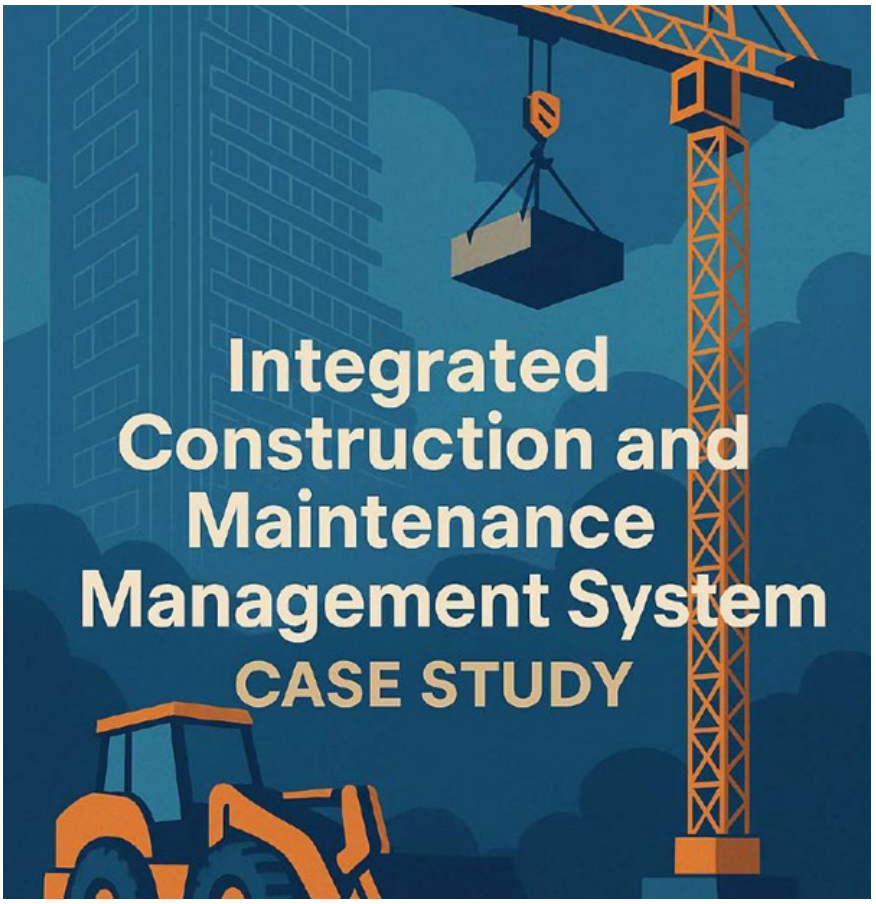
INSY7315 / WIL

Submission: Portfolio of Evidence (POE)

Date Created: 10th October 2025

A team presents…



Group Name: a team

Project Name:

Integrated Construction Maintenance Management System

(ICMMS) Development Project

Scrum Agile Board URL: Trello: link
GitHub repository URL: link
Code quality URL: to be added
Security report URL: to be added
Lucid Chart (all): link

Other relevant URLs: to be added

Members:

Name	Student Number	Role	Contact Email
Denzel Zimba	ST10383606	DevOps	ST10383606@vcconnect.edu.za
Daniel Jung	ST10324495	DevOps	ST10324495@vcconnect.edu.za
Braydon Wooley	ST10394807	Scrum Master, Front End Dev	ST10394807@vcconnect.edu.za
Nicolas Christofides	ST10339570	Front End Dev	ST10339570@vcconnect.edu.za
Max van der Walt	ST10354483	Lead Developer	ST10354483@vcconnect.edu.za

Table of Contents

DIAGRAMS: NOTE TO VIEWER

The diagrams included in this report are highly detailed and complex, which affects their print quality when converted to PDF or image formats. While printed versions have been provided, they may appear unclear or difficult to read due to the level of detail and scale.

For optimal clarity and full visibility, we strongly recommend accessing the diagrams using the provided Lucidchart links.
Links to all charts: https://lucid.app/lucidchart/ebc85260-6801-4178-a378-dd99698ea8e7/edit?viewport_loc=-742%2C-1085%2C4866%2C2059%2Cc3KVyjod~nFq&invitationId=inv_7bf09684-d572-42df-9856-a59685633a67

Part 1 - Project Roadmap & Work Agreement

1.1. Introduction

1.1.1. Project problem:

Dr Majoko Projects is a property development company managing construction and maintenance projects across several provinces. At the moment, they rely on separate tools like spreadsheets, paper forms, email, and WhatsApp to coordinate tasks between clients, contractors, project managers, and admin staff. This scattered approach leads to delays, poor accountability, and difficulty keeping projects on schedule and within budget.

1.1.2. Proposed solution:

The solution is to create an integrated construction and maintenance management system that works on both web and mobile. This system will connect all user roles in one platform, allowing them to log issues, track progress, assign work, share documents, manage budgets, and communicate efficiently. Key features will include real-time dashboards, automated alerts, secure file storage, quotation and invoice workflows, and an AI assistant to help estimate costs from blueprints and identify risks early.

1.1.3. Project background/context:

Dr Majoko Projects specialises in residential and commercial property development. Their services cover the full project lifecycle which is inherently difficult to keep track of. As their client base and project portfolio grow across multiple provinces, managing these operations has become increasingly complex.

Without a centralised system, project managers and site coordinators struggle to enforce deadlines, track expenses, and maintain clear communication. Clients often experience frustration from delays and lack of updates, while internal teams spend time chasing information instead of making progress.

This project will deliver a responsive, AI-assisted web and mobile application that replaces manual processes with digital workflows. It will make project data accessible in real time, standardise communication between all parties, and provide tools to keep budgets, timelines, and quality in check. By integrating construction and maintenance functions into one platform, Dr Majoko Projects will be able to operate more efficiently, make informed decisions faster, and offer a better experience for both clients and contractors.

1.2. Work Agreement

1.2.1. Team meeting schedule:

The team will meet every Thursday until 1 November 2025. Meetings will follow a sprint review and sprint planning format to ensure all members are aligned on progress, blockers, and upcoming work. Sprints are planned to end on Thursdays so that completed tasks can be reviewed immediately, and new work can be assigned without delays. Additional ad-hoc meetings may be scheduled for urgent technical issues or decision-making.

1.2.2. Communication tools and response times:

All group communication will be managed through a dedicated WhatsApp group for quick updates, brainstorming, and support requests. Trello will serve as the official project management board, tracking tasks, deadlines, and responsible members. The Trello board will be kept up to date before and after each work session, and members are expected to check it regularly.

Responses to messages should be given within 24 hours for standard items. Low-priority items may take up to 72 hours, but sprint-critical issues should be addressed as soon as possible. For high-priority technical problems, the responsible member must acknowledge the issue within 12 hours to avoid blocking other tasks.

1.2.3. Conflict resolution process:

Conflicts should be addressed as soon as they arise. Members should first attempt to resolve issues directly and professionally between themselves. If this does not work, and the matter is between two members, a neutral third member will be asked to mediate and record the resolution steps. If the conflict remains unresolved or affects multiple members, the issue will be escalated to the lecturer for formal mediation.

1.2.4. Role clarity and accountability:

Each member has a defined role in the project (Scrum Master, Lead Developer, DevOps, Front-End Developer), and is expected to deliver tasks assigned in Trello according to the sprint plan. Missed deadlines must be communicated in advance with reasons and an updated completion date. Persistent failure to deliver without communication will be escalated to the lecturer.

1.2.5. Collaboration standards:

All code will be stored in the team GitHub repository, following branch naming conventions and using pull requests with mandatory peer review before merging. Documentation will be kept current in the shared collaboration folder. All members must contribute to maintaining code quality (SonarQube), security checks (Snyk), and testing according to the agreed plan.

1.2.6. Backlog Items

In Scrum, the product backlog is the single source of all work that the team will take on. It is updated as the project evolves, with items small enough for one sprint and detailed through refinement. The team sizes tasks, while the Product Owner helps clarify priorities and trade-offs (Schwaber & Sutherland, 2020).

1.3. Definition of Ready (DoR)

The DoR is a set of conditions a backlog item must meet before it can be added to a sprint. It ensures the item is clear, testable, and has all the information and resources the team needs to finish it within the sprint. This avoids starting work on poorly defined tasks or those blocked by unresolved dependencies (PremierAgile, 2023).

- **Clear description** : The task has a clear title and description in Trello, including acceptance criteria that define exactly what needs to be delivered.
- **Linked to a user story**: The task is tied to a user story in the backlog with business priority and story points already assigned.
- **Designs and references attached:** Any required mockups, diagrams, API documentation, or reference material are attached or linked.
- **Dependencies identified**: All technical or resource dependencies are listed, and blocking items are either resolved or scheduled to be resolved before work starts.
- **Team understanding**: The assigned member confirms they understand the task, the scope, and the expected outcome before it is moved to “In Progress”.

1.4. Definition of Done (DoD)

The DoD is a checklist that confirms the work is finished to an agreed standard. It includes both technical quality and meeting acceptance criteria so the outcome can be delivered to the user without further fixes (PremierAgile, 2023).

- **Meets acceptance criteria**: The work matches the agreed requirements and passes functionality checks.
- **Code quality verified**: The code passes automated quality scans (SonarQube) with no new critical or high-severity issues introduced.
- **Security checks complete:** Snyk scan is run with no unaddressed vulnerabilities.
- **Testing complete**: Unit and integration tests are written or updated, and the feature has been tested on all relevant devices (web and/or mobile).
- **Merged to main branch:** The feature has been merged into the main branch via pull request with at least one peer review approval.
- **No breaking changes:** The new work does not cause regressions in other parts of the system and passes the build pipeline.
- **Documentation updated**: Any related documentation, diagrams, or setup instructions have been updated to reflect changes.
- **Demo ready:** The feature can be demonstrated in a sprint review without additional setup or fixes.

1.5. High-Level Roadmap

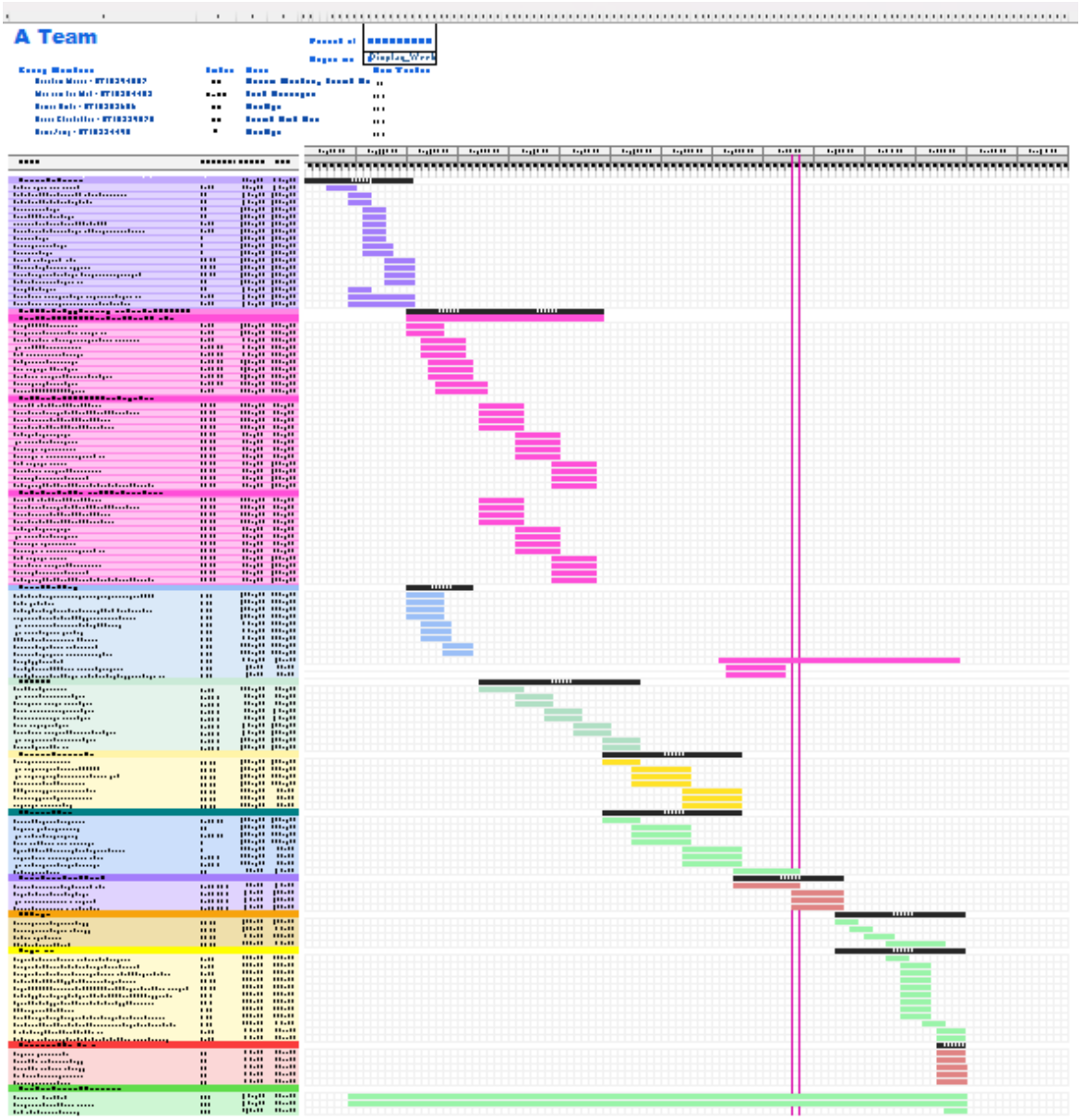


Figure 1.1 - Gantt Chart (Excerpt)

(Source: [Github] Docs/Part1/P1_01_GanttChart_ProjectRoadmap)

1.5.1. Sprint 1 - 4 Aug 2025 to 18 Aug 2025

Focus**: Complete Portfolio of Evidence.**
Deliverables:

- Full requirements capture, user roles, and user stories with backlog table.
- All system design diagrams (use case, ERD, bounded context, class, sequence, activity).
- UX assets including web/mobile mockups, journey maps, and navigation flows.
- Non-functional requirements documented.
- GitHub repository setup.
- Deployment, tech stack, and tools documentation.

1.5.2. Sprint 2 - 18 Aug 2025 to 28 Aug 2025

Focus: **Set up ASP.NET back-end, security, and API foundations.**
Deliverables:

- ASP.NET solution structure, authentication, role management, and database models.
- CRUD controllers, maintenance workflow, quotation/invoice logic, messaging, document, and reporting endpoints.
- API security (JWT, SSL, CORS).
- Test cases, bug tracking workflow, and SonarQube integration.
- Core security features including failed login tracking, role-based access control, and audit logs.

1.5.3. Sprint 3 - 28 Aug 2025 to 13 Sep 2025

Focus: **Develop Web and Mobile Front-End UIs**.
Deliverables:

- Role-specific dashboards (Admin, Project Manager, Contractor, Client) for web and mobile.
- Login/registration and role-based navigation.
- Forms for project creation and maintenance requests.
- Messaging system, document upload/view, quotation/invoice UI.
- Reporting and dashboard interfaces with visual charts (Gantt, Pie, Bar, Line).

1.5.4. Sprint 4 - 28 Aug 2025 to 18 Sep 2025

Focus: **Develop and test REST API.**
Deliverables:

- Planned API structure.
- Authentication, project management, maintenance request, and contractor assignment endpoints.
- Messaging and document management endpoints.
- Quotation and invoice endpoints.
- Postman testing of all endpoints.

1.5.5. Sprint 5 - 14 Sep 2025 to 2 Oct 2025

Focus: **Implement Quotation & Invoice System and begin AI Assistant.**
Deliverables:

- Quotation creation, plan upload, and basic plan parsing.
- Cost breakdown generation and quote approval/rejection.
- Invoice conversion and payment tracking.

- AI research for blueprint analysis, sample file preparation, and basic blueprint parsing.
- Manual cost estimation, exploration of AI APIs, integration into quotation module, and project delay prediction logic.

Initial AI testing and refinement

1.5.6. Sprint 6 - 2 Oct 2025 to 16 Oct 2025

Focus: **Add Voice Feedback feature and complete DevOps & Security.**
Deliverables:

- Research and design for web/mobile voice recording.
- Contractor voice memo upload with database storage/retrieval.
- Snyk security testing.
- CI/CD documentation and cloud deployment workflow.

1.5.7. Sprint 7 - 16 Oct 2025 to 2 Nov 2025

Focus: **Finalise UX Design.**
Deliverables:

- Responsive design finalised for web and mobile.
- UI testing on multiple devices.
- Adjustments based on feedback.
- Production Firebase environment configured (Hosting, Firestore, Storage, Authentication).
- API deployed to Firebase/Cloud Run with security settings (JWT, CORS, HTTPS).
- Final security review (Firestore/Storage rules, SSL, Snyk, SonarQube).
- Smoke testing on production environment.

1.5.8. Sprint 8 - 30 October 2025 to 2 Nov 2025

Focus: **Final Presentation and Collaboration Review.**
Deliverables:

- Presentation slides and combined web/mobile demo videos.
- Practice and delivery preparation.
- Commit history and branch documentation maintained.
- Submission of individual contribution logs.

1.6. Initial Risk Register:

This table shows the highest-priority risks to the project based on probability and impact.

Risk Description	% Prob	Impact	Mitigation Strategy
Delays in task completion due to academic workload or personal commitments	**High**	**High**	**Implement sprint buffer time in roadmap, redistribute tasks to available team members, enforce progress check-ins during standups.**
Version control issues on GitHub (conflicting changes, merge errors)	**Medium**	**Medium**	**Adopt strict branch naming conventions, use pull requests with mandatory peer review, schedule merge meetings before major pushes.**
Limited experience integrating AI functionalities	**High**	**Medium**	**Assign research responsibilities early (Max), use OpenAI documentation and GPT testing environment, start AI feature in earlier sprint (Sprint 4) to allow buffer.**
Poor coordination or communication among group members	**Medium**	**High**	**Use Trello for all task tracking, WhatsApp for updates, and enforce asynchronous standups every Thursday. Max will create and share the collaboration Word doc for updates.**
Technical setup challenges for code quality tools (e.g., SonarQube)	**Medium**	**Medium**	**Max will explore alternatives if setup fails (e.g., GitHub Actions). Pair programming or screen sharing if setup blockers occur.**
Security reporting tool (Snyk) not properly integrated	**Medium**	**Medium**	**Nicolas to set up Snyk and validate using dummy vulnerability, provide short guide or walkthrough for team if needed.**
Trello board or documentation not regularly updated	**Medium**	**Medium**	**Scrum Master (Braydon) to check-in twice weekly to ensure backlog and board reflect actual progress. Team encouraged to update cards before/after work sessions.**
Overreliance on GPT outputs without contextual editing	**High**	**Low**	**All GPT-generated outputs must be reviewed and customized (Daniel to revise Risk Register, Max to trim DoD). Assign reviewers for all content before submission.**

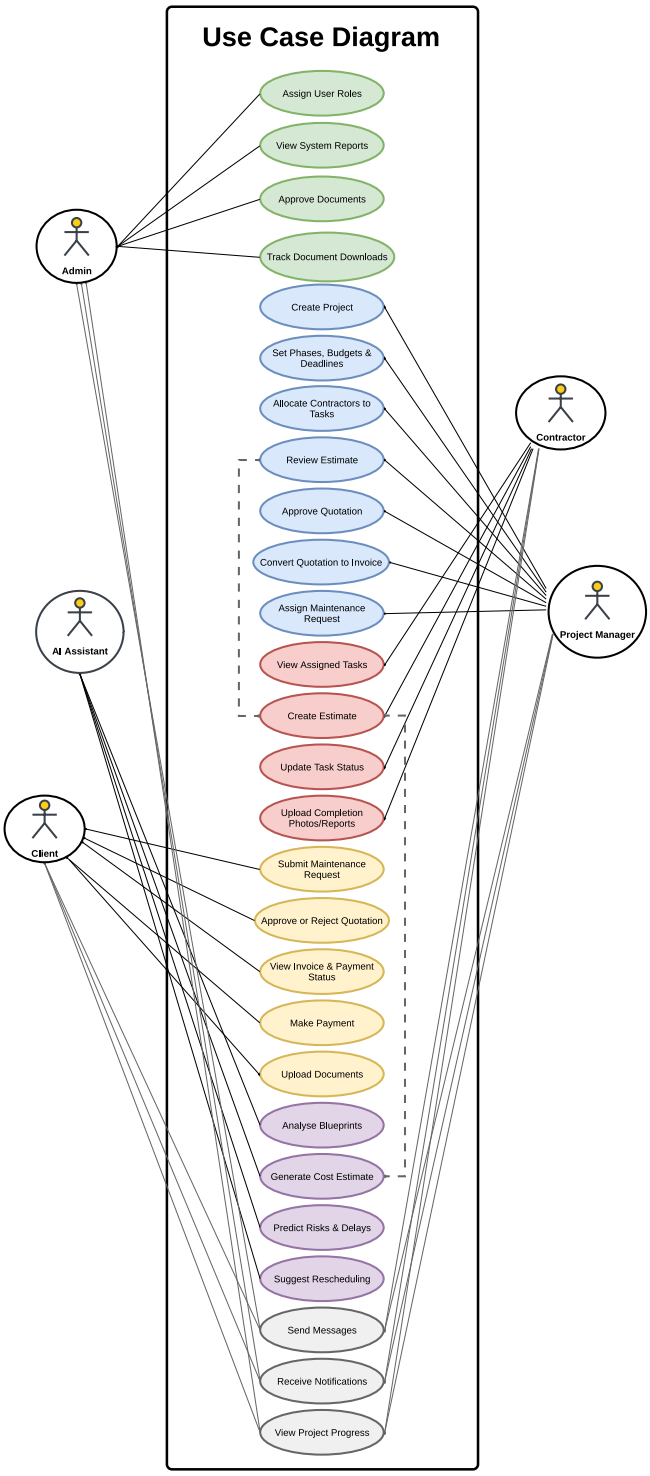
Figure 1.2 - Risk Register for ICMMS

Part 2 - Requirements & UX Design

2.1. Introduction

This section defines the functional and user-experience foundations of the Integrated Construction & Maintenance Management System (ICMMS). It documents all user roles, workflows, and requirements that shaped the application's architecture and interface design. The artefacts presented here including use-case, user-flow, and journey-map diagrams establish how the platform meets operational needs of administrators, project managers, contractors, and clients within a single secure environment.

2.2. Use Case Diagram



The ICMMS Use Case Diagram provides a high-level overview of how the system's actors interact with its primary modules. It visualises every role's responsibilities, system-triggered processes, and shared interactions between components such as quotation approval, maintenance requests, and AI-based estimation.

Key interactions include:

- **Administrator:** Assign user roles, approve documents, and monitor reports and downloads.
- **Project Manager:** Create projects, define phases and budgets, allocate contractors, review estimates, and convert quotations to invoices.
- **Contractor:** View assigned tasks, update status, and upload completion reports and photos.
- **Client:** Submit maintenance requests, approve or reject quotations, and view invoices and payments.
- **AI Assistant:** Analyse blueprints, generate cost estimates, predict risks and delays, and suggest rescheduling.

Figure 2.1 – ICMMS Use Case Diagram.
Source: Lucid Chart*:* link here

2.3. User Roles, Stories, Data Flow, UX Journey Maps

ICMMS is designed around four main user roles. Each role has distinct permissions, CRUD capabilities, and interface access.

2.3.1.1. Administrator Roles:

Oversees the entire platform's operation, ensuring smooth workflows and system compliance.

Key CRUD Tasks:

- **Create:** User accounts, role permissions, system configurations.
- **Read:** All communication logs, reports, documents, project statuses.
- **Update:** User permissions, notifications, system settings.
- **Delete:** Remove outdated records, deactivated users, or invalid documents.

Notes:

- Has direct messaging ability with Project Managers when necessary.
- Oversees all communication logs and can initiate conversations if needed.
- Approves quotations in workflow; client must accept final quote before invoice generation.
- Monitors notifications and ensures SMS/email/app alerts reach the right stakeholders.

UI screens:

- Admin Dashboard
- Reports Panel
- Assignment Manager
- Messaging & Notification Panel

2.3.1.2. Administrator Stories:

- As an Admin, I want to create and manage user accounts so that each user has the correct role permissions.
- As an Admin, I want to view all communication logs so that I can oversee and ensure compliance.
- As an Admin, I want to approve quotations in the workflow so that the client can give final acceptance.
- As an Admin, I want to monitor system notifications so that alerts reach the correct stakeholders.

2.3.1.3. Admin User Flows

The administrator's flow begins at login and navigation to the User Management module, where all CRUD actions are performed. The system validates inputs for new users (email, password, role) before creating records in both Firebase Auth and Firestore.
Admins can create, view, edit, toggle status, or delete users.

Admin User flows

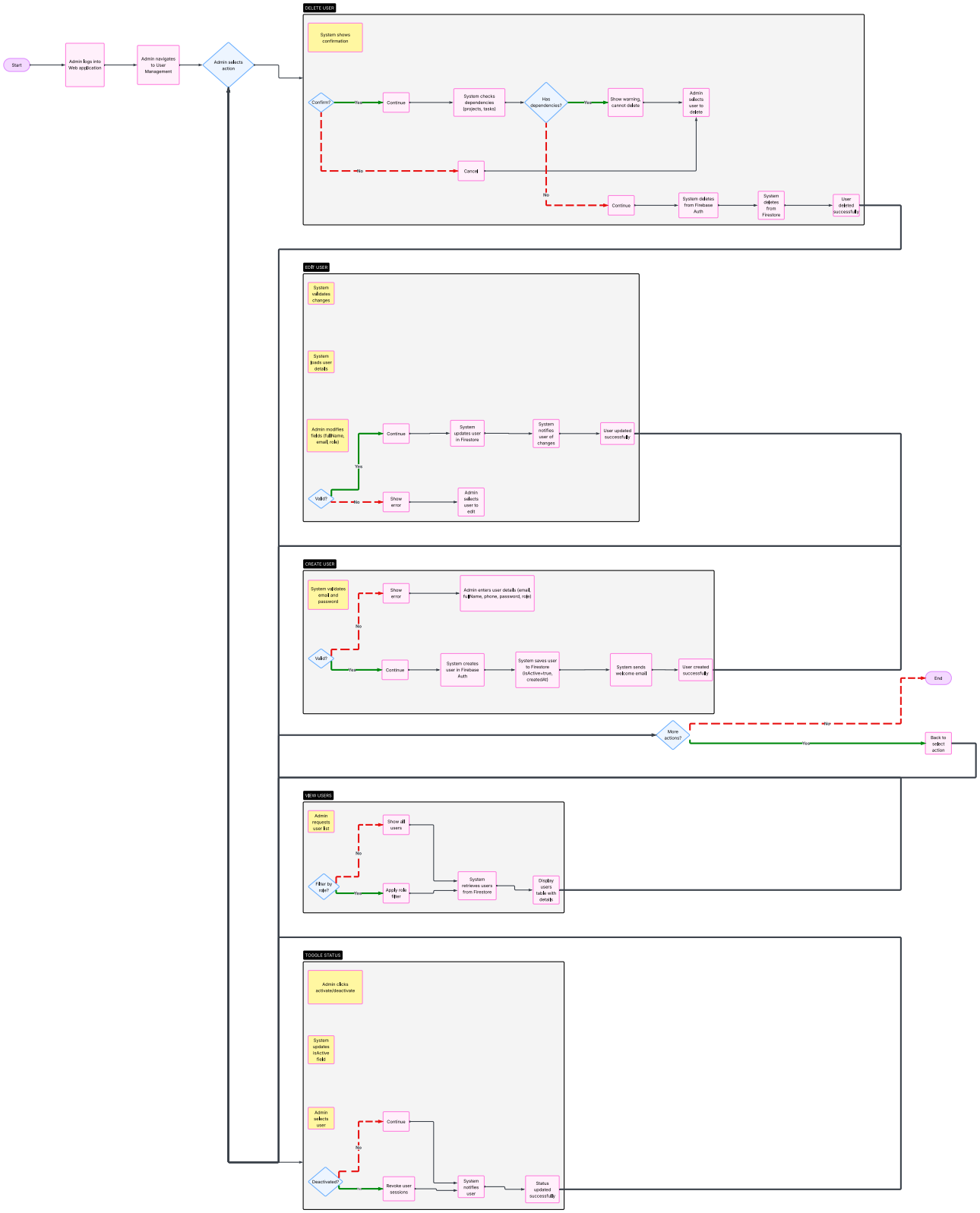


Figure 2.1.1 - Administrator User Flow (CRUD and Validation Process) Source: Lucid Chart: link here

2.3.1.4. Admin UX Journey Map

The Administrator journey covers six operational stages, from login to daily maintenance. It highlights how system control, configuration, and analytics are managed centrally. After login, the Admin accesses a system health dashboard showing uptime, performance metrics, and recent activity. The user then manages accounts, assigns roles, and configures integrations such as Firebase and Supabase. Monitoring, audit logging, and backup processes follow in the daily workflow, ending with support and scheduled maintenance tasks.Key pain points include complex configurations and alert overload, while positive moments include automated backups, role-based security, and comprehensive audit trails.

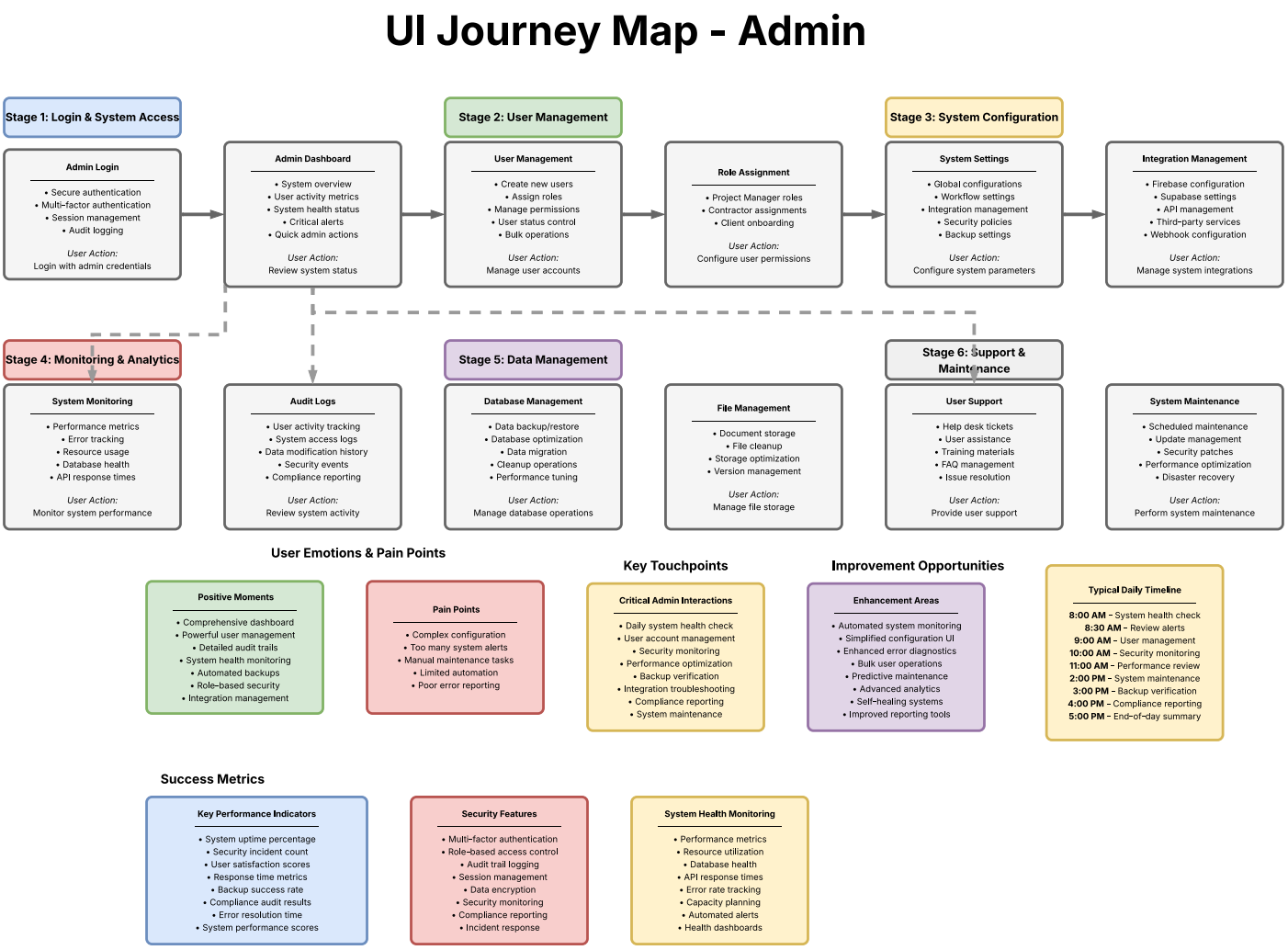


Figure 2.1.2 – Administrator UX Journey Map. Source: Lucid Chart: link here

2.3.2.1. Project Manager Roles:

Manages construction projects from creation to completion, ensuring delivery within budget and timelines.

Key CRUD Tasks:

- Create: New construction projects with quotes &invoice including budgets, phases, tasks per phase, deadlines,
- Read: Project status, budget vs actual reports, contractor updates.
- Update: Resource allocations, task assignments, document approvals, progress reports.
- Delete: Cancel projects or remove incorrect data.

Notes:

- Sole role that creates projects (admins do not).
- Assigns contractors to tasks and approves documents.
- Uses AI-generated quotes for cost planning but does not approve quotes.
- Communicates actively with contractors and clients for updates and clarifications.

UI Screens:

- PM Dashboard
- Quotes Creation & Invoice View
- Individual Tab Per Project's Page
- Review Screen
- Contractor Tracker
- Notification Panel

2.3.2.2. Project Manager Stories:

- As a Project Manager, I want to create construction projects, create quotes and get approvals with budgets, phases, tasks, and deadlines so that work is planned effectively.
- As a Project Manager, I want to assign contractors to specific tasks so that work is completed on time.
- As a Project Manager, I want to approve documents so that only verified files are used.
- As a Project Manager, I want to use AI-generated quotes for planning so that I can manage project costs.

2.3.2.3. Project Manager User Flows

The Project Manager workflow comprises three core streams:

1. **Project & Phase Management**: create, edit, or delete projects and phases after validation of dates and budgets. Each phase is stored with status tracking and linked to assigned contractors.
2. **Quotation & Invoice Management**: create & review draft quotations, approve or reject with reasons, and convert client-approved quotations to invoices.
3. **Task Assignment & Monitoring**: assign contractors, track progress, review completion, and verify deliverables. Overdue or unsatisfactory tasks trigger alerts and rework requests.

Every stage includes automated notifications to clients and contractors for status updates and invoice issues, ensuring accountability and project transparency

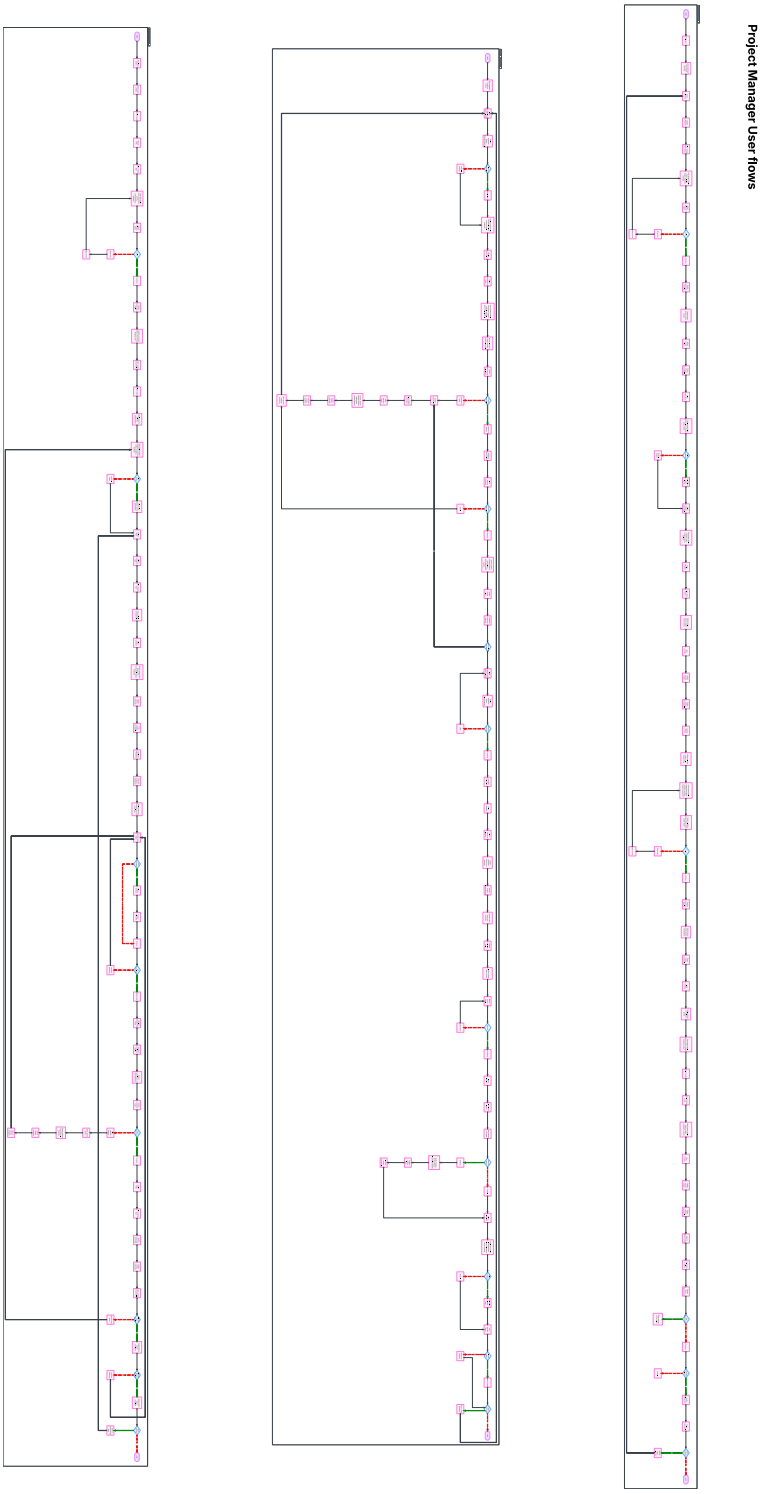


Figure 2.2.1 - Project Manager User Flow (End-to-End Lifecycle). Source: Lucid Chart*:* link here

2.3.2.4. Project Manager UX Journey Map

The Project Manager journey illustrates a full cycle of project oversight. Upon login, the PM dashboard displays all active projects, budgets, pending approvals, and maintenance requests. The user navigates between project creation, quotation approval, and contractor assignments through a unified interface. Daily touchpoints include quotation reviews, team communications, and analytics dashboards. Positive experiences include real-time notifications, clear project visualisation, and strong collaboration tools, while challenges arise from heavy data entry and long approval workflows.

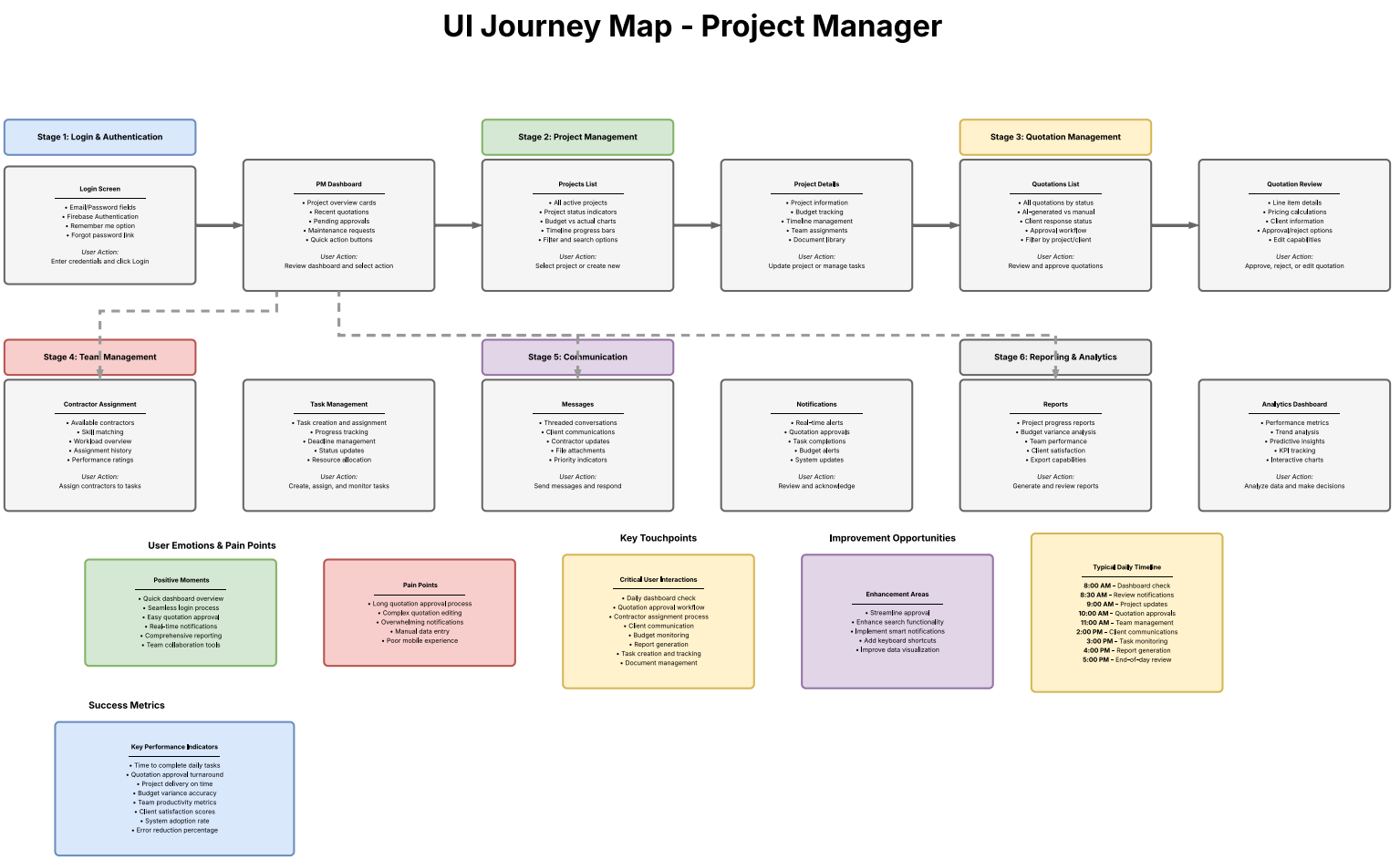


Figure 2.2.1 - Project Manager UX Journey Map. Source: Lucid Chart***:*** link here

2.3.3.1. Contractor Roles:

The Contractor executes assigned tasks on projects or maintenance jobs.

Key CRUD Tasks:

- Create: Task completion updates, upload completion photos/reports.
- Read: Assigned tasks, deadlines, instructions.
- Update: Task progress, status changes (Pending → In Progress → Completed).
- Delete: Remove incorrect (their own) submissions before final approval.

Notes:

- Works only on tasks assigned by Project Managers.
- Communicates progress to Project Managers via messaging.
- Cannot approve quotes or create projects.

UI Screens

- Contractor Dashboard including Task List
- Individual Tab Per Project's Page (simplified)
- Completion Report Form
- Notification Panel

2.3.3.2. Contractor Stories:

- As a Contractor, I want to view assigned tasks so that I know my responsibilities.
- As a Contractor, I want to update task progress so that the Project Manager is informed.
- As a Contractor, I want to upload completion photos so that task verification is easier.

2.3.3.3. Contractor User Flows

The Contractor flows cover three processes:

1. **Task Management**: View assigned tasks, start work, log progress, and mark completion. System automatically notifies the Project Manager for verification.
2. **Document Management**: Upload reports and media files to Supabase Storage; metadata is recorded in Firestore with status tracking and version updates.
3. **Estimate & Quotation Creation**: Generate AI-assisted estimates from blueprints or manual entry, convert to quotations, and submit for PM approval.

The flows ensure work accuracy and speed by combining AI cost estimation with manual verification, supporting secure document handling and transparent communication with PMs and clients.

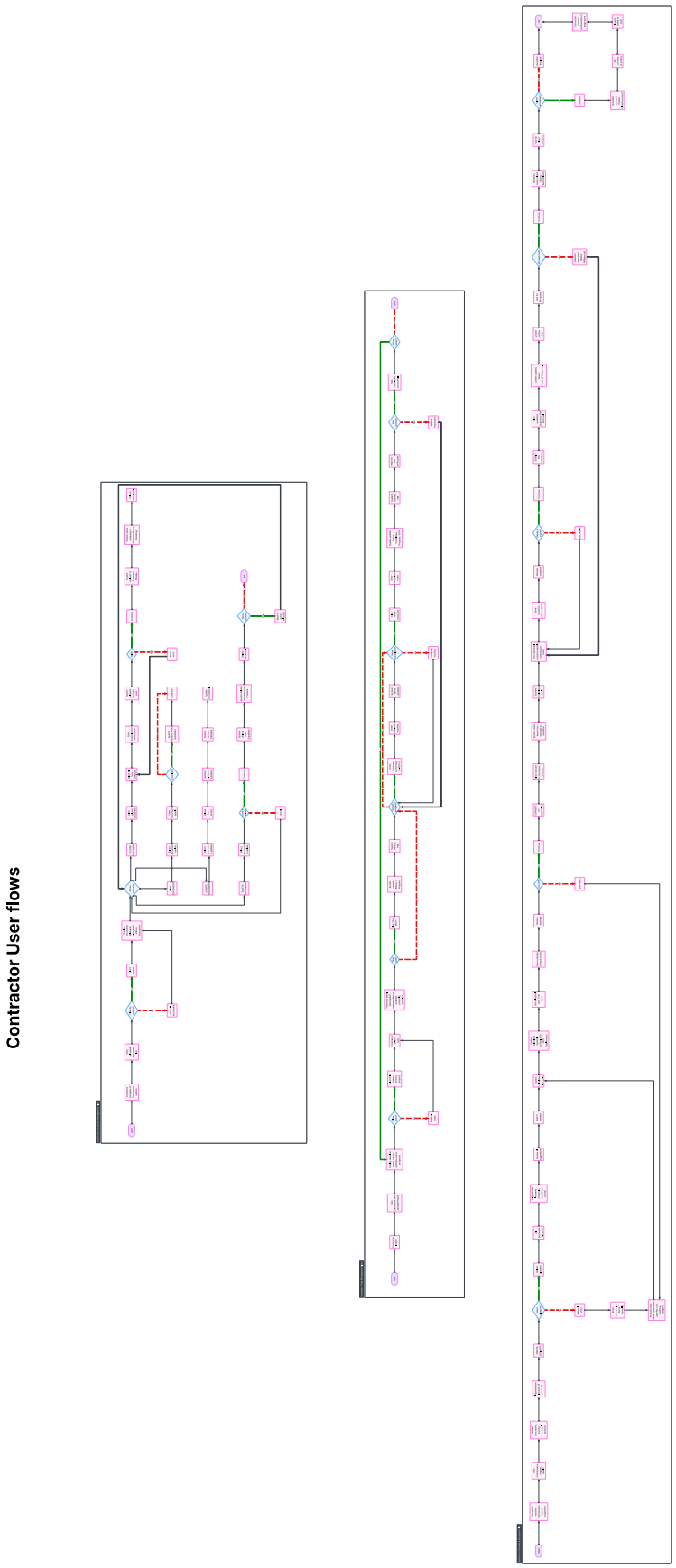


Figure 2.3.1 – Contractor User Flow (Task, Document & Quotation Process). Source: Lucid Chart*:* link here

2.3.3.4. Contractor UX Journey Map

The Contractor journey focuses on mobile-first usability, task execution, and communication. Starting with biometric login and offline access, contractors immediately see assigned jobs and maintenance requests on their dashboard. They can review details, track progress, upload before/after photos, and communicate with project managers.
The workflow concludes with task completion reports, performance tracking, and earnings summaries.
Positive aspects include intuitive mobile access and instant notifications, while major pain points involve weak connectivity, unclear instructions, and limited offline capability.

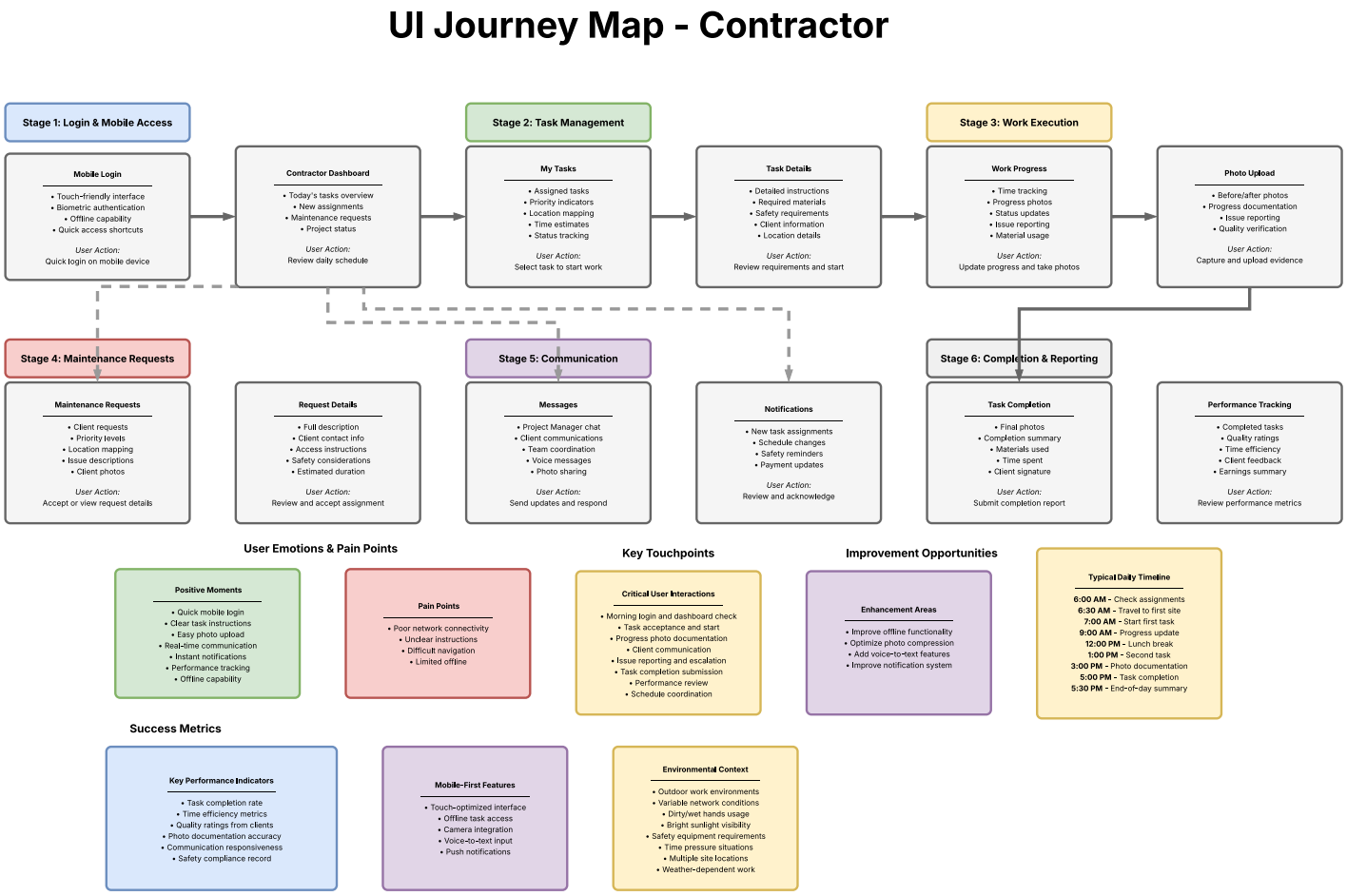


Figure 2.3.2 - Contractor UX Journey Map. Source: Lucid Chart*:* link here

2.3.4.1. Client Roles:

Requests maintenance, receives project updates, and approves final quotations.

Key CRUD Tasks:

- Create: Maintenance requests with descriptions/images.
- Read: Progress tracking, quotations, invoices, reports.
- Update: Approve/reject quotations, confirm completion of maintenance.
- Delete: Cancel pending maintenance requests before work starts.

Notes:

- Only role that can approve/reject quotations before invoice generation.
- Communicates with Project Managers and Contractors for status and clarifications.
- Receives automated alerts for every project/maintenance stage.

UI Screens

- Client Dashboard
- Maintenance Request Form
- Notification Panel
- Individual Tab Per Project's Page (simplified)

2.3.4.2. Client Stories:

- As a Client, I want to submit maintenance requests with descriptions and images so that issues can be addressed
- As a Client, I want to track the status of my requests so that I know when they will be resolved.
- As a Client, I want to approve or reject quotations so that only agreed work is billed.

2.3.4.3. Client User Flows

The Client journey is divided into three operational flows:

1. **Maintenance Request Flow**: Clients submit requests with descriptions and media attachments; system validates fields and creates unique IDs, then notifies the assigned Project Manager.
2. **Quotation Review Flow** : Clients evaluate quotations sent by PMs, accept or decline with notes, and automatically trigger invoice creation upon acceptance.
3. **Payment Flow**: Clients pay via EFT or PayFast/PayPal gateways; Project Manager records transactions, updates invoice status to 'Paid,' and issues receipts to both client and PM.

These flows illustrate the client's entire experience from maintenance submission to payment confirmation, highlighting the system's automation and audit trail capabilities.

Client User flows

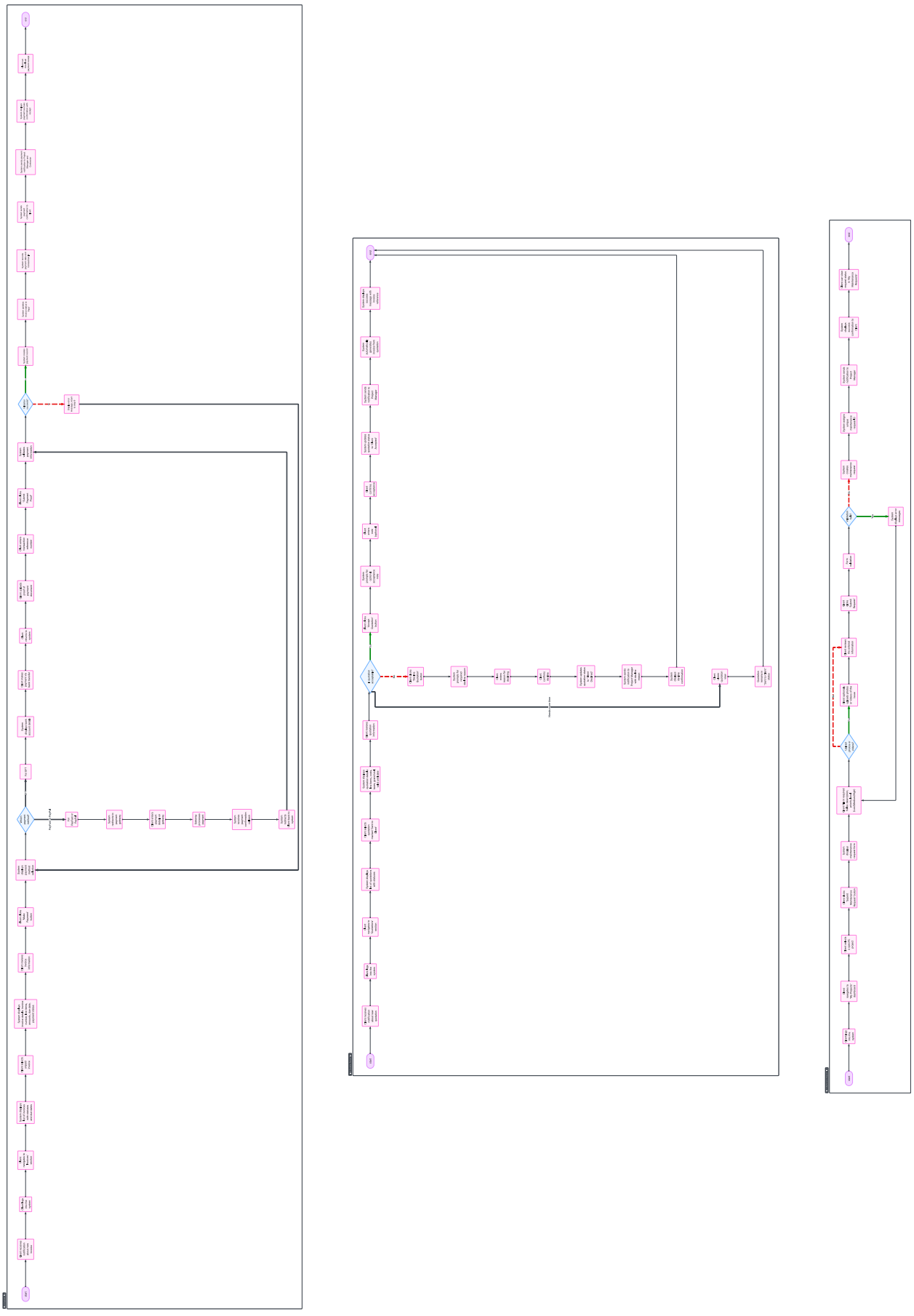


Figure 2.4.1 – Client User Flow (Maintenance, Quotation & Payment. Source: Lucid Chart***:*** link here

3.1.4.1. Client UX Journey Map

The Client journey traces the customer experience from registration to project completion and payment. After onboarding, clients can view project timelines, monitor progress, and access quotations.
Key stages include quotation acceptance, maintenance request submission, and payment confirmation. The interface offers visibility into every project milestone, communication thread, and invoice record.Positive feedback centres on transparency and secure payments, while challenges include complex quotation details and occasional processing delays.

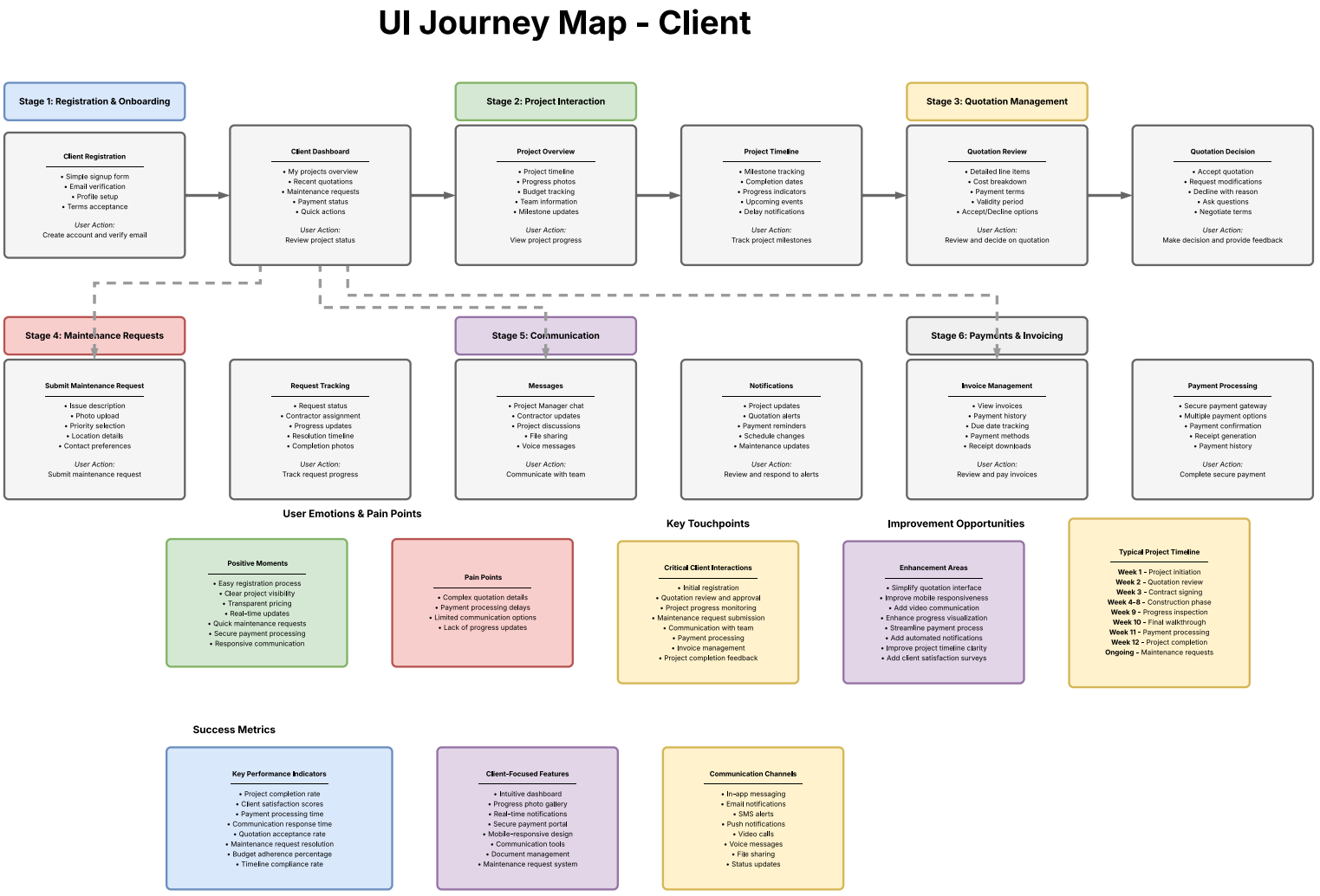


Figure 2.4.2 - Client UX Journey Map. Lucid Chart*:* link here

2.4. User Story Backlog Table

The user-story backlog consolidates all functional requirements defined during early sprints.
Each story links directly to a module in ICMMS.
Stories are prioritised by business impact and sprint allocation.

ID	User Story	Priority	Module/Feature
US-01	As an Admin, I want to create and manage user accounts so that each user has correct access.	High	User Management
US-02	As an Admin, I want to view all communication logs so that I can ensure compliance.	High	Communication & Notifications
US-03	As an Admin, I want to monitor system notifications so alerts reach correct stakeholders.	Medium	Communication & Notifications
US-04	As a Project Manager, I want to create construction projects with budgets and deadlines.	High	Project Management
US-05	As a Project Manager, I want to assign contractors to tasks so that work is completed on time.	High	Project Management
US-06	As a Project Manager, I want to approve documents so that only verified files are used.	High	Document Management
US-07	As a Project Manager, I want to use AI-generated quotes for planning so I can manage costs.	Medium	AI-Powered Estimation
US-08	As a Contractor, I want to view assigned tasks so that I know my responsibilities.	High	Contractor Portal
US-09	As a Contractor, I want to update task progress so that the Project Manager is informed.	High	Contractor Portal
US-10	As a Contractor, I want to upload completion photos so that task verification is easier.	Medium	Contractor Portal
US-11	As a Client, I want to submit maintenance requests so that issues can be addressed.	High	Maintenance Request Management
US-12	As a Client, I want to track the status of my requests so that I know when they'll be resolved.	High	Maintenance Request Management
US-13	As a Client, I want to approve or reject quotations so that only agreed work is billed.	High	Quotation Management

Table 2.1 - User Story Backlog Overview
(Source: docs/diagrams/Part2/P2_03_UserBacklogTable.pdf).
Each record specifies ID, story statement, priority level, and functional module to assist sprint planning and progress tracking.
This table also serves as the foundation for your Functional Requirements Matrix in Part 3.

2.5. UX Journey Map - All Roles (System Overview)

The consolidated UX Journey Map presents a holistic view of how the system interconnects across roles and modules. It visualises the entire ICMMS ecosystem from authentication to project delivery and reporting. It also illustrates how multi-role coordination supports real-time decision-making and communication.

The diagram maps each actor (Admin, Project Manager, Contractor, Client) against the system core modules, including Firebase Authentication, Firestore Database, Supabase Storage, and ASP.NET Core MVC interfaces.
It highlights major workflows such as:

- **Quotation Cycle:** Project Manager creates → Client reviews → Contractor executes → Admin monitors.
- **Maintenance Workflow:** Client submits request → Project Manager assigns → Contractor completes → Client approves.
- **Communication Flow:** Real-time messaging, push notifications, and email/SMS alerts keep all stakeholders in sync.

The consolidated journey map demonstrates how the ICMMS enables seamless collaboration across all stakeholders, improving communication and project transparency. By unifying user interactions within one integrated platform, it ensures faster decision-making, accurate information flow, and complete accountability from project creation to completion.

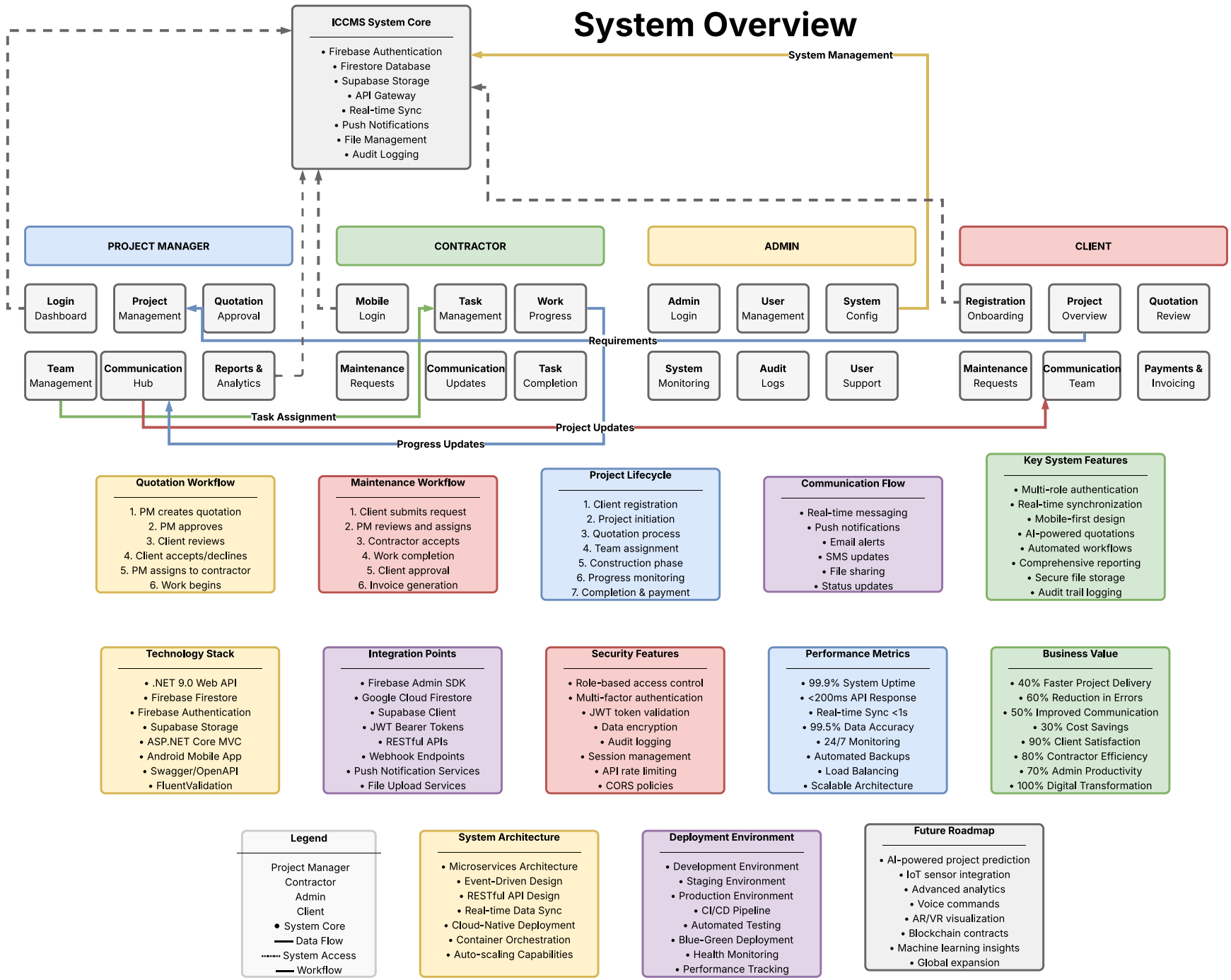


Figure 2.5 - UX Journey Map for All Roles (System Overview). Lucid Chart: link here

2.6. Functional and Non-Functional Requirements

The following sections formalize the operational scope of the Integrated Construction & Maintenance Management System (ICMMS). While the preceding diagrams and user stories describe how users interact with the platform, these requirement specifications define what the system must deliver and how it must perform.

Together, they ensure alignment between business needs, technical implementation, and evaluation criteria used during testing.

2.6.1. Functional Requirements

The functional requirements document defines all features and behaviours expected from the ICMMS, grouped into nine major modules.
It specifies the capabilities required to support each user role and provides the baseline for development, testing, and traceability through later phases.

FR 1.0: User Management

- **FR 1.1** User Onboarding and Role Management - Admins shall create, update, deactivate, and assign roles (Admin, Project Manager, Contractor, Client)
    - **FR 1.1.1** Admins shall be able to view all created users.
    - **FR 1.1.2** Admin shall be able to reassign roles.
- **FR 1.2** User Authentication and Access Control - The system shall authenticate users via Firebase Authentication and enforce role-based access rules.
    - **FR 1.2.1** Authentication shall require email and password.
    - **FR 1.2.2** Access rules shall restrict actions to permitted roles.
    - **FR 1.2.3** Login Attempt Tracking - The system shall track, and limit failed login attempts.
        - **FR 1.2.3.1** After 5 consecutive failed login attempts, the account shall be locked for 5 minutes.
        - **FR 1.2.3.2** Successful login shall reset the failed attempt counter.
        - **FR 1.2.3.3** Lockout timer shall display remaining time to user.

FR 2.0: Project Management

- **FR 2.1** Project Creation - Only Project Managers shall create projects with timelines, budgets, and phases
- **FR 2.2** Project Managers shall be able to allocate resources per phase.
- **FR 2.3** Project Tracking - Project Managers shall track actual progress vs planned milestones.
    - **FR 2.3.1** Progress tracking shall include percentage completion.
    - **FR 2.3.2** Milestone status updates shall be logged.

FR 3.0: Maintenance Management

- **FR 3.1** Maintenance Request Submission - Clients shall submit maintenance requests with descriptions, photos, and videos.
    - **FR 3.1.1** Priority Levels - Maintenance requests shall include priority designation (Low, Medium, High, Critical).
- **FR 3.2** Maintenance Request Tracking - Clients shall be able to track maintenance updates.

- **FR 3.3** Contractor Assignment - Project Managers shall assign contractors to maintenance requests or project tasks.
- **FR 3.4** Task Management - Contractors shall view assigned tasks, update progress, and upload completion evidence.
    - **FR 3.4.1** Evidence Upload – Contractors shall upload photos, videos, or documents as task completion evidence.
    - **FR 3.4.2** Evidence Storage 0 Evidence files shall be stored in Supabase and linked to specific tasks.
- **FR 3.5** Task Corrections - Contractors shall delete or correct their own submissions before final approval.

FR 4.0: Document Management

- **FR 4.1** Document Upload & Access – All users shall upload, access, and download project documents according to permissions.
- **FR 4.2** Document Upload Type – All users shall be able to upload PDF, DOCX, PNG or DWG files.
- **FR 4.3** Document Version Control – The system shall maintain version history for documents.
- **FR 4.4** Document Verification – Project Managers shall review and verify uploaded documents for accuracy and completeness.
- **FR 4.5** Document Approval – Admins shall approve verified documents to ensure all project stakeholders are working from approved and accurate files.
- **FR 4.6** Document Access Logging – The system shall log all document views, and modifications.
    - **FR 4.6.**1 Admins shall have full visibility of document access logs.
    - **FR 4.6.2** Project Managers shall have visibility of document access logs for projects they manage.

FR 5.0: Communication

- **FR 5.1 Messaging Infrastructure** - The system shall provide comprehensive messaging capabilities for all authenticated users.
    - **FR 5.1.1** All authenticated users (Admin, Project Manager, Contractor, Client) shall have access to the messaging system.
    - **FR 5.1.2** Messages shall be stored in Firebase Firestore with real-time synchronization.
- **FR 5.2 Message Types** - The system shall support multiple message communication patterns.
    - **FR 5.2.1** Direct Messages - Users shall send one-to-one messages to specific recipients.
    - **FR 5.2.2** Thread Messages - Users shall participate in multi-party threaded conversations.
    - **FR 5.2.3** Broadcast Messages - Admins shall broadcast messages to all active users in the system.
- **FR 5.3 Message Attachments** - Users shall attach files to messages for enhanced communication.
    - **FR 5.3.1** Attachment Storage - Files shall be stored in Supabase Storage with public URLs generated for authorized access.

    - **FR 5.3.2** Attachment Management - Users shall be able to view and download attachments and uploaders may delete their own attachments.
    - **FR 5.3.3** Attachment Status – Deleted attachments shall be marked as "deleted" rather than physically removed, preserving audit trail.
- **FR 5.4 Message Search and Filtering** - Users shall be able to search and filter messages efficiently.
    - **FR 5.4.1** Search by Content - Users shall search messages by text content in subject and body.
    - **FR 5.4.2** Filter by User - Users shall filter messages by sender or receiver.
    - **FR 5.4.3** Filter by Project - Users shall view all messages related to a specific project.
    - **FR 5.4.4** Read Status Filtering - Users shall filter messages by read/unread status.
- **FR 5.5 Message Status and Tracking -** The system shall track message delivery and read status.
    - **FR 5.5.1** Read Receipts - Messages shall track read status with ReadAt timestamp.
    - **FR 5.5.2** Unread Count - The system shall maintain unread message counts per user and per thread.
- **FR 5.6 Broadcast Messaging** - Admins shall broadcast messages to multiple users simultaneously.
    - **FR 5.6.1** Admin Broadcast - Only users with Admin role shall create broadcast messages.
    - **FR 5.6.2** Recipient Selection - Broadcast messages shall be sent to all active users in the system.
    - **FR 5.6.3** Individual Message Records - System shall create individual message records for each recipient to track read status independently.
- **FR 5.7 Message Validation** - The system shall validate message content before delivery.
    - **FR 5.7.1** Content Validation - Messages shall be validated for: Required fields (SenderId, ReceiverId, Content), Content length limits, Valid participant references
    - **FR 5.7.2** Validation Feedback 0 Validation errors shall be returned to sender with specific error messages.
    - **FR 5.7.3** Validation Warnings 0 Non-critical issues shall generate warnings without blocking message delivery.
- **FR 5.8 Workflow Messaging** - The system shall send automated messages for business process events.
    - **FR 5.8.1** Quotation Workflow Messages - System shall send automated notifications when: Quotation is submitted for approval, Quotation is approved by Project Manager, Quotation is rejected by Project Manager, Quotation is sent to client, Quotation is accepted or declined by client
    - **FR 5.8.2** Invoice Workflow Messages - System shall send automated notifications when: Invoice is generated from accepted quotation, Invoice is sent to client, Payment is received, Invoice becomes overdue
    - **FR 5.8.3** Project Update Messages - System shall send automated notifications for: Project milestone completions, Task assignments, Project status changes

    - **FR 5.8.4** System Alert Messages - Admins shall send system-wide alerts for critical notifications.
- **FR 5.9 Message Notifications** - Users shall receive real-time notifications for new messages.
    - **FR 5.9.1** Push Notifications - New messages shall trigger push notifications to recipient's registered devices via Firebase Cloud Messaging.
    - **FR 5.9.2** Notification Content - Push notifications shall include: Sender name, Message subject or preview, Message type indicator, Attachment indicator (if present)
    - **FR 5.9.3** Notification Data - Notifications shall include metadata for direct navigation to the message or thread.

FR 6.0: Notifications

- **FR 6.1** Real-Time Notifications - The system shall send real-time alerts for new assignments, updates, and overdue tasks.
    - **FR 6.1.1** Notifications shall be delivered via email and push notifications using Firebase Cloud Messaging (FCM).
    - **FR 6.1.2** Notifications shall be sent out to project shareholders regarding the completion of project milestones.
    - **FR 6.1.3** Users shall register device tokens for receiving push notifications.
    - **FR 6.1.4** Push notifications shall include rich data for direct navigation to relevant content (messages, tasks, quotations, invoices).

FR 7.0: Quotation and Invoice Management

- **FR 7.1 Estimate Generation -** The system shall generate cost estimates from uploaded blueprints using AI processing.
    - **FR 7.1.1 Blueprint Upload** - Contractors and Project Managers shall upload building plans (blueprints) for AI processing.
    - **FR 7.1.2 AI Blueprint Processing -** The AI system shall parse uploaded building plans using computer vision and natural language processing**.**
        - **FR 7.1.2.1** The AI shall identify rooms, dimensions, and construction elements from blueprints.
        - **FR 7.1.2.2** The AI shall extract construction line items including material types and quantities.
        - **FR 7.1.2.3** Each extracted line item shall include AI confidence score (0.0 to 1.0).
    - **FR 7.1.3 Material Database Integration -** The system shall maintain a construction materials pricing database.
        - **FR 7.1.3.1** Materials shall be categorized (Concrete, Masonry, Finishing, Steel, Electrical, Plumbing, etc.).
        - **FR 7.1.3.2** AI-extracted line items shall be matched against the material database for automatic pricing.
    - **FR 7.1.5 Estimate Calculation -** The system shall automatically calculate: Subtotal (sum of all line item totals), Tax Total (15% VAT on subtotal), Total Amount (subtotal + tax)
- **FR 7.2 Estimate Review and Conversion -** Project Managers shall review and convert estimates to quotations.

    - **FR 7.2.1 Estimate Review -** Project Managers shall review AI-generated estimates for accuracy.
    - **FR 7.2.2 Manual Pricing -** Items marked "REQUIRES MANUAL PRICING" shall be priced by Project Manager.
    - **FR 7.2.3 Estimate Editing -** Project Managers may edit line items, quantities, and prices before conversion.
    - **FR 7.2.4 Conversion to Quotation -** Project Managers shall convert approved estimates to quotations for client presentation.
        - **FR 7.2.4.1** All estimate line items shall be converted to quotation items.
- **FR 7.3 Quotation Workflow -** Quotations shall follow a defined approval and acceptance workflow.
    - **FR 7.3.1 Quotation Creation -** Project Managers shall create quotations for clients.
        - **FR 7.3.1.1** Quotations may be created from estimates or manually.
        - **FR 7.3.1.2** Each quotation shall be assigned to a specific ClientId.
        - **FR 7.3.1.3** Quotations may be associated with Projects or MaintenanceRequests.
    - **FR 7.3.4 Quotation Status Workflow -** Quotations shall transition through the following statuses:
        - **FR 7.3.4.1 Draft -** Initial creation state; quotation can be edited.
        - **FR 7.3.4.2 PendingPMApproval** - Submitted for Project Manager approval; requires PM review.
        - **FR 7.3.4.3 SentToClient -** Approved by PM and sent to client for decision.
        - **FR 7.3.4.4 ClientAccepted -** Client has accepted the quotation; ready for invoice generation.
        - **FR 7.3.4.5 ClientDeclined -** Client has rejected the quotation.
        - **FR 7.3.4.6 PMRejected -** Project Manager has rejected the quotation; requires revision.
- **FR 7.4 Quotation Approval Process -** Project Managers shall approve quotations before client presentation.
    - **FR 7.4.1 Submit for Approval -** Project Managers shall submit Draft quotations for approval.
        - **FR 7.4.1.1** Quotations must have at least one item and GrandTotal > 0 to submit.
        - **FR 7.4.1.2** Status shall change from Draft to PendingPMApproval.
    - **FR 7.4.2** PM Approval Project Managers shall approve PendingPMApproval quotations.
        - **FR 7.4.2.1** Approval shall change status to SentToClient.
        - **FR 7.4.2.2** Automated workflow message shall be sent to client.
    - **FR 7.4.3 PM Rejection -** Project Managers may reject quotations with reason.
        - **FR 7.4.3.1** Status shall change to PMRejected.
        - **FR 7.4.3.2** Rejection reason shall be stored.
    - **FR 7.4.4 PM Editing** - Project Managers may edit quotations in Draft or PMRejected status.
- **FR 7.5 Client Quotation Decision -** Clients shall accept or decline quotations sent to them.

    - **FR 7.5.1 Client Access -** Clients shall view only quotations where ClientId matches their UserId.
    - **FR 7.5.2 Quotation Validity -** Clients may only accept/decline quotations before a deadline.
    - **FR 7.5.3 Client Acceptance -** Clients shall accept quotations they wish to proceed with.
    - **FR 7.5.4 Client Rejection -** Clients shall decline quotations they do not wish to proceed with.
- **FR 7.6 Invoice Generation -** Invoices shall be automatically generated from accepted quotations.
- **FR 7.7 Payment Tracking -** The system shall log and track payments against invoices.
    - **FR 7.7.2 Mock Payment System -** The application shall use a mock non-functional payment system for all specified payment options during development and testing.
    - **FR 7.7.3 Payment Status -** The system shall update invoice status upon payment confirmation:
    - **FR 7.7.4 Payment History -** The system shall maintain complete payment history for all invoices.
- **FR 7.8 Multi-Currency Support -** Quotations and invoices shall support currency designation.
    - **FR 7.8.1 Default Currency -** Default currency shall be ZAR (South African Rand).
    - **FR 7.8.2 Currency Field -** All quotations and invoices shall include a Currency field.
    - **FR 7.8.3 Currency Consistency -** Currency shall remain consistent from estimate through quotation to invoice.
- **FR 7.9 Pricing Recalculation -** System shall automatically recalculate totals when items are modified.

FR 8.0: Reporting and Analytics

- **FR 8.1** Reporting & Dashboards - The system shall provide project, task, and financial performance reports.
    - **FR 8.1.1** The dashboard shall display project status.
    - **FR 8.1.2** The dashboard shall display budget vs actual expenditures.
    - **FR 8.1.3** The dashboard shall display contractor ratings.
    - **FR 8.1.4** The dashboard shall make use of Gantt Chart, Pie Chart, Bar Graph and Line graph to display data.
    - **FR 8.1.5** Admins and Project Managers shall be able to view contractor ratings.
- **FR 8.2** AI Risk Analysis - AI shall identify potential project delays and risks.
- **FR 8.3** AI Maintenance Forecasting - AI shall predict future maintenance needs based on historical data.

FR 9.0: Security and Compliance

- **FR 9.1** Audit Logging - All critical actions shall be logged with timestamps.
    - **FR 9.1.2** Audit logs shall be immutable once created.
    - **FR 9.1.3** Admins shall be able to search and filter audit logs by date range, user, log type, and entity.

- **FR 9.2** Role-Based Data Visibility - Data visibility shall be restricted according to user roles.
- **FR 9.3** Escalation Handling - Admins shall intervene in stalled workflows or disputes.
- **FR 9.4** System Configuration - Admins shall configure global settings, notification templates, and retention policies.\

2.6.2. Non-Functional Requirements

The non-functional requirements (NFRs) define the quality attributes of the ICMM. While the functional requirements describe what the system should do, non-functional requirements define how the system should perform. These requirements ensure usability, security, scalability, and performance of the application.

NFR 1.0: Performance Requirements

- **NFR-1.1:** The system shall support a minimum of **500 concurrent users** without degradation in performance.
- **NFR-1.2:** Average page load time for the web application shall not exceed **5 seconds** under normal load.
- **NFR-1.3:** Mobile API responses shall return within **5 seconds** for 95% of requests.
- **NFR-1.4:** The AI-based blueprint analysis shall complete within **60 seconds** for plans up to 20MB in size.
- **NFR-1.5:** Database queries shall be optimised to execute within **2 seconds** for standard operations.
- **NFR-1.6**: Message delivery shall occur within 4 seconds for 95% of messages.

NFR 2.0: Reliability Requirements

- **NFR-2.1:** The system shall achieve **99.5% uptime** (excluding planned maintenance).
- **NFR-2.2:** All critical business transactions (quotations, invoices, payments, project updates) must be logged to prevent data loss.
- **NFR-2.3:** The system shall implement retry and logging mechanisms for failed notifications (SMS/Email).

NFR 3.0: Availability Requirements

- **NFR-3.1:** The web application shall be available 24/7 except during scheduled maintenance
- **NFR-3.2:** Planned downtime shall not exceed **4 hours per month** and must be communicated to stakeholders at least 24 hours in advance.

NFR 4.0: Security Requirements

- **NFR-4.1:** User authentication shall be implemented using **Firebase Authentication** with email and password.
- **NFR-4.2:** User passwords shall be managed by Firebase Authentication, which uses scrypt hashing algorithm with automatic salting.
- **NFR-4.3:** All communications between client and server shall be encrypted using **HTTPS/TLS 1.3**.
- **NFR-4.4:** Role-based access control (RBAC) shall enforce least-privilege principles.

- **NFR-4.5:** All sensitive actions (logins, document downloads, invoice updates) shall be recorded in an **audit log**.
- **NFR-4.6:** The system shall comply with **POPIA (Protection of Personal Information Act, South Africa)** and **GDPR (EU General Data Protection Regulation)**.
- **NFR-4.7**: Brute Force Protection – System shall implement login attempt tracking.

NFR 5.0: Usability Requirements

- **NFR-5.1:** The web interface shall be responsive and accessible on desktop, tablet, and mobile browsers.
- **NFR-5.2:** The mobile app shall support Android version **9 (Pie)** and above.
- **NFR-5.3:** All role-specific dashboards shall be intuitive and require **no more than 4 clicks** to access key tasks.
- **NFR-5.4:** The system shall comply with **WCAG 2.1 accessibility standards** for visually impaired users.
- **NFR-5.5:** Online help and tooltips shall be provided for all complex workflows.

NFR 6.0: Maintainability Requirements

- **NFR-6.1:** The system shall follow a **modular architecture** (separation of concerns between project management, maintenance, documents, AI, etc.).
- **NFR-6.2:** The source code shall follow **SOLID principles** and industry-standard naming conventions.
- **NFR-6.3:** Automated unit tests shall cover at least **70% of core business logic**.
- **NFR-6.4:** New developers should be able to onboard and contribute within **1 day**, aided by documentation.

NFR 7.0: Scalability Requirements

- **NFR-7.1:** The system shall support up to **10,000 registered users** without redesign.
- **NFR-7.2:** The messaging and notifications module shall handle **up to 50 messages per second**.

NFR 8.0: Portability & Compatibility Requirements

- **NFR-8.1:** The web application shall be compatible with latest versions of Chrome, Edge, Firefox, and Safari.
- **NFR-8.2:** The mobile application shall run on both **emulators and real devices**.
- **NFR-8.3:** The system shall support migration to different cloud providers (AWS, Azure, GCP) with minimal changes.

NFR 9.0: Compliance Requirements

- **NFR-9.1:** The system shall comply with **South African ICT regulations**, **POPIA**, and where applicable, **GDPR**.
- **NFR-9.2:** Audit logs must be retained for **at least 5 years**.
- **NFR-9.3:** All financial data shall comply with **IFRS (International Financial Reporting Standards)**.

NFR 10.0: File Storage Requirements

- **NFR-10.1:** Document Storage - All files shall be stored in Supabase Cloud Storage.
- **NFR-10.2**: Storage Buckets - Files shall be organized in separate storage buckets
- **NFR-10.5:** Storage Retention - Deleted files shall only be soft-deleted and retained indefinitely or until otherwise permanently deleted.

NFR 11.0: Push Notification Requirements

- **NFR-11.1**: Delivery Platform - Push notifications shall be delivered via Firebase Cloud Messaging.
- **NFR-11.3**: Notification Delivery - Push notifications shall be sent within 5 seconds of triggering event.

NFR 12.0: Data Validation Requirements

- NFR-12.1: Input Validation - All user inputs shall be validated before processing.

2.7. Summary: Requirements & UX Design

The functional and non-functional requirements together form the backbone of the ICMMS specification. The functional requirements define how modules such as Project Management, Quotation Processing, Document Storage, and Communication will perform in practice. Each requirement directly traces back to one or more user stories from Part 2, ensuring that all critical business objectives are implemented through measurable system functions.

Complementing this, the non-functional requirements capture the quality benchmarks that guarantee stability, reliability, and security under operational stress. These include explicit thresholds for performance, availability, scalability, usability, and legal compliance. In doing so, they ensure that ICMMS is not only feature-complete but also robust, efficient, and maintainable within real-world conditions. Together, the FRs and NFRs create the technical rulebook that guides every design, coding, and testing activity in later stages.

The following section, **Part 3 - Analysis and System Design**, builds directly upon these specifications. It converts the documented requirements into structured system models, data architectures, and secure design components that represent the ICMMS in its complete technical form.

Part 3 - Analysis & System Design

3.1. Introduction

Part 3 translates the requirements foundation established in the previous section into a structured system design that defines how the Integrated Construction and Maintenance Management System (ICMMS) will operate in practice.
This stage focuses on the underlying logic, data architecture, and security infrastructure that enable those behaviours to function reliably in production.

The analysis phase begins by modelling the problem domain to identify key entities and their relationships within bounded business contexts. These models guide the design of the database schema and inform the logical architecture used to connect the web application, API services, Firebase / Firestore data layer, and authentication framework.

3.2. Domain Modelling

The ICMMS domain model defines the logical structure of the system and how all entities interact within bounded business contexts. It represents the static relationships between key objects such as Users, Projects, Phases, Tasks, Quotations, Invoices, Payments, and Documents, showing how data flows across the construction and maintenance lifecycle.

As shown in Figure 3.1, the central entity is Project, which links directly to Phase, Task, and Document classes; reflecting the hierarchical structure used in real construction workflows. Each *Project* is created by a ProjectManager, associated with a specific Client, and executed through assigned Contractors. The Quotation and Invoice entities connect the financial lifecycle, referencing the same ProjectId for full traceability from cost estimation to payment. Maintenance is designed as the final phase of every project. If only a Maintenance request is required, an account is first needed with a project setup.

The User superclass defines shared attributes (e.g., UserId, Email, Role) inherited by the system's four role types; *Administrator*, *Project Manager*, *Contractor*, and *Client*. This inheritance model simplifies authentication logic and enables role-based permissions through a unified data schema.

Other supporting entities include MaintenanceRequest, which originates from the Client and links to both the *ProjectManager* and *Contractor*, and AuditLog, which records system-wide actions for accountability and compliance. Together, these relationships outline the real-world dependencies between administrative control, operational execution, and client oversight, forming the technical foundation for database and architectural design that follows.

UML Class Diagram

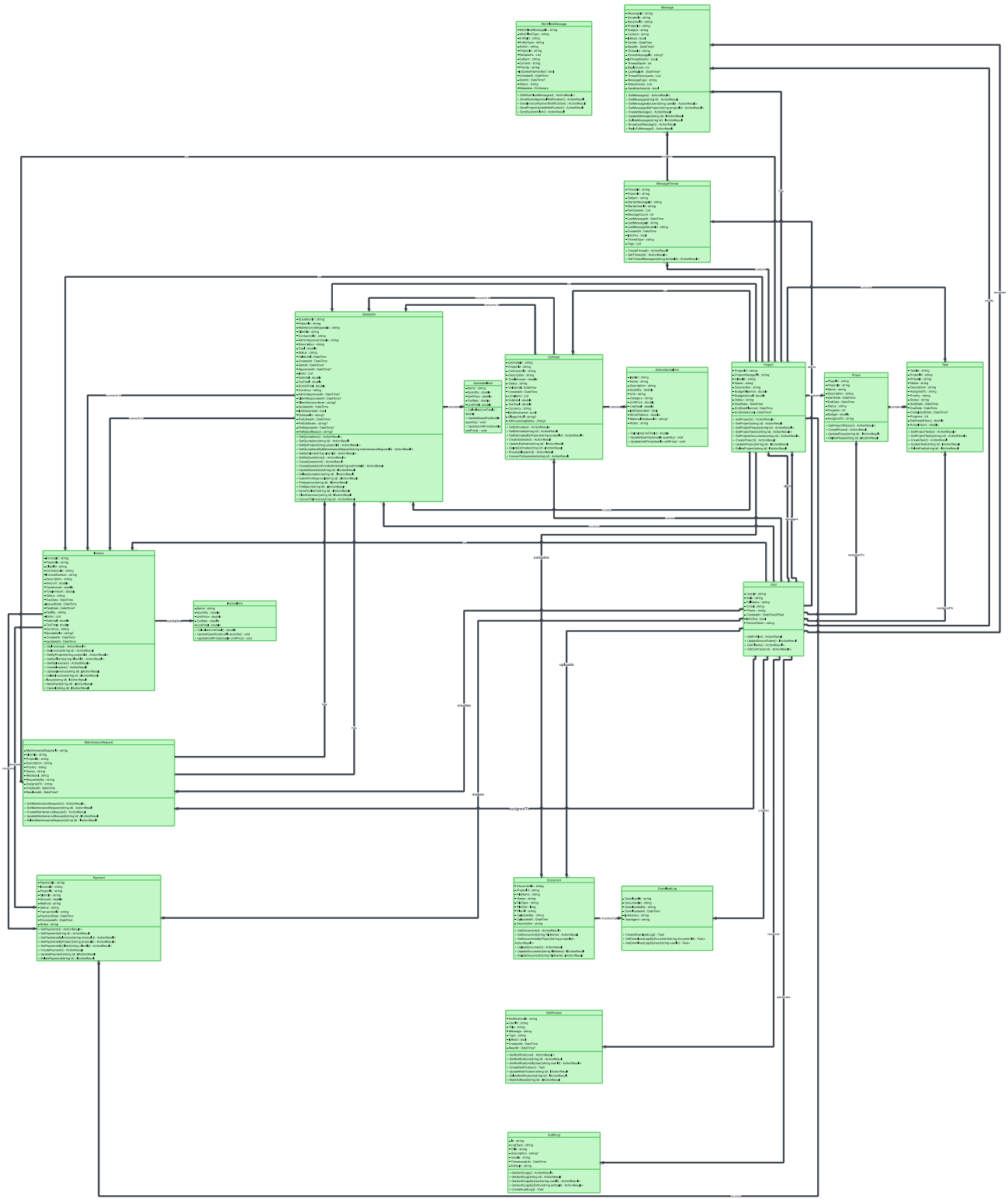


Figure 3.1 - UML Class and Domain Model. Source: Lucid Chart*: link here*

3.2.1. Bounded Context Diagram

The intention of this model is to separate the system's logic into clear business domains including preventing overlap, reducing data coupling, and improving maintainability across the full lifecycle of construction and maintenance operations.

As shown in Figure 3.2, the architecture is divided into four main contexts:

- **User and Access Management Context:** Handles authentication, authorisation, and user configuration. This context governs all role-based access and security policies managed through Firebase Authentication and the Admin module.
- **Project Lifecycle Management Context:** Represents the core of the system where projects are created, divided into phases, and monitored through task assignments, document uploads, and progress tracking. It integrates tightly with the Contractor and Client contexts via shared ProjectId references.
- **Quotation and Billing Context:** Manages cost estimation, quotation approval, invoicing, and payment tracking. It interacts directly with the Project Lifecycle context and references both Project Managers and Clients to maintain a full financial audit trail.
- **Maintenance and Support Context:** Focuses on client-submitted service or maintenance requests. Each request triggers a workflow linking the responsible Project Manager and assigned Contractor to ensure accountability and timely completion.

Bounded Context Diagram

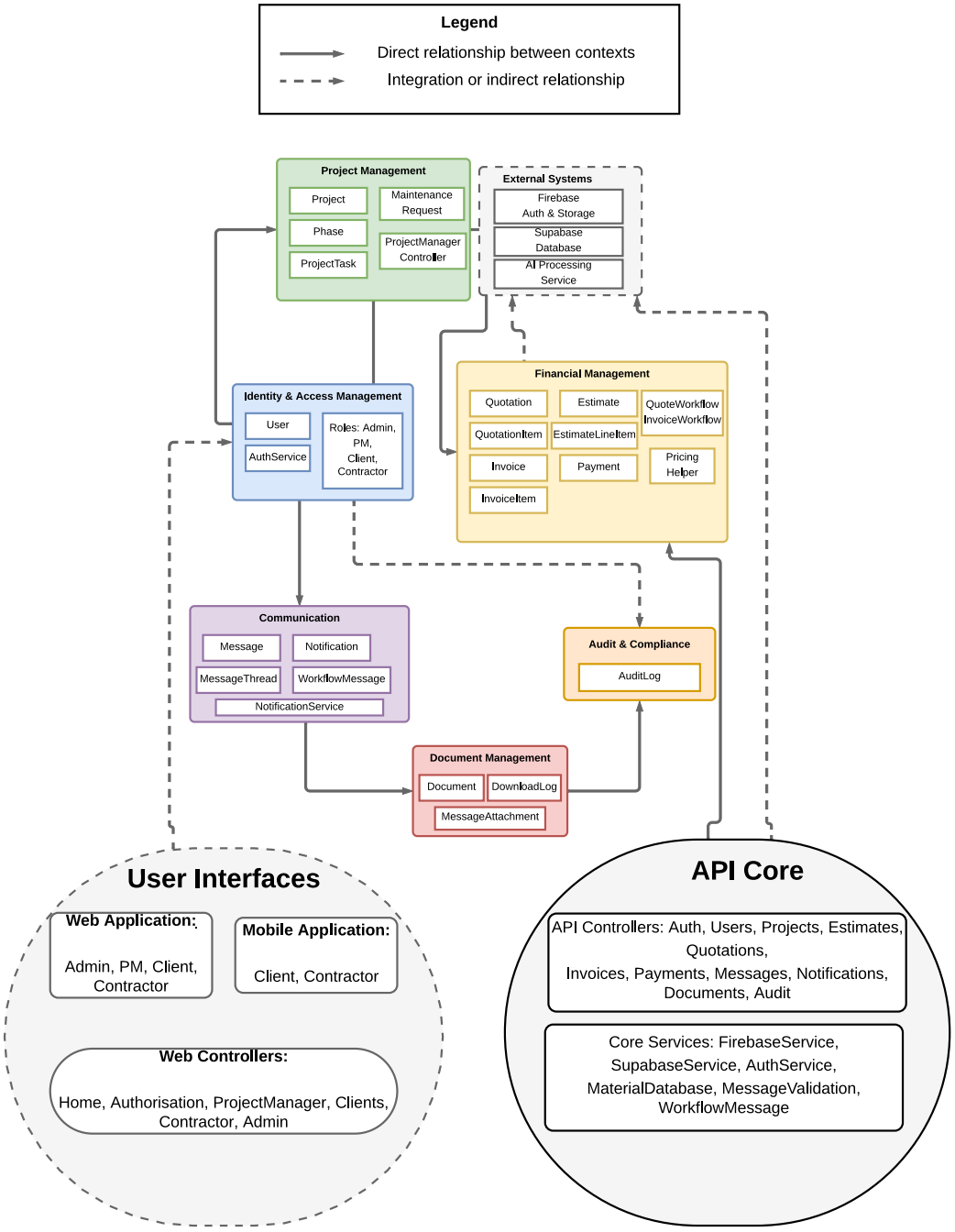


Figure 3.2 – Bounded Context Diagram. Source: Lucid Chart: link here

3.3. Entity–Relationship Model (ERD) and Database Schema

The Entity–Relationship Diagram (ERD) defines how project creation, task management, document control, and quotation processing interconnect to support all system operations

As illustrated in Figure 3.3, the core entities include User, Project, Phase, Task, Quotation, Invoice, Payment, and Document.

The *User* entity is associated with multiple *Projects* via the ProjectManagerId or ClientId foreign keys, while each *Project* cascades into multiple *Phases* and *Tasks*. This one-to-many relationship mirrors the practical structure of project execution workflows.

Financial processes are represented through *Quotation* and *Invoice*, both of which maintain direct references to their parent *Project* for traceability and reporting. The *Document* entity links to both *Project* and *Task*, capturing all uploaded reports, blueprints, and completion images. Additional utility entities; *Notification*, *AuditLog*, and *MaintenanceRequest*, support real-time communication, activity tracking, and maintenance scheduling within the system.

3.3.1. Entity–Relationship Model

ERD Class Diagram

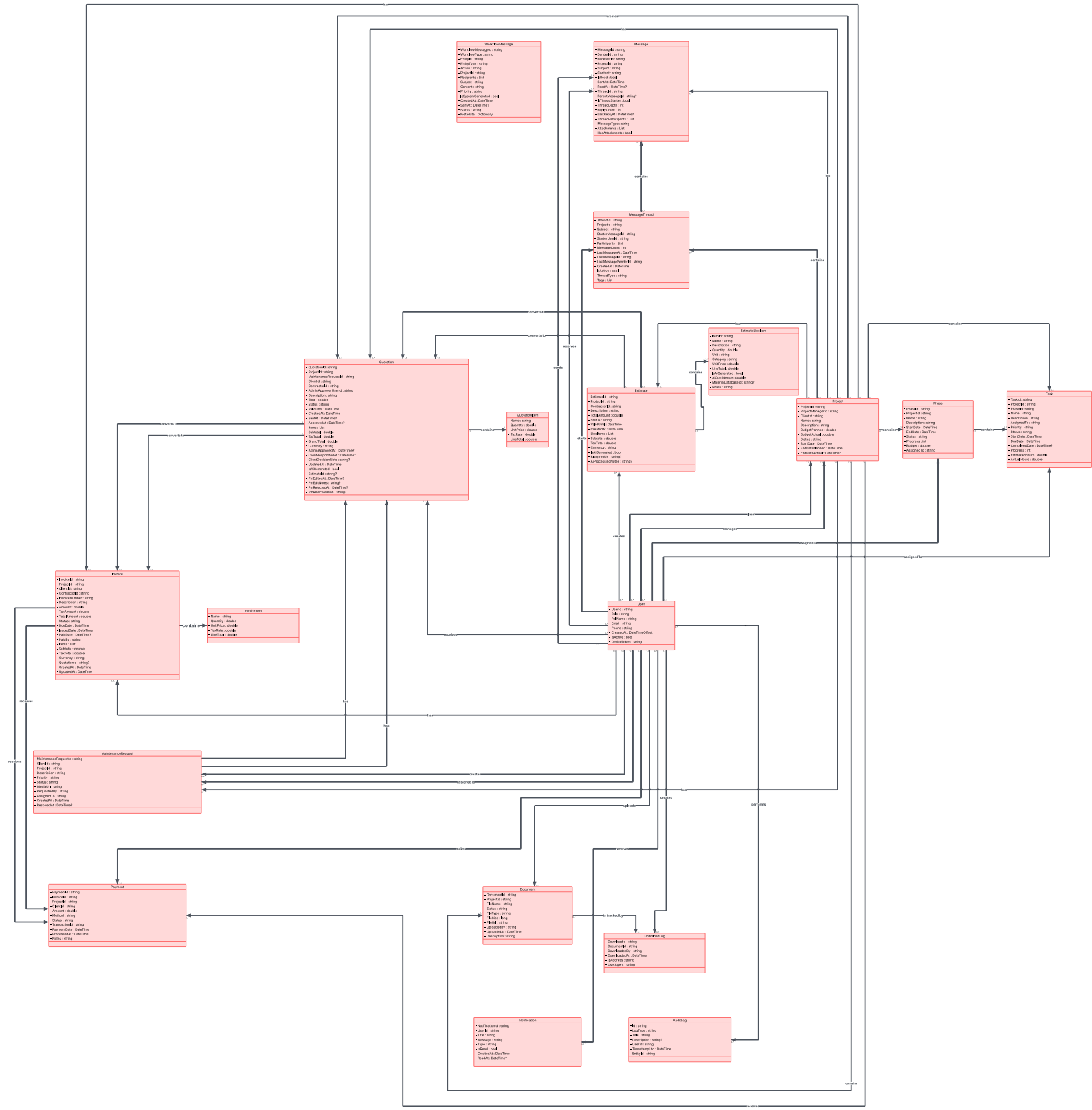


Figure 3.3 – Entity Relationship Diagram. Source: Lucid Chart: link here

3.3.2. Database Justification

During the design phase of the Integrated Construction and Maintenance Management System, we explored multiple database strategies, including a polyglot architecture combining SQL for structured, transactional data, and NoSQL for dynamic, operational data.
While the hybrid approach offers theoretical advantages in data integrity for financial records, our research identified several practical constraints that made this architecture less suitable for the scope and requirements of the project. Thus, the decision to go with only Firestore's NoSQL database was made.

Hosting and Cost Constraints:

Hosting a polyglot architecture solution would require additional infrastructure, configuration and maintenance. Additionally, many hosting solutions are catered for production grade SQL databases, which would incur recurring costs. (SQL, n.d.)

Complexity:

Hosting and maintaining two databases would result in complex backend logic to coordinate between the two databases and would pose synchronization challenges with additional security and access control layers.

Adequacy of Firestore:

While SQL naturally enforces schema and relational integrity (Teak, 2024), Firestore's flexibility allows us to model both structured and unstructured data into a single database. (Firebase, n.d.). We can enforce strict server-side data validation on our backend to ensure correctness without relying on a schema.

File Storage Strategy – Supabase Bucket

For storing and managing uploaded documents, we are using a Supabase Bucket. Supabase provides a scalable object storage that integrates seamlessly with our backend services. (Supabase, n.d.). This approach allows us to separate static file storage from dynamic application data in Firestore, ensuring efficient data retrieval and optimized bandwidth usage. Additionally, Supabase's authentication and control features align with our existing Firebase Authentication setup, mainting consistent security standard across both systems.

Seamless Integration into Tech Stack:

Firestore integrates natively with our current Tech Stack that already includes

- *Firebase Authentication*
- *Firebase Storage*
- *Supabase Bucket*
- *Kotlin*
- *ASP.NET (C#)*

Suitability

The system's goal is to demonstrate functionality and architectural principles, not to operate as a production-grade, compliance regulated financial system. A Firestore only architecture, complemented by Supabase for file storage, will allow us to focus on feature completeness, user experience, and maintainable code.

3.4. System Architecture Design

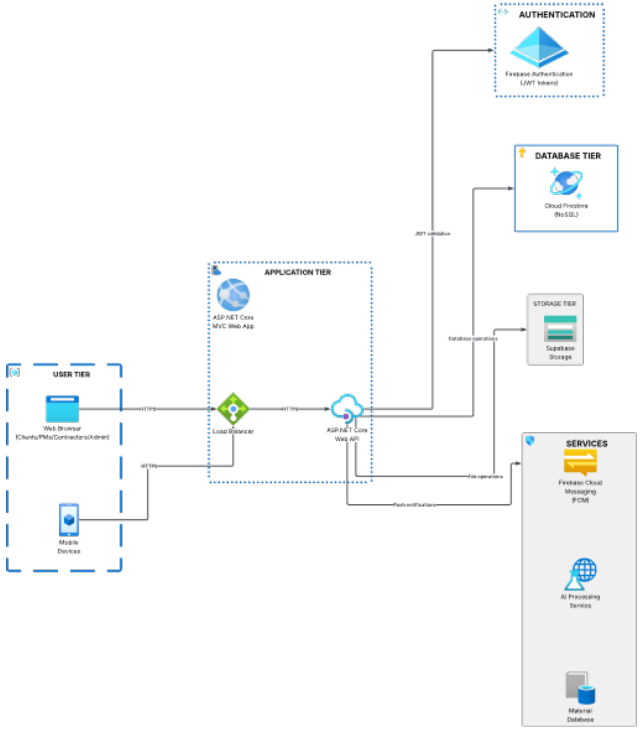
3.4.1. Logical Architecture

The logical architecture, illustrated in Figure 3.4, outlines the data and control flow across all major components. The core system comprises three primary layers:

- **Presentation Layer:** Web App (ASP.NET MVC) and Mobile App (Android Kotlin) provide role-specific dashboards and CRUD interfaces.
- **Application Layer (API):** The .NET 6 Web API exposes endpoints for Projects, Users, Quotations, and Documents. It enforces validation, authentication, and audit logging.
- **Data Layer:** Firestore (NoSQL) stores application data; Supabase Buckets handle static file storage; Firebase Auth secures login sessions; and Azure App Service hosts the backend and web frontend.

Data flows from the client layer to the API via HTTPS, authenticated by Firebase Auth tokens, then routed to Firestore for persistence.
This multi-tier approach isolates business logic, promotes scalability, and enables independent updates without downtime.

System Overview Diagram



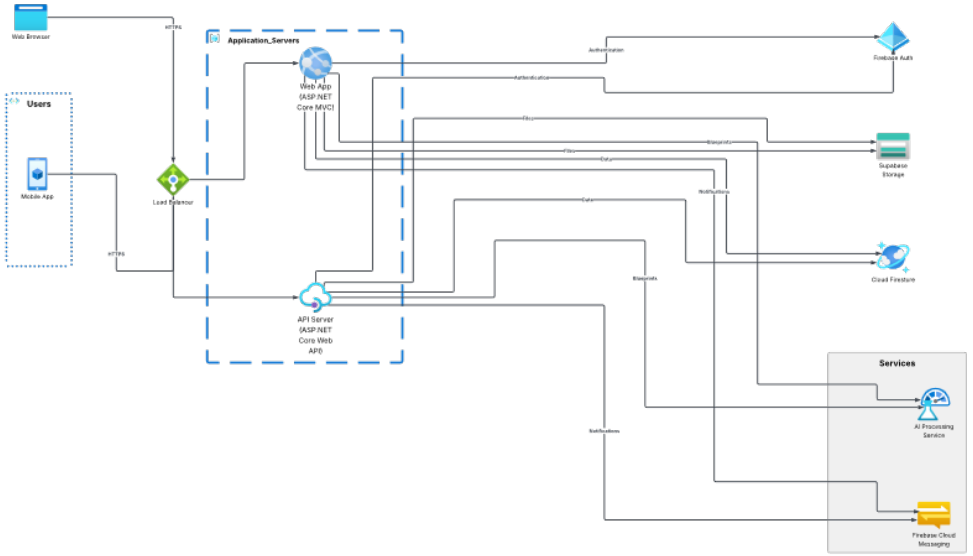


Figure 3.4 – System Overview Diagram. Source: Lucid Chart*:* link here

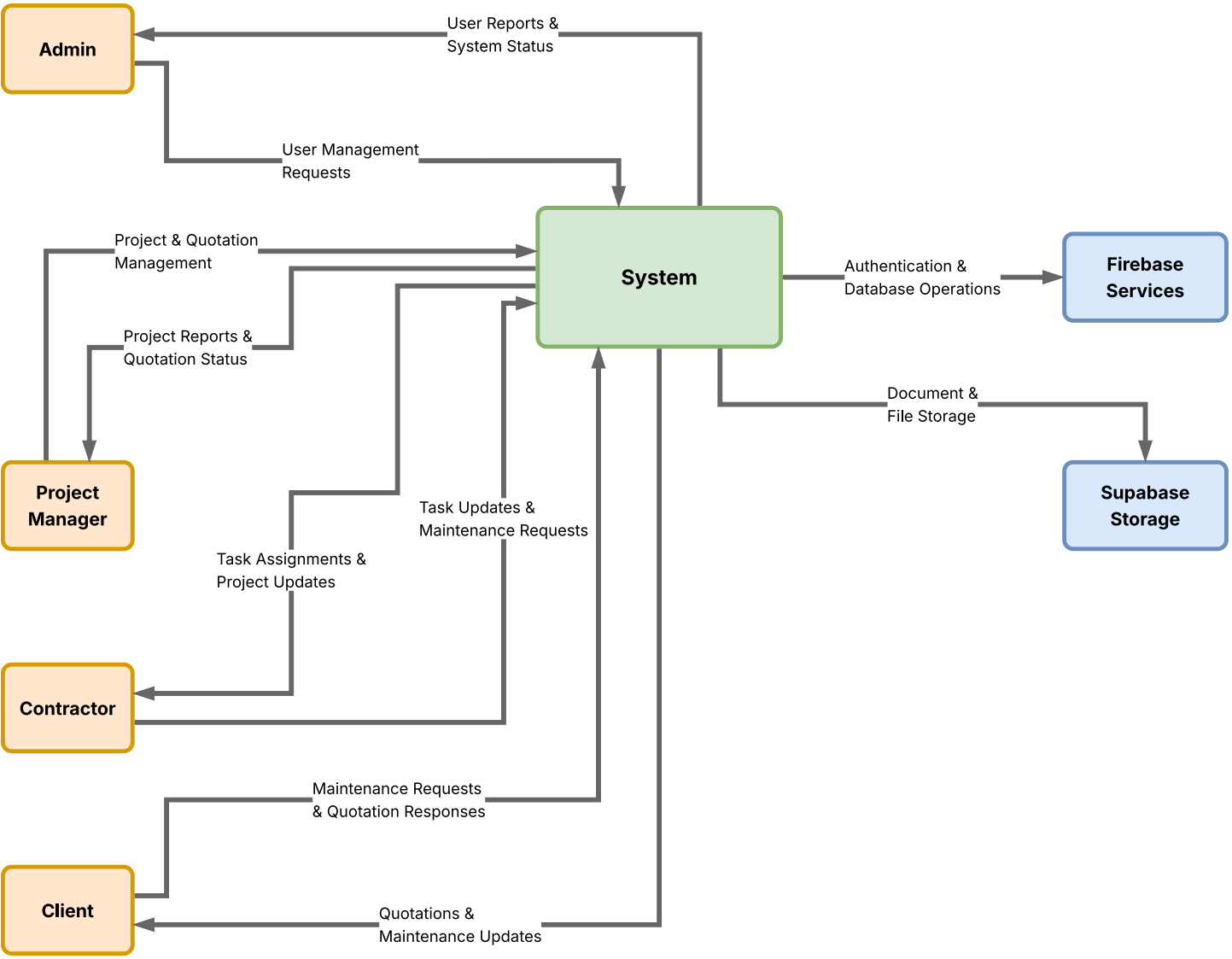
3.4.2 Data Flow Architecture

The supporting data-flow models expand the logical architecture into granular process views:

- **Level 0:** Context Data Flow Diagram (Figure 3.5) shows external entities (Admin, Project Manager, Contractor, Client) interacting with the system through secure API requests.
- **Level 1:** Authentication and User Management DFD (Figure 3.6) details credential validation via Firebase Auth and user-role retrieval from Firestore.
- **Level 1:** Quotation and Invoice Management DFD (Figure 3.7) models quote creation, approval, and invoice generation between Project Manager, Admin, and Client roles.

These flows ensure clear traceability of data movement and process responsibilities across all system components.

Context Level Data Flow Diagram (Level 0)



Notes:
- ISystem includes Web App, API, and Mobile App components
- Firebase handles authentication and Firestore database
- Supabase provides document and file storage services
- All data flows are bidirectional with appropriate security controls

Figure 3.5 – Context Level Data Flow Diagram Source: ***Lucid Chart***: ***link here***

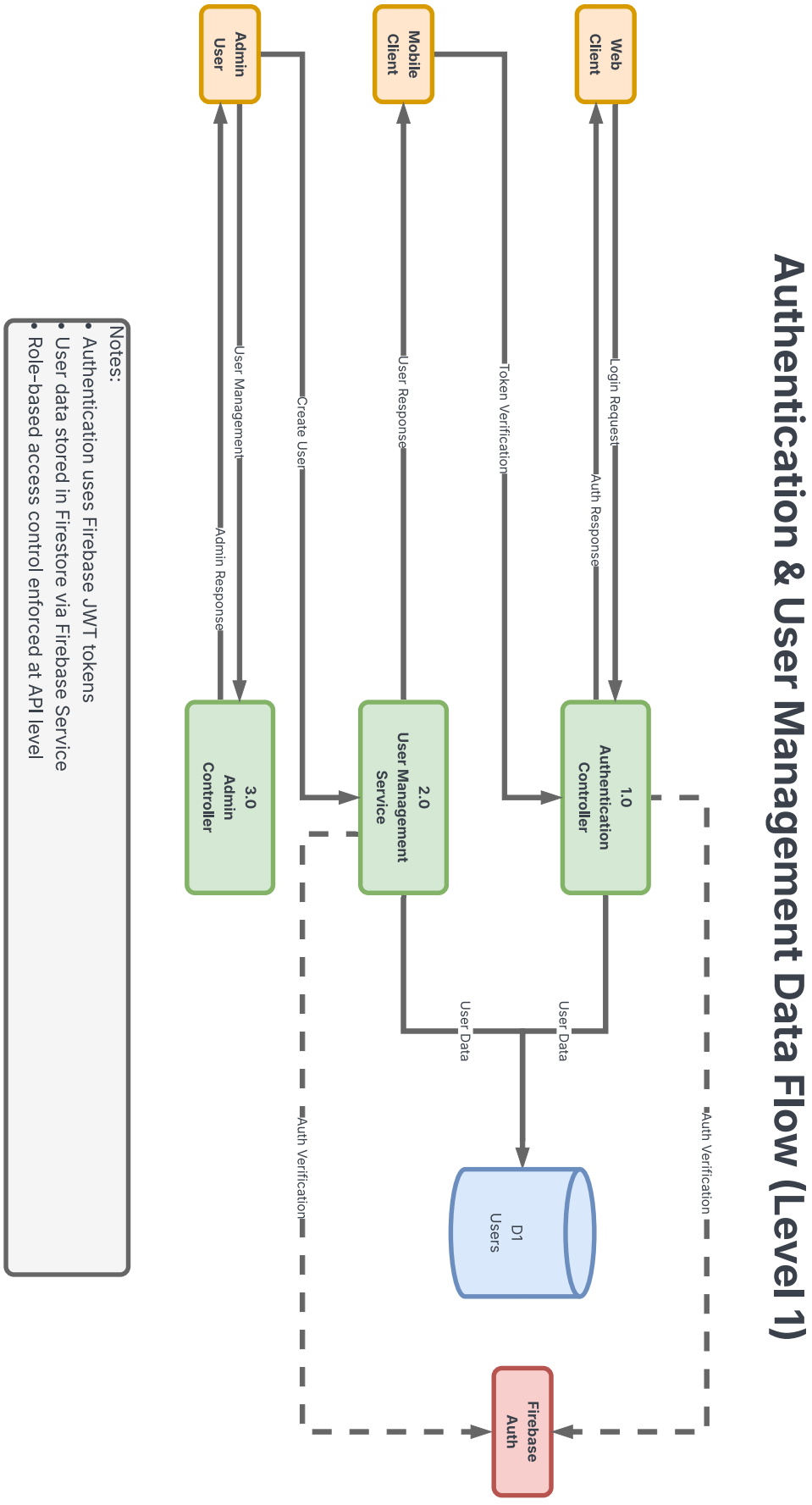


Figure 3.6 – Authentication and User Management Data Flow (Level 1) Source: ***Lucid Chart***: ***link here***

Quotation & Invoice Management Data Flow (Level 1)

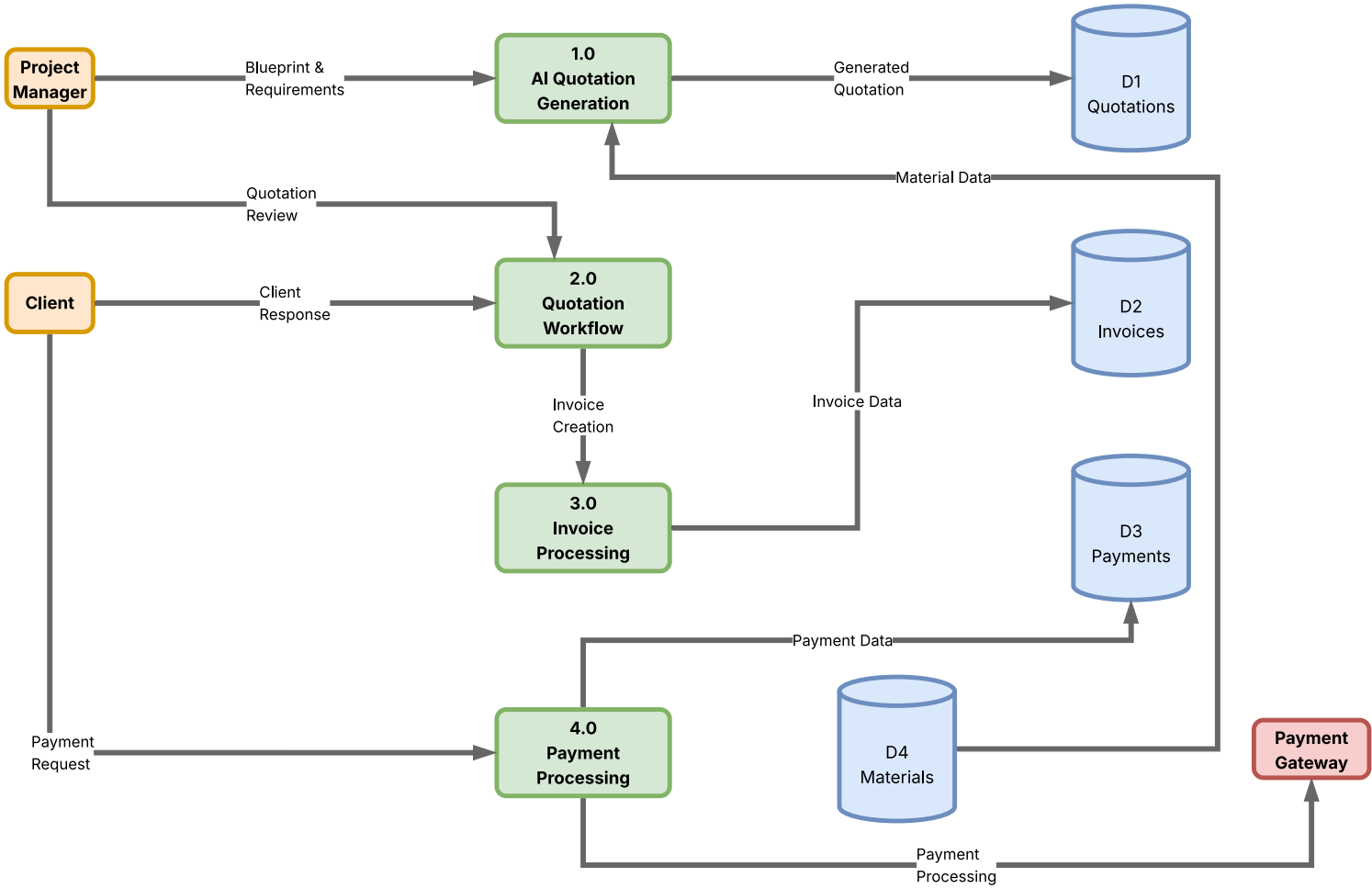


Figure 3.7 – Quotation and Invoice Management Data Flow (Level 1) Source: ***Lucid Chart***: ***link here***

3.4.3 Physical Deployment Architecture

Prerequisites

Azure Account

- Azure subscription
- Resource Group
- App Service Plan
- App Service
- Web App

Firebase Setup

- Create a Firebase project
- Link Google Cloud project
- Enable the following services
  1. Authentication: Go to Authentication -> sign-in method -> enable Email/Password & Google
  2. Firestore: Go to Firestore Database -> Create Database -> Start in test mode
  3. App Check: Go to App Check -> Register apps
- Generate Configuration Files
  1. Go to Project Settings -> General -> Your apps
  2. Add Android app -> download 'google-services.json' and place in app directory.
  3. Add Web app -> copy config object -> use in web app.

Development Environment

Required downloads

- .NET SDK 6.0+
- Azure CLI
- Git
- Visual Studio / VS Code
- Android Studio

3.4.2. Deployment Processes

Backend API Deployment

1. Build and Deploy
   - Navigate to API project
     `cd API/ICCMS-API`
   - Build the application
     `dotnet publish --configuration Release --output ./publish`
   - # Create deployment package
     `Compress-Archive -Path "./publish/*" -DestinationPath "./publish.zip" -Force`
   - Deploy to Azure
     `az webapp deployment source config-zip --name ICCMS-API --resource-group abcretailers_group --src ./publish.zip`
2. Restart App Service
   - Restart to apply runtime changes
     `az webapp restart --name ICCMS-API --resource-group abcretailers_group`

Alternative:
Run ./deployAPI bat script from project root

Web Frontend Deployment

- Navigate to Web project
  ` cd Web\ICCMS-Web`
- Build the application
  ` dotnet publish --configuration Release --output ./publish`
- # Create deployment package
  `Compress-Archive -Path "./publish/*" -DestinationPath "./publish.zip" -Force`
- Deploy to Azure
  ` az webapp deployment source config-zip --name icmms --resource-group abcretailers_group --src ./publish.zip`

Alternative:

Run ./deployWeb bat script from project root

Mobile App Build Process

Requirements:

- Android Studio
- ‘google-services.json’ file in app directory
- Proper API endpoint configuration pointing to Cloud Run API

Build steps:

```
# Generate debug APK for testing
cd Mobile
./gradlew assembleDebug
# APK location: app/build/outputs/apk/debug/app-debug.apk

# Generate release APK for distribution
./gradlew assembleRelease
# APK location: app/build/outputs/apk/release/app-release.apk
```

Installation:

- Transfer APK to Android device
- Enable “Install from unknown sources”
- Install APK directly
  OR
- Can install directly if device is connected using

```
adb install app-debug.apk
```

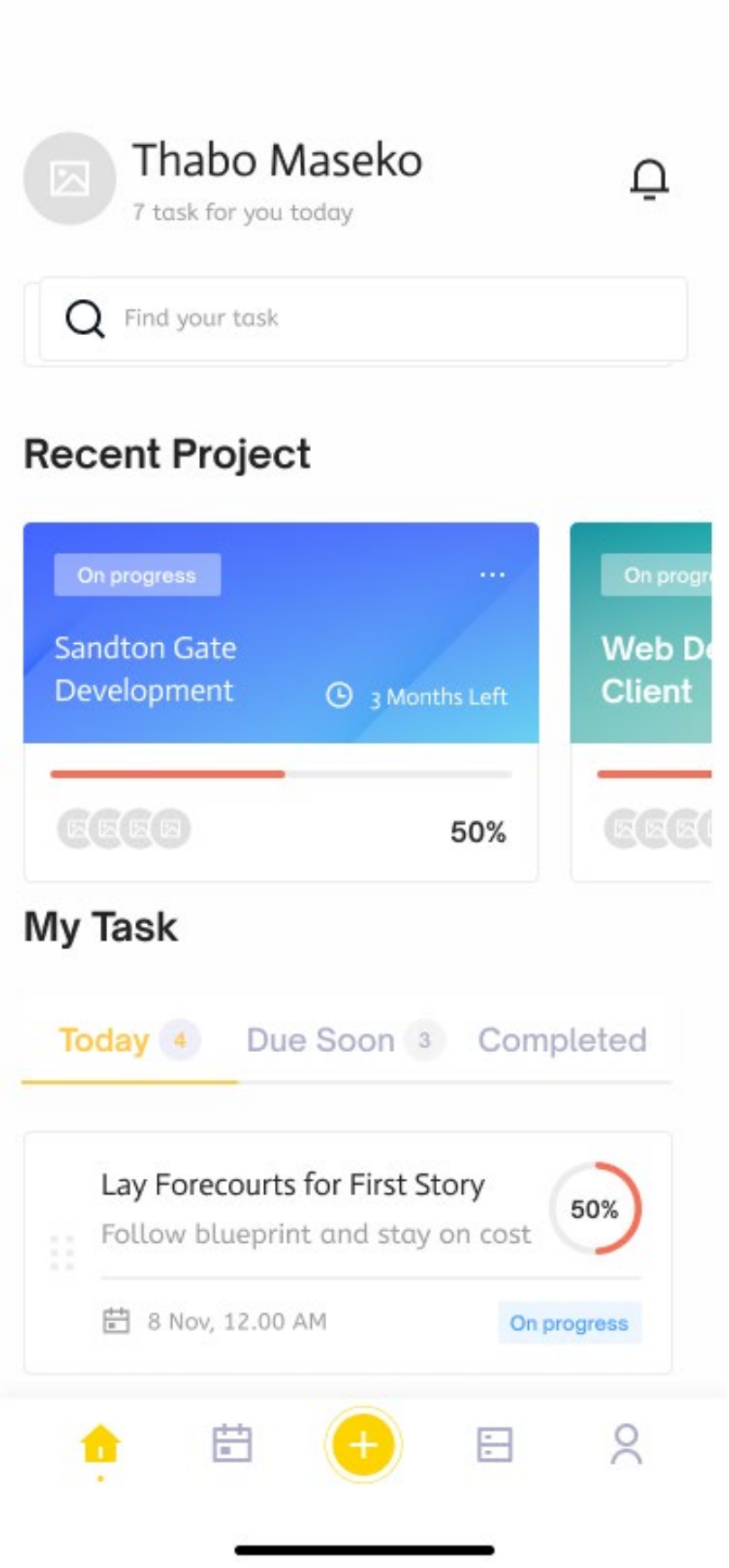
3.5. Wireframes and UI Mockups

The following wireframes visualise the Contractor Mobile Interface of the Integrated Construction and Maintenance Management System (ICMMS). They illustrate the user experience from a functional perspective. How daily tasks, project updates, and communication are delivered through a responsive and intuitive design.

The Contractor role was chosen for detailed mock-ups because it represents the most active operational user within the platform. Contractors interact with multiple system modules: task tracking, schedule management, document uploads, and real-time notifications. The mobile layout ensures accessibility on-site, where field workers require fast status updates and minimal navigation overhead.

Each screen maintains consistent design language, using standard spacing, high-contrast typography, and a clear bottom navigation bar to reinforce familiarity across views. Visual feedback (progress bars, task chips, and colour states) provides at-a-glance understanding of project progress.

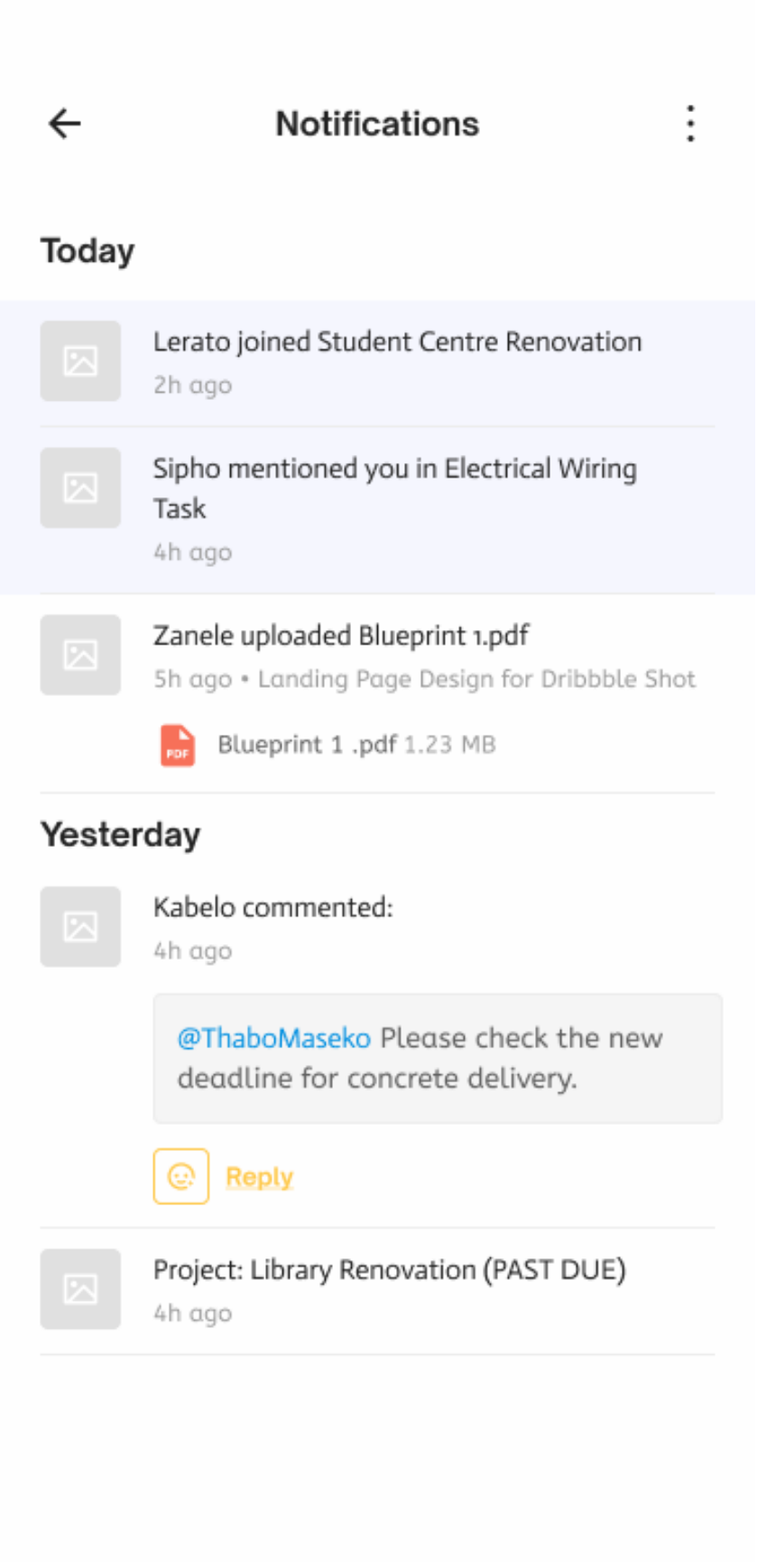
3.5.1 Home Screen



The Home Screen displays an overview of active projects, upcoming tasks, and progress metrics. Dynamic widgets summarise deadlines, completion percentages, and task priorities. The top search bar enables quick access to specific jobs or clients.

Figure 3.8 – Contractor Home Page Mock-up

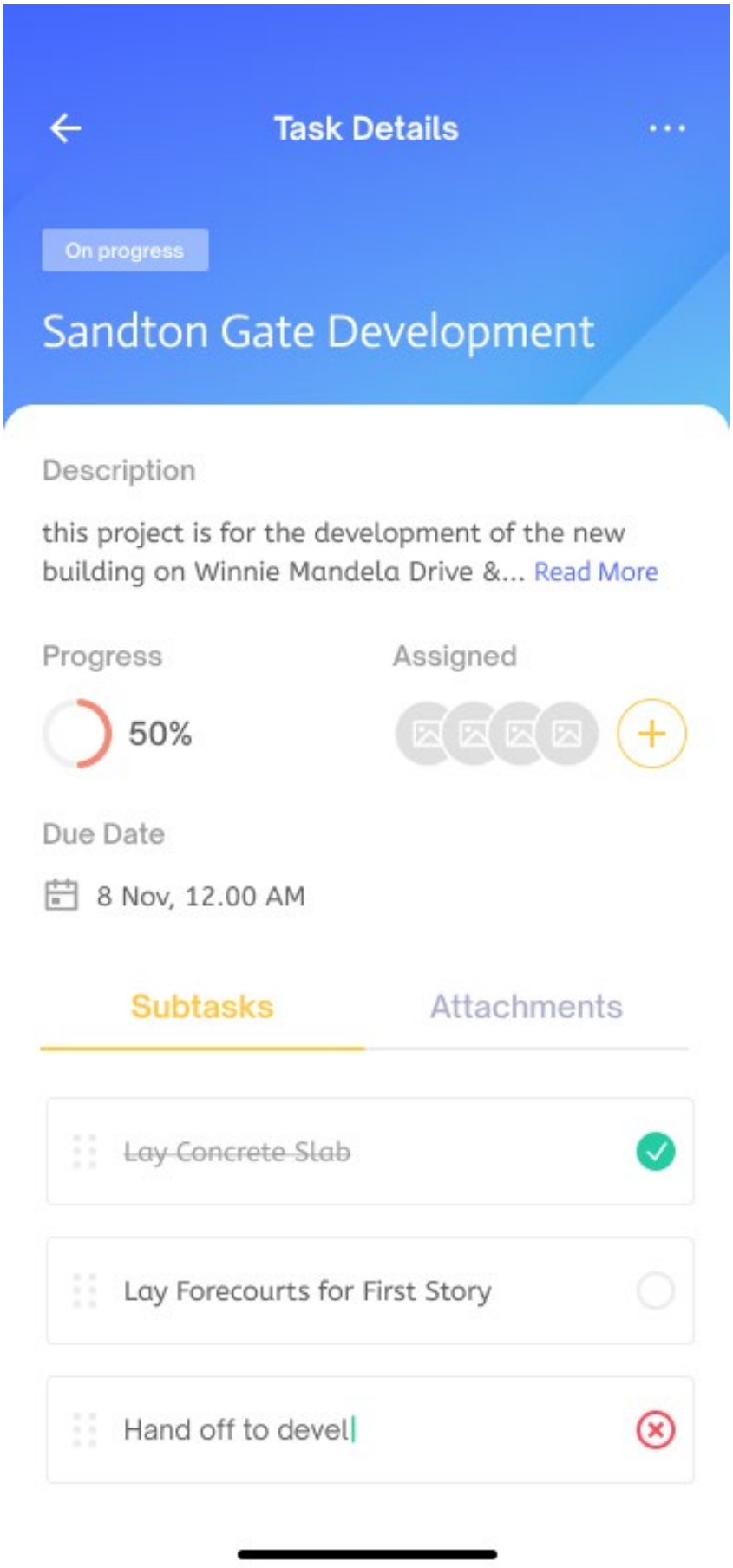
3.5.2 Notifications Screen



The Notifications interface aggregates all real-time system alerts; new task assignments, document uploads, and project mentions. Grouped by "Today" and "Yesterday," it helps contractors respond immediately to time-critical updates while maintaining a chronological activity log.

Figure 3.9 – Contractor Notifications Page Mock-up

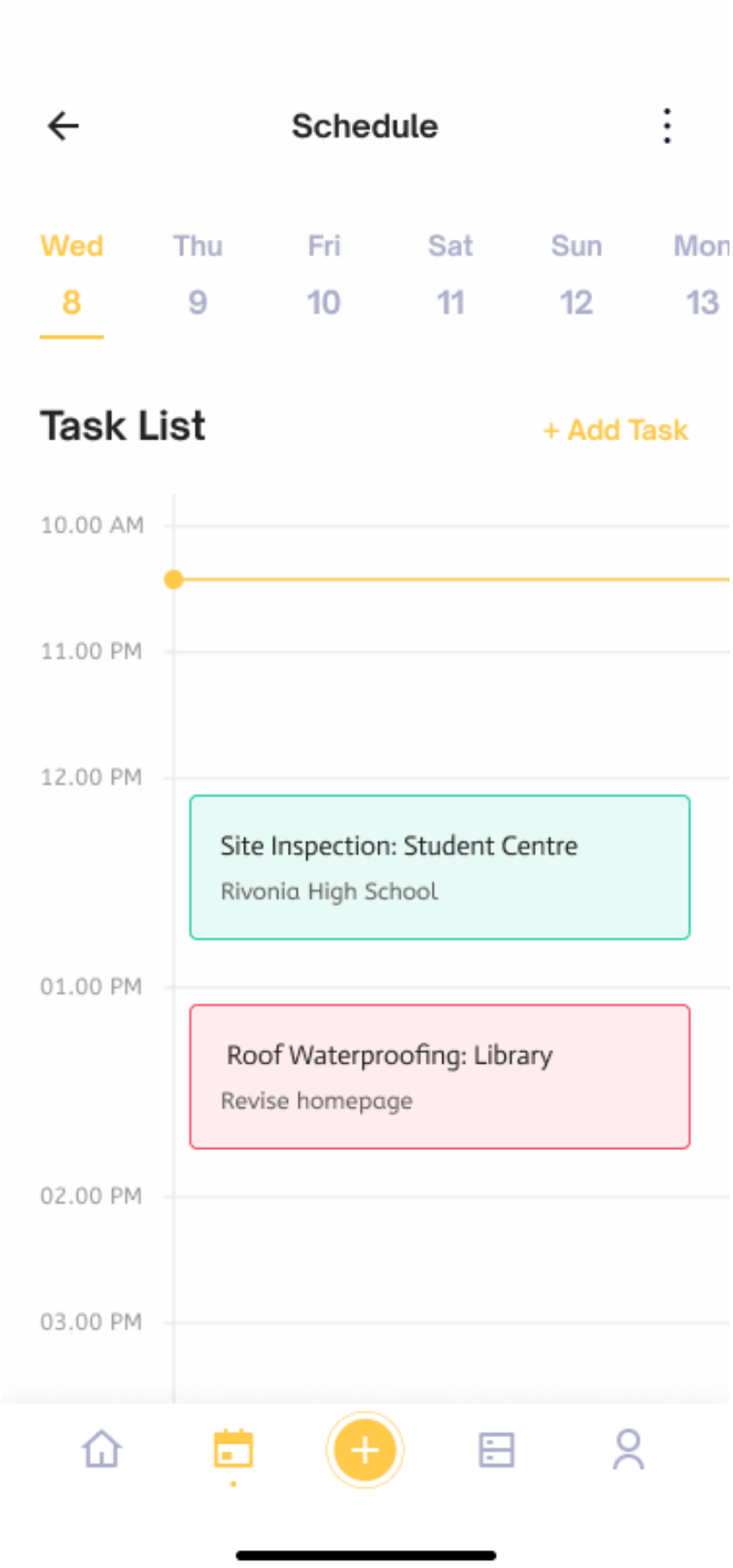
3.5.3 Task Details Screen



The Task Details view provides comprehensive insight into a specific job, including description, due date, progress, and subtasks. Users can mark individual subtasks as complete and attach photos or PDFs directly from the mobile interface; feeding data to Firestore in real time.

Figure 3.10 – Contractor Task Details Page Mock-up

3.5.4 Schedule Screen



The Schedule view integrates all assigned activities into a daily calendar timeline. Colour-coded tasks differentiate project categories (e.g., inspections vs. waterproofing). This interface supports intuitive drag-and-drop rescheduling and integrates with push notifications for reminders.

Figure 3.11 – Contractor Schedule Page Mock-up

3.5.5 Profile Screen

The Profile screen consolidates performance metrics; total projects, ongoing work, and completed tasks. It also serves as an entry point to manage user credentials, privacy preferences, and quick navigation to assigned projects.

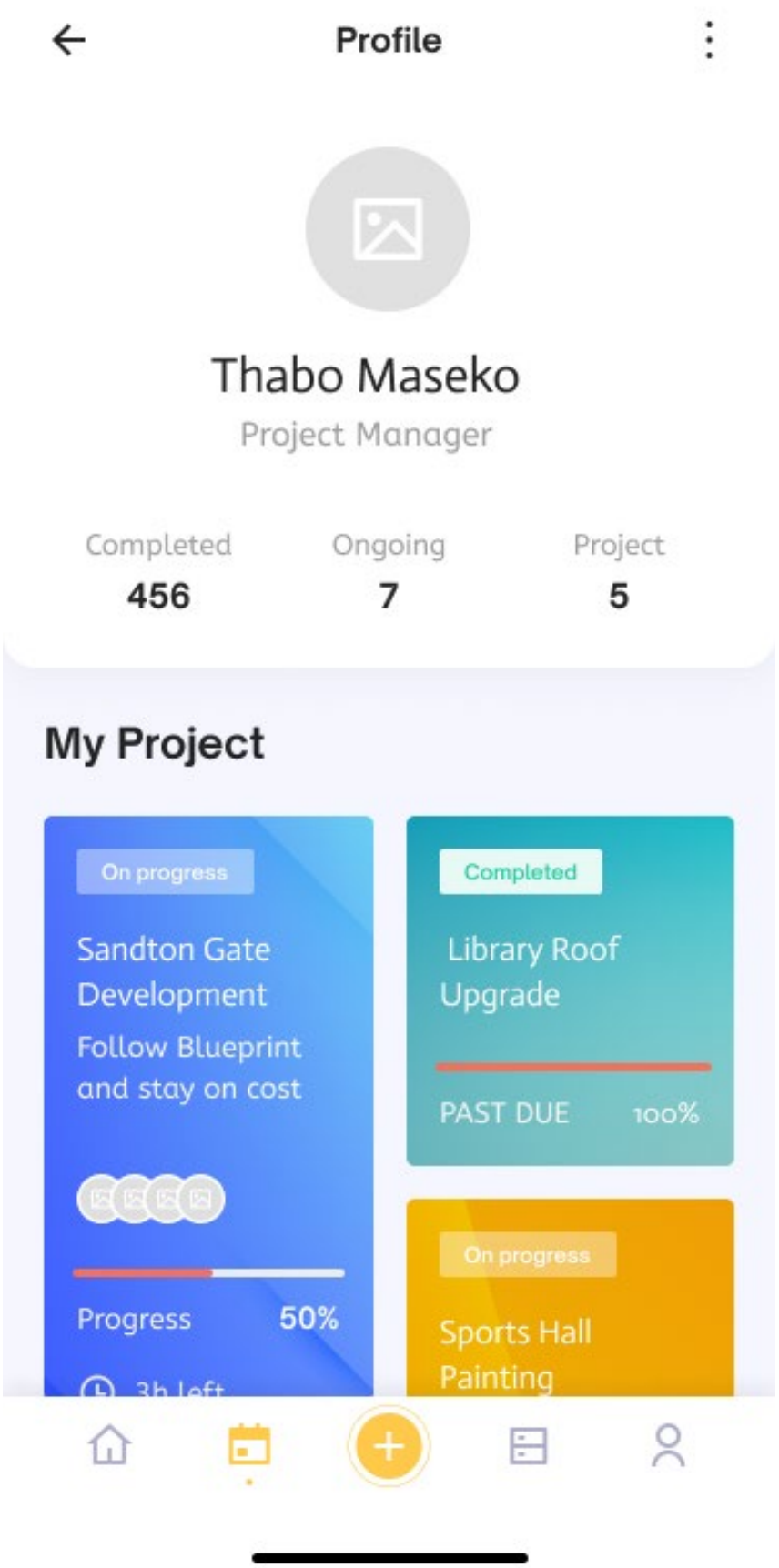


Figure 3.12 – Contractor Profile Page Mock-up

3.6. Security Design

The ICMMS implements a multi-layered security architecture that protects data integrity, enforces role-based access, and ensures full accountability across the application stack. Security is embedded from authentication through data transmission and storage, using industry-standard protocols and cloud-native controls from Firebase, Firestore, and Azure.

3.6.1 Authentication & Authorisation

User authentication and authorisation are handled through Firebase Authentication, which validates credentials, generates secure ID tokens, and maps each login session to its corresponding role within the Firestore user document.
As shown in Figure 3.6 – Authentication and User Management Data Flow (Level 1) each login request from the web or mobile client is transmitted via HTTPS to the backend API. The API verifies the token with Firebase, retrieves the user's Role, and enforces access rules at the controller level.
Only verified tokens can access endpoints; unauthorised or expired sessions are immediately rejected. This guarantees that Administrators, Project Managers, Contractors, and Clients operate strictly within their assigned permissions, preventing privilege escalation or unauthorised CRUD actions.

3.6.2 Secure Data Flow & Transmission

Data movement within the system follows tightly controlled pathways illustrated in Figure 3.5 – Context Level Data Flow Diagram. All communication between the front-end, API, and Firestore occurs over TLS-encrypted channels. Requests are validated by the API gateway before being routed to the corresponding microservice.
Document uploads and large media files are offloaded to Supabase Buckets, ensuring that static file storage remains isolated from dynamic Firestore data collections. This reduces exposure to injection or data-tampering risks while maintaining efficient retrieval performance.

3.6.3 Transaction and Workflow Integrity

Financial and operational workflows rely on multiple verification points to maintain transactional integrity. As detailed in Figure 3.7 – Quotation and Invoice Management Data Flow (Level 1), quotation approvals and invoice generations are authenticated through both Project Manager and Client sessions.
All approval actions generate immutable entries in the AuditLog collection, which captures UserId, ActionType, and Timestamp. These logs form a complete, non-repudiable trail of every modification made within the system, reinforcing transparency and compliance.

3.6.4 Threat Mitigation & Data Protection

The ICMMS applies proactive defences against common attack vectors:

- Injection prevention: Input validation and parameterised queries on all API endpoints.
- Cross-site request forgery (CSRF): Token-based session verification in client-server communications.
- Access control enforcement: Role-based middleware ensures least-privilege operations.
- Audit logging and monitoring: Every data change and login attempt is logged for anomaly detection.
- Secure storage: Firestore documents and Supabase files are encrypted at rest and transmitted over SSL.

This layered strategy minimises breach probability while maintaining system performance and user experience**.**

3.7. Summary: Analysis & System Design

Part 3 established the complete technical foundation of the Integrated Construction and Maintenance Management System (ICMMS). It transformed the project's requirements into a practical, scalable design through structured modelling, architecture definition, and secure data management. The domain model and bounded context diagrams clarified system boundaries and relationships between entities such as Users, Projects, Tasks, and Quotations, while the ERD and database justification confirmed the choice of Firestore's NoSQL architecture supported by Supabase for document storage.

The logical and physical architecture diagrams demonstrated how the web, mobile, and API layers integrate through Firebase Authentication and Azure cloud services to deliver a unified operational platform. Data-flow diagrams traced secure interactions between modules, ensuring consistent communication and traceability. Wireframes and mobile mock-ups then translated these technical designs into functional user interfaces that prioritise clarity, responsiveness, and role-specific efficiency. The section concluded with a robust security design that defined authentication, authorisation, and threat-mitigation strategies.

Part 4 - Implementation Documentation

4.1. Introduction

Part 4 demonstrates how the Integrated Construction and Maintenance Management System (ICMMS) was implemented, deployed, and maintained through structured coding, automation, and documentation practices.

4.2. UML Sequence Diagrams

The sequence diagrams illustrate how each major feature of ICMMS executes end-to-end through coordinated service calls between users, the web application, the API layer, and the database.

- **Figure 4.1- Project and Task Assignment Flow: (Figure 4.1)** This diagram visualises how a Project Manager creates a project, assigns tasks to contractors, and how acknowledgments are relayed via the API and Firebase event triggers. It highlights asynchronous notifications and secure role-based data exchange.
- **Figure 4.2 - Quote-to-Cash Workflow: (Figure 4.2)** Demonstrates the quotation approval lifecycle, from AI-generated estimates to invoice creation and payment confirmation. It captures how financial logic synchronises across Firestore and Supabase storage.
- **Figure 4.3 - Document Control and Access: (Figure 4.3)** Outlines secure file handling between Admin, Project Managers, and Clients. Access tokens and metadata validation ensure role-based file visibility.
- **Figure 4.4 - Maintenance Request Sequence: (Figure 4.4)** Shows the Client–PM–Contractor workflow where requests move through statuses (Pending → In Progress → Completed), automatically updating the project dashboard and notification system.
- **Figure 4.5 - Invoice Reminder and Dunning: (Figure 4.5)** Describes automated notification triggers for overdue invoices, integrating Firestore scheduling with email alerting.
- **Figure 4.6 - Contractor Task Sequence Diagram**: **(Figure 4.6)** visualises how task data is retrieved, updated, and synchronised across the platform in real time. The sequence begins when a Contractor logs into the mobile interface and retrieves a list of assigned tasks from Firestore through the API. Each task entry includes key metadata such as project ID, deadline, priority level, and linked documentation.

Together, these artefacts confirm that the implemented workflows mirror the functional intentions set out in the requirements phase.

Project & Task Assignment Sequence Diagram

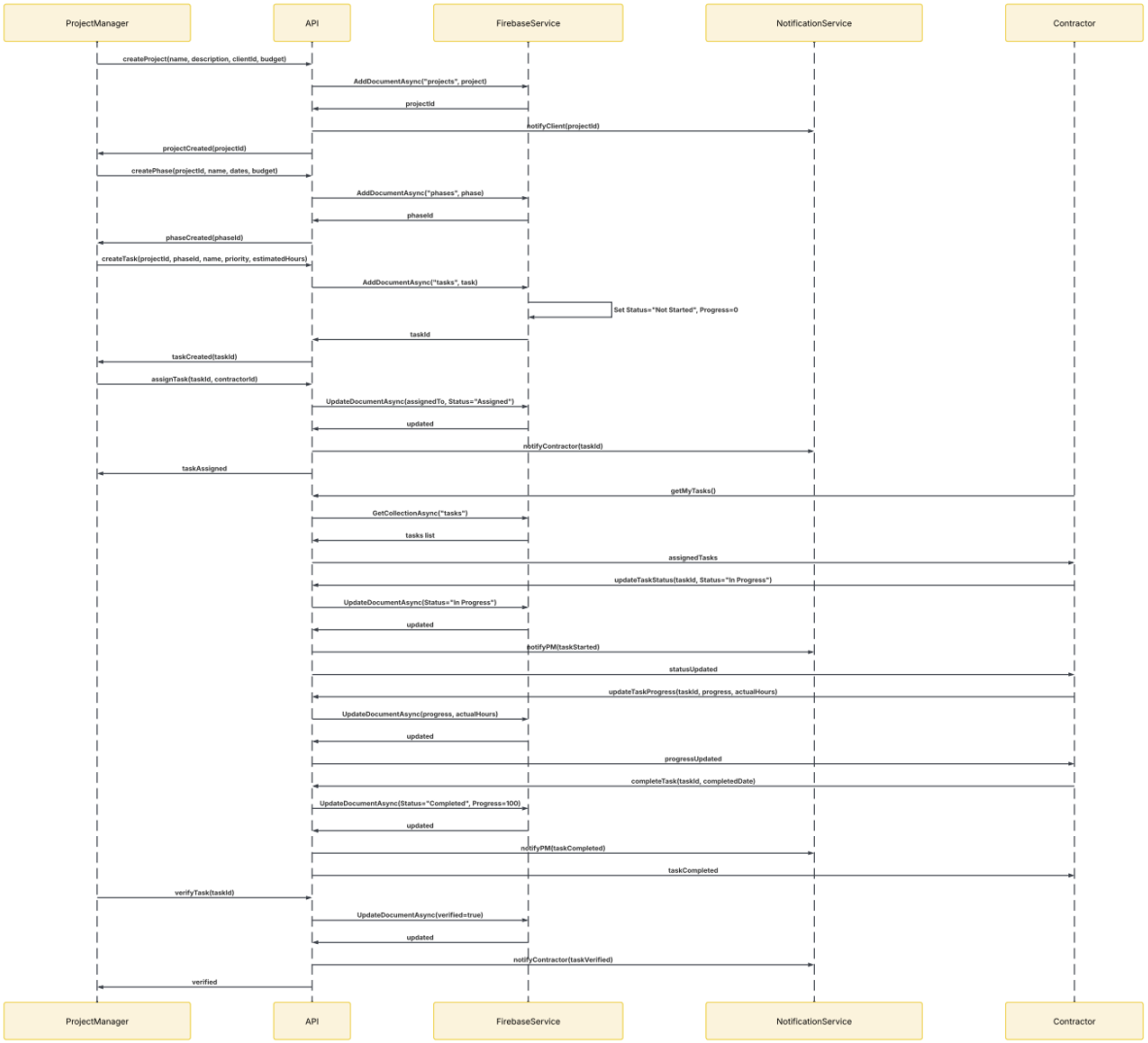


Figure 4.1 – Project & Task Assignment Sequence Diagram Source: link here

Quote to Cash Sequence Diagram

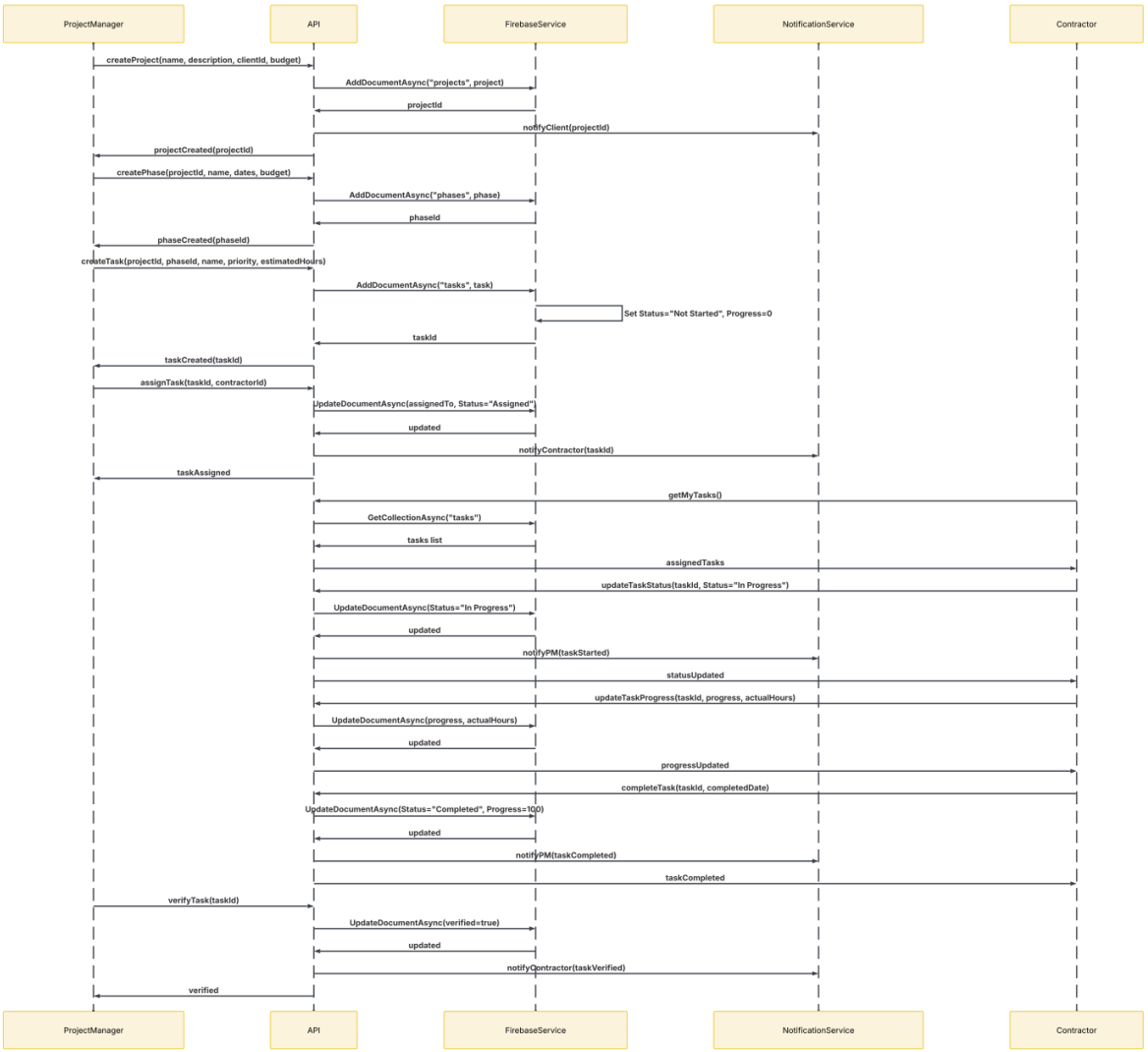


Figure 4.2 – Quote to Cash Sequence Diagram. Source: link here

Document Control & Access Sequence Diagram

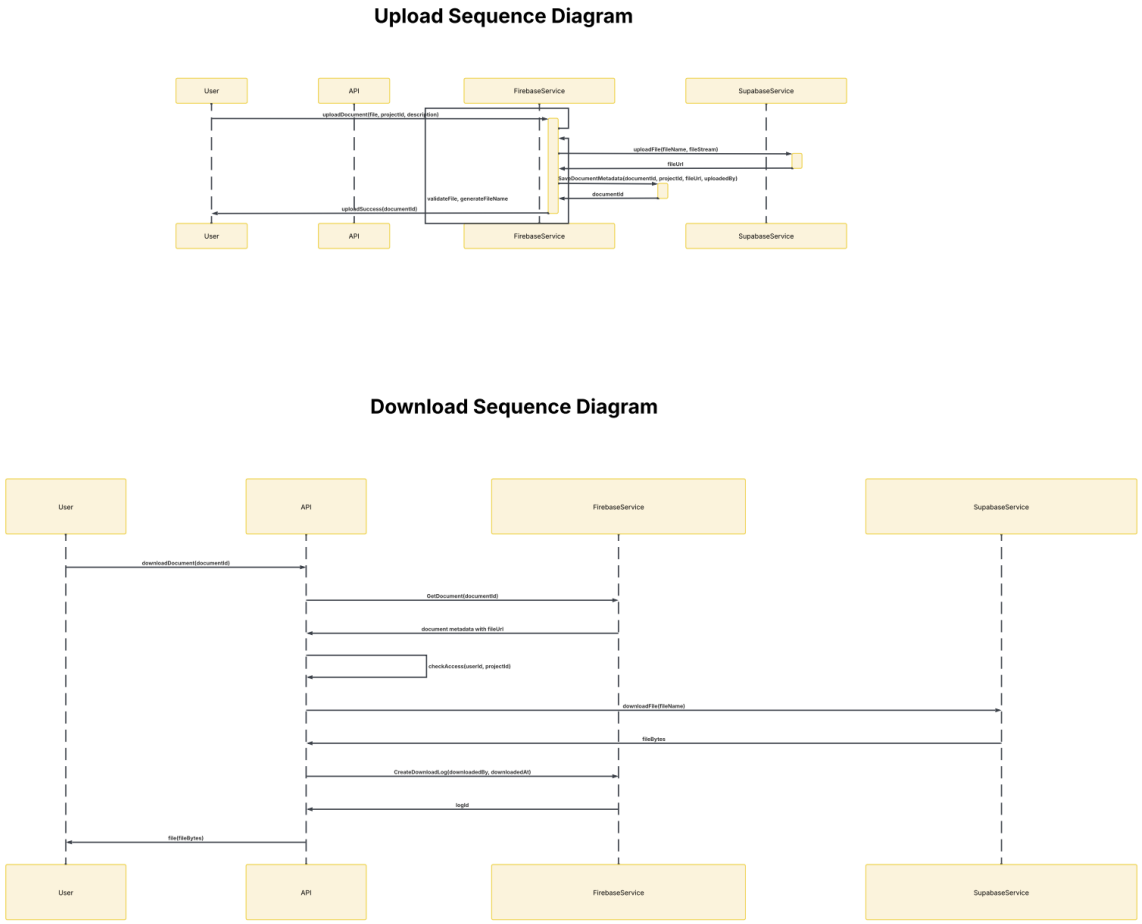


Figure 4.3 – Document Control & Access Sequence Diagram. Source: link here

Maintenance Request Sequence Diagram

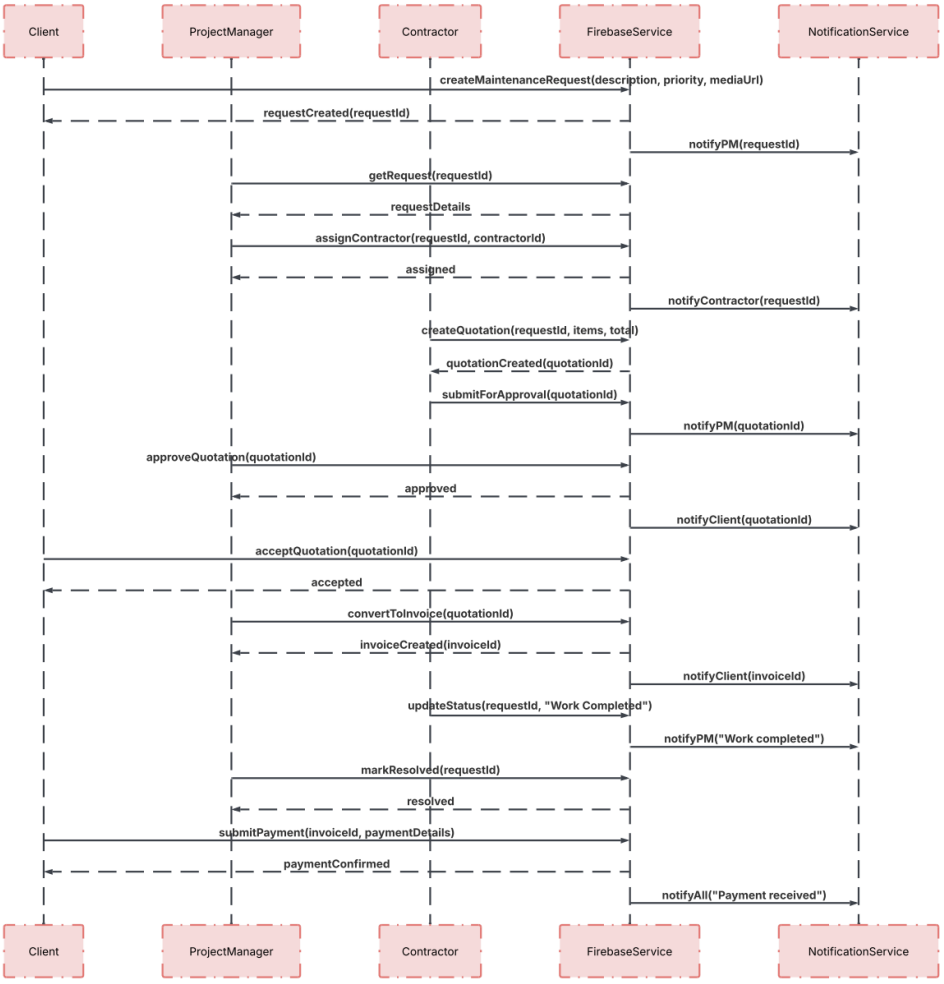


Figure 4.4 – Maintenance Request Sequence Diagram. Source: link here

Invoice Reminder & Dunning Sequence Diagram

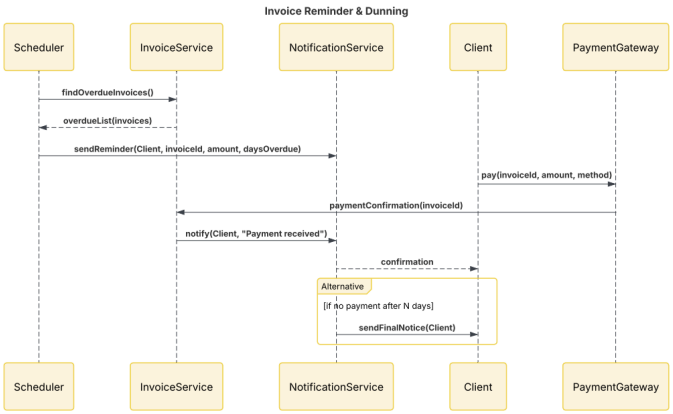


Figure 4.5 – Invoice Reminder & Dunner Sequence Diagram. Source: link here

Contractor Task Sequence Diagram

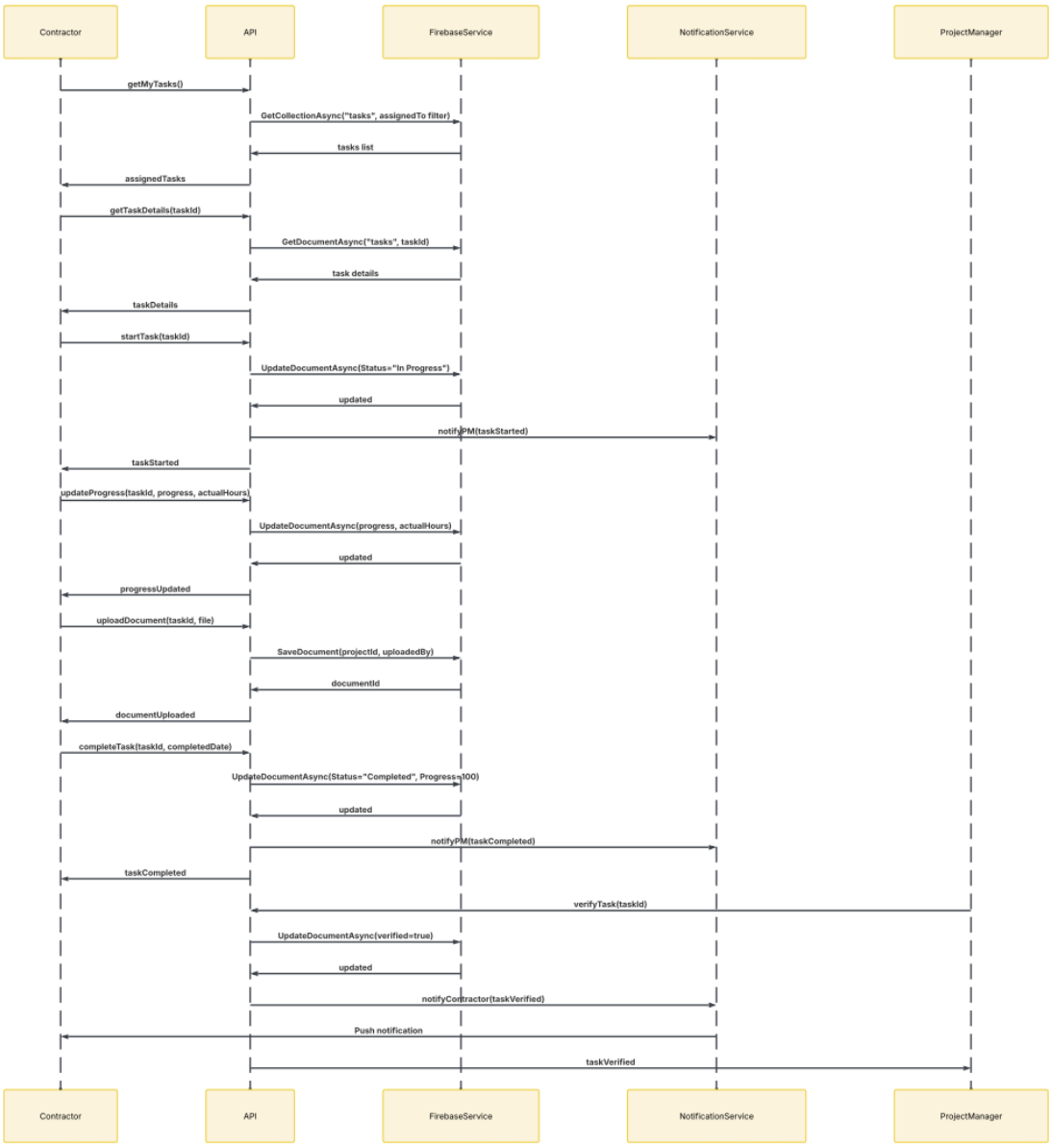


Figure 4.6 – Contractor Task Sequence Diagram. Source link here

4.3. Deployment Documentation

4.3.1 Prerequisites

Azure Account

- Azure subscription
- Resource Group
- App Service Plan
- App Service
- Web App

Firebase Setup

- Create a Firebase project
- Link Google Cloud project
- Enable the following services
    - Authentication: Go to Authentication -> sign-in method -> enable Email/Password C Google
    - Firestore: Go to Firestore Database -> Create Database -> Start in test mode
    - App Check: Go to App Check -> Register apps
- Generate Configuration Files
    - Go to Project Settings -> General -> Your apps
    - Add Android app -> download ‘google-services.json’ and place in app directory.
    - Add Web app -> copy config object -> use in web app.

4.3.2 Development Environment\

Required downloads

- .NET SDK 6.0+
- Azure CLI
- Git
- Visual Studio / VS Code
- Android Studio

4.3.3 Deployment Processes

Backend API Deployment

1. **Build and Deploy**
    - Navigate to API project

    `cd API/ICCMS-API`

    - Build the application

    `dotnet publish --configuration Release --output ./publish`

    - # Create deployment package

    `Compress-Archive -Path "./publish/*" -DestinationPath "./publish.zip" -

    Force`

    - Deploy to Azure

    `az webapp deployment source config-zip --name ICCMS-API --resource- group abcretailers_group --src ./publish.zip`

2. **Restart App Service**
    - Restart to apply runtime changes

    `az webapp restart --name ICCMS-API --resource-group abcretailers_group`

    Alternative:

    Run ./deployAPI bat script from project root

    Web Frontend Deployment

    - Navigate to Web project

    ` cd Web\ICCMS-Web`

    - Build the application

    ` dotnet publish --configuration Release --output ./publish`

    - # Create deployment package

    `Compress-Archive -Path "./publish/*" -DestinationPath "./publish.zip" -

    Force`

    - Deploy to Azure

    ` az webapp deployment source config-zip --name icmms --resource- group abcretailers_group --src ./publish.zip`

Alternative

Run ./deployWeb bat script from project root

Mobile App Build Process

Requirements:

- Android Studio
- ‘google-services.json’ file in app directory
- Proper API endpoint configuration pointing to Cloud Run API

Build steps:

```
# Generate debug APK for testing cd
Mobile

./gradlew assembleDebug
# APK location: app/build/outputs/apk/debug/app-debug.apk


# Generate release APK for distribution
./gradlew assembleRelease
# APK location: app/build/outputs/apk/release/app-release.apk
```

Installation:

- Transfer APK to Android device
- Enable “Install from unknown sources”
- Install APK directly OR
- Can install directly if device is connected using

adb **install** app-debug.apk

4.4. GitHub Actions Pipeline

The GitHub Actions CI/CD Pipeline automates every phase of the ICMMS development lifecycle ensuring consistency and traceability.

Figure 4.7 – GitHub Actions CI/CD Pipeline demonstrates how each commit triggers a workflow that sequentially builds, tests, and deploys both the web and API layers. When a developer pushes code to the repository, the pipeline automatically runs syntax validation, executes unit tests, and checks dependencies against predefined security gates. Only successful builds progress to the deployment stage.

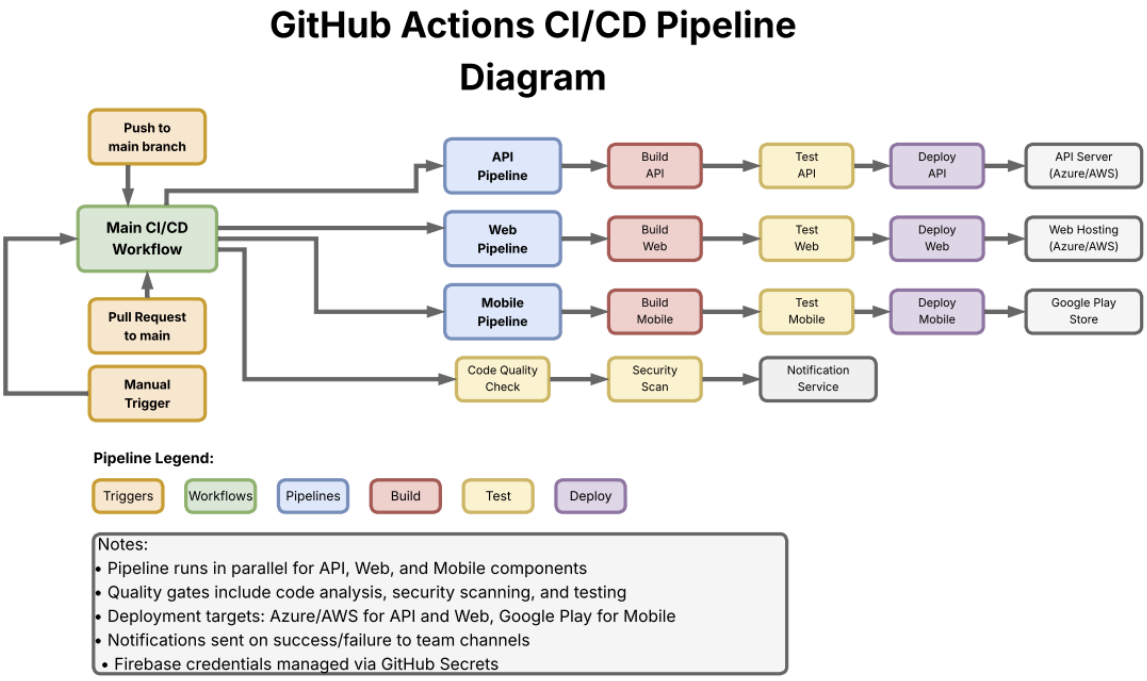


Figure 4.6 – GitHub Actions CI/CD Pipeline Diagram. Source link here

The process integrates with Azure and Firebase using secured environment variables stored within GitHub Secrets. Upon successful validation, the pipeline compresses the project files, transfers them to the Azure App Service, and triggers a post-deployment verification script that confirms endpoint availability and API connectivity. Failed jobs automatically roll back to the last stable build and notify the team via email and Trello webhook.

4.5. Scrum Evidence and Team Progress Documentation

Throughout the ICMMS development lifecycle, the team maintained consistent and transparent documentation of its collaboration and sprint activities through structured meeting minutes. These artefacts, compiled in the “ICMMS Meeting Minutes.zip” folder, capture all key discussions, decisions, and iterations from Sprint 2 through Sprint 9, ensuring that every development milestone is verifiable and traceable.

The recorded meetings began with the early planning and setup sessions held in July and August 2025, where foundational elements such as role assignment, version control, Firebase integration, and database strategy were finalised. Early documents such as *[02 Meeting – Physical Meeting with Junior INSY7135 WIL]* and *[03 Meeting – Group Discussion during Sprint 1]* reflect the team’s initial establishment of development priorities and the agreement to align all components under the ICCMS architecture. These discussions provided the groundwork for sprint-based delivery.

By the end of Sprint 1, the team conducted a review (*[04 Meeting – End of Sprint 1]*) to evaluate the completion of the initial dashboard, quotation flow, and project setup modules. Constructive feedback from this session shaped the improved UI flow and prompted early attention to security and validation measures. Subsequent meetings, such as *[05 Meeting – During Sprint 2]* and *[06 Meeting – End of Sprint 2, Start of 3 and 4]*, documented more advanced backend integration, including AI quotation logic, role-based authentication, and document management enhancements.

The mid-project meetings (Sprints 4–5) recorded detailed discussions on contractor task assignment, messaging implementation, and database refactoring for performance. These discussions, particularly in *[07 Meeting – Sprint Discussions (4,5)]* and *[08 Meeting – Sprint Discussions (4,5)]*, reveal strong inter-role collaboration between backend and frontend contributors.

Finally, *[09 Meeting – Preparing for POE Submission]* and *[11 September Meeting – Project Refinement]* illustrate the consolidation phase, where diagrams were finalised, documentation aligned, and final debugging tasks delegated. The team demonstrated full Scrum adherence, conducting retrospectives after each sprint, verifying deliverables against sprint goals, and maintaining accountability through Trello and GitHub task tracking.

4.6. Summary: Implementation Documentation

Part 4 consolidated the transition from design to full-scale implementation of the Integrated Construction and Maintenance Management System (ICMMS). It demonstrated how conceptual models evolved into the current program.

Implementation focused on structured coordination between roles, data layers, and services, verified through detailed UML sequence diagrams. Each workflow was modelled to confirm the correct order of system interactions and data persistence. These diagrams validated that the live implementation accurately mirrors the intended system behaviour defined in the earlier design phase.

Overall, Part 4 confirmed that ICMMS is not only functional but production-ready, built on traceable, automated, and secure practices. The next section, Part 5 - Testing & DevOps Plan, expands on how these systems were validated under controlled conditions to ensure stability, security, and long-term scalability.

Part 5 - Testing & DevOps Plan

5.1. Introduction

Part 5 outlines the approach used to validate, secure, and continuously deploy the Integrated Construction and Maintenance Management System (ICMMS). The goal was not only to confirm functionality but also to ensure long-term reliability, scalability, and security through disciplined testing and automation practices.

Principles:

Developers test their own code first using unit tests and manual checks.

Known bugs: issues that are expected or recognised will be logged in Trello and resolved as part of the sprint.

Unknown bugs: defects found during app usage, integration, or UAT will also be logged and tracked until resolved.

Major testing rounds will happen after Sprint 3 (Front-End UIs) and at the end of the application (Sprint 7/8).

In between, smaller integration and feature tests will happen after Sprint 4, 5, and 6.

5.2. Test Plan

Testing Levels

1. Unit Testing

- **Focus**: Backend services, database models, and UI components.
- **Responsibility**: Each developer tests their own code before merging.
- **Tools**: NUnit (ASP.NET), JUnit (Kotlin), built-in test frameworks.
- **Expected Outcome**: Modules work in isolation without errors.

2. Integration Testing

- **Focus**: How modules talk to each other e.g., project creation ↔ contractor assignment, AI quote ↔ invoice generation, document upload ↔ access logs.
- **Responsibility**: Team pairs (frontend ↔ backend) test connected features.
- **Tools**: Postman for API, Firebase staging environment, CI/CD pipelines.
- **Expected Outcome**: Data flows smoothly between modules with no breakage.

3. User Acceptance Testing (UAT)

- **Focus**: Real workflows from the perspective of Admin, Project Manager, Contractor, and Client.
- **Responsibility**: Full team, coordinated by Braydon.
- **Method**: Execute Sample Test Cases (to be written separately) in staging.

- **Expected Outcome**: Features align with user stories, critical workflows succeed, and
- major bugs are resolved.

Test Schedule (by Sprint)

- **Sprint 3 (28 Aug - 13 Sep 2025)**
    - First major testing round.
    - Test dashboards, login/registration, forms, messaging, document upload, quotation UI, reporting dashboards.
    - Goal: confirm frontend works and links correctly to backend stubs.
- **Sprint 4 (28 Aug - 18 Sep 2025)**
    - Postman API testing for all endpoints.
    - Verify role-based authentication and access control.
- **Sprint 5 (14 Sep - 2 Oct 2025)**
    - Test quotation workflow, invoice generation, and AI estimation logic.
    - Goal: confirm cost breakdowns and approval/rejection work as expected.
- **Sprint 6 (2 Oct - 16 Oct 2025)**
    - Test new voice memo features and storage.
    - Run security tests with Snyk.
- **Sprint 7 (16 Oct - 2 Nov 2025)**
    - **Final major testing round**.
    - Full UAT across all roles.
    - Test responsive design across devices.
    - Smoke testing on Firebase production.
- **Sprint 8 (5 - 10 Nov 2025)**
    - Demo prep: retest critical workflows.

Deliverables

- Unit test results (from GitHub Actions).
- Postman collections for API integration tests.
- UAT execution logs (Trello board + test cases).
- Final test summary report before submission.

Risks & Mitigation

- **Time pressure during UAT** → Mitigation: Prioritise quotation, maintenance, and project workflows.
- **AI module unpredictability** → Mitigation: Provide fallback manual cost estimation.

- **Integration issues between web/mobile and backend** → Mitigation: Test staging builds after Sprint 4.

5.3. Sample Test (Approach):

All test cases followed a standard template: Test ID, Module, Role, Preconditions, Steps, Expected Result, Actual Result, and Outcome. Testing covered authentication, CRUD operations, workflow logic, and error handling. Each sprint introduced at least one regression round to confirm no prior functionality was compromised.

Format:
ID | Module | Role | Precondition | Steps | Expected Result | Actual Result | Outcome

Access & Security

TC-001 | Auth | Any
Pre: User exists with role
Steps: Login with correct email/password
Expected: Login succeeds; redirected to role-aware dashboard
When: Sprint 3, UAT

TC-002 | Auth | Any
Pre: Existing user
Steps: Attempt login with wrong password 3×
Expected: Account temporarily locked or warning; failed-attempt logged
When: Sprint 3, Security pass in Sprint 7

TC-003 | RBAC | Admin/PM/Contractor/Client
Pre: Logged in as each role
Steps: Try to access a screen outside permissions (e.g., Client → User Management)
Expected: Access denied (UI hidden + API 403)
When: Sprint 4, UAT

TC-004 | HTTPS/CORS | Any
Pre: Deployed staging
Steps: Hit API from web and mobile builds
Expected: Only HTTPS allowed; CORS allows app origins; others blocked
When: Sprint 7

Project Management

TC-010 | Create Project | PM
Pre: PM logged in
Steps: Create project with name, budget, phases, deadlines
Expected: Project saved; appears in PM dashboard; audit entry created
When: Sprint 3 (UI), Sprint 4 (API)

TC-011 | Assign Contractor | PM
Pre: Project exists; contractor exists
Steps: Assign contractor to phase/task
Expected: Assignment saved; contractor sees task; notification sent
When: Sprint 4-5

TC-012 | Budget vs Actual | PM
Pre: Project has tasks with costs/time logged

Steps: Open project dashboard
Expected: Budget vs Actual chart renders with correct totals
When: Sprint 5-7

Maintenance Requests

TC-020 | Create Request | Client
Pre: Client logged in
Steps: Submit maintenance request with description + image
Expected: Request saved; status = Pending; PM notified
When: Sprint 3–4

TC-021 | Assign & Progress | PM/Contractor
Pre: Request exists (Pending)
Steps: PM assigns contractor → Contractor marks In Progress → Completed with photos
Expected: Status transitions tracked; timestamps stored; client sees updates
When: Sprint 4–5

TC-022 | Client Edit Before Assignment | Client
Pre: Request Pending (not assigned)
Steps: Edit description or cancel request
Expected: Edits allowed; cancel sets status = Cancelled; audit stored
When: Sprint 3–4

Document Management

TC-030 | Upload Blueprint | PM/Client
Pre: Logged in as PM or Client; project exists
Steps: Upload PDF/DWG/image
Expected: File stored; metadata saved; role-based access set
When: Sprint 3-4

TC-031 | Download Logs | Admin/PM
Pre: At least one download done by any role
Steps: Admin/PM opens document logs
Expected: Who/when/file recorded correctly
When: Sprint 4-5

TC-032 | Access Control | Contractor
Pre: Contractor not assigned to project
Steps: Try to open project document URL
Expected: Access denied (UI + API 403)
When: Sprint 4-5

Communication & Notifications

TC-040 | Messaging Flow | PM ↔ Contractor
Pre: PM and Contractor assigned on same project
Steps: PM sends message; Contractor replies
Expected: Messages appear in thread for both; timestamps correct
When: Sprint 3-4

TC-041 | Alerts | Any
Pre: Project milestone or status change
Steps: Trigger event (assignment, quote ready, overdue)

Expected: In-app notification + email/SMS (if configured) to correct role(s)
When: Sprint 5-7

Quotation & Invoices (AI Workflow)

TC-050 | AI Quote from Blueprint | PM
Pre: Project exists; blueprint uploaded
Steps: Request AI quote generation
Expected: Cost breakdown generated; includes materials + labour; status = Draft
When: Sprint 5

TC-051 | Quote Review | Admin/PM
Pre: Draft quote exists
Steps: Open quote; review line items; adjust allowed fields; send to Client → status = Sent
Expected: Client receives notification; quote visible to Client
When: Sprint 5

TC-052 | Client Approval/Reject | Client
Pre: Quote = Sent
Steps: Approve → or Reject
Expected: Approve → status = Accepted; Reject → status = Rejected; audit logged
When: Sprint 5

TC-053 | Invoice Generation | System
Pre: Quote = Accepted
Steps: System converts to invoice
Expected: Invoice created with correct amounts; payment status = Unpaid
When: Sprint 5

Reporting & Dashboards

TC-060 | KPIs Render | Admin
Pre: Seeded data for projects, maintenance, quotes, invoices
Steps: Open admin dashboard
Expected: KPIs show totals; charts (Gantt, Pie, Bar, Line) render without errors
When: Sprint 3-7

TC-061 | Contractor Ratings | Admin/PM
Pre: Some completed tasks with ratings
Steps: View contractor performance report
Expected: Accurate averages and counts
When: Sprint 5-7

Voice Feature

TC-070 | Voice Memo Upload | Contractor
Pre: Assigned task
Steps: Record/upload voice memo
Expected: File saved; linked to task; playable by PM/Admin
When: Sprint 6

Security & Quality Gates

TC-080 | Snyk Scan | Repo
Pre: Pipeline configured
Steps: Run Snyk on dependencies

Expected: No high/critical issues, or documented exceptions
When: Sprint 6-7

TC-081 | SonarQube Quality Gate | Repo
Pre: Pipeline configured
Steps: Run analysis on PR
Expected: Passes gate; coverage thresholds met on changed code
When: Sprint 7

UAT Packs (Major Rounds)

UAT-1 (post Sprint 3):

- TC-001, 003, 010, 020, 030, 040, 060

UAT-2 (final, Sprint 7/8):

- Full flow: 001–004, 010–014 (if added), 020–022, 030–032, 040–041, 050–054, 060–061, 070, 080–081

Every test case was peer-reviewed and revalidated following code merges or bug fixes to maintain consistent quality control.

5.4. Bug Tracking and Resolution Workflow

Defect management followed a clear workflow that ensured full visibility from discovery to resolution. Each bug was logged with a severity level (Critical, High, Medium, Low), a reproducibility indicator, and a version reference.

The resolution lifecycle followed this sequence:
New → Assigned → In Progress → Code Review → Testing → Verified → Closed.

5.4.1. Bug Tracking Process

Tool

- Trello Board will serve as the single source of truth for all bugs.
- All defects, discussions, and tracking will take place inside Trello or WhatsApp, or if needed Google Meet.
- GitHub will only be used for code management, pull requests, and linking fixes.

Roles (for bug management)

Triage Owners (Handle & Delegate Errors): Braydon (Scrum Master) or Max (Lead Developer) who are responsible for triaging, labelling, and prioritising bugs.

- Max - Architecture / Overall system health
- Denzel - Backend & Security (lead)
- Daniel - Backend support & documentation
- Nico & Braydon - Web & Mobile UI

Workflow (States)

1. **New** → Bug is created in Trello (and/ or WhatsAp) by any team member.
2. **Triage** → Bug is labelled (severity, priority, area) and assigned by Braydon or Max.
3. **In Progress** → Assigned developer begins work on a fix and calls other members if needed.
4. **In Review** → Fix is committed to GitHub, code is reviewed by a peer.
5. **Ready for Test** → Fix is deployed to staging for verification.
6. **Verified** → Bug is resolved, Trello card is moved to “Done.”
7. **Reopen** → If issue persists, Trello card is returned to “In Progress.”

SLA Targets (Time Expectations)

- **Critical (S1)** → triage within 4 hours, fix within 24 hours.
- **High (S2)** → triage within 24 hours, fix within 3 days.
- **Medium (S3)** → fix in next sprint.
- **Low (S4)** → fix when time allows.

Labels (Consistent Usage in Trello)

- **Type**: bug, regression, security, ui/ux, performance, docs
- **Severity**: S1-critical, S2-high, S3-medium, S4-low
- **Priority**: P1, P2, P3
- **Area**: backend, api, web-frontend, mobile-frontend, auth, storage, ci-cd, security
- **Module**: projects, maintenance, contractor, documents, messaging, quotation, invoice, reporting, ai, voice
- **Status**: triage, in-progress, in-review, ready-for-test, verified, reopen

Bug Card Template (Trello)

Each bug card must include the following:

Summary
Short, one-line description of the problem.

Steps to Reproduce
1., 2., 3.

Expected Result
What should happen.

Actual Result
What actually happens.

Scope / Area / Module
Area: [backend|api|web-frontend|mobile-frontend|auth|storage|ci-cd|security]
Module:
[projects|maintenance|contractor|documents|messaging|quotation|invoice|reporting|ai|voice]

Environment
Development / Staging / Production, commit ID, device/browser.

Attachments
Screenshots, logs, Postman exports, console output.

Severity & Priority
Severity: S1|S2|S3|S4
Priority: P1|P2|P3

Definition of Done (for a bug)

- Bug can be reproduced and confirmed.
- Fix has been implemented and pushed to GitHub.
- Peer review completed.
- Tests (manual and automated if available) pass successfully.
- Fix is verified on staging environment.
- Trello card is moved to **Done/Closed** with commit ID linked.

Weekly triage meetings reviewed all open defects and linked each resolution to a sprint deliverable, ensuring accountability and transparency throughout the process.

5.5. Security and Quality Testing

Security and quality testing were embedded into every stage of the development process to ensure that the Integrated Construction and Maintenance Management System (ICMMS) not only functioned correctly but also maintained strict protection of sensitive data. Instead of treating security as an isolated step near deployment, the team incorporated validation measures across the full CI/CD pipeline.

5.4.1. Authentication and Access Control Testing:

Each sprint included repeated verification of Firebase Authentication workflows. Tokens were validated for expiration, tampering, and correct role assignment before granting access to any protected endpoints. Role-based access control (RBAC) was stress-tested to confirm that users such as Contractors or Clients could not execute restricted Admin or Project Manager operations.

5.4.2. Data Validation and Input Handling:

All inputs were tested for injection vulnerabilities and malformed data entries. The backend API employed strict model binding and data type validation to prevent unauthorised schema changes. Client-side sanitisation was paired with backend verification to eliminate the risk of XSS and SQL-like payload injection.

5.4.3. Encryption and Transmission Testing:

End-to-end HTTPS communication was verified between web, API, and Firebase layers. Testing confirmed that sensitive data (login credentials, quotations, financial details) remained encrypted both in transit and at rest within Firestore and Supabase storage.

5.4.4. Automated Quality Scans:

Each commit to the development branch triggered static code analysis and vulnerability scanning through integrated tools. These scans assessed code complexity, coverage percentage, and dependency health, allowing the team to identify critical vulnerabilities long before deployment.

5.4.5. Load and Performance Validation:

During final sprints, concurrent user simulations were run to monitor system response times and confirm resource allocation under load. The API and web app demonstrated stable performance without degradation beyond 20% under simulated traffic.

Through this continuous, layered approach, the team achieved a stable and secure application build that met all predefined non-functional requirements and passed internal audit verification.

5.6. DevOps and Continuous Integration

5.5.1. Branching and Version Control:

The GitHub repository followed a disciplined branching structure; feature branches were created for each new functionality, merged into a central dev branch after peer review, and then into a protected main branch for deployment. Each merge triggered automated build and test sequences within GitHub Actions, ensuring that no untested code reached the live environment.

5.5.2. CI/CD Pipeline Automation:

The CI/CD pipeline compiled the backend API and web frontend, executed unit tests, and deployed successful builds directly to Azure App Services. The mobile app (Kotlin) followed a parallel workflow, building and packaging signed APKs for distribution. Deployment stages were clearly defined:

1. **Build Stage:** Compiles and validates project dependencies.
2. **Test Stage:** Executes automated test suites for logic and data integrity.
3. **Deploy Stage:** Publishes new builds to Azure and Firebase after successful test completion.

5.5.3. Collaboration and Review Workflow:

All commits required at least one peer review before merging. Build and test results were automatically posted to the team's communication platform, ensuring visibility and accountability for each change. The entire process was fully traceable through GitHub Actions logs and Azure deployment histories.

This DevOps model ensured rapid delivery without sacrificing quality or stability. The automated testing, security validation, and deployment processes established a reliable development rhythm and positioned ICMMS for long-term scalability and maintenance efficiency.

5.7. Summary: Testing & DevOps Plan

Part 5 demonstrated how the team embedded testing, security, and DevOps practices into every phase of ICMMS development to maintain professional-grade reliability. Testing followed a structured, sprint-based rhythm to ensure predictable and repeatable results.

Bugs and quality issues were logged, triaged, and tracked through a transparent workflow on Trello, with strict severity levels and turnaround targets. Security testing was continuous rather than reactive. Automated quality gates scanned every commit for code integrity and dependency health.

The DevOps framework unified the process: GitHub Actions pipelines automated build, test, and deployment cycles across Azure and Firebase environments

Together, these integrated practices delivered a tested, secure, and continuously deployable system foundation. This foundation now supports **Part 6 – Final Report and Handover**, where results, operational readiness, and team reflections complete the project's lifecycle documentation.

Part 6 - Final Report and Handover

6.1. Executive Summary

6.1.1. Project Overview

Part 6 concludes the Integrated Construction and Maintenance Management System (ICMMS) project by consolidating the system's technical, operational, and organisational outcomes into a unified reflection. This section highlights how the team transformed the original concept into a fully deployed cloud-based ecosystem —

The deliverables presented here capture the maturity of the system: a production-ready platform hosted on Azure, secured by Firebase Authentication, and extended with Supabase for structured file management. The system consists of three integrated components a RESTful .NET 9 API, an ASP.NET MVC web interface, and a Kotlin-driven mobile application.

This section also introduces the final handover components: cloud cost forecasting, adoption and training strategies, and post-launch support mechanisms. Collectively, these artefacts demonstrate a project that is not only technically successful but strategically prepared for long-term institutional and commercial adaptation.

6.1.2. Project Achievements

Successful Cloud Deployment

- Successfully deployed a .NET 9.0 RESTful API to Azure App service.
- Deployed ASP.NET MVC web application to Azure App Service.
- Established robust cloud infrastructure using Azure App Service, Firebase, and Supabase.
- Implemented GitHub actions for automated deployment and continuous integration.

Technical Implementation

- Implemented layered architecture with clear separation of concerns.
- Integrated Firebase Authentication for secure user management.
- Successfully connected Firestore and Supabase for data persistence.
- Deployed Swagger documentation for API testing and integration.
- Deployed testing platform for API testing.
- Developed solutions for both mobile and web applications.

System Features Delivered

- Complete user creation, authentication, and role-based access control.
- Project creation, tracking, and management capabilities.
- Comprehensive stakeholder management system between clients and contractors.
- Real-time communication between project stakeholders.
- Automated workflow processes quotes, invoices and approvals.
- Foundation for AI-powered features including blueprint parsing and risk assessment.

6.1.3. Project Impact

Operational Efficiency

- Automated project management processes by reducing manual overhead.
- Enhanced stakeholder communication through integrated messaging.
- Centralized document storage and retrieval system.
- Automated quote and invoice generation workflows.

Technical Excellence

- Cloud-native design supporting future growth and expansion.
- Robust authentication and authorization systems.
- Efficient data handling and API response times.
- Clean code architecture following industry best practices.

Business Value

- Automated processes reduce administrative overhead
- Enhanced communication between clients, contractors, and project managers.
- Comprehensive reporting and analytics capabilities.
- Field-ready mobile application for on-site project management.

6.1.4. Project Outcomes

Deliverables Completed

- RESTful API
- Web application
- Mobile application
- Cloud deployment
- Database integration
- Authentication system

Quality Metrics

- Comprehensive test coverage for critical business logic.
- Optimized API response times and efficient data handling.
- Implemented industry-standard security practices.
- Architecture designed to handle increased user load and data volume.

Future Readiness

- Foundation established for Gemini API integration.

6.2. Final System Walkthrough

(To be Completed)

6.3. Architecture Patterns Used

6.3.1. Explanation and Justification

Component	Primary Patterns	Secondary Patterns
API	• Layered Architecture • Service Layer • Repository	• DI • DTO • Authentication Handler
Web	• MVC • Layered Architecture • ViewModel	• Service Layer • DI
Mobile	• MVVM • Repository • Observer	• DI • Factory • Strategy

1. Multi-Tiered Architecture

Our system follows a classic 3-tier architecture pattern

- Presentation Tier: Web application and Mobile Application.
- Business Logic Tier: API layer with controllers and services.
- Data Tier: Firestore (NoSQL) and Supabase for file storage.

This separation of concerns provides clear boundaries between user interface, business logic, and data storage. This makes the system maintainable and scalable. Each tier can be deployed, tested and deployed independently.

2. Microservices-Orientated Architecture

While our system does not fully use microservices, our system demonstrates microservice principles.

- API Layer: Centralized business logic and data access.
- Web Layer: Separate presentation layer.
- Mobile Layer: Independent client application
- External Services: Firebase Authentication, Firestore, and Supabase.

This approach allows for independent scaling of different components and enables different teams to work on different parts of the system simultaneously.

3. Cloud-Native Architecture

Our system is designed for cloud deployment

- Firebase Integration: Authentication and NoSQL Database.
- Supabase: File storage
- Azure Deployment: Cloud hosting

Cloud-native design ensures scalability, reliability, and cost-effectiveness while providing accessibility.

4. API-First Architecture

The API acts as the central hub

- RESTful API: Standard HTTP methods and status codes.
- Swagger Documentation: API-first documentation
- CORS Configuration: Cross-origin resource sharing for multiple clients

API-first approach enables multiple client applications to consume same backend services, promoting code reuse and consistency.

6.4. Design Patterns Used

6.4.1. Explanation and Justification

1. *Dependency Injection*

Used extensively throughout the application.

This pattern promotes loose coupling, testability, and maintainability. Services can easily be mocked for testing and swapped for different implementations.

2. *Repository Pattern*

Implemented through the service interfaces

The repository pattern provides a clean abstraction layer between business logic and data access, making it easy to change data sources without affecting the business logic.

3. *Service Layer*

Clear separation of business logic.

- Controllers handle HTTP requests/responses.
- Services contain the business logic.
- Models handle data transfer.

Using a service layer allows for separation of concerns and makes the code more maintainable and testable.

4. *Factory Pattern*

Used in service initialization.
This provides flexible object creation and configuration management.

5. *Template Method Pattern*

This pattern is used in workflow message templates.

A template defines the skeleton of an algorithm while allowing subclasses to override specific steps.

6. *State Machine Pattern*

Implemented in workflow services

- Quote Workflow: Draft -> PendingPMApproval -> SentToClient -
  > ClientAccepted/ClientDeclined
- Invoice Workflow: Draft -> issues -> Paid
- Project Workflow: Various state transitions.

The state machine pattern ensures proper business flow and prevents invalid state transitions.

7. *Observer Pattern*

Used in notification system

- WorkflowMessageService: Observes system events
- NotifcaitonService: Notifies users of changes
- Event-driven architecture: System events trigger notifications

Using the observer pattern enabled loose coupling between components and supports event-driven architecture.

8. Strategy Pattern

Used for different validation strategies.
The strategy pattern allows runtime selection of algorithms and makes the system extensible.

9. Builder Pattern

Used in configuration setup.
This pattern provides a flexible way to construct complex objects step by step.

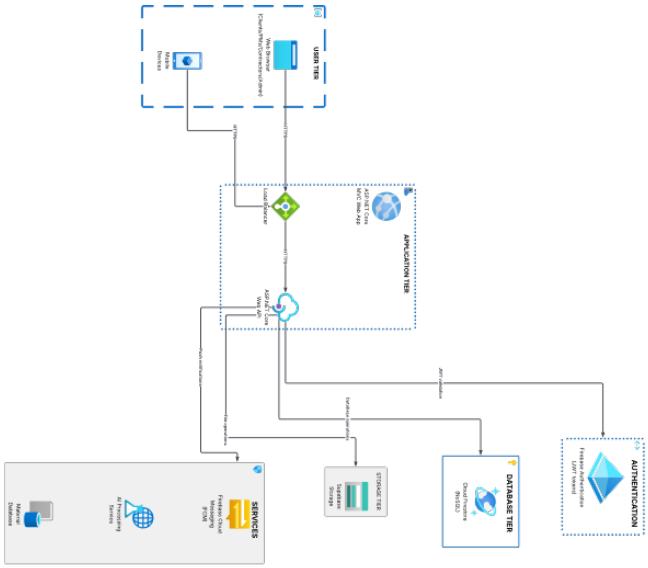
10. Facade Pattern

Controllers act as facades.

- AuthController: Simplifies authentication operations
- AdminController: Simplifies administrative operations
- MessageController: Simplifies messaging operations
- Etc

This pattern provides a simple interface to complex subsystems.

6.5. Cloud Architecture Diagram



System Overview Diagram

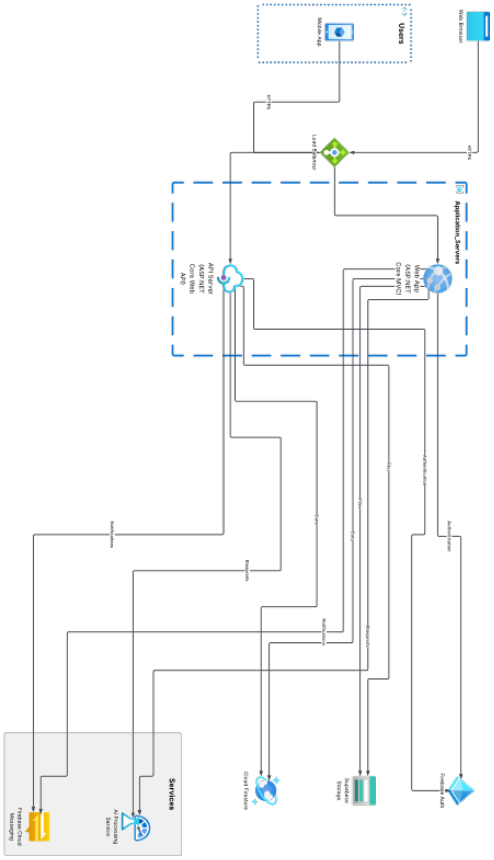


Figure 6.1 (&3.4) – System Overview Diagram. Source: Lucid Chart*:* Link Here

6.6. Predictive Running Cost:

6.6.1. Executive Cost Breakdown

This section provides cost projections for running the Integrated Construction and Maintenance Management System over the next two years, including scaling scenarios based on project volume.

Technology Stack Cost Components

Based on the current implementation:

- Firebase Firestore – NoSQL database
- Firebase Authentication – User authentication
- Firebase Cloud Messaging – Push notifications
- Supabase Storage – File storage (documents, images, videos)
- Azure App Service – API and Web hosting (.NET 8.0)
- AI/ML Services – Blueprint processing (Azure Cognitive Services or similar)
- SSL Certificates – Security
- Bandwidth & CDN – Content delivery

6.6.2. Table 1: Quarterly Running Costs (2 Years)

Baseline Assumptions

- 50 active users
- 20 active projects
- 100 maintenance requests per quarter
- 500 documents uploaded per quarter
- 10,000 messages per quarter
- 50 AI blueprint analyses per quarter

Quarter	Firebase	Supabase Storage	Azure Hosting	AI Services	SSL & Domain	Bandwidth	Total Cost
2025 Q4	R2 775	R1 480	R7 400	R4 625	R925	R1 850	**R19 055**
2026 Q1	R3 053	R1 758	R7 770	R5 088	R278	R2 035	**R19 982**
2026 Q2	R3 330	R2 035	R7 770	R5 550	R278	R2 220	**R21 183**
2026 Q3	R3 608	R2 313	R8 140	R6 013	R278	R2 405	**R22 757**
2026 Q4	R3 885	R2 590	R8 140	R6 475	R925	R2 590	**R24 605**
2027 Q1	R4 163	R2 960	R8 510	R6 938	R278	R2 775	**R25 624**
2027 Q2	R4 440	R3 330	R8 510	R7 400	R278	R3 053	**R27 011**
2027 Q3	R4 810	R3 700	R8 880	R7 863	R278	R3 330	**R28 861**
2027 Q4	R5 180	R4 070	R8 880	R8 325	R925	R3 608	**R30 988**

Annual Totals		
Year	**Total Annual Cost**	**Average Monthly Cost**
2026	**R88 527,00**	**R7 377,00**
2027	**R112 484,00**	**R9 374,00**
Combined	**R201 011,00**	**R16 751,00**

- 2-Year Total: R201011

Table 2: Cost Scaling by Project Volume

This table shows how monthly costs scale as the number of active projects increases.

Scaling Assumptions

- Users scale at 2.5 per project
- Maintenance requests: 5 per project per month
- Documents: 25 per project per month
- Messages: 500 per project per month
- AI analyses: 2–3 per project per month

6.6.3. Table 2: Scaling Costs per Project

Projects	Users	Fire-base	Supa base	Azure Hosting	AI Services	Bandwidth	Monthly Cost	Annual Cost
10	25	R740	R463	R2 220	R1 480	R648	**R5 550**	**R66 600**
20	50	R1 295	R925	R2 590	R2 313	R925	**R8 048**	**R96 570**
50	125	R3 330	R2 590	R3 700	R5 920	R2 035	**R17 575**	**R210 900**
100	250	R7 030	R5 550	R5 180	R12 025	R3 885	**R33 670**	**R404 040**
200	500	R14 430	R12 025	R7 770	R24 050	R7 400	**R65 675**	**R788 100**
500	1250	R36 075	R31 450	R13 875	R60 125	R17 575	**R159 100**	**R1 909 200**
1000	2500	R72 150	R66 600	R22 200	R120 250	R33 300	**R314 500**	**R3 774 000**

6.6.4. Table 3: Cost Per Project Analysis

This table shows the cost per project once scaled

Metric	20 Projects	50 Projects	100 Projects	200 Projects	500 Projects
Cost Per Project	R402	R352	R337	R328	R318
Cost Per User	R161	R141	R135	R131	R127
Monthly Cost	**R8 048**	**R17 575**	**R33 670**	**R65 675**	**R159 100**

6.7. Change Management Strategy:

6.7.1. Who Needs to Adopt?

User Group	Number	Main Concern	How to Help
Project Managers	~8	"Will this slow me down?"	Show time savings, quick training
Contractors	~15	"Too complicated"	Simple mobile app
Clients	~20	"Do I have to?"	Make it easier than calling or emailing
Admin Staff	~5	"More work for me?"	Show how it reduces paperwork

6.7.2. 3-Month Rollout Plan

Month 1: Get Ready

- Week 1-2: Create simple user guides (videos and PDF)
- Week 3-4: Train 3 “champions” who love tech
- Set up WhatsApp support group

Month 2: Start Using It

- Week 1: Train all Project Managers (4 hours)
- Week 2: Train Admin staff (2 hours)
- Week 3-4: Train Contractors in small groups (1 hour each)
- Launch with daily support available

Month 3: Get Everyone On Board

- Week 1-2: Activate client portal with simple emails
- Week 3-4: Check who’s not using it and help them
- Celebrate quick wins

By the end of Month 3, everyone should be using it daily.

6.7.3. Simple Training Plan

For Project Managers (4 hours)

1. Login and dashboard – 30 minutes
2. Create and track projects – 1 hour 30 minutes
3. Assign tasks to contractors – 1 hour
4. Documents and quotations – 1 hour

For Contractors (1 hour)

1. Download app and login – 10 minutes

2. See your tasks – 20 minutes
3. Update progress with photos – 20 minutes
4. Mark complete – 10 minutes

For Clients (Self-serve, 30-minute video)

1. Login to portal
2. See your project progress
3. Submit maintenance requests
4. View invoices

For Admin Staff (1.5 hours)

1. User management – 45 minutes
2. Reports and exports – 45 minutes

6.7.4. Support Plan

Week 1-4: Two people available 8am–6pm for questions (WhatsApp and phone)
Month 2-3: One person available 9am–5pm
After Month 3: Normal IT support

Support Channels:

- WhatsApp: 078 404 9189
- Email: support@icmms.co.za
- Phone: 078 404 9189

6.7.5. Handling Resistance

"It's too hard"
→ Offer 1-on-1 session, pair with a tech-savvy colleague

"I don't have time"
→ Show how it saves time (10 minutes training = 1 hour saved weekly)

"The old way works"
→ Get their manager to set expectations, show benefits

"It doesn't work"
→ Fix the actual problem quickly, acknowledge frustration

6.7.6. Success Metrics

- 80% of people logging in weekly by end of Month 3
- Contractors using the mobile app for task updates
- Less than 5 support requests per week by Month 4
- Positive feedback in surveys

6.7.7. Budget Breakdown

Item	Cost
Training materials (videos and guides)	R15 000
Extra support person (3 months)	R90 000
Small rewards for early adopters	R20 000
Lunch and learns, snacks, printing	R10 000
Contingency	R15 000
TOTAL	**R150 000**

6.7.8. Communication

Before Launch:

- Email: “New system coming – here’s why it’s good for you.”
- Meeting: Demo the system and answer questions.

During Launch:

- Weekly email: Tips, success stories, and contact details for support.
- WhatsApp group: Quick help and encouragement.

After Launch:

- Monthly “What’s New” email.
- Celebrate wins publicly.

6.7.9. Champion Program

Find 5–6 people who:

- Like technology
- Are respected by others
- Want to help

They will:

- Test the system first and give feedback
- Help train their colleagues
- Be the “go-to” person in their team

6.8. Lessons Learned:

To be completed during Final Submission

6.9. Summary: Final Report & Handover

Part 6 represents the culmination of the ICMMS development lifecycle, uniting the project's engineering depth with its business rollout readiness. Across the subsections, the documentation provides evidence of complete system delivery.

The executive summary outlined the successful integration of Azure App Services, Firebase, and Supabase into a cohesive, cloud-native architecture. The architecture and design pattern analysis proved that maintainability and scalability were achieved through layered and event-driven design, while the predictive cost model translated technical success into measurable financial sustainability over a two-year horizon. The change management plan demonstrated pragmatic rollout sequencing; balancing user resistance, training accessibility, and ongoing support through WhatsApp-based communication and internal champions.

7. Appendices

7.1. Declaration of Authenticity

We, the undersigned members of this project team, collectively declare that this Integrated Construction and Maintenance Management System represents our original collaborative work. AI-assisted tools including Cursor and ChatGPT were utilized to assist with code boiler plating and development efficiency, while all core logic, architecture decisions, and implementations remain our original work. Each team member's contributions have been genuine and appropriately documented. We confirm that this system was developed for our academic coursework and has not been submitted elsewhere for assessment. We acknowledge that we have adhered to all institutional academic integrity policies and ethical guidelines throughout this project. We take collective responsibility for the authenticity and originality of this work.

Team Members:

- Denzel Zimba - ST10383606

- Daniel Jung - ST10324495

- Braydon Wooley - ST10394807

- Nicolas Christofides - ST10339570

- Max van der Walt - ST10354483

Date: 2025/10/10

7.2. Scrum Artifacts

Attached Scrum Meeting Minutes

8. References

Core Technologies

Firebase. (n.d.) *Cloud Firestore Documentation.* Available at: https://firebase.google.com/docs/firestore (Accessed: 9 October 2025).
Firebase. (n.d.) *Firebase Authentication.* Available at: https://firebase.google.com/docs/auth (Accessed: 9 October 2025).
Firebase. (n.d.) *Firebase Storage.* Available at: https://firebase.google.com/docs/storage (Accessed: 9 October 2025).
Supabase. (n.d.) *Storage Guide.* Available at: https://supabase.com/docs/guides/storage (Accessed: 9 October 2025).
Microsoft Azure. (n.d.) *Azure App Service Documentation.* Available at: https://learn.microsoft.com/en-us/azure/app-service (Accessed: 9 October 2025).
Microsoft Azure. (n.d.) *Azure CLI Reference.* Available at: https://learn.microsoft.com/en-us/cli/azure (Accessed: 9 October 2025).
Microsoft. (n.d.) *.NET 6 SDK Documentation.* Available at: https://learn.microsoft.com/en-us/dotnet/core/sdk (Accessed: 9 October 2025).
GitHub. (n.d.) *GitHub Actions Documentation.* Available at: https://docs.github.com/en/actions (Accessed: 9 October 2025).
Docker. (n.d.) *Docker Overview Documentation.* Available at: https://docs.docker.com/get-started/overview/ (Accessed: 9 October 2025).
SonarSource. (n.d.) *SonarQube Documentation.* Available at: https://docs.sonarsource.com/sonarqube/latest/ (Accessed: 9 October 2025).
Snyk Ltd. (n.d.) *Snyk Vulnerability Scanning Documentation.* Available at: https://docs.snyk.io (Accessed: 9 October 2025).
Google Cloud SQL. (n.d.) *Focus on your application, and leave the database to us.* Available at: https://cloud.google.com/sql (Accessed: 13 August 2025).
Teak, T. (2024) *Database Schema: Why It Matters in SQL Data Management.* Available at: https://www.pingcap.com/article/database-schema-why-it-matters-in-sql-data-management/ (Accessed: 13 August 2025).

Development & Tooling

Kotlin. (n.d.) *Official Kotlin Programming Language Documentation.* Available at: https://kotlinlang.org/docs/home.html (Accessed: 9 October 2025).
Android Developers. (n.d.) *Room Database Overview.* Available at: https://developer.android.com/training/data-storage/room (Accessed: 9 October 2025).
Visual Studio Code. (n.d.) *VS Code Documentation.* Available at: https://code.visualstudio.com/docs (Accessed: 9 October 2025).
Google LLC. (n.d.) *Postman API Testing Tool Overview.* Available at: https://www.postman.com/product/api-testing/ (Accessed: 9 October 2025).

Project Management & Methodology

Scrum.org. (2020) *The Scrum Guide – The Definitive Guide to Scrum: The Rules of the Game.* Available at: https://scrumguides.org/scrum-guide.html (Accessed: 9 October 2025).
PremierAgile. (2023) *Definition of Ready and Definition of Done in Agile.* Available at: https://premieragile.com/definition-of-ready-vs-definition-of-done (Accessed: 9 October 2025).

Internal Artefacts

ICMMS Development Team. (2025) *ICMMS GitHub Repository.* Available at: https://
https://github.com/INSY7314-a-team/ICMMS-INSY7315-WIL
ICMMS Development Team. (2025) *ICMMS Lucidchart Repository – System Diagrams.*
Available at: https://lucid.app/lucidchart/ebc85260-6801-4178-a378-
dd99698ea8e7/edit?viewport_loc=-742%2C-
1085%2C4866%2C2059%2Cc3KVyjod~nFq&invitationId=inv_7bf09684-d572-42df-9856-
a59685633a67
ICMMS Development Team. (2025) *ICMMS Trello Scrum Board.* Available at:
https://trello.com/invite/b/6894e29e769ddcb284d6acc1/ATTI8519a919cb44ffbe35a8cbc5591
fe6bbA8532CEB/insy7315-wil-project-management