**Part 2 Implementation and Feedback Report**

October 24, 2025

PROG6212 DISD0601 DIS2

Student Name | ST10454956

Programming 2B

Mr. Deogratias Phiri

Table of Contents

[1. Introduction 3](#_Toc212228294)

[2. System Overview 3](#_Toc212228295)

[3. Key Functionalities Implemented 3](#_Toc212228296)

[4. Updated Architecture 3](#_Toc212228297)

[5. Database Integration 3](#_Toc212228298)

[6. Dependency Injection Implementation 4](#_Toc212228299)

[7. Unit Testing 4](#_Toc212228300)

[8. Screens and Views 5](#_Toc212228301)

[9. Version Control Summary 5](#_Toc212228302)

[10. Lecturer Feedback Implementation 5](#_Toc212228303)

[11. Conclusion 6](#_Toc212228304)

## 1. Introduction

This report presents the final implementation of the Claims Management and Verification System (CMCS) developed using ASP.NET Core MVC. Building on the prototype from Part 1, this version introduces full functionality, including claim submission, approval workflows, file uploads, database integration, and automated unit testing.

## 2. System Overview

The CMCS application streamlines the management of lecturer claims. Lecturers can submit claims with supporting documents, while Programme Coordinators and Academic Managers can verify and approve or reject these claims. The system ensures transparency, real-time tracking of claim statuses, and secure data handling.

## 3. Key Functionalities Implemented

The following functionalities were implemented for Part 2:

- \*\*Lecturer Claim Submission:\*\* Lecturers can submit claims via a simple form that captures sessions, hours, rate, and notes.  
- \*\*File Uploads:\*\* Supporting documents (.pdf, .docx, .xlsx) can be attached securely with file validation.  
- \*\*Verification and Approval:\*\* Programme Coordinators and Academic Managers can review, approve, or reject claims.  
- \*\*Status Tracking:\*\* Claim status updates are reflected instantly as 'Pending', 'Approved', or 'Rejected'.  
- \*\*Error Handling:\*\* User-friendly error messages and exception handling improve reliability.  
- \*\*Unit Testing:\*\* Core functionalities are tested using xUnit and Moq.  
- \*\*Version Control:\*\* Project committed to GitHub with five descriptive commits as required.

## 4. Updated Architecture

The application adheres to the Model-View-Controller (MVC) architecture. The static singleton pattern previously used in DbHelper has been replaced with dependency injection through the IDbHelper interface, improving testability and scalability. Database initialization is now synchronous to ensure the database is available before serving user requests.

## 5. Database Integration

The application uses SQL LocalDB, automatically initialized at startup. The connection string is stored securely in appsettings.json, and the database helper (DbHelper) manages all CRUD operations asynchronously.

## 6. Dependency Injection Implementation

builder.Services.AddSingleton<IDbHelper, DbHelper>();  
  
public class ClaimsController: Controller  
{  
 private readonly IWebHostEnvironment \_env;  
 private readonly IDbHelper \_db;  
  
 public ClaimsController(IWebHostEnvironment env, IDbHelper db)  
 {  
 \_env = env;  
 \_db = db;  
 }  
}

This change removes the need for static references (DbHelper.Instance) and allows easier mocking in unit tests.

## 7. Unit Testing

xUnit tests were implemented to verify the behavior of the ClaimsController. Using the Moq framework, IDbHelper was mocked to simulate database interactions. The tests verify that claims are created successfully, invalid models return errors, and unauthorized access is blocked.

[Fact]  
public async Task New\_Post\_ValidClaim\_RedirectsToIndex()  
{  
 var dbMock = new Mock<IDbHelper>();  
 dbMock.Setup(d => d.CreateClaimAsync(It.IsAny<Claim>())).ReturnsAsync(1);  
  
 var envMock = new Mock<IWebHostEnvironment>();  
 var controller = new ClaimsController(envMock.Object, dbMock.Object);  
  
 var claim = new Claim { number\_of\_sessions = 2, number\_of\_hours = 3, amount\_of\_rate = 100, module\_name = "Math", faculty\_name = "Science" };  
  
 var result = await controller.New(claim, null);  
 var redirect = Assert.IsType<RedirectToActionResult>(result);  
 Assert.Equal("Index", redirect.ActionName);  
}

## 8. Screens and Views

The graphical user interface remains simple and intuitive. Key views include:  
- Lecturer claim submission form  
- Coordinator verification dashboard  
- Manager approval panel  
- Claim status tracking page

## 9. Version Control Summary

The project was committed to GitHub with clear messages as per the assessment guidelines:  
1. feat: implemented IDbHelper and dependency injection  
2. fix: corrected DbHelper data retrieval logic  
3. test: added xUnit tests for controller validation  
4. style: improved UI and user flow  
5. docs: updated documentation for Part 2

## 10. Lecturer Feedback Implementation

**Lecturer Feedback**

| **Feedback** | **Implementation** |
| --- | --- |
| **“Add buttons to reject/accept.”** | The verification and approval modules were updated to include **Approve** and **Reject** buttons for Programme Coordinators and Academic Managers. Each claim in the pending claims view now displays these buttons clearly. Selecting either option triggers an update to the claim\_status in the database using the UpdateClaimStatusAsync() method in DbHelper. This ensures real-time updates and transparent tracking. |
| **Improve design clarity and layout.** | The GUI layout was refined with consistent spacing, color themes, and intuitive navigation. Buttons for submission, upload, and approval were made prominent. The overall layout adheres to responsive design principles, ensuring accessibility and user-friendliness. |
| **Enhance documentation depth for design rationale.** | The Part 2 documentation now includes detailed explanations for all design choices, including database schema, dependency injection for scalability, and MVC structure for maintainability. Each decision is supported with logical reasoning and alignment to system requirements. |
| **Add unit testing for reliability.** | xUnit tests were developed to validate claim creation, approval workflows, and data consistency. Mocked dependencies (using Moq) allow testing without connecting to the database. This demonstrates reliability and professional testing standards. |
| **Improve project planning details.** | The project plan was refined with explicit task dependencies, achievable timelines, and milestone checkpoints (e.g., “Implement DB Integration”, “Add File Upload”, “Testing & Validation”). This ensured timely completion and realistic progress tracking. |
| **Ensure version control compliance.** | The project includes at least five commits with descriptive messages (e.g., feat: implement IDbHelper and DI, fix: corrected DbHelper logic, test: add xUnit tests, style: improved UI, docs: updated Part 2 documentation). This meets version control criteria and reflects consistent development progress. |

**Summary of Implementation Impact**

The feedback directly informed system enhancements that increased usability, testability, and professional completeness. The inclusion of approval/rejection controls, improved documentation clarity, structured commits, and validated test coverage collectively strengthened the project to meet and exceed assessment standards.

## 11. Conclusion

The Part 2 implementation of the CMCS demonstrates a fully functional, well-structured application that adheres to modern development standards. It provides a reliable, testable, and user-friendly system for claim management within an academic environment.