**ST10378552**

**DEVESH NAIDOO**

**PROG6212 POEPART 1**

**Table Entities and Database Structure:**

**1. Table: Claims\_Submission-** This table is used for submitting claims by lecturers

* Claim\_ID (int, PK, auto-increment) - Unique identifier for each claim.
* Lecturer\_ID (int, FK) - Reference to the lecturer who submitted the claim.
* TotalHoursWorked (decimal) - Total hours worked by the lecturer.
* HourlyRate (decimal) - Hourly rate of the lecturer.
* ClaimAmount (decimal) - Total claim amount (calculated as TotalHoursWorked \* HourlyRate).
* SubmissionDate (datetime) - Date and time when the claim was submitted.

**2. Table: Claim\_Approval-** Programme Coordinators and Academic Managers use this table to approve lecturer claims

* Approval\_ID (int, PK, auto-increment) - Unique identifier for each review.
* Claim\_ID (int, FK) - Reference to the claim being reviewed.
* ApprovalDate (datetime) - Date and time when the claim was reviewed.
* ApprovalStatus (varchar) - Status of the claim after review (e.g., "Approved", "Rejected").
* Lecturer\_FullName- Full Name of the lecturer.
* ToatlAmount- The Total Amount of the Claim.

**3. Table: Claim\_Tracking-** Lecturers can use this table to monitor their claim submissions.

* Tracking\_ID (int, PK, auto-increment) - Unique identifier for each tracking record.
* Claim\_ID (int, FK) - Reference to the claim being tracked.
* TrackingStatus (varchar) - Current status of the claim.
* SubmissionDate (datetime) - Date and time of when the status was submitted.
* TotalAmount(int) – The amount of the claim that was submitted.

**4. Table: Lecturer**- This table stores key information about lecturers

* Lecturer\_ID (int, PK, auto-increment) - Unique identifier for each lecturer.
* Lecturer\_FullName (varchar) - Name of the lecturer.
* Lecturer\_PhoneNumber (varchar) - Contact information for the lecturer.
* Lecturer\_Email (varchar) - Email address of the lecturer.
* TotalHoursWorked – The total number of hours that the lecturer worked for.
* HourlyRate – The amount that the lecturer receives per hour they worked.
* TotalClaimSubmitted- The total number of claims submitted by that lecturer.

**5. Table: Claim\_Reviewals**- : This table allows Programme Coordinators and Academic Managers to review lecturer claims.

* Review\_ID (int, PK, auto-increment) - Unique identifier for each review.
* Claim\_ID (int, FK) - Reference to the claim being reviewed.
* TotalAmount(int) – The amount of the claim that was submitted.
* HourlyRate – The amount that the lecturer receives per hour they worked.
* TotalHoursWorked – The total number of hours that the lecturer worked for.
* ApprovalStatus (varchar) - Status of the claim after review (e.g., "Approved", "Rejected").

**Entity Relationships:**

* Each lecturer can submit multiple claims, creating a one-to-many relationship between the Lecturer and Claim tables.
* A lecturer can track several claims, establishing a one-to-many relationship between the Lecturer and Track Claims tables.
* Each claim's status can be individually monitored, leading to a one-to-one relationship between the Submit Claims and Track Claims tables.
* Every claim is reviewed only once, resulting in a one-to-one relationship between the Submit Claims and Review Claims tables.
* Each claim is either approved or rejected exactly once, forming a one-to-one relationship between the Submit Claims and Approve Claims tables.

**Design Choices**

**GUI Layout/Design:**

1. **Home Page:** Provides users with easy access to various functionalities within the application. Includes the button to 'Submit Claim. This central navigation hub helps users quickly find and access different sections of the application. Consists of a navigation bar at the top of the page for constant accessibility, with clear labels and intuitive icons.
2. **Submit Claim Page:** Divided into fields for 'Total Hours Worked,' 'Hourly Rate,' and 'Claim Amount.' Each field is clearly labeled and designed to ensure users can input their data correctly. Allows users to attach supporting documents relevant to their claim. The file upload functionality supports multiple file types (e.g., PDFs, images) and includes a preview option if possible. Utilizes Bootstrap's form controls for consistency, with a card layout to group related fields. The page uses spacing and alignment to enhance readability and reduce user input errors.
3. **Approve Claims Page:** Displays a list or table of submitted claims. Each claim includes essential details like lecturer name, claim amount, and submission date. Provides options to approve or reject each claim directly from the list. Each claim is presented in a card format for ease of management. Incorporates a clean card layout with color-coded statuses (e.g., pending, approved, rejected) to visually differentiate between claim stages. Action buttons are prominently placed for easy access.
4. **Track Claims Page:** Allows users to monitor the status of their submitted claims over time.Displays a card layout of claims with tracking details such as submission date, current status, and any comments or updates from reviewers.Uses a card layout to present claim tracking information. Cards include key details like claim ID, status, submission date, and action buttons for more details or updates. Color-coded statuses and icons to quickly convey the progress of claims (e.g., “In Review,” “Approved,” “Rejected”). Ensures that the layout adjusts for different screen sizes, with an easy-to-navigate interface on both mobile and desktop views.
5. **Lecturer Profiles Page:** Displays detailed information about each lecturer, including their personal details and claim history.Includes profile picture, name, contact details, and a summary of their submitted and approved claims.Allows users or administrators to update lecturer details if needed.A card or profile layout with a circular profile picture at the top, followed by sections for personal details and claim history. Uses a consistent design with Bootstrap’s card components, clear typography, and color accents to highlight important information. Ensures that the profile page looks good on different devices, with flexible image and text positioning.
6. **Review Claims Page:** Allows administrators or reviewers to view and process claims that are pending review.Displays claims with details such as lecturer name, claim amount, submission date, and any attached supporting documents.Provides options for reviewers to approve, reject, or request additional information for each claim.Uses a card or table format to organize claims, with distinct sections for each claim's details and action buttons.Incorporates color-coded status indicators and clear, actionable buttons to facilitate easy claim processing.Adapts to different screen sizes, ensuring that the review interface remains functional and accessible.
7. **Login Page:** Allows users to securely log in to the application. Includes fields for username/email and password, with options for password recovery or reset.Ensures that only authorized users can access the system. Login button and, if necessary, multi-factor authentication prompts.Simple and focused on user authentication, with a clean, centered form on the page. Includes branding elements and clear instructions for logging in.Uses Bootstrap’s form controls for consistency, with a modern, minimalistic design. Incorporates colors and branding to align with the rest of the application’s design. Ensures the login form is easily usable on both mobile and desktop devices, with appropriate field sizing and button placement.

**Design Rationale:**  
The database structure is designed with separation of concerns, allowing for independent management of lecturer data, claim submissions, and claim tracking. This approach simplifies maintenance and enhances scalability.

**Reason for Field Selection:**

* **Lecturer Profile Table:** Storing fields like *Total\_Hours\_Worked* and *Total\_Claims\_Submitted* directly in the Lecturer Profile table increases efficiency by eliminating the need to recalculate these values for every query. This allows for quick access to a lecturer’s claims and work history without reprocessing historical data.
* **Submit Claims Table:** To maintain the accuracy of financial records, the *Hourly\_Rate* is included in the Submit Claims table when the claim is made. This ensures that historical claims remain accurate even if the hourly rate changes in the future, preventing retroactive changes from affecting past claims.
* **Approve Claims Table:** The *Claim\_ID*, *Lecturer Name*, and *Status* are stored in this table to record the final outcome of each claim. Storing these fields allows for clear, unambiguous tracking of claims that have been either approved or declined, ensuring there is no confusion regarding the final status of a claim.
* **Track Claims Table:** Storing fields such as *Submitted\_Date*, *Status*, and *Total\_Amount* in the Track Claims table enables lecturers to view the current state and total value of each claim. This allows lecturers to monitor the progress of their claims without needing to access the entire claim history.
* **Review Claims Table:** The *Lecturer Name*, *Hours Worked*, *Hourly Rate*, and *Total* are stored in this table to provide a clear summary for review by Programme Coordinators and Academic Managers. These fields allow for an efficient review process by presenting all necessary information about the claim in a concise manner, facilitating informed decision-making.

The design is built for scalability, allowing for the easy addition of features such as claim history and document versioning. For instance, new claim statuses like "Under Review" or "Rejected" can be incorporated into the Track Claims table. To support claim history, a ClaimHistory table can track changes over time. Additionally, the Submit Claims table can be expanded to manage multiple document uploads with version numbers, facilitating document versioning. This approach ensures that future enhancements can be implemented efficiently, without requiring a major redesign.

**Assumptions and Constraints**

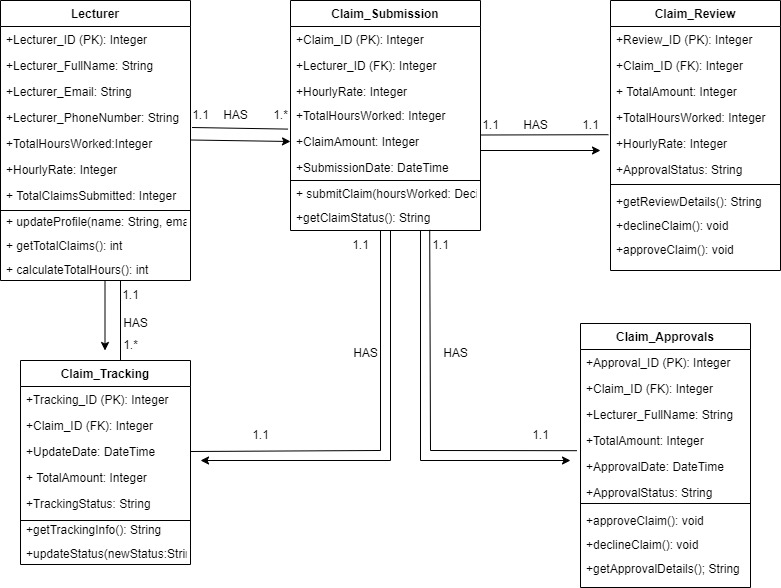
**Assumptions:**

1. User Knowledge: Users are expected to understand basic web interactions like filling forms and uploading files.
2. Internet Stability: The application assumes stable internet connections for document uploads and system access.
3. File Handling: Users are familiar with handling and uploading various document types (e.g., PDFs, images).

**Constraints:**

1. Prototype Limitations: The prototype focuses on design and layout, without functional backend logic or data processing.
2. Technology Stack: Developed with .NET Core and Bootstrap, limiting advanced features or customizations.
3. Data Handling: The prototype does not include advanced document handling like validation or error checking.

**UML Class Diagram:**



**3. Project Plan**

**Project Overview**

The Contract Monthly Claim System (CMCS) project aims to develop a comprehensive prototype that includes various functionalities for claim submission, approval, and tracking. The project leverages .NET Core for the backend and Bootstrap for the front-end design to ensure a modern, responsive, and user-friendly interface.

**Detailed Project Plan**

**Week 1-2: Define Requirements**

**1.1. Requirement Gathering and Scope Definition**

* **Tasks:**
  + Identify and document key functionalities and user needs for the Contract Monthly Claim System.
  + Create use case diagrams to visually represent different user interactions with the system.
  + Set clear project goals and deliverables.
  + Develop a scope statement detailing the project boundaries.
* **Dependencies:**
  + Development of the scope statement and project roadmap depends on the finalized requirements specification document.
* **Deliverables:**
  + Requirements Specification Document outlining all functionalities, user roles, and system constraints.
  + Use Case Diagrams and Scope Statement

**Week 3-4: Design Phase**

**2.1. GUI Design**

* **Objectives:**
  + Ensure that the design meets user requirements and is aligned with best practices.
  + Create wireframes for Home Page, Submit Claim, Approve Claims, Track Claims, Lecturer Profiles, Review Claims, and Login Page using tools like Figma or Microsoft MVC.
  + Develop mockups with Bootstrap components to visualize the final look and feel of the application.
* **Dependencies:**
  + The creation of wireframes and mockups requires the finalized requirements from Week 1-2.
* **Deliverables:**
  + Wireframes and Mockups for all pages.

**2.2. UML Class Diagram for Database Structure**

* **Objectives:**
  + Design a comprehensive UML class diagram using Draw.io to represent the database schema.
  + Ensure that the diagram accurately reflects the relationships and attributes of the entities.
  + Define and document classes (Lecturer, Claim, SupportingDocument) with their attributes and relationships.
* **Dependencies:**
  + Requires finalised requirements to accurately define classes and relationships.
* **Deliverables:**
  + UML Class Diagram.

**Week 5: Development Setup**

**3.1. Development Environment Setup**

* **Objectives:**
  + Set up .NET Core development environment, including necessary tools and libraries.
  + Create and configure the project repository on GitHub or another version control platform.
* **Dependencies:**
  + Requires completion of design phase for understanding the development needs.
* **Deliverables:**
  + Configured Development Environment.
  + Project Repository and Task Tracking Setup.

**Week 6-8: GUI Implementation**

**4.1. Front-End Development**

* **Objectives:**
  + Develop the user interface for each page using Bootstrap, ensuring responsiveness and adherence to design specs.
  + Implement HTML, CSS, and JavaScript based on approved mockups, and perform usability testing to validate and adjust the user experience.
* **Dependencies:**
  + Requires finalized wireframes and mockups from the design phase.
* **Deliverables:**
  + Completed Front-End Code for Home, Submit Claim, Approve Claims, Track Claims, Lecturer Profiles, Review Claims, and Login Pages.
  + Usability Testing Report.

**4.2. Integration with Backend (Mock Data)**

* **Objectives:**
  + Integrate the front-end with a mock backend to simulate data interaction and test data flow.
  + Create mock data and services for backend simulation, implementing databinding and interaction features.
  + Conduct integration testing to ensure proper GUI-backend interaction.
* **Dependencies:**
  + Requires front-end development to be completed.
* **Deliverables:**
  + Integrated GUI with Mock Backend.
  + Integration Testing Report.

**Week 9: Database Design**

**5.1. Database Schema Implementation**

* **Objectives:**
  + Design and implement the database schema from the UML diagram, setting up tables and relationships for Lecturer, Claim, Tracking, Submission and Approval.
  + Validate the schema against application requirements using a database management system (e.g., SQL Server, MySQL).
* **Dependencies:**
  + Requires the UML class diagram from the design phase.
* **Deliverables:**
  + Implemented Database Schema.
  + ORM Configuration and Setup.

**Week 10-11: Integration**

**6.1. GUI and Database Integration**

* **Objectives:**
  + Integrate the front-end with the database, ensuring seamless data flow and functionality through backend services and APIs.
  + Conduct end-to-end testing to validate integration.
* **Dependencies:**
  + Requires completion of front-end development and database schema implementation.
* **Deliverables:**
  + Integrated System with Real Database.
  + End-to-End Testing Report.

**Week 12: Testing and Refinement**

**7.1. Prototype Testing**

* **Objectives:**
  + Validate that all functionalities work as expected.
  + Perform unit testing, integration testing, and user acceptance testing (UAT) to address any issues or bugs.
* **Dependencies:**
  + Requires completed integration of the system with the real database and fully developed GUI.
* **Deliverables:**
  + Prototype Testing Report.
  + List of Issues and Resolutions.

**7.2. Design Refinement**

* **Objectives:**
  + Refine the design and functionality based on feedback and testing results.
  + Implement design changes and improvements based on feedback.
* **Dependencies:**
  + Requires feedback and issues identified from prototype testing.
* **Deliverables:**
  + Refined Prototype.
  + Final Design Adjustments Report.

**Week 13: Documentation and Final Review**

**8.1. Documentation Preparation**

* **Objectives:**
  + Complete detailed and accurate project documentation, including GUI layouts, UML diagrams, and database schema, then review and finalize for submission.
* **Dependencies:**
  + Requires completion of all design and development work, including testing and refinements.
* **Deliverables:**
  + Comprehensive Project Documentation.
  + User Guides and Technical Documentation.

**8.2. Final Review and Submission**

* **Objectives:**
  + Perform a final review of the prototype and documentation.
  + Prepare and submit the final project package.
* **Dependencies:**
  + Requires all previous deliverables to be complete and reviewed.
* **Deliverables:**
  + Final Review Report.
  + Submitted Project Package.

**4. GUI**

A screenshot of a computer

Description automatically generated

A screenshot of a login page

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a chat box

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated