2024

PROG6212

POE – PART 1

KAYLA FERREIRA – ST10259527

Table of Contents

1. Design Choices

.NET Core MVC

I selected .NET Core MVC due to its ability to separate concerns using the Model-View-Controller (MVC) pattern. This allows the system to be modular, maintainable, and scalable, with each layer (Model, View, and Controller) handling specific tasks.

The Model represents the data and business logic, the View focuses on displaying data and user interaction, and the Controller manages communication between the Model and View.

This separation makes it easier to implement changes in one layer without affecting others, supporting a smoother development process, especially in large-scale applications. .NET Core MVC is also cross-platform, meaning it can run on Windows, macOS, and Linux, providing flexibility for deployment in different environments.

C# Language

C# is chosen for its tight integration with the .NET environment, offering a wide array of built-in libraries and tools. It supports features such as strong typing, automatic garbage collection, and LINQ (Language Integrated Query), which simplifies data manipulation. C# also promotes secure coding practices through compile-time error checking, reducing the risk of runtime errors. Additionally, its modern syntax and object-oriented principles make it easier to implement complex business logic while ensuring the codebase is maintainable and scalable. The strong community support and continuous evolution of C# further ensure that the language remains up to date with the latest industry trends.

SQL Database

A relational database was chosen to store structured data, ensuring data integrity through the use of foreign keys, relationships, and constraints. SQL databases are well-suited for handling complex queries, particularly those involving multiple tables and aggregations required for reporting, analytics, and auditing. In this project, SQL is crucial for maintaining relationships between entities like Lecturers, Claims, and Documents, ensuring that data remains consistent and accurate throughout the system. The use of stored procedures and triggers can also enhance performance, security, and data validation at the database level.2. Database Structure

Database Tables

Lecturers

Stores information about independent contractor lecturers, including their identification details and hourly rate. This table is central to managing claims.

LecturerID {PK} : int
Name : string
Surname : string
Email : string
Password : string
HourlyRate : decimal

Claims

Contains details of each claim submitted by lecturers, including the hours worked, total amount claimed, and current status (Pending, Reviewed, Approved, Rejected).

ClaimID {PK} : int
LecturerID {FK} : int
ClaimDate : DateTime
HourlyRate : decimal
HoursWorked : int
TotalAmount : decimal
Status: string

Documents

Stores metadata about supporting documents uploaded by lecturers for each claim, including the file name and type.

DocumentID {PK} : int
ClaimID {FK} : int
FileName : string
UploadDate : DateTime
FileType : string

Admin

Manages and oversees claims, including approving, denying, or processing them.

AdminID {PK} : int
Name : string
Surname : string
Email : string
Password : string

Relationships

- **Lecturers ↔ Claims:** One-to-Many relationship, where one lecturer can submit multiple claims.
- **Claims ↔ Documents:** One-to-Many relationship, allowing multiple documents to be attached to a single claim.
- **Admin ↔ Claims:** One-to-Many relationship, where one admin handles multiple claims.

3. GUI Layout

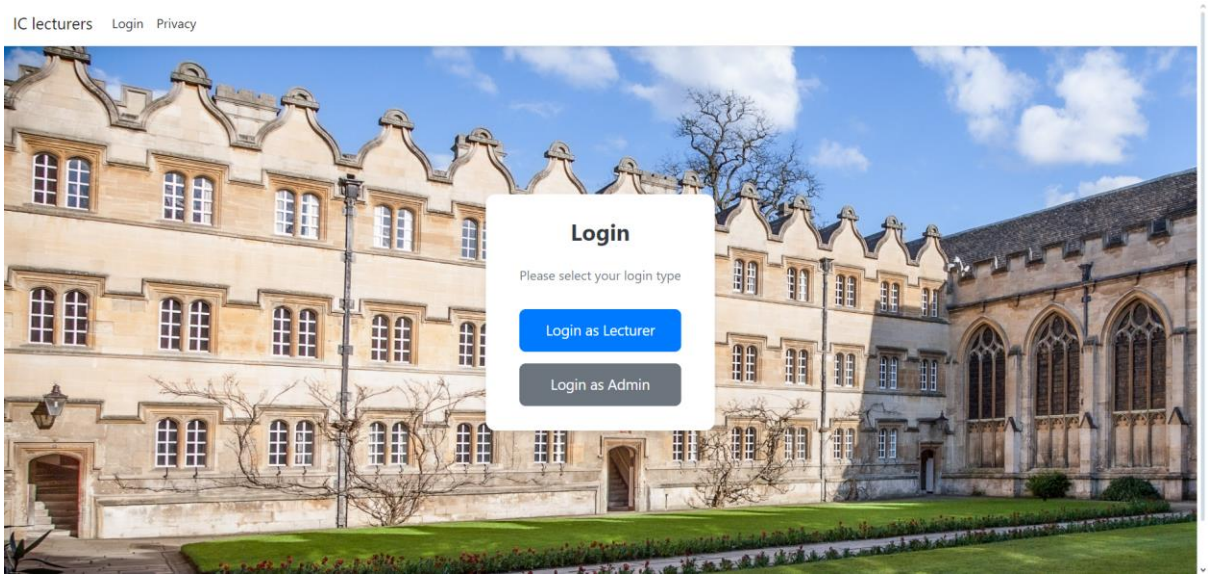
Design Considerations

- **User-Centric Design:** The interface is designed with a focus on ease of use and accessibility. Given that the primary users are lecturers and administrators, the design aims for simplicity and clarity.
- **Responsive Design:** The layout is responsive to different screen sizes, ensuring that the system is accessible from desktops, tablets, and mobile devices.
- **Consistent Navigation:** A consistent navigation bar is included across all pages, providing quick access to key features like claim submission, claim review, and document upload.

Layout

Startup Page

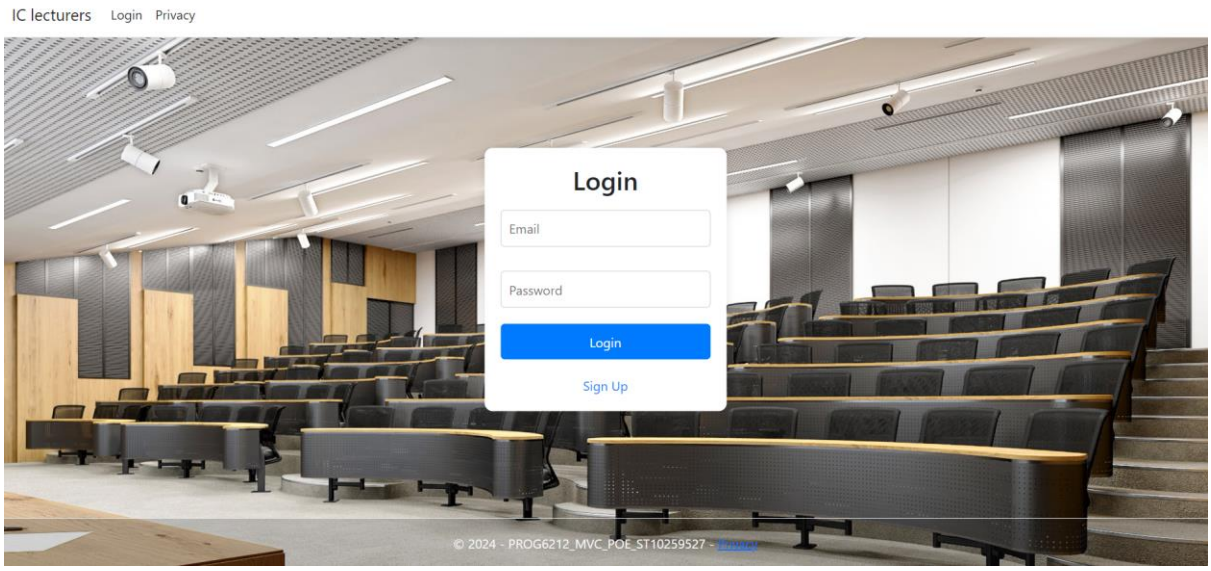
The startup page offers users the option to log in either as a lecturer or an administrator.

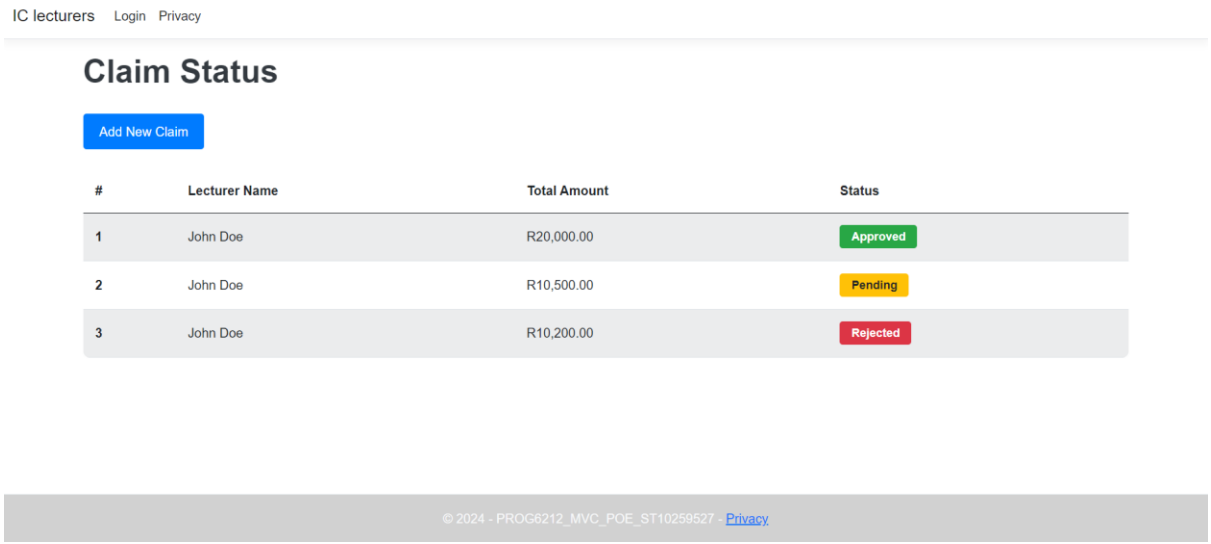


Lecturers

Lecturer Login

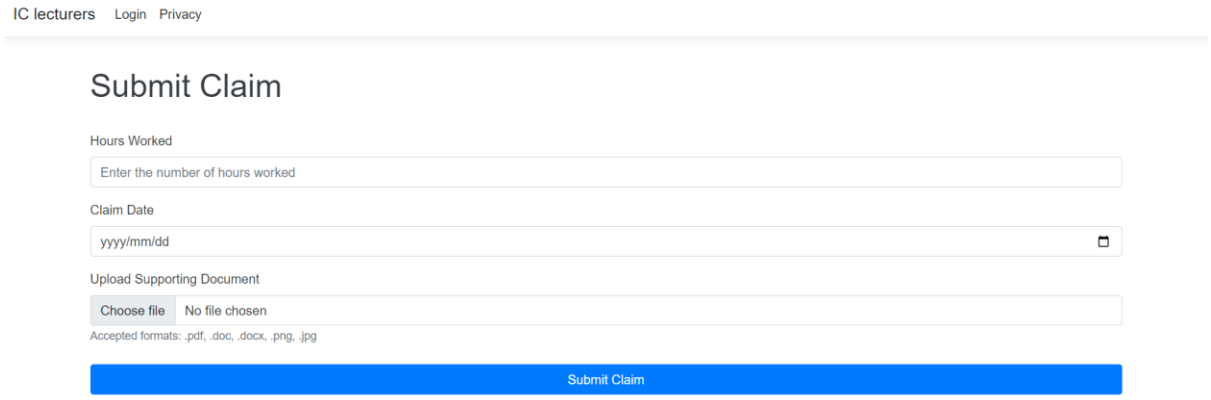
After logging in, lecturers are directed to a page where they can view and create new claims.





Create Claims

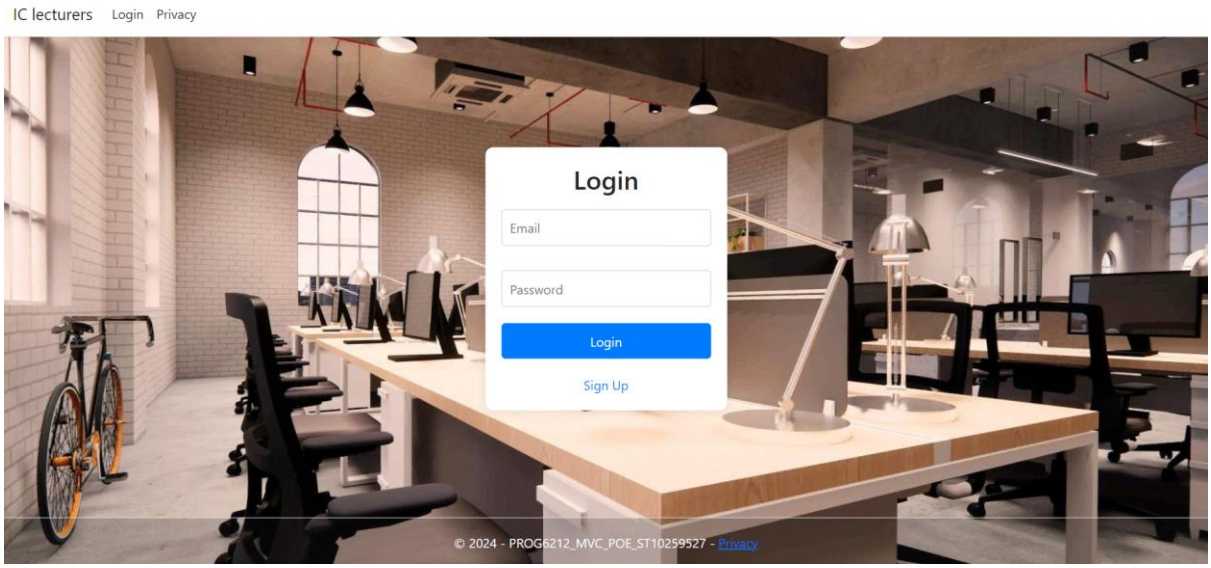
Lecturers can submit a claim by providing their hours worked, the date, and any supporting documents.



Administrators

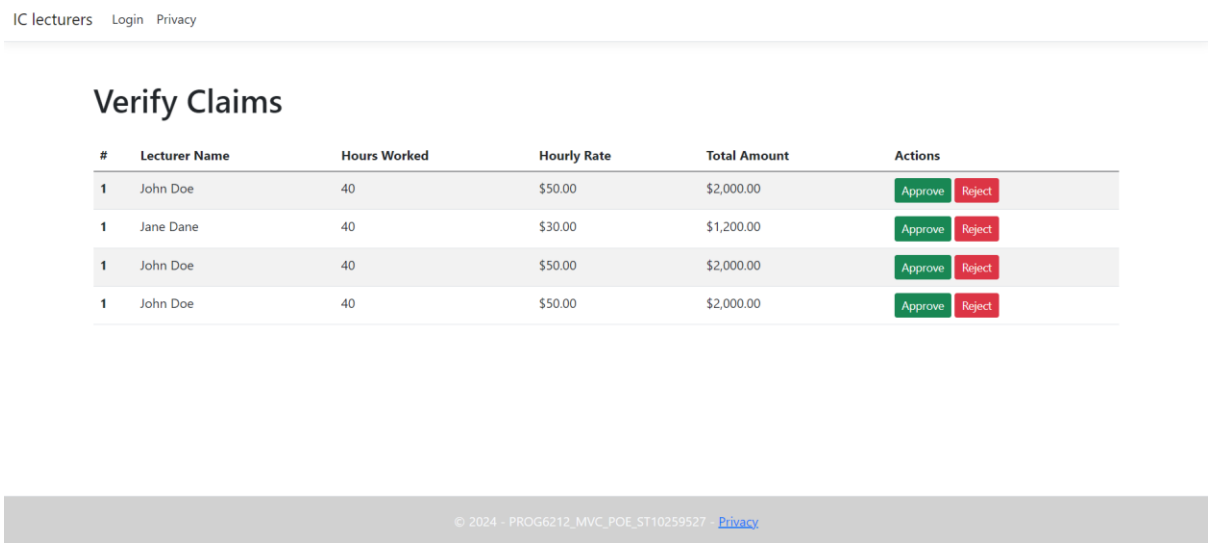
Admin Login

Administrators log in by entering their credentials, similar to the process for lecturers.



Admin Status Verification

Once logged in, administrators have the authority to approve or reject lecturers' claims, ensuring that each claim is thoroughly reviewed.



4. Assumptions and Constraints

Assumptions:

- **Data Integrity:** It is assumed that all data entered the system is accurate and that lecturers provide correct information when submitting claims.
- **User Roles:** The system assumes a clear distinction between user roles, with lecturers only able to submit claims and administrators responsible for reviewing and approving them.
- **Internet Connectivity:** The application is web-based, so it assumes users have a stable internet connection to access the system.

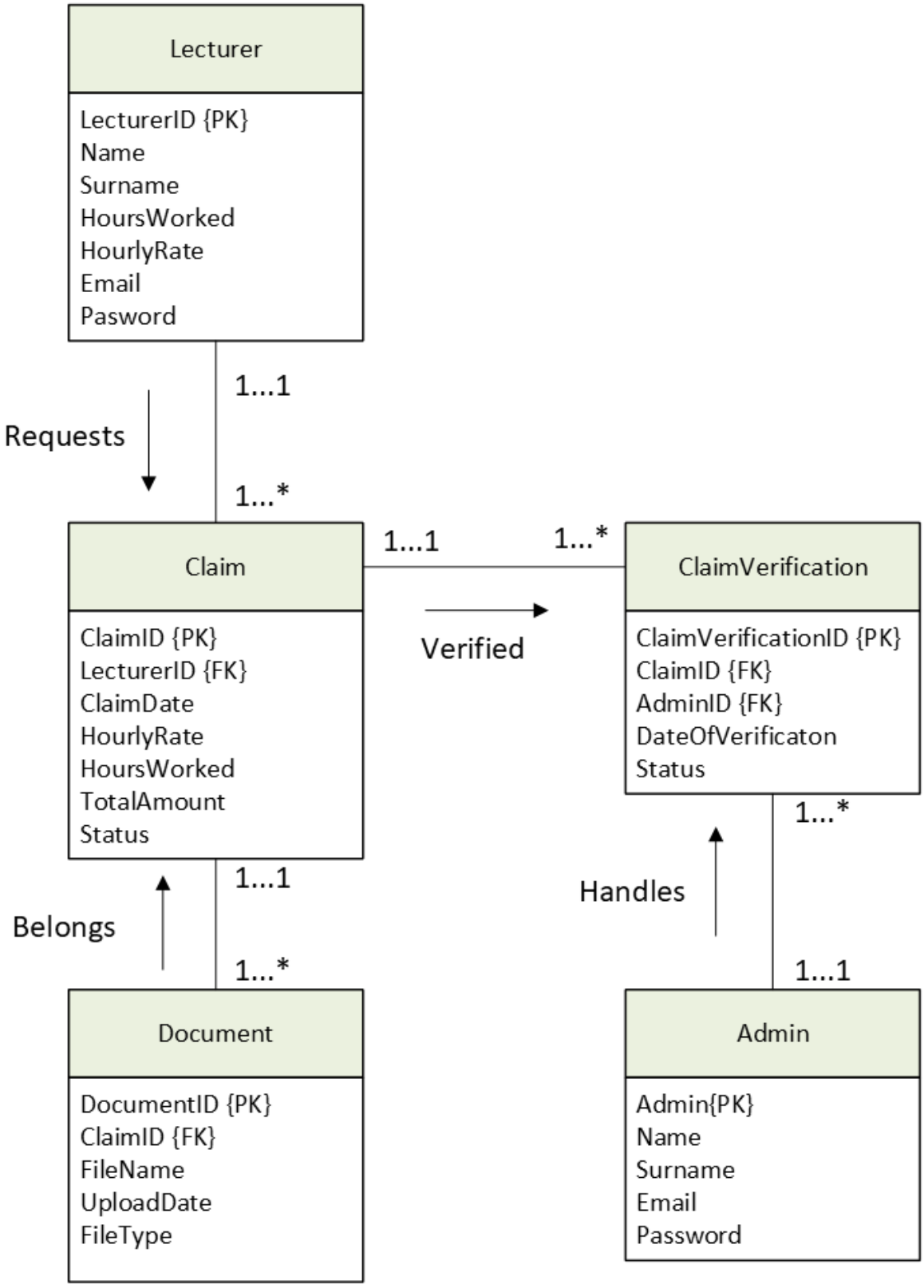
Constraints:

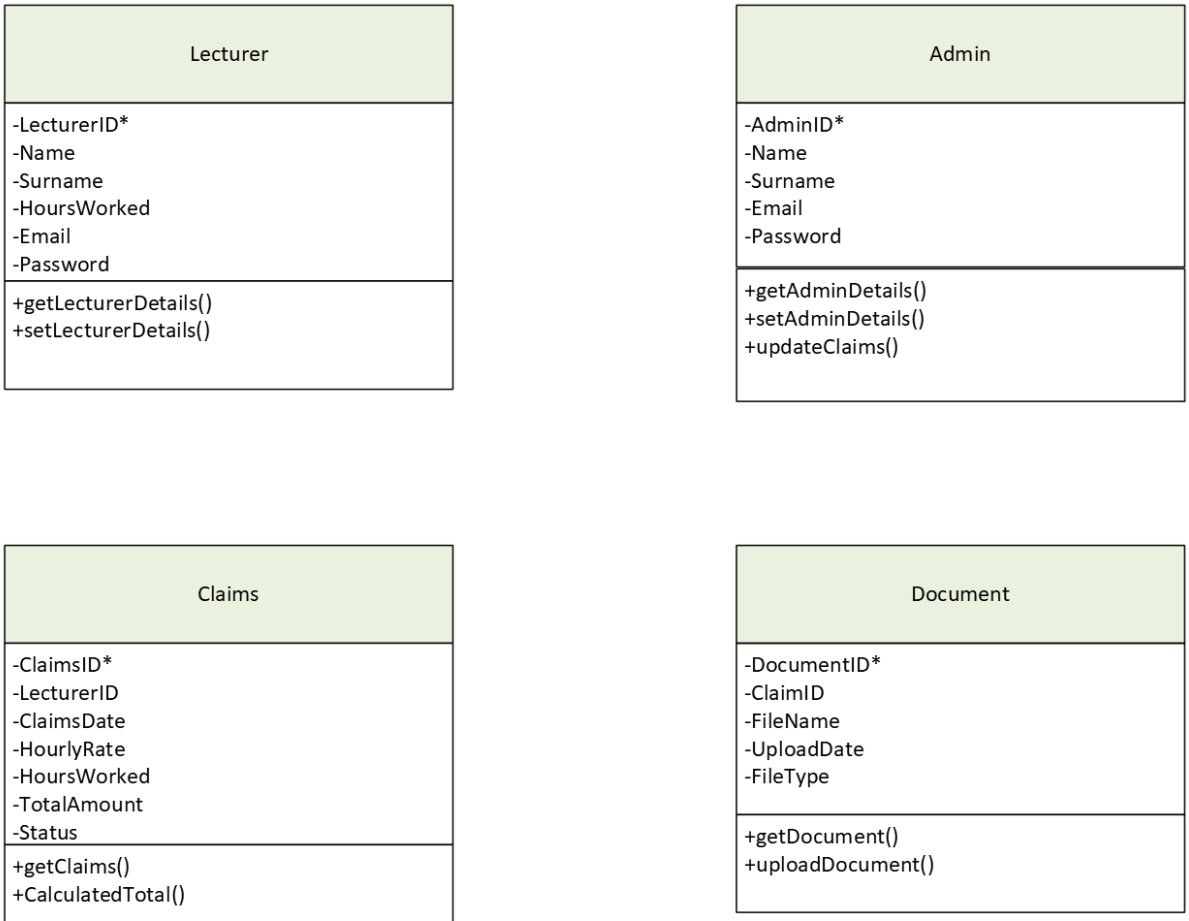
- **Security:** The system must implement strong security measures to protect sensitive information, particularly financial data and personal information. This includes encryption of sensitive data and secure user authentication.
- **Performance:** The system must be optimized for performance, especially when handling large volumes of data, such as multiple claims being submitted and reviewed at the same time.
- **Scalability:** The application must be designed to handle an increasing number of users and claims without significant degradation in performance.

Rationale Behind Design Decisions

The design decisions are guided by the need for a system that is both user-friendly and robust. By choosing the .NET Core MVC framework, the system benefits from a flexible architecture that separates concerns, making it easier to develop, maintain, and scale. The database design focuses on ensuring data integrity and supporting complex workflows, while the GUI layout emphasizes simplicity and responsiveness to accommodate a diverse user base. The assumptions and constraints considered ensure that the system meets the functional and non-functional requirements while being secure and performant.

UML Class Diagram




Lecturer
-LecturerID*
-Name
-Surname
-HoursWorked
-Email
-Password
+getLecturerDetails()
+setLecturerDetails()
Admin
-AdminID*
-Name
-Surname
-Email
-Password
+getAdminDetails()
+setAdminDetails()
+updateClaims()
Claims
-ClaimsID*
-LecturerID
-ClaimsDate
-HourlyRate
-HoursWorked
-TotalAmount
-Status
+getClaims()
+CalculatedTotal()
Document
-DocumentID*
-ClaimID
-FileName
-UploadDate
-FileType
+getDocument()
+uploadDocument()

Project Plan

Phase 1: Planning and Requirements Gathering (1 week)

This phase focuses on laying a strong foundation for the project by understanding the system's needs, defining the scope, and identifying key players.

Task 1.1: Define System Requirements

Conduct meetings and workshops with stakeholders to gather detailed requirements for the Contract Monthly Claim System (CMCS). This includes identifying functional and non-functional requirements, understanding user needs, and documenting use cases.

Dependency: None

Task 1.2: Identify Key Stakeholders and Roles

Identify all stakeholders, including independent contractor lecturers, programme coordinators, academic managers, and IT staff. Define their roles and responsibilities within the project and establish communication channels.

Dependency: Task 1.1

Task 1.3: Draft Initial Project Scope and Objectives

Based on the gathered requirements and stakeholder input, draft a clear and concise project scope document. Outline the key objectives, deliverables, and success criteria for the CMCS.

Dependency: Task 1.1, Task 1.2

Task 1.4: Create a Project Timeline with Milestones

Develop a detailed project timeline that includes all phases, tasks, and milestones. Ensure that the timeline is realistic and aligns with the project's scope and objectives. This timeline will guide the team throughout the project.

Dependency: Task 1.3

Phase 2: Design (2 weeks)

This phase involves translating the requirements into a technical design that will guide the development of the CMCS.

Task 2.1: Design the UML Class Diagram for the Database

Create a detailed UML class diagram that outlines the structure of the database, including tables, relationships, primary and foreign keys, and data types. This diagram will serve as the blueprint for the database implementation.

Dependency: Task 1.4

Task 2.2: Choose the Technology Stack (.NET Core MVC)

Evaluate and select the technology stack for the project, focusing on .NET Core MVC for the backend, along with any necessary front-end frameworks, databases, and tools. Document the rationale behind each choice.

Dependency: Task 1.4

Task 2.3: Design the GUI Layout for the System

Design wireframes and mock-ups for the CMCS's user interface. Focus on creating an intuitive and user-friendly layout for claim submission, review, and approval processes. Ensure consistency with the system's requirements and user roles.

Dependency: Task 2.1, Task 2.2

Task 2.4: Document Design Choices and Constraints

Compile a detailed design document that includes all decisions made during the design phase. Highlight any constraints, such as performance considerations, security requirements, and technical limitations.

Dependency: Task 2.3

Phase 3: Prototype Development (3 weeks)

This phase is dedicated to building a functional prototype of the CMCS based on the design created in Phase 2.

Task 3.1: Set Up the Development Environment

Install and configure the necessary software tools, libraries, and frameworks for the development team. This includes setting up version control, IDEs, and configuring the project repository.

Dependency: Task 2.4

Task 3.2: Develop the Initial Project Structure using MVC

Implement the foundational structure of the CMCS using the MVC pattern in .NET Core. This includes creating models, views, and controllers that align with the system's design and requirements.

Dependency: Task 3.1

Task 3.3: Create the Basic UI Forms for Claim Submission and Approval

Develop the essential user interface forms that allow lecturers to submit claims and administrators to review and approve them. Ensure that the forms are functional and meet the design specifications.

Dependency: Task 3.2

Task 3.4: Implement Basic Navigation Between Forms

Ensure seamless navigation between different forms and pages within the CMCS. This includes setting up routing, menus, and any necessary links or buttons to enhance user flow.

Dependency: Task 3.3

Phase 4: Review and Refinement (1 week)

The goal of this phase is to review the developed prototype with stakeholders and make any necessary adjustments based on their feedback.

Task 4.1: Review the Prototype with Stakeholders

Conduct a walkthrough of the CMCS prototype with key stakeholders, demonstrating its functionality and gathering feedback on the user interface, workflow, and overall system performance.

Dependency: Task 3.4

Task 4.2: Collect Feedback and Document Necessary Changes

Compile and categorize all feedback from stakeholders. Identify critical issues, areas for improvement, and potential enhancements. Document these changes for implementation.

Dependency: Task 4.1

Task 4.3: Refine the Prototype Based on Feedback

Implement the changes and improvements identified in the feedback process. Focus on addressing any critical issues and ensuring that the system meets the expectations of all stakeholders.

Dependency: Task 4.2

Phase 5: Final Documentation and Submission (1 week)

In this final phase, you will compile all documentation, finalize the project, and prepare the prototype for submission.

Task 5.1: Compile All Design Documentation

Gather and organize all design documents, including the UML diagrams, design choices, constraints, and any additional documentation created during the project. Ensure that the documentation is comprehensive and well-structured.

Dependency: Task 4.3

Task 5.2: Finalize the Project Plan and Timeline

Review and update the project plan and timeline to reflect the final state of the project. Ensure that all milestones and deliverables are accurately documented.

Dependency: Task 5.1

Task 5.3: Prepare the Prototype for Submission

Conduct a final review of the CMCS prototype, ensuring that all functionalities are working as expected. Prepare the prototype for submission, including packaging the code, documentation, and any other required materials.

Dependency: Task 5.2

Task 5.4: Submit the Completed Prototype and Documentation

Submit the final version of the CMCS prototype along with all accompanying documentation. Ensure that the submission meets all the requirements and deadlines set by the project guidelines.

Dependency: Task 5.3

Total Duration: 8 weeks

Explanation of Dependencies:

Phase 1: Establishes the foundational requirements and goals, necessary before moving on to design and development.

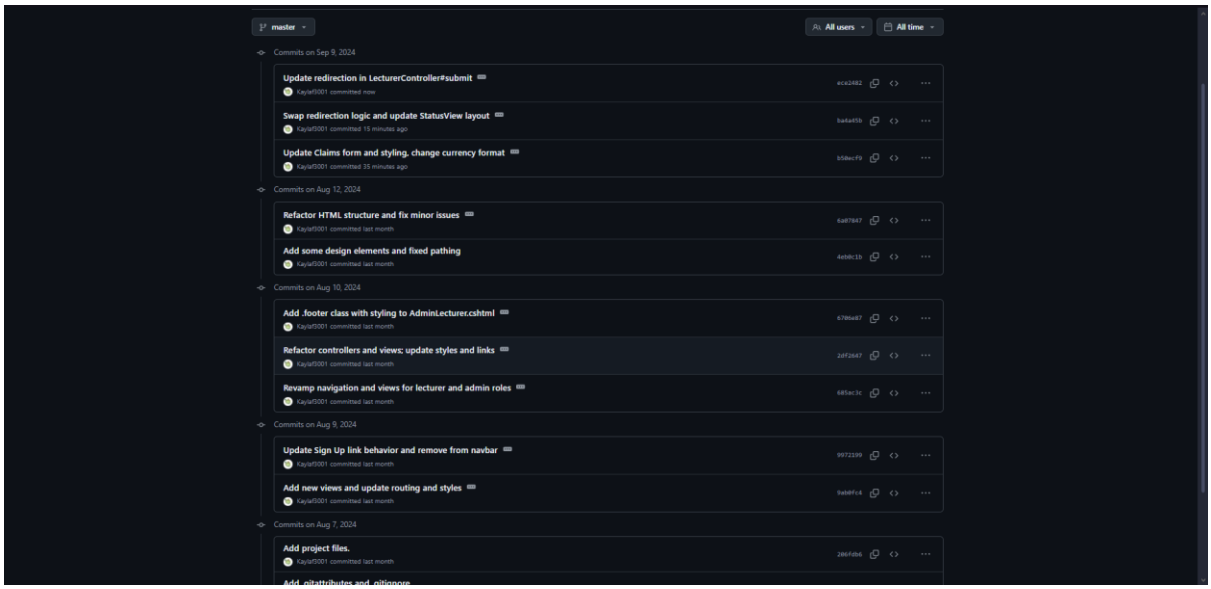
Phase 2: The design tasks depend on the completion of planning tasks to ensure alignment with the project's scope and objectives.

Phase 3: The development tasks build directly on the design work, requiring a completed design before implementation.

Phase 4: Involves reviewing and refining the prototype, dependent on the initial development's completion.

Phase 5: Final documentation and submission depend on the project's completion and review.

GitHub



GitHub Link

https://github.com/Kaylaf3001/PROG6212_MVC_POE_ST10259527.git

References

Armstrong, T. (2016, May 10). *Project Step #5: List Assumptions and Constraints*. Retrieved from The leadership program: https://www.tlpnyc.com/blog/project-step-5-list-assumptions-and-constraints

Eby, K. (2022, July 20). *How to Identify and Manage Dependencies in Project Management*. Retrieved from smartsheet: https://www.smartsheet.com/content/project-dependencies