# CA1 – Lab 2

## Introduction:

This lab is aimed at exploring test frameworks and supporting software as part of a DevOps pipeline. It will explore options to incorporate testing frameworks into the DevOps pipeline using industry standard tooling and principles focused on delivering quicker feedback in the software development lifecycle with the end goal of producing better quality software more efficiently than older software development lifecycle models. It will illustrate the use of the tools and also a high level conclusion on the system as a whole and the value it delivers under DevOps as well as a personal reflection on the learnings and discoveries made in completing the lab.

## Aims/Objectives:

* Research and explore available solutions for testing in the DevOps pipeline
* Choose a suitable tool
* Illustrate the use of this tool in a project
* Demonstrate the value produced by the selected tool based on reports generated from the system with the tool incorporated.
* Explore the concept of testing in the DevOps and Software Development Lifecycle.
* Gain a better understanding of the different types of testing tools and where they fit in at the various stages of software development and the DevOps pipeline and the value they bring to each stage.
* Draw up a conclusion on why it is important to incorporate these ideas into the pipeline and the benefit they have over not having it as well a comparison to using older testing methodologies.
* Explore the concept of “shifting left” in regards to testing and illustrate this understanding in the conclusion

## Method:

* Researched testing frameworks in DevOps and categorised different tools for different areas and stages of the pipeline
* Decided on a language and framework that would be used as a pre-requisite to choosing a testing framework.
* Chose two tools that could be used at various stages of the pipeline, where one would be integrated into the pipeline and another would be used as a comparison to explore the differences in available frameworks and where each one would fit into a pipeline.
* Created a repository to host the solution that would be used to integrate the chosen testing framework.
* Created a C# ASP.Net Web App using the default Visual Studio 2022 template targeting .Net 7
* Followed Microsoft Learn tutorial cited in conclusion to build up a working minimal API to be used for testing.
* Downloaded Postman desktop app to use for testing API.
* Created a collection in Postman to store tests using different HTTP methods on the developed API
* Ran Postman integration tests on the API as the solution was being developed further
* Created a xUnit Test project in API solution to include unit testing
* Created a unit test to use with the Test Explorer in Visual Studio

## Results:

* There is a wide variety of tools available for testing in different areas and different languages all providing different features at different price points.
* There is a lot of freely available options for testing in .Net mostly categorised into unit testing, integration testing and load testing
* This lab covers automated testing instead of manual testing
* Automated testing does not necessarily mean the test runs automatically in a pipeline but it can mean that a suite of test scripts can be created once so they can be ran again and again by a tester without having to recreate the entire test each time testing takes place.
* Good planning is required to determine where to include test automation and what should be automated in testing.
* There are many factors that can influence this not limited to the frequency of tests, number of configurations a specific test can have and the duration of tests which should all be considered when deciding what tests to automate.
* The chosen tools, Visual Studio 2022 and Postman, were very easy to set up and configure a project to be tested and the tests respectively.
* There is help available online for most frameworks and testing integrations and it some of it is provided out the box with IDEs once tests are made available to a project, in this example Test Explorer in Visual Studio.
* Postman can be downloaded locally which allows local API testing during development which provides quicker feedback to the developer than if the code had to be deployed to a remote endpoint.
* The Postman tests generated in the app can be exported as JSON and used in the command line tool Newman (Postman’s CLI tool) and stored in version control systems to track changes in tests over time.
* Postman can also do unit testing on the API, as well as Integration Testing.
* xUnit felt easier to maintain in source control and integrates better with Visual Studio Test Explorer for unit testing.

## Conclusion:

This lab was focused on researching available testing frameworks that would allow automated testing on an application or piece of code and implementing it into a solution where the results could be observed and verified. It was interesting to note the differences in testing methodologies such as functional and non-functional testing, as described in Software Testing Methodologies (Smartbear, 2022), where they described functional testing as testing on the business requirements of the application with unit testing, integration tests etc. versus non-functional testing as testing on the operational aspect of the application in terms of security, performance etc. This lab incorporated functional testing methodologies as it used both unit testing and integration testing with xUnit and Postman respectively. The brief asked to choose one tool to incorporate automated testing so research was conducted on a suitable tool as well as a suitable framework for the application that could incorporate this tool and also what type of testing would be done with the tool. There are many different tools for different frameworks but at a high level when you follow the listed functional testing areas you can see it is split between unit, integration, system and acceptance testing. A C# solution written in .Net 7 was the chosen solution for the application and Postman was chosen as the testing framework. A comparison of automated testing tools, (Katalon, 2022), provided a comprehensive list of the most popular tools and some questions suited to choosing the tool. Postman was a good choice as it had the required features, it can be used by developers or QA, it could be integrated into a pipeline in future, it is easy to maintain the testing scripts, it did satisfy the budget constraint as it is free for limited use and there is support for it freely available online through various online communities. It is also something I had previous exposure to but with a basic understanding of its use and this appealed to me as one of the personal objectives of this lab was to gain a better understanding of these tools and a better working knowledge of how to use them in a solution which could be developed further.

I enjoyed the lab as it gave me an opportunity to understand and tackle a problem I have struggled with for years in my career, as a qualified developer who was hired into an operations role at the start of my career I never had much opportunities to advance my skills as a developer any further than the basic education received at post graduate level and producing simple applications to aid operations. Although over my career I have managed to expose myself to all areas of the business and software development lifecycle, gaining a deep understanding of how it all works together, our process is waterfall and my place in the pipeline was normally gathering requirements from our clients, maintaining the in house legacy build system when it failed during development, packaging the build artifacts when they were signed off by QA and then deploying them to production, rinse, repeat. Due to the fact the majority of our legacy products are developed in a waterfall methodology, I often found that all aspects of testing and QA were in a grey area for me and testing always happened later in the development lifecycle, sometimes even weeks after code was committed to version control and was built. Depending on how resources were scheduled QA could happen before operations packaged the software for production meaning either QA work was doubled as tests needed to be run again on the package, or no further testing was done as the code had already been tested prior to packaging and was deemed operations responsibility now. From an operational perspective this was a nightmare as packages could be misconfigured compared to what was tested and often issues in production were very costly for the team as it would need to go through the whole process again with each change and normally with the question being asked “Was this tested?”.

In a white paper I found during my research, (Postman, 2022), it described a “shift left” in terms of automated testing and this is something that very much appeals to me as it is a solution to the problem above with my own teams and to many other teams who are dealing with this every day. Although on its own moving the QA process left will not solve the problem, my understanding of this is that we would be able to replace or reduce the need for manual testing if we were to integrate more automated testing earlier in the development lifecycle. This automated testing comes in the form of unit testing and integration tests that can be written by the developer and ran at build time so that issues are caught sooner and can be addressed before moving on to the next stage of the pipeline. This is invaluable to the business and is a measurable thing where you can see reduction in the time spent on rework of the code, reduction in the time spent on manual testing and an increase in productivity because previously tied up resources now have more time available to work on other areas such as further automation of the testing suites or other high value areas. This works in conjunction with methodologies like Agile which promote quicker feedback loops in the development lifecycle in comparison to waterfall where feedback can be much slower.   
  
During the course of the research and implementation of the tool into the solution it became apparent to me there was a similar tool to Postman which I would have known from seeing it used in the past but was not aware of it’s exact use. This tool is called xUnit and it allows you to create both unit tests and integration tests in .Net as well as producing reports on the success and failure rate of both. This is very similar to what Postman provides but from reading the documentation, (xUnit 2022) and (Postman 2022), both can create unit tests and integration tests for .Net. xUnit.net, as the name implies, is exclusive to .Net but it is free and open source so there is no charge to use it no matter how extensively you implement it into your code. It works well with the Test Explorer in Visual studio and provides comprehensive documentation on how to integrate it into your app and write tests, again with a strong online community across various forums. Although one might be available, it does not appear to have a visual element like Postman has with it’s desktop app. Another thing of note is that Postman will integrate with any API despite what it is built in as it accesses the API using REST so it would not care about the underlying architecture as long as the API conforms to a level of the REST standard. In comparison, xUnit would require the developer to write the API test in C# so it is limited to this. Postman also seems much easier to use and intuitive for a non-developer/QA person who might not have the knowledge to develop tests in C#. This is important as one of the requirements initially was that the solution could be used by people in a developer or QA role. In saying all of that, I find that the two tools compliment each other if used together as xUnit provides a nice platform for a developer to create and run their own unit or integration tests directly in the IDE while Postman allows its own tests to be extended further and ran later on in the pipeline or manually against a running API. Postman allows for sequencing of the tests also so a QA engineer could write a suite of API tests that would simulate a user on the system creating data, modifying data, querying the data, deleting the data etc. in a sequence that might be expected by the end user. These sequences or collections can be exported from Postman and kept in source control so QA teams can collaborate across a project and also the exported JSON can be used with Postman’s CLI tool Newman. Newman can be integrated into a CICD pipeline so that tests can be ran at any stage here too.

If I refer back to the whitepaper cited previously, (Postman 2022), there are some other interesting sections which are of interest to me around automated testing, more specifically about which tests to implement and where, and it lists some common considerations when answering these questions. One of these considerations is about the frequency of the testing and it argues that tests that are ran frequently and can be automated are great candidates for automation. In my experience one of the issues with test automation on smaller, less frequent things is that the effort to automate the test can cost more time than to just run the test. This applies to other workflows too but I believe that in the case of testing like this that you will want repeatable results and often in manual testing there is a human element that can cause discrepancies in the results where a step might have been missed or some other factor. Also, you might only run an automated test daily through the likes of Postman because it requires effort or a context switch from another task you are doing where if the test was automated into a pipeline and on a schedule you might schedule it to run more frequently than before which offsets the initial cost of automation. It is important not to confuse pipeline automation vs automation in the sense there is a script or set of scripts created that will automatically test something which is another thing this lab has highlighted for me. The report also factors in duration of the tests, if a test takes a very long time for someone to run manually then it should be automated. Smoke testing, quality gates etc. are all frequently repeated but also high value tests that would want to be ran on every release or at least every major release and should be automated.

Another thing, from my own experience with testing, is that in modern day software clients have become much more savvy on what is being delivered to their devices and depending on the industry or the scale things like automated testing and specifically the generation of reports from the automated testing facilities are being requested more and more. It provides a seal of approval as such when you produce a piece of software that has ran through your development pipeline and passed all of the tests before being delivered into UAT or a production environment. It gives some reassurance of the quality of the software but it also provides a baseline where if an issue is found in production then the tests can be scrutinised and extended to cover the newly found scenario if necessary. This all helps to build resilience into the software and helps minimise the chances of re-introducing an issue that might have been previously fixed but manually tested on that release and not in subsequent ones. This can be very embarrassing for the development team and the business and in not addressed properly can be detrimental to customer relationships.

## References:

[15 Best Automation Testing Tools & Frameworks | List for 2022 (katalon.com)](https://katalon.com/resources-center/blog/automation-testing-tools)

[Software Testing Methodologies | SmartBear](https://smartbear.com/learn/automated-testing/software-testing-methodologies/)

[Tutorial: Create a minimal web API with ASP.NET Core | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api?view=aspnetcore-7.0&tabs=visual-studio)

[How to unit test Minimal APIs in .NET 6 (and why it's hard) - YouTube](https://www.youtube.com/watch?v=VuFQtyRmS0E)

[Microsoft fixed my biggest complaint with Minimal APIs in .NET 7 - YouTube](https://www.youtube.com/watch?v=-i4rP0LGY5U)

[Test Minimal API apps | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis/test-min-api?view=aspnetcore-7.0)

[Testing in .NET - .NET | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/core/testing/)

[AspNetCore.Docs.Samples/fundamentals/minimal-apis/samples/MinApiTestsSample at main · dotnet/AspNetCore.Docs.Samples (github.com)](https://github.com/dotnet/AspNetCore.Docs.Samples/tree/main/fundamentals/minimal-apis/samples/MinApiTestsSample)

[Debug unit tests with Test Explorer - Visual Studio (Windows) | Microsoft Learn](https://learn.microsoft.com/en-us/visualstudio/test/debug-unit-tests-with-test-explorer?view=vs-2022)

[Automated API Testing | Postman](https://www.postman.com/automated-testing/#:~:text=Postman%20can%20be%20used%20to,human%20error%20and%20streamlines%20testing.)

[automated-testing-whitepaper-postman.pdf](https://voyager.postman.com/doc/automated-testing-whitepaper-postman.pdf)

[Minimal APIs in .NET 6 but where are the Unit Tests? - Scott Hanselman's Blog](https://www.hanselman.com/blog/minimal-apis-in-net-6-but-where-are-the-unit-tests)