Tshiamo Mbatha

ST10070515@RCCONNECT.EDU.ZA

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.   
When you’re ready to add your content, just click here and start typing.]

PROJECT PLANNING & PROTOTYPE DEVELOPMENT

PART1 DOCUMENTATION

Table of Contents

[Documentation 1](#_Toc207666658)

[Introduction 1](#_Toc207666659)

[Database Structure 1](#_Toc207666660)

[GUI Layout 2](#_Toc207666661)

[Assumptions & Constraints 2](#_Toc207666662)

[UML CLASS DIAGRAM FOR DATABASE 3](#_Toc207666663)

[PROJECT PLAN 3](#_Toc207666664)

[Project Objectives 3](#_Toc207666665)

[Scope 3](#_Toc207666666)

[Deliverables 4](#_Toc207666667)

[Timeline/Schedule 4](#_Toc207666668)

[Resources 4](#_Toc207666669)

[Budget 4](#_Toc207666670)

[Roles and Responsibilities 5](#_Toc207666671)

[Risk Management 5](#_Toc207666672)

[Communication Plan 5](#_Toc207666673)

[Quality Management 5](#_Toc207666674)

[Stakeholder Management 5](#_Toc207666675)

[Assumptions and Constraints 5](#_Toc207666676)

[References 6](#_Toc207666677)

# Documentation

## Introduction

The Contract Monthly Claim System (CMCS) is a GUI application enabling independent contractors to submit claims, with approvals managed by program coordinators and academic managers based on authorization and valid documentation. ICs can track statuses, while the system mitigates administrative errors, enhancing efficiency and user satisfaction. This non-functional prototype, developed using .NET Core, establishes a foundation for future functionality. WPF is preferred over MVC for its rich, responsive desktop UI, leveraging GPU acceleration, data binding, offline capabilities, and seamless Windows integration, which support role-specific dashboards for ICs, coordinators, and managers (Stack Overflow, 2010; Quora, 2023; iFour Technolab, 2024).

## Database Structure

The UML diagram includes five entities: IC, CLAIM, DOCUMENT, COORDINATOR, and MANAGER, adhering to relational database principles. Attributes are:

* IC: IC\_ID (primary key), FirstName, LastName, Email, Number.
* CLAIM: Claim\_ID (primary key), HoursWorked, hourly-rate, status, IC\_ID (foreign key).
* DOCUMENT: Doc\_ID (primary key), Claim\_ID (foreign key), FileName, FileType, UploadDate.
* COORDINATOR: COORDINATOR\_ID (primary key), FirstName, LastName, Email,
* MANAGER: MANAGER\_ID (primary key), FirstName, LastName, Email,

Cardinalities are: IC to CLAIM (1:\*), COORDINATOR to CLAIM (1:\*), MANAGER to CLAIM (1:\*), CLAIM to DOCUMENT (1:\*).

This structure ensures data integrity via primary and foreign keys, mitigating inaccuracies by enforcing entity and referential integrity (Acceldata, 2025; Agile Data, n.d.; Tennessee, n.d.). One-to-many relationships support scalability, allowing multiple documents per claim and multi-level approvals, enhancing reliability. UploadDate tracks workflow stages, ensuring accountability, while HoursWorked and hourly-rate enable accurate claim calculations, reducing financial errors.

## GUI Layout

The GUI requires login, directing ICs to a window for status tracking, claim creation, and profiles, while coordinators and managers access dashboards for review, calculations, and approvals/rejections. It features textboxes, labels, buttons, navigation bars, data grid views, and progress bars. User flow starts at login, leading to role-specific homepages with navigation for submissions and tracking. Using WPF, the design separates UI from logic, supporting future functionality. Accessibility is ensured with high-contrast colors (blue on white), 12pt Arial fonts, and keyboard navigation, aligning with WCAG standards, enhancing user satisfaction within the non-functional scope.

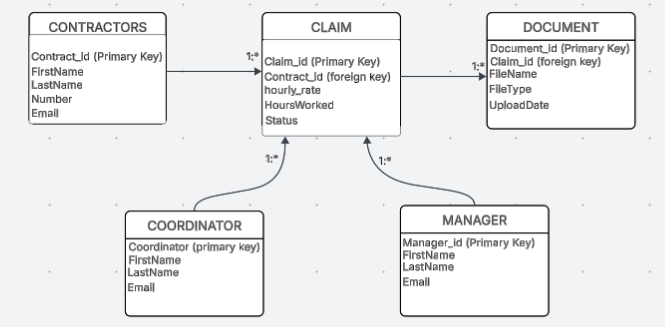
## Assumptions & Constraints

**Assumptions:** Basic user literacy guides UI design with top menus, reducing training needs within the 45-hour limit, though novices may struggle. Stable local database access supports visualization, but linkage failure risks data isolation.

**Constraints:** The 45-hour timeframe limits scope to non-functional prototyping, excluding coding or real-time notifications, focusing on core workflows. Resource constraints (free tools) may limit diagram complexity.

# UML CLASS DIAGRAM FOR DATABASE

The UML diagram visually represents five entities (IC, CLAIM, DOCUMENT, COORDINATOR, MANAGER) with attributes and 1:\* relationships. IC links to CLAIM via IC\_ID, CLAIM to DOCUMENT via Claim\_ID, and COORDINATOR/MANAGER to CLAIM via their IDs.



# PROJECT PLAN

## Project Objectives

Design a non-functional CMCS GUI prototype for claim submission, approval, and tracking, enhancing efficiency and satisfaction.

## Scope

Includes UML diagram, project plan, WPF GUI layout, and documentation; excludes functional code, APIs, and real-time features.

## Deliverables

- 400-500 word Word document.

- UML class diagram.

- GUI wireframes.

- Five GitHub commits.

## Timeline/Schedule

| **Day** | **Tasks** | **Dependencies** | **Estimated Time Allocation** |
| --- | --- | --- | --- |
| 1 | • Review POE Part 1 requirements and background information.  • Brainstorm design choices for database structure and GUI layout.  • Begin drafting documentation: Explain initial design choices, database structure, and assumptions/constraints.  • Set up GitHub repository and make initial commit | None | 6-8 hours |
| 2 | • Design UML Class Diagram: Identify classes, attributes, relationships, and database representation.  • Continue documentation: Integrate UML explanation and refine rationale for design decisions.  • Commit UML diagram draft to GitHub. | Day 1 brainstorming | 6-8 hours |
| 3 | •Start GUI/UI design using WPF (.NET Core): Create non-functional prototype layouts for lecturer claim submission, approval views, document upload, and status tracking.  • Update documentation with GUI layout details. - Commit initial GUI prototype code to GitHub. | UML diagram for data alignment | 6-8 hours |
| 4 | •Refine GUI/UI design: Ensure user-friendly elements  • Develop project plan: Outline tasks, dependencies, and overall timeline for prototype | GUI start from Day 3 | 6-8 hours |
| 5 | •Finalize documentation: Ensure detailed explanation of all in a 400-500 word report.  •Integrate project plan into the report.  •Commit finalized documentation draft to GitHub. | All prior components | 6-8 hours |
| 6 | •Review and polish all elements: UML diagram, GUI prototype, project plan, and documentation for coherence and completeness.  •Make additional commits for any refinements  • Format report as Microsoft Word document. | All prior days | 4-6 hours |
| 7 | •Final review of the report and prototype. - Push all source code, UML, GUI files, and documentation to GitHub.  • -Prepare for submission: Ensure report is well structured, clear, and concise. | Completion of all tasks | 4-6 hours |

## Resources

**Human:** Individual with Microsoft Learn and WPF guides.

**Financial:** ZAR 50-100 (electricity, internet).

**Material:** PC, Word, GitHub.

## Budget

Negligible costs (ZAR 50-100), covered by personal resources.

## Roles and Responsibilities

**Developer (me):** Handles all tasks.

## Risk Management

**Risk:** Time overrun from .NET unfamiliarity.

**Mitigation:** 5-hour buffer, prioritize UML, weekly reviews.

**Risk:** Documentation inconsistency.

**Mitigation:** Rubric templates, bi-weekly proofreads.

## Communication Plan

Self-documented in Word and GitHub; self-review every three days.

## Quality Management

Adheres to .NET best practices, with UML validated for integrity and GUI meeting WCAG standards, final proofread ensures rubric alignment.

## Stakeholder Management

Module assessor engaged via compliant submission and GitHub.

## Assumptions and Constraints

**Assumptions:** User literacy, stable local database.

**Constraints:** 45-hour limit, free tool reliance.

# References

[1] Acceldata, "How to Maintain Entity Integrity for Accurate Data Systems," [Online]. Available: <https://www.acceldata.io/blog/how-to-maintain-entity-integrity-for-accurate-data-systems>. (Accessed: Sept. 2, 2025).

[2] Agile Data, "Implementing Referential Integrity and Shared Logic in a RDB," [Online]. Available: <https://agiledata.org/essays/referentialintegrity.html>. (Accessed: Sept. 2, 2025).

[3] Tennessee, "Relational Database Design Clearly Explained," [Online]. Available: <https://www.voteforthepig.tennessee.edu/Resources/5P8050/default.aspx/RelationalDatabaseDesignClearlyExplained.pdf>. (Accessed: Sept. 2, 2025).

[4] Stack Overflow, "WPF vs Windows Forms in desktop applications," [Online]. Available: <https://stackoverflow.com/questions/2054059/wpf-vs-windows-forms-in-desktop-applications>. (Accessed: Sept. 2, 2025).

[5] Quora, "Benefits of using C# and WPF to build desktop applications," [Online]. Available: <https://www.quora.com/What-are-the-benefits-of-using-C-and-WPF-to-build-desktop-applications>. (Accessed: Sept. 2, 2025).

[6] iFour Technolab, "UWP vs WPF - Key Differences Explained!," [Online]. Available: <https://www.ifourtechnolab.com/blog/uwp-vs-wpf>. (Accessed: Sept. 2, 2025).