Part 1 — Project Planning and Prototype Development

Aneeqah Fisher

St10454674

1. Documentation: Design Choices and Structure

The CMCS has been kept simple, scalable, and usable by keeping simplicity, scalability, and usability at the forefront of the design process. The system will include three main user roles: Lecturer, Programme Coordinator, and Academic Manager.

Database Design: The database follows a relational normal structure to avoid redundancy. Core entities are:

Lecturer (LecturerID, Name, Surname, ContactDetails

Claim (ClaimID, LecturerID, HoursWorked, HourlyRate, TotalAmount, Status, SubmissionDate)

Document (DocumentID, ClaimID, FilePath, FileType)

UserRoles (RoleID, RoleName, Permissions).

Relationships are one-to-many between Lecturer → Claim and Claim → Document, to properly associate data.

GUI Layout: Simple WPF proof-of-concept with navigation menu and dashboards based on roles.

Lecturer View: Form to submit claim, upload supporting documents, track status of claim.

Coordinator View: Display pending claims, with "Approve/Reject" buttons.

Manager View: Verification overview and reporting facility.

The interface emphasizes usability highly, with good functional grouping, minimal clutter simplicity, and consistent feedback messages.

Design Decisions: WPF is selected for its UX flexibility, strong binding support (MVVM), and native .NET Core integration. Database design allows for future scalability (e.g., HR extension or automated payroll integration).

2. Assumptions and Constraints

Internal use within the institution by only authorized users.

Claims are entered month by month; partial entry during the middle of the month is not allowed.

Uploads will be limited to PDF, DOCX, XLSX under 10MB for security and storage management.

Internet connection is assumed to be stable to support cloud-hosted deployment.

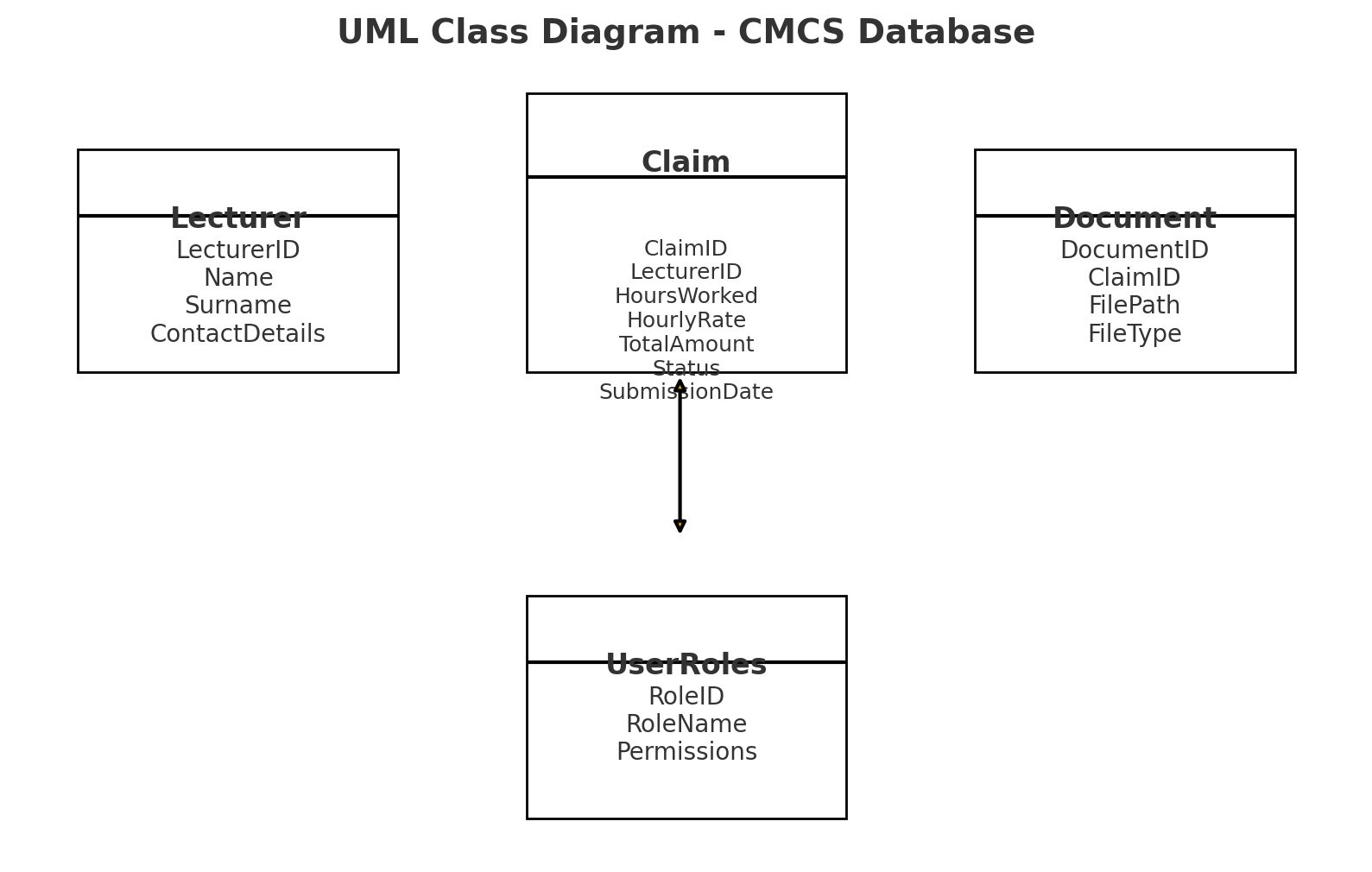
Restrictions: Development is limited to .NET Core, WPF, and SQL Server, as stipulated in module scope.

3. Databases UML Class Diagram

Diagram Structure:

Lecturer 1──∞ Claim 1──∞ Document

Claim ∞──1 UserRoles (approval handled using roles).



4. Project Plan

| **Task** | **Description** | **Duration** | **Dependency** |
| --- | --- | --- | --- |
| Requirements Analysis | Identify data requirements & role functions | 3 days | – |
| Database Design | Create UML + schema | 4 days | Requirements |
| GUI Wireframes | Create low-fidelity prototypes for each role | 5 days | Requirements |
| Prototype Development | Build WPF front-end with non-functional buttons/forms | 7 days | Database + GUI wireframes |
| Documentation | Write report + finalize assumptions | 3 days | Parallel |
| Version Control | 5 GitHub commits with descriptive logs | Ongoing | All stages |

5. GUI Design (Prototype Only)

Lecturer Dashboard: Field to enter hours, hourly rate, upload files, and submit claims.

Coordinator Dashboard: Outstanding claims table with information and approve/reject buttons.

Manager Dashboard: High-level overview, claim history, and approval reports.

All the screens share one common idea: plain buttons, structured layouts, and status feedback (pending/approved/rejected).

6. Version Control

We will maintain a GitHub repository with at least five commits:

Installation of the initial project (.NET Core + WPF template).

UML diagram

Lecturer GUI

CoordinatorGUI

Manager GUI

documentation upload.

Commit messages will be descriptive, informative, and to the point, following best practices.

Reference

Troelsen, A. and Japikse, P. (2023) Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. 11th ed. Apress.