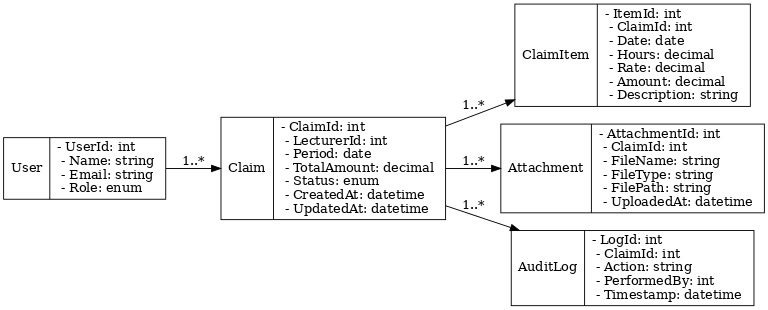
Contract Monthly Claim System — Part 1: Project Planning & Prototype (POE)

**Design Choices, Database Structure, and GUI Layout (Summary)**  
The system prioritises clarity, auditability, and minimal user effort. A modular domain model was selected, separating Users (Lecturer, Programme Coordinator, Academic Manager), Claims, ClaimItems, Attachments, and Workflows to ensure maintainability and enforce clear role-based access control (RBAC). A relational database such as Azure SQL or SQL Server is assumed to store normalized tables with foreign keys for referential integrity.

Core tables include: Users (UserId, Name, Role, Email), Claims (ClaimId, LecturerId, Period, TotalAmount, Status, CreatedAt, UpdatedAt), ClaimItems (ItemId, ClaimId, Date, Hours, Rate, Amount, Description), Attachments (AttachmentId, ClaimId, FileName, FileType, FilePath, UploadedAt), and AuditLogs (LogId, ClaimId, Action, PerformedBy, Timestamp). Constraints include unique indexes on ClaimId + Period per Lecturer, non-null foreign keys, and file size limits for uploads. Key assumptions are that lecturers are pre-registered, claims follow hourly-rate calculations, and offline or partial saves are not required at this prototype stage.

**UML Class Diagram**

* User { UserId: int; Name: string; Email: string; Role: enum; }
* Claim { ClaimId: int; LecturerId: int; Period: date; TotalAmount: decimal; Status: enum; CreatedAt: datetime; UpdatedAt: datetime; }
* ClaimItem { ItemId: int; ClaimId: int; Date: date; Hours: decimal; Rate: decimal; Amount: decimal; Description: string; }
* Attachment { AttachmentId: int; ClaimId: int; FileName: string; FileType: string; FilePath: string; UploadedAt: datetime; }
* AuditLog { LogId: int; ClaimId: int; Action: string; PerformedBy: int; Timestamp: datetime; }  
  Relationships: User (1) — (M) Claim; Claim (1) — (M) ClaimItem; Claim (1) — (M) Attachment; Claim (1) — (M) AuditLog.



**Project Plan (4-Week Prototype)**  
Week 1: Requirements refinement, UML & database schema (2 days), wireframes (2 days), GitHub repo setup & branch strategy (1 day).  
Week 2: UI prototype (WPF or MVC Razor) — core screens & navigation (4 days), static asset creation (logo, icons) (1 day).  
Week 3: File upload UI (mock), claim submission flow (front-end only), status tracking UI (5 days).  
Week 4: Documentation finalisation, report preparation, 5 Git commits distributed evenly across weeks, internal review, and packaging (3 days).  
Dependencies: UML → Wireframes → UI prototype → Documentation. Milestones: UML complete (end of Week 1), UI prototype (end of Week 2), report & repository submission (end of Week 4).

**GUI Design**  
Core screens: Login (mock), Dashboard (claims summary with submission button), New Claim (form for Claim + ClaimItems, add attachment control), Claim Detail (status timeline, attachments, audit log), and Approvals Panel (Programme Coordinator/Manager: pending claims with approve/reject). The design ensures usability through clear CTAs, consistent status badges (Submitted, Under Review, Approved, Rejected), upload progress indicators, and breadcrumb navigation.

**Technology Choice & Version Control**  
For GUI development, either WPF (desktop-focused, strong data binding) or MVC with Razor (web-based, scalable) is suitable. WPF is intuitive for standalone prototypes, while MVC better mirrors enterprise deployment. Version control will be managed through GitHub with at least five descriptive commits documenting progress from UML creation to final documentation. This ensures traceability, collaboration, and professional development practice.