

Web Driver IO

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

WebdriverIO is used to automate any tests designed for a present-day application developed in React, Angular, Polymerer Vue.js, and so on. This tutorial shall provide you with a thorough insight on WebdriverIO and its different terminologies. The tutorial contains practical examples on all important topics.

Audience

This tutorial is designed for professionals working in software testing who want to hone their skills on a robust automation testing tool like WebdriverIO. It is implemented in Node.js and comes under the umbrella of Selenium.

Prerequisites

Prior to going through this tutorial, you should have a fair knowledge on JavaScript and object oriented programming concepts. Besides, a good understanding of basics in testing is important to proceed with this tutorial.

Copyright & Disclaimer

© Copyright 2021 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial	i
Audience.....	i
Prerequisites.....	i
Copyright & Disclaimer	i
Table of Contents	ii
1. WebdriverIO – Introduction	1
2. WebdriverIO – Prerequisite.....	4
3. WebdriverIO – Architecture	6
4. WebdriverIO – Getting Started with NodeJS.....	7
5. WebdriverIO – Installation of NPM	11
Create NPM Project.....	11
6. WebdriverIO – VS Code Installation	13
7. WebdriverIO – Package.json.....	15
8. WebdriverIO – Mocha Installation.....	18
9. WebdriverIO – Selenium Standalone Server Installation	20
10. WebdriverIO – Configuration File generation	22
Create Mocha Spec File	24
11. WebdriverIO – VS Code Intellisense	28
Add intellisense to VS Code.....	28
12. WebdriverIO – Wdio.conf.js file	31
13. WebdriverIO – Xpath Locator	35
Xpath Locator with Text	38
14. WebdriverIO – CSS Locator.....	41
15. WebdriverIO – Link Text Locator	44
Partial Link Text Locator	45
16. WebdriverIO – ID Locator	48

17. WebdriverIO — Tag Name Locator	51
18. WebdriverIO — Class Name Locator	53
19. WebdriverIO — Name Locator	55
20. WebdriverIO — Expect statement for assertions	57
Assertions applied to browsers	57
Assertions applied on elements	57
Assertions applied to mock objects.....	61
21. WebdriverIO — Happy path flow	64
22. WebdriverIO — General Browser Commands	66
23. WebdriverIO — Handling Browser Size	71
24. WebdriverIO — Browser Navigation Commands	73
25. WebdriverIO — Handling Checkboxes and Dropdowns	76
Handling Dropdowns	78
26. WebdriverIO — Mouse Operations	82
27. WebdriverIO — Handling Child Windows/Pop ups	84
28. WebdriverIO — Hidden Elements	87
29. WebdriverIO — Frames	89
30. WebdriverIO — Drag and Drop	92
31. WebdriverIO — Double Click	95
32. WebdriverIO — Cookies	97
Methods for Cookies	97
33. WebdriverIO — Handling Radio Buttons	102
34. WebdriverIO — Chai Assertions on webelements	105
35. WebdriverIO — Multiple Windows/Tabs	111
Methods for Multiple Windows	111
36. WebdriverIO — Scrolling Operations	114
37. WebdriverIO — Alerts	116
Methods for Alerts	116

38. WebdriverIO — Debugging Code	119
Enable Debugging.....	119
39. WebdriverIO — Capturing Screenshots.....	123
40. WebdriverIO — JavaScript Executor.....	125
Actions with Javascript Executor	125
41. WebdriverIO — Waits	126
42. WebdriverIO — Running Tests in Parallel.....	129
43. WebdriverIO — Data Driven Testing	133
44. WebdriverIO — Running Tests from command-line parameters.....	138
45. WebdriverIO — Execute Tests with Mocha Options.....	143
46. WebdriverIO — Generate HTML reports from Allure	146

1. WebdriverIO – Introduction

WebdriverIO helps to automate any tests designed for a present-day application developed in React, Angular, Polymerer Vue.js, and so on. Besides, it can also be used in Android and iOS platforms.

WebdriverIO is implemented in Node.js and the automation code is written in JavaScript. It comes under the umbrella of Selenium. All the capabilities of Selenium are also available in WebdriverIO, along with certain accessory assertions available for validations.

Now-a-days, the front end of the majority of applications is developed with the JavaScript frameworks like React, Angular, and so on. WebdriverIO is really useful for testing these applications.

This is because WebdriverIO coding is also done in JavaScript. This tool falls under the roof of Selenium and also there are some additional APIs. If we are aware of Selenium, then gaining knowledge in WebdriverIO is a simple task.

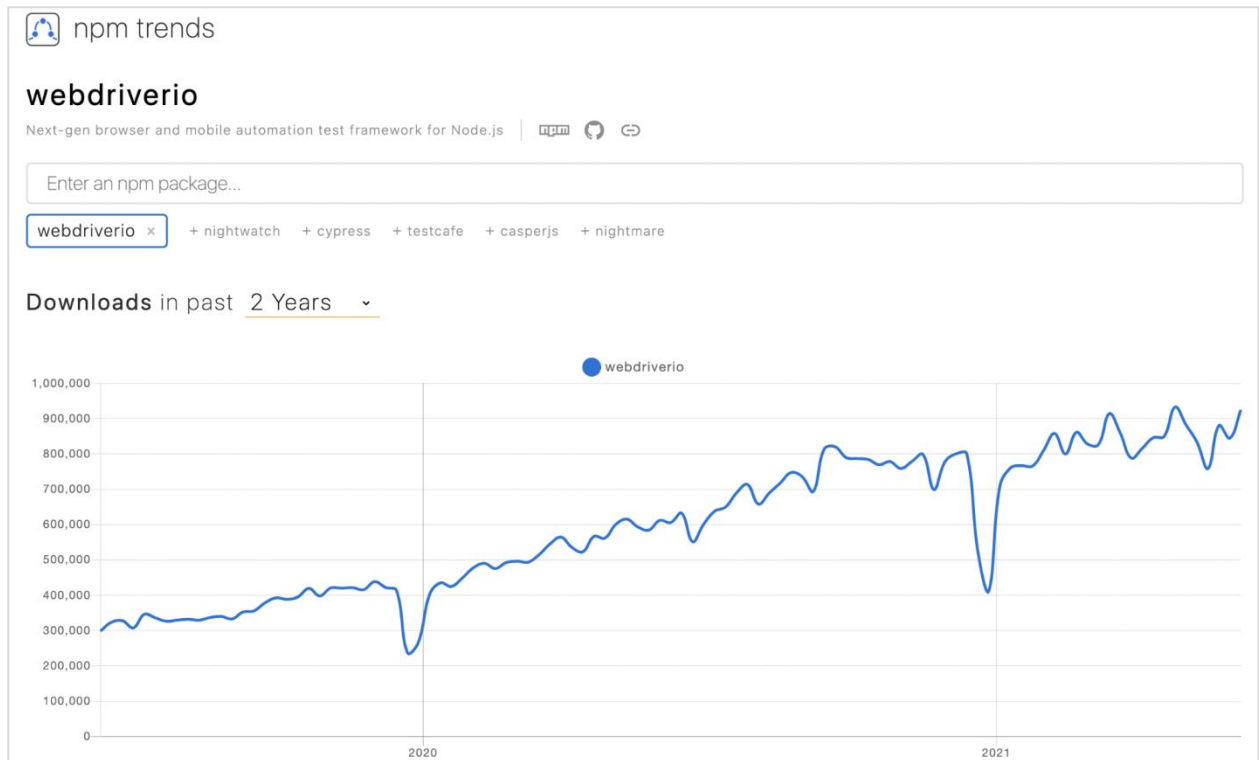
WebdriverIO can also be used for testing normal applications but if we are using WebdriverIO for verifying any application implemented in React, Angular, Polymerer Vue.js, and so on, we can enjoy an additional edge in building a robust framework.

If we are creating Selenium tests in JavaScript, then WebdriverIO should be the choice. There are other tools like Cypress which is based on the JavaScript framework but it does not fall under the umbrella of Selenium.

If we follow the npm trends for WebdriverIO downloads for the last few years, we shall observe an upward trend towards the use of WebdriverIO available from the link mentioned below:

<https://www.npmtrends.com/webdriverio>

The following screen will appear on your computer:



Reports

Some of the reports generated in WebdriverIO are as follows:

- Allure
- Spec
- JUnit
- HTML
- JSON
- Cucumber JSON

Services

Some of the services offered by WebdriverIO are as follows:

- Appium
- Docker
- Selenium Standalone
- ChromeDriver
- Firefox Profile
- DevTools

Testing Frameworks

Some of the testing frameworks supported by WebdriverIO are as follows:

- Cucumber

- Jasmine
- Mocha

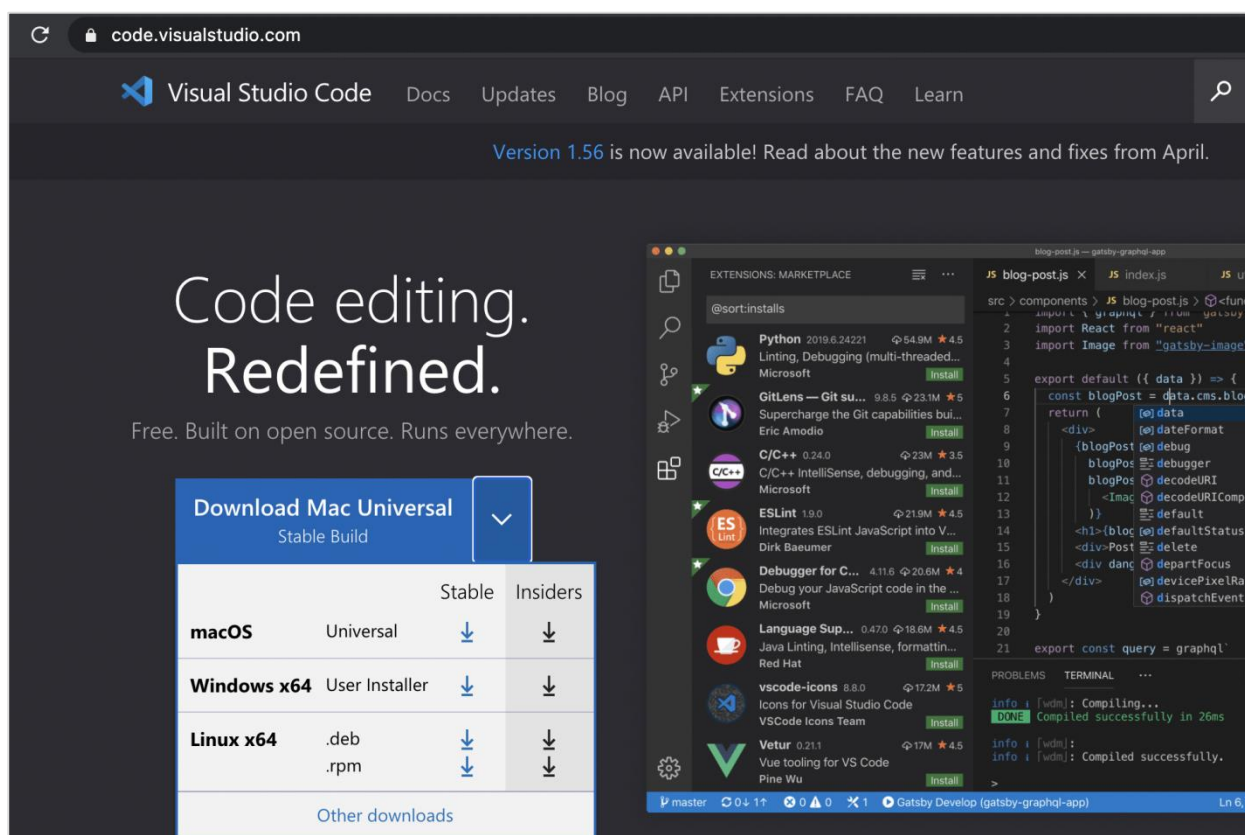
2. WebdriverIO — Prerequisite

As a prerequisite for WebdriverIO, we need to have an editor to write the JavaScript code. For this, we can use the Visual Studio Code. We can download it from the below link:

<https://code.visualstudio.com/>

Step 1: Based on the local operating system we have for example - macOS, Linux or Windows, we need to select the link for download.

The following screen will appear on your computer:

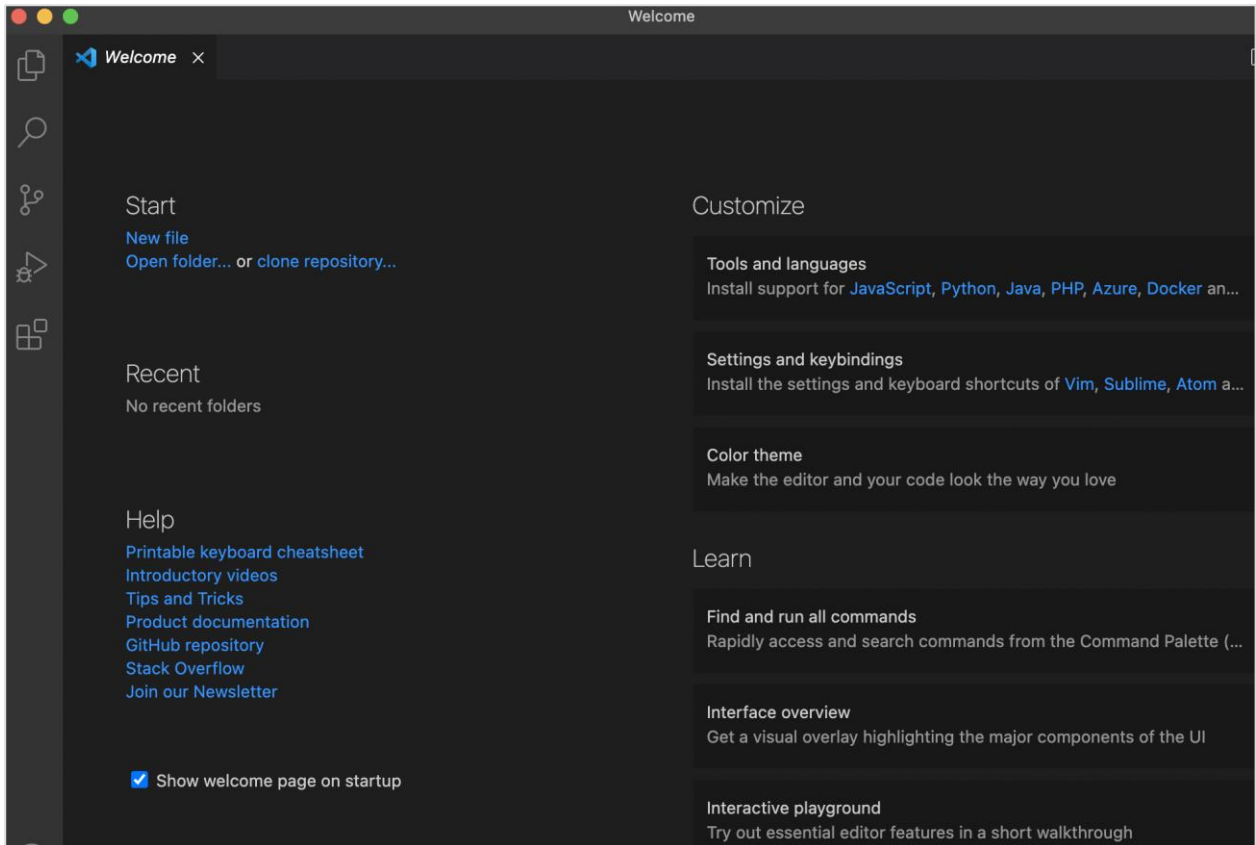


Step 2: A zip file gets downloaded after clicking the Download button. Click on this zip file and the Visual Studio Code application should be available on the machine.

The following screen will appear on your computer:



Step 3: Double-click on Visual Studio Code and it gets launched along with the welcome page. The following screen will appear on your computer:



3. WebdriverIO — Architecture

WebdriverIO architecture consists of the following components:

- NodeJS
- WebdriverIO
- JavaScript
- JSON Wire Protocol
- Services
- Browsers
- Application

Nodejs is enabled to execute the JavaScript runtime environment. It is actually an open-source project. WebdriverIO is developed on Nodejs and JavaScript is the script implemented by the end-user using the WebdriverIO library.

Thus the JavaScript implemented by the end-user passes a request using the WebdriverIO via Nodejs to the Services (in the format of an HTTP command). The entire process is done following the JSON Wire Protocol.

Services send the request to the browsers like Chrome, Firefox, and so on to execute a test against the application under test. Thus the Services can be termed as a middle-layer between the browser and the automation framework.

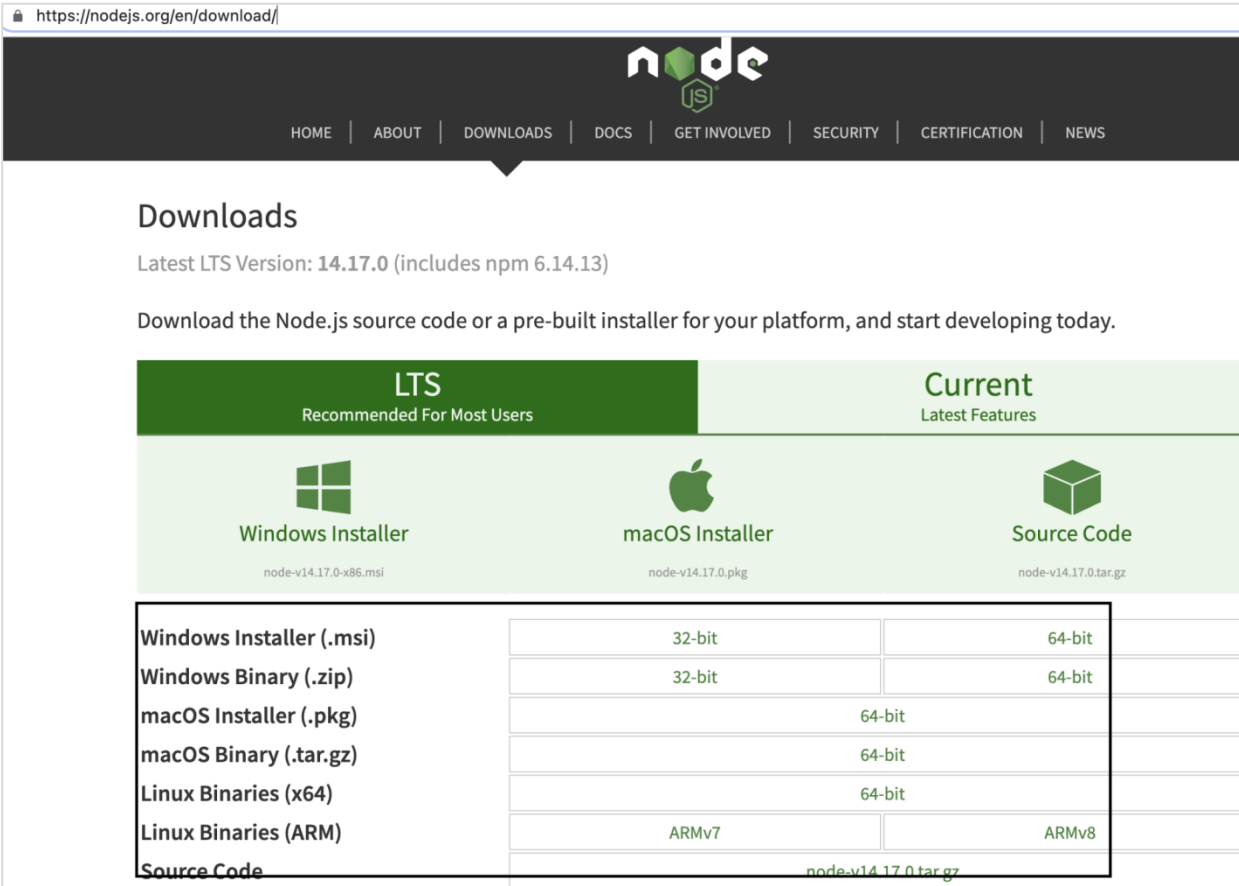
4. WebdriverIO — Getting Started with NodeJS

WebdriverIO coding is done using JavaScript. For this, NodeJS has to be installed since it is a JavaScript engine. Only after its installation, we can execute WebdriverIO tests. The steps to configure NodeJS are listed below:

Step 1: Launch the application using the below link:

<https://nodejs.org/en/download/>

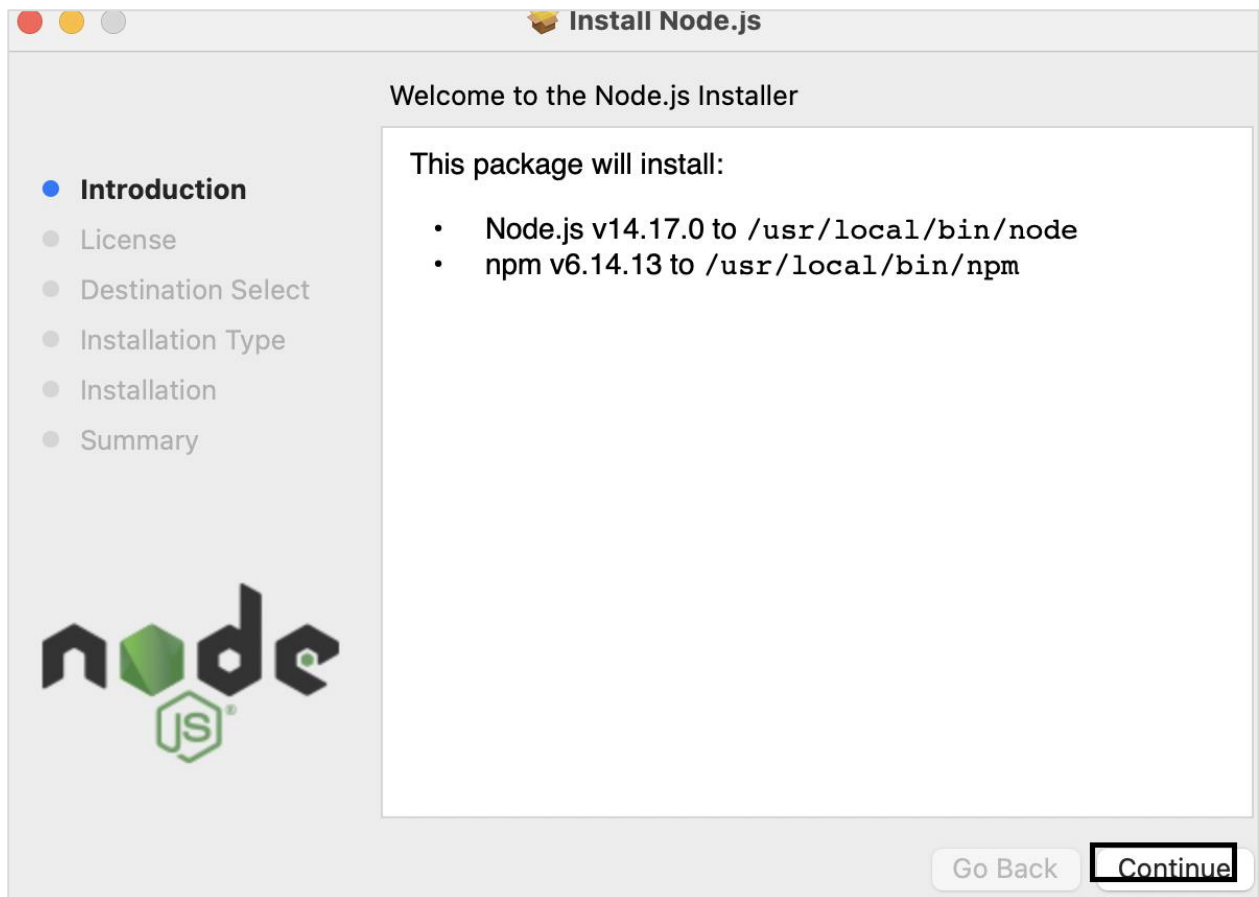
Step 2: As per the local operating system (Windows, Mac or Linux) we are using, click on the link to download the Installer. The following screen will appear on your computer:



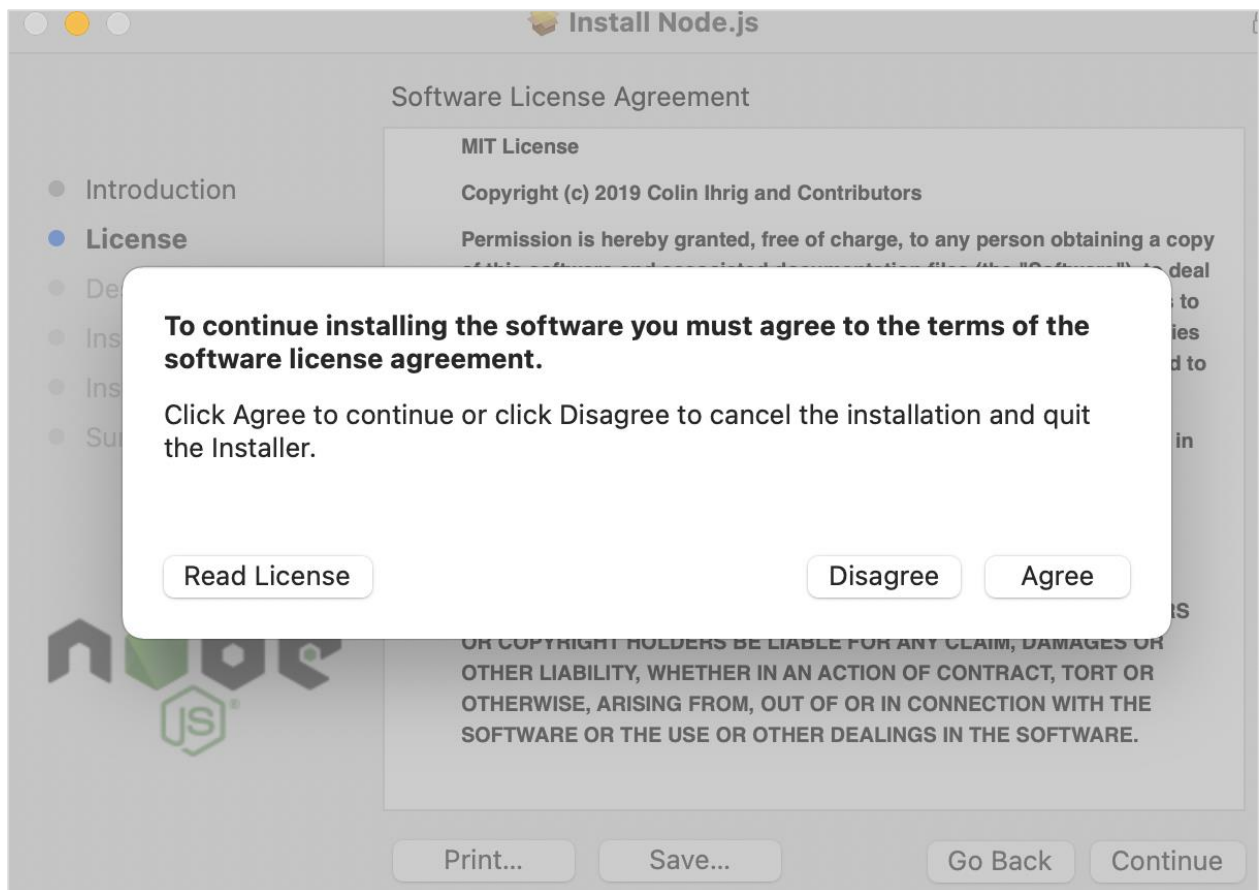
The screenshot shows the Node.js download page. The page title is "Downloads" and it displays the latest LTS version as 14.17.0 (including npm 6.14.13). It offers two main download paths: "LTS Recommended For Most Users" and "Current Latest Features". Under the LTS section, there are three options: "Windows Installer" (node-v14.17.0-x86.msi), "macOS Installer" (node-v14.17.0.pkg), and "Source Code" (node-v14.17.0.tar.gz). A table below these options lists various download formats and their bit architectures.

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)		64-bit
macOS Binary (.tar.gz)		64-bit
Linux Binaries (x64)		64-bit
Linux Binaries (ARM)	ARMv7	ARMv8
Source Code		node-v14.17.0.tar.gz

Step 3: Once the installer is downloaded, click on it. Navigate to the Node.js Installer welcome screen. Click on Continue. The following screen will appear on your computer:

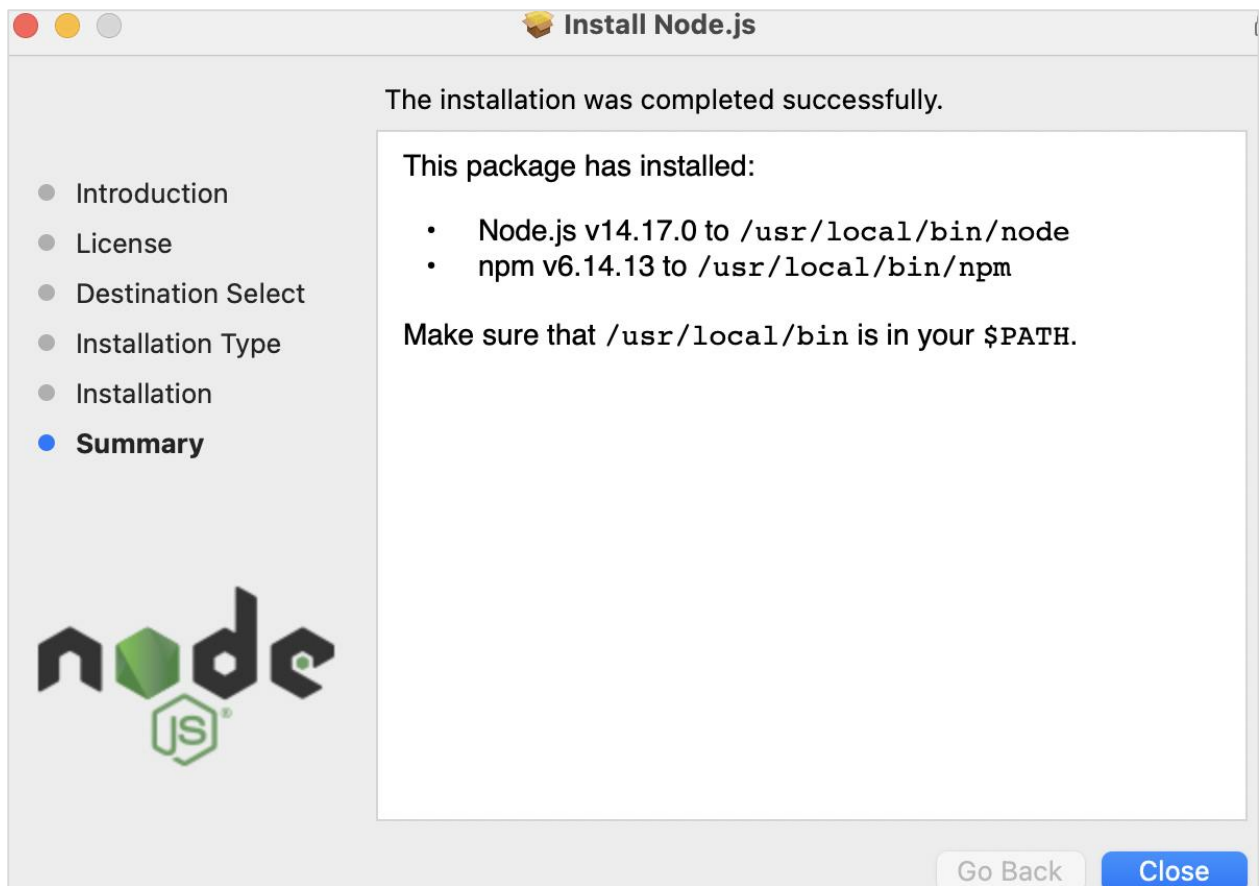


Step 4: Agree to the terms of agreement of Nodejs. The following screen will appear on your computer:



Step 5: Click on Install.

Step 6: Once the success message of Nodejs installation is displayed, click on Close. The following screen will appear on your computer:



Step 7: To check if Nodejs is installed successfully, open the terminal and run the command:

```
node
```

The following screen will appear on your computer:

```
[...]
(base) debomitabhattacharjee@Debomitas-MacBook-Air ~ % node
Welcome to Node.js v14.17.0.
Type ".help" for more information.
> █
```

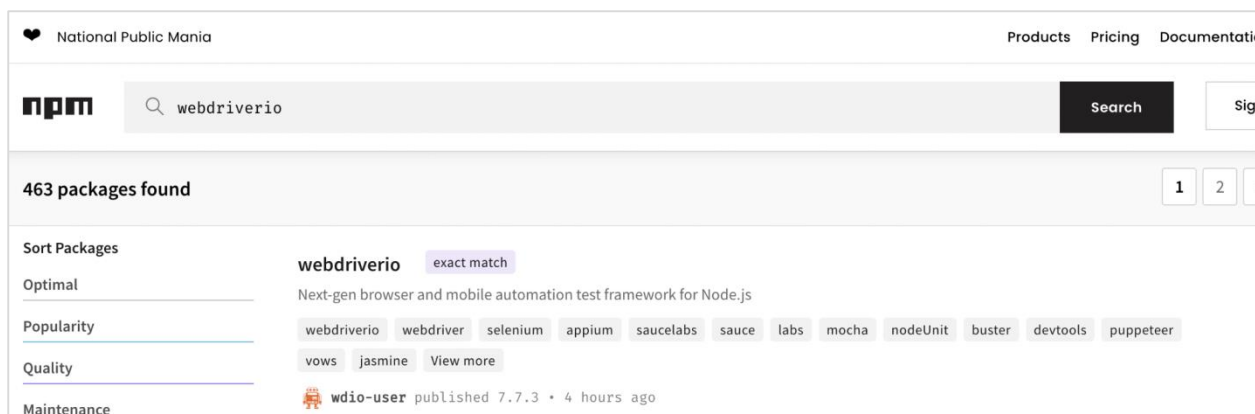
The version of the Nodejs installed in the machine should be displayed.

5. WebdriverIO — Installation of NPM

Once Nodejs has been installed, we have to create a NPM folder. NPM is actually the package manager for writing tests in JavaScript. The official page for NPM is available in the below link:

<https://www.npmjs.com/search?q=webdriverio>

Once we launch this page, enter WebdriverIO in the search box and click on Search, to get the npm packages for WebdriverIO. The following screen will appear on your computer:



Create NPM Project

The steps to create a NPM project are listed below:

Step 1: Create an empty folder, say webdriverIO in a location.

Step 2: Open the terminal and move from the current directory to the directory of the empty folder that we have created.

Step 3: Run the following command:

```
npm init -y
```

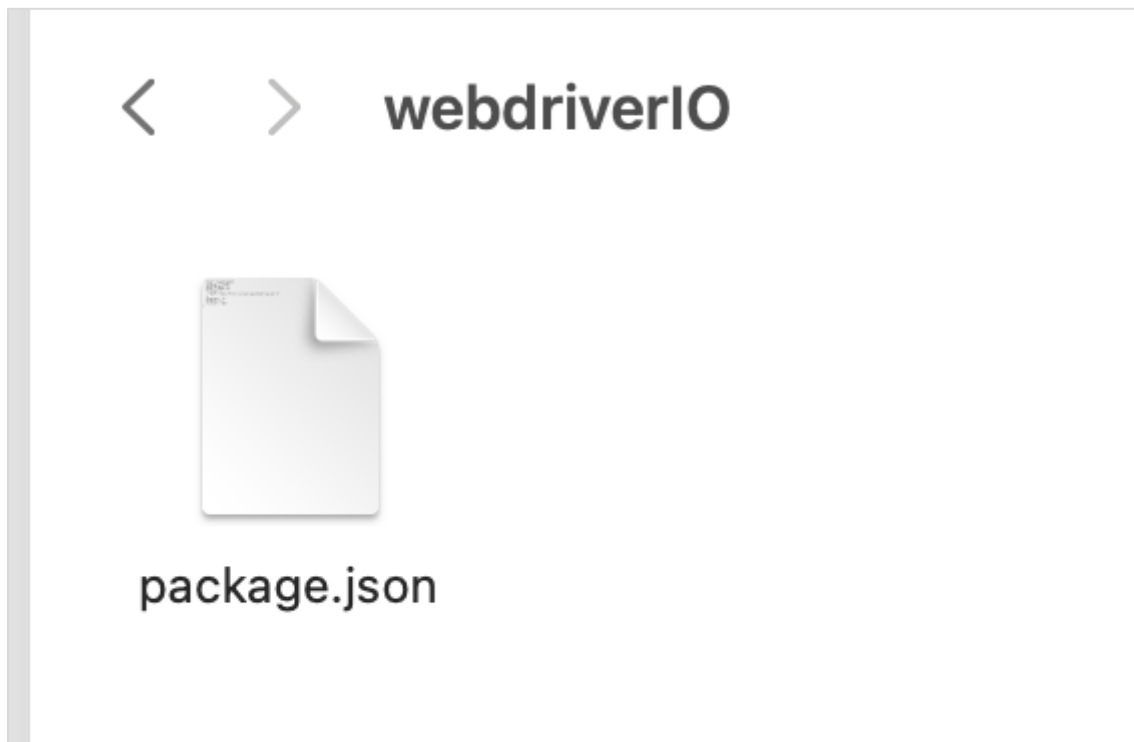
The y parameter is given to set the default values. The following screen will appear on your computer:


```
[(base) debomita@bhattacharjee@Debomitas-MacBook-Air ~ % cd webdriverIO
[(base) debomita@bhattacharjee@Debomitas-MacBook-Air webdriverIO % npm init -y
Wrote to /Users/debomita@bhattacharjee/webdriverIO/package.json:

{
  "name": "webdriverIO",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Step 4: The output obtained on running the command in Step 3 says that all the default configurations have been captured within the package.json file. It is generated within the folder we have created (named webdriverIO) in Step 1.

The following screen will appear on your computer:



This package.json contains all the dependencies which we need to work with the WebdriverIO project. To get any package under NPM, we can refer to the link:

<https://www.npmjs.com/>.

6. WebdriverIO — VS Code Installation

In this chapter, let us understand how to install the Visual Studio (VS) Code in WebdriverIO.

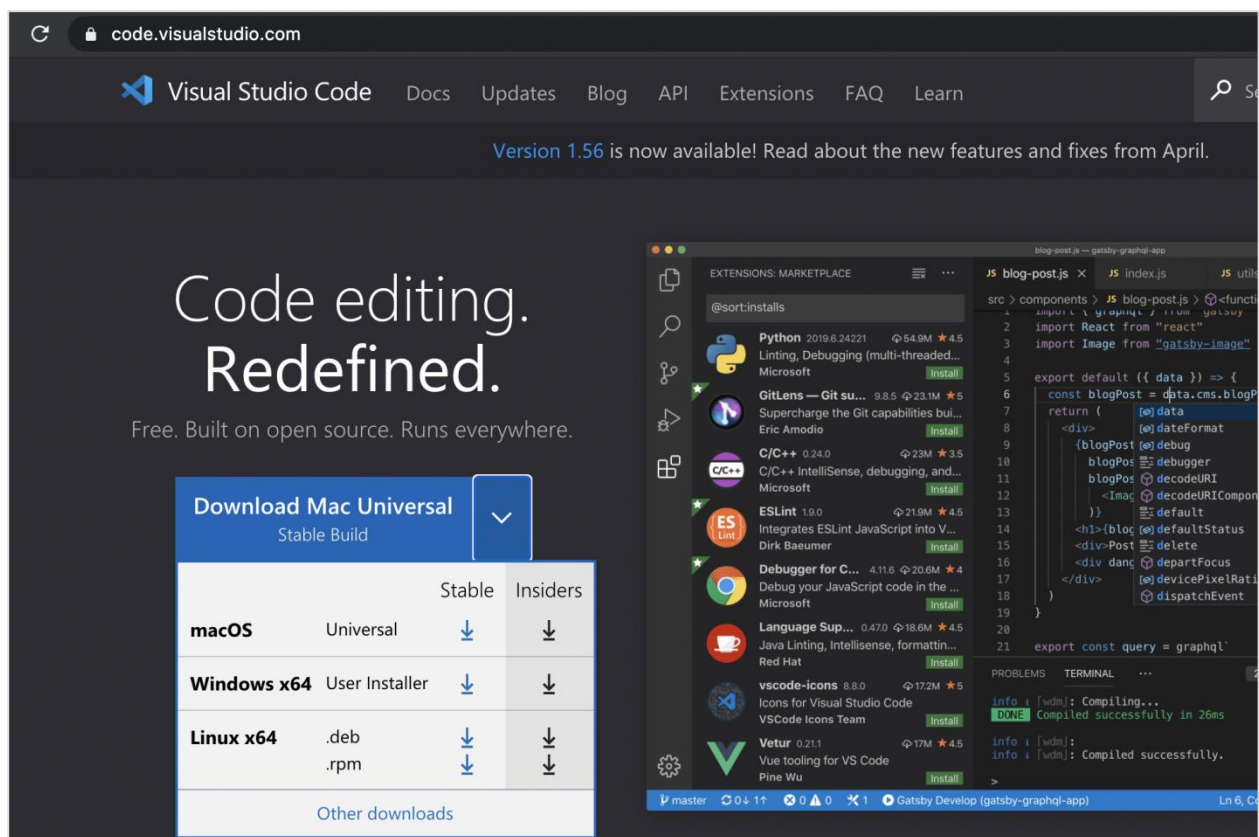
The steps to install the Visual Studio Code are listed below:

Step 1: Navigate to the below link:

<https://code.visualstudio.com/>

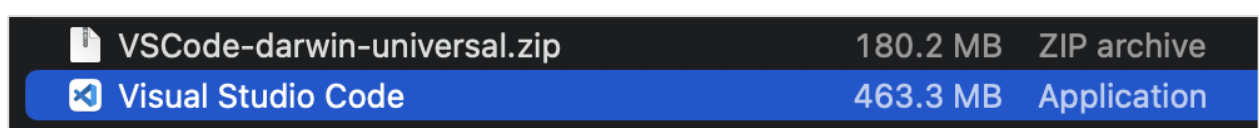
Step 2: Depending on the local operating system we have for example - macOS, Linux or Windows, we need to choose the link for download.

The following screen will appear on your computer:



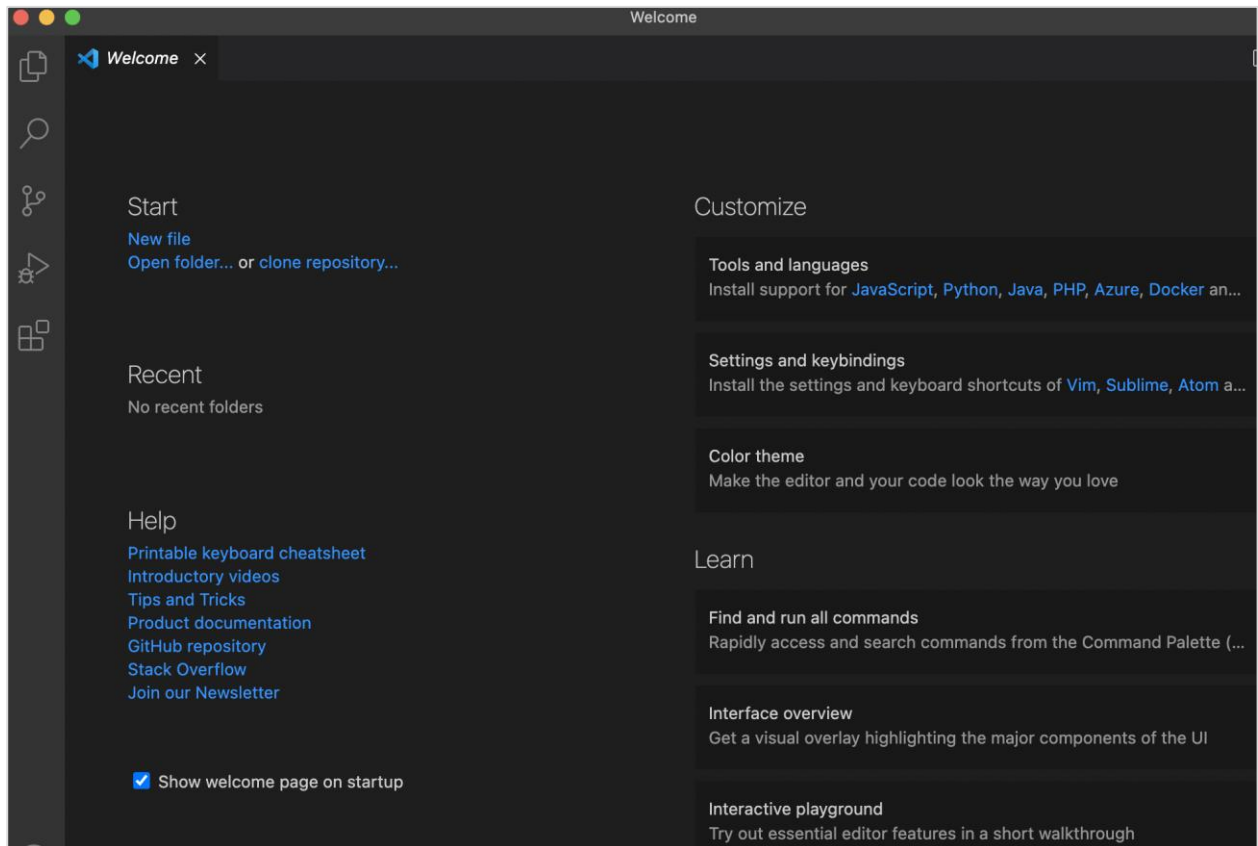
Step 3: A zip file gets downloaded after clicking the Download button. After downloading this file has completed, click on it and the Visual Studio Code application should become available for use.

The following screen will appear on your computer:



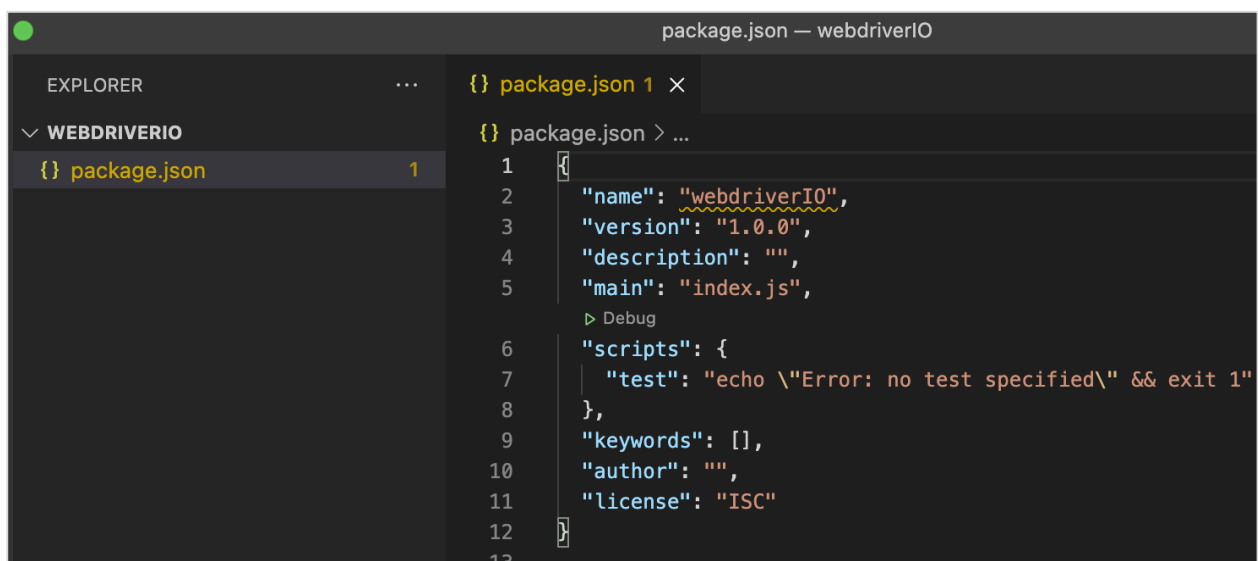
Step 4: Double-click it and the Visual Studio Code application should launch along with the welcome page.

The following screen will appear on your computer:



Step 5: Click on the Open folder link and import the folder that contains the package.json file. The details of how the package.json file got created are discussed in detail in the Chapter titled Installation of NPM.

The following screen will appear on your computer:



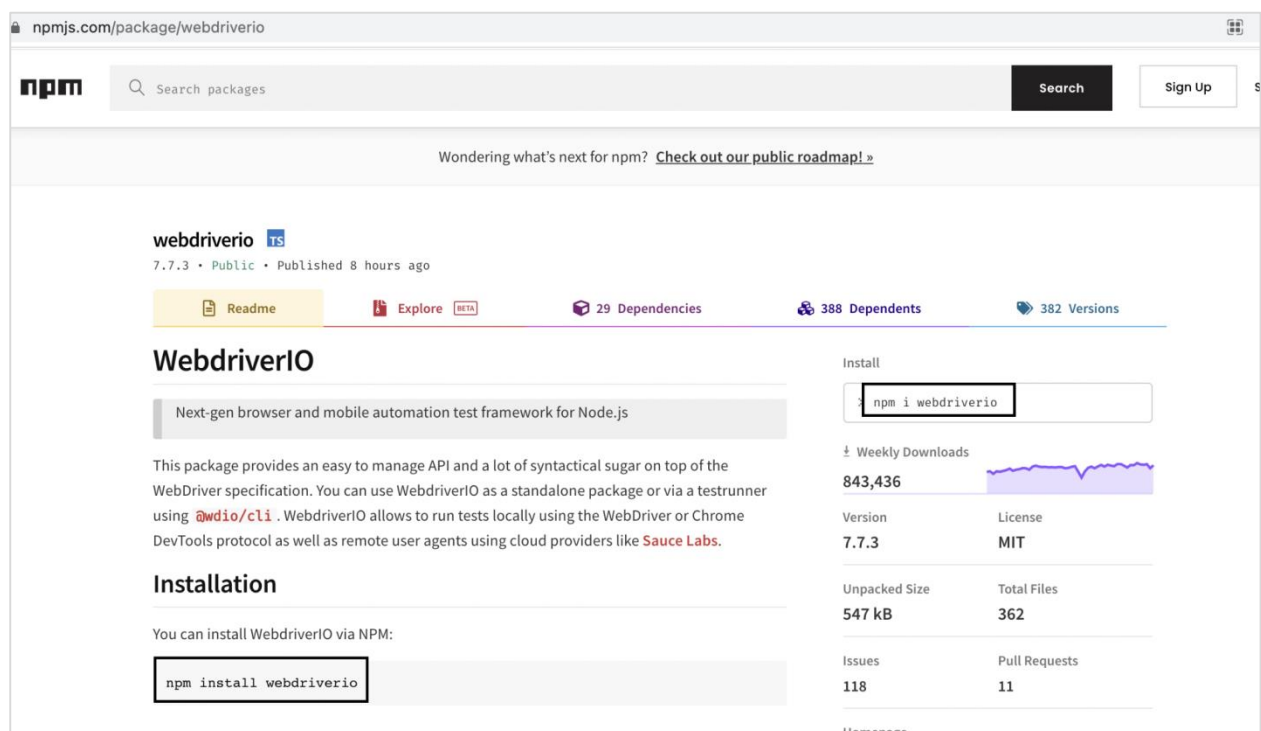
7. WebdriverIO — Package.json

Once the package.json file gets generated, we need to install other npm packages for WebdriverIO. The details of how the package.json file got created are discussed in detail in the Chapter titled Installation of NPM.

The necessary packages for WebdriverIO in the NPM registry can be found in the below link:

<https://www.npmjs.com/package/webdriverio>

The following screen will appear on your computer:



The screenshot shows the npmjs.com page for the `webdriverio` package. The page includes a search bar, a navigation menu, and a main content area. The main content area features the package name `webdriverio` with a TypeScript icon, version `7.7.3`, and a description: "Next-gen browser and mobile automation test framework for Node.js". It also includes an "Installation" section with the command `npm install webdriverio` highlighted in a box. A table on the right side of the page provides additional details about the package, including weekly downloads, version, license, unpacked size, total files, issues, and pull requests.

Weekly Downloads	Version	License
843,436	7.7.3	MIT

Unpacked Size	Total Files	Issues	Pull Requests
547 kB	362	118	11

For installation of WebdriverIO, we have to run the below command from the terminal:

```
npm i webdriverio
```

or

```
npm install webdriverio.
```

The following screen will appear on your computer:

The screenshot shows a code editor with the Explorer view on the left showing a project named 'WEBDRIVERIO' with files 'node_modules', 'package-lock.json', and 'package.json'. The 'package.json' file is open in the editor, showing the following content:

```

1 {
2   "name": "webdriverIO",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "webdriverio": "^7.7.3"
14  }
15 }

```

The terminal output below shows the execution of the command `npm install webdriverio` and its results:

```

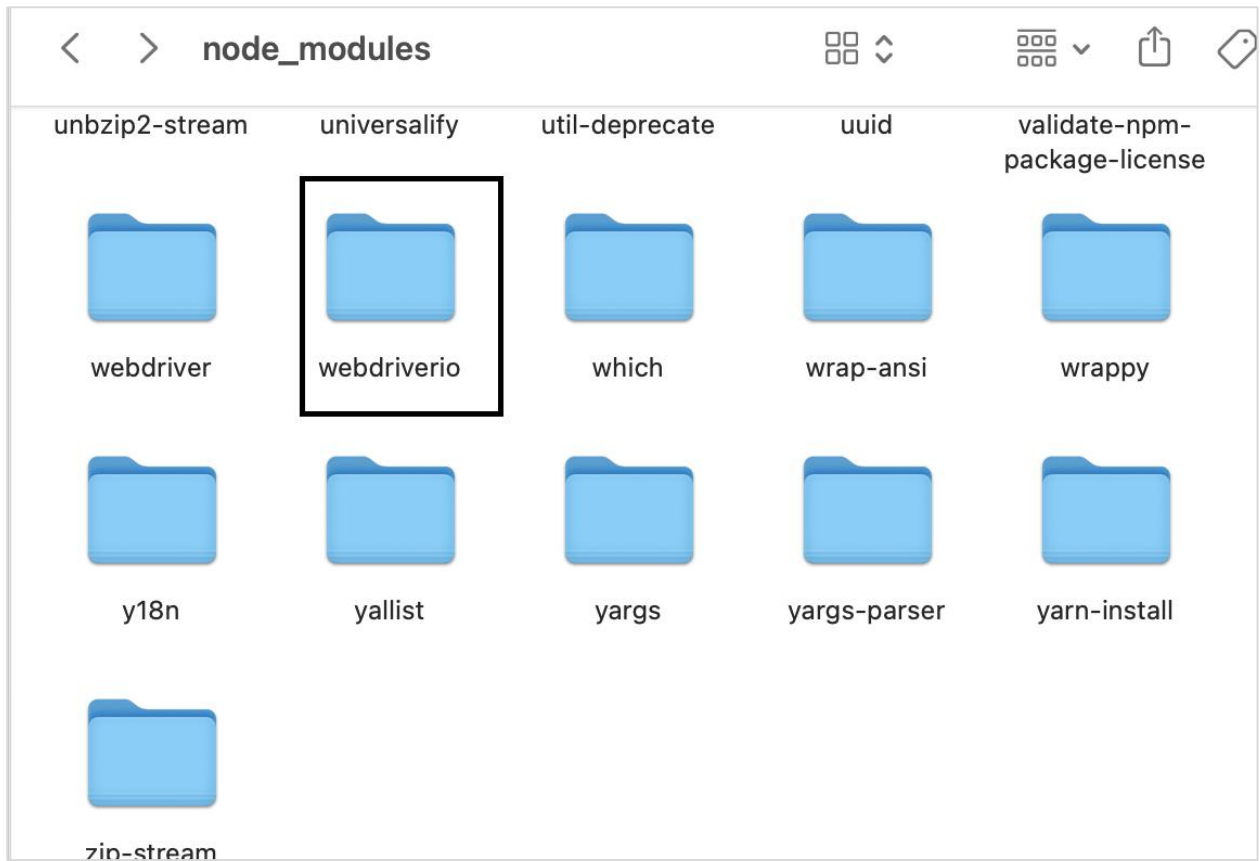
(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npm install webdriverio
> core-js-pure@3.13.1 postinstall /Users/debomtabhattacharjee/webdriverIO/node_modules/core-js-pure
> node -e "try{require('./postinstall')}catch(e){}"
Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!
The project needs your help! Please consider supporting of core-js on Open Collective or Patreon:
> https://opencollective.com/core-js
> https://www.patreon.com/zloirock
Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN webdriverIO@1.0.0 No description
npm WARN webdriverIO@1.0.0 No repository field.
+ webdriverio@7.7.3
added 166 packages from 242 contributors and audited 166 packages in 10.995s
22 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO %

```

After the command gets executed successfully, the package.json now displays the WebdriverIO version installed.

We can verify if the WebdriverIO has installed successfully, if the folder `node_modules` created within the project contains the `webdriverio` folder.

The following screen will appear on your computer:



8. WebdriverIO — Mocha Installation

Mocha is a testing framework based on JavaScript which is built on Nodejs. It makes asynchronous test execution flow interesting and simple. Mocha tests can be run serially.

It is capable of producing accurate and customizable reports. Also, the uncaught exceptions can be easily tagged with the proper test cases. The details of Mocha can be found in the below link:

https://www.tutorialspoint.com/tesults/tesults_integrating_your_automated_tests.htm

To install Mocha packages in the NPM registry, the command is as follows:

```
npm install mocha
```

The following screen will appear on your computer:

```

{} package.json > ...
1  {
2    "name": "webdriverIO",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "mocha": "^8.4.0",
14     "webdriverio": "^7.7.3"
15   },
16   "devDependencies": {
17     "@wdio/cli": "^7.7.3"
18   }
19 }
20

```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE 1: zsh

```

(base) debomita@Debomitas-MacBook-Air: ~/webdriverIO % npm install mocha
npm WARN webdriverIO@1.0.0 No description
npm WARN webdriverIO@1.0.0 No repository field.

+ mocha@8.4.0
added 37 packages from 25 contributors and audited 339 packages in 3.417s

45 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

```

After the command has been executed successfully, the Mocha version installed gets reflected within the package.json file.

9. WebdriverIO — Selenium Standalone Server Installation

WebdriverIO works under the roof of Selenium. To establish communication with the browser, we are required to install the Selenium standalone server.

To install Selenium standalone server, we have to run the following command:

```
npm install selenium-standalone
```

Or,

```
npm i selenium-standalone.
```

The following screen will appear on your computer:

```
{ } package.json > ...
1  {
2    "name": "webdriverIO",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    > Debug
7    "scripts": {
8      "test": "echo \"Error: no test specified\" && exit 1"
9    },
10   "keywords": [],
11   "author": "",
12   "license": "ISC",
13   "dependencies": {
14     "mocha": "^8.4.0",
15     "selenium-standalone": "^6.23.0",
16     "webdriverIO": "^7.7.3"
17   },
18   "devDependencies": {
19     "@wdio/cli": "^7.7.3"
20   }
21 }
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE 1: zsh

found 0 vulnerabilities

```
(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npm install selenium-standalone
npm WARN webdriverIO@1.0.0 No description
npm WARN webdriverIO@1.0.0 No repository field.

+ selenium-standalone@6.23.0
added 10 packages from 13 contributors and audited 349 packages in 2.393s

45 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

After the command has been executed successfully, the Selenium standalone server package version installed gets reflected within the package.json file.

10. WebdriverIO — Configuration File generation

WebdriverIO tests are controlled from a Configuration file. It is often considered the heart of WebdriverIO. It contains details on what test cases to be executed, browser on which the tests should run, global information - timeout, reports, screenshots and so on.

In WebdriverIO we do not execute a single test. We are required to trigger the Configuration file with the help of the Test Runner. Test Runner scans the information provided in the Configuration file and then triggers the tests accordingly.

To get the Test Runner, we have to install the WebdriverIO CLI dependencies. To install this and save it in the package.json file, we have to run the below mentioned command:

```
npm i --save-dev @wdio/cli
```

After this command has been executed successfully, the version of CLI dependency shall be reflected within the package.json file. The following screen will appear on your computer:

```

{} package.json > ...
1  {
2    "name": "webdriverIO",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "mocha": "^8.4.0",
14     "selenium-standalone": "^6.23.0",
15     "webdriverio": "^7.7.3"
16   },
17   "devDependencies": {
18     "@wdio/cli": "^7.7.3"
19   }
20 }
21

```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE 1: zsh

```

(base) debomitabhattacharjee@Debomitas-MacBook-Air webdriverIO % npm i --save-dev @wdio/cli
npm WARN webdriverIO@1.0.0 No description
npm WARN webdriverIO@1.0.0 No repository field.

+ @wdio/cli@7.7.3
updated 1 package and audited 349 packages in 4.649s

35 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

(base) debomitabhattacharjee@Debomitas-MacBook-Air webdriverIO %

```

To create a Configuration file, we have to run the below mentioned command:

```
npx wdio config -y
```

After this command has been executed successfully, the configuration file called the wdio.conf.js gets created within our project. Also, the package.json file should now contain some more dependencies under the devDependencies field.

The following screen will appear on your computer:

```

14     "selenium-standalone": "^6.23.0",
15     "webdriverio": "^7.7.3"
16   },
17   "devDependencies": {
18     "@wdio/cli": "^7.7.3",
19     "@wdio/local-runner": "^7.7.3",
20     "@wdio/mocha-framework": "^7.7.3",
21     "@wdio/spec-reporter": "^7.7.3",
22     "chromedriver": "^91.0.0",
23     "wdio-chromedriver-service": "^7.1.0"
24   }
25 }

```

Apart from the dependencies marked in the above image, we have to add one more dependency so that the WebdriverIO commands can execute synchronously.

We have to add the dependency - "@wdio/sync": "<version number>" under the devDependencies field. Then run the following command:

```
npm install
```

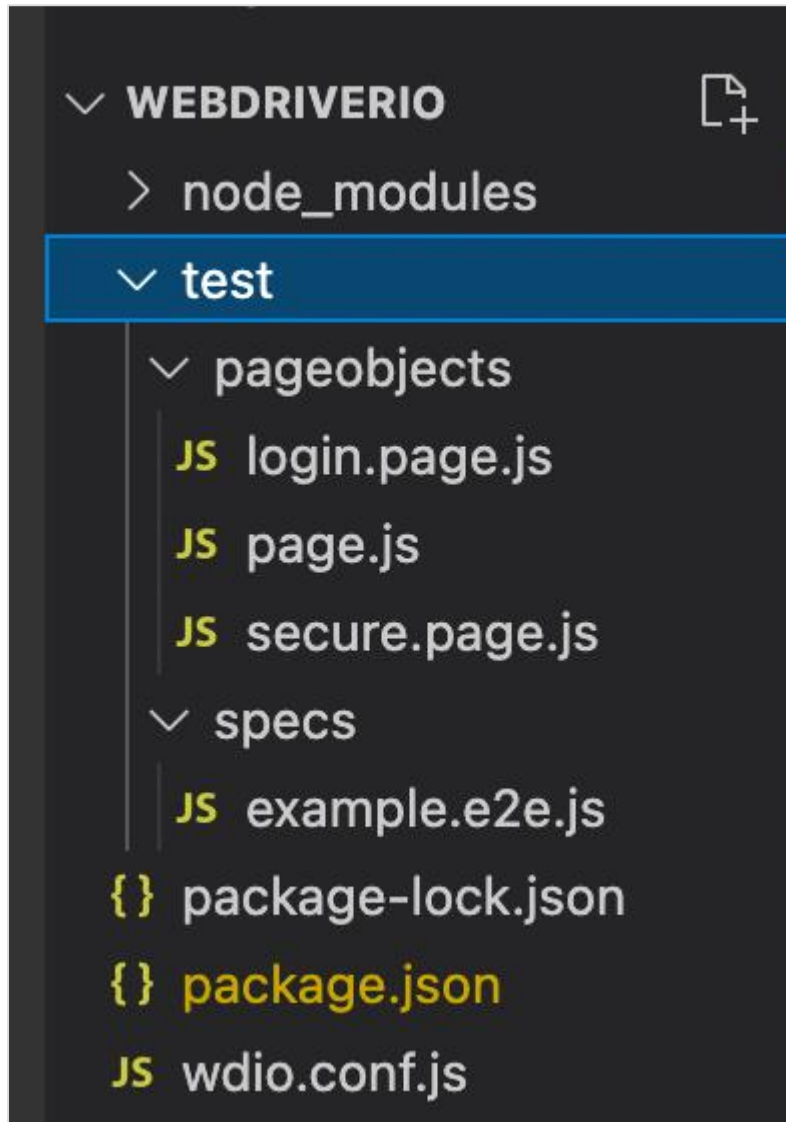
To run a Configuration file from the test runner, we have to run the below given command:

```
npx wdio run wdio.conf.js
```

Create Mocha Spec File

After a Configuration file is created, we shall find a test folder generated within the WebdriverIO project. The details on how to create a Configuration file are described in the Chapter titled Configuration File generation.

The following screen will appear on your computer:



If we expand this folder, we shall find two sub-folders - pageobjects and specs containing JavaScript files created by default. These are basically sample tests provided to guide the first time users to get accustomed with the Mocha framework.

Mocha is a testing framework based on JavaScript which is built on Nodejs. It makes asynchronous test execution flow interesting and simple. Mocha tests can be run serially.

It is capable of producing accurate and customizable reports. Also, the uncaught exceptions can be easily tagged with the proper test cases. The details of Mocha can be found in the below link:

https://www.tutorialspoint.com/tesults/tesults_integrating_your_automated_tests.htm

As per the Mocha testing framework, all the test files are known as the spec files and they should reside within the specs folder.

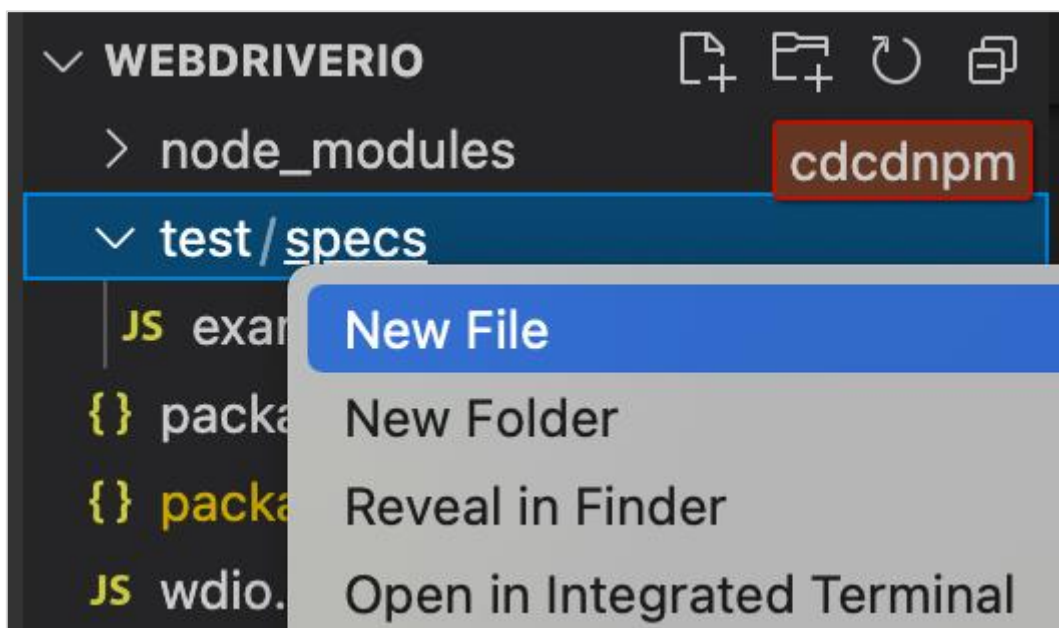
Blocks in Test File

A test file should have the following blocks:

- **describe:** This is higher in hierarchy than the it block. A test file can have multiple describe blocks. A describe block represents a test suite. It has two arguments - description of the test suite and an anonymous function.
- **it:** This is lower in hierarchy than the describe block. A describe can have multiple it blocks. An it block represents a test case and should be mandatory within a describe block. It has two arguments - description of the test case and an anonymous function. The actual WebdriverIO code is implemented within the it block.

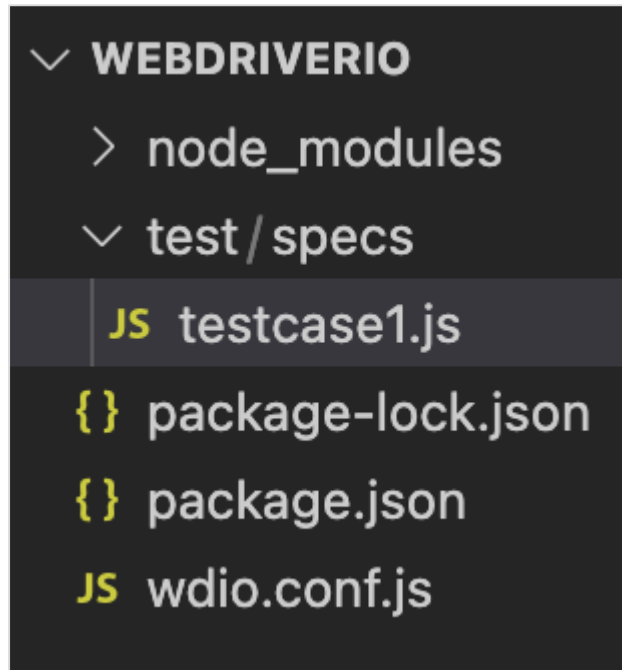
To create a Mocha file, let us follow the below steps:

Step 1: Right-click on the specs folder (which is within the test folder), then select New File. The following screen will appear on your computer:



Step 2: Enter a filename, say testcase1.js.

The following screen will appear on your computer:



Step 3: Add the below code in this file:

```
// test suite name
describe('Tutorialspoint Application', function () {
// test case name
it('Get Page Title', function (){
// URL launching
    browser.url("https://www.tutorialspoint.com/about/about_careers.htm")
//print page title in console
    console.log(browser.getTitle())
});
});
```

In the above code, the browser is the global object exposed by the WebdriverIO.

Please note: We cannot run this individual file directly. We shall take the help of the Configuration file in order to execute it.

11. WebdriverIO — VS Code Intellisense

Once we have completed installation of the Visual Studio Code, we should add the intellisense in the editor so that once we begin writing the WebdriverIO commands, the auto-suggestions of the WebdriverIO methods are displayed.

The details on how to do a VS Code installation are discussed in detail in the Chapter titled VS Code Installation.

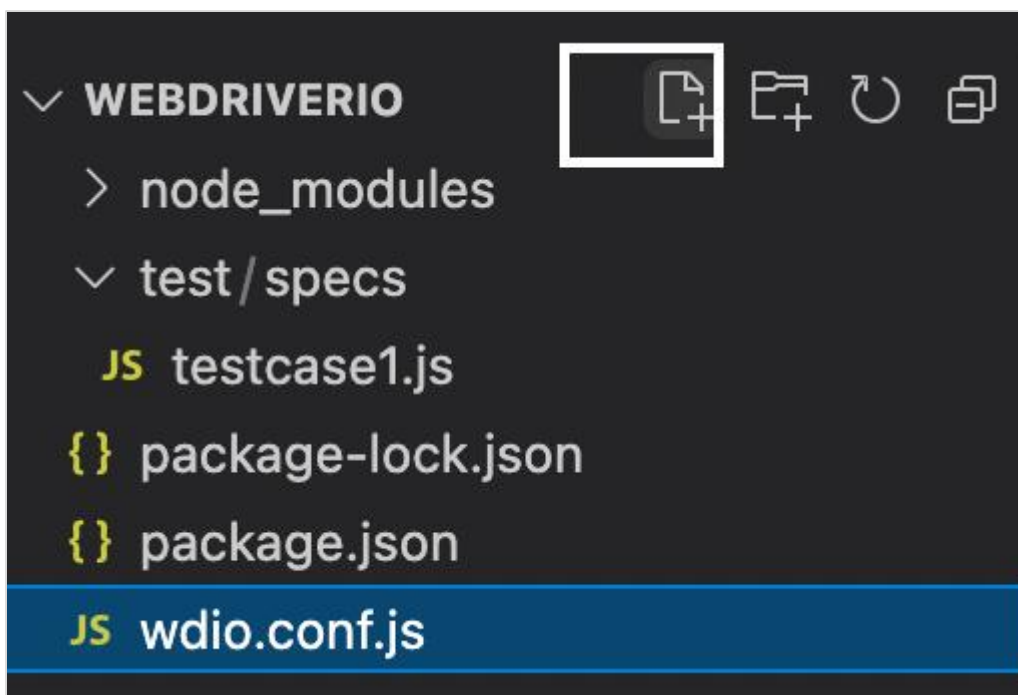
This is a very important feature that should be added so that the end-users do not need to memorize the raw code for the WebdriverIO.

Add intellisense to VS Code

The steps to add intellisense to the VS Code for the WebdriverIO are listed below:

Step 1: Click on the New File button appearing to the right of the WebdriverIO project.

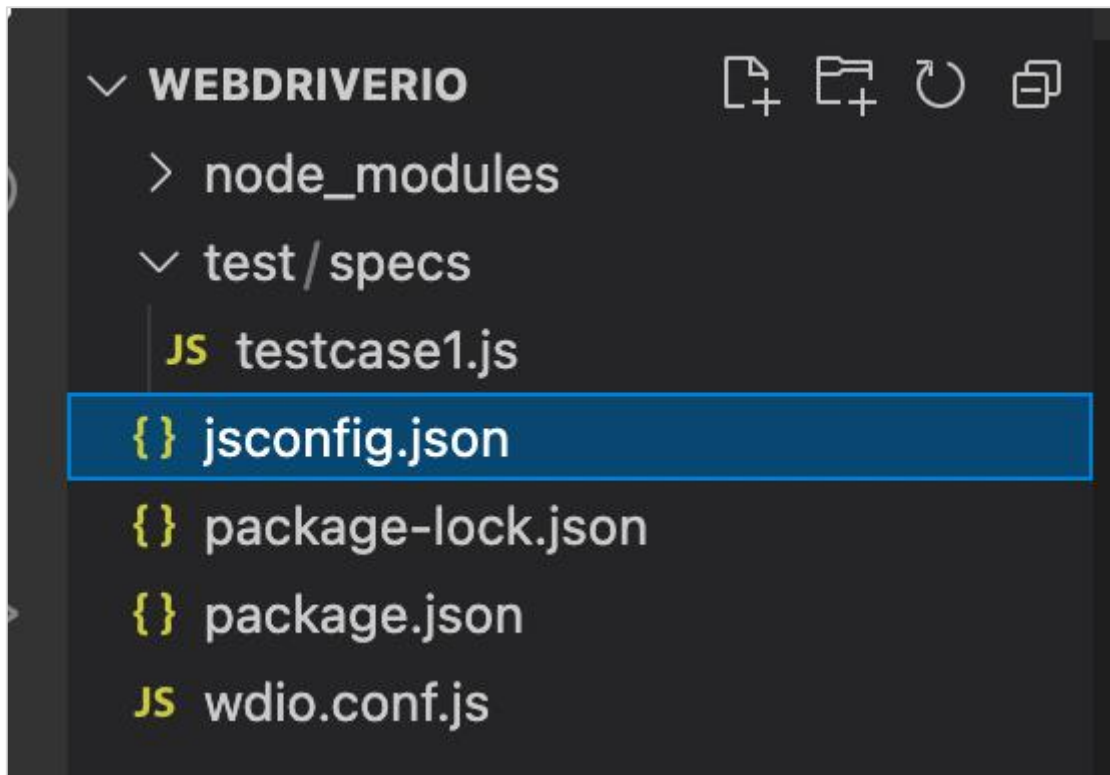
The following screen will appear on your computer:



Step 2: Enter the file name as jsconfig.json. Here, we have to specify the path of the spec files where we are implementing our test.

If we want to apply intellisense feature to all the spec files within the test folder, we can specify the relative path as **test/spec/*.js**.

The following screen will appear on your computer:



Step 3: Add the below code inside the file.

```
{
  "include": [
    //relative path of all spec files
    "test/specs/*.js",
    "**/*.json",
    "node_modules/@wdio/sync",
    "node_modules/@wdio/mocha-framework"
  ]
}
```

Step 4: In the spec file, start writing a WebdriverIO object or a method and we shall obtain the entire auto - suggestions.

The following screen will appear on your computer:

JS testcase1.js ●

```
test > specs > JS testcase1.js > describe('Tutorialspoint application')
1
2 // test suite name
3 describe('Tutorialspoint application', function(){
4 //test case
5 it('Get Page Title', function(){
6
7 // launch url
8 browser.
9 //get abc browser
10 conso abc console
11 abc describe
12 abc getTitle
13 abc it
14 abc log
15 });
16 });
```

12. WebdriverIO — Wdio.conf.js file

WebdriverIO tests are controlled from the Configuration file. It is often considered the heart of WebdriverIO. It contains details on which spec files to be executed, browser on which the tests should run, global information - base URL, timeout, reports, screenshots and so on.

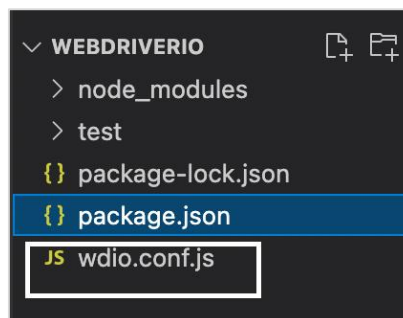
In WebdriverIO we do not execute a single test. We are required to trigger the Configuration file with the help of the Test Runner. Test Runner scans the information provided in the Configuration file and then triggers the tests accordingly.

To create a Configuration file, we have to run the below command:

```
npx wdio config -y
```

After this command has been executed successfully, the Configuration file called the `wdio.conf.js` gets created within our project.

The following screen will appear on your computer:



Within this file, we have to specify the path of the spec file that we want to execute within the specs parameter.

By default, the path provided is: `./test/specs/**/*.js`. This means any `.js` file under the sub-folder specs (which is under the folder test) should be picked for execution.

The following screen will appear on your computer:

```

✓ WEBDRIVERIO JS wdio.conf.js > config
  > node_modules 20 // process simply enclose the
  > test/specs 21 //
  JS testcase1.js 22 // If you are calling `wdio`
  {} jscfg.json 23 // then the current working d
  {} package-lock.json 24 // will be called from there.
  {} package.json 25 //
  JS wdio.conf.js 26 specs: [
  27 |   './test/specs/**/*.js'
  28 | ],

```

To execute the test with the help of the wdio.conf.js file, we have to run the command:

```
npx wdio run wdio.conf.js
```

The following screen will appear on your computer:

```

(base) debomtabhattarjee@Debitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-06T03:34:04.671Z
2021-06-06T03:34:04.716Z INFO @wdio/cli:launcher: Run onPrepare hook
2021-06-06T03:34:04.717Z INFO chromedriver: Start Chromedriver (/Users/debomtabhattarjee/webdriverIO/node_modules/chromedriver/lib/chromedriver/chromedriver) with args --port=9515 --url-base=
2021-06-06T03:34:04.774Z INFO chromedriver: Starting Chromedriver 91.0.4472.19 (1bf021f248676a0b2ab3ee0561d83a59e424c23e-refs/branch-heads/4472@{#288}) on port 9515
2021-06-06T03:34:04.774Z INFO chromedriver: Only local connections are allowed.
2021-06-06T03:34:04.774Z INFO chromedriver: Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping Chromedriver safe.
2021-06-06T03:34:04.808Z INFO chromedriver: Chromedriver was started successfully.
2021-06-06T03:34:04.848Z INFO @wdio/cli:launcher: Run onWorkerStart hook
2021-06-06T03:34:04.841Z INFO @wdio/local-runner: Start worker 0-0 with arg: run,wdio.conf.js
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] 2021-06-06T03:34:05.254Z INFO webdriver: Initiate new session using the WebDriver protocol
[0-0] 2021-06-06T03:34:05.296Z INFO webdriver: [POST] http://localhost:9515/session
[0-0] 2021-06-06T03:34:05.296Z INFO webdriver: DATA {
[0-0]   capabilities: {
[0-0]     alwaysMatch: { browserName: 'chrome', acceptInsecureCerts: true },
[0-0]     firstMatch: [ {} ]
[0-0]   },
[0-0]   desiredCapabilities: { browserName: 'chrome', acceptInsecureCerts: true }
[0-0] }
[0-0] 2021-06-06T03:34:08.190Z INFO webdriver: COMMAND navigateTo("https://www.tutorialspoint.com/about/about_careers.htm")
[0-0] 2021-06-06T03:34:08.191Z INFO webdriver: [POST] http://localhost:9515/session/bca1501bf3b926241529448932037566/url
[0-0] 2021-06-06T03:34:08.191Z INFO webdriver: DATA { url: 'https://www.tutorialspoint.com/about/about_careers.htm' }
[0-0] 2021-06-06T03:34:12.617Z INFO webdriver: COMMAND getTitle()
[0-0] 2021-06-06T03:34:12.617Z INFO webdriver: [GET] http://localhost:9515/session/bca1501bf3b926241529448932037566/title
[0-0] 2021-06-06T03:34:12.643Z INFO webdriver: RESULT About Careers at Tutorialspoint
[0-0] About Careers at Tutorialspoint - Tutorialspoint
[0-0] 2021-06-06T03:34:12.643Z INFO webdriver: COMMAND deleteSession()
[0-0] 2021-06-06T03:34:12.643Z INFO webdriver: [DELETE] http://localhost:9515/session/bca1501bf3b926241529448932037566
[0-0] PASSED in chrome - /test/specs/testcase1.js
2021-06-06T03:34:12.822Z INFO @wdio/cli:launcher: Run onComplete hook

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: bca1501bf3b926241529448932037566
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] > /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0] < Get Page Title
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (4.5s)

Spec Files:   1 passed, 1 total (100% completed) in 00:00:08
2021-06-06T03:34:12.823Z INFO @wdio/local-runner: Shutting down spawned worker
2021-06-06T03:34:13.073Z INFO @wdio/local-runner: Waiting for 0 to shut down gracefully
2021-06-06T03:34:13.075Z INFO @wdio/local-runner: Shutting down

```

After the command has been executed successfully, the page title of the application launched is obtained in the console.

However, a lot of the logs got captured in the console. This is because the parameter logLevel is set to info by default in the wdio.conf.js file.

The following screen will appear on your computer:

```

✓ WEBDRIVERIO
  > node_modules clear 73 // Define all optio
  ✓ test/specs 74 //
    JS testcase1.js 75 // Level of logging
    {} jsconfig.json 76 logLevel: 'info',
    {} package-lock.json 77 //
    {} package.json 78 // Set specific log
    JS wdio.conf.js 79 // loggers:
    JS wdio.conf.js 80 // - webdriver, web

```

In order to get rid of some of the logs and to obtain only those which the test case directs, we can set this parameter to silent.

The following screen will appear on your computer:

```

WEBDRIVERIO
  > node_modules clear 73 // Define all option
  ✓ test/specs 74 //
    JS testcase1.js 75 // Level of logging
    {} jsconfig.json 76 logLevel: 'silent',
    {} package-lock.json 77 //
    {} package.json 78 // Set specific log
    JS wdio.conf.js 79 // loggers:
    JS wdio.conf.js 80 // - webdriver, web
    JS wdio.conf.js 81 // - @wdio/appliteo

```

Again run the Configuration file with the following command:

```
npx wdio run wdio.conf.js
```

The following screen will appear on your computer:

```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  1: zsh

2021-06-06T03:19:16.936Z INFO @wdio/local-runner: Start worker 0-0 with arg: run,wdio.conf.js
[0-0] 2021-06-06T03:19:17.273Z INFO @wdio/local-runner: Run worker command: run
[0-0] RUNNING in chrome - /test/specs/testcase1.js
(base) debomitabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js

Execution of 1 workers started at 2021-06-06T03:46:45.951Z

[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] About Careers at Tutorials Point - Tutorialspoint
[0-0] PASSED in chrome - /test/specs/testcase1.js

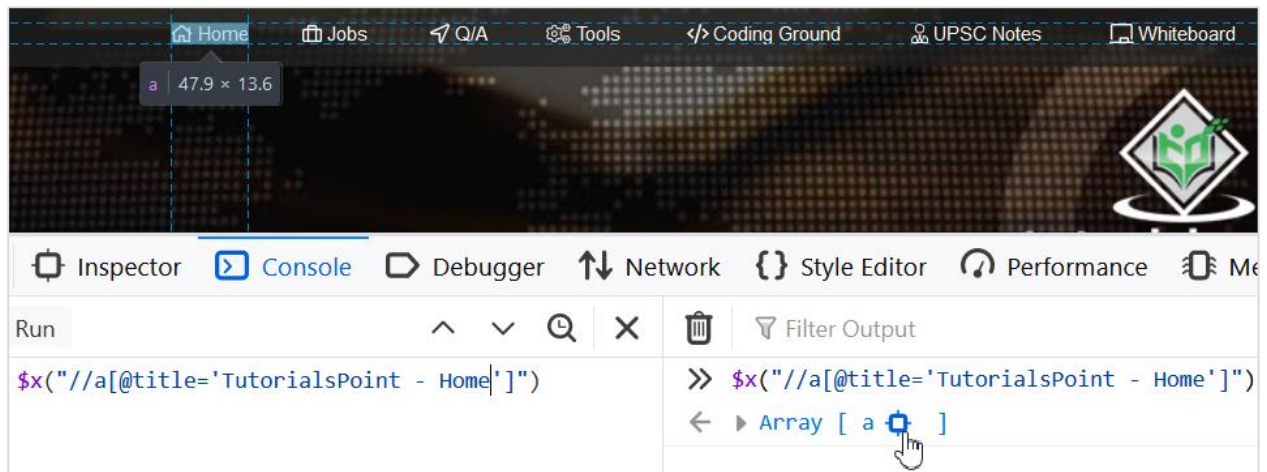
"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: c87c645bc7851f5292b7331122e17927
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Get Page Title
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (3.8s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:07

```

After the command has been executed successfully, we find lesser logs and the page title of the application launched - About Careers at Tutorials Point - Tutorialspoint is obtained in the console.

The following screen will appear on your computer:



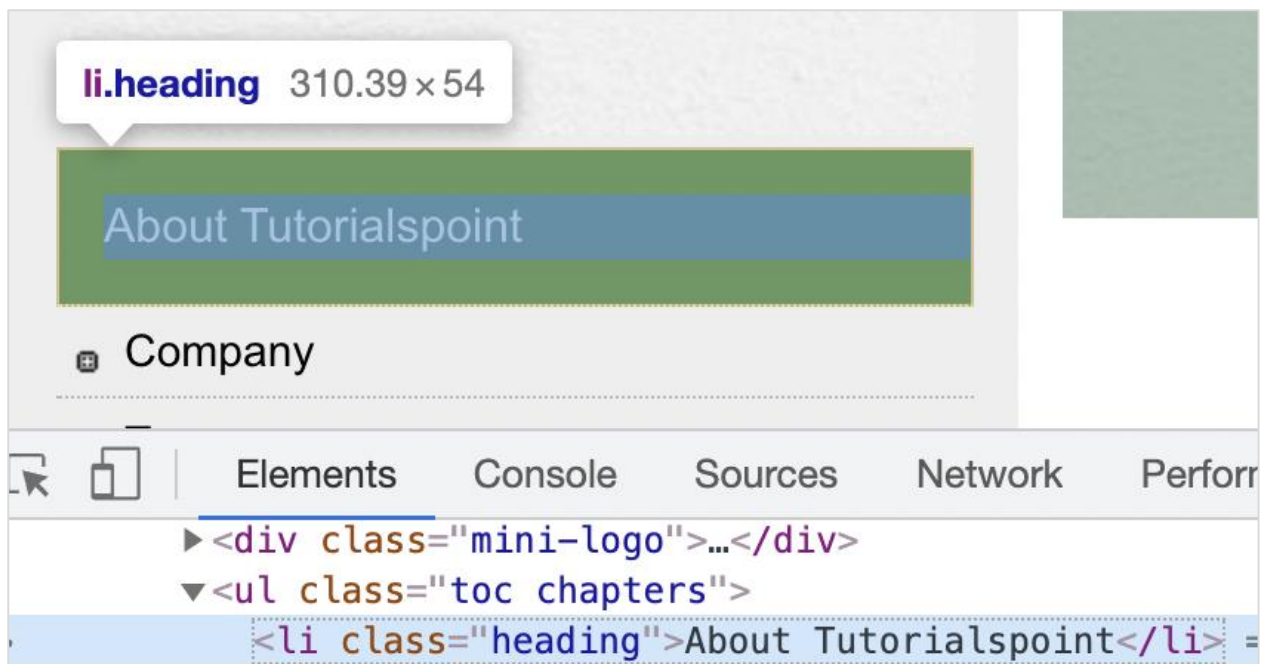
In the WebdriverIO code, we can specify the xpath expression of an element in the below format:

```
$('#value of the xpath expression')
```

Or, we can store this expression in a variable:

```
const p = $('#value of the xpath expression')
```

Let us identify the text highlighted in the below image and obtain its text:



The xpath of the above highlighted element should be as follows:

```
//li[@class='heading']
```

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Identify element with Xpath', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
    //identify element with xpath then obtain text
    console.log($(".//li[@class='heading']").getText() + " - is the
text.")
  });
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js.
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation

The following screen will appear on your computer:

```
(base) debomita@debtacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-06T18:31:14.754Z
0-0] RUNNING in chrome - /test/specs/testcase1.js
0-0] About Tutorialspoint - is the text.
0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: e95d7ab4f10bb5e3a3c7d3be631db7d3
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Identify element with Xpath
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (3.9s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:07
```

After the command has been executed successfully, the text of the element - About Tutorialspoint is printed in the console.

Xpath Locator with Text

Once we navigate to a webpage, we have to interact with the webelements available on the page like clicking a link/button, entering text within an edit box, and so on to complete our automation test case.

We can create an xpath for an element for its identification. However, there are scenarios where there are no HTML attributes or tagname available to uniquely identify an element.

In such a situation, we can create an xpath for an element with the help of the text visible on the page by using the text function. The text function is case-sensitive.

The rule to create a xpath expression with visible text is discussed below:

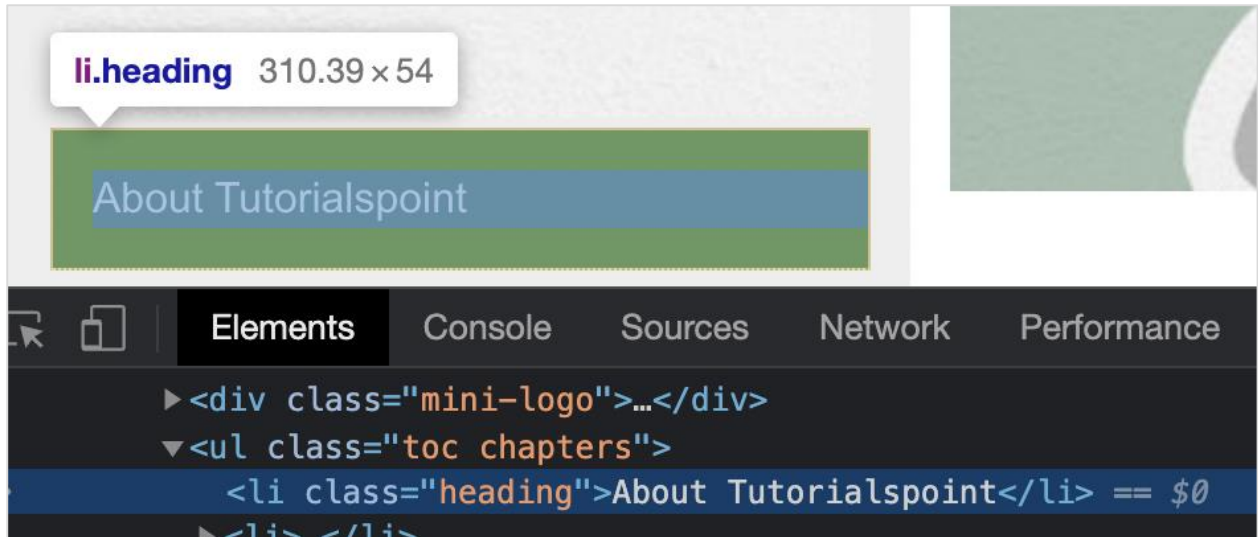
The syntax of xpath is as follows:

```
//tagname[text()='displayed text'].
```

For example,

```
//li[text()='WebdriverIO']
```

Let us identify the element highlighted in the below image with the help of the visible text in xpath:



The xpath of the above highlighted element using the text() function shall be as follows:

```
//li[text()='About Tutorialspoint']
```

To begin, follow Steps 1 to 5 from the Chapter - Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```

// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Identify element with Xpath - text()', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
    //identify element with xpath - visible text then obtain text
    console.log($(".//li[text()='About Tutorialspoint']").getText() + " - is
the text.")
  });
});

```

```
});
```

Run the Configuration file - wdio.conf.js file with the command:

```
npx wdio run wdio.conf.js.
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation

The following screen will appear on your computer:

```
(base) root@Debomitas-MacBook-Air webdriverIO # npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-14T02:48:10.347Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] About Tutorialspoint - is the text.
[0-0] PASSED in chrome - /test/specs/testcase1.js
Spec Files:      1 passed, 1 total (100% completed) in 00:00:08
```

After the command has been executed successfully, the text of the element - About Tutorialspoint is printed in the console.

14. WebdriverIO — CSS Locator

Once we navigate to a webpage, we have to interact with the webelements available on the page like clicking a link/button, entering text within an edit box, and so on to complete our automation test case.

For this, our first job is to identify the element. We can create a css for an element for its identification. The rules to create a css expression are discussed below:

The syntax of css is as follows:

```
tagname[attribute='value']
```

Here, the tagname is optional. We can also specifically use the id and class attribute to create a css expression.

With id, the format of a css expression should be tagname#id. For example, input#txt [here input is the tagname and the txt is the value of the id attribute].

With class, the format of the css expression should be tagname.class.

For example,

```
input.cls-txt
```

Here, input is the tagname and the cls-txt is the value of the class attribute.

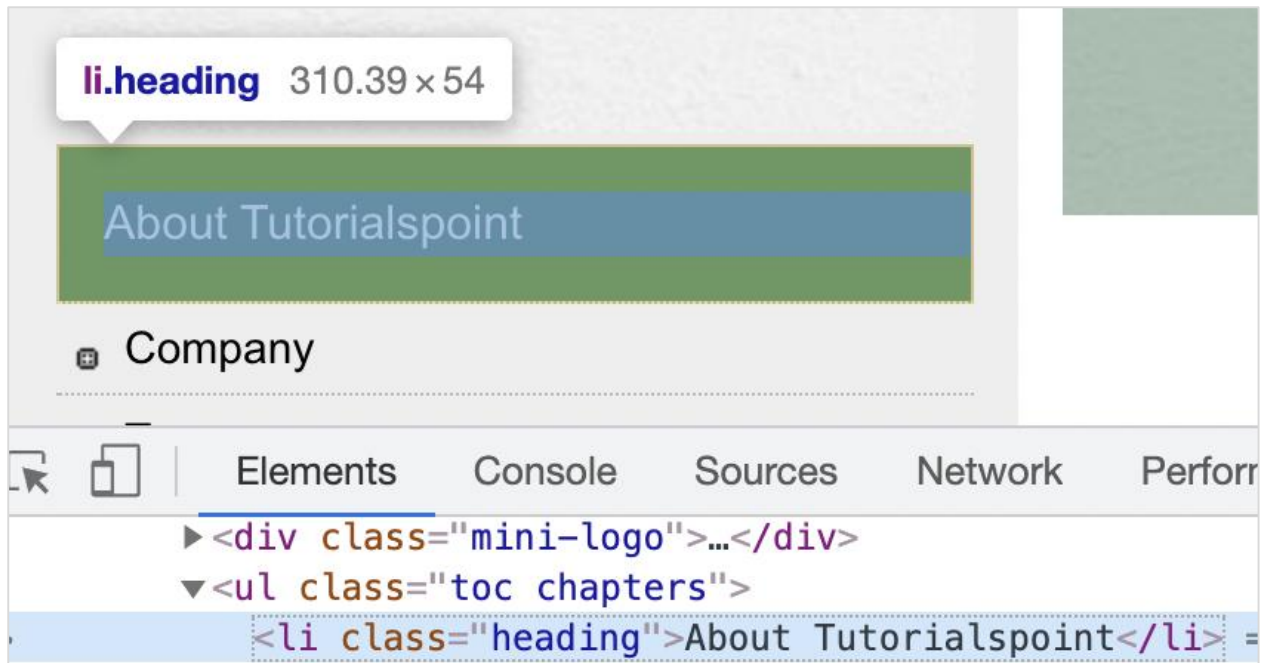
In the WebdriverIO code, we can specify the css expression of an element in the below format:

```
$('.value of the css expression')
```

Or, we can store this expression in a variable as follows:

```
const p = $('.value of the css expression')
```

Let us identify the text highlighted in the below image and obtain its text:



The css of the above highlighted element should be li.heading.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Identify element with CSS', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
    //identify element with CSS then obtain text
    console.log($(".li.heading").getText() + " - is the text.")
  });
});
```

```
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```
(base) debomitabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-06T18:39:50.634Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] About Tutorialspoint - is the text.
[0-0] PASSED in chrome - /test/specs/testcase1.js
"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 89fdb38998116955f66a22caff62be4c
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Identify element with CSS
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (3.9s)
Spec Files:      1 passed, 1 total (100% completed) in 00:00:07
```

After the command has been executed successfully, the text of the element - About Tutorialspoint is printed in the console.

15. WebdriverIO — Link Text Locator

Once we navigate to a webpage, we may interact with a webelement by clicking a link to complete our automation test case. The locator link text is used for an element having the anchor tag.

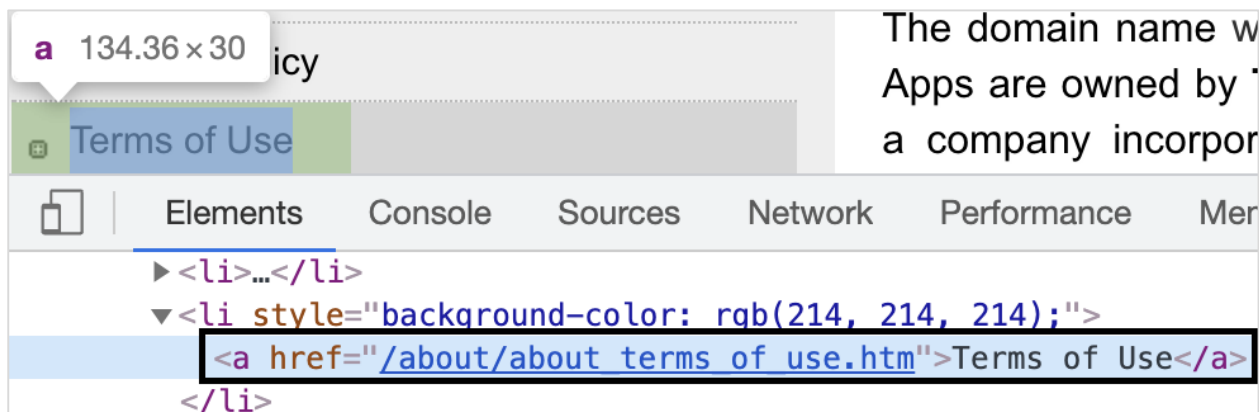
We can identify an anchor element with a matching text. In the WebdriverIO code, we have the option to specify the link of an element in the below format:

```
$('.=value of the anchor text')
```

Or, we can store this expression in a variable as follows:

```
const p = $('.=value of the anchor text')
```

Let us identify the link highlighted in the below image and click on it:



The link highlighted in the above image has a tagname - a and the anchor text - Terms of Use.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Identify element with Link Text', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
    //identify element with link text then click
    $("=Terms of Use").click()
    console.log('Page title after click: ' + browser.getTitle())
  });
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js.
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```
(base) debomitabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-06T19:12:09.754Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] Page title after click: Terms of Use - Tutorialspoint
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 228a1968d1fd02a1f5457116baa84df6
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Identify element with Link Text
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (6.6s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:09
```

After the command has been executed successfully, the title of the page after clicking - Terms of Use - Tutorialspoint is printed in the console.

Partial Link Text Locator

Once we navigate to a webpage, we may interact with a webelement by clicking a link to complete our automation test case. The locator partial link text is used for an element having the anchor tag.

We can identify an anchor element with a matching text. In the WebdriverIO code, we have the option to specify the partial link of an element in the below format:

```
$('.*=value of the anchor text which is matching')
```

Or, we can store this expression in a variable as follows:

```
const p = $('.*=value of the anchor text which is matching')
```

The partial link text is similar to link text with the only difference being that it assists in scenarios where a few characters of an anchor element are fixed and the remaining ones are dynamic.

Let us identify the link highlighted in the below image and click on it:



The link highlighted in the above image has a tagname - a and the anchor text - Terms of Use.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Identify element with Partial Link Text', function(){
```

```

// launch url
browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
//identify element with partial link text then click
$("#*=Terms of").click()
console.log('Page title after click: ' + browser.getTitle())
});
});

```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-07T00:07:49.347Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] Page title after click: Terms of Use - Tutorialspoint
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 9a732a0be2e677e9bf8df241510834f6
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Identify element with Partial Link Text
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (6s)

Spec Files:   1 passed, 1 total (100% completed) in 00:00:09

```

After the command has been executed successfully, the title of the page after clicking - Terms of Use - Tutorialspoint is printed in the console.

16. WebdriverIO — ID Locator

Once we navigate to a webpage, we have to interact with the webelements available on the page like clicking a link/button, entering text within an edit box, and so on to complete our automation test case.

For this, our first job is to identify the element. We can use the id attribute for an element for its identification. It is a very useful locator and speeds up the execution of automation tests in comparison to all the locators.

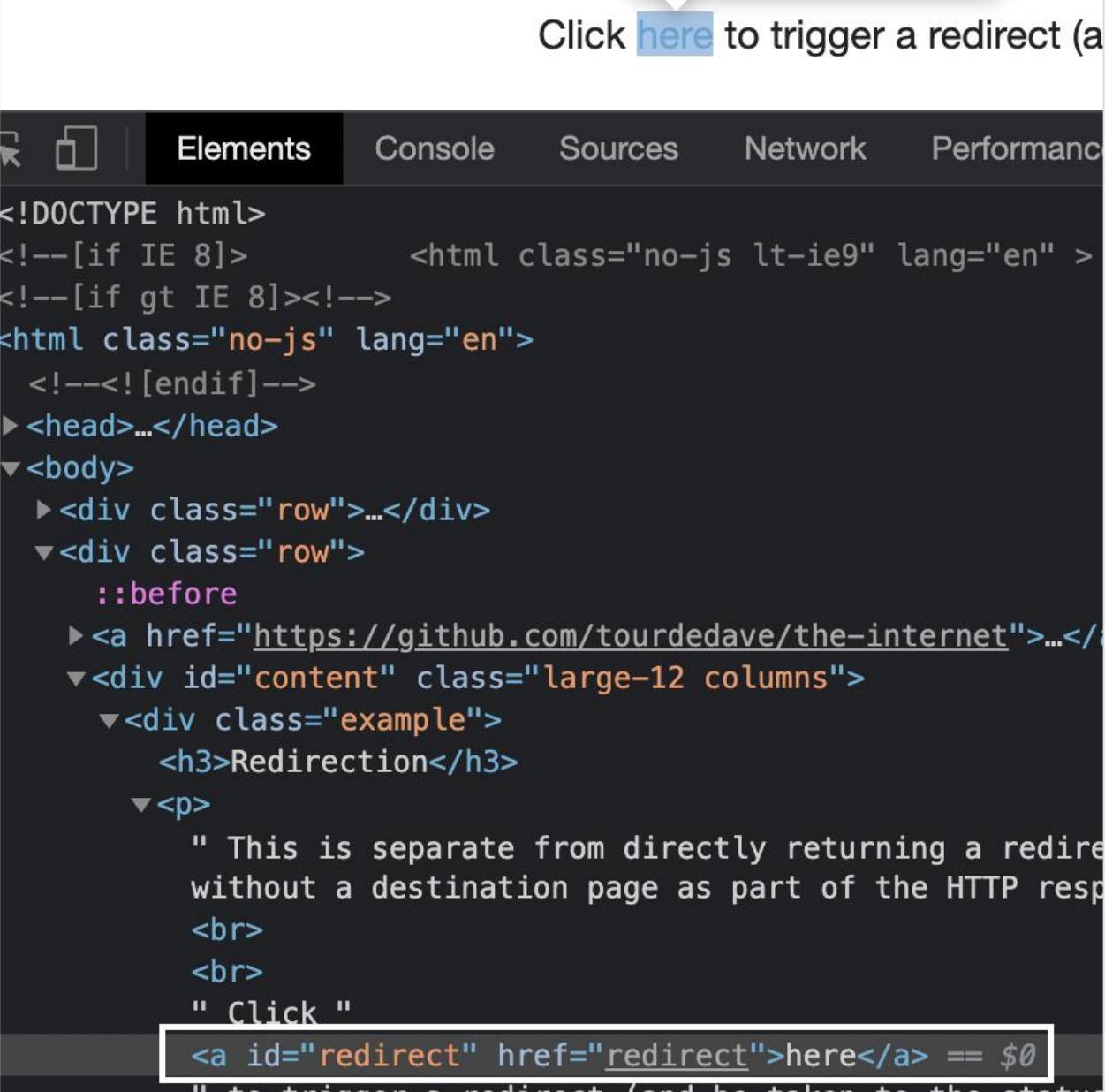
In the WebdriverIO code, we have the option to specify the value of the id attribute of an element in the below format:

```
$('#value of id attribute')
```

Or, we can store this expression in a variable as follows:

```
const p = $('#value of id attribute')
```

Let us identify the element highlighted in the below image and click on it:



```

Click here to trigger a redirect (a
Elements Console Sources Network Performance
<!DOCTYPE html>
<!--[if IE 8]> <html class="no-js lt-ie9" lang="en" >
<!--[if gt IE 8]><!-->
<html class="no-js" lang="en">
  <!--<![endif]-->
  <head>...</head>
  <body>
    <div class="row">...</div>
    <div class="row">
      ::before
      <a href="https://github.com/tourdedave/the-internet">...</a>
      <div id="content" class="large-12 columns">
        <div class="example">
          <h3>Redirection</h3>
          <p>
            " This is separate from directly returning a redirect
            without a destination page as part of the HTTP response
            <br>
            <br>
            " Click "
          <a id="redirect" href="redirect">here</a> == $0
            " to trigger a redirect (and be taken to the status

```

The link highlighted in the above image has a tagname - a and the id attribute value - redirect.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Identify element with Id', function(){
    // launch url
    browser.url('https://the-internet.herokuapp.com/redirector')
    //identify element with id then click
    $("#redirect").click()
    //obtain page title
    console.log('Page title after click: ' + browser.getTitle())
  });
});
```

Run the Configuration file - wdio.conf.js file with the command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```
(base) debomitabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-07T02:48:39.461Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] Page title after click: The Internet
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 9dfbdc7ec9a0c0bbff005a0a5a5c47a4
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Identify element with Id
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (2.4s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:06
```

After the command has been executed successfully, the title of the page after clicking - The Internet is printed in the console.

17. WebdriverIO — Tag Name Locator

Once we navigate to a webpage, we have to interact with the webelements available on the page like clicking a link/button, entering text within an edit box, and so on to complete our automation test case.

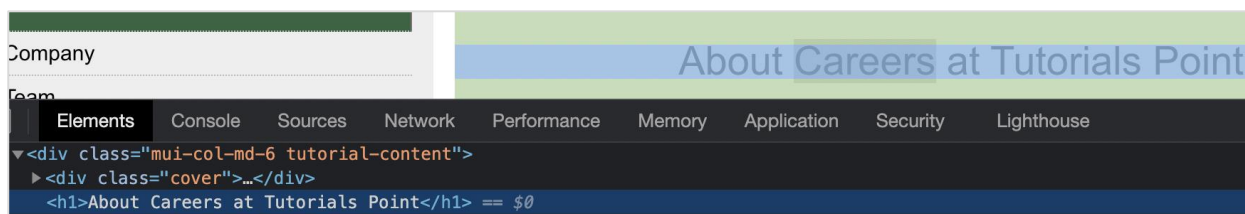
For this, our first job is to identify the element. We can use the HTML tagname for an element for its identification. In the WebdriverIO code, we have the option to specify the tagname of an element in the below format:

```
$('#<element tagname>')
```

Or, we can store this expression in a variable as follows:

```
const p = $('#element tagname')
```

Let us identify the element highlighted in the below image and obtain its text:



The element highlighted in the above image has a tagname - h1.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
```



```

it('Identify element with Tagname', function(){
  // launch url
  browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
  //identify element with tagname then obtain text
  console.log($(".<h1>").getText() + " - is the text.")
});
});

```

Run the Configuration file - wdio.conf.js file with the command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-07T03:05:06.698Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] About Careers at Tutorials Point - is the text
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 20e85934197ecdb8f7f9cbf898105b45
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Identify element with Tagname
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (4.2s)

```

After the command has been executed successfully, the text of the element - About Careers at Tutorials Point is printed in the console.

18. WebdriverIO — Class Name Locator

Once we navigate to a webpage, we have to interact with the webelements available on the page like clicking a link/button, entering text within an edit box, and so on to complete our automation test case.

For this, our first job is to identify the element. We can use the class name attribute for an element for its identification. It is a very useful locator and speeds up the execution of automation tests in comparison to xpath.

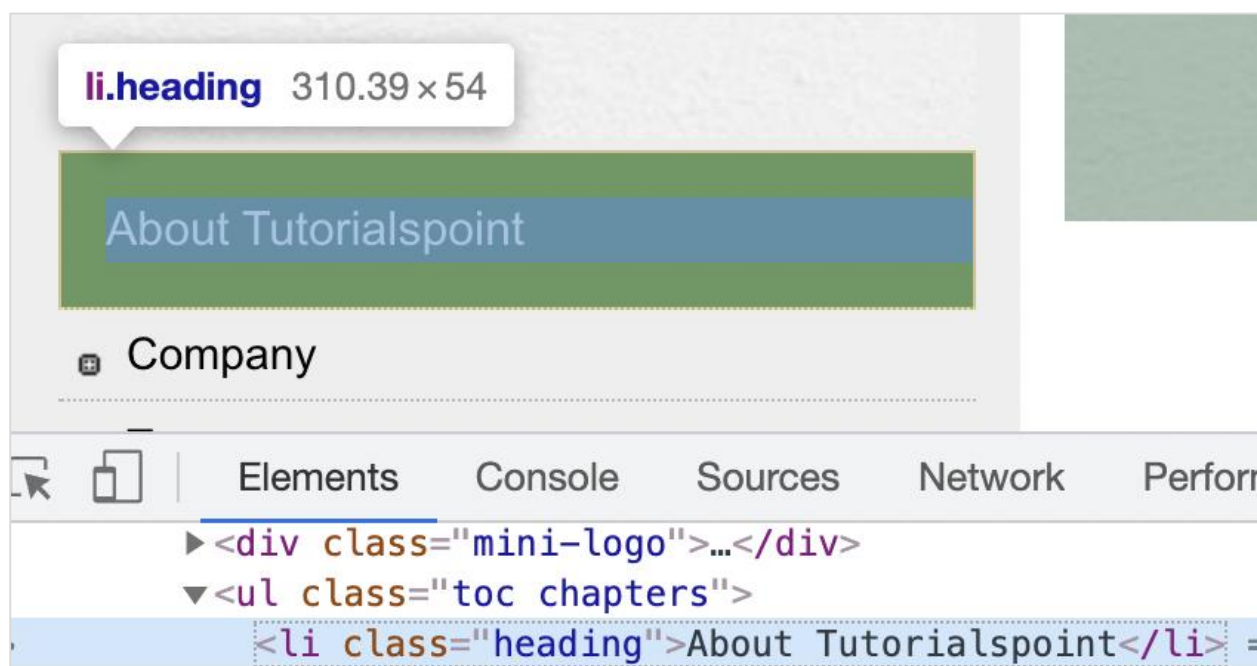
In the WebdriverIO code, we have the option to specify the value of the class name attribute of an element in the below format:

```
$('.value of class attribute')
```

Or, we can store this expression in a variable as follows:

```
const p = $('.value of class attribute')
```

Let us identify the text highlighted in the below image and obtain its text:



The element highlighted in the above image has the class attribute value as heading.

The Code Implementation is as follows:

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Identify element with Class Name', function(){
```

```

    // launch url
    browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
    //identify element with Class Name then obtain text
    console.log($(".heading").getText() + " - is the text.")
  });
});

```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation. The following screen will appear on your computer:

```

(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-07T18:59:15.272Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] About Tutorialspoint - is the text.
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: cc813b1601c56a29b8502e39a8c71129
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Identify element with Class Name
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (3.8s)

```

After the command has been executed successfully, the text of the element - About Tutorialspoint is printed in the console.

19. WebdriverIO — Name Locator

Once we navigate to a webpage, we have to interact with the webelements available on the page like clicking a link/button, entering text within an edit box, and so on to complete our automation test case.

For this, our first job is to identify the element. We can use the name attribute for an element for its identification. This locator is deprecated now and is only compatible with old browsers that are based on JSONWireProtocol or Appium.

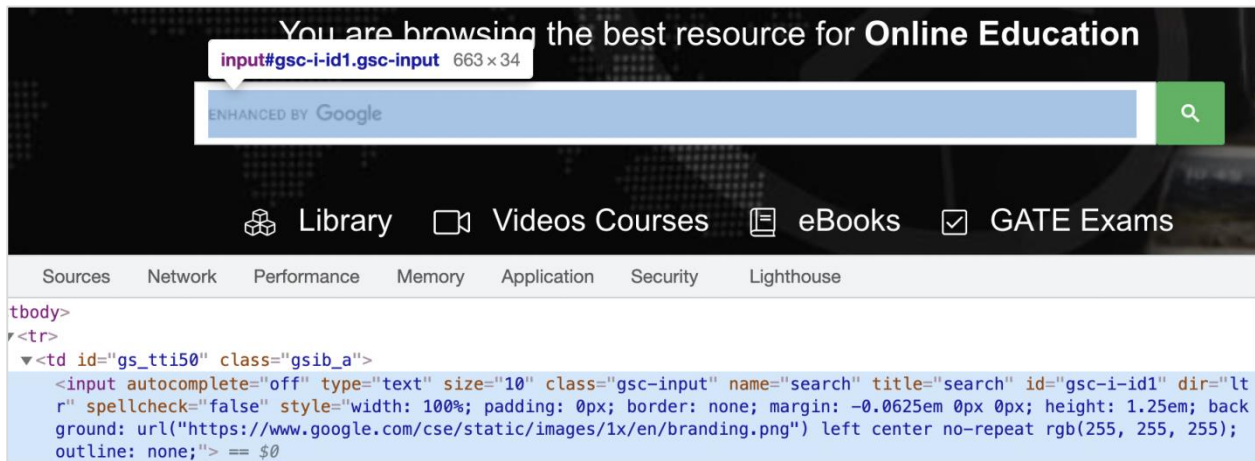
In the WebdriverIO code, we have the option to specify the value of the name attribute of an element in the below format:

```
$('#[name attribute='value']')
```

Or, we can store this expression in a variable as follows:

```
const p = $('#[name attribute='value']')
```

Let us identify the edit box highlighted in the below image and enter text:



The element highlighted in the above image has the name attribute value as search.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Identify element with Name', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/index.htm')
    //identify element with Name then input text
    $('[name="search"]').setValue('Selenium')
  });
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

20. WebdriverIO — Expect statement for assertions

To use WebdriverIO as an automation testing tool, we need to have checkpoints which will help us to conclude if our test has passed or failed. There are various assertions available in WebdriverIO with which we can verify if the test has successfully validated a step.

In assertion, we can compare an expected result of a test with an actual. If both are similar, a test should pass, else it should fail. The expect statement in WebdriverIO can be applied on the browser, a mock object, or an element.

We have to add a NodeJS library called Chai. Chai library contains the expect statement that is used for the Assertion.

We have to add the below statement in our code to implement the Chai Assertion:

```
const e = require('chai').expect
```

Assertions applied to browsers

These assertions are listed below:

toHaveUrl

It checks whether the browser has opened a particular page. The syntax is as follows:

```
expect(browser).toHaveUrl('https://www.tutorialspoint.com/index.htm')
```

toHaveUrlContaining

It checks whether the URL of a page has a particular value.

The syntax is as follows:

```
expect(browser).toHaveUrlContaining('tutorialspoint')
```

toHaveUrl

It checks whether the page has a particular title.

The syntax is as follows:

```
expect(browser).toHaveTitle('Terms of Use - Tutorialspoint')
```

Assertions applied on elements

These assertions are listed below:

toBeDisplayed

It checks whether an element is displayed.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toBeDisplayed()
```

toExist

It checks whether an element exists.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toExist()
```

toBePresent

It checks whether an element is present.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toBePresent()
```

toBeExisting

It is similar to toExist.

toBeFocussed

It checks whether an element is focused or not.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toBeFocussed()
```

toHaveAttribute

It checks whether an element attribute has a particular value.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toHaveAttribute('name', 'search')
```

toHaveAttr

It is similar to toExist.

toHaveAttributeContaining

It checks whether an element attribute contains a particular value.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toHaveAttributeContaining('name', 'srch')
```

toHaveElementClass

It checks whether an element has a particular class name.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toHaveElementClass('name', { message: 'Not available!', })
```

toHaveElementClassContaining

It checks whether an element class name contains a particular value.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toHaveElementClassContaining('nam')
```

toHaveElementProperty

It checks whether an element has a particular property.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toHaveElementProperty('width', 15)
//verify negative scenario
expect(e).not.toHaveElementProperty('width', 20)
```

toHaveValue

It checks whether an input element has a particular value.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toHaveValue('Selenium', { ignoreCase: false})
```

toHaveValueContaining

It checks whether an input element contains a particular value

The syntax is as follows:

```
const e = $('#loc')
expect(e).toHaveValueContaining('srch')
```

toBeClickable

It checks whether an element is clickable.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toBeClickable()
```

toBeDisabled

It checks whether an element is disabled.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toBeDisabled()
//verify negative scenario
expect(e).not.toBeEnabled()
```

toBeEnabled

It checks whether an element is enabled.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toBeEnabled()
```

toBeSelected

It is the same as toBeEnabled.

toBeChecked

It is the same as toBeEnabled.

toHaveHref

It checks whether a link element has a particular link target.

The syntax is as follows:

```
const e = $('<a>')
expect(e).toHaveHref('https://www.tutorialspoint.com/index.htm')
```

toHaveLink

It is same as toHaveHref.

toHaveHrefContaining

It checks whether a link element contains a particular link target.

The syntax is as follows:

```
const e = $('<a>')
expect(e).toHaveHrefContaining('tutorialspoint.com')
```

toHaveLinkContaining

It is the same as HaveHrefContaining.

toHaveId

It checks whether an element has a particular id attribute value.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toHaveId('loc')
```

toHaveText

It checks whether an element has a particular text.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toHaveText('Learning WebdriverIO')
```

toHaveTextContaining

It checks whether an element contains a particular text.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toHaveTextContaining('Learning WebdriverIO')
```

toBeDisplayedInViewpoint

It checks whether an element is within the viewpoint.

The syntax is as follows:

```
const e = $('#loc')
expect(e).toBeDisplayedInViewpoint()
```

Assertions applied to mock objects

The assertions are listed below:

toBeRequested

It checks whether a mock was called.

The syntax is as follows:

```
const m = browser.mock('**/api/list*')
expect(m).toBeRequested()
```

toBeRequestedTimes

It checks whether a mock was called for an expected number of times.

The syntax is as follows:

```
const m = browser.mock('**/api/list*')
expect(m).toBeRequestedTimes(2)
```

To begin, follow the steps 1 to 5 from the Chapter titled Happy path flow with webdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Assertion with expect', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
    //identify element with link text then click
    $("=Terms of Use").click()
    browser.pause(1000)
    //verify page title with assertion
    expect(browser).toHaveTitleContaining('Terms of Use - Tuter')
  });
});
```

Run the Configuration file - wdio.conf.js file with the command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```
(base) debomitabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js

Execution of 1 workers started at 2021-06-08T03:34:00.956Z

[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] Error in "Tutorialspoint application Assertion with expect"
Error: Expect window to have title containing

Expected: "Terms of Use - Tutor"
Received: "Terms of Use - Tutorialspoint"
    at Context.<anonymous> (/Users/debomitabhattacharjee/webdriverIO/test/specs/testcase1.js:12:28)
[0-0] FAILED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 6fc2c9be945e441a08b913c418214806
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] > /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0] * Assertion with expect
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 failing (17s)
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1) Tutorialspoint application Assertion with expect
[chrome 91.0.4472.77 mac os x #0-0] Expect window to have title containing

Expected: "Terms of Use - Tutor"
Received: "Terms of Use - Tutorialspoint"
[chrome 91.0.4472.77 mac os x #0-0] Error: Expect window to have title containing
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] Expected: "Terms of Use - Tutor"
[chrome 91.0.4472.77 mac os x #0-0] Received: "Terms of Use - Tutorialspoint"
[chrome 91.0.4472.77 mac os x #0-0] at Context.<anonymous> (/Users/debomitabhattacharjee/webdriverIO/test
/specs/testcase1.js:12:28)

Spec Files: 0 passed, 1 failed, 1 total (100% completed) in 00:00:20
```

After the command has been executed successfully, we find the result as 1 failed. Since the Expected: is Terms of Use - Tutor and the Received: output is Terms of Use - Tutorialspoint.

Also, the WebdriverIO expect statement has highlighted the part of the text where the Expected: and the Received: texts are not matching.

21. WebdriverIO — Happy path flow

Let us create a simple happy flow to demonstrate how to create a basic WebdriverIO test:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Happy Flow', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
    //identify element with link text then click
    $("=Team").click()
    //verify URL of next page with assertion
    expect(browser).toHaveUrlContaining('team')
  });
});
```

Step 7: Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```
(base) debomitabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-08T04:09:52.960Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 3f581b5f012df7e2cbaf604b9fb34605
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Happy Flow
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (6s)

Spec Files:   1 passed, 1 total (100% completed) in 00:00:09
```

Step 8: On investigating further on the output, we shall see the test within the spec file testcase1.js is marked as PASSED.

The browser version and operating system on which the test has been executed, session id, name of the spec file, test suite name - Tutorialspoint Application, test case name - Happy Flow, duration of test execution, and so on, have also been captured in the console.

22. WebdriverIO — General Browser Commands

Some of the general browser commands used in WebdriverIO are listed below:

browser.url(URL)

This command is used to launch an application whose URL is passed as a parameter.

The syntax is as follows:

```
browser.url('https://the-internet.herokuapp.com/redirector')
```

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with webdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Identify element with Id', function(){
    // launch url
    browser.url('https://the-internet.herokuapp.com/redirector')
    //identify element with id then click
    $("#redirect").click()
    //obtain page title
    console.log('Page title after click: ' + browser.getTitle())
  });
});
```

browser.getTitle()

This command is used to get the title of a page presently launched in the browser. The value is returned in the form of a string. This command does not accept any parameters. If the page has no title, a null string is returned.

The syntax is as follows:

```
browser.getTitle()
```

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint Application', function () {
// test case name
it('Get Page Title', function (){
// URL launching
browser.url("https://www.tutorialspoint.com/about/about_careers.htm")
//print page title in console
console.log(browser.getTitle())
});
});
```

browser.getUrl()

This command is used to get the URL of a page presently launched in the browser. The value is returned in the form of a string. This command does not accept any parameters.

The syntax is as follows:

```
browser.getUr1()
```

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint Application', function () {
// test case name
it('Get Url', function (){
// URL launching
browser.url("https://www.tutorialspoint.com/index.htm")
//print URL in console
console.log(browser.getUrl())
});
});
```

browser.getPageSource()

This command is used to get the page source of a page presently launched in the browser. The value is returned in the form of a string. This command does not accept any parameters.

The syntax is as follows:

```
browser.getPageSource()
```

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint Application', function () {
// test case name
it('Get Page Source', function (){
// URL launching
browser.url("https://www.tutorialspoint.com/index.htm")
//print URL in console
console.log(browser.getPageSource())
});
});
```

browser.maximizeWindow()

This command is used to maximise the present browser window.

The syntax is as follows:

```
browser.maximizeWindow()
```

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint Application', function () {
// test case name
it('Maximise Browser', function (){
// URL launching
```

```
        browser.url("https://www.tutorialspoint.com/questions/index.php")
    //maximize browser
        browser.maximizeWindow()
    });
});
```

23. WebdriverIO — Handling Browser Size

While working on automation tests in WebdriverIO, we may be required to set the size of the window and obtain the size of the window. The window size refers to the window height and width.

browser.setWindowSize(250, 450)

This command is used to set the window size. Here, the window size shall be set to width - 250 and height - 450.

The syntax is as follows:

```
browser.setWindowSize(250, 450)
```

browser.getWindowSize()

This command is used to get the window dimension.

The syntax is as follows:

```
browser.getWindowSize()
```

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Dimension', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/index.htm')
    //set window size
```

```

        browser.setWindowSize(500, 450)
        //get window size
        console.log(browser.getWindowSize())
    });
});

```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) debomita@hattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-08T22:57:59.605Z

[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] { width: 500, height: 450 }
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: f08902ed0a907719377d5fbea9716f18
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]     ✓ Dimension
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (2.6s)

```

After the command has been executed successfully, the dimension of the browser window- {width: 500, height: 450} is printed in the console.

24. WebdriverIO — Browser Navigation Commands

Some of the browser navigation commands used in WebdriverIO are listed below:

browser.navigateTo(URL)

This command is used to navigate to an application whose URL is passed as a parameter.

The syntax is as follows:

```
browser.navigateTo('https://the-internet.herokuapp.com/redirector')
```

browser.back()

This command is used to navigate back in browser history.

The syntax is as follows:

```
browser.back()
```

browser.forward()

This command is used to navigate forward in browser history.

The syntax is as follows:

```
browser.forward()
```

browser.refresh()

This command is used to refresh the present webpage.

The syntax is as follows:

```
browser.refresh()
```

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Navigation', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
    // navigate to another url
    browser.navigateTo("https://www.tutorialspoint.com/codingground.htm")
    //navigate back in history
    browser.back()
    //get title back in browser history
    console.log('Back in Browser history: ' + browser.getTitle())
    //navigate forward in history
    browser.forward()
    //get title forward in browser history
    console.log('Forward in Browser history: ' + browser.getTitle())
    //refresh browser
    browser.refresh()
    //get title after refresh
    console.log('Page Title after refresh: ' + browser.getTitle())
  });
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```
(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-09T03:25:42.932Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] Back in Browser history: About Careers at Tutorials Point - Tutorialspoint
[0-0] Forward in Browser history: Free Online IDE and Terminal
[0-0] Page Title after refresh: Free Online IDE and Terminal
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: bac7437532e7136b828097e52ebbc02a
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Navigation
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (7.2s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:10
```

After the command has been executed successfully, the page title obtained on navigating back in browser history - About Careers at Tutorials Point - Tutorialspoint is printed.

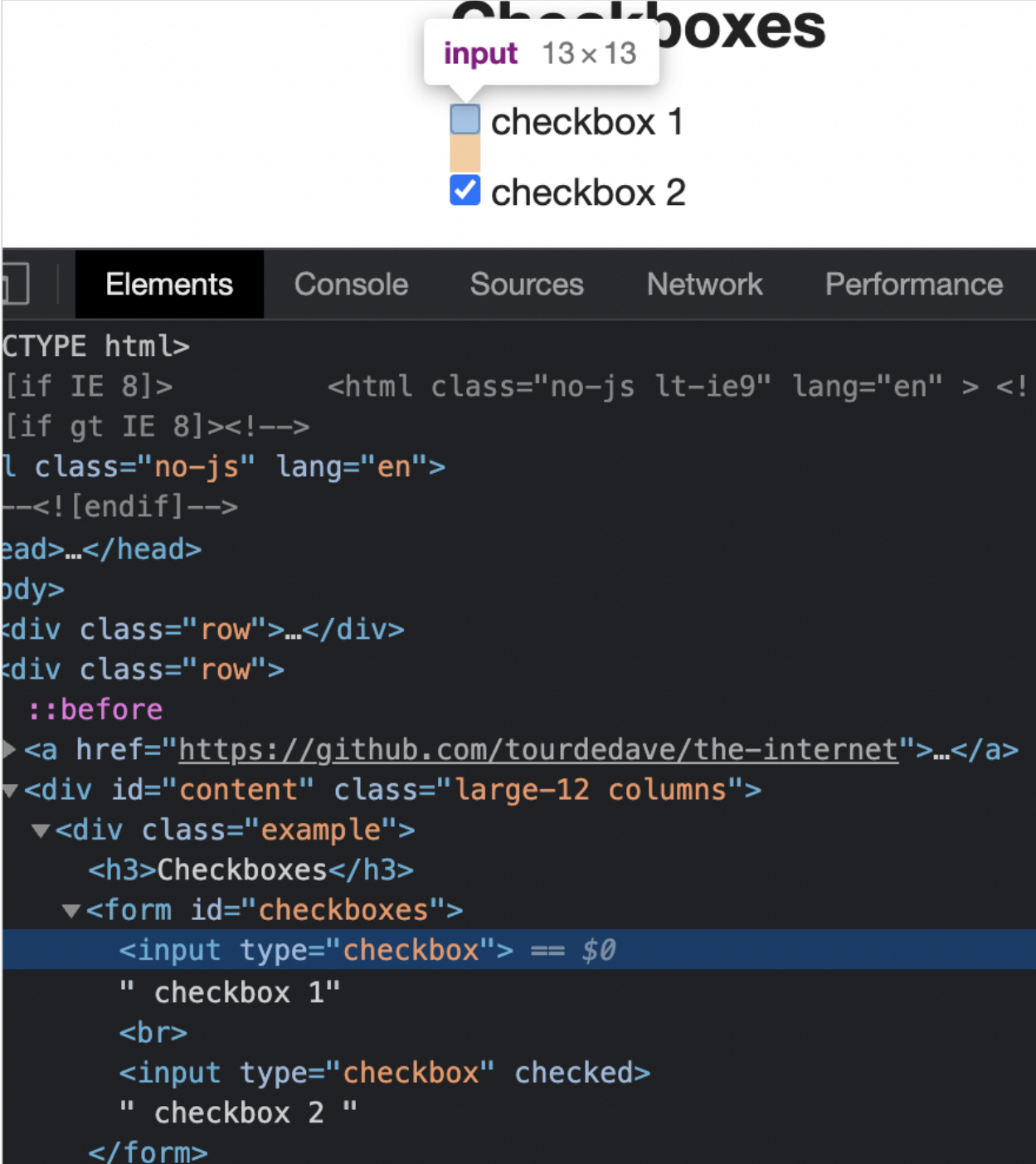
Then, the page title obtained on navigating forward in browser history - Free Online IDE and Terminal is printed.

Finally, the page title obtained after page refresh - Free Online IDE and Terminal is printed.

25. WebDriverIO — Handling Checkboxes and Dropdowns

We can handle checkboxes in the UI while automating a test using WebDriverIO. The checkboxes are identified in the html code with the tagname as input and type as checkbox.

The following screen will appear on your computer:



The screenshot shows a web browser window with a form titled "Checkboxes". The form contains two checkboxes: "checkbox 1" (unchecked) and "checkbox 2" (checked). A tooltip over the first checkbox indicates it is an "input" element with dimensions "13 x 13". Below the browser window, the Chrome DevTools "Elements" panel is open, showing the HTML structure of the page. The selected element is the first checkbox, with the following HTML code:

```
CTYPE html>
[if IE 8]>          <html class="no-js lt-ie9" lang="en" > <!
[if gt IE 8]><!-->
l class="no-js" lang="en">
--<![endif]-->
ead>...</head>
ody>
<div class="row">...</div>
<div class="row">
  ::before
  ><a href="https://github.com/tourdedave/the-internet">...</a>
  ><div id="content" class="large-12 columns">
    ><div class="example">
      ><h3>Checkboxes</h3>
      ><form id="checkboxes">
        ><input type="checkbox"> == $0
        " checkbox 1"
        <br>
        ><input type="checkbox" checked>
        " checkbox 2 "
      </form>
```

Methods to work with Checkboxes

Some methods to work with checkboxes are as follows:

click()

It is used to check a checkbox.

The syntax is as follows:

```
let p = $('#loc')
p.click()
```

isSelected()

It is used to check if an element of type checkbox is selected or not. It returns a Boolean value (true if checked, false if not).

The syntax is as follows:

```
let p = $('#loc')
p.isSelected()
```

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Checkbox', function(){
    // launch url
    browser.url('https://the-internet.herokuapp.com/checkboxes')
    //identify checkbox with CSS then click
    const p = $("input[type='checkbox']")
    p.click()
```

```

    //verify if checked with assertion
    expect(p).toBeSelected()
    //uncheck checkbox
    p.click()
    //verify if not checked with assertion
    expect(p).not.toBeSelected()
  });
});

```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) debomitabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-09T03:59:12.410Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 2f378923bbe3d9a95860a855c9e9d2db
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Checkbox
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (1.8s)

Spec Files:   1 passed, 1 total (100% completed) in 00:00:05

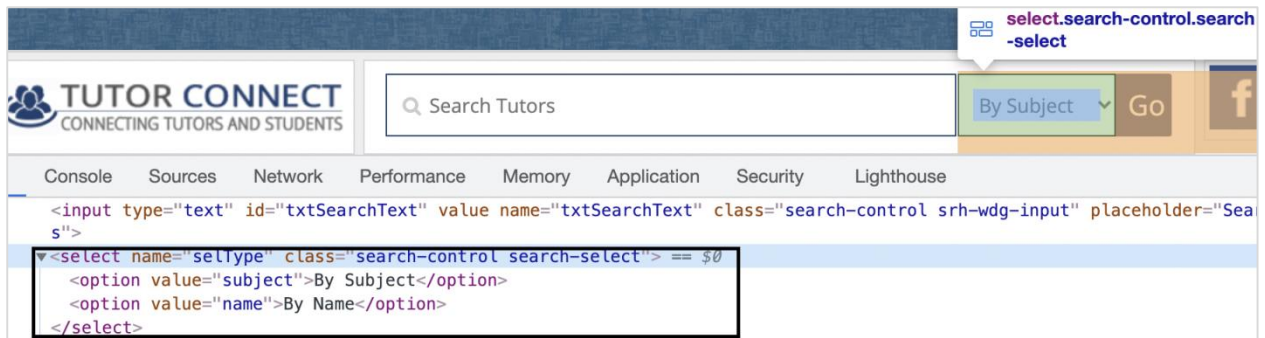
```

After the command has been executed successfully, all the Assertions are executed as per expectation and we have received a passed test.

Handling Dropdowns

We can handle drop downs in the UI while automating a test using WebdriverIO. The static drop downs are identified in the html code with the tagname as select and its options have the tagname as option.

The following screen will appear on your computer:



Methods for Static Dropdowns

Some methods to work with static dropdowns are as follows:

selectByVisibleText

This method is used to select an option which matches with the visible text of an option passed as a parameter to this method.

The syntax is as follows:

```
let p = $('#loc')
p.selectByVisibleText('By Subject')
```

selectByAttribute

This method is used to select an option which matches with the value of any attribute passed as a parameter to this method.

The syntax is as follows:

```
let p = $('#loc')
p.selectByAttribute('value', 'subject')
```

Here, the option has the attribute with value as subject.

selectByIndex

This method is used to select an option which matches with the index/position of an option passed as a parameter to this method. The index starts with 0.

The syntax is as follows:

```
let p = $('#loc')
p.selectByIndex(1)
```

getValue()

This method is used to get the attribute value of an option selected in the dropdown.

The syntax is as follows:

```
let p = $('#loc')
```

```
p.getValue()
```

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Drodowns', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/tutor_connect/index.php')
    //identify dropdown
    const p = $("select[name='selType']")
    //select by index
    p.selectByIndex(1)
    //get option selected
    console.log(p.getValue() + ' - option selected by index')
    //select by visible text
    p.selectByVisibleText('By Subject')
    //get option selected
    console.log(p.getValue() + ' - option selected by visible text')
    //select by value attribute
    p.selectByAttribute('value', 'name')
    //get option selected
    console.log(p.getValue() + ' - option selected by attribute value')
  });
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```
(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-09T19:14:09.109Z

[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] name - option selected by index
[0-0] subject - option selected by visible text
[0-0] name - option selected by attribute value
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: e5df4d75b46310a452572e9a1a0c122c
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Drodowns
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (4.4s)

Spec Files:   1 passed, 1 total (100% completed) in 00:00:07
```

After the command has been executed successfully, first the value of the option selected with the option index - name is printed in the console.

Then, the value of the option selected with the option visible text - subject is printed in the console.

Finally, the value of the option selected with the option attribute value - name is printed in the console.

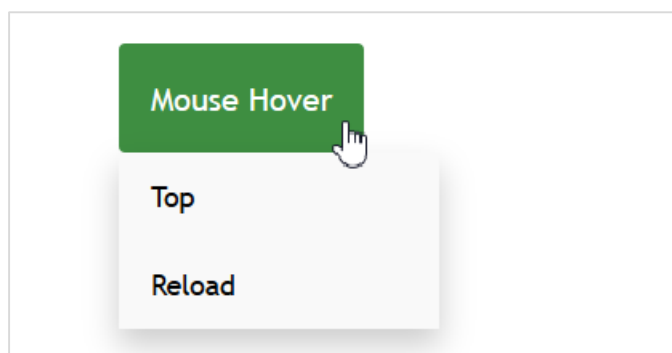
26. WebdriverIO — Mouse Operations

WebdriverIO can perform operations like hovering a mouse on an element by using the `moveTo` method. This method shall move the mouse to the middle of the element.

The syntax is as follows:

```
let p = $('#loc')
p.moveTo()
```

In the below image, on hovering over the Mouse Hover button, the Top and Reload buttons get displayed.



On moving the mouse out of the Mouse Hover button, the Top and Reload buttons get hidden.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
```

```

it('Mouse Operatio', function(){
  // launch url
  browser.url('https://courses.letscodeit.com/practice')
  //identify element then hover mouse
  const e = $(".dropbtn")
  //scroll to element then mouse hover
  e.scrollIntoView()
  e.moveTo()
  browser.pause(2000)
  //verify if sub-element display on hovering
  console.log($('=Top').isDisplayed())
});
});

```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-12T03:09:53.386Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] true
[0-0] PASSED in chrome - /test/specs/testcase1.js
"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: c00949c8dc1b7a987d2e18c39536a738
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Mouse Operatio
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (5.3s)
Spec Files:      1 passed, 1 total (100% completed) in 00:00:08

```

After the command has been executed successfully, the boolean value is printed in the console. This is returned by the isDisplayed() function which returns true if an element is displayed on the page.

27. WebdriverIO — Handling Child Windows/Pops

A new child window can open on clicking a link or a button. WebdriverIO by default has control over the main browser window, in order to access the elements on the child window, the WebdriverIO control has to be switched from the main page to the child window.

Methods for Child Windows

Some of the methods to work with child windows are as follows:

browser.getWindowHandles()

This method yields the window handle ids of all the currently opened browser windows in the form of a list. If there are two opened windows, the zero index of the list has the handle id of the parent window and the first index shall point to the window handle of the child.

The syntax is as follows:

```
var x = browser.getWindowHandles()
```

browser.getWindowHandle()

This method yields the window handle id of the browser which is in focus.

The syntax is as follows:

```
let l = browser.getWindowHandle()
```

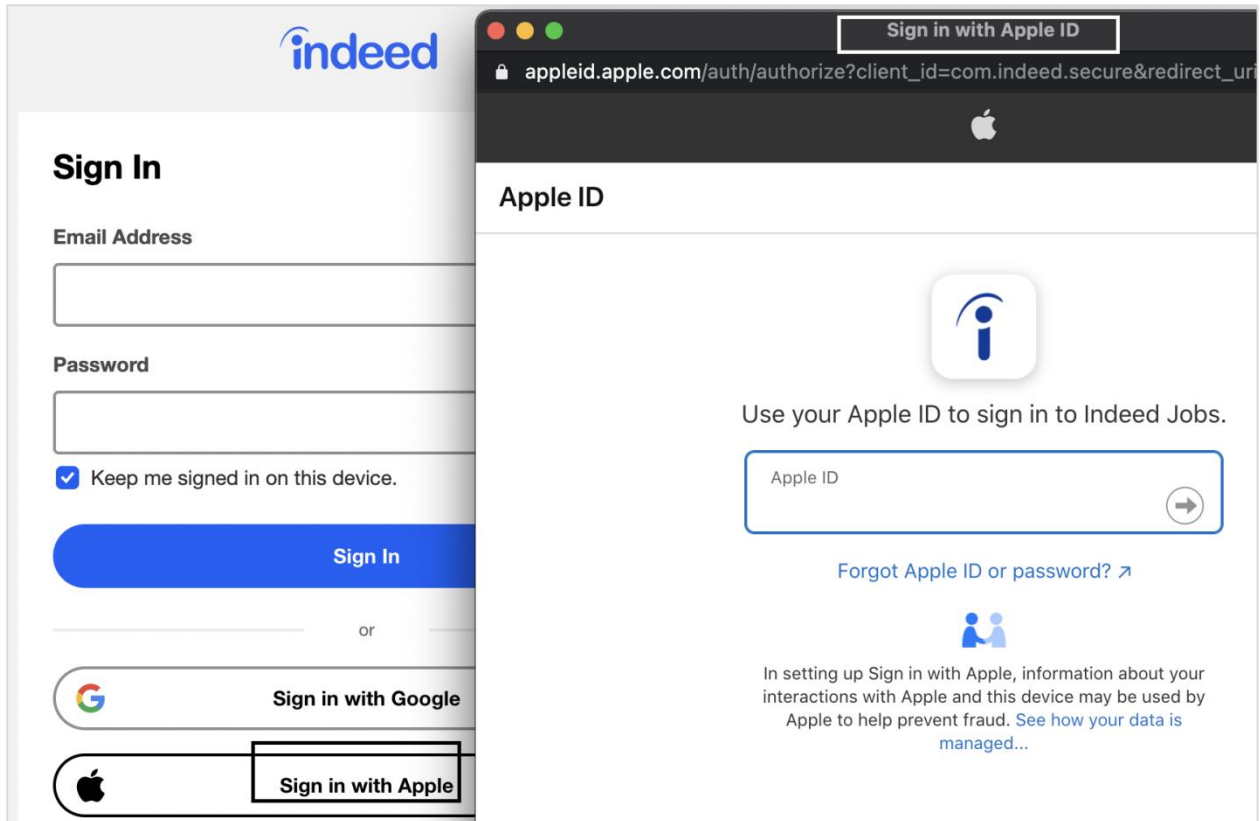
browser.switchToWindow('<window handle id>')

This method is used to switch focus from one browser window to another opened window whose window handle id is passed as a parameter to this method.

The syntax is as follows:

```
browser.switchToWindow(x)
```

In the below image, on clicking the Sign in with Apple button, a child window opens having the browser title as Sign in with Apple ID. Let us try to switch to the child window and access elements there.



To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Child Window', function(){
    // launch url
    browser.url('https://secure.indeed.com/account/login')
    //identify element then click
```

```

    $('#apple-signin-button').click()
    //get all window handle ids in list
    var l = browser.getWindowHandles()
    //switch to child window
    browser.switchToWindow(l[1])
    //get page title of child window
    console.log(browser.getTitle() + ' - Page title of child window')
    //close child window
    browser.closeWindow()

    //switch to parent window
    browser.switchToWindow(l[0])
    //get page title of parent window
    console.log(browser.getTitle() + ' - Page title of parent window')
  });
});

```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) debomitabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-10T03:26:16.566Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] Sign in with Apple ID - Page title of child window
[0-0] Sign In | Indeed Accounts - Page title of parent window
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 0198f27d2200da68a69295edc51b6762
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]     ✓ Drowdowns
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (8.5s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:11

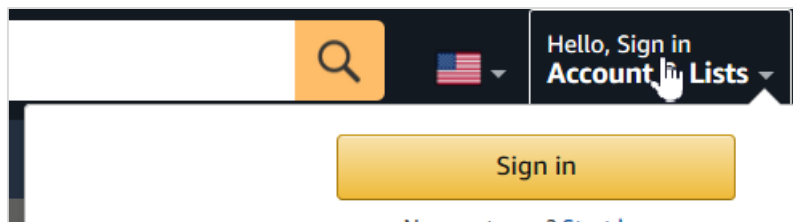
```

After the command has been executed successfully, first the page title of the child window - Sign in with Apple ID gets printed in the console. Then, the page title of the parent window - Sign In | Indeed Accounts get printed in the console.

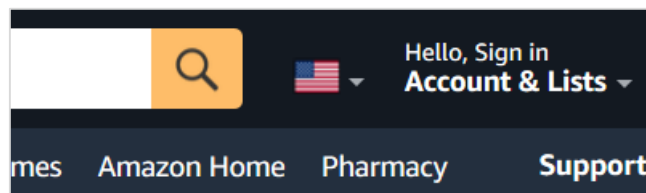
28. WebdriverIO — Hidden Elements

WebdriverIO can handle hidden elements. There are occasions when submenus get displayed only on hovering over the main menu. These submenus are initially hidden with the CSS property - display:none.

In the below image, on hovering over the Sign in menu, the Sign in button gets displayed.



On moving the mouse out of the Sign in menu, the Sign in button gets hidden.



To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Invisible Element', function(){
    // launch url
```

```

    browser.url('https://www.amazon.com/')
    //identify element then hover mouse
    const e = $("#nav-link-accountList")
    e.moveTo()
    browser.pause(2000)
    //click on hidden element
    $('=Sign in').click()
    //get page title
    console.log(browser.getTitle() + ' - Page title after click')
  });
});

```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-12T03:18:24.802Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] Amazon Sign-In - Page title after click
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 421213bb2e2bda830cdb5534447b68e3
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Invisible Element
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (15.5s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:18

```

After the command has been executed successfully, the page title obtained by clicking the hidden Sign in button - Amazon Sign-In gets printed in the console.

29. WebDriverIO — Frames

The frames in an html code are represented by the frames/iframe tag. WebDriverIO can handle frames by switching from the main page to the frame.

Methods for Frames

Some methods to work with frames are as follows:

browser.switchToFrame('<frame id/index/locator>')

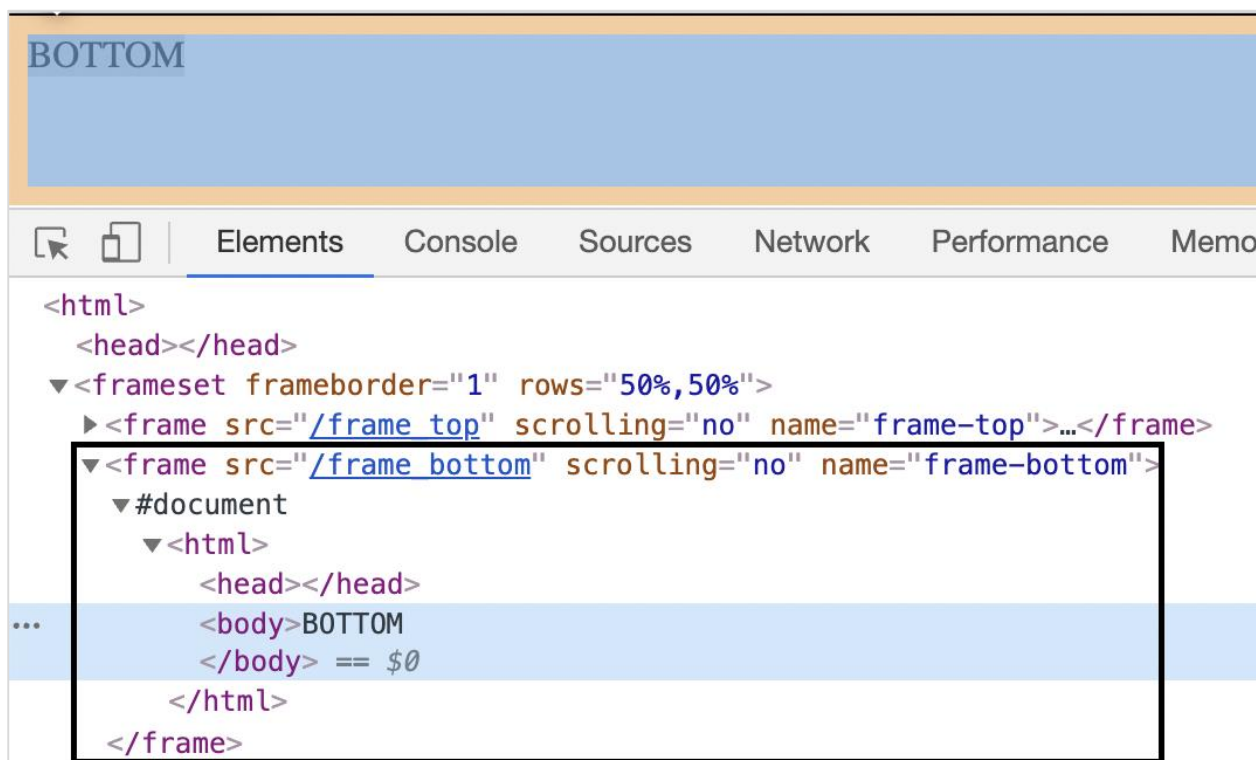
This method is used to switch focus from the main page to a frame. The frame id, index or locator is passed as a parameter to this method.

The syntax is as follows:

```
browser.switchToWindow(x)
```

To switch the focus from the frame to the main page, we have to pass null as a parameter to the browser.switchToFrame method.

Let us see the html code of an element inside a frame and obtain the text - BOTTOM inside it.



The screenshot shows a browser window with a blue frame containing the text "BOTTOM". Below the browser window, the developer tools are open to the "Elements" tab. The DOM tree shows a frameset with two frames: "frame-top" and "frame-bottom". The "frame-bottom" frame is selected and expanded, showing its content: a document with an HTML body containing the text "BOTTOM".

The tagname highlighted in the above image is frame and the value of its name attribute is frame-bottom.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Frames', function(){
    // launch url
    browser.url('https://the-internet.herokuapp.com/nested_frames')
    //switch to frame
    browser.switchToFrame($"frame[name='frame-bottom']")
    //identify element with tagname
    const p = $('<body>')
    //get text inside frame
    console.log(p.getText() + ' - Text inside frame')
    //switch to main page
    browser.switchToFrame(null)
  });
});
```

Run the Configuration file - wdio.conf.js file with the command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation

The following screen will appear on your computer:

```
(base) debomita@debutacharjee@Debutacharjee-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-10T04:19:56.611Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] BOTTOM - Text inside frame
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: f495a3bf72622fc0e79b49d4876d48d2
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Frames
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (769ms)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:04
```

After the command has been executed successfully, the text inside the frame - BOTTOM gets printed in the console.

30. WebdriverIO — Drag and Drop

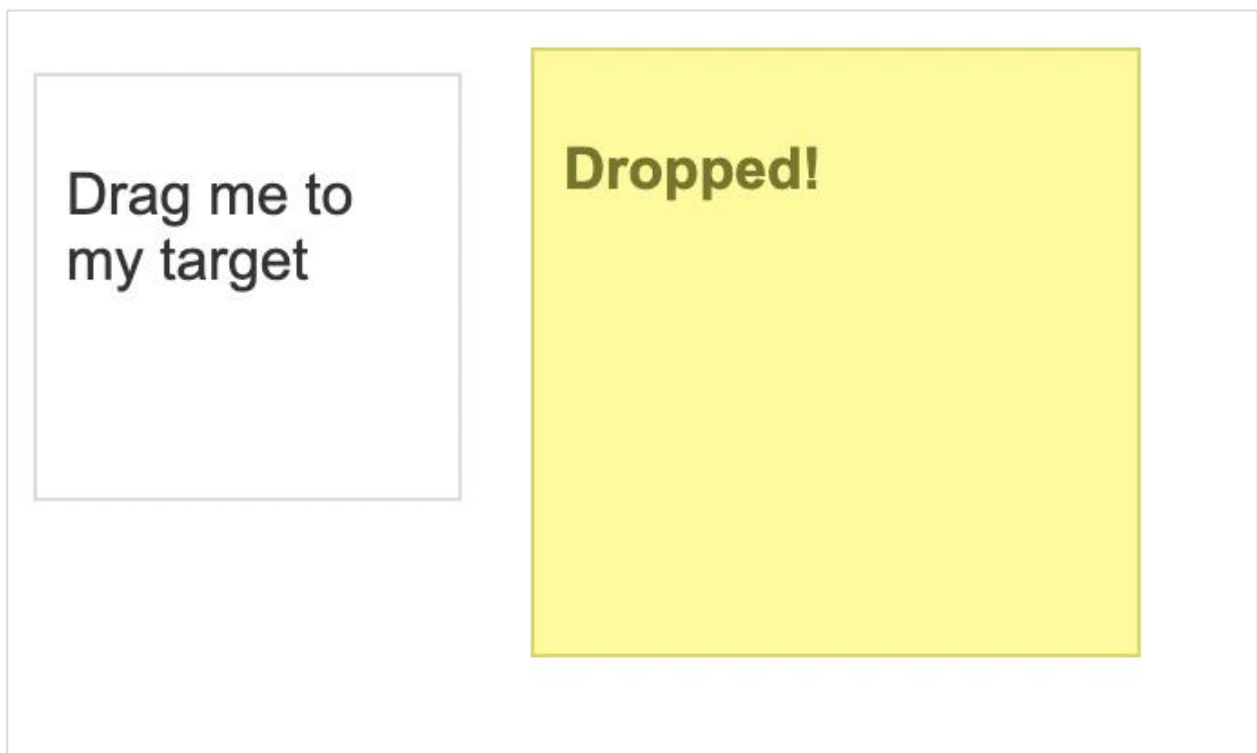
WebdriverIO can perform mouse operations like drag and drop using the `dragAndDrop` method. With this, we execute clicking and holding events on the present object (source), then pass the object to the target element. Finally, release the mouse.

The syntax is as follows:

```
let p = $('#loc')
let t = $('#target')
p.dragAndDrop(t)
```

Here, `p` is the source locator and `t` is the destination locator.

Let us perform the drag and drop functionality for the below elements:



In the above image, the element with the name - Drag me to my target has to be dragged and dropped on the element - Dropped!

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which is as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

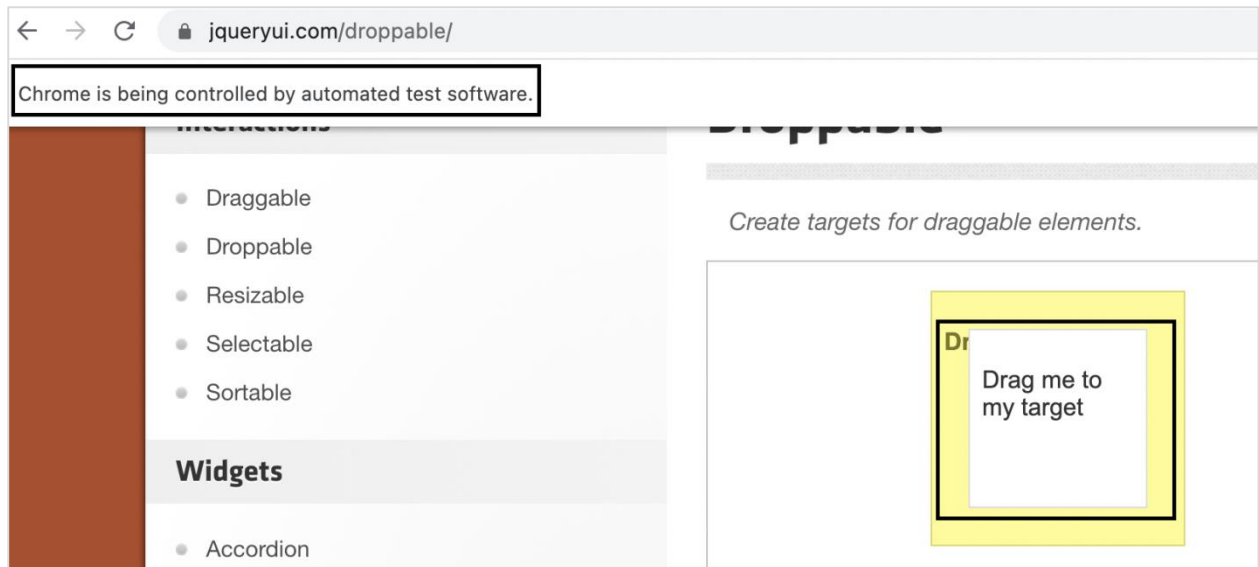
```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Drag and Drop', function(){
    // launch url
    browser.url('https://jqueryui.com/droppable/')
    //maximize browser
    browser.maximizeWindow()
    //switch to frame
    browser.switchToFrame($(".demo-frame"))
    //identify source element
    const src = $('#draggable')
    //identify target element
    const trg = $('#droppable')
    //drag and drop
    src.dragAndDrop(trg)
  });
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:



After execution, the element with the name - Drag me to my target has been dragged and dropped on the element - Dropped!

31. WebdriverIO — Double Click

WebdriverIO can perform mouse operations like double click using the `doubleClick` method. With this, we can perform double clicking on the given element on the webpage.

The syntax is as follows:

```
let p = $('#loc')
p.doubleClick()
```

Let us perform the double click on the below element:



Here, it is seen that on double clicking the Double-Click me To See Alert button, an alert box gets generated.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Double Click', function(){
```

```

    // launch url
    browser.url('http://only-testing-
blog.blogspot.com/2014/09/selectable.html')
    //identify element then double click
    $("button").doubleClick()
    //get Alert Text
    console.log(browser.getAlertText() + ' - Alert Text')
    //accept Alert
    browser.acceptAlert()
  });
});

```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) debomitabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-11T02:55:38.465Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] You double clicked me.. Thank You.. - Alert Text
[0-0] PASSED in chrome - /test/specs/testcase1.js
"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 802f7ca4c4dee587d96b873050173e2c
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Double Click
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (1.9s)
Spec Files:      1 passed, 1 total (100% completed) in 00:00:04

```

After the command has been executed successfully, the Alert text is generated on double-click - You double clicked me.. Thank You.. gets printed in the console.

32. WebdriverIO — Cookies

We can handle cookies using WebdriverIO. A cookie helps to identify a user. It is an efficient technique to pass information from one site session to another or in between sessions of two connected websites.

Methods for Cookies

We can add, delete and obtain a cookie with WebdriverIO using the following methods:

browser.setCookies

This is used to set a single cookie or multiple cookies for the present page. To set a cookie for a page, we have to first launch and stay on that page.

The syntax is as follows:

```
browser.setCookies({cookie, cookie.name, cookie.value, cookie.path,  
cookie.domain, cookie.secure, cookie.httpOnly, cookie.expiry} )
```

Here, cookie is the cookie object or object array and can contain the following values:

- **cookie.name**: It is an optional parameter and refers to the cookie name.
- **cookie.value**: It is an optional parameter and refers to the cookie value.
- **cookie.path**: It is an optional parameter and refers to the cookie path. The default value is / (if it is not added while adding a cookie).
- **cookie.domain**: It is an optional parameter and refers to the cookie domain. The default value is the present browsing context's active document's URL domain (if it is not added while adding a cookie).
- **cookie.secure**: It is an optional parameter to check if the cookie is secured. The default value is false (if it is not added while adding a cookie).
- **cookie.httpOnly**: It is an optional parameter to check if the cookie is of type HTTP. The default value is false (if it is not added while adding a cookie).
- **cookie.expiry**.

browser.getCookies

This is used to get a cookie from the existing page. If the cookie name is provided as a parameter to this method, then that particular cookie shall be obtained. Else, all the cookies from the present page shall be obtained.

The syntax is as follows:

```
//to get a specific cookie  
browser.getCookies(['Topic'])
```

Or,

```
//to get all cookies
browser.getCookies()
```

browser.deleteCookies

This is used to delete a cookie from the existing page. If the cookie name is provided as a parameter to this method, then that particular cookie shall be deleted. Else, all the cookies from the present page shall be deleted.

The syntax is as follows:

```
//to delete a specific cookie
browser.deleteCookies(['Topic'])
```

Or,

```
//to delete all cookies
browser.deleteCookies()
```

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Cookies', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/index.htm')
    //set cookies
    browser.setCookies([
      {name: 'topic1', value: 'WebdriverIO'},
```

```
        {name: 'topic2', value: 'Selenium'}
    ])
    //get a particular cookie
    const t = browser.getCookies(['topic1'])
    console.log(t);
    //get all cookies
    const a = browser.getCookies()
    console.log(a);
    //delete a cookie with name topic2
    browser.deleteCookies(['topic2'])
    d = browser.getCookies()
    console.log(d)
    //delete all cookies
    browser.deleteCookies()
    m = browser.getCookies()
    console.log(m)
});
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:


```
(base) debomita@Debomita-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-11T23:24:55.646Z

[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] [
[0-0]   {
[0-0]     domain: 'www.tutorialspoint.com',
[0-0]     httpOnly: false,
[0-0]     name: 'topic1',
[0-0]     path: '/',
[0-0]     secure: true,
[0-0]     value: 'WebdriverIO'
[0-0]   }
[0-0] ]
[0-0] [
[0-0]   {
[0-0]     domain: 'www.tutorialspoint.com',
[0-0]     httpOnly: false,
[0-0]     name: 'topic2',
[0-0]     path: '/',
[0-0]     secure: true,
[0-0]     value: 'Selenium'
[0-0]   },
[0-0]   {
[0-0]     domain: 'www.tutorialspoint.com',
[0-0]     httpOnly: false,
[0-0]     name: 'topic1',
[0-0]     path: '/',
[0-0]     secure: true,
[0-0]     value: 'WebdriverIO'
[0-0]   },
[0-0] ]
```

After the command has been executed successfully, first the cookie details having the name as topic1 get printed in the console. Then, both the cookie details having names as topic1 and topic2 get displayed.

The following screen will appear on your computer:

```

-0] [
-0]   {
-0]     domain: 'www.tutorialspoint.com',
-0]     httpOnly: false,
-0]     name: 'topic1',
-0]     path: '/',
-0]     secure: true,
-0]     value: 'WebdriverIO'
-0]   },
-0]   {
-0]     domain: '.tutorialspoint.com',
-0]     expiry: 1623453960,
-0]     httpOnly: false,
-0]     name: '_gat_gtag_UA_232293_6',
-0]     path: '/',
-0]     secure: false,
-0]     value: '1'
-0]   },
-0]   {
-0]     domain: '.tutorialspoint.com',
-0]     expiry: 1623540300,
-0]     httpOnly: false,
-0]     name: '_gid',
-0]     path: '/',
-0]     secure: false,
-0]     value: 'GA1.2.1837915299.1623453901'
-0]   },
-0]   {
-0]     domain: '.tutorialspoint.com',
-0]     expiry: 1686525900,
-0]     httpOnly: false,
-0]     name: '_ga',
-0]     path: '/',
-0]     secure: false,
-0]     value: 'GA1.2.2126140080.1623453901'
-0]   }
-0] ]
-0] []
-0] PASSED in chrome - /test/specs/testcase1.js

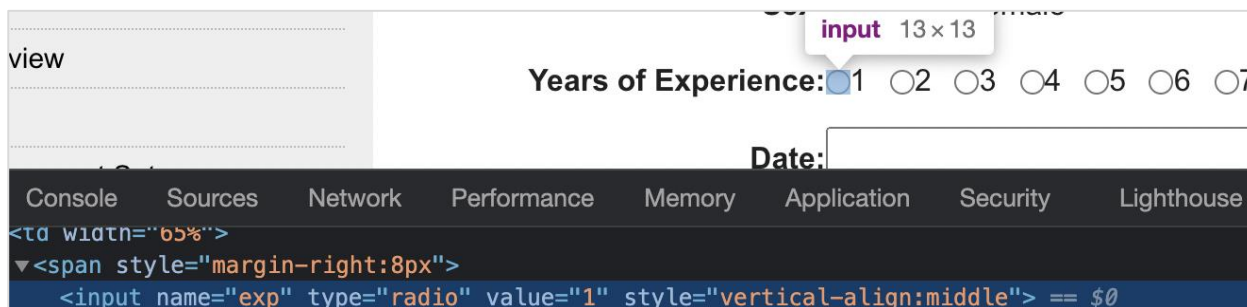
```

Then, we deleted the cookie with the name topic2, so the other cookies got printed in the console. Finally, on deleting all the cookies, an empty array is printed in the console.

33. WebdriverIO — Handling Radio Buttons

We can handle radio buttons in the UI while automating a test using WebdriverIO. The radio buttons are identified in the html code with the tagname as input and type as radio.

The following screen will appear on your computer:



Methods for Radio Buttons

Some methods to work with radio buttons are as follows:

click()

It is used to select a radio button.

The syntax is as follows:

```
const l = $('rad')
l.click()
```

isSelected()

It is used to check if an element of type radio is selected or not. It returns a Boolean value (true if selected, false if not).

The syntax is as follows:

```
const l = $('rad')
l.isSelected()
```

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Radio Button', function(){
    // launch url
    browser.url
    ('https://www.tutorialspoint.com/selenium/selenium_automation_practice.htm')
    //identify radio button with CSS then click
    const p = $("input[value='1']")
    p.click()
    //verify if selected
    console.log(p.isSelected())
  });
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```
(base) debomita@debomita:~/webdriverio % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-09T04:22:39.920Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] true
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 077d95469828ba64ae91892feeb0daed
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Radio Button
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (4.5s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:07
```

After the command has been executed successfully, the boolean value is printed in the console. This is returned by the `isSelected()` function which returns `true` as the radio button is selected in the previous step.

34. WebdriverIO — Chai Assertions on webelements

Chai is an assertion library for nodes. It is mainly used in the BDD and TDD framework. It can easily be integrated with any JavaScript testing framework. The official documentation of Chai is available in the below link:

<https://www.npmjs.com/package/chai>

For installation of Chai and making its entry in the package.json file, run the following command:

```
npm install --save-dev chai
```

The details on the package.json file are discussed in detail in the Chapter titled Package.json.

The following screen will appear on your computer:

```

{} package.json 1 x
{} package.json > {} devDependencies
16   "webdriverio": "^7.7.3"
17   },
18   "devDependencies": {
19     "@wdio/cli": "^7.7.3",
20     "@wdio/local-runner": "^7.7.3",
21     "@wdio/mocha-framework": "^7.7.3",
22     "@wdio/spec-reporter": "^7.7.3",
23     "@wdio/sync": "7.7.3",
24     "chai": "^4.3.4",
25     "chromedriver": "^91.0.0",
26     "wdio-chromedriver-service": "^7.1.0"
27   }
28 }
29
PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE
[0-0] Team @ Tutorials Point
[0-0] PASSED in chrome - /test/specs/testcase1.js
(base) debomita@MacBook-Air:~/webdriverIO % npm install --save-dev chai
npm WARN webdriverIO@1.0.0 No description
npm WARN webdriverIO@1.0.0 No repository field.
npm WARN The package @wdio/sync is included as both a dev and production dependency.

+ chai@4.3.4
added 7 packages from 20 contributors and audited 445 packages in 2.128s

52 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

New major version of npm available! 6.14.13 -> 7.17.0
Change log: https://github.com/npm/cli/releases/tag/v7.17.0
Run npm install -g npm to update!

```

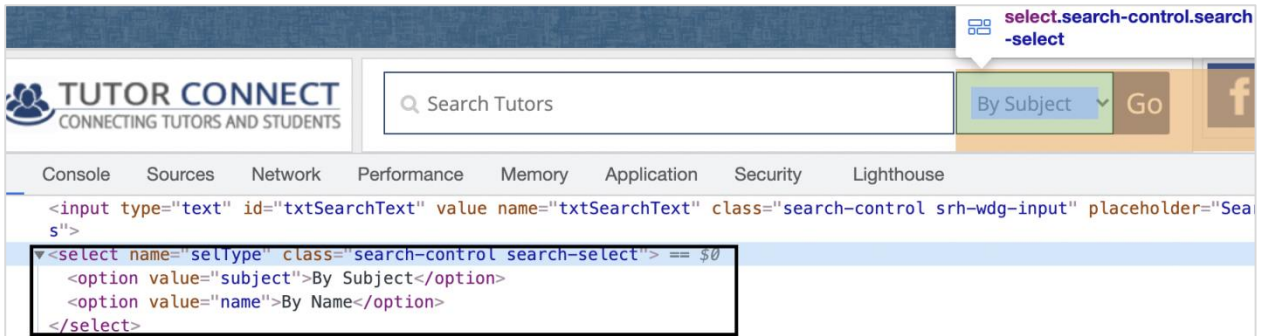
After installation we have to add the below statement to add expected style Chai in our code.

```
require('chai').expect
```

The syntax for Chai assertion is as follows:

```
const c = require('chai').expect
c(p.getValue()).to.equal('subject')
```

Let us implement a Chai assertion and verify if the option selected in the below dropdown is as per the expected result.



The details on how to handle a dropdown is discussed in detail in the Chapter - Handling Dropdowns.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
require('chai').expect
//import chai library
const c = require('chai').expect
describe('Tutorialspoint application', function(){
  //test case
  it('Drodowns with Chai Assertion', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/tutor_connect/index.php')
    //identify dropdown
    const p = $("select[name='selType']")
    //select by index
    p.selectByIndex(1)
    //get option selected
    console.log(p.getValue() + ' - option selected by index')
    //verify option selected with chai assertion
```



```

    c(p.getValue()).to.equal('name')
    //select by visible text
    p.selectByVisibleText('By Subject')
    //get option selected
    console.log(p.getValue() + ' - option selected by visible text')
    //verify option selected with chai assertion
    c(p.getValue()).to.equal('subject')
    //select by value attribute
    p.selectByAttribute('value', 'name')
    //get option selected
    console.log(p.getValue() + ' - option selected by attribute value')
    //verify option selected with chai assertion
    c(p.getValue()).to.equal('name')
  });
});

```

Run the Configuration file - wdio.conf.js file with the command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js

Execution of 1 workers started at 2021-06-12T23:15:04.290Z

[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] name - option selected by index
[0-0] subject - option selected by visible text
[0-0] name - option selected by attribute value
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: fda14ecc0da87e41864e1de2eee24b85
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Drodownds with Chai Assertion
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (4.2s)

Spec Files:   1 passed, 1 total (100% completed) in 00:00:08

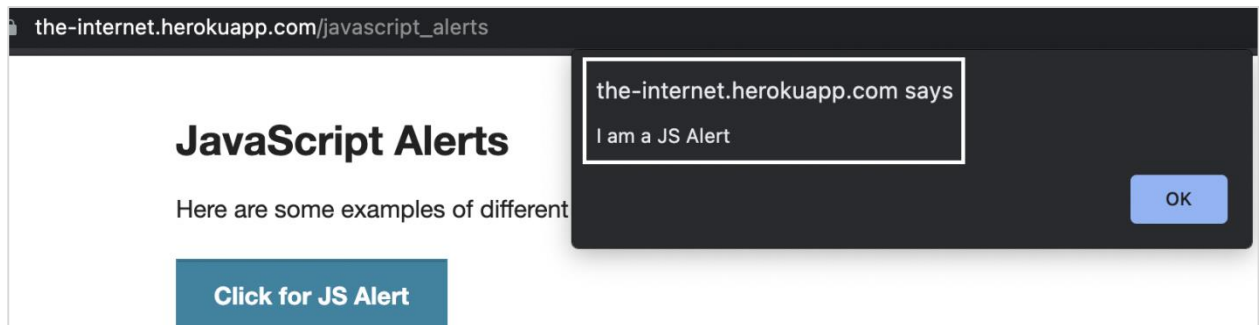
```

After the command has been executed successfully, first the value of the option selected with the option index - name is printed in the console. Then, the value of the option

selected with the option visible text - subject is printed in the console. Finally, the value of the option selected with the option attribute value - name is printed in the console.

Also, we get a PASSED result, pointing to the fact that all the Chai assertions applied on the dropdown have passed.

Let us implement another Chai assertion and verify if the alert text obtained is as per the expected result.



The details on how to handle an alert are discussed in detail in the Chapter titled Alerts.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
//import chai library
const c = require('chai').expect
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Alerts with Chai Assertion', function(){
    // launch url
    browser.url('https://the-internet.herokuapp.com/javascript_alerts')
    //identify element with xpath then click
    $("//*[text()='Click for JS Prompt']").click()
```

```

    //check if Alert is open
    console.log(browser.isAlertOpen())

    //get Alert Text
    console.log(browser.getAlertText() + ' - Alert Text')
    //verify Alert text with Chai assertion
    c(browser.getAlertText()).to.equal("I am a JS prompt")
    //accept Alert
    browser.acceptAlert()
  });
});

```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) debomita@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-12T23:27:57.875Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] true
[0-0] I am a JS prompt - Alert Text
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 3ea9df9720fdb61ed4f6d7989f964d5b
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Alerts with Chai Assertion
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (1.6s)

Spec Files:   1 passed, 1 total (100% completed) in 00:00:04

```

After the command has been executed successfully, at first true is printed in the console as it is returned by the browser.isAlertOpen() method. Then, the Alert text - I am a JS prompt is printed in the console.

Also, we get a PASSED result, pointing to the fact that the Chai assertion applied on the alert text has passed.

35. WebdriverIO — Multiple Windows/Tabs

Multiple windows/tabs can open on clicking a link or a button. WebdriverIO by default has control over the main browser, in order to access the elements on the other tabs, the WebdriverIO control has to be switched from the main browser window to the opened tab.

Methods for Multiple Windows

Some methods to work with multiple windows or tabs are as follows:

browser.getWindowHandles()

This method yields the window handle ids of all the currently opened browser windows in the form of a list. If there are two opened windows, the zero index of the list has the handle id of the parent window and the first index shall point to the window handle of the tab.

The syntax is as follows:

```
var x = browser.getWindowHandles()
```

browser.getWindowHandle()

This method yields the window handle id of the browser which is in focus.

The syntax is as follows:

```
let l = browser.getWindowHandle()
```

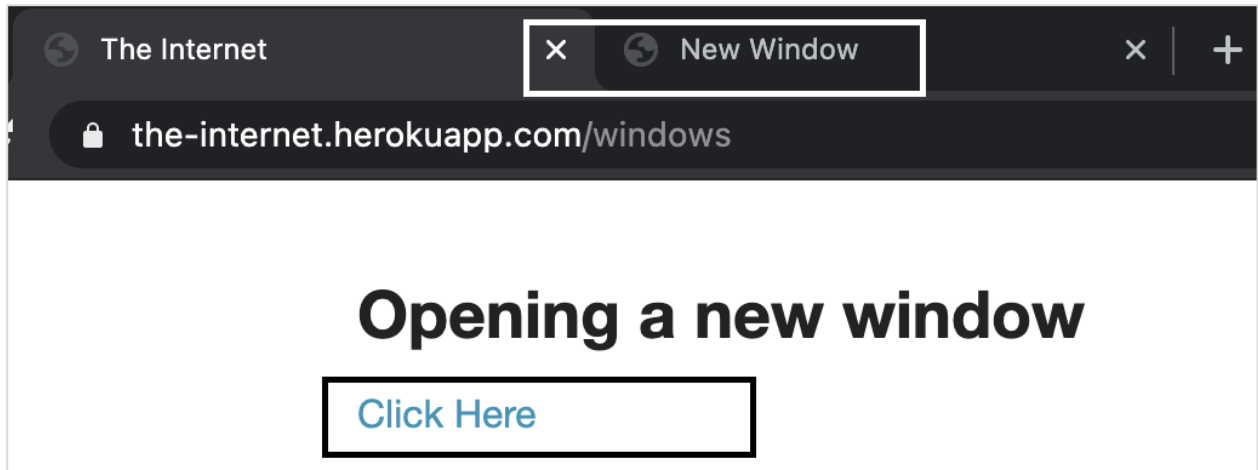
browser.switchToWindow('window handle id')

This method is used to switch focus from the browser window in focus to another opened browser window whose window handle id is passed as a parameter to this method.

The syntax is as follows:

```
browser.switchToWindow(x)
```

In the below image, on clicking the Click Here link, a new tab opens having the browser title as New Window. Let us switch to the new tab and access elements in there.



To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which is as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Tab windows', function(){
    // launch url
    browser.url('https://the-internet.herokuapp.com/windows')
    //identify element then click
    $('=Click Here').click()
    //get all window handle ids in list
    let w = browser.getWindowHandles()
    //switch to tab
    browser.switchToWindow(w[1])
    //get page title of tab
    console.log(browser.getTitle() + ' - Page title of tab')
```

```

    //close child window
    browser.closeWindow()

    //switch to parent window
    browser.switchToWindow(w[0])

    //get page title of parent
    console.log(browser.getTitle() + ' - Page title of parent window')
  });
});

```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) debomita@bhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-10T03:45:27.701Z

[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] New Window - Page title of tab
[0-0] The Internet - Page title of parent window
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 374dc0c5e6469aa89a1ffff0a96722af
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Tab windows
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (2.1s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:06

```

After the command has been executed successfully, the page title of the tab window - New Window gets printed in the console. Then, the page title of the parent window - The Internet gets printed in the console.

36. WebdriverIO — Scrolling Operations

We can perform scrolling operations with the WebdriverIO by using the `scrollIntoView` method. This method does not accept any parameter and can be applied to the browser object or on a particular element.

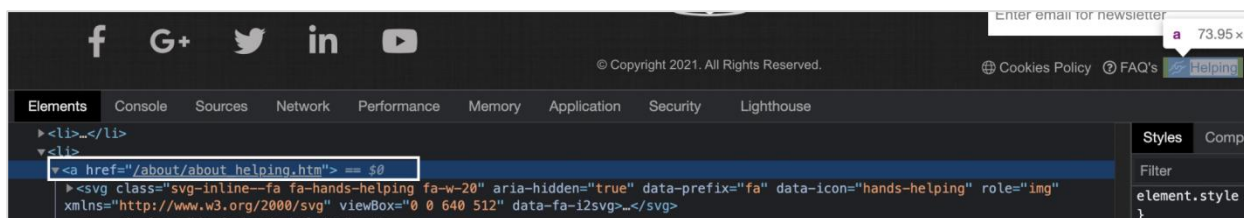
The syntax is as follows:

```
const p = $('#loc')
p.scrollIntoView()
```

Or,

```
browser.scrollIntoView()
```

In the below image, let us scroll to the footer element link - Helping and click on it.



To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO.

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Scroll', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/index.htm')
```

```

    //identify element
    const e = $("=Helping")
    //scroll to element
    e.scrollIntoView()
    e.click()
    //get page title
    console.log(browser.getTitle() + ' - Page time after click')
  });
});

```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) debomita@hattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-11T03:47:43.961Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] Helping Tutorials Point - Tutorialspoint - Page time after click
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: ab7de1ed95fbfcf447d46fde56fd62da
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Scroll
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (6.7s)

Spec Files:   1 passed, 1 total (100% completed) in 00:00:09

```

After the command has been executed successfully, the page title of the page obtained on clicking the link after scrolling - Helping Tutorials Point - Tutorialspoint gets printed in the console.

37. WebdriverIO — Alerts

WebdriverIO is capable of handling Alerts.

Methods for Alerts

Some methods to work with Alerts are listed below:

browser.isAlertopen()

This method is used to verify if there is an alert in the page. It returns true, if the Alert is present, else returns false

The syntax is as follows:

```
browser.isAlertopen()
```

browser.getAlertText()

This method is used to get the text present in the Alert.

The syntax is as follows:

```
browser.getAlertText()
```

browser.acceptAlert()

This method is used to accept an Alert.

The syntax is as follows:

```
browser.acceptAlert()
```

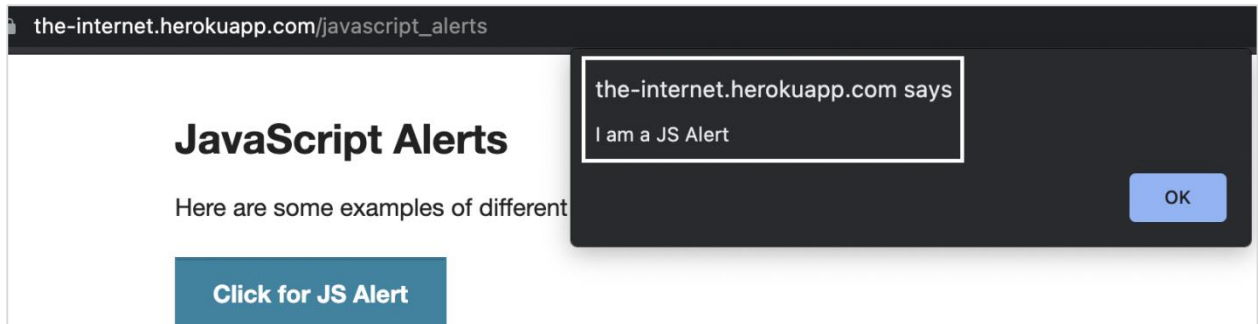
browser.dismissAlert()

This method is used to dismiss an Alert.

The syntax is as follows:

```
browser.dismissAlert()
```

In the below image, on clicking Click for JS Alert, an Alert is displayed. Let us obtain the text on the Alert.



To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Alerts', function(){
    // launch url
    browser.url('https://the-internet.herokuapp.com/javascript_alerts')
    //identify element with xpath then click
    $("//*[text()='Click for JS Prompt']").click()
    //check if Alert is open
    console.log(browser.isAlertOpen())
    //get Alert Text
    console.log(browser.getAlertText() + ' - Alert Text')
    //accept Alert
    browser.acceptAlert()
  });
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```
(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-10T04:45:52.730Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] true
[0-0] I am a JS prompt - Alert Text
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: cccd1fc75d0aeab4ada784ff490305ba
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Alerts
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (1.9s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:05
```

After the command has been executed successfully, the first true is printed in the console as it is returned by the browser.isAlertOpen() method. Then the Alert text - I am a JS prompt is printed in the console.

38. WebdriverIO — Debugging Code

To debug the WebdriverIO code in the Visual Studio Code editor, we have to enable the nightly version of JavaScript Debugger. Debugging is one of the most important steps for identifying the root cause of an error in code.

It also helps to understand the program flow.

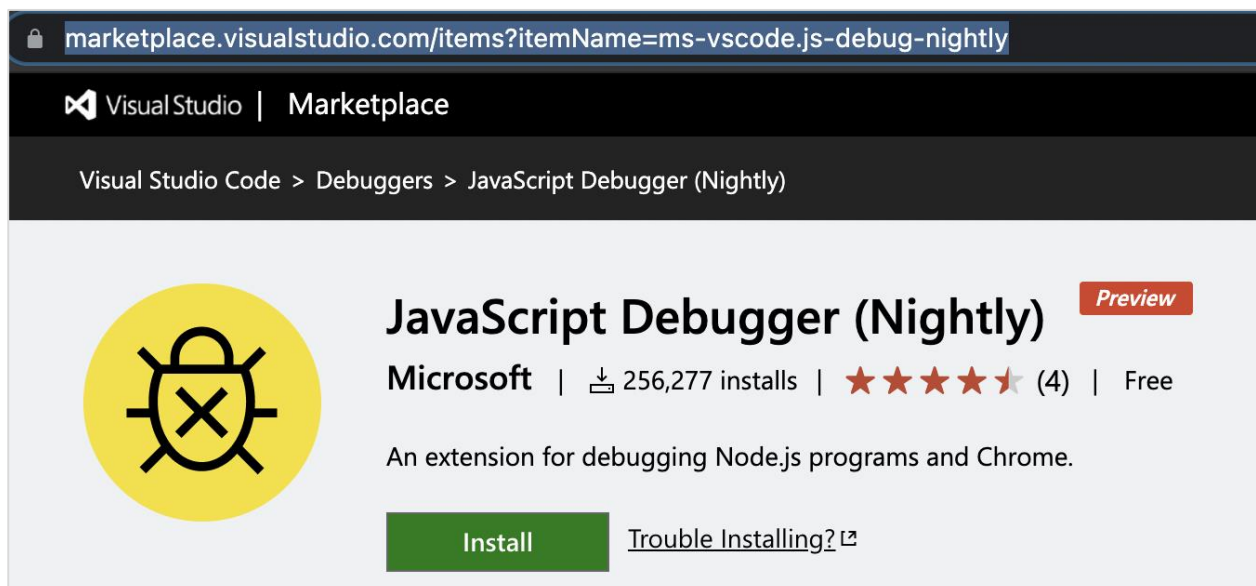
Enable Debugging

The steps to enable debugging are listed below:

Step 1: Navigate to the link below if you are using Windows or Linux operating system:

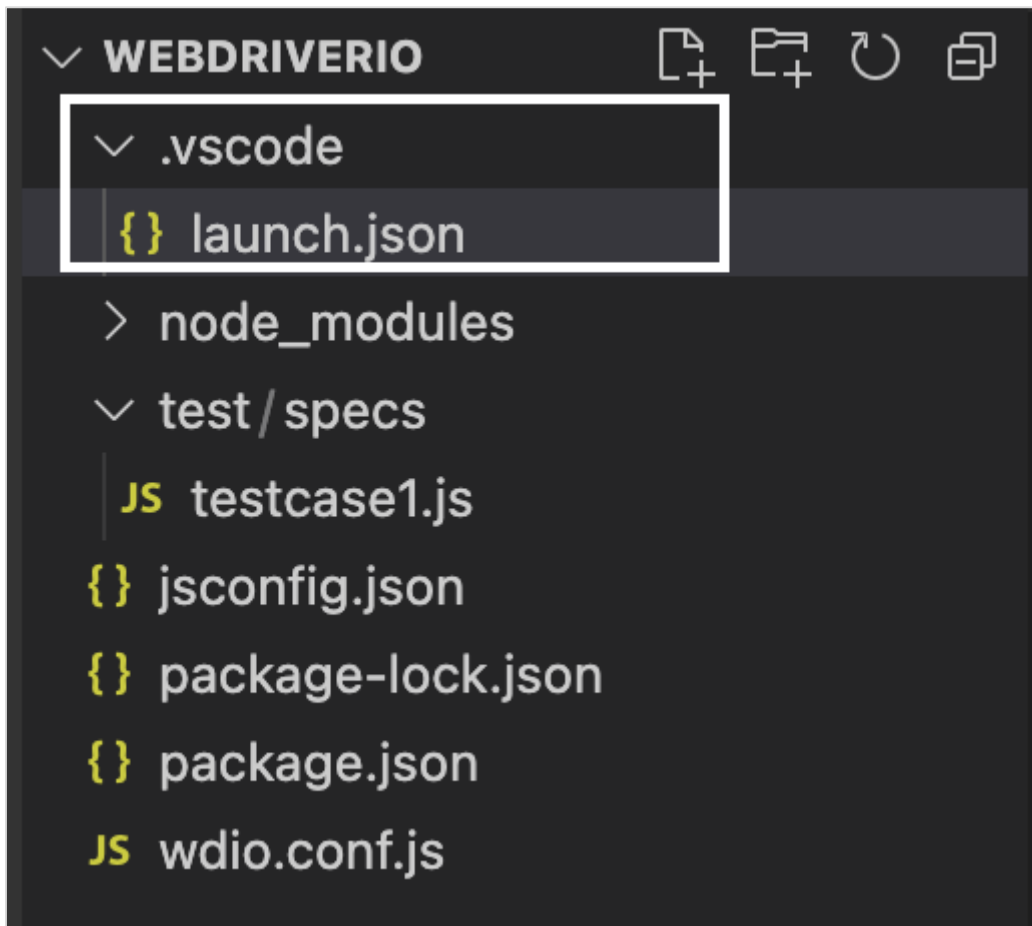
<https://marketplace.visualstudio.com/items?itemName=ms-vscode.js-debug-nightly>

Step 2: Click on Install. The following screen will appear on your computer:



If we are using a Mac operating system, we can skip Steps 1 and 2.

Step 3: Create a folder called the .vscode within the project. Then create a file launch.json within this folder. The following screen will appear on your computer:



Step 4: Add the below code in the launch.json file.

```
{
  "configurations": [
    {
      "name": "Webdriver IO",
      "type": "node",
      "request": "launch",
      "args": ["wdio.conf.js", "--spec", "${file}"],
      "cwd": "${workspaceFolder}",
      "autoAttachChildProcesses": true,
      "program": "${workspaceRoot}/node_modules/@wdio/cli/bin/wdio.js",
      "console": "integratedTerminal",
      "skipFiles": [
        "${workspaceFolder}/node_modules/**/*.js",
        "${workspaceFolder}/lib/**/*.js",
        "<node_internals>/**/*.js"
      ]
    }
  ]
}
```

```

    },
  ]
}

```

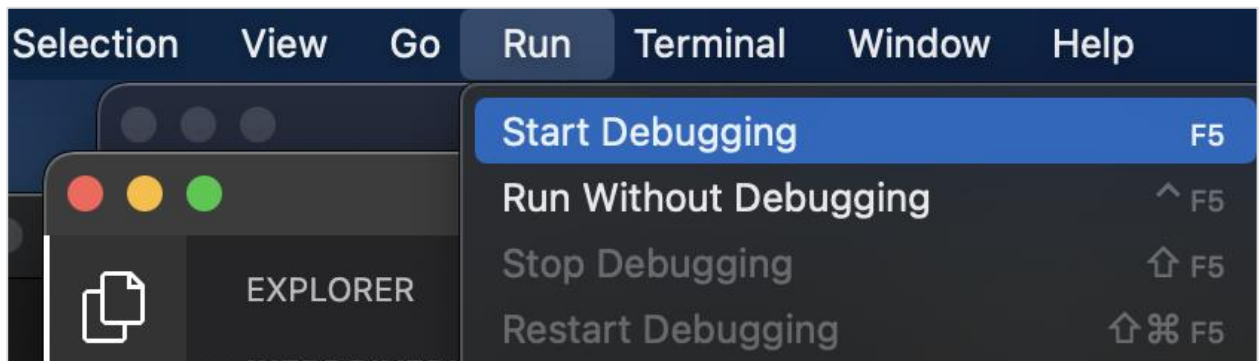
Step 5: Add a breakpoint in the spec file. The following screen will appear on your computer:

```

test > specs > JS testcase1.js > ...
1
2 // test suite name
3 describe('Tutorialspoint application', function(){
4 //test case
5 it('Happy Flow', function(){
6 // launch url
7 browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
8 //identify element with link text then click
9 $("=Team").click()
10 //verify URL of next page with assertion
11 expect(browser).toHaveUrlContaining('team')
12 });
13 });

```

Step 6: Go to the Run menu and select the option Start Debugging. The following screen will appear on your computer:



Step 7: The execution shall get triggered in Debugger mode, with an orange band at the bottom. Debugger attached message should be reflected in the Terminal console. Also, the execution shall halt at the breakpoint. We have to manually resume it again.

The following screen will appear on your computer:

The screenshot displays the Visual Studio Code editor with a WebdriverIO test suite. The test suite is paused at a breakpoint on line 9, which is the `click()` method. The interface includes a left sidebar with variables, watch, call stack, and breakpoints. The main editor shows the test code. The bottom panel shows the terminal with the message "Debugger attached." and the status bar at the bottom indicates "Ln 9, Col 12".

```

test > specs > JS testcase1.js > describe('Tutorialspoint application') callback > it('Happy Flow') callback
1
2 // test suite name
3 describe('Tutorialspoint application', function(){
4   //test case
5   it('Happy Flow', function(){
6     // launch url
7     browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
8     //identify element with link text then click
9     $('=Team').click()
10    //verify URL of next page with assertion
11    expect(browser).toHaveUrlContaining('team')
12  });
13 });
14
15

```

CALL STACK

- Webdriver IO: ... RUNNING
- r. PAUSED ON BREAKPOINT
- <anonymous> test/spec... Show 2 More: Skipped by skipPromise.then
- wrapCommandFn node_...
- <anonymous> test/spec... Show 2 More: Skipped by skipAsync function

LOADED SCRIPTS

BREAKPOINTS

- Caught Exceptions
- Uncaught Exceptions
- testcase1.js test/specs 9

TERMINAL

```

"/Applications/Visual Studio Code.app/Contents/Resources/app/extensions/ms-vscode.js-debug/src/bootloader.bundle.js" --inspect-publish-uid=http://VSCODE_INSPECTOR_OPTIONS={"inspectorIpc":"/var/folders/vv/lyn164sx11bchyr6fxn256jw000gn/T/node-cdp.1591-1.sock","deferredMode":false,"waitForDebugger":"","execPath":"/usr/local/bin/node","onlyEntrypoint":false,"autoAttachMode":"always","fileCallback":"/var/folders/vv/lyn164sx11bchyr6fxn256jw000gn/T/node-debug-callback-0104d6a4206b0a57"} /usr/local/bin/node . /node_modules/@wdio/cli/bin/wdio.js wdio.conf.js --spec /Users/debomtabhattacharjee/webdriverIO/test/specs/testcase1.js
Debugger attached.
Execution of 1 workers started at 2021-06-08T04:52:25.274Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js

```

Ln 9, Col 12 Spaces: 4 UTF-8 LF JavaScript

39. WebdriverIO — Capturing Screenshots

We can capture screenshots while working on automation tests developed in WebdriverIO using the `saveScreenshot` method. A screenshot is generally captured if we encounter an application error. An Assertion has failed, and so on.

The syntax for capturing screenshots is as follows:

```
browser.saveScreenshot("name along with path to store screenshot")
```

Here, the name along with the path where the screenshot is to be saved is passed as a parameter to the method. In the WebdriverIO, we don't have the option to capture a screenshot for a particular element.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint application', function(){
  //test case
  it('Screenshot', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/index.htm')
    //identify element then enter text
    const e = $("#gsc-i-id1")
    e.setValue('WebdriverIO')
    //capture screenshot of page
    browser.saveScreenshot("screenshot.png")
  });
});
```

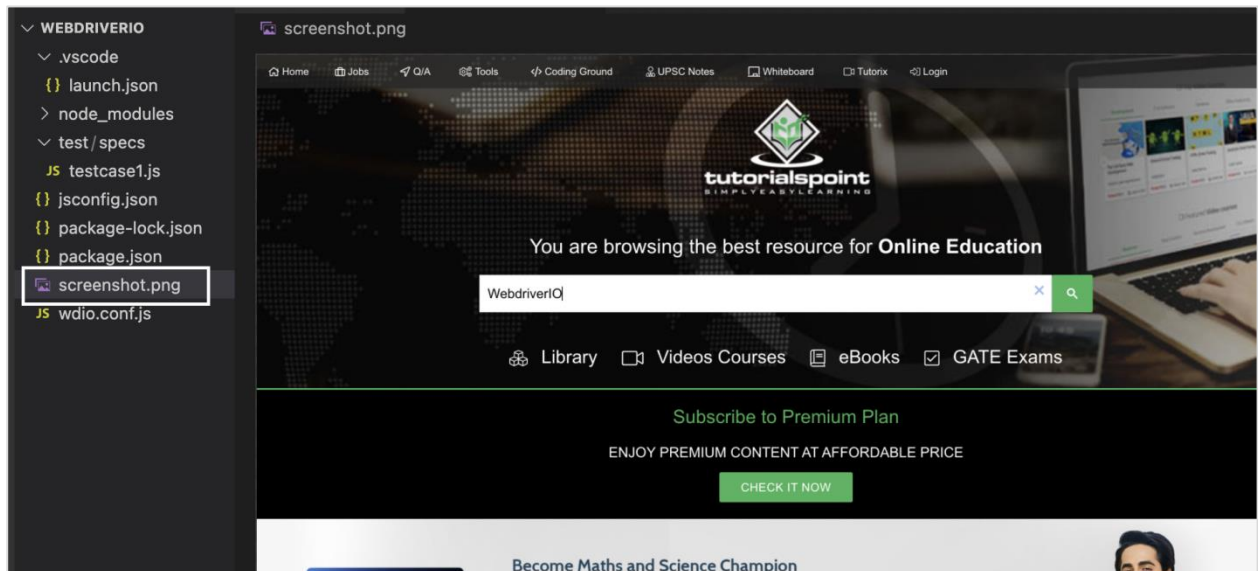


```
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation. The following screen will appear on your computer:



After the command has been executed successfully, a file named screenshot.png gets generated within the project folder. It contains the captured screenshot of the page.

40. WebdriverIO — JavaScript Executor

Inside the WebdriverIO, the JavaScript Executor is bundled and called executeScript. The JavaScript Executor is capable of performing all the tasks on a page whenever normal WebdriverIO methods are not working as expected.

The syntax for the Javascript executor is as follows:

```
browser.executeScript("JavaScript command")
```

Actions with Javascript Executor

Some actions performed with JavaScript Executor are as follows:

To enter a text - AB into an edit box having id as txt, use the command given below:

```
browser.executeScript("document.getElementById('txt').value='AB'")
```

To click a link, use the command given below:

```
browser.executeScript("document.querySelector('.lnk').click()")
```

The command given below is used for refreshing windows:

```
browser.executeScript("history.go(0)")  
var t = js.executeScript("return  
document.getElementById('b1n').innerHTML").toString()
```

The command to scroll down a page by 350 pixels is as follows:

```
browser.executeScript("window.scrollTo(0,350)")  
browser.executeScript("window.scrollTo(0, document.body.scrollHeight)")
```

The command given below is used to scroll down upto an element having class as tcl.

```
browser.executeScript("document.querySelector('.tcl').scrollIntoView()")  
browser.executeScript("window.history.back()")
```

Following command is used to go forward in browser history:

```
browser.executeScript("window.history.forward()")  
browser.executeScript("return document.title")
```

41. WebdriverIO — Waits

The `waitUntil` method in WebdriverIO is a standard method to wait for an action /element on the page. It waits for a criterion to be met (a true value).

For example, we often wait for a text to appear on the page.

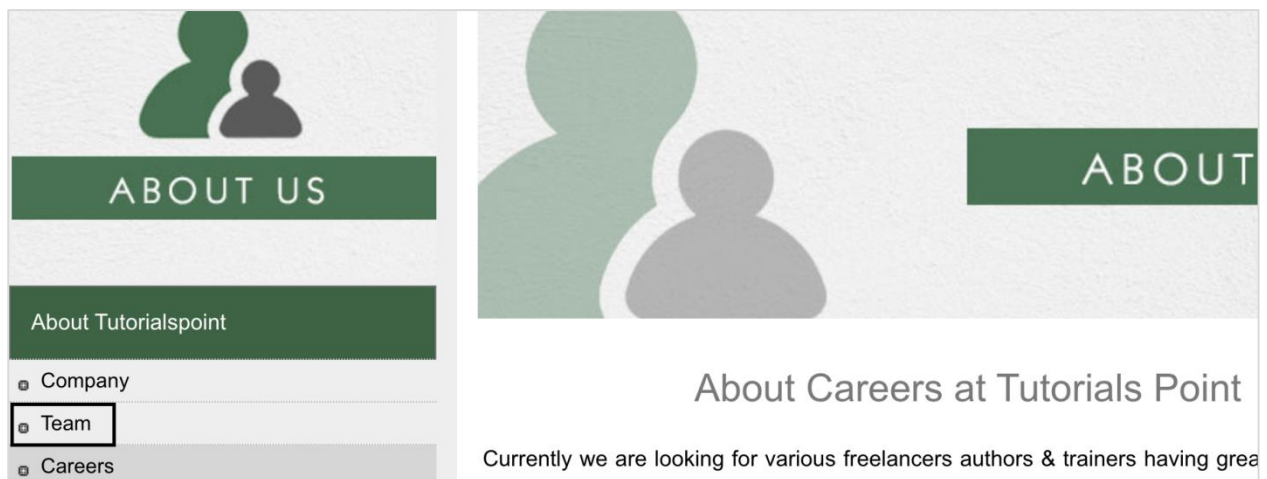
The syntax for `waitUntil` method is as follows:

```
browser.waitUntil(condition, { timeout, timeoutMsg, interval })
```

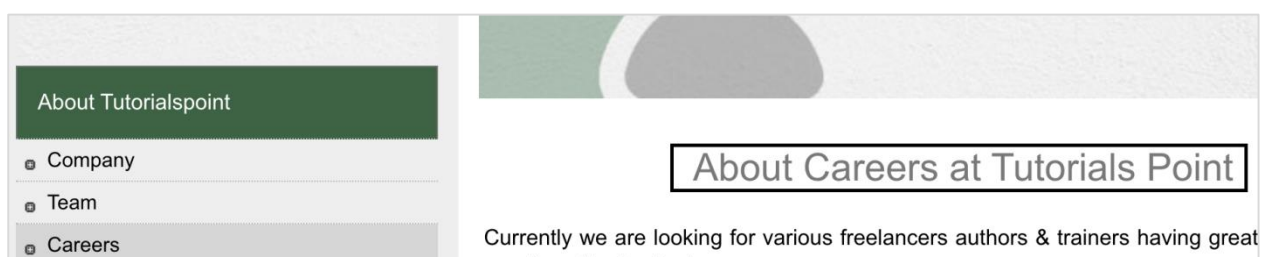
Here,

- `condition` = condition for waiting on.
- The `timeout` is in milliseconds. The default value is 5000 and is an optional parameter.
- The `timeoutMsg` is the error message thrown when there is a timeout and it is an optional parameter.
- The `interval` is the interval in between verification. The default value is 500 and it is also an optional parameter.

In the below image, let us click on the link - Team and wait for the text - Team @ Tutorials Point to appear on the page.



On clicking the link Team, the highlighted message is displayed on the page.



To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows: **Step 1:** Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
// test suite name
describe('Tutorialspoint Application', function(){
  //test case
  it('Waits', function(){
    // launch url
    browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
    //identify then click link - Team
    const p = $('=Team')
    p.click()
    //wait for text
    browser.waitUntil(
      () => $('<h1>').getText() === 'Team @ Tutorials Point',
      {
        timeout: 6000,
        timeoutMsg: 'expected text did not match'
      }
    );
    //identify required text
    const m = $('<h1>')
    console.log(m.getText())
  });
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation. The following screen will appear on your computer:

```
(base) debomitahattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js

Execution of 1 workers started at 2021-06-12T04:53:30.014Z

[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] Team @ Tutorial's Point
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 764bf139a5bcfbfa427661b25af4f7e8
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint Application
[chrome 91.0.4472.77 mac os x #0-0]   ✓ Waits
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (6.2s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:10
```

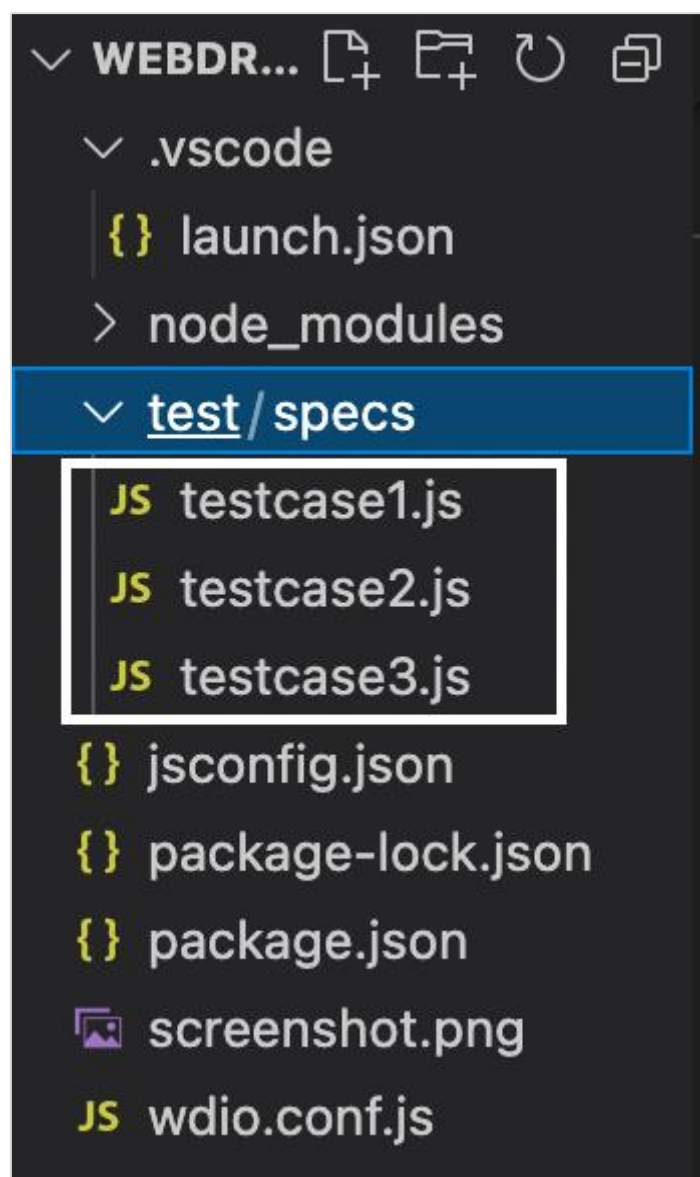
After the command has been executed successfully, the text generated on clicking the Team link - Team @ Tutorial's Point gets printed in the console.

42. WebdriverIO — Running Tests in Parallel

We can run WebdriverIO tests in parallel mode. For this we have to create more than one spec file within the test folder. The numbers of threads in which parallel tests can run are defined by the parameters in the Configuration file - wdio.conf.js file.

The details on how to create a Configuration file are discussed in detail in the Chapter - Wdio.conf.js file and Chapter - Configuration File generation to store WebdriverIO settings.

Let us take a project having three spec files within the test folder. The following screen will appear on your computer:

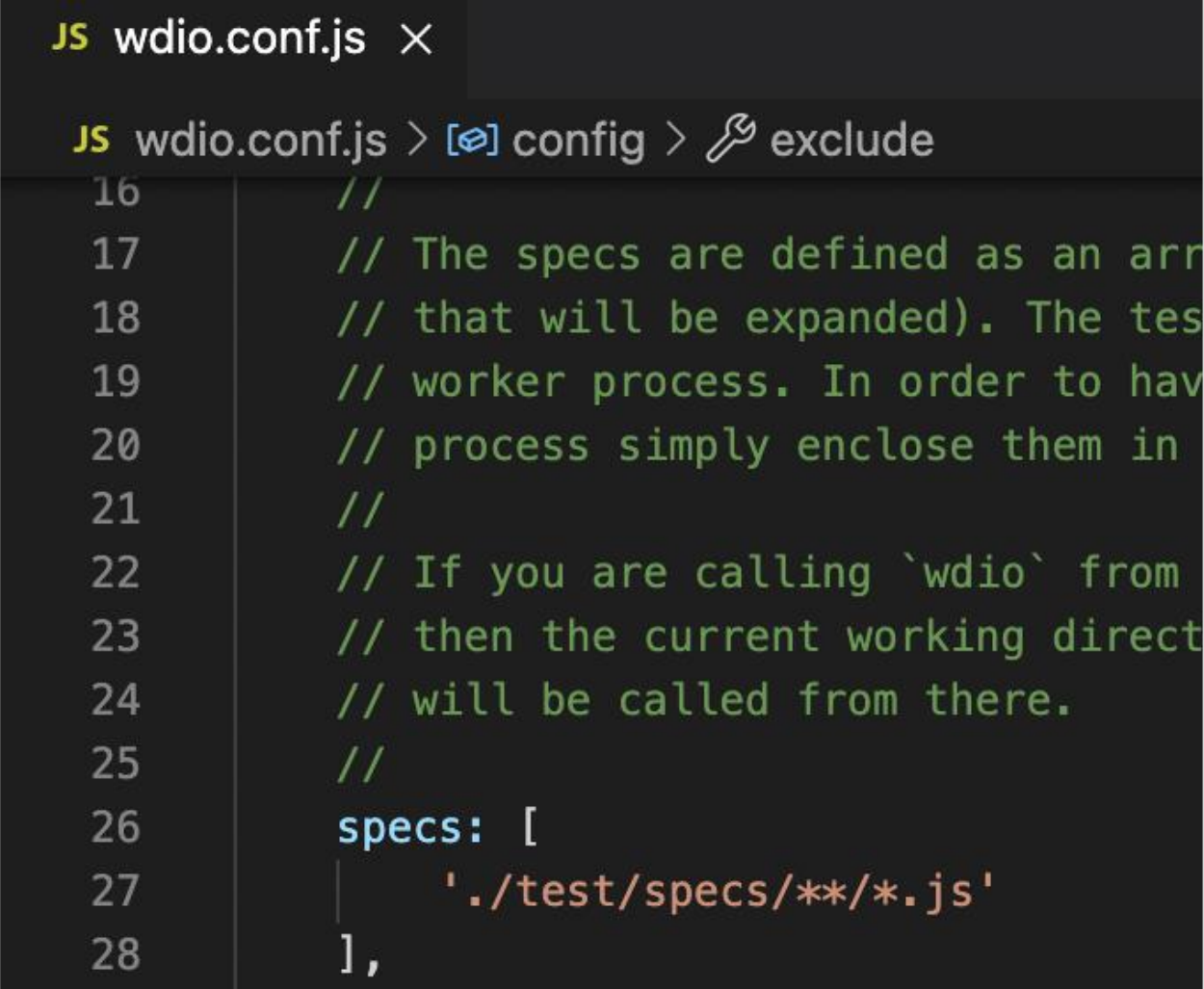


To execute all these files in a parallel mode, we have to first specify `'./test/specs/**/*.js'` under the specs field in the wdio.conf.js file. This means all the

spec files within the test folder would get triggered on running the command given below:

```
npx wdio run wdio.conf.js.
```

The following screen will appear on your computer:



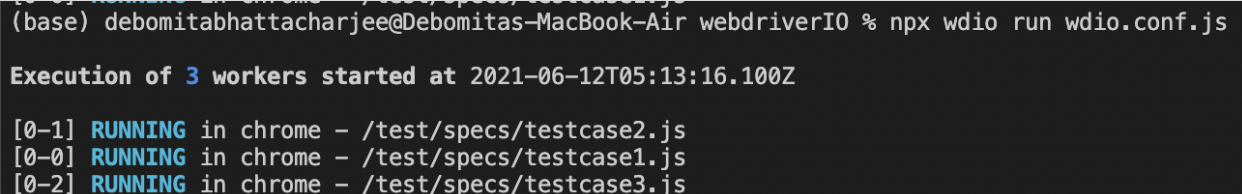
```

JS wdio.conf.js ×
JS wdio.conf.js > [🔍] config > 🔧 exclude
16 //
17 // The specs are defined as an array (normally based on your file globbing
18 // that will be expanded). The test runner will call each file in this
19 // worker process. In order to have a single process simply enclose them in
20 // an array.
21 //
22 // If you are calling `wdio` from a non-repository context you should
23 // then the current working directory will be called from there.
24 //
25 //
26 specs: [
27   './test/specs/**/*.js'
28 ],

```

After the command has been executed successfully, we shall see all the three spec files - testcase1.js, testcase2.js and testcase3.js getting triggered for execution simultaneously.

The following screen will appear on your computer:



```

(base) debomitabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 3 workers started at 2021-06-12T05:13:16.100Z
[0-1] RUNNING in chrome - /test/specs/testcase2.js
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-2] RUNNING in chrome - /test/specs/testcase3.js

```

Also, the maxInstances field in the wdio.conf.js determines the maximum number of threads possible to trigger the parallel execution. By default, the value is set to 10. Here, we have three spec files, so the maxInstances = 10, holds true.

The following screen will appear on your computer:

```

JS wdio.conf.js ×
JS wdio.conf.js > [⚙] config > 🔧 exclude
45 // files and you see maxInstances to
46 // and 30 processes will get spawned.
47 // from the same test should run tests
48 //
49 maxInstances: 10,
50 //
51 // If you have trouble getting all imp
52 // Sauce Labs platform configurator -
53 // https://docs.saucelabs.com/referenc
54 //
55 capabilities: [{
56
57 // maxInstances can get overwrite
58 // grid with only 5 firefox instan
59 // 5 instances get started at a ti
60 maxInstances: 5,
61 //
62 browserName: 'chrome',
63 acceptInsecureCerts: true

```

There is another field called capabilities within the wdio.conf.js file. Within this, we have a parameter called the maxInstances. It determines the number of instances that can be opened simultaneously by the Chrome browser during the parallel run.

Let us set the value 3 for the parameter maxInstances outside the capabilities field and then set the value 2 for the field maxInstances inside the capabilities field. The value set for maxInstances within the capabilities overrides the value set for maxInstances outside the capabilities.

Run the following command:


```
npx wdio run wdio.conf.js
```

After the command has been executed successfully, we shall see two spec files - testcase1.js and testcase2.js getting triggered for execution simultaneously in Chrome. They are initially in RUNNING status.

Once the status of testcase2.js moved to PASSED, the third spec testcase3.js moved to the status of RUNNING. The following screen will appear on your computer:

```
(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 3 workers started at 2021-06-12T18:44:20.549Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-1] RUNNING in chrome - /test/specs/testcase2.js
[0-1] PASSED in chrome - /test/specs/testcase2.js
[0-2] RUNNING in chrome - /test/specs/testcase3.js
[0-0] Team @ Tutorials Point
[0-0] PASSED in chrome - /test/specs/testcase1.js
[0-2] PASSED in chrome - /test/specs/testcase3.js
```

43. WebdriverIO — Data Driven Testing

We can achieve data driven testing with WebdriverIO. Data driven testing is required when we need to execute the same test case multiple times with different combinations of data. Here, we shall see how to use an external JSON file to hold data.

In the WebdriverIO project all the test files are created within the specs folder. The specs folder resides within the test folder. We shall create another folder, say testData within the test folder.

The testData folder shall contain the JSON files which hold the different sets of data in key-value pairs. Also, if we have three test files within the spec folder and we want to have data driven testing for all these files, we need to create three JSON files.

Each of these JSON files should be used dedicatedly for each test file within the spec folder. We shall create a JSON file, say test1.json within the testData folder.

Now, add the below data within this file:

```
[
  {
    "email":"test@gmail.com",
    "password":"12"
  },
  {
    "email":"test12@gmail.com",
    "password":"34"
  }
]
```

The following screen will appear on your computer:

```

EXPLORER
  WEBDRIVERIO
    .vscode
    node_modules
    test
      specs
      JS testcase1.js
      testData
        {} test1.json
        {} jsconfig.json
        {} package-lock.json
        {} package.json
        JS wdio.conf.js
  {} test1.json x
test > testData > {} test1.json > {} 0 > abc email
1  [
2  |
3  |   "email": "test@gmail.com",
4  |   "password": "12"
5  | },
6  | {
7  |   "email": "test12@gmail.com",
8  |   "password": "34"
9  | }
10 | ]
11 ]

```

We shall parse this JSON file and convert it in string format. This is done by adding the below library:

```
const s =require('fs')
```

Then to parse the JSON file, we shall use the `readFileSync` method and pass the relative path of the JSON file file as a parameter to this method. Finally, store this in an object, say `c`. This object shall contain all the data.

```
let c = JSON.parse(s.readFileSync('test/testData/test1.json'))
```

Then, we shall iterate the same test case over the two sets of data with the help of the loop. This loop has to be implemented just before the block and it should pass the data keys as declared in the JSON file.

With the above set of data, we shall validate the login page of the LinkedIn application. On clicking on the Sign in button after entering an email and password of less than 6 characters, an error message - The password you provided must have at least 6 characters should be thrown.

The following screen will appear on your computer:

LinkedIn

Sign in

Stay updated on your professional world

Email or Phone
test123@gmail.com

Password
..... [show](#)

The password you provided must have at least 6 characters.

[Forgot password?](#)

Sign in

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
//import chai library
const c = require('chai').expect
//library for parsing JSON file
const s =require('fs')
let h = JSON.parse(s.readFileSync('test/testData/test1.json'))
// test suite name
describe('Tutorialspoint application', function(){
  //iterate the test case
  h.forEach( ({email,password}) =>{
    //test case
    it('Data Driven testing', function(){
      // launch url
      browser.url('https://www.linkedin.com/login')
      //identify the email field then enter key - email
      $("#username").setValue(email)
      //identify password field then enter key - password
      $("#password").setValue(password)
      //identify Sign in button then click
      $("button[type='submit']").click()
      //verify error message
      const e = $('#error-for-password')
      console.log(e.getText() + ' - Error Text')
      //verify Alert text with Chai assertion
      c(e.getText()).to.equal("The password you provided must have at least 6
characters.")
    });
  });
});
```

Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

because element wasn't found
(base) debomita@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 1 workers started at 2021-06-13T04:38:07.827Z

[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] The password you provided must have at least 6 characters. - Error Text
[0-0] The password you provided must have at least 6 characters. - Error Text
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: 45cea181d72bfb061fc32825e0a535ad
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0] ✓ Data Driven testing
[chrome 91.0.4472.77 mac os x #0-0] ✓ Data Driven testing
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 2 passing (5s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:08

```

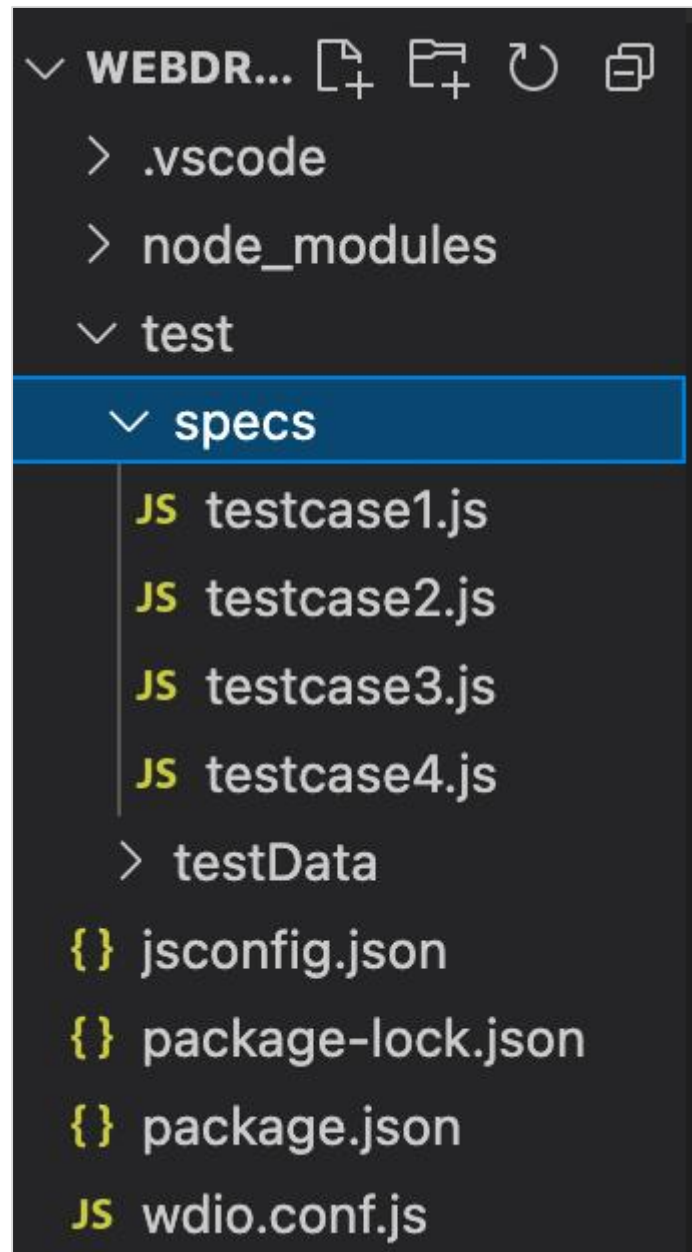
After the command has been executed successfully, the error text - The password you provided must have at least 6 characters gets printed in the console twice.

Also, it shows the message 2 passing as the same test case defined in one block has executed two times with two different sets of data.

44. WebdriverIO — Running Tests from command-line parameters

We can control running tests using the command-line parameters. Let us take a scenario, where we have four test files within the spec folder in the WebdriverIO project.

The following screen will appear on your computer:



Suppose we want to trigger only the files `testcase1.js` and `testcase2.js` using the command-line parameters. To do this we have to add a parameter called `suites` in the Configuration file `wdio.conf.js` file.

The details on how to create a Configuration file are discussed in detail in the Chapter titled `Wdio.conf.js` file and Chapter titled `Configuration File generation`.

Let us consider that the files `testcase1.js` and `testcase2.js` belong to a suite called the `group1` and the files `testcase3.js` and `testcase4.js` belong to a suite called the `group2`. We need to add this information to the `wdio.conf.js` file under the `suite` parameter as given below.

```
suites: {  
  group1: ['test/specs/testcase1.js', 'test/specs/testcase2.js'],  
  group2: ['test/specs/testcase3.js', 'test/specs/testcase4.js']  
},
```

The following screen will appear on your computer:

```
JS wdio.conf.js ●  
JS wdio.conf.js > [🔗] config  
24 // will be called from there.  
25 //  
26 suites: {  
27   group1: ['test/specs/testcase1.js', 'test/specs/testcase2.js'],  
28   group2: ['test/specs/testcase3.js', 'test/specs/testcase4.js']  
29 },  
30  
31 specs: [  
32   './test/specs/**/*.js'  
33 ],
```

To trigger the test files `testcase1.js` and `testcase2.js` belonging to `group1`, we have to run the command given below:

```
npx wdio run wdio.conf.js --suite group1
```

The following screen will appear on your computer:


```
(base) debomita@bhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js --suite group1

Execution of 2 workers started at 2021-06-13T05:15:31.368Z

[0-1] RUNNING in chrome - /test/specs/testcase2.js
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-1] Page title after click: The Internet
[0-1] PASSED in chrome - /test/specs/testcase2.js
[0-0] The password you provided must have at least 6 characters. - Error Text
[0-0] The password you provided must have at least 6 characters. - Error Text
[0-0] PASSED in chrome - /test/specs/testcase1.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-1] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-1] Session ID: 19da714f4d0433fb3202927d746dbd6a
[chrome 91.0.4472.77 mac os x #0-1]
[chrome 91.0.4472.77 mac os x #0-1] » /test/specs/testcase2.js
[chrome 91.0.4472.77 mac os x #0-1] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-1] ✓ Identify element with Id
[chrome 91.0.4472.77 mac os x #0-1]
[chrome 91.0.4472.77 mac os x #0-1] 1 passing (2.6s)
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: cbaa95af8d56e3a33a96950b2006c091
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase1.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0] ✓ Data Driven testing
[chrome 91.0.4472.77 mac os x #0-0] ✓ Data Driven testing
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 2 passing (5.5s)

Spec Files:      2 passed, 2 total (100% completed) in 00:00:09
```

After the command has been executed successfully, we see only the two test files `testcase1.js` and `testcase2.js` under the `specs` folder have been triggered for execution.

Suppose we want to trigger only the file `testcase3.js` using the command-line parameters. To trigger only the test file `testcase3.js`, we have to run the following command:

```
npx wdio run wdio.conf.js --spec test/specs/testcase3.js
```

The following screen will appear on your computer:

```
(base) debomita@bhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js --spec test/specs/testcase3.js

Execution of 1 workers started at 2021-06-13T05:20:57.653Z

[0-0] RUNNING in chrome - /test/specs/testcase3.js
[0-0] About careers at Tutorialspoint - is the text.
[0-0] PASSED in chrome - /test/specs/testcase3.js

"spec" Reporter:
-----
[chrome 91.0.4472.77 mac os x #0-0] Running: chrome (v91.0.4472.77) on mac os x
[chrome 91.0.4472.77 mac os x #0-0] Session ID: c52378e458c76f6b42b17b7c6450e4db
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] » /test/specs/testcase3.js
[chrome 91.0.4472.77 mac os x #0-0] Tutorialspoint application
[chrome 91.0.4472.77 mac os x #0-0] ✓ Identify element with Tagname
[chrome 91.0.4472.77 mac os x #0-0]
[chrome 91.0.4472.77 mac os x #0-0] 1 passing (3.8s)

Spec Files:      1 passed, 1 total (100% completed) in 00:00:07
```

After the command has been executed successfully, we see only the test file `testcase3.js` under the `specs` folder has been triggered for execution.

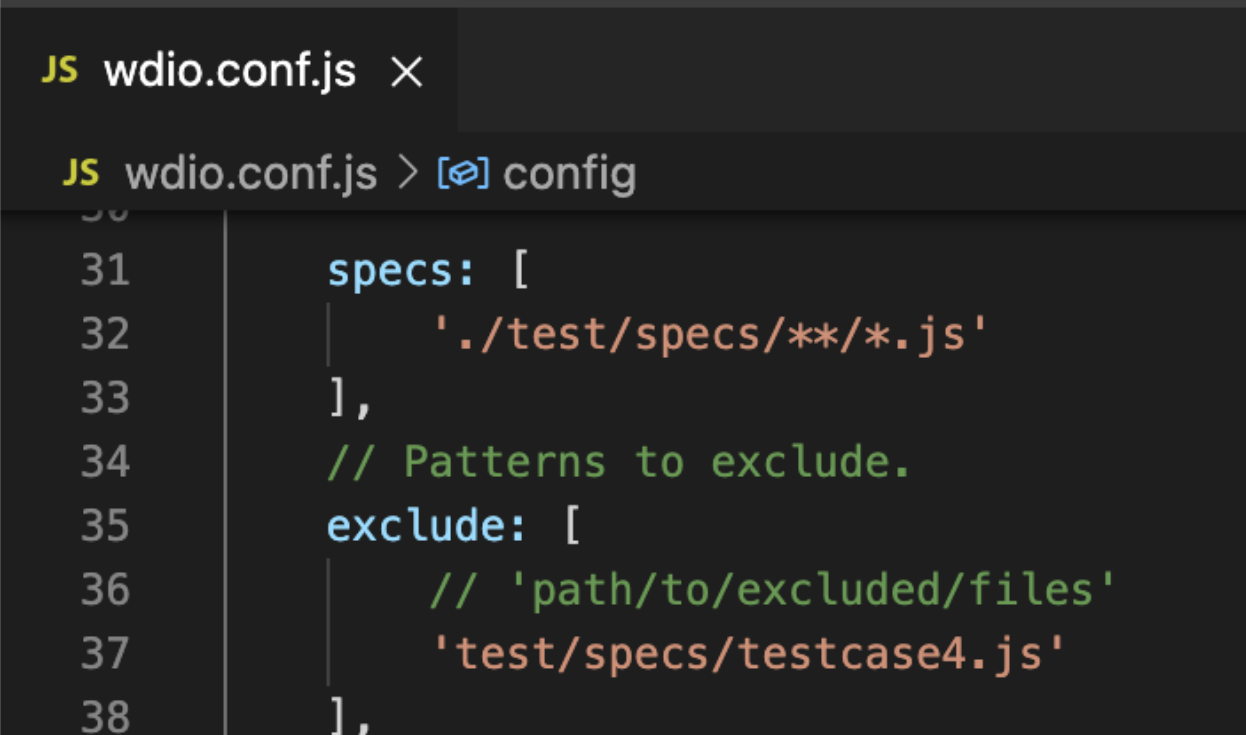
Besides, if we want to trigger multiple test files testcase3.js and testcase4.js, the command should be as follows:

```
npx wdio run wdio.conf.js --spec test/specs/testcase3.js,  
test/specs/testcase4.js
```

Suppose we want to exclude only the file testcase4.js from execution. To do this we have to add a relative path of the file that we want to exclude under the exclude parameter in the Configuration file wdio.conf.js file as given below.

```
exclude: [  
  // 'path/to/excluded/files'  
  'test/specs/testcase4.js'  
],
```

The following screen will appear on your computer:



```
JS wdio.conf.js ×  
JS wdio.conf.js > [🔗] config  
31 specs: [  
32 |   './test/specs/**/*.js'  
33 | ],  
34 // Patterns to exclude.  
35 exclude: [  
36 |   // 'path/to/excluded/files'  
37 |   'test/specs/testcase4.js'  
38 | ],
```

Then, we have to run the below command:

```
npx wdio run wdio.conf.js
```

The following screen will appear on your computer:

```
(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npx wdio run wdio.conf.js
Execution of 3 workers started at 2021-06-13T05:31:19.170Z
[0-1] RUNNING in chrome - /test/specs/testcase2.js
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-2] RUNNING in chrome - /test/specs/testcase3.js
[0-1] Page title after click: The Internet
[0-1] PASSED in chrome - /test/specs/testcase2.js
[0-2] About Careers at Tutorials Point - is the text.
[0-2] PASSED in chrome - /test/specs/testcase3.js
[0-0] The password you provided must have at least 6 characters. - Error Text
[0-0] The password you provided must have at least 6 characters. - Error Text
[0-0] PASSED in chrome - /test/specs/testcase1.js
```

After the command has been executed successfully, we see the test file `testcase4.js` under the `specs` folder has been excluded from execution.

45. WebdriverIO — Execute Tests with Mocha Options

A test file within the specs folder consists of the describe and it blocks. A describe block refers to the test suite and the it block refers to the test case. A describe block can have multiple blocks.

The details on how to create describe and it blocks are discussed in detail in the Chapter titled Happy path flow with Webdriverio.

To verify if a new build obtained from the development team is a healthy one, we need not execute all the test cases within a suite. A few test cases are identified for smoke/sanity testing and they are executed once we have a new build.

We can use the Mocha option called Grep to group test cases and run them together. For this, we have to add a keyword, say Smoke within the it description. Then at the runtime, we can instruct the WebdriverIO test to only trigger the it blocks which have Smoke in its description.

Let us take a test file having four it blocks. Out of the four it blocks, there are two it blocks having the keyword Smoke in description.

To begin, follow Steps 1 to 5 from the Chapter titled Happy path flow with WebdriverIO which are as follows:

Step 1: Install NodeJS. The details on how to perform this installation are given in detail in the Chapter titled Getting Started with NodeJS.

Step 2: Install NPM. The details on how to perform this installation are given in detail in the Chapter titled Installation of NPM.

Step 3: Install VS Code. The details on how to perform this installation are given in detail in the Chapter titled VS Code Installation.

Step 4: Create the Configuration file. The details on how to perform this installation are given in detail in the Chapter titled Configuration File generation.

Step 5: Create a spec file. The details on how to perform this installation are given in the Chapter titled Mocha Installation.

Step 6: Add the below code within the Mocha spec file created.

```
//import chai library
const c = require('chai').expect
//library for parsing JSON file
const s =require('fs')
let h = JSON.parse(s.readFileSync('test/testData/test1.json'))
// test suite name
describe('Tutorialspoint application', function(){
    //iterate the test case
```

```

h.forEach( ({email,password}) =>{
  //test case
  it('Data Driven testing', function(){
    // launch url
    browser.url('https://www.linkedin.com/login')
    //identify the email field then enter key - email
    $("#username").setValue(email)
    //identify password field then enter key - password
    $("#password").setValue(password)
    //identify SSign in button then click
    $("button[type='submit']").click()
    //verify error message
    const e = $('#error-for-password')
    console.log(e.getText() + ' - Error Text')
    //verify Alert text with Chai assertion
    c(e.getText()).to.equal("The password must be provided.")
  });
});
// it is blocked with Smoke keyword
it('Identify element with Id - Smoke', function(){
  // launch url
  browser.url('https://the-internet.herokuapp.com/redirector')
  //identify element with id then click
  $("#redirect").click()
  //obtain page title
  console.log('Page title after click: ' + browser.getTitle())
});
// it block with Smoke keyword
it('Identify element with Tagname - Smoke', function(){
  // launch url
  browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
  //identify element with tagname then obtain text
  console.log($("#<h1>").getText() + " - is the text.")
});
//test case
it('Identify element with Class Name', function(){
  // launch url

```

```

    browser.url('https://www.tutorialspoint.com/about/about_careers.htm')
    //identify element with Class Name then obtain text
    console.log($(".heading").getText() + " - is the text.")
  });
});

```

To trigger only the it blocks connected with Smoke, run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js --mochaOpts.grep Smoke
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:

```

(base) root@Debomitas-MacBook-Air webdriverIO # npx wdio run wdio.conf.js --mochaOpts.grep Smoke
Execution of 1 workers started at 2021-06-13T23:34:19.122Z
[0-0] RUNNING in chrome - /test/specs/testcase1.js
[0-0] Page title after click: The Internet
[0-0] About Careers at Tutorials Point - is the text.
[0-0] PASSED in chrome - /test/specs/testcase1.js
Spec Files:      1 passed, 1 total (100% completed) in 00:00:09

```

After the command has been executed successfully, we find out of the four it blocks, only two it blocks (having Smoke tag in description) have been executed.

46. WebdriverIO — Generate HTML reports from Allure

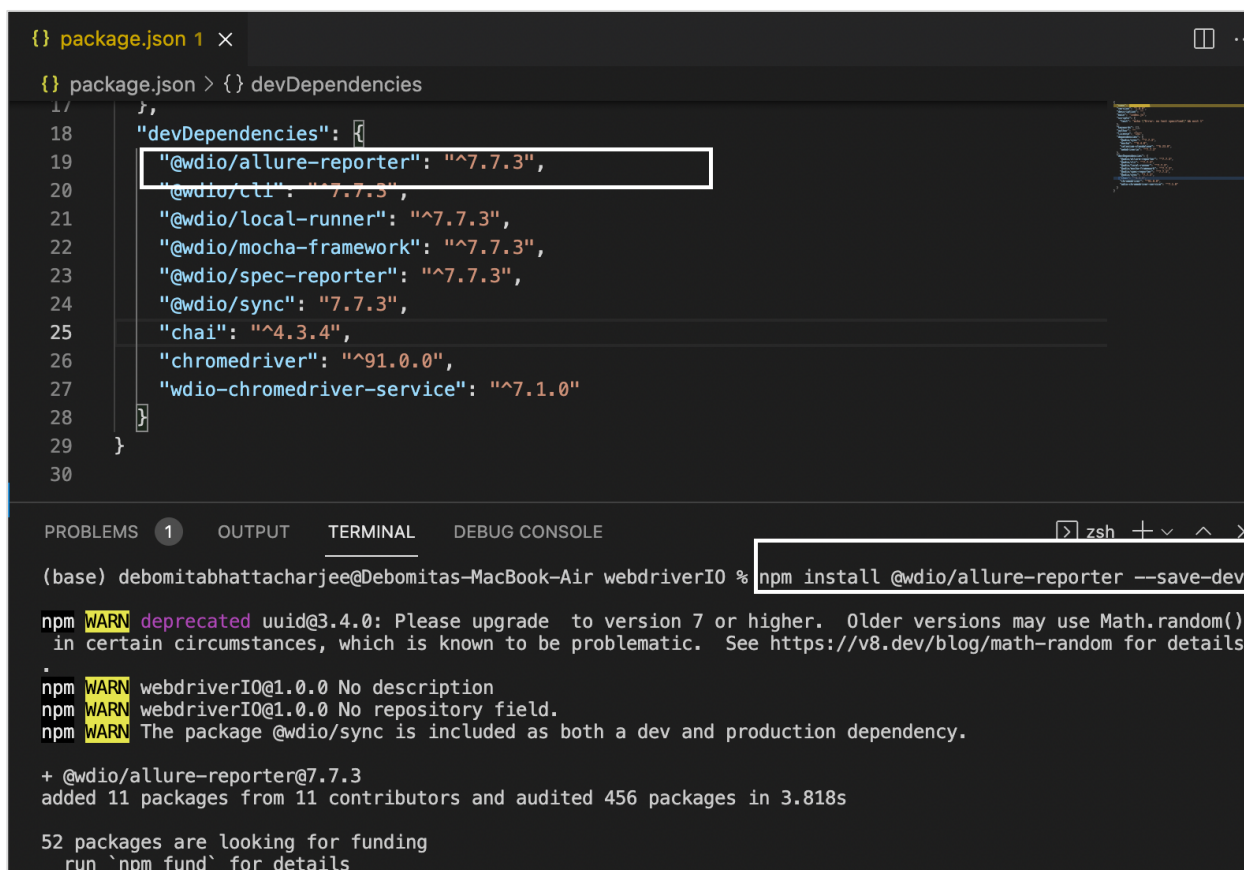
In WebdriverIO, we have a reporter plugin to generate Allure Test Reports. An Allure is a light-weight test reporter tool that creates a brief and well-documented report based on the test results from an automation run.

For installation of Allure and creating its entry in the package.json file, we have to run the below mentioned command:

```
npm install @wdio/allure-reporter --save-dev
```

The details on package.json are discussed in the Chapter titled Package.json file.

The following screen will appear on your computer:



The screenshot shows a code editor with a file named 'package.json' open. The 'devDependencies' section of the file is visible, with the following entries:

```
"devDependencies": {
  "@wdio/allure-reporter": "^7.7.3",
  "@wdio/cli": "^7.7.3",
  "@wdio/local-runner": "^7.7.3",
  "@wdio/mocha-framework": "^7.7.3",
  "@wdio/spec-reporter": "^7.7.3",
  "@wdio/sync": "7.7.3",
  "chai": "^4.3.4",
  "chromedriver": "^91.0.0",
  "wdio-chromedriver-service": "^7.1.0"
}
```

Below the code editor, the terminal output is shown. The command executed is `npm install @wdio/allure-reporter --save-dev`. The output includes several warnings and a successful installation message:

```
(base) debomtabhattacharjee@Debomitas-MacBook-Air webdriverIO % npm install @wdio/allure-reporter --save-dev
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions may use Math.random()
in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details
npm WARN webdriverIO@1.0.0 No description
npm WARN webdriverIO@1.0.0 No repository field.
npm WARN The package @wdio/sync is included as both a dev and production dependency.

+ @wdio/allure-reporter@7.7.3
added 11 packages from 11 contributors and audited 456 packages in 3.818s

52 packages are looking for funding
run `npm fund` for details
```

After installation of the Allure, we have to configure the output directory in the Configuration file `wdio.conf.js` within the reporter options by adding the below code.

The details on how to create a Configuration file are discussed in detail in the Chapter titled `Wdio.conf.js` file and Chapter titled Configuration File generation.

```
reporters: [['allure', {
  outputDir: 'allure-results',
```

```

    disableWebdriverScreenshotsReporting: false,
  }]],

```

The following screen will appear on your computer:

```

JS wdio.conf.js ×
JS wdio.conf.js > [🔍] config > [🔧] reporters
78 // -----
79 // Define all options that are relevant for the Web
80 //
81 // Level of logging verbosity: trace | debug | info
82 logLevel: 'silent',
83
84 reporters: [['allure', {
85   outputDir: 'allure-results',
86   disableWebdriverScreenshotsReporting: false,
87 }]],

```

Here, the outputDir has the default directory of /allure-results. After automation is completed, we shall find this directory generated. It shall contain the .xml files for each of the test files within the specs folder included in the run along with .txt, .png and other files.

Also, to attach the screenshot of the failure test, we have set the parameter disableWebdriverScreenshotsReporting to false.

However, we also need to add an afterStep hook in the wdio.conf.js file having the code as shown below:

```

afterStep: function (test, scenario, { error, duration, passed }) {
  if (error) {
    browser.takeScreenshot();
  }
}

```

The following screen will appear on your computer:

```

JS wdio.conf.js ×
JS wdio.conf.js > [🔍] config > [🔗] afterStep
273 afterStep: function (test, scenario, { error, duration, passed }) {
274   if (error) {
275     browser.takeScreenshot();
276   }
277 }

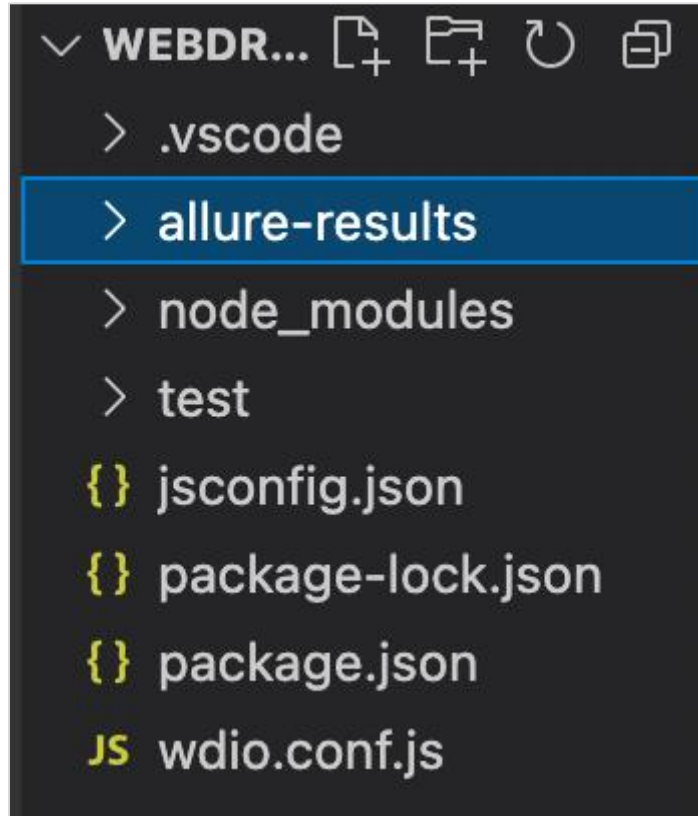
```


Run the Configuration file - wdio.conf.js file with the following command:

```
npx wdio run wdio.conf.js
```

The details on how to create a Configuration file are discussed in detail in the Chapter titled Wdio.conf.js file and Chapter titled Configuration File generation.

The following screen will appear on your computer:



After the command has been executed successfully, a folder called allure-results(as specified in the wdio.conf.js) gets generated within the WebdriverIO project. It contains the reports in xml format.

Next, we have to convert these reports to the HTML format. For this, we shall first install the Allure Commandline tool for generating Allure reports from the test results.

This is done by running the below given command:

```
npm install -g allure-commandline --save-dev
```

After the installation, we can generate the results in HTML format with the below mentioned command:

```
allure generate [allure_output_dir] && allure open
```

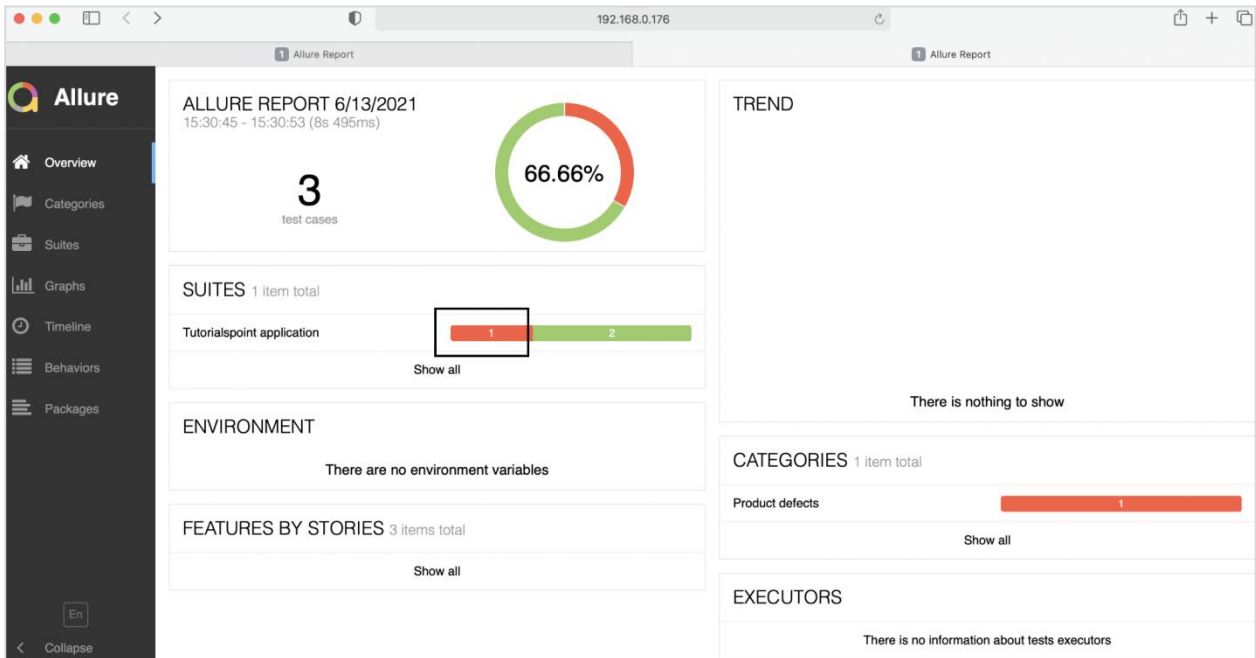
To override an existing result, we have to run the following command:

```
allure generate [allure_output_dir] --clean && allure open
```

The following screen will appear on your computer:

```
(base) root@Debomitas-MacBook-Air webdriverIO # allure generate allure-results && allure open
Report successfully generated to allure-report
Starting web server...
2021-06-13 15:25:25.756:INFO::main: Logging initialized @412ms to org.eclipse.jetty.util.log.StdErrLog
Server started at <http://192.168.0.176:49958/>. Press <Ctrl+C> to exit
```

After the command has been executed successfully, a browser is opened containing the test result. The following screen will appear on your computer:



On clicking the failed test (marked with red), we shall get the details of the test along with the expected, actual output and screenshot of the failure (obtained on expanding Response).

The following screen will appear on your computer:

