**FIKILE SEKATI**

**ST10158923**

**Programming2B-PROG6212**

**Portfolio of Evidence (PoE)\_Part 1**

**GROUP 15**

**Yonela Nuba**

**Part 1 — Project Planning and Prototype Development:**

**My Contract Monthly Claim System**

**Design Choices**:

* The Contract Monthly Claim System (CMCS) will be developed using .NET Core to leverage cross-platform compatibility and modern features of the .NET ecosystem.
* The Model-View-Controller (MVC) pattern will be used to separate concerns, which allows for better code organization, scalability, and ease of testing.
* For the GUI, a clean, user-friendly interface will be prioritized, making it simple for users (lecturers, program coordinators, and academic managers) to navigate and complete tasks efficiently.

**Database Structure**:

* The database will use SQL Server, taking advantage of its robustness and integration with .NET Core.
* Tables will include Lecturers, Claims, ClaimApprovals, and SupportingDocuments:
  + **Lecturers**: Contains details of the lecturers, such as LecturerID, Name, Email, and HourlyRate.
  + **Claims**: Contains claim details with attributes such as ClaimID, LecturerID, HoursWorked, ClaimDate, and Status.
  + **ClaimApprovals**: Manages the approval process with attributes like ApprovalID, ClaimID, ApprovedBy, and ApprovalDate.
  + **SupportingDocuments**: Stores paths or references to documents uploaded by lecturers with attributes DocumentID, ClaimID, DocumentName, and UploadDate.

**GUI Layout**:

* The main dashboard will have navigation for different user roles:
  + **Lecturer Dashboard**: A section for submitting new claims, viewing claim history, and uploading supporting documents.
  + **Program Coordinator Dashboard**: A section for reviewing and approving claims.
  + **Academic Manager Dashboard**: A section to view all claims and their statuses, with additional functionalities to approve or reject claims.
* The interface will include input forms, submission buttons, and status indicators to make the user experience intuitive.

**Assumptions and Constraints**:

* All users will have access to the internet and the necessary permissions to use the application.
* Claims can only be submitted once per month per lecturer.
* The system is expected to handle a moderate number of users simultaneously without performance degradation.

**UML Class Diagram for the Contract Monthly Claim System:**

The UML class diagram will include the following classes to represent the data requirements of the system:

1. **Lecturer**
2. **Claim**
3. **ClaimApproval**
4. **SupportingDocument**

UML Class Diagram Description

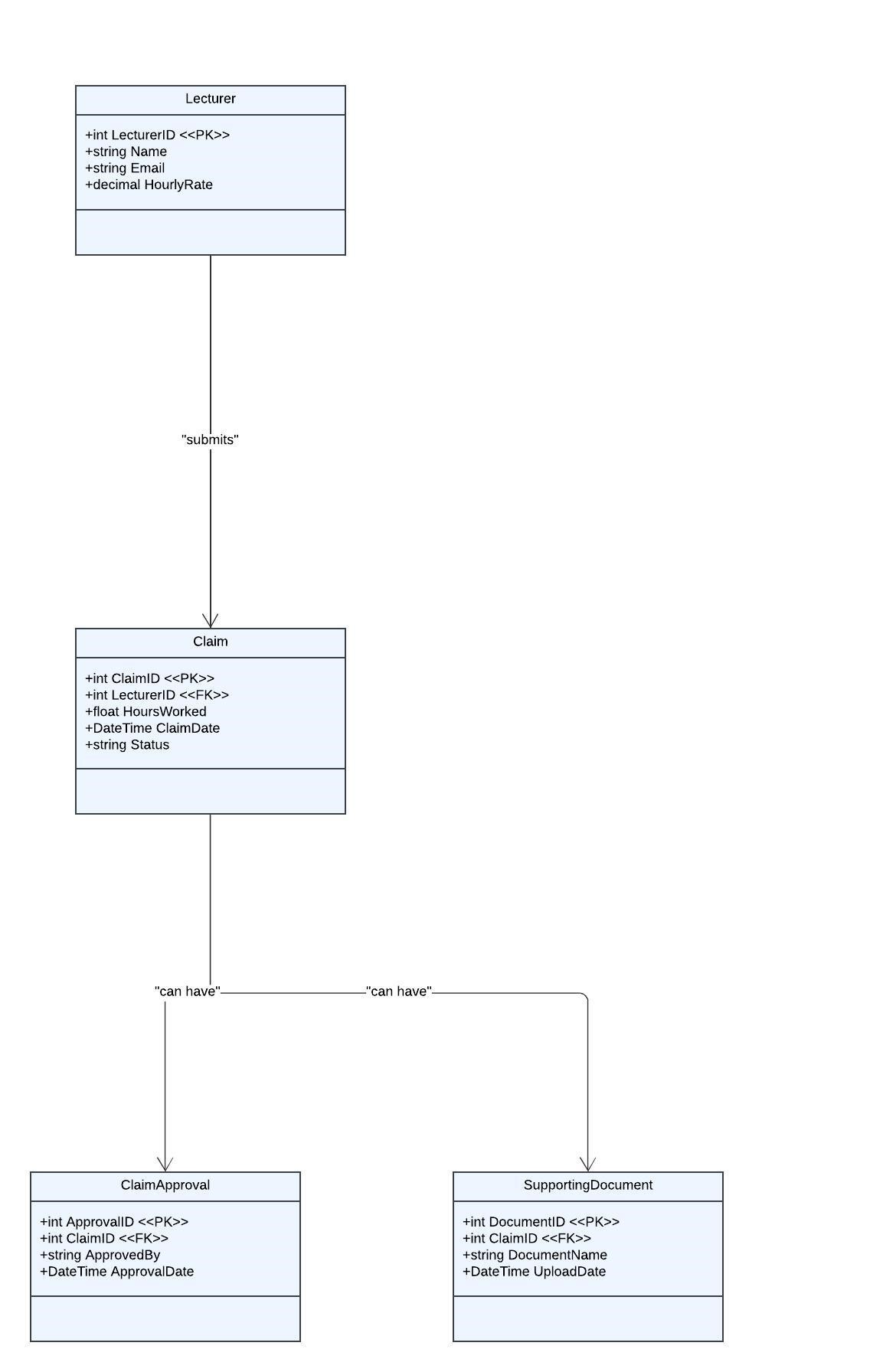
* **Lecturer**: Represents an independent contractor lecturer.
  + **Attributes**:
    - LecturerID: int - Unique identifier for each lecturer.
    - Name: string - Full name of the lecturer.
    - Email: string - Contact email of the lecturer.
    - HourlyRate: decimal - Hourly rate of payment for the lecturer.
* **Claim**: Represents a claim made by a lecturer for hours worked.
  + **Attributes**:
    - ClaimID: int - Unique identifier for each claim.
    - LecturerID: int - Foreign key referencing the lecturer who made the claim.
    - HoursWorked: float - Total hours worked for the claim period.
    - ClaimDate: DateTime - The date when the claim was submitted.
    - Status: string - The status of the claim (e.g., "Submitted", "Approved", "Rejected").
* **ClaimApproval**: Represents the approval or rejection of a claim.
  + **Attributes**:
    - ApprovalID: int - Unique identifier for each approval record.
    - ClaimID: int - Foreign key referencing the claim being approved or rejected.
    - ApprovedBy: string - Name of the person who approved or rejected the claim (could be a program coordinator or academic manager).
    - ApprovalDate: DateTime - The date when the claim was approved or rejected.
* **SupportingDocument**: Represents documents uploaded by a lecturer to support their claim.
  + **Attributes**:
    - DocumentID: int - Unique identifier for each supporting document.
    - ClaimID: int - Foreign key referencing the claim that the document supports.
    - DocumentName: string - Name or description of the document.
    - UploadDate: DateTime - The date when the document was uploaded.

**UML Class Diagram Relationships**

* **Lecturer** to **Claim**: One-to-Many relationship (one lecturer can submit multiple claims).
* **Claim** to **ClaimApproval**: One-to-Many relationship (one claim can have multiple approval records).
* **Claim** to **SupportingDocument**: One-to-Many relationship (one claim can have multiple supporting documents).

**Visual Representation of the UML Class Diagram**

Here’s a text-based representation of what the UML class diagram would look like:



**Project Plan for CMCS Protype Development**

A project plan typically includes the following components:

**1.Tasks**: Specific actions that need to be completed.

**2.Dependencies**: The relationship between tasks, indicating which tasks must be completed before others can begin.

**3.Timeline**: Estimated time required for each task, including start and end dates.

**4. Milestones**: Key points in the project when significant portions of work are completed.

**Project Plan Table**

Here’s an example of a project plan for your CMCS prototype:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task ID:** | **Task Description:** | **Dependencies:** | **Duration (Days):** | **Start Date:** | **End Date:** | **Milestone:** |
| 1 | Initial Planning and Setup | 2 | 8 | 09/03/2024 | 09/04/2024 | Project Initiation |
| 2 | Design Database Schema | 4 | 9 | 09/05/2024 | 09/07/2024 | UML Diagram Created |
| 3 | Create UML Class Diagram for Database | 6 | 9 | 09/06/2024 | 09/06/2024 | UML Diagram Created |
| 4 | Develop Project Plan Document | 3 | 0 | 09/08/2024 | 09/08/2024 | Project Plan Completed |
| 5 | Set Up GitHub Repository and Version Control | 2 | 8 | 09/09/2024 | 09/09/2024 | |  | | --- | |  |  |  | | --- | | GitHub Setup Completed | |
| 6 | Prepare Documentation and Screenshots for Report | 1 | 1 | 09/10/2024 | 09/12/2024 | GUI Layout Designed |
| 7 | Implement Basic GUI Prototype (XAML) | 2 | 9 | 09/06/2024 | 09/06/2024 | Basic GUI Prototype Implemented |
| 8 | Add Event Handlers and Simulate Interactions | 5 | 3 | 09/01/2024 | 09/02/2024 | Event Handlers Added |
| 9 | Review and Refine Prototype Based on Feedback | 8 | 2 | 09/07/2024 | 09/07/2024 | Prototype Refined |
| 10 | Prepare Documentation and Screenshots for Report | 9 | 1 | 09/03/2024 | 09/05/2024 | Documentation Completed |
| 11 | Finalize and Submit Part 1 of PoE (Portfolio of Evidence) | 10 | 1 | 09/05/2024 | 09/09/2024 | Submission of Part 1 |

**Report for the Contract Claim System (CMCS) Prototype**

**1.Introduction**

The Contract Monthly Claim System (CMCS) is a .NET-based application designed to streamline the process of submitting and approving monthly claims for independent contractor lecturers. This report outlines the design choices, database structure, and GUI layout for the CMCS prototype, providing insight into the system’s development process and highlighting key design considerations.

**2.Design Choices**

The CMCS prototype focuses on creating a user-friendly interface and a robust backend to manage the various aspects of claim submissions and approvals. The choice of **Windows Presentation Foundation (WPF) with XAML** for the GUI allows for a rich, interactive user experience while leveraging the .NET ecosystem's capabilities. This choice facilitates a responsive design that can be easily adapted for future enhancements.

For the backend, the **Entity Framework Core** was chosen for database management to provide a seamless object-relational mapping (ORM) experience. This enables easy manipulation of data as objects within the application while maintaining high performance and scalability. The decision to use **C#** as the primary programming language aligns with .NET's powerful features, such as strong typing and asynchronous programming, which are crucial for developing a responsive and reliable application.

**3.Structure of the Database**

The database for CMCS is designed to efficiently manage lecturer information, claims, approvals, and supporting documents. The structure is organized around four main entities: **Lecturer, Claim, ClaimApproval,** and **SupportingDocument**.

* **Lecturer**: This table stores details about each lecturer, including a unique identifier (LecturerID), name, email, and hourly rate. The lecturer data is fundamental to the system, as each claim submission is associated with a specific lecturer.
* **Claim**: This table captures the details of each claim, such as ClaimID, LecturerID (foreign key), hours worked, claim date, and status. The relationship between Lecturer and Claim is one-to-many, meaning each lecturer can have multiple claims.
* **ClaimApproval**: This table records the approval or rejection status of each claim. It includes an ApprovalID, ClaimID (foreign key), the name of the person who approved or rejected the claim, and the date of approval or rejection. This setup allows tracking of each claim’s approval process through a one-to-many relationship with the Claim table.
* **SupportingDocument**: This table holds information about the documents uploaded to support a claim, with attributes like DocumentID, ClaimID (foreign key), document name, and upload date. This structure ensures that each claim can have multiple associated supporting documents.

The database design emphasizes relational integrity and normalization to reduce redundancy and ensure data consistency across the system.

**4.Layout of the GUI**

The GUI of the CMCS prototype is designed using **WPF and XAML** to ensure a clean, intuitive, and responsive user experience. The main layout consists of several key components:

* **Dashboard**: The main interface where lecturers can view their submitted claims and their statuses. This section provides quick access to common actions like submitting a new claim or uploading supporting documents.
* **Claim Submission Form**: A dedicated form for entering details about new claims, such as hours worked and the date range. This form includes input validation to ensure data accuracy and completeness.
* **Approval Interface**: A separate interface for Programmed Coordinators and Academic Managers, where they can view pending claims, approve, or reject them, and add comments. This interface provides filtering and sorting options to manage claims efficiently.
* **Document Upload Section**: A user-friendly interface for uploading supporting documents, with features to preview documents before submission and check the file format and size constraints.

Each component of the GUI is designed to be **intuitive and accessible**, with consistent styling and navigational elements to enhance usability. The layout also includes feedback mechanisms, such as confirmation dialogs and status indicators, to guide users through their interactions with the system.

6.Conclusion

The CMCS prototype development involves a structured approach to database design and GUI layout, focusing on usability, data integrity, and maintainability. The design choices made, from the use of WPF and XAML for the front end to Entity Framework Core for the back end, are geared towards creating a scalable and efficient application. This report has outlined the foundational elements of the prototype, providing a clear pathway for future development and functionality enhancements.

By combining robust database architecture with a user-centered interface, the CMCS aims to streamline administrative processes and improve the efficiency of managing monthly claims, benefiting both lecturers and administrators.

**Reference List:**

•Microsoft Documentation, "ASP.NET Core MVC", https://docs.microsoft.com/en-us/aspnet/core/mvc

•Freeman, A., & Sanderson, S., Pro ASP.NET Core MVC 2, Apress, 2017.

•"Database Design Best Practices," Stack Overflow, https://stackoverflow.com

•"ASP.NET Core MVC Crash Course," YouTube, Code Maze, https://www.youtube.com/watch?v=BfEjDD8mWYg

•"Learn ASP.NET Core MVC (.NET 5) - Full Course," YouTube, FreeCodeCamp.org, <https://www.youtube.com/watch?v=fmvcAzHpsk8>

**Final Commit Log**

* Initial ASP.NET Core MVC project setup
* Add Claim model with properties for claim submission
* Add ClaimController with Index, Submit, and Details actions
* Add views for displaying claims list, submit form, and claim details
* Add custom CSS and Bootstrap for improved UI styling