**Ntando Nxumalo**

**ST10456704**

**PROG6212**

**PROG POE PART 1**

**REPORT**

PART 1 Documentation

**1. Overview**

The Contract Monthly Claim System (CMCS) is a System that is there to make life easier for independent contractor lecturers by simplifying how they submit, approve, and manage monthly claims. For this first phase, I’m focusing on laying the groundwork—planning the project and building a prototype using the MVC design pattern in ASP.NET Core. This document covers my design decisions, database setup, assumptions, project timeline, and the initial GUI mockups.

**2.Objective**

The Contract Monthly Claim System (CMCS) aims to make life easier for lecturers and admins by:

* Simplifying how claims are submitted and approved, cutting down on hassle.
* Keeping everyone in the loop with clear, trackable claim statuses from start to finish.
* Boosting efficiency with role-based access, so lecturers and admins can focus on their tasks without extra headaches.

**3. Roles**

The system involves three main user roles:

* **Lecturer**: Submits monthly claims and uploads supporting documents.
* **Programme Coordinator**: Reviews submitted claims and either approves or rejects them.
* **Academic Manager**: Provides oversight, final approvals, and access to claim summaries.

**4. Assumptions and Limitations**

**Assumptions**:

* Lecturers are already registered and can log in to submit claims.
* HR sets the hourly rates, so lecturers don’t have to input them manually.

**Limitations**:

* Only PDF, DOCX, and XLSX files are allowed for document uploads.
* For now, the prototype is just the front-end—there’s no database connection yet.

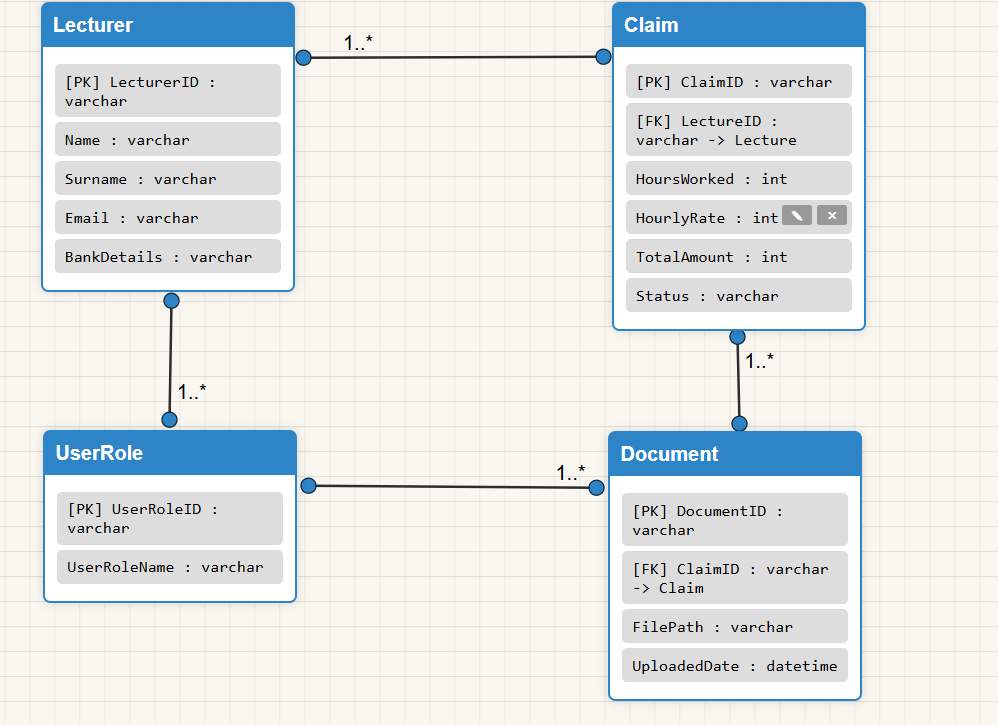
**5. System Workflow**

1. A **Lecturer** submits a claim form with hours worked and uploads documents.
2. The **Claim** enters a “Pending” state.
3. The **Coordinator** reviews the claim and either approves or rejects it.
4. Approved claims move forward to the **Manager** for final validation and tracking.
5. Documents remain linked to each claim for reference and auditing.

This workflow ensures accountability, consistency, and traceability throughout the claim process.

**6. UML Class Diagram**

The Contract Monthly Claim System (CMCS) is a user-friendly database design for managing lecturers' claims. It has four main components: Lecturers (who file claims), Claims (the payment requests), Documents (files supporting each claim), and User Roles (like Lecturer, Coordinator, or Manager to control access). A lecturer can submit multiple claims, each with several documents attached. Roles are assigned to lecturers to manage permissions, with one role covering multiple people. The system stores key details for each component and links them to keep data organized, secure, and consistent. It’s built to scale easily, ensuring smooth and accurate claim processing for everyone involved.



**Relationship**

* One **Lecturer** can make many **Claims** (1..\*).
* One **Claim** can have many **Documents** (1..\*).
* One **UserRole** can be assigned to many **Lecturers** (1..\*).

**7. Requirement Gathering**

The following requirements were identified for the prototype:

* **Functional Requirements**:
  + Lecturers can submit claims and upload supporting documents.
  + Coordinators can approve or reject claims.
  + Managers can track claim statuses.
* **Non-Functional Requirements**:
  + The system should be user-friendly and intuitive.
  + It must be scalable for future enhancements.
  + It should ensure data consistency and security.

**8. Project Timeline**

I’ve broken the work into weekly tasks to stay on track:

* **Week 1**: Set up the GitHub repo and basic MVC framework.
* **Week 2**: Create the UML diagram and finalize the database design.
* **Week 3**: Build the Lecturer’s claim submission interface prototype.
* **Week 4**: Finish the Coordinator and Manager interface mockups.
* **Week 5**: Wrap up the documentation, double-check the rubric, and prepare for submission.

This schedule keeps things manageable and ensures I meet deadlines.

**9.Design Decision**

For the Contract Monthly Claim System (CMCS), I chose the **Model-View-Controller (MVC)** approach because it keeps everything neat and tidy. It separates the database stuff (Model), the user interface (View), and the behind-the-scenes logic (Controller). This setup makes the system easier to maintain, expand, and fix if issues pop up.

**Database Setup**

The database is built around four main pieces:

* **Lecturer**: Holds info like name, email, and banking details.
* **Claim**: Keeps track of claim specifics, like hours worked, hourly rate, total amount, and whether it’s approved.
* **Document**: Stores uploaded files (like receipts or proof) with their file paths and upload dates.
* **UserRole**: Sets up user types (Lecturer, Coordinator, Manager) to control who can do what.

**How They’re Linked:**

* A Lecturer can submit multiple Claims.
* Each Claim can have several Documents attached.
* Every Lecturer is assigned a UserRole to manage their access.

**10. GUI Prototype**

The prototype includes three main views:

* **Lecturer View**: A simple form where lecturers enter hours worked, view their preset hourly rate, and upload files.
* **Coordinator View**: A list of pending claims with buttons to approve or reject them.
* **Manager View**: A dashbosard summarizing claim statuses at a glance.

Right now, these are just visual mockups built in the MVC project’s Views folder, with no back-end functionality.

**11. Security and Validation**

* **File Validation**: Only PDF, DOCX, and XLSX files are permitted to prevent unsafe uploads.
* **User Authentication**: Each lecturer, coordinator, and manager is assigned a role for controlled access.
* **Data Validation**: Input fields such as hours worked and hourly rate will be validated to prevent incorrect submissions.
* **Error Handling**: Even in the prototype, error messages will be displayed to maintain usability.

**12. UI/UX Design Choices**

The prototype focuses on **simplicity and clarity**:

* **Lecturer View**: A clean form with labeled fields and a clear “Submit” button.
* **Coordinator View**: Tabular layout of claims with Approve/Reject buttons for efficiency.
* **Manager View**: A dashboard view with status summaries for at-a-glance decision-making.
* **Consistency**: Standard color themes and navigation menus across all roles.

**13. Version Control**

I’m using **GitHub** to manage versions, with at least five commits planned for key milestones: project setup, adding the UML diagram, creating GUI prototypes, and finalizing documentation. I’ll use clear, descriptive commit messages to keep everything transparent.