Angular

AngularJS → Angular 2 → Angular 4 (released couple of days before) applications are there for a while and is actively maintained by Google and used by millions of developers right now.

Typescript

Typescript is the typed superset of *Javascript* that compiles(transpiles) to Javascript

Protractor

Protractor is an end-to-end test framework for Angular and AngularJS applications. Protractor runs tests against your application running in a real browser, interacting with it as a user would.

1. Angular
2. Protractor
3. Selenium
4. Cucumber

Angular 2

Angular applications are made up of *components*. A *component* is the combination of an HTML template, css styles etc and a component class that controls a portion of the screen

Angular applications are popular because they are

1. Cross platform
2. Easy to write code
3. Native support of Typescript which makes coding faster and highly maintainable
4. Native Supports of testing tools like Karma and Protractor

```
> npm install -g @angular/cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

Bindings

Button events

Calling a method from component from html code

Calling a variable from component from html code

Using ngModel for dynamic binding

Angular automation with protractor + Typescript + cucumber

Part 4 – An introduction to Jasmine

Jasmine

Jasmine is a behaviour-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests

Installation

Jasmine can be installed in many different ways, but we are going to use our NPM way

```
# Local installation:
npm install --save-dev jasmine

# Global installation
npm install -g jasmine
```

Jasmine + Typescript

Since we are going to use Typescript for writing our Jasmine test code, since by default you can write code only in JS.

We need to adding typings for Jasmine using the command

```
npm install --save @types/jasmine
```

https://github.com/executeautomation/EACourseApp

download code

- Import the code into IDE
- npm install
- npm start
- Create one folder "Test"
- Install jasmine types (npm install --save @types/jasmine
- npm install -g typescript
-

Describe and it

Describe and it are Jasmine global functions which are used in conjunction to describe a scenario (problem statement) to make code more understandable

Since describe and it blocks are functions, they can contain any executable code necessary to implement the test. JavaScript scoping rules apply, so variables declared in a describe are available to any it block inside the suite.

```
describe("Testing first", () =>{

    it("should always be happy testing", () =>{
        let a = true
        expect(a).toBe(false);
    })

    it("should feel we are great testers", () =>{
        let a = true
        expect(a).toBe(false);
    })
})
```

Fad arrow syntax

Expect

Jasmine It function cannot complete unless until there is atleast one *Expectation*

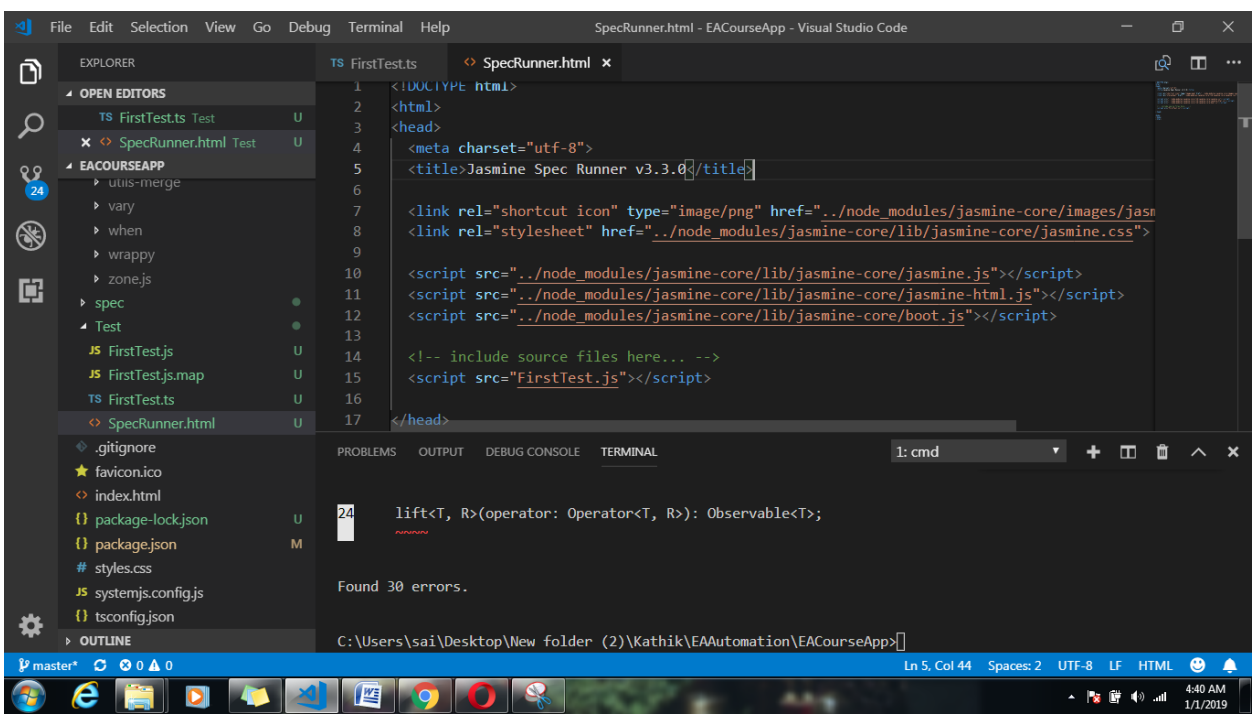
The Expectation are set using expect() method of jasmine and is chained with Matcher

```
describe("Going to write first test", () => {

    it("should pass without any issue", () => {
        let a = 12
        expect(a).toBe(12);
    });

    it("should not pass as the values are undefined", () =>{
        let u;
        expect(u).toBeDefined("Not definied");
    })

});
```

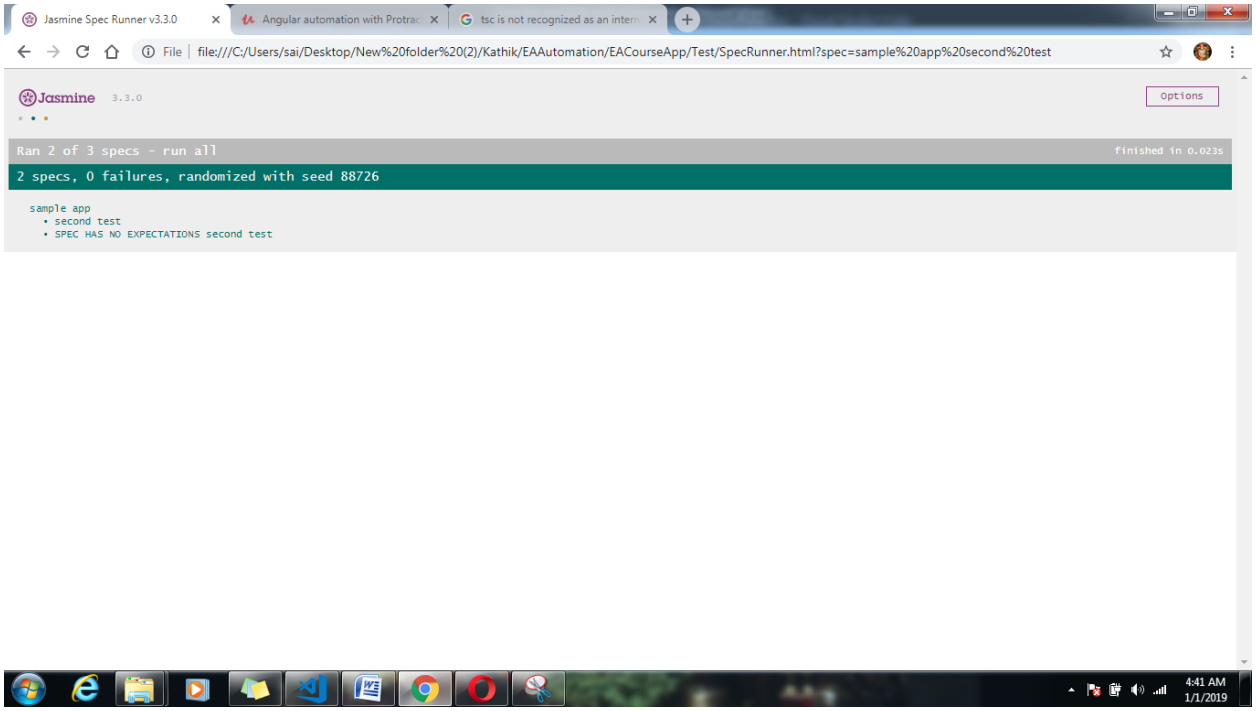
To run spec we need runner , we can download from here https://github.com/jasmine/jasmine/releases

Download the jar and open SpecRunner.html file and change path details

```
TS FirstTest.ts ×    <> SpecRunner.html

describe("sample app",() =>{
    it("first test",() =>{
        let a = 10;
        expect(a).toBe(20);

    });
});
```



Run tsc command and then open specRunner.html file



Introduction → Installing → Configuration of Protractor with VS code

Install protractor

Npm install –save-dev protractor

Webdriver-manager update

Npm install jasminewd2

Working with Locators

The heart of end-to-end tests for webpages is finding DOM elements, interacting with them, and getting information about the current state of your application

As we protractor is built on the top of WebdriverJs and has extended many functionalities within it, finding element in protractor can be done without having to have reference to WebDriver after initializing it for the first time in our code.

ElementFinder

ElementFinder class which extends WebDriverWebElement interface has different action methods used to perform action in UI elements such as

```
headings.
    all
    allowAnimations
    browser_
    clear
    click
    clone
    element
    elementArrayFinder_
    equals
    evaluate
    findElement
    findElements
```

Importing elements in protractor

As we know in Typescript we need to import all the classes/interfaces/variables before using it, even the element property has to be imported as well, something like this

```
import { browser, element, by, ElementFinder } from 'protractor';
```

element() in protractor is equivalent to IWebElement.FindElement (C#) or WebElement.FindElement (Java)

Finding all the elements

element.all() – will do the job for you !

element.all() in protractor is equivalent to IWebElement.FindElements (C#) or WebElement.FindElements (Java)

```
//Getting the first element from collection
var headings = element.all(by.css(".well.hoverwell.thumbnail > div")).first();
```

Finding single/multiple css element

Instead of finding element by element or element.all as shown below

```
//Getting the first element from collection
var headings = element.all(by.css(".well.hoverwell.thumbnail > div")).first();
```

Finding single/multiple css element

Instead of finding element by element or element.all as shown below

```
//Getting the first element from collection
var headings = element.all(by.css(".well.hoverwell.thumbnail > div")).first();
```

We can do this as well

```
var element = $(".well.hoverwell.thumbnail > div");
```

```
var element = $$(".well.hoverwell.thumbnail > div");
```

Element within element

```
element(by.xpath("//course-thumb/div/h2[text()=' Selenium Framework development ']"))
                                .element(by.xpath("//span[contains(text(), '4th')]")).click();
```

FirstTestSpec.ts ● conf.ts

```
1  port { browser, element, by, protractor, $$, $ } from 'protractor';
2
3  scribe("Going to write first test", () => {
4
5    it("should pass without any issue", () => {
6
7        //navigate application
8        browser.get("http://localhost:8808/");
9
10       var headings = element(by.css("//course-thumb/div/h2[text()=' Selenium Framework development ']")).
11                                .element(by.xpath;
12
13       //var headings = $(".well.hoverwell.thumbnail>h2");
14
15       headings.getText().then((text) => {
16           console.log("The heading is :" + text);
17       });
18
19       headings.click();
```

Working with Suites (Protractor + Cucumber Tidbits)

Suites are yet another great way to run your features and scenarios as a whole

```
suites: {
    "homepage": "../features/Home.feature",
    "coursedetail": "../features/CourseDetails.feature"
},
```

```
protractor config.js --suite homepage
```

Configuring in VS code debugger

```
"program": "${workspaceRoot}/node_modules/protractor/bin/protractor",
"stopOnEntry": false,
"args": [
    "${workspaceRoot}/Test/steps/config.js --suite homepage"
],
```

Page Object Model

POM is mainly used to

- ✓ Reduce the number of duplicate code which does same operation
- ✓ Maintain object in separate class file
- ✓ Improved readability of code (as objects will have their identification attribute set)
- ✓ Will have handle of each page using its instance
- ✓ Establish the relation between each pages directly in code, so that performing an operation in one page will return another page (which is expected) can be maintained in POM

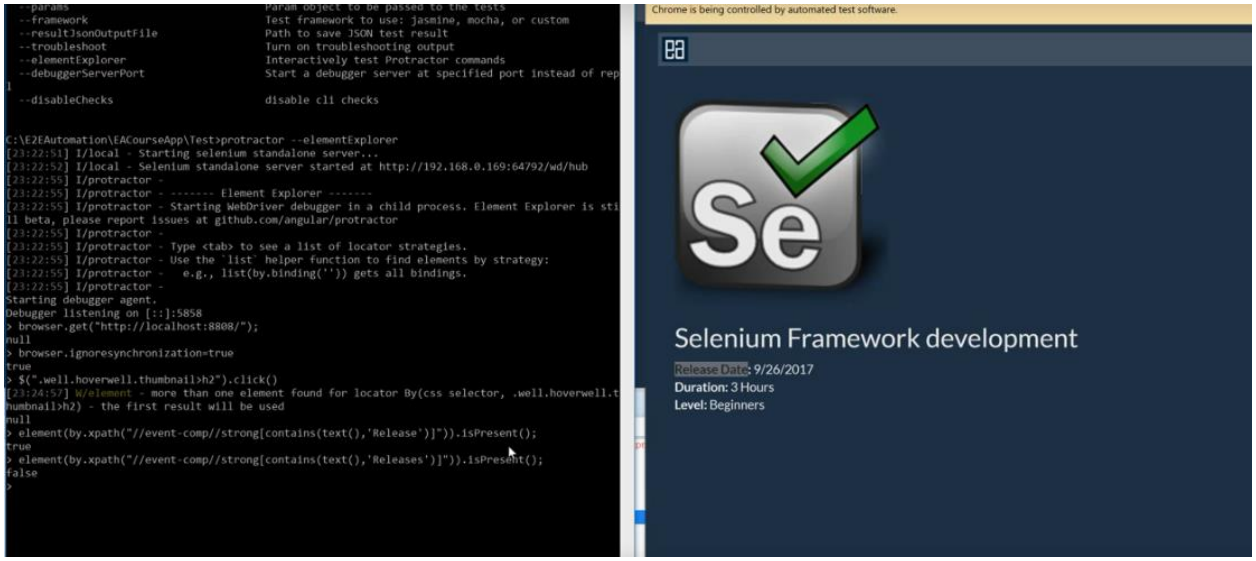
```
import { browser, element, by, ElementFinder, $, $$ } from 'protractor';

export class HomePage {

    OpenBrowser() {
        browser.get("http://localhost:8808/");
    }

    GetText(headings: ElementFinder) {
        headings.getText().then((text) => {
            console.log("The text is :" + text);
        })
    }
}
```

Using Interactive shell

When debugging or first writing test suites, you may find it helpful to try out Protractor commands without starting up the entire test suite. You can do this with the element explorer.

```
protractor --elementExplorer
```

https://www.youtube.com/watch?v=6aPfHrSl0Qk&feature=youtu.be&t=1051



Types of debugging

There are two ways you can debug protractor t

1. Using protractor debugger `browser.debugger();`
2. Using Editor (VS code)

How Protractor works ?

Now we know that Protractor is used for automating Angular applications via real browser.

We also know that Protractor is built on the top of WebDriverJS and NodeJS.

How Protractor works ?

Now we know that Protractor is used for automating Angular applications via real browser.

We also know that Protractor is built on the top of WebDriverJS and NodeJS.

Then eventually we should be aware that both WebDriverJS and NodeJS are *asynchronous* by itself (if not before), but since WebdriverJS is based on Promise, it in turn is wrapped around what is called as Webdriver control flow

WebDriverJS

The WebDriverJS library uses a promise manager to ease the pain of working with a purely asynchronous API. Rather than writing a long chain of promises, the promise manager allows you to write code as if WebDriverJS had a synchronous, blocking API (like all of the other Selenium language bindings)

```
const {Builder, By, until} = require('selenium-webdriver');
new Builder()
    .forBrowser('firefox')
    .build()
    .then(driver => {
      return driver.get('http://www.google.com/ncr')
        .then(_ => driver.findElement(By.name('q')).sendKeys('webdriver'))
        .then(_ => driver.findElement(By.name('btnG')).click())
        .then(_ => driver.wait(until.titleIs('webdriver - Google Search'), 1000))
        .then(_ => driver.quit());
    });
```

```
const {Builder, By, until} = require('selenium-webdriver');

let driver = new Builder()
    .forBrowser('firefox')
    .build();

driver.get('http://www.google.com/ncr');
driver.findElement(By.name('q')).sendKeys('webdriver');
driver.findElement(By.name('btnG')).click();
driver.wait(until.titleIs('webdriver - Google Search'), 1000);
driver.quit();
```

Protractor adaptation

Protractor adapts Jasmine so that each spec automatically waits until the control flow is empty before exiting.

Protractor adaptation

Protractor adapts Jasmine so that each spec automatically waits until the control flow is empty before exiting.

Jasmine expectations are also adapted to understand promises. That's why this line works - the code actually adds an expectation task to the control flow, which will run after the other tasks

(Source: http://www.protractortest.org/#/control-flow)

In future release neither promise manager nor control flow are going to there with Protractor or WebDriverJS, since control flow is set to be removed, Instead of the control flow, you can synchronize your commands with promise chaining or the upcoming ES7 feature **async/await**

Async/Await in Typescript

Async - Await has been supported by TypeScript since version 1.7. Asynchronous functions are prefixed with the *async* keyword; *await* suspends the execution until an asynchronous function return promise is fulfilled and unwraps the value from the *Promise* returned. It was only supported for **target es6** transpiling directly to **ES6 generators**.

Async/ Await

Its a way to tell the runtime to pause the executing of code on the *await* keyword when used on a *promise* and resume only once (and if) the promise returned from the function is settled

When the promise settles execution continues,

- if it was fulfilled then await will return the value,
- if it's rejected an error will be thrown synchronously which we can catch.

This suddenly (and magically) makes asynchronous programming as easy as synchronous programming.
Three things needed for this thought experiment are:

- Ability to *pause function* execution.
- Ability to *put a value inside* the function.
- Ability to *throw an exception inside* the function.

Source: https://basarat.gitbooks.io/typescript/docs/async-await.html

FirstTestSpec.ts ● HomePage.ts

```
1  import { browser, element, by, protractor, $$, $ } from 'protractor';
2  import { HomePage } from "./pages/HomePage";
3
4  describe("Going to write first test", ()
5      //Globally
6      var homePage = new HomePage();
7
8      it("should pass without any issue", async () => {
9          //Open browser
10         await homePage.OpenBrowser("http://localhost:8808/");
11
12                 //Get the headings
13         homePage.GetAllHeadings();
14
15                 //Click the first heading
16         homePage.ClickFirstHeading();
17     });
18
19     it("should not pass as the values are undefined", () => {
20
21     })
```

[ts] An async function or method in ES5/ES3 requires the 'Promise' constructor. Make sure you have a declaration for the 'Promise' constructor or include 'ES2015' in your `--lib` option.

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false,
    "suppressImplicitAnyIndexErrors": true,
    "typeRoots": [
      "./node_modules/@types/"
    ],
    "lib": [
      "es2015"
    ]
  },
  "compileOnSave": true,
  "exclude": [
    "node_modules/*",
```

```
node_modules/@types/core-js/index.d.ts(2091,39): error TS2339: Property 'species' does not exist on type 'SymbolConstructor'.
node_modules/@types/core-js/index.d.ts(2095,37): error TS2339: Property 'split' does not exist on type 'SymbolConstructor'.
node_modules/@types/core-js/index.d.ts(2099,43): error TS2339: Property 'toPrimitive' does not exist on type 'SymbolConstructor'.
node_modules/@types/core-js/index.d.ts(2102,43): error TS2339: Property 'toStringTag' does not exist on type 'SymbolConstructor'.
node_modules/@types/core-js/index.d.ts(2107,43): error TS2339: Property 'unscopables' does not exist on type 'SymbolConstructor'.
node_modules/rxjs/Observable.d.ts(66,60): error TS2693: 'Promise' only refers to a type, but is being used as a value here.
11:34:32 AM - Compilation complete. Watching for file changes.

11:38:22 AM - File change detected. Starting incremental compilation...
```

```
export class HomePage extends BasePage {

    //Selenium framework development course heading
    heading = this.ElementLocator(Locators.heading).element(by.xpath("//span[contains(text(),'4th')]"));


    //All heading
    headings = this.ElementLocator(Locators.headings);

    //Open browser
    async OpenBrowser(url: string){
        await browser.get(url);
    }

    async GetAllHeadings(){
        await this.headings.getText().then((text) => {
            console.log("The heading is :" + text);
        });
    }

    ClickFirstHeading(){
```

```
The heading is :Selenium Framework development
2 .


3 specs, 0 failures
Finished in 28.253 seconds

[11:25:58] I/launcher - 0 instance(s) of WebDriver still running

[11:25:59] I/launcher - chrome #01 passed
```

BDD – Behavioral Driven Development is based on Test Driven Development (TDD) and it aims to bridge the gap between Business analyst and developers.

BDD not only bridges the gap between business analyst and developers but also between

- Manual QA with Automation testers (who write BDD)
- Manual QA with Developers

BDD seems to be like a plan text, but they have their own syntax based on certain tools (which we will discuss next)

There are many tools available to support BDD, some most famous tools are

- ✓ Cucumber
- ✓ Jbehave
- ✓ Nbehave
- ✓ SpecFlow

All the above tools are used in conjunction with many different platforms and languages like JAVA, C#, Ruby, Python, Jruby, JS etc

Gherkin is the format for Cucumber specification.

It is a business readable, Domain specific language which will let anybody to understand the software behavior easily (effortlessly), since they are PLAIN TEXT.

Gherkin has some spaces and indentation to define structure.

Gherkin has very few syntax which make the parser (the tool which uses Gherkin) to behave based on the structure.

The syntax of Gherkin are very simple and are pretty readable as plain text (which we will discuss next)

Gherkin - Syntax

Here are few syntax of Gherkin

1. Feature
2. Background
3. Scenario
4. Given
5. When
6. Then
7. And
8. But
9. Scenario outline
10. Examples
11. Scenario Templates

```
Feature: UserFormFeature
  In this feature we will test user form of the application

  Scenario: Test User Form page
    And I enter UserName and Password
      | UserName | Password |
      | admin    | password |
    And I click login button
    And I enter User Details
      | Initial | Name    | Email                  |
      | KK      | Karthik | karthik@techgeek.co.in |
    And I click save button
    Then I see if the user details saved
```

BDD vs traditional

BDD	Traditional Automation
Its plain text and easy to understand	Its full of code and hard to understand
Its easy to understand by BA/QA/DEV/Automation Test Engineer and all will be in same page	The code are understood only by Automation test engineer (Some times Dev)
Since its plain text format, BDD can be shared even to stakeholders	Impossible
Easy to learn and implement	More knowledge is required while designing

Installation

Here are few npm packages we need to install before we get started with cucumber

```
npm install --save-dev cucumber
```

```
npm install --save-dev @types/cucumber
```

```
npm install --save-dev protractor-cucumber-framework
```

```
npm install --save-dev ts-node
```

```
npm install --save-dev chai-as-promised
```

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

```
Feature: To work with home page

@smoke
Scenario: Click course of application
    Given I navigate to application
    And I get all the heading
    And I click the 'Selenium framework development' course
    Then I should see 'Selenium framework development' course in coursedetails page
```

```
//Globally
var homePage = new HomePage();
var courses = new CourseDetailsPage();

Given(/^I navigate to application$/, async () => {
    await homePage.OpenBrowser("http://localhost:8808/");
});

When(/^I get all the heading$/, async () => {
    await homePage.GetAllHeadings();
});

When(/^I click the '([^\"]*)' course$/, async (headingText) => {
    await homePage.ClickFirstHeading(headingText.toString());
});

Then(/^I should see '([^\"]*)' course in coursedetails page$/, async (course) => {
    expect(courses.GetHeadingtoVerify()).to.be.not.null;
});
```

- Test
  - features
    - Home.feature
  - hooks
    - scenarioHooks.ts
  - pages
    - Base.ts
    - BasePage.ts
    - CourseDetails.ts
    - HomePage.ts
  - steps
    - config.ts
    - Homesteps.ts

File Edit Selection View Go Debug Help

Home.feature | Homesteps.ts | conf.ts | config.ts

```
import { Config } from 'protractor';

export let config: Config = {

    seleniumAddress: 'http://localhost:4444/wd/hub',
    specs: ["../features/*.feature"],

    framework: 'custom',
    frameworkPath: require.resolve('protractor-cucumber-fram

    baseUrl: "http://localhost:8808/",

    cucumberOpts: {
        compiler: "ts:ts-node/register",
        strict: true,
        format: ['pretty'],
        require: ['../steps/*.js', '../hooks/*.js'],
        tags: '@smoke'
    }
}
```

conf.ts

```
import {Config} from 'protractor';

export let config: Config = {
  framework: 'jasmine',
  capabilities: {
    browserName: 'chrome'
  },
  specs: [ './FirstTestSpec.js' ],
  seleniumAddress: 'http://localhost:4444/wd/hub',

  // You could set no globals to true to avoid jQuery '$' and protrac
  // collisions on the global namespace.
  noGlobals: true
};
```

Hooks

Cucumber provides a number of hooks which allow us to run blocks at various points in the Cucumber test cycle

Hooks are used for setup and teardown the environment before and after each scenario. The first argument will be a ScenarioResult for the current running scenario. Multiple *Before* hooks are executed in the order that they were defined. Multiple *After* hooks are executed in the **reverse** order that they were defined.

Hooks type

✓Tagged hooks

✓Event handlers

Event Handlers

Handlers will be passed the associated object as the first argument. Handlers can be synchronous, return a promise, accept an additional callback argument, or use generators.

Event	Object
BeforeFeatures	array of Features
BeforeFeature	Feature
BeforeScenario	Scenario
BeforeStep	Step
StepResult	StepResult
AfterStep	Step
ScenarioResult	ScenarioResult
AfterScenario	Scenario
AfterFeature	Feature
FeaturesResult	FeaturesResult
AfterFeatures	array of Features

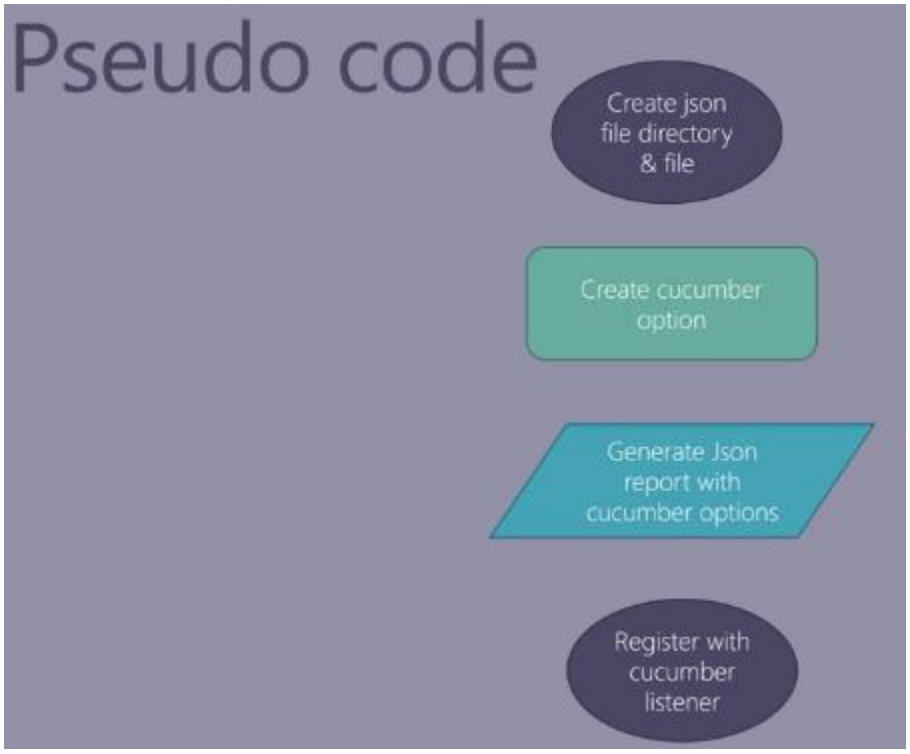
Using npm start command we can launch application

Npm start –port 8808

Cucumber HTML Report

Provide Cucumber JSON report file created from your framework and this module will create pretty HTML reports

```
npm install cucumber-html-reporter --save-dev
```



Create File/Directory

```
private CreateReportFile(dirName, fileName, fileContent) {
    //Check of the directory exist
    if (!fs.existsSync(dirName))
        mkdirp.sync(dirName);
    try {
        //Write create the file
        fs.writeFileSync(fileName, fileContent);
    }
    catch (message) {
        console.log("Filed to create File/Directory :" + message);
    }
}
```

Create cucumber reporting option

```
private cucumberReporterOptions = {
    theme: "bootstrap",
    jsonFile: this.jsonFile,
    output: this.htmlDir + "/cucumber_reporter.html",
    reportSuiteAsScenarios: true,
    metadata: {
        "App Version":"0.0.1",
        "Test Environment": "Testing",
        "Browser": "Chrome  59.0.945",
        "Platform": "Windows 10",
        "Parallel": "Scenarios",
        "Executed": "Local"
    }
};
```

Generating JSON report

```
private GenerateCucumberReport(cucumberReportOption) {
    report.generate(cucumberReportOption);
}

JsonFormatter = new Cucumber.JsonFormatter({
    log: jLog => {
        this.CreateReportFile(this.jsonDir, this.jsonFile, jLog);
        this.GenerateCucumberReport(this.cucumberReporterOptions);
    }
});
```

Finally

```
registerListener(JsonFormatter);
```

Working with Suites

Suites are yet another great way to run your features and scenarios as a whole

```
suites: {
    "homepage": "../features/Home.feature",
    "coursedetail": "../features/CourseDetails.feature"
},
```

How to execute the code ?

```
protractor config.js --suite homepage
```

Configuring in VS code debugger

```
"program": "${workspaceRoot}/node_modules/protractor/bin/protractor",
"stopOnEntry": false,
"args": [
    "${workspaceRoot}/Test/steps/config.js --suite homepage"
],
```

```
\\ - protractor config.js --suite homepage
        at Object.<anonymous> (C:\E2EAutomation\EACourseApp\Test\steps\HomeSteps.ts:24:53)
        at step (C:\E2EAutomation\EACourseApp\Test\steps\HomeSteps.js:32:23)
        at Object.next (C:\E2EAutomation\EACourseApp\Test\steps\HomeSteps.js:13:53)
        at C:\E2EAutomation\EACourseApp\Test\steps\HomeSteps.js:7:71
        at __awaiter (C:\E2EAutomation\EACourseApp\Test\steps\HomeSteps.js:3:12)
        at World.<anonymous> (C:\E2EAutomation\EACourseApp\Test\steps\HomeSteps.ts:23:81)

2 scenarios (1 failed, 1 passed)
5 steps (1 failed, 4 passed)
0m00.524s
Cucumber HTML report C:\E2EAutomation\EACourseApp\Test\steps/reports/html/cucumber_reporter.html generated successfully.

[22:58:17] I/launcher - 0 instance(s) of WebDriver still running
[22:58:17] I/launcher - chrome #01 failed 1 test(s)
[22:58:17] I/launcher - overall: 1 failed spec(s)
[22:58:17] I/launcher - Process exited with error code 1

C:\E2EAutomation\EACourseApp\Test\steps>protractor config.js
[22:58:36] I/launcher - Running 1 instances of WebDriver
[22:58:36] I/hosted - Using the selenium server at http://localhost:4444/wd/hub
0 scenarios
0 steps
0m00.000s
Cucumber HTML report C:\E2EAutomation\EACourseApp\Test\steps/reports/html/cucumber_reporter.html generated successfully.

[22:58:41] I/launcher - 0 instance(s) of WebDriver still running
[22:58:41] I/launcher - chrome #01 passed

C:\E2EAutomation\EACourseApp\Test\steps>protractor config.js --suite homepage
```

Data Tables are handy for passing a list of values to a step definition:

```
Then I should see all  course in coursedetails page
    |Courses    |Duration   |
    |Selenium   |2          |
    |Java       |3          |
```

How it looks like in protractor ?

```
TableDefinition
```

```
Then(/^I should see all  course in coursedetails page$/, async (table: TableDefinition) => {
```

```
table.rows()
```

CucumberReportExtension.ts | ScenarioHook.ts | Home.feature | CourseDetails.feature | config.ts | conf.ts | HomeSteps.ts

```
Feature: To work with home page

@smoke
Scenario: Click course of application
    And I get all the heading
    And I click the 'Selenium framework development' course
    Then I should see 'Selenium framework development' course in coursedetails page
    Then I should see all course information in coursedetails page
        |Courses    |Duration   |
        |Selenium   |2          |
        |Java       |3          |

@regression
Scenario: Click course of application second time
    And I get all the heading
    And I click the 'Selenium framework development' course
    Then I should see 'Selenium framework development' course in coursedetails page
```

```
Then(/^I should see all course information in coursedetails page$/, async (table: TableDefinition) => {

    table.rows().forEach(element => {
        console.log(element);
    });

});
```

```
\\ - protractor  config.js --suite coursedetails
[23:08:58] I/launcher - Running 1 instances of WebDriver
```

```
Then(/^I should see all course information in coursedetails page$/, async (table: TableDefinition) => {

    let localTable = [
        [ 'Selenium', '2' ],
        [ 'Java', '3' ]
    ]

    table.rows().forEach(element => {
        console.log(element);
    });

    assert.deepEqual(localTable, table.rows(), "The datasource does not matches with the step definitio

});

});
```

Data Driven testing

Data Driven testing is one of the important concept which helps to pass data to tests from external data source

Again, this is one of the best way to remove data from hard coding rather use it from external data source.

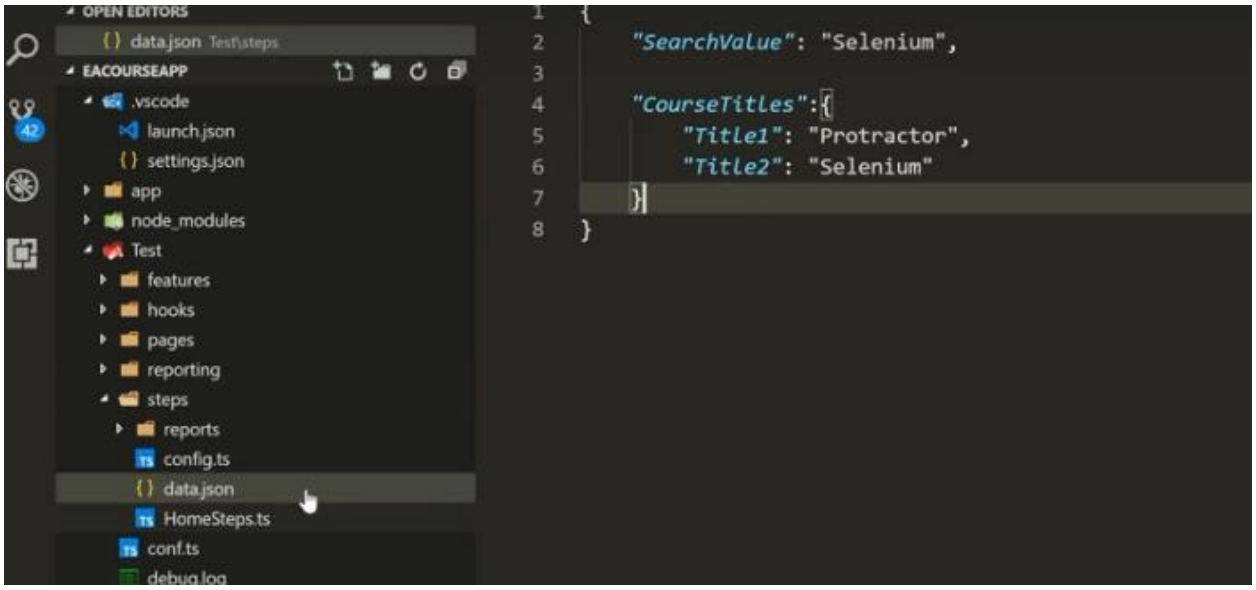
Different data source

There are different external data sources available to read from such as

1. Excel
2. JSON
3. XML
4. Database tables

```
npm install --save load-json-file
```

```
npm install --save ts-xlsx
```




```
@regression
Scenario: Search for course from External DataSource
    Given I navigate to application
    And I enter text in search from exterral data source
    And I get all the heading
```

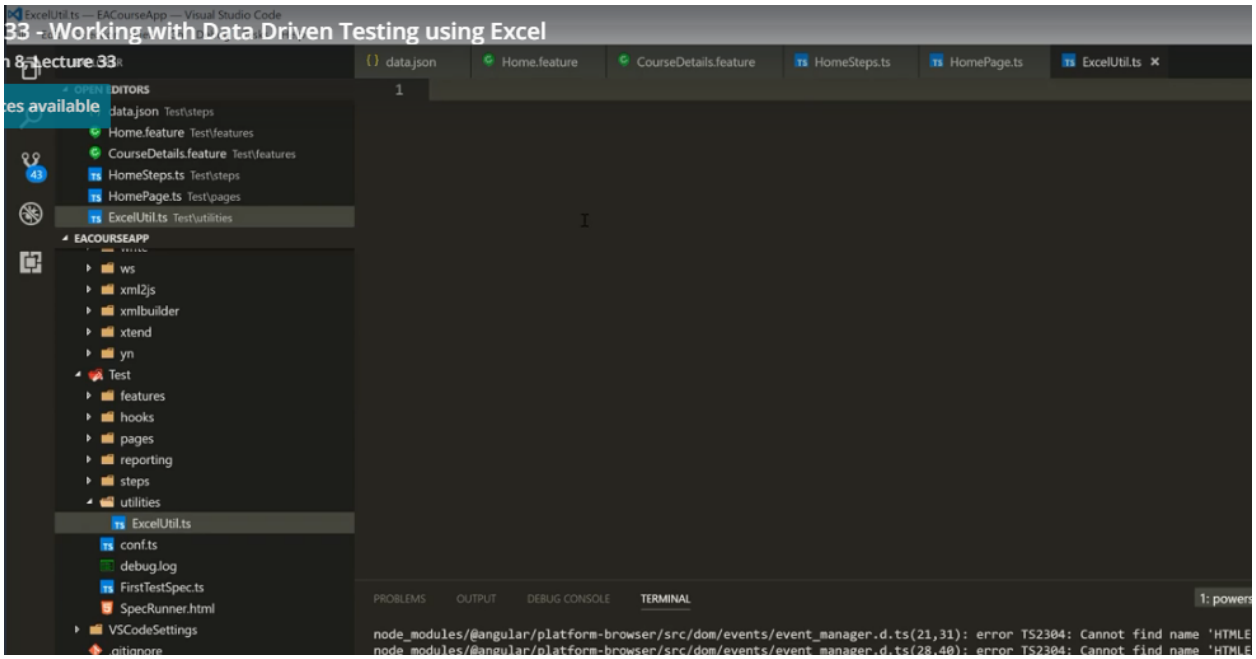
```
import * as json from 'load-json-file'
```

```
PS C:\E2EAutomation\EACourseApp> npm install --save-dev @types/load-json-file
```

```
async EnterDataInSearchFromJson(){
    json("./data.json").then( (x) => {
        console.log(x);
    });
}
```

```
async EnterDataInSearchFromJson() {
    json("./data.json").then((x) => {
        console.log(x);
        this.searchText.sendKeys((<any>x).SearchValue);
    });
}
```

```
npm install --save ts-xlsx
```



Part 33 - Working with Data Driven Testing using Excel
Section 8, Lecture 33
Resources available

CourseDetails.feature | HomeSteps.ts | HomePage.ts | ExcelUtil.ts

```typescript

import * as excel from 'ts-xlsx';
import { IWorkSheet } from "xlsx";

export class ExcelUtil {

    ReadExcelSheet(filepath: string): IWorkSheet{
        let file = excel.readFile(filepath);
        let sheet = file.Sheets["Sheet1"];
        return excel.utils.sheet_to_json(sheet)[0];
    }
}
```

```typescript
import {excel } from '../utilities/ExcelUtil';
```

C5

	A
1	SearchValue
2	Selenium automation
3	

```typescript
async EnterDataInSearchFromExcel() {
    let sheet = ExcelUtil.ReadExcelSheet("./data.xlsx");

    console.log(sheet);
}
```

```typescript
async EnterDataInSearchFromExcel() {
    let sheet = ExcelUtil.ReadExcelSheet("./data.xlsx");

    console.log((<any>sheet).SearchValue);
}
```

```
    async EnterDataInSearchFromExcel() {
        let sheet = <SearchData>ExcelUtil.ReadExcelSheet("./data.xlsx");

        console.log(sheet.);

    }


}

interface SearchData{
    SearchValue: string,
    CourseTitle: string,
    Durations: string
}
```

```
async EnterDataInSearchFromExcel() {
    let sheet = <SearchData>ExcelUtil.ReadExcelSheet("./data.xlsx");

    console.log(sheet.SearchValue);

    this.searchText.sendKeys(sheet.SearchValue);
}
```

Taking screenshot while test fails

We can verify if test fails using isFailed() method of StepResult in Cucumber, but using that in our existing registerHandler is important to attach those taken screenshots in existing Cucumber test report.

```
if (StepResult.isFailed()) {
    return browser.takeScreenshot().then(function (screenShot) {
        var decodedImage = new Buffer(screenShot, 'base64');
        StepResult.attachments.push({
            data: decodedImage.toString('base64'),
            mimeType: 'image/png'
        });
    });
}
```

Working with Gherkin in VS Code

So far we have been going throw various trouble while working with VS code to navigate to step definition, matching duplicate step definition implementation etc which are pretty normal with other languages and IDE like Specflow with C# or Cucumber in Java with Visual studio and IntelliJ respectively.

```
@regression
Scenario: Search for course from External DataSource
    Given I navigate to application
    And I enter text in search from external data source
    And I get all the heading
```

Cucumber (Gherkin) plugin

It has following feature

- *Syntax highlight*
- *Basic Snippets support*
- *Auto-parsing of feature steps from pathes, provided in settings.json*
- *Autocompletion of steps*
- *Ontype validation for all the steps*
- *Definitions support for all the steps parts*
- *Document format support, including tables formatting*
- *Supporting of many spoken languages*
- *Gherking page objects native support*

Handling global timeout

Handling global timeout throughout protractor is really important if you would have encountered problem something like this

```
Scenario: Search for course from External DataSource
Unhandled rejection VError: a handler errored, process exiting: ..\hooks\ScenarioHook.ts:12: function timed out after 5000 milliseconds
    at C:\Protractor\node_modules\cucumber\lib\runtime\event_broadcaster.js:78:21
    at Generator.next (<anonymous>)
    at Generator.tryCatcher (C:\Protractor\node_modules\cucumber\node_modules\bluebird\js\release\util.js:16:23)
    at PromiseSpawn._promiseFulfilled (C:\Protractor\node_modules\cucumber\node_modules\bluebird\js\release\generators.js:97:49)
    at Promise._settlePromise (C:\Protractor\node_modules\cucumber\node_modules\bluebird\js\release\promise.js:574:26)
    at Promise._settlePromise0 (C:\Protractor\node_modules\cucumber\node_modules\bluebird\js\release\promise.js:614:10)
    at Promise._settlePromises (C:\Protractor\node_modules\cucumber\node_modules\bluebird\js\release\promise.js:693:18)
    at Async._drainQueue (C:\Protractor\node_modules\cucumber\node_modules\bluebird\js\release\async.js:133:16)
```

Understanding timeouts

There are different timeouts available in protractor as we already know

- Wait for Page to load
- Wait for angular
- Asynchronous script timeout
- Timeout from Jasmine
- Timeout from Sauce lab

Time outs

Because WebDriver tests are asynchronous and involve many components, there are several reasons why a timeout could occur in a Protractor test and this can b handled very easily using one global time out

```
defineSupportCode(({ registerHandler, registerListener, setDefaultTimeout }) => {

    setDefaultTimeout(50000);
```

Go to Dashboard

Resources available

```
    });

    registerHandler('AfterFeature', async () => {
        console.log("Executing After feature !!");
    });

    registerHandler("StepResult", async(StepResult) =>{
        if(StepResult.isFailed()){
            return browser.takeScreenshot().then(function (screenShot){
                var decodedImage = new Buffer(screenShot, 'base64');
                StepResult.attachments.push({
                    data: decodedImage.toString('base64'),
                    mimeType: 'image/png'
                });
            });
        }
    });

    registerListener(JsonFormatter);
});
```

```
2 scenarios (1 failed, 1 passed)
7 steps (1 failed, 6 passed)
0m02.500s
Cucumber HTML report C:\E2EAutomation\dd\Test\steps/reports/html/cucumber_reporter.html generated successfully.
[12:47:37] I/launcher - 0 instance(s) of WebDriver still running
[12:47:37] I/launcher - chrome #01 failed 1 test(s)
[12:47:37] I/launcher - overall: 1 failed spec(s)
[12:47:37] E/launcher - Process exited with error code 1
PS C:\E2EAutomation\dd\Test\steps>
```

Capabilities in protractor

In Protractor config file (see config.ts), all browser setup is done within the capabilities object. This object is passed directly to the WebDriver builder

```
capabilities :{
    'browserName': 'firefox'
},
```

Handling multiple browsers

Protractor has a new capability called `multiCapabilities:` which let our existing test run in parallel without needing to do any code change

```
multiCapabilities: [{
    'browserName': 'firefox'
  }, {
    'browserName': 'chrome'
  }],
```

protractor/browser-setu

https://github.com/angular/protractor/blob/master/docs/browser-setup.md

Install | Execute Automation | MVC

In your Protractor config file (see config.ts), all browser setup is done within the `capabilities` object. This object is passed directly to the WebDriver builder (builder.js).

See DesiredCapabilities for full information on which properties are available.

Using Mobile Browsers

Please see the Mobile Setup documentation for information on mobile browsers.

Using Browsers Other Than Chrome

To use a browser other than Chrome, simply set a different browser name in the capabilities object.

```
capabilities: {
  'browserName': 'firefox'
}
```

You may need to install a separate binary to run another browser, such as IE or Android. For more information, see SeleniumHQ Downloads.

Adding Chrome-Specific Options

Chrome options are nested in the `chromeOptions` object. A full list of options is at the ChromeDriver site. For example, to show an FPS counter in the upper right, your configuration would look like this:

```
capabilities: {
  'browserName': 'chrome',
  'chromeOptions': {
    'args': ['show-fps-counter=true']
  }
```

Upgrading Protractor 2.x to 4.x

Its been a while we have upgraded our protractor code from 2.x to latest version and since then many things have changed, in fact many ***breaking changes*** happened and we started to see many problems reported by YOU who subscribed the course

Documentation

- update nodejs example (#898, João Guilherme Farias Duda)
- fix typo and make punctuation consistent (#909, Dmitry Shirokov)
- normalize CHANGELOG (#915, Jayson Smith)

3.0.0 (2017-08-08)

BREAKING CHANGES

- `pretty` formatter has been removed. All errors are now reported in a `pretty` format instead. The `progress` formatter is now the default.
- Major changes to custom formatter and custom snippet syntax APIs due to rewrite in support of the event protocol. Please see the updated documentation.
- Remove `registerHandler` and `registerListener`. Use `BeforeAll` / `AfterAll` for setup code. Use the event protocol formatter if used for reporting. Please open an issue if you have another use case.
- Remove deprecated `addTransform`. Use `defineParameterType` instead.
- `cucumber-expressions`:
  - using an undefined parameter type now results in an error
  - `{stringInDoubleQuotes}` is now `{string}` which works for strings in single or double quotes.

4.0.0 (2018-01-24)

BREAKING CHANGES

- cucumber now waits for the event loop to drain before exiting. To exit immediately when the tests finish running use `--exit`. Use of this flag is discouraged. See here for more information
- remove `--compiler` option. See here for the new way to use transpilers
- remove binaries `cucumber.js` and `cucumberjs`. Use `cucumber-js`

New Features

- can now use glob patterns for selecting what features to run
- update `--require` to support glob patterns
- add `--require-module <NODE_MODULE>` to require node modules before support code is loaded
- add snippet interface "async-await"
- add `--parallel <NUMBER_OF_SLAVES>` option to run tests in parallel. Note this is an experimental feature. See here for more information

Bug Fixes

- revert json formatter duration to nanoseconds

Deprecations

- `defineSupportCode` is deprecated. Require/import the individual methods instead

```
var {defineSupportCode} = require('cucumber');

defineSupportCode(function({Given}) {
  Given(/^a step$/, function() {});
```

Some of the changes

```
defineSupportCode(({ registerHandler, registerListener }) => {

    registerHandler('BeforeFeature', async () => {
        console.log("Executing before feature !!");
    });

    registerHandler('BeforeScenario', async () => {
        await browser.get(config.baseUrl);
    });
```

```
BeforeAll(async () => {
    CucumberReportExtension.CreateReportFile(jsonReports)
    await browser.get(config.baseUrl);
});



After(async function(scenario) {
```

```
import { defineSupportCode, TableDefinition } from 'cucumber'
import { HomePage } from "../pages/HomePage";
import { expect, assert } from 'chai'
import { CourseDetailsPage } from "../pages/CourseDetails";

defineSupportCode(({Given, When, Then}) => {

    var homePage = new HomePage();
    var coursedetails = new CourseDetailsPage();

    Given(/^I navigate to application$/, async () => {
        await homePage.OpenBrowser("http://localhost:8808/");
    });
```

```
const { Given, When, Then } = require("cucumber");
import { HomePage } from "../pages/HomePage";
import { expect, assert } from 'chai'
import { CourseDetailsPage } from "../pages/CourseDetails";

    var homePage = new HomePage();
    var coursedetails = new CourseDetailsPage();

    Given(/^I navigate to application$/, async () => {
        await homePage.OpenBrowser("http://localhost:8808/");
    });
```

Upgrading cucumber-html-report

There is a breaking change in our cucumber-html-report as well

For CucumberJS

This module converts Cucumber's JSON format to HTML reports.

> The code has to be separated from CucumberJS execution (after it).

In order to generate JSON formats, run the Cucumber to create the JSON format and pass the file name to the formatter as shown below,

```
$ cucumberjs test/features/ -f json:test/report/cucumber_report.json
```

Multiple formatter are also supported,

```
$ cucumberjs test/features/ -f summary -f json:test/report/cucumber_report.json
```

> Are you using cucumber with other frameworks or running cucumber-parallel? Pass relative path of JSON file to the `options` as shown here

Options

`theme`

```
AfterAll(async () => {
    CucumberReportExtension.GenerateCucumberReport();
});
```

Rather we should use it in config.ts

```
onComplete: () => {
    CucumberReportExtension.GenerateCucumberReport();
},
```

How to upgrade ?

```
"devDependencies": {
  "@types/chai": "^4.0.1",
  "@types/core-js": "~0.9.34",
  "@types/load-json-file": "^2.0.5",
  "@types/node": "~6.0.46",
  "chai": "^4.0.2",
  "chai-as-promised": "^7.0.0",
  "concurrently": "2.2.0",
  "cucumber": "^2.3.1",
  "cucumber-html-reporter": "^2.0.0",
  "jasmine": "^2.5.3",
  "load-json-file": "^2.0.0",
  "protractor": "^5.1.1",
  "protractor-cucumber-framework": "^3.1.2",
  "ts-node": "^3.1.0",
  "ts-xlsx": "0.0.11",
  "typescript": "^2.3.4"
},
```

```
                                    Use --skip-unused to skip this check.
                                    To remove this package: npm uninstall --save-dev chai-as-promised

concurrently                 MAJOR UP  Major update available. https://github.com/kimmobrunfeldt/concurrently
                                       npm install --save-dev concurrently@3.5.1 to go from 2.2.0 to 3.5.1

cucumber                     MAJOR UP  Major update available. http://github.com/cucumber/cucumber-js
                                       npm install --save-dev cucumber@4.0.0 to go from 2.3.1 to 4.0.0

cucumber-html-reporter       MAJOR UP  Major update available. https://github.com/gkushang/cucumber-html-reporter#readme
                                       npm install --save-dev cucumber-html-reporter@4.0.2 to go from 2.0.3 to 4.0.2

jasmine                      MAJOR UP  Major update available. http://jasmine.github.io/
                                       npm install --save-dev jasmine@3.1.0 to go from 2.99.0 to 3.1.0
                             NOTUSED?  Still using jasmine?
                                       Depcheck did not find code similar to require('jasmine') or import from 'jasmine',
                                       Check your code before removing as depcheck isn't able to foresee all ways dependencies can be used.
                                       Use --skip-unused to skip this check.
                                       To remove this package: npm uninstall --save-dev jasmine

load-json-file               MAJOR UP  Major update available. https://github.com/sindresorhus/load-json-file#readme
                                       npm install --save-dev load-json-file@4.0.0 to go from 2.0.0 to 4.0.0

protractor-cucumber-framework MAJOR UP Major update available. https://github.com/protractor-cucumber-framework/protractor-cucumber-framework
                                       npm install --save-dev protractor-cucumber-framework@4.2.0 to go from 3.1.2 to 4.2.0

ts-node                      MAJOR UP  Major update available. https://github.com/TypeStrong/ts-node
                                       npm install --save-dev ts-node@5.0.1 to go from 3.3.0 to 5.0.1
                             NOTUSED?  Still using ts-node?
                                       Depcheck did not find code similar to require('ts-node') or import from 'ts-node',
                                       Check your code before removing as depcheck isn't able to foresee all ways dependencies can be used.
                                       Use --skip-unused to skip this check.
                                       To remove this package: npm uninstall --save-dev ts-node
```

NCU –U

package.json - Protractor_old - Visual Studio Code

ading Protractor, Cucumber, Cucumber HTML report to latest ...

n 9 Lecture 39

OPEN EDITORS
- Workspace Settings .vscode
- ScenarioHook.ts Test\hooks
- HomePage.ts Test\pages
- config.ts Test\steps
- package.json

PROTRACTOR_OLD
- CourseDetails.ts
- HomePage.ts
- reporting
- steps
  - reports
    - html
      - cucumber_reporter.ht...
    - json
  - config.ts
  - data.json
  - Data.xlsx
  - HomeSteps.ts
- utilities
- conf.ts
- debug.log
- FirstTestSpec.ts
- SpecRunner.html
- VSCodeSettings
- .gitignore
- favicon.ico
- favicon.zip
- index.html
- npm-debug.log
- package-lock.json

Workspace Settings | ScenarioHook.ts | HomePage.ts | config.ts | package.json

```
40    "devDependencies": {
41      "@types/chai": "^4.0.1",
42      "@types/core-js": "~0.9.34",
43      "@types/load-json-file": "^2.0.5",
44      "@types/node": "~6.0.46",
45      "chai": "^4.0.2",
46      "chai-as-promised": "^7.0.0",
47      "concurrently": "2.2.0",
48      "cucumber": "^2.3.1",
49      "cucumber-html-reporter": "^2.0.0",
50      "jasmine": "^2.5.3",
51      "load-json-file": "^2.0.0",
52      "protractor": "^5.1.1",
53      "protractor-cucumber-framework": "^3.1.2",
54      "ts-node": "^3.1.0",
55      "ts-xlsx": "0.0.11",
56      "typescript": "^2.3.4"
57    },
58    "repository": {}
59  }
60
```

PROBLEMS 2 | OUTPUT | DEBUG CONSOLE | TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Protractor_old> ncu -u
```

```
cucumberOpts: {
    compiler: "ts:ts-node/register",
    strict: true,
    // format: ['pretty'],
    format: "json:./reports/json/cucumber_report.json",
    require: ['../steps/*.js', '../hooks/*.js'],
    tags: '@smoke or @regression'
}
```

Important

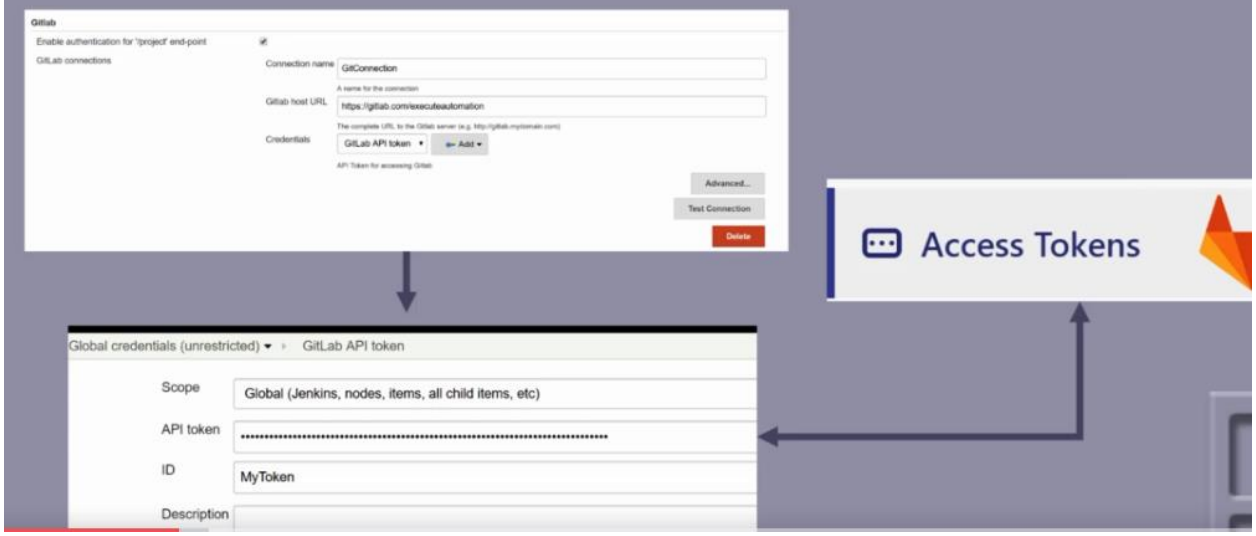
This video assumes you have following pre-requisite

1. Jenkins installed in your machine
2. GitLabs account already created and project pushed into the gitlabs repo
3. Good basic knowledge in Jenkins

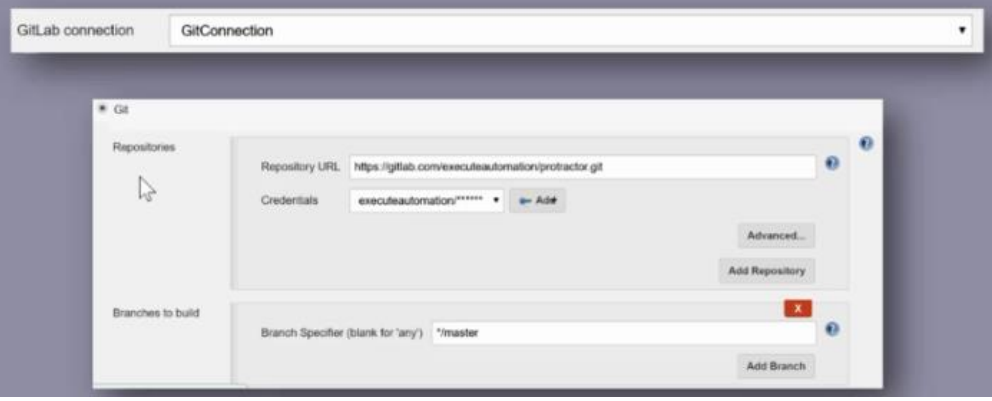
Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

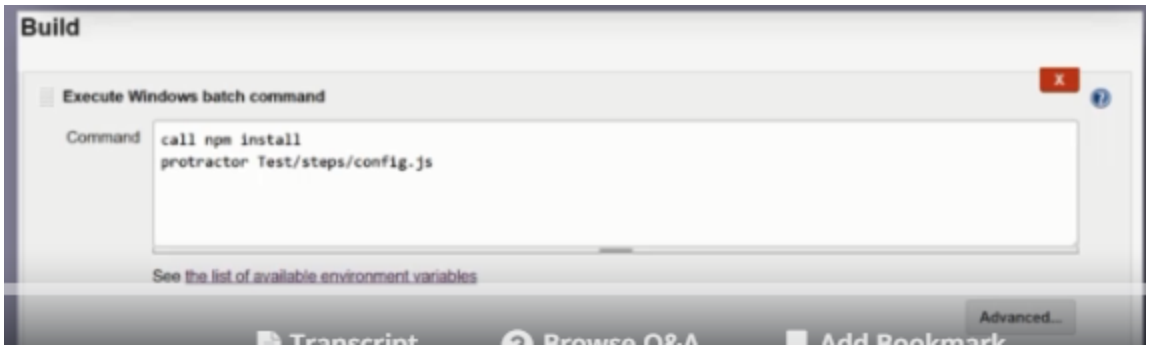
GitLab is a web-based Git-repository manager with wiki and issue-tracking features, using an open-source license, developed by GitLab Inc.

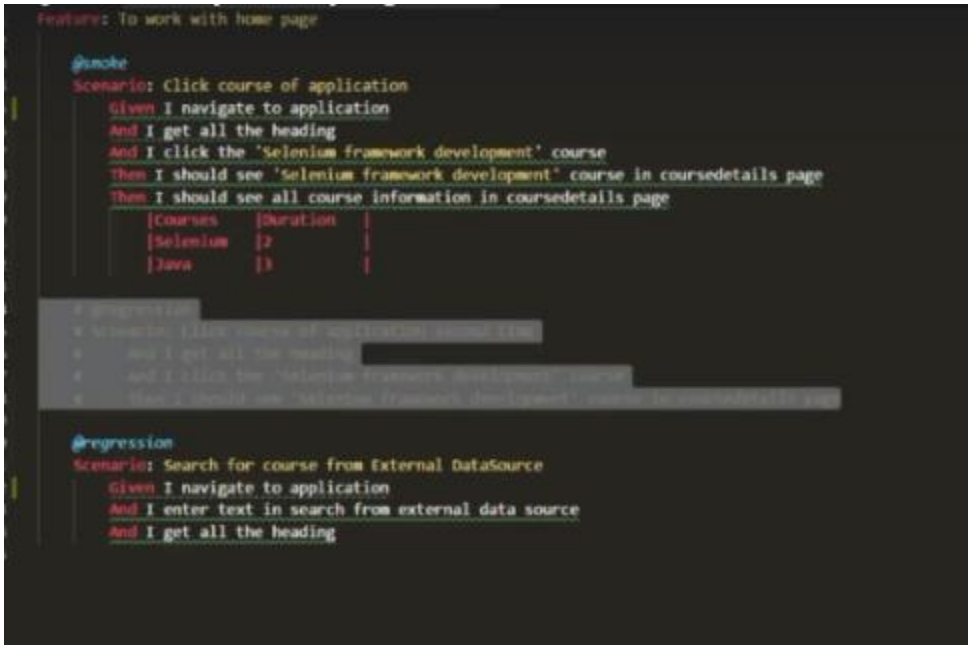
GitLabs configuration



Freestyle Jenkins project







Angular automation with Protractor + Typescript + Cucumber