Q.1

The Contract Monthly Claim System (CMCS) has been designed to simplify and improve how Independent Contractor (IC) lecturers submit their monthly claims, and how Programme Coordinators and Academic Managers verify and approve them. The system aims to remove unnecessary manual processes, reduce delays, and give all users a clear, transparent view of claim progress. From the start, the design focused on usability, clarity, and scalability, with a structure that can support more advanced features in later stages.

**Design Choices**

The system follows a layered design to separate the interface, business logic, and data management. This will make the application easier to maintain, upgrade, and test as it grows. For the front end, I chose either Windows Presentation Foundation (WPF) or ASP.NET Core MVC because both offer strong support for building rich, responsive interfaces in a .NET environment. The GUI will use a role-based design so that each user sees only the features relevant to them:

* Lecturers will access claim forms, upload supporting documents, and view claim statuses.
* Programme Coordinators and Academic Managers will see dashboards listing pending claims for review, approval, or rejection, with comment options.

This approach supports clear navigation, reduces user error, and ensures sensitive actions are only available to the correct roles.

**Database Structure**

The database will include five key entities: Lecturer, Claim, Document, Approval, and UserRole. A Lecturer can create many Claims, and each Claim can link to multiple Documents as proof. The Approval entity tracks the verification and approval process, including who approved it and when. The UserRole entity will manage user permissions. This structure ensures data integrity, supports clear tracking of claims, and will allow for detailed reporting and auditing later.

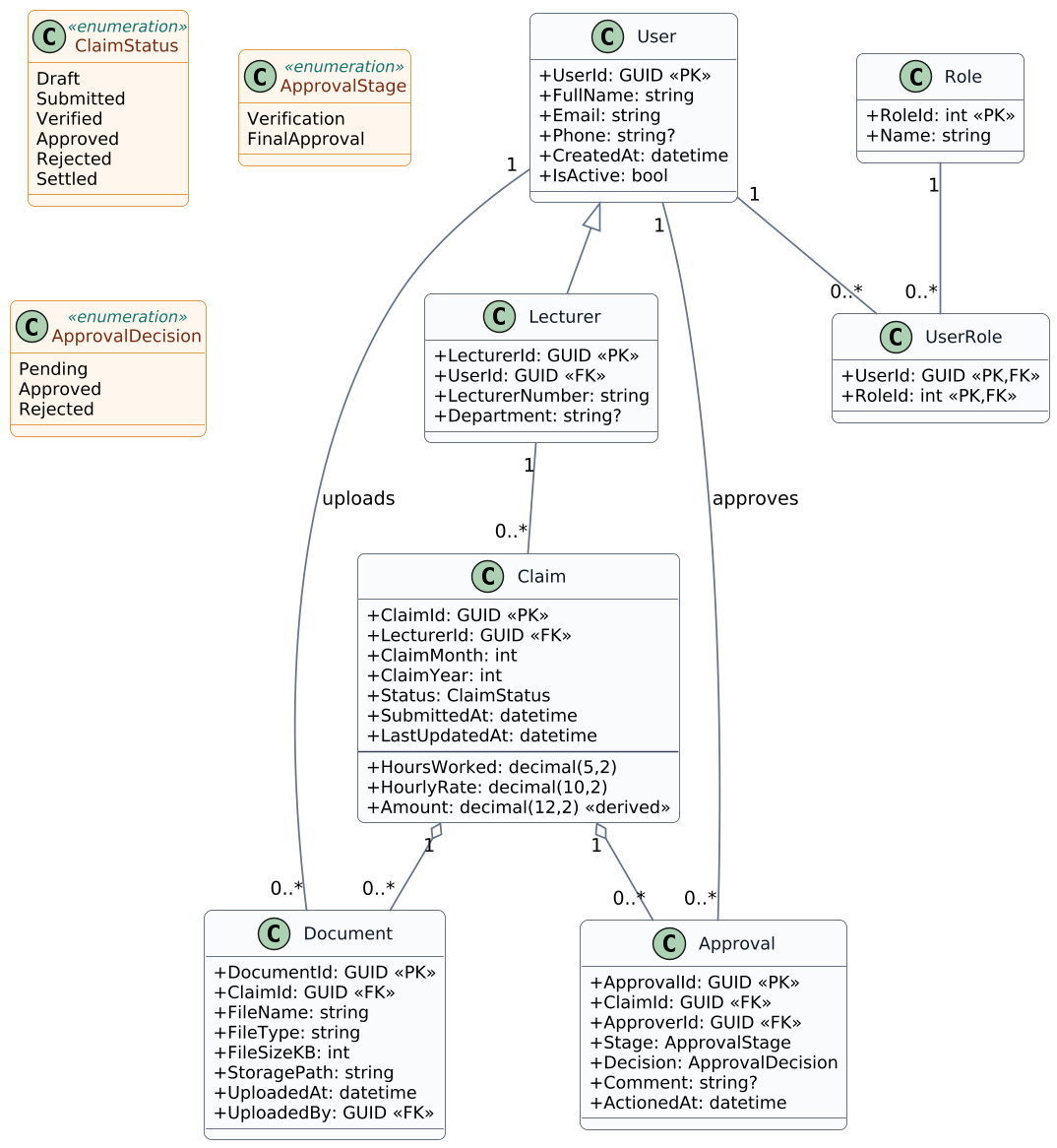
**GUI Layout**

The GUI will use a clean, dashboard-style layout. Lecturers will have a submission page with simple forms, file uploads, and a table showing all their claims and statuses. Coordinators and managers will have a verification page listing all claims, with buttons to approve or reject and a status tracker. A consistent top navigation bar, colour-coded statuses, and clear labelling will make the interface intuitive and quick to learn.

**Assumptions and Constraints**

It is assumed that users have basic computer literacy and stable internet access. Because this is only a prototype, no login or security features will be included yet — these will be implemented later. The main constraint is that the system at this stage will be non-functional and only show the design and layout. This ensures the design can be reviewed and refined before development begins.

Q.2



Q.3

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Tasks & Activities** | **Dependencies** | **Timeline** |
| Part 1 – Prototype UI | System Analysis & Requirements • Gather requirements, define features and user roles. | None | Week 1 |
| Part 1 – Prototype UI | Database & UML Design • Design DB schema & UML class diagram. | Requirements completed | Week 1 |
| Part 1 – Prototype UI | GUI Wireframes & Layout • Design mock UI screens using Bootstrap. | UML & entity structure | Week 2 |
| Part 1 – Prototype UI | Front-End Prototype Build • Build static ASP.NET Core MVC UI pages. | Wireframes completed | Week 2 |
| Part 1 – Prototype UI | Documentation & Report • Write design documentation and assumptions. | Prototype completed | Week 3 |
| Part 2 – Functional Web App | Implement Claim Submission • Build claim form (hours worked, hourly rate, notes). | Database schema complete | Week 4 |
| Part 2 – Functional Web App | Approval Workflow • Create coordinator/manager view to approve/reject claims. | Claim submission functional | Week 4-5 |
| Part 2 – Functional Web App | Document Upload • Upload control linked to claims, allow only PDF/DOCX/XLSX. | Claim submission functional | Week 5 |
| Part 2 – Functional Web App | Status Tracking • Show and update claim status dynamically. | Approval logic functional | Week 6 |
| Part 2 – Functional Web App | Testing & Error Handling • Add unit tests and user-friendly error handling. | All major features complete | Week 6 |
| Part 3 – Automation & POE | Automation – Lecturer View • Auto-calculate claim totals; add validation. | Part 2 completed | Week 7 |
| Part 3 – Automation & POE | Automation – Coordinator/Manager • Auto-check claims vs rules, streamline approval. | Lecturer view automation | Week 7 |
| Part 3 – Automation & POE | Automation – HR View • Generate invoices/reports and manage lecturer profiles. | Approvals complete | Week 8 |
| Part 3 – Automation & POE | PowerPoint Presentation • Show system architecture, UI, features, and flow. | System feature-complete | Week 9 |
| Part 3 – Automation & POE | Version Control & Delivery • Ensure 10+ Git commits, submit repo and docs. | All work complete | Week 9 |