PROG6212 PART 1

# Contract Monthly Claim System (CMCS)

The Contract Monthly Claim System is a .NET web-based application designed to streamline monthly claim submissions and approvals for independent contractor lecturers.

**Key User Interfaces Needed:**

**For Lecturers:**

* Login/Dashboard page
* Claim submission form
* Document upload interface
* Claim status tracking page
* Historical claims view

**For Programme Coordinators:**

* Claims review dashboard
* Claim details view with approval options
* Bulk approval interface
* Reporting dashboard

**For Academic Managers:**

* Final approval dashboard
* System reports and analytics
* User management interface
* System configuration

**Design Principles:**

* Intuitive navigation
* Role-based access control
* Clear visual hierarchy
* Responsive design
* Accessibility compliance

### **5. Version Control Strategy**

**Commit Structure (5 minimum commits):**

1. Initial project setup and structure
2. UML class diagram and database design
3. Basic GUI framework and layout
4. User interface components and styling
5. Documentation and final prototype polish

**PROG6212 – Part 1**

**Contract Monthly Claim System (CMCS) – Prototype Report**

### **1. Documentation**

The Contract Monthly Claim System (CMCS) is a .NET Core WPF prototype designed to streamline claim submissions for Independent Contractor (IC) lecturers and manage their approval workflow. The system routes each claim through two review stages where verification by the Programme Coordinator and final approval by the Academic Manager. The design prioritizes efficiency, accuracy, and transparency. At this stage, the prototype is non-functional and focuses only on structure, user flows, and interface layout.

**Technology Choice:** WPF was selected over MVC for Part 1 because it is suited for desktop-based prototypes, offers strong XAML layout support, and makes it easier to build role-specific interfaces without back-end complexity. MVC may be considered in later iterations if a web deployment is required.

**Database Design:** The schema centers on **Lecturers, Claims, Documents, Approvals, and Users** (for Coordinators and Managers). Each claim is tied to a lecturer, may include multiple documents, and must progress through structured approval steps. Primary keys ensure unique identification of entities, while foreign keys define relationships. Normalization ensures no redundant data (e.g., lecturer details stored once in Lecturer).

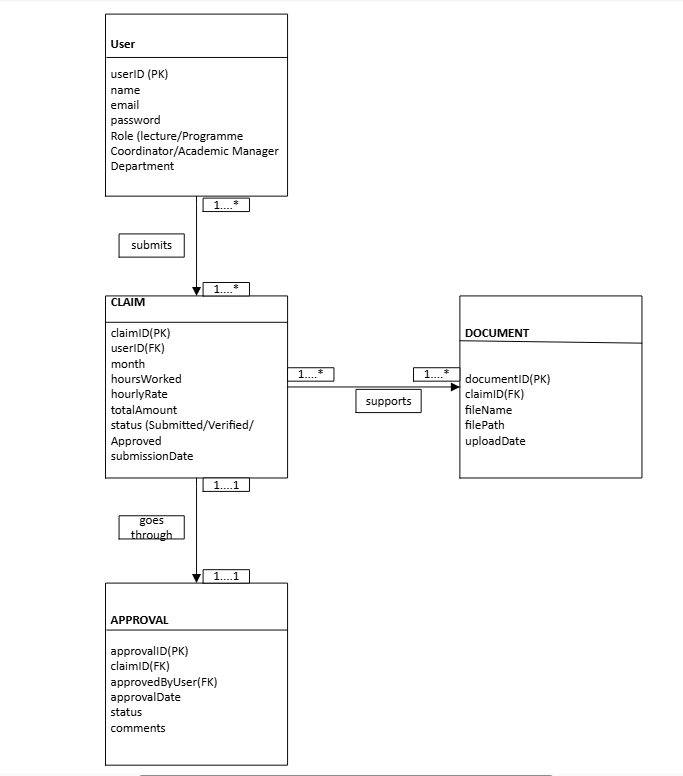
**UI/UX Principles:** Layouts were designed for clarity and accessibility. Role-based dashboards (Lecturer, Coordinator, Manager) minimize clutter and show only relevant functions. Navigation is straightforward with a consistent design language across pages. Visual hierarchy is achieved through clear sectioning, consistent typography, and buttons styled for recognizable actions. Accessibility is considered with readable labels, sufficient contrast, and predictable layouts.

### **Assumptions & Constraints**

* Lecturers are paid hourly, with variable rates depending on their contract.
* A claim is monthly and linked to one lecturer only.
* Supporting documents (PDF, Word, Excel) are uploaded and linked to a claim.
* The POE uses WPF for GUI
* Approvals are strictly ordered: Programme Coordinator then Academic Manager.
* Authentication and back-end logic are excluded for Part 1.
* Constraint: Prototype must not include working data or logic (no functional logic, database binding, or file uploads exist yet).

## 2. UML CLASS DIAGRAM FOR DATABASES

* Claim: ClaimID, LecturerID, ClaimPeriod, TotalHours, HourlyRate, TotalAmount, Status, Description
* Document: DocumentID, ClaimID, FileName, FilePath, UploadDate, DocumentType
* Approval: ApprovalID, ClaimID, ApproverID, Date, Status, Comments, ApprovalLevel
* User: UserID, Name, Email, Role (Coordinator/Manager), Department



Key relationships:

* Lecturer (1) → (M) Claim
* Claim (1) → (M) Document
* Claim (1) → (M) Approval
* User (1) → (M) Approval

### **3.Project Plan (Timeline & Dependencies)**

* **Week 1** – Requirements analysis, personas, UML draft, wireframes.
* **Week 2** – WPF solution setup, main windows, basic navigation.
* **Week 3** – Build prototype screens Submit Claim, Review, Approval, Status.).
* **Week 4** – Style refinement, polish layouts, final UML.
* **Week 5** – Documentation, screenshots, GitHub cleanup, submission.

Dependencies: Wireframes → UML → WPF shell → Screens → Styling → Docs.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Task** | **Dependencies** | **Week** | **Timeline (Days)** | **Responsible** | **Deliverables** |
| Requirement Gathering | None | 1 | 3 | Analyst | Requirement document |
| System Analysis | Requirement Gathering | 1 | 4 | Analyst | Problem statement & objectives |
| UML & Database Design | System Analysis | 2 | 3 | Analyst/DB Designer | UML class diagram, database schema |
| GUI Wireframes | UML & Database Design | 2 | 2 | UI/UX Designer | Wireframes |
| Prototype Setup | GUI Wireframes | 3 | 3 | Developer | Initial WPF project with navigation |
| Prototype Screens | Prototype Setup | 3–4 | 4 | Developer | Prototype GUI screens |
| Documentation & Report | All previous tasks | 5 | 4 | Analyst/Developer | Final Part 1 report (400–500 words) |
| Version Control | Ongoing across all tasks | 1–5 | Continuous | Developer | GitHub repository (≥5 commits) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Phase** | **Task** | **Timeline** | **Dependencies** | **Responsible** | **Deliverables** |
| **Phase 1: Planning & Design** | Requirements Gathering | 3 days (Week 1) | None | Analyst | Requirement document |
|  | UML Diagram Creation | 3 days (Week 1–2) | Requirements Gathering | Analyst | UML diagrams |
|  | UI/UX Wireframes | 2 days (Week 2) | Requirements Gathering | Analyst / Designer | Wireframes |
|  | Database Schema Design | 3 days (Week 2) | UML & Wireframes | Analyst / DB Designer | Database schema |
| **Phase 2: Prototype Development** | GUI Framework Setup | 3 days (Week 3) | Database schema | Developer | GUI skeleton |
|  | Static UI Components | 2 days (Week 3) | GUI Framework | Developer | UI components |
|  | Navigation Structure | 1 day (Week 3–4) | UI Components | Developer | Navigation menus |
|  | Basic Styling | 1 day (Week 4) | Navigation | Developer / Designer | Styled prototype |
| **Phase 3: Documentation** | Technical Documentation | 2 days (Week 5) | All development tasks | Analyst | GitHub repository (≥5 commits) |
|  | User Guides | 1 day (Week 5) | Prototype | Analyst / Team | User guide |
|  | Testing Documentation | 1 day (Week 5) | Prototype | Tester / Analyst | Test cases & results |

*Dependencies:* UML before GUI design; database schema impacts prototype structure; version control established early.



### **4. GUI/WPF Layout (Prototype Only)**

* **Login (Mock)** – entry point with role selection and buttons/links to role dashboards.
* **Lecturer Dashboard** – “New Claim”, “My Claims”, “Upload Documents”.
* **Submit Claim Page** – Month selector, Hours worked, Hourly rate, line items (Date, Hours, Description), total preview, Upload area (disabled for Part 1).
* **Coordinator Dashboard** – List of submitted claims, claim details, approve/reject buttons (disabled).
* **Manager Approval** – Claims pending final approval, approve/reject interface, reports placeholder.
* **Claim Status Page** – Claim history with status badges (Draft, Submitted, Approved, Rejected).

**GUI Layout (non-functional prototype)**

### **Version Control Strategy**

GitHub repository with minimum five commits:

1. Initial project setup (solution + repo).
2. UML diagram + database schema draft.
3. Base GUI layout (Login, dashboards).
4. Prototype screens + styling.
5. Final documentation and cleanup.

<https://github.com/ST10451618/prog6212part1.git>

## **Success Criteria**

**Documentation:** Clear, concise 400-500 word report

**UML Diagram:** Complete class diagram with all relationships

**Project Plan:** Realistic timeline with clear dependencies

**GUI:** Non-functional but complete visual prototype

**Version Control:** Minimum 5 meaningful commits with clear messages