|  |  |
| --- | --- |
| A picture of a winding road and trees  CONTACT MONTHLY CLAIM SYSTEM  Programming 2B | Abstract  The Contract Monthly Claim System (CMCS) is a web-based prototype designed to simplify the submission and management of academic claims. It enables lecturers to register, log in, and submit claims with supporting details and documents. Programme coordinators and management can review, approve, or reject claims through a structured workflow. Built with C#, MySQL, HTML, CSS, and JavaScript, the system follows an MVC design for security and scalability. The prototype promotes efficiency, transparency, and accountability in the claims approval process.  Amogelang Refilwe Matlhaga  PROG6212 |

**Contract Monthly Claim System (CMCS) - Development Process, Design Choices, and Database Structure**

1. **Project Overview**
2. **Introduction**

The Claims Management Application is a system created to make it easier for lecturers to submit, track, and complete their academic claims. At the moment, claims are often handled using manual or partly digital methods, which can cause delays, confusion, and poor record-keeping. This system solves those problems by giving lecturers, coordinators, and management a clear digital platform that can be used on both computers and mobile devices. Lecturers can register, log in, and send in claims with details such as the number of sessions, hours worked, hourly rate, total amount, module, faculty, and supporting documents. Every claim starts with a Pending status. Programme coordinators then review the claim and either approve or reject it. If a claim is approved, it goes to programme management, which makes the final choice to either accept it as complete or reject it.

This process makes sure that every step is clear and that each role is responsible for its part. On the technical side, the application is built using the Model–View–Controller (MVC) design, which helps separate the user interface, data, and system logic. The backend is created with C#, and a MySQL database is used to safely store and manage information about users and claims. The front end is built with HTML, CSS, and JavaScript, which makes the system easy to use and allows it to work well on both desktop and mobile devices. Together, these choices make the system simple, secure, and effective for its purpose.

1. **Objectives of the System:**

* To provide a secure user-friendly platform for lecturers to submit claims.
* To introduce a transparent approval process involving both the programme coordinator and programme management.
* To ensure data integrity and accountability through a well-structured database design.

1. **Core User Stories**

* The lecturer can register, log in, submit a claim using all the supporting documents with detailed descriptions of the sessions, hours, rates, module and faculty they are lecturing. They can also track the claim status with its default claim creating date.
* The programme coordinator can review claims and either approve or reject them.
* The programme management can finalize the claims and either approve or reject them.

1. **Roles**
2. Lecturer (Claimant)

* Registers.
* Logs in.
* Submits claims.
* Tracks claims.

1. Programme Coordinator

* Reviews claims.
* Approves or rejects claims.

1. Project Management (PM)

* Finalizes or rejects claims.

1. **Workflow**
2. Register 🡪 Login (Lecturer).
3. Submit Claim 🡪 System sets status = Pending and records created date.
4. Programmer Coordinator reviews 🡪 Approved 🡺 Status transition to Approved.
5. Programme Coordinator reviews 🡪 Rejected 🡺 Status transition to Rejected.
6. Programme Management (PM) sees the approved queue 🡪 Finalize 🡺 Status transition to Finalized.
7. Programme Management (PM) sees the rejected queue 🡪 Finalize 🡺 Status transition to RejectedByPM.
8. **Development Process**
9. Requirements

* The application must be able to get functional features from the following:
* Register page.
* Login page.
* Submit claim page.
* Track claim page.
* Coordinator approval
* PM finalization.
* The application must also extract fields to capture each form and table.

1. Architecture and Technology Decisions

* The application will be an MVC application since it provides clear separation of concerns and responsive web delivery across desktop and mobile applications.
* Front-End Tools
* HTML
* CSS
* JavaScript
* Those are the tools that will be used because they provide a broad device accessibility.
* Back-End Tools
* C#
* MySQL DataClient Extension
* These tools will be used because they can persist users and claims with a structured relational model.

1. Detailed Design

* Model entities
* Users Model
* Claims Model
* Claim Workflow
* User register 🡪 fills the form 🡪 login.
* Users submit claims 🡪 fill the form and attach documents 🡪 claim captured as pending.
* Coordinator reviews claim 🡪 approves/rejects claims.
* PM finalizes claim 🡪 finalize/reject claims.

1. **Architectural Design Choices and Rationale**
2. MVC Template

* Separation of concerns: Business logic in controllers or services and persistence in models.
* Web Focus and Responsiveness: MVC aligns with a responsive web application accessible from desktop devices to mobile devices.

1. Font-End: HTML + CSS + JavaScript

* Commonness and Accessibility: Works across browsers and devices without native installs and they will support the project’s mobile to desktop accessibility goal.
* Progressive Enhancement: Server-rendered pages remain usable even if JavaScript is limited.

1. Back-End: C# + MySQL DataClient Extension

* Type Safety and Productivity: IN C# for modelling claims logic and validation.
* Relational Consistency: In MySQL for enforcing referential integrity between users and claims and for structured reporting on statuses and amounts.

1. Database Shape: Users + Claims

* Core Tables:
* Users Table
* For identity and roles
* Claims Table
* For claims submissions
* This will map clearly to MVC models and the workflow.

1. **Security, Validation and Authorization**
2. Password Storage

* Store only a salted hash like “Amo2511” instead of raw passwords and this will support password encryption and have better security standards, and it will also avoid reversable storage.

1. Authentication:
2. Authorization:

* Lecturer:
* Create and read their own claims.
* Programme Coordinator:
* View all the pending claims (approved/rejected).
* Programme Management (PM)
* View all the claims approved or rejected by the coordinator (finalize/reject).

1. Input Validation:

* Email must be unique and valid.
* Sessions, hours, and rates:
* Total Amount = Sessions × Rate
* Documents:
* Restrict to allowed types (PDF, PNG, JPG) and the maximum size.
* Auditing:
* Record who changed status and when they changed the status.

1. **Database Structure**
2. Create the Users table and it must show the column name, data type and description inside the table

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| User ID [PK] | INT | User’s ID |
| Full Names | VARCHAR | User’s First & Last Name |
| Surname | VARCHAR | User’ Surname |
| Email | VARCHAR | User’s Email |
| Gender | VARCHAR | User’s gender |
| Role | VARCHAR | User’s role |
| Password | VARCHAR | User’s password |
| Date | DATE | User’s date of account creation |

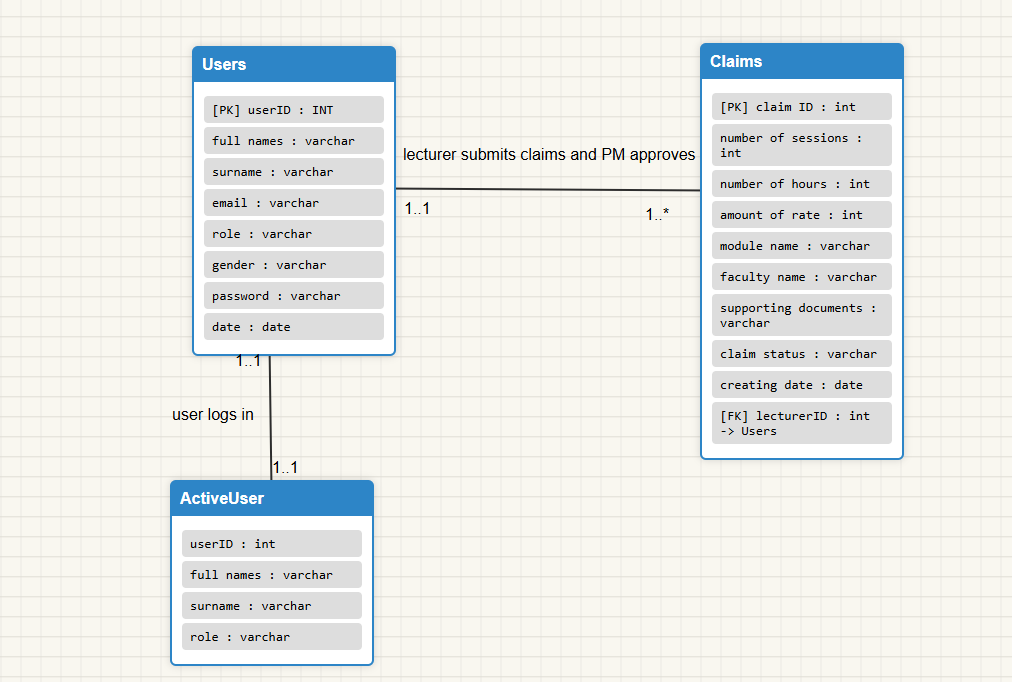
1. Create Claims Table and it must show column name, data type and description

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| Claim ID [PK] | INT | Claims ID |
| Number of sessions | INT | Number of the sessions attended |
| Number of hours | INT | Hours attended |
| Amount of rate | INT | Amounts of rates |
| Total amount | VARCHAR | Total amount claiming |
| Module name | VARCHAR | Module attended |
| Faculty name | VARCHAR | Faculty name attended |
| Documents | VARCHAR | Supporting documents |
| Claim status | VARCHAR | Claiming status progress |
| Date collection | DATE | Date claim submitted |
| Lecturer ID [FK] | INT | Lectures ID |

1. Create the Active User table

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| User ID | INT | Users Id |
| First Names | VARCHAR | The first and last name of the user |
| Surname | VARCHAR | The surname of the user |
| Role | VARCHAR | The role the user is currently occupying |
|  |  |  |

1. **UML Class Diagrams**

****

1. **UI Pages and required fields**
2. Lecturer (claimant) Pages:

* Registration Page:
* Fields:
* First Name(s).
* Surname.
* Email.
* Gender (Male, Female, Other).
* Role (Lecturer, programme Coordinator) default = Lecturer.
* Password
* System:
* Auto captures account creating date.
* Validates email uniqueness.
* Stores password as a hash
* Login Page:
* Fields:
* Email.
* Password.
* System:
* Authenticates credentials.
* Redirects lecturer to their dashboard.
* Submit Claim Page
* Fields:
* Number of sessions.
* Number of hours.
* Rate amount (per session/hour).
* Total amount (auto-calculated = session × rate).
* Module name
* Faculty name
* Upload supporting documents (PDF/JPG/PNG).
* System:
* Default claim status = pending.
* Captures claim submission date.
* Links claim to lecturer’s UserID.
* Track Claim Page:
* Table Columns:
* Row number.
* Sessions.
* Hours.
* Rate.
* Total amount.
* Module.
* Faculty.
* Supporting document (download/view link).
* Status (Pending, Approved, Rejected, Finalized).
* Date created.

1. Programme Coordinator Pages:

* Login Page:
* Fields:
* Email.
* Password.
* System:
* Authenticates and redirects to Coordinator Dashboard.
* Pending Claim Review Page:
* Table Columns:
* Lecturer Full Names.
* Lecturer Surname.
* Module Name.
* Faculty Name.
* Total Amount.
* Supporting Document (download/view link).
* Date Submitted.
* Action (Approve/Reject).
* System:
* Only shows claim with status = pending.
* Updates status to ApprovedBy Coordinator or Rejected.

1. Programme Management Pages:

* Login Page:
* Fields:
* Email.
* Password.
* System:
* Authenticates and redirects to Management Dashboard.
* Pending Claim Review Page:
* Table Columns:
* Lecturer Full Names.
* Lecturer Surname.
* Module Name.
* Faculty Name.
* Total Amount.
* Supporting Document (download/view link).
* Date Submitted.
* Action (Approve/Reject).
* System:
* Only shows claim with status = ApprovedBy Coordinator.
* Updates status to Finalized or RejectedByPM.

1. **Project Plan: CMCS Prototype Development**
2. Project Phases and Tasks:

* Phase 1: Planning and Requirements (Week 1 – 2)
* Task 1:
* Review and finalize functional requirements (Registration, Login, Submit Claim, Track Claim, Coordinator Approval, PM Finalization).
* Task 2:
* Define database schema (Users, Claims).
* Task 3:
* Identify non-functional requirements (Security, Validation, Auditing).
* Dependencies:
* None – starting phase.
* Phase 2: Design (Week 2 – 3)
* Task 1:
* Design database (ERD, Table structures).
* Task 2:
* Create system architecture diagrams (MVC flow, role workflow).
* Task 3:
* Wireframe UI pages (Lecture dashboard, Coordinator dashboard and PM dashboard).
* Dependencies:
* Requirements finalized (Phase 1).
* Phase 3: Back-End Development (Week 3 – 6)
* Task 1:
* Set up project environment (C# MVC Template, MySQL integration).
* Task 2:
* Implement user authentication and authorization (registration, login, password hashing).
* Task 3:
* Implement claims submission logic (form handling, validation, file upload).
* Task 4:
* Implement PM finalization logic (finalize/reject).
* Task 5:
* Implement PM finalization logic (finalize/reject).
* Dependencies:
* Database design (Phase 2).
* Phase 4: Front-End Development (Week 4 – 7)
* Task 1:
* Develop responsive UI pages (HTML, CSS, JS).
* Task 2:
* Integrate UI with back-end controller.
* Task 3:
* Build dashboard (lecturer, coordinator, PM) with role-based access.
* Dependencies:
* Wireframes and Database design (Phase 2).
* Phase 5: Integration and Testing (Week 7 – 8)
* Task 1:
* Unit testing for back-end logic (claims workflow).
* Task 2:
* Integration testing (UL and Database and Workflow).
* Task 3:
* Security and validation testing (input, file uploads, authentication).
* Task 4:
* User acceptance testing (simulate lecturer, coordinator, PM roles).
* Dependencies:
* Completion of back-end and front-end core functions.
* Phase 6: Deployment and Review (Week 9)
* Task 1:
* Deploy prototype on test server.
* Task 2:
* Gather feedback from sample users (lecturer, coordinator, management).
* Task 3:
* Final review and documentation.
* Dependencies:
* Successful system testing.

1. Timeline

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Tasks** | **Duration** | **Dependencies** |
| Phase 1  (Week 1 – 2) | Planning and Requirements | 2 Weeks | None |
| Phase 2  (Week 2 – 3) | Design (Database, Architecture, Wireframes) | 1 Week | Requirements |
| Phase 2  (Week 3 – 6) | Back-End Development | 4 Weeks | Design |
| Phase 4  (Week 4 – 7) | Front-End Development | 4 Weeks | Design (overlaps back-end) |
| Phase 5  (Week 7 – 8) | Integration and Testing | 2 Weeks | Back-End and Front-End |
| Phase 6  (Week 9) | Deployment and Review | 1 Week | Testing |

1. Key Dependencies:

* Requirements must be approved before design starts.
* Database design must be ready before coding the back-end.
* Front-end depends on wireframes but can overlap with back-end coding.
* Testing requires both back-end and front-end to be integrated.
* Deployment depends on successful testing.