Functional Automation using Protractor & Jasmine framework with Selenium.

# Assumptions:

* User knows the basics of Selenium (Either Java/c#).
* User has understanding of Page Object model design pattern.
* User has understanding of Selenium Architecture.
* User knows about Element locators available in Selenium.
* Understanding of Asynchronous Programming is a plus.

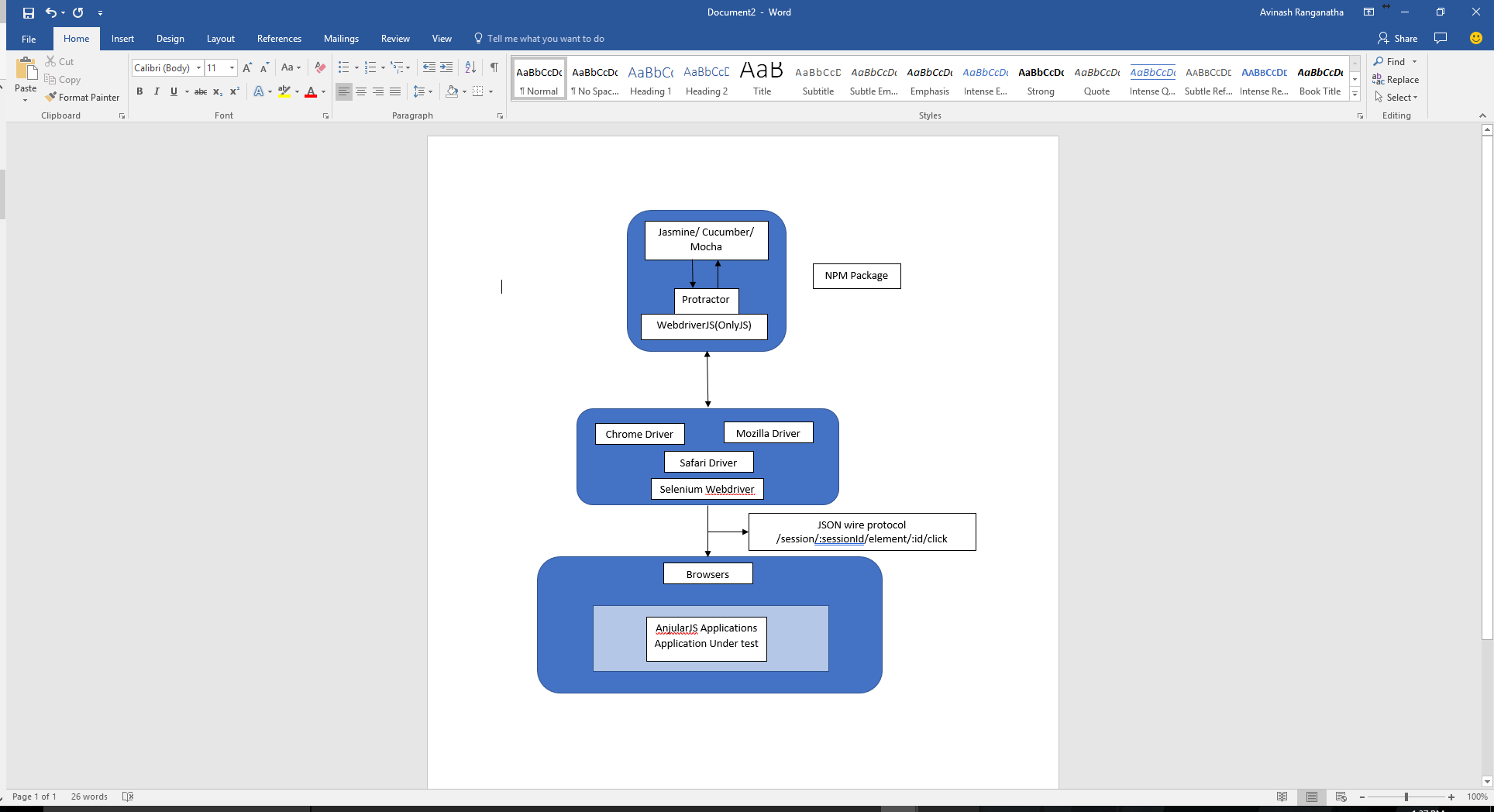
## What is Protractor? How is it different from Selenium?

Protractor is an end to end functional Automation framework which is built on Selenium webdriver and automates web applications developed using AngularJS/ReactJS. Protractor uses Javascript to communicate with selenium using WebdriverJS. One can use Protractor to automate applications which are built using non-AngularJS as well. One reason to use Protractor with AngularJS is Protractor handles asynchronous calls very well. User does not have to give specific waits for the objects/webpage to get loaded which can be painful to handle in Selenium synchronous calls. It provides additional web element locators with selenium Selenium. So basically Protractor is wrapper for webdriverJS so we get all the features of webdriverJS(selenium classes and metods) as well as additional features of Protractor.

## Similarities and differences between Protractor and Selenium with Java/C#



## Protractor and webdriver architecture



## Installations

1. Install Node.JS
2. Install Protractor using command: npm install protractor -g
3. Install/Update webdriver manager: webdriver-manager update
4. Install latest version of NetBeans/phpStorm

Set up conf.js file

Conf.js file is the configuration file which controls the execution, where reports can be configured, environment details and browsers can be configured for a run. It looks something like this

exports.config = {

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

baseUrl: ‘https://confr.com/’, // application URL for example using https://confr.com/

params: {

username: ‘userName’, //user name

password : ‘Password’, //password

invalidPassword: ‘invalidPassword’, //invalid Password

},

capabilities: {

'browserName': 'chrome' //Browser on which tests has to run

},

specs: [spec/\*spec.js'],

jasmineNodeOpts: {

showColors: true

},

onPrepare: function() {

browser.driver.manage().window().maximize(); // Launches browser and Maximises the window

}

};

## Object Locators

Apart from the usual element by xpath, by id, by name, by css etc, Protractor offers additional bindings like by model, by repeater, by buttontext, by bindings, by partialbuttontext

Usage

Click [here](http://www.protractortest.org/#/api?view=ElementFinder) to get the list

* Application

<button ngmodel=’mytestmodel’, name =’loginButton’, id =’id1’>Login</button>

* In script

var loginButton = element(by.model(‘’mytestmodel’)); //returns element which has ngmodel =’mytestmodel’

var loginButton = element(by.buttontext(‘Login’));//assuming there are no other buttons with this text.

//If there are, then the first element will be taken by default after prompting warning to user

* Dealing with Multiple elements

Let’s assume the page has multiple elements with same definition which user has to verify/use in the script

<div name =”abc”>

<textbox ngmodel=’mytxtmodel’> </ textbox >

</div>

<div name “xyz”>

<textbox ngmodel=’mytxtmodel’> </ textbox >

</div>

In script

element.all(by.model(‘’mytxtmodel’’).then(function(mytxtBoxes){ // element.all returns a promise. We’ll discuss about

//promises further section

mytxtBox[0].sendKeys(“text1”); //enters text1 in the first text box

mytxtBox[1].sendKeys(“text2”); //enters text2 in the second text box

})

## Executing Events

Let’s take the same example of button and textbox in the application

Example 1

Application

<button ngmodel=’mytestmodel’, name =’loginButton’, id =’id1’>Login</button>

In script

var loginButton = element(by.model(‘’mytestmodel’)); //returns element which has ngmodel =’mytestmodel’

loginButton.Click(); //Simulates Click Action on the button

Example 2

Application

<div name “xyz”>

<textbox ngmodel=’mytxtmodel’, id=’accountID’> </ textbox >

</div>

In script

var txtBox1 = element(by.id(‘’accountID’’)); //returns element with id=’accountID’

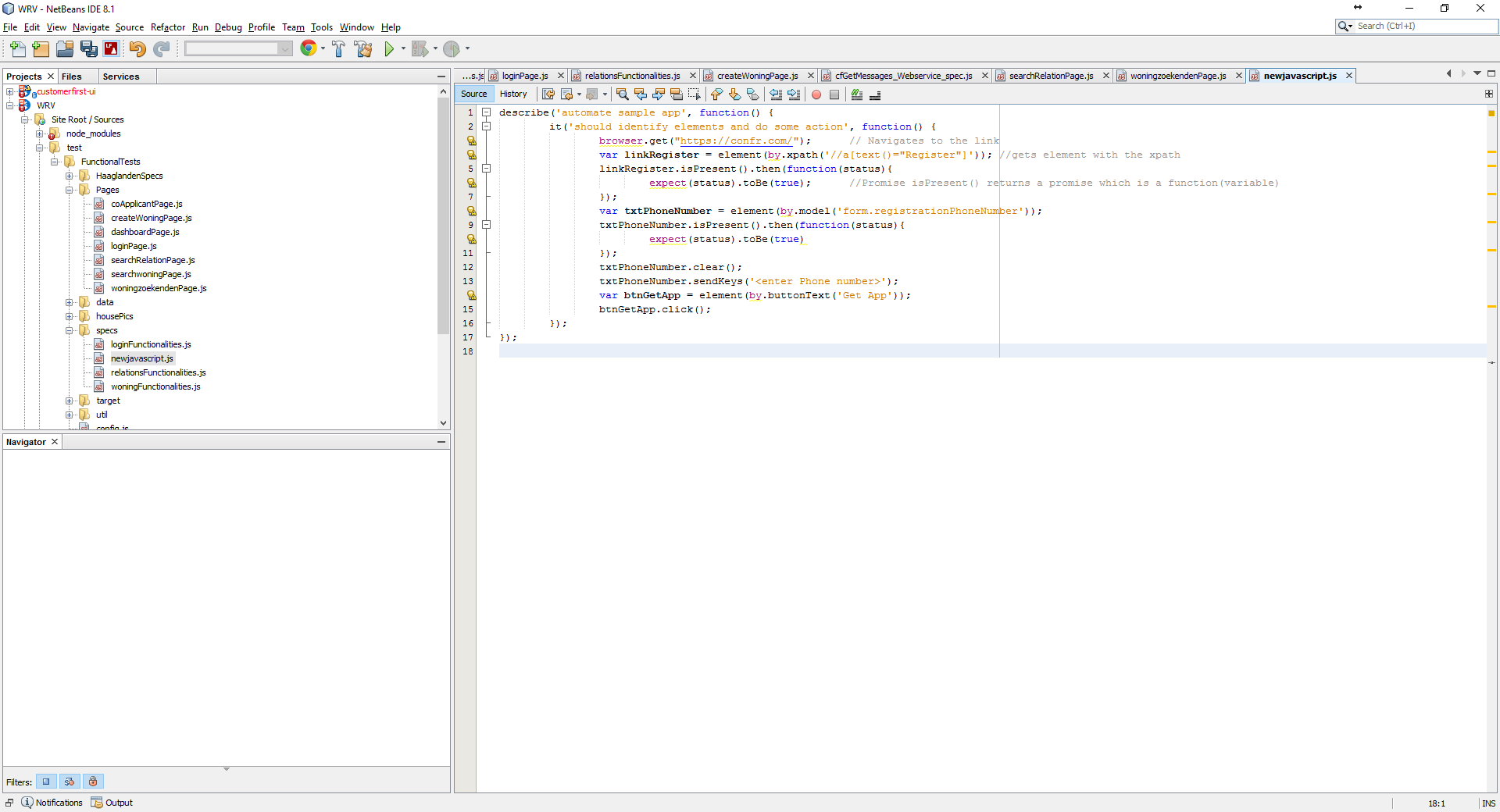
txtBox1.sendKeys(“Sample Text”); //Simulates sendKeys on the element

## Writing first test case using Jasmine

I’m using an example of a sample application <https://confr.com/> mentioned in conf.jsFile

Save this as testcase1spec.js and place it under a folder called as spec

The test case would look something like this



Running the test case

Open the terminal

Start webdriver Manager:

Run webdriver-manager start

Execute test case using configuration file

protractor test/e2e/config.js //this is the relative path to your config.js file.

## Handling promises

This is the tricky part of Asynchronos programming.

Behind the scenes of WebDriver, every call that interacts with the browser, such as get(), element(), sendKeys(), and click(), isPresent() is being scheduled and pushed onto the WebDriver ControlFlow allowing us to worry about using .then() on the resulting promises (unless we specifically want the result returned by the call).

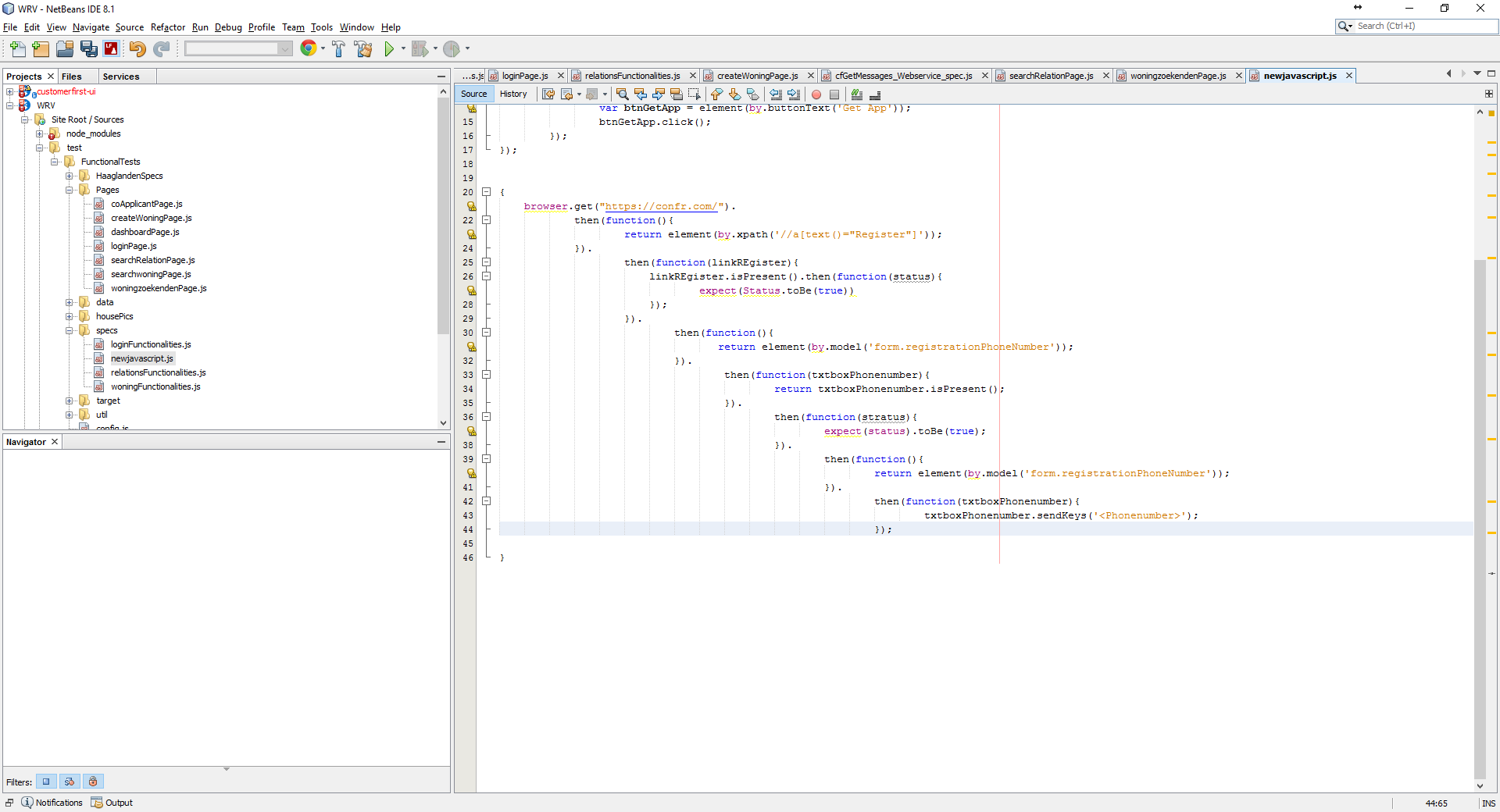
Calls like .get(), .getText(), isEnabled(), isPresent(), element.all(bylocator) would return promise. So user can use

.then(function(returnedPromise){

//Do something with the returned promise;

})

What truly happens in the above example:



## Control flow

Any time an asynchronous call is invoked, it’s put in the queue where it will wait for all previous commands to complete prior to execution.

Since control flow acts like a cashier in fast food where he serves multiple customers one after the other(there’s no scope for small talk). The control takes the order, promise to serve the order and asks the customer to move aside to take the next customer’s order. Eventually each customer gets served in the order queue.

Basically the method element(by.xpath('//a[text()="Register"]')) would only run once the browser.get() promise is resolved. More details can be found [here](https://spin.atomicobject.com/2014/12/17/asynchronous-testing-protractor-angular/).

Will explain Page object model in future blogs

# Conclusion:

There are considerable advantages of using Protractor over Java/Selenium. They are as follows:

1. Typecasting: This can be a major headache in object oriented programming. When a function/method returns something, the calling function needs to handle the datatype.
2. Asynchronous Programming: Though it might be a Little tricky to understand at first, once the user gets the hang of it, it’s pretty simple. The control doesn’t have time for small like – “what’s your data type”? “have you converted the data”?, “can you send me only the type of object I want”. It can become a huge headache sometimes. This makes the execution faster.
3. User does not have to write explicit/implicit waits. The control flow waits for the page to load (rest calls to finish).
4. Simple to code and oo need to create objects, initialize it as static. Everything is managed internally so that memory is utilized. Come to think of it, programmers are bored with OOP.
5. Thousands of NPM modules available to utilize. Ex reporter, name and password generators which can be used in data generation.