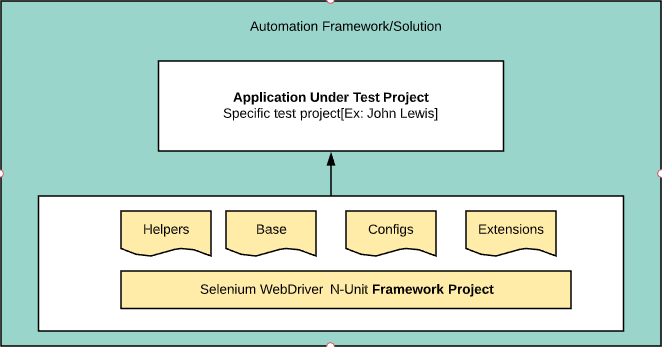
**Documentation on Automation Framework:**

I created this BDD automation framework using:

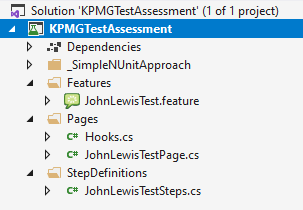
* C# [.Net Core 3.1]
* N-Unit with SpecFlow
* Selenium with ChromeDriver

With POM and Hybrid framework [combination of Data Driven and Hybrid Driven].

*Usually I build the automation framework with basic principles like Reusability, Maintainability, Scalability, Understandability and platform generic.* ***High Level architecture****:*



As per requirements to keep it simple and also, I’ve time boxed this task to one hour, so created single project for both reusable framework logic along with AUT [Application Under Test].



KPMGTestAssessment **solution** contains KPMHTestAssessment **project** which contains three main subfolders:

**Features**: Added feature file which contains the test scenarios.

**Pages**: Added page file which contains all web elements along with web actions/methods that are required as part of this test.

**StepDefinitions**: Added test step file which integrates the Gherkin steps with actual automation implementation *[using regular expressions via SpecFlow]* along with some assertions.

**Configuration and Execution:**

Pre-Requisite: To execute the code without any issues:

* Visual studio with .Net Core 3.1 SDK
* SpecFlow for Visual studio 2019 Extension
* NUnit 2 Test Adapter
* Chrome v84 as I am using ChromeDriver 84.0.4147.3001

I’ve added all required project dependencies under packages as part of project repository, so you just need to open the solution and build it. Then you should able to see and execute the test scenarios under Test Explorer.

**Test Details:**

I’ve scripted two scenarios and tagged them as test 1 & 2.

*@Test1*

Scenario: User journey of purchasing flow for a particular item on the John Lewis site.

This covers full end to end user flow as per requirements – Opens the John Lewis site, accept cookie, search for product ‘Laptop’, choose the first item in the results, update the quantity to ‘2’ then add to basket, deletes from basket and finally assert basket is empty and clear the cookies.

Though I am searching for product laptop and picking the first time from product list – not directly linked to specific model or brand – but just in case if John Lewis stop selling laptops.

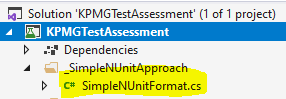
Then as per **additional bonus points requirement** ‘i.e. the test isn’t linked with a specific product.’, I created the second test which random picks item from gift category.

*@Test2*

Scenario: User journey of purchasing flow for a random item on the John Lewis site.

Which reuses all steps except choosing the product [created a new step to choose random product from Gift category].

Along with this two BDD test scenarios, I’ve also added **one simple N-unit automation test** [which is self-isolated and uses simple logic to perform the same]. This test case is located under this folder:



Finally, the **second additional bonus points requirement**: ‘*How can we analyse the performance of the test?’.* This part is a bit ambiguous:

* Performance of the automation test? (or)
* Performance of actual application under test?

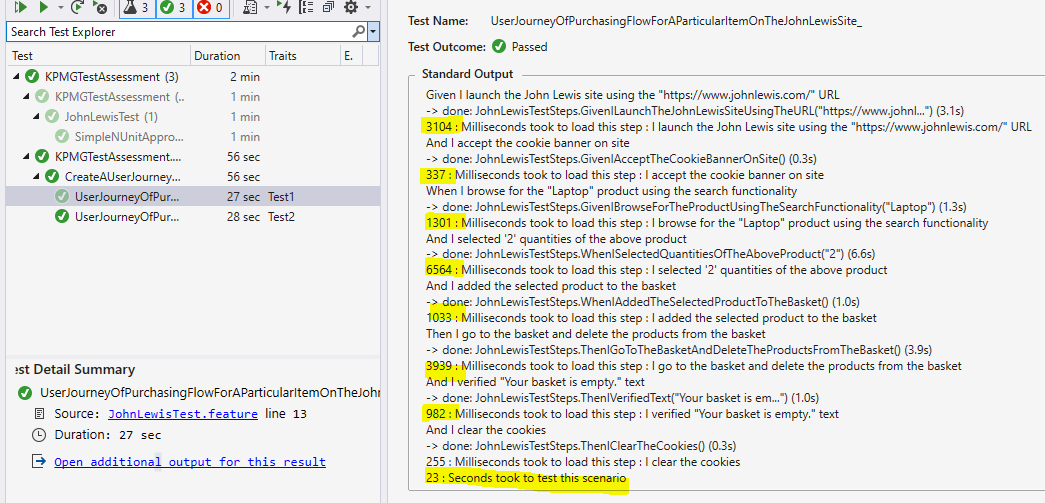
If it’s the performance of automation test, its straight forward as we can extract the test execution time from Test explorer.

If it’s the web performance of actual application then we can extract information at different level using the JavaScript, like: Using the Navigation Timing calculate the loading of page [window.performance.timing.domComplete - window.performance.timing.navigationStart] – but its include the client/user side rendering timings

(or)

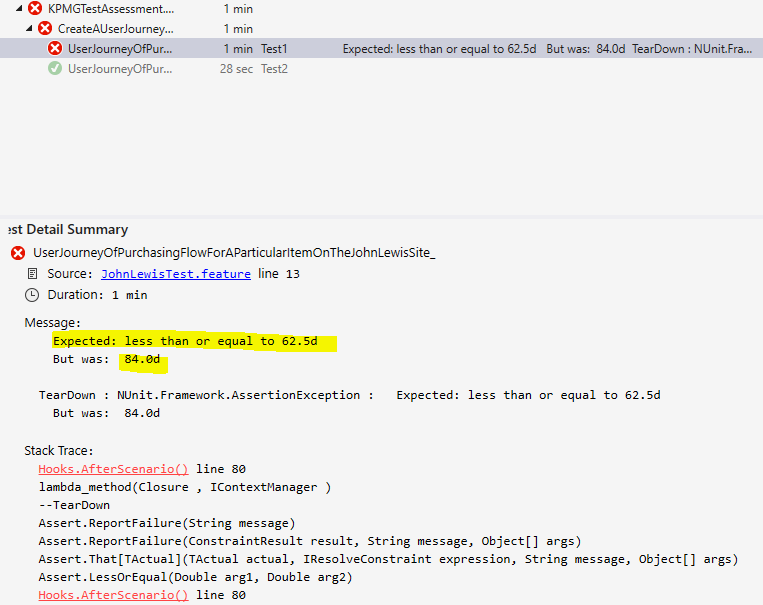
Using selenium 4 and access the dev tools and extract the information like firstPaint & startTime to measure the web performance. Again, it doesn’t provide full execution time and also the Selenium 4 is still in alpha stage.

So, I implement simple logic to capture the overall execution time *[this includes the browser rendering and user input timings]* for each step using the SpecFlow hooks along with the overall scenario execution time and asserting it not breaching the NFR as shown below.



Logic that assert overall execution time and fails the test if it breaches NFR:

I ran the test in debug mode and paused it to demo the assertion logic:



If we want to capture the web performance of application under load we can achieve it using the J-Meter + Selenium using the plugin [Selenium/WebDriver Support] and reuse most existing script.