Test Automation for Web application testing using Protractor, Cucumber and JavaScript

LTTS
GLOBAL
ENGINEERING
ACADEMY

L&T Technology Services

Name of Mentor: MONICA D ANAND
PS No. : 20140815

Document History

Ver. Rel. No.	Release Date	Prepared. By	Reviewed By	Approved By	Remarks/Revision Details
1.0	29/04/2021	Mithun M R (99003716)			
	29/04/2021	Shaik Rehana (99003549)			
	29/04/2021	Sneha Anand (99003525)			

Contents

PROJECT DETAILS:

AIM:

Engineers shall be able to understand the automation testing process and activities involved in automation testing for Web applications using protractor tool. Develop test automation framework with cucumber BDD using JavaScript.

PROBLEM STATEMENT:

- Understanding of UI Automation for web application.
- How BDD framework helps get the Automation done for web application.
- Framework development to support web application automation.

OBJECTIVES & LEARNING OUTCOMES OF THE PROJECT

- Understanding end to end process involved in test automation for web application.
- Configuration of the test environment and infrastructure . To install and use protractor and web drivers.
- Identifying web elements uniquely and develop page object model -repository.
- Creating test scripts using JavaScript.
- Able to leverage open-source tools and understanding of UI Automation concepts.
- Understand cucumber framework and its advantages.
- Generate automated html test summary report

HARDWARE & SOFTWARE RECOMMENDATIONS

- Pc with windows os
- Node js
- Protractor
- BDD framework cucumber
- Assertion framework – chai
- Visual studio code - editor

WEB APPLICATION TEST AUTOMATION

Test Automation:

Test automation is the practice of running tests automatically, managing test data, and utilizing results to improve software quality. It is a software testing technique that performs using special automated testing software tools to execute a test case suite.

Benefits of Test automation:

1. Increased testing efficiency,
2. Increased testing effectiveness,
3. Faster time to market.

Feasibility of Test Automation:

While doing a feasibility Study for test automation the factors to be considered and percentage of importance are as shown below:

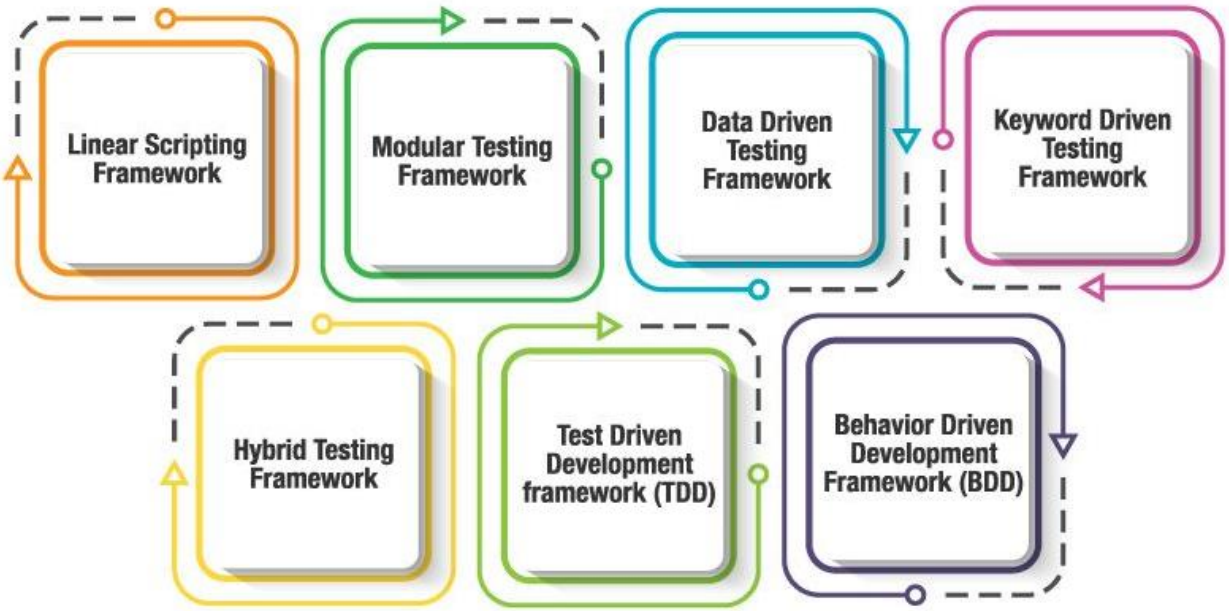
- Technical analysis (20)
- Complexity(15)
- Application stability(15)
- Test Data(15)
- Application size(10)
- Reusability of Automation scripts(10)
- Execution across Environments(15)

Automation Framework:

A Test Automation Framework is a set of guidelines like coding standards, test-data handling, object repository treatment etc. Which when followed during automation scripting produces beneficial outcomes like increased code re-usage, higher portability, reduced script maintenance cost etc. It integrates various functions like libraries, generic functions, test data, properties of test environment, and various reusable modules.

- Enhances efficiency during the design and development of automated test scripts by enabling the reuse of components or code
- Provides a structured development methodology to ensure uniformity of design across multiple test scripts to reduce dependency on individual test-cases
- Enables reliable issue and bug detection and delivers proper root-cause analysis with minimum human intervention for the system under test
- Reduces dependence on teams by automatically selecting the test to execute according to test scenarios
- Refines dynamically test scope according to changes in the test strategy or conditions of the system under test
- Improves utilization of various resources and enables maximum returns on efforts

Types of Test Automation Framework:



Linear Automation Framework:

It is the simplest of all Testing Automation Frameworks and also known as **"Record & Playback"**. In this Automation Testing Framework, Tester manually records each step (Navigation and User Inputs), Inserts Checkpoints (Validation Steps) in the first round. He then, plays back the recorded script in the subsequent rounds. In this type, the creation, and execution of test scripts are done individually for each test case individually. Testers capture each test step such as browsing, navigation, user inputs, enforcing checkpoints. Testers then play the scripts to carry out the tests.

Modular Driven Framework:

In the modular testing framework, testers create test scripts module wise by breaking down the complete application under test into smaller, independent tests.Abstraction is the concept on which this framework is built. Based on the modules, independent test scripts are developed to test the software. Specifically, an abstraction layer is built for the components to be hidden from the application under tests

In simple words, testers divide the application into multiple modules and create test scripts individually. These individual test scripts can be combined to make larger test scripts by using a master script to achieve the required scenarios. This master script is used to invoke the individual modules to run end to end test scenarios.

The main reason for using this framework is to build an abstraction layer to safeguard the master module from any changes made in individual tests.In this framework, testers write function libraries to use it whenever required. This is AKA modularity framework or module-based framework.

Data- Driven Framework:

The data-driven test automation framework is focused on separating the test scripts logic and the test data from each other.It allows us to create test automation scripts by passing different sets of test data. The test data set is kept in the external files or resources such as MS Excel Sheets, MS Access Tables, SQL Database, XML files, etc., The test scripts connect to the external resources to get the test data.

By using this framework we could easily make the test scripts work properly for different sets of test data.This framework significantly reduces the number of test scripts compared to the module-based framework. This framework gives more test coverage with reusable tests and flexibility in the execution of tests only when required and by changing only the input test data.It is reliable in terms of no impact on tests by changing the test data but it has its own drawbacks such as testers who work on this framework needs to have the hands-on programming knowledge to develop test scripts

Keyword Driven Framework:

It is also known as table-driven testing or action word based testing.
In Keyword-driven testing, we use a table format to define keywords or action words for each function or method that we would execute.It performs automation test scripts based on the keywords specified in the excel sheet.By using this Framework, testers can work with keywords to develop any test automation script, testers with less programming knowledge would also be able to work on the test scripts.

The logic to read keywords and call the required action mentioned in the external excel sheet is placed in the main class. Keyword-driven testing is similar to data-driven testing. Even though to work on this framework doesn't require much programming skills but the initial setup (implement the framework) requires more expertise.

Hybrid Driven Framework:

Hybrid Test automation framework is the combination of two or more frameworks mentioned above. It attempts to leverage the strengths and benefits of other frameworks for the particular test environment it manages. Most of the teams are building this hybrid driven framework in the current market.

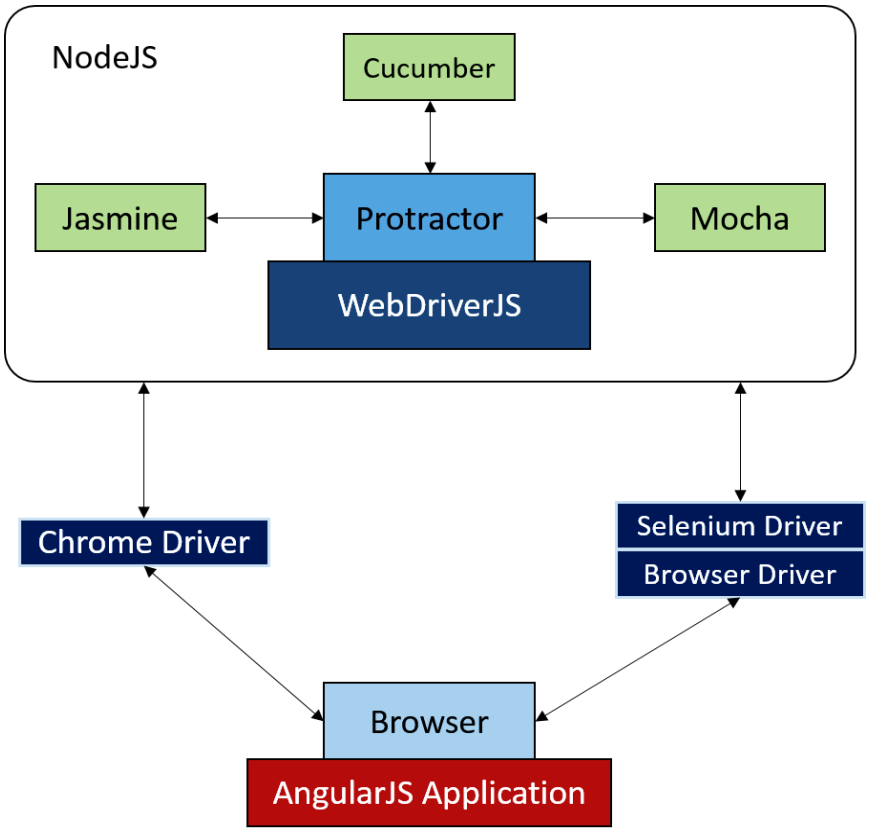
Behavior Driven Development Framework:

The purpose of this Behaviour Driven Development framework is to create a platform that allows everyone (such as Business Analysts, Developers, Testers, etc.,) to participate actively. It requires increased collaboration between Development and Test Teams. It doesn't require the users to be acquainted with a programming language. We use non-technical, natural language to create test specifications. Some of the tools available in the market for Behaviour Driven Development is JBehave, Cucumber, etc.

SOFTWARE TOOLS AND PACKAGES INSTALLED

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.
It also works as a solution integrator that combines powerful technologies such as NodeJS, Selenium, Jasmine, WebDriver, Cucumber, Mocha etc. Along with testing of AngularJS application, it also writes automated regression tests for normal web applications. It allows us to test our application just like a real user because it runs the test using an actual browser.



Setup

Use npm to install Protractor globally with:

```
npm install -g protractor
```

This will install two command line tools, `protractor` and `webdriver-manager`. Try running `protractor --version` to make sure it's working.

The `webdriver-manager` is a helper tool to easily get an instance of a Selenium Server running. Use it to download the necessary binaries with:

```
webdriver-manager update
```

Now start up a server with:

```
webdriver-manager start
```

This will start up a Selenium Server and will output a bunch of info logs.

Software Testing Life Cycle (STLC) is a sequence of different activities performed during the software testing process.

CUCUMBER:

Cucumber is a testing tool that supports Behavior Driven Development (BDD). It offers a way to write tests that anybody can understand, regardless of their technical knowledge. In BDD, users (business analysts, product owners) first write scenarios or acceptance tests that describe the behavior of the system from the customer's perspective, for review and sign-off by the product owners before developers write their codes.

Advantages of Cucumber

1. It is helpful to involve business stakeholders who can't easily read code
2. Cucumber Testing tool focuses on end-user experience
3. Style of writing tests allow for easier reuse of code in the tests
4. Quick and easy set up and execution
5. Cucumber test tool is an efficient tool for testing

SETUP:

Cucumber should be installed in the same place as Protractor - so if protractor was installed globally, install Cucumber with -g.

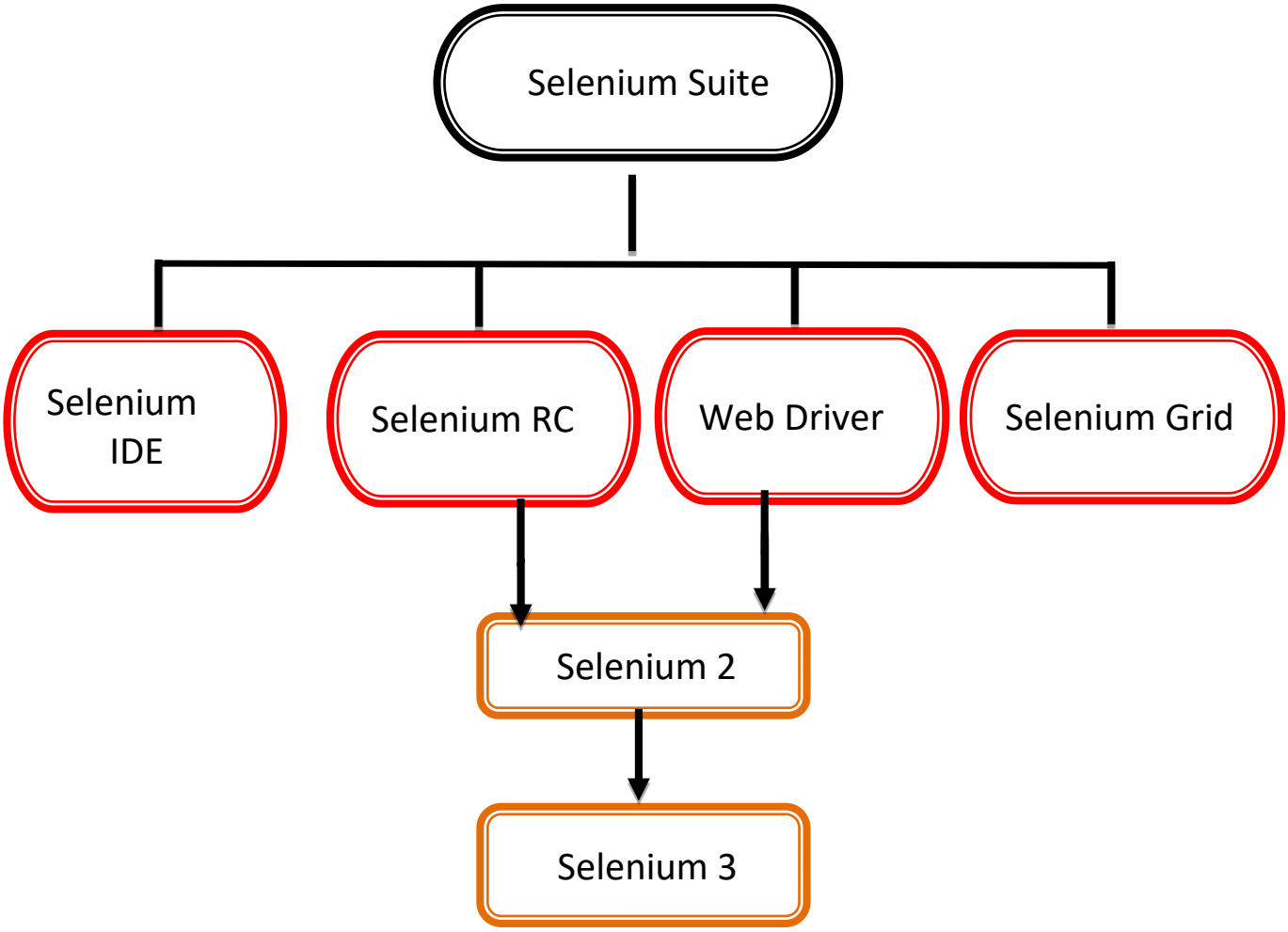
```
npm install -g cucumber

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

SELENIUM

Selenium is a portable framework for testing web applications. It is a popular open source web-based automation tool used to validate web applications across different browsers and platforms. We can use multiple programming languages like Java, C#, Python etc. to create Selenium Test Scripts. The tests can then be run against most modern web browsers. Selenium runs on Windows, Linux and macOS. Selenium was released under the Apache License 2.0.

Selenium Software is not just a single tool but a suite of software, each piece catering to different Selenium QA testing needs of an organization. The complete list of tools is as followed:

- Selenium Integration Development Environment (IDE)
- Selenium Remote Control (RC)
- Web Driver
- Selenium Grid



Selenium Integration Development Environment (IDE)

Selenium IDE is a complete integrated development environment (IDE) for Selenium tests. It is implemented as a Firefox Add-On and as a Chrome Extension. It allows for recording, editing and debugging of functional tests.

Scripts may be automatically recorded and edited manually providing auto completion support and the ability to move commands around quickly. Scripts are recorded in Selenese, a special test scripting language for Selenium. Selenese provides commands for performing actions in a browser.

Why choose Selenium IDE?

- To create tests with little or no prior knowledge in programming.
- To create simple test cases and test suites that you can export later to RC or WebDriver.
- To test a web application against Firefox and Chrome only.

Selenium WebDriver

Selenium WebDriver is the successor to Selenium RC. Selenium WebDriver accepts commands and sends them to a browser. This operation is implemented through a browser-specific browser driver, which sends commands to a browser and retrieves results.
Selenium WebDriver does not need a special server to execute tests. The WebDriver directly starts a browser instance and controls it. However, Selenium Grid can be used with WebDriver to execute tests on remote systems. WebDriver uses native OS level functionality rather than browser-based JavaScript commands to dive the browser thus bypassing problems with subtle difference between native and JavaScript commands including security restrictions.

Why choose Selenium WebDriver?

- To use a certain programming language in designing our test case.
- To test applications that are rich in AJAX-based functionalities.
- To create customized test results.

CHAI

- Chai is a BDD / TDD assertion library for node and the browser
- Chai can be paired with any javascript testing framework (for instance Mocha)
- Chai has several interfaces that allow the developer to choose. The chain-capable BDD styles provide an expressive language & readable style, while the TDD assert style provides a more classical feel. BDD (more popular) - expect, should TDD - assert

Installation

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

Assertion Styles

Chai provides the following assertion styles:

1. Assert style

```
var assert = require('chai').assert;
var foo = "bar";

assert.typeOf(foo, 'string');
assert.equal(foo, 'bar');
```

2. Expect style

```
var expect = require('chai').expect;
var foo = "bar";

expect(foo).to.be.a('string');
expect(foo).to.equal('bar');
```

LANGUAGES USED

JAVASCRIPT

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

Important features of JavaScript:

- Light weight scripting language
- Dynamic typing
- Object-oriented programming support
- Functional style
- Platform independent
- Prototype-based
- Interpreted language
- Async processing
- Client-side validation

Advantages of JavaScript

The merits of using JavaScript are −

- **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

GHERKIN

Gherkin is a business readable language which helps you to describe business behaviour without going into details of implementation. It is a domain specific language for defining tests in Cucumber format for specifications. It uses plain language to describe use cases and allows users to remove logic details from behaviour tests. The text in Gherkin language acts as documentation and skeleton of our automated tests.

Gherkin Syntax

Gherkin is line-oriented language just like YAML and Python. Each line called step and starts with keyword and end of the terminals with a stop. Tab or space are used for the indentation.

In this script, a comment can be added anywhere you want, but it should start with a # sign. It read each line after removing Gherkin's keywords as given, when, then, etc.

```
Feature: Title of the Scenario
Given [Preconditions or Initial Context]
When [Event or Trigger]
Then [Expected output]
```

A Gherkin document has an extension .feature and simply just a test file with a fancy extension. Cucumber reads Gherkin document and executes a test to validate that the software behaves as per the Gherkin syntax.

Important Terms used in Gherkin

Feature: The file should have extension .feature and each feature file should have only one feature. The feature keyword being with the Feature: and after that add, a space and name of the feature will be written.

Background: Background keyword helps you to add some context to the scenario. It can contain some steps of the scenario, but the only difference is that it should be run before each scenario.

Scenario: Each feature file may have multiple scenarios, and each scenario starts with Scenario: followed by scenario name.

Given: The use of Given keyword is to put the system in a familiar state before the user starts interacting with the system. However, you can omit writing user interactions in Given steps if Given in the "Precondition" step.

When : When the step is to define action performed by the user.

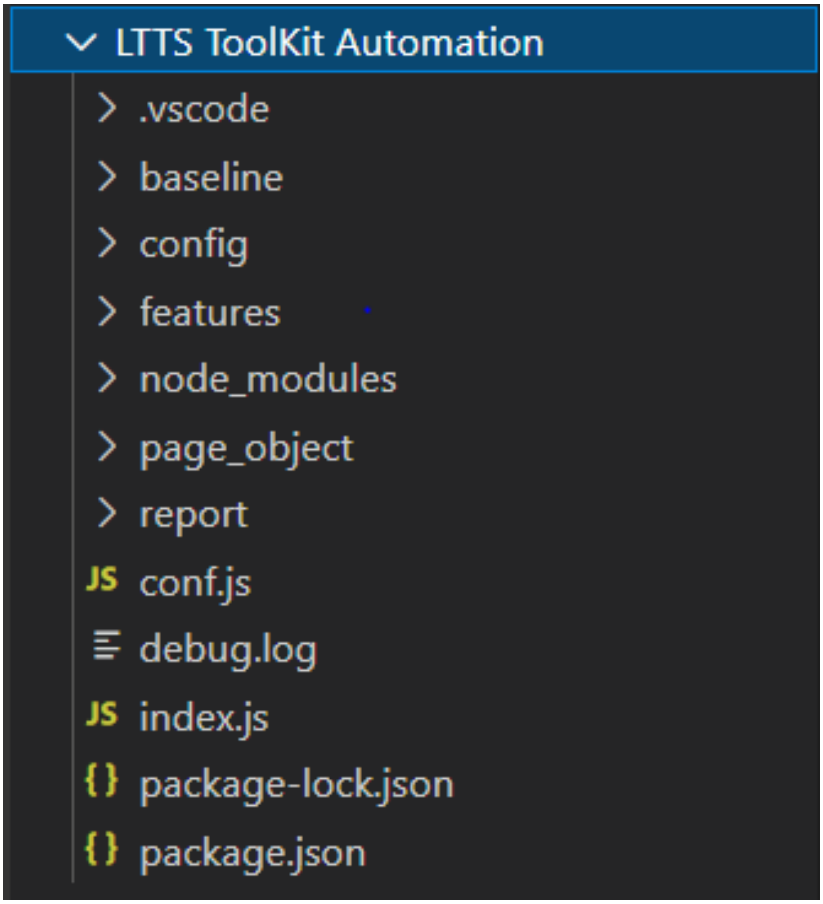
Then : The use of 'then' keyword is to see the **outcome** after the action in when step. However, you can only verify noticeable changes.

Advantages of Gherkin

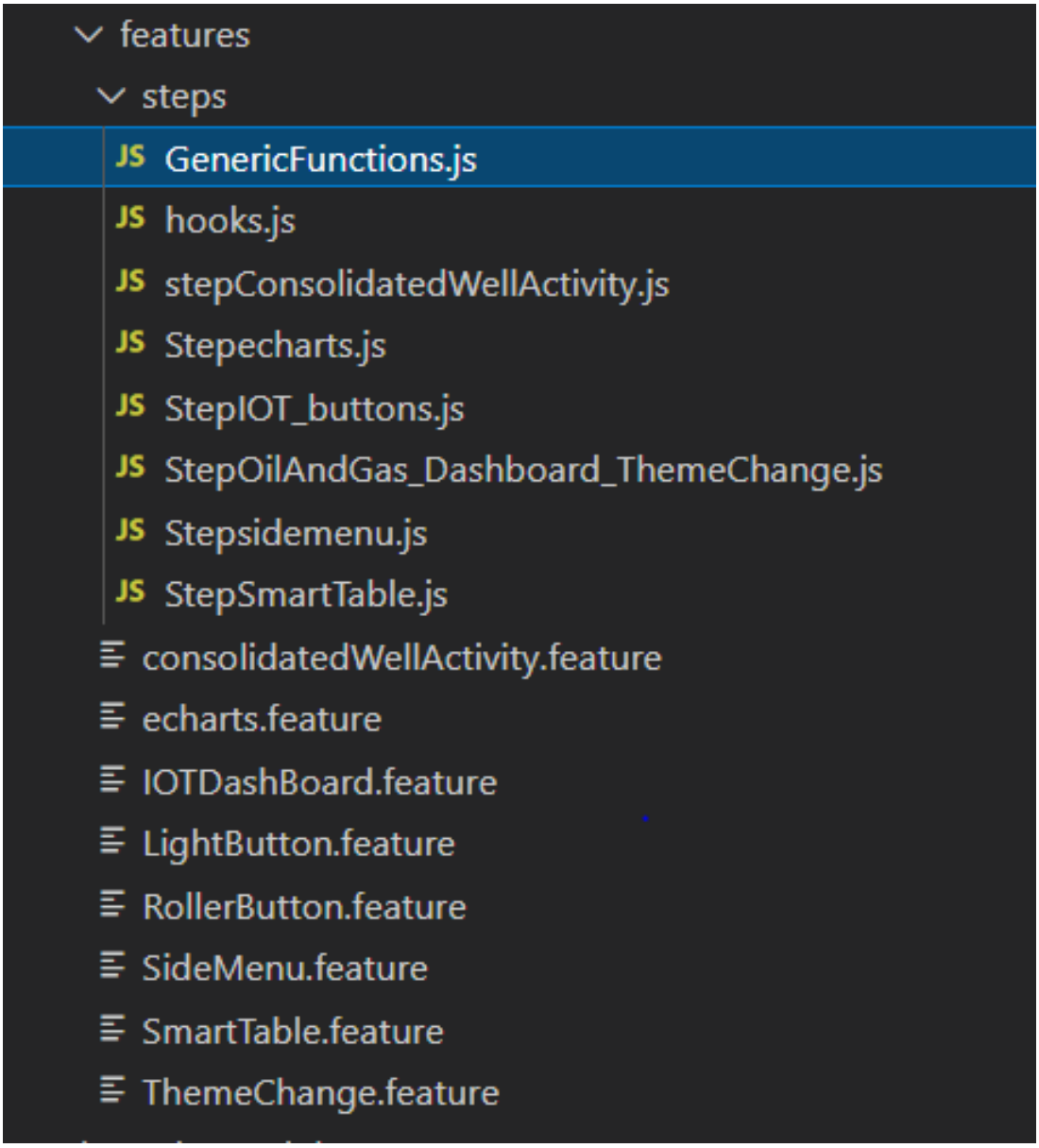
- Gherkin is simple enough for non-programmers to understand
- Programmers can use it as a very solid base to start their tests
- It makes User Stories easier to digest
- Gherkin script can easily understand by business executives and developers
- Gherkin Testing targets the business requirements
- A significant proportion of the functional specifications is written as user stories
- You don't need to be expert to understand the small Gherkin command set
- Gherkin Test cases link acceptance tests directly to automated tests
- Style of writing tests cases are easier to reuse code in other tests

IMPLEMENTATION

Project Folder Structure



Features tested



Feature Files

For echartS:

LTTS ToolKit Automation > features > echarts.feature

```
1   Feature: Verify images of echarts
2
3   Scenario: Verify echarts
4   Given I navigate to echarts page
5   When I click on Line two
6   Then Correct garph is displayed
7
```

For consolidated well activity:

LTTS ToolKit Automation > features > consolidatedWellActivity.feature

```
1   Feature: Verify values displayed in consolidated well activity table
2
3   Scenario: Verify values displayed in the well activity table by month
4   Given The well activity table is displayed
5   Then The values are displayed by month
6
7   Scenario: Verify values displayed in the well activity table by week
8   Given The option to display values by week is available
9   When I select option by week
10  Then Values are displayed by week
```

Step Definitions:

For echarts:

```
Given('I navigate to echarts page', async function () {
        homepage.getCharts().click();
        browser.sleep(5000);
        expect(homepage.geteCharts().isPresent()).to.eventually.equal(true);
        await homepage.geteCharts().click();
      });

When('I click on Line two', async function () {
        expect(homepage.LineGraphsExist().isPresent()).to.eventually.equal(true);
        await homepage.geteCharts().click();

      });

Then('Correct garph is displayed', async function () {
        expect(homepage.getLineGraph().isPresent()).to.eventually.equal(true);
        await homepage.getLineGraph().click();

      });
```

For consolidated well activity:

```
Given('The option to display values by week is available',async function () {
    //await homepage.scrollTo().click();
    await expect(homepage.getChangeOption().isPresent()).to.eventually.equal(true);
    await homepage.getChangeOption().click();

  });


  When('I select option by week',async function () {
    await expect(homepage.getByWeek().isPresent()).to.eventually.equal(true)
    await homepage.getByWeek().click();
  });


  Then('Values are displayed by week', async function () {
      console.log("\n");
      days = ["Mon", "Tue","Wed", "Thu", "Fri", "Sat"]
    await homepage.getVisitedDate().then(function(items){
        for(i=0;i<6;i++)
        {
            var weektext = items[i].getText();
            weektext.then(elementText => {
                console.log(elementText);
            })
        expect(items[i].getText()).to.eventually.equal(days[i]);

        }
       });
  });
```

Page Object:

For Echarts :

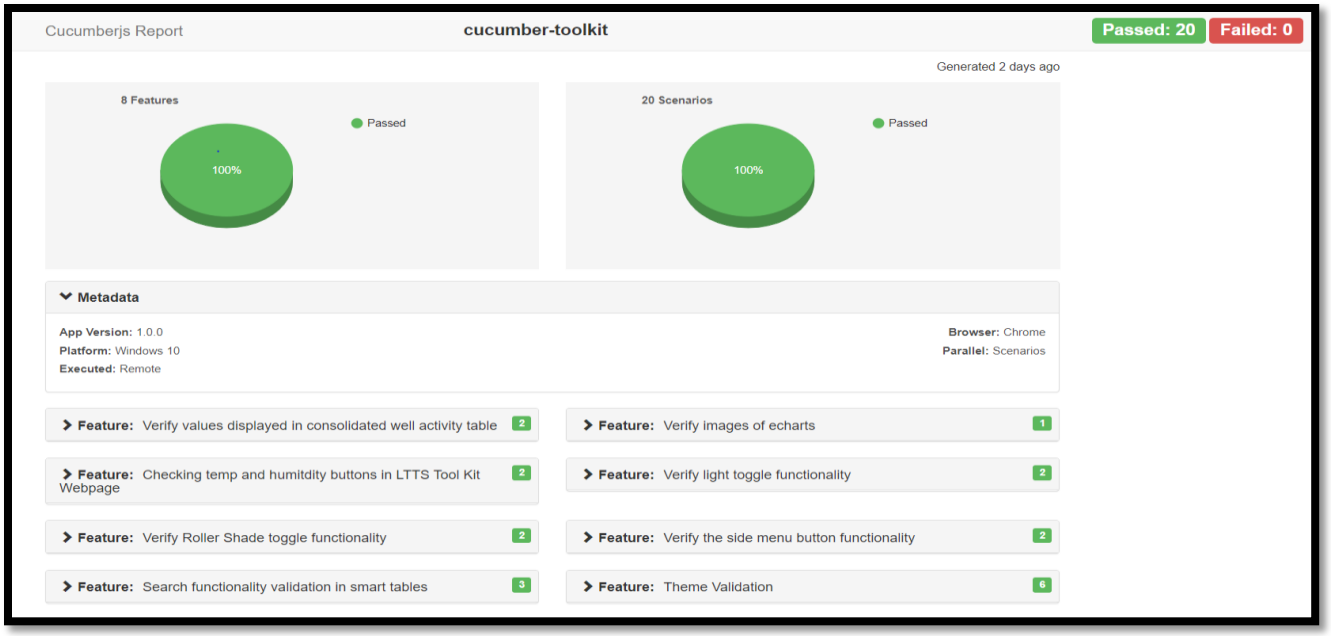
pageobjectmod > chartpom.js > Charts

```
1   //admin Chart POM
2
3   var Charts = function()
4   {
5       this.Line = element(by.xpath("//*[text()='Line']"))
6       this.LineGraph = element(by.xpath("(//*[@class='echart'])[3]"))
7       this.charts = element(by.className('menu-title ng-tns-c108-6'))
8       this.echarts = element(by.className('menu-title ng-tns-c108-7'))
9
10
11      this.getCharts = function()
12      {
13          return this.charts;
14      }
15      this.geteCharts = function()
16      {
17          return this.echarts;
18      }
19      this.LineGraphsExist = function()
20      {
21          return this.Line;
22      }
23
24      this.getLineGraph = function()
25      {
26          return this.LineGraph;
27      }
28
29
30  }
31
32  module.exports = new Charts();
```

Test result obtained:

```
WARNING: See https://github.com/lorenwest/node-config/wiki/Strict-Mode
[19:53:48] I/launcher - Running 1 instances of WebDriver
[19:53:48] I/direct - Using ChromeDriver directly...

DevTools listening on ws://127.0.0.1:50281/devtools/browser/8566eb4d-7045-4ed6-9150-4a48a07fd22d
[12536:23496:0428/195357.791:ERROR:device_event_log_impl.cc(214)] [19:53:57.791] USB: usb_device_handle_win.cc:1054 Failed to read descriptor from node connection:
A device attached to the system is not functioning. (0x1F)
..DATA= Consolidated Well Activity
.....................................................................ID = 5
First Name= Jack
Last Name= Sparrow
User Name= @jack
Mail ID= jack@yandex.ru
Age= 30
..........ID = 5
First Name= Jack
Last Name= Sparrow
User Name= @jack
Mail ID= jack@yandex.ru
Age= 30
..........First Name= Roronoa
Last Name= Zoro
User Name= @jack
Mail ID= jack@yandex.ru
Age = 30
.......................ID = 5
.........

20 scenarios (20 passed)
72 steps (72 passed)
0m52.391s
[19:55:20] I/launcher - 0 instance(s) of WebDriver still running
[19:55:20] I/launcher - chrome #01 passed
PS C:\Users\USER\Desktop\LTTS ToolKit Automation\LTTS ToolKit Automation>
```

Test Summary Report (HTML):



PROJECT OUTCOMES

- Final Aim of project is achieved
- Able to leverage open source tools and understanding of end to end process involved in test automation of an applitcation
- Able to identitfy web elements uniquely and develop page object model repository
- Able to create test scripts using JavaScript
- Able to Generate automated HTML test summary report.

CHALLENGES FACED

- Installation of required tools
- Technical Challenges

CHALLENGES OVERCOME

- Required tools were installed.
- And team work by proper coordination and sharing the work.

CONTRIBUTIONS

INDIVIDUAL CONTRIBUTIONS:

Mithun M R (99003716)

- Worked on features : Iot- Dashboard , Roller shades Toggle and Light Toggle

Shaik Rehana (99003549)

- Worked on features : Theme, Echarts and Consolidate well activity

Sneha Anand (99003525)

- Worked on features : Smart Table , Side menu

TEAM CONTRIBUTIONS:

- We all worked together on the complete setup of framework and overall testing of the mentioned features.

SHAREPOINT LINK:

SHADOW PROJECT IMPLEMENTATION