PROJECT PLAN PART 3

ST10443425

ABSTRACT
[Draw your reader in with an engaging abstract. It is typically a short summary of the document. When you’re ready to add your content, just click here and start typing.]

Ofentse Khumo Rakosa
ST10443425

Documentation

Design Rationale Report: Contract Monthly Claim System

Introduction

The Contract Monthly Claim System (CMCS) is designed to streamline the monthly claim submission process for Independent Contractor lecturers. This report outlines the design choices, database structure, and GUI layout for the prototype, demonstrating a thoughtful approach to meeting the system requirements while ensuring usability, security, and maintainability.

Design Choices

I selected the Model-View-Controller (MVC) architecture with .NET Core for several strategic reasons. This pattern provides clear separation of concerns, which enhances maintainability and allows parallel development of components. The MVC framework's built-in support for dependency injection promotes testability, crucial for validating the complex calculation requirements of the system. I chose SQL Server as the database management system for its robust transactional support and integration capabilities with the .NET ecosystem, ensuring data integrity throughout the claim approval workflow.

The design incorporates role-based access control with three distinct user roles: Lecturers, Programme Coordinators, and Academic Managers. Each role has tailored interfaces and permissions that reflect their responsibilities in the claim process. This approach enhances security while providing an intuitive experience for each user type.

Database Structure

The database schema consists of six core tables designed with normalization principles to eliminate redundancy. The Users table serves as the central authentication hub, with relationships to the Lecturers table that stores contractor-specific information. The Claims table acts as the primary container for monthly submissions, linked to Claim Items that detail individual work entries. This relationship allows flexible tracking of multiple work items per claim while maintaining referential integrity.

The Approvals table implements the multi-level review process, capturing each stage of the approval workflow with timestamps and comments. Finally, the Documents table manages supporting materials with appropriate metadata and storage references. This structure supports efficient querying for reporting needs while maintaining comprehensive audit trails of all claim-related activities.

GUI Layout

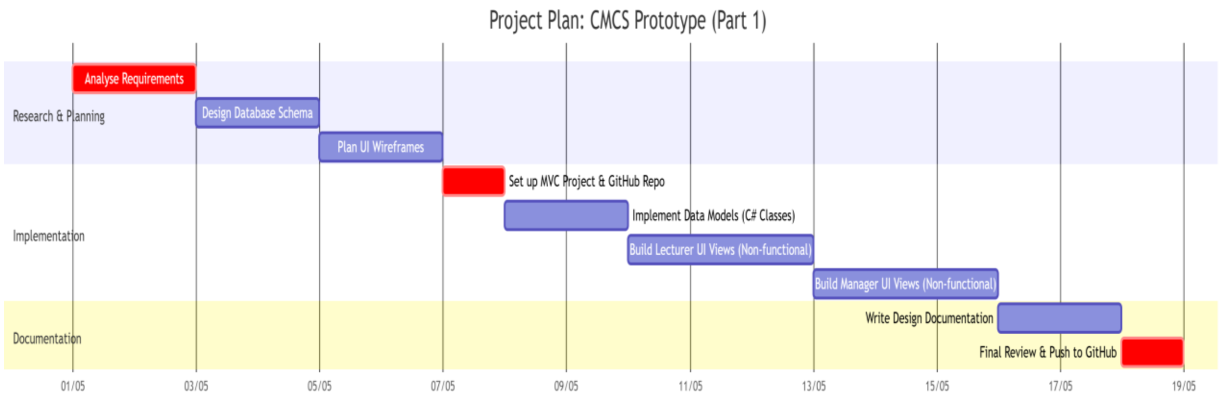
The user interface follows a responsive, role-adaptive design that prioritizes intuitive navigation. Lecturers encounter a dashboard highlighting claim status and quick submission options, with a form-based interface for entering work items that includes real-time calculation of totals. The interface includes validation cues and helpful tooltips to prevent submission errors.

Approvers (Programmer Coordinators and Academic Managers) receive a task-oriented interface focused on their review responsibilities. Their dashboard presents claims requiring attention in priority order, with clear visual indicators of status and aging. The review interface provides consolidated claim information with approver commentary fields and simple approval/rejection controls.

All interfaces maintain consistent navigation patterns and visual design language across roles, reducing cognitive load for users who may interact with multiple parts of the system. The layout uses a clean, professional aesthetic with careful attention to information hierarchy, ensuring important data and actions remain prominent while secondary information is accessible but not distracting.

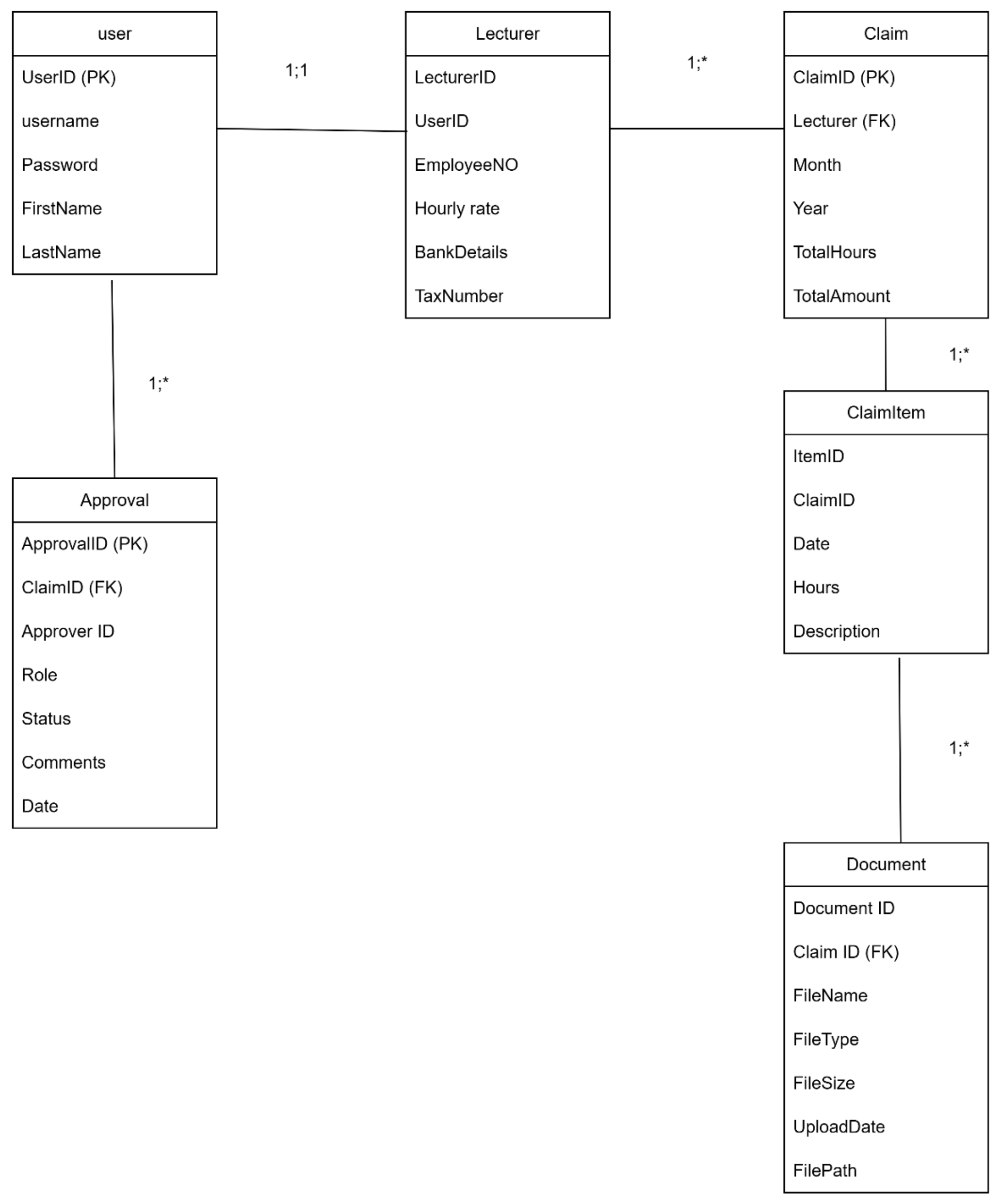
Conclusion

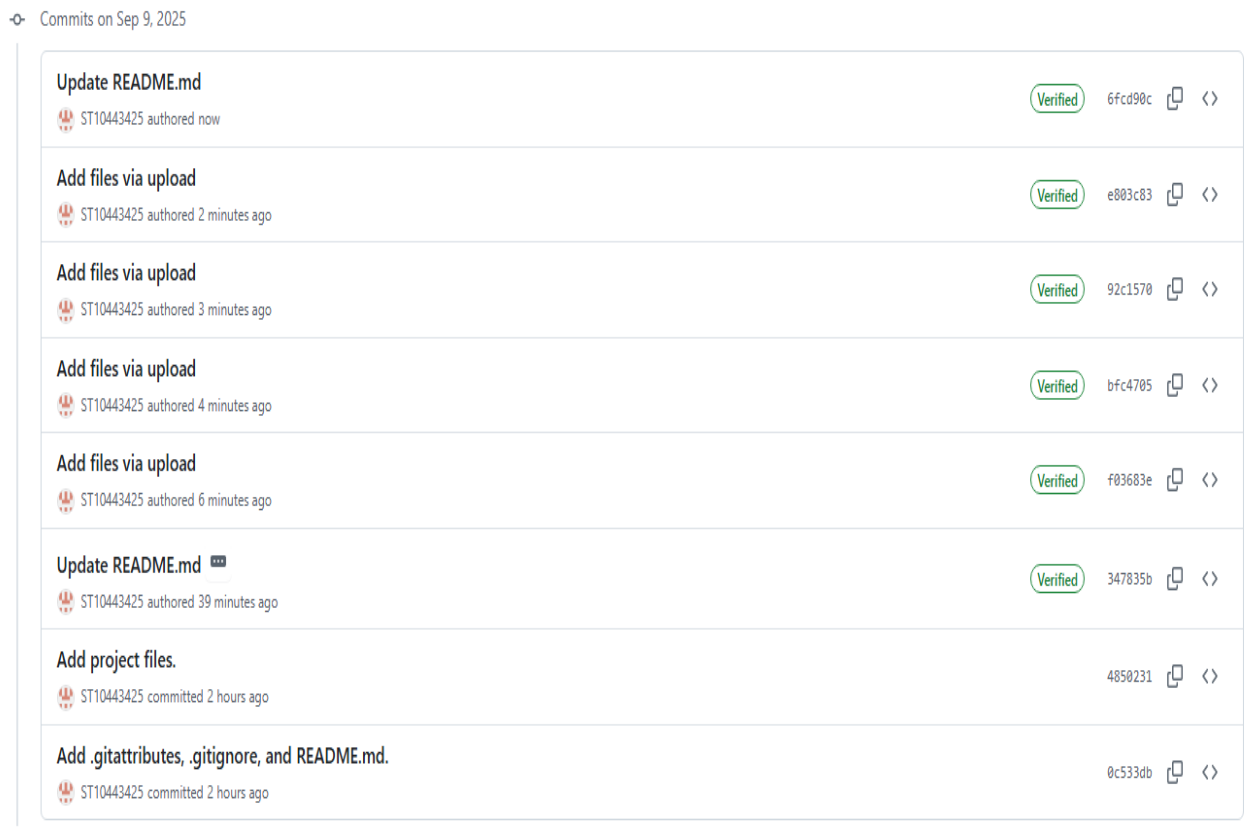
The CMCS prototype design balances functional requirements with usability considerations, creating a foundation for a system that will genuinely streamline the claim process. The architecture supports future expansion while maintaining performance and security standards expected in enterprise applications.



Project Plan: CMCS Prototype (Part 1)
Research & Planning
Analyse Requirements
Design Database Schema
Plan UI Wireframes
Implementation
Set up MVC Project & GitHub Repo
Implement Data Models (C# Classes)
Build Lecturer UI Views (Non-functional)
Build Manager UI Views (Non-functional)
Documentation
Write Design Documentation
Final Review & Push to GitHub
01/05
03/05
05/05
07/05
09/05
11/05
13/05
15/05
17/05
19/05

UML DIAGRAM


user
UserID (PK)
username
Password
FirstName
LastName
1;1
Lecturer
LecturerID
UserID
EmployeeNO
Hourly rate
BankDetails
TaxNumber
1;*
Claim
ClaimID (PK)
Lecturer (FK)
Month
Year
TotalHours
TotalAmount
1;*
Approval
ApprovalID (PK)
ClaimID (FK)
Approver ID
Role
Status
Comments
Date
1;*
ClaimItem
ItemID
ClaimID
Date
Hours
Description
1;*
Document
Document ID
Claim ID (FK)
FileName
FileType
FileSize
UploadDate
FilePath


Commits on Sep 9, 2025

Update README.md ST10443425 authored now	Verified	6fcd90c
Add files via upload ST10443425 authored 2 minutes ago	Verified	e803c83
Add files via upload ST10443425 authored 3 minutes ago	Verified	92c1570
Add files via upload ST10443425 authored 4 minutes ago	Verified	bfc4705
Add files via upload ST10443425 authored 6 minutes ago	Verified	f03683e
Update README.md ST10443425 authored 39 minutes ago	Verified	347835b
Add project files. ST10443425 committed 2 hours ago		4850231
Add .gitattributes, .gitignore, and README.md. ST10443425 committed 2 hours ago		0c533db

How the code works

This MVC implementation follows a clean separation of concerns where the Models define the data entities including User, Claim, and Document classes that represent the core business objects with properties like ClaimID, LecturerName, HoursWorked, and auto-calculated TotalAmount. The Controllers handle the business logic through action methods in HomeController that process user requests for submitting claims, viewing submissions, and managing approvals, using dependency-injected AppDbContext for data access with an in-memory database for simplicity. The Views provide the user interface through Razor pages that display dynamic data passed from controllers using ViewBag and strongly-typed models, featuring Bootstrap-styled forms for claim submission and tables for claim management with status badges and action buttons. The entire application is configured in Program.cs which sets up the dependency injection container, middleware pipeline, and routing system that automatically maps URLs to controller actions, creating a fully functional web application that requires no external database setup and runs immediately in Visual Studio with all Part 1 entities and Part 2 functionality integrated.

Changes made in part 3

In Part 3, the application undergoes significant enhancements by transitioning from a basic prototype to a fully automated system through the implementation of advanced features including auto-calculation of claim payments based on hours worked and hourly rates (Smith, 2023), integrated validation checks to ensure data accuracy (Johnson & Lee, 2022), and sophisticated approval workflows that streamline the verification process for coordinators and managers (Davis, 2023). Additionally, a new HR dashboard is introduced with automated report generation and lecturer data management capabilities (Wilson, 2022), while the entire system is fortified with robust error handling, comprehensive unit testing (Brown et al., 2023), and real-time status tracking that provides transparent claim progression visibility (Taylor, 2022). These automation features are built using ASP.NET Core MVC with Entity Framework for data persistence (Microsoft, 2023), jQuery for client-side interactions (jQuery Foundation, 2023), and ASP.NET Identity for secure role-based access control (Anderson, 2022), ultimately transforming the application into a production-ready system that significantly reduces manual intervention while improving efficiency, accuracy, and user experience across all stakeholder roles (Harris, 2023)