What is protractor:

Protractor is node.js program written on top of webdrive.js to test mainly for Angular.js applications. So it is like wrapper on top of webdriver.js.

Applications written in angular has some extra elements which is difficult to selenium to locate these elements like ng:model, ng:repeater, ng:binding. ng:controller etc. In protractor we can find Angular elements directly like:

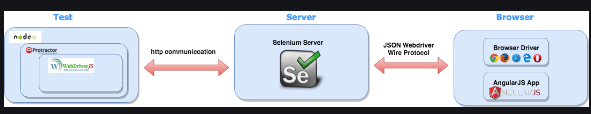
element(by.model(“”)) or element(by.repeater(“”))

We can test non-angular applications as well by protractor.

There are also sync issues while writing test in Selenium for Angular application, Protractor manages all sync related issues.

Angular application are asynchronous application and Protract manages all async call backs. In normal web application for any event entire page loads which is scnchronus behavior and selenium can manage this. But in Angular application there is no page load happens, only data is changed at that particular section at client side, Protractor manages all these call backs and there is no sync issue between elements. In selenium we face sync issues while doing automation for Angular applications and we need to put some wait conditions. Protractor automatically manage all this behavior.

Protractor workflow:



Test scripts send commands to the Selenium server.

Selenium Server interacts with browser driver.

Browser driver interact with application on the real browser.

What is Jasmine:

It is BDD framework to test any java script application, while we do automation testing with Protractor we use Jasmine as default BDD framework. Like Junit is default automation framework for Selenium, Jasmine is for Protractor. It is downloaded default with Protractor. It is open source.

It has generic methods which are used to write test case like in Cucumber we have when then and, Jasmine has describe, it and except keywords which are mostly used. There are other methods as well like afterall, aftereach, beforeall, beforeeach etc.

Every test case should have at least one describe and it method.

describe(“Testing Search Engine”,function()){

it(“Open browser and search”,function()){

browser.get(<http://google.com>);

element(by.name(“q”);

}

}

Working with Protractor in Eclipse:

Download Tern plugin for eclipse from marketplace.

Create new java project in eclipse and covert project to Tern project and select node.js, Jasmine and protractor and convert.

Tern plugin gives all intellisence while writing code.

Create one conf.js file which will have all information about browser, testfile, seleniumAddress and other capabilities.

exports.config = {

capabilities: {

'browserName': 'chrome',

'ignoreProtectedModeSettings': **true**

},

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

specs: ['../Day1/Test1.js']

};

Now create test.js where you will have all your tests like:

describe("Open Google", **function**() {

it("Search in Google", **function**() {

browser.get("http://www.google.com");

browser.sleep(3000);

element(by.name("q")).sendKeys("Protractor");

})

})

**Interview questions**

1. **What are the prerequisites required to run Protractor?**

To run Protractor, we need to have Selenium WebDriver and Node.js installed. We can download Protractor package using npm.

1. **How Protractor, Selenium Server, and Selenium WebDriver work together?**

**Defined above.**

**What are the test frameworks supported by Protractor?**

Protractor supports two behavior driven development (BDD) test frameworks such as Jasmine & Mocha.

**What is Typescript?**

TypeScript is a super set of JavaScript compiled to JavaScript. TypeScript is a strongly typed, object oriented, compiled language developed and maintained by Microsoft. TypeScript may be used to develop JavaScript applications for both client-side and server-side (Node.js) execution

**What is a conf file in Protractor?**

The configuration file tells Protractor how to set up the Selenium Server, which tests to run, how to set up the browsers, and which test framework to use. The configuration file can also include one or more global settings. The config file provides explanations for all of the Protractor configuration options. Default settings include the standalone Selenium Server, the Chrome browser, and the Jasmine test framework.

**11. What is a spec file in Protractor?**

Spec file is the one where we write actual test code. It contains the logic and locators to interact with an application.

**How to exclude a spec file in Protractor**?

You can exclude the spec’s by adding them in the exclude tag in your conf.js file. Say we want to exclude test name “myTest.js”  
exclude: [myTest.js]

**What is the use of directConnect in Protractor?**

Protractor can test directly against Chrome and Firefox without using a Selenium Server. To use this, in your config file set directConnect: true.  
directConnect: true – Your test script communicates directly Chrome Driver or Firefox Driver, bypassing any Selenium Server. If this is true, settings for seleniumAddress and seleniumServerJar will be ignored. If you attempt to use a browser other than Chrome or Firefox an error will be thrown.  
The advantage of directly connecting to browser drivers is that your test scripts may start up and run faster.

**What are the locators in Protractor?**

Protractor supports all the element location strategies given by Selenium and it also has unique set of locators particularly to identify elements based on AngularJS attributes.

Selenium locators:

by.className  
by.css  
by.id  
by.linkText  
by.name  
by.partialLinkText  
by.tagName  
by.xpath

Angular Specific Locators

by.binding  
by.exactBinding  
by.model  
by.repeater  
by.exactRepeater  
by.options

**16. How do you check the status of a webdriver manager and how do you update webdriver manager?**

To check the status of webdriver manager, run the following in your command prompt

**webdriver-manager status**

To update webdriver manager, run the following code in your command prompt

***webdriver-manager update***

**19. What is browser.refresh in Protractor?**

browser.refresh makes a full reload of page.

**What is the difference between “GET” and “NAVIGATE in Protractor?**

**Get:**  
Get method is used to navigate to the given destination.  
browser.get(“https://www.softwaretestingmaterial.com”);

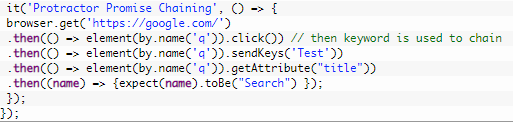
browser.navigate()

if we want to load a previous page or next page in a browser we can use back and forward.

browser.navigate().back();  
browser.navigate().forward();

What is promise in Protractor or use of then?

A promise in protractor confirms that next action will be performed only when last action is done, it manages all sync issues. For ex:



1. *browser.get(‘https://google.com/’) executes and then it waits for page to load*
2. *once the page is loaded it executes element(by.name(‘q’)).click() then it ensures action is performed*
3. *Then it executes the line element(by.name(‘q’)).sendKeys(‘Test’) once this action is done next line and continues…*
4. *.then(() => element(by.name(‘q’)).getAttribute(“title”)) .then((name) => { expect(name).toBe(“Search”) }); watch this line we are getting attribute once this operation is successful it returns value that is resolved now the name will hold the actual attribute value.*

So by using then, there is no need for any sleep or wait as it will manage all syncing issues.

Protractor automatically manage sync issues but still if you are facing any you can use then method.

**How do you verify tooltip text using Protractor?**

**We can read title attribute which is nothing but tooltip like:**

**23. What are the mouse actions that can be performed using Protractor?**

1. click()  
2. doubleClick()  
3. contextClick()  
4. mouseUp()  
5. mouseDown(element)  
6. mouseMove(element)  
7. mouseMove(xOffset, yOffset)  
8.dragAndDrop(element1, element2)

**25. How to get**text**from a textbox in Protractor?**

By using getAttribute method by passing argument as value. getAttribute() method returns a promise which contains String.

element(**by**.id(‘username’)).getText();

**26. How do you verify whether an element exists on a webpage using Protractor?**

To determine whether an element exists on a webpage with isPresent function

// Element exists  
expect(element(by.binding(‘person.name’)).isPresent()).toBe(true);

// Element not present  
expect(element(by.binding(‘notPresent’)).isPresent()).toBe(false);

**27. What are the methods present in Alerts Class?**

Methods present in Alerts Class are  
1. accept()  
2. dismiss()  
3. getText()  
4. sendKeys()

**28. How to set value to prompt Alert in Protractor**?

By using sendKeys() method of Alerts class, we set value to the prompt Alert in Protractor.

browser.switchTo().alert().sendKeys("softwaretestingmaterial")

**29. How to accept Alert in Protractor**?

By using accept() method of Alerts class, we accept Alerts in Protractor.

|  |
| --- |
| Note: We have to switch the control to the alert using switchTo().alert() before accepting the alert browser.switchTo().alert().accept() |

**30. How to cancel Alert in Protractor?**

By using dismiss() method of Alerts class, we cancel or close alerts in Protractor.

Note: We have to switch the control to the alert using switchTo().alert() before cancel the alert.

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

**31. How do you stop Page Loading in Protractor when an element is loaded?**

By using Key.ESC to body element in Protractor.

browser.actions().sendKeys(protractor.Key.ESC).perform();

**32. How to verify whether an element is displayed on a webpage or not?**

To verify whether an element is displayed on a webpage or not, we use isDisplayed() method in Protractor. isDisplayed() method returns a Boolean value based on the state of the element in a webpage. It returns true if the element is displayed and false if the element is not displayed.

**33. How to verify whether an element is enabled or not?**

To verify whether an element is enabled or not, we use isEnabled() method in Protractor. isEnabled() method returns a Boolean value based on the state of the element. It returns true if the element is enabled and false if the element is not enabled.

**34. How to verify whether a dropdown is selected or not in angular applications?**

To verify whether a dropdown list is selected or not in angular applications using Protractor, we use isSelected() method. isSelected() method returns Boolean value. It returns true if the element is selected and false if the element is not selected.

**35. How do you find Angular on your page?**

Open chrome devtools or firefox and find if ‘angular’ is defined. For AngularJS apps, the ‘angular’ variable is expected to be available in the global context.

**36. What are the browsers supported by Protractor?**

Protractor supports the two latest major versions of Chrome, Firefox, Safari, and IE.

**What Are Element Finder And Element Array Finder?**

To write end to end tests in any test automation, we first identify elements and then interact with them using locators.

In selenium we use findElement to find element like driver.findElement, in protractor we use element() method to find an element.

ElementFinder is used to select an element and do operations on it. ElementFinder are like webelement in selenium, but webelement doesn't wait for angular elements to load. Unlike webelement ElementFinder waits for angular elements to wait.

ElementFinder will not attempt to find the element until the actions such as click (), getText (), or sendKeys has been called on it.

element(**by**.className('submit')).click();

**getText()**If the element has any text in it, then getText( ) function is used to get that text from the element.

element(**by**.id(‘username’)).getText();

**sendKeys()** If you want to send text to the input field, we use sendKeys( ) function.

element(**by**.id('username')).sendKeys(key1);

**ElementArray (element.all)**:

ElementArrayFinder is used to get an array/list of elements and perform operations on them. You can call an array of elements and filter them by condition to return a new element.

We can use index to filter and get the required element from array of elements.  
Example:

<**ul** class="numbers">

  <**li**>One</**li**>

  <**li**>Two</**li**>

  <**li**>Three</**li**>

</**ul**>

From the above html code we have to take second element and perform operation on it using index :

var numbers = element.all(**by**.className(numbers));

**console**.log(numbers[1].getText());

And now we select an element using then() function :

element.all(by.className(numbers)).then(**function**(numbers){

    consol.log(numbers[1].getText());

});

**Few methods of element.all**:

**First()** : This method is used for selecting the first element from the array of elements.

var elem = element.all(**by**.className('numbers')).first();

**console**.log(elem.getText());

**Last()** : This method is used for selecting the last element from the array of elements.

var elem = element.all(**by**.className('numbers')).last();

**console**.log(elem.getText());

**Count()** : This method is used to get the count/number of elements present in the array elements.  
**Example**:

element.all(by.css('.numbers li')).then(**function**(numbers){

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

});

**get()** : To get an element within the ElementArrayFinder by index and index starts from 0. get(index) to return a single ElementFinder at specified 'index'

Example :

**let** list = element.all(by.css('.numbers li'));

expect(list.**get**(0).getText()).toBe('One');

expect(list.**get**(2).getText()).toBe('Three');

What is **protractor chaining locators or how to find child elements?**

To find an element in another element we use chaining of locators. First we need to find the parent element then using the parent element we find the child element and perform operation on it.



First we are selecting element with id and assigning it as parent. Then using parent, we are selecting another element by className and named it child. Now we have an child element which is chained to parent element. By using child element we are selecting another element chained to it by id and performing operation by sending data into it. This whole selection of elements is done using chaining of locators.

**Question 7. How To Assert Something In Protractor?**

**Answer :**

It depends on the assertion framework you’re using. In general, most of the e2e implementation done is based out of the default Jasmine 2.0, which provides the assertion in this format

expect (something).toEqual (someotherthing).

expect(latestResult.getText()).toEqual('3');

expect(browser.getTitle()).toEqual('Super Calculator');

**Question 10. I Do Not Want To Start The Selenium Server Every Time. How Can I Overcome This?**

**Answer :**

In order to not have to do this, you need to set the direct Connect flag to true in your conf.js file.

**Question 11. What Does Ignore Synchronization Do?**

**Answer :**

To use Protractor on web pages which doesn't have Angular, we have to suppress the default Protractor behaviour of waiting for Angular components.

Setting the browser.ignoreSynchronization to true means that we’re telling Protractor to not wait for the Angular promises to be resolved.

browser.ignoreSynchronization=true;

**How to handle reporting in Protractor?**

Similar to Selenium, Protractor does not comes with an inbuilt reporting tool of itself. Although this is counted as one of the draw-backs of frameworks like Selenium, Protractor, but it also gives a chance for the person using this tool to customise the reporting tool as per their choice.

Since Protractor is a Javascript e2e testing library and works with a host of default frameworks like jasmine or mocha or cucumber, so the reporting too is dependent on the framework you’re using. There are a lot of reporting tools available in the npm repository that can be customised and used in your e2e frameworks.

Some of the Protractor Reporting tools are as follows.

1. Jasmine Reports like Allure
2. Protractor Beautiful Reporter

We have used protractor beautiful reporter

We know that reporting in software testing plays vital role. This Protractor Beautiful Reporter generate a beautiful report for your protractor tests.

First, let’s see the features of Protractor Beautiful Reporter

Features of Protractor Beautiful Reporter

* Browser’s Logs
* Log Stack Trace
* Screenshot UI
* Filters (can display only Passed/Failed/Pending/Has Browser Logs)
* Details (Browser/Session ID/OS)
* Duration time for test cases (only Jasmine2)

Steps to generate Protractor Beautiful Reporter:

**Step 1: Download beautiful report from repository**

npm install protractor-beautiful-reporter

**Step 2: Open protractor conf.js file**

var HtmlReporter = require('protractor-beautiful-reporter');

// Add a screenshot reporter and store screenshots to `/Reports/screenshots`:

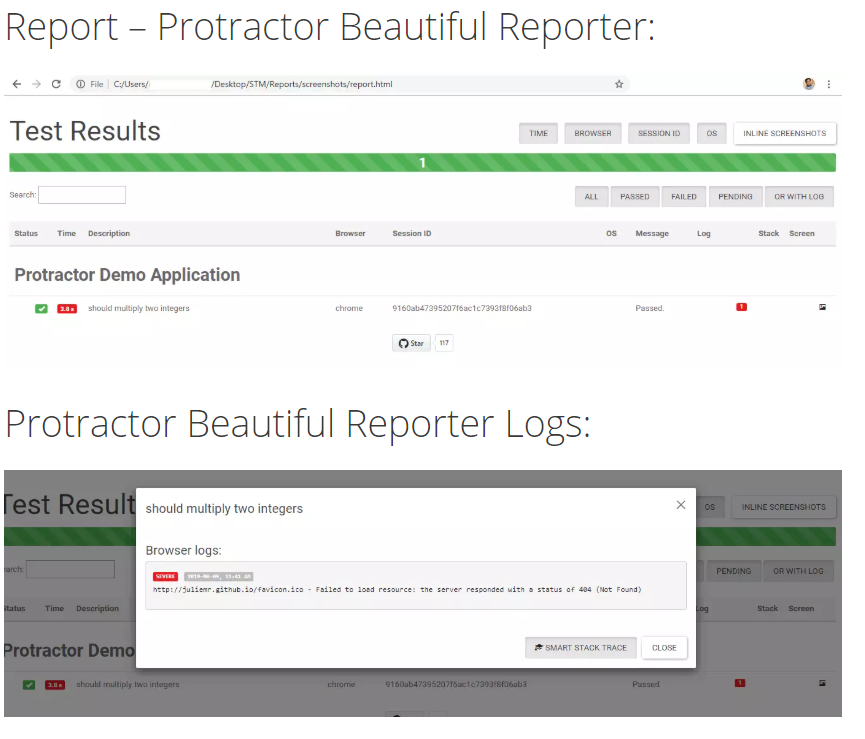
jasmine.getEnv().addReporter(new HtmlReporter({

         baseDirectory: 'Reports/screenshots'

      }).getJasmine2Reporter());

   }

After execution, the screenshot reporter will generate JSON and PNG files for each test. Open the file name report.html which is under ***Report/screenshots*** folder to view the report.



**Question 16. What Is The Default Time For A Protractor With Jasmine Spec To Fail And Can I Change It?**

**Answer :**

By default, any spec in Protractor will fail after a period of 30 seconds because that is the default time out for a Jasmine spec to fail.

Yes, this timeout period can be configured to be increased or decreased.  In order to change the default timeout period for all specs, you need to add

JasmineNodeOpts: {defaultTimeoutInterval: timeout in millis}

To you conf.js file, where the second option is the time period that you want to set.

**Question 17. How To Run Multiple Specs In Protractor?**

**Answer :**

In order to run multiple spec files in Protractor, you just need to mention them in the spec flag in an array.

For eg, let’s say I have two different spec files test1\_spec.js and test2\_spec.js, so I can do this

Specs: ['. /test/test1\_spec.js','./test/test2\_spec.js']

This will make Protractor run these multiple spec files.

**Question 18. Select A Checkbox Using Protractor?**

you got to make sure that the checkbox is selected. To verify it we need to use the is Selected () method and get check whether it is really selected.

**Question 26. Write A Code To Wait For An Alert To Appear?**

**Answer :**

Waiting for an alert to appear on a page can be performed using explicit wait in protractor.

Browser. Wait (ExpectedConditions.alertIsPresent (), 30000)

**Question 28. Can Protractor Handle Windows Based Pop Up?**

**Answer :**

Protractor is a web automation testing tool which supports only web application testing. Therefore, windows pop up cannot be handled using Protractor.

**Question 30. What Are The Advantages Of Automation Framework In Protractor?**

**Answer :**

* Re-usability of code
* Maximum coverage
* Recovery scenario
* Low cost maintenance
* Minimal manual intervention
* Easy Reporting
* Logging for debugging
* Easy Coding

**Question 32. What Is The Order Of Fastest Browser Implementation For Protractor?**

**Answer :**

HTML Unit Driver is the fastest browser implementation as it does not involves interaction with a browser; this is followed by Firefox driver and then IE driver which is slower than FF driver and runs only on Windows.

**Question 34. What Are The Web Pages Elements In Web Applications?**

**Answer :**

Link Button Image, Image Link, Image Button Text box Edit Box Text Area Check box Radio Button Drop down box List box Combo box Web table /HTML table Frame.

**Question 35. What Is The Difference Between Browsers. Close () And Browser. Quit () Command?**

**Answer :**

**Close ():** Protractor close () method closes the web browser window that the user is currently working on or we can also say the window that is being currently accessed by the Web Driver. The command neither requires any parameter nor does is return any value.

**Quite ():** Unlike close () method, quit() method closes down all the windows that the program has opened. Same as close () method, the command neither requires any parameter nor does is return any value.

**How do you run headless mode browsers in Protractor ?**

Headless browser doesn’t require a graphics driver at all, They just get commands and execute them directly in memory. To achieve this, you need to specify headless in cong.js under capabilities.  
  
For chrome

**capabilities: {**

**browserName: 'chrome',**

**chromeOptions: {**

**args: [ "--headless", "--disable-gpu", "--window-size=800,600" ]**

**}**

**}**

**Write a code to wait for a particular element to be visible on a page using protractor ?**

Visibility means that the element is not only displayed but also has a height and width that is greater than 0. You can use visibilityOf function to check the visibility of the element

**var condition = ExpectedConditions.visibilityOf(element(by.id("hidden")))**

**browser.wait(condition, 30000)**

**Is there a way to do drag and drop?**

You can perform drag and drop using the browser.action() in protractor

**// perform drag and drop**

**browser.actions().dragAndDrop(**

**element(by.id("drag1")),**

**element(by.id("div2"))**

**).perform();**

**How to fetch the current page URL in Protractor ?**

Using getCurrentURL() command we can fetch the current page URL-

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

**console.log("Web page url is : " +url )**

**})**

**How to handle alerts and confirmation boxes.?**

We can switch to the alert using switchTo().alert() method in protractor

**let abc:Alert = browser.switchTo().alert();**

**// typescript assigns the type dynamically, so don't have to provide type explicitly**

**let abc = browser.switchTo().alert();**

**abc.accept()**

**abc.dismiss()**

**abc.getText()**

**abc.sendKeys()**

Difference between toBe(true), toBeTrue() and toBeTruthy()

I know everybody loves an easy-to-read list:

* toBe(<value>) - The returned value is the same as <value>
* toBeTrue() - Checks if the returned value is true

toBeTruthy() - Check if the value, when cast to a boolean, will be a truthy value

Truthy values are all values that aren't 0, '' (empty string), false, null, NaN, undefined or [] (empty array)\*.

**How to set default browser window size in Protractor.**

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

browser.driver.manage().window().setPosition(x, y);

**getText() returns an object or String in Protractor?**

Unlike selenium, element.getText() return an object in Protractor and we need to resolve this.

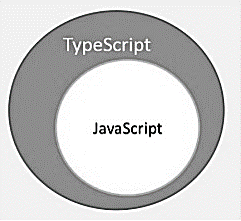
var foo = element(by.id('foo'));

expect(foo.getText()).toEqual('Inner text');

What is TypeScript?

JavaScript was introduced as a language for the client side. The development of Node.js has marked JavaScript as an emerging server-side technology too. However, as JavaScript code grows, it tends to get messier, making it difficult to maintain and reuse the code.

TypeScript is a strongly typed, object oriented, compiled language. It was designed by Microsoft. ***TypeScript***is an open-source programming language developed and maintained by [***Microsoft***](https://www.microsoft.com/). If you are aware of OOPS principles, ***typescript support all OOPS concepts just like C# and Java***. It is a typed superset, which means it provides an additional layer to javascript. So that it will be easy for developers coming from another programming background and knows the OOPS concept.

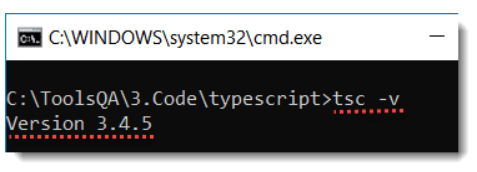


How to use TypeScript in Protractor?

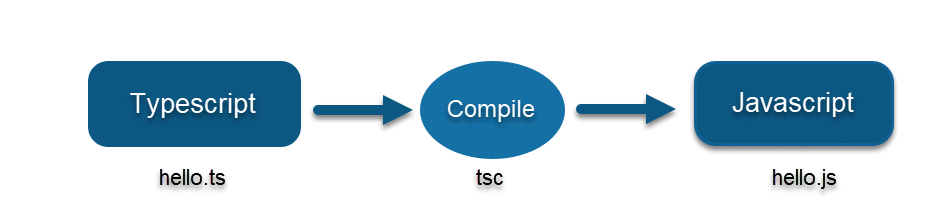
Install typescript first

***npm install -g typescript***

***check version***



***How does typescript work?***

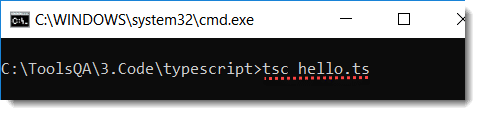
****

Create a new typescript file as hello.ts

|  |
| --- |
| //hello.ts  class Greeter {      greeting: string;      constructor() {          this.greeting = "World";      }      greet() {          return "Hello, " + this.greeting;      }  }    let greeter = new Greeter();  console.log(greeter.greet()); |

***How to compile typescript?***

TypeScript’s commands can be run in the ***command prompt***. Command to compile into java is: ***tsc <ts\_file>***



After successful compilation, the equivalent javascript ***.js*** file will be generated (i.e hello.js in our example) in the same folder.

How to add custom locator in Protractor?

Protractor addLocator method can then be used with element(by.locatorName(args)) to add your own custom locator in your framework.