**ST10448839 – PROG6212 – POE Part 1 – Raihaan Rajah**

**Design Choices:**

* **Technology:** WPF (.NET Core) chosen for the prototype instead of MVC. WPF is better suited for desktop applications with a rich GUI, and it allows advanced layout, styling, and data binding. This matches the POE requirement while keeping the prototype **non-functional** at this stage.
* **Architecture:** Using **MVVM (Model-View-ViewModel)** pattern to separate UI (XAML), business logic, and data structures. This makes it easier to bind controls to future backend logic.
* **Styling & Layout:** WPF **Grids**, **StackPanels**, and **DockPanels** used for layout. Consistent theme with status colours (badges for Draft/Submitted/Approved).
* **User Experience:** Emphasis on simplicity — lecturers see only “Submit Claim” and “Claim History,” while coordinators/managers see “Verify” and “Approve.” Navigation provided via a simple **Menu bar** or **TabControl**.

**The Structure of the Database:**

* **Lecturer**
  + LecturerId (PK), StaffNo, FirstName, LastName, Email
* **Claim**
  + ClaimId (PK), LecturerId (FK), Month, Status (Draft/Submitted/Approved/etc.), CreatedAt, SubmittedAt, VerifiedBy, ApprovedBy
* **ClaimLine**
  + LineId (PK), ClaimId (FK), ActivityType, Hours, HourlyRate, Notes
* **Document**
  + DocumentId (PK), ClaimId (FK), FileName, FileType, UploadedAt
* **User**
  + UserId (PK), Name, Email, Role (Lecturer, Coordinator, Manager)

This database model ensures **one-to-many** between Lecturer → Claim, and Claim → ClaimLines/Documents.

**Layout of the GUI:**

**Main Window (Navigation Menu)**

* Menu strip at the top with role-based options (Submit Claim, Claim History, Review Claims).
* Status bar at bottom showing current user role and system date.

**Submit Claim Window (Lecturer)**

* DatePicker for selecting the claim month.
* DataGrid for entering line items (Activity, Hours, Hourly Rate, Notes).
* “Add Row” button to add more activities.
* File upload placeholder (Button → “Attach Document”).
* Buttons: **Save Draft** | **Submit Claim** (disabled for now).

**Claim History Window (Lecturer)**

* DataGrid listing previous claims with columns: ClaimId, Month, Status, SubmittedAt.
* Status shown with coloured badges (e.g., Green = Approved, Orange = Pending).

**Review Queue Window (Coordinator/Manager)**

* DataGrid listing submitted claims awaiting verification.
* Double-click row → opens **Claim Detail View** showing line items + documents.
* Buttons: Verify / Approve / Reject (disabled placeholders).

**Shared UI Elements**

* Consistent WPF **Styles** for buttons, labels, and badges.
* Tooltip hints: e.g., hovering over Submit shows “Functionality to be added later.”

**Assumptions:**

* Each lecturer submits one claim per month, multiple activities allowed.
* Authentication is mocked (role chosen manually for demo).
* File upload is simulated (no actual storage).
* Claim total calculations will be implemented later.

**Constraints:**

* **Prototype only:** All buttons and grids are placeholders; no DB connection.
* Must use **WPF XAML layout**, not MVC.
* UI must clearly communicate role differences, but without backend logic.
* Accessibility considered (labels, tab order, high-contrast themes).

**UML Class Diagram:**

**Claim**

ClaimID: int (PK)

LecturerID: int (FK)

Month: Date

Status: string

SubmitAt: DateTime

**Lecturer**

LecturerID: int (PK)

FirstName: string

LastName: string

Email: string

1..

**Approver**

ApproverID: int (PK)

Name: string

Role: string

\*..1

1..

1..

1..

**ClainItem**

ItemId: int (PK)

ClaimID: int (FK)

Activity: string

Hours: double

Rate: decimal

Notes: string

\*..1

**ClaimApproval**

ApprovalID: int (PK)

ClaimID: (FK)

ApproverID: int (FK)

Action: string

ActionDate: DateTime

**SupportingDocument**

DocumentID: int (PK)

ClaimID: int (FK)

FileName: string

FilePath: string

UploadedAt: DateTime

**Project Plan:**

The development of the Contract Monthly Claim System (CMCS) prototype will follow a structured project plan to ensure clarity and timely delivery. The first phase focuses on requirements gathering and analysis, where the scope of the system, user roles, and necessary features are defined. Once requirements are clear, the second phase involves database design, where entities such as Lecturers, Claims, Supporting Documents, and Approvals are modelled into a UML class diagram. The third phase is GUI design using WPF, focusing on creating an intuitive and user-friendly prototype interface with tabs for submitting claims, uploading documents, viewing claim history, and reviewing claims. The GUI at this stage will remain non-functional but visually representative of the intended system. The fourth phase will involve creating the project structure in Visual Studio Code 2022 and organizing the source code with proper version control through GitHub, including regular commits with descriptive messages. Throughout the process, constraints such as limited functionality at this stage and reliance on WPF for front-end design will be considered. By following this plan, the Part 1 prototype will provide a clear foundation for subsequent functional implementation in later phases of the POE.

**Below is a table that summarises the project plan for easier understanding:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task: | Owner: | Duration: | Dependencies: | Planned Dates: |
| Requirements & wireframes | Raihaan | 2 Days | None |  |
| UML database class diagram | Raihaan | 1 Day | Wireframes |  |
| Create WPF solution skeleton | Raihaan | 1 Day | UML Diagram |  |
| Design XAML windows (Submit Claim, Claim History, Review Queue) | Raihaan | 3 Days | Skeleton of Solution |  |
| Add placeholder controls (DataGrids, buttons, file uploads) | Raihaan | 2 Days | XAML Windows |  |
| Apply styling, layout adjustments, and accessibility tweaks | Raihaan | 2 Days | Placeholoders |  |
| Prepare Part 1 report, screenshots, and Git commits | Raihaan | 1 Day | All Prior |  |

**Designing the GUI using WPF:**

I have attached the WPF file (with code) to this ZIP folder but below is a screenshot of the GUI.

