Atlantic Technological University

SecFlo

A Secure Workflow Application

with Role-Based Access Control

Michael McKibbin

Contemporary Software Development 2025/2026

Atlantic Technological University

Computing Department, ATU Donegal, Port Road, Letterkenny, Co. Donegal, Ireland.

SecFlo

Design, Implementation, and Cloud Deployment of a Containerised Secure Workflow Application with Role-Based Access Control

SecFlo.net

or

workflow.michaelmckibbin.cloud.

GitHub repository

https://github.com/MichaelMcKibbin/secure-workflow-system

Author: Michael McKibbin

Supervised by: Dr. Lusungu Mwasinga

Declaration

I hereby certify that the material, which I now submit for assessment on the programmes of study leading to the award of Bachelor of Science (Hons) in Contemporary Software Development, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution. I understand that it is my responsibility to ensure that I have adhered to ATU's rules and regulations.

I hereby certify that the material on which I have relied on for the purpose of my assessment is not deemed as personal data under the GDPR Regulations. Personal data is any data from living people that can be identified. Any personal data used for the purpose of my assessment has been pseudonymised and the data set and identifiers are not held by ATU. Alternatively, personal data has been anonymised in line with the Data Protection Commissioners Guidelines on Anonymisation.

I consent that my work will be held for the purposes of education assistance to future students and will be shared on the ATU Computing website (http://www.atucomputingdonegal.com/) and Research THEA website (https://research.thea.ie/). I understand that documents once uploaded onto the website can be viewed throughout the world and not just in Ireland. Consent can be withdrawn for the publishing of material online by emailing Jade Lyons; Head of Department at jade.lyons@atu.ie to remove items from the ATU Computing website and by email emailing Denise McCaul; Systems Librarian at denise.mccaul@atu.ie to remove items from the Research THEA website. Material will continue to appear in printed formats once published and as websites are public medium, ATU cannot guarantee that the material has not been saved or downloaded.

Signature of Candidate: Michael McKibbin Date: 20/05/2026

Acknowledgements

I would like to thank my project supervisor, Dr. Mwasinga, for his guidance, feedback, and support throughout the development of this project.

I would also like to thank the lecturers, staff, and students of Atlantic Technological University for their support during my studies.

I would like to acknowledge the use of modern development and collaboration tools, including GitHub, Docker, and Microsoft development technologies, which supported the implementation and deployment of this project.

Finally, I would like to thank my family for their patience and encouragement throughout the completion of this project.

Abstract

This project describes the design, implementation, testing, and cloud deployment of SecFlo, a secure workflow management system developed using modern software engineering and DevOps practices. The system was designed to provide a structured workflow platform supporting secure user authentication, role-based access control, workflow case management, and containerised cloud deployment.

The application was developed using ASP.NET Core Blazor with PostgreSQL as the backend database platform. User authentication and authorization were implemented using ASP.NET Core Identity with role-based access control policies to support standard users, staff users, and administrative users. Workflow functionality was implemented using controlled workflow states and status history tracking to support structured case progression and accountability.

The project also focused heavily on deployment automation and cloud-hosted infrastructure. Docker was used to containerise the application and supporting services, while GitHub Actions provided automated CI/CD workflows for building, testing, publishing, and deploying the system to a live Ubuntu VPS environment. HTTPS communication and reverse proxy routing were implemented using Traefik with automatic TLS certificate management.

Testing was performed across multiple layers of the system, including unit testing, component testing, integration testing, workflow validation, security testing, and deployment verification. Several deployment and operational issues were identified and resolved during development, providing practical experience in secure cloud application deployment and DevOps-oriented software engineering practices.

The completed project demonstrates the successful implementation of a secure cloud-hosted workflow platform while also highlighting the importance of testing, deployment automation, authentication security, and operational reliability within modern software systems.

Acronyms

Acronym	Meaning
API	Application Programming Interface
ASP.NET	Active Server Pages .NET
BUnit	Blazor Unit Testing Framework
CI/CD	Continuous Integration / Continuous Deployment
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DevOps	Development and Operations
EF Core	Entity Framework Core
GHCR	GitHub Container Registry
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTPS	HyperText Transfer Protocol Secure
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
JWT	JSON Web Token
ORM	Object-Relational Mapping
PR	Pull Request
RBAC	Role-Based Access Control
SaaS	Software as a Service
SHA	Secure Hash Algorithm
SQL	Structured Query Language
TLS	Transport Layer Security
UI	User Interface
VPS	Virtual Private Server
XML	eXtensible Markup Language
xUnit	.NET Unit Testing Framework

Table of Contents

Declaration 3

Acknowledgements 4

Abstract 5

Acronyms 6

Chapter 1 - Introduction 14

1. Introduction 15

1.1 Background Research 16

1.2 Security Considerations in Workflow Systems 17

1.3 Existing Enterprise Workflow Management Tools 18

1.4 Task Workflow Tools 18

1.5 How this project compares 20

1.6 Workflow Modelling Approaches 20

1.7 Comparison of Petri Nets, Finite State Machines (FSM), and Business Process Model and Notation (BPMN) 21

1.8 Purpose / Rationale 23

1.9 Aim 23

1.10 Objectives 24

1.11 System Overview 24

1.12 Report Structure 25

Chapter 2: Design 27

2. Design 28

2.1 System Overview 28

2.2 Project Requirements 29

2.2.1 Functional Requirements 29

2.2.2 Non-Functional Requirements 30

2.2.3 Potential Future Enhancements 31
2.3 System Design Overview 32
2.3.1 Use-case diagram 32
2.3.2 Simplified Use Case Diagram 33
2.3.3 Architecture diagram 34
2.4 Technology Selection and Justification 35
2.4.1 Blazor Server (.NET 10 / C#) 35
2.4.2 ASP.NET Core Identity (Authentication & RBAC) 35
2.4.3 PostgreSQL (Database) 36
2.4.4 Entity Framework Core (ORM) 36
2.4.5 Docker & Docker Compose (Containerisation) 37
2.4.6 Traefik (Reverse Proxy & HTTPS) 37
2.4.7 Ubuntu VPS (Cloud Hosting Environment) 37
2.4.8 GitHub & GitHub Actions (Version Control & CI) 38
2.4.9 Summary 38
2.5 Database schema design 39
2.6 Security Design 40
2.6.1 Role-Based Access Control (RBAC) Decision Flow Diagram 41
2.7 Workflow Design 42
2.7.1 Workflow State Diagram 43
2.8 Deployment / DevOps Design 44
Chapter 3 : Implementation 46
3. Implementation 47
3.1 Development Environment and Setup 47
3.2 Database and Data Layer Implementation 48

3.3 Authentication and Authorization 53

3.4 Case Workflow Implementation 58

3.5 Workflow State and Audit Management 64

3.6 DevOps and Deployment Implementation 66

Chapter 4: Testing 76

4. Testing 77

4.1 Testing Strategy and Development Approach 77

The Testing Pipeline 77

4.2 Testing Objectives 77

4.2.1 Functional Requirements Validation 78

The following sections describe the specific testing methods and results used to validate each of these objectives. 78

4.3 Testing Environment 79

4.4 Unit and Component Testing 80

4.5 Integration Testing 82

4.6 Functional and Workflow Testing 84

4.7 Security and RBAC Testing 88

4.8 CI/CD and Deployment Testing 90

4.9 Negative Testing / Error Testing 93

4.10 Troubleshooting and Defect Resolution 94

4.11 Evaluation Against Objectives 95

4.12 Test Results Summary 96

4.13 Limitations 96

4.14 Reflection on Testing Outcomes 97

4.15 Summary 97

Chapter 5 – Conclusions 98

5. Conclusions and Future Work 99
5.1 Project Summary 99
5.2 Achievement of Objectives 99
5.3 Challenges and Lessons Learned 100
5.4 Limitations 101
5.5 Future Work 102
5.6 Final Reflection 103
References 104
Glossary 106

Table of Figures

Figure 1: Conceptual Comparison Diagram - Petri Nets, FSM, & BPMN 22
Figure 2: Use Case Diagram 32
Figure 3: Simplified Use Case Diagram 33
Figure 4: RBAC Permissions Diagram 40
Figure 5: RBAC Permissions Matrix 40
Figure 6: Role-Based Access Control (RBAC) Decision Flow Diagram 41
Figure 7: Workflow Diagram 42
Figure 8: Workflow State Diagram 43
Figure 9: DevOps Deployment Process 44
Figure 10: Sample case data showing user ownership, assignment, and workflow status stored in the PostgreSQL database 49
Figure 11: Sample case data showing user ownership, assignment, and workflow status 50
Figure 12: Case–User relationship configuration - Restrict delete behaviour 50
Figure 13: CaseStatusHistories table 51

Figure 14: Sequential workflow state transitions for a selected case 52

Figure 15: Server-side authorization enforcement for Admin Role 54

Figure 16: Authorisation based on role membership 54

Figure 17: Staff level user attempting to access Admin level "Admin Users" page 55

Figure 18: The Role Management page 56

Figure 19: Adding a new case using the Create Case page 58

Figure 20: Case listing interface showing role-based filtering and workflow status - list view 59

Figure 21: Case listing interface showing role-based filtering and workflow status – Card view 59

Figure 22: My Cases view for a logged in User 60

Figure 23: Unauthorised User View 60

Figure 24: Case details view with access control enforced based on user permissions and with status update history functionality 61

Figure 25: New Case awaiting assignment 62

Figure 26: Case assigned to a user with status change success message 62

Figure 27: Case Details with Status History 63

Figure 28: Case status update functionality demonstrating workflow progression over time 64

Figure 29: Docker Desktop containers running the application, database, and administration services locally for development. 67

Figure 30: Multi-container configuration using Docker Compose 68

Figure 31: Deployment architecture diagram 70

Figure 32: Live cloud deployment secured using HTTPS and Traefik reverse proxy routing. 70

Figure 33: GitHub Actions snippet from deploy.yml 71

Figure 34: Docker container images published to GitHub Container Registry (GHCR) during the CI/CD deployment process 72

Figure 35:GitHub Actions Successful Deployment 73

Figure 36: VPS Docker PS Output showing all current containers 73

Figure 37: Verifying Container Operational Status (note the image digest identifier) 74

Figure 38: Using Docker Manager to view running containers 74
Figure 39: The Testing Pipeline 77
Figure 40: Testing Frameworks and Tools Used 80
Figure 41: Docker test containers created during integration tests 83
Figure 42: The lifecycle of a case 84
Figure 43: The Case Creation Form 85
Figure 44: The All Cases List and Workflow Status Display 85
Figure 45: Case Details View 86
Figure 46: Case Status History View 86
Figure 47: HTTPS browser security 89
Figure 48: Role Management View 89
Figure 50: GitHub Actions Successful Workflow 91
Figure 51: Docker container images published to GitHub Container Registry (GHCR) 92
Figure 52: Docker ps Output on VPS using Terminal 92
Figure 53: Health check endpoint 93
Figure 54: Invalid Login Attempt Response 94
Figure 55: Unauthorized Access Denied 94
Figure 56: Successful Redeployment After Fix 95

Table of Tables

Table 1: Workflow complexity comparison 19
Table 2: Comparison of Petri Nets, FSMs, and BPMNs 22
Table 3: Current Authentication Implementation vs Potential Enhancements 57
Table 4: : Using the docker compose command 74
Table 5: Implemented Features and Technologies 75
Table 6 Bunit test features: 82
Table 7: Workflow tests 84
Table 8: Workflow Validation Testing Octcomes 87
Table 9: Manual/browser testing of the deployed app 87
Table 10: User Permissions 90
Table 11: Container and Endpoint Checks 91
Table 12: Negative & Error Testing Results 93
Table 13: : Troubleshooting and Defect Resolution Summary 95
Table 14: Evaluation vs Objectives 96
Table 15: Test Results 96

Chapter 1- Introduction

1. Introduction

Modern organisations increasingly rely on digital systems to manage internal processes, coordinate tasks, and track the progression of work. These processes, commonly referred to as workflows, often involve multiple stages, users, and decision points. As organisations grow in complexity, the need for structured workflow management systems becomes more significant.

Traditional workflow and case management systems are frequently implemented as large enterprise platforms. While these systems provide powerful automation and orchestration capabilities, they can be complex to configure, resource-intensive to deploy, and unsuitable for smaller-scale environments or educational projects. At the same time, simpler task management tools often lack the security controls required to manage sensitive data or enforce structured process rules.

A key challenge in workflow system design is balancing functionality, security, and deployment simplicity. In particular, ensuring that users can only access appropriate data and perform authorised actions is essential in systems that handle organisational processes. Role-Based Access Control (RBAC) is commonly used to address this challenge by restricting system functionality based on user roles.

This project addresses these challenges through the design and implementation of a secure workflow management system, referred to as *SecFlo*. The system focuses on controlled workflow progression, secure authentication, and role-based access control, while remaining lightweight and suitable for containerised cloud deployment.

The project also explores modern software development and deployment practices, including containerisation, continuous integration, and cloud hosting. By combining these approaches, the project demonstrates how secure workflow systems can be developed using contemporary technologies while maintaining clarity, maintainability, and deployment efficiency.

1.1 Background Research

Workflow is a generic term for orchestrated and repeatable patterns of activity, enabled by the systematic organization of resources into processes that transform materials, provide services, or process information (Workflow Management Coalition, 1996).

Workflow systems are software applications designed to manage, coordinate, and track the progress of tasks through a defined sequence of stages. They provide structured mechanisms for assigning work, monitoring progress, enforcing rules, and ensuring that tasks are completed in the correct order. These systems help organisations standardise processes, improve efficiency, and maintain accountability across teams (Van der Aalst, 2016).

Workflow systems are commonly used in environments where tasks pass through multiple stages or require input from different roles. Examples include customer service case management, helpdesk ticket systems, document approval workflows, healthcare patient management, and legal or administrative case tracking. In these contexts, workflow systems allow organisations to track requests from submission through resolution while maintaining a record of actions taken and decisions made (Weske, 2024).

A **Workflow Management System (WfMS)** provides the infrastructure required to define, execute, and monitor workflows. According to the Workflow Management Coalition definition, a workflow management system supports the setup, performance, and monitoring of tasks organised as workflow applications (Workflow Management Coalition, 1996). These systems coordinate the interaction between users, tasks, and supporting software services.

The theoretical foundation of many workflow systems can be traced to **Petri nets**, a mathematical modelling technique used to represent distributed systems and concurrent processes. Petri nets allow workflows to be represented as networks of states and transitions, enabling the modelling of task dependencies, synchronisation points, and parallel activities within a workflow process (Peterson, 1981).

A workflow application is therefore a software application that automates, to some degree, a defined process or series of tasks. These processes are typically business-related but can apply to any structured activity involving multiple steps. While some steps may be automated entirely by the system, others may require human intervention, such as approvals, data entry, or decision-making.

Advanced workflow applications also allow users to modify or extend processes by introducing new tasks or conditions within the workflow.

Workflow management plays an important role in modern organisations by helping to standardise and optimise business processes. Many organisations operate with complex internal procedures that involve multiple teams and approval stages. Without structured workflow management, tasks may become delayed, duplicated, or overlooked. By clearly defining task ownership and sequencing, workflow systems can improve productivity, reduce errors, and provide transparency regarding the current status of ongoing work.

1.2 Security Considerations in Workflow Systems

As organisations increasingly rely on digital workflow systems to manage operational processes, security becomes an essential consideration. Workflow and case management systems often store sensitive information, including personal data, internal communications, organisational decisions, or operational records. Without appropriate access controls, unauthorised users may gain access to confidential information or perform actions that fall outside their responsibilities (Sandhu, 1996).

One widely adopted mechanism for controlling access in such systems is **Role-Based Access Control (RBAC)**. RBAC restricts system actions based on the role assigned to each user within the organisation. For example, administrators may have permission to configure workflows, while staff members may only create or update tasks within their assigned scope. Proper enforcement of RBAC helps ensure confidentiality, integrity, and accountability within workflow systems (Sandhu, 1996).

Modern workflow systems are also increasingly deployed using **cloud infrastructure and containerised environments**, which introduce additional considerations such as secure network configuration, authentication mechanisms, and protection of exposed services.

1.3 Existing Enterprise Workflow Management Tools

A number of enterprise level workflow management platforms and tools already exist that provide automation and orchestration capabilities for organisational processes. These systems vary widely in complexity, intended use, and deployment requirements.

Camunda is an open-source workflow automation platform that supports Business Process Model and Notation (BPMN) for modelling and executing business processes. It is widely used in enterprise environments to orchestrate complex workflows that integrate multiple systems and services. While powerful, Camunda often requires significant configuration and infrastructure, which may make it less suitable for lightweight deployments or smaller projects.

Temporal is a modern workflow orchestration platform designed for managing long-running and distributed business processes. It provides strong reliability guarantees and fault tolerance for complex microservice architectures. Temporal is particularly suited to large-scale distributed systems but introduces additional architectural complexity that may not be necessary for simpler workflow applications.

Jira Service Management provides workflow capabilities as part of a broader issue tracking and service management platform. It is widely used in software development and IT service management environments to track issues, service requests, and project tasks. While Jira provides configurable workflows, it is primarily designed as a commercial enterprise platform rather than a lightweight standalone workflow system.

Apache Airflow is a workflow orchestration tool primarily designed for scheduling and managing data pipelines. Airflow workflows are defined as directed acyclic graphs (DAGs) that describe task dependencies. Although powerful for data processing workflows, Airflow is typically used in data engineering contexts rather than general business process management.

1.4 Task Workflow Tools

In addition to enterprise workflow engines, a number of work management platforms provide simplified workflow capabilities for teams. Tools such as Trello, Asana, Monday.com, Smartsheet, and

Airtable allow users to organise tasks, track progress, and manage collaborative workflows through visual interfaces such as boards, lists, or tables.

For example, Trello uses a Kanban-style board where tasks move between columns representing stages of a workflow. Asana and Monday.com provide configurable project management environments that allow teams to track tasks, deadlines, and dependencies. Airtable combines spreadsheet-style data management with workflow automation features, enabling users to build lightweight workflow applications without traditional software development.

While these platforms provide useful workflow management features, they are typically designed for project coordination and team collaboration rather than secure workflow execution or process enforcement. They often lack fine-grained access control models or tightly controlled workflow state transitions. As a result, they are best suited for organisational task management rather than the implementation of secure workflow systems.

The table below shows a simplified comparison of the complexity levels of enterprise workflow management systems, task workflow tools, and this project.

System	Primary Use	Workflow Complexity
Camunda	Business process automation	High
Temporal	Distributed workflow orchestration	High
Apache Airflow	Data pipeline workflows	High
Jira	Issue tracking and service workflows	Medium
Trello / Asana / Monday	Task and project workflows	Low–Medium
This Project	Secure lightweight workflow system	Medium

Table 1: Workflow complexity comparison

1.5 How this project compares

The system developed in this project differs from these platforms in its scope and objectives. Rather than attempting to replicate the full feature set of enterprise workflow engines, the project focuses on designing and implementing a **lightweight, secure workflow system** that demonstrates key architectural concepts such as role-based access control, workflow state transitions, containerised deployment, and secure cloud hosting. The project therefore emphasises clarity of design and security considerations rather than enterprise-scale functionality.

1.6 Workflow Modelling Approaches

Workflow systems rely on formal or semi-formal modelling approaches to represent processes and task flows. These models define how tasks move between different stages, how decisions are made, and how different actors interact within the workflow. Several modelling approaches are commonly used in workflow management systems, including Petri nets, state machines, and Business Process Model and Notation (BPMN).

One of the earliest theoretical models used to represent workflows is the Petri net. Petri nets provide a mathematical framework for modelling concurrent and distributed systems. In a Petri net, processes are represented using places, transitions, and tokens that move through the network to represent changes in system state. This model is particularly useful for analysing parallel activities, synchronisation points, and resource constraints within complex workflows (Peterson, 1981). Many early workflow systems were built upon Petri net concepts due to their strong theoretical foundation and ability to formally verify system behaviour.

Another common modelling approach is the finite state machine (FSM) model. In a state machine representation, a workflow is described as a set of states connected by transitions. Each transition represents a change in the status of a workflow item, often triggered by a user action or system event. State machines are frequently used in software systems where workflows follow a clear lifecycle, such as issue tracking systems, approval processes, or ticket management platforms. The workflow system developed in this project uses a simplified state-transition model in which tasks move through defined stages such as creation, review, and completion.

A widely adopted industry standard for modelling business processes is Business Process Model and Notation (BPMN). BPMN provides a graphical notation that allows workflow processes to be represented using diagrams that are understandable by both technical and non-technical stakeholders. BPMN models include elements such as tasks, gateways, events, and process flows, enabling organisations to visualise and analyse business processes before implementing them in software systems (Weske, 2024).

Modern workflow platforms may support one or more of these modelling approaches depending on their intended use. Enterprise workflow engines such as Camunda and Activiti frequently rely on BPMN-based models, while lightweight workflow systems often use simplified state transition models implemented directly in application logic.

For the purposes of this project, a state-transition workflow model was selected due to its simplicity and suitability for the type of case management workflow being implemented. This approach provides clear lifecycle stages for tasks while remaining straightforward to implement within a web application architecture.

1.7 Comparison of Petri Nets, Finite State Machines (FSM), and Business Process Model and Notation (BPMN)

This analysis compares Petri Nets, State Machines (Finite State Machines - FSM), and Business Process Model and Notation (BPMN), highlighting their purposes, strengths, and structural differences. (ResearchGate, 2025), (Van Den Berg & Weber, 2019), (Mutarraf, et al., 2018)

The table below shows a comparison of Petri Nets, FSMs, and BPMNs

Feature	**Petri Nets**	**State Machines (FSM)**	**BPMN 2.0**
Primary Purpose	Formal analysis, verification, simulation	Modelling reactive, single-threaded system behaviour	Business process documentation, automation
Core Elements	Places (circles), Transitions (rectangles), Tokens	States (rectangles/rounded), Transitions (arrows)	Tasks, Gateways, Events, Sequence Flows

Concurrency	Excellent (natural handling of parallel processes)	Poor (limited to orthogonal regions/concurrent state machines)	Excellent (parallel gateways)
Data Representation	Explicit via Coloured Petri Nets (CPN)	Implicit (based on current state)	Data objects, data stores
Complexity	High (too academic for business stakeholders)	Low/Moderate (conceptually simple)	High (easy to read, hard to master)
Execution	Formal Simulation	Directly executable (code generation)	Executable (via Workflow Engine)
Common Use Cases	Industrial plants, concurrency analysis	UI workflows, protocol design, embedded systems	Business Process Management (BPM), workflows

Table 2: Comparison of Petri Nets, FSMs, and BPMNs

The following diagram illustrates the relationship between Petri Nets, Finite State Machines (FSM), and Business Process Model and Notation (BPMN).

Petri Nets, FSM, and BPMN Relationships

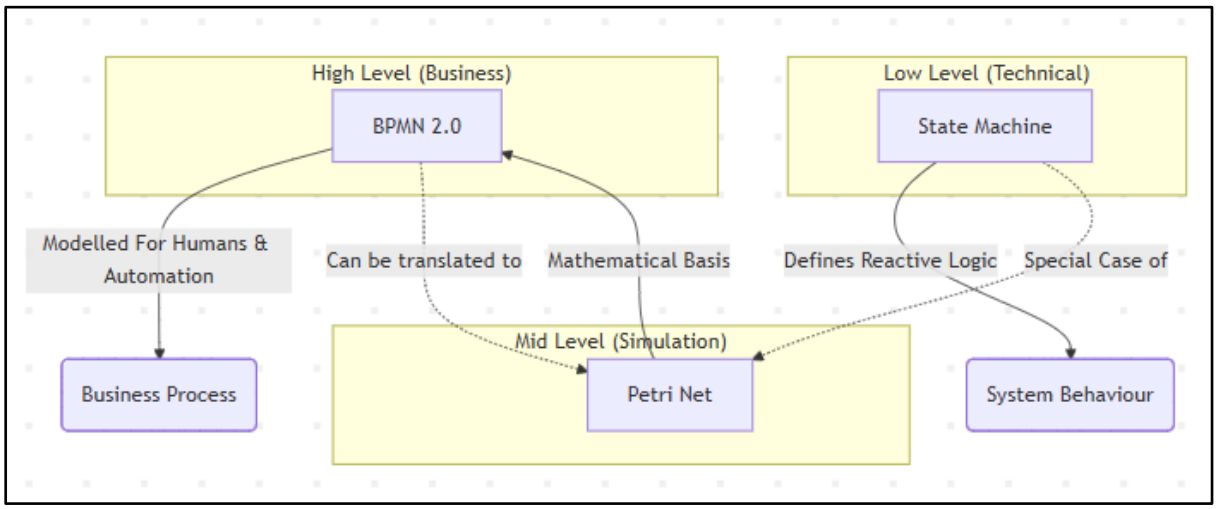


Figure 1: Conceptual Comparison Diagram - Petri Nets, FSM, & BPMN

1.8 Purpose / Rationale

Many existing workflow and case management systems are designed as large enterprise platforms that require significant infrastructure, configuration, and operational expertise. While these systems provide extensive functionality, they are often complex to deploy, resource-intensive, or unsuitable for smaller organisations, development environments, or educational contexts (Weske, 2024).

At the same time, lightweight workflow solutions or prototype systems frequently lack robust security controls. In particular, insufficient enforcement of authentication and role-based permissions can allow users to access or modify workflow data beyond their intended authority. This creates potential risks for organisations relying on such systems to manage operational processes or sensitive information (Sandhu, 1996).

In addition, the increasing adoption of cloud-based infrastructure and containerised deployment platforms has created opportunities for developing workflow systems that are easier to deploy, maintain, and scale than traditional enterprise solutions.

As organisations increasingly adopt digital systems to manage internal processes, the need for secure and well-structured workflow management solutions continues to grow. Effective workflow systems must not only coordinate tasks and responsibilities but also enforce security policies, maintain accountability, and protect sensitive organisational data (Van der Aalst, 2016).

In response to these challenges, this project investigates the design and implementation of a secure workflow management system that balances functionality, security, and deployment simplicity. The project focuses on demonstrating core workflow concepts while applying modern software engineering practices, including structured workflow state management, role-based access control, and containerised deployment. By developing and deploying the system within a cloud-hosted environment, the project aims to illustrate how secure workflow applications can be implemented using contemporary technologies while remaining lightweight and accessible for smaller organisations and development contexts.

1.9 Aim

The aim of this project is to design, implement, and deploy a secure workflow management system that can be hosted in a cloud environment. The system will demonstrate key architectural principles

including role-based access control, secure authentication, and controlled workflow state transitions. In addition, the project will explore modern software engineering and deployment practices such as containerised infrastructure, automated build pipelines, and cloud-based hosting.

1.10 Objectives

To achieve the project aim, the following objectives were defined:

- Implement secure authentication and session management using ASP.NET Core Identity.
- Enforce Role-Based Access Control (RBAC) restricting system functionality by user roles.
- Design and implement workflow state transitions to manage task lifecycles within the system.
- Develop functionality for creating, assigning, and managing workflow tasks within the system.
- Store workflow data securely using a relational database system.
- Develop the application using a modern web framework and structured architectural design.
- Deploy the system using containerised infrastructure.
- Implement a Continuous Integration (CI) pipeline to automatically build and deploy the application.
- Develop unit tests to verify key components of the system and improve software reliability.

In addition to the core objectives listed above, some optional enhancements may be explored if time permits. These may include additional workflow visualisation features, extended reporting capabilities, or further security enhancements.

1.11 System Overview

The artefact developed in this project is a secure workflow management web application designed to demonstrate controlled task progression, role-based access control, and secure cloud deployment. The system is implemented as a Blazor Web App (Server) using the ASP.NET Core framework, which provides integrated support for authentication, authorisation, and modern web application development. User accounts and roles are managed using ASP.NET Core Identity, enabling the enforcement of Role-Based Access Control policies that restrict access to workflow operations based on user roles.

Persistent data storage is provided by a PostgreSQL relational database, accessed through Entity Framework Core using the Npgsql provider. The application and database services are containerised using Docker, ensuring consistent deployment across development and production environments. The system is deployed on a cloud-hosted Ubuntu Virtual Private Server (VPS), where containers are orchestrated using Docker Compose. A Traefik reverse proxy is used to manage HTTP and HTTPS routing, automatically provision TLS certificates via Let's Encrypt, and securely expose the application over the internet.

Together, these components provide a secure and modern infrastructure for implementing and deploying a lightweight workflow management system and demonstrate key principles of secure system design, modern deployment practices, and structured workflow management.

A Continuous Integration pipeline is also implemented to automatically build the application and publish container images, supporting reliable and repeatable deployment.

1.12 Report Structure

Chapter 1 presents a review of workflow management systems and related technologies. It discusses the theoretical foundations of workflow systems, including workflow modelling concepts and security considerations such as role-based access control. The chapter also examines existing workflow and work management tools and compares them to the system developed in this project.

Chapter 2 describes the system design. It outlines the architectural decisions made during development, including the application structure, data model, workflow state design, security mechanisms, and deployment architecture used to support the Secure Workflow System.

Chapter 3 presents the implementation of the system. This chapter explains how the workflow application was developed using the Blazor framework and how supporting technologies such as PostgreSQL, Docker, GitHub Actions, and container-based cloud deployment were integrated.

Chapter 4 describes the testing and evaluation of the system. It discusses the testing strategies used throughout development, including unit testing, integration testing, role-based access control testing, workflow validation, security testing, and deployment verification. The chapter also evaluates how effectively the system meets the objectives defined for the project.

Finally, Chapter 5 summarises the outcomes of the project and reflects on the development process. It also discusses limitations, lessons learned, and potential future improvements that could further extend the capabilities of the system.

Chapter 2: Design

2. Design

This chapter presents the design of SecFlo, a Secure Workflow System, including the system architecture, database structure, workflow model, security mechanisms, and deployment strategy that formed the foundation for implementation.

2.1 System Overview

The Secure Workflow System, *SecFlo*, developed in this project is a web-based application designed to manage structured task progression through a defined workflow lifecycle. The system enables users to create, track, and manage workflow cases while enforcing role-based access control to ensure that actions are restricted according to user responsibilities.

At a high level, the system follows a client-server architecture. Users interact with the application through a web browser, where requests are processed by a server-side application responsible for business logic, authentication, and workflow management. The system supports multiple user roles, including standard users, staff, and administrators, each with distinct permissions that govern access to system functionality.

The core functionality of the system centres on workflow case management. Users can create cases, which then progress through a series of defined states such as creation, assignment, processing, and completion. These transitions are controlled by the system to ensure that workflow progression follows a structured and predictable lifecycle. Staff users are responsible for managing and progressing cases, while administrators oversee user management and system-level operations.

From a data perspective, the system uses a relational database to store user information, workflow cases, status history, and associated notes. This supports data consistency, traceability, and auditability of workflow actions.

The system is designed with security as a primary concern. Authentication mechanisms ensure that only authorised users can access the system, while role-based access control enforces restrictions on available actions. This helps maintain confidentiality, integrity, and accountability within the workflow process.

In addition to functional requirements, the system is designed to be deployable in a modern cloud environment. The application and database are containerised, allowing for consistent deployment across environments and supporting scalable, maintainable infrastructure. A reverse proxy is used to manage secure external access, including HTTPS communication.

This high-level design provides the foundation for the detailed system architecture, technology selection, and component design described in the following sections.

2.2 Project Requirements

The project requirements define the expected functionality and constraints of the Secure Workflow System. These requirements are divided into functional requirements, which describe what the system must do, and non-functional requirements, which describe how the system should perform and operate.

2.2.1 Functional Requirements

Core Functionality

The system must provide the following core functionality:

- Users must be able to log in, and log out securely.
- The system must support role-based access control with at least three roles:
    - User
    - Staff
    - Admin
- Users must be able to create workflow cases.
- Users must be able to view and manage their own cases.
- Staff users must be able to view all cases in the system.
- Staff users must be able to assign cases to users or staff members.
- Staff users must be able to update the status of a case. e.g.
    - New
    - Assigned
    - In Progress
    - Resolved

    - Closed
- Staff users must be able to add notes to cases.
- Admin users must be able to manage user accounts.
- Admin users must be able to manage roles and permissions.
- Admin users must be able to view system-level information or audit data.
- The system must enforce valid workflow state transitions.

2.2.2 Non-Functional Requirements

The system must meet the following quality and operational requirements:

Security

- The system must enforce authentication and role-based authorisation.
- Users must only access data and functionality appropriate to their role.
- Sensitive data must be handled securely.

Performance

- The system should respond to user actions with minimal delay.
- The application should support multiple concurrent users.

Scalability

- The system should be deployable in a containerised environment.
- The architecture should allow for future scaling if required.

Reliability

- The system should maintain data integrity across operations.
- Database transactions should be handled safely.

Usability

- The user interface should be simple and intuitive.
- Navigation between workflow stages should be clear.

Maintainability

- The system should follow a structured architecture.
- Code should be modular and testable.

Deployment

- The system must be deployable to a cloud-hosted environment (VPS).
- The system must support containerised deployment using Docker.
- A CI pipeline should automate build and deployment processes.

These requirements directly support the project aim and objectives outlined in Chapter 1 and provide a foundation for system design and implementation aligned with established software quality models, including ISO/IEC 25010, which defines key attributes such as performance, reliability, and security. (Sommerville, 2016), (ISO.org, 2023)

2.2.3 Potential Future Enhancements

- Audit logging of workflow actions
- Notification system for case updates
- Advanced role permissions or policy-based access control
- Dashboard or reporting features
- External Registration Authentication Service

2.3 System Design Overview

The following diagrams were developed to represent the functional, structural, and behavioural design of the Secure Workflow System. Together, they provide a visual overview of user interactions, system architecture, data relationships, role-based permissions, and workflow progression.

2.3.1 Use-case diagram

The use case diagram illustrates the main interactions between the different user roles and the Secure Workflow System.

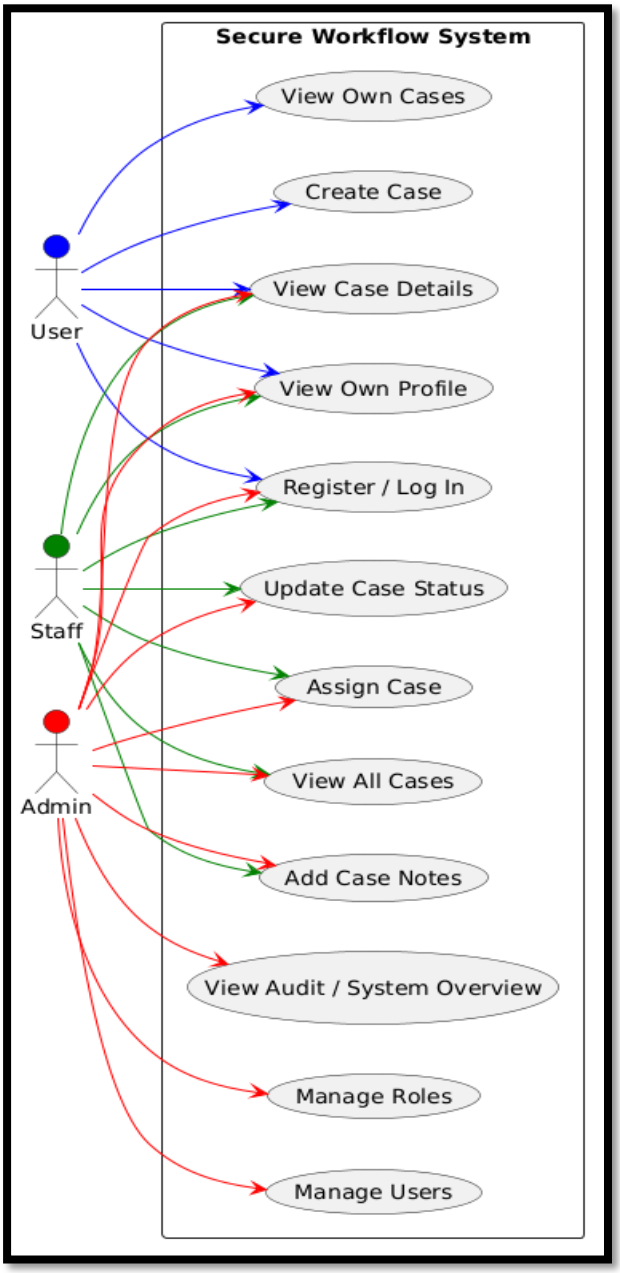


Figure 2: Use Case Diagram

Three primary actors are identified: User, Staff, and Admin.

Standard users can register, log in, create cases, and view the cases they have submitted.

Staff users have additional permissions that allow them to view all cases, assign cases, update workflow status, and add notes.

Administrators have the highest level of access and can additionally manage users, manage roles, and view system-level information.

This diagram provides a high-level view of the functional scope of the system and demonstrates how access to features is controlled according to role.

2.3.2 Simplified Use Case Diagram

This simplified Use Case Diagram makes it easier to understand which additional permissions each actor has beyond those of lower-level actor(s).

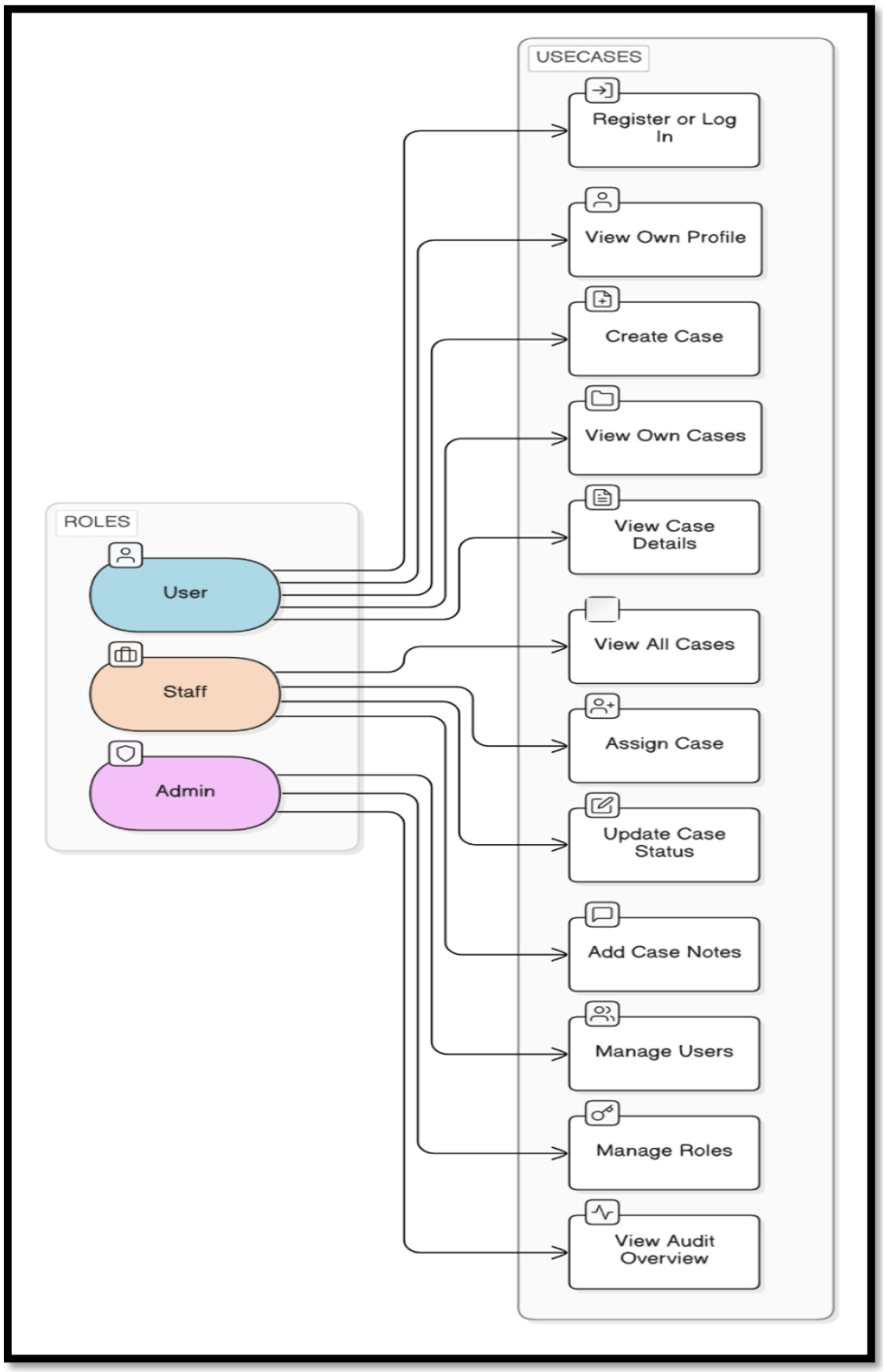
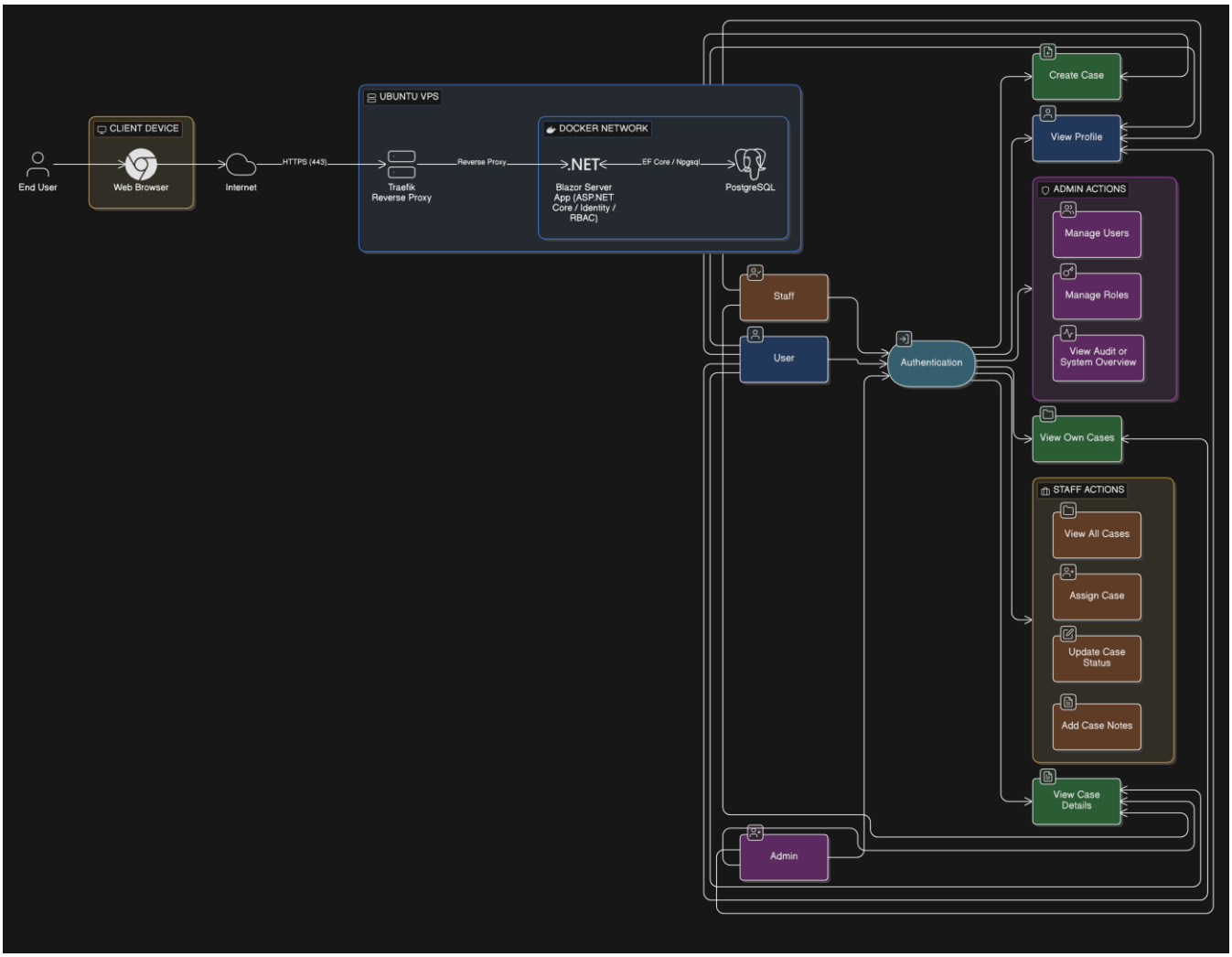


Figure 3: Simplified Use Case Diagram

2.3.3 Architecture diagram

This diagram shows the deployed runtime architecture.



The architecture diagram shows the main technical components of the system and how they interact in the deployed environment. End users access the application through a web browser over HTTPS. Requests are routed through a Traefik reverse proxy, which handles secure routing and TLS termination. Traffic is then forwarded to the Blazor Server application, which contains the user interface, business logic, authentication, and role-based access control functionality. The application connects to a PostgreSQL database using Entity Framework Core and the Npgsql provider for persistent data storage. These services are deployed in Docker containers on an Ubuntu VPS, providing a modern and portable cloud-hosted architecture.

2.4 Technology Selection and Justification

The Secure Workflow System was developed using a modern, full-stack technology stack chosen to balance security, scalability, maintainability, and alignment with contemporary software development practices. The selected technologies support rapid development, strong authentication and authorisation capabilities, and seamless deployment to a cloud-hosted environment.

The chosen technologies incorporate the developer's existing skill set in C#, .NET, and DevOps practices, allowing the project to focus on architectural quality and security design rather than excessive learning overhead.

2.4.1 Blazor Server (.NET 10 / C#)

The application was implemented using Blazor Server, part of the ASP.NET Core framework, with C# as the primary programming language. This approach allows both frontend and backend logic to be written in a single, strongly typed language, reducing complexity compared to multi-language stacks such as JavaScript-heavy frameworks.

Blazor Server was selected due to its support for:

- Real-time UI updates via SignalR
- Built-in integration with ASP.NET Core Identity
- Strong typing and compile-time checking
- Clean separation of concerns through component-based architecture

This makes it particularly suitable for a workflow system where user interaction and state updates occur frequently.

2.4.2 ASP.NET Core Identity (Authentication & RBAC)

Authentication and authorisation are handled using ASP.NET Core Identity, which provides a secure and extensible framework for managing users, roles, and credentials.

This was chosen because it:

- Provides secure, industry-standard authentication mechanisms
- Supports role-based access control (RBAC) natively

- Integrates directly with Entity Framework Core
- Reduces the need to implement custom security logic

Using Identity ensures that the system adheres to best practices in user management and access control, which is critical for a system handling potentially sensitive workflow data.

2.4.3 PostgreSQL (Database)

The system uses PostgreSQL as its primary relational database, accessed through the Npgsql Entity Framework Core provider.

PostgreSQL was selected due to:

- Its robustness and reliability as an open-source database system
- Strong support for relational data and transactional consistency
- Compatibility with cloud-hosted environments and containerised deployment
- Seamless integration with EF Core

It provides a stable and scalable data storage solution suitable for managing workflow cases, user data, and audit information.

2.4.4 Entity Framework Core (ORM)

Entity Framework Core (EF Core) was used as the Object-Relational Mapper (ORM) to manage database interactions.

EF Core was chosen because it:

- Simplifies database access using strongly typed C# models
- Supports code-first development and automated migrations
- Integrates directly with ASP.NET Core Identity
- Improves developer productivity and maintainability

This allows the database schema to evolve alongside the application while maintaining consistency across environments.

2.4.5 Docker & Docker Compose (Containerisation)

The application and database are deployed using Docker containers, orchestrated with Docker Compose.

This approach was selected because it:

- Ensures consistent environments across development and production
- Simplifies deployment and configuration management
- Isolates application components (web app, database, reverse proxy)
- Supports scalability and portability

Containerisation aligns with modern DevOps practices and is particularly suitable for cloud-based deployment. (Docker, 2026)

2.4.6 Traefik (Reverse Proxy & HTTPS)

Traefik is used as a reverse proxy to manage routing and HTTPS configuration.

It was chosen because it:

- Automatically handles TLS certificate generation via Let's Encrypt
- Integrates directly with Docker through service labels
- Simplifies domain-based routing for multiple services
- Reduces manual server configuration

This enables secure external access to the application while keeping internal services isolated.

2.4.7 Ubuntu VPS (Cloud Hosting Environment)

The system is deployed on a Ubuntu 24.04 LTS Virtual Private Server (VPS).

This environment was selected due to:

- Stability and long-term support (LTS)
- Compatibility with Docker and modern deployment tooling
- Full control over server configuration
- Cost-effectiveness compared to managed cloud platforms

This provides a flexible and realistic production environment for deploying the system.

2.4.8 GitHub & GitHub Actions (Version Control & CI)

GitHub is used for version control, with GitHub Actions (GitHub, 2026) implementing a Continuous Integration (CI) pipeline.

This setup was chosen because it:

- Supports collaborative development and version tracking
- Automates build and deployment processes
- Integrates with container registries (GHCR)
- Reflects industry-standard DevOps workflows

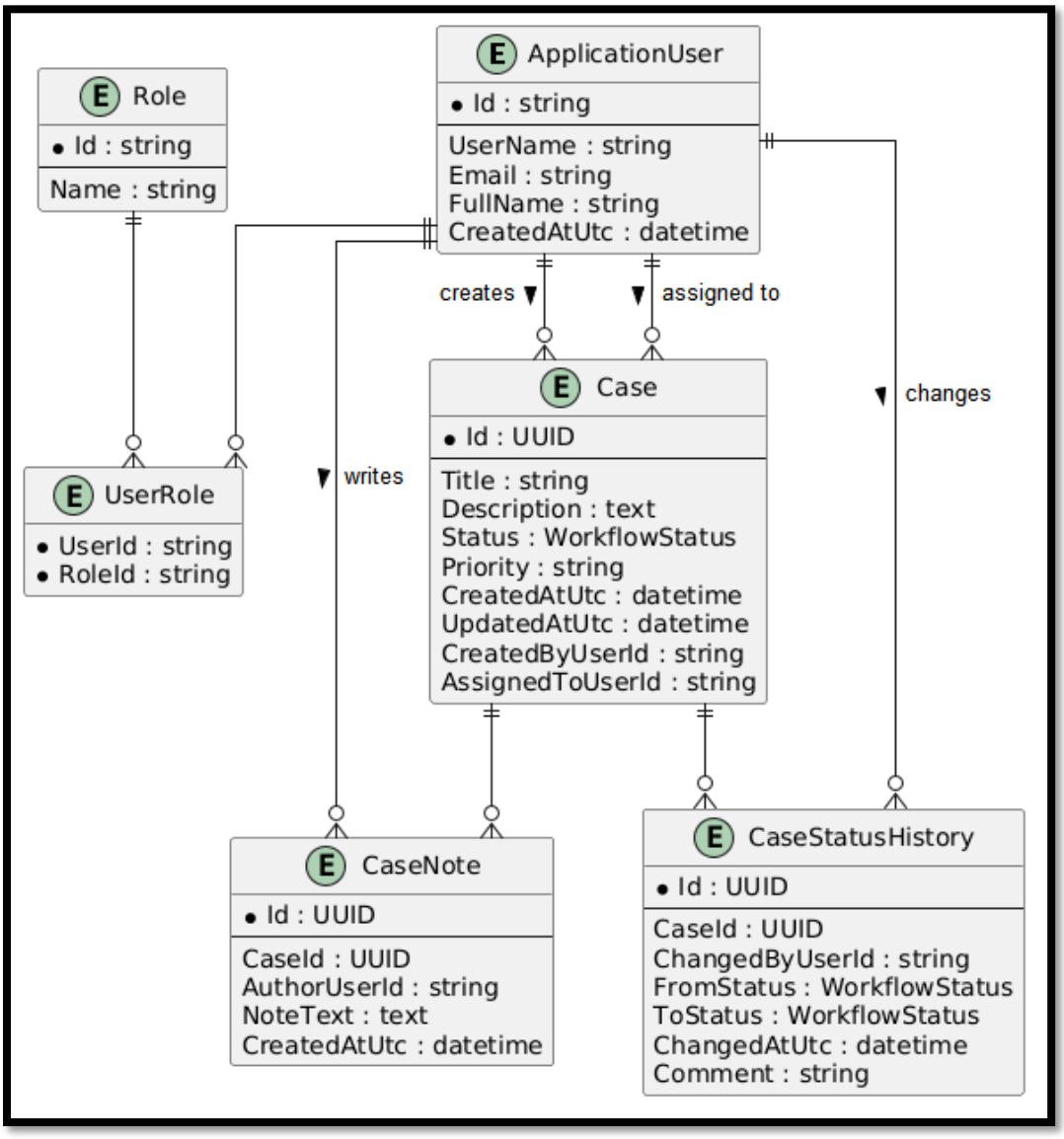
This ensures that code changes are automatically tested and deployed, improving reliability and consistency.

2.4.9 Summary

The selected technology stack provides a cohesive and modern foundation for the Secure Workflow System. Each component was chosen to support secure authentication, structured workflow management, and reliable cloud deployment, while also aligning with current industry practices in software development and DevOps.

2.5 Database schema design

The Entity Relationship diagram represents the main data structures used by the Secure Workflow System and the relationships between them.



The system is centred around the Case entity, which stores the core workflow item, including its title, description, status, timestamps, and ownership information. Cases are created by users and may also be assigned to staff members for processing. Additional entities such as CaseNote and CaseStatusHistory are included to support collaboration, auditability, and workflow tracking. User authentication and authorisation are supported through ASP.NET Core Identity, represented by the ApplicationUser, Role, and UserRole entities. Together, these entities provide the data model needed to support workflow creation, role-based access control, and status progression.

2.6 Security Design

The RBAC model defines how permissions are distributed across the different system roles. The system uses three main roles: User, Staff, and Admin.

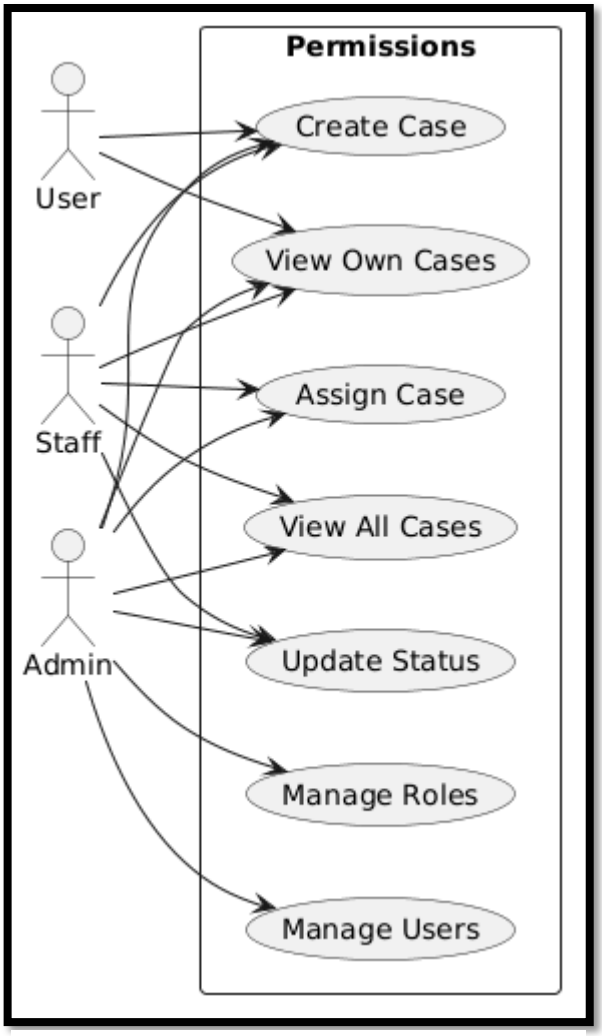


Figure 4: RBAC Permissions Diagram

Users have access to core self-service functions such as creating cases and viewing their own submissions.

Staff users have broader operational access, including the ability to view all cases, assign work, and update workflow status.

Administrators have full system access and are responsible for user and role management, and system oversight. This ensures that users can only perform actions appropriate to their responsibilities (Sandhu, 1996), and forms an important part of the system's overall security design.

The RBAC matrix provides a concise summary of how permissions are allocated across roles. It helps demonstrate that access control is applied consistently and that higher-privileged roles inherit or extend the permissions available to lower-privileged roles.

Permission / Role	User	Staff	Admin
Register / Log in	X	X	X
View own profile	X	X	X
Create case	X	X	X
View own cases	X	X	X
View all cases		X	X
View case details	Own	X	X
Assign case		X	X
Update workflow state		X	X
Add case notes	Own	X	X
Manage users			X
Manage roles			X
View audit / overview			X

Figure 5: RBAC Permissions Matrix

2.6.1 Role-Based Access Control (RBAC) Decision Flow Diagram

This diagram illustrates the role-based access control (RBAC) logic used within the system. It shows how a user action is evaluated by checking the user's assigned role, after which access to specific system functions is granted or denied.

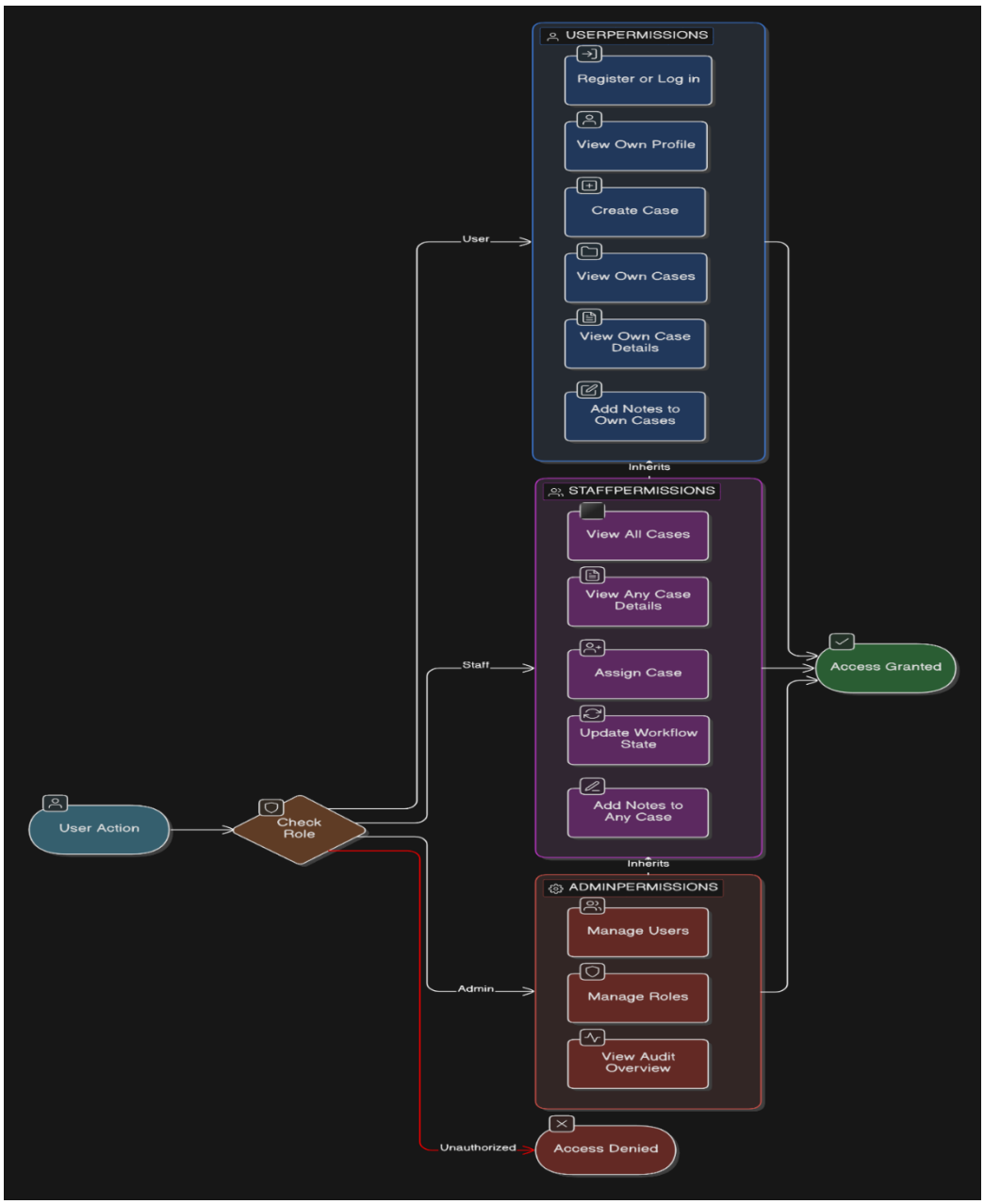


Figure 6: Role-Based Access Control (RBAC) Decision Flow Diagram

Each role (User, Staff, and Admin) is associated with a defined set of permissions, and unauthorised access attempts result in access being denied. The diagram highlights how the system enforces security by ensuring that only users with appropriate roles can perform specific actions consistent with established RBAC principles (Sandhu, 1996).

2.7 Workflow Design

This workflow diagram illustrates how users interact with the system following successful authentication. All roles—User, Staff, and Admin—are first routed through a central authentication process, after which access to system functionality is determined based on role.

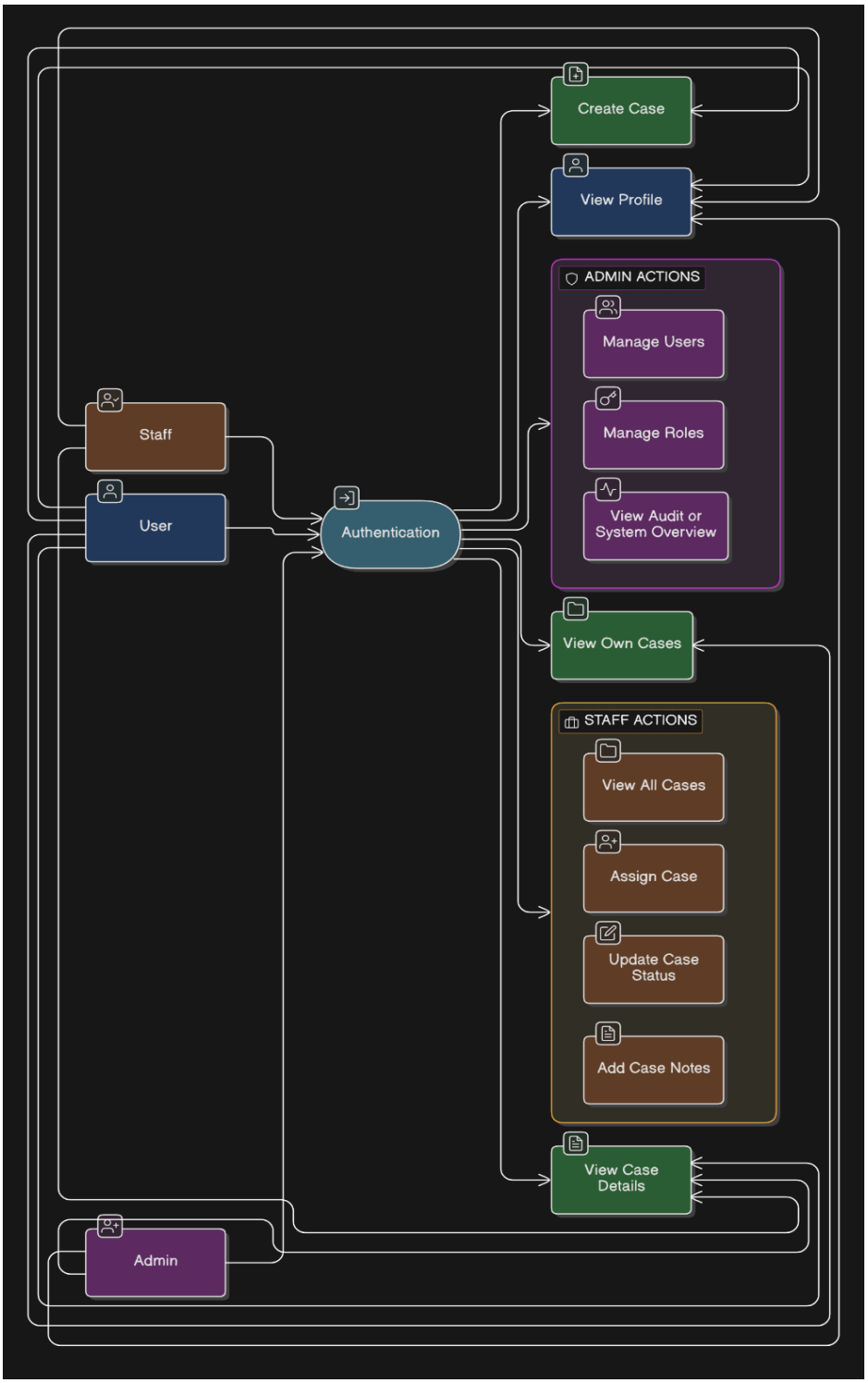


Figure 7: Workflow Diagram

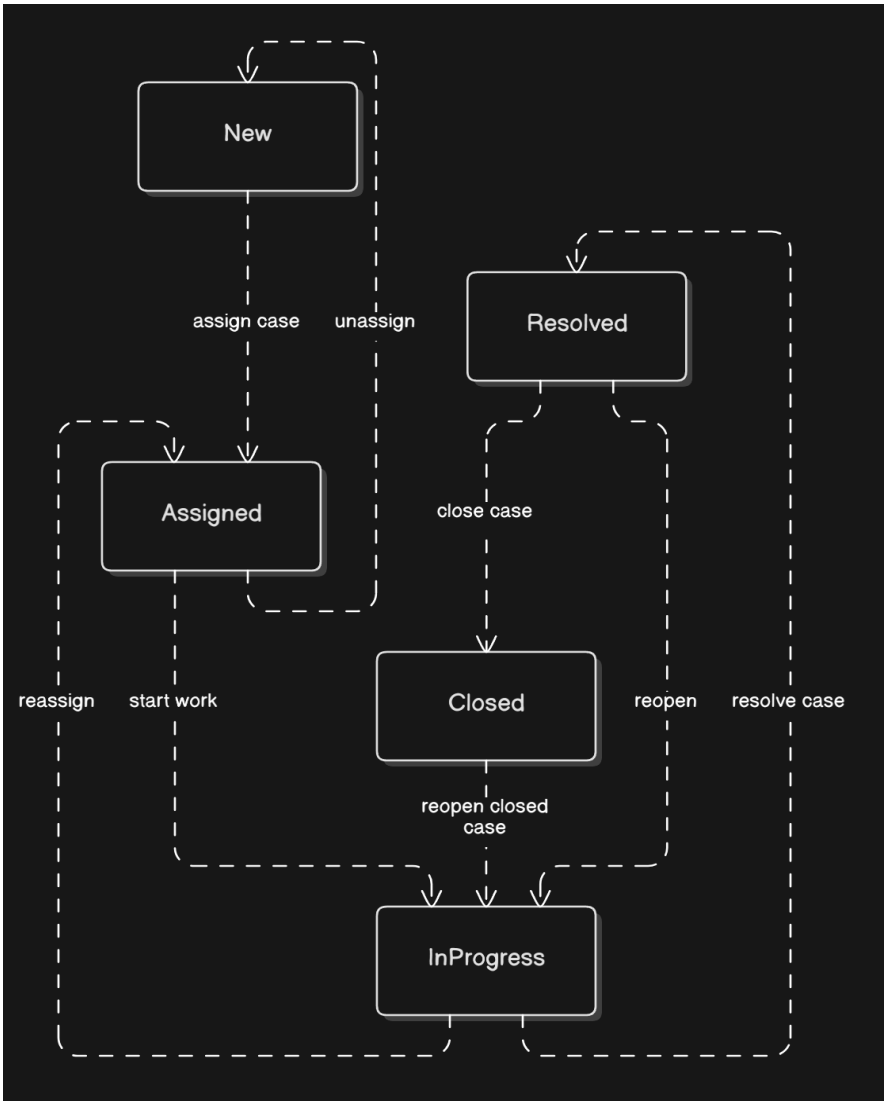
Standard users can perform core actions such as creating cases, viewing their profile, and accessing their own cases.

Staff users have extended capabilities, including viewing all cases, assigning cases, updating case status, and adding notes.

Administrators have the highest level of control, with additional access to user management, role management, and system audit features. The diagram highlights how authentication acts as the entry point to the system and how functionality is structured and grouped according to role-based permissions.

2.7.1 Workflow State Diagram

The workflow state diagram illustrates the lifecycle of a case as it progresses through the system.



A newly created case begins in the **New** state and may then move to **Assigned** once responsibility has been allocated.

From there, it can progress to **In Progress** while work is being carried out, followed by **Resolved** when the issue has been addressed, and finally **Closed** when the workflow is complete.

In some cases, controlled backward transitions are allowed such as reopening a resolved case.

Figure 8: Workflow State Diagram

This state-based approach provides a clear and structured mechanism for managing workflow progression and ensures that transitions occur in a controlled and predictable manner.

These diagrams establish the design foundation for the implementation phase and help ensure that system functionality, security controls, and deployment architecture remain aligned throughout development.

2.8 Deployment / DevOps Design

The deployment architecture of the Secure Workflow System is designed to support automated, secure, and reproducible delivery of application updates using modern DevOps practices. The system integrates Continuous Integration and Continuous Deployment (CI/CD), artefact management, and secure remote deployment to ensure consistent and reliable releases.

A Git-based workflow is used for source code management, with the project hosted on GitHub. Changes pushed to the main branch automatically trigger a GitHub Actions workflow, implementing a CI/CD pipeline. During the Continuous Integration phase, the application is built, and a Docker container image is created to encapsulate the application and its dependencies in a consistent runtime environment.

The Deployment Process

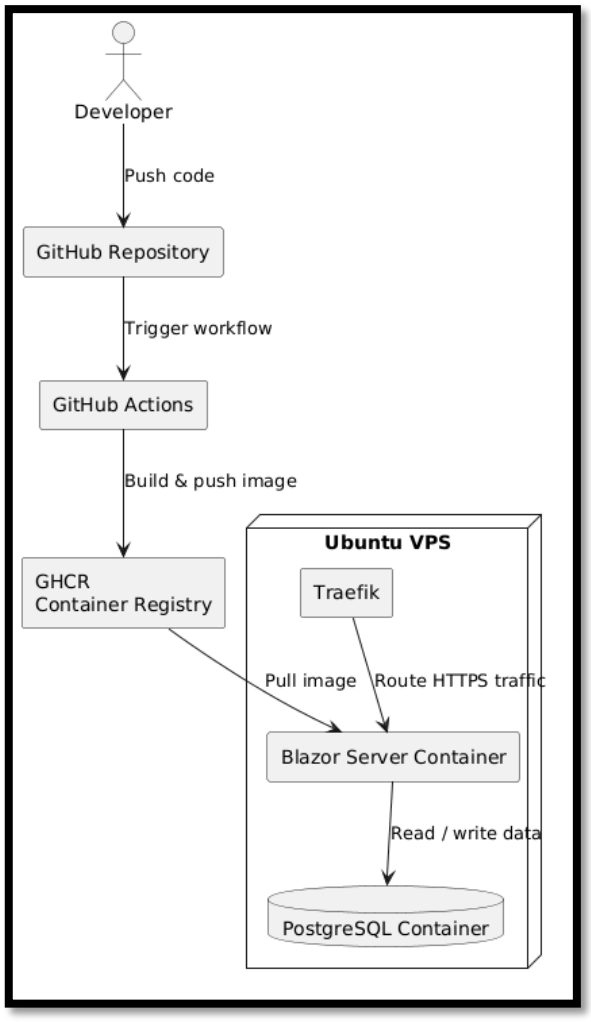


Figure 9: DevOps Deployment Process

Following a successful build, the container image is published to the GitHub Container Registry (GHCR), which serves as the system's artefact repository. This approach ensures that each build produces a versioned, reusable artefact that can be deployed consistently across environments. Storing images in GHCR also supports traceability and rollback, as specific image versions can be referenced and redeployed if required.

The Continuous Deployment phase is implemented using secure remote execution over SSH. The GitHub Actions workflow connects to the cloud-hosted Virtual Private Server (VPS) using an SSH key stored securely in GitHub Secrets. This eliminates the need for hard-coded credentials and ensures that sensitive deployment information is protected.

Once connected, the deployment process updates the running system by pulling the latest container images from GHCR and restarting the application services using Docker Compose. This approach allows the system to be updated with minimal downtime and ensures that the deployed environment remains consistent with the tested build artefacts.

To further enhance security, all sensitive configuration values, including SSH keys and environment variables, are managed through encrypted secrets within the CI/CD platform. This prevents exposure of credentials in source code and aligns with best practices for secure DevOps workflows.

Overall, this deployment architecture demonstrates a fully automated CI/CD pipeline, centralised artefact management, and secure infrastructure access. It reflects modern DevOps principles by enabling rapid, reliable, and secure delivery of application updates while maintaining strong control over deployment processes.

This approach aligns with DevOps principles such as automation, consistency, and security, and supports key performance metrics including reduced deployment time and improved reliability (Kim, et al., 2021).

Chapter 3 : Implementation

3. Implementation

This chapter describes the practical implementation of the Secure Workflow System, including the development environment, database architecture, authentication mechanisms, workflow functionality, and deployment infrastructure used to build and deploy the application.

3.1 Development Environment and Setup

The Secure Workflow System, SecFlo, was developed using a modern full-stack development environment designed to support rapid development, testing, and deployment of web applications. The primary development platform was Microsoft Visual Studio, which provided integrated support for .NET development, debugging, and project management.

The application was built using .NET 10 and the Blazor Web App (Server) framework. This framework was selected due to its ability to support server-side rendering, integrated authentication mechanisms, and seamless interaction with backend services. The use of Blazor Server enabled the implementation of dynamic, stateful user interfaces while maintaining a simplified deployment model compared to fully client-side architectures.

Entity Framework Core was used as the Object-Relational Mapping (ORM) tool, allowing the application to interact with the database using strongly-typed C# objects. This reduced the need for manual SQL queries and improved maintainability. Database schema changes were managed using EF Core migrations, enabling incremental updates and ensuring consistency between the application model and the underlying database.

PostgreSQL was selected as the database management system, accessed via the Npgsql provider for .NET. PostgreSQL provides a robust, open-source relational database solution with strong support for data integrity, performance, and scalability. The database was configured to run locally within a Docker container, ensuring consistency across development environments and simplifying deployment.

Docker Desktop was used to manage containerised services, including the PostgreSQL database and pgAdmin. Containerisation allowed the application to run in an isolated and reproducible environment, reducing configuration issues and supporting a DevOps-oriented workflow. Docker

Compose was used to define and manage multi-container configurations, including the application, database, and database administration tools.

pgAdmin 4 was included as a containerised database administration tool, providing a web-based interface for inspecting database tables, executing SQL queries, and verifying application data. This was particularly useful during development for validating database operations such as case creation, assignment, and status history tracking.

The development environment also incorporated Git for version control, with the project hosted on GitHub. This enabled structured version management, incremental development, and integration with automated build and deployment pipelines.

Overall, the chosen development environment supported efficient implementation, testing, and debugging of the system, while aligning with modern software engineering and DevOps practices. This ensured that the system could be developed and tested in a manner consistent with its intended deployment architecture.

3.2 Database and Data Layer Implementation

The database layer of the Secure Workflow System was implemented using Entity Framework Core in conjunction with a PostgreSQL relational database. This approach enabled the use of strongly-typed domain models while maintaining a structured and consistent underlying database schema.

A custom ApplicationDbContext class was developed, inheriting from IdentityDbContext<ApplicationUser>, allowing seamless integration between application-specific entities and ASP.NET Core Identity tables. This unified context enabled the management of both authentication data and workflow-related data within a single database.

3.2.1 Case Entity

The primary domain entity in the system is the Case, which represents an individual workflow item. Each case contains key attributes including a title, description, workflow status, and timestamps for creation and updates. The entity also includes relationships to users, allowing each case to be associated with both a creator and an optional assigned user.

Data validation and schema constraints were enforced using a combination of data annotations and Fluent API configuration. Fields such as Title, Description, and Status were marked as required and constrained by maximum lengths to ensure data integrity and prevent invalid input.

Sample case data in the database

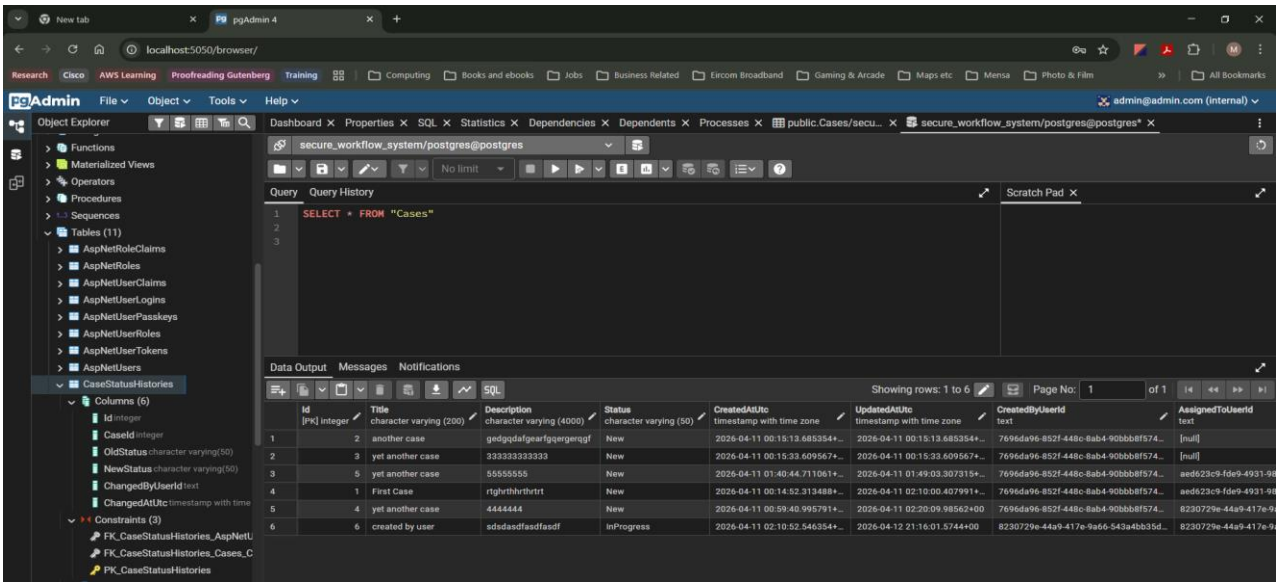


Figure 10: Sample case data showing user ownership, assignment, and workflow status stored in the PostgreSQL database

The relationship between cases and users is reflected in the database, where each case record is linked to both a creator and an optional assigned user. Figure 10 illustrates sample case data retrieved from the PostgreSQL database, demonstrating the storage of workflow state, user associations, and timestamps.

Database query of case data

Dashboard × Properties × SQL × Statistics × Dependencies × Dependents × Processes × public.Cases/secu... × secure_wo

secure_workflow_system/postgres@postgres

No limit

Query Query History

```
1  SELECT
2      c."Id",
3      c."Title",
4      c."Status",
5      creator."Email" AS "CreatedBy",
6      assigned."Email" AS "AssignedTo",
7      c."CreatedAtUtc"
8  FROM "Cases" c
9  JOIN "AspNetUsers" creator ON c."CreatedByUserId" = creator."Id"
10 LEFT JOIN "AspNetUsers" assigned ON c."AssignedToUserId" = assigned."Id"
11 ORDER BY c."CreatedAtUtc" DESC;
```

Data Output Messages Notifications

SQL Showing rows: 1 to 6

	Id integer	Title character varying (200)	Status character varying (50)	CreatedBy character varying (256)	AssignedTo character varying (256)	CreatedAtUtc timestamp with time zone
1	6	created by user	InProgress	user@user.com	user@user.com	2026-04-11 02:10:52.546354+...
2	5	yet another case	New	admin@local.test	mmckibbin1@gmail.com	2026-04-11 01:40:44.711061+...
3	4	yet another case	New	admin@local.test	user@user.com	2026-04-11 00:59:40.995791+...
4	3	yet another case	New	admin@local.test	[null]	2026-04-11 00:15:33.609567+...
5	2	another case	New	admin@local.test	[null]	2026-04-11 00:15:13.685354+...
6	1	First Case	New	admin@local.test	mmckibbin1@gmail.com	2026-04-11 00:14:52.313488+...

Figure 11: Sample case data showing user ownership, assignment, and workflow status

Relationships between cases and users were explicitly defined using the Fluent API. As shown in Figure 12, each case is linked to a creator via the CreatedByUserId foreign key, with deletion restricted to prevent accidental data loss. Additionally, cases may be assigned to a user via the AssignedToUserId field, with a SetNull delete behaviour to preserve cases if an assigned user is removed.

```
entity.HasOne(c => c.CreatedByUser)
    .WithMany()
    .HasForeignKey(c => c.CreatedByUserId)
    .OnDelete(DeleteBehavior.Restrict);
```

Figure 12: Case–User relationship configuration - Restrict delete behaviour

Indexes were introduced to support efficient querying of cases. A composite index on CreatedByUserId and CreatedAtUtc enables fast retrieval of cases created by a user in chronological order, while an index on AssignedToUserId supports efficient filtering of assigned cases.

3.2.2 Case Status History

To support auditability and workflow tracking, a CaseStatusHistory entity was implemented. This entity records all status transitions for a case, including the previous status, the new status, the user responsible for the change, and the timestamp at which the change occurred.

Each status history record is associated with a specific case through a foreign key relationship, with cascading delete behaviour ensuring that related history records are removed if a case is deleted. The relationship to the user who performed the change is configured with restricted deletion to preserve historical accuracy.

Indexes were added to the CaseStatusHistory table to support efficient querying. These include indexes on CaseId, ChangedByUserId, and a composite index on CaseId and ChangedAtUtc to allow efficient retrieval of status changes in chronological order. Figure 13 below shows the result of a "CaseStatusHistories" query.

Case Status Histories Table

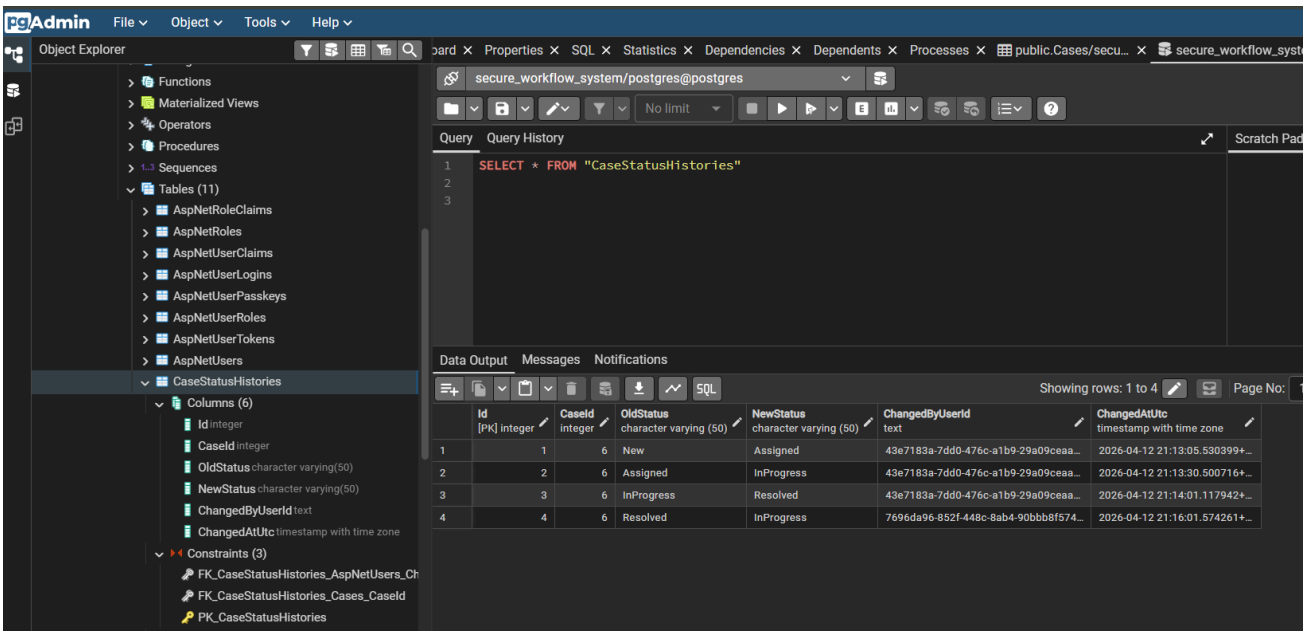


Figure 13: CaseStatusHistories table

The introduction of a dedicated status history entity provides a complete audit trail of workflow progression, ensuring that all state transitions are recorded and traceable. This enhances the transparency and reliability of the system.

Sequential workflow state transitions

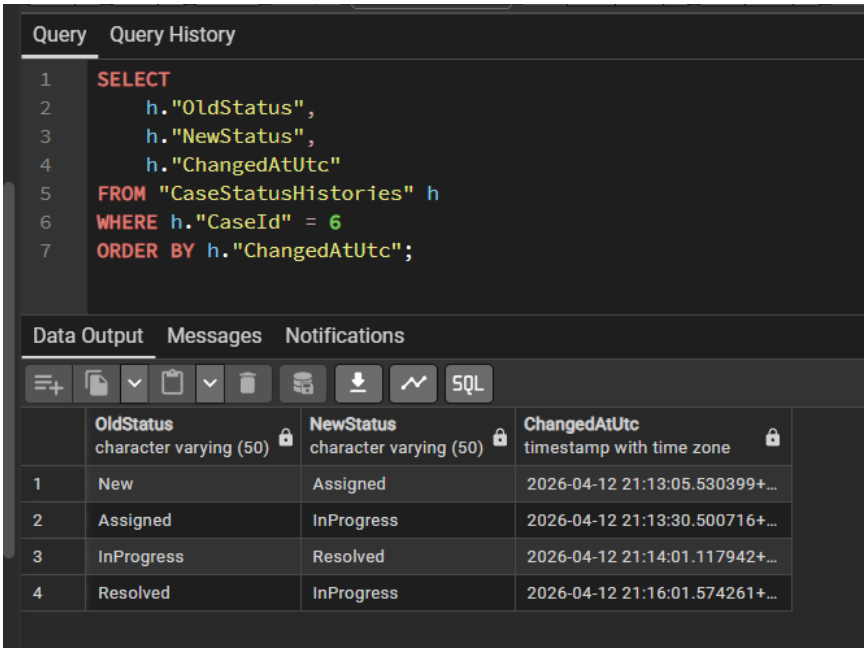


Figure 14: Sequential workflow state transitions for a selected case

3.2.3 Database Context Configuration

The ApplicationDbContext was configured using the Fluent API to define entity relationships, constraints, and indexes. This approach provides greater control over the database schema compared to relying solely on data annotations.

Configuration included:

- Defining required properties and maximum field lengths
- Establishing foreign key relationships between entities
- Specifying delete behaviours to maintain referential integrity
- Creating indexes to optimise common query patterns

This explicit configuration ensures that the database schema accurately reflects the domain model and supports efficient data access operations.

3.2.4 Schema Management with Migrations

Database schema changes were managed using Entity Framework Core migrations. This enabled incremental evolution of the database structure as the application developed, without requiring manual database modification.

Migrations were generated using the dotnet ef migrations add command and applied using dotnet ef database update. These migrations included the creation of the Cases and CaseStatusHistories tables, as well as subsequent schema refinements such as making timestamp fields nullable.

During development, migrations were also applied automatically at application startup in the development environment, ensuring that the database remained synchronised with the application model.

This approach ensured consistency between the application code and the database schema, while also supporting reproducibility across different environments.

3.3 Authentication and Authorization

Authentication and authorization within the Secure Workflow System were implemented using ASP.NET Core Identity, providing a secure and extensible framework for user management and access control.

3.3.1 Authentication

User authentication was implemented using the built-in ASP.NET Core Identity system. This provides support for user registration, login, password hashing, and session management. The system allows users to create accounts and authenticate using email and password credentials, with authentication state maintained through secure cookies.

The Identity framework integrates directly with Entity Framework Core, storing user information within the PostgreSQL database. This includes user credentials, login information, and associated security metadata. By leveraging this framework, the system benefits from established security practices without requiring custom implementation of sensitive authentication logic.

3.3.2 Role-Based Access Control

Authorization was implemented using a role-based access control (RBAC) model. Three primary roles were defined within the system:

- User: Can create cases and view cases they have created or are assigned to
- Staff: Can view all cases, assign cases, and update case status
- Admin: Has full access to case management functionality and can manage user roles

Roles were seeded into the database during application startup, along with a default administrative user account. This ensured that role-based functionality was immediately available during development.

3.3.3 Access Control Enforcement

Access control was enforced at both the application and data levels.

At the application level, access to pages and functionality was restricted using the [Authorize] attribute. This ensured that only authenticated users could access protected routes such as case management pages. Additional role-based restrictions were applied to specific features, such as case assignment and status updates, limiting these actions to users with appropriate roles.

```
@page "/admin/users"
@rendermode InteractiveServer
@attribute [Authorize(Roles = "Admin")]

@inject UserManager<ApplicationUser> UserManager
@inject RoleManager<IdentityRole> RoleManager
```

Figure 15: Server-side authorization enforcement for Admin Role

Access to administrative functionality was enforced using ASP.NET Core authorization attributes. The `[Authorize(Roles = "Admin")]` attribute restricts access to the user management page to authenticated users assigned the Admin role. This approach provides declarative, server-side authorization enforcement and ensures that administrative functionality cannot be accessed through direct URL manipulation or client-side bypass attempts.

```
<AuthorizeView Roles="User,Staff,Admin">
```

Figure 16: Authorisation based on role membership

In addition to backend authorization enforcement, the user interface dynamically adapts based on role membership using Blazor AuthorizeView components. This improves usability by hiding navigation options that are not relevant to the current user role. Using Blazor Server simplified authentication and reduced frontend complexity, although it introduced reliance on persistent SignalR connections.

At the data level, further restrictions were implemented to ensure that users could only access data they are authorised to view. For example, standard users are restricted to viewing cases that they have created or that have been assigned to them. This was enforced through filtered database queries that limit results based on the authenticated user's identity.

Access Denied View

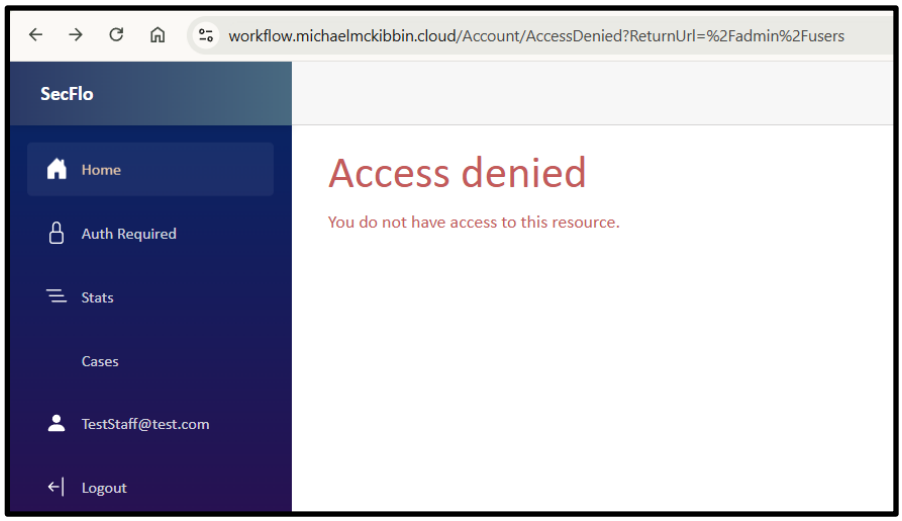


Figure 17: Staff level user attempting to access Admin level "Admin Users" page

This dual-layer approach ensures that access control is not solely dependent on user interface restrictions, reducing the risk of unauthorised data access through direct URL manipulation or client-side bypass.

3.3.4 Administrative Role Management

An administrative interface was implemented to allow users with the Admin role to manage user roles within the system. This interface enables administrators to assign or modify roles for existing users, supporting flexible control over system permissions.

Managing User Roles

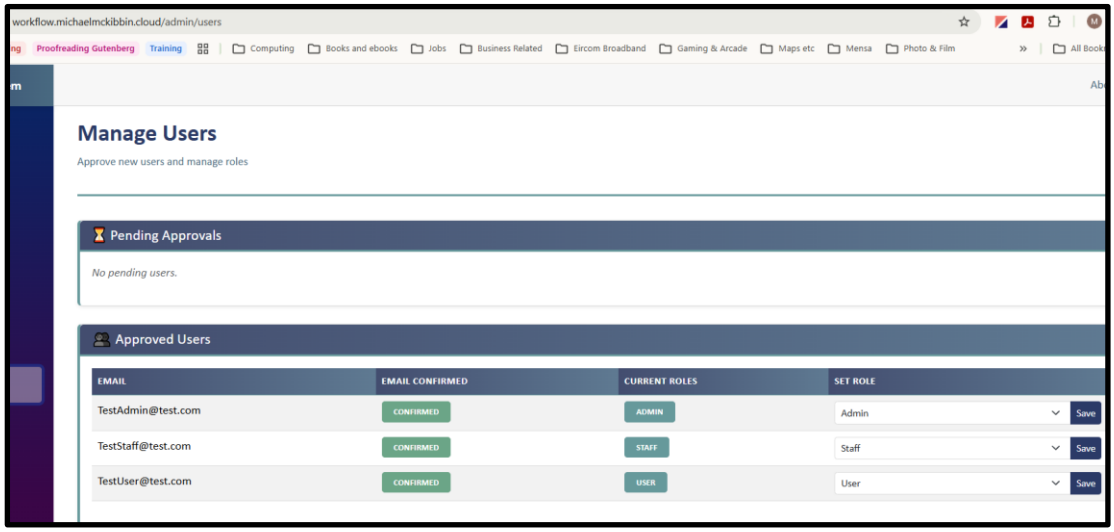


Figure 18: The Role Management page

This functionality provides a practical mechanism for managing access control policies and demonstrates the integration of RBAC within the application.

This implementation ensures that both authentication and authorization are enforced consistently across the application, forming a secure foundation for workflow management.

3.3.5 Production Authentication Considerations

While ASP.NET Core Identity provided a secure and reliable authentication framework for the implementation of the Secure Workflow System, additional considerations would be required for a full production deployment.

The current implementation uses cookie-based authentication with locally managed Identity accounts stored within the application database. This approach is suitable for development and smaller-scale deployments, as it simplifies user management and integrates directly with the existing RBAC architecture. However, larger production environments may require integration with external identity providers or enterprise authentication systems.

Potential future enhancements could include integration with external authentication providers such as Microsoft Entra ID (Azure Active Directory), Google Authentication, or OAuth/OpenID Connect identity services. This would allow organisations to centralise identity management and support features such as Single Sign-On (SSO), multi-factor authentication (MFA), and federated identity management.

Additional production security considerations include:

- enforcing stronger password policies,
- implementing account lockout mechanisms,
- rate limiting authentication requests,
- audit logging of authentication events,
- and monitoring for suspicious login activity.

During deployment testing, secure HTTPS communication was enforced through the Traefik reverse proxy using TLS certificates automatically provisioned by Let's Encrypt. This ensured that authentication credentials and session cookies were encrypted during transmission between users and the deployed application.

The current implementation also highlighted the importance of persistent ASP.NET Core Data Protection key storage within containerised environments. Initially, anti-forgery validation failures occurred following container recreation because encryption keys were stored only within the container filesystem. Persisting the Data Protection key ring using a Docker volume resolved this issue and ensured that authentication sessions and anti-forgery tokens remained valid across deployments.

Authentication Implementation vs Enhancement

Authentication Feature	Current Implementation	Potential Future Enhancement
User Authentication	ASP.NET Core Identity	External Identity Provider
Authorization	RBAC Roles	Policy-Based Access Control
Session Security	Secure Cookies	MFA + Conditional Access
Transport Security	HTTPS/TLS	WAF / Zero Trust Controls

Table 3: Current Authentication Implementation vs Potential Enhancements

Overall, the authentication architecture provides a secure foundation suitable for a lightweight workflow system while also supporting future extension toward more advanced enterprise authentication models.

3.4 Case Workflow Implementation

The core functionality of the Secure Workflow System is centred on the implementation of a structured case workflow. This workflow enables users to create, manage, and progress cases through a defined lifecycle, with access and actions controlled based on user roles.

3.4.1 Case Creation

Case creation is available to authenticated users and represents the entry point into the workflow. Users can create a new case by providing a title and description via a web form. Upon creation, the system automatically assigns the case a default status of *New*, records the creation timestamp, and associates the case with the currently authenticated user.

Creating a Case

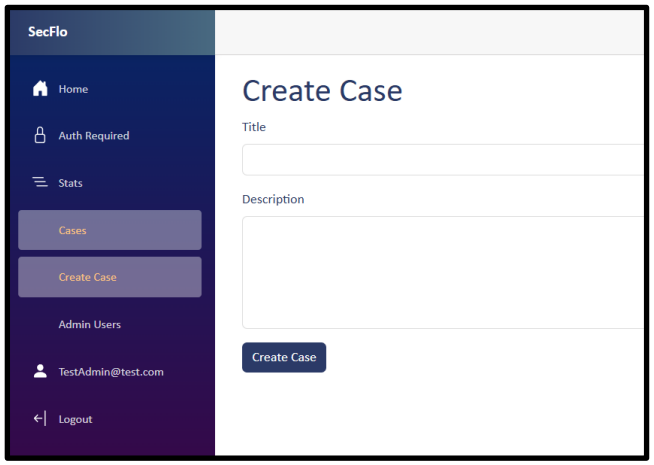


Figure 19: Adding a new case using the Create Case page

This process ensures that all cases are consistently initialised and that ownership is clearly established from the outset. The use of server-side logic ensures that critical fields, such as the creator and timestamps, cannot be manipulated by the client.

3.4.2 Case Listing and Retrieval

The system provides a case listing interface that allows users to view relevant cases. The data returned to the user is filtered based on their role and relationship to the case.

Standard users are restricted to viewing cases that they have created or that have been assigned to them. In contrast, users with the Staff or Admin roles are permitted to view all cases within the

system. This filtering is implemented at the data access level, ensuring that unauthorised data is never exposed to the user interface.

Cases are displayed in a choice of card or list views. Key information such as title, status, and creation date is included. This provides users with an overview of workflow items and their current state.

All Cases List View

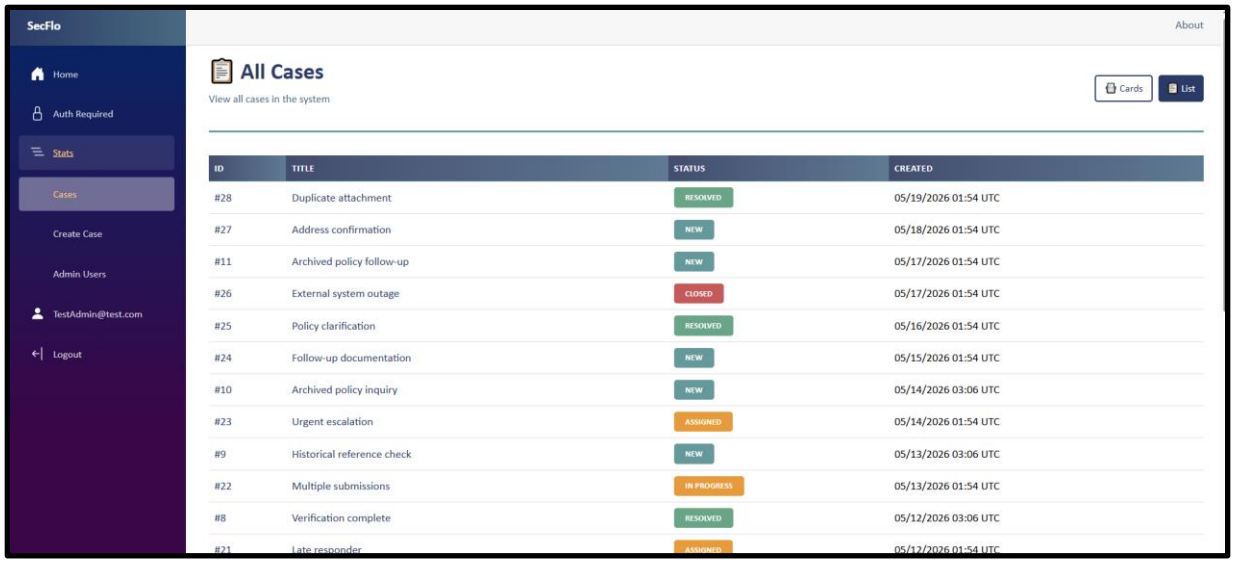


Figure 20: Case listing interface showing role-based filtering and workflow status - list view

All Cases Card View



Figure 21: Case listing interface showing role-based filtering and workflow status – Card view

3.4.3 Case Details and Access Control

A detailed view is provided for individual cases, allowing users to inspect full case information. Access to this view is restricted based on user permissions, ensuring that users can only access cases they are authorised to view.

This is enforced by validating the authenticated user's identity against the case data during retrieval. If a user attempts to access a case outside of their permissions, the system returns a safe response, preventing unauthorised data exposure.

User Level Cases View

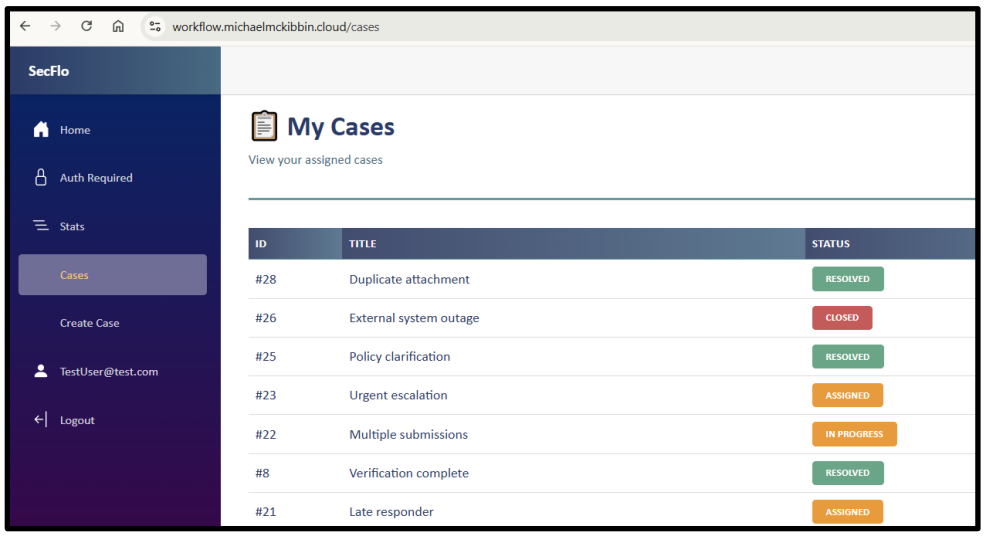


Figure 22: My Cases view for a logged in User

Unauthorised User View

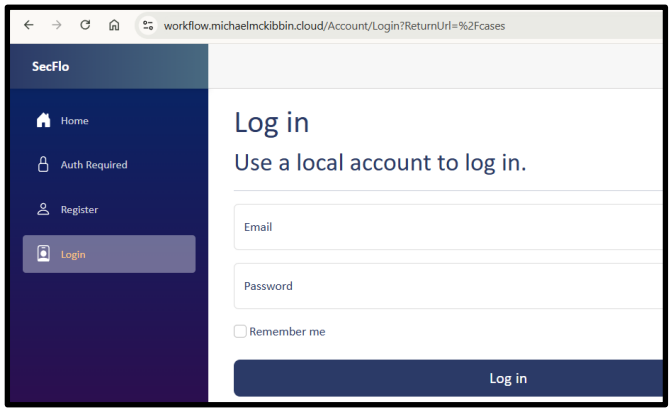


Figure 23: Unauthorised User View

This approach protects against common security risks such as direct URL manipulation and ensures that access control is enforced consistently across the application.

Case Details View

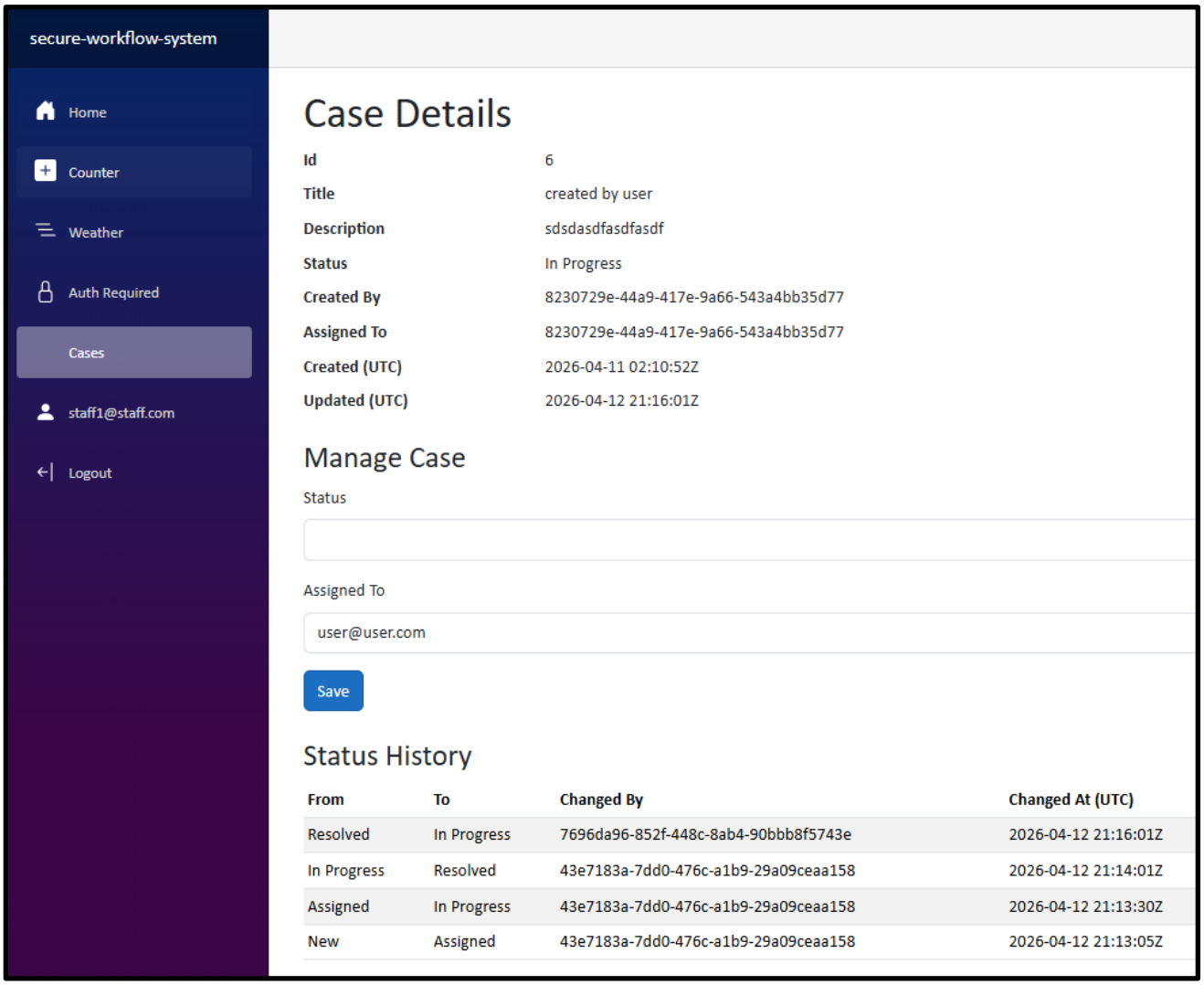


Figure 24: Case details view with access control enforced based on user permissions and with status update history functionality

3.4.4 Case Assignment

Case assignment functionality is available to users with Staff or Admin roles. This allows cases to be allocated to specific users for processing, supporting a structured workflow where responsibility can be transferred.

Unassigned Case

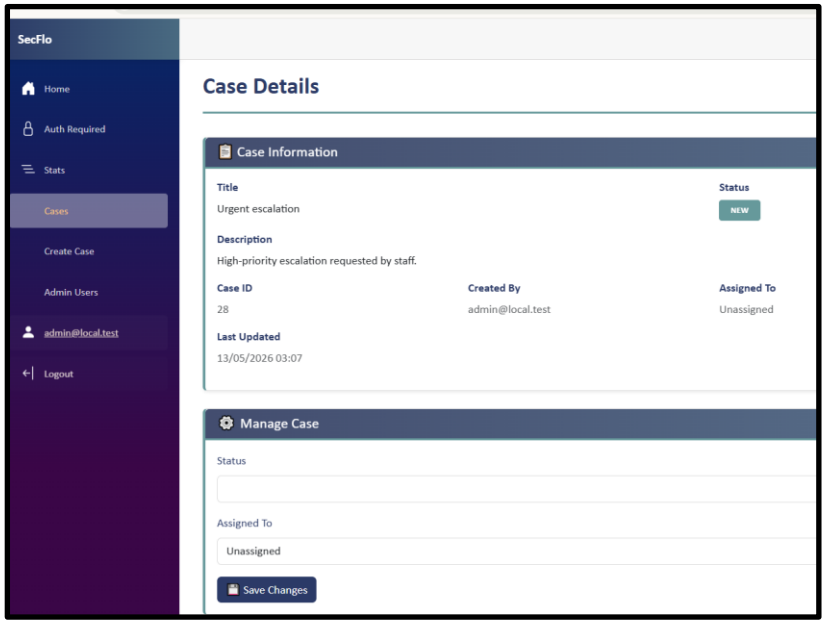


Figure 25: New Case awaiting assignment

When a case is assigned, the system updates the AssignedToUserId field and records the change in the database. The assignment is reflected in the user interface, allowing assigned users to easily identify cases requiring their attention.

Assigned Case

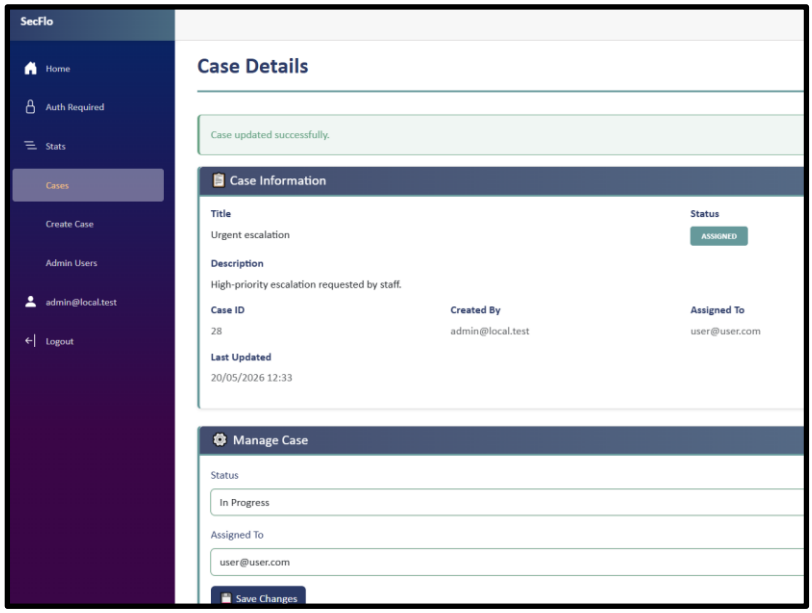


Figure 26: Case assigned to a user with status change success message

The design ensures that assignment does not alter the ownership of the case, preserving the distinction between the creator and the assigned user.

3.4.5 Case Status Updates

Cases progress through a defined set of workflow states, including *New*, *Assigned*, *In Progress*, *Resolved*, and *Closed*. Status updates are restricted to users with appropriate roles, ensuring that only authorised personnel can advance or modify the workflow state.

Case Details Page

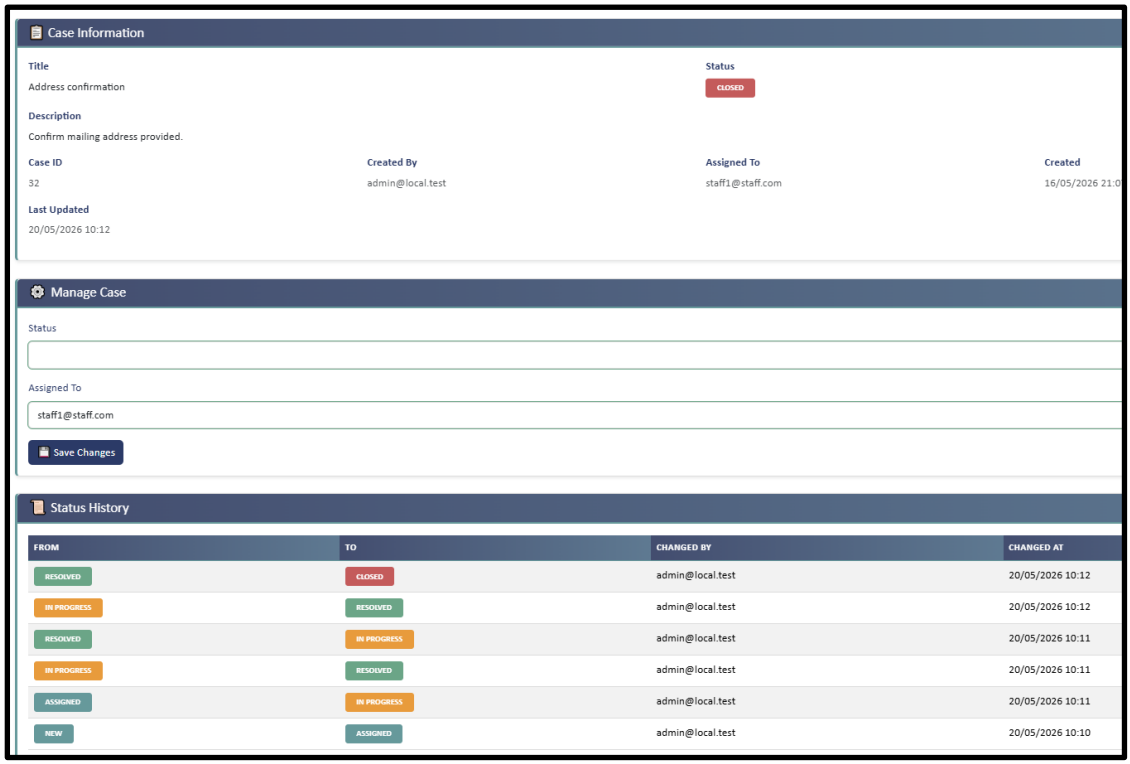


Figure 27: Case Details with Status History

When a status change is performed, the system updates the case record and records the modification timestamp. The use of controlled status values ensures consistency in workflow progression and supports predictable system behaviour.

Status History



Status History

From	To	Changed By	Changed At (UTC)
Resolved	In Progress	7696da96-852f-448c-8ab4-90bbb8f5743e	2026-04-12 21:16:01Z
In Progress	Resolved	43e7183a-7dd0-476c-a1b9-29a09ceaa158	2026-04-12 21:14:01Z
Assigned	In Progress	43e7183a-7dd0-476c-a1b9-29a09ceaa158	2026-04-12 21:13:30Z
New	Assigned	43e7183a-7dd0-476c-a1b9-29a09ceaa158	2026-04-12 21:13:05Z

Figure 28: Case status update functionality demonstrating workflow progression over time

3.4.6 Workflow Behaviour and State Management

The workflow implementation enforces a structured lifecycle for cases, ensuring that cases move through a predictable sequence of states. Although the current implementation allows status changes through role-based permissions, the system design supports the introduction of stricter transition validation rules if required.

This approach balances flexibility with control, allowing users to manage cases efficiently while maintaining a coherent workflow structure. The use of clearly defined states enables future enhancements, such as validation of permitted transitions and automated workflow rules.

3.5 Workflow State and Audit Management

The Secure Workflow System implements a structured approach to managing the lifecycle of workflow cases through a defined set of states. These states represent the progression of a case from creation to completion and provide a clear model for tracking and controlling workflow behaviour.

3.5.1 Workflow States

Each case is associated with a status value that represents its current position within the workflow. The system currently supports the following states:

- New
- Assigned
- In Progress
- Resolved
- Closed

These states were selected to reflect a typical case-handling lifecycle, enabling cases to move from initial creation through processing and eventual resolution.

The use of a fixed set of states ensures consistency across the system and prevents the introduction of invalid or ambiguous workflow values. This supports predictable system behaviour and simplifies both user interaction and backend logic.

3.5.2 State Transitions

Cases progress through the workflow via status updates performed by authorised users. While the current implementation allows status changes based on role permissions, the system is designed to support controlled transitions between states.

For example, a case would typically progress from *New* to *Assigned*, then to *In Progress*, followed by *Resolved*, and finally *Closed*. In certain scenarios, transitions such as reopening a resolved case may also be permitted.

Although strict transition validation is not fully enforced in the current implementation, the structure of the system supports the introduction of rules to restrict invalid transitions. This provides a foundation for more advanced workflow control, such as enforcing mandatory progression paths or preventing regression to earlier states.

3.5.3 Status Change Tracking

To enhance transparency and traceability, all status changes are recorded in the CaseStatusHistory entity. Each status update generates a new record containing:

- the previous status
- the new status
- the user responsible for the change
- the timestamp of the update

This mechanism provides a complete audit trail of workflow progression, allowing the history of each case to be reconstructed and analysed.

The recording of status changes is implemented within the application logic, ensuring that history entries are created automatically whenever a valid status update occurs. This approach ensures consistency and removes reliance on manual logging.

3.5.4 Design Considerations

The workflow state management design prioritises simplicity and clarity while allowing for future extensibility. By defining a fixed set of states and recording all transitions, the system achieves a balance between flexibility and control.

Although string-based workflow states simplified early development and improved readability during debugging, a strongly typed enumeration would reduce the risk of invalid state values in larger systems.

Overall, this approach provides a clear and extensible framework for managing workflow state, supporting both current functionality and future enhancements.

3.6 DevOps and Deployment Implementation

The Secure Workflow System was deployed and managed using a containerised architecture supported by modern DevOps practices. This approach enabled consistent development, simplified environment configuration, and provided a foundation for automated build and deployment processes.

3.6.1 Containerisation with Docker

Docker was used to containerise the application and its supporting services, allowing each component of the system to run in an isolated and reproducible environment. The primary services implemented include:

- A PostgreSQL database container for persistent data storage
- A pgAdmin container for database administration and inspection
- The local application container hosting the Blazor Web App

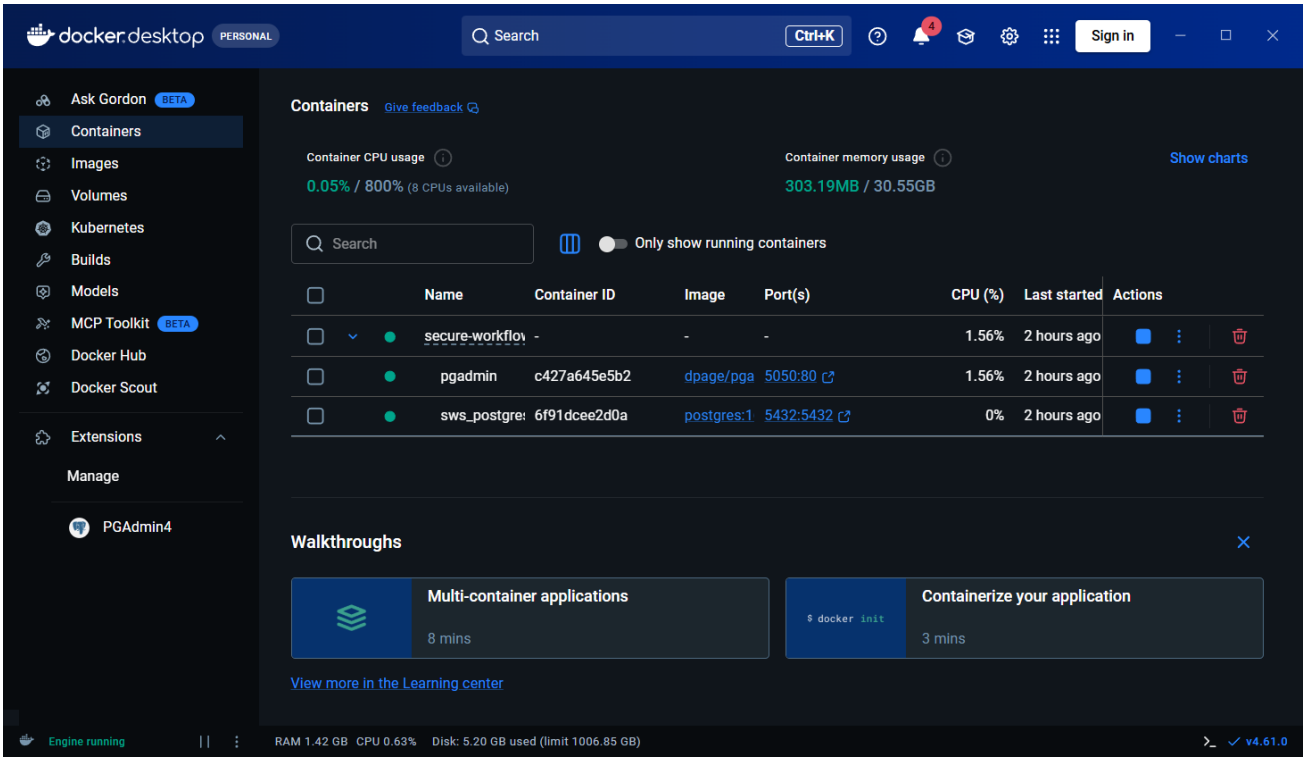


Figure 29: Docker Desktop containers running the application, database, and administration services locally for development.

Containerisation ensures that all required dependencies are packaged together, eliminating inconsistencies between development environments and simplifying deployment.

3.6.2 Multi-Container Configuration with Docker Compose

Docker Compose was used to define and manage the multi-container architecture of the system. The configuration specifies the services, networking, environment variables, and port mappings required to run the application locally.

docker-compose.yml

```
services:
  workflow:
    image: ghcr.io/michaelmckibbin/secure-workflow-system@${IMAGE_DIGEST:?IMAGE_DIGEST must be set to a valid
image digest}
    restart: unless-stopped
    depends_on:
      postgres:
        condition: service_healthy
    environment:
      ASPNETCORE_ENVIRONMENT: ${ASPNETCORE_ENVIRONMENT:-Production}
      ASPNETCORE_URLS: http://+:8080
      DATA_PROTECTION_KEY_RING_PATH: /home/app/.aspnet/DataProtection-Keys
      ConnectionStrings__DefaultConnection: Host=postgres;Port=5432;Database=${POSTGRES_DB:-
secure_workflow_system};Username=${POSTGRES_USER:-postgres};Password=${POSTGRES_PASSWORD}
      SEED_ADMIN_EMAIL: ${SEED_ADMIN_EMAIL}
      SEED_ADMIN_PASSWORD: ${SEED_ADMIN_PASSWORD}
      SEED_USER_EMAIL: ${SEED_USER_EMAIL}
      SEED_USER_PASSWORD: ${SEED_USER_PASSWORD}
      SEED_STAFF_EMAIL: ${SEED_STAFF_EMAIL}
      SEED_STAFF_PASSWORD: ${SEED_STAFF_PASSWORD}
    labels:
      - traefik.enable=true
      - traefik.docker.network=traefik-proxy
      - traefik.http.routers.workflow.rule=Host(`workflow.michaelmckibbin.cloud`)
      - traefik.http.routers.workflow.entrypoints=websecure
      - traefik.http.routers.workflow.tls=true
      - traefik.http.routers.workflow.tls.certresolver=le
      - traefik.http.services.workflow.loadbalancer.server.port=8080
    networks:
      - default
      - traefik-proxy
    ports:
      - "${WORKFLOW_PORT:-8080}:8080"
    volumes:
      - dpkeys:/home/app/.aspnet/DataProtection-Keys

  postgres:
    image: postgres:16
    restart: unless-stopped
    environment:
      POSTGRES_DB: ${POSTGRES_DB:-secure_workflow_system}
      POSTGRES_USER: ${POSTGRES_USER:-postgres}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    ports:
      - "${POSTGRES_PORT:-5432}:5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER:-postgres} -d ${POSTGRES_DB:-secure_workflow_system}"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 10s
    volumes:
      - pgdata:/var/lib/postgresql/data

  # Optional local/dev admin UI. Not required for cloud production deployment.
  pgadmin:
    image: dpage/pgadmin4
    container_name: pgadmin
    restart: unless-stopped
    profiles: ["dev"]
    environment:
      PGADMIN_DEFAULT_EMAIL: ${PGADMIN_DEFAULT_EMAIL:-admin@local.test}
      PGADMIN_DEFAULT_PASSWORD: ${PGADMIN_DEFAULT_PASSWORD}
    ports:
      - "${PGADMIN_PORT:-5050}:80"
    depends_on:
      - postgres
    volumes:
      - pgadmin_data:/var/lib/pgadmin

networks:
  traefik-proxy:
    external: true

volumes:
  pgdata:
  pgadmin_data:
  dpkeys:
```

Figure 30: Multi-container configuration using Docker Compose

This approach allows all services to be started and managed using a single command, significantly reducing setup complexity. It also ensures that the application, database, and administration tools are consistently configured and can communicate over a shared Docker network.

The use of Docker Compose supports a modular architecture, enabling individual services to be updated or replaced without affecting the entire system.

3.6.3 Local Deployment and Environment Configuration

The system was deployed locally using Docker Desktop, with services exposed via defined port mappings. The application is accessible through a web browser, while PostgreSQL and pgAdmin are accessible through their respective ports.

Environment variables were used to configure key settings such as database connection strings and service credentials. This approach separates configuration from code, improving security and flexibility. It also enables different configurations to be used for development and production environments without modifying the application source code.

The local deployment environment closely mirrors the production setup, supporting reliable testing and validation of system behaviour prior to deployment.

3.6.4 Reverse Proxy and HTTPS Configuration

To support secure public access to the deployed application, a Traefik reverse proxy was implemented within the VPS container environment. Traefik acted as the external entry point for incoming web traffic and automatically routed requests to the appropriate application container based on the configured domain name.

A reverse proxy provides several advantages in a containerised cloud environment. Rather than exposing the application container directly to the internet, Traefik handled incoming HTTP and HTTPS requests and forwarded them internally to the Blazor application container. This simplified network configuration, improved security, and allowed TLS certificate management to be centralised within a single component.

Domain-based routing was configured using Docker labels within the container orchestration configuration. Requests for the configured domain (workflow.michaelmckibbin.cloud) were

automatically routed to the Secure Workflow System container through Traefik's routing engine. This approach supports scalable multi-service hosting while avoiding direct exposure of internal container ports.

Traefik was also configured to automatically provision and manage TLS certificates using Let's Encrypt. This enabled HTTPS encryption without requiring manual certificate generation or renewal. HTTPS termination occurred at the reverse proxy layer, where encrypted client traffic was decrypted before being forwarded securely across the internal Docker network to the application container. The use of HTTPS ensured that authentication data, session cookies, and application traffic were transmitted securely between users and the deployed system.

The deployment architecture used in the cloud-hosted environment is illustrated below.

Deployment Architecture

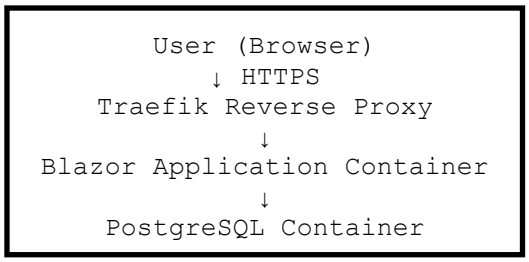


Figure 31: Deployment architecture diagram

Live Cloud Deployment

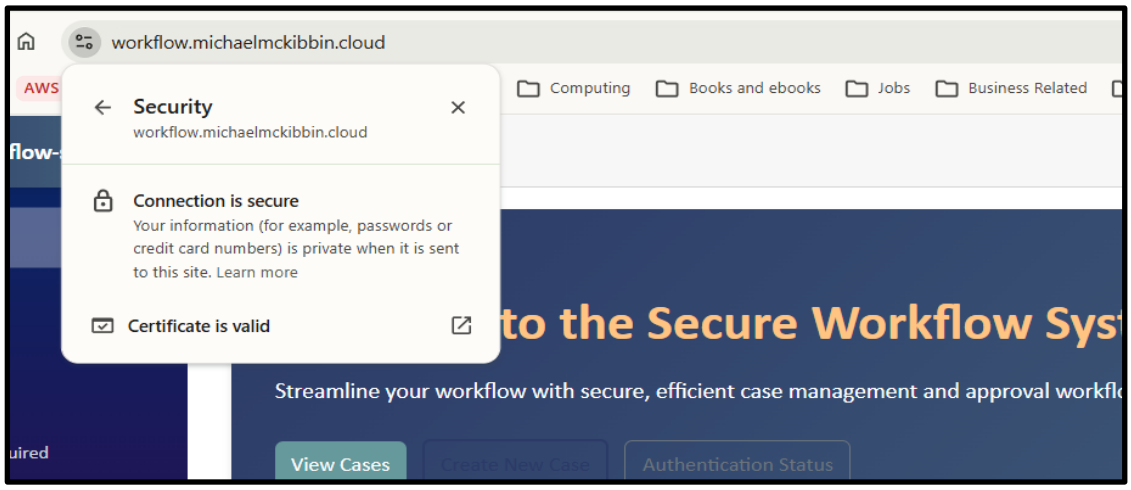


Figure 32: Live cloud deployment secured using HTTPS and Traefik reverse proxy routing.

Traefik automatically handled HTTPS certificate provisioning and renewal using Let's Encrypt. This simplified secure deployment management while ensuring all traffic between users and the application was encrypted. The successful HTTPS deployment also demonstrated that the containerised application could operate correctly behind a reverse proxy within a cloud-hosted environment.

3.6.5 Continuous Integration and Deployment Pipeline

A Continuous Integration and Continuous Deployment (CI/CD) pipeline was implemented using GitHub Actions. This pipeline automates the process of building, packaging, and deploying the application.

GitHub Actions – Build and Push

```
- name: Build and push Docker image
  id: build_image
  uses: docker/build-push-action@v6
  with:
    context: .
    push: true
    tags: |
      ghcr.io/michaelmckibbin/secure-workflow-system:sha-${{ github.sha }}

- name: Upload docker-compose.yml to VPS
  uses: appleboy/scp-action@v0.1.7
  with:
```

Figure 33: GitHub Actions snippet from deploy.yml

When changes are pushed to the repository, the pipeline performs the following steps:

- Builds the application using the .NET SDK
- Creates a Docker image containing the application
- Publishes the image to GitHub Container Registry (GHCR)

This process ensures that each build produces a consistent and versioned artefact that can be deployed across environments. The use of GHCR provides centralised storage for container images and supports version control and traceability.

During later stages of deployment testing, the deployment strategy was improved by transitioning from mutable image tags such as 'latest' and SHA-labelled tags to immutable Docker image digests.

Image digests uniquely identify a specific container image version using a cryptographic hash, ensuring that the exact tested image is deployed consistently across environments. Unlike mutable tags, digests cannot be reassigned to different images, reducing the risk of accidental deployment inconsistencies or cached image conflicts. (Google, 2026)

The use of image digests also improved deployment traceability and rollback capability. Specific deployment versions could be verified directly on the VPS host and redeployed if required, supporting more reliable CI/CD operations and stronger alignment with immutable infrastructure principles commonly used in modern DevOps workflows. (Microsoft, 2026)

Automated Deployment using the GitHub Container Registry (GHCR)

GitHub Container Registry (GHCR) was used to store and distribute the Docker container images generated during the CI/CD pipeline. Using GHCR simplified deployment management by allowing the VPS environment to automatically pull versioned application images directly from a central registry during deployment. The use of immutable image tags and digests improved deployment traceability, supported rollback procedures, and ensured consistent application behaviour across environments. Integrating GHCR with GitHub Actions also streamlined the automated build and deployment workflow used throughout the project.

GitHub Container Registry (GHCR) Images

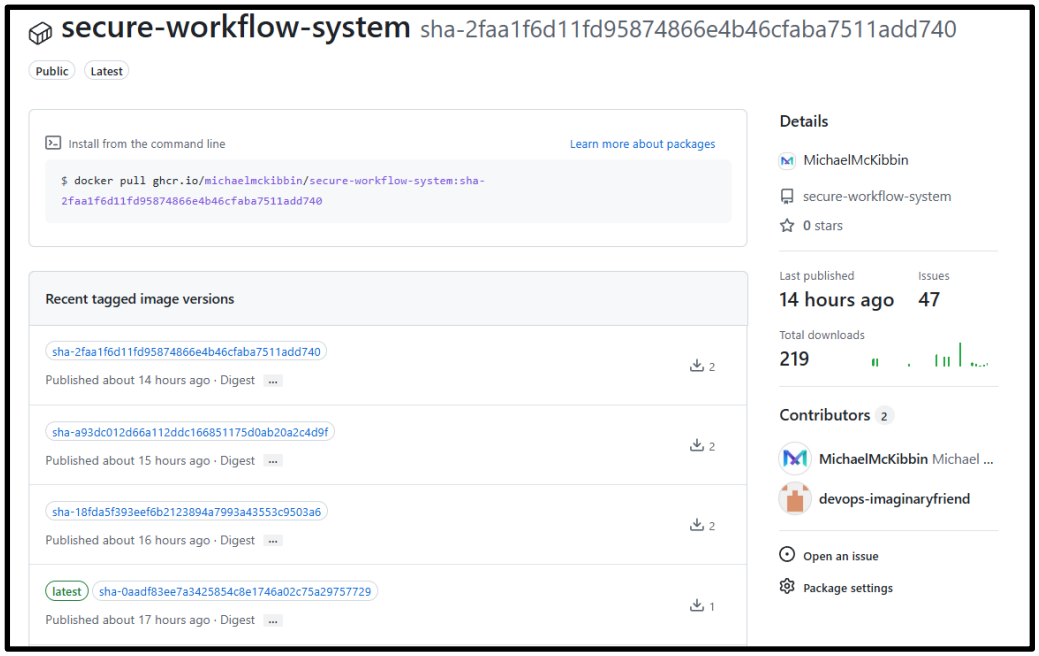


Figure 34: Docker container images published to GitHub Container Registry (GHCR) during the CI/CD deployment process

GitHub Actions Successful Deployment



Figure 35:GitHub Actions Successful Deployment

During deployment testing, an issue was identified in the CI/CD pipeline where an incorrect environment variable reference prevented the correct image tag from being deployed. This resulted in the VPS continuing to run a previously cached container image rather than the newly built version.

The issue was resolved by correcting the image SHA substitution logic and introducing additional deployment logging, including verification of the generated environment file and resolved container configuration. This ensured that the correct image tag was applied during deployment and improved the reliability and observability of the pipeline.

Docker Containers

The deployed container environment was validated using Docker container management tools on the VPS host. The running containers shown below demonstrate the operational deployment of the Secure Workflow System application alongside its supporting PostgreSQL database and reverse proxy infrastructure. This confirmed that the containerised services were functioning correctly within the production cloud environment and communicating successfully across the internal Docker network.

VPS Docker PS Output

```
root@michaelmckibbin:~# cd /docker/workflow
root@michaelmckibbin:/docker/workflow# docker ps
CONTAINER ID   IMAGE                                          COMMAND                  CREATED        STATUS                  PORTS
   NAMES
db2e1da298e8   81c25068e7ef                                   "dotnet secure-workf…"   15 hours ago   Up 15 hours             0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp
   workflow-workflow-1
4b30ce97e027   postgres:16                                    "docker-entrypoint.s…"   15 hours ago   Up 15 hours (healthy)   0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp
   workflow-postgres-1
675311e41429   ghcr.io/michaelmckibbin/michaelmckibbindotcom  "docker-entrypoint.s…"   6 weeks ago    Up 2 weeks              3000/tcp
   myportfolio-michaelmckibbin-1
63adc256cf14   ghcr.io/michaelmckibbin/a1postal:latest        "docker-entrypoint.s…"   7 weeks ago    Up 2 weeks              3000/tcp
   a1postal-a1postal-1
c316837a049a   traefik:v3.6.8                                 "/entrypoint.sh --pr…"   2 months ago   Up 7 days               0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/tcp
   traefik-traefik-1
```

Figure 36: VPS Docker PS Output showing all current containers

The docker `compose -p workflow --env-file workflow.env ps` command was used to inspect the status of the production deployment containers. This verified that the workflow application and

supporting database services were running correctly within the Docker Compose project environment using the configured deployment variables.

Verifying the operational status of the deployed container

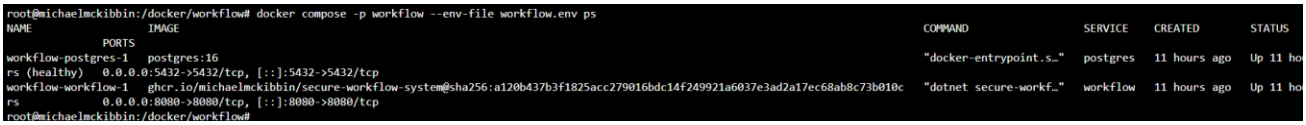


Figure 37: Verifying Container Operational Status (note the image digest identifier)

Breaking down the docker compose command

Component	Purpose
`docker compose`	Docker Compose management command
`-p workflow`	Specifies the deployment project name
`--env-file workflow.env`	Loads deployment environment variables
`ps`	Displays running container status

Table 4: : Using the docker compose command

Docker Manager on VPS

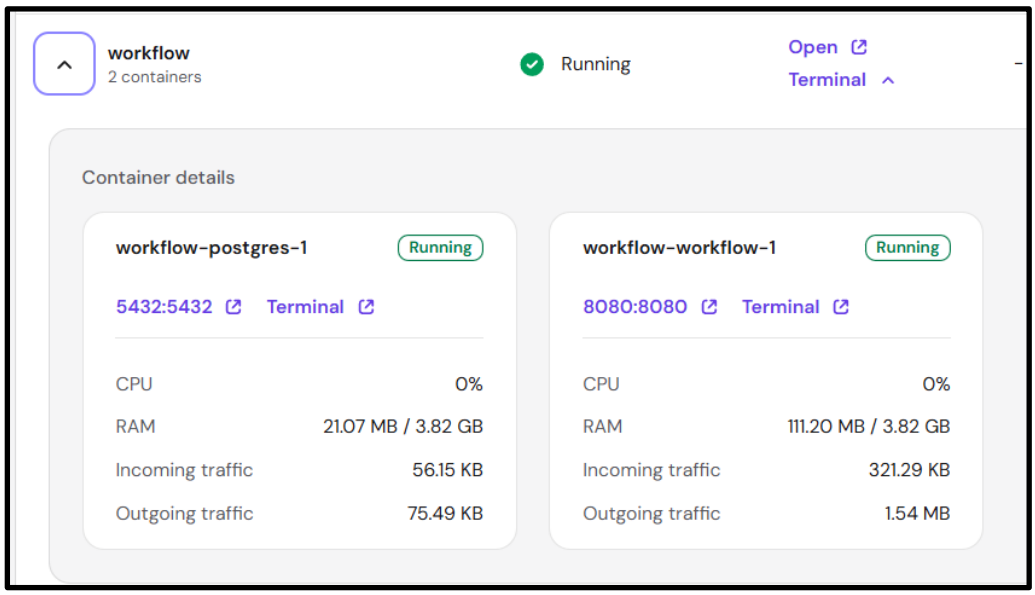


Figure 38: Using Docker Manager to view running containers

The Docker Manager GUI on the VPS allows for easy editing of deployment and environment files, rapid redeployment of containers. It also provides a CLI terminal window that allows for easy access to the VPS Operating System, and all containers.

3.6.6 Secure Deployment Practices

Security considerations were incorporated into the deployment process using environment variables and secret management. Sensitive information, such as database credentials and SSH keys, is not stored directly in source code. Instead, these values are managed using secure configuration mechanisms, including GitHub Secrets.

This approach reduces the risk of credential exposure and aligns with best practices for secure application deployment. It also supports safe automation within the CI/CD pipeline by allowing secure access to external systems during deployment.

3.6.7 Summary

The adoption of containerisation and DevOps practices provides a robust foundation for the Secure Workflow System. The use of Docker and Docker Compose ensures consistent environments, while the CI/CD pipeline enables automated and reliable delivery of application updates.

During deployment testing, ASP.NET Core Data Protection keys were initially stored within the container filesystem. This caused anti-forgery token validation failures after container recreation because the encryption keys were lost. The issue was resolved by persisting the Data Protection key ring using a mounted Docker volume, ensuring session continuity across deployments.

Implemented Features and Technologies

Feature	Technology Used	Purpose
Authentication	ASP.NET Core Identity	Secure login and RBAC
ORM	Entity Framework Core	Database abstraction
Database	PostgreSQL	Persistent relational storage
Containerisation	Docker	Consistent deployment
Reverse Proxy	Traefik	HTTPS and routing
CI/CD	GitHub Actions	Automated deployment

Table 5: Implemented Features and Technologies

This implementation supports scalability, maintainability, and security, while aligning with modern software engineering practices.

Chapter 4: Testing

4. Testing

Testing was performed locally using Visual Studio, Docker Desktop, and PostgreSQL.

Testing on the VPS Cloud made use of Docker containers reachable at:

SecFlo.net

or

workflow.michaelmckibbin.cloud.

4.1 Testing Strategy and Development Approach

This chapter describes the testing and evaluation process used throughout the development of the Secure Workflow System. Testing was performed across multiple layers of the application, including unit testing, integration testing, component testing, workflow validation, security verification, and deployment testing. The objective of the testing process was to verify that the system met the functional and non-functional requirements defined in Chapter 2 while ensuring reliability, security, and deployment stability within the cloud-hosted environment.

The Testing Pipeline

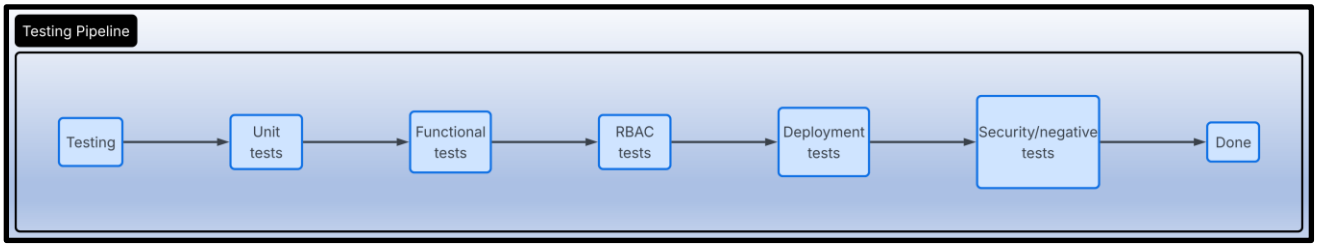


Figure 39: The Testing Pipeline

4.2 Testing Objectives

The testing process was designed to verify that the Secure Workflow System satisfied both the functional and non-functional requirements defined during the design phase. Testing focused not only on application functionality, but also on security, workflow validation, deployment reliability, and

cloud-hosted operation. The primary testing objectives and validation requirements are summarised below.

4.2.1 Functional Requirements Validation

Requirement	Validation Method	Result
Users can log in	Functional authentication testing	Pass
Admin can manage users	Administrative RBAC testing	Pass
User can only view own cases	Authorization and access testing	Pass
Staff can view all cases	Staff role validation testing	Pass
Cases move through workflow states correctly	Workflow transition testing	Pass
System deploys successfully to VPS	Deployment and health endpoint test	Pass

4.2.2 Testing Objectives

Testing was performed across multiple layers of the system, including isolated unit testing, component testing, integration testing, functional testing, security validation, and deployment verification.

Testing Objective	Purpose	Result
Verify authentication and RBAC	Ensure users can only access authorised functionality	Pass
Validate workflow transitions	Ensure cases follow valid lifecycle states	Pass
Test database operations	Verify persistence and integrity	Pass
Validate deployment pipeline	Ensure reliable automated deployment	Pass
Test cloud hosting environment	Ensure production stability	Pass
Verify UI behaviour	Ensure components render correctly	Pass

The following sections describe the specific testing methods and results used to validate each of these objectives.

4.3 Testing Environment

Testing of the Secure Workflow System was performed across both local development and cloud-hosted deployment environments in order to validate application functionality, security, workflow behaviour, and deployment reliability.

Local development and testing were carried out on a Windows-based development workstation using Visual Studio 2026 and the .NET 10 framework. Docker Desktop was used to host local containerised services, including the PostgreSQL database used by the application during development and integration testing. This allowed the application to be tested in an environment closely aligned with the final cloud deployment architecture.

The production deployment environment consisted of an Ubuntu 24.04 VPS hosted within a cloud infrastructure environment. The application was deployed using Docker Compose and consisted of multiple containers, including the Blazor application container, PostgreSQL database container, and Traefik reverse proxy container. HTTPS support was provided through Traefik with automatic TLS certificate management using Let's Encrypt.

Automated testing and deployment validation were integrated into the GitHub Actions CI/CD pipeline. Source code changes pushed to the GitHub repository triggered automated build and deployment workflows, allowing application changes to be validated before deployment to the VPS environment.

Several testing frameworks and tools were used throughout development:

Testing Frameworks and Tools Used

Tool / Technology	**Purpose**
Visual Studio 2026	Application development and debugging
xUnit	Unit testing framework
BUnit	Blazor component testing
Docker Desktop	Local container hosting
PostgreSQL 16	Relational database platform
TestContainers	Integration testing using disposable containers
GitHub Actions	CI/CD automation and deployment validation

GitHub Container Registry (GHCR)	Container image storage
Docker Compose	Multi-container orchestration
Traefik	Reverse proxy and HTTPS routing

Figure 40: Testing Frameworks and Tools Used

This combination of local and cloud-hosted testing environments allowed both isolated application behaviour and full end-to-end deployment scenarios to be validated throughout the development process.

4.4 Unit and Component Testing

Unit testing was used throughout development to validate the behaviour of individual application components and business logic in isolation. The primary goal of unit testing was to ensure that core workflow functionality behaved as expected before integration into the wider application and deployment environment.

The Secure Workflow System used the xUnit testing framework for automated unit testing. Tests were designed to verify the correctness of application services, workflow operations, validation logic, and database interactions. Where appropriate, isolated in-memory databases were used to simulate Entity Framework Core operations without requiring a full PostgreSQL deployment.

Testing focused particularly on the workflow management functionality implemented within the CaseService class. This included validating case creation, case retrieval, workflow state transitions, assignment logic, and status history generation. Automated tests were also used to confirm that invalid operations were handled correctly and that application data remained consistent during workflow updates.

Examples of behaviours validated through unit testing included:

- creating workflow cases successfully,
- verifying default case status values,
- recording workflow status history correctly,
- updating assigned users,
- validating permitted workflow transitions,
- and handling invalid or missing data safely.

Blazor Component Testing

In addition to service-level unit testing, BUnit was used to validate the behaviour of Blazor UI components in isolation. Component tests verified that pages rendered correctly for different user roles and that workflow information was displayed appropriately based on the authenticated user context.

The MyCases component was tested using mocked authentication states and mocked application services. Tests verified that standard users only viewed their assigned workflow cases, while staff users were presented with administrative views containing all available cases. The tests also confirmed that workflow statuses such as Assigned, In Progress, and Resolved rendered correctly within the user interface.

Mock implementations of the ICaseService interface were created using the Moq framework, allowing predictable test data to be injected into the component during execution. Authentication behaviour was simulated using a custom AuthenticationStateProvider, enabling role-based rendering behaviour to be validated without requiring a live authentication system.

The component tests demonstrated that:

- workflow case information rendered correctly,
- role-specific interface behaviour functioned as expected,
- authorized users received appropriate data views,
- and unauthorized data access paths were not exposed through the user interface.

BTest Features

Feature	Demonstrated?
RBAC	Yes
Workflow states	Yes

Authentication mocking	Yes
Service abstraction	Yes
Dependency injection	Yes
Blazor component rendering	Yes
UI conditional logic	Yes

Table 6 Bunit test features:

The table shows successful execution of the automated BUnit component tests during development.

A layered testing approach was adopted to ensure that both successful and unsuccessful operations were validated. Positive test cases confirmed that valid operations behaved correctly, while negative test cases verified that invalid operations generated appropriate exceptions or validation failures.

Unit testing was integrated into the GitHub Actions CI/CD pipeline so that tests were executed automatically during the build process. This ensured that regressions or implementation errors could be detected before deployment to the production VPS environment.

4.5 Integration Testing

Integration testing was used to validate interactions between the application, database, and containerised infrastructure components. Unlike isolated unit tests, integration tests verified that multiple parts of the system operated correctly together within a realistic deployment environment.

PostgreSQL integration testing was performed using containerised database instances hosted through Docker Desktop and TestContainers. This allowed the application to be tested against a real PostgreSQL environment while avoiding dependency on shared development databases.

Integration tests verified:

- database connectivity,
- Entity Framework Core persistence,
- relational data integrity,
- workflow history storage,
- and correct interaction between application services and the PostgreSQL database.

Testing also confirmed that workflow case data, user assignments, and status history records remained consistent following application restarts and container recreation.

Figure 41: Docker test containers created during integration tests in the PostgreSQL container environment.

Integration Test Execution

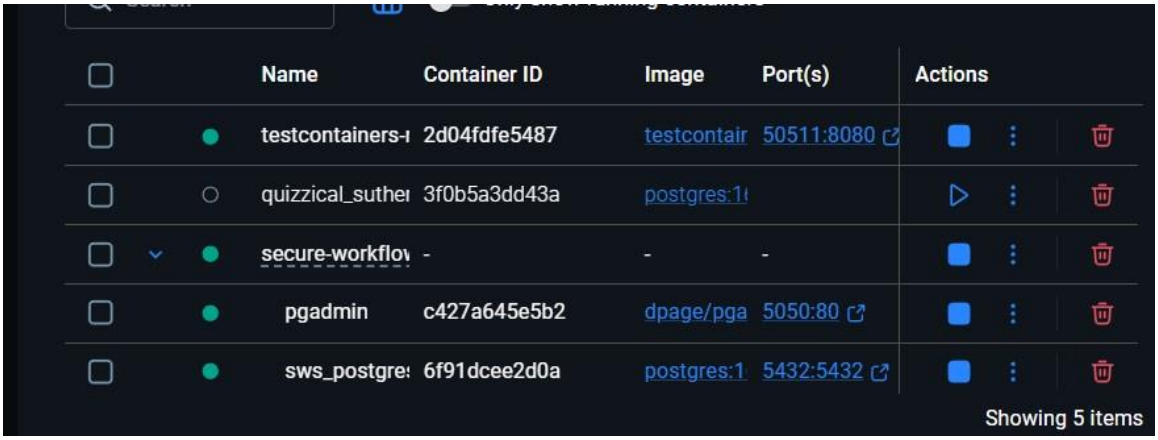
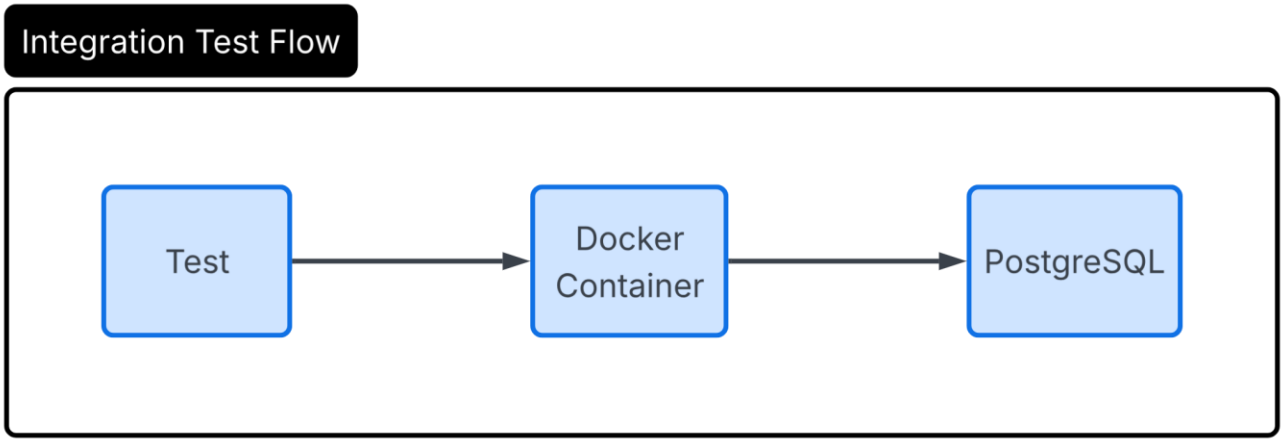


Figure 41: Docker test containers created during integration tests

The Integration Test Flow

The diagram shows the event flow during Integration Testing.



4.6 Functional and Workflow Testing

The lifecycle of a case

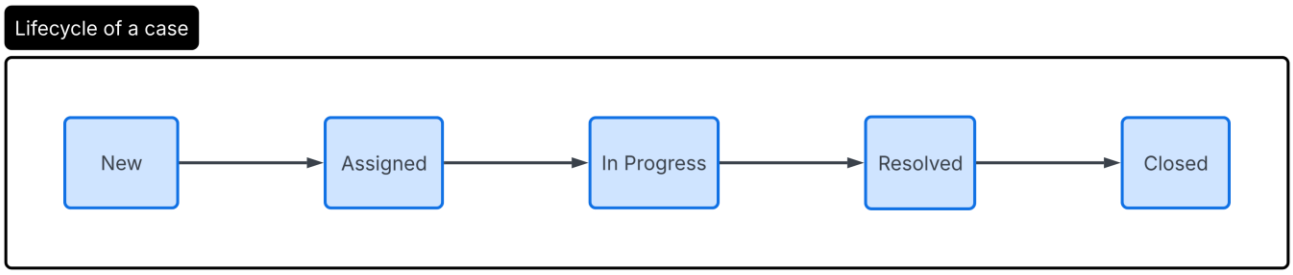


Figure 42: The lifecycle of a case

Workflow testing verified that cases moved correctly through the defined lifecycle states implemented within the Secure Workflow System. Both valid and invalid workflow transitions were tested to ensure that workflow rules were enforced consistently.

Test table

Test Case	Expected Result	Outcome
User login	User authenticated successfully	Pass
Create case	Case saved to database	Pass
Staff assignment	Case assigned correctly	Pass
Invalid transition	Transition rejected	Pass

Table 7: Workflow tests

The Lifecycle Steps of a Case from creation to closure

Case Creation Form

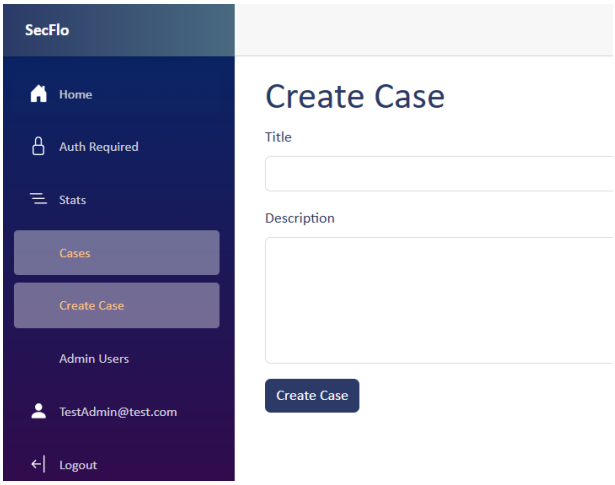


Figure 43: The Case Creation Form

Case List and Workflow Status Display

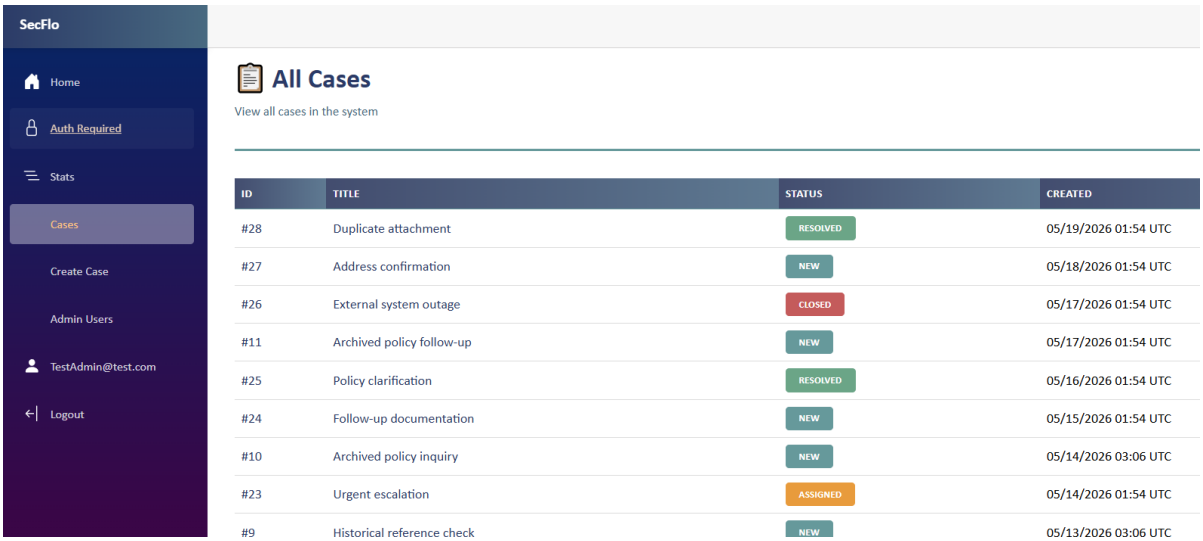


Figure 44: The All Cases List and Workflow Status Display

Case Details and Status History



Figure 45: Case Details View

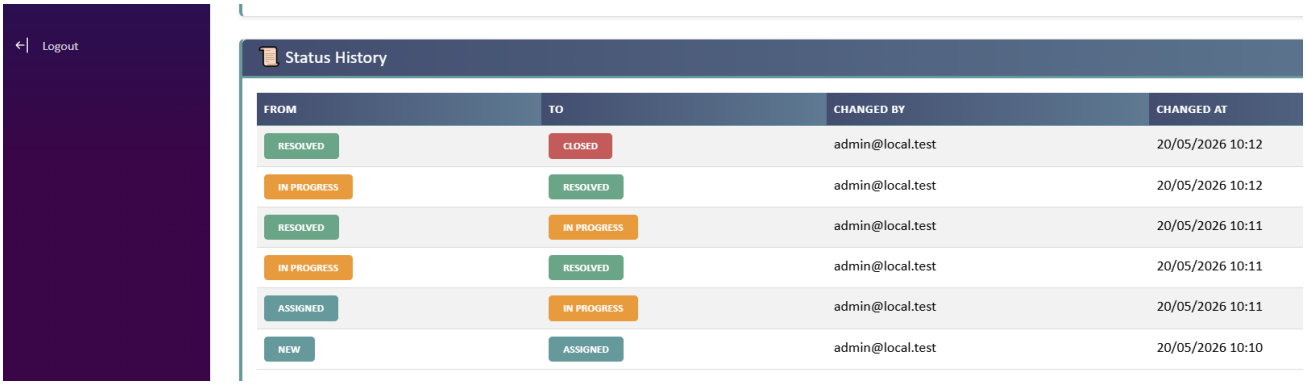


Figure 46: Case Status History View

4.6.1 Workflow Validation Testing

Validation

Current State	Attempted Transition	Expected Result	Outcome
New	Closed	Rejected	Pass
Assigned	Resolved	Rejected	Pass
InProgress	Resolved	Allowed	Pass

Table 8: Workflow Validation Testing Octcomes

4.6.2 Bunit Testing

BUnit was used to perform automated component testing of the Blazor user interface within the Secure Workflow System. These tests verified that UI components rendered correctly and responded appropriately based on authentication state and user role. Component tests were implemented for pages such as MyCases, where mocked authentication contexts and mocked application services were used to simulate standard users and staff users interacting with the system. The tests confirmed that workflow information and case data were displayed correctly, while role-specific functionality and restricted content remained appropriately controlled through RBAC policies. Using BUnit allowed Blazor components to be tested in isolation without requiring a full running web server, improving test reliability and supporting automated validation within the CI/CD pipeline.

Manual/browser testing of the deployed app

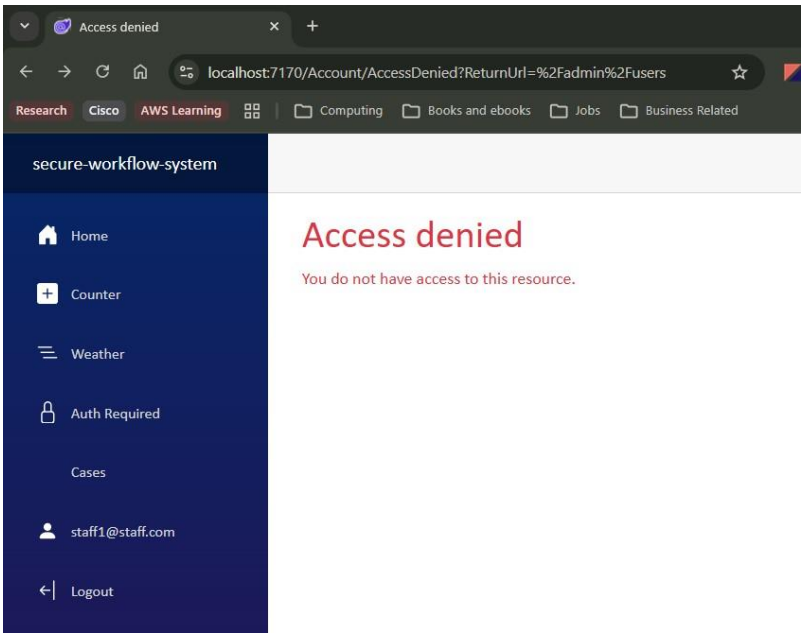
ID	Test	Steps	Expected Result	Actual Result
FT01	Register user	Complete registration form	User account created	Pass
FT02	Login	Login with valid credentials	Dashboard loads	Pass
FT03	Create case	Submit case form	Case appears in list	Pass
FT04	View case details	Open case	Details and history shown	Pass
FT05	Update status	Change case status	Status and history update	Pass

Table 9: Manual/browser testing of the deployed app

4.7 Security and RBAC Testing

Security and role-based access control testing were performed to verify that users could only access functionality appropriate to their assigned roles. Authentication and authorization behaviour were validated using multiple test accounts representing standard users, staff users, and administrators.

Access denied



HTTPS browser security

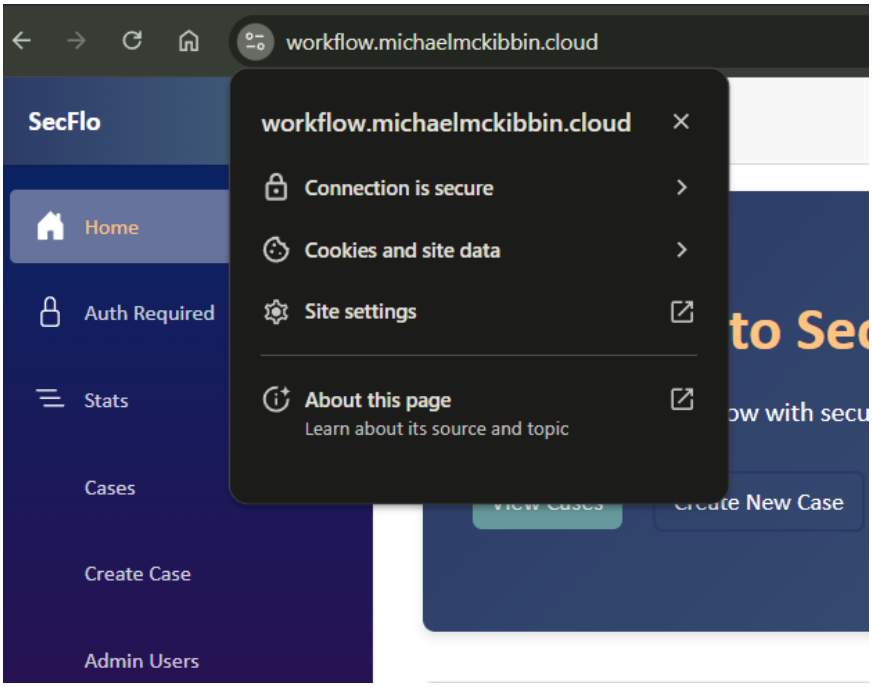


Figure 47: HTTPS browser security

Test accounts were created for each access level:

Role Management

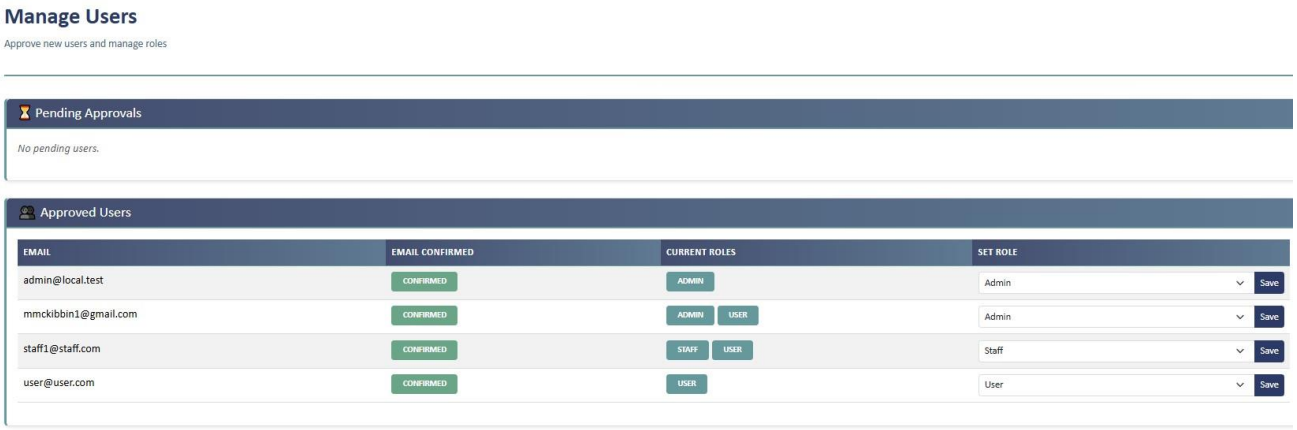


Figure 48: Role Management View

User Permissions

Feature	User	Staff	Admin
Create case	Pass	Optional/Pass	Pass
View own cases	Pass	Pass	Pass
View all cases	Denied	Pass	Pass
Assign case	Denied	Pass	Pass
Update status	Denied	Pass	Pass
Manage users	Denied	Denied	Pass

Table 10: User Permissions

4.8 CI/CD and Deployment Testing

Continuous Integration and Continuous Deployment (CI/CD) testing was used to validate the automated build, packaging, and deployment pipeline implemented for the Secure Workflow System.

GitHub Actions workflows were configured to automatically build the application, execute automated tests, publish container images to GitHub Container Registry (GHCR), and deploy updated containers to the Ubuntu VPS environment.

Deployment testing verified:

- successful application builds,
- container image publication,
- automated VPS deployment,
- Docker container startup,
- HTTPS availability,
- and health endpoint responses.

During later stages of development, the deployment pipeline was improved by transitioning from mutable image tags to immutable Docker image digests. This ensured that deployments referenced exact tested container versions and improved rollback reliability.

Deployment validation also included verification of:

- persistent database storage,
- authentication persistence,
- Data Protection key persistence,
- and successful redeployment following container recreation.

Deployment verification included confirming the running container digest on the VPS host.

Container and Endpoint Checks

Test	Expected Result	Result
Container starts	Healthy	Pass
Database reachable	Connection succeeds	Pass
HTTPS valid	TLS certificate trusted	Pass
Health endpoint	HTTP 200 OK	Pass
Persistent login	User remains after restart	Pass
Correct image digest deployed	Running container matches CI/CD artifact digest	Pass

Table 11: Container and Endpoint Checks

GitHub Actions Successful Workflow

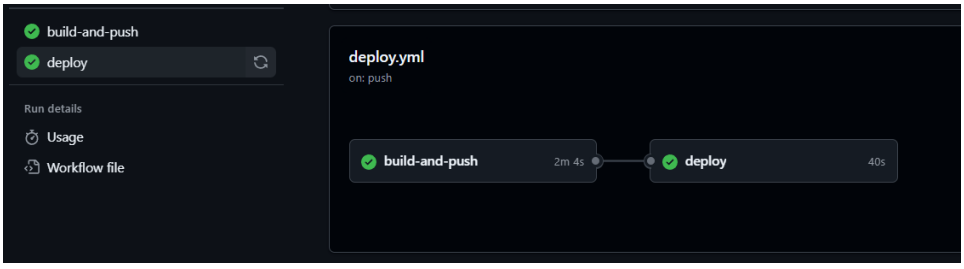


Figure 49: GitHub Actions Successful Workflow

GitHub Container Registry (GHCR) Images

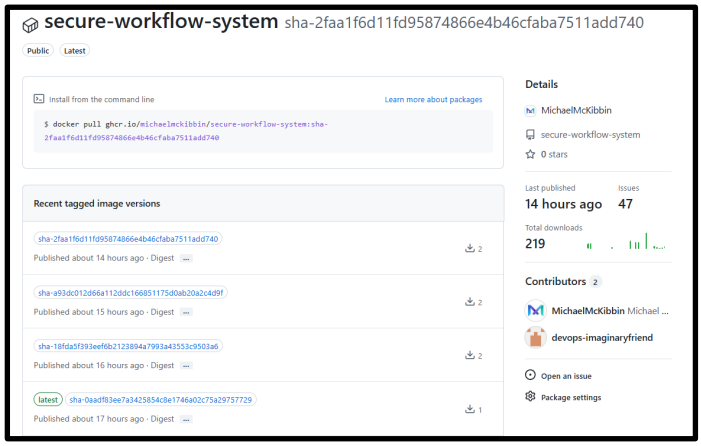


Figure 50: Docker container images published to GitHub Container Registry (GHCR)

Docker ps Output



```
root@michaelmckibbin:/docker/workflow# docker ps
CONTAINER ID   IMAGE                                          COMMAND                  CREATED        STATUS                  PORTS
   NAMES
22f973864a41   a120b437b3f1                                   "dotnet secure-workf…"   20 hours ago   Up 20 hours             0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp
   workflow-workflow-1
1776b6ed3be1   postgres:16                                    "docker-entrypoint.s…"   20 hours ago   Up 20 hours (healthy)   0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp
   workflow-postgres-1
675311e41429   ghcr.io/michaelmckibbin/michaelmckibbindotcom  "docker-entrypoint.s…"   6 weeks ago    Up 2 weeks              3000/tcp
   myportfolio-michaelmckibbin-1
63adc256cf14   ghcr.io/michaelmckibbin/a1postal:latest        "docker-entrypoint.s…"   7 weeks ago    Up 2 weeks              3000/tcp
   a1postal-a1postal-1
c316837a049a   traefik:v3.6.8                                 "/entrypoint.sh --pr…"   2 months ago   Up 10 days              0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->4
   traefik-traefik-1
```

Figure 51: Docker ps Output on VPS using Terminal

A health check is easily performed in the localhost, via browser on the Cloud deployment and in the terminal on the VPS.

Health Endpoint Verification

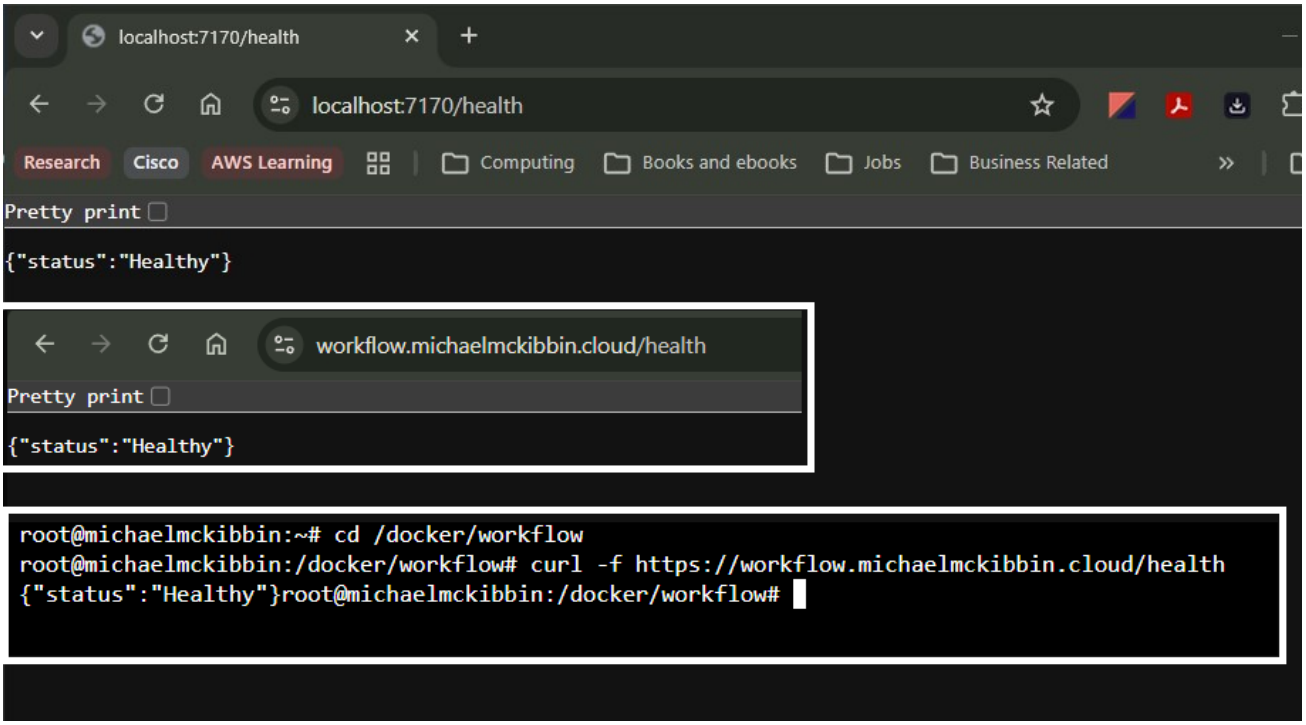


Figure 52: Health check endpoint

4.9 Negative Testing / Error Testing

Negative testing was performed to verify that invalid operations, unauthorized access attempts, and deployment edge cases were handled safely by the application.

Tests performed

Test	Expected Result	Outcome
Access protected page while logged out	Redirect to login/denied	Pass
Standard user accesses `/admin/users`	Access denied	Pass
User tries to view another user's case by URL	Denied/not found	Pass
Submit empty case form	Validation errors shown	Pass
Wrong password login	Login rejected	Pass
Restart container and login again	Session/auth still works if valid	Pass
Antiforgery token after redeploy	No persistent key error after fix	Pass

Table 12: Negative & Error Testing Results

During initial deployment testing, antiforgery validation failed after container recreation because Data Protection keys were not persisted. This was resolved by storing the key ring in a persistent Docker volume.

An invalid login attempt gives the required feedback.

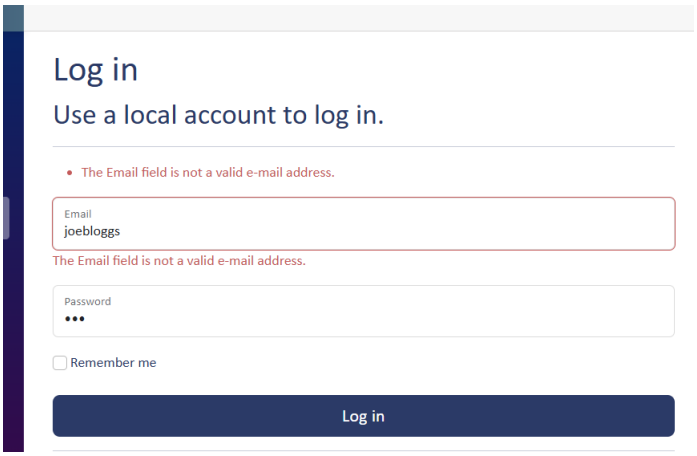


Figure 53: Invalid Login Attempt Response

Unauthorized Access Denied

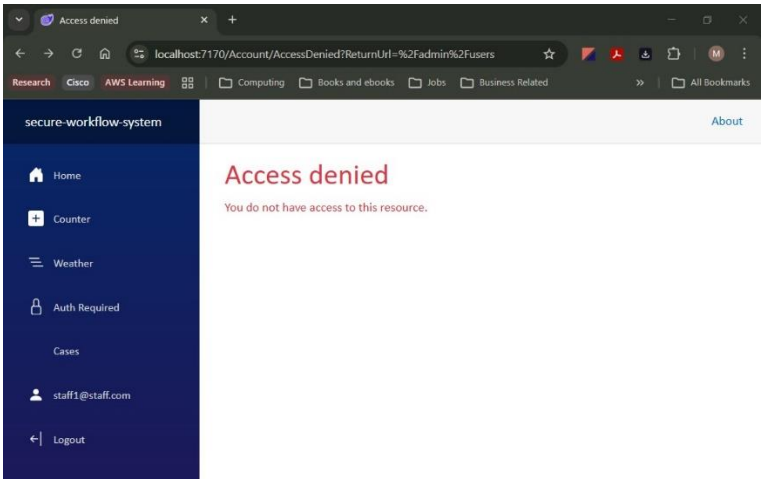


Figure 54: Unauthorized Access Denied

4.10 Troubleshooting and Defect Resolution

Several deployment and application issues were identified and resolved during development and deployment testing. Resolving these issues improved application reliability, deployment stability, authentication persistence, and user interface behaviour within the cloud-hosted environment. Table 13: : Troubleshooting and Defect Resolution Summary, summarises the most significant defects encountered during testing and the corresponding resolutions implemented.

Defect Resolution Table

Issue	Cause	Resolution
Seed users missing	Environment variable mismatch	Standardised naming
Antiforgery failure	Non-persistent keys	Docker volume persistence
GUID displayed instead of username	Missing Include()	EF eager loading
UI buttons not responding	Missing render mode	Added InteractiveServer

Table 13: : Troubleshooting and Defect Resolution Summary

Successful Redeployment After Fix

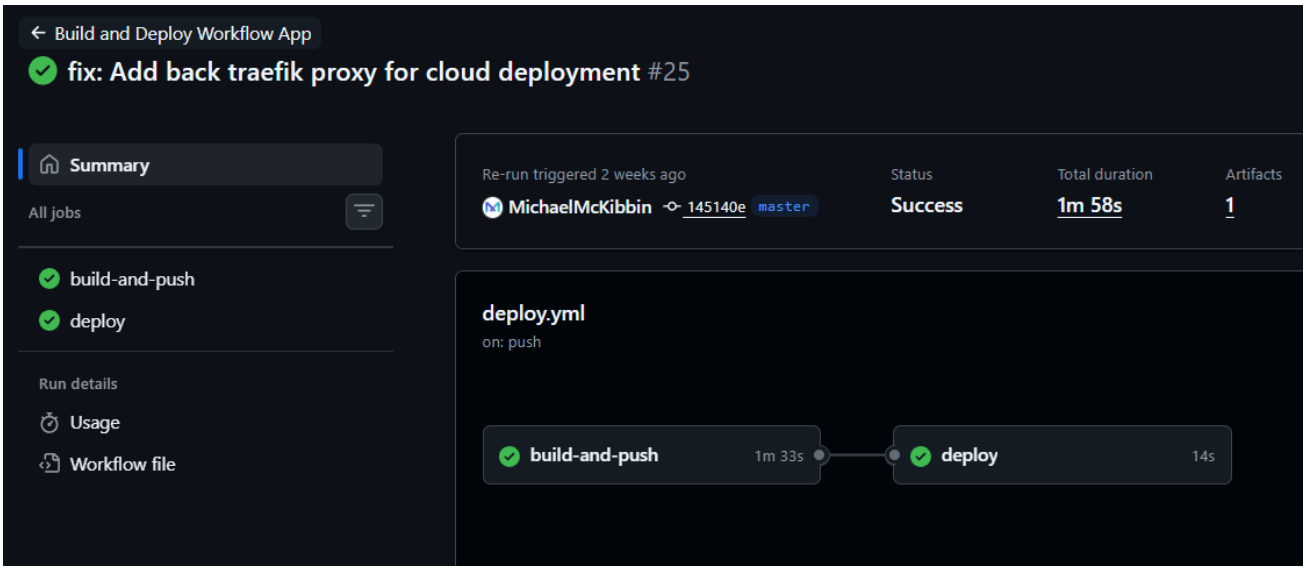


Figure 55: Successful Redeployment After Fix

4.11 Evaluation Against Objectives

The completed testing process demonstrated that the primary project objectives defined during the design phase were successfully achieved.

Evaluation vs Objectives

Objective	Achieved?	Evidence
RBAC	Yes	Admin/User/Staff roles implemented
Cloud deployment	Yes	VPS deployment with Docker

CI pipeline	Yes	GitHub Actions automated deployment
Secure authentication	Yes	ASP.NET Identity
Unit testing	Yes	xUnit test suite

Table 14: Evaluation vs Objectives

4.12 Test Results Summary

Test Results

Test Area	Number of Tests	Passed	Failed	Notes
Unit tests	40+	All	0	Core logic verified
Functional tests	10+	All	0	Browser-tested
RBAC tests	8+	All	0	Role restrictions verified
Workflow tests	10+	All	0	Status lifecycle confirmed
Deployment tests	6+	All	0	VPS deployment verified
Security/negative tests	8+	All	0	Access denial tested

Table 15: Test Results

4.13 Limitations

Although extensive functional and deployment testing was completed, some limitations remain within the current implementation.

Large-scale load testing and performance benchmarking were not performed due to time and infrastructure constraints. Automated penetration testing and advanced security auditing were also outside the scope of the project.

The current deployment architecture uses a single VPS host and does not include multi-node orchestration or horizontal scaling. Additional workflow visualisation and reporting features could also be implemented in future versions of the system.

4.14 Reflection on Testing Outcomes

Testing confirmed that the main functional, security, and deployment objectives were met. The most valuable issues found during testing related to deployment configuration, particularly image tag handling and Data Protection key persistence. Resolving these improved the reliability of the live deployment and provided stronger evidence of secure cloud operation. Some limitations remain, including the need for broader automated integration testing and stricter workflow transition validation.

4.15 Summary

Testing demonstrated that the Secure Workflow System successfully met its primary functional, security, and deployment objectives. Unit testing, workflow validation, RBAC testing, and deployment verification confirmed that the application operated reliably within both local and cloud-hosted environments.

The testing process also identified several deployment and authentication issues during development, including Data Protection key persistence and deployment image versioning. Resolving these issues improved the reliability and maintainability of the final deployed system while providing valuable practical experience in secure cloud-hosted application deployment and DevOps workflows.

Chapter 5 – Conclusions

5. Conclusions and Future Work

5.1 Project Summary

The aim of this project was to design, implement, and deploy a secure workflow management system using modern software engineering and DevOps practices. The resulting application, SecFlo, provides a containerised cloud-hosted workflow platform supporting secure authentication, role-based access control, workflow state management, and automated deployment through a CI/CD pipeline.

The project successfully combined several modern technologies into a single integrated solution, including ASP.NET Core Blazor, PostgreSQL, Docker, GitHub Actions, Traefik, and GitHub Container Registry. The final system was deployed to a live Ubuntu VPS environment using containerised infrastructure and automated deployment workflows.

Throughout development, emphasis was placed on security, deployment reliability, maintainability, and practical cloud-hosting considerations. Authentication and authorization were implemented using ASP.NET Core Identity with role-based access control policies to restrict functionality appropriately between standard users, staff users, and administrators. Workflow functionality allowed cases to progress through controlled lifecycle states while recording workflow history and user assignments.

The project also demonstrated practical DevOps principles through the implementation of automated CI/CD workflows, container image management, deployment validation, and deployment rollback procedures. Deployment reliability was improved during development through iterative troubleshooting and refinement of the cloud-hosted environment.

5.2 Achievement of Objectives

The primary objectives defined at the beginning of the project were successfully achieved.

The system successfully:

- implemented secure user authentication and role-based access control,
- supported workflow-based case management,
- provided administrative user management functionality,
- deployed successfully within a containerised cloud environment,

- integrated automated CI/CD deployment workflows,
- and demonstrated secure HTTPS communication using TLS certificates.

Testing confirmed that the system operated reliably across both local and cloud-hosted environments. Unit testing, integration testing, component testing, workflow validation, and deployment testing all demonstrated that the application behaved as expected under normal operational conditions.

The project also achieved several secondary objectives beyond the original minimum requirements, including:

- automated deployment through GitHub Actions,
- GitHub Container Registry integration,
- immutable deployment image versioning using image digests,
- deployment rollback capability,
- persistent authentication key storage,
- and structured workflow transition validation.

These additional features helped transform the project from a simple CRUD-style application into a more realistic secure cloud-hosted workflow platform.

5.3 Challenges and Lessons Learned

Several technical and operational challenges were encountered during development. Many of these issues occurred during deployment and cloud-hosting configuration rather than during initial application development.

One significant issue involved ASP.NET Core Data Protection key persistence within Docker containers. Authentication and antiforgery failures occurred following container recreation because encryption keys were stored only within temporary container storage. Persisting the key ring using Docker volumes resolved the issue and improved deployment stability.

Additional challenges included:

- environment variable mismatches during deployment,
- container image versioning inconsistencies,
- workflow deployment synchronization problems,
- and Entity Framework Core relationship loading issues.

Resolving these issues provided valuable practical experience in debugging distributed cloud-hosted applications and highlighted the importance of deployment automation, configuration consistency, and operational testing within DevOps-oriented environments.

The project also reinforced the importance of:

- iterative testing,
- infrastructure validation,
- container orchestration,
- secure configuration management,
- and automated deployment pipelines.

5.4 Limitations

Although the project successfully achieved its primary objectives, several limitations remain within the current implementation.

The system was designed primarily as a prototype workflow platform and therefore does not yet include features such as:

- large-scale performance optimisation,
- advanced audit reporting,
- external identity provider integration,
- multi-factor authentication,

- or enterprise-scale workflow orchestration.

The deployment architecture currently uses a single VPS host and does not include high-availability clustering or horizontal scaling mechanisms. Automated penetration testing and large-scale load testing were also outside the scope of the project.

Additionally, while the workflow engine supports structured state transitions, future versions could provide more advanced configurable workflow definitions and graphical workflow visualisation.

5.5 Future Work

Several areas for future enhancement were identified during development.

Potential future improvements include:

- integration with external authentication providers such as Microsoft Entra ID or OAuth/OpenID Connect,
- implementation of multi-factor authentication,
- expanded audit logging and monitoring,
- real-time notifications and email integration,
- enhanced workflow reporting dashboards,
- and configurable workflow templates.

Additional DevOps improvements could include:

- automated security scanning,
- infrastructure-as-code deployment,
- Kubernetes orchestration,
- centralized logging and monitoring,
- and automated rollback validation.

The workflow system could also be extended into a multi-tenant SaaS-style platform supporting multiple organisations and configurable workflow environments.

5.6 Final Reflection

This project provided valuable practical experience across multiple areas of modern software engineering, including secure web application development, cloud deployment, DevOps automation, containerisation, authentication systems, and workflow management design.

The combination of application development and operational deployment challenges provided insight into the full software development lifecycle beyond traditional application coding alone. In particular, the deployment and infrastructure components of the project highlighted the importance of automation, testing, configuration management, and operational reliability within modern cloud-hosted systems.

Overall, the project successfully demonstrated the design and implementation of a secure workflow management system using contemporary cloud-native development practices while also providing a strong foundation for future extension and continued development.

References

Docker, 2026. *Deployment and orchestration.* [Online]
Available at: https://docs.docker.com/guides/orchestration/

Elsevier B.V., 2026. *Workflow System.* [Online]
Available at: https://www.sciencedirect.com/topics/computer-science/workflow-system
[Accessed 14 03 2026].

ftb.ca.gov, 2009. *Business Process Management Center of Excellence Glossary.* s.l.:ftb.ca.gov.

GitHub, 2026. *GitHub Actions documentation.* [Online]
Available at: https://docs.github.com/en/actions
[Accessed 9 4 2026].

Google, 2026. *About container image digests.* [Online]
Available at: https://docs.cloud.google.com/kubernetes-engine/docs/concepts/about-container-images
[Accessed 15 5 2026].

ISO.org, 2023. *ISO/IEC 25010:2023 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model.* [Online]
Available at: https://www.iso.org/standard/78176.html
[Accessed 10 04 2026].

Kim, G. et al., 2021. *The DevOps Handbook.* Second ed. Portland, OR: IT Revolution Press, LLC.

Microsoft, 2026. *DevOps Security.* [Online]
Available at: https://learn.microsoft.com/en-us/security/benchmark/azure/mcsb-v2-devops-security
[Accessed 15 5 2026].

Mutarraf, M. U. et al., 2018. *Transformation of Business Process Model and Notation models onto Petri nets and their analysis.* [Online]
Available at: https://vbn.aau.dk/ws/files/291572941/1687814018808170.pdf
[Accessed 14 03 2026].

Peterson, J. L., 1981. *Petri Net Theory and the Modeling of Systems.* Englewood Cliffs, N.J.: Prentice-Hall.

ResearchGate, 2025. *Petri Nets - Science topic.* [Online]
Available at: https://www.researchgate.net/topic/Petri-Nets
[Accessed 13 03 2026].

Sandhu, R. C. E. F. H. a. Y. C., 1996. Role-Based Access Control Models. *IEEE Computer,* 29(2), pp. 38 - 47.

Sommerville, I., 2016. *Software Engineering.* Tenth ed. s.l.:Pearson Education Limited.

Van Den Berg, K. & Weber, R. (. O., 2019. *What's the difference of Petri Nets and Finite State Machines?.* [Online]
Available at: https://stackoverflow.com/questions/53980748/whats-the-difference-of-petri-nets-and-finite-state-machines
[Accessed 14 03 2026].

Van der Aalst, W. M. P., 2016. *Process Mining : Data Science in Action.* 2nd ed. Heidelberg: Springer.

Weske, M., 2024. *Business Process Management: Concepts, Languages, Architectures.* 4th ed. Berlin: Springer.

Wikipedia, 2025. *Workflow management system.* [Online]
Available at: https://en.wikipedia.org/wiki/Workflow_management_system
[Accessed 14 03 2026].

Wikipedia, 2026. *Petri net.* [Online]
Available at: https://en.wikipedia.org/wiki/Petri_net
[Accessed 13 03 2026].

Workflow Management Coalition, 1996. *Workflow Management Coalition Terminology & Glossary.* [Online]
Available at: https://www.aiai.ed.ac.uk/project/wfmc/ARCHIVE/DOCS/glossary/glossary.html
[Accessed 14 03 2026].

Glossary

Term	Description
API	Application Programming Interface – a method allowing software systems to communicate with each other.
ASP.NET Core	Microsoft's cross-platform framework for building web applications and services.
Authentication	The process of verifying the identity of a user.
Authorization	The process of determining what an authenticated user is permitted to access.
Blazor	A .NET web framework used to build interactive web applications using C#.
BUnit	A testing framework for Blazor UI component testing.
CI/CD	Continuous Integration / Continuous Deployment – automated software build, testing, and deployment practices.
Container	An isolated software environment used to package and run applications consistently across systems.
CRUD	Create, Read, Update, Delete – the four basic operations used in data-driven applications.
CSS	Cascading Style Sheets – used to style and format web pages.
Data Protection Keys	Encryption keys used by ASP.NET Core for authentication cookies and antiforgery tokens.
DevOps	A software engineering approach combining development and operational practices through automation and collaboration.
Docker	A containerisation platform used to package and deploy applications.
Docker Compose	A tool used to define and manage multi-container Docker applications.
EF Core	Entity Framework Core – Microsoft's object-relational mapping framework for .NET applications.
Git	A distributed version control system used to track source code changes.
GitHub Actions	GitHub's CI/CD automation platform used for build and deployment workflows.
GHCR	GitHub Container Registry – a service used to store and distribute container images.
HTTPS	HyperText Transfer Protocol Secure – encrypted communication over HTTP using TLS.
Integration Testing	Testing interactions between multiple application components or services.
JWT	JSON Web Token – a token-based authentication format commonly used in APIs.

Term	Description
ORM	Object-Relational Mapping – a technique for mapping database tables to application objects.
PostgreSQL	An open-source relational database management system used by the project.
RBAC	Role-Based Access Control – an authorization model that restricts access based on user roles.
Reverse Proxy	A server that forwards client requests to backend services while handling routing and security functions.
Role	A defined set of permissions assigned to users within the system.
SaaS	Software as a Service – software delivered through cloud-hosted platforms.
SHA	Secure Hash Algorithm – a cryptographic hashing function used for image identification and integrity verification.
TLS	Transport Layer Security – a protocol used to encrypt network communications.
Traefik	A reverse proxy and load balancer used to route HTTPS traffic to application containers.
Unit Testing	Testing individual software components in isolation.
VPS	Virtual Private Server – a virtualized cloud-hosted server environment.
Workflow	A sequence of structured process states or tasks used to manage work activities.
xUnit	A unit testing framework for .NET applications.