Cardiff Metropolitan University | Prifysgol Metropolitan Caerdydd

DevAppDeploy: A Platform for Internal Mobile Application Distribution for User BETA Testing

DAT6001 WRIT1

Kyle Cuthbert

ST20249341

Date: 11/06/2025

Declaration

Student Declaration In Respect of Individual Work

I declare that the whole of this work is the result of my individual effort and that all quotations from other authors have been acknowledged. This dissertation is submitted in partial fulfilment of the requirements of Cardiff Metropolitan University for the Degree of Bachelor of Applied Software Engineering.

Signed: Kyle Cuthbert Date: 11/06/2025

Abstract

With the retirement of Microsoft's Visual Studio App Center, a gap has been created in the software development lifecycle for mobile applications, particularly for Cardiff Council's internal beta testing process. This report showcases DevAppDeploy, a prototype system based on App Center's distribution functionality that will allow for the controlled release of Android mobile application versions for the purpose of user quality assurance testing. The project follows a structured process to identify software development methodologies and technologies, which involves conducting a literature review before carrying out requirement analysis, system design, implementation, and evaluation. The system features include secure authentication, role-based access control, package file storage, and application version management. The project integrates cloud technologies, web frameworks, and database management to showcase a comprehensive prototype that aims to be scalable, secure, and user-friendly. While focusing on Android applications, the prototype has scope for future improvements and enhancements, such as integrating iOS applications, implementing CI/CD, and utilising analytics. The outcome of the project is a feasible internal distribution tool that can be tailored and evolved to make DevAppDeploy a valuable asset for maintaining an efficient beta testing workflow.

Table of Contents

List of Abbreviations

Abbreviation	Definition
CI/CD	Continuous integration and continuous delivery
SDLC	Software Development Life Cycle
SMART	Specific Measurable Achievable Relevant Time-Bound
UML	Unified modelling language
GUI	Graphical user interface
UI	User interface
CRUD	Create, Read, Update and Delete
Blob	Binary Large Object
SaaS	Software-as-a-service
PaaS	Platform-as-a-service
IaaS	Infrastructure-as-a-service
GCP	Google Cloud Platform
AWS	Amazon Web Services
DOM	Document Object Model
MSSQL	Microsoft SQL Server
GUID	Globally Unique Identifier

List of Figures

List of Tables

1. Introduction

Mobile application development is a complex and constantly evolving software engineering discipline, with efficient beta testing playing a crucial role in ensuring the quality of applications on various devices. Microsoft App Center is a tool that is widely within mobile application development to assist developers with distributing and testing beta versions of applications, along with other essential offerings such as crash analytics and releases to app stores. Its planned retirement has created a gap within the software development lifecycle for mobile application teams that rely on it.

This final year work-based project showcases DevAppDeploy, a prototype platform that is based on Microsoft App Center's distribution functionality that could be used for Cardiff Council's internal beta testing of mobile applications. The platform aims to facilitate controlled releases of mobile applications, enabling quality assurance testers to evaluate various versions of the applications on their test devices. Although there are alternative tools that can offer distribution functionality, they cannot roll back to previous versions or distribute multiple versions for parallel comparative testing.

The project will employ a structured approach to integrating software engineering best practices, developing a robust, scalable, and secure system that follows a comprehensive software development lifecycle from start to finish. This approach will enable the showcase of a fully implemented prototype solution. By conducting thorough research, analysis, design, and implementation, DevAppDeploy will deliver a viable solution that can be refined, tailored, and evolved to meet the changing needs of Cardiff Council's digital services team.

1.1 Background

App Center is a cloud-based product offered by Microsoft that includes tools to manage the development lifecycle of mobile applications. These tools include mobile app analytics and monitoring, crash reporting, release management and distribution for beta testing (Microsoft, 2025).

App Center allowed development teams to distribute application builds for mobile and desktop applications, either manually or automatically from code repositories such as Bitbucket, GitHub and Azure DevOps. This allowed beta versions to be tested by quality assurance teams before being released to the stores.

Regarding distribution for beta testing, a member of the development team will create an application with various details, such as the operating system (e.g., Android or iOS). These applications will have a collection of releases within each application. From the releases, testers can then download the build file for a particular release directly to their test device. The following figures show the pertinent screens for this functionality.

Figure 1 shows a screenshot of various mobile applications within App Center.

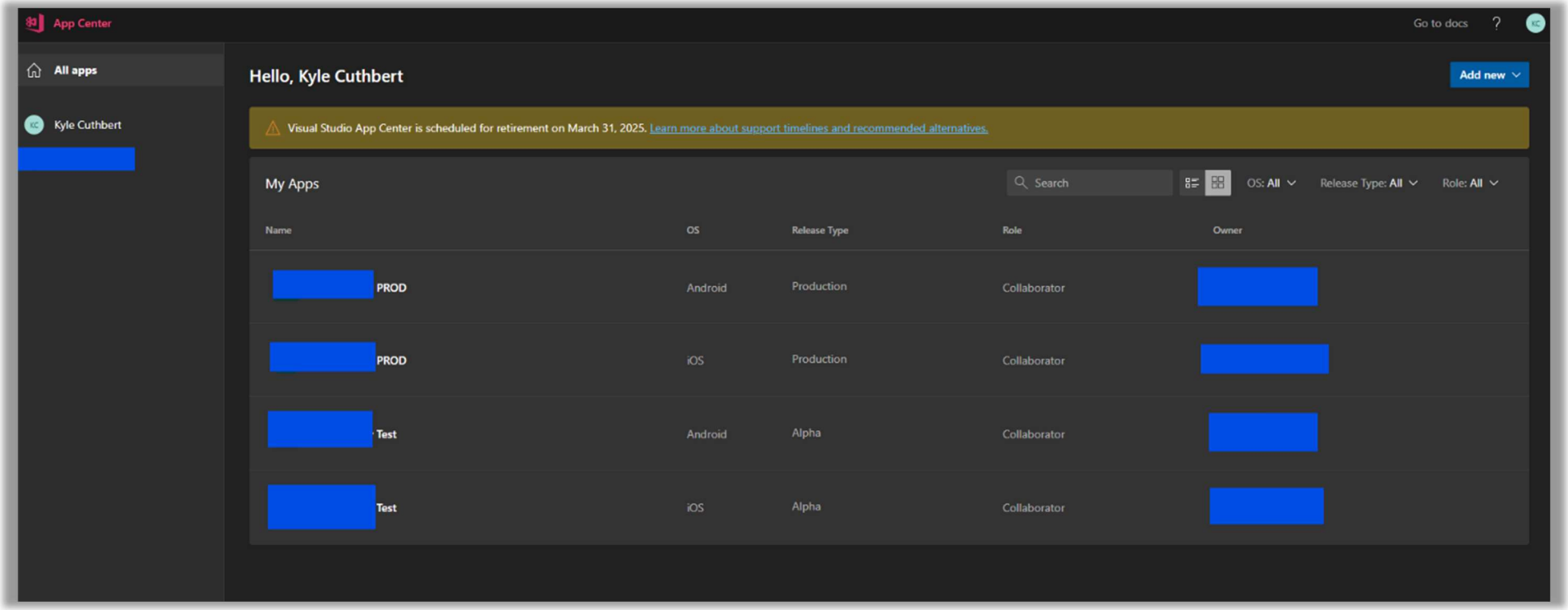

Figure 1 App Center showing mobile applications

Figure 2 shows a screenshot of different releases for one of the applications above.

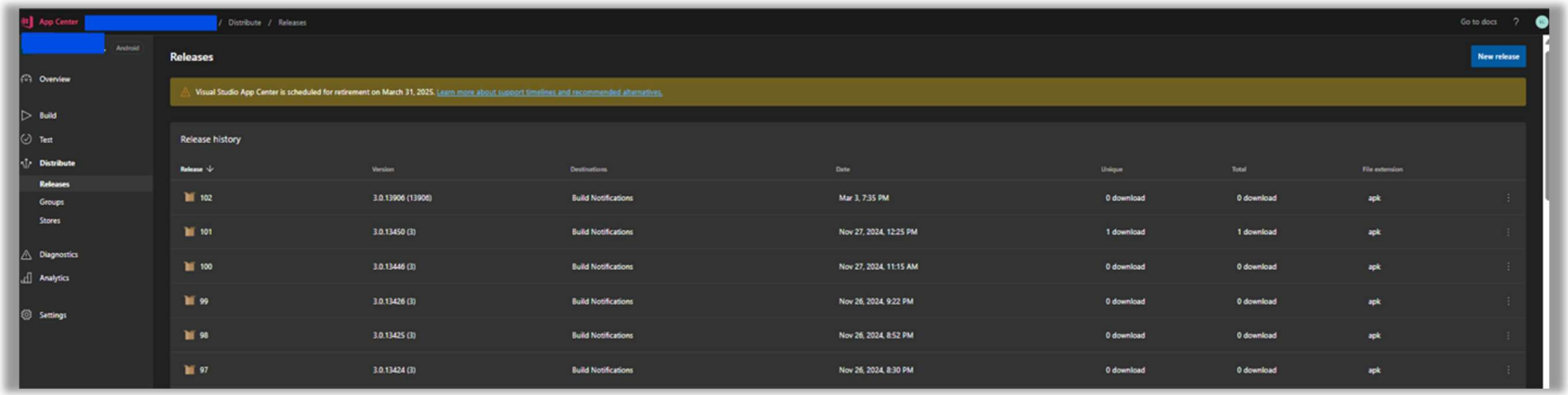

Figure 2 App Center releases for an application

Figure 3 shows a screenshot of a specific release within App Center that can be downloaded onto a device.

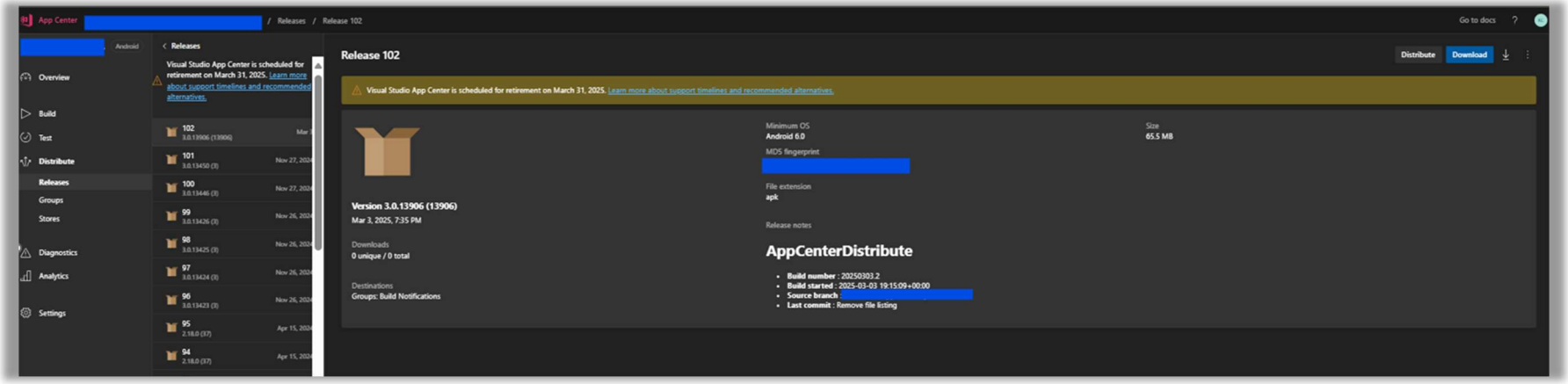

Figure 3 App Center release details

1.2 Problem Statement

Cardiff Council is a local authority whose current process of distributing beta versions of its Xamarin or .NET MAUI mobile application(for both iOS and Android) involves using a tool called Visual Studio App Center(also known as App Center).

In March 2024, Microsoft announced that it would be retiring App Center on 31st March 2025 (Microsoft, 2025), and would not be replacing the service with a replacement that encompasses all the functionality offered by App Center; instead, Microsoft offered workarounds for each of the separate functionalities (Ramel, 2024). Figure 4 shows a table displaying all the suggested workarounds.

App Center capability	Recommended alternative solution
Build	We recommend migrating your builds from App Center to **Azure Pipelines**, leveraging the **Export App Center Build feature**.
Test	For app device **testing**, we recommend **BrowserStack App Automate**. BrowserStack provides access to 20,000+ real iOS and Android devices. BrowserStack has developed the Device Testing CLI to support migrating from Microsoft App Center to BrowserStack App Automate. You can find the full guidance for migration to BrowserStack **here**.
Distribution	We recommend Apple's **AppStore** for iOS app production releases and Apple's **TestFlight** for iOS app test releases. For Android applications we recommend **Google Play** for production releases and Google Play Console for test releases. **Azure Pipelines** tasks can be used for **distributing** to **AppStore/TestFlight** and **Google Play**. For Microsoft Store applications, we recommend using **Package Flight**s feature available in Partner Center.
Code Push	We have prepared a special version of **CodePush** to integrate into your app and run independently from App Center. If you'd like to get access to the codebase of this CodePush standalone version, please reach out to our support team at **support@appcenter.ms** for more information.
Analytics & diagnostics	We recommend one of the Azure Native ISV services which provide rich capabilities for mobile **analytics and diagnostics**. By leveraging these Azure Native ISV services, you will be able to monitor your complete stack from device to your backend infrastructure on Azure. Click on each link below to see the documentation for Azure Native ISV service of your choice. **Azure Native ISV Services**: • **Datadog** • **Dynatrace** • **New Relic** Microsoft Store app developers can access analytics & diagnostics data through **Partner Center Dashboard**. Additionally, UWP applications published in Microsoft Store can log custom events through the **Microsoft Store Services SDK**.

Figure 4 Recommended Tools to Replace App Center Functionalities (Ramel, 2024)

Whilst these alternatives will allow the life cycle to continue, it has been identified that TestFlight and Google Play Console's offerings for distribution functionality for beta testing are not as comprehensive as App Center. One such drawback is that these alternatives do not store older versions of the application, so they are not available to app testers to roll back to previous versions. This ability means the testers cannot test the user's experience when upgrading from one version to another. Another drawback would be that testers can't test multiple different versions of the application at the same time via different branches.

1.3 Motivation

The leading motivation for undertaking this project is the retirement of fundamental services offered by App Center, specifically the distribution feature, which allows app testers at the local authority to install versions of the mobile application onto their test devices to conduct beta user testing.

This functionality is integral to identifying bugs in the mobile application's code, highlighting usability issues, and providing feedback from a quality assurance perspective.

With App Center's discontinuation, a gap in the software development life cycle has been created. Still, it also offers the opportunity to build a new internal tool that can evolve with bespoke functionality tailored to the needs of the local authority's digital services teams.

This will initially allow the local authority to:

- Retain control of the distribution of mobile application versions.
- Maintain continuity of internal testing.

Future opportunities for this tool will include:

- Future potential to grow the tool with functionality such as CI/CD from development to testing by sending the APK/IPA directly from the pipelines.
- Create analytics dashboards and implement custom crash reporting.
- Allow for builds to be sent to 3rd parties, such as accessibility testers from external governing bodies.

From a developer's perspective, this project presents an exciting opportunity and poses a challenge in recreating essential functionality by building a bespoke solution from scratch. The tool can be created as a prototype for this project, but it can then grow and evolve into a tailor-made, in-house tool that addresses further organisational needs.

1.4 Scope of Project

This project's scope is to develop a functional prototype, from start to finish of the SDLC, of a tool that internal developers at Cardiff Council can use for distributing mobile applications for beta testing purposes. At this stage, the project will not include build, test, analytics or crash reporting functionality.

The key deliverables for this project will include the following:

- Secure login functionality along with role-based access rights to certain views and functionality.
- Authorised personnel will have the ability to upload new application package files. With the focus of this project being the Android operating system, these files will be in the .apk format.
- The package files for each version will be uploaded and stored securely to enable the installation of old builds on devices.
- An accessible way for the quality assurance teams to view and download versions of the mobile application files to their Android test devices.

This project will exclude the upload of iOS package files, with the focus being on creating a prototype to enable Android application uploads.

2. Aims and Objectives

This section outlines the project's aim and specific objectives, which aim to result in the successful development of a tool that can replace the distribution functionality void left by the recently retired App Center. This will address the current gap in the local authority's mobile application development life cycle.

2.1 Aim

The primary objective of this project is to design, develop, implement, and evaluate a web application that is both secure and user-friendly, making versions of the mobile application easily accessible.

The system will enable authorised users to upload and manage beta versions of the local authority's mobile application for the Android operating system, facilitating quality assurance testing on physical Android devices.

2.2 Objectives

To achieve this aim, specific objectives following a SMART approach will need to be in place to guide the project.

1. Conduct a literature review – A comparative analysis that will focus on pertinent elements of the project, such as SDLC, cloud technology platforms, version control, web frameworks, database options, blob storage, and authentication. This literature review will transition to a methodology where the chosen elements will be discussed.
2. Requirement analysis – Gather and document requirements for both functional and non-functional purposes. Functional requirements will relate to the system behaviour, such as package upload, package download, and version management. Non-functional requirements will include aspects such as security, usability and performance. Performing this requirement analysis will ensure that the application addresses the needs of the local authority.
3. System design – This will include creating detailed design artefacts, including UML diagrams such as Use Case, Activity, Class, Sequence, and Entity-Relationship. These UML diagrams aim to model the system structure and behaviour. In addition to the UML diagrams, wireframes will be created to show the proposed design of the GUI, which will define the user experience.
4. Implement functional prototype – This will include writing code to implement all the functionality gathered as part of the requirement analysis, and all the steps from version control through to creating CI/CD pipelines to automate build, testing, and deployment.

5. Conduct system testing and quality assurance – Various testing methods will be employed to ensure the system is working properly, including user testing, unit testing, and load testing.
6. Evaluate the system – The system will be evaluated to identify what is working well and areas that can be improved or functionality that can be implemented in the future.

4. Literature Review

4.1 Introduction

This section aims to identify and establish the theoretical framework that will underpin the development project of the mobile application release management tool. A literature review will be conducted to examine key technologies and methodologies employed throughout the project life cycle, drawing on past research and best practices related to Software Development Life Cycle (SDLC), cloud computing, version control, the chosen development frameworks, cloud data storage, and blob storage.

Finally, a conclusion will be drawn to illustrate the technologies and approaches chosen to progress into the project's requirements analysis and design phase.

4.2 Software Development Life Cycle

SDLC is a methodology for planning, analysing, designing, developing, testing, and deploying software, and there are many different SDLC methodologies to choose from, with each having its advantages and disadvantages (Ikbal Hossain, 2023). SDLC allows for a systematic approach to progressing through the development process promptly, with the goal of each methodology being to minimise the risk of failures while also maximising the quality of the system or product (Gurung, et al., 2020).

4.2.1 Waterfall Model

The **Waterfall Model** is one of the earliest SDLC models, which follows a linear sequential flow downward through the various stages of the life cycle, as can be seen in Figure 5 (Gurung, et al., 2020).

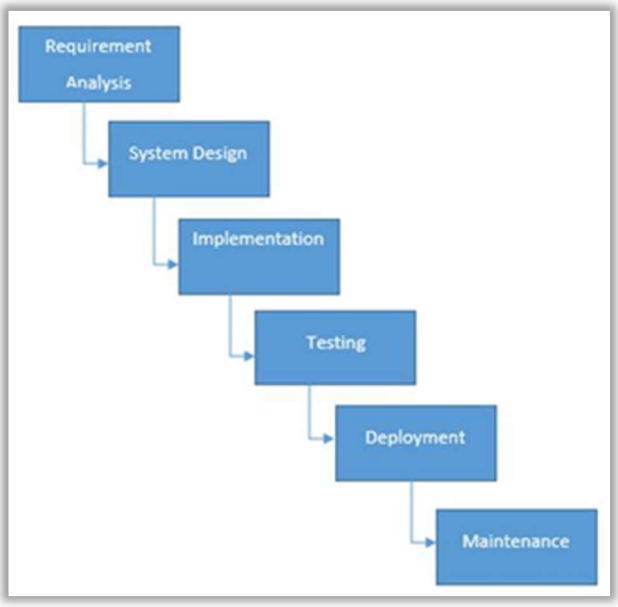

Figure 5 Waterfall Model (Gurung, et al., 2020)

With the waterfall model, each phase of the SDLC must be completed before cascading down to the next phase (Saeed, et al., 2019), the approach does not allow the process to go back to a previous stage when there is feedback or changes to the requirements, making it too rigid to cater for revisions (Gurung, et al., 2020). While this model is well-suited for small projects or projects with well-defined specifications that aren't likely to change, it is often considered too inflexible for most software projects and requires each phase of the model to have extensive documentation to record the progress (Ikbal Hossain, 2023).

4.2.2 Iterative Model

The **Iterative model** is a methodology that builds on the waterfall model by overcoming its shortfalls by allowing requirements to be gathered at every phase to allow for feedback from the client and stakeholders (Alshamrani & Bahattab, 2015). This allows for more flexibility, enabling developers to add new features with each iteration based on the feedback provided, thereby increasing stakeholder engagement and involvement. However, the iterative model does have drawbacks. The model increases the complexity of managing activities and coordinating resources, scope creep that can lead to delays, and excessive documentation, causing more overheads of developer time (Ikbal Hossain, 2023). Figure 6 shows the iterative model phases and flow.

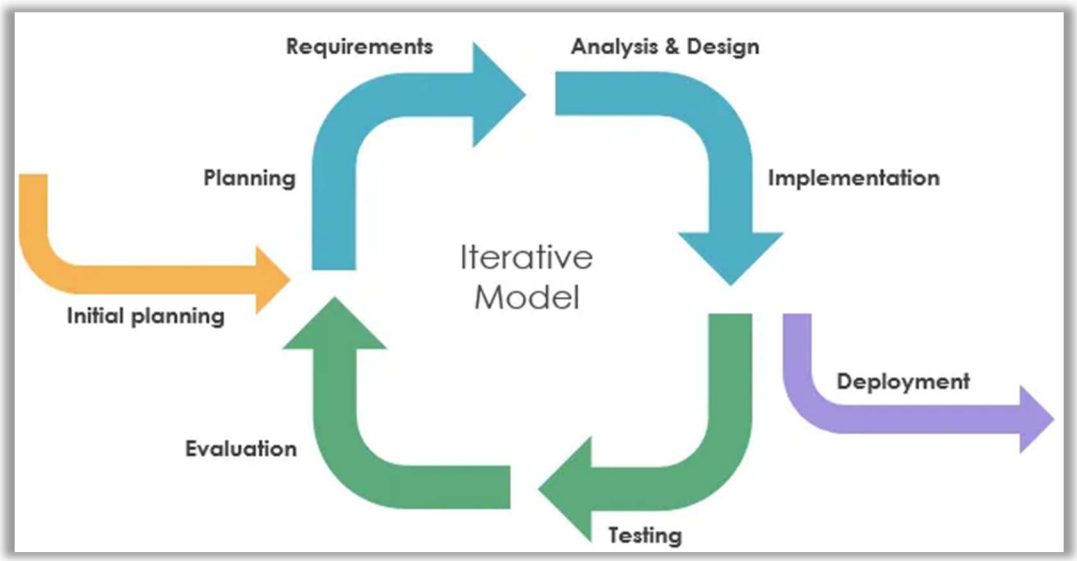

Figure 6 Iterative Model (Jin, 2023)

Other incremental models exist that build upon the iterative model, such as the **V-Model**. The V-model also verifies and validates progress after each phase but can result in costly adaptations to requirements during each phase (Balaji, et al., 2012).

4.2.3 Agile Model

In contrast to the previous models, the **Agile Model** was created to ensure that a project can adapt quickly to changes (Gurung, et al., 2020) thus, resulting in quicker project completion. While following an iterative approach where development happens in smaller increments within a given period(also referred to as sprints), it also encompasses other methodologies such as Scrum, Extreme Programming and lean development (Gurung, et al., 2020). Agile promotes close collaboration between the stakeholders, including the client and the developers, with pair programming often being used to ensure code quality. With Scrum, the project phases will be split into sprints, accompanied by regular team meetings (sprint planning, daily stand-ups, and sprint retrospectives), allowing for the progress of the development project to be monitored and tracked. Figure 7 shows the steps in an Agile SDLC.

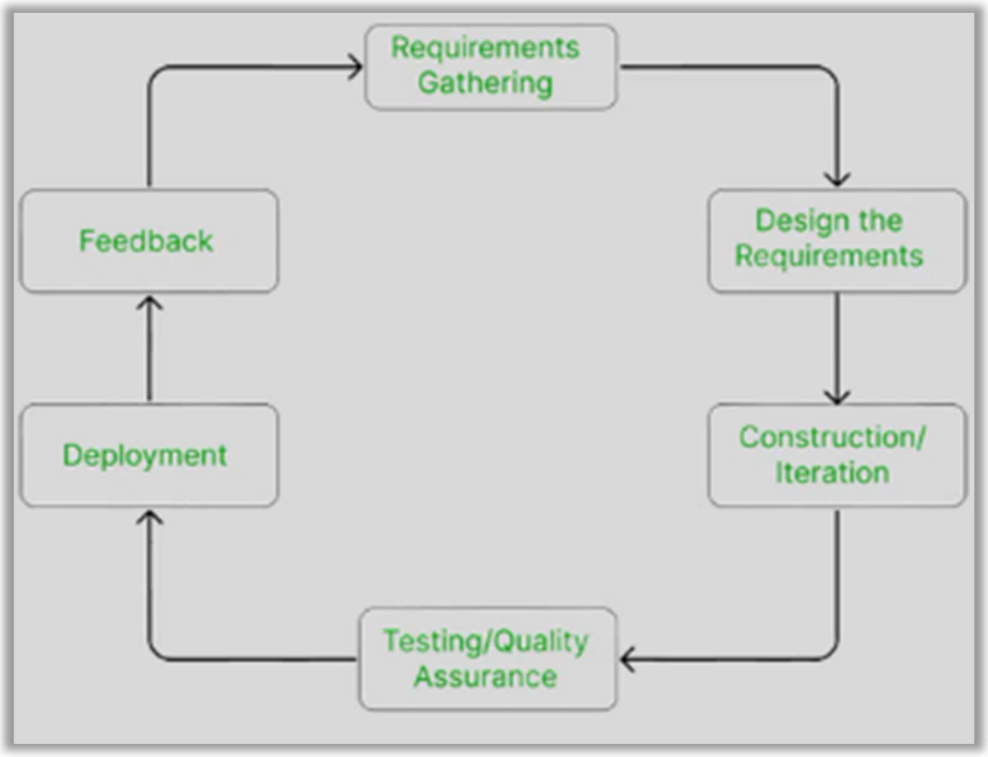

Figure 7 Agile Model (GeeksForGeeks, 2025)

The agile methodology is particularly suitable for projects that are anticipated to evolve with amended requirements being incurred frequently, allowing the project teams to respond quickly (Ikbal Hossain, 2023). One of the challenges that (Ikbal Hossain, 2023) highlights is managing stakeholder expectations and ensuring the coordination of the team this is also backed up by (Balaji, et al., 2012). A decision matrix has also been included to help determine which methodology is best suited for a project, as shown in Figure 8.

Aspect	Traditional SDLC	Agile Methodology
Project Approach	Sequential and linear.	Iterative and incremental.
Requirements	Often fully defined upfront, with limited flexibility for changes.	Embraces changing requirements and encourages flexibility.
Phases	Distinct, sequential phases (e.g., planning, design, development, testing, deployment).	Iterative phases, with overlapping development and testing.
Documentation	Comprehensive documentation is a key aspect.	Focuses on working software over extensive documentation.
Customer Involvement	Limited customer involvement until later stages.	High customer involvement throughout the project.
Testing	Testing typically occurs after development is complete.	Continuous testing throughout development.
Project Control	High degree of control and predictability.	Less predictability, but greater adaptability.
Change Management	Changes can be costly and disruptive.	Embraces changes and accommodates them gracefully.
Quality Assurance	Strong emphasis on quality assurance through extensive documentation and reviews.	Focuses on code quality through practices like pair programming and continuous integration.
Progress Reporting	Typically requires comprehensive progress reporting and milestones.	Focuses on regular progress updates and frequent releases.
Risk Management	Risk management occurs at specific stages with limited adaptability.	Proactive risk management with continuous adaptation and mitigation.
Team Structure	Roles are often well-defined and specialized.	Emphasizes cross-functional teams with shared ownership.
Suitability	Well-suited for stable, well-understood projects with fixed requirements.	Best for projects with changing or evolving requirements, high customer involvement, and a need for rapid delivery.

Figure 8 The distinction between traditional and agile methodologies (Ikbal Hossain, 2023)

4.3 Cloud Technologies

Cloud computing has transformed how many software engineers develop, deploy and manage applications. Cloud computing delivers computing services, such as servers, storage, databases, software and analytics, remotely and on-demand (Microsoft, 2025). By being on-demand, cloud services have a pay-as-you-go pricing structure, avoiding the need to buy and maintain servers as well as offering access to other technology services (Amazon, 2025).

Cloud computing offers three distinct service models tailored to the requirements and needs of a project: 'Software-as-a-service', 'Platform-as-a-service', and 'Infrastructure-as-a-service'. A definition of these service models from (Ashraf, 2014) can be seen in Table 1.

Table 1 Cloud Service Model Definitions

Service Models	Definition
IaaS	Enables the user to rent infrastructure resources, such as hardware and software components, by emphasising resource virtualisation, allowing users to deploy their own guest operating systems in the cloud. Maintenance, administration, and deployment are the responsibility of the service provider.
PaaS	Offers a development environment platform for users to create and maintain applications. This model encompasses both development tools and infrastructure.
SaaS	This is the top layer of service models offered by cloud computing, whereby users are independent of their resources. Users will instead utilise ready-to-use software, and they don't need to manage hardware or software.

Figure 9 shows the difference between these service models:

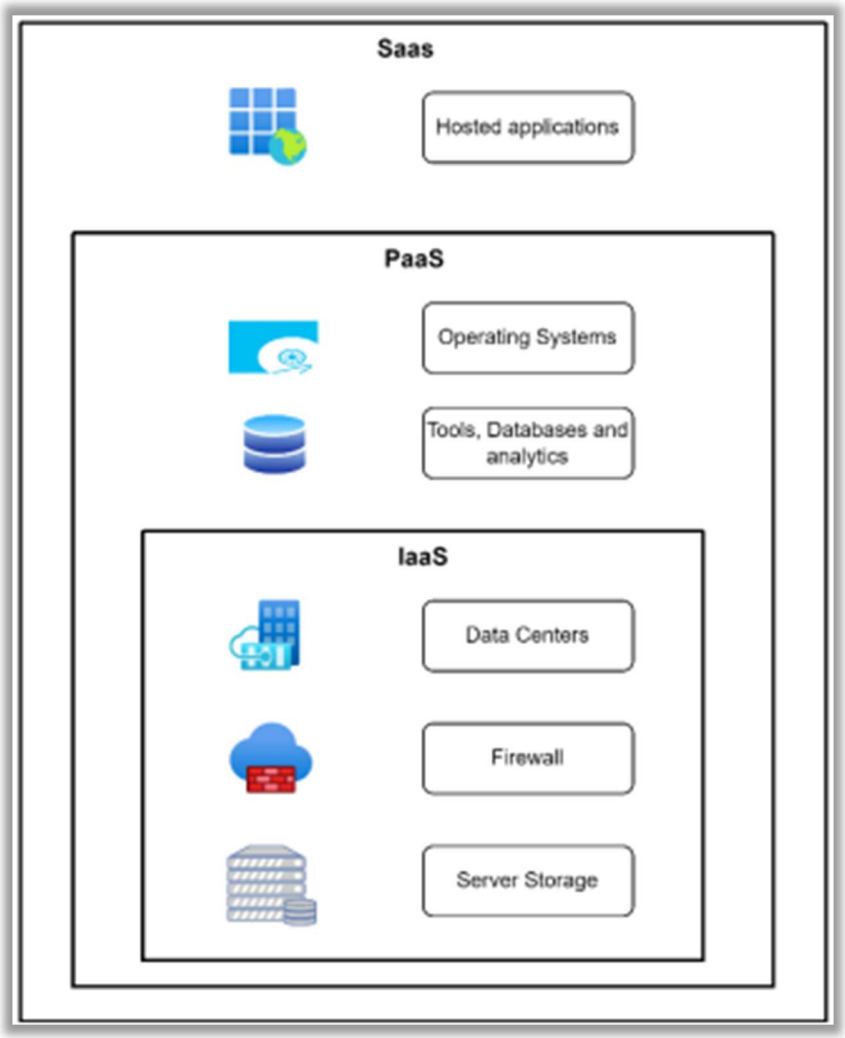

Figure 9 Differences between service models (Borra, 2024)

Many cloud service providers offer services for IaaS and PaaS, and some of the most popular include AWS(Amazon), Azure(Microsoft), GCP(Google). Figure 10 illustrates a comparison of these providers, along with their respective market shares as per (Borra, 2024).

	AWS	Azure	GCP
Launch Year	2006	2010	2008
Market Share	32	23	10
Availability Regions	30+	60+	40+
Countries	200+	200+	200+
Services	200+	200+	120+

Figure 10 Comparison of Major Cloud Service Providers (Borra, 2024)

For IaaS, customers would be effectively hiring the use of virtual machines, that will require the customer responsible for the configuration of the operating system, updating the required runtimes, updating security patches, setting up web servers and managing the scaling of the application by manually setting up and configuring additional virtual machines and load balancers (Borra, 2024).

In contrast, with PaaS services, cloud providers manage the infrastructure, so developers are only required to deploy their code. This means that the runtimes are automatically installed, security patches are managed, and the scaling is handled automatically. This allows developers to spend their time focusing on the code by reducing the burden of handling and managing the infrastructure, increasing productivity and ultimately resulting in quicker releases (Borra, 2024). Table 2 shows the services that various providers offer as IaaS and PaaS services:

Table 2 Cloud Provider IaaS and PaaS Services

Cloud Provider	IaaS Service	PaaS Service
Azure	Azure Virtual Machine	Azure App Service
AWS	EC2 Instance	AWS Elastic Beanstalk
GCP	Compute Engine VM	Google Cloud App Engine

(Younis, et al., 2024) says that PaaS is best suited for companies or developers who focus on application development, thereby removing the need to manage infrastructure or acquire the necessary skills to manage it. This is especially pertinent when creating custom applications or microservices.

4.4 Web Frameworks

Web frameworks serve as pre-written code that provides tools, libraries, and architectural structure to a project (Rathinam, 2022). By using a framework, a software project is already provided with a basis for development, helping to streamline the development process. Web frameworks promote the re-use of code and provide the foundation for building scalable and efficient web applications, which helps to reduce the time and cost spent developing the application (IONOS, 2022).

While many web frameworks are used for both server-side and client-side development, this section will compare various frameworks, including the .NET Framework, .NET Blazor, and React, to highlight the characteristics of each framework and identify their suitability for different applications and developer needs.

4.4.1 ASP.NET Framework

The **ASP.NET Framework** is a server-side framework that has been in existence since 2002. It benefits from extensive developer tooling and a large library. While still supported, **ASP.NET Framework** is no longer actively promoted, with a key restriction being that it is not cross-platform, as it is dependent on the Windows operating system (Nile Bits, 2023).

4.4.2 ASP.NET Core

ASP.NET Core is a framework that encapsulates and unifies several different web frameworks to build web applications and APIS. Unlike the ASP.NET Framework, it is open source and cross-platform (Microsoft, 2025). For UI projects, ASP.NET Core offers several options that include:

- ASP.NET Core Blazor
- ASP.NET Core Razor Pages
- ASP.NET Core MVC

Many recent blog posts and articles, such as (Anderson, 2025) note that whilst Razor Pages and MVC are still popular frameworks, Microsoft has stated that Blazor will be its main investment for web UI.

4.4.3 .NET Blazor

Blazor is a relatively recent framework offered by Microsoft as part of the ASP.NET Core family. Blazor has two specific approaches, which are server-side rendering and client-side rendering.

In the paper (Medavarapu, 2021) the author performs a series of experiments to compare the two approaches within Blazor. Table 3 summarises the outcome results that the author has provided to help developers decide on which approach would be more suitable for a given project.

Table 3 Blazor Rendering Approaches

Rendering Type	Advantages	Disadvantages
Server-Side-Rendering	• Faster initial load times since the page is pre-rendered. • Reduces computational load on the client. • Ensures users always see up-to-date content. • Beneficial for SEO Optimisation.	• Higher server load, especially under high concurrency. • Requires frequent server roundtrips, introducing latency. • Less suited for applications needing offline functionality.
Client-Side-Rendering	• Provides rich client-side interactivity. • Better scalability since processing happens on the client. • Enables offline functionality and reduces server dependency. • Less strain on the server, allowing higher user concurrency.	• Slower initial load times due to WebAssembly runtime download. • Larger payload sizes are impacting network performance. • Increased energy consumption on the client devices.

These results are further compounded by (Grace, 2021) who has also run experiments comparing Blazor WebAssembly(client-side rendering), Blazor Server, and an ASP.NET Core MVC application.

WebAssembly once again has a slower initial load time due to a lot of activity and a larger number of HTTP requests, however, it is mentioned that Blazor Server takes longer when updating page elements after performing click actions (Grace, 2021). When it comes to ASP.NET Core MVC (Grace, 2021) mentions that this framework has the quickest initial load but is then outperformed when clicks are performed, however, the author notes that it does utilise caching.

Blazor is becoming more popular, as discussed by (Fishuck & Datsyshyn, 2025) which is shown in Figure 11.

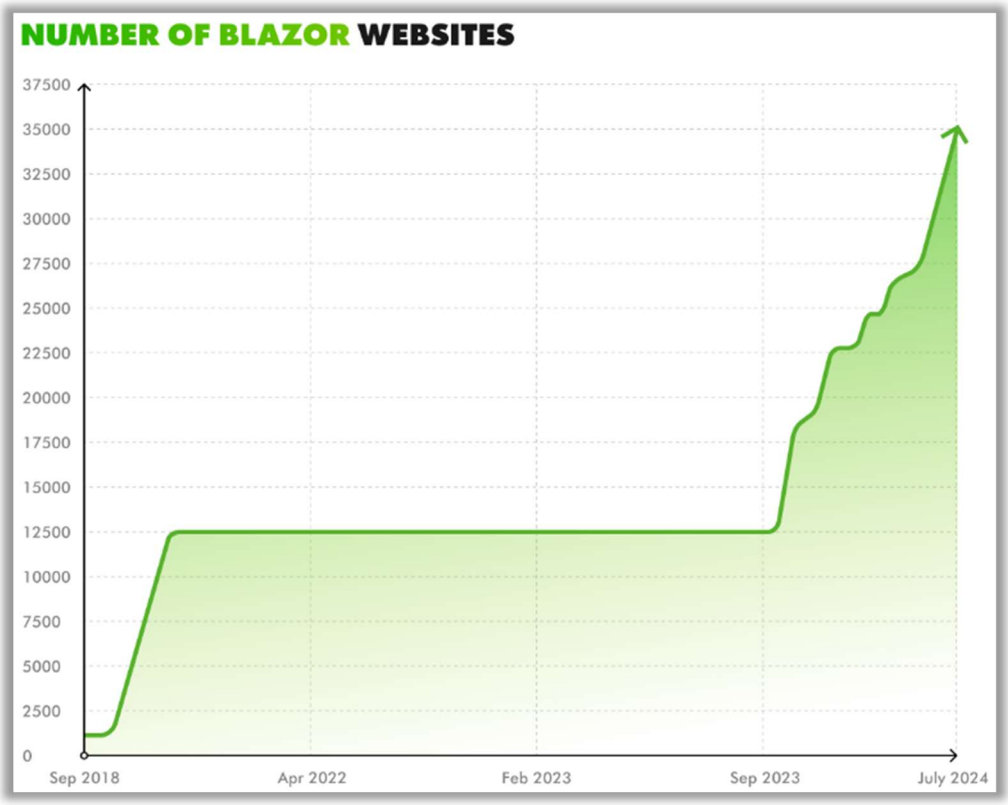

Figure 11 Number of Blazor Websites Since Sep 2018 (Fishuck & Datsyshyn, 2025)

The article attributes its increasing popularity to the fact that Blazor allows C# developers to develop both front-end and back-end applications with just C#, thereby avoiding the need to use JavaScript. Another advantage that is mentioned by (Fishuck & Datsyshyn, 2025) That is, Blazor integrates seamlessly with Azure services and supports integration with services such as authentication, database connectivity, and API services.

4.4.4 React

React is an open-source JavaScript library designed to build dynamic, reusable UI components and leverages the virtual DOM for efficient updates and supports declarative programming, making it highly performant for single-page applications and progressive web applications (Devessence, 2025).

React boasts a vast ecosystem, featuring tools such as Redux for state management and Next.js for server-side rendering. It also benefits from strong community support to ensure long-term viability. (Metha, 2025). React also integrates well with various back-end technologies and benefits from its flexibility to front-end interactivity.

4.4.5 Comparison Blazor vs React

Köping & Persson (2021) compares the React framework with Blazor based on several criteria to determine whether the Blazor framework can be considered a viable alternative to traditional JavaScript-based frameworks. Table 4 below summarises the key findings of the study:

Table 4 Comparison between Blazor and React

Criteria	Blazor	React
Lines of Code	Fewer lines of code required.	Requires more lines of code.
Cyclomatic Complexity	Easier to maintain, lower complexity.	Higher complexity could be more challenging to manage.
State Management	Uses built-in state management.	Uses established tools such as Redux.
Debugging Tools	Visual Studio.	Visual Studio Code.
Community Support	Less established but has a growing community.	Large, well-established community.
Available Libraries	Fewer third-party libraries.	Extensive mature libraries.

(Köping & Persson, 2021) Concludes that Blazor is now a viable option for web development.

4.4.6 Summary

The article (Devessence, 2025) Favours Blazor for enterprise applications within Microsoft-heavy environments due to its seamless .NET integration, and full-stack development for C# focused developer teams. This is backed up by (Metha, 2025) Who also points out that Blazor has strong integration with .NET APIs and strong backend connectivity. In the article (Prioxis, 2025) It is highlighted, however, that React has a stronger ecosystem and high-performance rendering via the virtual DOM, making it ideal for dynamic, user-centric applications that require fast UI updates.

4.5 Databases

Relational database management systems are a fundamental element of business applications. They play a crucial role for organisations as an efficient way of storing and retrieving structured data.

Two of the most popular RDBMS are Microsoft SQL Server and PostgreSQL. This section will examine and compare these two systems by discussing three critical elements that are important when choosing which RDBMS to employ.

4.5.1 Pricing

SQL Server is a product owned by Microsoft, which requires commercial businesses to pay high licensing costs that vary depending on the pricing plan the organisation is signed up to. However, Microsoft offer a free version to developers who are not using the database for production purposes (Google Cloud, 2025). Table 5 shows the up-to-date pricing for each license.

Table 5 MS SQL pricing (Microsoft, 2025)

Editions	Price(US Dollar)	Licensing Model
Enterprise	$15,123	2 core pack
Standard – per core	$3,945	2 core pack
Standard – server	$989	Server
Developer	Free	Per user
Express	Free	N/A

PostgreSQL is an open-source product that has been released under a liberal open-source license, which means it is free to use. There are no plans to change this license to any other type or pricing (PostgreSQL, 2025). This means that the product is free to use, even for commercial purposes.

4.5.2 Performance

Regarding RDBMS performance, several studies have been conducted using TPC-H and TPC-C to benchmark performance.

One such study was conducted by (Alves, et al., 2023) that used TPC-H to compare the performance of two RDBMS when handling semi-structured data. The paper finds that MSSQL performs better when inserting data, which is attributed to PostgreSQL performing validation on input, thereby reducing its speed.

When it comes to query execution, the paper finds that MSSQL runs six times faster when the queries are indexed. However, storage efficiency is better optimised for PostgreSQL, as MSSQL requires significantly more storage for semi-structured data. Overall, the paper

finds that MSSQL is faster for data insertion and indexing, but PostgreSQL is more efficient for querying and storage.

Another study that compares performance is (Vershinin & Mustafina, 2021) which compares three RDBMS, PostgreSQL, MySQL, and MSSQL. The study once again uses TPC-H to benchmark test elements of performance by generating differently sized databases and importing data with various commands. Each query was executed ten times.

In contrast to the first study, the paper finds that MSSQL has better query execution, especially for complex queries, with PostgreSQL performing better than MySQL but still having processing delays.

4.5.3 Scalability

Another crucial element to consider when choosing databases is how well they scale with growing volumes of data or an increase in the number of concurrent users.

The research paper by (Chande, 2025) compares high-availability solutions for MSSQL and PostgreSQL, with one of its primary focuses being scalability. The paper examines vertical scaling, which involves adding more resources to a single machine, and horizontal scaling, which involves distributing the workload across multiple resources. By reviewing PostgreSQL, the paper finds that this RDBMS approaches scalability with streaming replication(read-heavy workloads can be handled via replica nodes), logical replication(allowing selective data replication and enabling cross- version compatibility), and supporting manual sharding or extensions such as Citus to distribute data across multiple nodes for horizontal scaling.

In contrast to the approach taken by PostgreSQL, (Chande, 2025) states that MSSQL leverages 'Always On Availability Groups' that distribute read requests efficiently, failover clustering to ensure seamless transition between nodes in case of failure, and has scale-up capabilities as MSSQL is optimised for enterprise use by employing intensive hardware to scale vertically. The paper also notes that MSSQL does well integrating into the Microsoft ecosystem, mainly when it is used with Azure SQL Database.

The paper concludes that, in terms of scalability, PostgreSQL is preferred for cloud-native, distributed systems, while MSSQL is preferred for enterprise workloads, offering highly performant and robust solutions.

4.5.4 Additional Features

There were additional features and tooling that were not explored within this literature review but were encountered from other resources. Table 6 provides a comparative overview of the features offered by MSSQL and PostgreSQL. While the literature review has focused on pricing, performance and scalability, other pertinent information was found from other resources such as (Sprinkle, 2024), (EDB, 2024), and (Geeksforgeeks, 2024).

The table highlights various features, development tools, and architectural differences between the two relational database management systems. The table will serve as a quick reference to evaluate and choose which is best suited by showcasing the strengths and limitations of each.

Table 6 Comparison of PostgreSQL and MS SQL features

Feature/Tool	PostgreSQL	Microsoft SQL Server
SQL Compliance	Strong compliance with SQL standards	Supports SQL but includes proprietary T-SQL extensions
Extensibility	Allows custom extensions, procedure language and data types	Limited, primarily T-SQL
Scalability & Concurrency	MVCC for high concurrency and scalability	Optimised for scalability, especially within enterprise environments
Advanced Data Types	Supports arrays, JSON, and geometric data types	XML, spatial data, and CLR integration
Security Features	SSL encryption, row-level security, and custom access controls	Transparent Data Encryption, Always Encrypted, and role-based access control
High Availability	Streaming, logical replication and failover mechanisms	Always On Availability Groups, mirroring, and clustering
Development Tools	pgAdmin, DBeaver, Navicat	SQL Server Management Studio, Azure Data Studio, Visual Studio integration
Integration & Ecosystem	Open-source third-party integrations	Strong integration with the Microsoft ecosystem
Community Support	Large open-source community and third-party extensions	Documentation and support from Microsoft
Licensing & Cost	Free, open source	Paid enterprise licenses or free developer & express versions

4.5.5 Summary

Based on the literature review, there appears to be a perception that MSSQL could be chosen over PostgreSQL primarily due to strengths in performance and enterprise scalability. The review suggests that MSSQL can offer faster data insertion and indexing, as well as potentially better query execution. MSSQL also has strong integration with other Microsoft products, such as Azure. Still, while Microsoft offers free tiers, the license pricing should be a factor that remains in consideration if this prototype is ever released to a production environment.

4.6 Blob Storage

Blob storage is a storage service that is fundamental to cloud computing and is designed to store unstructured data that does not conform to a particular file format, for example, images, videos, documents or other binary files (Cloudflare, 2025). Blob storage is designed to handle the storage of very large amounts of data.

Cloud providers such as Azure, AWS, and GCP all offer blob storage services, which provide nearly limitless capacity, allowing businesses to manage large amounts of data on their physical hardware, including servers. With the popularity of blob storage, providers are in heavy competition to offer cheaper pricing and to compete on performance (Noah, 2024).

The authors (Daher & Hajjdiab, 2018), offer a guide for developers to decide on the type and provider to select by comparing two of the most widely used cloud services, Amazon S3 and Azure. Table 7 has been created to collate a summary of the advantages and disadvantages that were the key findings of (Daher & Hajjdiab, 2018).

Table 7 Comparison table between Amazon S3 and Azure Blob Storage

Feature	Amazon S3	Azure Blob Storage
Granularity	Object-level tier management for flexibility.	Requires a uniform access tier within an account.
Availability	All AWS Regions.	Limited availability in some regions.
Scalability	100 buckets per account, 5TB per object.	100 x 500 TB accounts, 500 TB per blob container.
Pricing	Charges apply for changing tier before 30 days.	Free tier changes, but access tier uniformity is enforced.
Security	Server-side encryption, TLS for data transfer.	Lacks server-side encryption.
Changing Access Tier	Object lifecycle policies automate transitions.	Account-wide tier changes, restrictive flexibility.
Performance	No targets published.	High request rates and bandwidth limits are documented.

The authors conclude that both providers offer cost-effective options for blob storage, with AWS offering more flexibility, and both provide robust security, although Azure offers less regional availability. With this said (Daher & Hajjdiab, 2018) and (Noah, 2024) Both point to user requirements and the cloud ecosystem being the key deciding factors when choosing between the two services, with no superior provider.

5. Methodology

5.1 Research Approach

For this project, it has been decided to adopt an applied research approach to developing a prototype replicating Microsoft App Center's application distribution functionality. The research methodology has employed a hybrid approach, utilising both qualitative and quantitative techniques to ensure a well-rounded assessment of whether the system is feasible and efficient.

5.1.1 Qualitative Approach

Technology selection, software architecture design, and development methodologies with a qualitative approach. This will include:

- Justification for choosing the various technology stacks based on usability, integration capabilities, and efficiency.
- Based on an analysis of various SDLC models within the literature review.
- A comparative evaluation of cloud solutions, focusing on the Azure, AWS, and GCP platforms based on scalability, reliability, and ease of deployment.
- Documentation of ease of development and best practices being observed throughout the implementation.

5.1.2 Quantitative Approach

The performance, efficiency, and scalability of the prototype will be evaluated using quantitative techniques. Quantitative aspects may include:

- Scalability Testing – Using Azure tools to simulate concurrent users and review system responsiveness.
- Monitor and review performance.

5.2 Software Development Methodology

While well-structured, the waterfall model is less adaptable to revisions, changes, and responding to problems that arise during development. Iterative and incremental methodologies, while improved, also have drawbacks that would not suit this project (Gurung, et al., 2020).

This project will follow an agile methodology, consisting of the requirements analysis, design, implementation, and quality assurance phases (GeeksForGeeks, 2025). It will require a high level of flexibility, as there is potential for numerous changes to the requirements, allowing the development to evolve, which will need to be addressed. For this reason, the agile model was chosen as the better approach (Ikbal Hossain, 2023).

With Agile chosen, the project will implement methodologies such as Extreme Programming to develop the prototype efficiently. Once the prototype is built, requirement adjustments will be introduced in future iterations to refine and enhance the functionality.

5.3 Technology Stack Selection

This section will discuss the selection criteria for the technology stack to be employed in application development. These choices will include web frameworks, cloud technologies, databases, and blob storage. The section will then identify relevant tools that can be employed to test and evaluate the application.

5.3.1 Web Framework

Following the preceding literature review, which performed a comparative analysis of the distinct approaches of ASP.NET Core (Blazor and MVC) and React, the selection of the web framework is an important aspect and a key factor in the success of this development.

Based on the literature review, the decision to develop the application with Blazor can be justified with several key findings.

Firstly, ASP.NET Framework is an older platform that lacks cross-platform support and is no longer actively promoted by Microsoft (Nile Bits, 2023).

While comparing Blazor to React, the review highlighted that both are component-based, but Blazor is potentially easier to maintain and is written in C# for both front-end and back-end (Fishuck & Datsyshyn, 2025). React does have a larger community and benefits from a more extensive library, but Blazor was found to have sufficient support and enough libraries to make it a viable option (Köping & Persson, 2021).

Given the requirements to build an interactive user interface and the developer's C# coding background, along with minimal experience with JavaScript, Blazor has been selected as the web framework to quickly and efficiently build the required application.

5.3.2 Cloud Technologies

After conducting the literature review on the types of cloud platforms and providers, it has been decided that a PaaS hosting model will be employed for this project, with the decision ultimately stemming from the advantages of PaaS reducing the burden to manage the infrastructure by eliminating the need to handle updates to the runtime, operating system, security patches and also having automatic scaling that does not require configuring by the developer (Borra, 2024).

The decision of which provider to employ came down to the seamless integration of the web framework (Blazor), which will speed up development time with ease of use (Fishuck & Datsyshyn, 2025).

5.3.3 Database Selection

The literature review synthesised research and technical documentation to assess database management systems, focusing on pricing, performance, and scalability. It employed a comparative analytical approach to examine PostgreSQL and MSSQL while also summarising system features, tooling, extensibility, and support.

PostgreSQL and Microsoft SQL Server (MSSQL) were chosen due to their prominence in enterprise systems. A systematic method identified credible resources, prioritising benchmarking and peer-reviewed case studies. Where unavailable, vendor documentation and authoritative sources were used. The key difference between these databases came down to pricing (Google Cloud, 2025). Still, with the available free models for the prototype (Microsoft, 2025), existing licences within the council, and given the selected web frameworks and cloud technologies, MSSQL was deemed optimal due to its competitive performance, scalability, and query execution. Its strong enterprise adoption and Azure SQL integration could further benefit production deployment.

5.3.4 Blob Storage Considerations

One of the fundamental requirements of this application is the storage and retrieval of Android mobile application build files(.apk files). Based on the outcomes highlighted in the literature review, it has been determined that implementing cloud-based blob storage would be beneficial.

The literature review highlighted that blob storage is the most cost-effective storage option for storing large files, as them being highly scalable to ensure that the application can accommodate more files over time (Daher & Hajjdiab, 2018). Availability was another factor discussed with AWS and Azure, both of which have high availability for these key artefacts that are so pertinent to the system.

By comparing two providers, the literature review concluded that both platforms offered cost-effective, secure, and reliable services. Still, ultimately, it was user requirements and the cloud ecosystem that would be the deciding factors (Noah, 2024). With Azure already being selected as the hosting platform, it has been decided to keep the blob storage with the same provider.

5.4 Testing and Evaluation Metrics

To validate the platform's functionality, performance and reliability, a structured testing strategy will be implemented. This methodology will seek to ensure that, based on evidence, the system is behaving and performing as expected.

Unit testing is a technique of writing code that is designed from test cases to automatically ensure the correctness of individual components or methods of the system (GeeksForGeeks, 2025). Automating these tests can provide results that confirm the stability and correctness of features and can be run at any given time to facilitate technical analysis. Unit tests can also be run via CI/CD pipelines to ensure quality control when merging code.

Manual user testing will also be conducted to verify that all actions within the distribution workflow process are functioning correctly. This will focus on core functionality, including application management, release management, authentication, file upload to blob storage, and download of versions to the tester's mobile device.

Another measure will be to perform a load test within the Azure Load Test.[1] To simulate how the system performs under heavy concurrent user interactions. Finally, App Insights[2] will be employed to evaluate availability, failures and performance.

[1] https://learn.microsoft.com/en-us/azure/load-testing/overview-what-is-azure-load-testing
[2] https://learn.microsoft.com/en-us/azure/azure-monitor/app/asp-net-core

6. Workplan

To maintain project progress across research, design, and implementation, several strategies have been used. Regular supervisor meetings from October 2024 to June 2025 ensured updates on developments. Additionally, a Gantt chart was created to track project phases, tasks, and timelines, encompassing the report, requirement analysis, design, and implementation phases.

6.1 Introduction

The project commenced in early March 2025, with the primary objective of identifying a problem and defining its scope. This ensured a clear direction for addressing specific challenges. Additionally, the work plan was developed based on previously established objectives and aims. Figure 12 presents the Introduction phase within the GANNT chart.

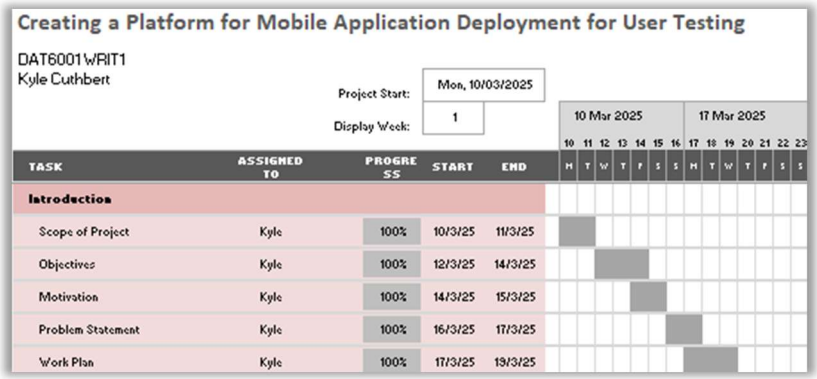

Figure 12 Introduction Workplan

6.2 Literature Review and Methodology

The literature review begins on March 19th, focusing on research and methodologies relevant to the application's features, functionality, and best practices for project planning and implementation. Insights gained will inform the design phase prior to the requirements analysis. Figure 13 presents the Literature Review phase of the work plan.

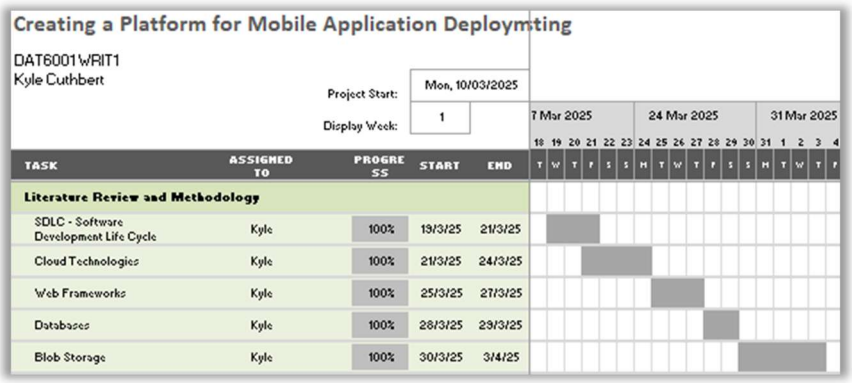

Figure 13 Literature Review Workplan

6.3 Requirement Analysis and Design

The “Requirement Analysis and Design” phase runs from April 4th to 23rd, focusing on gathering and documenting functional and non-functional requirements. Functional requirements define system actions and behaviours, while non-functional requirements ensure performance, scalability, security, and usability.

Once analysed, the design stage translates these requirements into UML artefacts, serving as blueprints for the system:

- Use Case Diagram – To show the key functionalities from a user perspective.
- Activity Diagram – To map out the system’s workflow. Showing an understanding of the processes documented in the use case diagram.
- Sequence Diagram – Highlighting the interactions between the different objects to show how operations are carried out in order of time (Visual Paradigm, 2025).
- Class Diagram – To visually communicate the structure of the system by showing the relationships of the classes (GeeksForGeeks, 2025).
- Entity-Relationship Diagram – This diagram defines the data relationships that determine the database structure.

Additionally, an initial GUI design will visualise the application's key screens. Figure 14 presents this phase of the work plan.

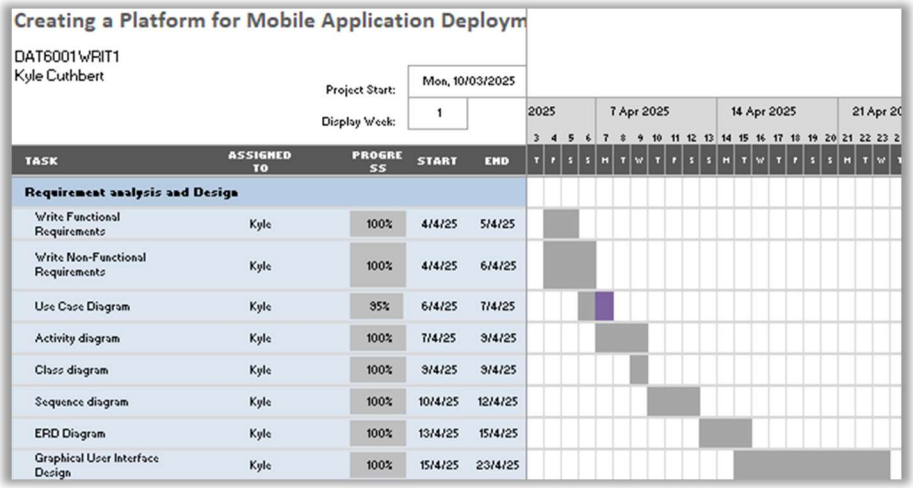

Figure 14 Requirement Analysis and Design Workplan

6.4 Implementation

The implementation phase, which runs from late April to early June, focuses on application development. Targeted research on key technologies, including .NET Blazor for frontend, MudBlazor for UI, and Azure for cloud services, begins earlier to ensure proficiency.

A dedicated code repository will be established for iterative development. The initial stages include authentication implementation, database creation, and integration of CRUD functionality. File upload and download features will follow, utilising Blob storage.

A basic mobile application will be developed for testing purposes, along with the creation of unit tests to ensure its functionality. Deployment will involve setting up cloud infrastructure, establishing CI/CD pipelines for automated releases, and utilising monitoring tools to track performance. Figure 15 illustrates this phase.

.

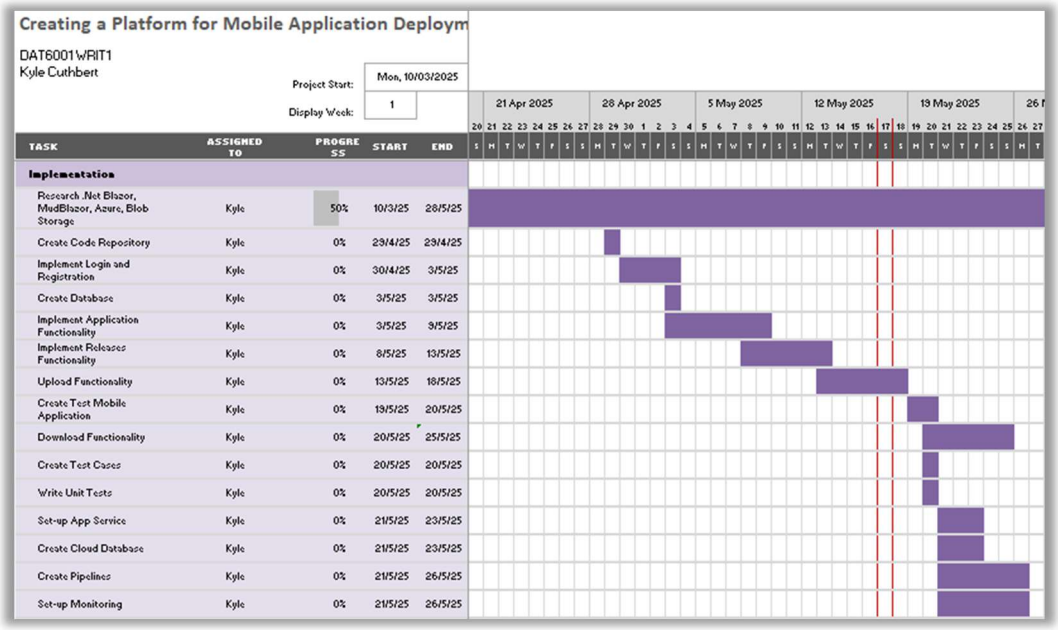

Figure 15 Implementation Workplan

7. Requirement Analysis and Design

The first phase within the Agile software development lifecycle is the requirement gathering and analysis. This section outlines both the functional and non-functional requirements of the DevAppDeploy system.

Following this phase will be the design of the application based on the requirements that have been analysed.

7.1 Functional Requirements

The functional requirements of the DevAppDeploy system will focus on the core functions that have been identified, considering the scope of the development.

7.1.1 User Roles and Permissions

Authentication and Authorisation

- The DevAppDeploy application will employ ASP.NET Core Identity for user authentication.
- The system will have two roles, Developer and Tester, each offering different access levels.

Role-based Access Control

All users (both Developer and Tester) can:

- View a table listing all the applications.
- View a table listing each release of a chosen application.
- View release details for each release.
- Download release APK files.

Only users with the Developer role can:

- Add new applications.
- Delete existing applications.
- Add new releases(upload APK to Azure Blob Storage and insert details into Azure SQL Database).
- Delete existing releases.

7.1.2 Application Management

Application Viewing

- Display a table listing all the applications available in the system.

Application Creation and Deletion

- Developers can create and delete applications from the system.

7.1.3 Release Management

Release Viewing

- Users can view all releases available per application.

APK Download

- Users can download APK files associated with a release.

Release, Create, and Deletion

- Developers can upload new release APK files to be stored in Azure Blob Storage.
- Developers can delete previously uploaded releases.

7.1.4 Data Storage

Database Integration

- Application data (users, roles, apps, releases) is stored in an Azure SQL Database via Entity Framework Core by using incremental data migrations.

Blob Storage

- APK files will be loaded to and stored in Azure Blob Storage, with the Blob Storage URLs securely stored and managed within the database.

7.2 Non-functional Requirements

The non-functional requirements identified for this development, which will be discussed in this section, include performance, scalability, security, availability, maintainability, and usability.

7.2.1 Performance

The system's performance will be monitored and evaluated with unit tests, load testing and end-to-end testing, and the performance requirements for the application are:

- Response Time – The application must have an average process response of 500ms under normal load.
- Concurrent Users – Conducted load tests with over 30,000 users to confirm stability.
- High Availability – The application's availability must be over 90%.
- Failure Rate – The application must have a low failure rate of
- Test Cases – All test cases must pass.

7.2.2 Scalability

With this application hosted on Azure App Service, it will benefit from horizontal scaling, which will handle increased user load. This prototype will utilise the free student tier, which does not include scaling. However, if the application is promoted to production, a more viable tier will be available, offering auto-scaling.

7.2.3 Security

The application will have the following security requirements:

- Utilises ASP.NET Core Identity with secure password hashing and role-based access.
- Utilises Entity Framework Core, which protects against SQL injection.
- Azure Blob Storage access is restricted and secured via Shared Access Signature(SAS token) or private containers with controlled access.
- HTTPS is enforced automatically with Azure for azurewebsites.net domains, to secure communication between client and server.

7.2.4 Availability

The application will be hosted on Azure App Service to achieve high availability, and by using Application Insights and Azure Monitor, availability analytics can be monitored.

Azure SQL Database will provide automatic backups and geo-redundancy options; however, these options will only be available with a paid tier of Azure licensing, which will not be utilised for this prototype.

7.2.5 Maintainability

The Application will be written with a separation of concerns, with user interface logic being stored within the code-behind files using the .NET Blazor architecture. The application will separate the business logic by using dependency injection and services.

Coding tools within the IDE will be utilised to ensure that uniform naming conventions and coding styles are consistently enforced.

EF Core migrations will provide an easy way to evolve the database schema.

7.2.6 Usability

DevAppDeploy should be a simple and intuitive web user interface. This will be achieved by employing Blazor Server and MudBlazor to provide real-time interaction.

Straightforward navigation, supported with a navigation bar and role-based UI elements to ensure that users can only use functionality and actions that they are authorised to use.

7.3 Use Case Diagram

Figure 16 shows a use case diagram that represents the release distribution use cases of the DevAppDeploy system. There are two primary actors in the use case diagram: the developer and the tester. The tester will be able to register and log in to view applications, as well as click on and view the corresponding releases, allowing them to download a release file.

In addition to having all the use cases of the tester, the developer also has access to add an application, create a release, and upload an app file to that release. The developer will also be able to delete a release.

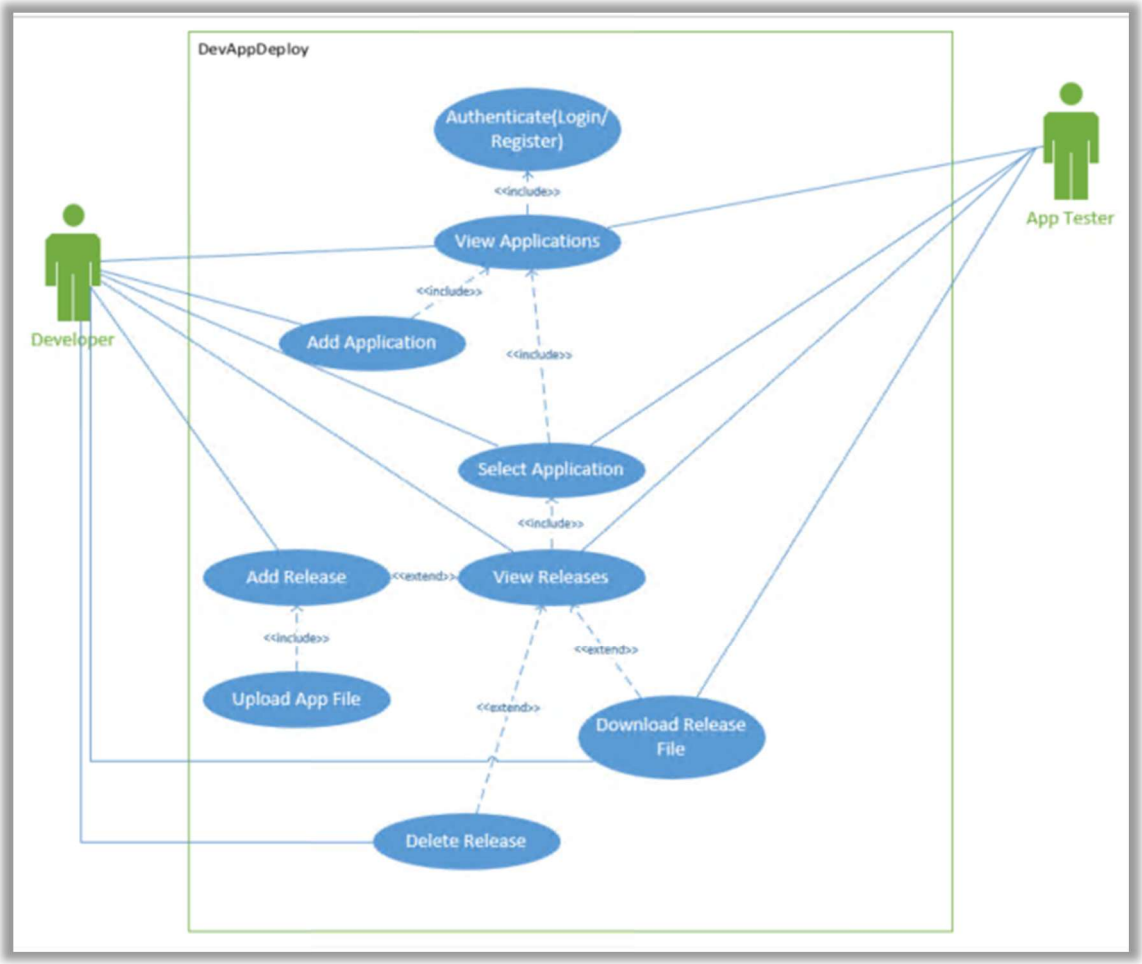

Figure 16 DevAppDeploy use case diagram

7.4 Activity Diagrams

7.4.1 Registration and login

Figure 17 illustrates the activity diagram for the login and registration process from the DevAppDeploy system that leverages .NET Identity for secure authentication. It shows the steps taken, including user input validation and account verification.

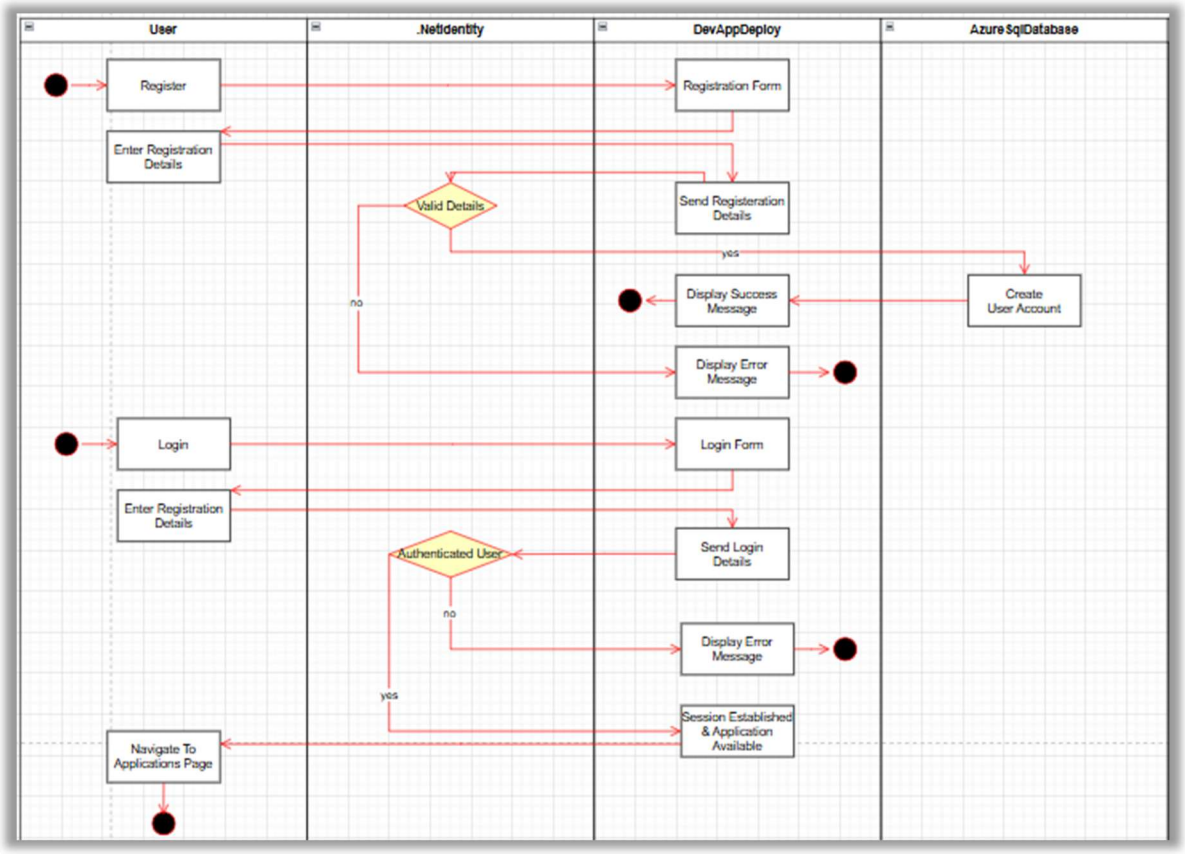

Figure 17 Activity showing the flow of registration and login

7.4.2 Application and Release Activities

Figure 18 illustrates the activity diagram for DevAppDeploy core functionality surrounding the release management for each application, showcasing the steps taken that lead up to uploading and downloading Android APK files to and from Azure Blob Storage.

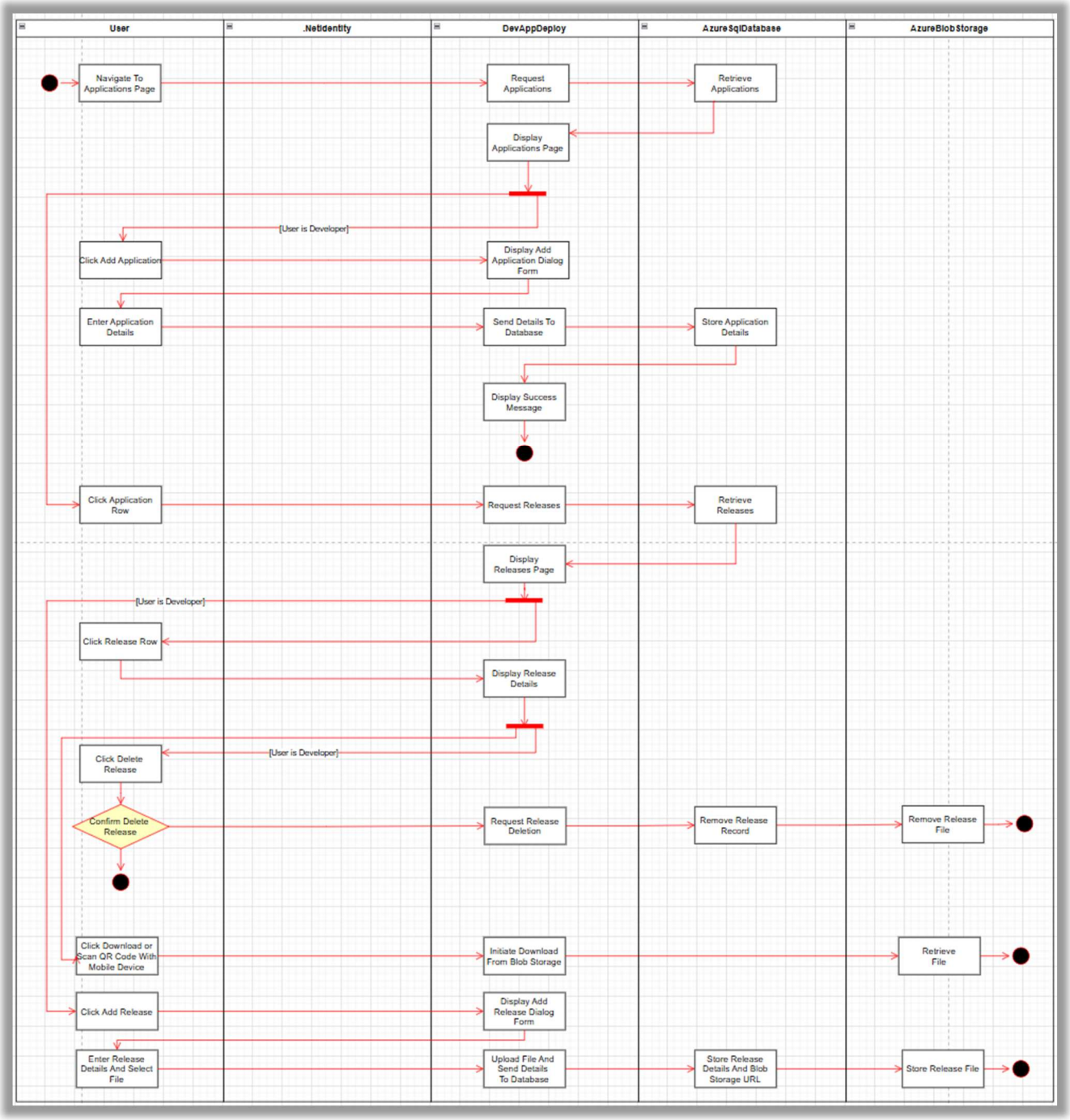

Figure 18 Application and release activity flow diagram

7.5 Class Diagram

Figure 19 illustrates the class diagram, which shows the core structure of the application and release management within the system. It showcases the creation and display of models for data transfer, as well as ApplicationTbl and ReleaseTbl, which represent the database entities. The ApplicationService manages the business logic, performing operations such as creating applications and releases, and fetching data by interacting with database entities. ApplicationTbl shows a 'has-a' relationship connecting it with ReleaseTbl, indicating that an application can have multiple releases. The BlobStorageService is also shown.

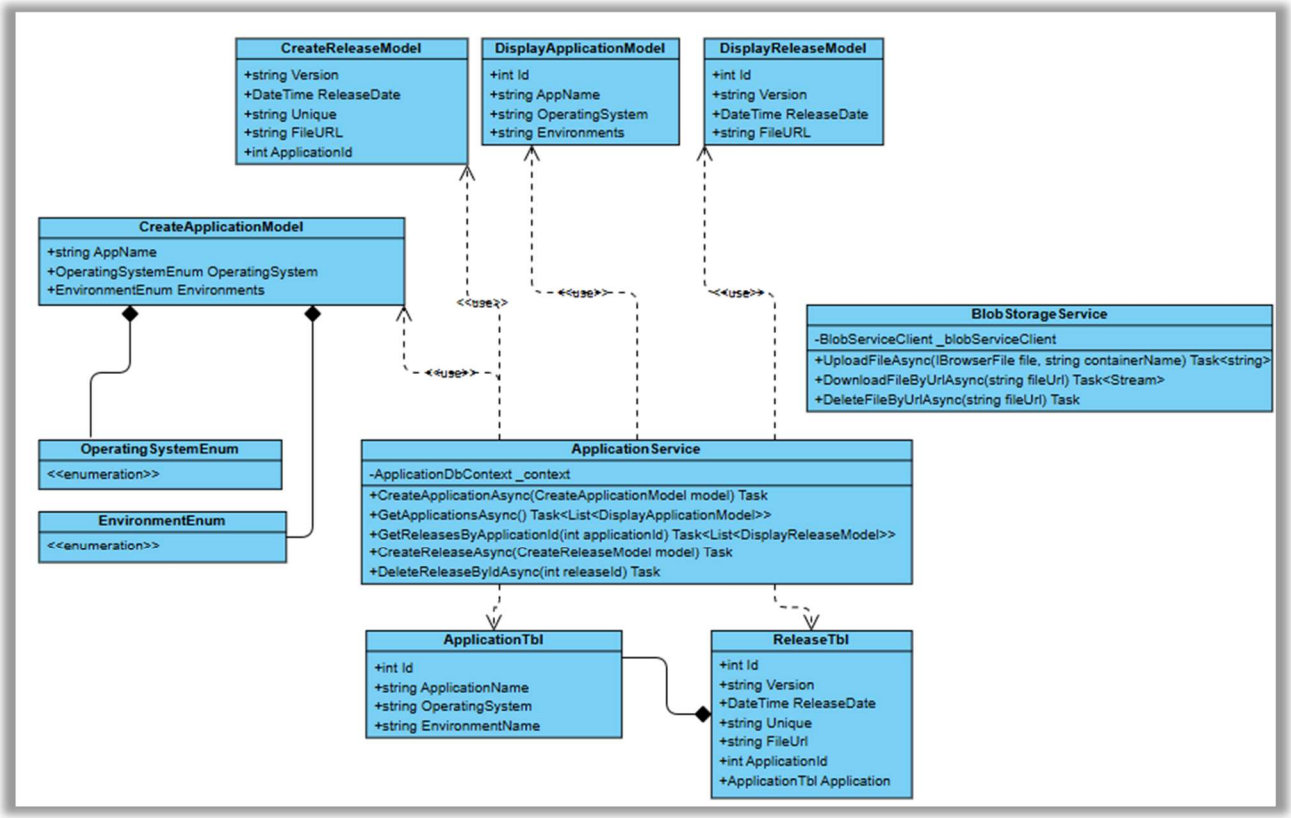

Figure 19 Class diagram showing the core services

7.6 Sequence Diagrams

Through a series of sequence diagrams, this next stage of the design phase of the SDLC will showcase key interaction flows of the DevAppDeploy system. These sequence diagrams will illustrate the sequence of actions, showing how the key interactions of the system are envisioned to flow.

7.6.1 Registration Sequence Diagram

The first process is the registration of a user. The sequence diagram in Figure 20 shows how the client will interact with the registration page, which will interact with the .NET identity services and the database.

As can be seen, the user will complete the pertinent form details before clicking the registration button, which will call the authentication service to attempt to insert the new user into the database. The service will then send the result of the action back to the user and display the status.

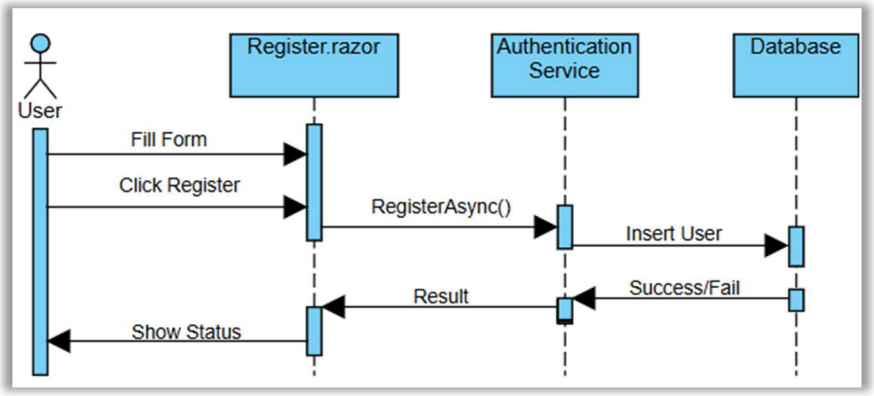

Figure 20 Registration sequence

7.6.2 Login Sequence Diagram

The login process attempts to authenticate the user by using their user details(email and password), and Figure 21 shows the user completing the login form to attempt validation via .NET Identity's authentication service.

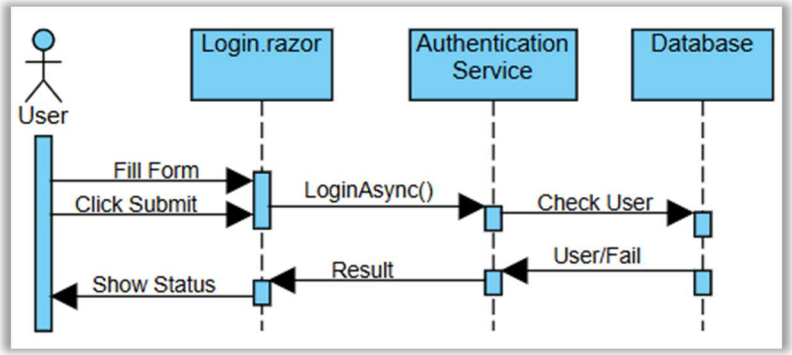

Figure 21 Login sequence

7.6.3 Get Applications Sequence Diagram

The application page can be called via the navigation bar by the user clicking 'applications', once this is clicked, the UI code will use the get application method within the application service to query the database and return a list of the applications so that they can be rendered to a table within the applications page. This can be seen in Figure 22.

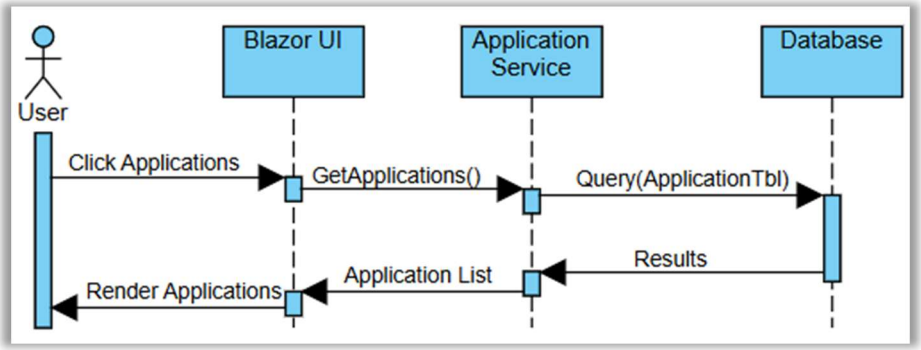

Figure 22 Get applications sequence

7.6.4 Add Application Sequence Diagram

To create a new application, the user will fill out the 'AddApplicationDialog' with the new application's details and click submit. The 'CreateApplicationAsync' method within the application service will then use the Entity Framework Core database context to add and save the changes, inserting the application into the database. If this has been successfully added to the database, the new application will also appear on the applications page. This is illustrated in Figure 23.

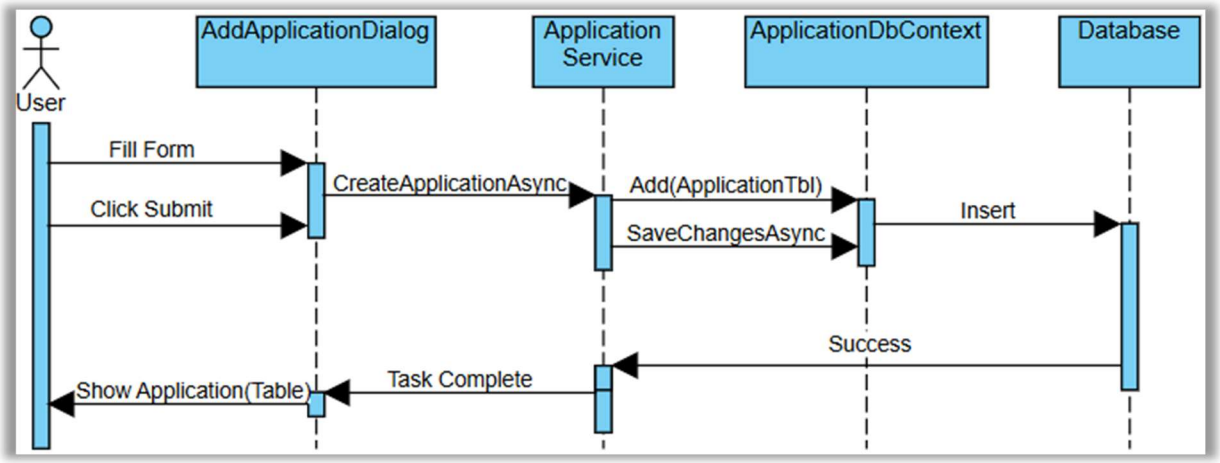

Figure 23 Add application sequence

7.6.5 Add Release Sequence Diagram

The sequence diagram shown in Figure 24 shows the process of adding a new release. Once the user has entered the chosen application, they will be able to click 'Add Release' to open a dialogue form that will collect the pertinent information and an .apk file. The information and the file will then be submitted with the application service, which will call the blob service to upload the file to Azure Blob storage before returning the blob URL to the application service. The release, including the URL, will then be stored in the database before returning a success message, closing the dialogue, and refreshing the table that lists the releases for the given application.

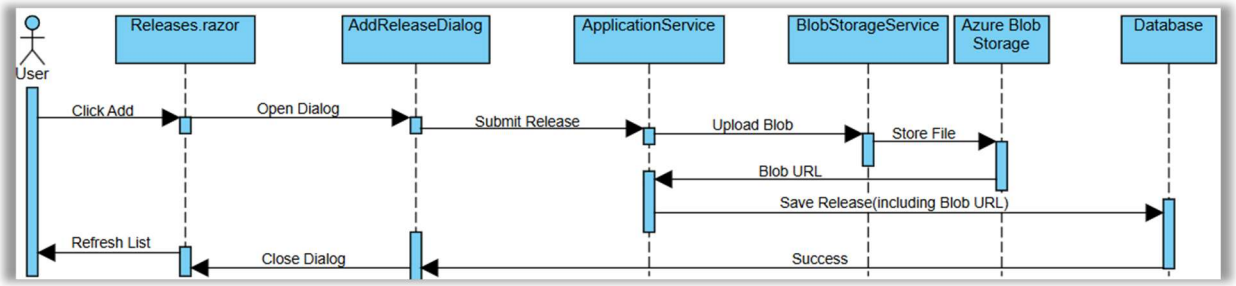

Figure 24 Add a release sequence diagram

7.7 Entity-Relational Diagram

An entity-relational diagram has been created to illustrate the expected relationships between entities, their data types, and fields. Included in this ERD is the EF Core migrations table, which stores an audit of changes to the database and .NET Identity standard tables, which may or may not be extended with change requirements. The fundamental tables for the mobile application releases are the 'Releases' table and the 'Applications' table, with applications having a one-to-many relationship (one application can have many releases, but a release can only have one application). This is managed with the ApplicationId(the primary key in the Application table) acting as the foreign key in the Releases table. With future understanding, the database tables can be optimised to ensure a more relevant data type is utilised for each field by not using unrestricted nvarchar(-1). The ERD is illustrated in Figure 25.

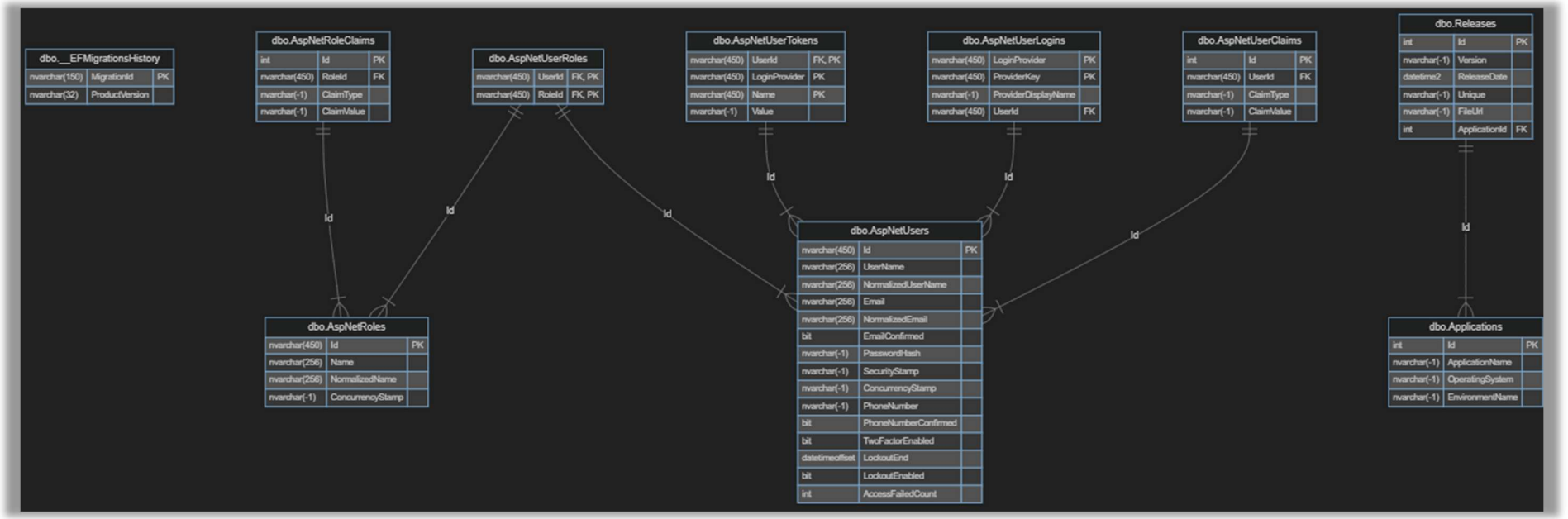

Figure 25 Entity-relational diagram

7.8 Graphical User Interface Design

This section showcases the graphical user interface (GUI) design of the system, highlighting key pages and interactive components. The UI has been developed using .NET Blazor, along with MudBlazor, for styling to ensure a modern and user-friendly experience.

Home Page

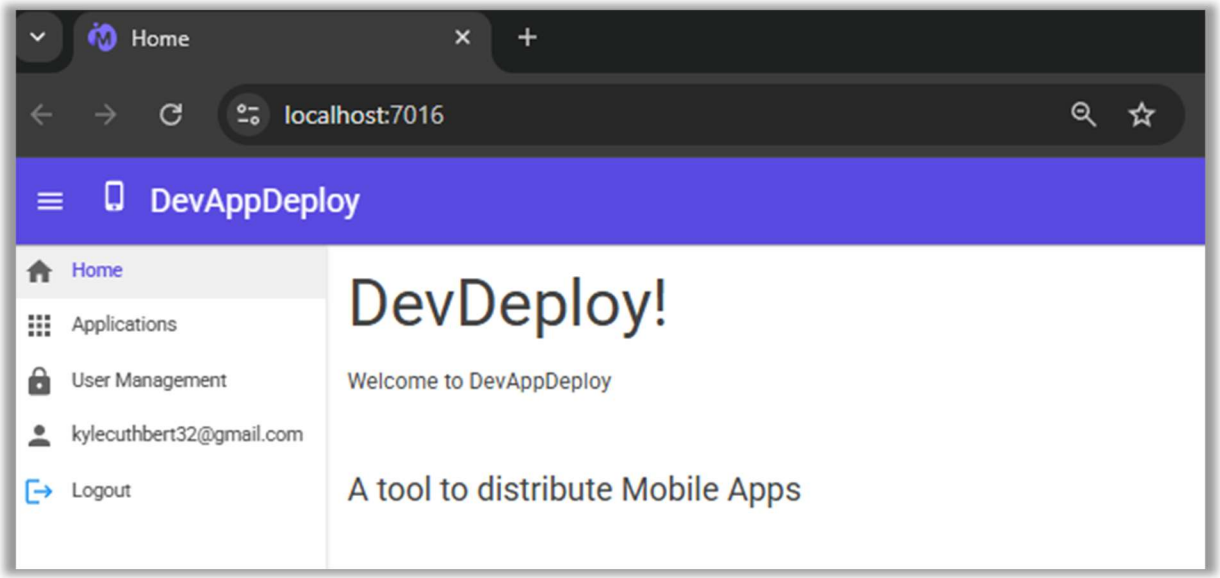

Figure 26 Home page

Application Page

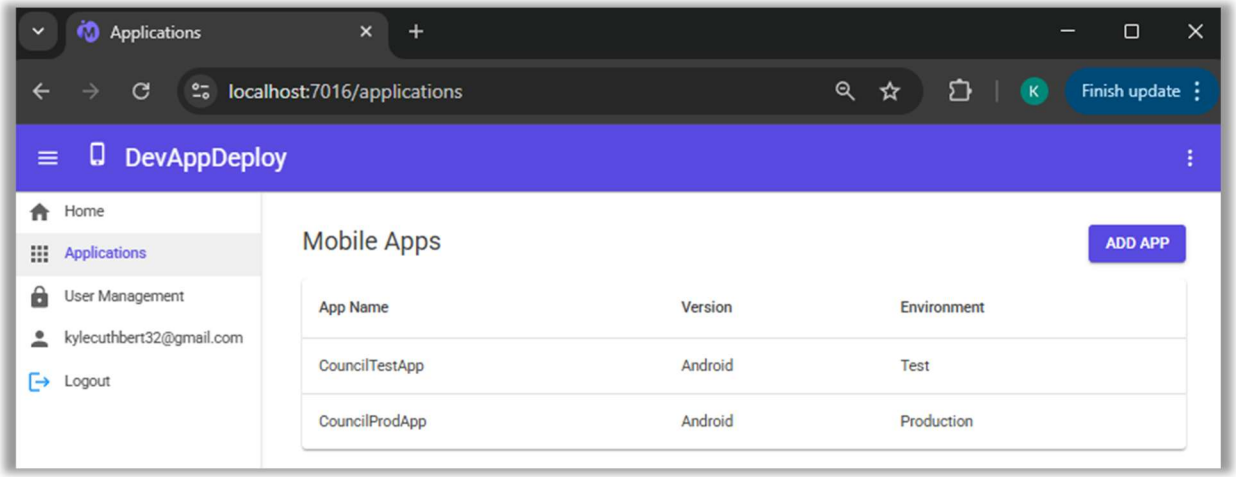

Figure 27 Applications page

Add Application Dialog

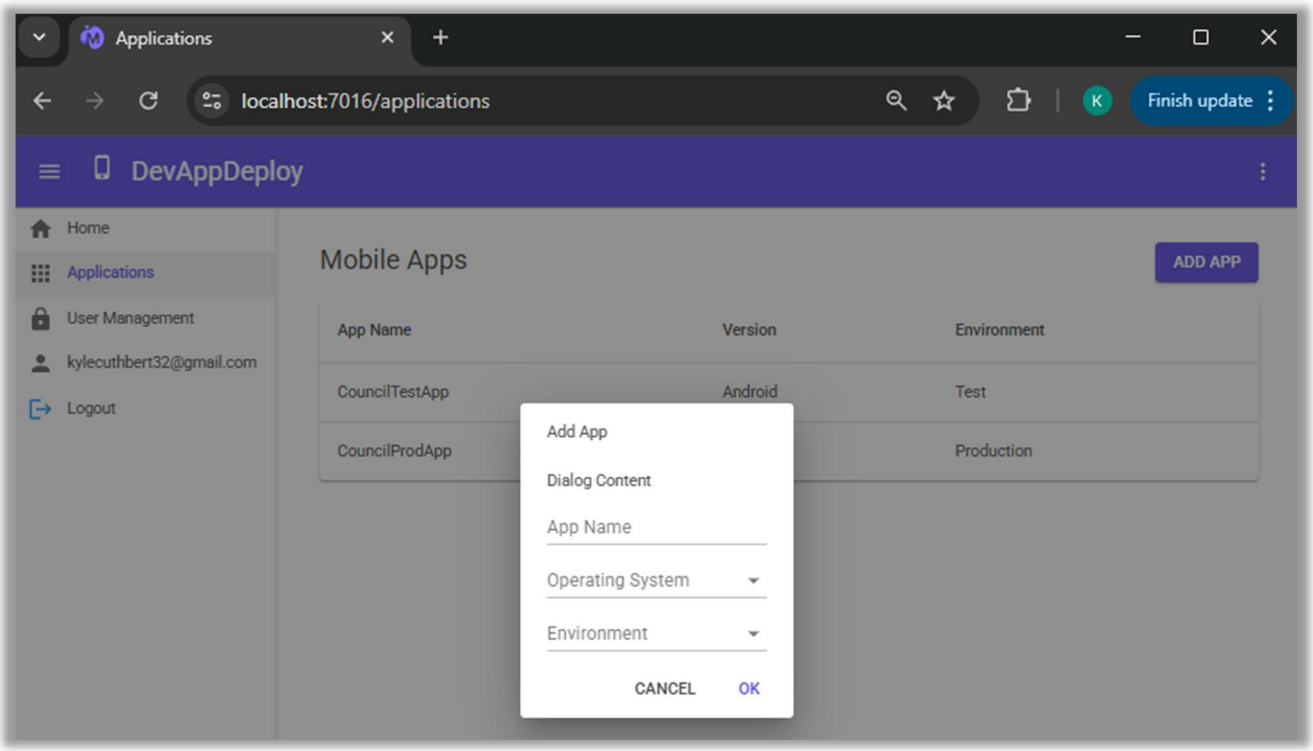

Figure 28 Add app dialogue

Releases Page

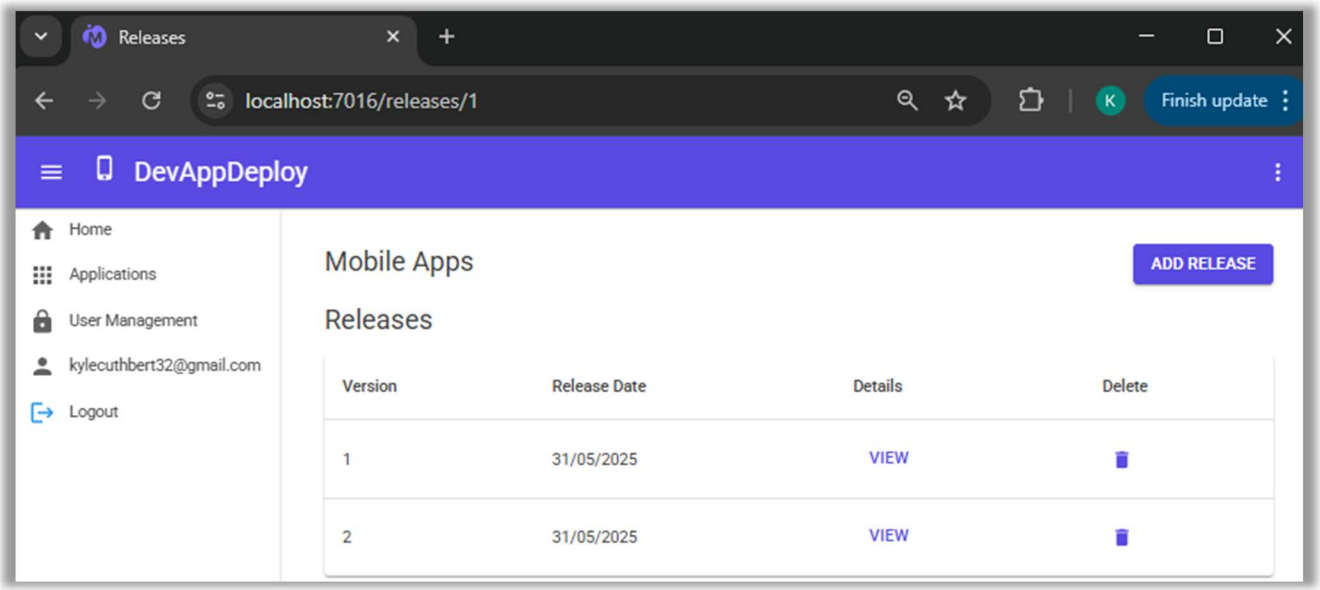

Figure 29 Releases page

Add Release Dialog

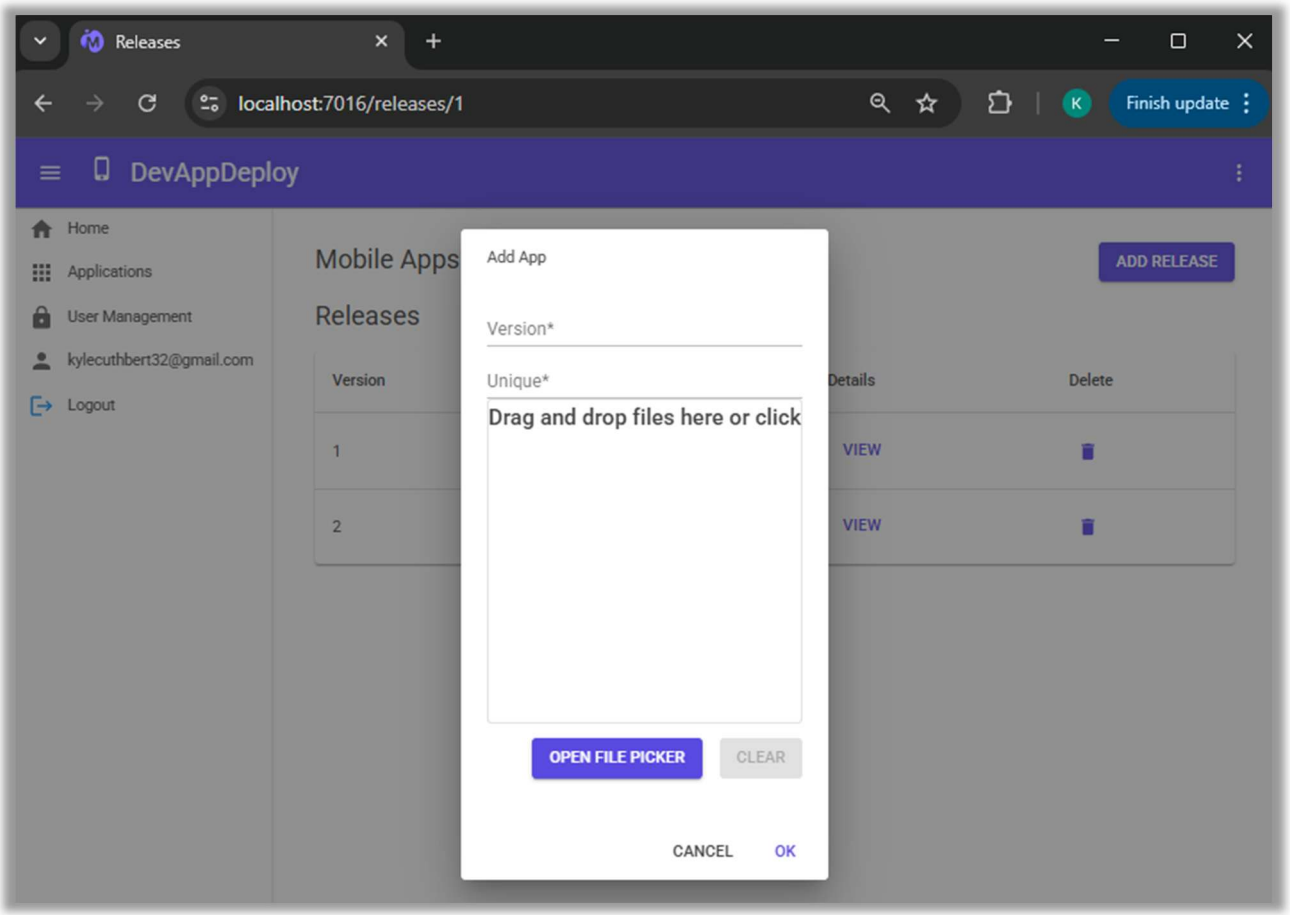

Figure 30 Add release dialogue

Release Details

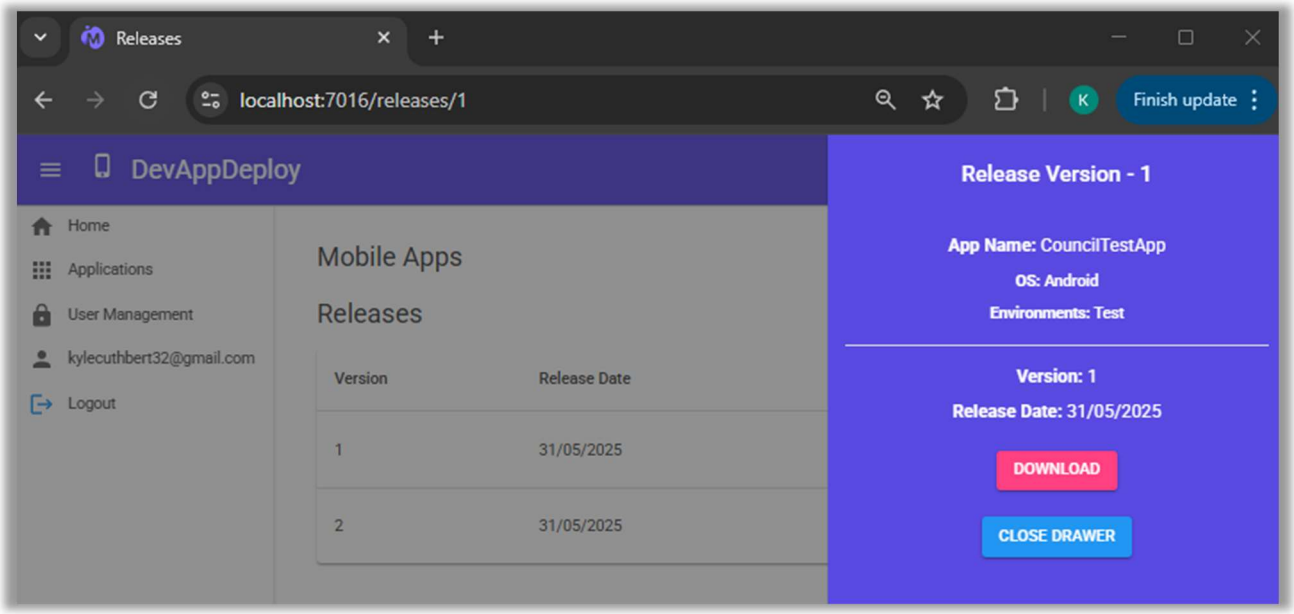

Figure 31 Release details page

8. Implementation and Quality Assurance

This section will discuss the steps taken to implement the DevAppDeploy prototype from start to finish. It will showcase the steps taken from setting up version control, configuring the blob storage container, and implementing the various code to add the CRUD functionality for both applications and the releases. Other steps include adding role-based access for restricted parts of the application, setting up the database and the app service on Azure, setting up CI/CD pipelines, testing and monitoring the application running within the cloud environment.

8.1 Version Control

For this project, it was decided to store the code repository using Azure Repos within Azure DevOps.

Initially, an organisation was created with the developer's Azure DevOps student account.

8.1.1 Project and Repository Set-up

Once the organisation was created, a new project called DevAppDeploy was created, and two separate branches were created called dev and master. The repository was set up this way to ensure that the master branch was protected from any rogue or invalid code making its way into the branch that is published to the production environment by allowing development work to take place on the dev branch before being reviewed and merged into the master branch. Figure 32 shows the repository and the two branches that were created.

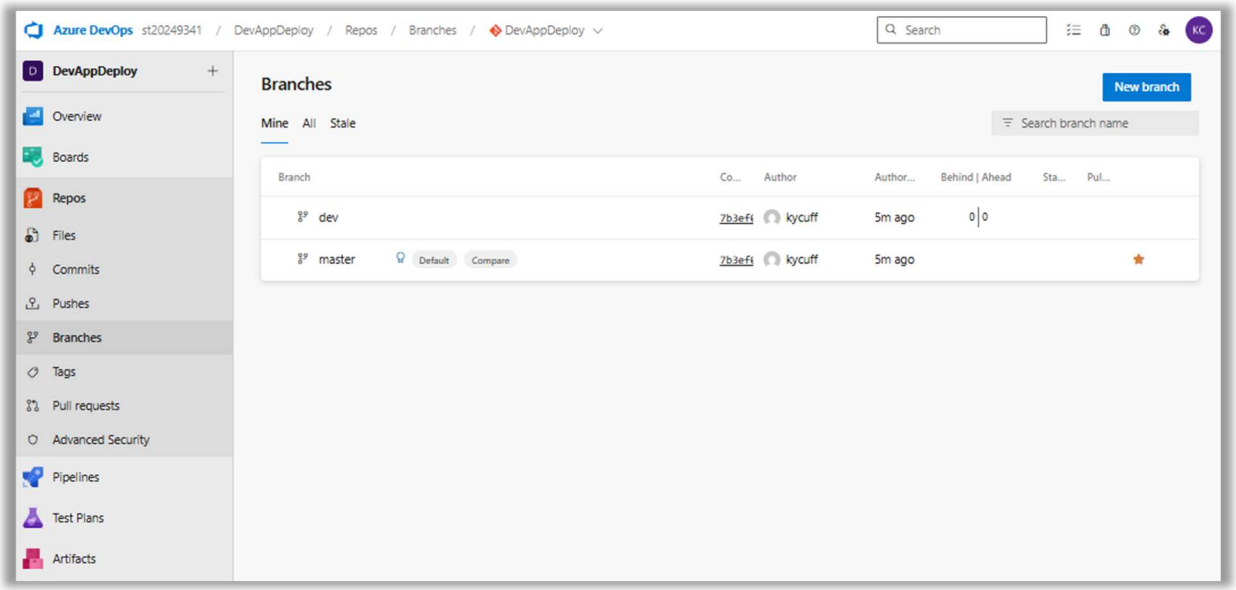

Figure 32 Project repository setup in Azure Repos

8.1.2 Branch Policies

As mentioned in the previous section, the dev branch acts as the development environment. This section shows the branch policies that were configured on the branches to ensure the branches behave as expected.

Development Branch

On the dev branch, 'Force Push' was denied, ensuring that it is not deleted once the dev branch is merged into the master branch. This is shown in Figure 33.

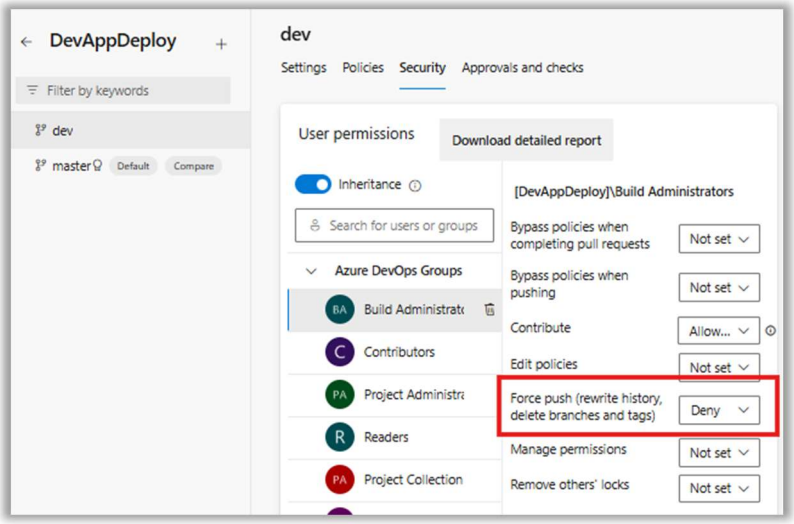

Figure 33 Dev branch policies

Master Branch

To protect the master branch, a policy was implemented, as illustrated in Figure 34, to ensure that pull requests are reviewed and approved before new code is merged into the branch. As this is a solo project, a minimum of one reviewer was configured.

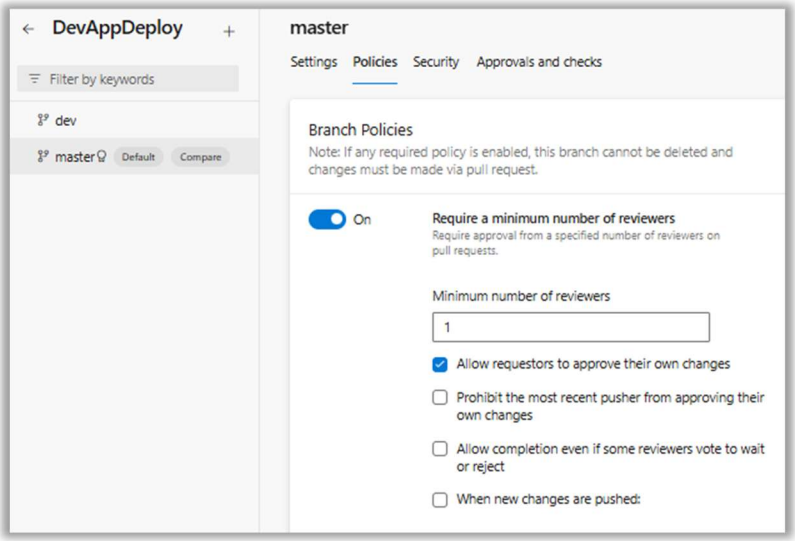

Figure 34 Master branch policies

8.2 Blob Storage

This section of the report details how the storage of the mobile application files was implemented within the DevAppDeploy application using Azure Blob Storage, and how the application integrates with Azure.

A service within the application was created to integrate with Azure Blob Storage, which can be injected as a dependency into any other services or frontend code behind files. The service called **'BlobStorageService'** was created, using an interface called 'IBlobStorageService' as the contract for dependency injection. Figure 35 shows the interface file.

```
namespace DevAppDeploy.Services.Blob;

4 references
public interface IBlobStorageService
{
    2 references
    Task<string> UploadFileAsync(IBrowserFile file, string containerName);
    2 references
    Task<Stream> DownloadFileByUrlAsync(string fileUrl);
    2 references
    Task DeleteFileByUrlAsync(string fileUrl);

}
```

Figure 35 IBlobStorageService interface class

Figure 36 shows the IBlobStorageService interface being configured for dependency injection within the program.cs file.

```
builder.Services.AddSingleton(x =>
    new BlobServiceClient(builder.Configuration.GetConnectionString("AzureBlobStorage")));
builder.Services.AddScoped<IdentityUserAccessor>();
builder.Services.AddScoped<IdentityRedirectManager>();
builder.Services.AddScoped<AuthenticationStateProvider, IdentityRevalidatingAuthenticationStateProvider>();
builder.Services.AddScoped<IBlobStorageService, BlobStorageService>();
```

Figure 36 Dependency injection of BlobStorageService

The BlobStorageClient will be initialised within the various service methods using a connection string from Azure to connect with the Azure Blob Storage account. The appsettings.json file is shown in Figure 37.

```
"ConnectionStrings": {
  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=DevAppDeploy;Trusted_Connection=True;MultipleActiveResultSets=true",
  "AzureBlobStorage": "DefaultEndpointsProtocol=https;AccountName=devappdeploystorage01;AccountKey=[redacted]==;EndpointSuffix=core.windows.net"
},
```

Figure 37 Azure Blob Storage connection string

8.2.1 Azure Configuration

Firstly, within Azure Portal under 'Storage Accounts', an account was created called devappdeploystorage01. Figure 38 shows the account being created using the developer's student subscription.

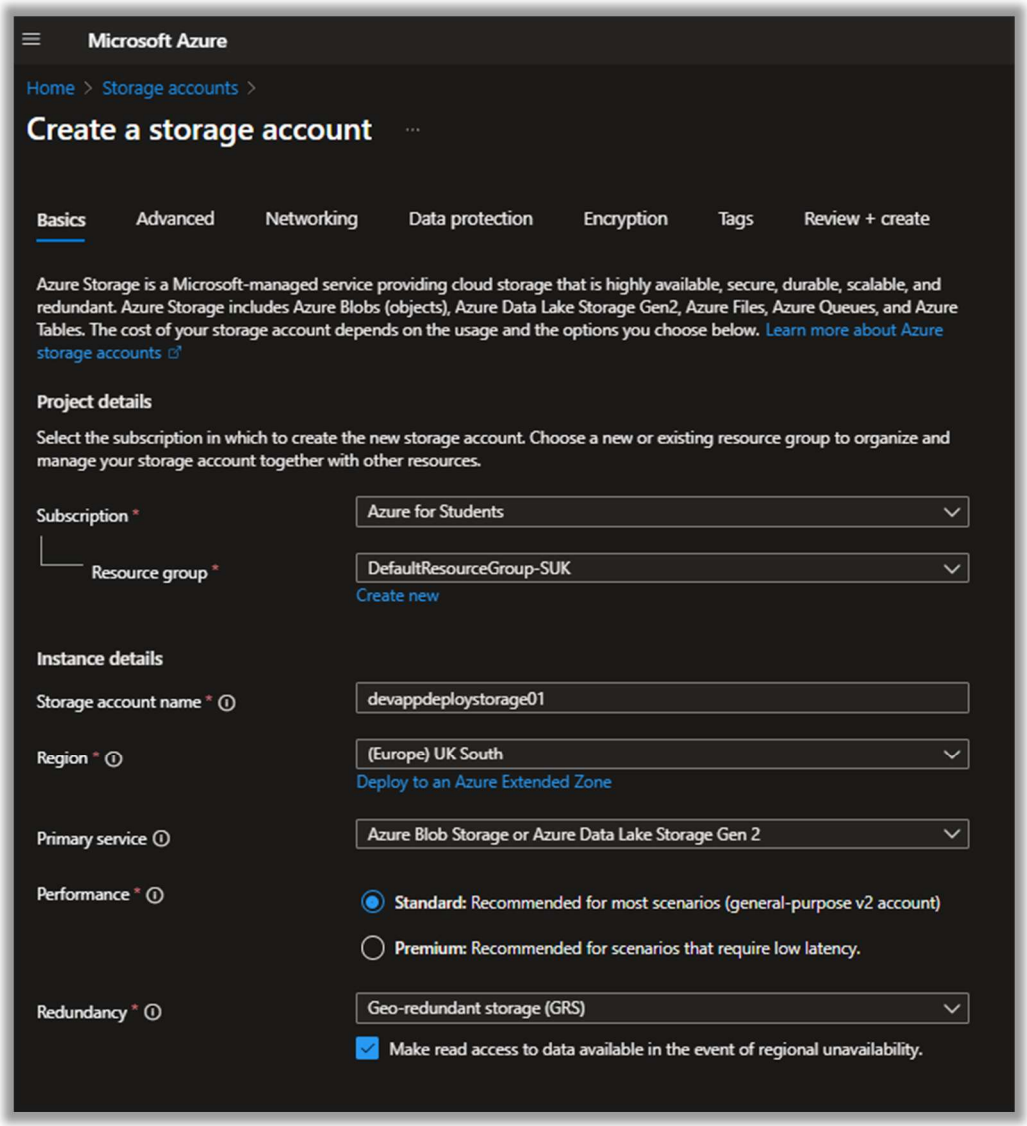

Figure 38 Storage account creation

Figure 39 shows a summary of the storage account configurations.

Figure 39 Storage account configuration

8.2.2 File Upload

To handle file uploads to the blob storage, a method called **UploadFileAsync** was created. This method retrieves a container client, creates the container if it doesn't already exist in Azure, and then it uploads the file with a unique name by using a GUID. The method then returns the URL of the uploaded blob. Figure 40 shows the **UploadFileAsync** Method.

```
2 references
public async Task<string> UploadFileAsync(IBrowserFile file, string containerName)
{
    var containerClient = _blobServiceClient.GetBlobContainerClient(containerName);
    await containerClient.CreateIfNotExistsAsync();

    var blobClient = containerClient.GetBlobClient($"{Guid.NewGuid()}/{file.Name}");

    using var stream = file.OpenReadStream();
    await blobClient.UploadAsync(stream, overwrite: true);

    return blobClient.Uri.ToString();
}
```

Figure 40 UploadFileAsync method

8.2.3 File Download

The download method that was created in the **BlobStorageService** receives a file URL as a parameter. It then parses the URL to extract the container and the blob file name, before generating a shared access signature(SAS) token for secure access and finally downloading the file into a memory stream. The SAS token can also be generated within Azure; however, due to its limited expiry time, it was decided that it should be generated via this method using code. Figure 41 shows the **DownloadFileByUrlAsync** method.

```
2 references
public async Task<Stream> DownloadFileByUrlAsync(string fileUrl)
{
    // Parse the URL to extract container and blob name
    var uri = new Uri(fileUrl);
    var segments = uri.AbsolutePath.TrimStart('/').Split('/', 2);
    if (segments.Length < 2)
        throw new ArgumentException("Invalid blob URL format.");

    string containerName = segments[0];
    string blobName = segments[1];

    var containerClient = _blobServiceClient.GetBlobContainerClient(containerName);
    var blobClient = containerClient.GetBlobClient(blobName);

    // Generate a SAS token (read-only, valid for 5 minutes)
    if (!blobClient.CanGenerateSasUri)
        throw new InvalidOperationException("BlobClient cannot generate SAS URI. Ensure the client uses a key credential.");

    var sasBuilder = new BlobSasBuilder
    {
        BlobContainerName = containerName,
        BlobName = blobName,
        Resource = "b",
        ExpiresOn = DateTimeOffset.UtcNow.AddMinutes(5)
    };
    sasBuilder.SetPermissions(BlobSasPermissions.Read);

    Uri sasUri = blobClient.GenerateSasUri(sasBuilder);

    // Download using the SAS URL
    var sasBlobClient = new BlobClient(sasUri);
    var response = await sasBlobClient.DownloadAsync();
    var memoryStream = new MemoryStream();
    await response.Value.Content.CopyToAsync(memoryStream);
    memoryStream.Position = 0;
    return memoryStream;
}
```

Figure 41 DownloadFileByUrlAsync method

8.2.4 File Deletion

The **DeleteFileByUrlAsync** method was created to remove files from blob storage that are no longer required. Like the download method, it takes a file URL string as a parameter and parses it to identify the container and the blob, before finally deleting the blob if it exists. Figure 42 shows the **DeleteFileByUrlAsync** method.

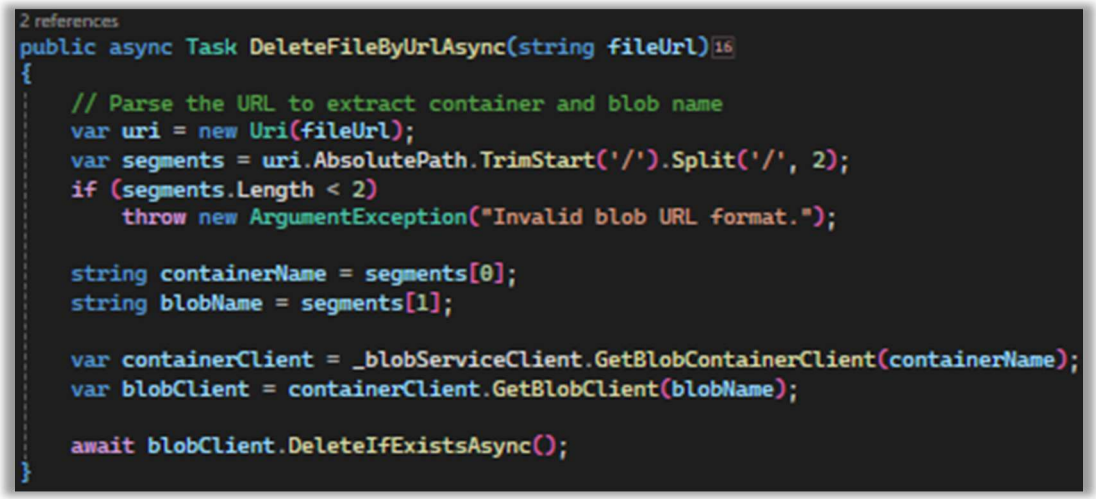

Figure 42 DeleteFileByUrlAsync method

8.3 Role-Based Access Control

A fundamental element of the system identified during the requirements analysis was the ability to restrict pages and actions based on a user's role. The roles showcased in the use case diagram were the developer, who can access all system elements, and the tester, who cannot add or delete applications or releases.

This section outlines the steps taken to implement this critical functionality by leveraging and extending the existing .NET Identity framework.

8.3.1 Seeding Roles Data

The first thing to ensure was that the developer and tester roles exist within the AspNetRoles table in the database when the application is first started. This is achieved by utilising .NET Identity's role manager and then verifying if the required roles exist; if not, they are created. The seeding code is illustrated in Figure 43.

```
using (var scope = app.Services.CreateScope())
{
    var roleManager = scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();
    await SeedRolesAsync(roleManager);
}

1 reference
async Task SeedRolesAsync(RoleManager<IdentityRole> roleManager)
{
    string[] roleNames = { "developer", "tester" };

    foreach (var roleName in roleNames)
    {
        if (!await roleManager.RoleExistsAsync(roleName))
        {
            await roleManager.CreateAsync(new IdentityRole(roleName));
        }
    }
}
```

Figure 43 Method to seed roles into database

8.3.2 User Management Page

To manage access roles among users, a page called the User Management page was created, which existing developer users can access to assign roles.

Figure 44 shows the user management page, which loads all users and displays their current assigned roles. The developer user can then select a new role via the picker and click assign to change the current roles.

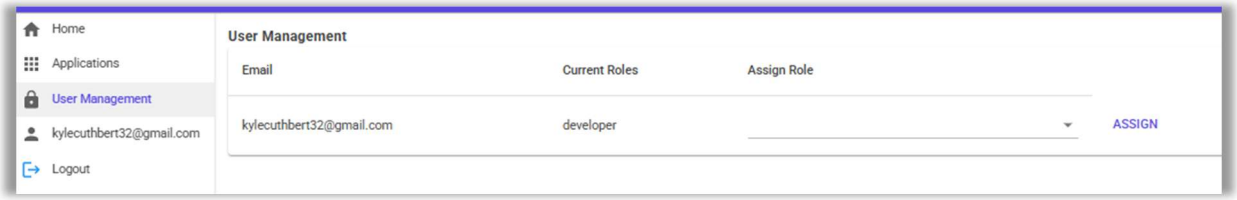

Figure 44 User management page

Figure 45 shows the methods from the code-behind file for the user management page. This includes the **LoadUsersAsync** and **AssignRole** methods that are triggered on the above page. As displayed in Figure 45, the **AssignRole** method calls the **AssignRoleAsync** method within the **RoleAssignmentService**.

```
private List<UserViewModel>? users;
private readonly string[] _roles = { "developer", "tester" };

protected override async Task OnInitializedAsync()
{
    users = await LoadUsersAsync();
}

private async Task<List<UserViewModel>> LoadUsersAsync()
{
    var allUsers = UserManager.Users.ToList();
    var userList = new List<UserViewModel>();

    foreach (var user in allUsers)
    {
        var roles = await UserManager.GetRolesAsync(user);

        userList.Add(new UserViewModel
            {
                Id = user.Id,
                Email = user.Email,
                Roles = roles.ToList()
            });
    }

    return userList;
}

private async Task AssignRole(string userId, string selectedRole)
{
    if (!string.IsNullOrEmpty(selectedRole))
    {
        await RoleAssignmentService.AssignRoleAsync(userId, selectedRole);
        users = await LoadUsersAsync();
    }
}
```

Figure 45 User management page code behind class

8.3.3 Role Assignment Service

The **RoleAssignmentService** injects the **UserManager<ApplicationUser>** within the class constructor that is held in a private field. The **AssignRoleAsync** method takes two parameters: the user's ID and the new role that is being assigned to the user, from the user management page. The user ID is used to find the existing user from the user manager, performing error handling if the user is not found.

Once the user has been found, **GetRolesAsync** is used to retrieve the currently assigned roles, before removing them and assigning the new role. The method can be seen in Figure 46.

```
using DevAppDeploy.Data;
using Microsoft.AspNetCore.Identity;

namespace DevAppDeploy.Services.Roles;

2 references
public class RoleAssignmentService : IRoleAssignmentService
{
    private readonly UserManager<ApplicationUser> _userManager;

    0 references
    public RoleAssignmentService(UserManager<ApplicationUser> userManager)
    {
        _userManager = userManager;
    }

    2 references
    public async Task AssignRoleAsync(string userId, string newRole)
    {
        var user = await _userManager.FindByIdAsync(userId);
        if (user == null)
        {
            throw new ArgumentException("User not found.");
        }

        // Remove all existing roles
        var currentRoles = await _userManager.GetRolesAsync(user);
        if (currentRoles.Any())
        {
            await _userManager.RemoveFromRolesAsync(user, currentRoles);
        }

        // Assign the new role
        await _userManager.AddToRoleAsync(user, newRole);
    }
}
```

Figure 46 RoleAssignmentService

8.3.4 Restricting Access

Role-based access restrictions have been applied, using Blazor's authentication and authorisation features, to the following elements. Please review the graphical user interface design(section 7.9) from the developer's perspective.

Applications page – Only the developer users can access the 'Add Application' button. Figure 47 shows the Application page from a tester's view with no 'Add Application' button.

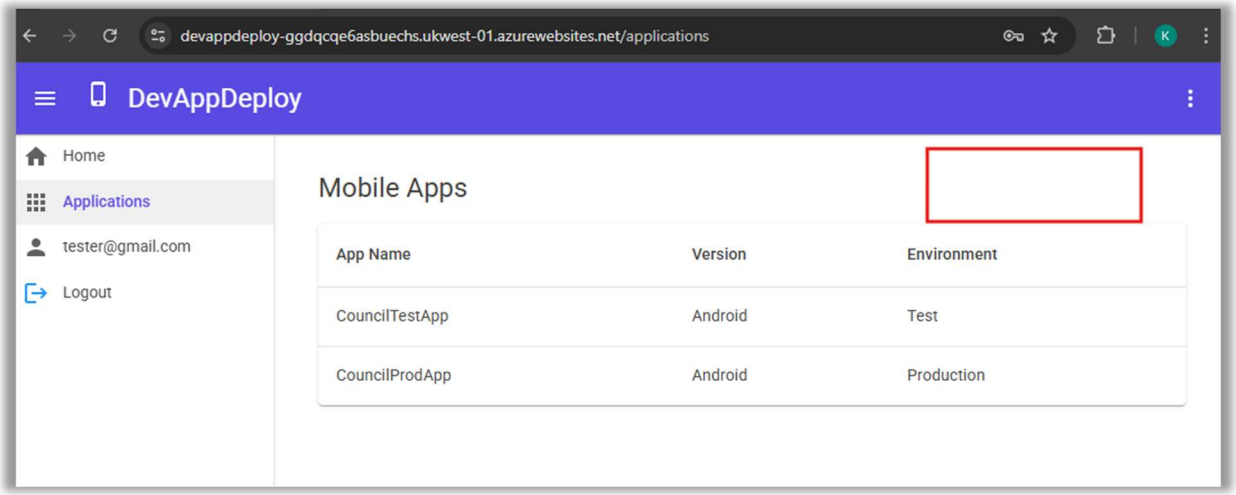

Figure 47 The applications page for a tester user

Release page - Only developer users can access the 'Add Release' button and the 'Delete' column (including the delete button for each release). This is achieved by using the AuthenticationStateProvider to retrieve the current user's authentication state and verifying if the user holds the developer role. Figure 48 shows the Release page with a tester user logged in. Note the absence of the 'Add Release' button and the delete comment.

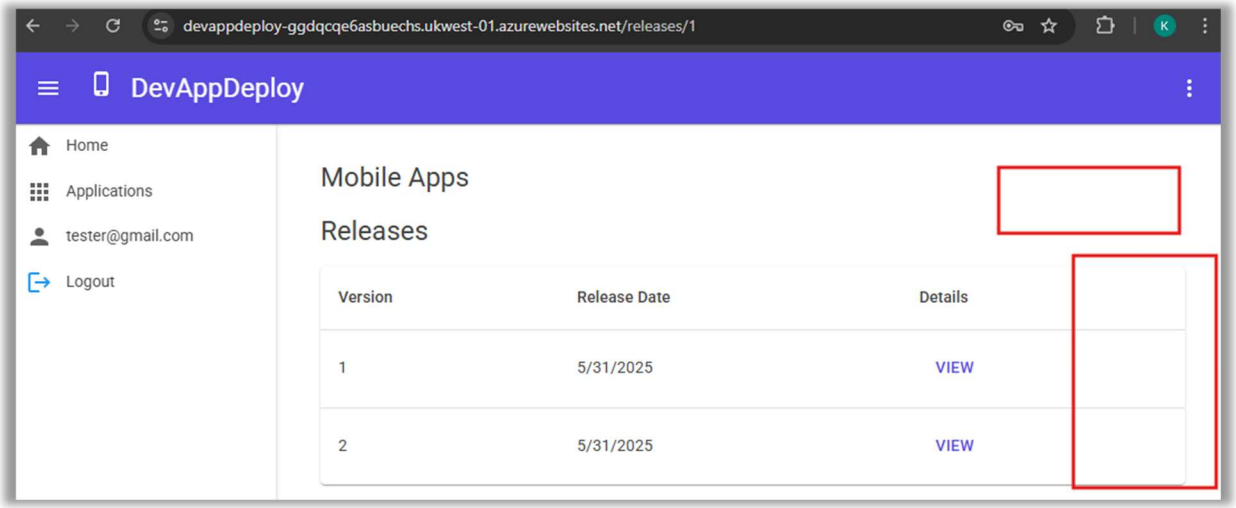

Figure 48 The release page for a tester user

8.4 Application Management

One of the main aspects of DevAppDeploy is the management of applications. The purpose of this section is to outline how this part of the application has been implemented, using Entity Framework Core to provide a robust data layer, business logic within a dedicated service that can be injected into the UI code or other services, and a user interface using Blazor and Mudblazor components.

8.4.1 Application Data

Following the design section of the software development lifecycle, the entity-relationship diagram guides the structure of this iteration of the data layer.

By using EF Core, a code-first approach was taken, utilising a class model to specify the design of the data table. The 'ApplicationTbl' captures the essential fields such as the unique Id, which will be the primary key, as well as the application name, the operating system and the environment name. Figure 49 shows the 'ApplicationTbl'.

Figure 49 ApplicationTbl

8.4.2 Application Service

The 'ApplicationService' implements the 'IApplicationService' and serves as the central point for all application-related business logic and interactions with the database. The service utilises the 'ApplicationDbContext' to communicate with the SQL database via EF Core.

The key functionalities provided by the ApplicationService include:

- CreateApplicationAsync – This method takes a parameter of 'CreateApplicationModel', which accepts all pertinent information from the user interface and maps the properties to the 'ApplicationTbl' entity before persisting it to the database and correctly recording the new application. Figure 50 shows the 'CreateApplicationAsync'.

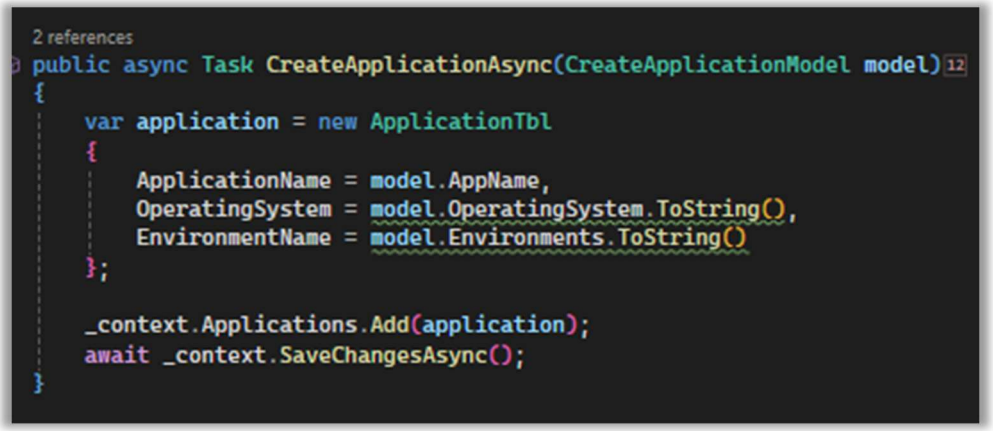

Figure 50 CreateApplicationAsync method

- GetApplicationsAsync – This method retrieves all applications currently stored in the database and returns them as a list of 'DisplayApplicationModel' instances for display in the user interface. This method is illustrated in Figure 51.

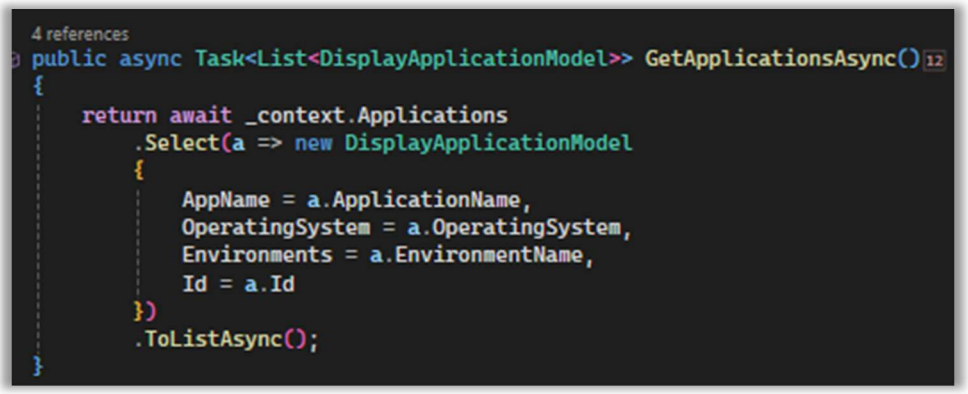

Figure 51 GetApplicationsAsync method

8.4.3 Application Page

The main page for managing applications is the 'Applications' page (Application.razor), which requires the user to be authenticated to access. Figure 52 shows the Applications page user interface, built with Blazor and utilising MudBlazor components, to interact with the ApplicationService.

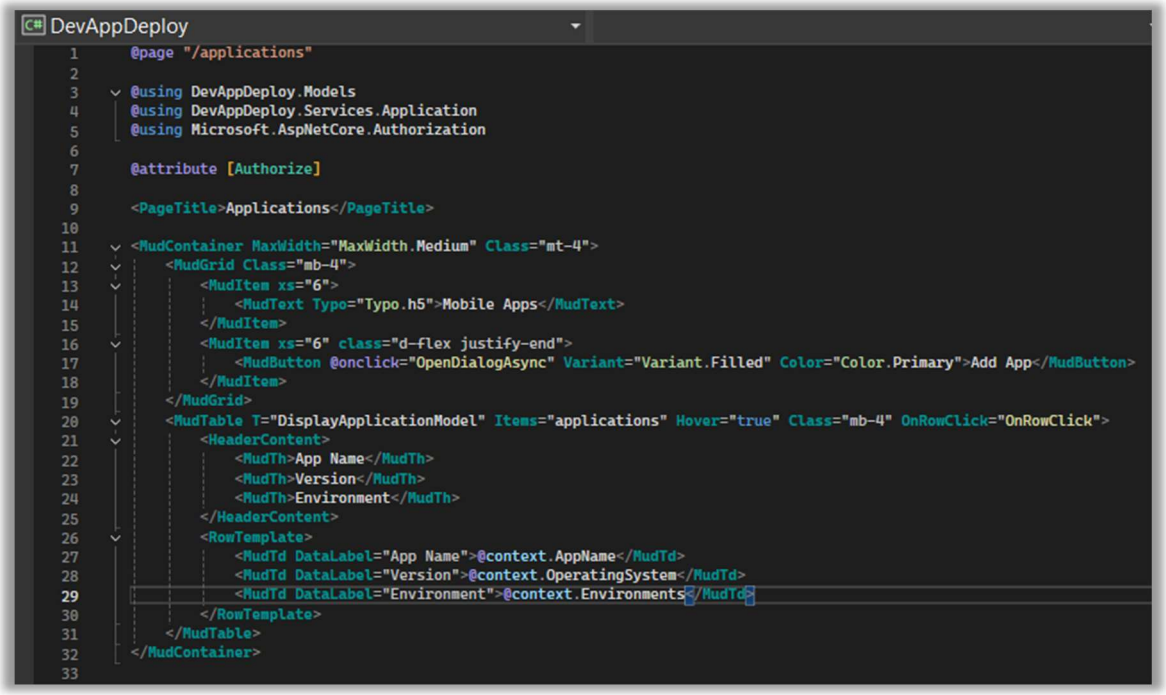

Figure 52 Applications page UI code

The Applications page is powered by a corresponding code-behind file that handles any user interactions with the page and uses the ApplicationService to retrieve data. The code behind file can be seen in Figure 53.

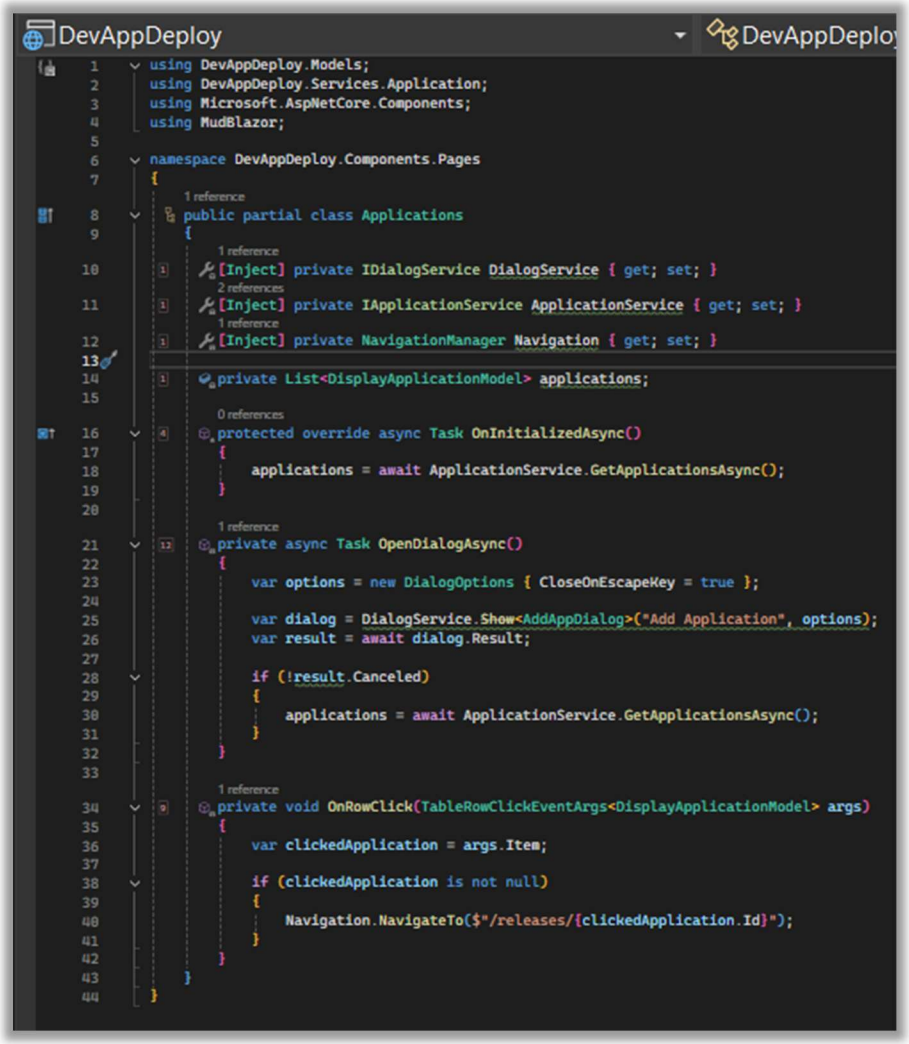

Figure 53 Application page code behind class

On initialisation, the page will use the ApplicationService.GetApplicationsAsync() to retrieve the list of applications and display them to the page using the MudTable component.

On the Applications page, there is also an 'Add App' button that allows authorised users (users with the developer role) to open the 'AddAppDialog' form, enabling them to create new applications.

Finally, the user can click on any row in the MudTable to navigate the user to a /releases/{applicationId} page, which allows the user to drill down into all the releases for the chosen application.

8.5 Release Management

As mentioned in Section 7.4.3, an application row on the Applications page can be clicked to display a list of all releases for that application. This section will examine some of the methods relevant to managing releases.

8.5.1 Getting Releases For An Application

For the user interface code to retrieve releases for the chosen application, it will once again use the ApplicationService by calling the **GetReleasesByApplicationId** method. This method will take the ID for the application as a parameter, then use the database context to return a list of releases (by populating a **DisplayReleaseModel**) where the release's foreign key matches the ID of the application. Figure 54 shows the **GetReleasesByApplicationId** method.

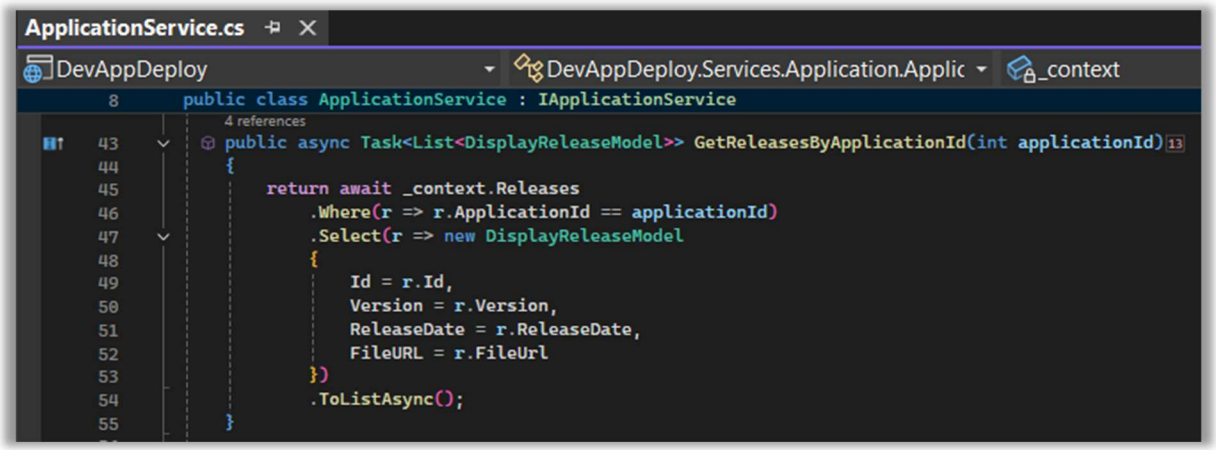

Figure 54 GetReleasesByApplicationId method

8.5.2 Creating A Release

On the page showing an application's releases, a developer user can click to add a new release. While the BlobStorageService handles the upload of the release's APK file, the ApplicationService, specifically the CreateReleaseAsync method, handles the insertion of the release into the Release table within the database. As shown in Figure 55, the process takes a **CreateReleaseModel** that has captured the data input from the **CreateReleaseDialog** form. It then uses the ID to find the application and checks if the application is null, throwing an exception if so. The model data is then added to the release data model(**ReleaseTbl**) and added to the database context.

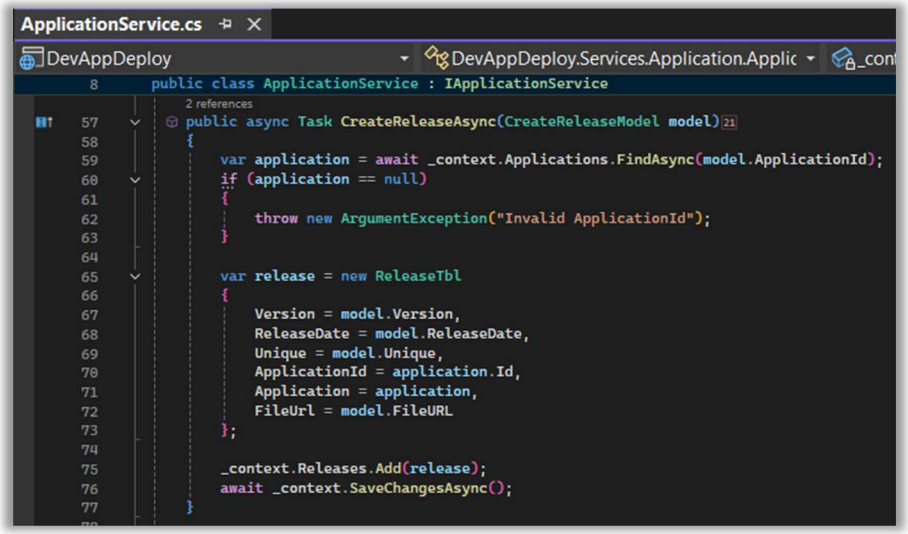

Figure 55 CreateReleaseAsync method

8.5.3 Deleting A Release

To remove a release from the database, a method called **DeleteReleaseByIdAsync** was added to the ApplicationService, as shown in Figure 56. It requires a release ID to find the release, checks if it is not null to ensure it is present in the database and then removes it from the database context and saves the changes.

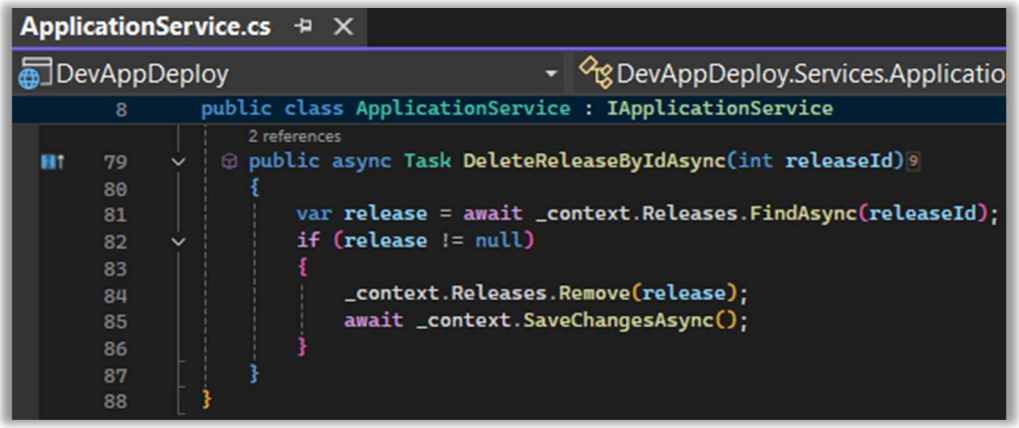

Figure 56 DeleteReleaseByIdAsync method

8.6 Hosting

Based on the literature review, it was decided, as part of the methodology, to employ Azure App Service to host the prototype of DevAppDeploy, along with an Azure SQL Database.

8.6.1 App Service Set-up

Within the Azure portal, the app service was created with a student subscription and configured to use the same runtime stack as the DevAppDeploy application. By using an app service on Azure, the runtime security patches and updates will be handled automatically. Figure 57 shows the app service being created under the resource group RG-1.

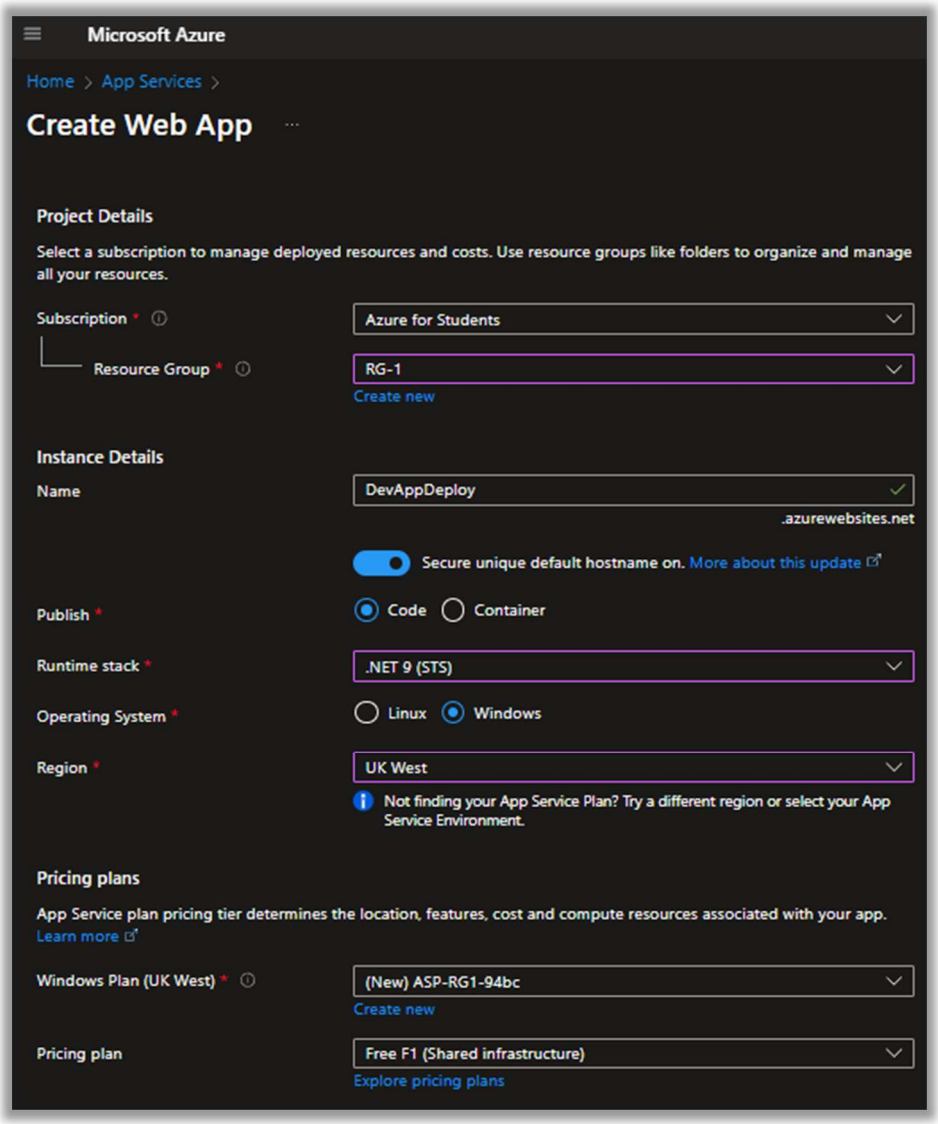

Figure 57 App Service creation

8.6.2 Database Set-up

Before the application can be deployed to the app service, the database needs to be created. Firstly, an SQL Server (devappdeployserver) was created on Azure before creating the database (devappdeploydb) itself. The database was also created in the same resource group as the app service, allowing costs to be monitored and organising all resources in one place. To enable EF Core migrations from the application database design to be implemented in the created database, the network settings were configured for public access, and a firewall rule was created to allow only the developer's IP address to access and modify the server. Figure 58 shows the database creation.

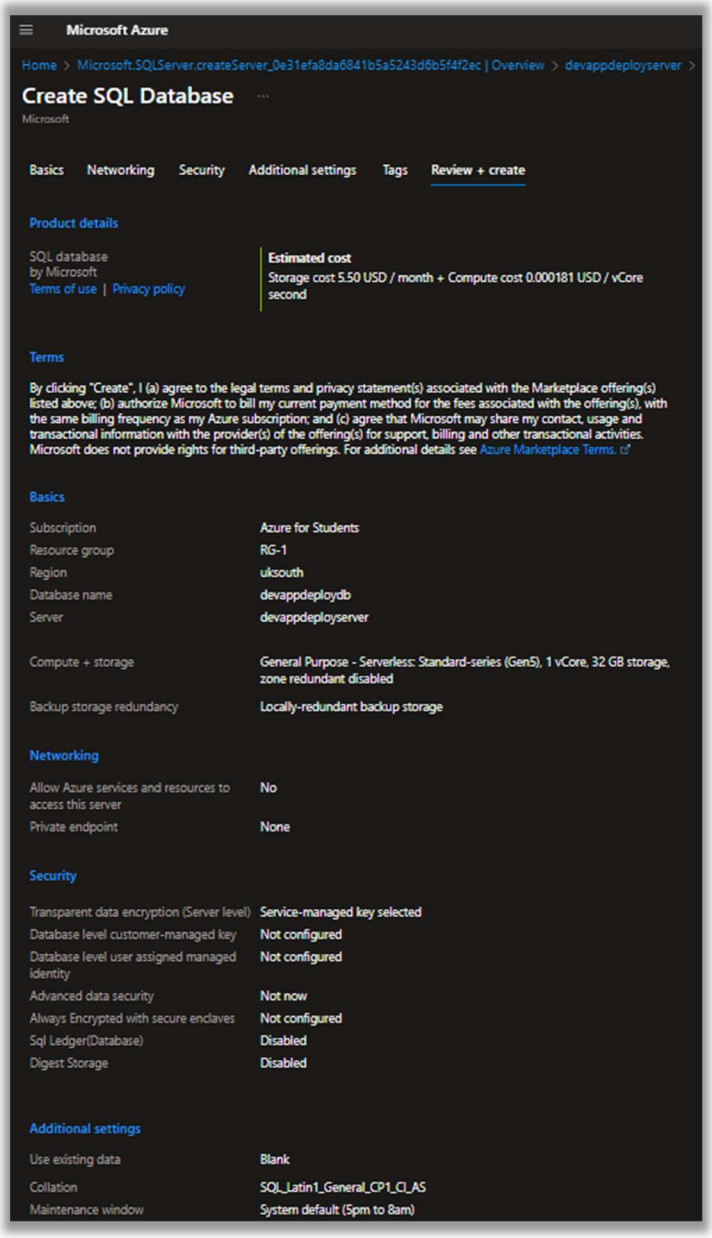

Figure 58 Database creation

8.7 Testing

Testing is an essential element for quality assurance that helps identify bugs and issues so that they can be addressed as early as possible. For DevAppDeploy, various types of testing have been employed to ensure thorough testing to maintain reliability throughout the entire solution.

By employing the Agile SDLC method, ensure that testing is integrated into the development by offering continuous feedback for iterative improvements. This section outlines the various testing approaches used in DevAppDeploy, establishing a solid foundation that can be expanded to achieve broader test coverage.

8.7.1 Unit Testing

The first method of testing was to create unit tests to verify the correctness of individual components of the application, to ensure functions and methods behaved as expected.

A new test project was created within the solution, referencing the Blazor project. A testing framework called XUnit[3] was utilised to create the unit tests as it works with .NET and can be used within Visual Studio. To assist with simulating dependencies, a mocking library called NSubstitute[4] was employed to help create the unit tests by mocking the services, allowing unit tests to be run in isolation from the actual services that interact with real data.

One of the methods in which unit tests were applied was the Submit method from the code-behind class of the AddReleaseDialog. Figure 59 illustrates the Submit method that will be unit-tested.

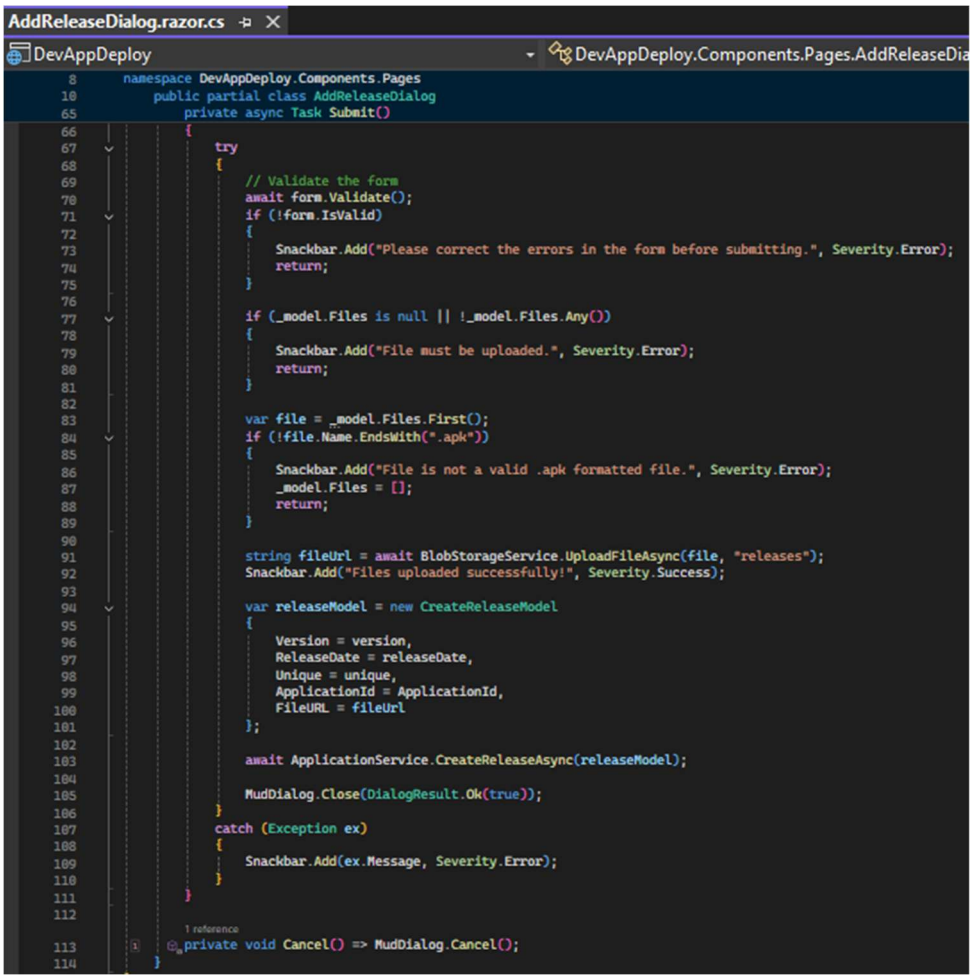

Figure 59 AddReleaseDialog Submit method

[3] https://xunit.net/
[4] https://nsubstitute.github.io/index.html

Table 8 summarises the test cases written before the unit tests.

Table 8 Submit method test cases.

Test Name	Form Valid	File Selected	File Type	Blob Upload	Release Created	Dialog Closed	Error Shown	Exception Simulated
Submit_ValidFormAndApkFile	Yes	Yes	APK	Yes	Yes	Yes	No	No
Submit_InvalidForm	No	N/A	N/A	No	No	No	No	No
Submit_NonApkFile	Yes	Yes	Not APK	No	No	No	Yes(“APK” error)	No
Submit_NoFileSelected	Yes	No	N/A	No	No	No	Yes(“file” error)	No
Submit_ApplicationServiceThrows	Yes	Yes	APK	No	No	No	Yes(“Created failed” error)	Yes(on create)

The following sections show the various unit tests that were written within the ‘DevDepoly.Tests’ project.

Submit_ValidFormAndApkFile_CreatesReleaseAndClosesDialog

```
namespace DevAppDeploy.Tests;
0 references
public class AddReleaseDialogTests
{
    [Fact]
    0 references
    public async Task Submit_ValidFormAndApkFile_CreatesReleaseAndClosesDialog()
    {
        // Arrange
        var appService = Substitute.For<IApplicationService>();
        var blobService = Substitute.For<IBlobStorageService>();
        var snackbar = Substitute.For<ISnackbar>();
        var dialog = Substitute.For<IMudDialogInstance>();

        var component = new AddReleaseDialog
        {
            ApplicationService = appService,
            BlobStorageService = blobService,
            Snackbar = snackbar,
            MudDialog = dialog,
            ApplicationId = 1
        };

        // Set private fields using reflection
        var form = Substitute.For<MudForm>();
        //form.IsValid.Returns(true);
        typeof(AddReleaseDialog)
            .GetField("form", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance)
            .SetValue(component, form);

        var file = Substitute.For<IBrowserFile>();
        file.Name.Returns("test.apk");
        var files = new List<IBrowserFile> { file };
        typeof(AddReleaseDialog)
            .GetField("_model", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance)
            .SetValue(component, new AddReleaseDialog.FileUploadModel { Files = files });

        blobService.UploadFileAsync(file, "releases").Returns(Task.FromResult("http://blob/file.apk"));

        // Act
        var submitMethod = typeof(AddReleaseDialog)
            .GetMethod("Submit", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance);
        var task = (Task)submitMethod.Invoke(component, null);
        await task;

        // Assert
        await blobService.Received(1).UploadFileAsync(file, "releases");
        await appService.Received(1).CreateReleaseAsync(Arg.Any<CreateReleaseModel>());
        dialog.Received(1).Close(Arg.Any<DialogResult>());
    }
```

Figure 60 Unit test - submit valid form

Submit_InvalidForm_DoesNotCreateReleaseOrCloseDialog

AddReleaseDialogTests.cs

DevAppDeploy.Tests

```
10      public class AddReleaseDialogTests
57
58          [Fact]
            0 references
59          public async Task Submit_InvalidForm_DoesNotCreateReleaseOrCloseDialog()
60          {
61              var appService = Substitute.For<IApplicationService>();
62              var blobService = Substitute.For<IBlobStorageService>();
63              var snackbar = Substitute.For<ISnackbar>();
64              var dialog = Substitute.For<IMudDialogInstance>();
65
66              var component = new AddReleaseDialog
67              {
68                  ApplicationService = appService,
69                  BlobStorageService = blobService,
70                  Snackbar = snackbar,
71                  MudDialog = dialog,
72                  ApplicationId = 1
73              };
74
75              // Use a real MudForm instance to avoid null reference
76              var form = new MudForm();
77              typeof(AddReleaseDialog)
78                  .GetField("form", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance)
79                  .SetValue(component, form);
80
81              var submitMethod = typeof(AddReleaseDialog)
82                  .GetMethod("Submit", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance);
83              var task = (Task)submitMethod.Invoke(component, null);
84              await task;
85
86              await blobService.DidNotReceive().UploadFileAsync(Arg.Any<IBrowserFile>(), Arg.Any<string>());
87              await appService.DidNotReceive().CreateReleaseAsync(Arg.Any<CreateReleaseModel>());
88              dialog.DidNotReceive().Close(Arg.Any<DialogResult>());
89          }
```

Figure 61 Unit test - submit invalid form

Submit_NonApkFile_ShowsErrorAndDoesNotCreateReleaseOrCloseDialog

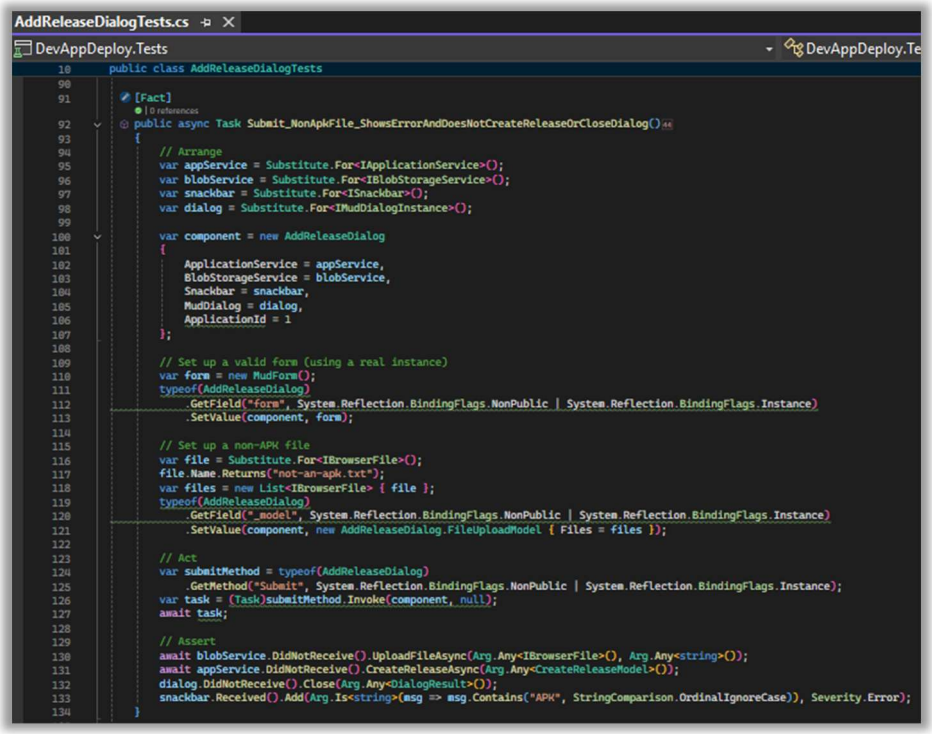

Figure 62 Unit test - submit non-apk file

Submit_NoFileSelected_ShowsErrorAndDoesNotCreateReleaseOrCloseDialog

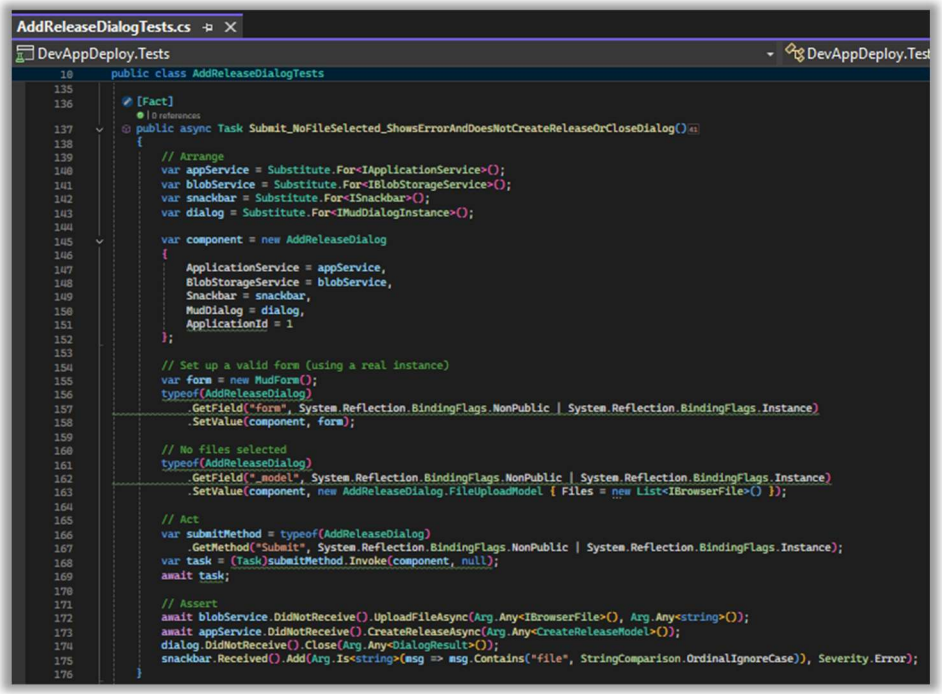

Figure 63 Unit test - submit no file selected

Submit_ApplicationServiceThrows_ShowsErrorAndDoesNotCloseDialog

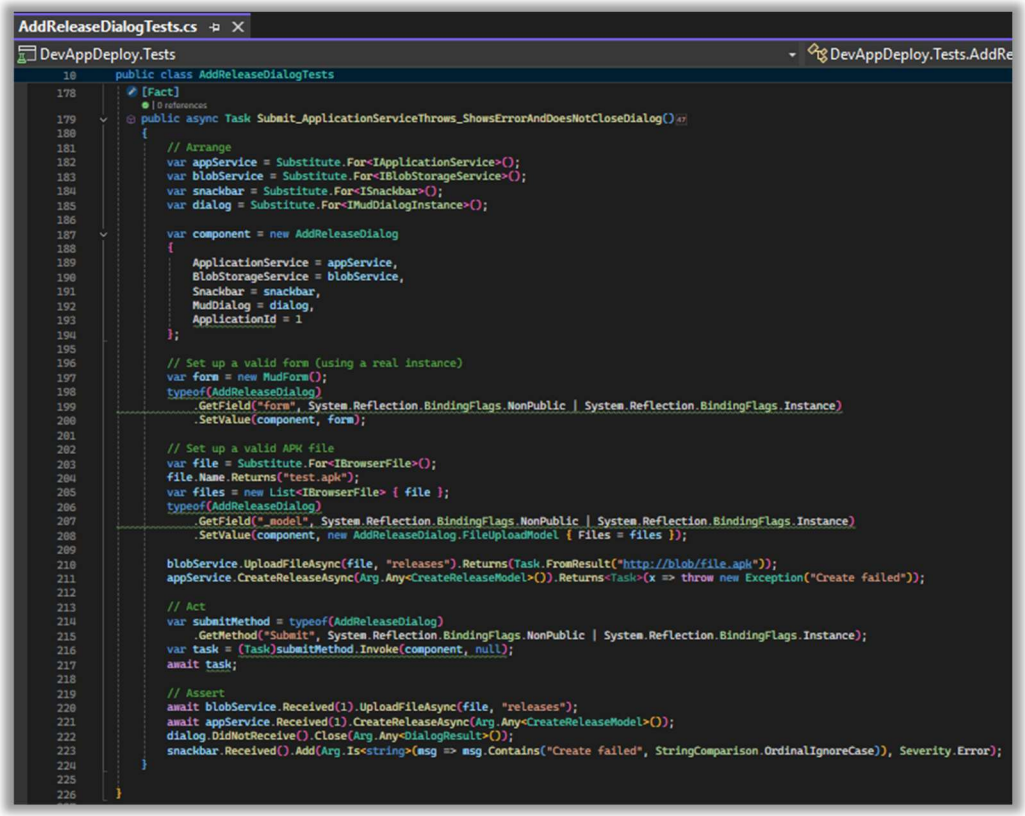

Figure 64 Unit test - submit with application service exception

Unit Test Results

Figure 65 shows the test runner showing the results of the unit tests.

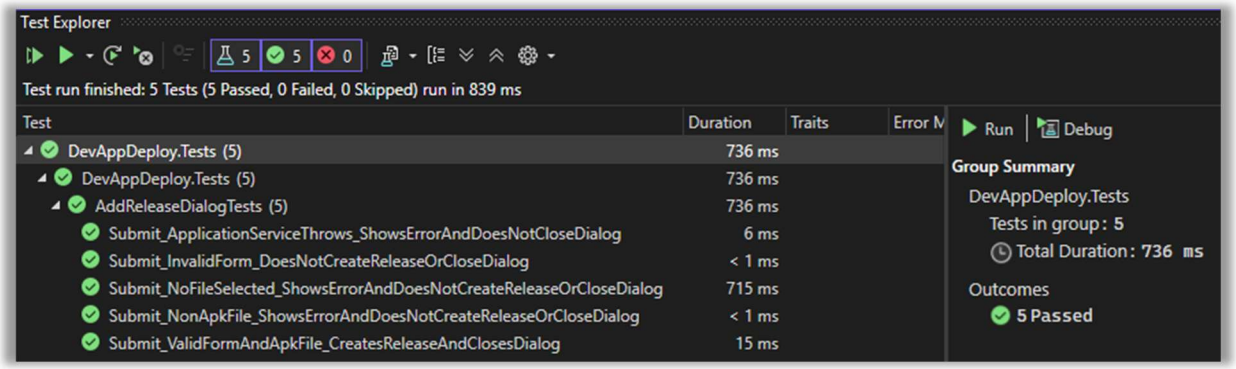

Figure 65 Test runner showing unit test results

8.7.2 End-to-End User Testing

In addition to the automated unit testing, manual testing was conducted on the application. This section outlines the pertinent test cases to validate the core functionalities of the DevAppDeploy application. The test cases focus on critical user flows such as registration, login, application and release management, as well as testing the download and installation of an Android APK file onto a user's device. Table 9 below summarises each of the test cases that have been conducted.

Table 9 Test cases

Test Case	Description	Testing Type	Expected Result	Status
Login_ValidDetails	User submits valid details to the login form	Functional/UI	User is signed in	Pass
Login_InvalidPassword	User submits an invalid password	Validation/UI	Error message is displayed	Pass
Login_InvalidUsername	User submits an invalid user email	Validation/UI	Error message is displayed	Pass
Registration_ValidDetails	User submits valid details that do not exist in the system	Functional/UI	User registered and signed in	Pass
Registration_AccountExists	User submits an existing user	Validation/UI	Error message is displayed	Pass
CreateApplication_ValidInput	Create a new application with valid data	Functional/UI	Application is added and appears in the application list	Pass
CreateApplication_MissingFields	Attempt to create application with missing required fields	Validation/UI	Validation errors shown, application not created	Pass
ListApplications_Display	View the list of all applications	Functional/UI	All created applications are displayed with correct details	Pass
CreateRelease_ValidInput	Create a new release for an existing application	Functional/UI	Release is added and appears in the release list for the application	Pass
CreateRelease_MissingFields	Attempt to create release with missing required fields	Validation/UI	Validation errors shown, release not created	Pass

ListReleases_Display	View all releases for a selected application	Functional/UI	All releases for the application are displayed with correct details	Pass
DeleteRelease_Valid	Delete a release from an application	Functional/UI	Release is removed from the list and database	Pass
CreateRelease_InvalidAppId	Attempt to create a release for a non-existent application	Validation/UI	Error message shown, release not created	Pass
Login_AndroidDevice	Log in to the DevAppDeploy system from an Android device	Functional/UI	User is authenticated and redirected to the home page	Pass
DownloadAPK_Android	Download the APK file from the application on an Android device	Functional/UI	APK file is downloaded successfully to the device(This can be seen demonstrated in the video link within the video demonstration section)	Pass
InstallAPK_Android	Install the downloaded APK on the Android device	Functional/UI	APK installs successfully, and the app launches without errors	Pass

8.7.3 Load Testing

Azure Load Testing was employed to evaluate the scalability and performance of the DevAppDeploy system by simulating multiple user requests to see how the system handles increased loads. However, there are limitations to using this feature in Azure when using the F1 free pricing tier, as it is predominantly designed for small-scale apps. Therefore, there are limitations, especially regarding CPU usage.

Firstly, a load testing resource needs to be created as shown in Figure 66.

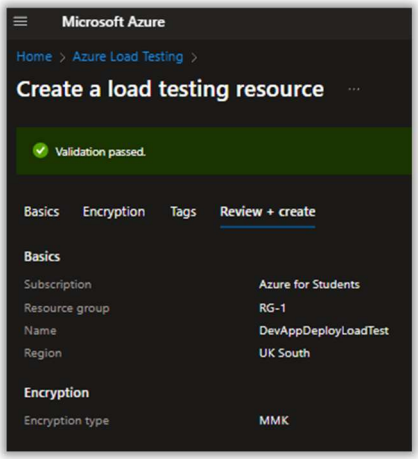

Figure 66 Create a load test resource

A URL-based test was created against the home page endpoint, and despite the free tier limitations, the test successfully ran over 62,000 load tests within a 1 minute 54 second duration. The results are presented in Figure 67, which shows an acceptable response time of 113 ms and an error rate of 0.12%. While more tests could be added to ensure a more comprehensive evaluation of performance, this test has demonstrated how non-functional performance requirements can be assessed.

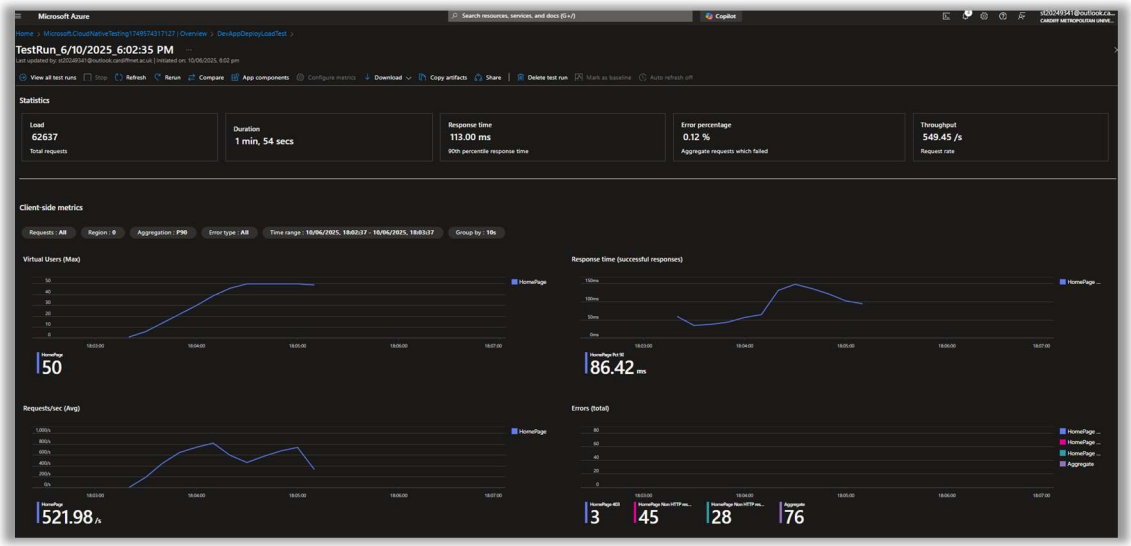

Figure 67 Load test results

8.8 CI/CD

To streamline the deployment process of publishing the application to Azure, as well as adding robust actions to ensure quality control and restrictions on what is published, it was decided to create CI/CD pipelines within Azure DevOps. The pipelines would leverage the two distinct branches created within the Azure Repos code repository, which were previously set up (dev and master).

8.8.1 Build Pipeline

To achieve continuous integration, a build pipeline was created that is automatically triggered when commits are made to the dev branch. The build pipeline performs a NuGet restore, builds the application, runs the automatic unit tests and publishes the artefacts in a zip file, including the DevAppDeploy deployment package. Figure 68 shows the build pipeline.

Figure 68 Build pipeline

Figure 69 shows the build pipeline running successfully, including the '**UnitTest**' step that has successfully run and passed all the unit tests written.

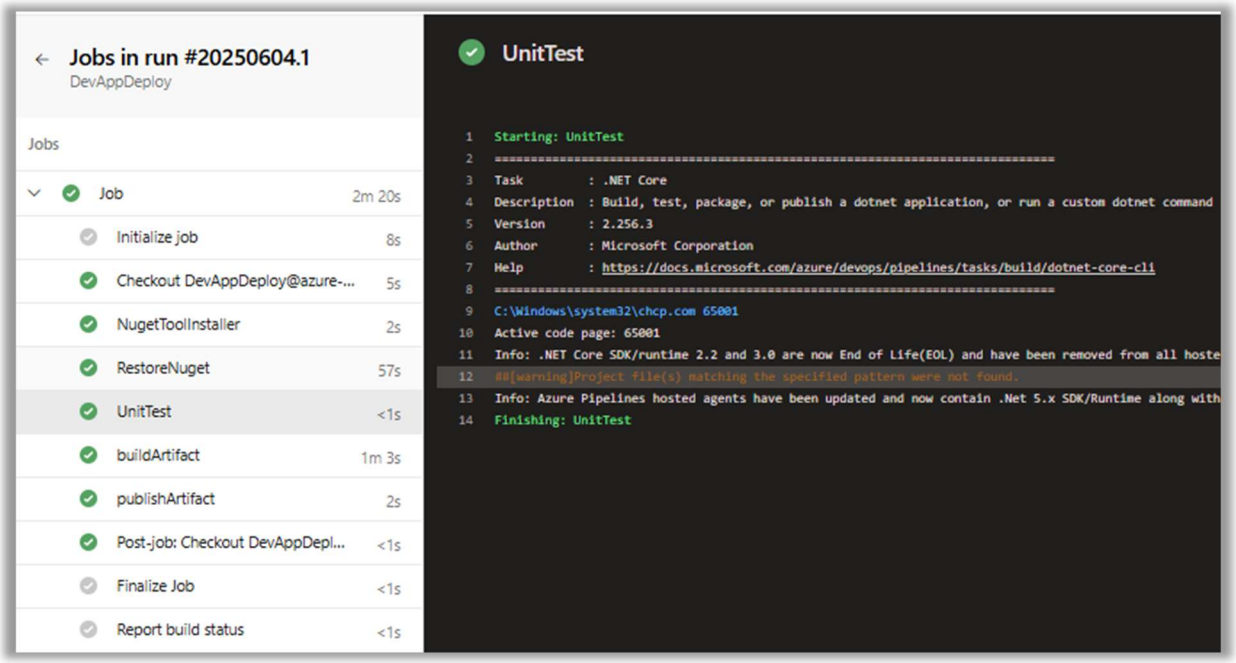

Figure 69 The build pipeline running successfully

8.8.2 Release Pipeline

The release pipeline handles continuous delivery. This pipeline is linked to the master branch and is triggered once a pull request has been accepted and merged into the master branch. This pipeline consists of two steps: the first is to download the artefact created by the build pipeline, and the second is to deploy the artefact to the app service using an Azure Service Connection with a managed identity for secure authentication. Figure 70 shows the release pipeline.

Figure 70 Release pipeline

Figure 71 shows a successfully run release pipeline highlighting the app service on which it was deployed.

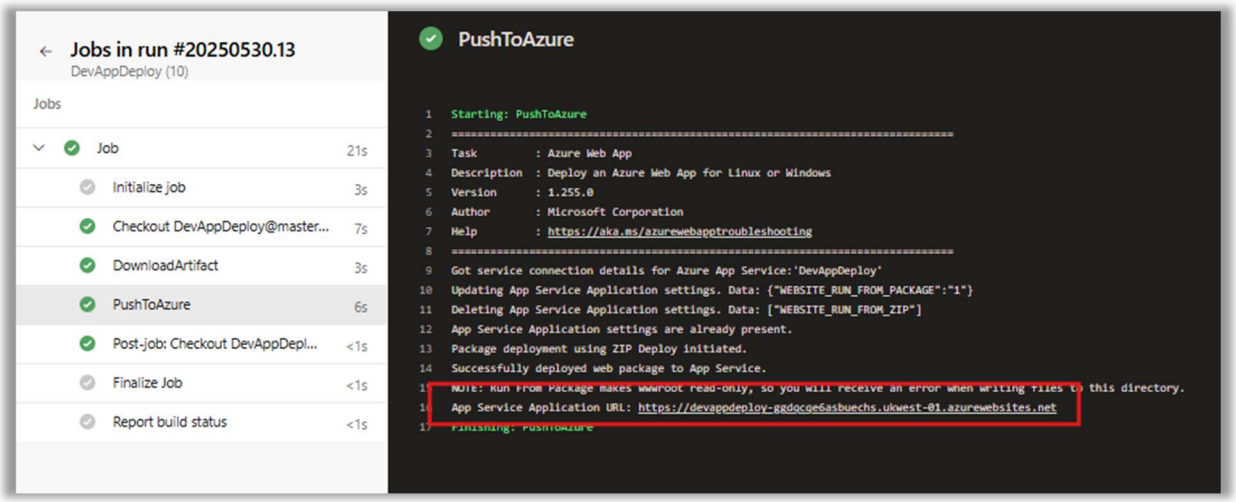

Figure 71 Successfully run release pipeline

By automating the workflow, it ensures an efficient, reliable, and repeatable deployment from development to production.

8.9 Monitoring

A crucial part of managing this system is monitoring to ensure reliability, performance, and availability. This section will discuss how Azure Application Insights was implemented to provide an understanding of the system's health, user engagement and operational metrics. These aspects directly contribute to ensuring that non-functional requirements specified in the previous sections are achieved, such as performance (7.2.1), availability (7.2.4), and maintainability (7.2.5).

8.9.1 Availability

Providing a system that users can reliably access at all times is a fundamental requirement for the overall system experience.

Setting up an availability test

Within the Azure portal, availability tests, such as the one shown in Figure 72, were configured to proactively detect any issues with availability or downtime. The test in Figure 72 was based on the URL of the home page being tested every 5 minutes from 5 different test locations.

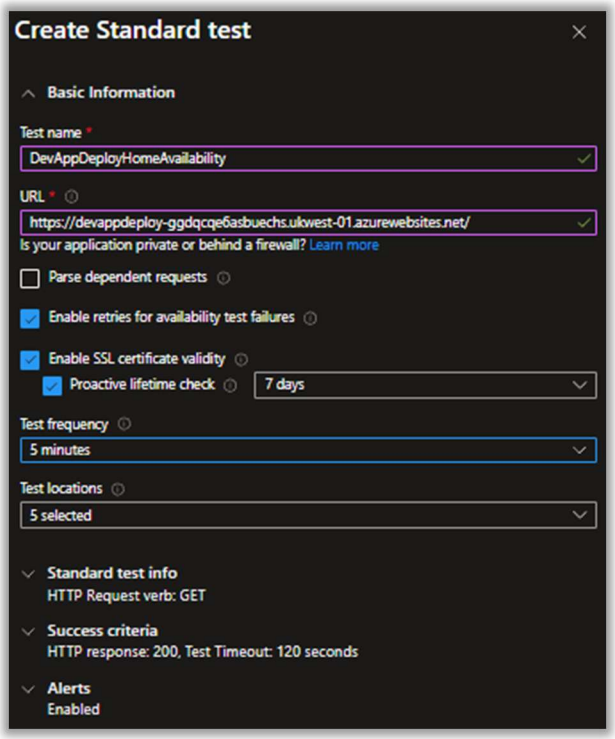

Figure 72 Creation of an availability test

Availability Stats

By configuring the test availability, uptime is tracked and presented within the availability dashboard, allowing developers to drill down into periods of downtime for a better understanding of issues, whilst also building confidence with the system's accessibility. Figure 73 shows the availability of DevAppDeploy and highlights a warning of a purposefully created downtime.

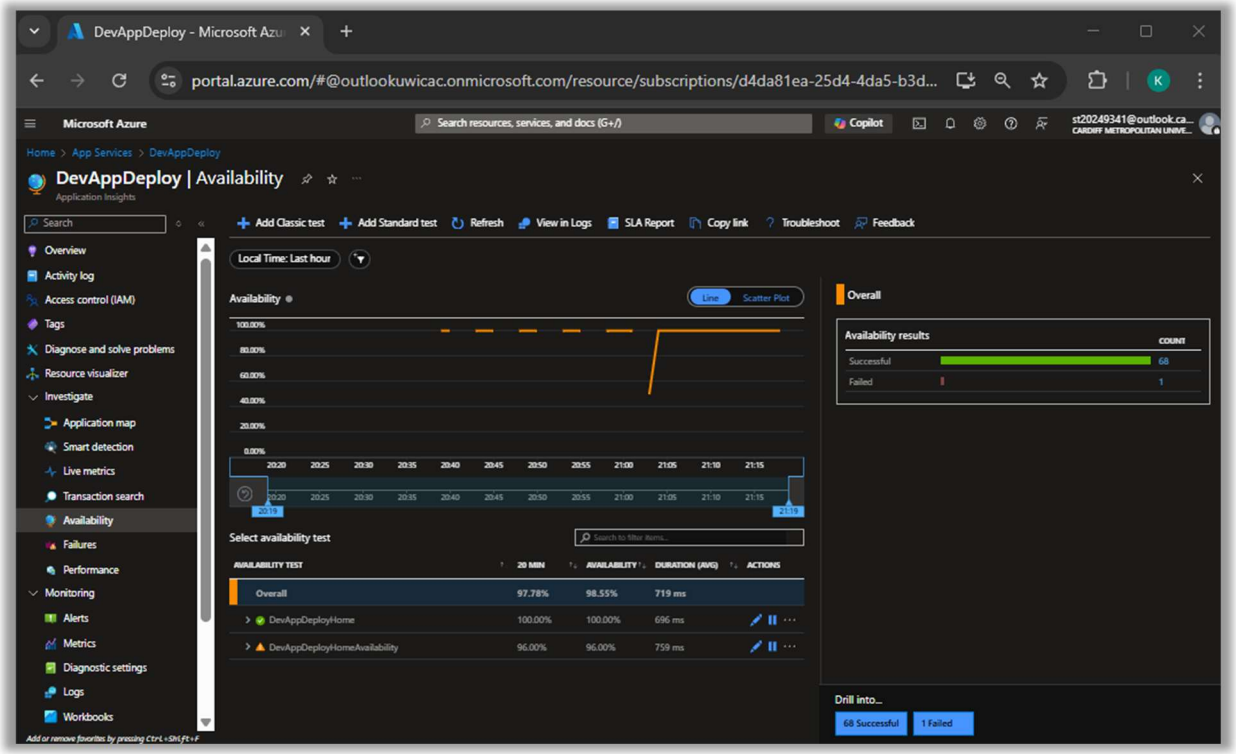

Figure 73 Availability monitoring

8.9.2 Performance & Failures

Monitoring performance is key to identifying bottlenecks and detecting failures, which aligns with the non-functional requirements outlined in Section 7.2.1, Performance, and Section 7.2.5, Maintainability.

Performance

Figure 74 shows the performance dashboard where insights such as latency, response times, and resource consumption are available to promote optimisation and can contribute to making the system more performant.

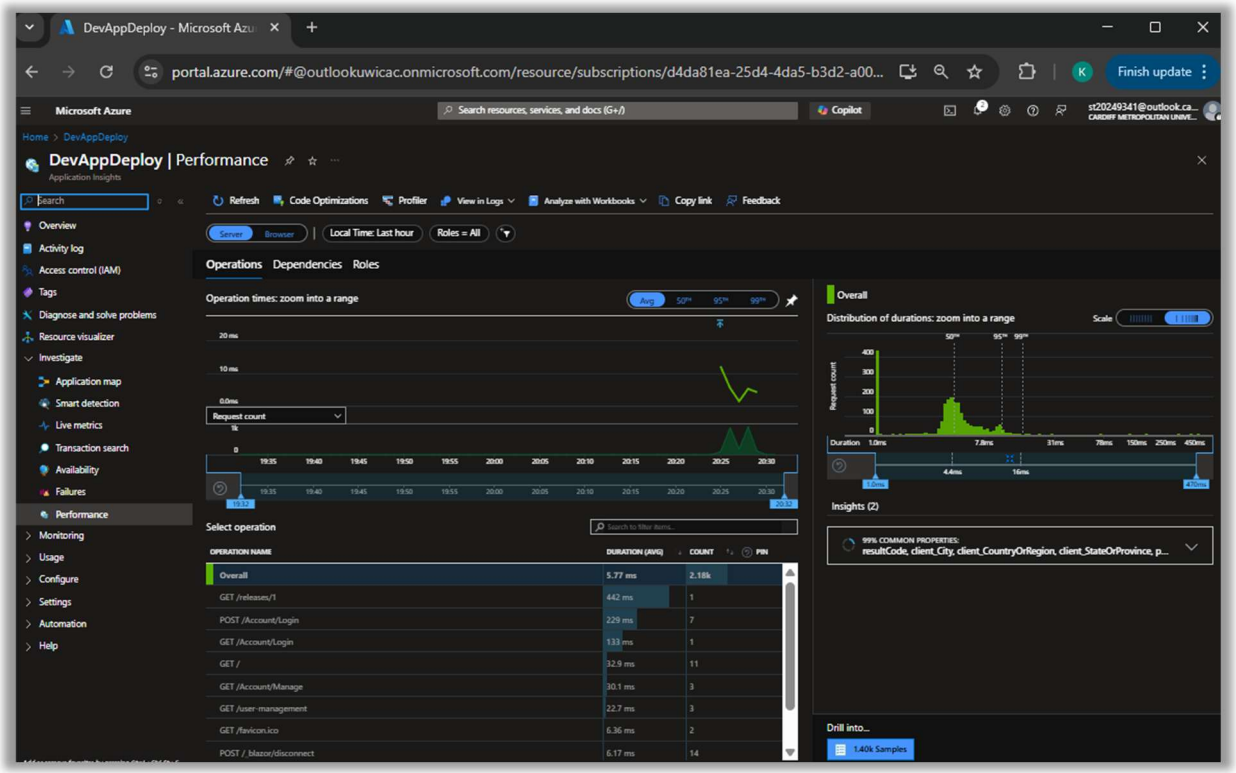

Figure 74 Performance monitoring

Failures and Live Metrics

The failures and live metrics insights can provide real-time error tracking and diagnostic capabilities, allowing for the immediate detection with the ability to drill down for more comprehensive information about the failures. Figure 75 shows the 'Failures' dashboard with simulated errors.

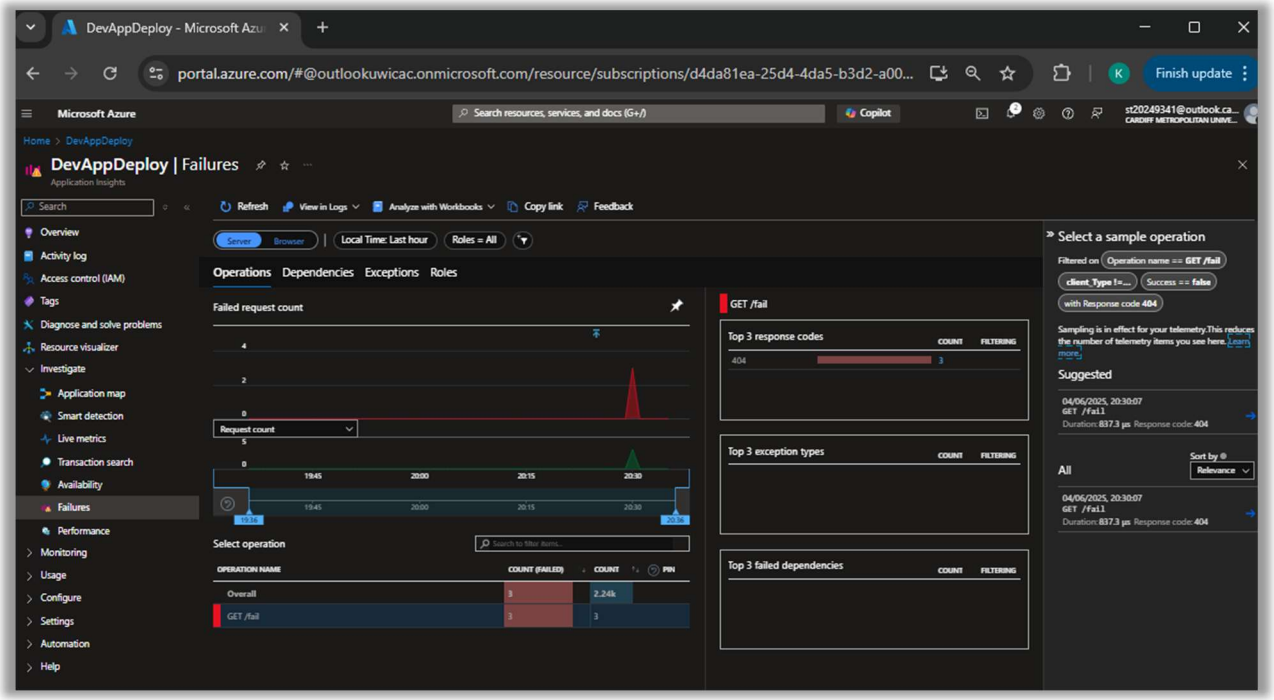

Figure 75 Failure monitoring

Figure 76 displays the live metrics screen, which provides various insights related to the DevAppDeploy application.

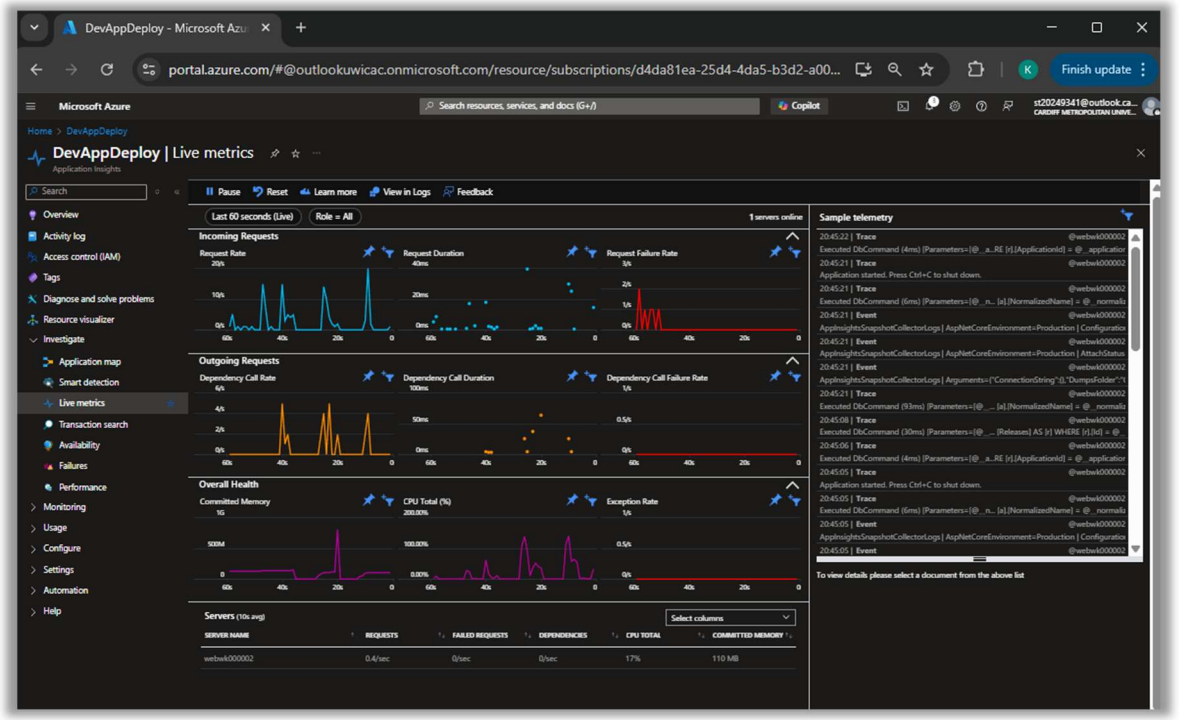

Figure 76 Live metrics showing various insights

8.9.3 Alerts

Alerts in Azure Application Insights provide notifications for any potential issues that require attention. For DevAppDeploy, the following alerts have been created:

- Availability Alert: This alert triggers when availability drops below 90%, helping to detect potential downtime.
- Failed Location Alerts: These alerts are raised when failures occur within two or more locations to identify regional service disruptions.

These alerts are illustrated in Figure 77.

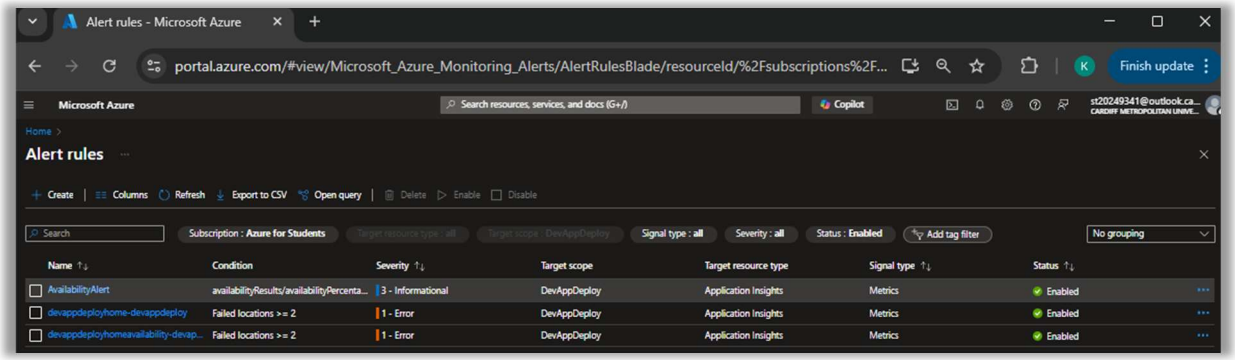

Figure 77 Alerts for availability and failed locations

9. Interpretation and Conclusions

The purpose of this project was to successfully develop DevAppDeploy as a prototype for mobile application distribution, tailoring it to meet the needs of Cardiff Council's internal beta testing. This platform addresses critical gaps left by the discontinuation of Microsoft's Visual Studio App Center, which is missing from suggested alternative tools, such as the Google Play Console.

Using the newly retired App Center as the basis and inspiration for this system, the distribution functionality has been replicated to provide controlled releases, role-based access, and the capability to roll back to previous versions, ensuring structured beta testing of mobile applications.

By using an extensive research-based approach to development, DevAppDeploy has been built by integrating cloud technologies, secure version control mechanisms, and role-based access control, in addition to offering a scalable architecture.

The results from the various system tests that have been conducted confirm that the system is effective in securely distributing the application version builds, offering structured user testing of mobile applications to evaluate different versions in parallel.

Performance assessments were undertaken and have demonstrated the platform's key indicators with usability, scalability and availability, while also demonstrating robustness with security evaluations to handling beta releases.

This project highlights the importance of internal beta testing within the mobile application development cycles for organisations such as Cardiff Council that need to comply with various compliance and security requirements. By replicating the distribution features once offered by App Center allows for the future evolution of the system that can be tailored to the needs of Cardiff Council, proving that custom built solutions can offer advantages over 'off the shelf' products or commercial tools while also removing the dependence on the corporations to support and maintain these tools. This means that reliance on them is not something that can hinder or cause disruption should they ever be removed.

One of the fundamental technical challenges for the implementation of DevAppDeploy was to ensure reliable blob storage integration. Using blob storage was not a topic that the developer had much knowledge of, nor any experience of using. Azure Blob Storage was employed to store version builds, which provided a new level of understanding in the benefits that this technology can bring to a system such as DevAppDeploy.

DevAppDeploy offers value to development teams that are currently not offered by existing tools, with its storage of previous builds allowing the roll-back of beta applications, and its parallel testing offering enhances assurance as testers can install different versions of the mobile apps.

Despite the success of the prototype, several limitations remain that could be addressed. Currently, the system is only capable of uploading and downloading Android applications as iOS compatibility has not yet been incorporated into the platform.

While DevAppDeploy benefits from CI/CD pipelines to automate the build and distribution of the system itself, mobile applications need to be manually uploaded by the users, rather than mobile application repos being connected to pipelines, allowing for the application build to be automatically inserted into DevAppDeploy.

Another area that the system would benefit from further refinement is the inclusion of monitoring and crash analytics, allowing developers and testers to evaluate the performance of mobile applications directly within the DevAppDeploy platform.

To summarise, potential future improvements to DevAppDeploy would include:

- Automate Mobile App Deployment Pipelines – Utilise CI/CD to automatically distribute builds from code repositories to DevAppDeploy.
- Analytics and Crash Reporting – A dashboard to provide testers with insights into performance and failure tracking.
- iOS compatibility – Introduce iOS application package builds to the system.
- Access Control – The system currently stores individual user accounts. In the future, this could be integrated with the organisation's Azure Entra accounts and employ multi-factor authentication.

By completing the prototype of the DevAppDeploy, the prototype serves as a foundation for Cardiff Council's mobile application testing needs and showcases its growth potential. It also has scope to be refined to align with any changes to testing strategies.

Video Demonstrations & Source Code Links

Source Code:

[RemovedLink]

Video Demonstration(showing access to DevAppDeploy from an Android device and the app file being downloaded and installed to the device):

[RemovedLink]

References

Alshamrani, A. & Bahattab, A., 2015. A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model. *International Journal of Computer Science Issues (IJCSI),* 12(1), p. 106.

Alves, L. et al., 2023. Comparison of semi-structured data on MSSQL and PostgreSQL. *In Marketing and Smart Technologies: Proceedings of ICMarkTech 2022,* Volume 2, pp. 31-43.

Amazon, 2025. *What is cloud computing?*. [Online]
Available at: https://aws.amazon.com/what-is-cloud-computing/
[Accessed 1 4 2025].

Anderson, T., 2025. *Microsoft designates Blazor as its main future investment in Web UI for .NET.* [Online]
Available at: https://devclass.com/2025/05/29/microsoft-designates-blazor-as-its-main-future-investment-in-web-ui-for-net/#:~:text=Home%20Development-,Microsoft%20designates%20Blazor%20as%20its%20main,in%20Web%20UI%20for%20.NET&text=At%20its%20Build%20developer%20event
[Accessed 30 5 2025].

Ashraf, I., 2014. An Overview of Service Models of Cloud Computing. *International Journal of Multidisciplinary and Current Research,* 2(1), pp. 779-783.

Balaji, S., Murugaiyan, S. & M, 2012. WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC. *International Journal of Information Technology and Business Management,* 2(1), pp. 26-30.

Borra, P., 2024. An Overview Of Cloud Computing And Leading Cloud Sservice Providers. *International Journal of Computer Engineering and Technology (IJCET),* 15(3), pp. 122-133.

Chande, M., 2025. High Availability Redefined: Architecting Scalable and Resilient Systems with SQL Server and PostgreSQL. *International Research Journal of Engineering and Technology (IRJET),* 12(02).

Cloudflare, 2025. *What is blob storage?*. [Online]
Available at: https://www.cloudflare.com/en-gb/learning/cloud/what-is-blob-storage/
[Accessed 25 4 2025].

Daher, Z. & Hajjdiab, H., 2018. Cloud Storage Comparative Analysis Amazon Simple Storage vs. Microsoft Azure Blob Storage. *International Journal of Machine Learning and Computing,* 8(1), pp. 85-89.

Devessence, 2025. *Blazor vs React in 2025: Which Framework is Right for Your Web App?*. [Online]
Available at: https://devessence.com/blog/!/58/blazor-vs-react-in-2025
[Accessed 28 5 2025].

EDB, 2024. *A Complete Comparison of PostgreSQL vs Microsoft SQL Server*. [Online]
Available at: https://www.enterprisedb.com/blog/microsoft-sql-server-mssql-vs-postgresql-comparison-details-what-differences
[Accessed 26 4 2025].

Fishuck, I. & Datsyshyn, V., 2025. *Blazor's Popularity Explained: Benefits and Real-World Use Cases*. [Online]
Available at: https://leobit.com/blog/blazors-popularity-explained-benefits-and-real-world-use-cases/#:~:text=Blazor%20simplifies%20development%20by%20allowing,frameworks%20like%20JavaScript%20or%20TypeScript.
[Accessed 24 4 2025].

Geeksforgeeks, 2024. *Difference between MS SQL Server and PostgreSQL*. [Online]
Available at: https://www.geeksforgeeks.org/difference-between-ms-sql-server-and-postgresql/
[Accessed 26 4 2025].

GeeksForGeeks, 2025. *Agile Development Models - Software Engineering*. [Online]
Available at: https://www.geeksforgeeks.org/software-engineering-agile-development-models/
[Accessed 23 3 2025].

GeeksForGeeks, 2025. *Class Diagram | Unified Modeling Language (UML)*. [Online]
Available at: https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/
[Accessed 20 03 2025].

GeeksForGeeks, 2025. *Unit Testing - Software Testing*. [Online]
Available at: https://www.geeksforgeeks.org/unit-testing-software-testing/
[Accessed 20 5 2025].

Google Cloud, 2025. *PostgreSQL vs SQL Server: What are the key differences?*. [Online]
Available at: https://cloud.google.com/learn/postgresql-vs-sql
[Accessed 25 4 2025].

Grace, D., 2021. *How Blazor Performs Against Other Frameworks*. [Online]
Available at: https://www.telerik.com/blogs/how-blazor-performs-against-other-frameworks
[Accessed 25 4 2024].

Gurung, G., Shah, R., Jaiswal, P. & Dhiraj, 2020. Software Development Life Cycle Models-A Comparative Study. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology,* 6(4), pp. 30-37.

Ikbal Hossain, M., 2023. Software Development Life Cycle (SDLC). *International Journal for Multidisciplinary Research (IJFMR),* Issue 5, pp. 1-36.

IONOS, 2022. *Web frameworks – overview and classification*. [Online]
Available at: https://www.ionos.co.uk/digitalguide/websites/web-development/web-frameworks-an-overview/
[Accessed 24 4 2025].

Jin, 2023. *Software Development Framework — Iterative Model*. [Online]
Available at: https://medium.com/geekculture/software-development-framework-iterative-model-68584bfad773
[Accessed 23 3 2025].

Köping, O. & Persson, E., 2021. *Evaluation of the Blazor framework: A comparison between Blazor and React,* Jönköping: School of Engineering at Jönköping.

Medavarapu, S. v., 2021. Server-Side Rendering vs. Client-Side Rendering in Blazor. *Journal of Scientific and Engineering Research,* 8(12), pp. 311-317.

Metha, T., 2025. *Blazor vs React in 2025: Pros, Cons, and Use Cases*. [Online]
Available at: https://wirefuture.com/post/blazor-vs-react-in-2025-pros-cons-and-use-cases
[Accessed 3 6 2025].

Microsoft, 2025. *Overview of ASP.NET Core*. [Online]
Available at: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-9.0
[Accessed 24 4 2025].

Microsoft, 2025. *SQL Server 2022 pricing and licensing*. [Online]
Available at: https://www.microsoft.com/en-us/sql-server/sql-server-2022-pricing
[Accessed 3 6 2025].

Microsoft, 2025. *Visual Studio App Center documentation*. [Online]
Available at: https://learn.microsoft.com/en-gb/appcenter/
[Accessed 09 04 2025].

Microsoft, 2025. *Visual Studio App Center Retirement*. [Online]
Available at: https://learn.microsoft.com/en-gb/appcenter/retirement
[Accessed 11 03 2025].

Microsoft, 2025. *What is cloud computing?*. [Online]
Available at: https://azure.microsoft.com/en-us/resources/cloud-computing-

dictionary/what-is-cloud-computing
[Accessed 1 4 2025].

Nile Bits, 2023. *ASP.NET Core vs. ASP.NET Framework: Which is Better for Your Project?*. [Online]
Available at: https://medium.com/@nile.bits/asp-net-core-vs-asp-net-framework-which-is-better-for-your-project-154a2d098211
[Accessed 24 4 2025].

Noah, G., 2024. Comparing Cloud Storage Solutions for Big Data: Amazon S3 vs. Google Cloud Storage vs. Azure Blob Storage.

PostgreSQL, 2025. *License*. [Online]
Available at: https://www.postgresql.org/about/licence/
[Accessed 25 4 2025].

Prioxis, 2025. *Blazor vs React: Key Differences*. [Online]
Available at: https://www.prioxis.com/blog/blazor-vs-react
[Accessed 15 5 2025].

Ramel, D., 2024. *Workarounds Listed for Upcoming Visual Studio App Center Retirement*. [Online]
Available at: https://visualstudiomagazine.com/Articles/2024/03/18/app-center-retirement.aspx
[Accessed 09 04 2025].

Rathinam, S., 2022. Analysis and Comparison of Different Frontend Frameworks. *International Conference on Applications and Techniques in Information Security,* pp. 243-257.

Saeed, S., Jhanjhi, N., Navqi, M. & Humayun, M., 2019. Analysis of Software Development Methodologies. *International Journal of Computing and Digital Systems ,* 8(5), pp. 446-460.

Sears, R., Van Ingen, C. & Gray, J., 2006. *To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem?,* California: University of California at Berkeley.

Sprinkle, 2024. *PostgreSQL vs. SQL Server: A Comprehensive Comparison of Database Systems*. [Online]
Available at: https://www.sprinkledata.com/blogs/postgresql-vs-sql-server-a-comprehensive-comparison-of-database-systems
[Accessed 26 4 2025].

Vershinin, I. & Mustafina, A., 2021. Performance Analysis of PostgreSQL, MySQL, Microsoft SQL Server Systems Based on TPC-H Tests. *2021 International Russian Automation Conference (RusAutoCon) ,* pp. 683-687.

Visual Paradigm, 2025. *What is Sequence Diagram?*. [Online]
Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/
[Accessed 20 03 2025].

Younis, R. et al., 2024. A Comprehensive Analysis of Cloud Service Models: IaaS, PaaS, and SaaS in the Context of Emerging Technologies and Trend. *International Conference on Electrical, Communication and Computer Engineering (ICECCE),* pp. 1-6.