POE Part 1

PROG 6212

ST10448858
N. MUKADDAM

Table of Contents

Report:

Contract Monthly Claim System (CMCS)

1. Introduction

The Contract Monthly Claim System (CMCS) is a web-based application designed to streamline the monthly claim submission and approval process for Independent Contractor (IC) lecturers. This system eliminates paper-based workflows by providing a single, integrated platform where lecturers submit claims, upload supporting documents, and track approval status. Programme Coordinators and Academic Managers use the same platform to verify and approve claims. This report outlines the initial planning, system design, and database modelling decisions for the CMCS prototype.

2. Purpose of the Prototype

This first stage focuses on creating a non-functional prototype using .NET Core MVC (Microsoft, 2025), Razor Views (Microsoft, 2025), and UML diagrams. At this stage the GUI serves as a visual representation of the system but does not yet implement any business logic or database storage. The purpose is to:

- Plan the system's structure and data model.
- Design a user-friendly interface for lecturers, coordinators, and managers.
- Build a clear roadmap for future development stages.

3. Design Decisions and Assumptions

3.1 System Architecture

We have chosen the Model-View-Controller (MVC) pattern, which is well supported in .NET Core (Microsoft, 2025). This structure separates the application into:

- Models: Data structures (Lecturers, Claims, Supporting Documents).
- Views: Razor pages that present information to users. (Microsoft, 2025)
- Controllers: Components that will later handle requests and apply business logic.

This approach ensures scalability, maintainability, and a smooth transition to a fully functional application later.

3.2 User Roles and Permissions

- Lecturers: Submit claims, upload documents, track claim status.
- Programme Coordinators: Verify submitted claims, return them for correction if needed.
- Academic Managers: Final approval of claims.

Each role will have its own dashboard and restricted access to certain functions.

3.3 Key Assumptions

- All lecturers already exist in the system with hourly rates recorded.

- Claims are monthly and tied to one lecturer but can contain multiple claim items.
- Authentication and authorisation will be implemented later, not at the prototype stage.
- Upload file types limited to PDF, JPG, PNG, maximum 10 MB.

3.4 Constraints

- This phase is front-end only – no back-end logic or database connections.
- Razor Views are placeholders and may display mock data.
- The system will evolve iteratively as per the POE requirements.

4. Database and UML Class Diagram

We modelled the system using several key entities. The UML Class Diagram and ERD are closely aligned. At the prototype stage, these structures serve as a blueprint for future database implementation.

Entities and relationships are detailed in Section 6 below.

5. GUI / User Interface Design

5.1 Lecturer View

- "Submit Claim" button with a form for claim details.
- File upload area for supporting documents.
- Table showing claim status (Draft, Submitted, Verified, Approved, Returned).

5.2 Programme Coordinator View

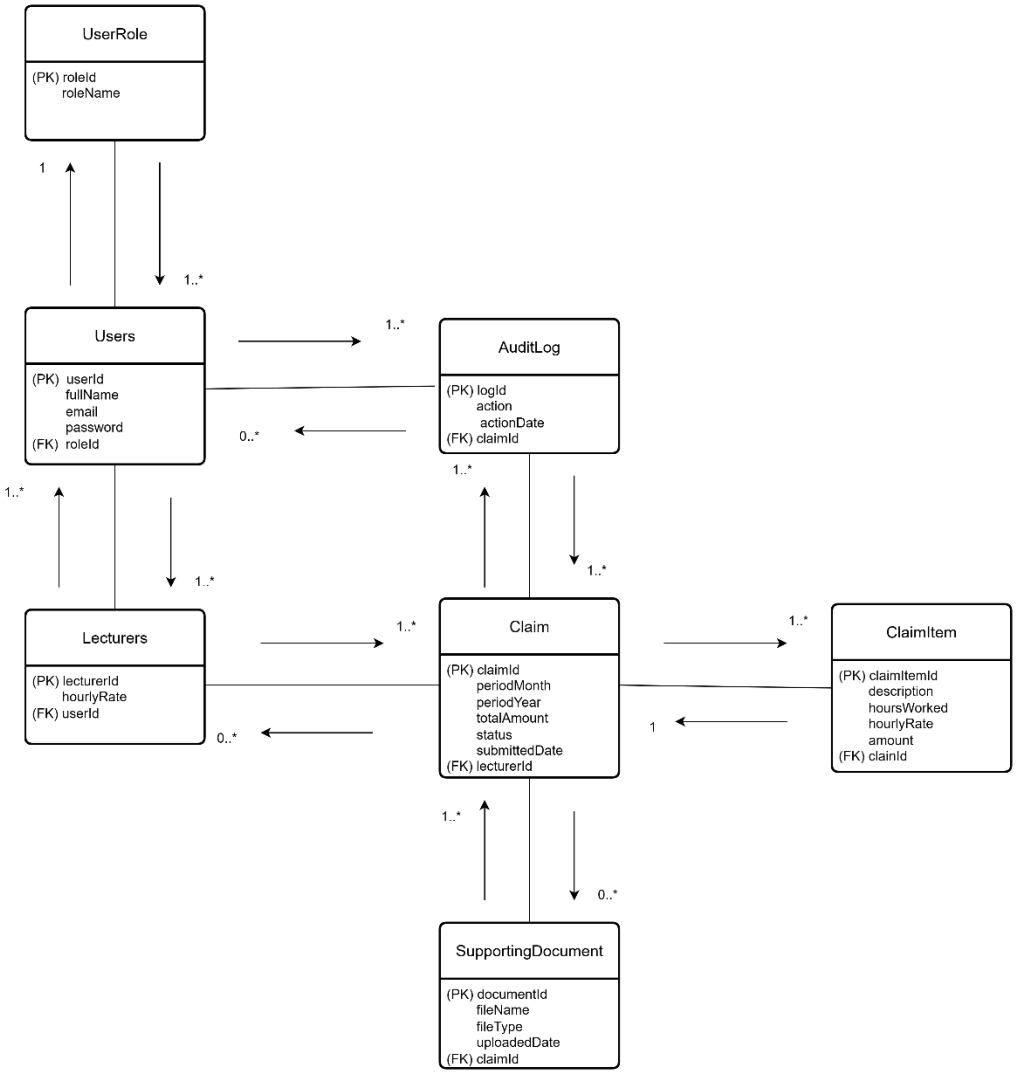
- Queue of claims awaiting verification.
- Ability to view claim details, supporting documents, and verify or return.

5.3 Academic Manager View

- List of verified claims ready for approval.
- Approval/rejection actions and audit history.

The GUI is visually simple but shows all required options.

6. UML/ Entity Relationship Diagram



(Draw.io, 2025)

Relationships:

- UserRole (1) → Users (Many)
- User (1) → Lecturer (0 or 1)
- Lecturer (1) → Claims (Many)
- Claim (1) → ClaimItems (Many)
- Claim (1) → SupportingDocuments (Many)
- Claim (1) → AuditLog (Many)
- User (1) → AuditLog (Many)

7. Project Plan

Task ID	Task Description	Duration (Days)	Dependencies
T1	Analyse requirements and outline prototype scope	2	—
T2	Draft detailed report (design choices, assumptions, database structure)	3	T1
T3	Develop UML/ERD in Draw.io (based on attributes above)	2	T2
T4	Create GUI wireframes and Razor View placeholders	5	T3
T5	Setup GitHub repository and push initial commit	1	T1
T6	Review prototype, update diagrams and documentation	2	T4
T7	Finalise Part 1 package (report + diagrams + repo)	1	T6

Timeline Suggestion (4 Weeks):

- Week 1: T1, T2
- Week 2: T3, T5
- Week 3: T4
- Week 4: T6, T7

8. Conclusion

This report presents the foundation of the CMCS prototype, including the rationale for using .NET Core MVC (Microsoft, 2025), the proposed database model, GUI layout, and project plan. By completing this phase, you establish a clear blueprint for adding functionality in subsequent stages. The modular approach ensures scalability, traceability, and adherence to good software development practices. Open AI was used for the correction of punctuation errors and layout (OpenAI, 2025).

References

Draw.io, 2025. *Draw.io*. [Online]
Available at: https://app.diagrams.net/

Microsoft, 2025. *MicrosoftLearn*. [Online]
Available at: https://learn.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/getting-started
[Accessed September 2025].

Microsoft, 2025. *MicrosoftLearn*. [Online]
Available at: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-9.0
[Accessed September 2025].

OpenAI, 2025. *ChatGPT*. [Online]
Available at: https://chat.openai.com/c/c7b7dc60-a206-406b-911b-96d0c7066eee
[Accessed 2025].