# Documentation — Design choices, database structure & GUI layout

The Contract Monthly Claim prototype is designed as a focused Model–View–Controller (MVC) web user interface that models the monthly claim lifecycle for contract lecturers. The process begins with claim submission and may include optional document attachments, followed by verification by Programme Coordinators, and finally approval and settlement by Academic Managers. At this stage, the prototype is deliberately non-functional, allowing the design to focus on data model correctness, user journey clarity, and extensible architecture. These elements ensure that the prototype can later be connected to services and persistence layers without major restructuring (GeeksforGeeks, 2025a).

### Key Design Choices and Rationale

A core design principle is the separation of concerns achieved through MVC architecture and a supporting service layer. In this model, controllers are responsible for routing and selecting views, while views render the user interface. Meanwhile, a small service layer (IClaimService/ClaimService) encapsulates data access, keeping the UI code testable and simplifying the transition between storage implementations, whether in-memory, Entity Framework Core, or eventually a cloud database (GeeksforGeeks, 2025b).

The data model has been normalized and modularized. Key entities—Claims, ClaimDocuments, and Approvals—are separated to avoid data duplication and to support future scalability, such as multiple attachments or multi-stage approval workflows. This approach aligns with database design best practices (GeeksforGeeks, 2025a).

Role-awareness is another significant design choice. The system layout supports role selection (Lecturer, Coordinator, Manager), allowing the later introduction of role-based authorization and differentiated queues or views per role, without altering the core model.

For the prototype phase, prototype-friendly data types were used. The “Month” field was represented as either a string or DateTime depending on the scenario, and file uploads were disabled. However, metadata fields for documents were included to prepare for future implementation.

### Database Structure

The database schema has been structured following UML-style conventions.

* Claim:
  + Attributes: Id (PK), LecturerName, Programme, Month, Amount, Notes, Status, SubmittedAt, CreatedBy.
  + Relationships: One-to-many with both ClaimDocument and Approval.
* ClaimDocument:
  + Attributes: DocumentId (PK), ClaimId (FK), Filename, MimeType, UploadedAt.
  + Purpose: Metadata storage only in the prototype, with binary storage delegated to a file system or blob store at a later stage.
* Approval:
  + Attributes: ApprovalId (PK), ClaimId (FK), UserId (FK), Decision, DecisionAt, Comments.
  + Purpose: Provides traceability of decisions in the verification and approval lifecycle.
* User/Role (design placeholder):
  + Attributes: UserId (PK), FullName, Role.
  + Purpose: To support authentication and authorization in future iterations.

This schema is designed for traceability, multiple attachments, and extensibility in workflow management, ensuring scalability as functionality grows (Gurnov, 2025).

### Graphical User Interface and Layout

The global layout is implemented in Index.cshtml, which uses a top navigation bar with the application title, a role selector, and primary navigation links (Dashboard, Submit, Approver Queue). Bootstrap was selected for quick, responsive styling (W3C, n.d.).

* Claims Dashboard (Index View): Displays claims in a compact table view with sortable-looking rows, including fields such as Lecturer Name, Programme, Month, Amount, Notes, and Status. A prominent “Submit Claim” button allows users to quickly access the claim submission form.
* Submit Form: Provides input fields for Lecturer, Programme, Month, Amount, and Notes. A file input placeholder exists but is disabled in this prototype. Submissions trigger a temporary “toast” message for visual confirmation.
* Approver Queue and Claim Detail (planned): Intended to provide action buttons (Verify/Approve) for coordinators and managers, along with a detailed timeline and attachments view.

Accessibility considerations were integrated into the prototype, with clear labels, confirmation prompts on delete actions, and status badges for rapid scanning (W3C, n.d.).

### Assumptions and Constraints

Several assumptions guided the prototype design. First, the prototype is non-functional, meaning that authentication, file storage, and payment settlement are not implemented in this phase. Instead, seeded mock data is used to populate the interface, enabling evaluators to assess user flows.

Furthermore, aspects of security, persistence, and concurrency have been deferred to later iterations, though the models already include relevant fields to facilitate secure implementation. Finally, ASP.NET Core MVC was chosen for its familiarity and efficiency in prototyping, while Entity Framework Core is planned for persistence in subsequent stages (LinkedIn, 2025).