**<Project> Test Automation**

Strategy Design Document

Contents

[1. Introduction 4](#_Toc471388541)

[1.1. Overview 4](#_Toc471388542)

[1.2. Intended Audience 4](#_Toc471388543)

[2. Test Automation Objectives 5](#_Toc471388544)

[2.1. Angularjs based AUT 5](#_Toc471388545)

[2.2. Testing in an Agile Methodology Maintenance Pattern (Sprints) 5](#_Toc471388546)

[2.3. Multiple Browser Testing 5](#_Toc471388547)

[2.4. Scenarios with Impersonate Feature 5](#_Toc471388548)

[3. Choice of Framework and Test Automation Tools 6](#_Toc471388549)

[3.1. Test Automation Tool – Selenium (Web Driverjs) 6](#_Toc471388550)

[3.2. Test Automation Base Framework – Protractor [Cucumber] 6](#_Toc471388551)

[4. Framework Architecture 7](#_Toc471388552)

[4.1. .feature Files 8](#_Toc471388553)

[Feature 9](#_Toc471388554)

[Scenario 10](#_Toc471388555)

[Steps 10](#_Toc471388556)

[Scenario Outline 10](#_Toc471388557)

[Examples 10](#_Toc471388558)

[4.2. pagesObject 10](#_Toc471388559)

[4.3. stepDefinition 11](#_Toc471388560)

[4.4. testData.json 12](#_Toc471388561)

[4.5. utils 13](#_Toc471388562)

[4.6. Protractor-cucumber-framework 15](#_Toc471388563)

[4.7. config.js 15](#_Toc471388564)

[4.8. gulpfile.js 16](#_Toc471388565)

[4.8. testSuite.js 16](#_Toc471388566)

[5. Test Automation Standards and Procedures 16](#_Toc471388567)

[5.1. Test Script Coding 17](#_Toc471388568)

[5.2. Page Object Creation 17](#_Toc471388569)

[5.4. Test Data creation 17](#_Toc471388570)

[5.5. Test Method debugging 18](#_Toc471388571)

[5.6. Test Script Maintenance 18](#_Toc471388572)

[5.7. Commit code to SVN/Any Provided Repository 18](#_Toc471388573)

[6. Test Automation Framework Usage 19](#_Toc471388574)

[6.1. Runner.js configuration 20](#_Toc471388575)

[6.3. Test Automation Framework Execution 20](#_Toc471388576)

[7. Reporting 21](#_Toc471388577)

[7.1. Protractor-Cucumber Report 21](#_Toc471388578)

[7.4. Snapshots of test script failures 22](#_Toc471388579)

[8. Manual vs. Automation (Per Cycle) 23](#_Toc471388580)

[9. ROI/CBA (Cost Benefit Analysis) 23](#_Toc471388581)

[10. Estimated Cost of maintenance 23](#_Toc471388582)

[11. Appendix 23](#_Toc471388583)

# 1. Introduction

The primary goal of this document is to describe the Design Strategy implemented for <Project> Test Automation, Automation Framework Architecture, and Strategy for Execution Control, Test Data Storage Design and Reporting Format(s).

## 1.1. Overview

This document presents a complete understanding of the Automation Framework used for Automation of <Project> Regression Test Suites. The Framework Architecture Document presents detailed description of the following:

1. Test Automation Challenges
2. Choice of Framework and Test Automation Tools
3. Framework Architecture
4. Test Automation Standards and Procedures
5. Test Automation Framework Usage
6. Reporting

## 1.2. Intended Audience

The Key Stakeholders listed below would use this document for making decisions related to various:

1. Test Automation Team
2. <Project> Core Team

# 2. Test Automation Objectives

The test automation framework is designed to cater the below mentioned Objectives and requirements:

* To Test the application developed using AngularJS technology.
* To test the application maintained using Agile Methodology (Sprints).
* To test the application for Multiple Browsers.
* To test Business scenarios with Impersonate Features.

## 2.1. Angularjs based AUT

<Project> ServiceNow has implemented AngularJS, a structural framework for dynamic web apps. It extends HTML *[with new attributes and elements]* through the use of [directives](http://docs.angularjs.org/guide/directive%22directives%22). AngularJS directives uses elements and attributes that define data bindings and the behavior of presentation components. The UI elements are loaded on the HTML page based on user’s interaction and are dynamically refreshed. The test automation framework is required to handle these delays / dynamic refresh and not report “false failure” that arises.

## 2.2. Testing in an Agile Methodology Maintenance Pattern (Sprints)

Agile Methodology is being followed for <Project> ServiceNow maintenance. The automation regression testing is required to test the application sprint wise for Enhancement/Upgrade release, Issues/Tickets raised and Custom Applications Post Migration.

## 2.3. Multiple Browser Testing

The <Project> ServiceNow is a web based solution and the usage will depend on the browser choice of business user. Hence the framework had to be designed to scale up for multiple browsers. Presently the framework is configured for below mentioned browsers:

* Mozilla Firefox (version 15 and above)
* Google Chrome (version 22 and above)
* Microsoft Internet Explorer (version 8 and above)

## 2.4. Scenarios with Impersonate Feature

The process flow of <Project> ServiceNow comprises of multiple actors (Impersonate) to perform application workflow steps in a given business use case. The framework is required to enable execution of these workflow scenarios based on the actor in the given workflow.

# 3. Choice of Framework and Test Automation Tools

The choice of test tools to be used for creating the test automation framework was based on the challenges/requirements described in the previous section.

As a result below mentioned test tools were identified:

* Test Automation Tool - Selenium (Web DriverjS)
* Test Automation Framework – Protractor [Cucumber]

## 3.1. Test Automation Tool – Selenium (Web Driverjs)

Selenium (Web Driverjs) is used to test AngularJs based applications, it returns Promises for all of its interactions with a browser. Promises are method of dealing with AngularJs asynchronous code. The programming language used for framework creation and test automation script creation is JavaScript. Selenium Web Driverjs has inherent power to build a common Test Automation Script that be used to test the application on multiple browsers.

## 3.2. Test Automation Base Framework – Protractor [Cucumber]

Protractor [Cucumber] is a test automation framework that is used for <Project> test automation. This is a Nodejs based solution which provides integration with the selected test tool (Selenium Web Driverjs)

# 4. Framework Architecture

The test automation framework structure and components are shown below:



Figure 1: <Project> ServiceNow Test Automation Framework folder structure

The above mentioned folder structure shows the following components of the automation framework:

* .feature files: the files contain executable specifications written in a plain simple English language called Gherkin.
* pageObject: The folder contains the Page Object Model (POM) (.js files), to map the <Project> ServiceNow AUT elements and write the functions to perform actions.
* stepDefinition: the folder contains the Test Automation Scripts in (.js files) using cucumber syntax. Also contains the export and required to connect POM and stepDefinition.
* testData.json: the folder contains configuration and Dynamic data required to execute the Test Automation Framework.
* Utils: the folder contains all the support (.js files) required by the Test Automation Framework and Scripts.
* Protractor-cucumber-framework – the folder contains all in build dependencies of framework in (.js files) format.
* Config.js: the file tells protractor where the test files are, and sets the communication channel with Selenium Server. The browser to be used for Test Suite execution.
* Gulpfile.js: the file starts execution from command prompt by triggering the Config.js file.
* testSuite.js: the file to organize the test for execution for specific goal.

## 4.1. .feature Files

The folder “<Name>-application” contains all feature files for <Project> ServiceNow test automation. Below shown is the folder structure with feature files.

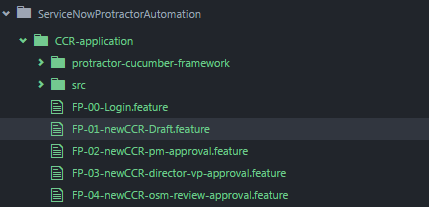


Figure 2: Feature File folder structure

Each feature file is a cucumber file, which contains executable specification written in plain-text English known as Gherkin language.

The file snippet shown below depicts a sample structure of a feature file.

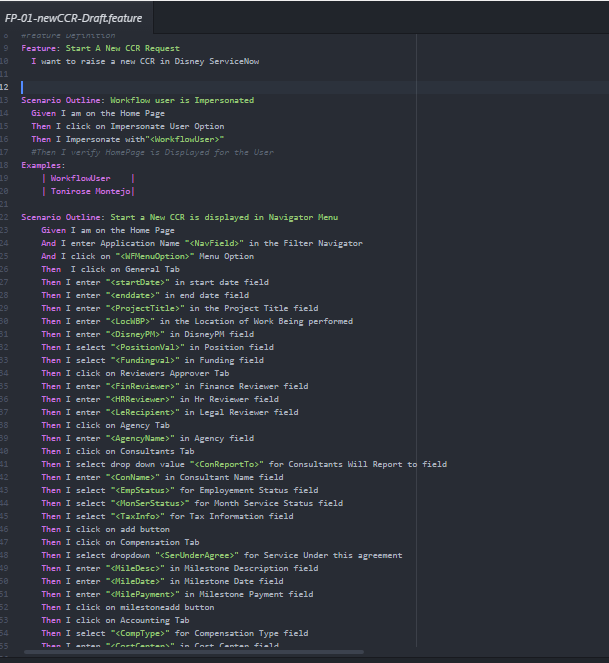


Figure 3: Feature File Statements

Gherkin *keywords used*

### Feature

A .feature file describe a single feature of the system, or a particular aspect of a feature. It's just a way to provide a high-level description of a software feature, and to group related scenarios.

### Scenario

A scenario is a *concrete example* that *illustrates* a business rule. It consists of a list of [steps](https://cucumber.io/docs/reference#steps).

### Steps

A step typically starts with Given, When, Then, And, But. Cucumber does not differentiate between the keywords, but choosing the right one is important for the readability of the scenario as a whole.

### Scenario Outline

When we have a complex business rule with severable variable inputs, we user scenario outline along with section to provide field level test data.

### Examples

A Scenario Outline section is always followed by one or more Examples sections, which are a container for a table. The table has a header row corresponding to the variables in the Scenario Outline steps. Each of the rows below will create a new Scenario, filling in the variable values.

## 4.2. pagesObject

The “pagesObject” folder consists of (.js files) that are logical representation of each HTML page that is part of <Project> ServiceNow application. This files would contain the web elements present on application page and the business functionalities that the page provides. A sample code snippet is shown below:

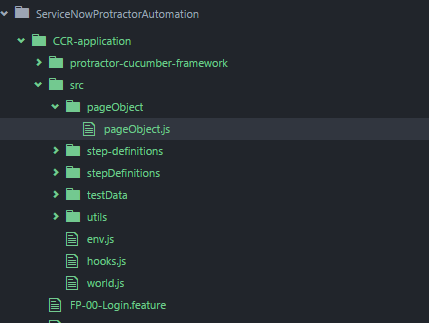


Figure 4: Page Object folder structure

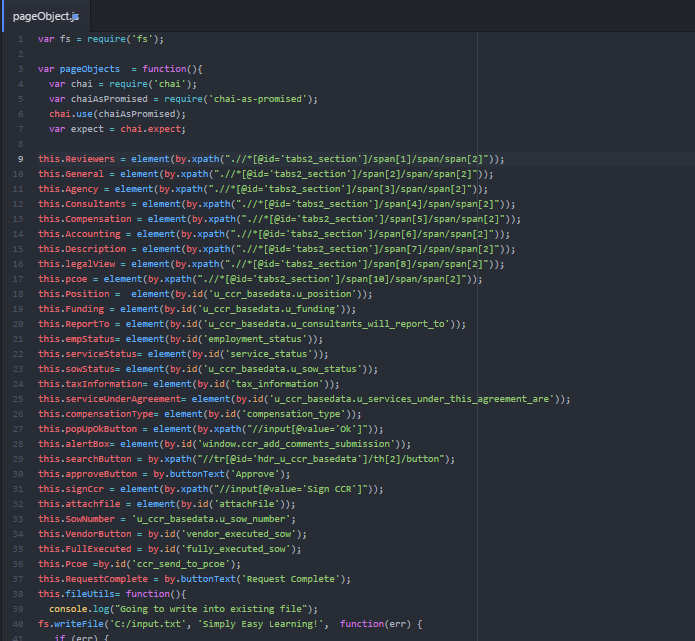


Figure 5: Page Object Sample Code

## 4.3. stepDefinition

The support folder contains all Test Automation Scripts in (.js files) using cucumber syntax. Also contains the “export and required” which connects POM and stepDefinition.

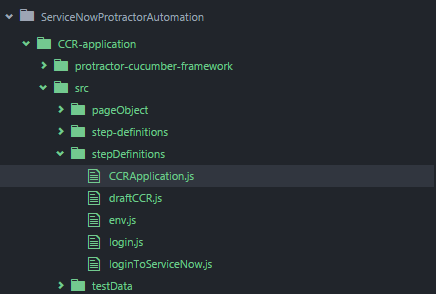


Figure 6: Step Definition folder structure

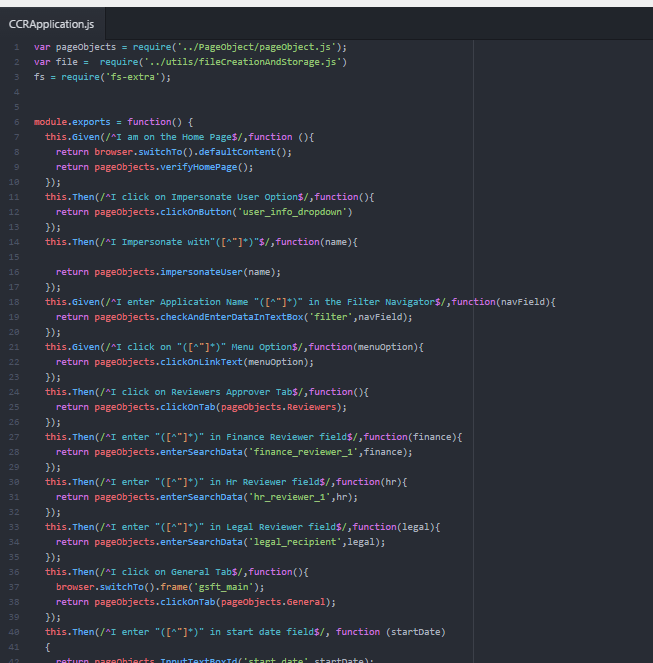


Figure 7: Step Definition Sample Code

## 4.4. testData.json

The support folder contains all configuration and Dynamic data required to execute the Test Automation Framework. Dynamic data will be URL and one time login to <Project> ServiceNow with Admin user.

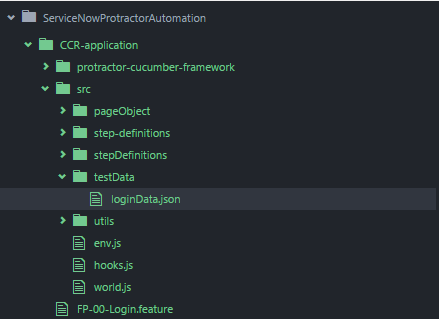


Figure 8: Test Data folder structure

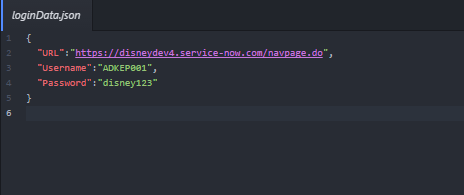


Figure 9: Test Data Sample code

## 4.5. utils

The folder contains all the dependencies (.js files) required by the Test Automation Framework and Scripts. The dependencies will be Application dependent, Global functions and set of functions need to run before and after feature file execution.

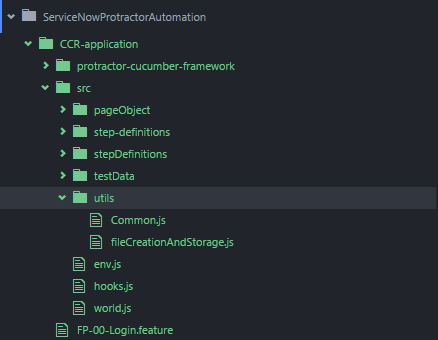


Figure 10: Utils folder structure



Figure 11: Utils Sample Code

## 4.6. Protractor-cucumber-framework

The folder contains all in build dependencies of framework Protractor and Cucumber. These dependences will be downloaded with node.js and will be made the part of the structure. The ‘framework’ property is set to custom and the ‘frameworkPath’ is set to 'protractor-cucumber-framework' to the [config file](https://github.com/angular/protractor/blob/master/spec/cucumberConf.js). Cucumber ‘format’ options will also be set to summary to the Config.js file.

## 4.7. config.js

The configuration file used by Protractor-Cucumber framework for establishing connection with selenium standalone server, setting browser options, path to feature files and all required .js file paths are mentioned in Config.js files.

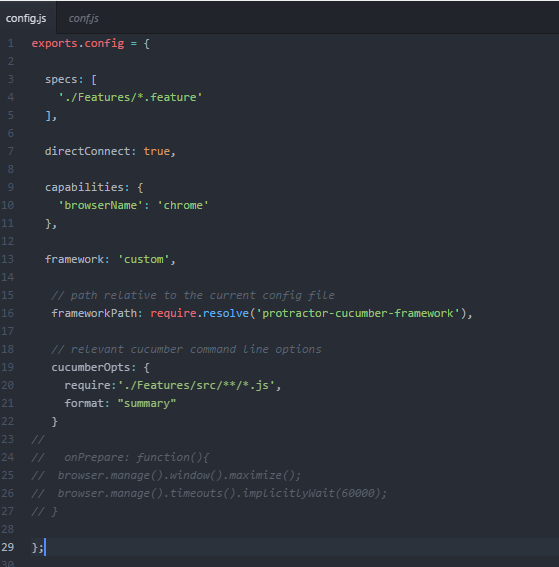


Figure 12: Sample config file

## 4.8. gulpfile.js

This file is used as build manager it starts the script execution from command prompt by triggering the Config.js file.Config.js: the file tells protractor where the test files are, and sets the communication channel with Selenium Server. The browser to be used for Test Suite execution.

Built tool Gulp will be used to run Gulp-protractor in order to create different setups for environment [Dev,Testing. etc], helps to run test or Suite by running a single task, instead of creating different Config files for different environments.



Figure 8: Sample format of Gulp file

For criteria based execution, the <method-selectors> tag will be used for creating the execution criteria. Combinations of groups defined for test methods in the @Test annotations will be used.

## 4.8. testSuite.js

The file to organize the test for execution for specific goal.

# 5. Test Automation Standards and Procedures

The Test Automation Scripter needs to understand the <Project> ServiceNow test automation framework. The automation scripter has to adhere to the rules of the framework for code uniformity.

The Test Automation Scripter will use this framework to automate the manual Regression test cases into automation test scripts. There are broadly 7 activities need to be performed as part of automation scripting:

* Test Script (Test Method) Coding
* Feature file creation
* Stub creation
* Test Steps (.js file) creation
* Page Object (.js file) creation
* Test Data creation
* Test Script Maintenance
* Commit code to SVN/ or any Repository

## 5.1. Test Script Coding

At the time of creating test script following standards need to be followed strictly to maintain the uniformity of the code

* The Feature File name should be unique and should represent the corresponding Use case/Story/Defect/Test Scenario/Test Case.
* Every Application will have a separate script Folder structure.
* Every Application will have a single PageObject Model (.js file).
* Dynamic data and configuration data will be provided in .json file under testData folder.
* Static Field level data will be provided in cucumber file.
* Every Application will have a single stepDefinition (.js file).
* The test method name should follow the format <automation script id><\_><test script description>.

## 5.2. Page Object Creation

The Automation scripter has to create new page object for every new page that is added in the <Project> ServiceNow application as part of sprint release by the development team.

For any change on the existing pages of the application, the corresponding page object class needs to be modified accordingly, for its web elements and/or for its services (business functionalities provided by the page).

The name of identifiers in the page object should uniquely point to the corresponding web element on the application page. This identifier name should match the web element name as much as possible

## 5.4. Test Data creation

The test data for the automation test scripts would reside in the corresponding Feature Files that are required for the current test script. The Dynamic Test Assertion data should be inserted in the .json file for the corresponding automation application suite.

## 5.5. Test Method debugging

The debugging of test script should be performed before a test script is included in the final Regression Test suite. Following features are provided for debugging of a test script:

1. Feature files (.features file)
2. Protractor-Cucumber inbuild report

## 5.6. Test Script Maintenance

The test scripts that fail in sprints, future to which it was created, need to be modified according to the new application release. For such cases all the above 5 points need to be followed before it is again included in the regression test suite.

## 5.7. Commit code to SVN/Any Provided Repository

Once the test scripts are complete, the new code has to be committed to the QA SVN repository using Tortoise SVN. The Repository address for Test Automation is [http://172.17.24.159:8080/svn/E15933/PDLC/Coding/Automation Script/Regression](http://172.17.24.159:8080/svn/E15933/PDLC/Coding/Automation%20Script/Regression)

# 6. Test Automation Framework Usage

The SPTN test automation framework will be executed in two modes:

1. Auto Scheduled Mode
2. Manually Triggered Mode

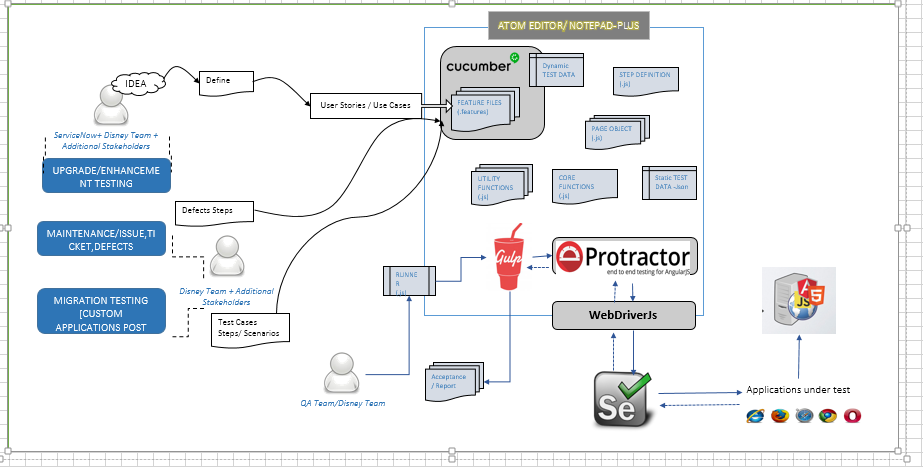


Figure 15: Process Diagram of test automation framework

In both above modes the following activities need to be performed manually before the execution starts

* Runner.js - Setting the execution criteria for current framework execution

## 6.1. Runner.js configuration

Build.xml contains properties that are configurable as shown in the image:

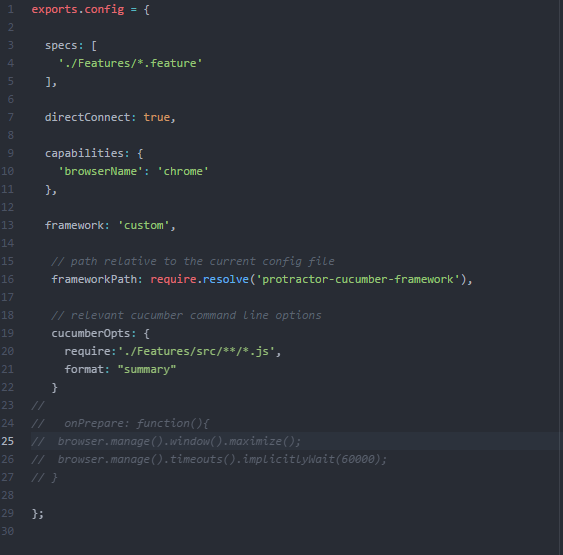


Figure 16: First set of configuration properties in Config.js

The change only needs to be done in the “Automation\_Execution” properties during the test automation framework execution phase.

Once the “Automation\_Execution” properties are updated, the below mentioned properties should be updated /verified so as to point to “Automation\_Execution” properties

The updated Runner should be committed to the SVN repository for Jenkins job to pick up the latest script for regression test suite execution

## 6.3. Test Automation Framework Execution

The location of Config.js file need to be accessed from command prompt. This is configured for manual execution needs to be executed which will start the test automation framework.

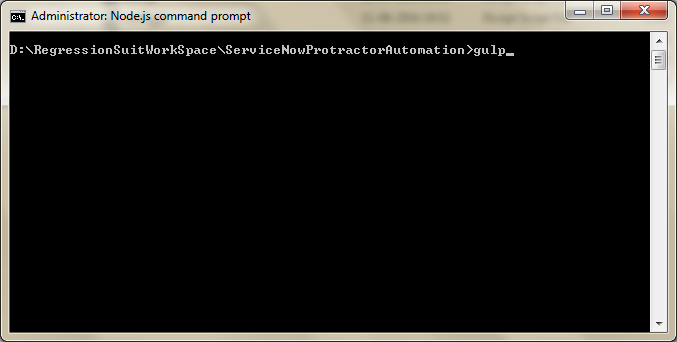


Figure 19: Command prompt from where the test automation framework will be executed

# 7. Reporting

The execution results are stored in following 4 modes:

* Protractor-Cucumber reports (html report)
* Snapshots of test script failures and Pass

## 7.1. Protractor-Cucumber Report

The execution reports would be saved on the remote server a structure as shown below:

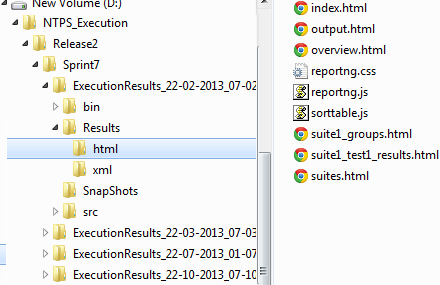
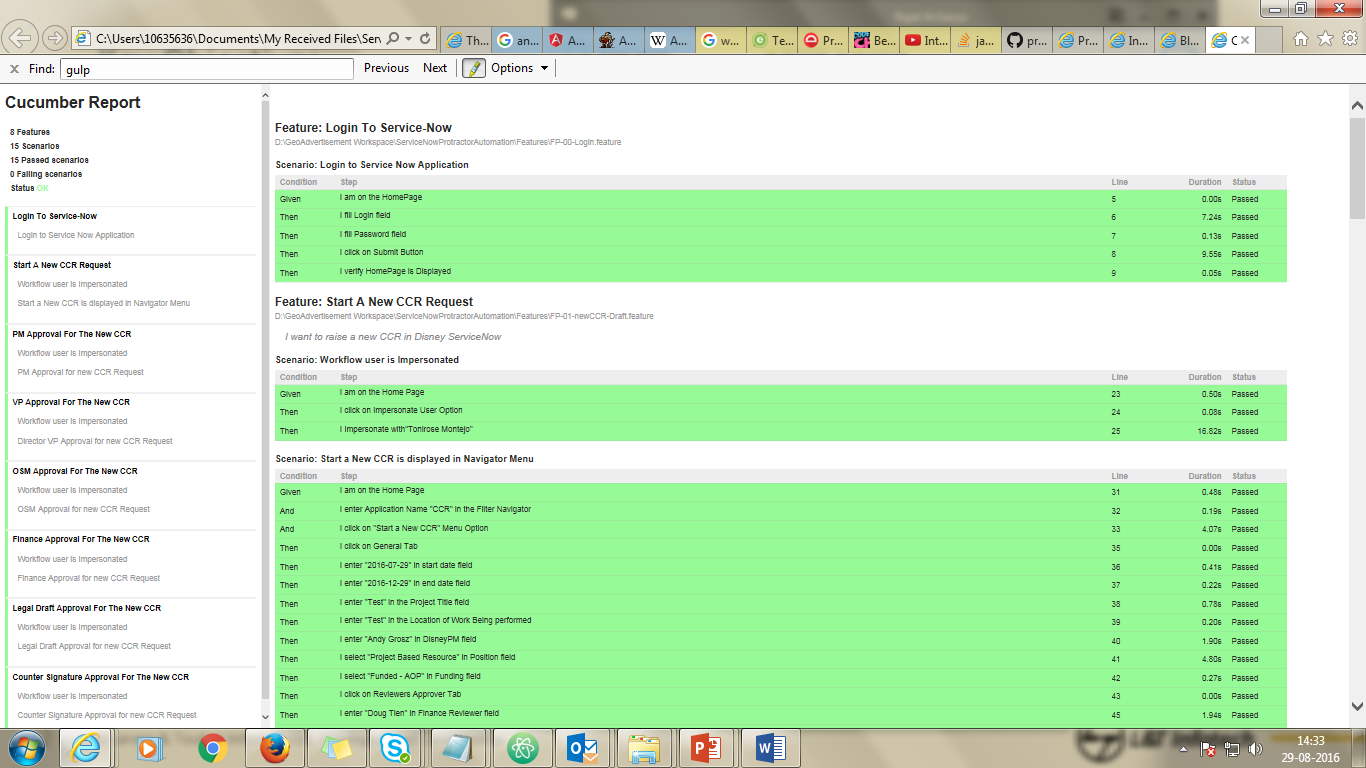


Figure 20: Folder structure of Current Execution folder and location of reports

By opening index.html a complete summary of the execution would be displayed as shown below:



The same result can be viewed using the execution criteria that was used for regression test suite execution.

## 7.4. Snapshots of test script failures

The test script failures are captured in snapshots and saved on the remote server in the shown folder structure:

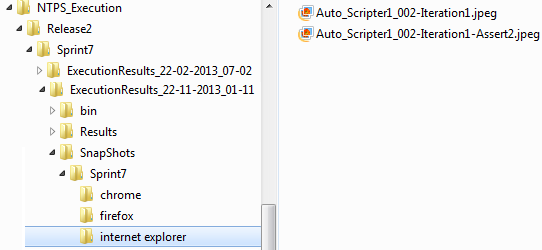


Figure 21: Folder structure of current execution and location test failure snapshots

# 8. Manual vs. Automation (Per Cycle)

The execution results are stored in following 4 modes:

* Protractor-Cucumber reports (html report)
* Snapshots of test script failures and Pass

# 9. ROI/CBA (Cost Benefit Analysis)

# 10. Estimated Cost of maintenance

# 11. Appendix

The list of softwares used while creating the complete workflow of Automation Framework, and also while creating Automation Test Scripts:

| **Setups/Plugins/Folders/Drives** | **Setup Available** | **Admin Access  (Reqd)** | **Env var  Setting (Reqd)** | **Description** |
| --- | --- | --- | --- | --- |
| Mozilla Firefox 15+ | Yes | Yes | No | Browser used to run test |
| Internet Explorer 8+ | Yes | Yes | No | Browser used to run test |
| Google Chrome 22+ | Yes | Yes | No | Browser used to run test |
| selenium-server-standalone-2.33.0.jar | Yes | No | No | Selenium server used to work as Hub/Nodes for Selenium2 Code to execute |
| chromedriver\_win32\_2.0.zip | Yes | No | No | Driver to enable Selenium2 to interact with Chrome |
| selenium-java-2.33.0.zip | Yes | No | No | Selenium2 API - Used as the main/basic test tool for Automation in the project |
| IEDriverServer\_x64\_2.33.0.zip | Yes | No | No | Driver to enable Selenium2 to interact with IE |
| Selenium IDE for Mozilla Firefox | No | Yes | No | Used for Object recognition/Record n Playback in Mozilla |
| Firebug plug in for Mozilla Firefox | No | Yes | No | Used for Object recognition in Mozilla Firefox |
| JDK7 | Yes | Yes | Yes | Used for Java code development - Test Automation code is in Java |
| JRE7 | Yes | Yes | Yes | Used to execute any java program |