# Contract Monthly Claim System

# Design Documentation (Prototype / Non-Functional) Summary.

**1) Goals, Scope & Audience**

Goal: Provide a visual prototype of the Contract Monthly Claim System focusing on claims

submission, verification, approval, and transparency.

Out of Scope: Database writes, authentication, notifications, payment integration.

**2) Users & Roles**

Lecturer: Submits claims.

Programme Coordinator: Verifies claims.

Academic Manager: Approves claims.

Finance (future): Records payments.

Admin (future): Manages users/contracts.

**3) High-Level Use Cases**

Submit Claim, My Claims, Review Claims, Claim Details. All screens are non-functional prototypes.

**4) Data Model — UML Overview**

Entities: User, Role, Contractor, Contract, Claim, ClaimLineItem, Attachment, Approval, Payment,

AuditLog.

Relationships: Contractor 1-\* Contract, Contract 1-\* Claim, Claim 1-\*

LineItem/Attachment/Approval, Claim 0.1 Payment.

**5) GUI Layout**

Navigation: Home/Dashboard, Submit Claim, My Claims, Review Claims, Claim Details.

Forms are disabled. Mock data is used consistently.

Accessibility: ARIA labels, colour contrast, focus indicators.

**6) Technology Choices**

.NET 8 MVC with Razor (primary), Bootstrap 5 for layout, PlantUML for diagrams. Optional WPF

XAML prototype.

**7) Assumptions & Constraints**

Claims are monthly per contract, VAT varies by contract, attachments support limited formats.

Prototype constraint: Mock only, no backend.

**Assumptions**

**User Roles and Responsibilities**

* Lecturers are responsible for preparing and submitting claims, including supporting documents (e.g., timesheets, receipts).
* Programme Coordinators are the first level of verification, checking completeness and accuracy.
* Academic Managers provide final academic approval, ensuring the claim complies with contractual and institutional requirements.
* Finance staff (future phase) handle payment processing after approvals.
* Admin manages contracts, users, and role assignments.
* Claim Process Workflow
* Each claim corresponds to one calendar month and a single contract.
* A contract cannot have multiple active claims for the same period (uniqueness enforced).
* Workflow states are sequential: Draft → Submitted → Under Review → Approved/Rejected → Paid (if approved).
* Claims must include at least one-line item, with VAT calculated automatically.
* Data Integrity and Auditability
* Every change in claim status generates an audit log entry.
* Attachments are mandatory for final submission (e.g., signed attendance or service reports).

* System maintains historical claims and approvals without overwriting past records.
* System Behaviour
* The system provides real-time claim status visibility to lecturers.
* Users may only see data relevant to their role (role-based access control in the functional version).
* Technical Environment
* Prototype GUI is implemented in .NET Core MVC (or WPF) but is non-functional.
* Database schema design is forward-looking, designed for later use with Entity Framework Core.
* Web version assumes access via standard browsers; WPF version assumes Windows desktop environment.

**Constraints**

* Prototype Constraints (Current Stage)
* GUI is front-end only; no persistence, authentication, or workflow logic.
* Buttons and upload fields are disabled, serving as placeholders.
* Mock data is used for consistency (claims, contracts, users).
* Security, performance, and integrations are out of scope for the prototype.
* Functional Constraints (Future Implementation)
* Data Privacy: Personal and financial data must comply with GDPR/POPIA (depending on jurisdiction).
* Integration: Finance system integration may be required in later phases (e.g., SAP, Oracle).
* Technical Constraints
* Database normalization to 3NF to reduce redundancy.
* Unique constraint on (ContractId, Period) to prevent duplicate monthly claims.
* Application must be scalable, able to transition from prototype to enterprise without redesigning schema.
* Offline support (for WPF desktop clients) is optional but considered a future requirement.

# UML Class Diagram for databases: Contract Monthly Claim System

Claim

Id Guid

Contractid Guid

Period: string

CreatedAt: Datefime

Subtotal: decimal

Vat: decimal

Total: decimal

Status: string

Draft: Submitfitted

UnderReview, Approved: Poid Rejected: string

Contract

Id Guid

ContraciSNo

Title

StartDate: Date Time

EndDate: DateTime

Notes: string?

ClaimLineltem

Id Guid

Claimid Guid

FileName: stirring

ContentType

StorageTath

Role

Id Guid

Name: string

Approval

Id Guid

Claimid Guid

ApproverUserGuid

ActionedAt: Datefime

Decision

Gumment \* string

Contractor

Id Guid

Name: string

Contract

Registration No, string

ContactEmail string

ContactPhone string

IsActive: bool

User

Id Guid

Username: string

Email: string

FullName: string

IsActive: bool

CreatedAt: Datefime

\*

1

0

Payment

Id Guid

Claimid Guid

PaymentRef strig

Payment Date: Datfime

Amount: dec

Action

Created: Updated

Deleted. StatusChanged

Detail: string

Pamen

Id Guid

Claimid Guid

PaymentRef

string

Approval

Id Guid

Claimid Guid

ApproverUserind

Entify/d Guid

# Project Plan: Contract Monthly Claim System (Prototype)

**Phase 1: Project Initiation (Week 1)**

• Define project scope and objectives

• Identify stakeholders (lecturers, coordinators, managers, finance, admin)

• Gather high-level requirements (interviews, documentation review)

• Develop initial project charter

Dependencies: None

Duration: 1 week

**Phase 2: Requirements & Analysis (Week 2–3)**

• Collect detailed requirements (functional + non-functional)

• Map user roles and workflows (claim submission → approval → payment)

• Define system constraints and assumptions (monthly uniqueness, VAT handling)

• Draft UML class diagram (database model)

• Validate requirements with stakeholders

Dependencies: Completion of initiation phase

Duration: 2 weeks

**Phase 3: System Design (Week 4–5)**

• Finalize UML class diagram for database structure

• Define ERD-to-table mapping (keys, constraints, indexes)

• Design GUI wireframes (Lecturer, Coordinator, Manager, Admin views)

• Prototype navigation flow (non-functional)

• Select technology stack (.NET Core MVC/WPF, Bootstrap for GUI)

Dependencies: Requirements sign-off

Duration: 2 weeks

**Phase 4: Prototype Development (Week 6–7)**

• Build non-functional front-end prototype in .NET Core MVC (or WPF)

• Create static pages (Submit Claim, My Claims, Review Claims, Claim Details)

• Integrate mock data for consistency across views

• Ensure accessibility features (labels, disabled buttons, ARIA support)

• Test navigation flow for correctness

Dependencies: Completion of design

Duration: 2 weeks

**Phase 5: Documentation (Week 8)**

• Compile design documentation (database + GUI layout + rationale)

• Document assumptions and constraints clearly

• Add mock data examples and UML diagrams to report

• Prepare submission package (PDF documentation, UML, Gantt chart, prototype UI files)

Dependencies: Prototype development completion

Duration: 1 week

**Timeline (Gantt Summary)**

|  |  |  |
| --- | --- | --- |
| Phase weeks | Milestone | Tasks |
| Week 1 | Initiation | Initiation, Requirements gathering, stakeholder alignment |
| Week 2-3 | Analysis and design | Analysis & Design, UML class diagram, data dictionary, wireframes |
| Week 4-5 | Prototype setup | Prototype Setup, Project setup, navigation layout, Claims scaffold |
| Week 6-7 | Prototype Expansion | Prototype Expansion, Contracts & Admin modules, UI polish |
| Week 8 | Review and closure | Review & Closure, Internal review, stakeholder demo, docs & handover |

**Key Realism Factors**

• Tasks broken into manageable, clear deliverables

• Dependencies reflect logical sequencing (no unrealistic overlaps)

• Timeline allows buffer for feedback and adjustments

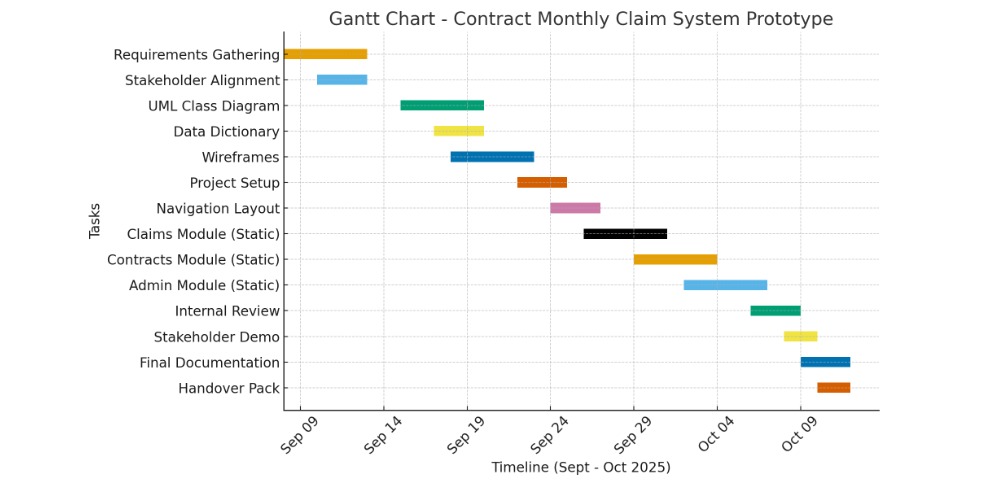
• Final output = Prototype + Documentation + UML + Gantt chart

**Roles & Responsibilities**

* Project Manager – oversees schedule, dependencies, stakeholder commas.
* Business Analyst – requirements gathering, UML/data model validation.
* UI/UX Designer – wireframes, layouts, accessibility.
* Developers – prototype build (MVC/WPF scaffolding).
* QA/Tester – walkthroughs, ensure usability & consistency.

**6. Risks & Mitigations**

* Scope creep → lock scope early, MoSCoW prioritisation.
* Timeline delays → weekly progress reviews, buffer in Week 5.
* Unclear workflows → stakeholder demos at Week 2 (wireframes) and Week 5 (prototype).



# GUI User interface: Design and User-Friendliness: Contract Monthly Claim System — GUI Design (Front-End Prototype, Non-Functional)

**Technology:** .NET Core MVC (or WPF) • Scope: Visual-only prototype • Audience: Lecturers, Programme Coordinators, Academic Managers

**Purpose**

Deliver a highly user-friendly and intuitive user interface that visually demonstrates the monthly claim workflow without any backend functionality.

**Prototype Constraint:** All actions are disabled. Forms, buttons and uploads are placeholders. Mock data is shown consistently to illustrate states.

**Key Options Demonstrated**

* Lecturers can submit claims (button exposed at all times).
* Programme Coordinators and Academic Managers can verify and approve claims (queue view + disabled actions).
* Lecturers can upload supporting documents (file input placeholder).
* Transparent status tracking from Draft → Submitted → Under Review → Approved/Rejected → Paid (badges).

**Navigation Map**

|  |  |
| --- | --- |
| Global Navigation | Home/Dashboard • Submit Claim • My Claims • Review Claims |
| Primary Flow | Home → Submit Claim (Lecturer) → My Claims → Claim Details → (Coordinator/Manager) Review Claims |
| Context Links | “View” on My Claims opens read-only Claim Details with tabs for Items, Attachments, Approvals |

**Screen Blueprints (Wireframe Specifications)**

**1) Home / Dashboard**

Three cards: Submit Claim, My Claims, Review Claims. Clear call-to-action buttons. A banner note stating “Prototype- Non-functional”.

**2) Submit Claim (Lecturer)**

* Form fields (disabled): Contract No, Period (YYYY-MM), Notes. Grid for Line Items (Description, Qty, Unit, Rate,
* Line Total). Totals section (Subtotal, VAT, Total). File upload control (placeholder). Buttons: Save Draft, Submit for Review (disabled).

**3) My Claims (Lecturer)**

* Tabular list with columns: Contract, Period, Subtotal, VAT, Total, Status, Actions. Status shown via color-coded badges. “View” opens read-only Claim Details.

**4) Claim Details (Read-only)**

* Header summary (Contract, Period, Status, Total). Tabs: Line Items, Attachments, Approvals. All content is static; used to communicate transparency.

**5) Review Claims (Programme Coordinator / Academic Manager)**

* Queue table with Lecturer, Contract, Period, Total, Status, Actions. Approve/Reject buttons are disabled for prototype, showing intended placement

**Accessibility & Usability**

1 Keyboard-friendly navigation and focus indicators.

2 Clear hierarchy and spacing (cards, tables, tabs).

3 High-contrast badges and labels; meaningful iconography (optional).

4 ARIA-friendly labels (if MVC web) / AccessibleName (if WPF).

5 Disabled states to signal non-functional prototype constraints.

**Sample User Flows (Prototype)**

|  |  |
| --- | --- |
| Role / Flow | Steps |
| Lecturer — Submit | Home → Submit Claim → Fill (disabled) → Attach file (placeholder) → Submit (disabled) → My Claims shows “Submitted”. |
| Lecturer — Track | Home → My Claims → View → Claim Details shows tabs and approval history (static). |
| Coordinator — Review | Home → Review Claims → Approve/Reject buttons shown but disabled, communicates intended actions. |

**Design Rationale**

The layout prioritizes clarity and speed: a three-card dashboard to reduce cognitive load, clean tables for tracking, and obvious tabs for drill-down. Status badges provide immediate feedback on progress. Disabled controls preserve, the non-functional scope while indicating future interactivity.

**Non-functional snippet**

**Submit Claim (Lecturers)**

@\* Views/Claims/Create.cshtml \*@

<h2>Submit Claim</h2>

<form>

<fieldset disabled>

<div class="mb-3">

<label class="form-label">Contract No</label>

<input class="form-control" placeholder="e.g., CN-001" />

</div>

<div class="mb-3">

<label class="form-label">Period</label>

<input class="form-control" placeholder="YYYY-MM" />

</div>

<div class="mb-3">

<label class="form-label">Supporting Documents</label>

<input type="file" class="form-control" />

</div>

<button class="btn btn-primary" type="button" disabled>Submit Claim</button>

</fieldset>

</form>

**My Claims (Lecturers)**

@\* Views/Claims/Index.cshtml \*@

<h2>My Claims</h2>

<table class="table table-striped">

<thead><tr><th>Contract</th><th>Period</th><th>Total</th><th>Status</th></tr></thead>

<tbody>

<tr><td>CN-001</td><td>2025-08</td><td>R1,250</td><td><span class="badge bg-warning">Submitted</span></td></tr>

<tr><td>CN-002</td><td>2025-07</td><td>R800</td><td><span class="badge bg-success">Approved</span></td></tr>

</tbody>

</table>

# Version Control: push source code and Documentation link

<https://github.com/YourUsername/Contract-Monthly-Claim-System>

