# PROGRAMMING 2B/PROG6212 POE PART 1

IMAAN EBRAHIM

ST10021922

GROUP 3



## PART 1:

1.Documentation:

**Design Choices**

**Modular Design**

1. **Explanation:**
   * **Concept Overview:**

The design of the Contract Monthly Claim System (CMCS) follows a modular approach. This approach includes splitting the system into separate, standalone modules, with each module handling a particular part of the system's functions. The primary modules consist of the Lecturer Module, the Admin Module, and the Document Management Module.  
  
Module for Lecturers: Handles tasks concerning lecturers, like submitting claims, accessing claim records, and uploading necessary documents.  
  
Admin Module performs administrative duties like reviewing and approving claims, managing users, and creating reports.  
  
Module for Managing Documents: in charge of managing the uploading, storing, and retrieving of files related to claims.

* + **Component Interaction:**

Every module interacts with other modules using clearly defined interfaces to ensure smooth communication between them. For example, if a lecturer sends a request, the Lecturer Module works with the Document Management Module to add attachments and with the Admin Module to pass the request for evaluation.

* + **Technical Implementation:**

The system consists of individual components known as modules, each of which can be created, evaluated, and implemented on its own. Separating components is done through common programming techniques and platforms like ASP.NET Core MVC or Windows Forms that endorse modularity.

1. **Justification:**
   * **Maintainability:**

Modular design improves maintainability by separating various system functions. Updating or repairing one module will not impact other components of the system. For example, if modifications are required in the Admin Module, they can be made without affecting the Lecturer Module, minimizing the chance of causing errors in other areas of the system.

* + **Scalability:**

As the system grows and expands, more features or modules can be incorporated without needing a total overhaul. For instance, if there is a requirement to include a new feature for managing feedback, a new module can be created and added to the current system structure.

* + **Development Efficiency:**

The use of modular design enables several developers to work on separate modules at the same time, thus expediting the development process. Clear module boundaries and interfaces enable the smooth progress of this simultaneous development, outlining how modules engage with each other.

* + **Testing and Debugging:**

Testing and debugging are made easier as modules can undergo individual testing prior to integration. This isolation makes it possible to locate and fix errors within the module they are found in, instead of searching through a massive codebase.

* + **Flexibility:**

The modular method provides flexibility in terms of technology stack and execution. Various modules have the option to utilize distinct technologies or frameworks that are most appropriate for their particular needs. In one scenario, the Document Management Module may utilize a specialized file storage platform or database, whereas the Lecturer Module prioritizes user interface and engagement.

* + **User Experience:**

A modular layout guarantees that users will have a well-structured and user-friendly interface. Every module offers specific features designed for various user roles' specific requirements. This approach based on roles helps create a smoother and more user-friendly experience.

* + **Security:**

Security can be controlled individually for each module, enabling customized security measures depending on the importance of the feature. The Admin Module could have extra security measures, like access controls and auditing, to safeguard administrative tasks.

In conclusion, the modular design explanation is concise and offers a thorough explanation of the system's structure.  
  
The technical aspects of how modules are implemented and interact are detailed to ensure a full understanding of the design. The reasons for choosing a modular design are explained thoroughly, covering benefits like maintainability, scalability, development efficiency, testing, debugging, flexibility, user experience, and security.  
  
The reason why a modular approach is appropriate for CMCS is explained, taking into account different aspects of system development and user interaction. The documentation presupposes that the system will consist of individual modules that can be managed and expanded independently. It also presupposes a knowledge of common programming techniques and frameworks.  
  
The project requirements and limitations are understood and taken into account by addressing constraints like the necessity for module-based security and user experience considerations.

2.UML Diagram for Databases

1. Identify Core Entities

Based on the requirements of CMCS, these are following core entities:

1. **Lecturer**
2. **Claim**
3. **Document**
4. **Approval**
5. **User**

2. Define Attributes for Each Entity

* **Lecturer:**
  + LecturerID (Primary Key)
  + Name
  + Department
  + ContactInformation
  + Email
* **Claim:**
  + ClaimID (Primary Key)
  + LecturerID (Foreign Key)
  + SubmissionDate
  + HoursWorked
  + HourlyRate
  + TotalAmount
  + Status (e.g., Submitted, Approved, Rejected)
* **Document:**
  + DocumentID (Primary Key)
  + ClaimID (Foreign Key)
  + FileName
  + FilePath
  + UploadDate
* **Approval:**
  + ApprovalID (Primary Key)
  + ClaimID (Foreign Key)
  + ApproverID (Foreign Key to User)
  + ApprovalDate
  + Status (e.g., Pending, Approved, Rejected)
* **User:**
  + UserID (Primary Key)
  + Role (e.g., Lecturer, Programme Coordinator, Academic Manager)
  + Username
  + Password
  + Email

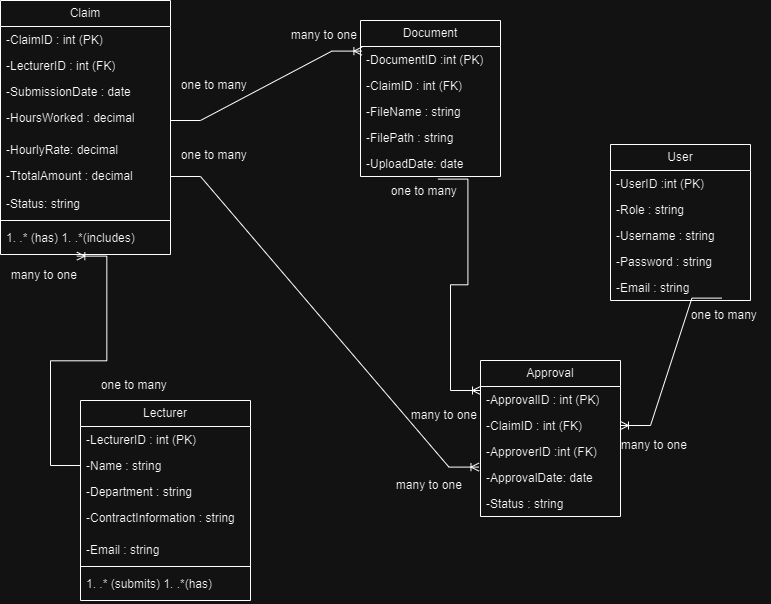
3. Define Relationships Between Entities

* **Lecturer to Claim:**
  + **Relationship:** One-to-Many
  + **Description:** A lecturer can submit multiple claims, but each claim is submitted by only one lecturer.
* **Claim to Document:**
  + **Relationship:** One-to-Many
  + **Description:** Each claim can have multiple associated documents, but each document is linked to only one claim.
* **Claim to Approval:**
  + **Relationship:** One-to-Many
  + **Description:** Each claim can have multiple approval records (e.g., if reviewed by multiple approvers), but each approval record is for one specific claim.
* **User to Approval:**
  + **Relationship:** One-to-Many
  + **Description:** A user (Programme Coordinator or Academic Manager) can approve multiple claims, but each approval is recorded for one user.

4. Create the UML Class Diagram

Here is how you might represent these entities and their relationships in a UML class diagram:

1. **Lecturer Class:**
   * Attributes: LecturerID, Name, Department, ContactInformation, Email
   * Relationships:
     + One-to-Many with **Claim**
2. **Claim Class:**
   * Attributes: ClaimID, LecturerID, SubmissionDate, HoursWorked, HourlyRate, TotalAmount, Status
   * Relationships:
     + Many-to-One with **Lecturer**
     + One-to-Many with **Document**
     + One-to-Many with **Approval**
3. **Document Class:**
   * Attributes: DocumentID, ClaimID, FileName, FilePath, UploadDate
   * Relationships:
     + Many-to-One with **Claim**
4. **Approval Class:**
   * Attributes: ApprovalID, ClaimID, ApproverID, ApprovalDate, Status
   * Relationships:
     + Many-to-One with **Claim**
     + Many-to-One with **User**
5. **User Class:**
   * Attributes: UserID, Role, Username, Password, Email
   * Relationships:
     + One-to-Many with **Approval**



UML made using draw.io <https://app.diagrams.net/>

3.Project Plan

1. Define Project Scope

**Project Scope:**

Create a non-operational prototype for the CMCS system.  
  
The prototype will consist of a UML class diagram, project plan, and a visual representation of the GUI utilizing either Windows Presentation Forms (WPF) or Model-View-Controller (MVC) in .NET Core.

**Objectives:**

Create a UML class diagram that illustrates the data needs of the system.  
  
Develop a comprehensive project schedule outlining tasks, dependencies, and timeframe.  
  
Create a non-operational GUI prototype for displaying the user interface.

2. Break Down Tasks

**Task Breakdown:**

**Project Initialization:**

* + Define project scope and objectives
  + Set up project repository on GitHub

**UML Class Diagram Development:**

* + Identify core entities and attributes
  + Define relationships between entities
  + Create the UML class diagram
  + Review and finalize the diagram

**Project Plan Creation:**

* + Outline project tasks and milestones
  + Identify dependencies between tasks
  + Develop a timeline with deadlines
  + Review and update the project plan

**GUI Prototype Design:**

* + Design the layout and user interface for each module
  + Develop mockups or wireframes
  + Implement the front-end prototype using WPF or MVC
  + Review and refine the GUI design

**Documentation:**

* + Write detailed explanations of design choices, database structure, and GUI layout
  + Document assumptions and constraints
  + Prepare the final project report

**Version Control:**

* + Regularly commit and push changes to the GitHub repository
  + Maintain clear and descriptive commit messages

**Final Review and Submission:**

* + Review all project components
  + Ensure all documentation and prototypes are complete
  + Submit the final project deliverables

3. Identify Dependencies

**Dependencies:**

* **UML Class Diagram Development** depends on:
  + Project Initialization (for understanding scope and setting up the repository)
* **Project Plan Creation** depends on:
  + Project Initialization (for setting up the framework)
* **GUI Prototype Design** depends on:
  + UML Class Diagram Development (for understanding data structure)
* **Documentation** depends on:
  + Completion of UML Class Diagram
  + Finalized GUI Prototype
* **Version Control** should be performed throughout all tasks.
* **Final Review and Submission** depends on:
  + Completion of Documentation
  + Finalized GUI Prototype

4. Develop Timeline

**Timeline:**

**Week 1: Beginning the Project**

* Outline the project's scope and goals.
* Establish GitHub repository.  
    
  **Week 2 will focus on designing UML class diagrams.**
* Determine main elements and characteristics within a 2-day timeframe.
* Establish connections between different entities (2 days)
* Develop and assess UML class diagram within a (3 day) timeframe.  
    
  **Week 3 involves creating the project plan.**
* Create a schedule of activities and goals to be achieved within a (2-day) time frame.
* Determine relationships and create schedule (2 days)
* Evaluate and complete project plan (1 day)  
    
  **Week 4 will focus on designing GUI prototypes**.
* Creating the visual arrangement and UI for a website or application will take 3 days.
* Create mockups or wireframes (2 days)
* Develop a prototype for the front-end within a time frame of 3 days.  
    
  **Week 5: Documentation**
* Provide in-depth explanations (over the course of 3 days)
* List assumptions and limitations (2 days)
* Create the concluding project summary within 2 days.  
    
  **Week 6 will cover Version Control and provide a final review.**
* Regularly make commits and push changes to GitHub (continuous)
* Assess every aspect of the project (2 days)
* Make sure all documentation and prototypes are finished within 2 days.
* Deliver final project materials by the deadline (1 day)

Gantt Chart:

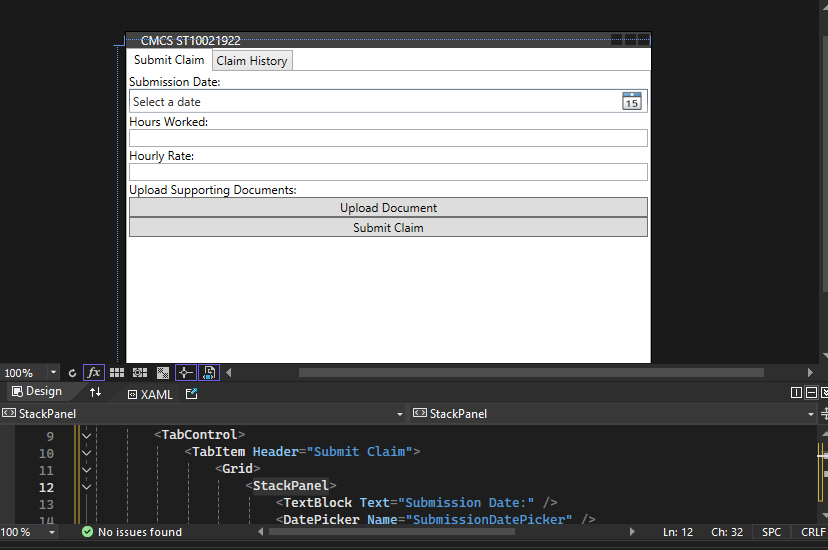
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TASK | WEEK 1 | WEEK 2 | WEEK 3 | WEEK 4 | WEEK 5 | WEEK 6 |
| Project Initialization | \* |  |  |  |  |  |
| UML Class Diagram Development |  | \* |  |  |  |  |
| Project Plan Creation |  |  | \* |  |  |  |
| GUI Prototype Design |  |  |  | \* |  |  |
| Documentation |  |  |  |  | \* |  |
| Version Control |  |  |  |  |  | \* |
| Final Review and Submission |  |  |  |  |  | \* |

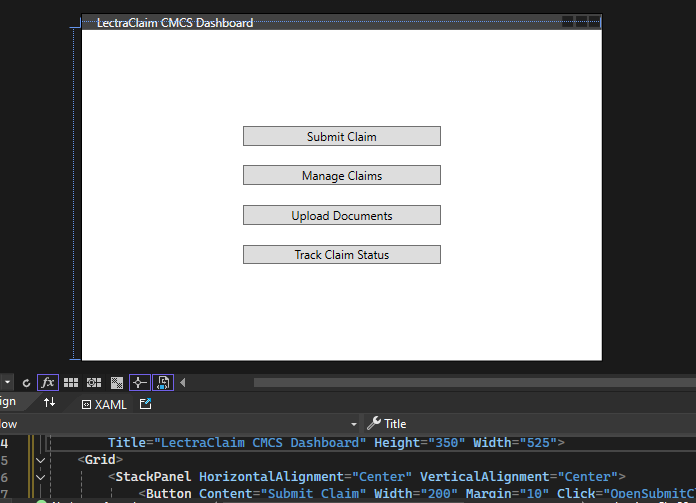
4. GUI UI

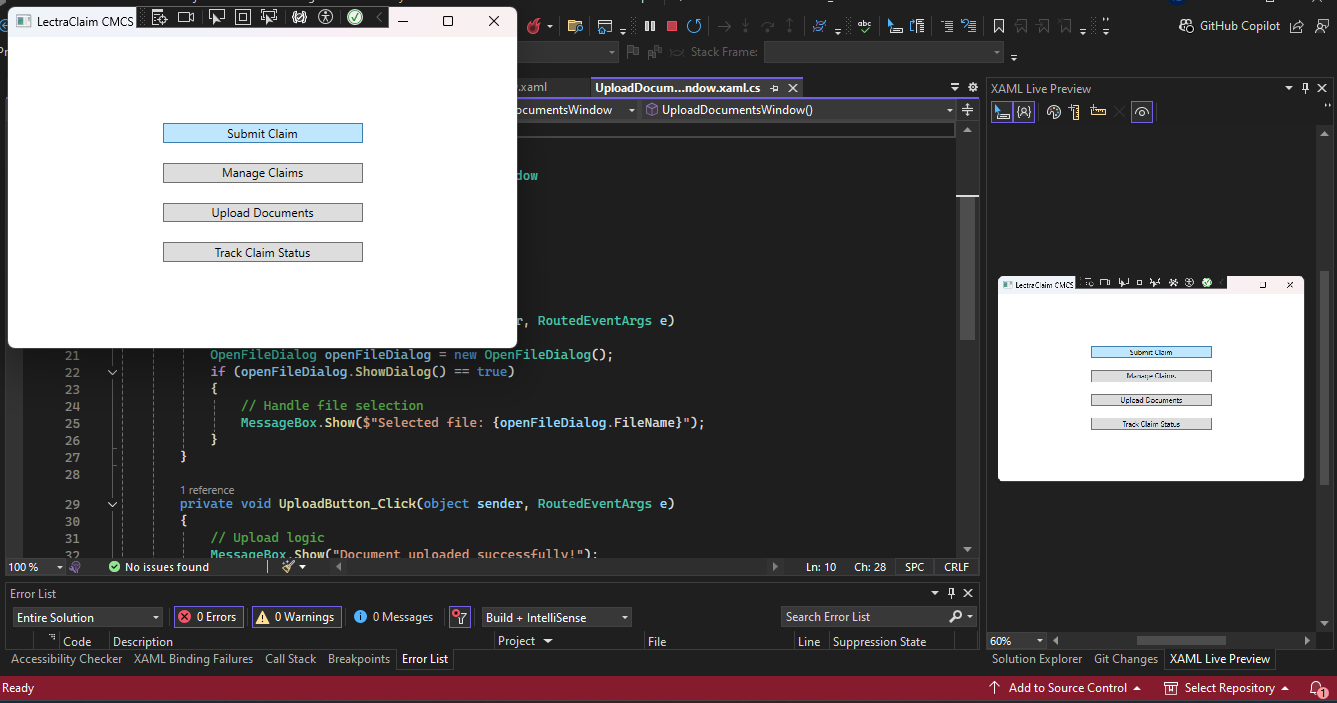
GutHub link to code:

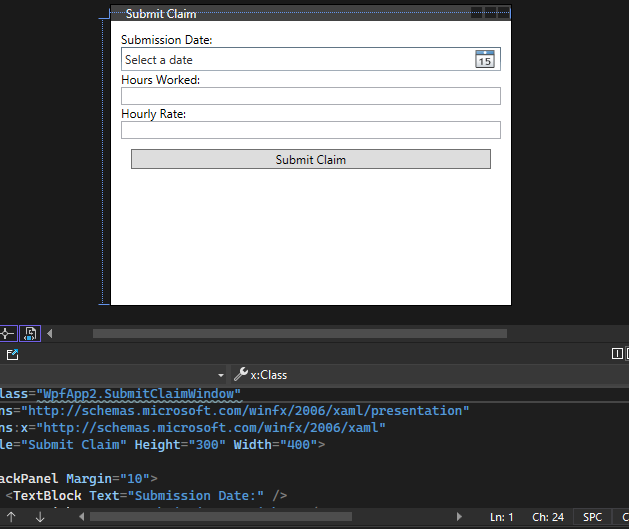
<https://github.com/ImaanEbrahim/PROG6212_poe_part_1_st10021922.git>

For the UI below I have used WPF to plan out and format a user friendly ui for users of the “LectraClaim” CMCS App

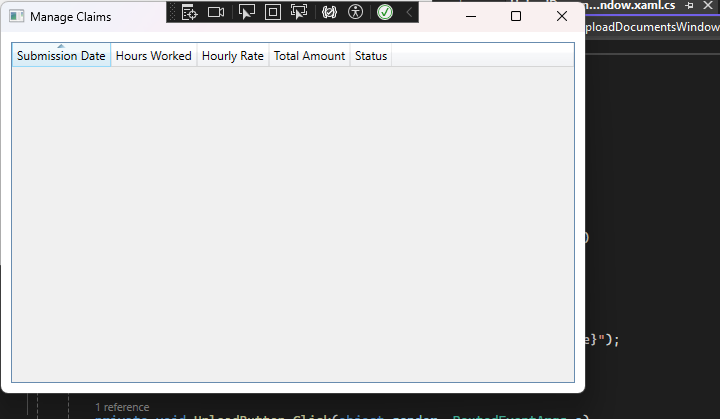
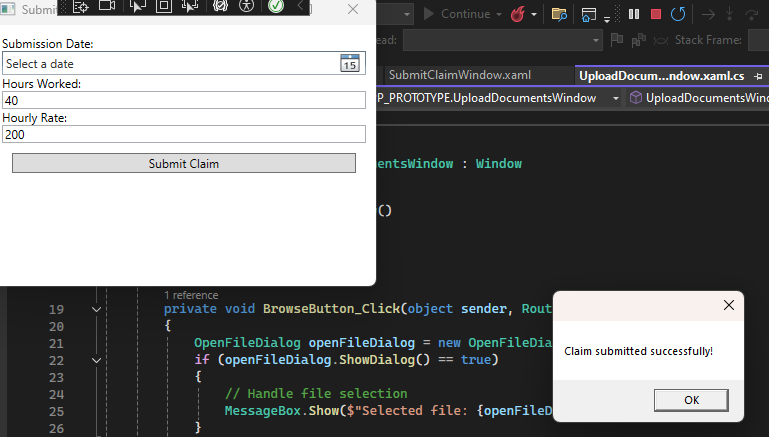
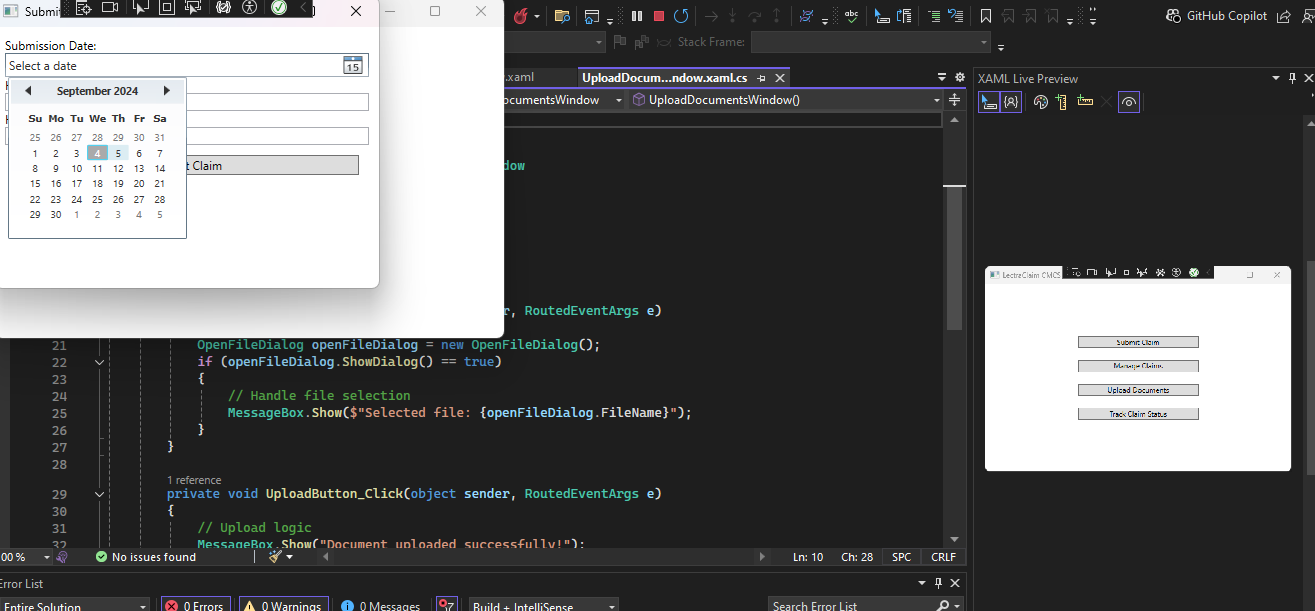
Here is one of the options for the main page, which acts as a menu and has all the features on it

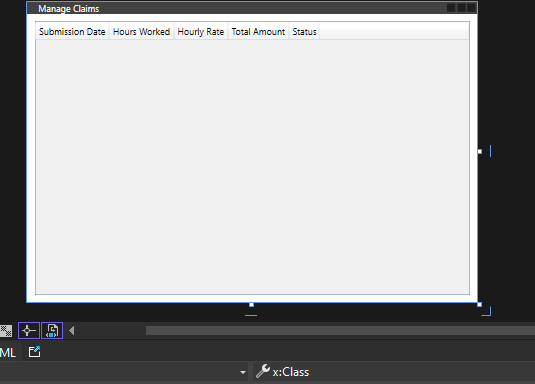
Here is the alternative option which gives users the option to choose what feature of the web app they want to use, using buttons



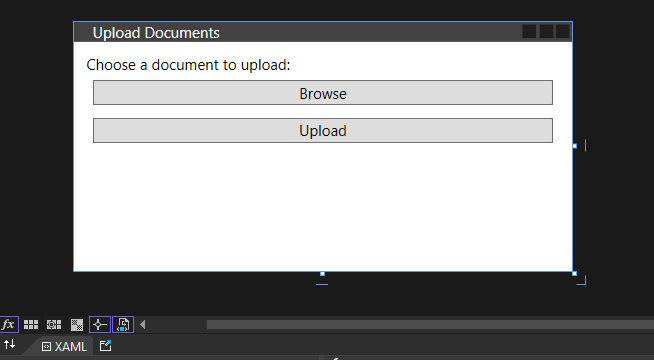


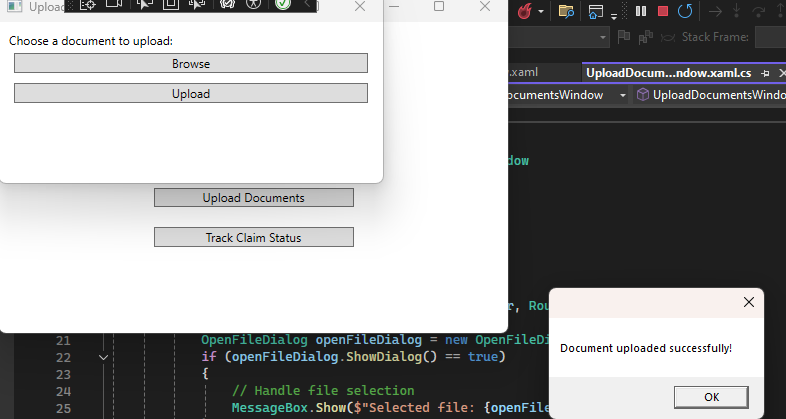
Here is the submit claim window when user clicks on the associated button where they can enter in their hours and dates

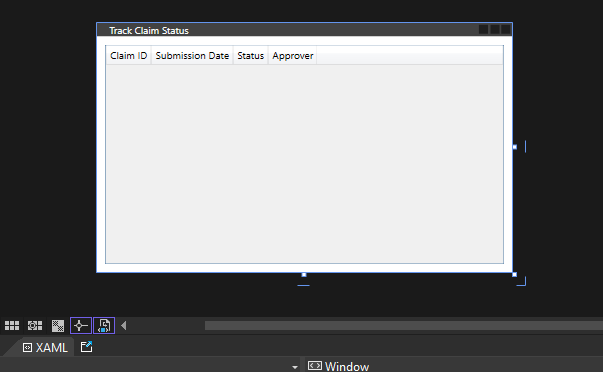




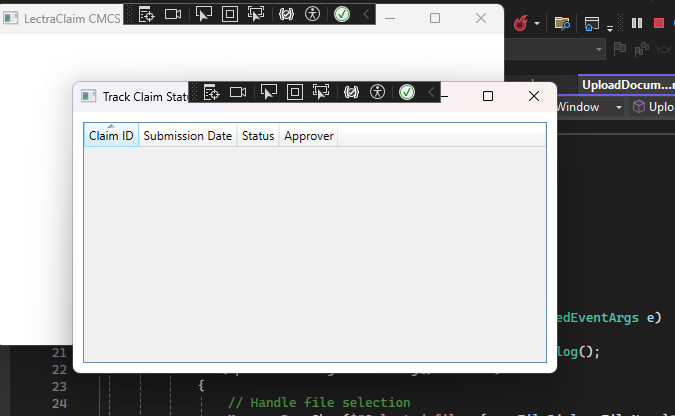
Here is where the claims are displayed and managed

Here is the window where the users can upload documents





The track status window where can users can view their claims statuses



Bibliography:

Webstacks.com (2024) *The Guide to Modular Web Design*. Available at: <https://www.webstacks.com/blog/modular-web-design> (Accessed: 4 September 2024).

Sandhu, R., Coyne, E. J., Feinstein, H. L. and Youman, C. E. (1996) 'Role-based access control models', *IEEE Computer*, 29(2), pp. 38-47.

O’Neill, M. and O’Neill, M. (2004) *The Role-Based Access Control System: An Overview*.

Sommerville, I. (2016) *Software Engineering*. 10th edn. Pearson.

Pressman, R. S. (2014) *Software Engineering: A Practitioner’s Approach*. 8th edn. McGraw-Hill Education.

Marcotte, E. (2011) *Responsive Web Design*. A Book Apart.

Wroblewski, L. (2011) *Mobile First*. A Book Apart.

Myers, G. J. (2011) *The Art of Software Testing*. 3rd edn. Wiley.

Boehm, B. W. and Papalambrou, A. (1990) 'Software Validation', *IEEE Transactions on Software Engineering*, 16(2), pp. 258-269.

Troelsen, A. and Japikse, P. (2017) *Pro C# 7: With .NET and .NET Core*. 8th edn. Apress.

Codd, E. F. (1970) 'A relational model of data for large shared data banks', *Communications of the ACM*, 13(6), pp. 377-387.

Elmasri, R. and Navathe, S. B. (2015) *Fundamentals of Database Systems*. 7th edn. Pearson.

Nielsen, J. (1993) *Usability Engineering*. Academic Press.

Garrett, J. J. (2010) *The Elements of User Experience: User-Centered Design for the Web and Beyond*. 2nd edn. New Rider.