PROTRACTOR

SELENIUM **2**
- Features; 2
- Automation; 2
- Components; 2
- Supported Programming Languages; 2
- WebDriverJS 2
- JSON Wire Protocol 3
- ASYNCHRONOUS PROBLEM of JS 3

PROTRACTOR **4**
- How Protractor Works? 5
- Why use Protractor? 5
- Why not Selenium? 5
- Protractor Configuration (conf.js) 6
- Protractor Test File (spec.js) 6
- BROWSER METHODS 7
- LOCATORS 8
- Protractor Locators 12
- CHECK VISIBILITY 17
- Interacting with Elements (Keyboard & Mouse) 18

JASMINE **20**
- SETUP & TEARDOWN 21
- ASSERTIONS to VALIDATE PROTRACTOR TESTs 22
- BETTER CONSOLE REPORT 24
- ALERTS, POPUPS, MULTIPLE WINDOWS and IFRAMES 24
- MULTIPLE WINDOWS & POPUPS 25
- FRAME & iFRAME 26
- WAITS 27
  - IMPLICIT WAIT 27
  - EXPLICIT WAY (We prefer) 28
- JAVASCRIPT EXECUTOR 28
  - JS EXECUTOR METHODS 29
  - FILE UPLOAD 29
  - BY.ADDLOCATORS 30
  - TAKING BASIC SCREENSHOT 30
- PROTRACTOR - BEAUTIFUL - REPORTER 31
- PAGE OBJECT MODEL 31
- FRAMEWORK STRUCTURE 32
- PACKAGE.JSON 33
- READING DATA and LOCATORS THROUGH JSON FILE 38
- EXECUTING TEST CASES 39
- JASMINE DATA PROVIDER 39
- READING & WRITING FROM EXCEL FILES 40

SELENIUM

- Selenium is an automation testing tool used to automate various types of web applications.
- Actually not a tool, library of tools.
- Free and Open Source (www.seleniumhq.com)
- Windows Desktop Applications = UFT (QTP)

Features;

- Flexible and extensible
- Multiple language & browsers supported

Automation;

- Simulate the interaction with browser like a real user.
- Selenium, automate browser.

TEST CASE →	Identify the web elements	Text Input Button Text Dropdown
	Interacting with them (Actions)	Navigating Page Clicking the Link or Button Enter Text
	Getting info or result of the Interaction	Result Submission Etc

Components;

- Selenium IDE (Int.Dev.Env) — Record & Playback, Firefox Add-on, Time-Sensitive prototypes
- Selenium RC (Remote Control) — *Selenium1*, JS, Needs Selenium Server
- WebDriver — API to send commands to browser, *Selenium2*, Replaced Selenium RC
- Selenium Grid — Run parallel Test, Different Machines & Browsers, Simultaneously
    - **OUR FOCUS → Se**

Supported Programming Languages;

LANGUAGE	FRAMEWORK
C#	**NUnit**
Haskell	
Java	**JUnit, TestNG**
JavaScript	**WebdriverJS, WebdriverIO, NightwatchJS, NemoJS**
Objective-C	
Perl	
PHP	**Behat + Mink**
Phyton	**unittest, pyunit, py.test, robot framework**
R	
Ruby	**RSpec, Test::Unit**

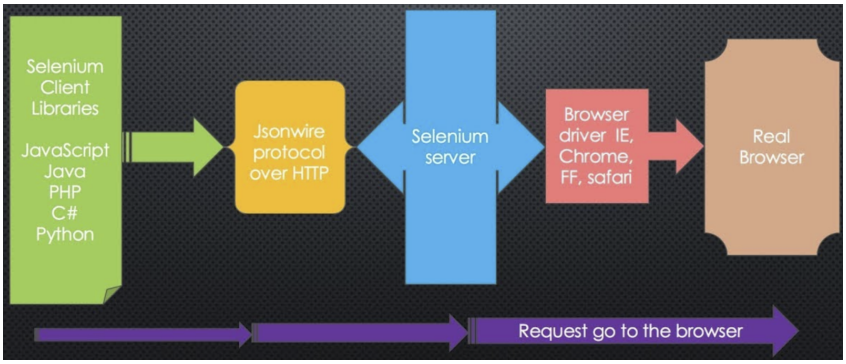
WebDriverJS

- Selenium has a lot of **language bindings.**
- **JS** being the language of choice for the web, WebDriverJS exists.
- It uses **JSONWire Protocol** to **interact** with browser.
- Difference? Asynchronous
- Async: in order to for the next action to start, previous action doesn't need to be finished.

JSON Wire Protocol

- JSON: JS Object Notation
- JS being the language of choice for the web, WebDriverJS exists
- It uses **JSONWire Protocol** to **interact** with browser.
- It is used to transfer data between a server and a client in the web
- XML and JSON are industry standard.

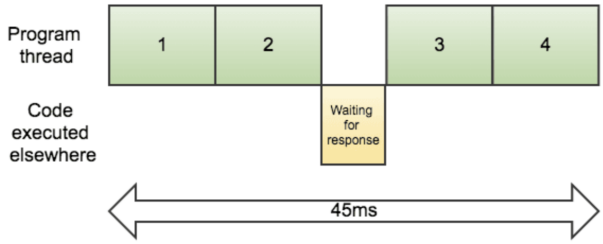
Selenium WebDriver Architecture



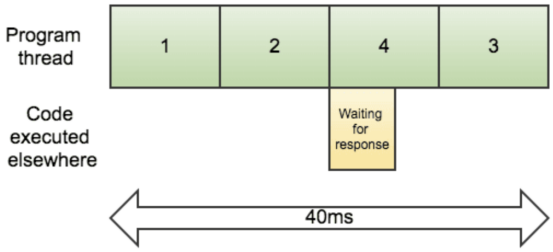
ASYNCHRONOUS PROBLEM of JS

- JS is Asynchronous → Every JS steps returns promise
- So, promise resembles state of our steps (1.pending 2.resolved 3.rejected)

Synchronous



Asynchronous



- *Synchronous*: 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 of our protractor API will not move to next step until promise is resolved (%10 problem)
      It means; if we want to perform any **actions** on the browser then it supports promise
    - %10 → If we want to retrieve anything from browser, then that particular functionality related methods have not support of promise resolving (Actually the problem is about JS, not the protractor: when we see the browser…. all of them is working synchronously, but console.log is JS and it works asynchronously. So we use **.then(function(){})** to solve this issue))

PROTRACTOR

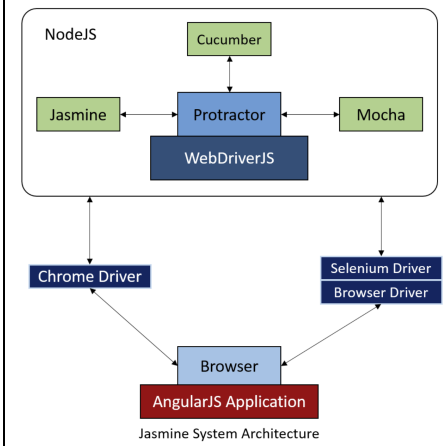
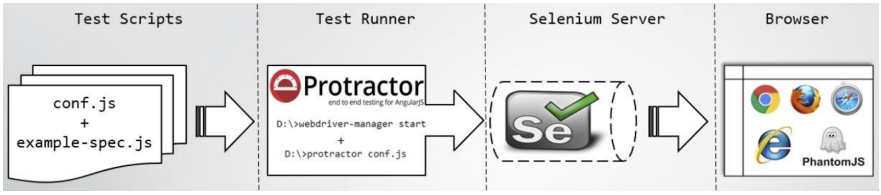
- It is an **end-to-end** test framework for Angular and AngularJS applications. Protractor runs tests against your application running in a real-browser, interacting with it as a user would.
- It has extra accessors to angular specific attributes such as **models, bindings, ng-options, ng-repeats**. These additions make accessing elements much easier.

End-to-end	Unit Testing
Interacting with the entire application stack	Interacting with a single, isolated unit at a time
Focus on what the user experiences	Focus on the functionality of the code
Runs in a browser	Runs in the console

END-TO-END

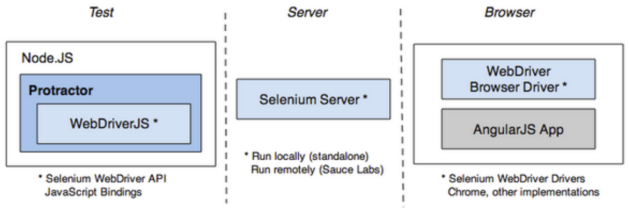
TEST CASE →	
Identify the web elements	Text Input Button Text Dropdown...
Interacting with them (Actions)	Navigating Page Clicking the Link or Button Enter Text
Getting info or result of the Interaction	Result Submission Etc

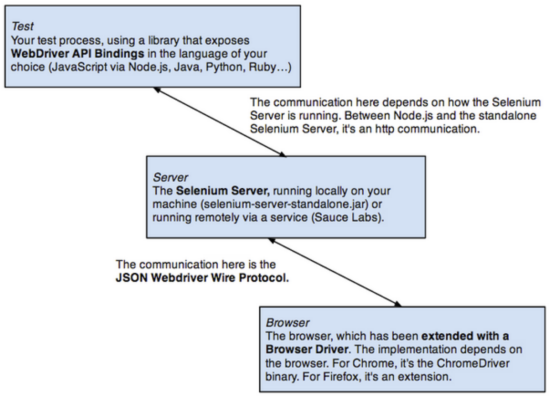
- It is built on the top of **WebDriverJS**
- It contains every feature that is available in the Selenium **WebDriver**. Additionally, protractor provides some **new locator strategies** and **functions** which are very helpful to automate the AngularJS
- Protractor is a **node.js** framework.

Architecture	Work Flow Diagram
	

Jasmine System Architecture

How Protractor Works?





- Protractor is a Node.js program that supports the Jasmine and Mocha test frameworks.
- Selenium is a browser automation framework. Selenium includes the Selenium Server, the WebDriver APIs, and the WebDriver browser drivers.
- Protractor works in conjunction with Selenium to provide an automated test infrastructure that can simulate a user's interaction with an Angular application running in a browser or mobile device.

Why use Protractor?

- It works in conjunction with Selenium to offer an end-to-end automated test infrastructure.
- It is like a bonus for Angular apps. Angular is designed on testing since the very beginning.
- Supports **behavior-driven development (BDD)** frameworks like **Jasmine**, **Mocha**, **Cucumber**
- No need to put wait Angular Apps. Protractor know when to run and check the DOM after Angular has done. (**Digest Cycle**)
- Test Case Execution Time is less. Time is money.
- It can use Selenium Grid to run multiple browsers at once.
- Makes our lives easier
    - Hooks to Angular
    - Wait for Angular
    - Get elements by (bindings, repeaters, model)

Why not Selenium?

- Custom HTML attributes remember? **NO - Selenium cannot find**
  * no-repeater * ng-controller *ng-model *ng-bindings
- Synchronization issue to handle
- It works in a harmony in node.js Echo System

Protractor Installation

- It is node.js program (http://nodejs.org/) and we need **JDK**
- Terminal **keywords**;
    - npm install -g protractor
    - protractor --version
    - START SELENIUM WEB SERVER
        - sudo webdriver-manager update or sudo webdriver-manager start
        - webdriver-manager start
        - webdriver-manager shutdown
    - RUN YOUR SPECS
        - protractor conf.js
- IDE Installation (Visual Studio Code)

Protractor Configuration (conf.js)

- Selenium Server Configurations
- Browser Options (Multiple Browsers option)
- Where the test scripts are located (spec.js)
- The Base URL for your spec files
- Jasmine Node Configuration

```
exports.config = {

    //address of our selenium server
    seleniumAddress: 'http://localhost:4444/wd/hub',

    //which browser would you want to use for your tests?
    capabilities: {'browserName': 'chrome'},
    /*FOR MULTIBROWSERS we use;
    multiCapabilities: [
        {'browserName': 'chrome'},
        {'browserName': 'firefox'}
    ],
    */
    //the name of your test scripts file
    specs: ['lab1.js'],

    //which BDD framework we're going to use
    framework: 'jasmine2',

    //option to be passed to Jasmine-node
    jasmineNodeOpts: {
        showColors: true,
        defaultTimeoutInterval: 30000

    }
};
```

Protractor Test File (spec.js)

describe function	- called **Test Suite**
	- Container for Test Cases
	- Compose your suite as a tree
	- Can be nested
	- Mandatory
	- A collection of similar type of test cases
	
	!!! xdescribe (disabled describe)
	!!! fdescribe (focus describe)
		describe(description, specDefinitions);
		
		`describe('Login Feature',function(){`
		
		`});`

it function	- called **Test Case**
	- Define a single spec. A spec should contain one or more **expectations**.
	- All expectations succeed -> Passed
	- Any failures ->Failed
	
	!!! xit (disabled it)
	!!! fit (focus it)
		it(description, testFunction$_{opt}$, timeout$_{opt}$);
		
		xit(description, testFunction$_{opt}$)
		fit(description, testFunction, timeout$_{opt}$)
		`describe ("Name of the test Suite", function(){`
		`  //xit --> Temporarily disabled the it`
		`  //fit --> We want to run only this test cases`
		`  it ('Name of the Test Case(spec)', function(){`
		`  console.log('1st test case');`
		`  });`
		`});`

BROWSER METHODS

- **browser** is a global object.

<table>
<tr><td>browser.sleep(time);</td><td>It simply makes the browser wait</td><td rowspan="2"><pre>
//to fix non-angular 1st keyword
//browser.ignoreSynchronization = true;
//to fix non-angular 2nd keyword
//browser.waitForAngularEnabled(false);
it("should open a webpage",function(){
    browser.get("https://www.jetblue.com/");
    browser.get("https://www.lego.com/");
    browser.sleep(4000);
});
</pre></td></tr>
<tr><td>browser.get(URL);</td><td>Navigates to the given URL

!!! to fix non-angular websites</td></tr>
<tr><td>browser.refresh();</td><td>Refreshes the page</td><td><pre>
it("should refresh a webpage",function(){
    browser.get("https://www.jetblue.com/");
    browser.refresh();
});
</pre></td></tr>
<tr><td>browser.navigate().back();
browser.navigate().forward();</td><td>Pressing back / forward button in your browser</td><td><pre>
it("should navigate back/forward a page",function(){
    browser.get("https://www.jetblue.com/");
    browser.get("https://www.lego.com/");
    browser.navigate().back();
    browser.navigate().forward();
});
</pre></td></tr>
<tr><td>browser.manage().window().maximize();
browser.manage().window().setSize(w,h)</td><td>-Maximizes the browser window
-Setting the size of the browser</td><td><pre>
it('should maximize/setSize windows', function(){
    browser.waitForAngularEnabled(false);

    browser.get('https://www.google.com/');
    browser.sleep(4000);
    browser.manage().window().maximize();
    browser.sleep(3000);
    browser.manage().window().setSize(1580,800);
    browser.sleep(4000);
});
</pre></td></tr>
<tr><td>browser.getCurrentUrl();</td><td>Return the URL of currently active page as a String</td><td><pre>
it("should get current URL", function(){
    browser.get("https://www.lego.com/");
    browser.getCurrentUrl().then(function(url){
        console.log(url);
    });
});
</pre></td></tr>
<tr><td>browser.getTitle();</td><td>Returns the title of the current page as a String</td><td><pre>
it("should verify title", function(){
    browser.get("https://www.lego.com/");
    browser.getTitle().then(function(title){
        console.log(title);
    });
});
</pre></td></tr>
</table>

LOCATORS

We will learn how to locate and perform actions on DOM elements using Protractor.

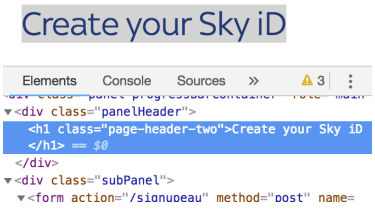
1- Element and by

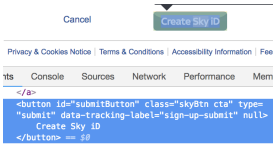
element and **by** are global functions created by protractor

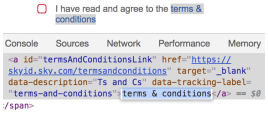
- element functions is used for finding and interacting HTML elements on your webpage.
- element(locator) ⇒ **element(by.attribute(' '))** → describe how to find the element
- **INTERVIEW QUESTION:** We need to find a unique item to locate. Selenium and Protractor uses Left to right, top to bottom, it is gonna fill the first one which is found by the tool. So **id** is the best one to use.
- We should give the address (somehow with locators) to protractor to locate the web elements. Locators should be unique. We inspect from Chrome and find the elements attributes.
- *Element:* starts with HTML tag. Everything we see in the page(DOM) is an element; <a .....>, <div .....>
- We use locators inside **element** function. This is our toolbox locating the elements.
- A locator tells Protractor how **to find** a certain DOM element. Protractor exports locator factories on the global **by** object.

by.id()	→ most reliable, desirable, 1st priority
	one **exception**: if you feel that the id has been created dynamically. If you see numbers in the id attribute, don't use it.
	 
		`describe("Selenium Locators", function(){`
		`  it("should login Facebook by.id() locator", function(){`
		`  browser.waitForAngularEnabled(false);`
		`  browser.get("https://www.facebook.com/");`
		`  //ACTIONS: sendKeys: send keys to the element`
		`  element(by.id("email")).sendKeys("myemail@gmail.com");`
		`  //sendKeys: wrong info --> Negative Testing`
		`  element(by.id("pass")).sendKeys("password123");`
		`  browser.sleep(3000);`
		`  //be careful about clicking id; it can be dynamic`
		`  //ACTIONS: .click(): click on the element`
		`  element(by.id('u_0_2')).click();`
		`  browser.sleep(3000);`
		`  });`
		`});`

by.name()		`it("by.name locator example", function(){`
		`  browser.waitForAngularEnabled(false);`
		`  browser.get("https://skyid.sky.com/signup");`
		`  element(by.name("lastname")).sendKeys("Stone");`
		`  browser.sleep(3000);`
		`});`

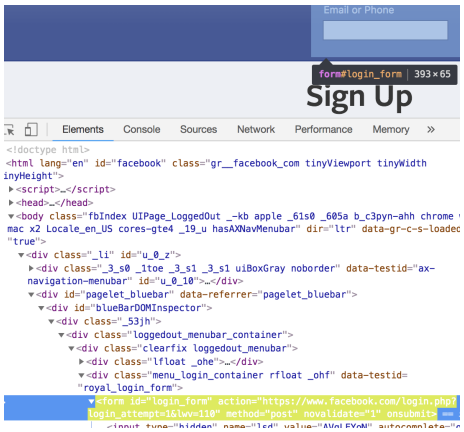
by.className()	→ only if there is one class name for your element
	 
		`it("by class name locator", function(){`
		`  browser.waitForAngularEnabled(false);`
		`  browser.get("https://skyid.sky.com/signup");`
		`  //ACTIONS: .getText() → PROMISE`
		`  element(by.className("page-header-two")).getText()`
		`  .then(function(text){`
		`  if(text=="Create your Sky iD"){`
		`  console.log("Passed");`
		`  }else{`
		`  console.log("Failed");`
		`  }`
		`  });`
		`  browser.sleep(3000);`
		`});`

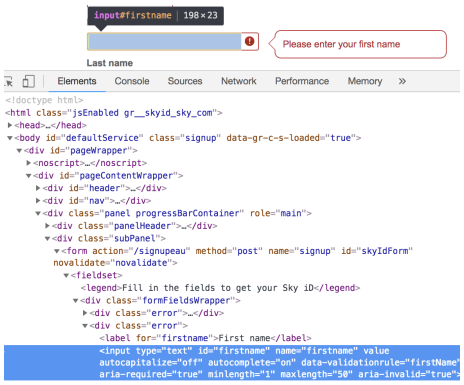
by.tagname()		`it("by class Facebook", function(){`
		`  browser.waitForAngularEnabled(false);`
		`  browser.get("https://skyid.sky.com/signup")`
		`  element(by.tagName("button")).click();`
		`  browser.sleep();`
		`});`

by.linkText()	this is only for links (<a....>), it should be exactly the same text of the link.
	 
		`it("by.linkText locator example", function(){`
		`  browser.waitForAngularEnabled(false);`
		`  browser.get("https://skyid.sky.com/signup");`
		`  browser.sleep(4000);`
		`  element(by.linkText('Terms & Conditions')).click();`
		`  browser.sleep(4000);`
		`});`

by.partialLinkText()	this is only for links (<a....>), but you have the luxury to use only a small part of the text of the link. e.x. Feedback --> eed (if it is unique, i can use)
	 
		`it('by.partialLinkText locator example 2',function(){`
		`  browser.waitForAngularEnabled(false);`
		`  browser.get('https://skyid.sky.com/signup');`
		`  browser.sleep(4000);`
		`  element(by.partialLinkText('ook')).click();`
		`  browser.sleep(4000);`
		`});`

If the locators above did not work, we use **xpath**

`by.xpath("Xpath")`	-Finding element on the page(XML path) -It makes our test execution slow	
Absolute Path → / Whole address starting from /html - Not effective - Fragile - Avoid using because if developer changes the dom, it wont work		/html/body/div[1]/div[2]/div[1]/div[1]/div[1]/div[1]/div[2]/form[1]

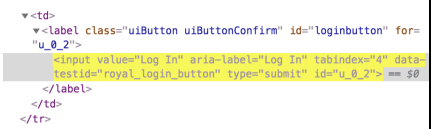
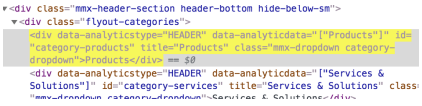
Relative Path → //
Can start from anywhere in the page
- More reliable
- Search element in the whole HTML page until find the unique one
- More flexible
- Change friendly
 	
DOM traveller → **traversing**
		**SYNTAX**
		//tagname[@attributeName='value']
		
		//input[@id='firstname']
		
		<u>Any tag works for me (Wildcard):</u>
		//*[@id='firstname']

		//a[@class="has-sub-menu js-top-navigation"]
		
		<u>with Child element;</u>
		//li[@data-active-level-1="mens"]/a
		
		<u>With Wildcard</u>
		//*[@id="navigation"]/ul[1]/li[1]/a
		
		<u>Finding a sibling;</u>
		//div[@class="megamenu"]/../a

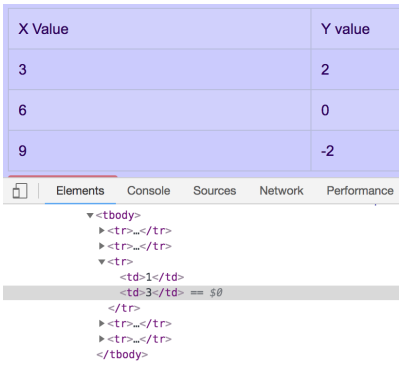
XPATH (Complex and **Dynamic** Elements)

Everything is an object on any webpage and we are trying to reach them with DOM.
So, XPATH is an address of the element
→ Can 2 elements have same xpath?
//*[@class='floatR'] - found 2 elements. Like home and forget password

//*[contains(@attributeName,'value')]		//p[contains(@class,'desktop')]
		
		//p[contains(@class,'desk')]
		
		//*[contains(@class,'desk')]

//*[contains(@attributeName,'value') and contains(@attributeName,'value')] **//*[@attributeName='value') or @attributeName='value']** **//*[@attributeName='value') and @attributeName='value']**		//input[@value="Log In" and @ type="submit"] //*[@value="Log In" and @ type="submit"] //*[contains(@value,'Log') and contains(@aria-label,'L')]
//*[starts-with(@attributeName,'value')]		//span[starts-with(@id,'your')] //*[starts-with(@id,'yourS')] //*[starts-with(@id,'u_') and contains(@value,"Log")]
//*[text()= '*exact* text']		//*[text()= 'Products']
//*[contains(text(), '*Partial* text')]		//div[contains(text(), 'Fill')]

TABLES

//**table**[contains(@class,'table')]/...		//table[contains(@class,'table')]/tbody/tr[3]/td[1]

CSS vs XPATH (might be interview question)

XPATH	CSS
//tagname[@attributeName='value']	tagname[attributeName='value']
slower	faster
Used only by tester	Used by developers, web scrapers, testers
Can traverse in reverse and forward	CSS can traverse only forward

class shortcut is → .

id shortcut is → #

^ : starts-with

$: ends-with

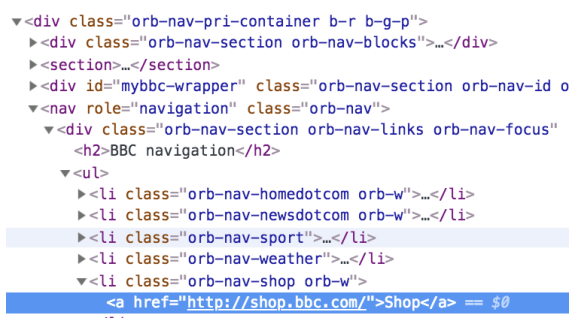
* : substring, contains

`<h1 class="page-header-two">Create your Sky iD</h1>`	h1[class='page-header-two'] → element(by.css("h1[class*='header']")); or → element(by.css("[class*='header']")); or → $("h1[class*='header']"); or → $("#header");
`<div class="defaultBubble emailError visible">`	div.defaultBubble.emailError.visible
`<input type="text" id="lastname" name="lastname" value autocapitalize="off" autocomplete="on" required="true" data-no-info="true" minlength="1" maxlength="50" aria-invalid="false"> == $0`	input[id='lastname'] input#lastname #lastname → element(by.css('#lastname')); or → $('#lastname');

css direct child → >

sub child → space

:nth-of-type(index) : to go to an element with index number

 	div.orb-nav-pri-container.b-r.b-g-p>nav.orb-nav li.orb-nav-shop a .orb-nav-section.orb-nav-links.orb-nav-focus>ul li.orb-nav-shop.orb-w a div.orb-nav-pri-container.b-r.b-g-p li:**nth-of-type(5)** a

Locator Precedence

1- id

2- Protractor Locators

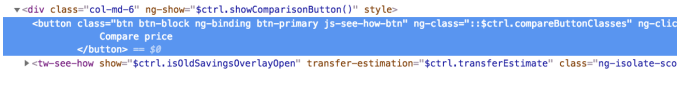
3- Selenium Locators

4- xPath

Protractor Locators

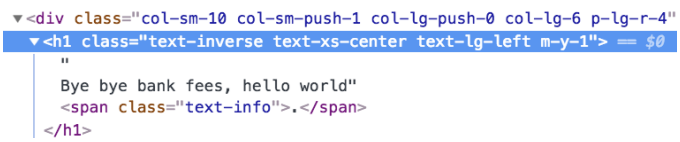
1- by.buttonText

2- by.partialButtonText



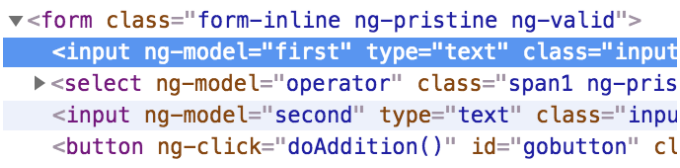

```
describe ("protractor locators", ()=>{
   it("buttonText example",()=>{
        browser.get("https://transferwise.com/us/");
        element(by.buttonText("Compare price")).click();
        //element(by.partialButtonText("Compa")).click();
   });
});
```

3- by.cssContainingText



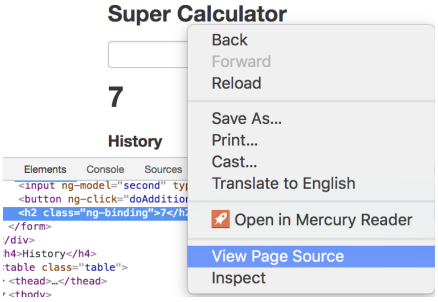

```
it("buttonText example",()=>{
       browser.get("https://transferwise.com/us/");
       element(by.cssContainingText('.text-inverse','hello
world')).getText().then((text)=>{
           console.log(text);
       });
   });
```

4- by.model

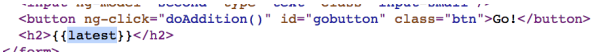


```
it("by.model example",()=>{
   browser.get("http://juliemr.github.io/protractor-demo/");
        element(by.model('first')).sendKeys(4);
        element(by.model('second')).sendKeys(4);

});
```

5- by.binding & 6- by.exactBinding



VIEW SOURCE



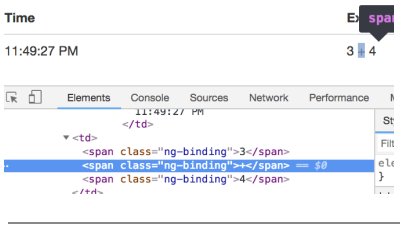

```
fit('binding',()=>{
    browser.get("http://juliemr.github.io/protractor-demo/");
    element(by.model("first")).sendKeys(5);
    element(by.model("second")).sendKeys(6);
    element(by.id("gobutton")).click();
    element(by.binding('latest')).getText().then((text)=>{
        console.log(text);
    })
})
```

7- by.repeater & 8- by.exactRepeater

```
<!-- ngRepeat: result in memory -->
<tr ng-repeat="result in memory" class="ng-scope">
  <td class="ng-binding">
            3:37:47 PM
          </td>
  <td>
    <span class="ng-binding">3</span>
    <span class="ng-binding">+</span>
    <span class="ng-binding">4</span>
  </td>
  <td class="ng-binding">7</td>
</tr>
<!-- end ngRepeat: result in memory -->
```

```
fit('bymodel with function', () => {
    browser.get("http://juliemr.github.io/protractor-demo/");

    element.all(by.repeater("result in memory"))
       .getText()
           .then(function(arrayText){
        console.log(arrayText);
    });
    element.all(by.css("option[value]")).getText()
       .then(y=>{console.log(y)});
});
```

```
function Add(a,b){
    element(by.model("first")).sendKeys(a);
    element(by.model("second")).sendKeys(b);
    element(by.id("gobutton")).click();
};
it('Open Angular js website',function() {
    browser.get('http://juliemr.github.io/protractor-demo/');
    Add(5,6);
    Add(7,8);
    Add(3,9);

    element.all(by.repeater('result in memory').row(2))
           .getText().then(function(text){
        console.log(text);
    })
});
```



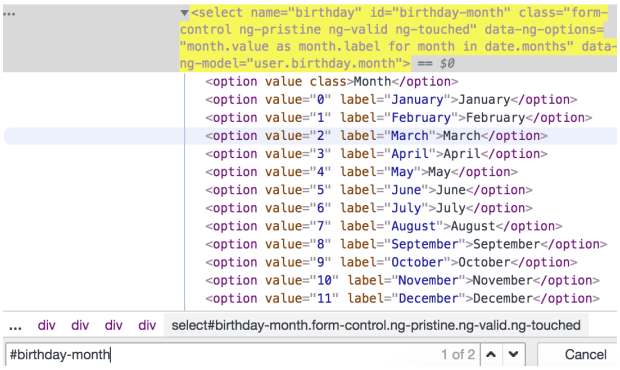
VIEW SOURCE

```
<tr ng-repeat="result in memory">
  <td>
    {{result.timestamp | date:'mediumTime'}}
  </td>
  <td>
    <span>{{result.first}}</span>
    <span>{{result.operator}}</span>
    <span>{{result.second}}</span>
  </td>
```

```
it('Open Angular js website',function() {
    browser.get('http://juliemr.github.io/protractor-demo/');
    Add(5,6);
    Add(7,8);
    Add(3,9);

    element.all(by.repeater('result in memory')
        .row(2)
            .column('result.operator'))
                .getText().then(function(text){
        console.log(text)
    })
});
```

9- by.options (for dropdown menus)


```
it ('ng-options', ()=>{
    browser.get('http://juliemr.github.io/protractor-demo/');

    var allOptions = element.all(by.options('value for (key,
value) in operators'));
    allOptions.each(function(option){
        option.getText().then(function(text){
            console.log(text);
        })
    }).then(function(){
        //print the third option
        var thirdOption= allOptions.get(2);
        thirdOption.getText().then(function(text){
            console.log(text);
        })
    })
});
```




```
it('another way to print options',()=>{
    browser.waitForAngularEnabled(false);
    browser.get('https://www.skout.com/');

$$('#birthday-month').first().all(by.tagName('option')).getText()
.then(function(text){
        console.log(text);
    })
})
it('selecting with a text',()=>{
    browser.waitForAngularEnabled(false);
    browser.get('https://www.skout.com/');
    //select item with cssContainingText method

$$('#birthday-month').first().all(by.cssContainingText('option',
'April')).getText().then(function(text){
        console.log(text);
    })
    //select element by text

element(by.model('user.birthday.month')).element(by.xpath('option
[.="April"]')).click();
    browser.sleep(4000);
    //selecting with a value

element(by.model('user.birthday.month')).element(by.css('option[v
alue="5"]')).click();
    browser.sleep(4000);
    //clicking an element with css/ value
element.all(by.id("birthday-month")).first() //Finding the
dropdown list
        .element(by.css('option[value="2"]')).click();

})
```

.all

element.all(by.css("locator")); → **$$("locator");**

Finds all elements depending on the locator and puts all the items in an **array**.

Methods : count - first - last - get - each

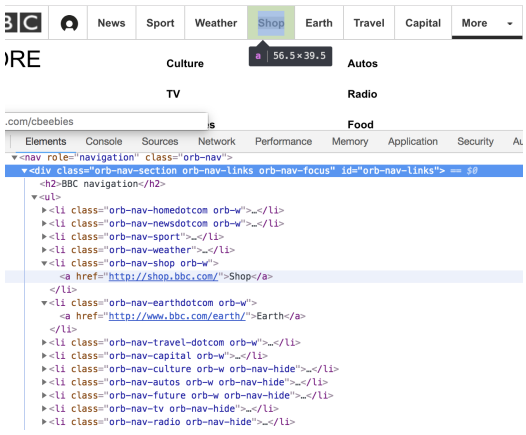
Chain Locators → By chaining the locators like CSS Child Element, we have the chance to find child elements.

a sub-element:

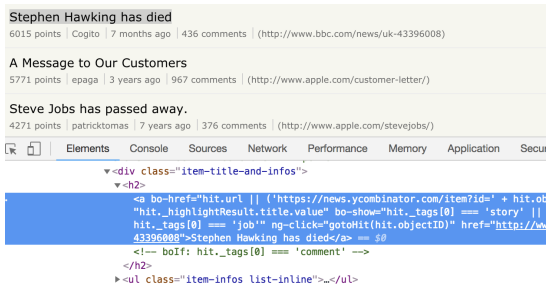
```
element(by.css('some-css')).element(by.tagName('tag-within-css'));
```

to find a list of sub-elements:

```
element(by.css('some-css')).all(by.tagName('tag-within-css'));
element.all(by.css('some-css')).first().all(by.tagName('tag-within-css'));
```

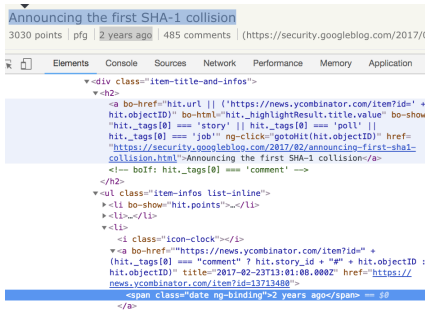



```
describe ("All",function(){
   xit('should explain element.all method',()=>{
        browser.waitForAngularEnabled(false);
        browser.get('https://www.bbc.com');
        element.all(by.css("*[class*='orb-nav-section'] ul li a"))
            .getText().then((result) => {
            console.log(result);
        });
   });
   fit('grab a specific list from BBC', () => {
        browser.waitForAngularEnabled(false);
        browser.get("http://www.bbc.com/");
        //CSS parent to child version
        $$("#orb-nav-links>ul>li").getText().then(function(links){
                console.log(links);
        })
        //Chain Locators
       element(by.css("#orb-nav-links")).all(by.tagName("li"))
            .getText().then(function(links){
              console.log(links);
        });
   });
});
```




```
fit ('should get all elements', ()=>{
        browser.get("https://hn.algolia.com");
        var results = $$(".item-title-and-infos>h2>a");
        results.getText().then(function(text){
            console.log(text);
        })});
```

```
it ('should try all methods elements', function(){
        browser.get("https://hn.algolia.com");
        //count method
        $$(".item-title-and-infos>h2>a").count().then(function(c){
                console.log("total items: " + c);
        });
        //first method
        $$(".item-title-and-infos>h2>a").first().getText()
          .then(function(a){
                console.log("First title: " + a);
        });
        //last method
        $$(".item-title-and-infos>h2>a").last().getText()
          .then(function(a){
            console.log("Last title: " + a);
        });
        //get method → Starts with "0" index
        $$(".item-title-and-infos>h2>a").get(2).getText()
           .then(function(a){
            console.log("Index title: " + a);
        });
});
```



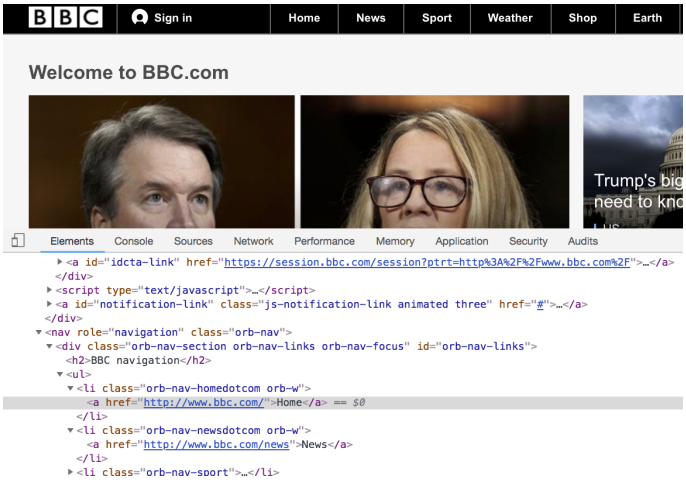

```
it("should all, chain", function(){
        browser.get("https://hn.algolia.com");
        var parent = $$(".item-title-and-infos");
        var title = parent.$$("h2>a");
        var time = parent.all(by.partialLinkText("ago"));
        title.get(10).getText().then(function(a){
            console.log("Index title: " + a);
        });
        time.get(10).getText().then(function(a){
            console.log("Index title: " + a);
        });
    });
```

.each method

```
describe ('each method examples', function(){
    it ('each method 1', ()=>{
        browser.waitForAngularEnabled(false);
        browser.get('https://www.bhtp.com/');

        $$('#top-nav-down li').each(function(item){
            item.getText().then(function(text){
                if(text!==""){
                console.log(text);
                }
            });
        });

    });
});
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

[11:11:17] I/direct - Using ChromeDriver directly...
Started
TRAVEL INSURANCE
BLOG
TRAVEL AGENTS
MORE
.
```

```
it ('each method #1', ()=>{
    browser.waitForAngularEnabled(false);
    browser.get('https://www.bhtp.com/');

    $$('#top-nav-down li').each(function(item){
        item.getText().then(function(text){
            if(text){
            console.log(text);
            }
        });
    });
});
```

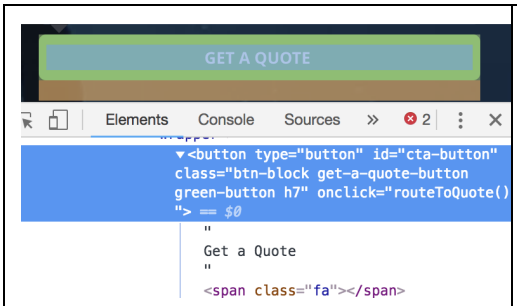



```
it ('each method #3', ()=>{
    browser.waitForAngularEnabled(false);
    browser.get('http://www.bbc.com/');
    browser.manage().window().setSize(2400,2400);
    $$('.orb-nav a').each(function(item, index){
        item.getText().then(function(text){
            if(text){
            console.log(text, index);
            }
        });
    });
});
```

```
Started
**Home 0
News 1
Sport 2
Weather 3
Shop 4
Earth 5
Travel 6
Capital 7
More 24
.
```

CHECK VISIBILITY

1- isDisplayed
2- isPresent
3- isEnabled




```
it('isPresent method',()=>{
    browser.get('https://www.bhtp.com/');
    element(by.id('cta-button')).isPresent().then(function(text){
        console.log(text); //true
    })
});
it('isDisplayed method',()=>{
    browser.get('https://www.bhtp.com/');
    element(by.id('cta-button')).isDisplayed().then(function(text){
        console.log(text); //true
    })
});
```

Without filling the form, it is not Enabled

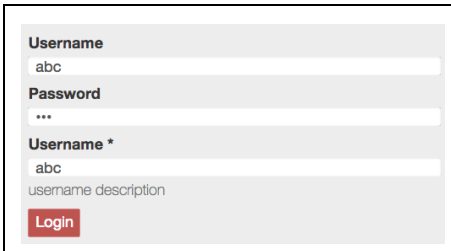

```
it('isEnabled method',()=>{
browser.get('http://www.way2automation.com/angularjs-protractor/registeration/#/login');
    $('.btn-danger').isEnabled().then(function(text){
        console.log(text); //false
    })
});
```

Enabled → clickable


```
it('isEnabled method',()=>{
browser.get('http://www.way2automation.com/angularjs-protractor/registeration/#/login');
    element(by.id('username')).sendKeys('abc');
    element(by.id('password')).sendKeys('123');
    element(by.id('formly_1_input_username_0')).sendKeys('abc');
    $('.btn-danger').isEnabled().then(function(text){
        console.log(text); //true
    })
});
```

Interacting with Elements (Keyboard & Mouse)

1- Clear Method




```
describe('keyboard and mouse practice',()=>{
   it('clear method',()=>{
   browser.get('http://www.way2automation.com/angularjs-protractor/registeration/#/login');
        var userName= $('#username')
        userName.sendKeys('Cybertek');
        browser.sleep(3000);
        userName.clear();
        browser.sleep(3000);
        userName.sendKeys('angular');
        browser.sleep(3000);
   });
});
```

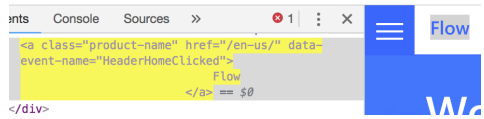
2- Special Simulating Keyboard Keys


```
   it('using keyboard buttons',()=>{
        browser.get('https://flow.microsoft.com/en-us/');
        var backSpace = $('[data-test="homepage-search-input"]')
        backSpace.sendKeys('dropbox');
        browser.sleep(3000);
        backSpace.sendKeys(protractor.Key.BACK_SPACE)
             .sendKeys(protractor.Key.ENTER);
        browser.sleep(3000);
   });
```

3- Getting Information from Elements

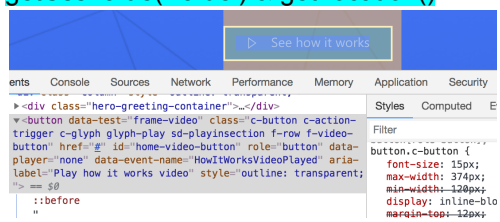
getText

getAttribute('attribute')



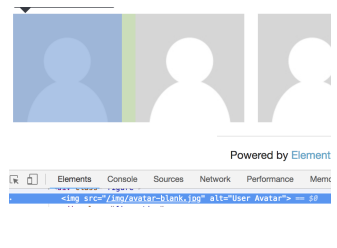

```
   it('Getting Information from Elements',()=>{
        browser.get('https://flow.microsoft.com/en-us/');
        //getText() => text
      $('.hero-greeting-container h1').getText().then(function(text){
            console.log(text);
        })
        //getAttribute('Attribute') => value
      $('.product-name').getAttribute('href').then(function(text){
            console.log(text);
        })
   })
```

getCssValue('value') & getLocation()



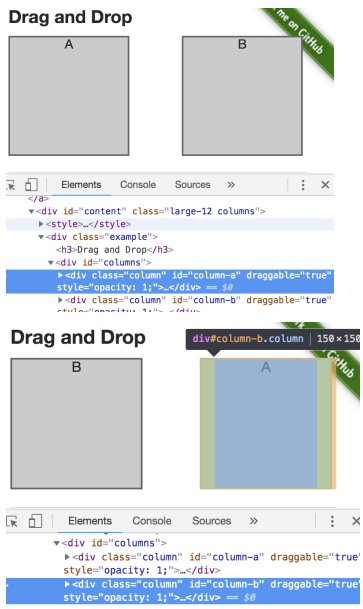

```
   it('Getting Information from Elements',()=>{
        browser.get('https://flow.microsoft.com/en-us/');
        //getCssValue('value') =>
        $('#home-video-button').getCssValue('font-size').then(function(text){
          console.log(text)
      });
        //getLocation
        $('#home-video-button').getLocation().then(function(location){
            var x = location.x;
            var y = location.y;
            console.log(x + '--------' +y)
        })
   })
```

4- Hover Over → ……..actions()……..perform()




```
   it('mouse hover over',()=>{
        browser.waitForAngularEnabled(false);
        browser.get('http://the-internet.herokuapp.com/');
        element(by.linkText('Hovers')).click();
        browser.sleep(2000);
        browser.actions().mouseMove($$('.figure img')
             .get(0)).perform()
        element(by.linkText('View profile')).click();
        browser.sleep(2000)
   })
```

5- Drag and Drop (....sudo npm install --save-dev html-dnd) → to Terminal for mac




```
it('Drag and Drop',()=>{

    var dragAndDrop = require('html-dnd').code;
    var elementA = element(by.id('column-a'));
    var elementB = element(by.id('column-b'));

    browser.waitForAngularEnabled(false);
    browser.get('http://the-internet.herokuapp.com/drag_and_drop')
    browser.sleep(3000);
    browser.executeScript(dragAndDrop,elementA,elementB,0,0);
    browser.sleep(5000);

})
```

6- Scroll Down & JS Slick

```
//Scroll down to the end of the page
browser.executeScript('window.scrollTo(0,document.
body.scrollHeigh t)").then…

//Scroll to a position (getLocation)
browser.executeScript('window.scrollTo(271,3127);'
).then…

//JavaScript click
browser.executeScript("arguments[0].click();",elem
ent)
```

```
   it('scroll down to find an element',()=>{
       browser.get('https://www.bhtp.com/');
       browser.sleep(3000);

browser.executeScript('arguments[0].scrollIntoView();',element(by.lin
kText('START A CLAIM'))).then(function(){
           browser.sleep(3000);
               element(by.linkText("START A CLAIM")).click();
       })
   })
});
```

JASMINE

- It is a Behavior Driven Development (BDD) testing framework for JS. Features of BDD
    - Shifting from thinking in "tests" to **thinking in "behavior:** the desired behavior of the software to be developed. **"**
    - Extends **TDD** (Test-driven development) by utilizing **natural language** that **non technical** stakeholders can understand
    - Collaboration between Business **stakeholders**, Business **Analysts**, **QA** Team and developers
    - Ubiquitous language, it is easy to describe
    - Driven by Business **Value**
    - BDD frameworks such as **Cucumber** or **Jasmine** are an enabler, acting a "bridge" between Business & Technical Language
    - **TDD**: start writing tests first. This test will fail and we will write more code to pass these tests.
- Tests are written in plain descriptive English type grammar/syntax.
- Tests are explained as behavior of application and more user focused.
- Allows us communicating **with business people** to engage the whole team even not technical.
- It is capable of testing any kind of JavaScript application.
- By following BDD procedure, Jasmine provides a syntax to test the entire application.
- Unit testing is a fundamental part of BDD.
- It provides **assertions** for our tests.
- It is like a combination of Cucumber+JUnit for Java/Selenium
- Why Jasmine: Jasmine **does not depend** on any other JavaScript framework.
- All the syntax used in Jasmine framework is **clean and obvious** so that you can easily write tests.
- Jasmine is an **open-source f**ramework and easily available in different versions like stand-alone, ruby gem, Node.js, etc.

INTERVIEW BASED:

Jasmine	This is the behavior-driven JavaScript testing framework. It only support java script to write test cases. While automation using Protractor we use jasmine keywords. It provides global keywords like Browser etc.It can be used with node.js project for dependency management and other supports like creating a project structure.
Mocha	This is the JavaScript testing framework, but it cannot run as a standalone testing framework, it needs plugins or library to run as a testing framework. But it is easy to install and good documentation is available. It is used at unit testing level.
Cucumber	This is also the behavior-driven JavaScript testing framework.It support java,c# etc to write test cases. It also support many other features like automatic generation of step definitions,for java cucumber jar is available on maven repo.

xdescribe (a temporarily disabled describe)

fdescribe (a focused describe)

SETUP & TEARDOWN

- Jasmine has 4 global function to manage for setup and teardown in test suites and They helps to find where we did mistakes

SETUP; (when we have some initial steps before you actually start validation we will put it in a **setup method)**

1- beforeAll → executes only **once** before any of the `it` block and before any of the beforeEach functions

2- beforeEach → executes before any spec in the describe block containing `it`.

SIMPLE SAMPLE

```
describe('Jasmine Framework', () => {
   describe('A single describe',()=>{
       beforeAll(function() {
           console.log('in beforeAll');
       });
       beforeEach(function() {
           console.log('in beforeEach');
       });
       it('spec A', function() {
           console.log('in spec A');
       });
       it('spec B', function() {
           console.log('in spec B');
       });
   });
});
```

```
in beforeAll
in beforeEach
in spec A
.in beforeEach
in spec B
.
```

```
describe('Jasmine Framework', () => {
   describe('A single describe', function() {
       beforeAll(function() {
           console.log('in beforeAll');
       });
       beforeEach(function() {
           console.log('in beforeEach');
       });
       it('spec A', function() {
           console.log('in spec A');
       });
       it('spec B', function() {
           console.log('in spec B');
       });
       describe('A single describe', function() {
           beforeAll(function() {
               console.log('in inner beforeAll');
           });
           beforeEach(function() {
               console.log('in inner beforeEach');
           });
           it('spec A', function() {
               console.log('in inner A');
           });
           it('spec B', function() {
               console.log('in inner B');
           });
       });
   });
});
```

```
in beforeAll
in beforeEach
in spec A
.in beforeEach
in spec B
.in inner beforeAll
in beforeEach
in inner beforeEach
in inner A
.in beforeEach
in inner beforeEach
in inner B
.
```

TEARDOWN; (if you have any steps like any automation steps once that complete functionalities done you will put in the **tearDown method**.) 1- afterAll, 2- afterEach

```
describe('Validating the Calculator app', () => {
   beforeAll(function(){
       console.log('This is printing before All of the blocks started');
   });
   beforeEach(function(){
       browser.get('http://way2automation.com/angularjs-protractor/calc/');
   });
   afterEach(function(){
       console.log('This is printing after each it block');
   });
   afterAll(function(){
       console.log('This is printing after all of the tests blocks finished');
   });
       it('validate 1 + 1=2', () => {
           element(by.model("first")).sendKeys(1);
           element(by.model('second')).sendKeys(1);
           element(by.buttonText('Go!')).click();
           element(by.binding('latest')).getText().then(function(text){
               console.log('Result is: '+text);
           });
       });
       it('validate 2 + 2 = 4', () => {
           element(by.model("first")).sendKeys(2);
           element(by.model('second')).sendKeys(2);
           element(by.buttonText('Go!')).click();
           element(by.binding('latest')).getText().then(function(text){
               console.log('Result is: '+text);
           });
       });
       it('validate 3 + 3 = 6', () => {
           element(by.model("first")).sendKeys(3);
           element(by.model('second')).sendKeys(3);
           element(by.buttonText('Go!')).click();
           element(by.binding('latest')).getText().then(function(text){
               console.log('Result is: '+text);
           });
       });
});
```

```
This is printing before All of the blocks started
Result is: 2
This is printing after each it block
.Result is: 4
This is printing after each it block
.Result is: 6
This is printing after each it block
.This is printing after all of the tests blocks finished
```

ASSERTIONS to VALIDATE PROTRACTOR TESTs

- When you are testing a code, you are "**expecting**" the output of the code to be something. That's exactly what **expect()** along with **toBe()** does. This is just like an assertion. If it is true, then the spec will pass; otherwise, it will fail.
- **Expectations** are built with the function expect which takes a value, called the actual. It is chained with a Matcher function, which tikes the expected value. It verifies if the actual and expected are matching based on matchers.
- **Matchers** → Each matcher implements a Boolean comparison between the actual value and the **expected** value. It is responsible for reporting to Jasmine if the **expectation is true or false**. Jasmine will then pass or fail the spec. E.g: expect(2+2).toEqual(4);
- Negating Matchers Any matcher can evaluate to a negative assertion by adding the **not** in front of it.
  E.g: expect(2+5).**not**.toEqual(4);

Matchers

```
expect(instance).toBe(instance);
expect(mixed).toEqual(mixed);
expect(number).toBeGreaterThan(number);
expect(number).toBeLessThan(number);
expect(number).toBeCloseTo(number, decimalPlaces);
expect(mixed).toBeDefined();
expect(mixed).toBeFalsy();
expect(number).toBeNaN();
expect(mixed).toBeNull();
expect(mixed).toBeTruthy();
expect(mixed).toBeUndefined();
expect(array).toContain(member);
```

True vs Truthy

- In JS there is true and truthy.
  - When something is true it is obviously true or false.
  - When something is truthy it may or may not be a boolean, but the "**cast**" value of is a **boolean**.
- toBe(<value>) - The returned value is the same as <value>. same as → **===**
  - toBe(true) - Checks if the returned value is true
  - toBeTruthy() - Check if the value, when cast to a boolean, will be a true value .
- In fact, **everything that is not 0, "", null, undefined, NaN or false is truthy.**
- This means that if you're testing a function that erroneously returns 'false' (a string) instead of false (a boolean), toBeTruthy() will match and toBeFalsy() will not.
- Therefore, we should **AVOID using Truthy** and Falsy in our test scripts. Developer use them.

```
describe('toBeTruthy', function() {
   it('passes if subject is true', function() {
       expect(true).toBeTruthy();  //PASSED
       expect(false).not.toBeTruthy();
   });
   it('passes if subject is a non empty string', function() {
       expect('Should pass').toBeTruthy();  //PASSED
   });
   it('passes if subject is a number not equal 0', function() {
       expect(1).toBeTruthy();    //PASSED
   });
   it('fails if subject is an empty string', function() {
       expect('').not.toBeTruthy();  //PASSED
   });
   it('fails if subject is false', function() {
       expect(false).not.toBeTruthy();  //PASSED
   });
   it('fails if subject is 0', function() {
       expect(0).not.toBeTruthy();  //PASSED
   });
   it('toBeTrue', function() {
       //expect(1).toBeTrue();
       expect(0).toBe(true); //FAILED
   });
});
```

Started

.....F

Failures:
1) toBeTruthy toBeTrue
   Message:
   Expected 0 to be true.

toBe vs toEqual

- For primitive values, both toEqual() and toBe() matchers behave the same. The difference can be seen in object level testing.
- The toBe() Jasmine matcher is basically the === operator (equal value and equal type)
- Using the == operator (Equality) → `true == 1;"2" == 2;`
- Using the === operator (Identity) → `true === 1;`
  - This is because the equality operator == does type coercion→ the interpreter implicitly tries to convert the values before comparing.
  - On the other hand, the identity operator === does not do type coercion, and thus does not convert the values when comparing.

```
describe('Validating the Calculator app', () => {
   var expectedText;
   beforeEach(function(){
     browser.get('http://way2automation.com/angularjs-protractor/calc/'); });
   afterEach(function(){
       browser.sleep(3);
       console.log('This is printing after it block');});
       it('validate 7 + 7 = 14', () => {
           element(by.model("first")).sendKeys(7);
           element(by.model('second')).sendKeys(7);
           element(by.buttonText('Go!')).click();
           expectedText= element(by.binding('latest')).getText();
             expectedText.then(function(text){
               expect(parseInt(text)).toBe(14);
               });
       });
       it('validate 7+7 !=10',function(){
           element(by.model("first")).sendKeys(7);
           element(by.model('second')).sendKeys(7);
           element(by.buttonText('Go!')).click();
           expectedText= element(by.binding('latest')).getText();
             expectedText.then(function(text){
               expect(parseInt(text)).not.toBe(10);
               });
       });
       it('validate 7+7 =13',function(){
           element(by.model("first")).sendKeys(7);
           element(by.model('second')).sendKeys(7);
           element(by.buttonText('Go!')).click();
           expectedText= element(by.binding('latest')).getText();
             expectedText.then(function(text){
               expect(parseInt(text)).toBe(12);
               });
       });});
```

```
Started
This is printing after it block
.This is printing after it block
.This is printing after it block
F


Failures:
1) Validating the Calculator
app validate 7+7 =13
  Message:
    Expected 14 to be 12.
```

```
describe('toBeDefined', function() {
   it('passes if subject and expectation are equivalent', function() {
       expect('Hello World!').toEqual('Hello World!');
       expect('Hello World!').not.toEqual('Goodbye!');
       expect([1, 2, 3]).toEqual([1, 2, 3]);
       expect("1").toEqual(1);
       expect({ foo: 1 }).toEqual({ foo: 1 });
   });
   it('passes if subject is not undefined', function() {
       expect({}).toBeDefined();
       expect(undefined).not.toBeDefined();
   });
   it('passes if subject is null', function() {
       expect(null).toBeNull();
       expect(undefined).not.toBeNull();
       expect({}).not.toBeNull();
   });
   it('passes if subject is false', function() {
       expect(false).toBeFalsy();
       expect(true).not.toBeFalsy();
   });
   it('expected item is equal to the actual item up to a decimal precision ', ()=>{
       expect(1.223).toBeCloseTo(1.22);
       expect(1.233).not.toBeCloseTo(1.22);
       expect(1.23326).toBeCloseTo(1.23324, 3);
   });
   it('expected item is an element in the actual array, or a substring of string', function() {
       expect([1, 2, 3]).toContain(2);
       expect([1, 2, 3]).not.toContain(4);
   });
   it('passes if the actual value is greater than the expected value',()=>{
      expect(347).toBeGreaterThan(300);
   });
   it('passes if the actual value is less than the expected value', ()=> {
       expect(2).toBeLessThan(3);
   });
});
```

```
F.......


Failures:
1) toBeDefined passes if
subject and expectation are
equivalent
  Message:
    Expected '1' to equal 1.
```

```
//To summarize 2 use cases for contain():
//To check if item is an element of an
array.
//To check if item is a a substring of a
string.
```

BETTER CONSOLE REPORT

1. Install jasmine-spec-reporter through the command:
    a. **npm install jasmine-spec-reporter --save-dev**
2. In the Protractor configuration file, import the package and configure the customizable options:

```
let SpecReporter = require('jasmine-spec-reporter').SpecReporter;
exports.config = {
  framework: 'jasmine',

  directConnect: 'true',
  specs: ['zzzElnar.js'],

  onPrepare: function(){
    browser.manage().window().maximize();
    jasmine.getEnv().addReporter(new SpecReporter({
      displayFailuresSummary: true,
      displayFailuredSpec: true,
      displaySuiteNumber: true,
      displaySpecDuration: true
    }));
  }
};
```

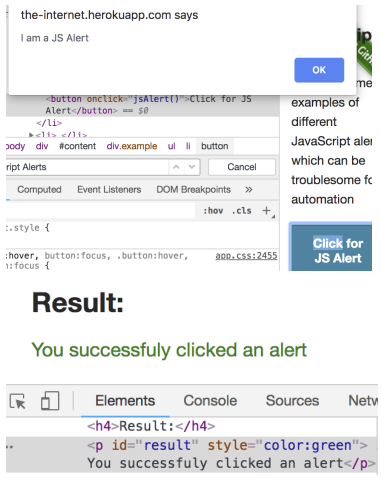
ALERTS, POPUPS, MULTIPLE WINDOWS and IFRAMES

Alert is a small message box which displays on-screen notification to give the user some kind of information or ask for permission to perform certain kind of operation. It may be also used for warning purpose. We cannot identify alerts using inspect tools

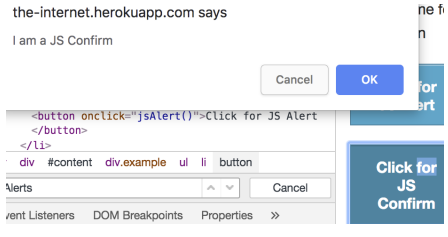
- We cannot just locate alerts.
- It is not a Window.
- It will not allow to perform any other operation o WebPage unless it is taken care of.
- We will need to switch to alert window via: `browser.switchTo().alert()`

Alert Methods:

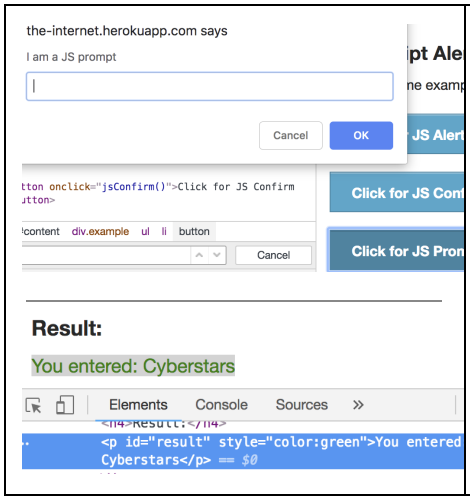
```
browser.switchTo().alert().getText();
browser.switchTo().alert().accept();
browser.switchTo().alert().dismiss();
browser.switchTo().alert().sendKeys();
```




```
describe ('Testing Alerts', ()=>{
   beforeAll(()=>{
        browser.waitForAngularEnabled(false);
        browser.get('http://the-internet.herokuapp.com/');
        element(by.linkText('JavaScript Alerts')).click();
   })
   it('should get text of alerts',()=>{
        element(by.buttonText('Click for JS Alert')).click();
        browser.sleep(2000);
        var myAlert = browser.switchTo().alert();
        expect(myAlert.getText()).toEqual('I am a JS Alert');
   });
   it('should click OK',()=>{
        var myAlert = browser.switchTo().alert();
        myAlert.accept(); //click OK
        expect($('#result').getText()).toEqual('You successfuly clicked an
alert');
   })
```



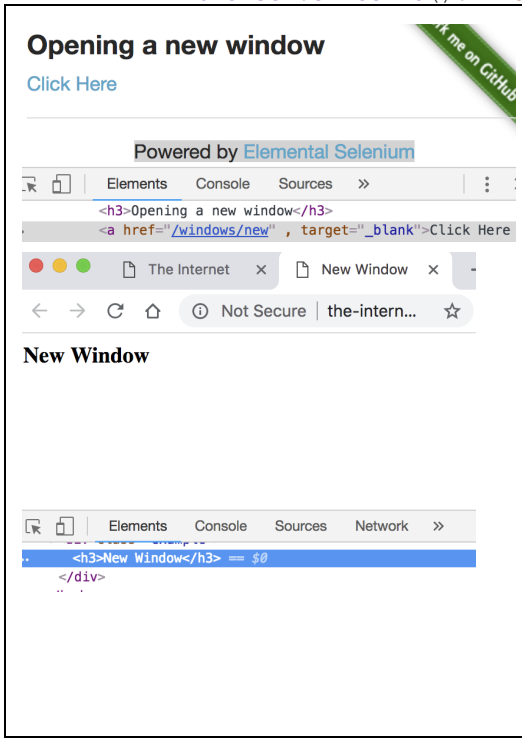

```
it('should dismiss the alert',()=>{
        element(by.buttonText('Click for JS Confirm')).click();
        browser.sleep(2000);
        var myAlert = browser.switchTo().alert();
        expect(myAlert.getText()).toEqual('I am a JS Confirm');
        browser.sleep(2000);
        myAlert.dismiss(); //click cancel
        expect($('#result').getText()).toEqual('You clicked: Cancel');
   });
```




```
it('should send some text to the alert',()=>{
    element(by.buttonText('Click for JS Prompt')).click();
    browser.sleep(2000);
    var myAlert = browser.switchTo().alert();
    expect(myAlert.getText()).toEqual('I am a JS prompt');
    browser.sleep(2000);
    var text= 'Cyberstars'
    myAlert.sendKeys(text);
    myAlert.accept();
    expect($('#result').getText()).toEqual('You entered: ' + text);
});
```

MULTIPLE WINDOWS & POPUPS

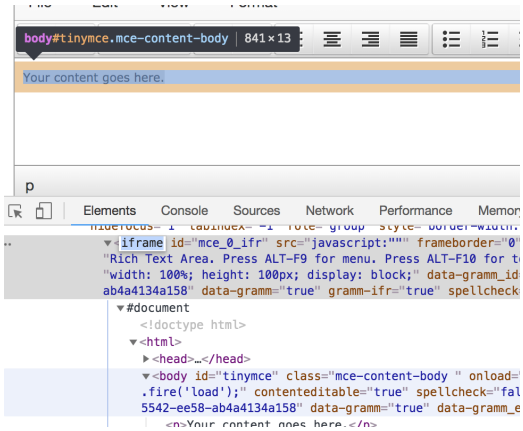
- There are some cases where you will have more than window/tab open
- When application opens up a new pop-up window or tab, Selenium/protractor will not automatically pass control to that page. Just because it is shown on the screen doesn't mean protractor has control on it by default.
- We have to pass control to the new tab/window perform our desired actions and then pass control back.
- Tabs and windows are treated as same in protractor. They both have unique GU IDs.
- To get the current (active) window/tab GU ID:
    - `browser.getWindowHandle();`
- To get all windows/tabs GU IDs:
    - `browser.getWindowHandles();`
    - Once you get the the window handles, it returns you art array as seen. we can loop through this array and do other operations as well.
- To switch control over to the target window/tab GU ID:
    - `browser.switchTo().window(guid);`



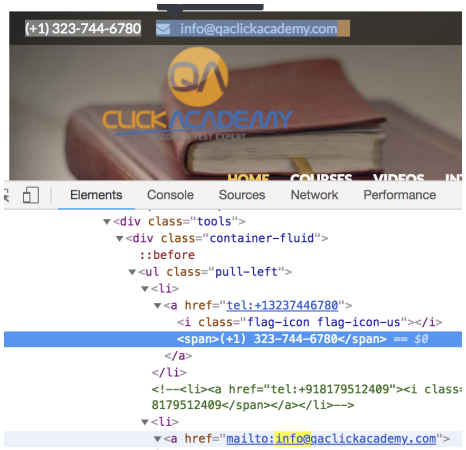

```
describe ('Testing Alerts', ()=>{

   var browserHandles = [];
   beforeAll(()=>{
       browser.waitForAngularEnabled(false);
       browser.get('http://the-internet.herokuapp.com/');
       element(by.linkText('Multiple Windows')).click();
       element(by.linkText('Click Here')).click();
   })
   it('should switch to new window',()=>{
       browser.getAllWindowHandles().then(function(handles){
           browserHandles = handles;
          browser.switchTo().window(browserHandles[1]).then(function(){
               browser.sleep(3000);
               expect(browser.getCurrentUrl()).toContain('windows/new')
           })
       });
   });
   it('should go back to previous window',()=>{
       browser.close().then(function(){
           browser.switchTo().window(browserHandles[0]);
           browser.sleep(2000);
           expect(element(by.linkText('Click
Here')).isDisplayed()).toBe(true);
       })
   });
});
```

FRAME & iFRAME

- iFrame is nothing but another webElement in html page, which displays another part of webpage.
- You **won't be able to interact with it via the DOM**. We have to **switch into frame** to see the elements present in the frame. Protractor throws NoSuchElementFound Exception unless we switch to the iframe
- We can handle iframes present in the webpage using **browser.switchTo()** command in protractor.
- There are couple of ways to switch to frame/iframe.
    - Using locator:
      browser.switchTo( ).frame(browser.driver.**findElement**(by.**tagName**('iframe'));
    - If using element array finder, then make sure to use .getWebElement to convert.
      browser.switchTo( ).frame(**element**(by.**tagName**('iframe')).**getWebElement**());
    - Using Index: (not preferred as order can change)
      // switch to 1st frame
      browser.switchTo( ).frame(1);
    - If nested iFrames, then to go back to outer iframe, use:
      browser.switchToParentFrame();
    - To switch back to default page:
      browser.switchTo( ).defaultContent();




```
describe ('Practising Iframe', ()=>{

   beforeAll(()=>{
       browser.waitForAngularEnabled(false);
       browser.get('https://the-internet.herokuapp.com/iframe');
   });

   it('should switch to iFrame',()=>{
       browser.sleep(3000);
       browser.switchTo( )
         .frame(browser.driver.findElement(by.tagName('iframe')));
           $('#tinymce').click();
           $('#tinymce').clear();
           $('#tinymce').sendKeys('Cyberstars')
           browser.sleep(6000);
   });
});
```




```
describe('Protractor Frames Steps',function() {
   beforeAll(()=>{
       browser.waitForAngularEnabled(false);
       browser.get("http://qaclickacademy.com/practice.php");
   })
   it('should open frame',function() {
       browser.switchTo().frame("courses-iframe");
       expect($$('.tools .container-fluid
span').get(0).getText()).toEqual('(+1) 323-744-6780')
       expect($$('.tools .container-fluid
span').get(1).getText()).toEqual('info@qaclickacademy.com')
       browser.switchTo().defaultContent();
       element(by.css('.logoClass')).click();
       browser.sleep(3000);
   });
});
```

WAITS

- Why do we need waiting?
    - Because world is not perfect.
    - You will come across scenarios where an element you are trying to interact is not loaded to dom, or not clickable at that moment etc.
- Most basic version of wait is: **browser.sleep**(5000);
    - Avoid using this in your test scripts, as it leads to **unnecessary longer test execution time**.
    - Forces browser to wait for the given amount of time, even if desired element shows up early.
    - Can be used while **troubleshooting** your script.
- When testing Angular pages, Protractor automatically waits for angular to be ready and then it executes the next step in the control flow. So, you don't need to wait.
- However, there are cases that you will need to use waits. Sometimes the tests outrun browser!
- You will face situations that we will need to use waits both on:
    - For some elements on **Angular pages**. (Not as common)
    - For some elements on **Non-angular pages**. (Very common)

1. IMPLICIT WAIT

- Implicitly wait in protractor, sets maximum time that we are going to wait for the element to be available in the Website.
- If we have given implicit wait of 30,000ms then, Implicit wait tries to find the element in first go, if element is not present, implicit wait tries to find the element after every 500ms, and it goes on till the time reaches the 30,000 milliseconds limit we set.
- Add following to the on Prepare option in conf.js file.
    - `browser.manage().timeouts().implicitlyWait(30000);`
- If **element is found** before implicitly wait time, **protractor moves to next commands** in the program **without waiting** to complete the implicitly wait time, this wait is also called dynamic wait.
- Implicitly wait is one of the way to request selenium not throw any exception until provided time.
    - NO_SUCH_ELEMENT Error
- Default wait time of the protractor is **500ms**. This gets overridden when added.
- Implicit wait is **set for the entire duration of your webdriver** and is set at the start of your program. If Mostly gets put in the **conf.js file.**
- **Issues:**
    - You are hardcoding same maximum wait time for all elements. If Element is taking longer than the set timeout, it will throw error.
    - One size fits all —> approach doesn't work.

CONFIG FILE (we add)

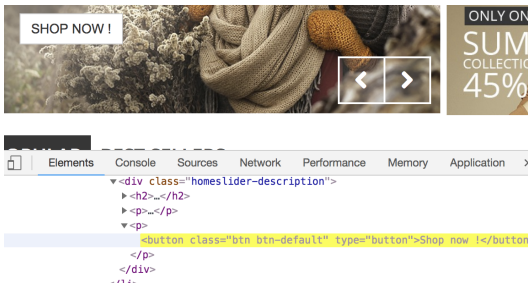
```
onPrepare: function(){
   browser.manage().timeouts().implicitlyWait(30000);
...
```

```
describe('Testing Implicit Wait',()=>{
   beforeAll(()=>{
       browser.waitForAngularEnabled(false);
       browser.get("http://www.target.com/");
   })
   it('should wait for a fixed amount of time when element not found',()=>{
   element(by.linkText('Categories2')).click();
   });
});
```

```
Testing Implicit Wait
  ✗ should wait for a fixed amount of time when element not found
    - Error: Timeout - Async callback was not invoked within timeout specified by jasmine.DEFAULT_TIMEOUT_INTERVAL.
    - Failed: No element found using locator: By(link text, Categories2)
```

2. EXPLICIT WAY (We prefer)

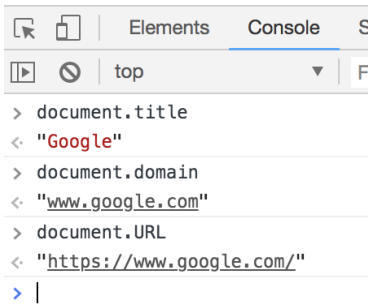
- What if we want to make sure element is in certain condition (displayed, checked, clickable etc) before protractor starts interacting with the element. This is when Explicit wait comes into picture.
- The explicit wait tells the protractor to **wait for certain conditions** or the **maximum time limit** before throwing an exception=error.
- Explicit Wait is a dynamic wait, which means it **will wait only if the condition is not met**.
- Explicit wait's **scope** is only the element it is sued for. (unlike whole script like in implicit wait)
- For Angular Page and Non-Angular waits, we use Expected Conditions.
- **Expected Conditions class** have several methods.
- ```
  var EC = protractor.ExpectedConditions;
  ```
  - ```
    browser.wait(EC.presenceOf(element(by.css('#GatewayTools'))),12000);
    ```
  - ```
    browser.wait(EC.visibilityOf(element(by.css('#GatewayTools'))),12000);
    ```
  - ```
    browser.wait(EC.invisibilityOf(element(by.css('#GatewayTools'))),12000);
    ```
  - ```
    browser.wait(EC.elementToBeClickable(element(by.css('#GatewayTools'))),12000);
    ```
  - ```
    browser.wait(EC.textToBePresentInElement($('#abc'),'foo'),12000);
    ```
  - ```
    browser.wait(EC.alertIsPresent(),12000);
    ```
  - ```
    browser.wait(EC.elementToBeSelected($('#abc'),'foo'),12000);
    ```




```
describe('Working with Explicit Waits', () => {
   var EC = protractor.ExpectedConditions;
   it('should wait for an element to be visible', () => {
       browser.waitForAngularEnabled(false);
       browser.get('http://automationpractice.com/index.php');
       browser.wait(EC.visibilityOf( $$('.homeslider-description .btn').
       get(2)),14000).then(function(){
           $$('.homeslider-description .btn').get(2).click();
       });
   });
});
```

JAVASCRIPT EXECUTOR

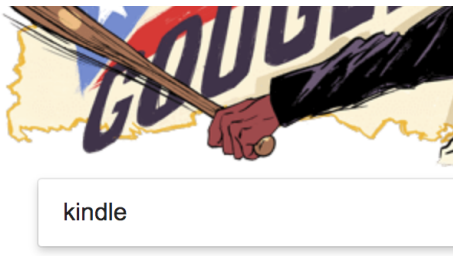
- JavaScriptExecutor is an Interface that helps to execute JavaScript through Selenium Webdriver.
  - (Irrespective of the Selenium language binding (Java, C#, Python etc.) you are using.)
- We should go forlaytaLtor only when we are not able to perform a particular task with our protractor:
  - Like sometimes, we may not be able click an element
- We have different javascript executor script method in protractor :
  - ```
    browser.executeScript("javascript command", "arguments");
    ```
  - ```
    browser.executeScriptWithDescription("javascript command", "description message", "arguments");
    ```
- Methods to open a page, get title, url and domain of the page.
  - ```
    browser.executeScript("window.location='http://automationpractice.com.index.php';");
    ```
  - ```
    browser.executeScript("return document.title");
    ```
  - ```
    browser.executeScript("return document.URL");
    ```
  - ```
    browser.executeScript("return document.domain");
    ```




```
describe('Practising with JS Executer',function() {
   beforeEach(()=>{
       browser.waitForAngularEnabled(false);
       browser.executeScript("window.location='https://www.google.com/';");
   });
   it('should get title/url/domain of the page ',function(){
       browser.executeScript("return document.title").then(text=>{
           console.log(text);
       });
       browser.executeScript("return document.URL").then(url=>{
           console.log('url is: ' +url)
       });
       browser.executeScript("return document.domain").then(domain=>{
           console.log('domain is: ' +domain)
       });
   });});
```

JS EXECUTOR METHODS

- Methods to click:
  - `browser.executeScript("document.getElementsByName('submit_search')[0].click()");`
  - `browser.executeScript('arguments[0].click();', element)};`
- sendKeys:
  - `browser.executeScript("document.getElementById('search_query_top').value='leather jacket'");`
- Scroll down:
  - `browser.executeScript("window.scrollBy(0,600)");`
  - `browser.executeScript("document.getElementById('defaults).scrollIntoView(true)");`
- Why is it not recommended as first choice ?
  - The **Protractor works similar way to an user**, protractor will perform the operation only if a user(physical person) can perform the operation on element.
  - We can rely on protractor, if protractor says that it cannot interact with an element, then is **physical person also cannot interact with the element**.
  - So, It is **better to fail the test case when protractor methods fails** instead of trying with javascript Executor. That is why manual check here is important when writing test case.




```
describe('Practising with JS Executer',function() {
   beforeEach(()=>{
       browser.waitForAngularEnabled(false);
       browser.executeScript("window.location='https://www.google.com/';");
   });
   it('should type',()=>{
       browser.executeScript("document.getElementsByName('q')[0].value='kindle'");
       browser.sleep(2000);
       browser.actions().sendKeys(protractor.Key.ENTER).perform();
       browser.sleep(2000);
       browser.executeScript("window.scrollBy(0,600)");
       browser.sleep(3000);
   });
});
```

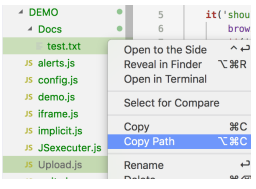
FILE UPLOAD

- File Upload:
  - `element(By.id('file-upload ')).sendKeys('C:/Users/emirzayev/Desktop/CTEK/README.md');`
- There is no need to simulate the clicking of the "Browse" button. WebDriver automatically enters the file path onto the file-selection text box of the **<input type="file">** element

```
function uploadFile (fileName) {
   // Append folder location of your current directory to your file to get full path.
   var absolutePath = path.resolve(__dirname,fileName);
   // Send file location for upload
   $$('input[type="file"]').get(0).sendKeys(absolutePath); };
```

- File Upload (When element not visible)
  - If you receive error that element is not visible.
  - You need to make the input visible using javascript executor. (You can add this to the above method.)

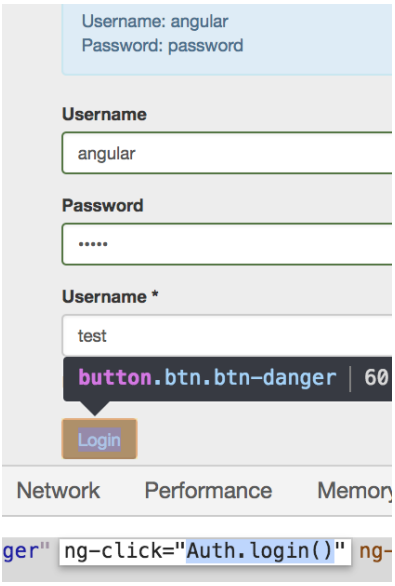
```
   // Unhide file input
browser.executeScript("arguments[0].style.visibility='visible'; ",fileElem.getWebElement());
$$('input[type="file"]').get(0).sendKeys(absolutePath);
Return; };
```




```
describe('Practising with JS Executer',function() {
   //when you see `require` --> means `importing`
   var path = require('path');
   it('should upload file',()=>{

browser.get('http://nervgh.github.io/pages/angular-file-upload/examples/simple');

        //__dirname = /Users/mac/Desktop/CtekProject/DEMO/
   $$('input[type="file"]').get(0).sendKeys(__dirname + '/Docs/test.txt');
        $('.btn-success').click();
   });
});
```

BY.ADDLOCATORS

- Add a locator to this instance of ProtractorBy. You can use **any attribute in the DOM** to create Custom Locators.
- This locator **can then be used** with element(**by.locatorCustomName(args)**).
- Practice: http://www.way2automation.com/angularjs-protractor/registeration/#/login
- Below method is created to locate any element with different ng-click values.

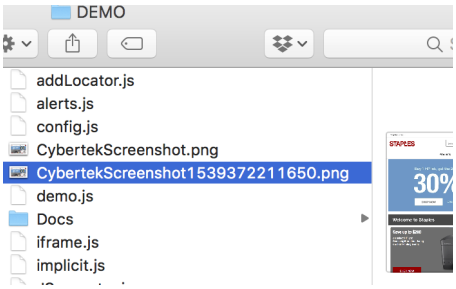



```
describe('Testing Adding Locators', ()=>{
   by.addLocator('ngClick', function(toState,parentelement) {
        //Below line tells to look for the element in the parent element if not in
then the entire document.
        var using = parentelement || document ;
        var prefixes = ['ng-click'];
        for (var p = 0; p < prefixes.length; ++p) {
            var selector = '*[' + prefixes[p] + '="' + toState + '"]';
            var inputs = using.querySelectorAll(selector);
            if (inputs.length) {
                return inputs;
            }
        }
   });
   it('should locate element using custom locator',()=>{

browser.get('http://www.way2automation.com/angularjs-protractor/registeration/#/login
');
        browser.sleep(3000);
        $('#username').sendKeys('angular');
        $('#password').sendKeys('password');
        $("[ng-model*='model']").sendKeys('Eagle');
        //This is our custom locator that we created.
        element(by.ngClick('Auth.login()')).click();
        browser.sleep(3000);
        expect(element(by.linkText('Logout')).isDisplayed()).toBe(true);
   })
});
```

- We can now call this method with our own parameters: **element(by.ngClick('Auth.login()')).click();** //very common attribute on Angular.

TAKING BASIC SCREENSHOT

- Screenshots are major part of our framework. They help with reporting, and viewing the results of your execution.
- Protractor provides the ability to take a screenshot with browser.takeScreenshot() function and save it.




```
describe('Taking Basic Screenshot',()=>{
   var fs = require('fs');
   it('should take screenshot', ()=>{
        browser.waitForAngularEnabled(false);
        browser.get('https://www.staples.com/');
        // for taking screenshot
        browser.takeScreenshot().then(function (png) {
            // create stream for writing the image
          var stream=fs.createWriteStream('CybertekScreenshot'+Date.now()+'.png');
            // write the stream to local file
            stream.write(new Buffer(png, 'base64'));
            // close the stream
            stream.end();
        });
   })
})
```

PROTRACTOR - BEAUTIFUL - REPORTER

- Another reporter available via npm.
- This reporters is visually better.
- `npm install protractor-beautiful-reporter --save-dev` **(sudo … for mac)**
- Also you can define if you want capture screenshots only from failed test cases:

  `takeScreenShotsOnlyForFailedSpecs: true`

to the config.js

```
let SpecReporter = require('jasmine-spec-reporter').SpecReporter;
var HtmlReporter = require('protractor-beautiful-reporter');

exports.config = {
  framework: 'jasmine',

  directConnect: 'true',
  specs: ['screenshot.js'],

  onPrepare: function(){
   browser.manage().window().maximize();
    jasmine.getEnv().addReporter(new SpecReporter({
      displayFailuresSummary: true,
      displayFailuredSpec: true,
      displaySuiteNumber: true,
      displaySpecDuration: true
    }));
    // Add a screenshot reporter and store screenshots:
    jasmine.getEnv().addReporter(new HtmlReporter({
    baseDirectory: 'report/screenshots'
    }).getJasmine2Reporter());
  }
};
```

```
// Other options we will use:
   // Add a screenshot reporter and store screenshots:
    jasmine.getEnv().addReporter(new HtmlReporter({
     baseDirectory: 'report/screenshots',
     preserveDirectory: false,
     screenshotsSubfolder: 'images',
     jsonsSubfolder: 'jsons',
     docName: 'Report.html'
    }).getJasmine2Reporter());
```

PAGE OBJECT MODEL

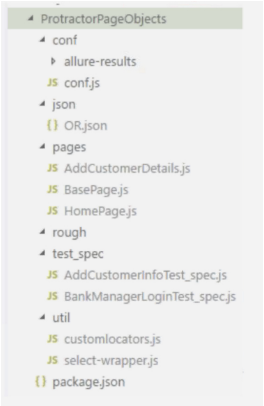
- The chief problem with script maintenance is that if 10 different scripts are using the same page element, with any change in that element, you need to change all 10 scripts. This is time consuming and error prone.
- A better approach to script maintenance is to create a separate file which would find web elements, fill them or verify them.
- This class can be reused in all the scripts using that element. In future, if there is a change in the web element, we need to make the change in just 1 file and not 10 different scripts.
- This approach is called **Page Object Model(POM)**. It helps make the code more
  - **Readable**
  - **Maintainable**
  - **Reusable**
- Page Object Model is a design pattern to create **Object Repository for web UI elements**.
- Under this model, for each web page in the application, there should be corresponding page file. (e.g: loginpage)
- We also store our reusable functions in this page-object files.
- The tests then use the functions of this page object class whenever they need to interact with that page of the UI.

FRAMEWORK STRUCTURE

- In POM Framework we organize our js files in folders
- We separate configuration files and utilities from test cases and object files
- Simple format of the Page Object model is:

```
var HomePage = function(){
      ...code...
} module.exports = new HomePage();
```

- The idea is to **move all the logic** required to interact with the page from the test **to the page objects**. Our suite now is more focused on the behavior of the page, than on how to select this or that element.



PAGE OBJECT FILE & SPEC FILE

```
require('../util/customlocators.js');
var HomePage = function(){
   this.loginAsCustomer = function(){
       element(by.partialButtonText("Customer")).click();
   };
   this.loginAsBankManager = function(){
       element(by.ngClick("manager()")).click();
       return require('./AddCustomerDetails.js');
   };
};
module.exports = new HomePage();
```

```
var base = require('../pages/BasePage.js');
var home_page = require('../pages/HomePage.js.');
describe("BankManager Login Test",function(){
   it("Login as Bank Manager",function(){
       base.navigateToURL(OR.testsiteurl);
       var customer = home_page.loginAsBankManager();
       var title = base.getPageTitle();
       expect(title).toBe("Protractor practice website
-Banking App");
       browser.sleep(3000);
   });
});
```

USING IT FROM SPEC FILE

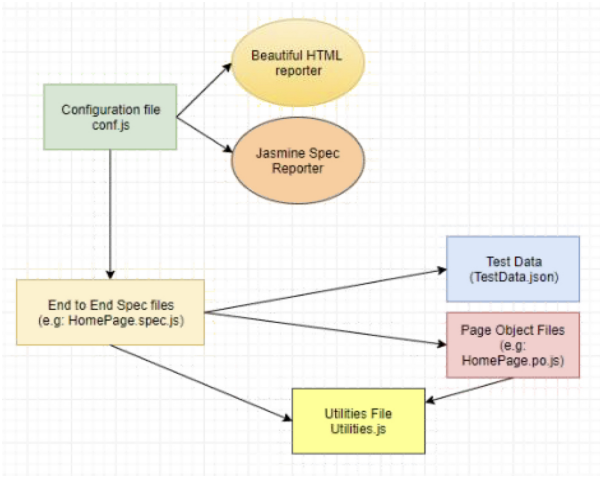
```
var toDoPage = require('../pages/toDoPage.js');
describe('Protractor Test',function(){
   it('should navigate to the AngularJS homepage',function(){
       toDoPage.go();
   });
   it('should let you add a new task',function(){
       toDoPage.addItem('New Task Item');
   });
});
```

POM DESIGN PATTERN

```
this.loginAsBankManager = function(){
   element(by.ngClick("manager()")).click();
   //By putting below line, we are confirming that this
method returns next page's object.
   //After login we are directed to the next page which is
AddCustomerDetails page.
   //If this method ended up in same page, then we would
write:
   // return this
   return require('./AddCustomerDetails.js');
};
```

```
it("Login as Bank Manager",function(){
   base.navigateToURL(OR.testsiteur1);
   var customer = home_page.loginAsBankManager();
   // Here we are instantiating an object from that method.
customer.gotoAddCustomer().addCustomerInfo(OR.locators.addcu
stomerdetailspage.fName,OR.locators.addcustomerdetailspage.l
Name,OR.locators.addcustomerdetailspage.pCode);
   var title = base.getPageTitle();
   expect(title).toBe("Protractor practice website - Banking
App");
   browser.sleep(3000);
}) ;
```

E2E BANKING APPLICATION

FRAMEWORK DIAGRAM	CREATING FRAMEWORK: BANKING APPLICATION
	1. Create package.json a. For managing Dependencies 2. Setup Conf.js file a. Includes SpecReporter b. Includes Beautiful Reporter 3. Create BasePage.js a. This includes global attributes we will be using throughout the application b. Such as Application URL. 4. Create Utilities directory a. Files include functions that will be used across entire application. 5. Create PageObject files a. One for each page 6. Create json file a. Includes test-data 7. Create spec files a. Includes actual test case specs.

PACKAGE.JSON

- Package.json is a **build tool** available for npm packages, it can **install the required packages** to run a particular system.
    - Basically, instead of installing all of the components of our framework separately, we put them in this file and install them in one shot.
    - It will install npm packages **only if** the mentioned packages are not present in system
- It is used to install and maintain project dependencies.
    - Such as: protractor version, jastnine-reporter version etc.
- Basically whenever you try to install a npm module, it will look for package.json file to make entry of the installation, but there is no such so we got the warning.
- To create package.json file: in the project folder we can run: **npm init**
    - This will prompt us to input values.
- After we have our package,json file, we run "npm install" to install all the dependencies listed in there.
    - **npm install**.
- This becomes especially useful, when we are installing our test framework on remote systems, Erg: on jenkins server etc.

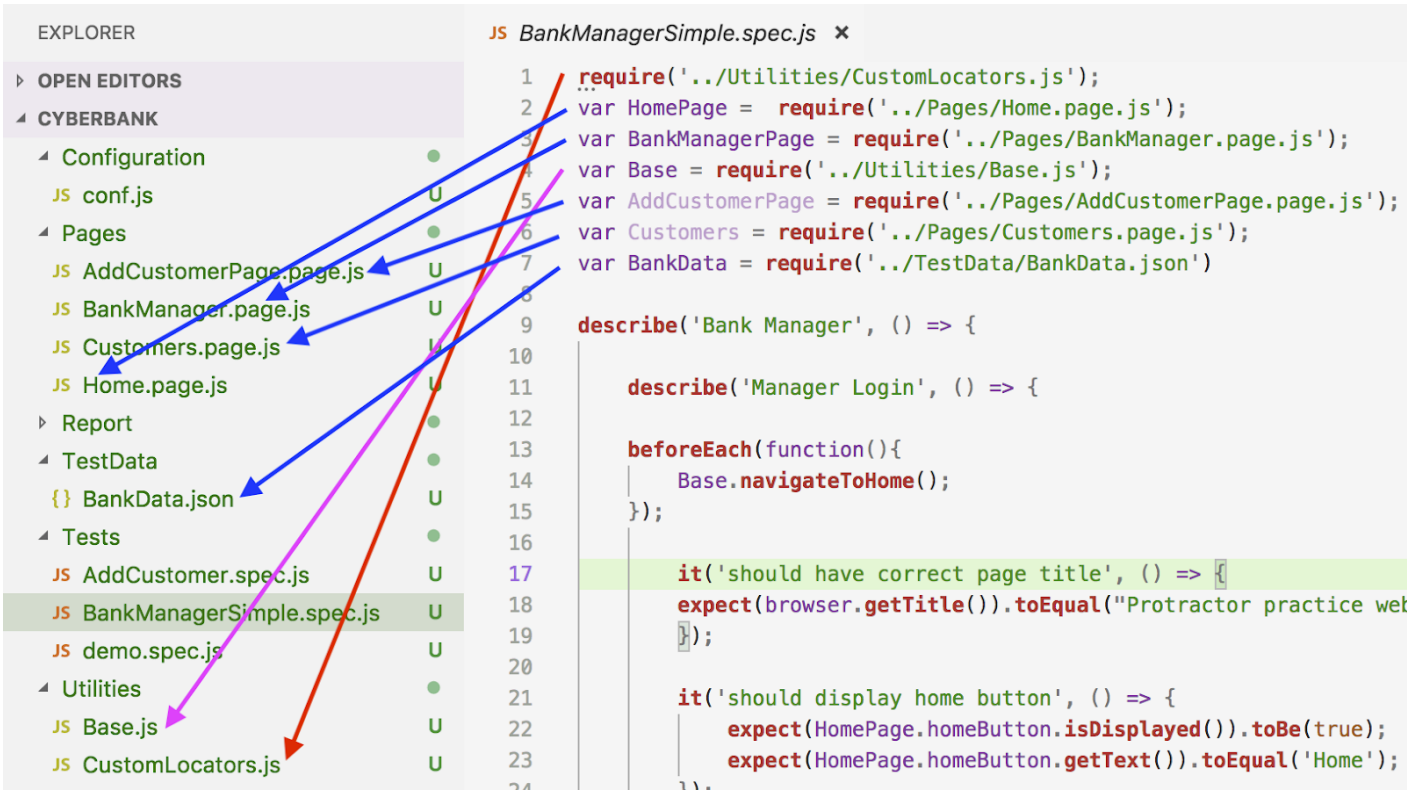
TERMINAL:

```
macs-MBP:CyberFramework mac$ sudo npm init
Password:
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.
Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (cyberframework) cyberframework
version: (1.0.0)
description: This is for CyberFramework.
entry point: (index.js)
test command: protractor conf.js
git repository:
keywords: protractor
author: CyberStar
license: (ISC)
→ → → → → → → → → → → → → → → → → → → → → → → → → → → → → → → → → → →
```

```
About to write to /Users/mac/Desktop/CyberFramework/package.json:
{
  "name": "cyberframework",
  "version": "1.0.0",
  "description": "This is for CyberFramework.",
  "main": "index.js",
  "scripts": {
    "test": "protractor conf.js"
  },
  "keywords": [
    "protractor"
  ],
  "author": "CyberStar",
  "license": "ISC"
}
Is this OK? (yes) yes
macs-MBP:CyberFramework mac$ sudo npm install protractor --save-dev
macs-MBP:CyberFramework mac$ sudo npm install jasmine-spec-reporter
--save-dev
macs-MBP:CyberFramework mac$ sudo npm install
protractor-beautiful-reporter --save-dev
```

Conf.js

```
let SpecReporter = require('jasmine-spec-reporter').SpecReporter;
var HtmlReporter = require('protractor-beautiful-reporter');
exports.config = {
   directConnect : true,
   capabilities: {
       browserName: 'chrome'
   },
     //specs: ['../Tests/BankManagerSimple.spec.js'],
   suites : {
       smoke: ['../Tests/BankManagerSimple.spec.js','../Tests/demo.spec.js'],
       regression: ['../Tests/*.spec.js']//* means go and run everything
    },

onPrepare: function () {
   browser.driver.manage().window().maximize();
   jasmine.getEnv().addReporter(new SpecReporter({
       displayFailuresSummary: true,
       displayFailuredSpec: true,
       displaySuiteNumber: true,
       displaySpecDuration: true,
       showstack: false
   }));
     // Add a screenshot reporter and store screenshots to `/tmp/screenshots`:
   jasmine.getEnv().addReporter(new HtmlReporter({
       baseDirectory: 'report/screenshots',
       preserveDirectory: false,
       screenshotsSubfolder: 'images',
       jsonsSubfolder: 'jsons',
       docName: 'CyberBank-Report.html'
   }).getJasmine2Reporter());
},
jasmineNodeOpts: {
   showColors: true,
   defaultTimeoutInterval: 30000,
   print: function() {}
} };
```



UTILITIES

CustomLocators.js

```
//we create this file for ng-click locator
var customlocators = function() {
    by.addLocator('ngClick', function(toState,parentelement) {
       var using = parentelement || document ;
       var prefixes = ['ng-click'];
       for (var p = 0; p < prefixes.length; ++p) {
                var selector='*['+prefixes[p]+'="'+toState+ '"]';
                var inputs = using.querySelectorAll(selector);
                if (inputs.length) {
                    return inputs;
                }
       }
   });
}
module.exports = new customlocators();
```

Base.js

```
var Base = function(){
   this.homeUrl =
'http://www.way2automation.com/angularjs-protractor/banking/#/login';
   this.navigateToHome = function(){
       browser.get(this.homeUrl);
   };
}
module.exports = new Base();
```

PAGES

Home.page.js

```
//to import the ng-click;
require('../Utilities/CustomLocators.js');
//to manage the changes of the locators; we create this;
var HomePage = function(){
   this.homeButton = $('button.home');
   this.pageHeader = $('.mainHeading');
   this.managerLoginButton = element(by.ngClick('manager()'));
};
module.exports = new HomePage();
```

BankManager.page.js

```
require('../Utilities/CustomLocators.js');
var BankManagerPage = function(){
   this.addCustomerButton = element(by.ngClick('addCust()'));
   this.openAccountButton = element(by.ngClick('openAccount()'));
   this.customersButton = element(by.ngClick('showCust()'));
};
module.exports = new BankManagerPage();
```

AddCustomerPage.page.js

```
require("../Utilities/CustomLocators.js");
var BankManagerPage = require('./BankManager.page');

var AddCustomerPage = function(){

   this.formLabels = $$('.form-group>label');
   this.firstNameInputBox = element(by.model('fName'));
   this.lastNameInputBox = element(by.model('lName'));
   this.postalCodeInputBox=element(by.model('postCd'));
   this.formRequiredFields=element.all(by.css('input:required'));
   this.formAddCustomerButton=$('.btn-default');
   this.customerForm = element(by.name('myForm'));

   this.goToAddCustomer = function(){
       BankManagerPage.addCustomerButton.click();
   };
}
module.exports = new AddCustomerPage();
```

Customers.page.js

```
require("../Utilities/CustomLocators.js");

var Customers = function(){

this.table=element(by.xpath("//table[contains(@class,'table-bordered')]"))
;

   this.getLastRowValue=(function(columnNumber){
      return
this.table.element(by.xpath("//tbody/tr[last()]/td["+columnNumber+"]"));
   });
};
module.exports = new Customers();
```

BankManagerSimple.spec.js

```
require('../Utilities/CustomLocators.js');
var HomePage =  require('../Pages/Home.page.js');
var BankManagerPage = require('../Pages/BankManager.page.js');
var Base = require('../Utilities/Base.js');
var AddCustomerPage = require('../Pages/AddCustomerPage.page.js');
var Customers = require('../Pages/Customers.page.js');
var BankData = require('../TestData/BankData.json')

describe('Bank Manager', () => {
   describe('Manager Login', () => {
   beforeEach(function(){
       Base.navigateToHome();
   });
       it('should have correct page title', () => {
       expect(browser.getTitle()).toEqual("Protractor practice website - Banking App");
       });
       it('should display home button', () => {
           expect(HomePage.homeButton.isDisplayed()).toBe(true);
           expect(HomePage.homeButton.getText()).toEqual('Home');
       });
       it('should display page header', () => {
           expect(HomePage.pageHeader.isDisplayed()).toBe(true);
           ////////////////////////////////////////////////////////////
           //expect(HomePage.pageHeader.getText()).toEqual('XYZ Bank');
           expect(HomePage.pageHeader.getText()).toEqual(BankData.appData.bankName);
       });
       it('should display login option for Bank Manager', () => {
           expect(HomePage.managerLoginButton.isDisplayed()).toBe(true);
           expect(HomePage.managerLoginButton.getText()).toEqual('Bank Manager Login');
       });
       it('should stay at the homepage when Home Button is clicked', () => {
           HomePage.homeButton.click();
           expect(browser.getTitle()).toEqual('Protractor practice website - Banking App');
           expect(HomePage.managerLoginButton.getText()).toEqual('Bank Manager Login');
       });
       it("should login as Bank Manager",function(){
           HomePage.managerLoginButton.click();
           expect(BankManagerPage.addCustomerButton.isDisplayed()).toBe(true);
       }) ;
       it('should display  options for manager', () => {
           HomePage.managerLoginButton.click();
           expect(BankManagerPage.addCustomerButton.isDisplayed()).toBe(true);
           expect(BankManagerPage.openAccountButton.isDisplayed()).toBe(true);
           expect(BankManagerPage.openAccountButton.getText()).toEqual('Open Account');
           expect(BankManagerPage.customersButton.isDisplayed()).toBe(true);
       });
       it('should navigate back to home page from Manager Login Page', () => {
           HomePage.managerLoginButton.click();
           HomePage.homeButton.click();
           expect(HomePage.managerLoginButton.getText()).toEqual('Bank Manager Login');
       });
   });
```

AddCustomer.spec.js

```
require('../Utilities/CustomLocators.js');
var HomePage =  require('../Pages/Home.page.js');
var BankManagerPage = require('../Pages/BankManager.page.js');
var Base = require('../Utilities/Base.js');
var AddCustomerPage = require('../Pages/AddCustomerPage.page.js');
var Customers = require('../Pages/Customers.page.js');
var BankData = require('../TestData/BankData.json')

describe('Add Customer', function() {
   describe('Adding a Customer', () => {
       beforeAll(function(){
           Base.navigateToHome();
           HomePage.managerLoginButton.click();
           AddCustomerPage.goToAddCustomer();
       });
       it('should display form for Adding Customer', () => {
           expect(AddCustomerPage.customerForm.isDisplayed()).toBe(true);
           expect(AddCustomerPage.formLabels.count()).toEqual(3);
       });
       it('should list all the labels', () => {
           expect(AddCustomerPage.formLabels.get(0).getText()).toEqual('First Name :')
           expect(AddCustomerPage.formLabels.get(1).getText()).toEqual('Last Name :')
           expect(AddCustomerPage.formLabels.get(2).getText()).toEqual('Post Code :')
       });
       it('should require first name', () => {
           expect(AddCustomerPage.formRequiredFields.get(0).getAttribute('required')).toEqual('true');
       });
       it('should require last name', () => {
           expect(AddCustomerPage.formRequiredFields.get(1).getAttribute('required')).toEqual('true');
       });
       it('should require post code', () => {
           expect(AddCustomerPage.formRequiredFields.get(2).getAttribute('required')).toEqual('true');
       });
       it('should add customer', () => {
               //AddCustomerPage.firstNameInputBox.sendKeys('Jeff');
               for (var i = 0; i < BankData.customers.length; i++) {
                   AddCustomerPage.goToAddCustomer();
                   AddCustomerPage.firstNameInputBox.sendKeys(BankData.customers[i].fName);
                   AddCustomerPage.lastNameInputBox.sendKeys(BankData.customers[i].lName);
                   AddCustomerPage.postalCodeInputBox.sendKeys(BankData.customers[i].pCode);
                   AddCustomerPage.formAddCustomerButton.click();
                   expect(browser.switchTo().alert().getText()).toContain('added successfully');
                   browser.switchTo().alert().accept();
                   BankManagerPage.customersButton.click();
                   expect(Customers.getLastRowValue(1).getText()).toEqual(BankData.customers[i].fName);
                   expect(Customers.getLastRowValue(2).getText()).toEqual(BankData.customers[i].lName);
                   expect(Customers.getLastRowValue(3).getText()).toEqual(BankData.customers[i].pCode);
               }
       });
   });
});
```

READING DATA and LOCATORS THROUGH JSON FILE

- We an use json file to store and retrieve :
    - Locators
    - test data
- To read data from this file we put in our spec file:
  var objects= require('Objects.json`);
  browser.get(Objects.testsiteurl)
  element(by.xpath(**Objects.locators.loginpage.username**)) .sendKeys(**Objects.userdetails.username1**);

BankData.json

```
{
   "appData":{
       "bankName": "XYZ Bank",
       "bankManagerLoginButtonText": "Bank Manager Login",
       "addCustomerButtonText": "Add Customer",
       "OpenAccountButtonText": "Open Account",
       "customersButtonText": "Customers"
   },
   "customers":[
       {
           "fName":"Mark",
           "lName":"Zuckerberg",
           "pCode":"21005",
           "accountNumber":""
       },
       {
           "fName":"Jack",
           "lName":"Ma",
           "pCode":"334455",
           "accountNumber":""
       },
       {
           "fName":"Jeff",
           "lName":"Bezos",
           "pCode":"221155",
           "accountNumber":""
       }
   ],
   "locators":{
       "homePage":{
           "bankManagerLogin":"manager()"
       }
   }
}
```

EXECUTING TEST CASES

There are several ways of executing test cases:

1. Executing just one test case: specs: [‘functional/loginTest.spec.js’]
2. Executing all spec files in the same folder: [functional/*.spec.js]
3. Executing test suites:
    a. Suite is a group of test cases or specs. Protractor provides a feasibility to run group of test cases or specs by dividing them as suites.

suites: {
smoke: ['./smoke/*.spec.js'],
regression: ['./regression/*.spec.js'],
functional: ['./functional/*.spec.js'],
all: ['./*/*.spec.js'],
seLected: ['./functional/addcustomer.spec.js','./regression/openaccount.spec.js'],
}

To run the suites, we use command: **protractor conf.js --suite smoke, selected**
We can also run different specs by passing spec parameter in command: **protractor conf.js --spec logintTest.spec.js**

JASMINE DATA PROVIDER

Someone created library and (downloaded like npm) we use it in our project.

sudo npm install jasmine-data-provider --save-dev

```
require('../Utilities/CustomLocators.js');
var HomePage =  require('../Pages/Home.page.js');
var BankManagerPage = require('../Pages/BankManager.page.js');
var Base = require('../Utilities/Base.js');
var AddCustomerPage = require('../Pages/AddCustomerPage.page.js');
var Customers = require('../Pages/Customers.page.js');
var BankData = require('../TestData/BankData.json')

var using = require('jasmine-data-provider')


fdescribe('Jasmine Data Provider ', () => {

   beforeAll(function(){
       Base.navigateToHome();
       HomePage.managerLoginButton.click();
       AddCustomerPage.goToAddCustomer();
   });

   function dataProvider(){
       return [
       {fName:"Elon", lName:'Musk',pCode:'334455'},
       {fName:"Warren", lName:'Buffet',pCode:'112233'},
       {fName:"Amanico", lName:'Ortega',pCode:'112233'}
   ]};

   using(dataProvider, function(data){ //instead of for loop --> using
       it('should add customer: '+data.fName+' '+data.lName, () => {
           AddCustomerPage.firstNameInputBox.sendKeys(data.fName);
           AddCustomerPage.lastNameInputBox.sendKeys(data.lName);
           AddCustomerPage.postalCodeInputBox.sendKeys(data.pCode);
           AddCustomerPage.formAddCustomerButton.click();
           expect(browser.switchTo().alert().getText()).
           toContain('Customer added successfully with customer id :');
           browser.switchTo().alert().accept();
           BankManagerPage.customersButton.click();
           expect(Customers.getLastRowValue(1).getText()).toEqual(data.fName);
           expect(Customers.getLastRowValue(2).getText()).toEqual(data.lName);
           expect(Customers.getLastRowValue(3).getText()).toEqual(data.pCode);
           AddCustomerPage.goToAddCustomer();
       });
   });
});
```

READING & WRITING FROM EXCEL FILES

- We can keep our test data in an excel, as well as json files
  sudo npm install exceljs --save-dev