Abongile Mazongolo

ST10043271

PROG6212 POE document

Contents

[Link to My GitHub Repository Part 2 2](#_Toc180157037)

[Contract Monthly Claim System (CMCS) Report 3](#_Toc180157038)

[Design Choices 3](#_Toc180157039)

[Structure of the Database 3](#_Toc180157040)

[Layout of The GUI Design 4](#_Toc180157041)

[Screenshots of The Layout of My GUI 5](#_Toc180157042)

[UML Class Diagram 8](#_Toc180157043)

[Project Plan 9](#_Toc180157044)

[Screenshot of Commits 9](#_Toc180157045)

[Conclusion 10](#_Toc180157046)

[Lecturer Feedback 10](#_Toc180157047)

Part 1 — Project Planning and Prototype Development

## Link to My GitHub Repository Part 2

https://github.com/st10043271/CMCS\_Part2.git

# Contract Monthly Claim System (CMCS) Report

The Contract Monthly Claim System (CMCS) is designed to streamline the submission, verification, and approval of claims by Independent Contractor (IC) lecturers. The system focuses on simplifying administrative tasks, improving user satisfaction, and reducing errors in processing claims. The CMCS prototype uses WPF (.NET Core) to provide a visually appealing, user-friendly interface. This report covers the design choices, database structure, UML class diagram, and the GUI layout of the system, as well as the assumptions and constraints considered during development.

## Design Choices

The primary goal was to design a system that is both intuitive and scalable, ensuring ease of use for lecturers, programme coordinators, and academic managers.

* Platform: I chose WPF (.NET Core) mainly because it is a platform I am familiar with I also chose it for its rich graphical capabilities and ease of integration with backend databases, which allows for a responsive and modern UI.
* Architecture: The system uses the Model-View-Controller (MVC) design pattern to separate concerns. This enhances code maintainability and allows independent updates to the business logic, database interactions, and UI.
* User Roles: Three user roles were defined—Lecturers, Programme Coordinators, and Academic Managers. Each role has specific tasks within the system, such as claim submission, claim verification, and claim approval/rejection.
* User Experience: The system is designed with a focus on simplicity. The lecturer's view includes a form for submitting claims, while coordinators and managers have access to more complex features, such as claim verification and approval workflows. This separation ensures that each user is presented with only the features relevant to their role.

## Structure of the Database

The database is structured around key entities that represent the core functions of the system. These entities include Lecturer, Claim, ProgrammeCoordinator, AcademicManager, and Document.

* Lecturer: Stores basic lecturer information like lecturerID, firstName, lastName, email, and phoneNumber.
* Claim: Represents claims submitted by lecturers, with attributes like claimID, lecturerID, hoursWorked, hourlyRate, status, submissionDate, and reviewDate.
* Document: Stores paths to supporting documents uploaded by lecturers. This allows claims to be accompanied by necessary files, such as PDF invoices.
* ProgrammeCoordinator and AcademicManager: Handle claim verification and approval, with attributes for tracking coordinators' and managers' actions on claims.

Each entity is related logically, with one-to-many relationships between Lecturer and Claim, Claim and Document, and ProgrammeCoordinator/AcademicManager and Claim. This structure ensures efficient querying and data retrieval.

## Layout of The GUI Design

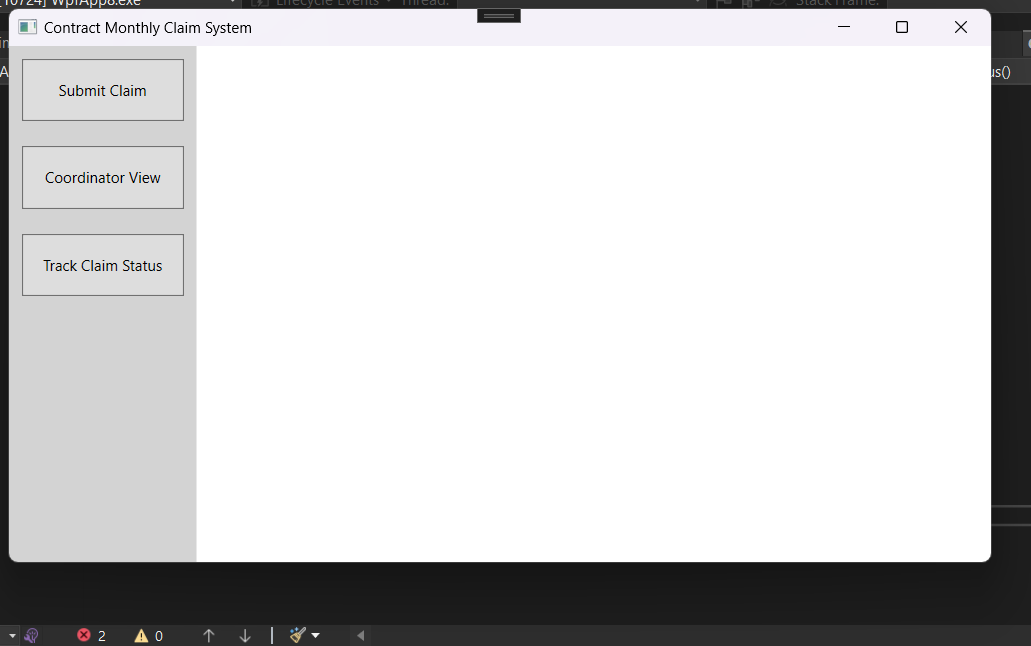
The GUI consists of three main views, tailored to each user role:

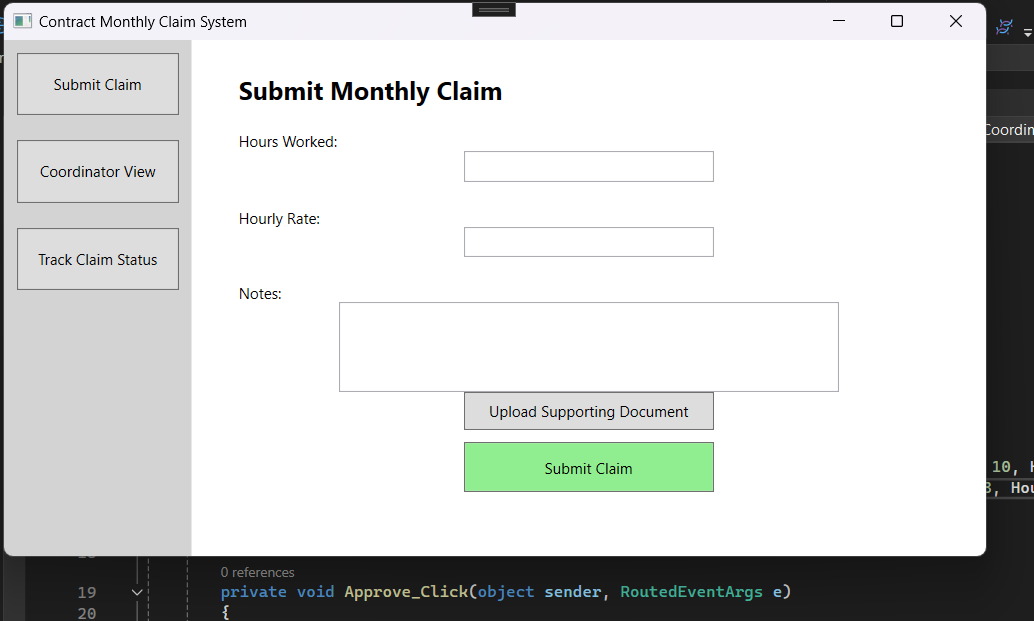
Lecturer’s View: The lecturer's user interface is a simple form for submitting claims, including fields for hoursWorked, hourlyRate, and notes. A button allows lecturers to upload documents (e.g., PDFs), and another button submits the claim for review. The focus here is on simplicity, ensuring lecturers can submit claims with minimal effort.

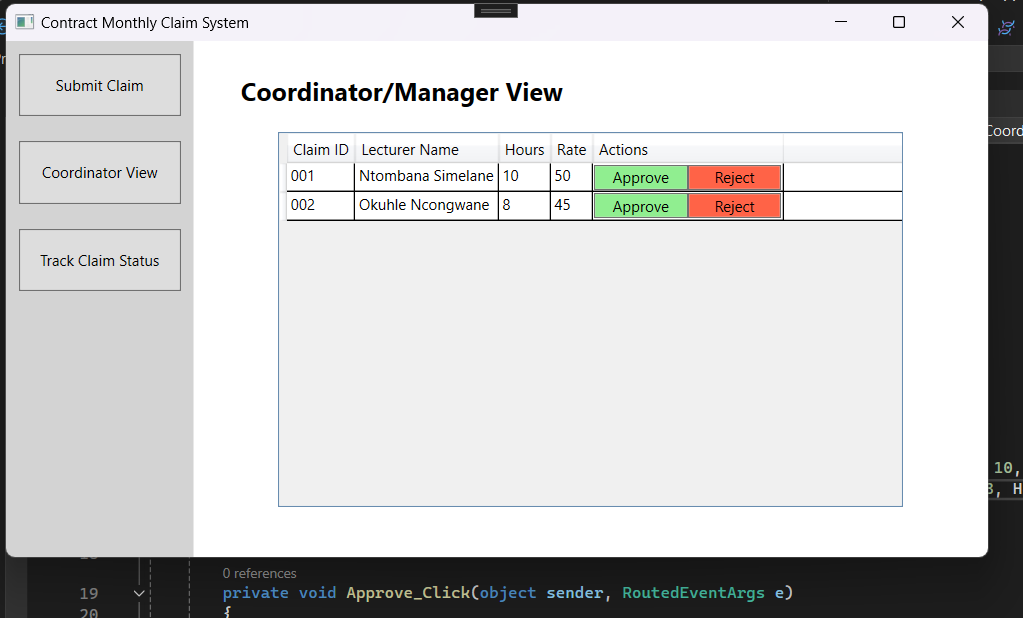
Coordinator/Manager View: This view is more complex and contains a table displaying all pending claims. Each row includes details such as lecturerName, hoursWorked, and hourlyRate, with Approve and Reject buttons for each claim. This layout ensures that programme coordinators and academic managers can easily verify claims at scale.

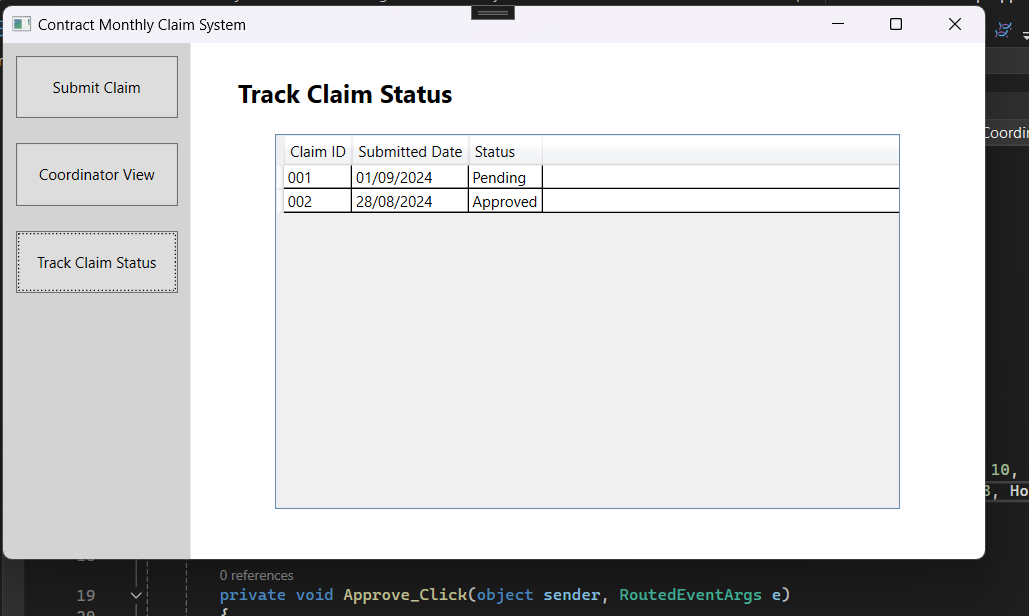
Claim Status View: A progress bar and table format allow lecturers to track the status of their submitted claims. This view provides real-time updates on whether the claim is Pending, Approved, or Rejected.

## Screenshots of The Layout of My GUI



****

****

****

**Assumptions and Constraints**

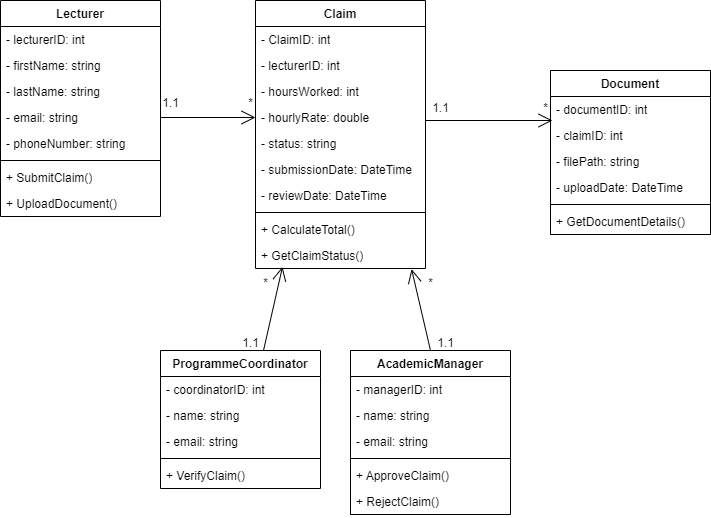
Assumptions:

* Each user has a defined role (Lecturer, Programme Coordinator, or Academic Manager).
* The lecturers are responsible for submitting claims with basic information like hours worked and supporting documents.
* Claims follow a simple approval workflow, moving from Pending to either Approved or Rejected.

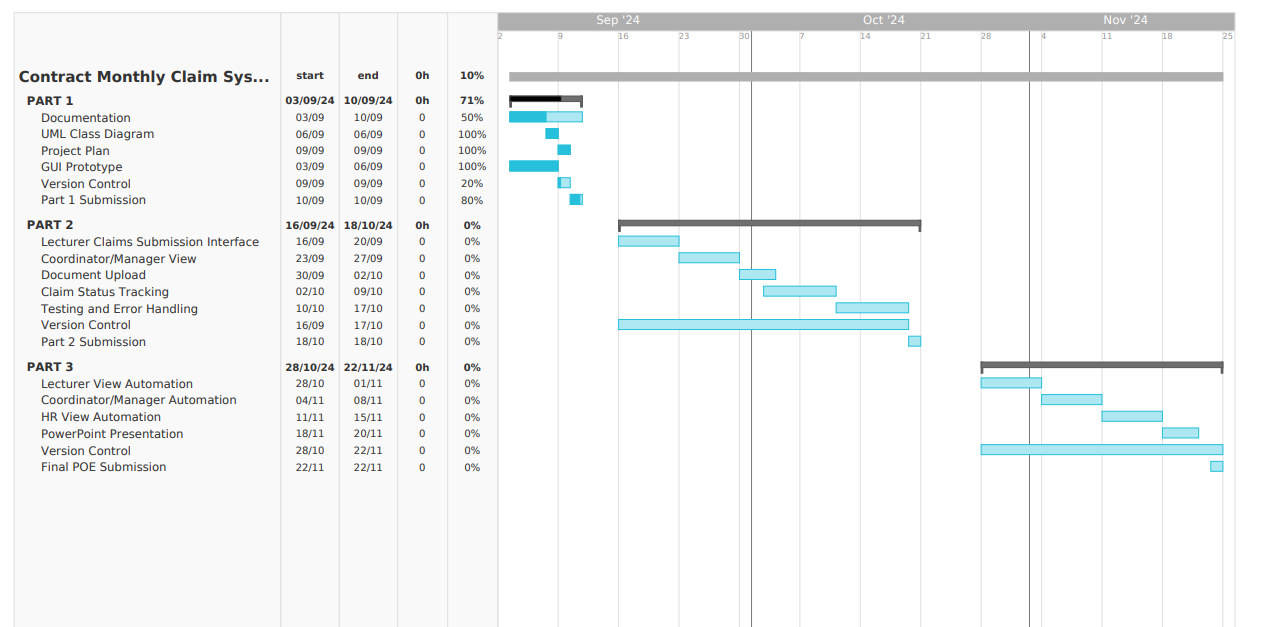
Constraints:

* The system currently does not include complex financial calculations or payroll integration. The focus is on creating a non-functional prototype with just the user interface.
* File uploads are limited to certain document types (PDF, DOCX) for security reasons.
* The system assumes a local or cloud-based SQL database for storing claims and user data.

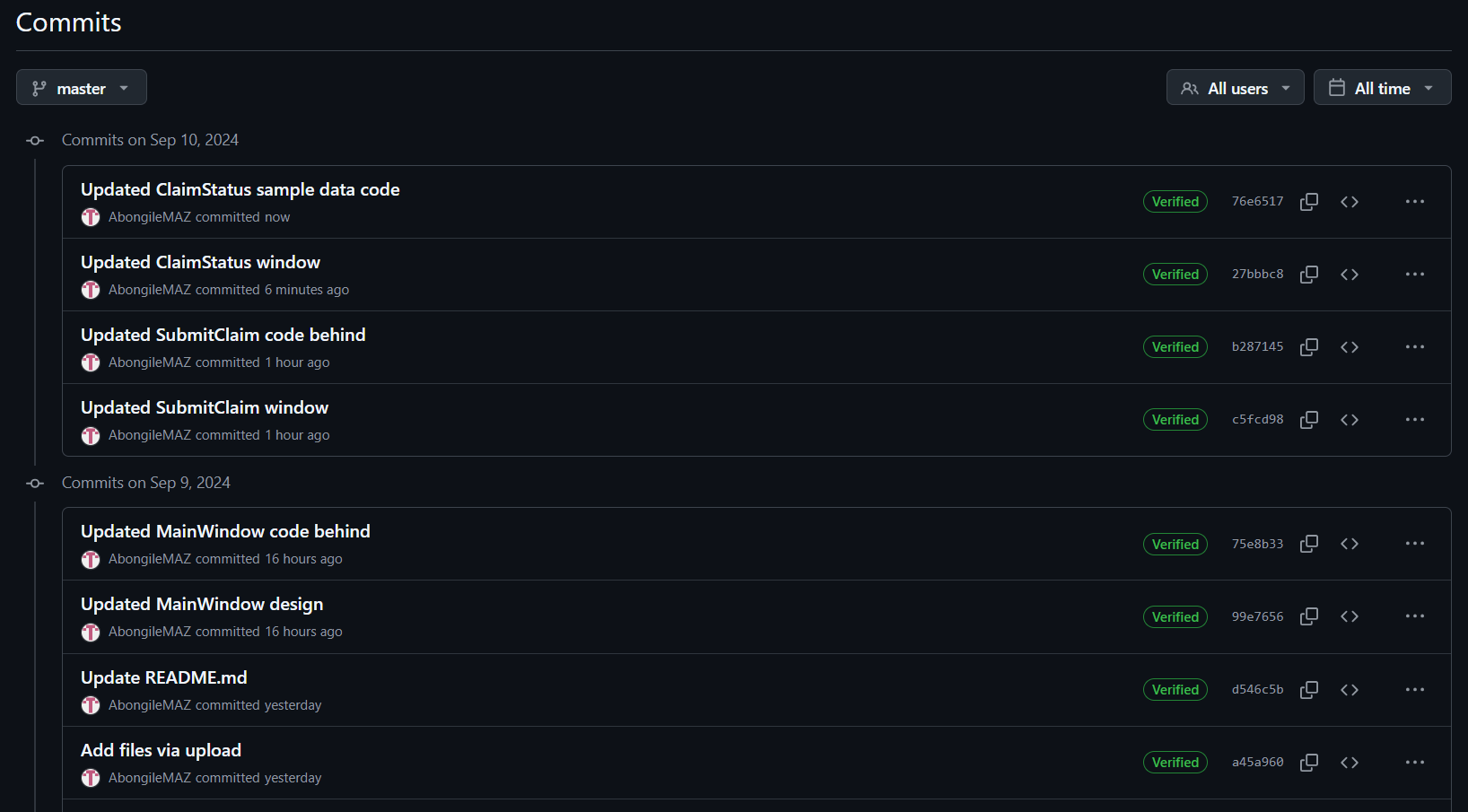
## UML Class Diagram



## Project Plan



## Screenshot of Commits



## Conclusion

The Contract Monthly Claim System (CMCS) has been designed to streamline the submission, verification, and approval process for lecturer claims while ensuring a user-friendly experience for all roles involved. The system's architecture, built using WPF (.NET Core) allows for clean separation of concerns, ensuring maintainability and future scalability. The database structure is organized to optimize data integrity and querying efficiency, with clear relationships between Lecturers, Claims, Programme Coordinators, Academic Managers, and Documents.

The design choices, including the modular GUI and role-specific functionality, focus on creating an intuitive interface for each user, while the use of a UML class diagram clearly defines the relationships and interactions between the core entities. Assumptions such as user roles and constraints like limiting the scope to non-functional prototypes in this stage, were carefully considered to create a functional and maintainable foundation for the system.

Ultimately, the CMCS provides a practical, scalable solution to streamline administrative tasks and improve efficiency, paving the way for real-world implementation in future phases of development.

# Lecturer Feedback

Lecturer’s feedback was for me to provide GitHub link of which I included on the page 2.