Test Automation as a Service

Author: Imran Abbas Satti

18/01/2021

Overview

This is helping guide and it will help you in understanding the TASS approach using C#. In the repo you will see three directories as below.

1. [**code**] This directory contains the test automation framework code in project root directory named as TestAutoService
2. [**test_cases**] This directory contains two feature files representing the case document
3. [**test_report**] This directory contains the test case report generated by automation script

Test Automation As a Service (TAAS)

I have not just only automated test scenarios using BDD but also designed a complete frame work with an Idea of Test Automation As a Service, my own Idea.

Approach/Idea of TAAS

TAAS is a centralized automation framework for performing automated testing different application deployment environments (test, stage, prod) over different platforms (web, android, IOS) over different testing layers (UI, API, Databases). All in one service.

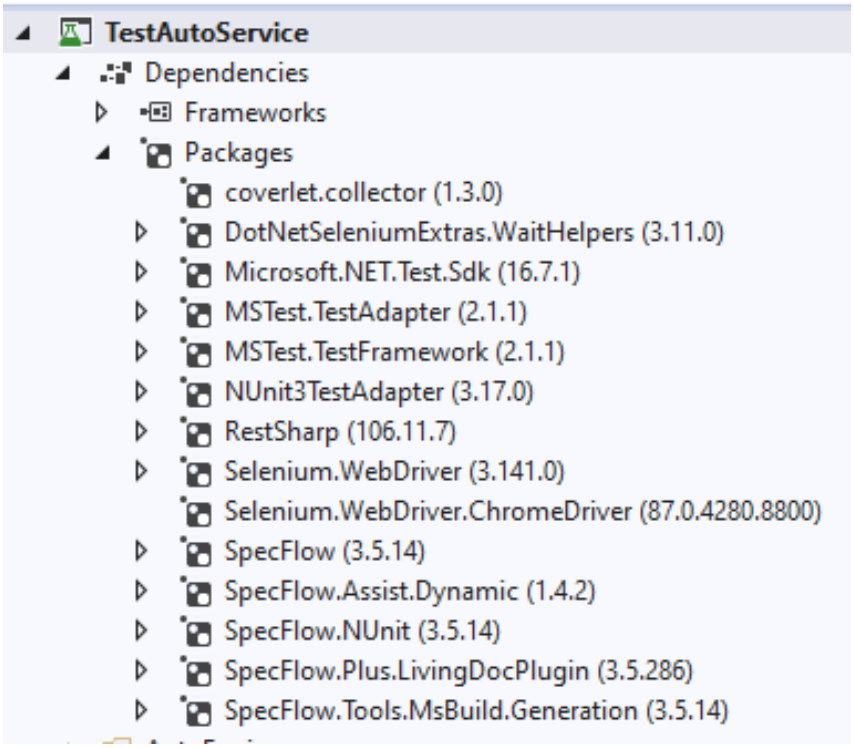
Implementation Technology Stack

Following technology stack is used in automating **6** test scenarios **3** at UI layer and **3** at API layer.

1. Visual Studio Community Version 2019
2. C#
3. Project
    - Specflow Template
    - .NET Core 3.1
    - Nunit as runner

4. Installed visual studio extensions
    - Specflow
    - NUnit 3 Test Adapter

5. Installed following Dependencies using Nuget Manger

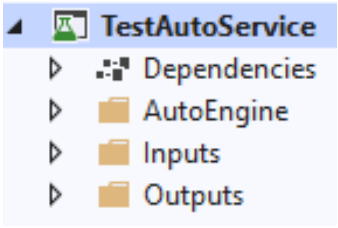


6. For reporting I have used
    - CLI tool for Specflow
    - Specflow Plus Living doc plugin
    - Find the link below

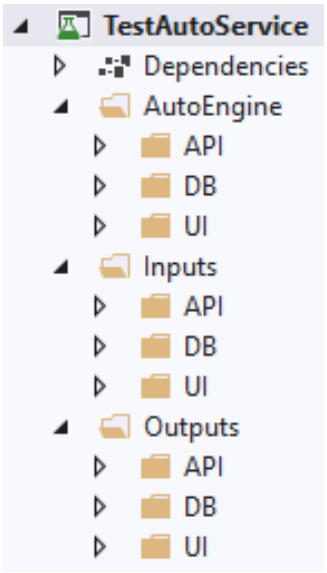
https://docs.specflow.org/projects/specflow/en/latest/Tools/Reporting.html

Understanding Framework Architecture

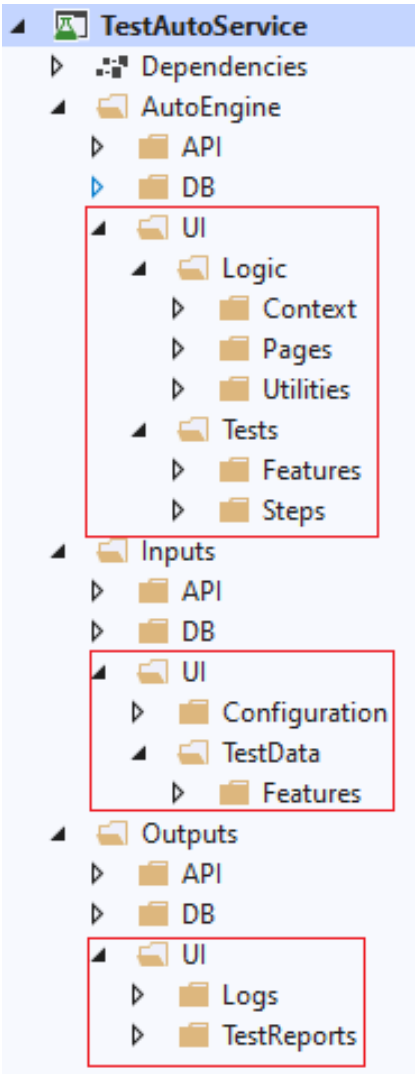
1. Framework have only three directories under project root folder [**TestAutoService**]
    a. **Auto Engine** contains the core logic and C# code files
    b. **Inputs** contains static files like test data, configurations etc
    c. **Outputs** also contains static files like test results,logs etc



2. Each above directory is divided into three subdirectories named [API,DB,UI] as we are following layer base approach, so we have segregated test cases on the base of layer. Also according to research 90 % issues lies under API and DB layer, so this segregation will help us in maintaining the test cases.

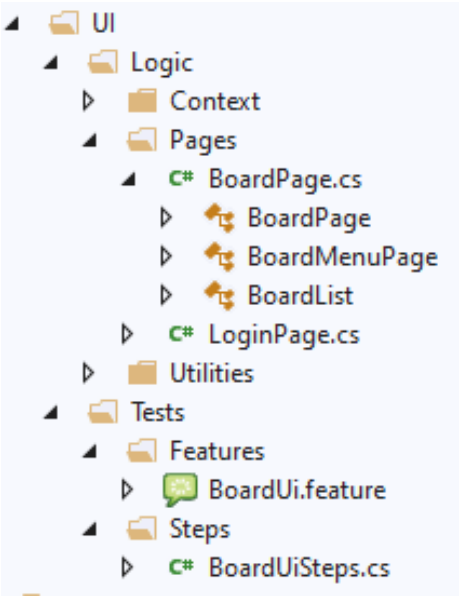


3. let's take look into **UI** layer first
    a. Under **AutoEngine** in UI we have **Logic** and **Tests** directory. Test directory contain further two folders **Features** and **Steps**
    b. In **Inputs** we have **Configuration** and **TestData** folder etc
    c. In **Outputs** under UI we have Logs and **TestReports** etc

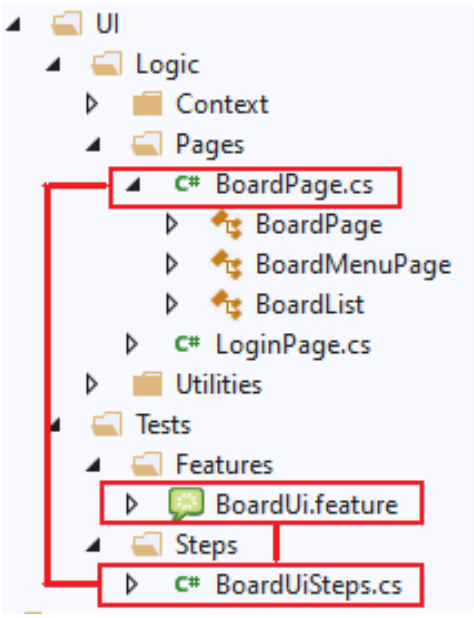


So this hierarchy is not random but is done after doing analysis on a lot of factors like maintenance, integration, modularity, separation of concerns, accessibility, DRY and readability.

4. For UI automation I have followed **Page Object pattern** and leveraged OOP principles. Like **BoardMenuPage** and **BoardList** class is derived from **BoardPage** class.

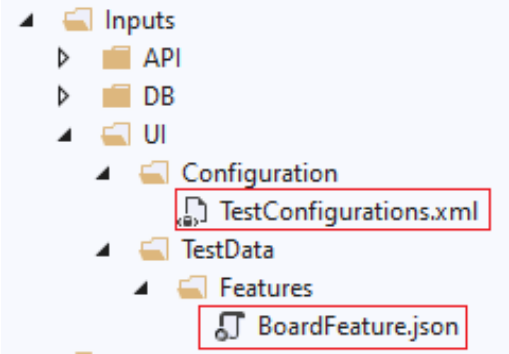


5. Linkage of files
    a. Each **feature file** is linked to one **step file** only, following one to one approach
    b. Each **step file** is linked to one **logic file** only

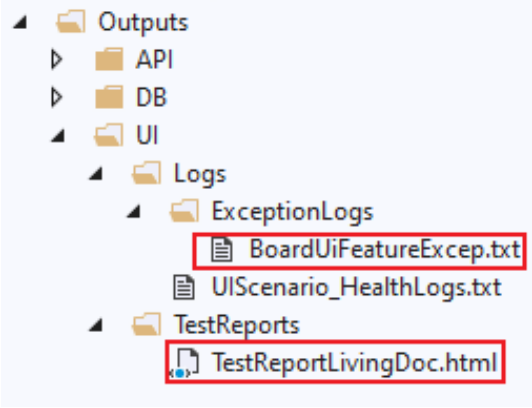


Same approach is used for **API** and **DB** layer

6. In **Input** folder we have static and centralized files like for environmental configuration we can refer **to TestConfiguration.xml** file. Also we have test data folder for placing test data on feature bases



7. In **Outputs** directory we have logs and test reports. Considering logs we can have exception logs, health logs etc.



8. For reporting I have used **Specflow CLI** tool.

9. Execution of test cases
    a. One way is using **cmd** to execute test cases for that
        i. Open cmd
        ii. Run the command **dotnet test <path to TestAutoService project>**
    b. Second way is to import **TestAutoService** project in visual code and use test explorer to execute test cases

Modular Level Architecture

Each module have its own unique responsibility and a defined dependency, as shown in figure below