PO

PROg6212

POE – PART 2

Enrique Arendse

ST10302006

Contents

[Introduction 2](#_Toc180665838)

[WPF Code: 2](#_Toc180665839)

[1. Claim Submission by Lecturers 4](#_Toc180665840)

[Code: 4](#_Toc180665841)

[2. Claim Approval and Rejection by Programme Coordinators and Academic Managers 5](#_Toc180665842)

[Code: 5](#_Toc180665843)

[3. Document Upload Feature 6](#_Toc180665844)

[Code: 6](#_Toc180665845)

[4. Real-Time Claim Status Tracking 7](#_Toc180665846)

[Code: 7](#_Toc180665847)

[5. Error Handling and Data Validation 8](#_Toc180665848)

[Code: 8](#_Toc180665849)

[6. Version Control 9](#_Toc180665850)

[Link: https://github.com/Rique14/MonthlyClaimContractSystemPart2 10](#_Toc180665851)

[7. Database 10](#_Toc180665852)

[Conclusion 11](#_Toc180665853)

[References 11](#_Toc180665854)

# Introduction

Expanding upon the Part 1 prototype, Part 2 concentrates on integrating.NET Core functionalities for the Contract Monthly Claim System (CMCS) into the GUI. This article outlines the methods and strategies utilized to put important functionality including filing claims, accepting or rejecting claims, uploading documents and monitoring claim statuses into practice. Real-time updates and error handling were also included to improve the application's usability and dependability.

# WPF Code:

<Window x:Class="MonthlyClaimContractSystemPart2.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:MonthlyClaimContractSystemPart2"

mc:Ignorable="d"

Title="Contract Monthly Claim System" Height="450" Width="800" Background="#F0F0F0">

<Grid >

<TabControl Background="#F0F0F0" BorderThickness="0">

<!-- Tab for Lecturer Claim Submission -->

<TabItem Header="Submit Claim">

<Grid Margin="10" >

<!-- Lecturer Claim Form -->

<TextBlock Text="Lecturer Name:" VerticalAlignment="Top" />

<TextBox x:Name="LecturerName" VerticalAlignment="Top" Margin="0,20,0,0"/>

<TextBlock Text="Hours Worked:" VerticalAlignment="Top" Margin="0,50,0,0" />

<TextBox x:Name="HoursWorked" VerticalAlignment="Top" Margin="0,70,0,0"/>

<TextBlock Text="Hourly Rate:" VerticalAlignment="Top" Margin="0,100,0,0" />

<TextBox x:Name="HourlyRate" VerticalAlignment="Top" Margin="0,120,0,0"/>

<TextBlock Text="Additional Notes:" VerticalAlignment="Top" Margin="0,150,0,0" />

<TextBox x:Name="AdditionalNotes" VerticalAlignment="Top" Margin="0,170,0,0"/>

<!-- Document Upload -->

<TextBlock Text="Upload Supporting Document:" VerticalAlignment="Top" Margin="0,200,0,0" />

<Button Content="Upload" VerticalAlignment="Top" Margin="0,220,0,0" Click="UploadDocument" Background="#007ACC" Foreground="White"/>

<TextBlock x:Name="UploadedFileName" VerticalAlignment="Top" Margin="0,250,0,0"/>

<!-- Submit Claim -->

<Button Content="Submit Claim" VerticalAlignment="Top" Margin="0,280,0,0" Background="#007ACC" Foreground="White" Click="SubmitClaim"/>

<TextBlock Text="Claim Status:" VerticalAlignment="Top" Margin="0,320,0,0" />

<TextBox x:Name="ClaimStatus" IsReadOnly="True" VerticalAlignment="Top" Margin="0,340,0,0" />

</Grid>

</TabItem>

<!-- Tab for Programme Coordinator/Manager Approval -->

<TabItem Header="Approve Claims">

<Grid Margin="10">

<!-- Pending Claims List -->

<TextBlock Text="Pending Claims:" VerticalAlignment="Top" />

<ListBox x:Name="PendingClaims" VerticalAlignment="Top" Margin="0,20,0,0" SelectionChanged="OnClaimSelected"/>

<!-- Claim Details for Approval -->

<StackPanel Orientation="Vertical" VerticalAlignment="Top" Margin="0,150,0,0">

<TextBlock Text="Lecturer Name:" />

<TextBox x:Name="LecturerNameDetails" IsReadOnly="True" />

<TextBlock Text="Hours Worked:" />

<TextBox x:Name="HoursWorkedDetails" IsReadOnly="True" />

<TextBlock Text="Hourly Rate:" />

<TextBox x:Name="HourlyRateDetails" IsReadOnly="True" />

<TextBlock Text="Total Amount:" />

<TextBox x:Name="TotalAmountDetails" IsReadOnly="True" />

<TextBlock Text="Additional Notes:" />

<TextBox x:Name="AdditionalNotesDetails" IsReadOnly="True" />

<TextBlock Text="Document:" />

<TextBox x:Name="DocumentPathDetails" IsReadOnly="True" />

</StackPanel>

<!-- Approve and Reject Buttons -->

<Button Content="Approve" VerticalAlignment="Bottom" Margin="0,300,700,0" Click="ApproveClaim" Background="#4CAF50" Foreground="White"/>

<Button Content="Reject" VerticalAlignment="Bottom" Margin="700,30,0,0" Click="RejectClaim" Background="#F44336" Foreground="White"/>

</Grid>

</TabItem>

</TabControl>

</Grid>

</Window>

# Claim Submission by Lecturers

With the help of an easy-to-use form, professors can now submit their monthly claims through the system. The form records crucial information about the claim, such as:

* Hours worked
* Hourly rate
* Additional notes

**Implementation Approach:**

• A StackPanel is used in the form's design to arrange the input fields in a tidy, vertical fashion.   
• TextBox controls and labels are used to gather data from the instructors.  
• At the bottom of the form, a "Submit" button is introduced to enable instructors to submit the claim data for processing.

## Code:

private void SubmitClaim(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(LecturerName.Text) || string.IsNullOrEmpty(HoursWorked.Text) ||

string.IsNullOrEmpty(HourlyRate.Text) || string.IsNullOrEmpty(UploadedFileName.Text))

{

MessageBox.Show("Please fill in all required fields and upload a supporting document.");

return;

}

try

{

double hoursWorked = Convert.ToDouble(HoursWorked.Text);

double hourlyRate = Convert.ToDouble(HourlyRate.Text);

double totalAmount = hoursWorked \* hourlyRate;

var claim = new Claim

{

LecturerName = LecturerName.Text,

HoursWorked = hoursWorked,

HourlyRate = hourlyRate,

TotalAmount = totalAmount,

Notes = AdditionalNotes.Text,

Status = "Pending",

DocumentPath = UploadedFileName.Text

};

using (var context = new AppDbContext())

{

context.Claims.Add(claim);

context.SaveChanges(); // Save the claim to the database

}

// After the claim is successfully saved, reset the form fields

MessageBox.Show("Claim Submitted Successfully!");

ClaimStatus.Text = "Pending";

LecturerName.Text = "";

HoursWorked.Text = "";

HourlyRate.Text = "";

AdditionalNotes.Text = "";

UploadedFileName.Text = "";

// \*\*Number 2: Update the pending claims list\*\*

pendingClaims.Add(claim); // Add the new claim to the pending claims list

LoadPendingClaims(); // Refresh the list to show the new claim

}

catch (DbUpdateException dbEx)

{

MessageBox.Show($"A database error occurred: {dbEx.InnerException?.Message}");

}

catch (Exception ex)

{

MessageBox.Show($"An error occurred: {ex.Message}");

}

}

# Claim Approval and Rejection by Programme Coordinators and Academic Managers

Academic managers and program coordinators now have their own view in the application, where they can:

* View pending claims submitted by lecturers.
* Choose to approve or reject each claim.

**Implementation Approach:**

• Each pending claim's details are shown in a grid format and buttons to approve and reject are included.

• The user interface updates the status of a claim when it is accepted or denied and a Label is used to show the current state in real time.

## Code:

private void ApproveClaim(object sender, RoutedEventArgs e)

{

var selectedClaim = (Claim)PendingClaims.SelectedItem;

if (selectedClaim != null)

{

using (var context = new AppDbContext())

{

var claim = context.Claims.Find(selectedClaim.ClaimID);

if (claim != null)

{

claim.Status = "Approved";

context.SaveChanges();

}

}

MessageBox.Show("Claim Approved!");

LoadPendingClaims();

}

}

private void RejectClaim(object sender, RoutedEventArgs e)

{

var selectedClaim = (Claim)PendingClaims.SelectedItem;

if (selectedClaim != null)

{

using (var context = new AppDbContext())

{

var claim = context.Claims.Find(selectedClaim.ClaimID);

if (claim != null)

{

claim.Status = "Rejected";

context.SaveChanges();

}

}

MessageBox.Show("Claim Rejected!");

LoadPendingClaims();

}

}

# Document Upload Feature

Instructors can now include supporting documents (such as Word and PDF files) with their claim submission. This guarantees that all required documentation is connected and that claims are properly documented.

**Implementation Approach:**

• The claim submission form now has an Upload Document button. Upon clicking, a file dialog box appears, enabling the user to choose a file from their local computer.

• The file name appears in the form next to the upload button and the uploaded file is safely kept.

• In order to guarantee that the system only accepts standard document formats (such as.pdf, .docx and.xlsx), file size and format limits were put in place.

## Code:

private void UploadDocument(object sender, RoutedEventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog

{

Filter = "PDF Files|\*.pdf|Word Documents|\*.docx|Excel Files|\*.xlsx",

Title = "Upload Supporting Document"

};

if (openFileDialog.ShowDialog() == true)

{

FileInfo fileInfo = new FileInfo(openFileDialog.FileName);

// Limit file size to 5MB

if (fileInfo.Length > 5 \* 1024 \* 1024)

{

MessageBox.Show("File size exceeds the 5MB limit.");

return;

}

UploadedFileName.Text = fileInfo.FullName;

MessageBox.Show("Document Uploaded Successfully!");

}

}

# Real-Time Claim Status Tracking

Real-time status tracking has been added to the system to increase transparency. After a claim is filed, instructors can view the claim's current status, including:

* Pending
* Approved
* Rejected

**Implementation Approach:**

• To show a claim's current status, utilize the Label control. Every time a claim is accepted or denied by an academic manager or program coordinator, the status is automatically updated. The status quickly refreshes to show where the approval process is at that moment.

## Code:

private void LoadPendingClaims()

{

using (var context = new AppDbContext())

{

// Fetch all pending claims from the database

var pendingClaims = context.Claims.Where(c => c.Status == "Pending").ToList();

// Refresh the PendingClaims ListBox with updated list

PendingClaims.ItemsSource = null; // Clear the old items

PendingClaims.ItemsSource = pendingClaims; // Load the updated list of claims

}

}

private void OnClaimSelected(object sender, RoutedEventArgs e)

{

var selectedClaim = (Claim)PendingClaims.SelectedItem;

if (selectedClaim != null)

{

LecturerNameDetails.Text = selectedClaim.LecturerName;

HoursWorkedDetails.Text = selectedClaim.HoursWorked.ToString();

HourlyRateDetails.Text = selectedClaim.HourlyRate.ToString();

TotalAmountDetails.Text = selectedClaim.TotalAmount.ToString("C");

AdditionalNotesDetails.Text = selectedClaim.Notes;

DocumentPathDetails.Text = selectedClaim.DocumentPath;

// Bind the status of the selected claim to the ClaimStatus TextBox

ClaimStatus.DataContext = selectedClaim;

ClaimStatus.SetBinding(TextBox.TextProperty, new System.Windows.Data.Binding("Status"));

}

}

}

public class Claim : INotifyPropertyChanged

{

private string \_status;

public int ClaimID { get; set; } // Add ClaimID for primary key

public string LecturerName { get; set; }

public double HoursWorked { get; set; }

public double HourlyRate { get; set; }

public double TotalAmount { get; set; }

public string Notes { get; set; }

public string DocumentPath { get; set; }

// Property for Claim Status with INotifyPropertyChanged implementation

public string Status

{

get => \_status;

set

{

\_status = value;

OnPropertyChanged(nameof(Status));

}

}

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

}

# Error Handling and Data Validation

Error handling techniques were incorporated into the program to guarantee that unforeseen circumstances are handled with grace. Among them are:

* Validating input fields (e.g., ensuring hours worked and hourly rate are numeric).
* Restricting file uploads to specific formats and sizes.
* Displaying user-friendly error messages when something goes wrong.

**Implementation Approach:**

* Try-catch blocks were used around critical code sections to capture any exceptions that may occur.
* Validation logic checks whether the required fields are populated and that numeric fields contain valid numbers.

## Code:

private void SubmitClaim(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(LecturerName.Text) || string.IsNullOrEmpty(HoursWorked.Text) ||

string.IsNullOrEmpty(HourlyRate.Text) || string.IsNullOrEmpty(UploadedFileName.Text))

{

MessageBox.Show("Please fill in all required fields and upload a supporting document.");

return;

}

try

{

double hoursWorked = Convert.ToDouble(HoursWorked.Text);

double hourlyRate = Convert.ToDouble(HourlyRate.Text);

double totalAmount = hoursWorked \* hourlyRate;

var claim = new Claim

{

LecturerName = LecturerName.Text,

HoursWorked = hoursWorked,

HourlyRate = hourlyRate,

TotalAmount = totalAmount,

Notes = AdditionalNotes.Text,

Status = "Pending",

DocumentPath = UploadedFileName.Text

};

using (var context = new AppDbContext())

{

context.Claims.Add(claim);

context.SaveChanges(); // Save the claim to the database

}

// After the claim is successfully saved, reset the form fields

MessageBox.Show("Claim Submitted Successfully!");

ClaimStatus.Text = "Pending";

LecturerName.Text = "";

HoursWorked.Text = "";

HourlyRate.Text = "";

AdditionalNotes.Text = "";

UploadedFileName.Text = "";

// \*\*Number 2: Update the pending claims list\*\*

pendingClaims.Add(claim); // Add the new claim to the pending claims list

LoadPendingClaims(); // Refresh the list to show the new claim

}

catch (DbUpdateException dbEx)

{

MessageBox.Show($"A database error occurred: {dbEx.InnerException?.Message}");

}

catch (Exception ex)

{

MessageBox.Show($"An error occurred: {ex.Message}");

}

}

# Version Control

At different phases of the development process, all changes were committed and posted to the GitHub repository in order to maintain appropriate version control. To keep track of the developments and record the modifications made, concise and informative commit messages were employed.

## Link: <https://github.com/Rique14/MonthlyContractClaim_Part2>

# Database

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MonthlyClaimContractSystemPart2

{

public class AppDbContext : DbContext

{

public DbSet<Claim> Claims { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

// Use MySQL connection with Pomelo

optionsBuilder.UseMySql("Server=localhost;Database=ContractClaimsDB;User=root;Password=;",

new MySqlServerVersion(new Version(8, 0, 21))); // Specify MySQL version

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Claim>().ToTable("Claims");

}

}

}

# Conclusion

Part 2 of the Contract Monthly Claim System (CMCS) now has additional features that give lecturers a comprehensive procedure for submitting claims, uploading supporting documentation and monitoring real-time progress updates. Academic managers and program coordinators can now easily check, accept or reject claims. The application provides a dependable and user-friendly experience through version control, error handling and user feedback.

# References

Enrique, 2024. *Part 2 - Monthly Contract Claim System,* s.l.: VIsual Studio 2022.