NAME: Mmasebotsana

SURNAME: Nkosi

STUDENT NUMBER: ST10452632

MODULE CODE: PROG6212

MODULE NAME: PROGRAMMING 2B

POE PART 1: Project Planning and Prototype Development

GitHub Repository Link:

https://github.com/Nkosi-Mmasebotsana/CMCS-Prototype

(1) DOCUMENTATION (REPORT):

(1) Reason for Design Choice:

For this project, the Model-View-Controller (MVC) architecture was selected as the framework to build the Contract Monthly Claim System (CMCS). MVC helps separate different parts of the system, which is important for managing both the user interface and the behind-the-scenes logic in a clear way. Here's how each part works:

- **The Model** part deals with all the data structures, like lecturers, claims, supporting documents, and user roles.
- It makes sure that the system can store and get data correctly.

- **The View** part shows the user interface, letting lecturers submit claims and coordinators or managers check them.
- This part is focused on making the interface easy to use, accessible, and clear.

- **The Controller** part connects everything.
- It handles user actions, tells the Model to do something, and shows the right View back to the user.

Using MVC brings several benefits for this project.
It makes the system easier to maintain, so changes to the interface or the back end can be made without messing up the other parts. It also makes the system scalable, which means we can add new features like automated approval processes and HR reports in future iterations. MVC is commonly used in .NET Core web development, which fits well with creating a professional and reliable application. Also, the framework supports unit testing, which will be important for checking the system's functions in Part 2 and Part 3 of the POE.

(2) Database Structure:

The database for CMCS is built in a way that makes sure all claim data is correct, reliable, and can be checked if needed. The main parts of the database are Lecturer, Claim, ClaimLine, SupportingDocument, and User.

- **Lecturer** holds all the important information about a lecturer, like their ID, full name, email, and hourly rate.
- Each claim is connected to a specific lecturer to make sure payments are accurate and that responsibilities are clear.

- **Claim** is for each monthly request a lecturer makes.
- It includes details like the claim ID, the month, total hours worked, total amount, the status of the claim, when it was submitted, and when it was approved.
- Each claim is linked to a lecturer using their ID to keep the data connected properly.

- **ClaimLine** shows all the small parts of a claim, like how many hours were worked, the rate per hour, and the total for that part.
- This helps track different tasks a lecturer might do, like teaching, grading, or giving advice.

- **SupportingDocument** keeps track of any files a lecturer uploads, including the document ID, name, where it's stored, and when it was uploaded.
- A claim can have several documents to help verify the work and create a record for checking later.

- **User** refers to people like Programme Coordinators and Academic Managers.
- They have a user ID, full name, email, and a role, which helps control who can approve or reject claims.

This setup allows each lecturer to have many claims, each claim to have many line items, and each claim to have several documents. The design is also flexible, so it can be expanded later to include new features like reports, HR tools, or invoicing.

(3) GUI Layout:

The CMCS prototype GUI was designed with user-friendliness and intuitive navigation as its main objectives. The interface is visually clear, featuring consistent design elements and straightforward access to all essential functions.

- **Lecturer View:** This section includes a clean submission form allowing lecturers to input hours worked, hourly rate, and additional notes.
- A clearly marked 'Submit' button ensures ease of use.
- The upload section displays file names after selection, confirming document attachment.

- **Coordinator/Manager View:** This view presents pending claims in an organized table, displaying crucial details such as lecturer name, month, total hours, total amount, and claim status.
- Prominent 'Approve' and 'Reject' buttons facilitate quick verification.

- **Design Considerations:** The interface employs a consistent colour scheme and font hierarchy to improve readability.
- Navigation remains consistent across views, enabling users to switch between screens with minimal effort.
- Dummy data and static messages are used in this prototype stage to simulate interactions without affecting actual database records.

The layout emphasizes efficiency and clarity, ensuring users can complete tasks quickly and accurately. The design also prepares for future enhancements, such as real-time status updates, automated calculations, and dynamic document management.

(4) Assumptions or Constraints:

Several assumptions and constraints were implemented during the design of the CMCS prototype:

(a) Each lecturer has a fixed hourly rate recorded in the system.
(b) Only Programme Coordinators and Academic Managers have the authority to approve or reject claims.
(c) Every claim is associated with a lecturer, and each supporting document is linked to a specific claim.
(d) Claim statuses are limited to predefined values: Draft, Submitted, Approved, and Rejected.
(e) The prototype's GUI uses dummy data; clicking buttons triggers static messages without modifying any actual data.
(f) The system is designed for desktop or tablet use; mobile responsiveness is not required at this stage.

These considerations ensure the prototype remains realistic and manageable, while allowing for future expansion and functional improvements in Part 2 and Part 3.

(5) AI Declaration:

I declare that I used Artificial Intelligence (AI) tools, including OpenAI's ChatGPT and DeepSeek, during the preparation of this report.

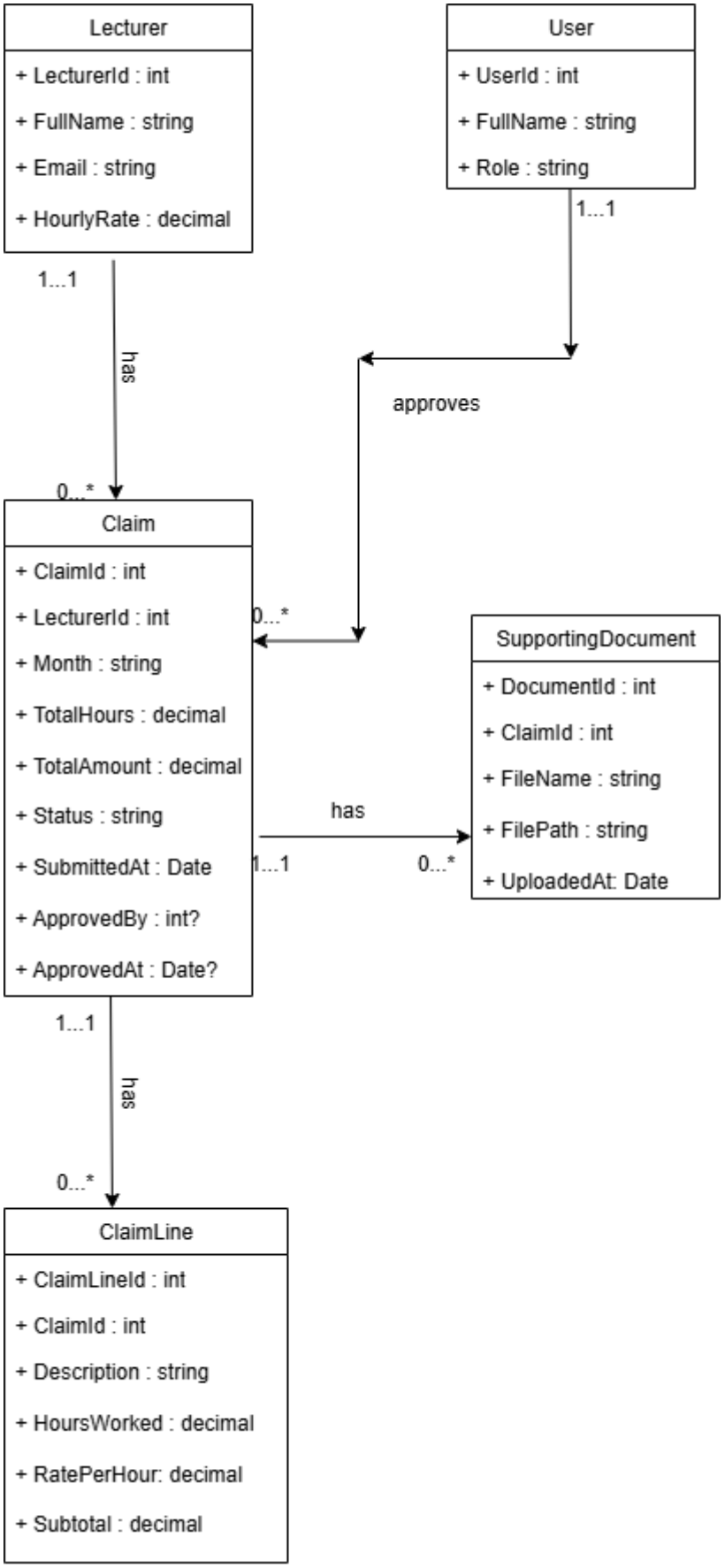
- I extensively used ChatGPT to assist in structuring, drafting, and refining the written sections of the report to enhance clarity, coherence, and professional presentation.

- I specifically asked ChatGPT to help me clean up my code using Visual Studio Code, to improve the UI design, and incorporate a Lecturer View to my interface, to make the prototype more user-friendly and realistic.

- I used DeepSeek to compare my explanations, design choices, and report content with ChatGPT's suggestions, which helped me verify accuracy and consistency before finalizing my submission.

All project implementation, database design, and coding were done independently, and I have ensured that all content accurately reflects the work I personally completed.

(2) UML CLASS DIAGRAM FOR DATABASES:



The UML class diagram illustrates the main entities and relationships needed for the Contract Monthly Claim System (CMCS). It organizes data in a logical manner, which allows for accurate submission for claims, approvals, and necessary documentation.

Entities and Attributes:

- **Lecturer:** stores information about contractors, which includes LecturerId, FullName, Email, and HourlyRate.

- **Claim:** refers to a monthly request that includes details like ClaimId, Month, TotalHours, TotalAmount, Status, SubmittedAt, and ApprovedAt.

- **ClaimLine:** contains a detailed breakdown of each claim item, including ClaimLineId, Hour, and Subtotal.

- **SupportingDocument:** contains metadata for uploaded claim documents, including DocumentId, FileName, FilePath. And UploadedAt.

- **User:** refers to Programme Coordinators or Academic Managers who handle and authorize claims. This includes their UserId, FullName, Email, and Role.

Relationships:

- **Lecturer – Claim (1 to Many):** A lecturer can have several claims associated with them.
- **Claim – ClaimLine (1 to Many):** A claim can have multiple line items that detail specific work entries.
- **Claim – SupportingDocument (1 to Many):** A claim can have multiple documents associated with it for verification purposes.
- **User – Claim (1 to Many):** A user can either approve or reject several claims at once.

(3) PROJECT PLAN:

PHASE	TASK	DESCRIPTION	DEPENDENCIES	ESTIMATED DURATION
PART 1: Planning & Prototype	Requirements Gathering	Identify core system requirements: claim, submission, approval, supporting documents, status tracking.	None	2 days
	UML Class Diagram	Design database structure including entities (Lecturer, Claim, ClaimLine, SupportingDocument,User) and relationships.	Requirements Gathering	2 days
	UI Wireframing	Create Figma or paper wireframes for all views (Lecturer, Coordinator, Manager, HR).	Requirements Gathering	1 day
	MVC Project Setup	Create ASP.NET Core MVC project skeleton in Visual Studio 2022.	None	1 day
	Build Dummy UI	Implement controllers and views with static data and messages (no functional claims).	MVC Project Setup, Wireframes	3 days
	Internal Review & Version Control	Test dummy UI and push to GitHub. Ensure clear commit messages.	Build Dummy UI	1 day
Part 2: Functional Implementation	Database Setup	Create SQL database and table according to UML diagram.	UML Class Diagram	2 days
	CRUD Implementation	Add Create, Read, Update, Delete functionality for Lecturers, Claims, and ClaimLines.	Database Setup	4 days

	File Upload Feature	Enable uploading of supporting documents and link to claims.	CRUD Implementation	2 days
	Claim Status Tracking	Implement dynamic status updates for claims (Pending, Approved, Rejected).	CRUD Implementation	2 days
	Role-Based Views	Design separate views for Lecturers, Coordinators, and Managers with relevant actions.	CRUD Implementation	2 days
	Error Handling & Unit Testing	Add validation, error messages, and unit tests to ensure data consistency and reliable functionality.	All Functional Tasks	3 days
	Version Control	Commit and push functional application to GitHub(min- 5 commits)	All Functional Tasks	1 day
Part 3: Automation & POE Finalization	Automate Lecturer Claim Submission	Implement auto-calculation of claim totals and input validations.	Functional Implementation	2 days
	Automate Coordinator/Manager Approval	Implement automated workflows for claim verification, approval, and rejection.	Functional Implementation	3 days
	Automate HR Processes	Implement auto-generation of reports/invoices and lecturer data management features.	Functional Implementation	3 days
	Integration Testing	Test all automated processes, workflows, and data consistency.	Automation Tasks	2 days
	Prepare POE Documentation & Presentation	Compile UML diagrams, project plan, screenshots, explanations, and	All Phases Completed	2 days

		PowerPoint overview of the system.		
	Final Review & Submission	Verify GitHub repository, ensure project runs, push final commits to ARC	Documentation & Presentation	1 day

REFERENCES:

app.diagrams.net. (n.d.). *Flowchart Maker & Online Diagram Software*. [online]

Available at: https://app.diagrams.net/?libs=general.

[Accessed 17th September 2025]

Lecturer slides by Toritseju, Toju Oni.

[Accessed 01 September 2025 – 03 September 2025]

Troelsen, A. and Japikse, P. 2021.Pro C# 9 with .NET 5: Foundational Principles and Practices in Programming. 10th ed.Apress.

[Accessed 12 – 18th September 2025]