



PROG 6212

POE PART

1

ST10438003

KEVOSHEN MOOLDEY



1.DOCUMENTATION

Part 1 Documentation

1. Design Choices

When I started planning the Contract Monthly Claim System (**CMCS**), I wanted to make sure my design was both **practical and realistic**. Since the whole purpose of this system is to help **Independent Contractor (IC)** lecturers easily submit their monthly claims while also giving Programme Coordinators and Academic Managers a way to review and approve them, I had to keep **simplicity, accuracy, and usability** in my mind.

I chose to build the prototype using **ASP.NET Core MVC** because it gives a very clean way to separate responsibilities:

- The **Models** represents the database entities like **Lecturer, Claim, SupportingDocument, ClaimReview, and FinalApproval**.
- The **View** is the GUI that the user interacts with. I wanted it to look professional but still simple enough for lecturers and coordinators to use without confusion.
- The **Controller** acts as the “middleman” that connects the interface to the data and logic.

By following MVC, my project is structured and future-proof. Later, when functionality like database queries and authentication come in, the foundation I’m laying now will make it much easier to expand the system without breaking things.

Another design choice I made was to think about **roles** right from the beginning. There are three very distinct roles in this system:

1. **Lecturers** who will log in, submit claims, and upload documents.
2. **Programme Coordinators** who will review and approve or reject claims with their own accounts.
3. **Academic Managers** who will have the final oversight, making sure of accountability through their own accounts.

There are three distinct roles managed through a single User table: Lecturers, Programme Coordinators, and Academic Managers. Each user has a specific role assigned (**via the Role field**), which determines what they can see and do in the system. This prevents redundancy and makes sure there is consistency across all users while maintaining clear role-based functionality. *****

2. Database Structure

My database structure was designed to reflect the real-world process of how monthly claims are submitted, reviewed, and approved. Each lecturer submits a claim based on their worked hours, attaches supporting documents, and then that claim goes through a two-step approval process: first by a **Programme Coordinator** and then by an **Academic Manager**. By designing the database around this, my system will guarantee accountability and a clear audit trail.

Here's how I structured my main entities:

User

- UserID (PK)
- FullName
- Email
- HourlyRate (*nullable – applies only to Lecturers*)
- Role (*Lecturer / Coordinator / Manager*)

[The User table merges Lecturer, ProgrammeCoordinator, and AcademicManager, with Role differentiating their responsibilities. This prevents redundancy while maintaining role-based access.]

• **Claim**

- ClaimID (PK)
- LecturerID (FK → Lecturer)
- HoursWorked
- TotalAmount
- ClaimMonth
- Status (*Pending/Approved/Rejected*)
- SubmissionDate

[This is the central entity. A claim is connected to a lecturer and records the number of hours worked. TotalAmount is calculated using $\text{HoursWorked} \times \text{HourlyRate}$. ClaimMonth allows grouping claims by month. “Status will be updated automatically based on the decisions in ClaimReview (*coordinator*) and FinalApproval (*manager*)]

• **SupportingDocument**

- SupportingDocumentID (PK)

- ClaimID (**FK** → **Claim**)
- DocumentType

[Each claim can have one or more documents attached, like timesheets or signed proof of hours. Linking documents directly to a claim gives evidence and accountability.]

- **ClaimReview**

- ClaimReviewID (**PK**)
- ClaimID (**FK** → **Claim**)
- ProgrammeCoordinatorID (**FK** → **ProgrammeCoordinator**)
- ReviewDate
- Decision (**Approved/Rejected**)

[ClaimReview is a linking entity that stores the interaction between a claim and a coordinator. This allows multiple coordinators to review claims if needed, while also keeping a log of decisions.]

- **FinalApproval**

- FinalApprovalID (**PK**)
- ClaimID (**FK** → **Claim**)
- AcademicManagerID (**FK** → **AcademicManager**)
- ApprovalDate
- Decision (**Approved/Rejected**)

[FinalApproval records the ultimate decision by the Academic Manager. By separating this from ClaimReview, the database distinguishes between first-level and final-level approval. This also creates an auditable trail.]

Relationships between the 4 entities:

- One **User (Lecturer)** can submit many **Claims**.
- One **Claim** can have many **ClaimReviews (by Coordinators)**.
- One **Claim** can have one **FinalApproval (by a Manager)**.
- **User** roles control what actions they can perform on **claims**.

3.GUI LAYOUT SECTION

My GUI layout was designed to directly reflect my database structure above, ensuring each entity has a corresponding interface screen or function.

1. Login Screen

- **Fields:** Username, Password.
- Role-Based Access determined by the **User.Role property** (Lecturer, Coordinator, or Manager).
 - If **Lecturer** logs in → dashboard shows their own claims.
 - If **Coordinator** logs in → dashboard shows claims pending review.
 - If **Manager** logs in → dashboard shows claims pending final approval.
- **Reasoning:** Ties into **User roles (Lecturer/Coordinator/Manager)** from documentation.

2. Lecturer Dashboard

Main functions for lecturers:

- **Submit New Claim** → opens a form.
- **View My Claims** → list of all claims with statuses.
- **Upload Supporting Documents** → attached per claim.

Submit Claim Form Fields:

- Hours Worked (**numeric field**).
- Claim Month (**dropdown: Jan–Dec**).
- Upload Supporting Document (**button** → **browse file**).
- Submit (**button**).

View My Claims Table:

- ClaimID | ClaimMonth | HoursWorked | TotalAmount | Status | SubmissionDate | Documents
 - Status will update as it moves through coordinator → manager.
-

3. Programme Coordinator Dashboard → User (Role: Coordinator)

This is the **first-level approval** screen.

- **Pending Claims List:** Shows claims submitted by lecturers awaiting review.
 - Columns: ClaimID | Lecturer | ClaimMonth | HoursWorked | TotalAmount | SubmissionDate.
 - **Review Panel (when a claim is selected):**
 - Lecturer details (**FullName, Email**).
 - Claim details (**Hours, TotalAmount**).
 - Supporting Documents (**clickable links**).
 - **Decision options:** Approve / Reject.
 - Comments text box (**for notes like “Missing timesheet” or “Verified against attendance”**).
 - Save Decision button → creates a record in **ClaimReview** table.
-

4. Academic Manager Dashboard → User (Role: Manager)

This is the **final approval** stage.

- **Pending Final Approvals List:**
 - Columns: ClaimID | Lecturer | Coordinator Decision | ClaimMonth | HoursWorked | TotalAmount.
- **Final Approval Panel (when claim is opened):**
 - All claim details.
 - Coordinator’s decision + comments.

- Supporting documents.
- **Final Decision options:** Approve / Reject.
- Approval Date auto filled when submitted.

Once approved → Status updates to **Approved**.

If rejected → Status = **Rejected**, visible back to lecturer.

5. Claim Review History (Optional)

- A screen/tab where all users (**depending on their role**) can see **past claims and their decisions**.
 - Table format: ClaimID | Lecturer | ClaimMonth | Coordinator Decision | Manager Decision | Final Status.
 - Purpose: Provides transparency + audit trail (ties into **ClaimReview** + **FinalApproval** entities).
-

6. Navigation Flow (Role-based GUI experience)

- **Lecturer:** Login → Lecturer Dashboard (**Submit/View Claims**).
 - **Coordinator:** Login → Coordinator Dashboard (**Pending Reviews → Review Decision**).
 - **Manager:** Login → Manager Dashboard (**Pending Approvals → Final Decision**).
-

Assumptions and Constraints

Assumptions :

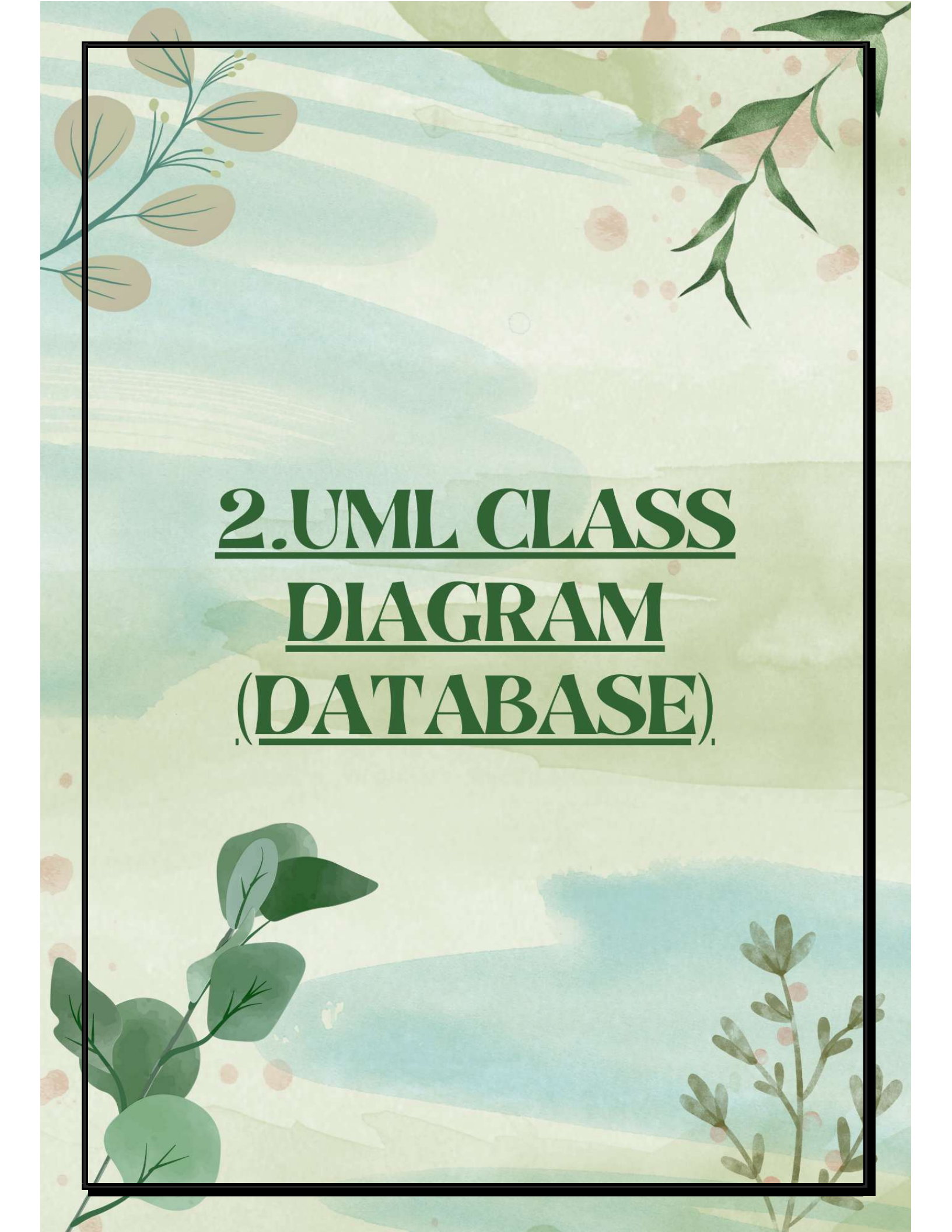
-Every lecturer has a fixed hourly rate stored in the system, making sure that claims can be calculated consistently.

- Lecturers will upload valid supporting documents such as timesheets or proof of hours. The system will not verify the accuracy of these documents, only that they have been attached.
- Coordinators and Managers are expected to log in with their own accounts to review and approve claims.
- The system will eventually automate all claim calculations so that lecturers don't need to calculate totals manually.
- It is assumed that all users will have stable internet access and use compatible web browsers to access the system effectively.
- The prototype will be developed using **ASP.NET Core MVC**, with the assumption that the university has access to Visual Studio and the necessary hosting resources for future deployment.

Constraints :

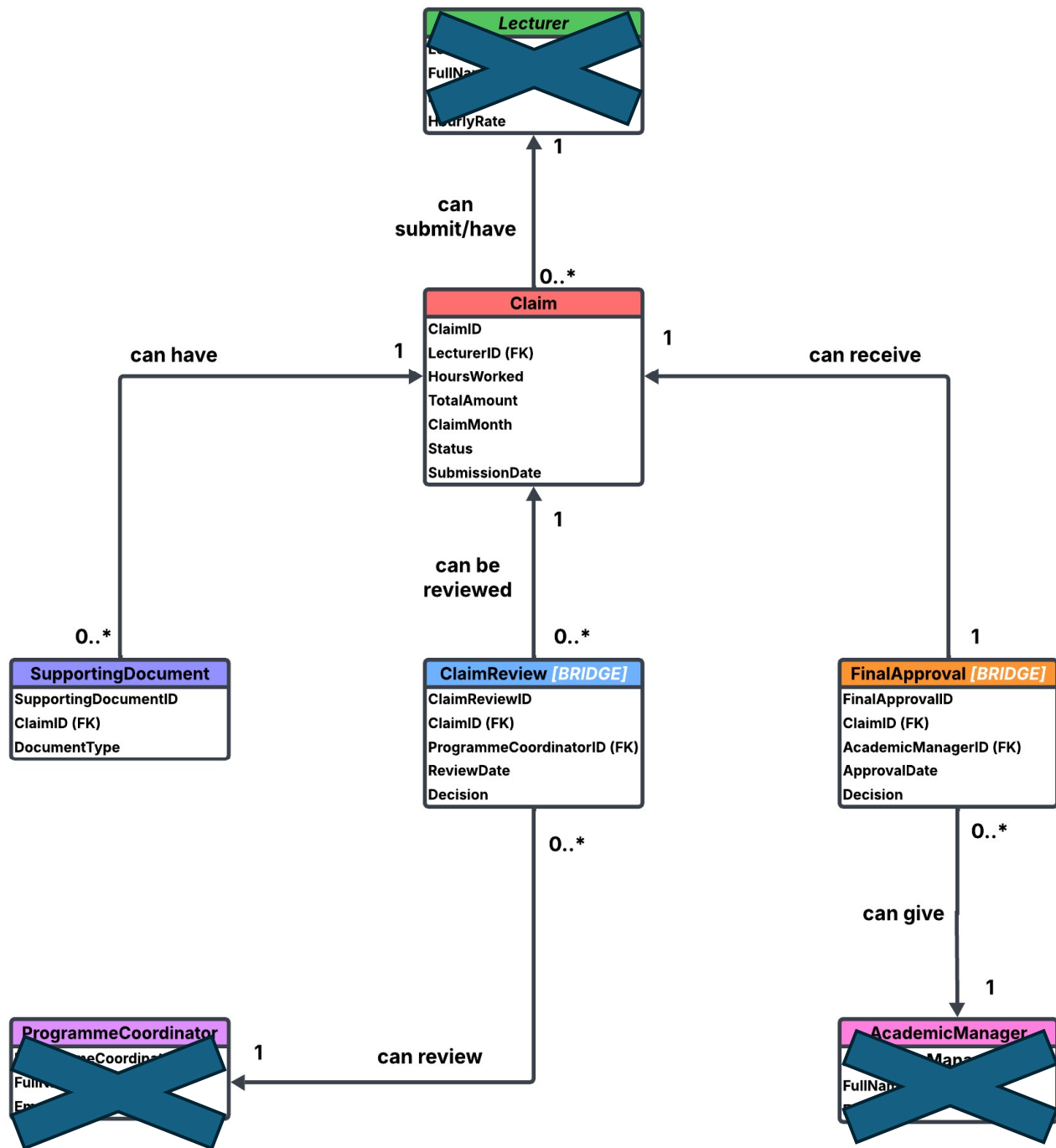
- Since this is only Part 1, the system functions purely as a prototype. It focuses on the core claim workflow, claim submission, supporting document uploads, coordinator review, and manager approval. None of the backend functionality (**such as database saving, authentication, or file uploads**) is active yet. This limits what can be tested, but the interface fully demonstrates the intended design and flow.
- Security features have not yet been implemented. For example, passwords are not being validated or encrypted at this stage. These elements will be added in later development phases.
- The prototype focuses exclusively on the claim workflow. Additional features like reporting, notifications, or payroll integration are beyond the current scope.
- Testing is currently limited to a local Visual Studio environment. The system has not yet been deployed online or to Azure.
- The design layout has been intentionally kept simple, following a clean university-style portal interface to make sure of clarity and ease of use.
- Future versions of the system will be required to comply with university data protection and storage policies, especially regarding lecturer information and uploaded documents.
- System performance and database scalability have not yet been tested, as these will only become relevant once full backend functionality is integrated.

MY FEED BACK WAS TOO “Consider additional system constraints which may apply”. SO, I DID THAT AND IT CAN BE SHOWN ABOVE WITH THE EXTENDED VERSION OF MY ASSUMPTIONS AND CONSTRAINTS.



2.UML CLASS DIAGRAM (DATABASE)

UML CLASS DIAGRAM FOR CMCS



***DIAGRAM WAS CREATED AND WHEN ACCORDING TO /
MATCHES THE INFORMATION DONE IN MY DOCUMENTATION
ABOVE. LINKS DIRECTLY TO MY DATABASE STRUCTURE***

DISCLAIMER: IN THE FEEDBACK FROM THE TEACHER I WAS TOLD FOR THE UML DIAGRAM THAT

“ Camel casing has not been utilized when naming the attributes, redundancy is present in terms of Lecturer, Academic Manager, and Programme Coordinator. Consider creating a User Table with a role type and relating it to the respective other entities ”

[I DID CHANGE IT TO USE A USER TABLE INSTEAD TO REDUCE REDUNDANCY BUT I HAVENT IMPLEMENTED IT IN PART 2, THIS CHAGE WILL BE USED IN PART 3, THE REASON WHY I DIDN'T USE IT IN PART 2 WAS BECAUSE:

In **MY** system, I chose not to create a combined **User model** or separate models for **Lecturer**, **Coordinator**, or **Manager** because the focus of the project was managing **claims**. All the necessary information, who submitted the claim, its status, notes, and supporting documents, was stored directly in my **Claim model**. This approach kept the system simple, avoided complexity, and still allowed me to track the full workflow of submission, review, and approval without needing extra tables or models.

“SINCE THERE WAS NO NEED FOR ROLES OR IDENTITIES, I USED THIS APPROACH BUT IF IT NEEDS TO BE CHANGED IN PART 3 IT WILL”.



3. PROJECT PLAN

PROOF OF PROJECT PLAN DONE IN EXCEL

CMCS PROJECT PLAN - INSY 6212.xlsx • Saved						
File Home Insert Draw Page Layout Formulas Data Review View Automate Help						
Clipboard Font Alignment Number Styles Cells Editing Sensitivity Add-ins						
A1 TASK ID						
TASK ID	TASK DESCRIPTION	START DATE	END DATE	DURATION (33 DAYS)	DEPENDANCIES	NO
1	1 REQUIREMENT GATHERING & CONFIRMATION	16-Sept-25	17-Sept-25	2	NULL	Scope must be crystal clear to
1.1	Review existing documentation – Carefully go through design notes, ERD draft, and GUI ideas to ensure they align with the simplified prototype workflow. Identify any PROBLEMS early.	16-Sept-25	16-Sept-25	1	NULL	Foundation for stakeholder si
1.2	Stakeholder confirmation – Present refined scope to the executive and confirm that only claim submission, review, and approval will be built.	17-Sept-25	17-Sept-25	1	TASK 1.1 - Review existing documentation	This approval locks requireme
2	2 DATABASE DESIGN AND VALIDATION	18-Sept-25	20-Sept-25	3	TASK 1 - Requirement Gathering & Confirmation	Entities must support the GUI
2.1	Draft entity-relationship structure – Create a clear ERD with all primary and foreign keys. Map relationships between claims, lecturers, and approval entities ETC.	18-Sept-25	18-Sept-25	1	TASK 1 - Requirement Gathering & Confirmation	Ensures consistency with UML
2.2	Validate relationships – Double-check connections (e.g., one lecturer → many claims, claim → many supporting documents). Ensure claim approval flow is correctly represented.	19-Sept-25	19-Sept-25	1	TASK 2.1 - Draft entity-relationship structure	Peer or lecturer review recom
2.3	Adjust for prototype focus – Remove non-scope extras like payroll integration, financial reporting, or notifications. Keep the database lean and workflow-centered.	20-Sept-25	20-Sept-25	1	TASK 2.2 - Validate relationships	Keeps prototype focused and
3	3 GUI LAYOUT AND WIREFRAMING	21-Sept-25	23-Sept-25	3	TASK 1 - Requirement Gathering & Confirmation	Provides visual confirmation
3.1	Draft low-fidelity wireframes – Create quick sketches (paper or software) showing basic layouts for login, dashboards, and claim submission forms.	21-Sept-25	21-Sept-25	1	TASK 1 - Requirement Gathering & Confirmation	Captures navigation flow with
3.2	Refine into final layout – Convert sketches into neat, detailed wireframes with Bootstrap styling references. Ensure role-based dashboards (lecturer, coordinator, manager) are represented.	22-Sept-25	23-Sept-25	2	TASK 3.1 - Draft low-fidelity wireframes	Final version guides GUI codin
4	4 UML CLASS DIAGRAM	24-Sept-25	25-Sept-25	2	TASK 2 - Database Design & Validation	Must align with UML CLASS D
5	5 BACKEND SETUP (IMPLEMENTING DATABASE)	26-Sept-25	29-Sept-25	4	TASK 2 - Database Design & Validation	The Database is the backbone
5.1	Create tables & constraints – Code schema in SQL Server for Lecturer, Claim, SupportingDocument, ClaimReview, and FinalApproval. Add foreign key constraints to preserve.	26-Sept-25	27-Sept-25	2	TASK 2 - Database Design & Validation	This ensures error-free schem
5.2	Insert sample test data – Add a small dataset of lecturers, claims, and approvals for use in GUI testing and validation.	28-Sept-25	29-Sept-25	2	TASK 5.1 - Create tables & constraints	Add dummy data which helps
6	6 GUI FRONTEND DEVELOPMENT	30-Sept-25	05-Oct-25	6	TASK 3 - GUI Layout & Wireframing AND TASK 5 - Backend Setup	Core of the prototype.
6.1	Build login & dashboard – Develop login page with role-based redirection (lecturer, coordinator, manager). Each role sees a customized dashboard.	30-Sept-25	01-Oct-25	2	TASK 5 - Backend Setup	Authentication is prototype-c
6.2	Build claim submission form – Code form for lecturers to submit new claims (hours worked, month, upload supporting documents).	02-Oct-25	03-Oct-25	2	TASK 6.1 - Build login & dashboard	Links claim to database
6.3	Build claim tracking view – Develop table showing lecturers' claims and their statuses (Pending, Approved, Rejected). Coordinators and managers will see review lists.	04-Oct-25	05-Oct-25	2	TASK 6.2 - Build claim submission form	Tracks claim statuses and pro
7	7 WORKFLOW LOGIC IMPLEMENTATION	06-Oct-25	10-Oct-25	5	TASK 6 - Frontend GUI Development	Make system/prototype func
7.1	Submission validation rules – Add checks for required fields, valid hours, and document uploads. Prevents incomplete claims.	06-Oct-25	06-Oct-25	1	TASK 6.2 - Build claim submission form	Ensures data accuracy.
7.2	Approval/rejection logic – Enable Programme Coordinators to approve/reject claims, and Managers to give final decision. Store actions in ClaimReview and FinalApproval tables.	07-Oct-25	08-Oct-25	2	TASK 6.3 - Build claim tracking view	Manager decision logic imple
7.3	Payment simulation (basic) – Add placeholder logic that updates claim status to "Paid" once approved. No real finance integration.	09-Oct-25	10-Oct-25	2	TASK 7.2 - Approval/rejection logic	Shows end-to-end workflow
8	8 TESTING AND DEBUGGING	11-Oct-25	13-Oct-25	3	TASK 7 - Workflow Logic Implementation	Internal testing that ensures
9	9 USER FEEDBACK	14-Oct-25	16-Oct-25	3	TASK 8 - Testing & Debugging	Show prototype to executive
10	10 FINAL DOCUMENTATION UPDATES	17-Oct-25	18-Oct-25	2	TASK 9 - User Feedback & Iteration	Update UML, docs, GUI layout

*****IN MY ARC SUBMISSION IM GOING TO SUBMIT THE PDF PRINTED VERSION SO YOU CAN GET A BETTER LOOK AT IT AS IT IS QUITE SQASHED HERE*****

FOR THIS QUESTION MY FEEDBACK WAS :

“Timeline and implemented dates require minor changes in order to be realistic”

THEREFORE TO CHANGE IT AND MAKE IT BETTER I CHANGED IT :

Task	Original	Revised Dates	Notes
Requirement Gathering & Confirmation	16–17 Sept	16–18 Sept	Adds 1 day for documentation and feedback.
Database Design & Validation	18–20 Sept	19–23 Sept	Adds 2 days to reflect testing and ERD refinement.
GUI Layout & Wireframing	21–23 Sept	24–27 Sept	Follows database completion. Adds weekend buffer.
JML Class Diagram	24–25 Sept	28–29 Sept	Aligns with wireframes and DB adjustments.
Backend Setup (DB Implementation)	26–29 Sept	30 Sept–04 Oct	Adds testing and data validation buffer.
GUI Frontend Development	30 Sept–05 Oct	06–12 Oct	More realistic for multi-role UI development.
Workflow Logic Implementation	06–10 Oct	13–16 Oct	Moves after GUI: aligns with system logic integration.
Testing & Debugging	11–13 Oct	20–23 Oct	Expands testing for stability.
User Feedback	14–16 Oct	24–26 Oct	Now follows testing logically.

Total Duration: 16 Sept – 28 Oct (NOW 43 days)

WHAT I CHANGED :

Based on lecturer feedback, my project timeline was adjusted to be more realistic and industry aligned. Some development and testing tasks were extended by 1–2 days to allow for internal reviews, feedback integration, and debugging. This made sure of smoother workflow transitions between database setup, GUI implementation, and logic integration. The total project duration was extended from 33 to 43 days, providing a more achievable and professional project plan.



4. GUI/ UI

*****FOR THE GUI/ MVC FOR CMCS, I DETAILED VIDEO HAS BEEN SUBMITTED ON ARC THROUGH A YOUTUBE VIDEO WHERE I WALK YOU THROUGH THE PROTOTYPE OF THE CMCS*****

YOUTUBE VIDEO LINK IS:

<https://youtu.be/SX4-FEeQWJU>

MY CODE FOR THE APPLICATION HAS ALSO BEEN SUBMITTED THROUGH GITHUB REPOSITORY.

MY GITHUB REPOSITORY LINK IS:

<https://github.com/VCWVL/prog6212-part-1-Kevoshen-ST10438003.git>

“whatever feedback I was given for this question, me improving it can be shown through my commits through GitHub”

REFERENCES :

1.OpenAI (2025). ChatGPT. [online] ChatGPT. Available at:
<https://chatgpt.com/>.

2.wadepickett (2024). Get started with ASP.NET Core MVC. [online] Microsoft.com. Available at: <https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-9.0&tabs=visual-studio>.

3.tutorialsEU (2024). ASP.NET 8 MVC Tutorial for Beginners - C# web development made easy. [online] YouTube. Available at:
<https://www.youtube.com/watch?v=xuFdrXqpPB0>.

4.Taylor, R. (2021) *How to Document Your Project*. Available at:
www.projectdocs.com (Accessed: 17 September 2025).