



TCOE AUTOMATION FRAMEWORK USER GUIDE

Kantharaj, Shankar SBOBNG-ITC/I/PF

SHELL INDIA

Table of Contents

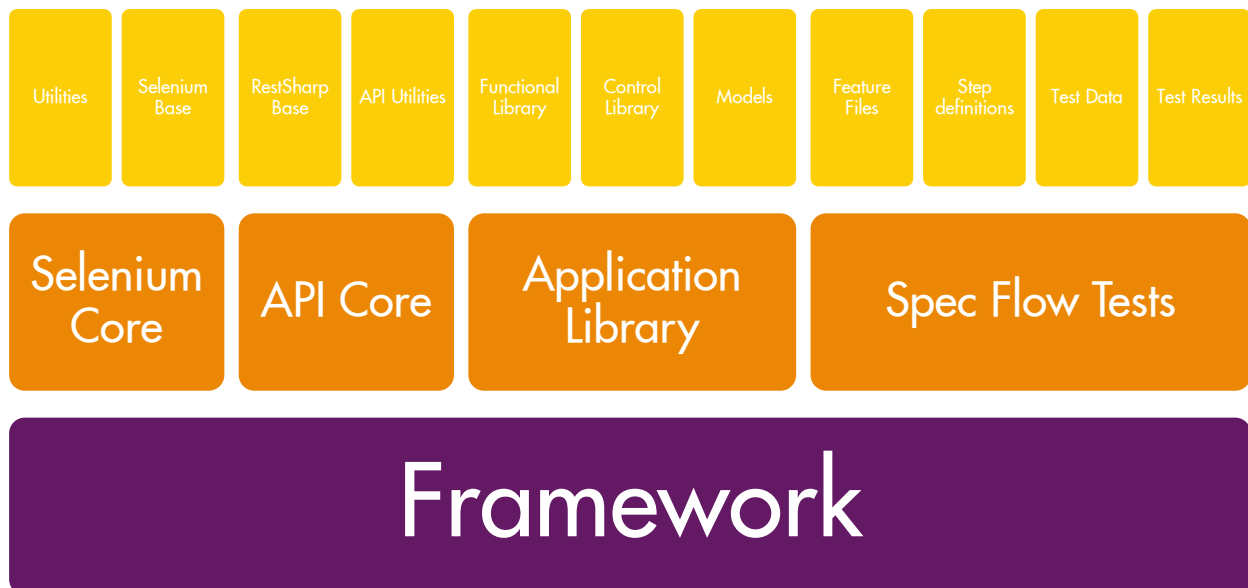
Introduction	1
Framework Architecture.....	2
Why should you use this Framework?	2
Framework Setup	3
Prerequisites Solution.....	3
Setup the Framework in less than 5mins.....	4
Optional Features setup.....	7
Framework Usage	8
Creating Feature Files and Generating step definitions	8
Creating Control Library and Function Library	11
Creating and Auto generating Test data.....	15
Using the Library functions and Test data in step definition	16
Setting up App.Config.....	17
Analyzing Test Reports.....	17

INTRODUCTION

This Automation framework designed by TCoE is currently used to automate web applications and REST APIs. The framework is built on selenium and Rest Sharp as its core components. Its designed for ease of usability, maintainability and scalability. To use this framework, one must have fair understating of

OOPS programming. The framework is written in C#, but even without any knowledge on C# one can use this framework with basic programming skills.

Framework Architecture



- The Framework consists of four major components.
- The Selenium core and API core which are the core components in form of NuGet packages provided by TCoE.
- The Application Library and Spec Flow tests are business components that contain the functionalities for Tests.
- Based on your requirements, you can use only selenium core or only API core or both.
- The Core components will be maintained and enhanced by TCoE while the business components will be maintained by the project teams.
- Selenium Core: It contains the Selenium base which can be registered by unity container to initialize your local or remote driver. The Utilities contains selenium extension methods and various utilities for browser, SQL, Excel and Waits.
- API Core: It contains RestSharp base which helps in registering the client and make all kind of GET and POST calls. It also contains methods for Authentication and Utility methods for serialize and deserialize the json or xml response content.
- Application Library: It contains the control library which is used to store the web element properties, function library that are designed by injecting control library to write business functions and Asserts, Models are property class to get and set Test Data.
- Spec Flow Tests: This contains your test suites written in BDD using spec flow. The Feature files contain a feature with all related scenarios written in plain English using gherkins. The Step definitions are reusable test methods that map to the scenarios which in turn calls function library and test data. The runs will generate test results with screen shots.

Why should you use this Framework?

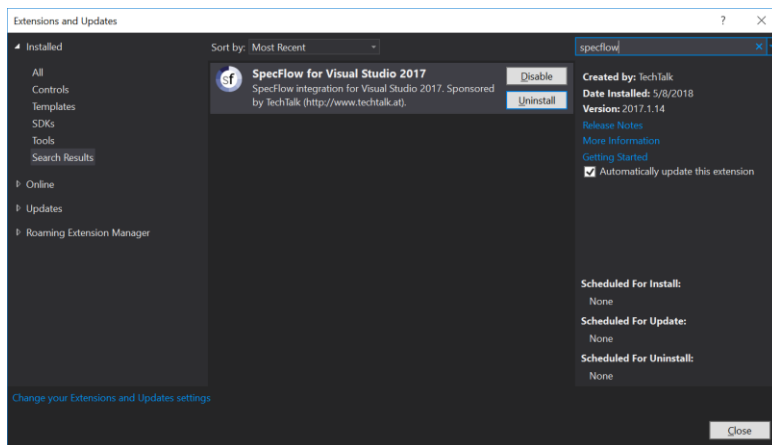
- A robust framework ready to adopt.

- 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 extensions methods to write your business functions.
- Easy maintenances of web elements in the control library which uses page factory pattern.
- A single framework for testing web applications and Rest APIs
- Interactive Reports in BDD format with screenshots on failure.
- Mongo DB Test Report server that gives insights and analytics on historic test runs for different projects.
- Supports customized parallel test executions.
- Auto generated Test data for non-specific data values.
- Supports responsive testing for various mobile devices
- Supports cloud execution with BrowserStack, Sauce 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 core components, so that there is no need to reinvent the wheel.

FRAMEWORK SETUP

Prerequisites Solution

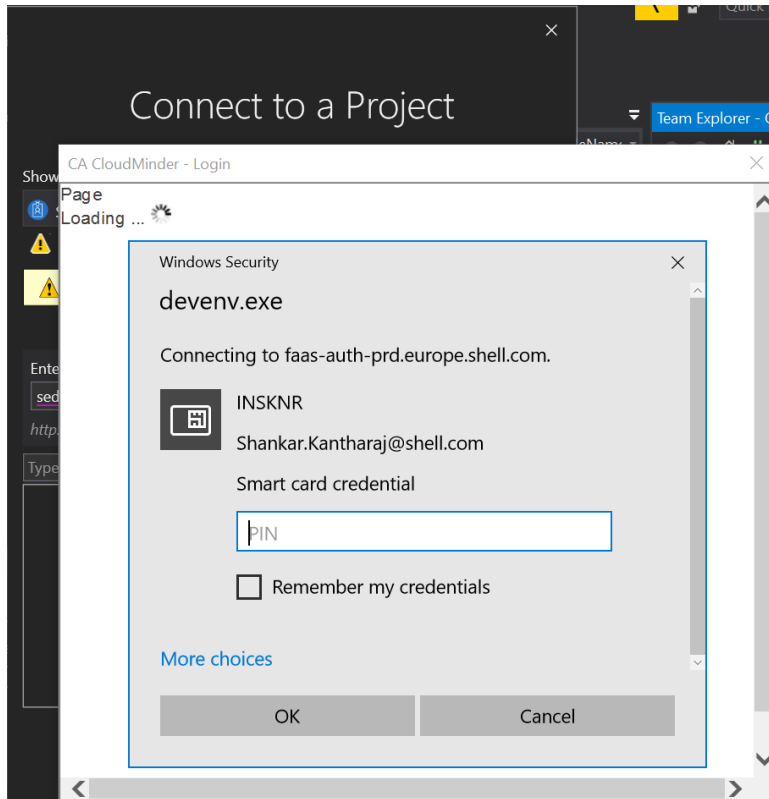
- To use the framework, you would need to use Visual Studio as the IDE, the frame work has been developed with Visual Studio 2017 Enterprise and is recommended to use the same.
- If you have visual studio subscription you can download from the subscription account or use the 30 days trail for the enterprise version.
- Once Visual studio is installed you need to install the extension for spec flow. To do this in your visual studio got to Tools> Extensions and Updates. Search for Spec Flow and install.



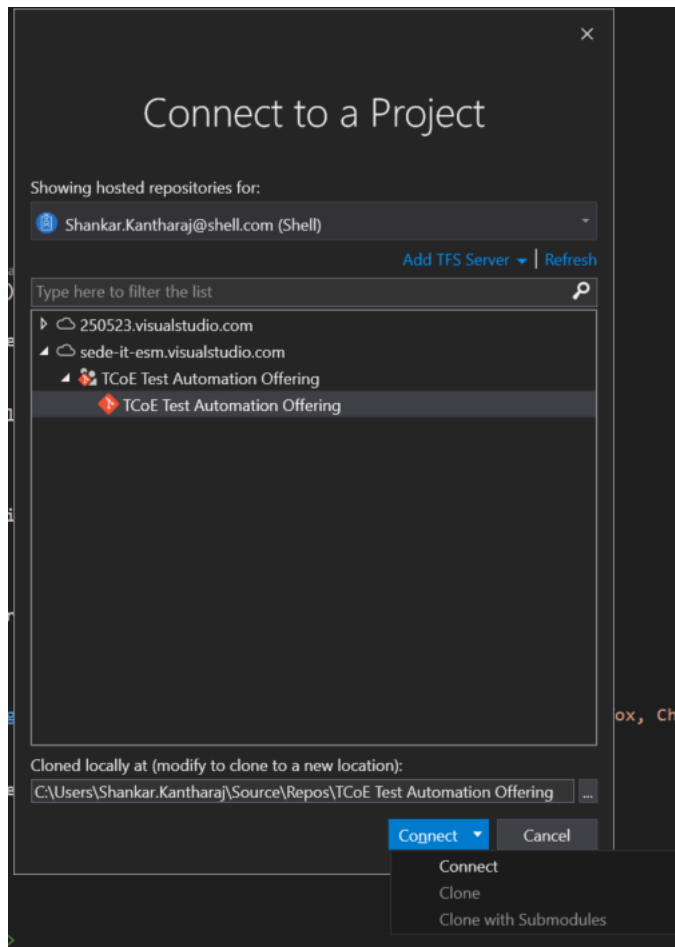
- Restart Visual studio
- Now you are all set to setup the framework

Setup the Framework in less than 5mins

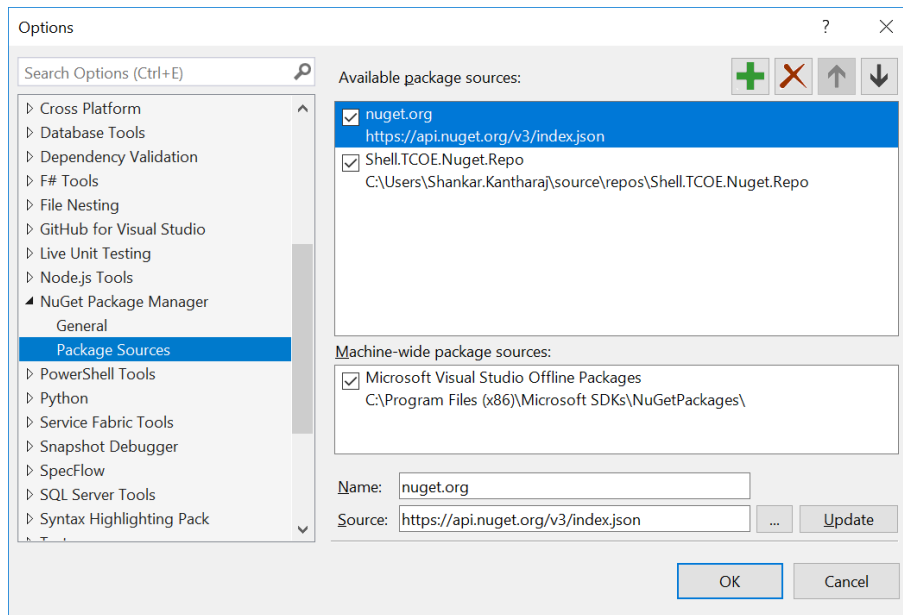
- First TCoE would provide you access to GIT repository in VSTS
- In Visual studio click on Team > Manage connection
- A pop up opens to connect to project and would ask you to sign in



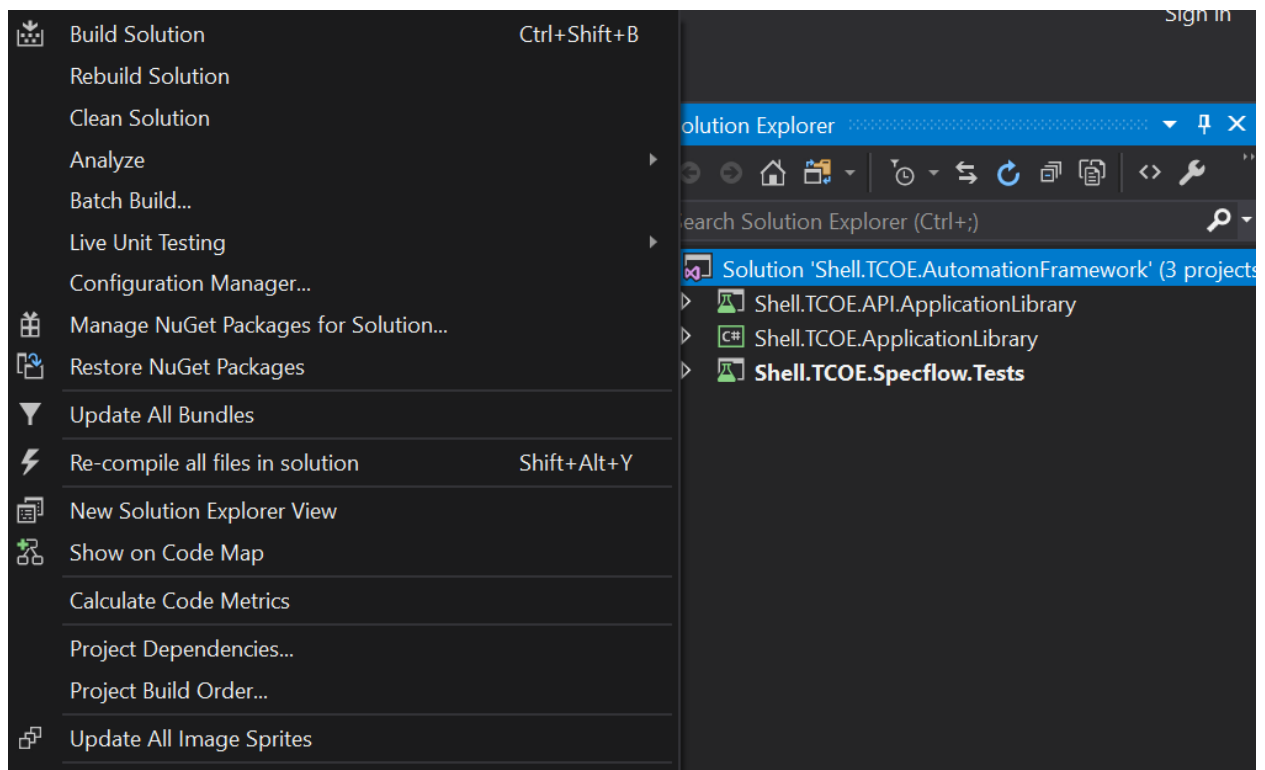
- After signing in you would see the git repository
- Click on Connect drop down and select clone.



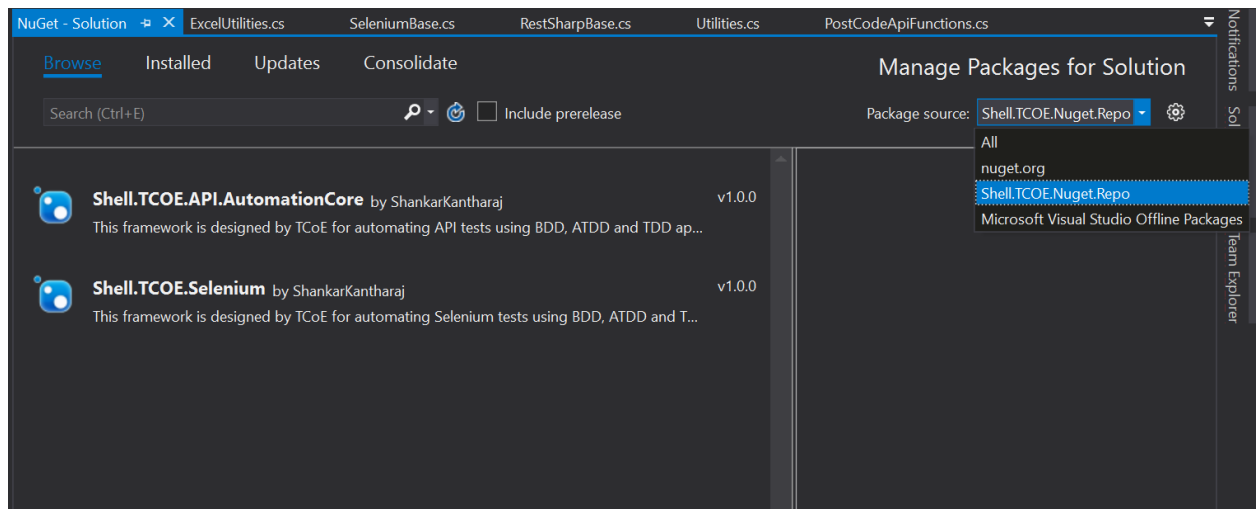
- The Framework solution will be downloaded to your local
- Click on Tools > NuGet package Manager > Package Manger settings
- Select package sources and click on Add [+], Provide the path to the local NuGet repository and Name the repo Shell.TCOE.Nuget.Repo



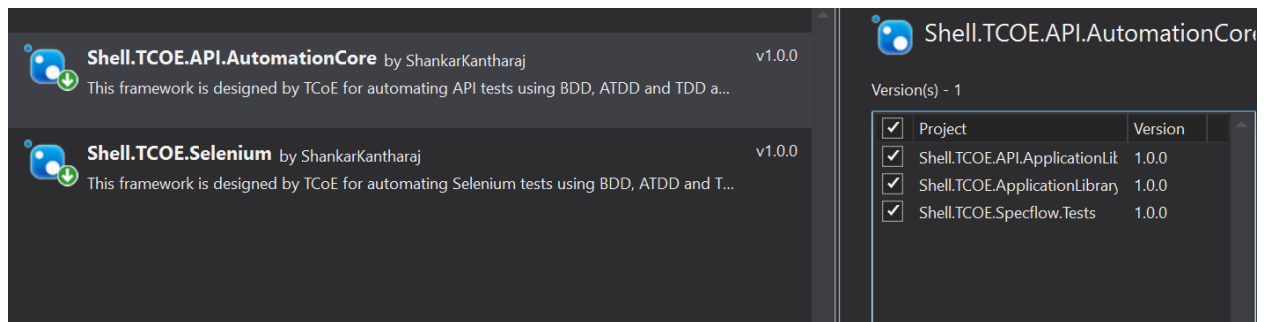
- Right click on the Solution and select Manage NuGet Packages for Solution



- In Package Source select Shell.TCOE.Nuget.Repo and select the Browse tab



- Install the Selenium package if your automating only web application, install API package if your automating only Rest API and Install both if your automating both to all the projects.



- Right click on solution and build or press ctrl+shift+B
- That's it, you're ready to automate.

Optional Features setup

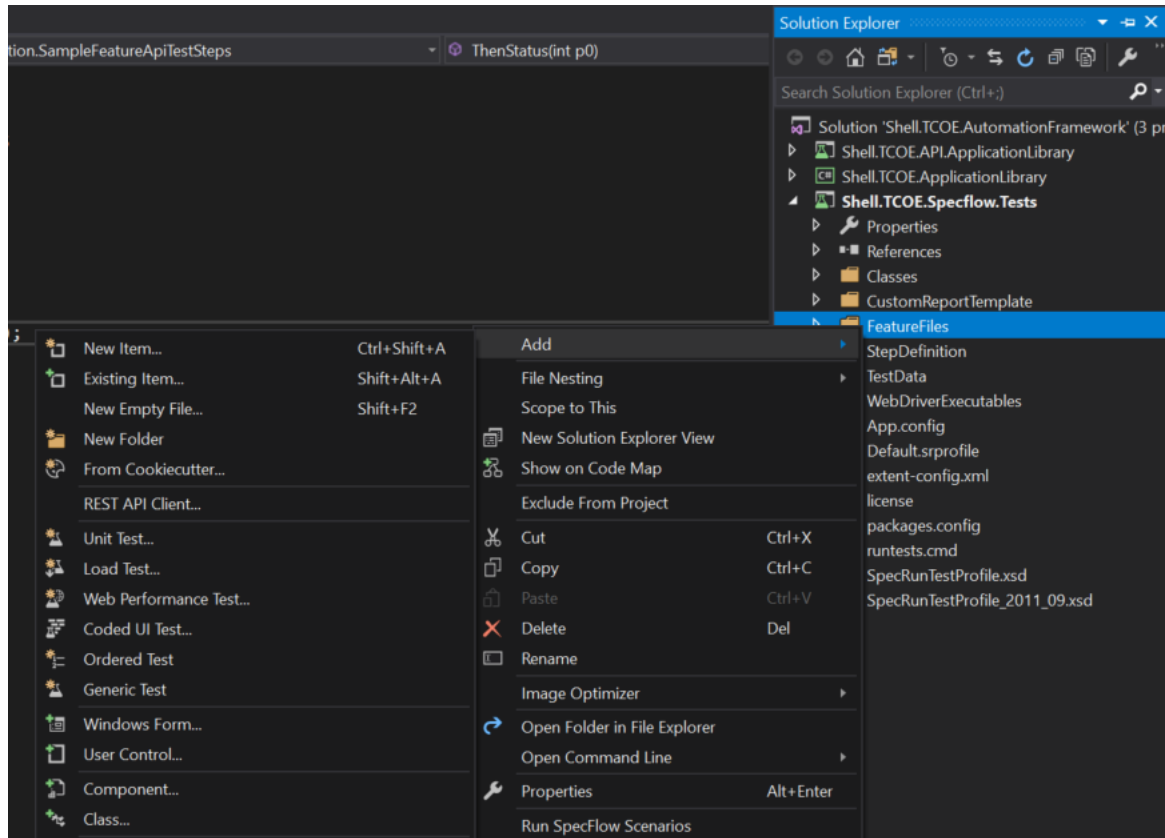
- The framework has support to configure a report server
- To use the report server, navigate to [Klov](#) web page and follow the instructions to install MongoDB server and Klov jar file.
- There no need to do any setup for Redis.
- In the solution open App.Config and set UseKlov to true


```
<add key="UseKlov" value="true" />
```
- Start the MongoDB server then the Klov jar
- Now run your test suits

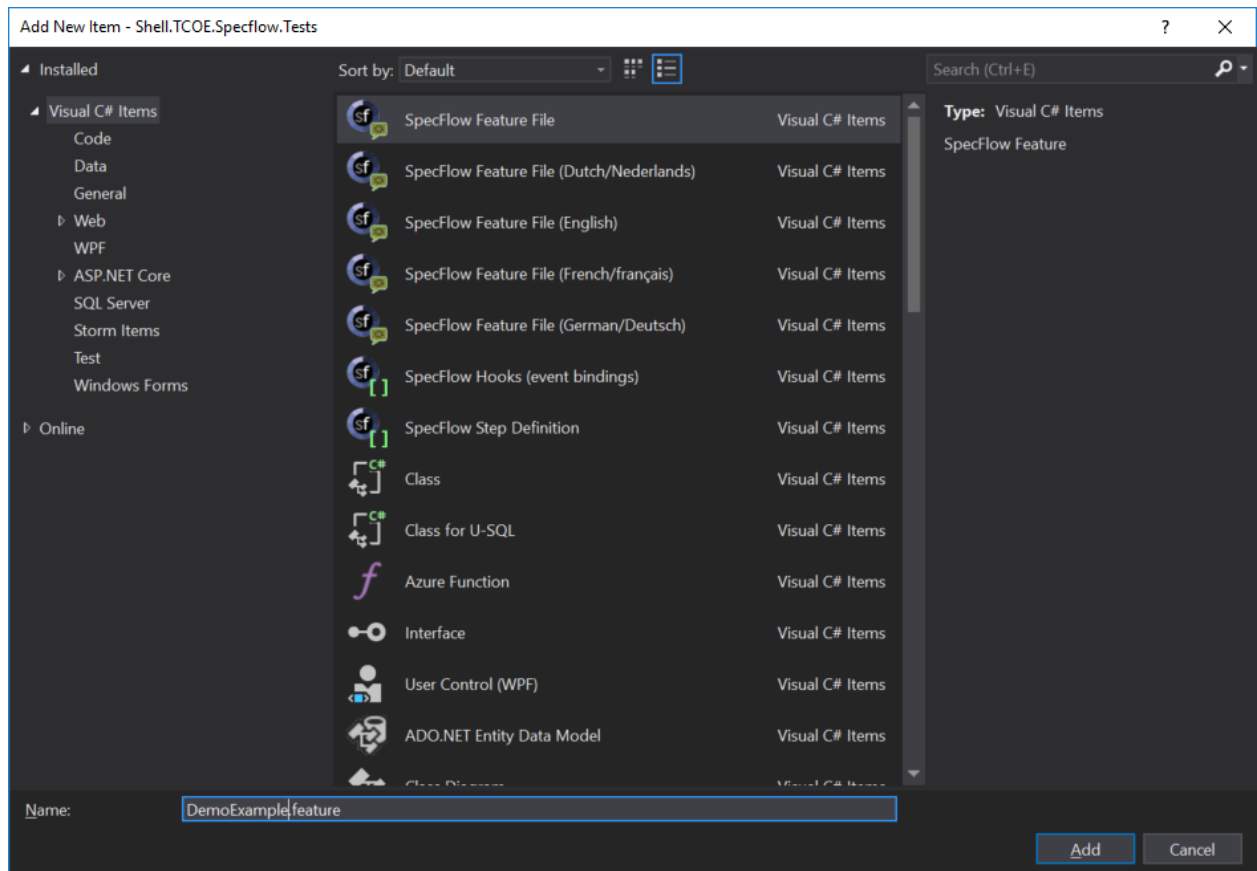
FRAMEWORK USAGE

Creating Feature Files and Generating step definitions

- Once your framework setup is done, your solution would look like this.



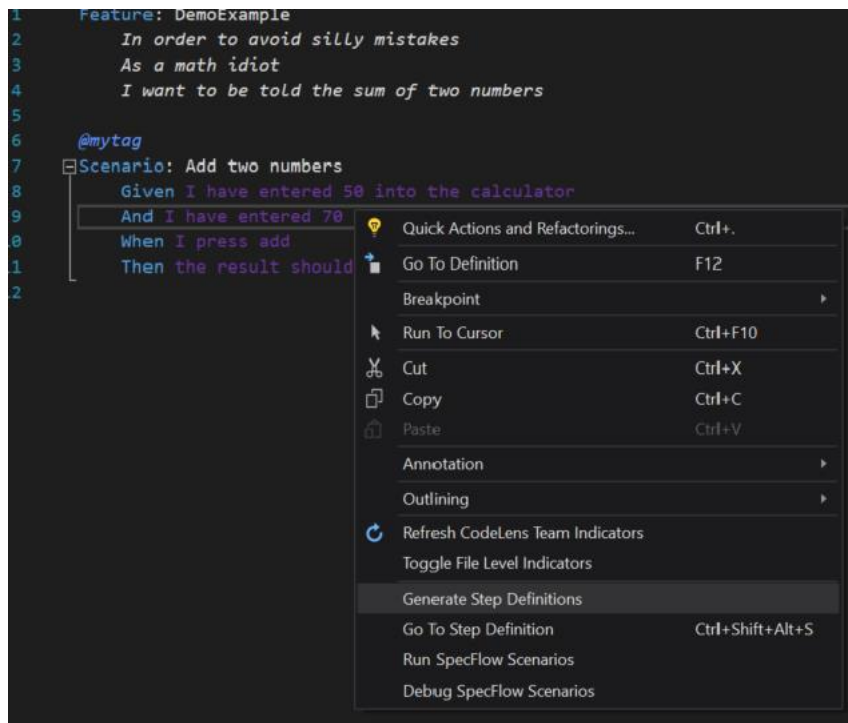
- As shown above right click on the Feature file folder and select Add> New Item
- Select Feature File and rename it as required and click on add



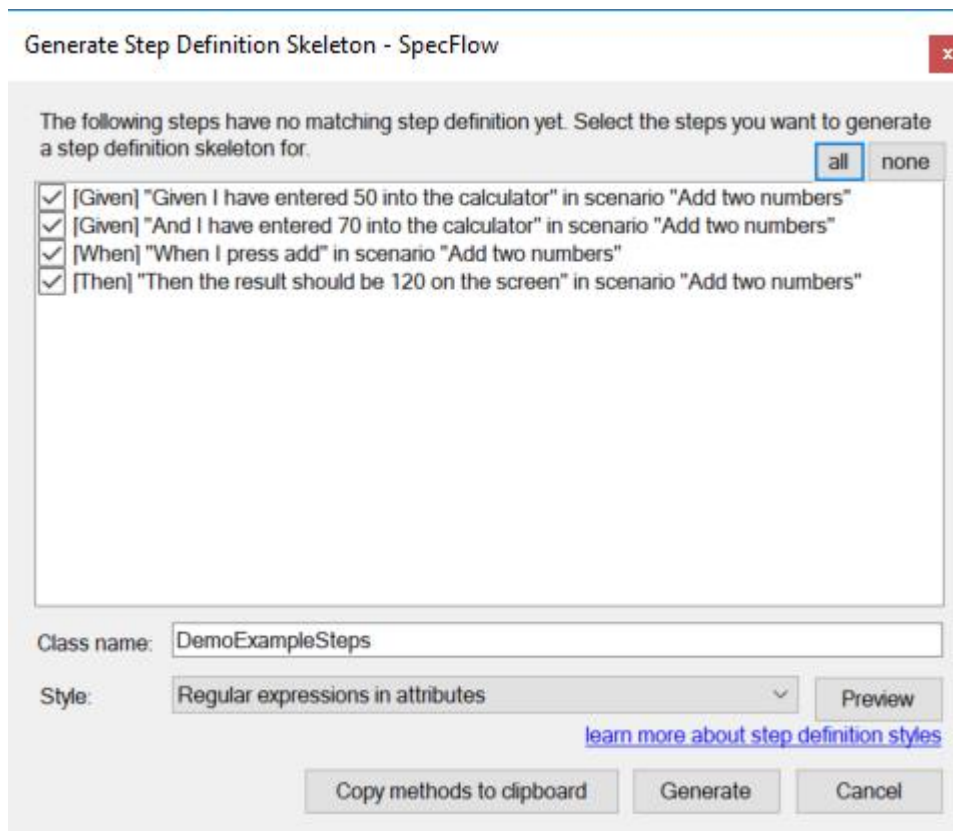
- Type in your feature and add the scenarios in Given, When and Then format.

```
DemoExample.feature  X SampleFeatureApiTestSteps.cs  LoginFunctions.cs
1  Feature: DemoExample
2      In order to avoid silly mistakes
3      As a math idiot
4      I want to be told the sum of two numbers
5
6      @mytag
7      Scenario: Add two numbers
8          Given I have entered 50 into the calculator
9          And I have entered 70 into the calculator
10         When I press add
11         Then the result should be 120 on the screen
12
```

- Once all the feature scenarios are written, right click on the editor and select generate Step definitions



- Now from the generate step definition window you can either click on generate and select the step definition folder to save the file, or click on copy methods to clipboard and paste it in a new or existing class.



- The generated step definition look like as shown below

```

using System;
using TechTalk.SpecFlow;

namespace Shell.TCOE.Specflow.Tests.StepDefinition
{
    [Binding]
    0 references | 0 changes | 0 authors, 0 changes
    public class DemoExampleSteps
    {
        [Given(@"I have entered (.*) into the calculator")]
        0 references | 0 changes | 0 authors, 0 changes
        public void GivenIHaveEnteredIntoTheCalculator(int p0)
        {
            ScenarioContext.Current.Pending();
        }

        [When(@"I press add")]
        0 references | 0 changes | 0 authors, 0 changes
        public void WhenIPressAdd()
        {
            ScenarioContext.Current.Pending();
        }

        [Then(@"the result should be (.*) on the screen")]
        0 references | 0 changes | 0 authors, 0 changes
        public void ThenTheResultShouldBeOnTheScreen(int p0)
        {
            ScenarioContext.Current.Pending();
        }
    }
}

```

- 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

- In your application Library project, right click on the control Library folder and add a new class. Name the class as <PageName>controls.cs, where the PageName could be Home , Login or any functional name that the web page does.
- Now in the Control class create IWebElement properties and use the FindBy annotations and assign attribute values like xpath or ID. These can be identified from the DOM explorer in the browser using F12.

```

using OpenQA.Selenium;
using OpenQA.Selenium.Support.PageObjects;

namespace Shell.TCOE.ApplicationLibrary.ControlLibrary
{
    /// <summary>
    /// Class of Webelements for Login Controls
    /// </summary>
    3 references
    public class LoginControls
    {
        #region Control Definitions

        [FindsBy(How = How.XPath, Using = "//input[@id='userName']")]
        1 reference
        public IWebElement LoginTextField { get; set; }

        [FindsBy(How = How.XPath, Using = "//input[@id='password']")]
        1 reference
        public IWebElement PasswordField { get; set; }

        [FindsBy(How = How.XPath, Using = "//button[@id='submitButton']")]
        2 references
        public IWebElement LoginButton { get; set; }

        [FindsBy(How = How.XPath, Using = "//button[@title='Log out']")]
        0 references
        public IWebElement LogoutButton { get; set; }

        #endregion
    }
}

```

- Once this is done, in your application Library project, right click on the Function Library folder and add a new class. Name the class as <PageName>Functions.cs, 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, in your application Library project, right click on the Models folder and add a new class. Name the class as <PageName>FormData.cs, 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. Regex and Data Annotations can also be added to format input data.

```

using System.ComponentModel.DataAnnotations;

namespace Shell.TCOE.ApplicationLibrary.Models
{
    /// <summary>
    /// Model properties for Login form data
    /// </summary>
    0 references
    public class LoginFormData
    {
        /// <summary>
        /// Login ID
        /// </summary>
        [RegularExpression(@"\d*[1-9]\d*", ErrorMessage = "Invalid Login ID")]
        0 references
        public string LoginID { get; set; }

        /// <summary>
        /// password
        /// </summary>
        0 references
        public string Password { get; set; }

        /// <summary>
        /// Language
        /// </summary>
        0 references
        public string Language { get; set; }
    }
}

```

- Now let's get back to function class, declare a private read only instance of the corresponding control class and Inject it in the constructor and initialize it with PageFactory as shown below.

```

namespace Shell.TCOE.ApplicationLibrary.FunctionalLibrary
{
    /// <summary>
    /// Login Page functions
    /// </summary>
    3 references
    public class LoginFunctions
    {
        #region Private Declarations

        private readonly LoginControls loginControls;

        #endregion

        #region Constructor

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="loginControls">The login controls to inject</param>
        1 reference
        public LoginFunctions(LoginControls loginControls)
        {
            this.loginControls = loginControls;

            PageFactory.InitElements(SeleniumBase.WebDriver, this.loginControls);
        }

        #endregion
    }
}

```

- Now you can write the business function methods and if there is a test data to set, then the methods would accept model form data as a parameter as shown below. Also include your Asserts for validation here.

```

/// <summary>
/// Fill and Verify Permit
/// </summary>
/// <param name="permitDetailsFormData"></param>
0 references
public void FillVerifyPermit(PermitDetailsFormData permitDetailsFormData)
{
    verifyPermitControls.PermittType.SelectValueFromDropDown(permitDetailsFormData.PermittType);
    verifyPermitControls.TurnAroundCheckBox.Click();
    verifyPermitControls.PermittTitle.EnterText(permitDetailsFormData.PermittTitle);
    verifyPermitControls.WorkDescription.EnterText(permitDetailsFormData.DescriptionOfWork);
    verifyPermitControls.PermittStartDate.EnterText(permitDetailsFormData.PlannedStartDate);
    verifyPermitControls.PermittEndDate.EnterText(permitDetailsFormData.PlannedEndDate);
    verifyPermitControls.PermittFromHour.EnterText(permitDetailsFormData.WorkingHoursFrom);
    verifyPermitControls.PermittToHour.EnterText(permitDetailsFormData.WorkingHoursTo);
}

```

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

Creating and Auto generating Test data

- Now let's switch back to the specflow tests project. Under TestData folder create a class. Create an internal instance of the corresponding form data class instance.
- Here we would use auto fixtures to populate data to all the properties, for those properties where data is specific that can be explicitly mentioned as shown below.

```
using Autofixture;
using Shell.TCOE.ApplicationLibrary.Models;

namespace Shell.TCOE.Specflow.Tests.TestData
{
    2 references
    public class PermitDetailTestData
    {
        internal PermitDetailsFormData PermitFeatureTestData;

        1 reference
        internal void CreatePermitTestData()
        {
            PermitFeatureTestData = CreatePermitDetailTestData();
        }

        1 reference
        private PermitDetailsFormData CreatePermitDetailTestData()
        {
            var fixture = new Fixture();
            var mc = fixture.Build<PermitDetailsFormData>()
                .With(x => x.PermitType, "Hot Work")
                .With(x => x.PlannedStartDate, "20 Mar 2018")
                .With(x => x.PlannedEndDate, "30 Mar 2018")
                .With(x => x.WorkingHoursFrom, "1200")
                .With(x => x.WorkingHoursTo, "1930")
                .Create();
            return mc;
        }
    }
}
```

- Alternatively, one can also use excel or SQL DB as a data source

Using the Library functions and Test data in step definition

- Now open the step definition class that you generated early and remove below line from all methods.
- `ScenarioContext.Current.Pending();`
- Now declare an instance of the necessary function classes and test data classes and Initialize them in a before scenario block as shown below.

```
using TechTalk.SpecFlow;

namespace Shell.TCOE.Specflow.Tests.StepDefinition
{
    [Binding]
    0 references
    public class SampleFeatureTestSteps
    {
        private LoginFunctions loginFunctions;
        private PermitMenuFunctions permitMenuFunctions;
        private VerifyPermitFunctions verifyPermitFunctions;
        private PermitDetailTestData permitDetailTestData;

        /// <summary>
        /// Test initializer
        /// </summary>
        ///
        [BeforeScenario("PermitFeature")]
        0 references
        public void TestInitialize()
        {
            //Initialize Application Library
            loginFunctions = new LoginFunctions(new LoginControls());
            permitMenuFunctions = new PermitMenuFunctions(new PermitMenuControls());
            verifyPermitFunctions = new VerifyPermitFunctions(new VerifyPermitControls());

            //Initialize Test Data Models Library
            permitDetailTestData = new PermitDetailTestData();

            //Setup Test Data
            permitDetailTestData.CreatePermitTestData();
        }
    }
}
```

- Now got to the respective steps methods and call the function library methods as shown below

```
[When(@"I navigate to verify template")]
0 references
public void WhenINavigateToVerifyTemplate()
{
    permitMenuFunctions.NavigateToVerifyPermit();
    verifyPermitFunctions.FillVerifyPermit(permitDetailTestData.PermitFeatureTestData);
}
```

- Your almost done, now all you need to do is update your configurations which is discussed in the next section.

Setting up App.Config

- One the App.Config file and update the test URL, machine name and environment.

```
<appSettings>
  <add key="TestsUrl" value="https://demoUrl.com/login?return_url=.%2F" />
  <add key="Environment" value="local" />
  <add key="RemoteEnvironment" value="browserstack" />
  <add key="MachinesToRunSelenium" value="DEMO-MachineName" />
  <add key="Browser" value="chrome" />
```

- You can update the report settings if your using the report server.

```
<!--Extent Report settings -->
<add key="UseKlov" value="false" />
<add key="MongodbUrl" value="localhost" />
<add key="MongodbPort" value="27017" />
<add key="KlovProjectName" value="Shell.TCOE.Automation Test Results" />
<add key="KlovUrl" value="http://localhost:5689" />
<add key="KlovReportName" value="Shell.TCoE_SampleTestRun" />
```

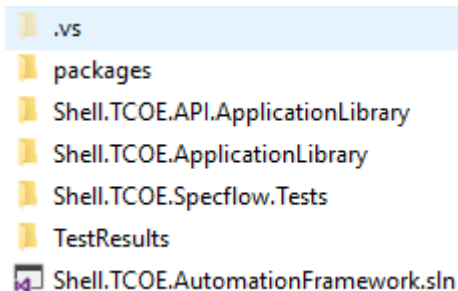
- If you want to test the responsiveness on mobile applications update the mobile emulation settings.

```
<!--Mobile Emulation : Please Note : to use emulation use Browser as Chrome-->
<add key="EnableMobileEmulation" value="NO" />
<add key="MobileDevice" value="iPhone 7" /><!--Select available emulated devices -->
```

- Now you are all set to run your test cases, go ahead and next we will discuss how to analyze test reports.

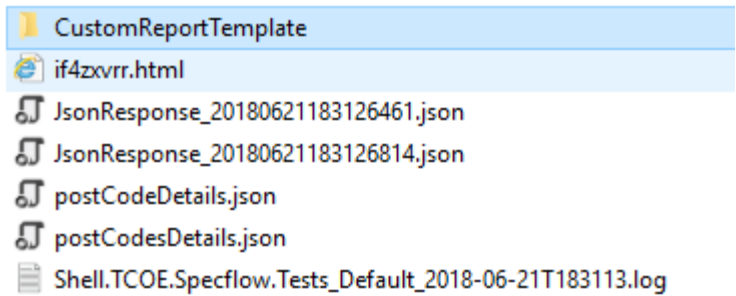
Analyzing Test Reports

- When it comes to test reports you have the flexibility to use test report that is generated by specflow or customize the razor view to your organizations theme. You can use the extent reports that are in BDD format and view the same on Klov report server.
- Once the tests are run, navigate to the projects physical location and one the test results folder

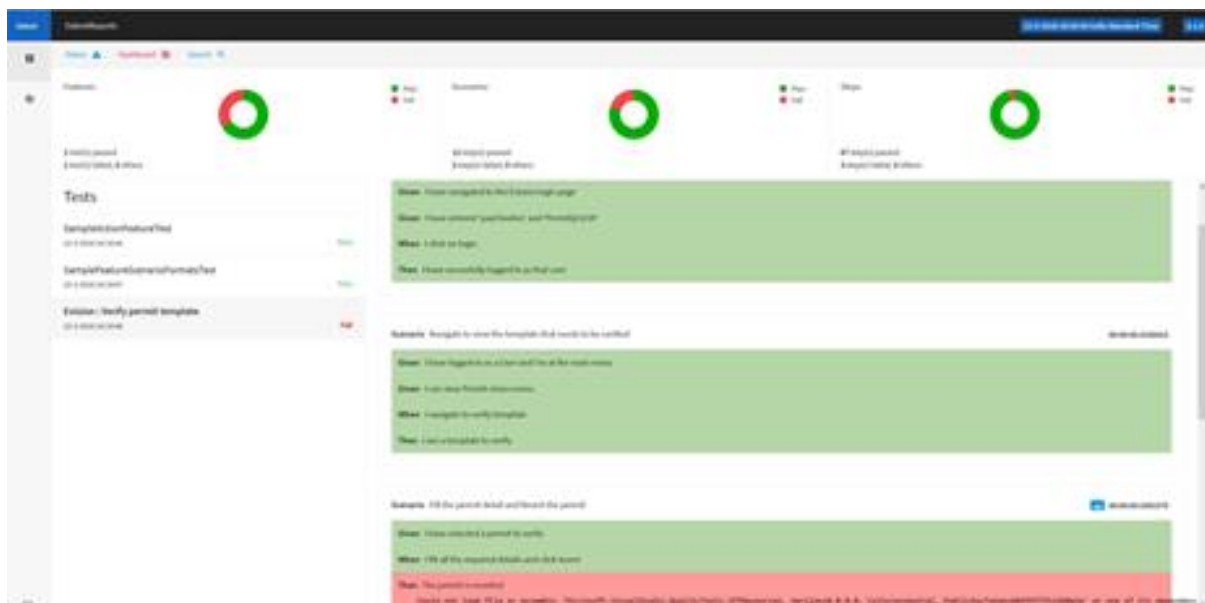


```
.vs
packages
Shell.TCOE.API.ApplicationLibrary
Shell.TCOE.ApplicationLibrary
Shell.TCOE.Specflow.Tests
TestResults
Shell.TCOE.AutomationFramework.sln
```

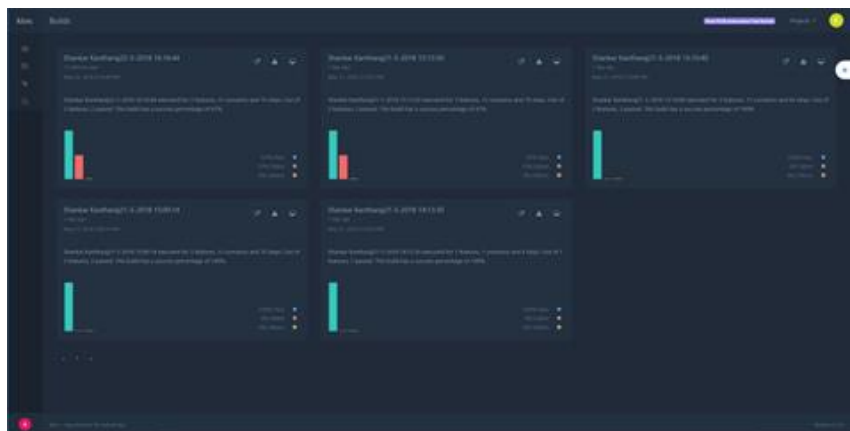
- Here both extent reports and specflow reports are generated with screen shots.
- Open the custom report template to view the specflow results and the last generated html file in test results for extent reports.



- The reports would look like as shown below.



- The report is grouped by features and there is screen shot attached for any failed tests.
- The report server will give you analytical information about past test run, trends and build information.



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

THANK YOU