D Mboni ST10437614— PROG 6212 Part 1 Report

**Background & Rationale.** In our department, Independent Contractor (IC) lecturers still submit monthly claims by emailing spreadsheets and attachments to different people. It’s easy for files to go missing, formulas to break, or for two versions of the same sheet to circulate. Approvals are also unclear—sometimes a coordinator signs off, but the manager hasn’t, and no one can tell where the claim is stuck.

The **Contract Monthly Claim System (CMCS)** is my proposed fix. It puts the whole process into one simple flow:

1. the lecturer captures their claim lines,
2. supporting documents are uploaded once, and
3. the Programme Coordinator (PC) and Academic Manager (AM) review in order.

The system always shows a visible status and keeps an audit trail, so we know who did what and when. For Part 1, I’m only building a front-end prototype (no backend logic yet). The class diagram I designed points to how the database will look when we add functionality.

**Design Choices & Assumptions.**

**Users & roles.** Instead of creating separate tables for each type of person, I used a simple User + Role model. A user can be a Lecturer, PC, or AM based on their role. This avoids duplication and makes permissions easier to manage.

**Claims & lines**. A Claim is the monthly “container”. Inside it, ClaimLine items store the actual work (date, module, hours, rate). Totals are calculated from these lines. I plan to store the totals on the Claim as well—this is a performance choice for later when there are many lines.

**Documents**. A Document entity holds file metadata (name, type, who uploaded, when). In a real build, files would live on a file server or S3, but the database would still track them neatly.

**Approvals**. Because our process needs two checks, I added an Approval entity with Stage (PC or AM), Decision, Comment, and DecidedAt. If policy changes, we can add more stages without redesigning everything.

**Status flow**. The Claim has a clear state machine: Draft → Submitted → UnderReview → Approved/Rejected. This feeds the UI so lecturers and admins always know where things stand.

**Assumptions & constraints.**

For this prototype, I will use one claim per lecturer per month to prevent duplicates; the hourly rate will default from the lecturer profile but can be overridden at line level for exceptions (e.g., special sessions); approvals occur in sequence—Programme Coordinator first, then Academic Manager; and for supporting evidence the system will accept only PDF or image files to keep verification consistent.

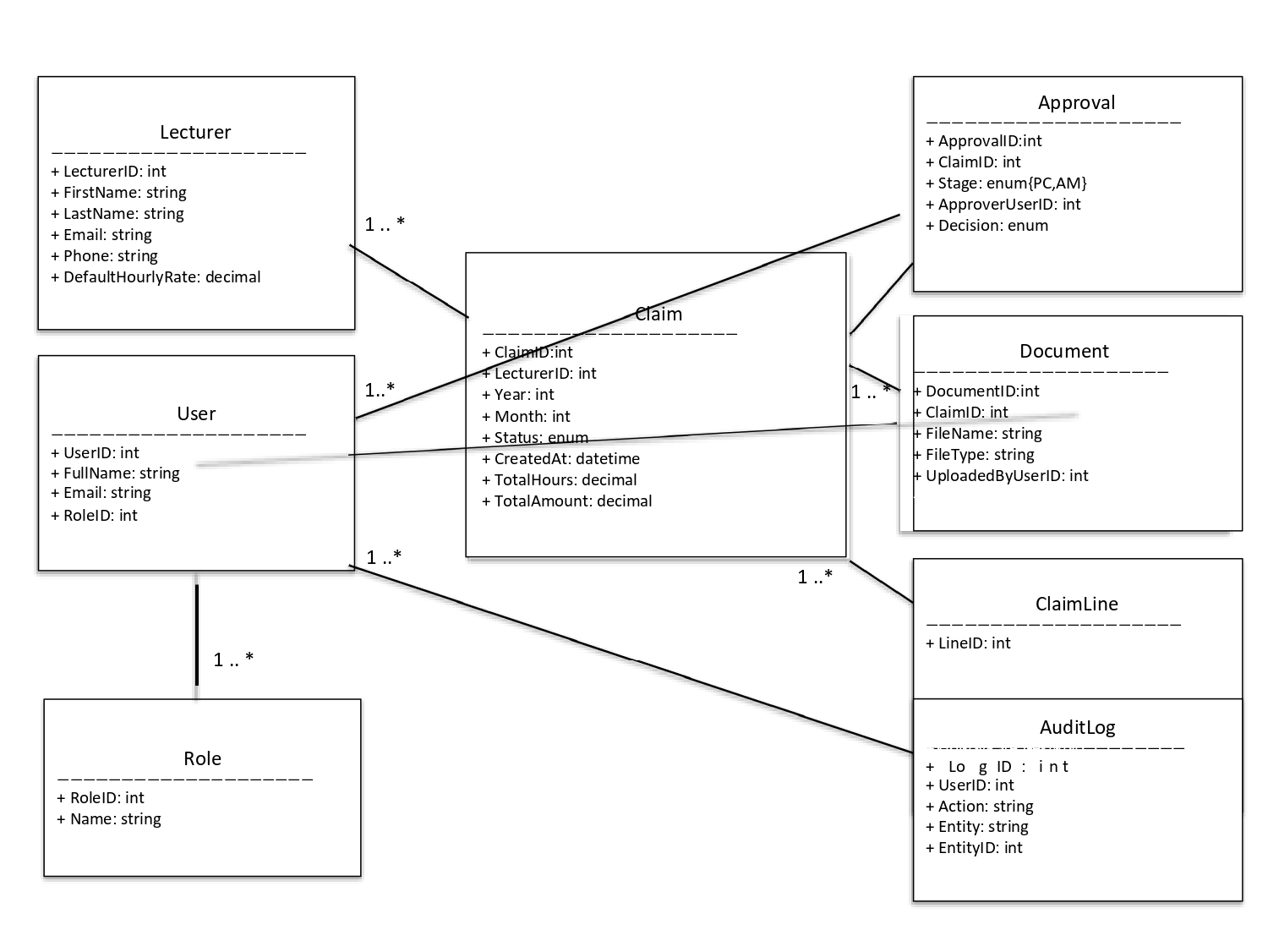


Figure 1: UML class diagram focused on database‑ready classes and relationships.

**Role → User**: one role can be assigned to many users; each user has **one** role. *(1 .. \*)*

**Lecturer → Claim**: one lecturer can create many monthly claims. *(1 ..\*)*

**Claim → ClaimLine**: a claim “owns” **many** line items (*1..\*)*

**Claim → Document**: a claim can have **many** supporting files. *(1..\*)*

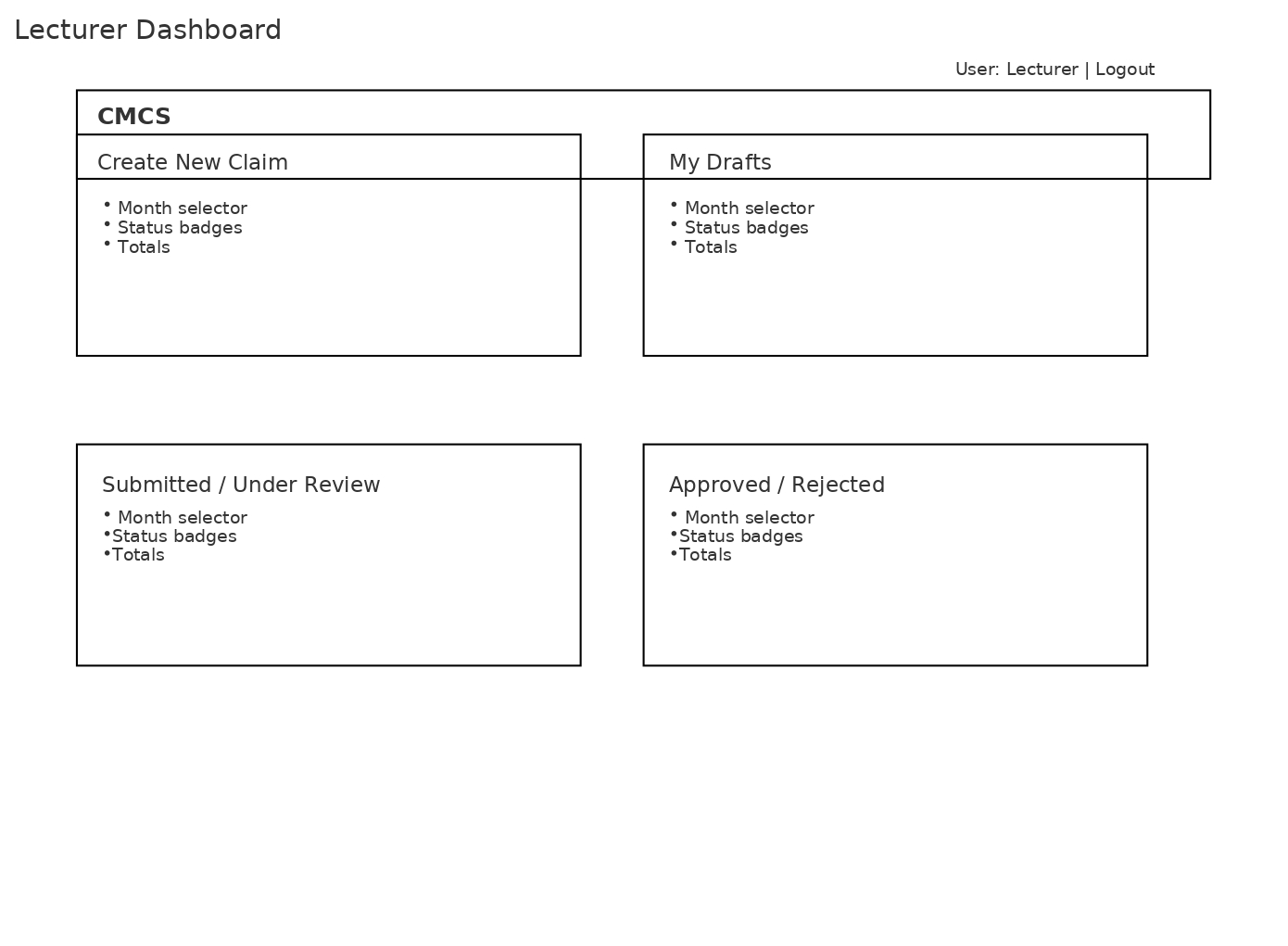
**Claim → Approval**: a claim gathers **multiple** approval records *(1..\*)*

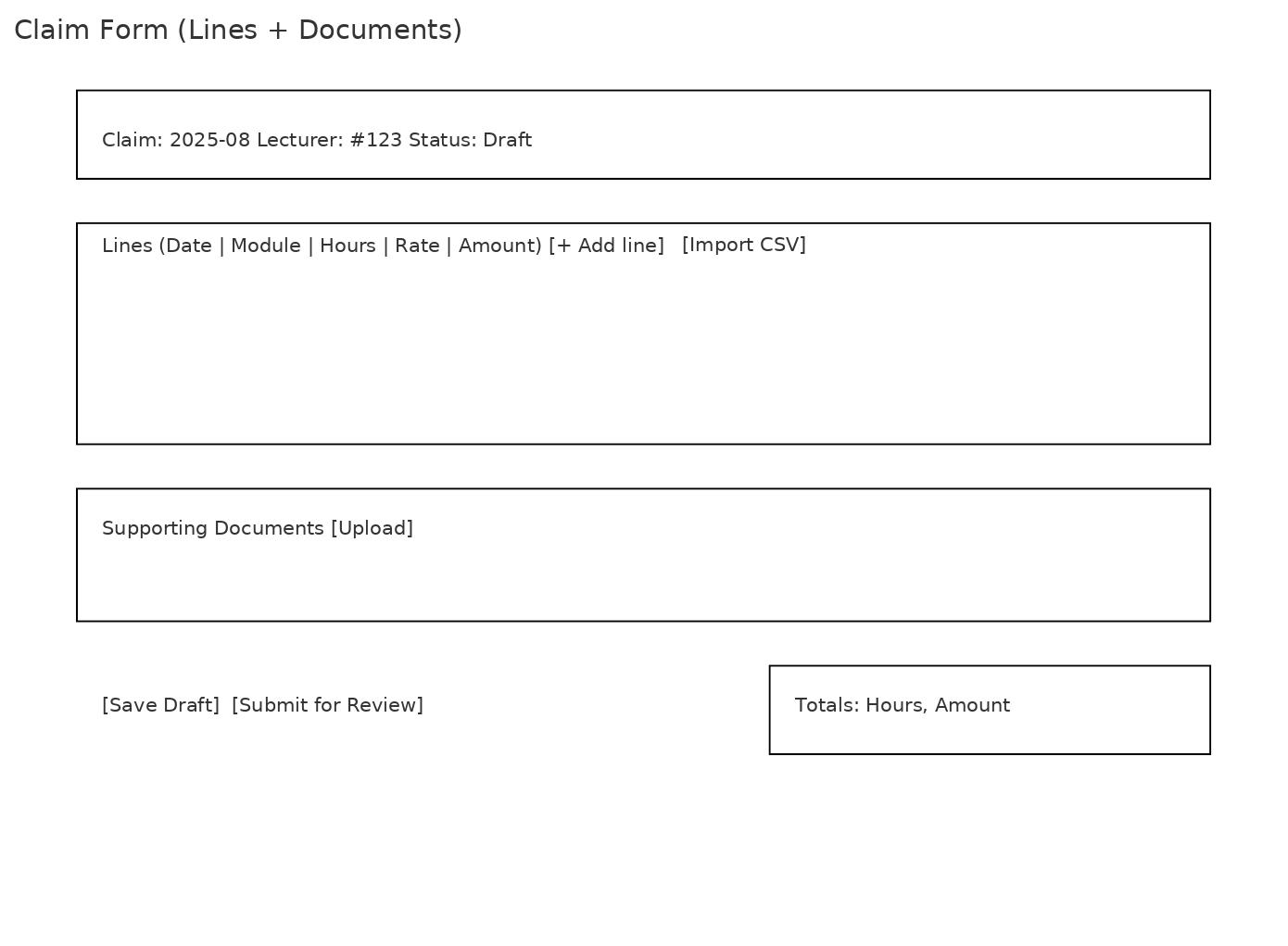
**User → Document**: a user can **upload many** documents *(1 ..\*)*

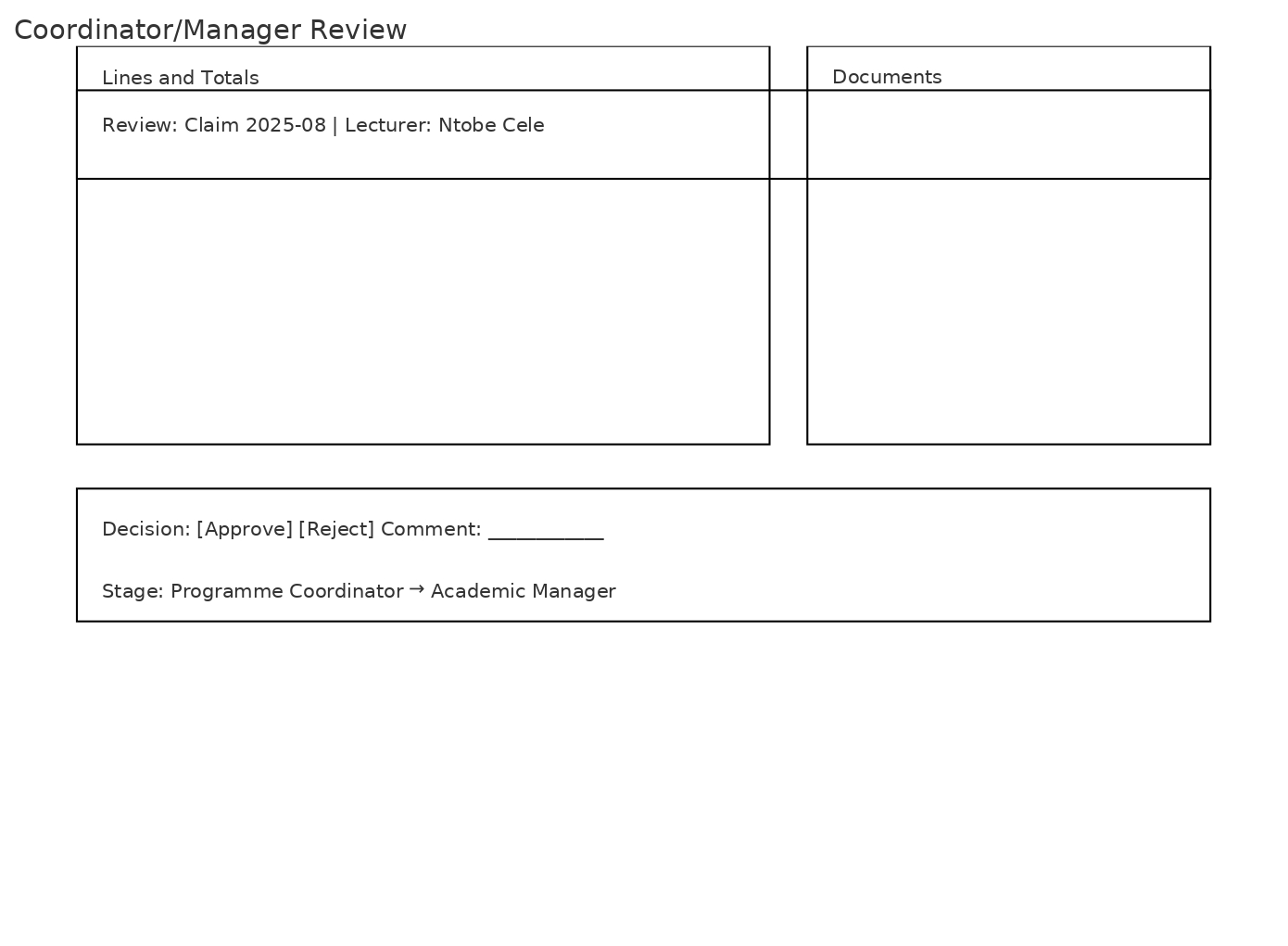
**User → Approval**: a user (PC/AM) can **approve/reject many** claims over time. *(1 ..\*)*

**User → AuditLog**: a user can generate **many** audit entries *(1 .. \*)*

**GUI/UX Overview.** Screen 1 (Lecturer Dashboard) surfaces month selection, draft/submitted/approved tiles and a clear call to “Create Claim”. Screen 2 (Claim Form) presents an editable grid for lines with inline totals and a document uploader; actions are Save Draft and Submit for Review. Screen 3 (Review) displays the claim alongside documents; reviewers record an Approve/Reject decision with a comment.

****

****

****

**Project Plan & Version Control.** I propose a 2-week project plan: weeks 1–2 requirements and UML; weeks 3–5 wireframes and repo scaffolding with at least five commits; weeks 6–8 prototype GUI and report; weeks 9–10 review and submission. Commit messages should be clear (e.g., “feat(ui): add claim line grid” / “chore: add approval model skeleton”).

