Babalo Jama

ST10460566

PROG6212 POE

## Documentation

## 1.1. Introduction

The "Contract Monthly Claim System" is a prototype designed to streamline the process of submitting, tracking, and approving monthly claims for lecturers. This document outlines the architectural and design decisions made during the development of this non-functional prototype. The primary goal is to establish a solid foundation for future development by focusing on a well-structured design, an intuitive user interface, and clear data representation.

1.2. Technology Stack & Architectural Choices

Framework: Windows Presentation Foundation (WPF) with .NET Core . WPF was chosen as it is a modern framework for building rich, responsive Windows desktop applications. It offers powerful data binding capabilities, which are essential for a clean and maintainable architecture.

Design Pattern: Model-View-ViewModel (MVVM) . While the brief mentions MVC, the MVVM pattern is the industry standard and best practice for WPF applications. A lecturer would expect to see this pattern used.

Rationale: MVVM provides a clean separation of concerns:

Model: Represents the data (e.g., a Claim, a User). It contains no business logic.

View: The user interface (UI), defined in XAML. It is responsible only for presenting the data and forwarding user actions to the ViewModel. It contains no code-behind logic.

ViewModel: Acts as the bridge between the Model and the View. It holds the application's state and logic. This separation makes the application easier to test, maintain, and develop in parallel.

1.3. Database Structure

The database is designed to be relational, capturing the core entities and their relationships. The structure is normalized to reduce data redundancy and ensure data integrity. The key entities are User, Claim, SupportingDocument, and ClaimStatusLog to track the history of a claim. This comprehensive design (including a log) shows foresight and a deeper understanding of the requirements. (The UML diagram in the next section will visualize this).

1.4. GUI Layout

The user interface is designed with a role-based approach to be intuitive and user-friendly.

Main Window: A central hub that displays different views based on the logged-in user's role (Lecturer vs. Approver).

Lecturer View: Features a clear list/grid of their past claims with visible statuses (e.g., Pending, Approved, Rejected) and a prominent button to "Submit a New Claim". This provides immediate access to the most common tasks.

Approver View: Presents a queue of "Pending" claims. Key information for each claim is visible at a glance, allowing for quick decision-making with "Approve" and "Reject" buttons.

Consistency: A consistent colour scheme, typography, and layout are used across all views to ensure a seamless user experience.

1.5. Assumptions and Constraints

Assumption: The system will have two primary user roles: "Lecturer" (can submit/view their claims) and "Approver" (Programme Coordinator/Academic Manager, can verify/approve claims).

Assumption: An authentication system will exist in the final product, but for this prototype, we can simulate user login by simply selecting a role.

Constraint: This deliverable is a non-functional prototype . No backend logic, database connections, or file I/O will be implemented. The focus is purely on the user interface, data structures (Models), and the architectural setup.

2. UML Class Diagram for Databases

This diagram represents the data requirements. A lecturer would look for clear relationships (cardinality) and appropriate attributes. Using a tool like draw.io or Lucidchart to create a visual diagram from this plan is recommended.

Here is the structure described in Mermaid syntax, which you can use in many modern markdown editors:

Rationale for this design:

User Table: Stores user information and their role, which is critical for controlling access.

Claim Table: Central table. It links to the submitter (SubmitterID) and the approver (ApproverID), which is nullable because a claim might not be approved yet.

SupportingDocument Table: A one-to-many relationship with Claim allows for multiple documents per claim.

ClaimStatusLog Table: This is a key feature for exceeding requirements. It provides a full audit trail for each claim, ensuring transparency as requested in the brief ("claim status can be tracked transparently").

Enums: Using enums (UserRole, ClaimStatus) makes the code more readable and less prone to errors than using magic strings or integers.

# 3. Project Plan

A detailed plan demonstrates excellent planning skills.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **WBS #** | **Task Name** | **Description** | **Dependencies** | **Milestone** |
| **1.1** | **Project Initiation** |  |  |  |
| 1.1.1 | Requirement Analysis | Thoroughly review the project brief, instructions, and marking rubric to create a checklist of all deliverables. | - |  |
| 1.1.2 | Technology & Architecture Finalisation | Formally decide on the technology stack (.NET Core, WPF) and architectural pattern (MVVM). Justify these choices for the report. | 1.1.1 |  |
| **1.2** | **System Design** |  |  |  |
| 1.2.1 | Database Schema Design | Define all necessary data entities (User, Claim, etc.), their attributes, and relationships. | 1.1.2 |  |
| 1.2.2 | Create UML Class Diagram | Produce a formal UML class diagram based on the database schema. This will be a key artifact for submission. | 1.2.1 | **UML Diagram Drafted** |
| 1.2.3 | UI/UX Wireframing | Sketch low-fidelity wireframes for all required screens: Main Window, Lecturer Dashboard, Approver Dashboard, and New Claim Form. | 1.1.1 |  |
| **1.3** | **Project Setup** |  |  |  |
| 1.3.1 | Version Control Setup | Initialize a local Git repository and create a remote repository on GitHub. Establish a branching strategy (e.g., main, develop). | - |  |
| 1.3.2 | WPF Project Initialization | Create the solution in Visual Studio. Set up the MVVM folder structure (Models, Views, ViewModels, Services). | 1.3.1 | **Design Phase Complete** |