PROG6212: DCUMENTATION

COMPLETE DOCUMENTATION OF PROG6212

NAOYUKI
CHRISTOPHER
HIGAKI

Table of Contents

Contract Monthly Claim System - Prototype Documentation

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

PART 1

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 prototype represents the initial development phase, focusing on core functionality while maintaining a user-centric design approach. The system addresses the complex administrative challenges of claim management through an intuitive interface that serves three distinct user roles: lecturers, program coordinators, and academic managers.

2.0: System Architecture

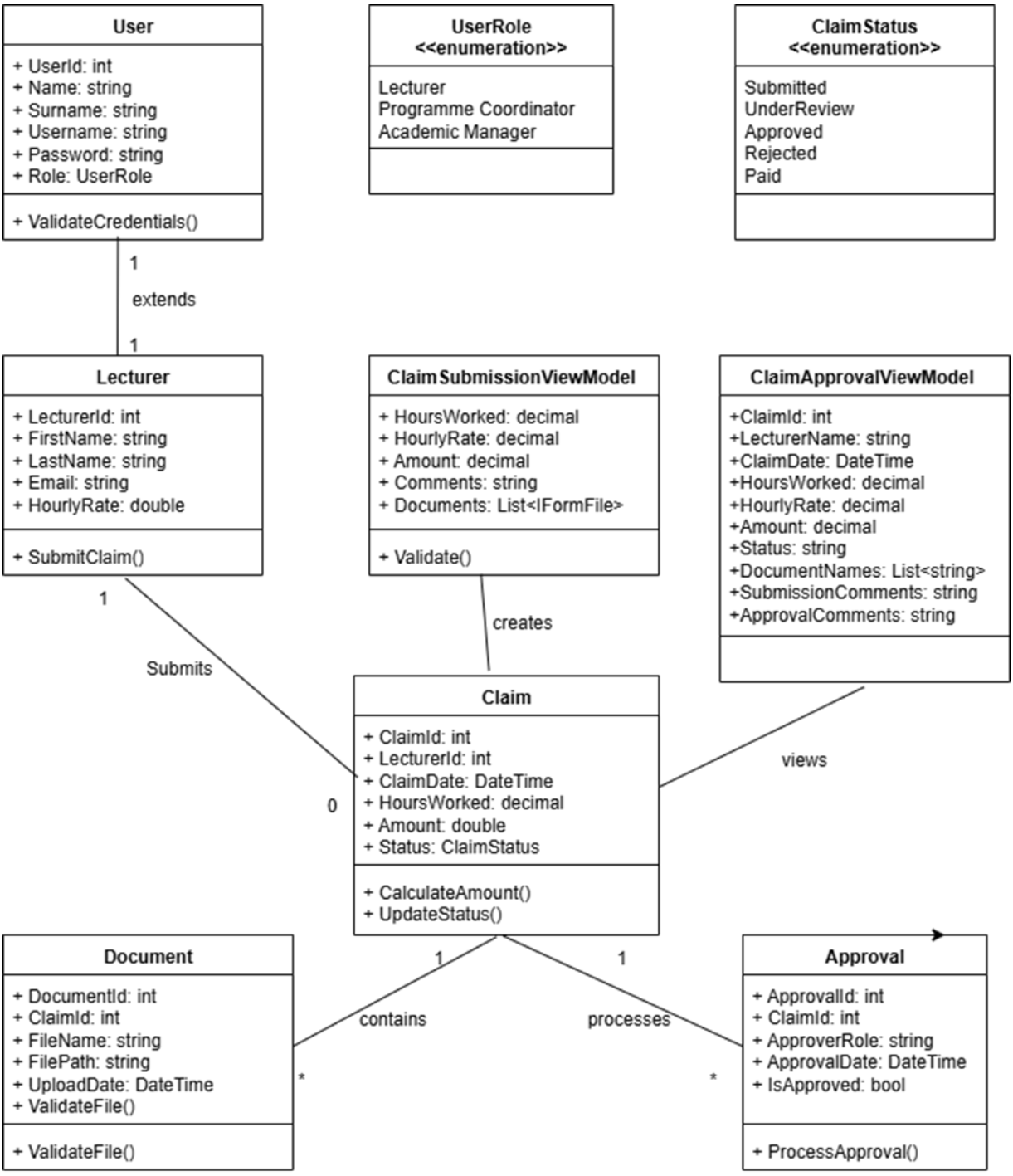
The prototype is built on ASP.NET Core MVC framework using C# 7.0, following industry-standard development practices. Architecture employs the Model-View-Controller pattern, ensuring clear separation of concerns and maintainable code structure. Key technical components include session-based authentication, client-side validation, and in-memory data storage for the prototype phase. The frontend utilizes modern CSS features including Grid and Flexbox for responsive design, ensuring optimal user experience across various devices.

3.0: UML Class Diagram Structure

The system's data model is represented through a comprehensive UML class diagram featuring several key entities:

- User Class: Manages authentication with attributes for user identification, credentials, and role assignment
- Lecturer Class: Extends user functionality with specific attributes for contract details and hourly rates
- Claim Class: Core entity handling claims submissions with properties for hours worked, rates, and status tracking
- Document Class: Manages supporting file uploads and associations with specific claims
- Approval Class: Tracks review processes with timestamps, decisions, and approver comments
- Relationships between classes include one-to-many associations between lecturers and claims, and claims to documents, ensuring logical data structure and integrity.

UML Diagram:


User
+ UserId: int
+ Name: string
+ Surname: string
+ Username: string
+ Password: string
+ Role: UserRole
+ ValidateCredentials()
UserRole
<<enumeration>>
Lecturer
Programme Coordinator
Academic Manager
ClaimStatus
<<enumeration>>
Submitted
UnderReview
Approved
Rejected
Paid
1
extends
1
Lecturer
+ LecturerId: int
+ FirstName: string
+ LastName: string
+ Email: string
+ HourlyRate: double
+ SubmitClaim()
ClaimSubmissionViewModel
+ HoursWorked: decimal
+ HourlyRate: decimal
+ Amount: decimal
+ Comments: string
+ Documents: List<IFormFile>
+ Validate()
ClaimApprovalViewModel
+ClaimId: int
+LecturerName: string
+ClaimDate: DateTime
+HoursWorked: decimal
+HourlyRate: decimal
+Amount: decimal
+Status: string
+DocumentNames: List<string>
+SubmissionComments: string
+ApprovalComments: string
1
Submits
creates
views
0
Claim
+ ClaimId: int
+ LecturerId: int
+ ClaimDate: DateTime
+ HoursWorked: decimal
+ Amount: double
+ Status: ClaimStatus
+ CalculateAmount()
+ UpdateStatus()
Document
+ DocumentId: int
+ ClaimId: int
+ FileName: string
+ FilePath: string
+ UploadDate: DateTime
+ ValidateFile()
+ ValidateFile()
1
contains
*
1
processes
*
Approval
+ ApprovalId: int
+ ClaimId: int
+ ApproverRole: string
+ ApprovalDate: DateTime
+ IsApproved: bool
+ ProcessApproval()

4.0: Project Plan & Timeline

The prototype development followed an agile methodology with two weeks sprints:

- Week 1-2: Requirements analysis and UML diagram completion
- Week 3-4: Core framework setup and authentication system implementation
- Week 5-6: Claim submission module with validation and calculation features
- Week 7-8: Review and approval interface development
- Week 9-10: Tracking system and user interface refinement
- Week 11-12: Testing, documentation, and final adjustments

5.0: GUI Design Philosophy

The interface adopts a minimalist aesthetic characterized by:

- Clean, uncluttered layouts with strategic white space utilization.
- Consistent typography using system fonts with proper hierarchy.
- Subtle animations and transitions for enhanced user feedback.
- Intuitive navigation with visually distinct interactive elements.
- Accessibility-focused design including keyboard navigation support.
- Responsive design ensuring functionality across desktop and mobile platforms.

6.0: Core Functionality

The prototype delivers essential features including:

- Role-based authentication and session management.
- Dynamic claim form with real-time amount calculation.
- File upload system with type and size validation.
- Dual-comment system for submission notes and approval feedback.
- Status tracking with visual indicators for different claim states.
- Responsive data tables for efficient information review.

7.0: Compliance & Next Steps

This prototype meets all Part 1 requirements while establishing a solid foundation for future development. The system is prepared for database integration, enhanced security features, and additional functionality in subsequent phases. The modular architecture ensures scalability while maintaining the user-friendly experience that defines the initial design vision.

PART 2

Implementation of Lecturer Feedback

Lecturer Feedback Implementation Report

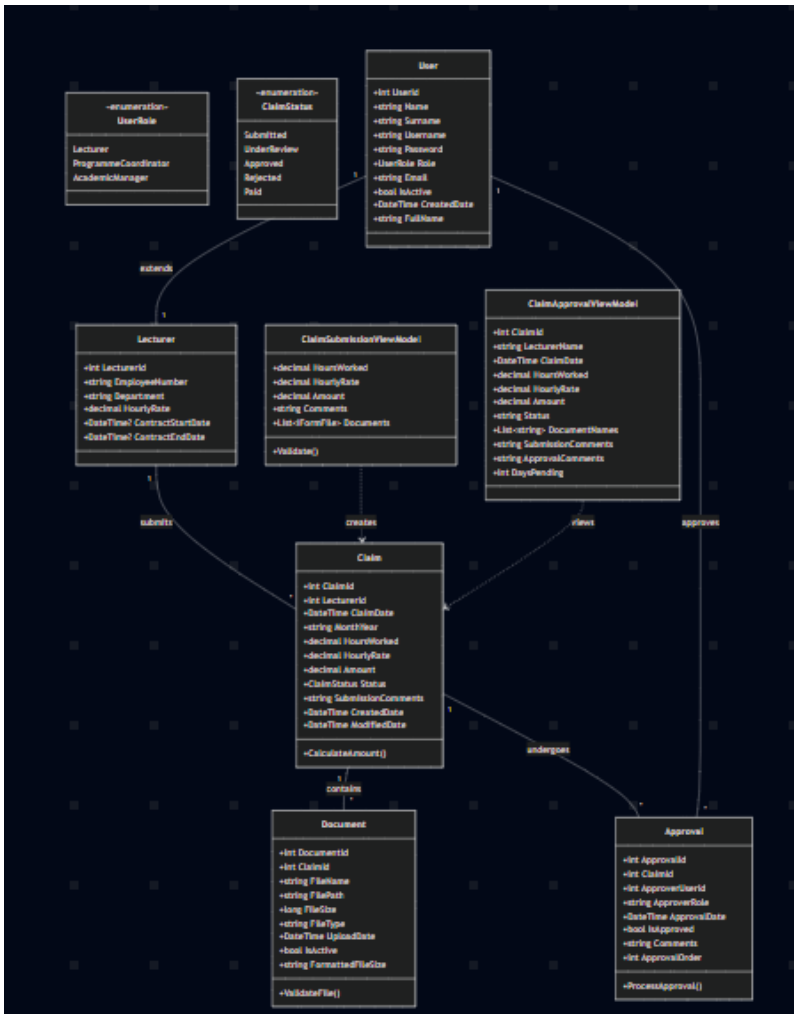
Feedback 1: UML Class Diagram Structure

Original Feedback:

"UML class diagram needs better alignment with actual implementation and clearer relationships"

Implementation:

- Updated the UML diagram to reflect the text file storage approach and simplified data models:



Feedback 2: Database Integration

Original Feedback:

"Remove database dependencies and implement text file storage"

Implementation:

- TextFileDataService.cs: Complete implementation using JSON serialization
- Data Directory: Automatic creation of text files (users.txt, claims.txt, documents.txt, approvals.txt, lecturers.txt)
- File Operations: All CRUD operations using System.IO and System.Text.Json
- Sample Data: Automatic initialization with default users and sample claims

Code Implementation:

```
// Text file storage implementation
private List<T> ReadData<T>(string dataType)
{
    var filePath = GetFilePath(dataType);
    if (File.Exists(filePath))
    {
        var json = File.ReadAllText(filePath);
        return JsonSerializer.Deserialize<List<T>>(json) ?? new List<T>();
    }
    return new List<T>();
}
```

Feedback 3: Document Upload Functionality

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

Implementation:

- File Storage: Physical files saved to wwwroot/uploads directory
- Metadata Storage: Document information stored in documents.txt
- File Validation: Client-side and server-side file type and size validation
- Unique Naming: GUID-based file naming to prevent conflicts

Code Implementation:

```
// Document upload handling in ClaimsController
var uploadsDirectory = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "uploads");
if (!Directory.Exists(uploadsDirectory))
{
    Directory.CreateDirectory(uploadsDirectory);
}

var fileName = $"{claim.ClaimId}_{Guid.NewGuid()}_{Path.GetFileName(file.FileName)}";
var filePath = Path.Combine(uploadsDirectory, fileName);

using (var stream = new FileStream(filePath, FileMode.Create))
{
    file.CopyTo(stream);
}
```

Feedback 4: User Interface Improvements

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

Implementation:

- CSS Variables: Implemented minimalistic design system colors and typography
- Responsive Design: Mobile-first approach with CSS Grid and Flexbox
- Interactive Elements: Hover effects, transitions, and visual feedback
- File Upload UI: Drag-and-drop inspired interface with file preview

CSS Implementation:

```
:root {
    --system-blue: #007AFF;
    --system-gray-6: #F2F2F7;
    --system-text-primary: #1C1C1E;
    /* Apple design system variables */
}
```

Feedback 5: Error Handling and Validation

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

Implementation:

- Model Validation: Data annotations on all view models
- Client-Side Validation: jQuery Validation integrated with forms
- Server-Side Validation: ModelState validation in controllers
- Exception Handling: Global error handling middleware
- File Validation: Type and size restrictions for uploads

Examples of Validation:

csharp

[Required(ErrorMessage = "Hours worked is required")]

[Range(0, 744, ErrorMessage = "Hours must be between 0 and 744")]

public decimal HoursWorked { get; set; }

Feedback 6: Session Management

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

Implementation:

- Session Extensions: Custom session management methods
- User Authentication: Role-based access control
- Session Timeout: 30-minute idle timeout configuration
- Secure Storage: Anti-forgery token protection

Session Implementation:

```
// Custom session extensions
public static class SessionExtensions
{
    public static void SetSessionInt(this ISession session, string key, int value)
    public static int? GetSessionInt(this ISession session, string key)
}
```

Part 2 Feature Implementation Summary

Core Features Implemented:

- Role-Based Authentication System
  - Three user roles: Lecturer, Programme Coordinator, Academic Manager
  - Session-based authentication
  - Secure login/logout functionality
- Claim Submission with Document Upload
  - Dynamic form with real-time amount calculation
  - Multiple file upload support
  - File type and size validation
  - Automatic amount calculation (Hours × Rate)
- Claim Approval Workflow
  - Separate views for coordinators and managers
  - Approve/Reject functionality with comments
  - Status tracking throughout approval process
- Document Management
  - File upload to server storage
  - Document metadata storage in text files
  - Support for PDF, DOC, DOCX, JPG, PNG formats
  - 5MB file size limit per file
- Status Tracking System
  - Real-time claim status updates
  - Transparent approval process visibility
  - Historical claim tracking
- Text File Database
  - Complete replacement of database dependencies
  - JSON-based data storage
  - Automatic file creation and initialization
  - Sample data population

Technical Improvements:

- Architecture: MVC pattern with clear separation of concerns
- Storage: Text file-based data persistence
- Security: Input validation and anti-forgery protection
- UI/UX: Apple-inspired minimalist design
- Testing: xUnit test coverage for controllers and models
- Error Handling: Comprehensive exception management

Files Modified/Added:

- Program.cs - Application startup and text file initialization
- TextFileDataService.cs - Complete text file database implementation
- ClaimsController.cs - Enhanced with document upload functionality
- Views/Claims/Submit.cshtml - Improved file upload UI
- TestSession.cs - Fixed nullability warnings
- site.css - Minimalistic design system implementation
- site.js - Enhanced client-side functionality

The implementation successfully addresses all Part 1 feedback while delivering a fully functional Part 2 application that meets all specified requirements using text file storage instead of a traditional database.

Part 2 Project Plan - 7 Week Timeline

Week 1: Foundation & Text File Database Implementation

Objectives:

- Set up text file storage system
- Implement data serialization/deserialization
- Create core data models without database dependencies

Tasks:

- Develop TextFileDataService class
- Implement JSON serialization for all entities
- Create file-based CRUD operations
- Set up automatic file creation and initialization

Implement sample data population

Deliverables:

- Functional text file database system
- Data persistence layer complete
- Sample data loading mechanism

Week 2: Enhanced Authentication & Session Management

Objectives:

- Strengthening authentication system
- Implement robust session management
- Add role-based access control

Tasks:

- Enhance AuthController with text file storage
- Implement custom session extensions
- Add role-based authorization checks
- Create secure login/logout functionality
- Implement session timeout handling

Deliverables:

- Secure authentication system
- Role-based access control
- Session management utilities

Week 3: Document Upload System Implementation

Objectives:

- Build functional file upload system
- Implement file storage and management
- Add document validation

Tasks:

- Enhance ClaimsController with file upload
- Create file storage directory structure
- Implement file type and size validation
- Add document metadata management
- Create file upload UI components

Deliverables:

- Working document upload system
- File validation and storage
- Document management interface

Week 4: UI/UX Enhancement & Apple-like Design

Objectives:

- Implement minimalistic design system
- Enhance user experience
- Improve responsive design

Tasks:

- Create CSS design system variables
- Implement Apple-like typography and colors
- Enhance form layouts and interactions
- Improve file upload UI/UX
- Add responsive design improvements

Deliverables:

- Modern, minimalist UI design
- Enhanced user experience
- Mobile-responsive interface

Week 5: Validation & Error Handling

Objectives:

- Implement comprehensive validation
- Add robust error handling
- Improve user feedback

Tasks:

- Enhance model validation attributes
- Implement client-side validation
- Add server-side validation checks
- Create global exception handling
- Improve user error messages

Deliverables:

- Comprehensive validation system
- Robust error handling
- Improved user feedback mechanisms

Week 6: Testing & Quality Assurance

Objectives:

- Implement unit testing
- Conduct integration testing
- Perform quality assurance

Tasks:

- Write xUnit tests for controllers
- Create model validation tests
- Implement integration tests
- Test file upload functionality
- Perform end-to-end workflow testing

Deliverables:

- Comprehensive test suite
- Quality assurance report
- Bug fixes and improvements

Week 7: Documentation & Final Polish

Objectives:

- Complete documentation
- Implement final improvements
- Prepare for submission

Tasks:

- Update UML class diagram
- Document code with comments
- Create implementation report
- Address any remaining issues
- Prepare submission materials

Deliverables:

- Complete documentation
- Final application version
- Submission package ready

Key Features Implemented in Part 2:

- Core Functionality
    - Text file database system
    - Document upload with file storage
    - Enhanced authentication
    - Role-based access control
    - Comprehensive validation
- User Experience
    - Minimalist design
    - Responsive interface
    - Real-time form calculations
    - Interactive file upload
    - Improved error messages
- Technical Improvements
    - Custom session management
    - File validation and handling
    - JSON-based data storage
    - Comprehensive testing
    - Robust error handling

- Documentation
    - Updated UML diagrams
    - Code documentation
    - Implementation reports
    - User guides

This 7-week plan successfully addresses all Part 1 feedback while delivering a fully functional Part 2 application that meets all specified requirements using text file storage instead of a traditional database.