ['todo-spec.js']

};

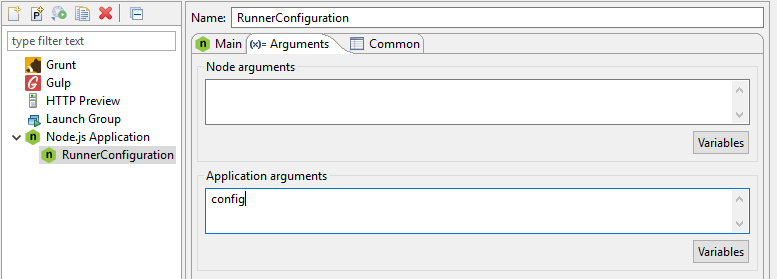
Specify all the spec file you want to run

**>> Setting up Run Configuration**

Step 1: Copy paste the protractor folder into the project folder to access **cli.js file**

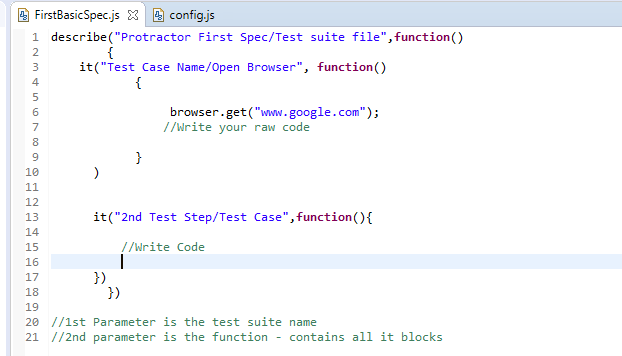


Step 2: Place the configuration file you wanted to run in arguments tab



Step 3: Type in your code

**Note: By default, protractor runs in chrome browser**



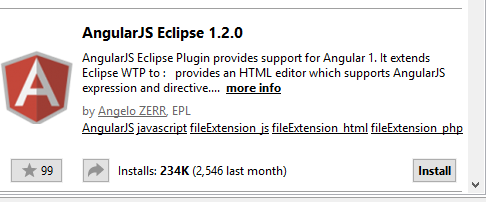
Step 4: Run the TestRunner configuration – Run Configuration

**3 mandatory fields are required.**

1. Project Name
2. Main file – cli.js
3. Add configuration file inside Arguments menu

***>>Install Angular JS plugin***

----Install from market place



>>***Convert the project to TERN project***

This is mainly to turn on auto suggestions when you type in your commands

**Right click on project ->configure -> convert to Tern Project**

**Why JavaScript is asynchronous?**

Refer the below link:

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

***Concept of Promise***

Promise resembles state of your step.

Types of states:

1. **Pending**
2. **Resolved**
3. **Rejected**

\*\*In Synchronous, unless or until the steps is resolved, it will not move to the next line.

It will move to next step only when promise returns resolved or rejected

\*\*In asynchronous, it will not look for resolved or rejected. It moves to next step even when promise is pending

**Note: 90% of the protractor API will not move to next step until promise is resolved**

If you want to act on browser, then protractor takes care of the promise.

Whereas, if you want to retrieve anything from browser, that functionality support (eg. getTitle(), getText(),getAttribute() etc) is not provided.

It make the remaining 10% to behave synchronously then make use of the keyword called “**then**” after each command

**Example:**

browser.get(“google.com” ).then(function(){

//Write the code that has to execute after the above said code

})

***Links about promise:***

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/>

***Global variables of Protractor***

Protractor exports these global variables to your spec (test) file:

1. **browser** - A wrapper around an instance of WebDriver, used for navigation and page-wide information. The browser.get method loads a page. Protractor expects Angular to be present on a page, so it will throw an error if the page it is attempting to load does not contain the Angular library. (If you need to interact with a non-Angular page, you may access the wrapped webdriver instance directly with browser.driver).

2. **element** - A helper function for finding and interacting with DOM elements on the page you are testing. The element function searches for an element on the page. It requires one parameter, a locator strategy for locating the element **–(Element Basis)**

3. **by** - A collection of element locator strategies. For example, elements can be found by CSS selector, by ID, or by the attribute they are bound to with ng-model. See [Using Locators](https://www.protractortest.org/#/locators).

4. **protractor** - The Protractor namespace which wraps the WebDriver namespace. Contains static variables and classes, such as protractor.Key which enumerates the codes for special keyboard signals.

***Basic/Common Locators in protractor***

The most common locators are:

// 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')

**Protractor Locators:**

| **Function** | **Description** |
| --- | --- |
| [addLocator](http://www.protractortest.org/#/api?view=ProtractorBy.prototype.addLocator) | Add a locator to this instance of ProtractorBy. |
| [binding](http://www.protractortest.org/#/api?view=ProtractorBy.prototype.binding) | Find an element by text binding. |
| [exactBinding](http://www.protractortest.org/#/api?view=ProtractorBy.prototype.exactBinding) | Find an element by exact binding. |
| [model](http://www.protractortest.org/#/api?view=ProtractorBy.prototype.model) | Find an element by ng-model expression. |
| [buttonText](http://www.protractortest.org/#/api?view=ProtractorBy.prototype.buttonText) | Find a button by text. |
| [partialButtonText](http://www.protractortest.org/#/api?view=ProtractorBy.prototype.partialButtonText) | Find a button by partial text. |
| [repeater](http://www.protractortest.org/#/api?view=ProtractorBy.prototype.repeater) | Find elements inside an ng-repeat. |
| [exactRepeater](http://www.protractortest.org/#/api?view=ProtractorBy.prototype.exactRepeater) | Find an element by exact repeater. |
| [cssContainingText](http://www.protractortest.org/#/api?view=ProtractorBy.prototype.cssContainingText) | Find elements by CSS which contain a certain string. |
| [options](http://www.protractortest.org/#/api?view=ProtractorBy.prototype.options) | Find an element by ng-options expression. |
| [deepCss](http://www.protractortest.org/#/api?view=ProtractorBy.prototype.deepCss) | Find an element by css selector within the Shadow DOM. |

**Extends webdriver locators:**

| **Function** | **Description** |
| --- | --- |
| [className](http://www.protractortest.org/#/api?view=webdriver.By.className) | Locates elements that have a specific class name. |
| [css](http://www.protractortest.org/#/api?view=webdriver.By.css) | Locates elements using a CSS selector. |
| [id](http://www.protractortest.org/#/api?view=webdriver.By.id) | Locates an element by its ID. |
| [linkText](http://www.protractortest.org/#/api?view=webdriver.By.linkText) | Locates link elements whose [visible text](http://www.protractortest.org/#/api?view=webdriver.WebElement.prototype.getText) matches the given string. |
| [js](http://www.protractortest.org/#/api?view=webdriver.By.js) | Locates an elements by evaluating a JavaScript expression, which may be either a function or a string. |
| [name](http://www.protractortest.org/#/api?view=webdriver.By.name) | Locates elements whose name attribute has the given value. |
| [partialLinkText](http://www.protractortest.org/#/api?view=webdriver.By.partialLinkText) | Locates link elements whose [visible text](http://www.protractortest.org/#/api?view=webdriver.WebElement.prototype.getText) contains the given substring. |
| [tagName](http://www.protractortest.org/#/api?view=webdriver.By.tagName) | Locates elements with a given tag name. |
| [xpath](http://www.protractortest.org/#/api?view=webdriver.By.xpath) | Locates elements matching a XPath selector. |

**Working with locators and usage:**

Protractor exports a global function element, which takes a Locator and will return an **ElementFinder**. This function finds a single element - if you need to manipulate multiple elements, use the **element.all** function.

The **ElementFinder** has a set of action methods, such as click(), getText(), and sendKeys. These are the core way to interact with an element and get information back from it.

**Syntax and Guidelines:**

element(by.model(“model-name”))

Css --- tagname[attribute=’value’]

//Resolving promise using then() – Here we get value from webpage

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

//resolving promise by using then

console.log(outputText)

})

***Jasmine Assertions***

Refer: <https://jasmine.github.io/2.0/introduction.html>

Example:

Here, value of element(by.css("h2[class='ng-binding']")).getText()) is 8(Actual Value) and we are going verify with toBe function with Expected value

>> expect(element(by.css("h2[class='ng-binding']")).getText()).toBe(“8”)

Expect, toBe are jasmine functions

Note: Jasmine Takes care of promises internally. If you want to make use of the attribute value from webpage then we have to take care of promises explicitly.

**Importance of chain locators:**

#### >>repeater locator(will generally contain child objects) –especially webtable concepts

element(by.repeater("result in memory")).element(by.css("td:nth-child(2)"));

Yellow highlighted is parent node and grey highlighted is child node.

**td:nth-child(2)** = when a tag contains duplicate child attributes with same value then we use the index to find the exact element

**Syntax : element(by.css(attributename:nth-child(location)))**

***>>element.all()***

Gives you all the connected elements.

**Difference between then() and each()?**

then() is mainly used to resolve the promise for single element.

each() is mainly used to resolve the promise for multiple elements

***JavaScript Basics (Refer Tutorials)***

JavaScript is dynamic type language, means you don’t need to specify the type of the variable because it is dynamically used by JavaScript engine. You need to specify var here to specify return type/data type.

***>> DataTypes***

Use **var** which holds all data types – int,String,Boolean float etc

***>>Conditional statement***

If else,

if --else if-- else if—else

***>>Loops***

for(), while and do while

***>>Array*** – collection of objects or values.

Var arrayValue = [“Prakash”, “Test Engineer”] //used when u know the size and value of an array

Or

Var arrayValue = new array(); //used when we get values from webpage which is dynamic

arrayValue[0]= “Prakash”

arrayValue[1]= “Test Engineer”

To determine the size of array**: length**

***>>String***

**Slice** is same as substring. – to get part of a string.

Slice(0,5) – starts from 0th index and ends on 5th index

**Note: starting index is inclusive whereas ending index is exclusive**