**Ntando Nxumalo**

**ST10456704**

**PROG6212**

**PROG POE PART 2**

**Lecturer Feedback Report**

I**ntroduction**

In this report, I explain how I implemented the recommendations given by my lecturer in the Contract Monthly Claim System (CMCS). My focus was on applying good programming practices and key technical concepts such as database design, entity relationships, data handling, and overall system functionality. I also show how I developed the system’s main features, including claims management, chatbots, dashboards, and real-time updates.

**Code First Approach**

I used the Code First approach to design the database. This meant that the database structure was created based on the models I defined in the application. I carefully structured entities like claims, claim documents, and users to reflect real-world scenarios. I also made sure that relationships between claims, documents, and users were correctly set up so that the data stays consistent and accurate. Validation rules were included to make sure all required fields are filled out properly.

**Entity Framework Integration**

Entity Framework Core was used to connect the application to the database. This made it easy to handle tasks like creating, updating, and retrieving claims and documents. I also used efficient querying patterns to make sure the dashboards and claim lists load quickly and without delays.

**Performing Migrations**

I performed database migrations to keep the database in sync with the models as they changed. The initial migration created the basic structure, and later migrations added new fields and features based on feedback from my lecturer. This approach allowed the system to grow and adapt while staying organized.

**Connection Setup**

I set up a secure connection to SQL Server so that the application could communicate with the database safely. The connection was configured to be environment-specific, which helps keep sensitive information secure.

**Model Creation**

The main models in the system are claims, claim documents, and users. Each model includes the necessary details to support the system’s features. For example, claims include information like hours worked, hourly rates, totals, category, and notes, while documents store information about files attached to a claim. I applied validation both on the user interface and server-side to make sure all data entered is correct.

**SQL Server Usage**

SQL Server is used as the main database for the system. It provides a reliable platform for storing claims, documents, and user data. All the standard operations, like creating, reading, updating, and deleting records, work smoothly, and dashboards can quickly retrieve the information needed.

**Following Best Practices**

Throughout the system, I made sure to follow good programming practices:

* Separation of concerns: Controllers manage actions, models hold data, and views display the information.
* Validation and error handling: I checked user inputs carefully and handled errors in a way that keeps the system stable.
* Security: Roles control access so only the right users can submit, approve, or view claims and documents.
* Performance: I optimized data retrieval and included real-time updates to make the system fast and responsive.

**Implemented Features**

* Claims Management: Users can submit claims with multiple documents, view them, and manage approvals. Totals are calculated automatically to make sure they are correct.
* Dashboards: Personalized dashboards show the right information for lecturers, coordinators, and managers, including pending approvals, total claims, and document summaries.
* Real-Time Updates: Users get instant notifications when claim statuses change.
* Document Handling: All files are stored securely, and only authorized users can download them.
* Chatbot Integration: A chatbot helps users by answering questions, analyzing claims, and giving guidance on documents and analytics.

**Unit Testing**

I included unit tests to make sure the system works correctly. These tests cover total calculations, input validation, and secure document access. Running tests regularly helped me maintain reliability throughout development.

**Libraries and Tools**

I used several tools and libraries to make the system work properly, including ones for database management, real-time updates, PDF analysis, and user authentication. This helped ensure that all the features are reliable and secure.

**Operating the System**

* Users can submit claims, attach multiple documents, and check the status of their claims.
* Approvers can review, approve, or reject claims.
* The chatbot can answer questions and analyze uploaded documents.
* Dashboards provide an overview of activity and personalized insights.

**Best Practices in the System**

* Security measures make sure that only the right users can access sensitive data.
* Error handling prevents the system from crashing and keeps it running smoothly.
* Optimizations ensure the system runs fast and efficiently.
* Usability improvements, like consistent dashboards and a responsive chatbot, make it easier to use.

**Conclusion**

I successfully implemented the recommendations, which made the system more secure, reliable, and user-friendly. By applying good programming practices and focusing on both functionality and usability, I was able to create a system that meets both academic requirements and real-world standards.