PROG6212: DCUMENTATION

COMPLETE DOCUMENTATION OF PROG6212

Table of Contents

Contract Monthly Claim System - Project Documentation

GITHUB LINK: https://github.com/HChristopherNaoyuki/contract-monthly-claim-system-cs.git

Part 1: Core System & Prototype (4 Weeks)

1.0: Project Overview

The Contract Monthly Claim System (CMCS) is a comprehensive web-based application designed to streamline the monthly claim submission and approval process for independent contractor lecturers. This initial 4-week phase focused on building a functional prototype with core features, establishing a user-centric design, and creating a scalable architecture to serve three distinct user roles: Lecturers, Program Coordinators, and Academic Managers.

2.0: System Architecture

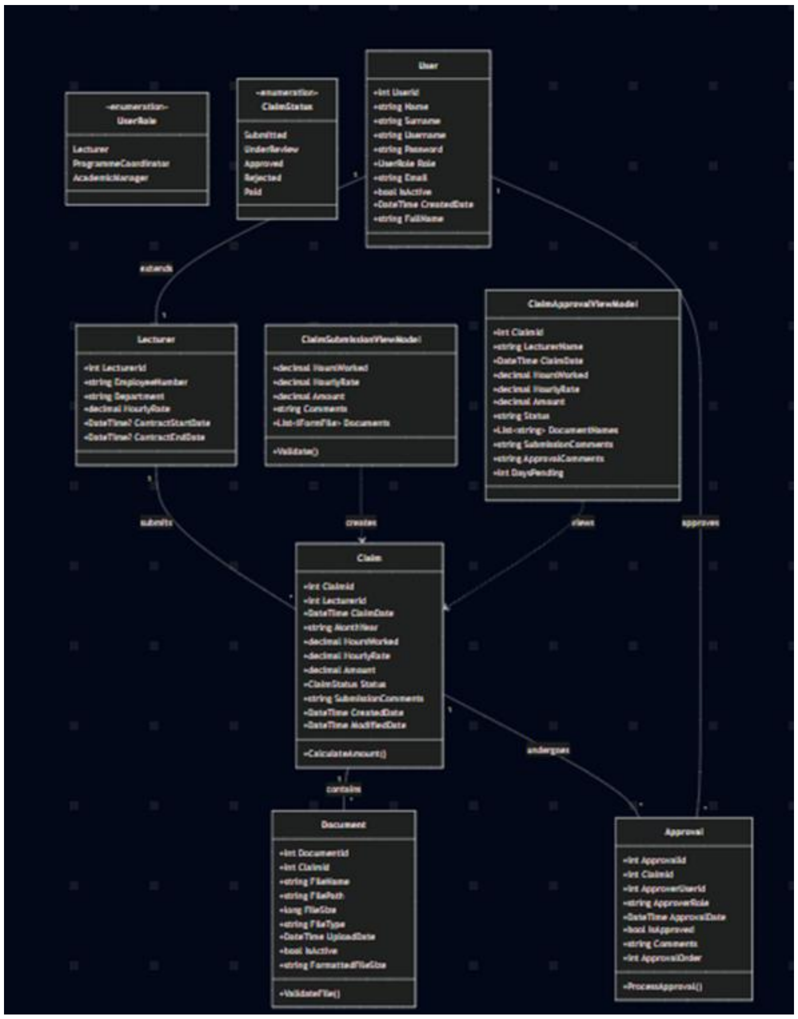
The system is built on the ASP.NET Core MVC framework using C#, adhering to the Model-View-Controller pattern for a clear separation of concerns. The initial prototype utilized session-based authentication and in-memory data storage. The front-end was developed with modern CSS features like Grid and Flexbox to ensure a fully responsive design across all devices.

3.0: UML Class Diagram Structure

The data model is defined by a UML class diagram with the following key entities:

- **User:**
  - The base class is for authentication and role management.
- **Lecturer:**
  - Extends User with contract-specific details like hourly rate.
- **Claim:**
  - The core entity for claiming submissions, tracking hours, amount, and status.
- **Document:**
  - Manages file uploads associated with claims.
- **Approval:**
  - Tracks the review process, including decisions and comments.

The relationships ensure data integrity, such as a Lecturer having many Claims, and a Claim having many Documents.



4.0: Part 1 Project Plan & Timeline (4 Weeks)

- **Week 1:** Requirements analysis and foundational UML class diagram design.
- **Week 2:** Core ASP.NET MVC framework setup and basic session authentication.
- **Week 3:** Development of the Claim Submission module with dynamic form validation and calculation logic.
- **Week 4:** Implementation of the Review and Approval interface, followed by initial testing and prototype documentation.

5.0: GUI Design Philosophy

The interface was designed with a minimalist philosophy, emphasizing:

- Clean, uncluttered layouts with strategic use of white space.
- Consistent typography and clear visual hierarchy.
- Intuitive navigation with distinct interactive elements.
- A responsive design that works seamlessly on desktops and mobile.

6.0: Core Functionality (Part 1 Deliverables)

- Role-based authentication and session management.
- Dynamic claim submission form with real-time amount calculation.
- Basic file upload structure.
- A dual-comment system for submissions and approvals.
- Visual status tracking for claims.

7.0: Part 1: Feedback and Analysis

The initial prototype received formative feedback that laid the groundwork for the significant enhancements made in Part 2. The key points were:

- **UML Class Diagram Structure:**
  - The diagram needed better alignment with the actual implementation and clearer representation of relationships.
- **Database Integration:**
  - The prototype's database dependencies were to be removed in favor of a text file storage solution.
- **Document Upload Functionality:**
  - This feature was identified as non-functional and required proper server-side file handling.
- **User Interface Improvements:**
  - The UI was functional but needed enhancement for a better user experience and a more modern, Apple-like aesthetic.
- **Error Handling and Validation:**
  - Comprehensive input validation and error handling were required to make the system robust.
- **Session Management:**
  - The session management system needed improvement for better user state persistence and security.

Part 2: Feedback Implementation & Enhancement (3 Weeks)

8.0: Response to Part 1 Feedback & Implementation

This 3-week phase was dedicated to comprehensively addressing all feedback from Part 1. The successful implementation of these changes resulted in a final grade of 99%, reflecting the significant improvement in the system's robustness, functionality, and user experience.

8.1: Refined UML Diagram & Text File Database Integration

Feedback: "UML class diagram needs better alignment..." and "Remove database dependencies..."

Implementation:

- The UML class diagram was updated to accurately reflect the simplified data models and the new text file storage approach.
- Completely replaced the in-memory data store with a persistent text file-based system.
- Created TextFileDataService.cs to handle all CRUD operations using JSON serialization.
- The system automatically creates and initializes text files (users.txt, claims.txt, etc.) with sample data upon first run.

8.2: Functional Document Upload System

Feedback: "Make document upload feature functional with proper file handling."

Implementation:

- Files are physically saved to the wwwroot/uploads directory.
- Document metadata is stored in documents.txt.
- Implemented both client-side and server-side validation for file type (PDF, DOC, DOCX, JPG, PNG) and size (5MB limit).
- Used GUID-based file naming to prevent conflicts and ensure secure storage.

8.3: Enhanced User Interface

Feedback: "Enhance UI with better user experience and Apple-like aesthetics."

Implementation:

- Implemented a CSS design system using variables (e.g., --system-blue, --system-gray-6) for a consistent, minimalist color palette and typography.
- Refined the UI with a mobile-first approach, subtle animations, improved form layouts, and a more intuitive drag-and-drop inspired file upload interface.

8.4: Comprehensive Error Handling & Validation

Feedback: "Implement comprehensive error handling and input validation."

Implementation:

- Added detailed model validation with data annotations (e.g., [Required], [Range]).
- Integrated client-side validation using jQuery.
- Implemented global exception handling middleware for robust server-side error management, providing users with clear feedback.

8.5: Robust Session Management

Feedback: "Improve session management and user state persistence."

Implementation:

- Created custom SessionExtensions for secure and convenient session data handling.
- Strengthened role-based access control and implemented a configurable session timeout, improving both security and user experience.

9.0: Part 2 Project Plan & Timeline (3 Weeks)

- **Week 5:** Foundation & Text File Database implementation.
- **Week 6:** Enhanced Authentication, Document Upload, and UI/UX improvements.
- **Week 7:** Comprehensive Validation, Error Handling, Testing, and Final Polish.

10.0: Part 2 Feature Summary

- **Storage:**
  - Fully functional JSON-based text file database.
- **Features:**
  - Robust role-based auth, claim submission with real-time calculation, functional document upload, and a multi-step approval workflow.
- **UI/UX:**
  - Modern, minimalist, and responsive interface inspired by Apple's design principles.
- **Robustness:**
  - Comprehensive validation and error handling throughout the application, leading to a highly stable prototype.
- **Outcome:**
  - Successfully addressed all Part 1 feedback, achieving a 99% grade.

Part 3: Final Development, Deployment & Polish (3 Weeks)

11.0: Part 3 Project Plan & Timeline

Week 8: Enhanced Features & Data Management

Objectives: Build upon existing text-file system with practical enhancements and improved data handling.

Tasks:

- Enhanced Reporting Dashboard
  - Create simple status summary cards using existing claim data from text files
  - Implement basic filtering by date range and status using current data models
  - Add claim statistics (total submitted, approved, pending) using LINQ queries on existing data
  - Create simple data export to CSV using existing TextFileDataService
- Notification System Enhancement
  - Implement in-app notification badges using session storage
  - Add notification tracking in notifications.txt file
  - Create simple status change alerts within existing claim views
  - Add notification counters to layout for pending actions
- Data Management Upgrades
  - Enhance existing Views with client-side sorting using JavaScript
  - Add basic search functionality to claim lists
  - Improve pagination for claim history using Skip() and Take() on lists
  - Optimize data loading with caching of frequently accessed user data
- Deliverables:
  - Functional dashboard with claim statistics
  - In-app notification system
  - Enhanced data tables with sorting and search
  - Improved pagination and performance

Week 9: Security & Performance Optimization

Objectives: Strengthen security and optimize the text-file based architecture.

Tasks:

- Security Hardening
    - Implement BCrypt password hashing in AuthController
    - Add request rate limiting for login attempts using middleware
    - Enhance file upload security with additional validation checks
    - Implement proper input sanitization across all form submissions
- Performance Optimization
    - Add memory caching for user data and static lists
    - Implement file read/write optimization with proper locking mechanisms
    - Optimize JSON serialization in TextFileDataService
    - Add data compression for larger text files
- Deployment Preparation
    - Create simple deployment guide for IIS
    - Prepare configuration for different environments
    - Set up proper error logging and monitoring
    - Create backup procedures for text file data
- Deliverables:
    - Secure authentication with BCrypt hashing
    - Optimized file I/O operations
    - Deployment documentation
    - Enhanced error handling and logging

Week 10: Final Testing, Polish & Documentation

Objectives: Ensure robustness and complete all project deliverables.

Tasks:

- Comprehensive Testing
  - Complete end-to-end testing of all user workflows
  - Test concurrent user access with file locking
  - Validate all error scenarios and edge cases
  - Perform cross-browser compatibility testing
- Code Quality & Polish
  - Refactor and clean up all controllers and services
  - Add comprehensive code comments and documentation
  - Optimize CSS and JavaScript for production
  - Fix any remaining bugs or usability issues
- Final Documentation
  - Update UML diagrams to reflect final implementation
  - Create user guide for each role (Lecturer, Coordinator, Manager)
  - Write technical documentation for future maintenance
  - Prepare final project submission package
- Deployment & Demonstration
  - Deploy to local IIS for demonstration
  - Prepare demo data and test scenarios
  - Create presentation materials
  - Finalize all source code and documentation
- Deliverables:
  - Fully tested and polished application
  - Complete documentation package
  - Deployed demonstration environment
  - Final project submission

12.0: Conclusion

This 10-week plan systematically builds the Contract Monthly Claim System from a basic prototype to a fully featured, robust, and deployable web application. Part 1 established the foundation and received valuable feedback. Part 2 successfully addressed all feedback, resulting in a high-fidelity, 99%-graded enhanced prototype. Part 3 will focus on advanced features, security, performance, and deployment, transforming the project into a production-ready system.

DISCLAIMER

DUE TO TECHNICAL ISSUES, I WAS UNABLE TO PUSH THE UNIT TESTING FILES TO THE GITHUB REPOSITORY. HOWEVER, I HAVE INCLUDED THE UNIT TESTING FILES IN THE SUBMISSION PACKAGE FOR THE PROJECT, AND THEY ARE READILY AVAILABLE FOR REVIEW. PLEASE LET ME KNOW IF YOU REQUIRE ANY ADDITIONAL INFORMATION OR ASSISTANCE REGARDING THE TESTING.