TCOE AUTOMATION

FRAMEWORK USER GUIDE FOR PROTRACTOR

Kantharaj, Shankar SBOBNG-ITC/I/PF

SHELL INDIA

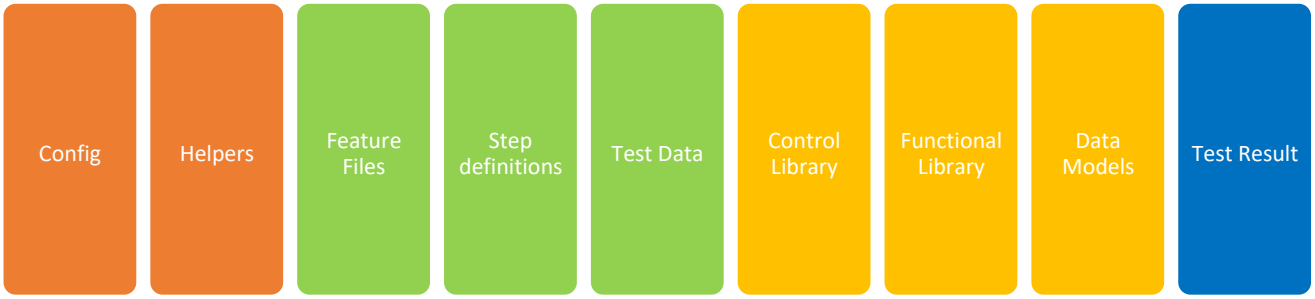
Table of Contents

INTRODUCTION

This Automation framework designed by TCoE is currently used to automate AngularJS web applications. The framework is built on Protractor. Its designed for ease of usability, maintainability and scalability. To use this framework, one must have fair understating of OOPS programing. The framework is written in TypeScript, but even without any knowledge on TypeScript one can use this framework with basic programing skills.

Framework Architecture

Framework



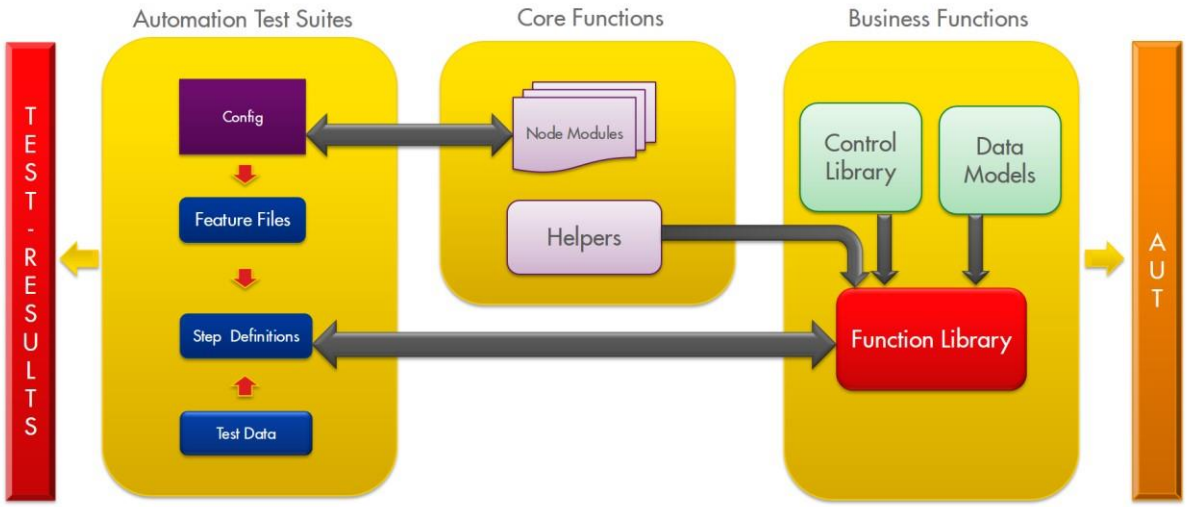
- The Framework consists of four major logical layers
    - Config and helpers as the base layer o Control Library, Function Library and Data Models as the Business functions layer o Feature files, Step definitions and Test data as the Test suite layer
    - Test Results as the Analysis Layer
- The base layer components will be maintained and enhanced by TCoE while the business layer will be maintained by the project teams.
- Config: This contains Config.ts which contains configurations for Protractor, CucumberJS and HTML reporter.
- Helpers: This contains a set of exportable Helper classes for Asserts, Waits, Model mappers and browser helpers.
- Feature Files: This contains description of feature and related scenarios in BDD format that maps to step definitions.
- Step Definitions: This contains the implementation for the mapped scenarios. In this framework the step definitions consume methods defined in function library and injects test data to corresponding data models using the custom mappers.
- Test Data: This contains test data that can be in any format like JSON, excel or Xml.
- Control Library: This contains Element finders definitions.
- Data Models: This contains model property definitions.
- Function Library: This contains reusable business functions that consume control library and Data model.

Why should you use this Framework?

- A robust framework ready to adopt.
- Write your Test cases using Gherkins in BDD format which can be understood by all stakeholders, PM/PO, Developers and testers.
- Good amount of utility methods and Helper methods to write your business functions.
- Easy maintenances of web elements in the control library which uses page factory pattern.
- Interactive Reports in BDD format with screenshots on failure.
- Supports cloud execution with BrowserStack, Sauces Labs, Selenium grid or PhantomJS
- Full support for integration of CI/CD with VSTS or Jenkins
- This framework has loosely coupled components that is flexible and extensible

- TCoE will be upgrading and releasing updated versions of base components and helper classes, so that there is no need to reinvent the wheel.

Framework Execution workflow



- When the test execution is triggered the settings in the Config.ts is applied the dependent node modules
- Execution continues by executing the hooks followed by the feature files declared in the scope.
- The Features file executes corresponding step definitions.
- Step definitions execute function library declared in it by injecting the test data to data models using the custom mapper developed.
- The function library executes by importing the control library and test data injected to data models and also by using relevant helpers
- As per the execution clear corresponding test results along with snapshots for failures is generated for the AUT (Application under test)

FRAMEWORK SETUP

Prerequisites

- To use the framework, you would need to use Visual Studio code as the IDE.
- Latest version of Node JS should be installed.
- Latest version of JDK and add java/bin to PATH variable.
- Install TypeScript globally by running the command “npm install -g typescript”.
- Install Protractor globally by running the command “npm install -g protractor”.
- Now you are all set to setup the framework

Setup the Framework in less than 5mins

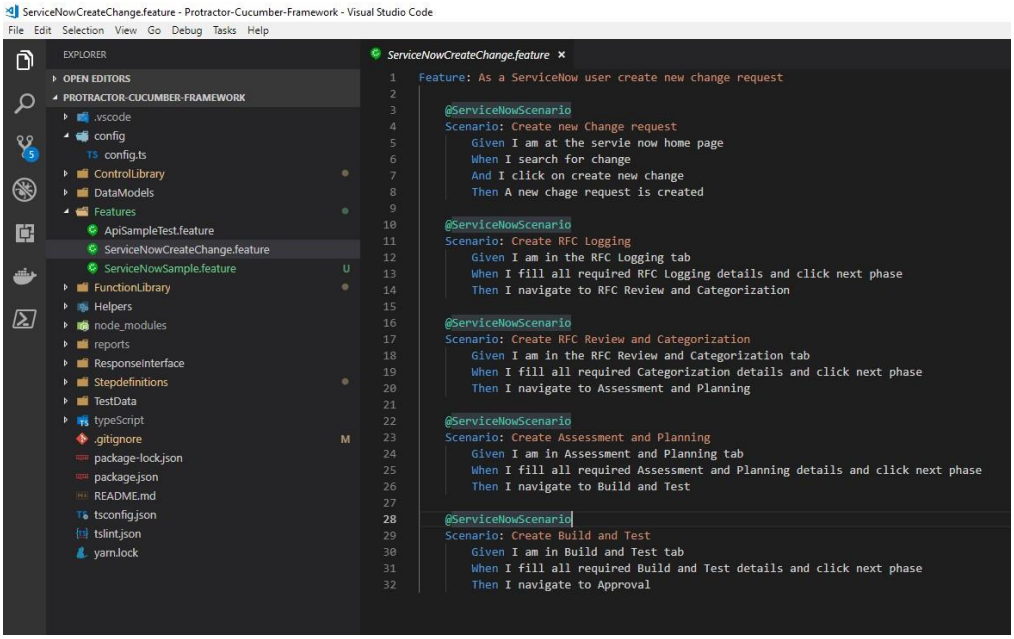
- First TCoE would provide you the base framework template and a PAT token.
- Once you have the framework you can open it in visual studio code
- In visual studio code open view > Terminal

- Check if user has a file by name '.npmrc' at the location C:\Users\<username>\
- If user doesn't have the above file
  1. Run npm install -g vsts-npm-auth in the Terminal
  2. Run vsts-npm-auth -config .npmrc in the Terminal
  3. User to find file by name '.npmrc' at the location C:\Users\<username>\
- Open '.npmrc' file at C:\Users\<username>\.npmrc and add the provided PAT token.
- In the Terminal run the command "npm install"
- Then run npm install protractor cucumber feed[Optional]
- Close visual studio code if required and reopen it again[Optional]
- Update selenium web driver by running command "npm run webdriver-update"
- Start Selenium server by running command "npm run webdriver-start".
- Compile the framework by running command "npm run build".
- Run tests by running the command "npm run test".

FRAMEWORK USAGE

Creating Feature Files and Generating step definitions

- Once your framework setup is done, your solution would look like this when opened in VS Code. While following the below steps, please refer to the provide examples in the solution for better understanding.



- The First step is to create a feature file, to do this just right click on the "Features" folder and create file with. feature extension.
- Please add your details for the feature and its related scenario in the BDD format.
- Once all the feature scenarios are written, right click on the "Step definitions" folder and create a .ts file.
- Define step definition methods as shown below

```
const { When, Then, Given, And } = require("cucumber");

//     Scenario: Create new Change request

Given(/^I am at the servie now home page$/, async () =>{

});

When(/^I search for change$/, async () => {
});

When(/^All the page Contents are loaded$/, async () => {

});

Then(/^A new chage request is created$/, async () => {

});
```

- Here your function library classes and Test data will be called, but before that let's see how to design those.

Creating Control Library and Function Library

- Now from create .ts file in the "Control library" folder and declare the web controls as shown below. Name the class as <PageName>controls.ts, where the PageName could be Home, Login or any functional name that the web page does.
- Now in the Control class create ElementFinder properties and Assign attribute values like xpath or ID in a private method. These can be identified from the DOM explorer in the browser using F12.

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

export class ServiceNowHomePageControls {

    public SearchTextBox: ElementFinder;
    public CreateChangeLink: ElementFinder;

    constructor() {
        this.SetChangeSearchControls();
    }

    private SetChangeSearchControls(){
        this.SearchTextBox = $('input[ng-model="filterTextValue"]');
        this.CreateChangeLink = element(by.xpath("//a[@id='323bb07bc611227a018aea9eb8f3b35e']//div[contains(text(),'Create New')]"));
    }
}
```

- Once this is done, right click on the Function Library folder and add a new .ts class. Name the class as <PageName>Functions.ts, where the PageName could be Home, Login or any functional name that the web page does.
- Now that you function class is ready, before you start writing the methods for the business function we need create a Model class with properties to inject test data.
- To do this, right click on the Data Models folder and add a new .ts class. Name the class as <PageName>FormData.ts, where the PageName could be Home, Login or any functional name that the web page does.
- In the model class define properties for all the fields that require data input.

```
import { BaseModel, ModelProperty } from '../Helpers/ModelMapper';

export class ChangeDataModel extends BaseModel {
    @ModelProperty()
    public SearchCriteria: string;

    @ModelProperty()
    public BussnessServiceOffering: string;

    @ModelProperty()
    public ShortDescription: string;


}
```

- Now let's get back to function library, declare a instance of the corresponding control library class and Inject it in the constructor and initialize an instance.

```
import { ServiceNowHomePageControls } from "../ControlLibrary/ServiceNowHomePageControls";
import { browser,protractor,promise,by } from "protractor";
import { waitFor } from "../Helpers/waitFor";
import { ChangeDataModel } from "../DataModels/ChangeDataModel";

const { click, waitToBeDisplayed, sendKeys, getText ,waitToBeNotPresent,selectOptionByText,selectOption  }
var driver = browser.driver;
var loc = by.tagName('iframe');
var el = driver.findElement(loc);
var path = require('path');

const chai = require('chai');
chai.use(require('chai-smoothie'));
const expect = chai.expect;


export class ServiceNowHomePageFunctions{

    serviceNowHomePageControls : ServiceNowHomePageControls;

    constructor(){
        this.serviceNowHomePageControls = new ServiceNowHomePageControls();
    }
```

- Define business function and use “chai-smoothie” for assertions and use Data Models as function parameter to represent data.

```
//#region Scenario: Create new Change request
public async AssertNavigateToServiceHomePage() : Promise<any> {
    await waitToBeDisplayed(this.serviceNowHomePageControls.SearchTextBox,10000);
    return await expect(this.serviceNowHomePageControls.SearchTextBox).to.be.displayed;
}


public async CreateNewChangeRequest(changeDataModel : ChangeDataModel): Promise<any> {
        await click(this.serviceNowHomePageControls.SearchTextBox);
        await sendKeys(this.serviceNowHomePageControls.SearchTextBox,changeDataModel.SearchCriteria);
        await waitFor.time(2000);
        await  click(this.serviceNowHomePageControls.CreateChangeLink);
        return await browser.waitForAngular();

}
```

- Next, we need to populate the Form data during execution which will be discussed in the next section.

Creating Test data

- Under TestData folder include the files you want to upload in the upload folder. Test data can be represented in a JSON format or can be deserialized to JSON using any other source. This Data is later mapped to Data Model using a custom mapper.

```
export const ChangeTestdata = {
    "SearchCriteria": "Change",
    "BussnessServiceOffering": "Altiris GB AB",
    "ShortDescription": "Example of a Short Description",
    "Description": "Example of a Description",
    "Justification": "Example of Justification",
    "RequestedByDate": "2019-02-07 08:49:56",
    "ChangeType": "Normal Change",
    "AssignedTo": "Adrian Smolarczyk (INASTO)",
    "ReasonForChange": "Project",
    "PlannedStartDate": "2019-02-16 09:46:25",
    "PlannedEndDate": "2019-02-27 09:47:23",
    "ChangePlan": "Example of change plane",
    "BackoutPlan": "Example of Back out Plan",
    "TestPlan": "Example of Test Plan",
    "SystemTestingExitCriteria": "Example of System Testing Exit Criteria",
    "Build_ChangeTask_AssignedTo": "Adrian Smolarczyk (INASTO)",
    "Build_ChangeTask_ClosureCode": "Task completed",
    "Build_ChangeTask_ClosingComments": "Example of Build Change Task Closing Comments",
    "Test_ChangeTask_TestPlanAttachment": "../TestData/UploadFiles/TestPlan.txt"
};

```

Using the function library and Test data in step definition

- Now open the step definition class that you created early

- Now declare an instance of the necessary function classes and test data classes and Initialize them as shown below.

```
const { When, Then, Given, And } = require("cucumber");
import { ServiceNowHomePageFunctions} from "../FunctionLibrary/ServiceNowHomePageFunctions";
import { ChangeDataModel } from "../DataModels/ChangeDataModel";
import { ChangeTestdata } from "../TestData/ChangeTestData"

const homePageFunctions = new ServiceNowHomePageFunctions();
const changeDataModel = new ChangeDataModel(ChangeTestdata);

//     Scenario: Create new Change request

Given(/^I am at the servie now home page$/, async () =>{
    await homePageFunctions.AssertNavigateToServiceHomePage();
});

When(/^I search for change$/, async () => {
    await homePageFunctions.CreateNewChangeRequest(changeDataModel);
});
```

- Now got to the respective steps methods and call the function library methods as shown above.
- Your almost done, now all you need to do is update your configurations which is discussed in the next section.

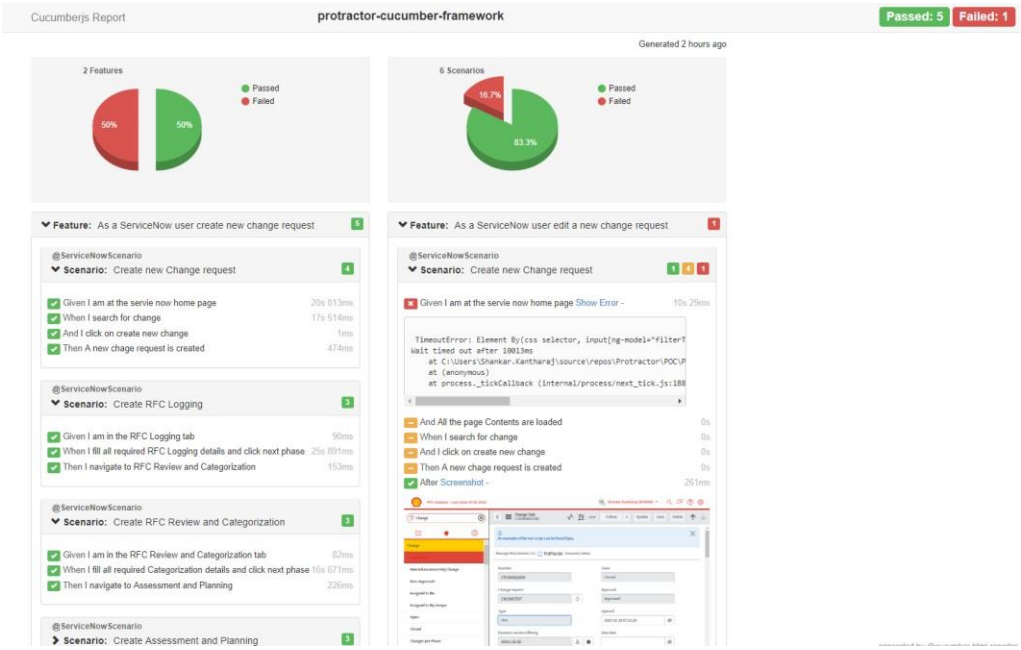
Setting up Config.ts

- Update Config.ts with selenium Server address, base URL and browser options.

Analyzing Test Reports

- Once the tests are run, navigate to the projects physical location and one the Reports folder •

The reports would look like as shown below.



Now go ahead and enjoy automating your application. For any question or query please feel free to reach out to TCoE team.

THANK YOU