RESIDENTIAL MAINTENANCE MANAGEMENT SYSTEM

FINAL YEAR PROJECT REPORT

A report submitted in partial fulfillment of the requirements for the award of the Bachelor of Science in Computer Science

College of Science, Computer Science and Cybersecurity

December 2025

DECLARATION

I declare that this report entitled "Residential Maintenance Management System" is the result of our own research and development work except as cited in the references. The report has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature: Tristan Kras, Quin Perkins, Bethlehem Tewodros, Sandip Poudel

Name: Tristan Kras, Quin Perkins, Bethlehem Tewodros, Sandip Poudel

Date: 12/08/25

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who have contributed to the successful completion of this project. We are especially thankful to our project supervisor Professor Dr. Ismail Bile Hassan for their continuous guidance, valuable feedback, and support throughout the development process. We also extend our appreciation to our faculty members who provided us with the necessary knowledge and skills to undertake this project. Special thanks to our family and friends for their encouragement and patience during this work. Finally, we acknowledge the various online resources and documentation that helped us understand the technical aspects of the system we developed.

ABSTRACT

The Residential Maintenance Management System is a comprehensive software solution designed to streamline and automate the maintenance request process in residential buildings. The system addresses the common challenges faced by tenants, building managers, and maintenance staff in managing repair and maintenance requests efficiently. Built using JavaFX for the user interface and integrated with a relational database, the system provides role-based access for four distinct user types: Admin, Tenant, Building Manager, and Maintenance Staff. The system enables tenants to submit maintenance requests with detailed descriptions and photo attachments, allows building managers to review and assign requests to appropriate maintenance staff based on their specialization and availability, and provides maintenance staff with tools to update work progress and complete assigned tasks. The project follows a structured software development methodology, incorporating detailed system design through use case diagrams, sequence diagrams, activity diagrams, entity-relationship diagrams, and UML class diagrams. The system aims to improve communication between all stakeholders, reduce response time for maintenance issues, increase accountability through tracking and documentation, and provide comprehensive reporting and analytics capabilities for building management.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figures found on: https://github.com/qperkins33/Residential-Maintenance-Management-System/tree/main/diagrams

LIST OF ABBREVIATIONS

NO	ABBREVIATION	DESCRIPTION
1	RMMS	Residential Maintenance Management System
2	JavaFX	Java Platform for Creating Rich Internet Applications
3	JDBC	Java Database Connectivity
4	UML	Unified Modeling Language
5	ER	Entity Relationship
6	UI	User Interface
7	CRUD	Create, Read, Update, Delete
8	API	Application Programming Interface
9	SQL	Structured Query Language
10	DAO	Data Access Object

1.0 INTRODUCTION

Residential buildings and apartment complexes face ongoing challenges in managing maintenance requests efficiently. Tenants often struggle to report issues clearly, building managers find it difficult to track and prioritize multiple requests, and maintenance staff lack proper tools to document their work progress. Traditional paper-based or email-based systems lead to lost requests, delayed responses, poor communication between stakeholders, and lack of accountability.

The Residential Maintenance Management System (RMMS) was developed to address these critical issues by providing a centralized, digital platform for managing all aspects of residential maintenance operations. This system enables seamless communication between tenants, building managers, and maintenance staff while maintaining a complete audit trail of all maintenance activities.

The primary motivation for developing this system stems from the need to improve operational efficiency in residential property management. By automating the maintenance request workflow, the system reduces response times, ensures proper assignment of tasks

based on staff specialization, provides real-time status updates to all stakeholders, and generates comprehensive reports for management decision-making.

This project demonstrates the practical application of software engineering principles, including requirements analysis, system design using UML diagrams, database design and normalization, object-oriented programming with JavaFX, and user interface design for multiple user roles. The system is built to be scalable, maintainable, and user-friendly, ensuring that it can be adopted by residential properties of various sizes.

The development of this system has provided valuable learning experiences in full-stack application development, including front-end user interface design, back-end database management, business logic implementation, and integration of various system components. Through this project, we have gained practical insights into real-world software development challenges and solutions.

2.0 VISION, BUSINESS CASE AND SCOPE

2.1 VISION

Our vision for the Residential Maintenance Management System is to create an efficient, transparent, and user-friendly platform that transforms the way residential properties handle maintenance operations. We envision a system where tenants can easily report issues, building managers can efficiently coordinate maintenance activities, and maintenance staff can effectively manage their workload, all while maintaining clear communication and accountability throughout the process.

The long-term vision includes expanding the system to support multiple properties under a single management company, integrating with IoT devices for predictive maintenance, providing mobile applications for on-the-go access, and incorporating artificial intelligence for intelligent task assignment and priority prediction.

2.2 BUSINESS CASE

The Residential Maintenance Management System addresses several critical business needs in property management:

- **Improved Efficiency**: Automating the maintenance request workflow reduces manual paperwork and eliminates lost requests, leading to faster response times and higher tenant satisfaction.
- **Cost Reduction**: Better tracking and resource allocation minimize unnecessary expenses and optimize maintenance staff utilization.
- **Enhanced Communication**: Real-time notifications and status updates keep all stakeholders informed, reducing phone calls and emails.
- **Accountability and Documentation**: Complete audit trails ensure accountability and provide documentation for disputes, insurance claims, or compliance requirements.

- **Data-Driven Decision Making**: Comprehensive reporting and analytics help building managers identify recurring issues, evaluate staff performance, and make informed decisions about property maintenance investments.

- **Tenant Retention**: Faster response times and better communication improve tenant satisfaction, leading to higher retention rates and positive property reputation.

2.3 SCOPE

The Residential Maintenance Management System encompasses the following functional areas:

In Scope:

- User authentication and authorization for four roles: Admin, Tenant, Building Manager, and Maintenance Staff

- Tenant functionality to submit maintenance requests with descriptions, categories, and photo attachments

- Tenant functionality to view their own maintenance request history and status

- Building Manager functionality to review pending maintenance requests and assign them to appropriate staff

- Building Manager functionality to monitor work progress and generate reports

- Maintenance Staff functionality to view assigned work orders, update status, and upload completion photos

- Admin functionality to manage user accounts, buildings, apartments, and system settings

- Database management for storing users, buildings, apartments, maintenance requests, work orders, and photos

- Automatic priority calculation based on request category and urgency

- Notification system to inform stakeholders of status changes

- Reporting and analytics dashboard for building managers

Out of Scope:

- Financial management and billing integration

- Vendor management for external contractors

- Inventory management for maintenance parts and supplies

- Mobile application versions (initially desktop-only)

- Integration with external property management systems

- Real-time chat functionality between users

3.0 FEASIBILITY STUDIES

3.1 TECHNICAL FEASIBILITY

The technical feasibility study evaluates whether the proposed system can be successfully developed and deployed using available technology and resources.

Technology Stack:

The system is built using JavaFX, a modern framework for building rich desktop applications with Java. JavaFX provides excellent support for creating responsive user interfaces, handling multimedia content such as images, and integrating with databases through JDBC. Java is a mature, well-documented platform with extensive community support and libraries.

Development Tools:

The development team has access to IntelliJ IDEA, a professional integrated development environment that provides excellent support for JavaFX development, including visual design tools, debugging capabilities, and version control integration. The project repository is hosted on GitHub, enabling collaborative development and version management.

Database Technology:

The system uses a relational database management system (RDBMS) to store all application data. The database schema has been designed following normalization principles to ensure data integrity and efficiency. JDBC provides reliable connectivity between the Java application and the database.

System Requirements:

The application requires Java Runtime Environment (JRE) version 21 or higher, which is freely available and widely adopted. The system can run on Windows, macOS, and Linux operating systems, making it accessible to a broad user base.

Technical Risks:

The primary technical risks include ensuring proper JavaFX runtime component configuration, which has been addressed through documented IntelliJ run configurations, managing concurrent database access when multiple users interact with the system simultaneously, and ensuring scalability as the number of users and maintenance requests grows over time.

Conclusion:

The technical feasibility analysis confirms that the Residential Maintenance Management System can be successfully developed using the chosen technology stack. The development team possesses the necessary technical skills, and all required tools and platforms are readily available.

3.2 ECONOMIC FEASIBILITY

The economic feasibility study examines whether the project is financially viable and provides sufficient return on investment.

Development Costs:

Development costs are minimal as the project uses open-source and free technologies, including Java and JavaFX, which require no licensing fees. The IntelliJ IDEA Community Edition provides all necessary development tools at no cost. GitHub offers free repository hosting for open-source projects. The primary cost is the time invested by the development team.

Deployment Costs:

Deployment costs include server infrastructure for hosting the database (if using a centralized server approach) and distributing the application to end users. For small to medium-sized residential properties, these costs are minimal and can be accommodated within typical IT budgets.

Benefits:

The system provides substantial economic benefits, including reduced administrative overhead through automation of manual processes, decreased response times leading to lower tenant turnover costs, optimized maintenance staff utilization reducing overtime expenses, and better documentation preventing disputes and potential legal costs.

Return on Investment:

Property management companies can expect to see return on investment through increased operational efficiency within the first year of deployment. Time savings alone, achieved through eliminating paper-based processes and reducing communication overhead, can offset the implementation costs.

Conclusion:

The economic feasibility analysis indicates that the Residential Maintenance Management System is financially viable with minimal upfront investment and substantial long-term benefits. The low development and deployment costs, combined with significant operational savings, make this project economically sound.

3.3 OPERATIONAL FEASIBILITY

The operational feasibility study assesses whether the proposed system will function effectively within the target organization and be accepted by users.

User Acceptance:

The system is designed with user-friendly interfaces tailored to each role. Tenants benefit from a simple, intuitive process for submitting maintenance requests without technical knowledge. Building managers gain centralized visibility into all maintenance operations, improving their ability to manage properties effectively. Maintenance staff receive clear work assignments with all necessary information, reducing confusion and improving job satisfaction. Administrators have comprehensive control over system configuration and user management.

Training Requirements:

The system's intuitive design minimizes training requirements. Basic computer literacy is sufficient for all user roles. Training can be conducted through simple demonstrations and

user manuals. The clear visual design and logical workflow make the system self-explanatory for most users.

Organizational Impact:

Implementing the system requires organizational changes, including transition from paper-based or email-based processes to the digital system, establishment of clear policies for request categorization and priority levels, and definition of response time expectations for different request types. These changes are manageable and bring long-term benefits.

System Maintenance:

The system is designed for easy maintenance with modular architecture allowing updates without disrupting operations, comprehensive documentation for future developers, and database backup procedures to prevent data loss.

Legal and Regulatory Compliance:

The system complies with data protection requirements by securely storing user information, provides audit trails for accountability and compliance purposes, and can be configured to meet specific property management regulations.

Conclusion:

The operational feasibility analysis confirms that the Residential Maintenance Management System can be successfully integrated into residential property management operations. The user-friendly design, minimal training requirements, and clear benefits ensure high user acceptance and operational success.

4.0 METHODOLOGY

The development of the Residential Maintenance Management System follows the Software Development Life Cycle (SDLC) approach, specifically using an iterative and incremental methodology. This approach allows for continuous refinement and testing throughout the development process.

Phase	Description
Requirements Analysis	Gathering and documenting system requirements from stakeholders
System Design	Creating UML diagrams, database schema, and interface designs
Implementation	Coding the application using JavaFX and Java
Testing	Unit testing, integration testing, and user acceptance testing
Deployment	Installing and configuring the system for production use
Maintenance	Ongoing support, bug fixes, and enhancements

Table 1: Development Methodology Phases

Requirements Analysis:

The requirements analysis phase involved identifying all stakeholders (tenants, building managers, maintenance staff, and administrators) and conducting interviews to understand their needs and pain points. We documented functional requirements detailing what the system should do, and non-functional requirements specifying performance, security, and usability standards. This phase resulted in a comprehensive requirements specification document.

System Design:

The system design phase focused on creating detailed visual representations of the system architecture and behavior. We developed use case diagrams to illustrate interactions between users and the system for each major function. Sequence diagrams show the flow of operations and message passing between system components. Activity diagrams depict the workflow for complex processes such as request assignment and status tracking. The database design phase involved creating an entity-relationship diagram and normalizing the database schema to ensure data integrity.

Implementation:

The implementation phase involved setting up the development environment with IntelliJ IDEA and configuring JavaFX runtime components. We implemented the database layer with DAO (Data Access Object) pattern for clean separation of concerns. Business logic was developed to handle request processing, assignment algorithms, and priority calculations. User interface components were created using JavaFX FXML for declarative UI design and JavaFX controllers for handling user interactions and updating the view.

Testing:

Testing was conducted at multiple levels including unit testing of individual methods and classes, integration testing to verify component interactions, database testing to ensure data integrity, and user interface testing for usability and responsiveness. Test scenarios covered normal operations, edge cases, error handling, and concurrent user access.

Deployment:

Deployment involves packaging the application as an executable JAR file, configuring the database connection parameters, setting up user accounts and initial data, and providing user documentation and training materials.

Maintenance:

Ongoing maintenance includes monitoring system performance, addressing bug reports, implementing user feedback, and adding new features based on evolving requirements.

This methodology ensures a systematic approach to development with clear milestones, continuous testing and quality assurance, flexibility to incorporate changes, and comprehensive documentation at each phase.

5.0 SYSTEM DESIGN

The system design section presents the visual models and architectural diagrams that represent the structure and behavior of the Residential Maintenance Management System.

5.1 USE CASE DIAGRAMS

Use case diagrams illustrate the interactions between different actors (users) and the system functionalities. The RMMS has four primary actors: Admin, Tenant, Building Manager, and Maintenance Staff.

Figure 1: Admin Use Case Diagram

Admin Use Case Diagram:

The Admin use case diagram (Figure 5.1) shows the administrative functions available to system administrators. The admin can login to the system using their credentials, view the admin dashboard displaying system overview and statistics, manage users by creating, updating, and deleting user accounts for tenants, building managers, and maintenance staff, manage buildings by adding new buildings and editing building details, manage apartments by assigning apartments to buildings and updating apartment information, assign building managers to specific buildings, view system reports including user activity and system performance, configure system settings such as request categories and priority levels, and manage maintenance staff by adding staff members and specifying their specializations.

Figure 2: Tenant Use Case Diagram

Tenant Use Case Diagram:

The Tenant use case diagram (Figure 5.2) illustrates the functions available to tenants. A tenant can login to the system, view their dashboard showing active and past maintenance requests, submit a maintenance request by providing a description, selecting a category, setting priority level, and uploading photos, view their own requests with detailed status information, track request status to see if it has been assigned, in progress, or completed, view work order details including assigned maintenance staff and scheduled date, view apartment information including building details and manager contact information, update their profile with contact information, receive notifications about request status changes, and logout from the system.

Figure 3: Building Manager Use Case Diagram

Building Manager Use Case Diagram:

The Building Manager use case diagram (Figure 5.3) shows the functions available to building managers. A building manager can login to the system, view the manager dashboard displaying pending requests and staff workload, review maintenance requests for their assigned buildings, approve or reject maintenance requests based on validity, create work orders by assigning approved requests to maintenance staff, monitor work progress by viewing status updates from maintenance staff, view all building requests with filtering and sorting options, generate reports on maintenance activities, costs, and staff performance, view staff workload to balance task assignments, manage apartments within their buildings, view building information including tenant list and apartment details, send notifications to tenants and staff, update request status manually if needed, and logout from the system.

Figure 4: Maintenance Staff Use Case Diagram

Maintenance Staff Use Case Diagram:

The Maintenance Staff use case diagram (Figure 5.4) depicts the functions available to maintenance staff members. A maintenance staff member can login to the system, view the staff dashboard showing assigned work orders, view assigned work orders with full details including location, description, and priority, accept a work order to begin work, update work order status to reflect progress (in progress, completed, etc.), upload completion photos as proof of work completed, view work order details including tenant contact information and request history, view apartment location to navigate to the maintenance site, view request history to understand recurring issues, report issues if additional work or specialist is needed, update their profile including availability status and contact information, and logout from the system.

5.2 SEQUENCE DIAGRAMS

Sequence diagrams show the interactions between objects and the order in which these interactions occur. They illustrate the flow of messages and operations for specific use cases.

Figure 5: Tenant Submit Maintenance Request Sequence Diagram

Tenant Submit Maintenance Request:

The sequence diagram (Figure 5.5) illustrates the process when a tenant submits a new maintenance request. The tenant logs into the system by providing username and password. The UI Controller forwards credentials to the Authentication Service. The Authentication Service validates the credentials against the Database. Upon successful authentication, the user object is returned and the tenant dashboard is displayed. The tenant clicks the "Create Request" button, and the system shows the request form. The tenant enters details including description and category, and optionally uploads photos. For each uploaded photo, a Photo object is created and stored in the Database. The UI Controller calls the Maintenance Request service to create the request. The Maintenance Request generates a unique request ID, calculates priority based on category and urgency, and sets the initial status to SUBMITTED. The request is saved to the Database, and a notification is sent to the Building Manager. The tenant receives confirmation with the Request ID displayed. This sequence ensures that all maintenance requests are properly logged and immediately communicated to the appropriate building manager.

Figure 6: Tenant View Request Status Sequence Diagram

Tenant View Request Status:

The sequence diagram (Figure 5.6) shows how a tenant checks the status of their maintenance requests. The tenant accesses "My Requests" section from the dashboard. The UI Controller retrieves requests by tenant ID from the Database, and the system returns a list of maintenance requests. The tenant sees the request list displayed with basic information. The tenant selects a specific request to view details. The UI Controller retrieves full request details including assigned staff, photos, and status history. If the request has an associated work order, the system retrieves work order information including assigned staff and scheduled date. The work order details are displayed, showing which maintenance staff member is assigned, current progress status, and completion information if applicable. This sequence provides tenants with full visibility into the status and progress of their maintenance requests.

Figure 7: Building Manager Review and Assign Request Sequence Diagram

Building Manager Review and Assign Request:

The sequence diagram (Figure 5.7) illustrates the process of reviewing and assigning maintenance requests. The building manager views all pending requests from their dashboard. The UI Controller retrieves pending requests from the Database and displays them. The manager selects a request to review, and the system displays full request details including tenant information, description, photos, and submission date. The manager assesses the priority level and can update it if necessary based on urgency, safety concerns, or tenant situation. The manager reviews available maintenance staff by querying staff availability and specialization from the Database. The manager selects an appropriate staff member considering their specialization match to the request category, current workload, and location proximity. The manager assigns the request to the selected staff member. The system creates a work order with the assigned staff, generates a work order ID, and sets the status to ASSIGNED. The Database is updated with the new work order and updated request status. Notifications are sent to both the maintenance staff member informing them of the new assignment and the tenant confirming that their request has been assigned. This sequence ensures proper request evaluation and optimal staff assignment.

Figure 8: Admin User Management Sequence Diagram

Admin User Management:

The sequence diagram (Figure 5.8) shows the process of managing user accounts in the system. The admin accesses the user management section from the admin dashboard. The UI Controller retrieves the list of all users from the Database across all roles including tenants, building managers, and maintenance staff. The user list is displayed with options to create, update, or delete users. When creating a new user, the admin clicks "Add New User" and a form is displayed. The admin enters user details and selects the role (Tenant, Building Manager, or Maintenance Staff). For tenants, the admin assigns them to a specific apartment. For building managers, the admin may assign them to manage specific buildings. For maintenance staff, the admin specifies their specialization. The system validates the entered information and creates the user record in the Database. The new user is saved, and confirmation is displayed. For updating a user, the admin selects the user from the list and modifies their information. For deleting a user, the admin selects the user and confirms deletion. This sequence provides comprehensive user account management capabilities.

Figure 9: Maintenance Staff Work on Assigned Request Sequence Diagram

Maintenance Staff Work on Assigned Request:

The sequence diagram (Figure 5.9) illustrates how maintenance staff members interact with their assigned work orders. The staff member logs into the system and views the staff dashboard. The UI Controller retrieves assigned work orders from the Database showing requests assigned to this staff member. The dashboard displays the work orders with priority and location information. The staff member selects a request to work on. The UI Controller retrieves full work order details including maintenance request information, apartment location, tenant contact details, and request photos. The staff member updates the status to IN PROGRESS, and notifications are sent to the building manager and tenant. The staff member travels to the location and assesses the issue. The staff member adds initial assessment notes to the work order. If parts are needed, the staff member identifies required parts and requests them. The system waits for parts to arrive. Once parts arrive, the staff

member updates status notes and sets an expected completion date. The staff member performs the repair work and updates progress notes regularly during the work. When work is completed successfully, the staff member enters the actual cost of materials and labor, enters resolution notes describing the work performed, takes completion photos showing the finished work, and uploads photos to the system. The staff member updates the status to COMPLETED and sets the completion date. Notifications are sent to both the building manager and tenant informing them that the work is complete. This comprehensive sequence ensures proper documentation of all maintenance activities.

Figure 10: Building Manager View Reports and Analytics Sequence Diagram

Building Manager View Reports and Analytics:

The sequence diagram (Figure 5.10) shows how building managers access reporting and analytics capabilities. The manager navigates to the reports section from the dashboard. The UI Controller provides report type options including reports by request status, reports by staff performance, and cost analysis reports. The manager selects a report type, such as "Report by Status". The UI Controller retrieves relevant data from the Database including request counts by status, average resolution time, and status distribution. The Report Generator processes the data and generates visual representations including charts and summary statistics. The generated report is displayed to the manager showing trends and insights. The manager can select another report type such as "Staff Performance" which shows completion rates for each staff member, average time per request, and workload distribution. The manager reviews the analytics to make informed decisions about resource allocation, identify high-performing staff members, and detect recurring maintenance issues that may require preventive measures.

5.3 ACTIVITY DIAGRAMS

Activity diagrams represent workflows and business processes, showing the sequence of activities and decision points in the system.

Figure 11: Admin Activity Diagram

Admin Activity Diagram:

The Admin Activity Diagram (Figure 5.11) illustrates the various workflows available to system administrators. The admin logs in with admin credentials and accesses the admin dashboard. From the dashboard, the admin can perform multiple activities. For user management, the admin can add a new user by selecting user type (Tenant, Manager, Staff), entering user details including name and contact information, assigning the user to a building or apartment, and saving the user. The admin can also view the user list, search for specific users, select a user to update, and modify user information, or select a user to delete and confirm deletion. For building management, the admin can add a new building by entering building details such as name, address, and total units, assigning a manager to the building, and saving the building information. The admin can also view existing buildings and update building information or display maintenance history. For system configuration, the admin can access system settings, configure email settings and notification preferences, update role-specific data such as request categories, and save the configuration. For reports and analytics, the admin can generate user activity reports showing login frequency and user engagement, export reports to PDF or Excel format, and review system analytics. After completing tasks, the admin can return to the dashboard or logout from the system.

Figure 12: Building Manager Activity Diagram

Building Manager Activity Diagram:

The Building Manager Activity Diagram (Figure 5.12) shows the workflow for building managers in handling maintenance requests. The manager logs into the system and accesses the manager dashboard. The dashboard shows pending requests and active work orders. The manager has two main workflow branches: reviewing requests and monitoring progress.

For reviewing requests, the manager views all pending requests and checks if new requests are available. If yes, the manager selects a request to review. The manager views request details including apartment number, description, photos, and submission date. The manager assesses the priority level and determines if the priority is correct. If not, the manager updates priority based on urgency, safety concerns, or tenant situation. The manager then determines required skills for the repair. The manager reviews available staff by querying all maintenance staff, checking staff availability, filtering by specialization matching the request category, and checking current workloads. The system checks if suitable staff is available. If yes, the manager ranks staff by specialization match, current workload, location proximity, and past performance, then selects the best fit staff member. The manager assigns the request to the staff member, generates a work order with request details and staff assignment, sets the status to ASSIGNED, updates staff workload records, and sends notifications to both the staff member and tenant. If no staff is available, the manager checks if external contractors are needed. If yes, the manager assigns the work to a contractor. If not, the manager escalates to senior manager for additional resources. If it is not an emergency request, the manager adds the request to a pending queue, sets an expected assignment date, and notifies the tenant of the delay. The manager returns to review more requests.

For monitoring progress, the manager views all active requests, checks request updates, and retrieves status updates from staff. The manager checks if the status is COMPLETED. If yes, the manager reviews completion details including work performed, parts used, photos, and estimated completion. The manager checks the actual cost and verifies if cost is approved. If additional approval is needed, the manager requests escalation and adds notes for specialist or budget approval. If approved, the manager marks the request as closed, notifies the tenant of completion, and displays a success message. If the status is IN PROGRESS, the manager monitors progress by checking staff notes and viewing estimated completion dates. The manager continues to check other active requests.

The manager can also generate reports by selecting report type such as by status, by staff performance, or cost analysis. The system generates the report, and the manager reviews analytics. After completing activities, the manager can logout.

Figure 13: Maintenance Staff Activity Diagram

Maintenance Staff Activity Diagram:

The Maintenance Staff Activity Diagram (Figure 5.13) illustrates the workflow for maintenance staff members as they handle assigned work orders. The staff member logs into the system and accesses the staff dashboard. The dashboard shows assigned requests and current workload.

For viewing assignments, the staff member views assigned requests and checks their workload showing the current number of active requests. If the staff member has assigned

requests, they can prioritize by urgency or update their availability status, then proceed. Otherwise, they can logout.

For working on requests, the staff member selects a request to work on. The system displays request details including location (apartment and building), description of the issue, photos submitted by tenant, and priority level. The staff member updates status to IN PROGRESS, which triggers notifications to the tenant and manager. The staff member travels to the location and assesses the issue by examining the problem and adding initial assessment notes. The system checks if parts are needed. If yes, the staff member identifies required parts, requests parts through the system, and waits for parts to arrive. Once parts arrive, the staff member updates status notes and sets an expected completion date. If no parts are needed initially, the staff member proceeds directly. The staff member enters an estimated cost for materials and labor, and performs the repair work. The staff member updates progress notes with regular updates during the work. The system checks if work is completed successfully. If no and more time is needed, the staff member requests escalation and updates the expected completion date. If yes and the work is successful, the staff member enters the actual cost of the work, enters resolution notes describing what was done, takes completion photos showing the finished repair, and updates status to COMPLETED, setting the completion date. Notifications are sent to both the tenant and manager. The system checks if there are more assigned requests. If yes, the staff member continues to the next request. If no, the staff member can proceed.

At the end of the day, the staff member reviews completed work and updates their availability status. The system checks if the staff member is available for new assignments. If yes, their status is set to AVAILABLE. If not, they set status to UNAVAILABLE and can add a reason if needed. The staff member then logs out.

Figure 14: Tenant Activity Diagram

Tenant Activity Diagram:

The Tenant Activity Diagram (Figure 5.14) shows the complete workflow for tenants interacting with the maintenance system. The tenant logs into the system using their username and password and accesses the tenant dashboard. The system checks if the tenant has maintenance needs. If yes, the tenant navigates to create a request. The tenant fills out the request form by providing a description of the issue, selecting a category (Emergency, Plumbing, Electrical, HVAC, Appliances, etc.), and adding detailed notes. The system checks if the tenant has photos to upload. If yes, the tenant uploads photos by selecting images and attaching them to the request. If no, the tenant proceeds without photos. The tenant submits the request, and the system generates a unique request ID, calculates priority based on category and urgency, and sets status to SUBMITTED. The tenant receives confirmation showing the Request ID. The request is saved to the database, and a notification is sent to the building manager. After submission, the tenant can check request status by navigating to "My Requests", viewing all submitted requests, and selecting a request to view details. The system checks if the status has changed. If the status is ASSIGNED, the tenant views assigned staff information showing the name and contact details of the assigned maintenance staff member. If the status is IN PROGRESS, the tenant views progress notes showing updates from the maintenance staff. If the status is COMPLETED, the tenant views resolution details including what work was performed, views the actual cost, and checks if satisfied with the work. If yes, the tenant archives the request. If no, the tenant contacts the building manager to request follow-up. After checking status, the tenant waits for updates and checks if the request is still open. If yes, the tenant can continue monitoring. If no, the tenant returns to the dashboard or logs out.

Figure 15: Tenant Request Tracking Activity Diagram

Tenant Request Tracking Activity Diagram:

The Tenant Request Tracking Activity Diagram (Figure 5.15) provides a detailed view of how tenants monitor their maintenance requests. The tenant logs into the system and navigates to "My Requests" section. The tenant views all submitted requests displayed with basic information. The system provides filtering options, and the tenant can apply filters by status (SUBMITTED, ASSIGNED, IN PROGRESS, COMPLETED, CANCELLED), by date range, by priority level, or by category. If the tenant applies a filter, the system displays the filtered list. Otherwise, the system displays all requests. The tenant selects a request to view from the list. The system displays request details including Request ID, submission date, current status, priority level, category, description, photos attached, assigned staff information, progress notes, scheduled date, and completion information if finished. The system checks the request status. If SUBMITTED, the tenant sees "Wait for Assignment" status. If ASSIGNED, the tenant views staff information and views the scheduled date. If IN PROGRESS, the tenant views progress updates including staff notes on the work, views the estimated cost, and can view staff notes for additional details. If COMPLETED, the tenant views resolution notes describing the completed work, views the actual cost of the repair, and views the completion date. The tenant checks if satisfied. If yes, the tenant archives the request to mark it as reviewed. If no, the tenant contacts the building manager by sending a message to request follow-up. If CANCELLED, the tenant views the cancellation reason. After reviewing request details, the tenant checks if they need to contact anyone about the request. If yes, the tenant sends a message to the manager with questions or concerns. The message is delivered, and the tenant waits for a response. If no contact is needed, the tenant returns to the dashboard to view other requests or logout.

Figure 16: Building Manager Staff Assignment Process Activity Diagram

Building Manager Staff Assignment Process Activity Diagram:

The Building Manager Staff Assignment Process Activity Diagram (Figure 5.16) shows the detailed logic for assigning maintenance requests to staff members. The process begins when the manager receives a new request notification. The manager opens the request details and enters the "Evaluate Request" phase. The manager reviews the description and photos submitted by the tenant and checks the priority level. The system checks if priority is correct based on the issue severity. If not correct, the manager adjusts priority based on urgency (immediate, within 24 hours, within a week), safety concerns (electrical hazards, water leaks, structural issues), and tenant situation (vulnerable tenants, special needs). After priority adjustment or confirmation, the manager determines the required skills needed for the repair.

The process enters the "Find Available Staff" phase. The manager queries all maintenance staff from the system and checks staff availability. The system filters by specialization to match skills to the request category (plumbing for water issues, electrical for power problems, HVAC for heating/cooling, general for basic repairs). The manager checks the current workloads of available staff. The system checks if suitable staff is available. If yes, the manager ranks staff by specialization match (exact match preferred), current workload (fewer active requests preferred), location proximity (closer staff for faster response), and past performance (completion rate and quality ratings). The manager selects the best fit staff member from the ranked list.

If no suitable staff is available, the system checks if this is an emergency request. If yes, the manager checks external contractors for availability. If a contractor is available, the manager assigns the work to the contractor. If no contractor is available, the manager escalates to senior manager for urgent resource allocation. If the request is not an emergency, the manager adds it to a pending queue, sets an expected assignment date based on staff availability forecasts, and notifies the tenant of the delay with an explanation.

Once staff is selected, the process enters the "Assign Request" phase. The manager creates an assignment including generating a work order with a unique work order ID, copying request details (description, photos, location), recording the staff assignment (staff member name and ID), and setting a schedule (expected start date and time). The manager sets the request status to ASSIGNED and updates the staff workload by incrementing their active request count. The system sends notifications including a notification to the staff member with request details, priority level, location, and contact information, and a notification to the tenant with assignment confirmation, staff details, and expected timeline. The manager saves the assignment to the database and adds it to the staff member's active queue. The manager monitors the assignment status by checking for updates and waiting for progress reports from the staff member.

5.4 DATABASE DESIGN

The database design section presents the data model and schema for the Residential Maintenance Management System.

Figure 17: Entity Relationship Diagram

Entity Relationship Diagram:

The Entity Relationship (ER) Diagram (Figure 5.17) shows the relationships between the main entities in the system. The diagram illustrates the following key relationships:

The **User** entity is the superclass with attributes including id (Primary Key), name, email, password, and role (Admin, Manager, Staff, Tenant). User has subtypes including **Admin** with admin-specific fields, **MaintenanceStaff** with specialty attribute, **BuildingManager** with manager-specific fields, and **Tenant** with phone attribute.

The **WorkOrder** entity has attributes including id (Primary Key), status (Open, InProgress, Completed, Cancelled), priority (Low, Medium, High, Urgent), description, request_id (Foreign Key), staff_id (Foreign Key), and admin_id (Foreign Key). WorkOrder fulfills MaintenanceRequest, is assigned to MaintenanceStaff, and is created by Admin.

The **MaintenanceRequest** entity has attributes including id (Primary Key), category (Plumbing, Electrical, General, Appliances), status (Pending, Approved, Denied, Completed), description, date_created, tenant_id (Foreign Key), and apt_id (Foreign Key). MaintenanceRequest has Photo entities, belongs to Tenant, and is for Apartment.

The **Photo** entity has attributes including id (Primary Key), url, and request_id (Foreign Key).

The **Building** entity has attributes including id (Primary Key), name, address, and manager_id (Foreign Key). Building contains Apartment entities and is managed by BuildingManager.

The **Apartment** entity has attributes including id (Primary Key), number, building_id (Foreign Key), and tenant_id (Foreign Key). Apartment is rented by Tenant.

The relationships show that a tenant can create multiple maintenance requests, each maintenance request belongs to one apartment, a building manager manages one or more buildings, a building contains multiple apartments, maintenance staff can be assigned to multiple work orders, each work order fulfills one maintenance request, and maintenance requests can have multiple photos attached.

Figure 18: UML Class Diagram

UML Class Diagram:

The UML Class Diagram (Figure 5.18) shows the object-oriented design of the system with classes, attributes, and methods. Key classes include:

User (base class) with attributes id, name, email, password, role, and methods getRole(), getName(), getEmail(), authenticatePw(password).

Admin extends User with method manageUsers().

MaintenanceStaff extends User with attribute specialty and method updateWorkOrder().

BuildingManager extends User with method assignStaff().

Tenant extends User with attribute phone and method submitRequest().

MaintenanceRequest with attributes id, category, status, description, date_created, tenant, and apt. Related to Photo (many), Apartment, and Tenant.

WorkOrder with attributes id, status, priority, description, request, staff, admin. Related to MaintenanceRequest (one-to-one), created by Admin, and assigned to MaintenanceStaff.

Photo with attributes id, url, request.

Apartment with attributes id, number, building, tenant. Belongs to Building, rented by Tenant.

Building with attributes id, name, address, manager, apartments. Managed by BuildingManager, contains many Apartments.

The diagram also shows DAO (Data Access Object) classes including TenantDAO, MaintenanceRequestDAO, WorkOrderDAO, AdminDAO, MaintenanceStaffDAO, ManagerDashboardController, ApartmentDAO, BuildingDAO, PhotoDAO, and LogicController. Additional service classes include AuthenticationService, NotificationService, Email, ViewFactory, IDGenerator, DatabaseManager, and DatabaseInitializer.

Figure 19: Database Schema Diagram

Database Schema Diagram:

The Database Schema Diagram (Figure 5.19) shows the detailed table structures with data types and constraints.

User Table:

- userId: varchar(50) NOT NULL, Primary Key
- username: varchar(50) NOT NULL
- password: varchar(255) NOT NULL
- firstName: varchar(50) NOT NULL
- lastName: varchar(50) NOT NULL
- email: varchar(100) NOT NULL
- phoneNumber: varchar(20)
- dateCreated: timestamp
- lastLogin: timestamp
- isActive: boolean

Tenant Table:

- tenantId: varchar(50) NOT NULL, Primary Key, Foreign Key to User
- userId: varchar(50) NOT NULL, Foreign Key to User
- apartmentNumber: varchar(20)
- leaseStartDate: date
- leaseEndDate: date
- emergencyContact: varchar(100)
- emergencyPhone: varchar(20)

BuildingManager Table:

- managerId: varchar(50) NOT NULL, Primary Key, Foreign Key to User
- userId: varchar(50) NOT NULL, Foreign Key to User
- department: varchar(100)
- yearsOfExperience: int

MaintenanceStaff Table:

- staffId: varchar(50) NOT NULL, Primary Key, Foreign Key to User
- userId: varchar(50) NOT NULL, Foreign Key to User
- specialization: varchar(100)
- certifications: text
- hourlyRate: decimal(10,2)
- availability: varchar(50)

Admin Table:

- adminId: varchar(50) NOT NULL, Primary Key, Foreign Key to User
- userId: varchar(50) NOT NULL, Foreign Key to User
- accessLevel: varchar(50)

Building Table:

- buildingId: varchar(50) NOT NULL, Primary Key
- managerId: varchar(50), Foreign Key to BuildingManager
- buildingName: varchar(100) NOT NULL
- address: varchar(255) NOT NULL
- totalUnits: int NOT NULL
- buildingType: varchar(50)
- constructionYear: int

Apartment Table:

- apartmentId: varchar(50) NOT NULL, Primary Key
- buildingId: varchar(50), Foreign Key to Building
- currentTenantId: varchar(50), Foreign Key to Tenant
- apartmentNumber: varchar(20) NOT NULL
- floorPlan: varchar(50)
- squareFootage: int
- monthlyRent: decimal(10,2)
- leaseStatus: varchar(20)

MaintenanceRequest Table:

- requestId: varchar(50) NOT NULL, Primary Key
- tenantId: varchar(50) NOT NULL, Foreign Key to Tenant
- assignedStaffId: varchar(50), Foreign Key to MaintenanceStaff
- apartmentNumber: varchar(20) NOT NULL
- description: text NOT NULL
- detailedDescription: text
- category: varchar(50) NOT NULL
- priority: varchar(20) NOT NULL
- status: varchar(30)
- submissionDate: timestamp
- lastUpdated: timestamp
- scheduledDate: timestamp

- completionDate: timestamp
- estimatedCost: decimal(10,2)
- actualCost: decimal(10,2)
- workOrderNumber: varchar(50)
- staffUpdateNotes: text
- resolutionNotes: text
- tenantArchived: boolean
- staffArchived: boolean

WorkOrder Table:

- workOrderId: varchar(50) NOT NULL, Primary Key
- requestId: varchar(50) NOT NULL, Foreign Key to MaintenanceRequest
- assignedStaffId: varchar(50), Foreign Key to MaintenanceStaff
- scheduledDateTime: timestamp
- estimatedDuration: int
- partsRequired: text
- instructions: text
- status: varchar(30)

Photo Table:

- photoId: varchar(50) NOT NULL, Primary Key
- requestId: varchar(50) NOT NULL, Foreign Key to MaintenanceRequest
- fileName: varchar(255) NOT NULL
- filePath: varchar(500) NOT NULL
- uploadDate: timestamp
- description: varchar(255)

CategoryType Table:

- value: varchar(50), Primary Key

RequestStatus Table:

- value: varchar(30), Primary Key

WorkOrderStatus Table:

- value: varchar(30), Primary Key

PriorityLevel Table:

- value: varchar(20), Primary Key

LeaseStatus Table:

- value: varchar(20), Primary Key

The schema implements referential integrity through foreign key constraints and uses appropriate data types for each field to ensure data validity.

6.0 PROJECT SYSTEM

6.1 SYSTEM FUNCTIONALITY AND FUNCTIONAL REQUIREMENTS

The Residential Maintenance Management System provides comprehensive functionality across four user roles. The functional requirements are organized by user type:

Admin Functionality:

- User account management with the ability to create new accounts for all user types (Tenant, Building Manager, Maintenance Staff), update existing user information including contact details and role assignments, delete user accounts when necessary, and reset user passwords.

- Building and apartment management including adding new buildings with details such as name, address, and total units, updating building information, assigning building managers to specific buildings, adding apartments to buildings with unit numbers and details, and updating apartment information including tenant assignments.

- System configuration with the ability to configure request categories (Plumbing, Electrical, HVAC, Appliances, General), define priority levels and their criteria, set up notification preferences, and manage system-wide settings.

- Reporting and analytics access to view comprehensive system usage statistics, generate user activity reports, and monitor system performance metrics.

Tenant Functionality:

- Maintenance request submission with the ability to create new maintenance requests with detailed descriptions, select appropriate category from predefined list, specify urgency level, upload multiple photos to illustrate the issue, and receive confirmation with unique request ID.

- Request tracking and status monitoring including viewing all personal maintenance requests, checking current status (Submitted, Assigned, In Progress, Completed), viewing assigned maintenance staff details, seeing scheduled dates and estimated completion times, and viewing work progress notes and updates.

- Notification receipt for receiving automatic notifications when request status changes, when staff is assigned to the request, when work begins, and when work is completed.

- Profile management to update personal contact information and view apartment and building details.

Building Manager Functionality:

- Request review and approval with the ability to view all pending maintenance requests for managed buildings, review request details including photos and descriptions, assess and adjust priority levels based on urgency and circumstances, and approve or reject requests based on validity.

- Staff assignment and work order creation including viewing available maintenance staff with their specializations and current workload, assigning requests to appropriate staff members based on skills and availability, creating work orders with detailed instructions, setting scheduled dates and estimated durations, and balancing workload across maintenance staff.

- Progress monitoring and oversight to view all active work orders, monitor status updates from maintenance staff, review completion details and costs, and approve completed work or request additional actions.

- Reporting and analytics access to generate reports on maintenance activities by status, staff performance, cost analysis, and recurring issues, view staff workload distribution, analyze response times and completion rates, and export reports for management review.

- Building management including viewing all apartments and tenants in managed buildings, updating building information, managing apartment assignments, and sending notifications to tenants and staff.

Maintenance Staff Functionality:

- Work order management with the ability to view all assigned work orders, access detailed request information including location, description, priority, and photos, accept work orders to begin work, and organize work queue by priority and location.

- Status updates and progress reporting including updating work order status (In Progress, Completed), adding progress notes and updates during repair work, recording parts used and labor hours, entering actual costs upon completion, and uploading completion photos as proof of work.

- Communication capabilities to access tenant contact information for clarification if needed, report issues requiring specialist assistance or additional resources, and receive notifications of new assignments.

- Profile and availability management to update availability status (Available, Unavailable), specify reasons for unavailability, and update contact information.

6.2 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements specify the quality attributes and constraints of the system:

Performance:

- The system should load user dashboards within 2 seconds under normal network conditions.

- Database queries should execute within 1 second for typical operations.

- The system should support at least 50 concurrent users without performance degradation.
- Photo uploads should support files up to 5MB in size and complete within 10 seconds.

Usability:

- The user interface should be intuitive and require minimal training for basic operations.
- Clear visual feedback should be provided for all user actions including success and error messages.
- The system should follow consistent design patterns across all screens and user roles.
- Help documentation should be accessible from all main screens.

Security:

- User passwords must be securely hashed using industry-standard algorithms.
- Session management should automatically log out users after 30 minutes of inactivity.
- Role-based access control must prevent unauthorized access to restricted functions.
- All database transactions should be logged for audit purposes.
- Sensitive user information must be protected against unauthorized access.

Reliability:

- The system should have 99 percent uptime during business hours.
- Database backups should be performed daily to prevent data loss.
- Error handling should prevent system crashes and provide graceful degradation.
- Transaction rollback mechanisms should ensure data consistency in case of failures.

Maintainability:

- Code should follow object-oriented design principles for easy maintenance.
- Comprehensive inline documentation should explain complex logic.
- Modular architecture should allow updates to individual components without affecting others.
- Database schema should support future extensions without major restructuring.

Scalability:

- The system architecture should support addition of new user roles with minimal changes.

- Database design should handle growth in users, buildings, and maintenance requests.
- The system should support future integration with mobile applications.
- API design should enable future web service interfaces.

6.3 SYSTEM REQUIREMENTS

The system requirements specify the software and hardware needed to develop, deploy, and run the application.

6.3.1 SOFTWARE REQUIREMENTS

Component	Specification
Programming Language	Java SE 21 or higher
UI Framework	JavaFX 21
Database	Relational Database (MySQL, PostgreSQL, or SQLite)
Database Connectivity	JDBC (Java Database Connectivity)
Development IDE	IntelliJ IDEA Community or Ultimate Edition
Build Tool	Maven or Gradle
Version Control	Git
Operating System	Windows 10/11, macOS 10.15+, or Linux
Java Runtime	JRE 21 with JavaFX runtime components

Table 2: Software Requirements

Development Environment:

- IntelliJ IDEA provides comprehensive support for JavaFX development including visual FXML editors, integrated debugging tools, and version control integration.
- Git and GitHub enable collaborative development with version control, code review capabilities, and project documentation.

Runtime Environment:

- Java Runtime Environment (JRE) 21 or higher must be installed on client machines.
- JavaFX runtime components must be properly configured with module path settings.
- Database server must be accessible via network connection or local installation.

Libraries and Dependencies:

- JavaFX Controls and FXML for user interface components and declarative UI design.
- JDBC driver specific to the chosen database system.

- Additional libraries may include file I/O utilities for photo management and encryption libraries for secure password storage.

6.3.2 HARDWARE REQUIREMENTS

Component	Minimum Specification
Processor	Intel Core i3 or equivalent, 2.0 GHz or higher
RAM	4 GB minimum, 8 GB recommended
Storage	500 MB free disk space for application
	10 GB recommended for database and photos
Display	1366 x 768 resolution minimum
	1920 x 1080 recommended
Network	Ethernet or Wi-Fi connection for database access
	Minimum 10 Mbps for photo uploads

Table 3: Hardware Requirements

Client Workstation:

- Standard desktop or laptop computers meeting the minimum specifications can run the application effectively.
- Higher specifications improve performance, especially when handling multiple photos or generating reports.

Server Requirements (if using centralized database):

- Processor: Intel Xeon or equivalent multi-core processor.
- RAM: 8 GB minimum, 16 GB recommended for larger deployments.
- Storage: SSD recommended for database storage to improve query performance.
- Network: Gigabit Ethernet connection for optimal data transfer speeds.

6.4 SYSTEM IMPLEMENTATION

The system implementation section describes how the major functionalities are realized in the application.

Architecture Overview:

The application follows a three-tier architecture pattern:

- **Presentation Layer**: JavaFX-based user interface with FXML layouts and controller classes. Each user role has dedicated views and controllers.

- **Business Logic Layer**: Service classes and Data Access Objects (DAOs) that implement business rules and data operations. This layer handles request validation, priority calculation, staff assignment logic, and notification generation.

- **Data Layer**: Relational database accessed through JDBC. The database stores all persistent data including users, buildings, apartments, maintenance requests, and work orders.

Key Implementation Components:

Authentication and Authorization:

The system implements secure authentication with hashed password storage using industry-standard algorithms. Upon successful login, the user's role is determined, and the appropriate dashboard is displayed. Session management tracks the current user and enforces role-based access control throughout the application.

Maintenance Request Workflow:

When a tenant submits a maintenance request, the system generates a unique request ID using the IDGenerator service. The request priority is calculated based on the selected category and urgency level. Photos are stored in the file system with paths recorded in the database. A notification is sent to the assigned building manager using the NotificationService.

Staff Assignment Algorithm:

The building manager views pending requests and initiates the assignment process. The system queries available maintenance staff from the database, filtering by specialization matching the request category. Staff workload is calculated by counting active work orders. The manager reviews the filtered list showing each staff member's specialization, current workload, and past performance. Upon selection, a WorkOrder entity is created linking the request to the assigned staff member.

Status Tracking and Updates:

Throughout the maintenance process, status updates are recorded with timestamps. Tenants can view their request history and status at any time. Maintenance staff update work order status as they progress through the repair. Completion triggers notifications to both the tenant and building manager.

Reporting System:

The reporting module queries the database for relevant data based on selected criteria such as date range, building, staff member, or request status. Data is aggregated and formatted for display including charts showing status distribution, tables listing staff performance metrics, and cost analysis summaries. Reports can be exported to PDF or Excel format for external use.

Database Operations:

All database operations are performed through DAO (Data Access Object) classes that encapsulate SQL queries and result set processing. This design pattern provides separation of concerns, centralizes database logic, enables easy testing and maintenance, and allows

database changes without affecting business logic. Each entity type (User, Tenant, Building, MaintenanceRequest, WorkOrder) has a corresponding DAO class.

Photo Management:

Photos uploaded by tenants and maintenance staff are stored in a designated directory on the file system. The Photo table stores metadata including file path, request ID, upload date, and description. The system validates file types and sizes before upload. Photos are displayed in the application using JavaFX ImageView components.

Notification System:

The NotificationService handles all system notifications including email notifications for critical updates (optional feature), in-app notifications displayed when users log in, and notification logging in the database for audit trails. Notifications are triggered by events such as new request submission, request assignment, status changes, and work completion.

7.0 SYSTEM TESTING

System testing ensures that the Residential Maintenance Management System functions correctly and meets all specified requirements.

Testing Approach:

The testing strategy follows a comprehensive multi-level approach including unit testing, integration testing, system testing, and user acceptance testing.

Unit Testing:

Individual methods and classes are tested in isolation to verify correct behavior. Test cases cover normal input scenarios, edge cases and boundary conditions, error handling, and exception cases. Key components tested include authentication logic, priority calculation algorithms, database DAO methods, notification generation, and data validation routines. Unit tests use sample data and mock objects to simulate database interactions.

Integration Testing:

Integration testing verifies that system components work together correctly. Test scenarios include testing the flow from UI controllers to DAOs to database, verifying that authentication integrates properly with authorization and session management, confirming that maintenance request submission triggers appropriate notifications, ensuring that staff assignment updates both request and work order tables, and validating that status updates propagate correctly to all affected entities.

System Testing:

System testing evaluates the complete application in an environment similar to production. Test cases are organized by user role and use case:

- **Admin Testing**: Create user accounts of each type, assign managers to buildings, configure system settings, generate reports, and verify data consistency.
- **Tenant Testing**: Submit maintenance requests with various categories and priorities, upload photos of different sizes and formats, view request status and history, track assigned staff information, and verify notifications are received.

- **Building Manager Testing**: Review pending requests, assign requests to staff with different specializations, monitor work progress, generate various types of reports, approve completed work, and manage building information.

- **Maintenance Staff Testing**: View assigned work orders, update status through all stages (In Progress to Completed), add progress notes and photos, record actual costs, report issues requiring escalation, and update availability status.

Performance Testing:

Performance testing evaluates system responsiveness and scalability by measuring dashboard load times with varying amounts of data, testing concurrent user access with multiple simultaneous logins, measuring database query execution times, evaluating photo upload speeds with different file sizes, and stress testing with large datasets (1000+ requests, 100+ users).

Usability Testing:

Usability testing involves real users from each role performing typical tasks while observers note any difficulties, confusion, or inefficiencies. Feedback is collected on interface clarity, workflow logic, error message helpfulness, and overall user satisfaction. Based on findings, interface improvements are implemented including clearer labels, better visual feedback, streamlined workflows, and improved error messages.

Security Testing:

Security testing verifies that the system protects sensitive data and prevents unauthorized access by attempting login with invalid credentials, verifying password storage is properly hashed, testing role-based access control by attempting unauthorized operations, checking session timeout functionality, and validating input to prevent SQL injection attacks.

Bug Tracking and Resolution:

All identified bugs are logged with description, severity level, steps to reproduce, and expected versus actual behavior. High-severity bugs affecting core functionality are prioritized for immediate fixing. Medium and low-severity issues are addressed in subsequent development iterations. Fixes are verified through regression testing to ensure they don't introduce new issues.

Test Results Summary:

The testing process revealed and addressed several issues including minor UI inconsistencies that were corrected for visual consistency, edge cases in date handling that were fixed with proper validation, performance optimization opportunities in database queries that were improved with indexing, and usability improvements suggested by test users that were implemented. All critical functionality passed testing, and the system is considered ready for deployment.

8.0 SYSTEM LIMITATIONS

While the Residential Maintenance Management System provides comprehensive maintenance management capabilities, several limitations exist in the current implementation:

Technology Limitations:

- **Desktop-Only Application**: The current version is a JavaFX desktop application requiring installation on each user's computer. There is no web or mobile version, which limits accessibility for users who prefer to access the system from smartphones or tablets while on the go.

- **Single Database Instance**: The system is designed for a single database instance, which may create scalability challenges for very large property management companies with multiple geographically distributed locations.

- **Limited Concurrent Users**: While the system supports concurrent access, it is optimized for small to medium-sized residential properties (up to 50 concurrent users). Very large deployments may require additional optimization.

Functional Limitations:

- **No Financial Integration**: The system tracks estimated and actual costs but does not integrate with financial systems, accounting software, or payment processing. Manual transfer of financial data is required.

- **Limited External Contractor Support**: While the system allows building managers to note external contractor assignments, it does not provide comprehensive vendor management or contractor portal access.

- **No Inventory Management**: The system does not track parts inventory or automatically order replacement parts. Maintenance staff must manually manage supplies.

- **Basic Reporting**: While the system provides essential reports, advanced analytics features such as predictive maintenance, trend analysis, and forecasting are not included.

- **Single Property Focus**: Each system instance is designed for a single residential property or small portfolio. Large property management companies managing dozens of buildings may need multiple separate installations.

Communication Limitations:

- **No Real-Time Chat**: The system uses notifications but does not provide real-time chat functionality between tenants, managers, and staff. Users must use external communication methods (phone, email) for complex discussions.

- **Email Notifications Optional**: While the system generates in-app notifications, email notification functionality requires additional configuration and is not guaranteed delivery.

- **No SMS Notifications**: The system does not support SMS or text message notifications for urgent requests.

Data Limitations:

- **Photo Storage Only**: The system supports photo attachments but does not handle video files, which might be useful for demonstrating complex issues.

- **Limited Document Management**: The system does not provide general document storage for manuals, warranties, or building documentation.

- **No Historical Analytics**: Long-term trend analysis and historical comparison features are limited. The system focuses on current operational data.

Integration Limitations:

- **No IoT Integration**: The system does not integrate with smart building sensors or IoT devices that could automatically detect and report maintenance issues.

- **No Third-Party System Integration**: The system is standalone and does not integrate with property management systems, CRM platforms, or other business software.

- **No API for External Access**: There is no public API for external applications to interact with the system programmatically.

User Experience Limitations:

- **Fixed User Roles**: The system has four predefined roles with fixed permissions. Custom roles or granular permission configuration is not supported.

- **Limited Customization**: While admins can configure categories and priority levels, extensive customization of workflows or business rules requires code changes.

- **Desktop Environment Required**: Users must have access to a computer with Java installed. The system does not support thin clients or browser-based access.

These limitations are documented to set appropriate expectations and to guide future development priorities.

9.0 FUTURE ENHANCEMENTS

Future enhancements will address current limitations and expand the system's capabilities based on user feedback and technological advances:

Mobile Application Development:

- Develop native mobile applications for iOS and Android platforms to allow tenants to submit requests and track status from their smartphones.

- Enable maintenance staff to access work orders, update status, and upload photos while on-site using mobile devices.

- Implement push notifications for immediate alerts on mobile devices.

- Support offline mode for maintenance staff in areas with poor connectivity, with automatic synchronization when connection is restored.

Web-Based Interface:

- Create a responsive web application accessible through standard browsers without requiring software installation.

- Implement progressive web app (PWA) features for offline capability and app-like experience.
- Enable cloud hosting for multi-tenant deployment serving multiple properties from a single instance.

Advanced Analytics and Reporting:

- Implement predictive maintenance algorithms that analyze historical data to predict when equipment is likely to fail.
- Provide trend analysis showing patterns in maintenance requests over time, identifying seasonal issues or deteriorating building systems.
- Develop customizable dashboard widgets allowing users to configure their own view priorities.
- Add forecasting capabilities to predict future maintenance costs and resource needs.
- Implement machine learning algorithms to optimize staff assignment based on historical performance and success rates.

IoT Integration:

- Integrate with smart building sensors that automatically detect issues such as water leaks, HVAC failures, and electrical problems.
- Enable automated request creation when sensors detect anomalies.
- Provide real-time monitoring dashboards showing building system status.
- Support integration with smart locks and access control systems for secure property access by maintenance staff.

Enhanced Communication Features:

- Implement real-time chat functionality allowing tenants, managers, and staff to communicate within the application.
- Add video call support for remote diagnosis of issues.
- Enable SMS and email notifications with customizable delivery preferences.
- Provide automated status update broadcasts to keep all stakeholders informed.

Financial Management Integration:

- Integrate with accounting software such as QuickBooks or Xero for automatic financial data synchronization.
- Implement cost tracking and budget management features with alerts when costs exceed budgets.
- Support invoice generation for maintenance services.

- Enable payment processing for tenants paying maintenance fees or repair costs.

Inventory Management:

- Add parts inventory tracking to monitor stock levels of commonly used maintenance supplies.
- Implement automatic reordering when inventory falls below threshold levels.
- Track parts usage by request to improve cost accuracy.
- Generate inventory reports showing consumption patterns and cost optimization opportunities.

Vendor Management:

- Create a vendor portal for external contractors to receive assignments and update work status.
- Maintain contractor profiles with certifications, specializations, and performance ratings.
- Implement contractor bidding functionality for major repairs.
- Track vendor performance and costs for informed decision-making.

Advanced Scheduling:

- Implement calendar-based scheduling with drag-and-drop assignment capabilities.
- Support recurring maintenance tasks and preventive maintenance schedules.
- Enable automated scheduling optimization based on staff availability, location, and workload.
- Provide schedule conflict detection and resolution.

Multi-Property Portfolio Management:

- Extend the system to support property management companies managing multiple buildings or properties.
- Provide consolidated reporting across all properties.
- Enable resource sharing between properties when appropriate.
- Support hierarchical organization structures with regional managers and site managers.

Customization and Configuration:

- Allow administrators to define custom workflows for different request types.
- Enable configurable approval chains for high-cost repairs.

- Support custom fields and forms for specific property needs.
- Provide template library for common maintenance scenarios.

Enhanced Security Features:

- Implement two-factor authentication for enhanced security.
- Add audit logging for all system activities.
- Provide role-based access control with granular permissions.
- Enable data encryption for sensitive information.

Tenant Portal Enhancements:

- Allow tenants to schedule amenity reservations (gym, party room, etc.) through the same interface.
- Enable document access for lease agreements and building policies.
- Provide community bulletin board for announcements and discussions.
- Support maintenance history viewing for transparency on past work in their unit.

Sustainability Features:

- Track energy consumption and environmental impact of maintenance activities.
- Provide green maintenance options and eco-friendly repair alternatives.
- Generate sustainability reports for environmentally conscious property management.

These enhancements will be prioritized based on user demand, resource availability, and strategic importance to create a comprehensive, modern property maintenance management platform.

10.0 PROJECT SCHEDULING

Project scheduling outlines the timeline and task allocation for the development of the Residential Maintenance Management System.

10.1 SCHEDULING TASKS AND DURATION

The project development was divided into distinct phases with specific tasks and durations:

Phase	Tasks	Duration
Requirements Analysis	Stakeholder interviews Requirements documentation Feasibility study	1 week

System Design	Use case diagram creation Sequence diagram creation Activity diagram creation Database design (ER and schema) UI mockups and wireframes	2 weeks
Database Implementation	Database schema implementation Sample data creation DAO class development	1 week
Core Development	User authentication system Admin functionality Tenant request submission Manager request assignment Staff work order management	3 weeks
UI Development	FXML layout creation Controller implementation CSS styling Navigation flow	2 weeks
Integration	Component integration Notification system Photo upload functionality	1 week
Testing	Unit testing Integration testing System testing Bug fixing	2 weeks
Documentation	Code documentation User manual creation Technical documentation Project report writing	1 week
Deployment	System packaging Installation testing User training preparation	1 week

Table 4: Project Timeline and Tasks

Total Project Duration: 14 weeks (approximately 3.5 months)

Team Size: 3 developers working collaboratively with task distribution based on expertise.

Development Methodology: Iterative and incremental approach with regular review meetings every week to assess progress, address challenges, and adjust priorities as needed.

10.2 PROJECT MILESTONES

Key milestones achieved during the project development:

- **Week 1**: Requirements specification completed and approved.
- **Week 3**: System design documents completed including all UML diagrams.
- **Week 4**: Database schema implemented and tested with sample data.
- **Week 7**: Core functionality completed for all user roles.
- **Week 9**: User interface development completed with all screens functional.
- **Week 10**: System integration completed with all components working together.
- **Week 12**: Testing phase completed with all critical bugs resolved.
- **Week 13**: Documentation completed including code comments and user manuals.
- **Week 14**: System deployed and ready for production use.

Risk Management:

Throughout the project, several risks were identified and mitigated including technical challenges with JavaFX configuration (addressed through team learning sessions and online resources), database design changes (managed through version control and careful migration planning), and scope creep (controlled through strict adherence to requirements specification and change control process).

Lessons Learned:

The project provided valuable learning experiences including the importance of thorough system design before coding, the value of iterative development with regular testing, the benefits of modular architecture for maintainability, and the need for comprehensive documentation throughout development.

10.3 PROJECT LOG

DATE	MEMBERS PRESENT	TASKS / ASSIGNMENT
09/08/25 [In Class]	All	• Choose Topic • Write a project description. • Discuss a project plan
09/15/25 [In Class]	Sandip, Bethlehem, Quin	• Organize Iteration 1 Report • Discuss UML Diagram • Discuss additional meeting plans.
09/20/25 [Zoom]	Tristan, Bethlehem, Quin	• Complete Iteration 1 Report • Complete Iteration 1 Slideshow
09/22/25 [In Class]	All	• Iteration 1 Presentation • Agree to halt application development to complete the topical presentation.
09/29/25 [In Class]	All	• Topical Presentation

10/06/25 [In Class]	All	• Assign Project Tasks and Individual Sprints:
		  ○ Login UI
		  ○ Dashboard UI
		  ○ Java Classes
		  ○ SQL Database

10/11/25 [Discord Check-in]	All	• Summarize Progress • Resolve Pull Requests
10/13/25 [In Class]	All	• Iteration 2 Presentation + Demo
10/14/25 [Zoom]	All	• Begin Sprint 1: Ticket edit functionality design and implementation
10/20/25 [In Class]	All	• Begin Sprint 2: Dashboard UI development with color coding and visual improvements
10/27/25 [In Class]	All	• Begin Sprint 3: Filtering and Easy Assign feature implementation

11/03/25 [In Class]	All	• Begin Sprint 4: Integration, comprehensive testing, bug fixes, and user acceptance testing.
11/10/25 [In Class]	All	• Iteration 3 Presentation + Demo
11/17/25 [In Class]	All	• Begin email feature
11/24/25 [In Class]	Sandip, Bethlehem, Quin	• Finalize email feature
12/01/25 [In Class]	All	• Finalize Report and Diagrams
12/08/25 [In Class]	All	• Iteration 4 Final Presentation + Demo

11.0 CONCLUSION

The Residential Maintenance Management System successfully addresses the critical challenges faced by residential properties in managing maintenance operations. Through systematic analysis, design, and implementation, we have created a comprehensive software solution that streamlines the maintenance request workflow from initial submission through completion.

Project Achievements:

The project has successfully achieved its primary objectives by developing a fully functional desktop application that serves four distinct user roles (Admin, Tenant, Building Manager, and Maintenance Staff), implementing a robust database design that ensures data integrity and supports efficient querying, creating intuitive user interfaces tailored to each role's specific needs, establishing automated workflows that reduce manual intervention and improve response times, providing comprehensive tracking and documentation of all maintenance activities, and enabling data-driven decision making through reporting and analytics capabilities.

The system demonstrates the practical application of software engineering principles learned throughout our academic program, including requirements analysis and documentation, object-oriented design and programming, database design and normalization, user interface design and usability considerations, software testing methodologies, and project management and scheduling.

Technical Accomplishments:

From a technical perspective, the project successfully implements a three-tier architecture separating presentation, business logic, and data layers. The use of JavaFX provides a modern, responsive user interface with rich functionality. The database design follows normalization principles ensuring data integrity and efficiency. The DAO pattern implementation provides clean separation of concerns and maintainability. The notification system keeps all stakeholders informed of relevant events and status changes.

Impact and Benefits:

The Residential Maintenance Management System provides significant benefits to all stakeholders. Tenants benefit from an easy-to-use system for reporting maintenance issues, transparent tracking of request status, faster response times, and improved communication with property management. Building managers gain centralized visibility into all maintenance operations, efficient tools for assigning and tracking work, comprehensive reporting for informed decision-making, and improved resource allocation and cost control. Maintenance staff receive clear work assignments with all necessary information, tools to document work progress and completion, reduced confusion and improved job satisfaction, and fair workload distribution based on skills and availability. Property owners and administrators benefit from increased operational efficiency and cost savings, improved tenant satisfaction and retention, comprehensive documentation for compliance and disputes, and data-driven insights for preventive maintenance planning.

Challenges Overcome:

During the development process, we encountered and successfully addressed several challenges including configuration of JavaFX runtime environment in IntelliJ, which was resolved through documented run configurations, designing an intuitive interface serving diverse user needs with different technical skill levels, implementing efficient database queries for complex reporting requirements, managing concurrent database access to prevent data conflicts, balancing feature richness with system simplicity and usability, and coordinating development efforts among team members with clear task allocation and version control.

Personal Growth and Learning:

This project provided invaluable hands-on experience in full-stack application development. We gained practical skills in translating requirements into working software, designing and implementing complex database schemas, creating user-friendly interfaces for diverse user groups, writing clean, maintainable, and well-documented code, conducting comprehensive testing at multiple levels, managing project timelines and deliverables, and collaborating effectively as a development team. Beyond technical skills, we developed problem-solving abilities, learned to make design tradeoffs between functionality and complexity, gained appreciation for the importance of user feedback and iterative development, and understood the value of thorough planning and documentation.

Project Success Factors:

Several factors contributed to the project's success including clear vision and well-defined requirements from the outset, systematic approach following the software development life cycle, comprehensive system design using UML diagrams before implementation, modular architecture facilitating parallel development and testing, regular progress reviews and course corrections, effective use of version control and collaboration tools, and dedication and teamwork of all project members.

Future Outlooks:

While the current system provides solid functionality for desktop environments, we recognize the growing need for mobile and web-based access. The future enhancements outlined in this report represent our vision for evolving the system into a comprehensive, cloud-based platform serving the modern property management industry. The modular architecture and clean code structure provide a strong foundation for these future developments.

Final Thoughts:

The Residential Maintenance Management System represents a significant accomplishment in applying computer science principles to solve real-world problems. The project demonstrates that well-designed software can substantially improve operational efficiency, enhance communication, and increase satisfaction for all stakeholders. As we complete this final year project, we are confident that the skills and knowledge gained will serve us well in our future careers as software developers. We are proud of what we have created and excited about the positive impact this system can have on residential property management operations.

This project has been a rewarding journey from concept to completion, and we are grateful for the opportunity to develop a practical, useful application that addresses genuine business needs while demonstrating our technical competencies and readiness to enter the professional software development field.

12.0 REFERENCES

The following resources were consulted during the development of the Residential Maintenance Management System:

Books and Publications:

Oracle. (2024). *JavaFX Documentation*. Oracle Corporation. https://openjfx.io/

Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley Professional.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

Horstmann, C. S. (2019). *Core Java Volume I: Fundamentals* (11th ed.). Prentice Hall.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database System Concepts* (7th ed.). McGraw-Hill Education.

Online Resources:

Jenkov, J. (2023). JavaFX Tutorial. Retrieved from http://tutorials.jenkov.com/javafx/

Baeldung. (2024). JavaFX Guide. Retrieved from https://www.baeldung.com/javafx

Oracle. (2024). Java Database Connectivity (JDBC) Documentation. Retrieved from https://docs.oracle.com/javase/tutorial/jdbc/

OMG. (2024). Unified Modeling Language (UML) Specification. Retrieved from https://www.omg.org/spec/UML/

GitHub. (2024). GitHub Documentation. Retrieved from https://docs.github.com/

Technical Documentation:

IntelliJ IDEA. (2024). JavaFX Application Development. JetBrains. Retrieved from https://www.jetbrains.com/help/idea/javafx.html

MySQL. (2024). MySQL Reference Manual. Oracle Corporation. Retrieved from https://dev.mysql.com/doc/

Stack Overflow. (2024). JavaFX Questions and Answers. Retrieved from https://stackoverflow.com/questions/tagged/javafx

Property Management Resources:

National Apartment Association. (2023). *Best Practices in Apartment Maintenance Management*. Retrieved from https://www.naahq.org/

BuildingEngines. (2023). *Property Maintenance Management Guide*. Retrieved from https://www.buildingengines.com/

Project Resources:

GitHub Repository. (2024). Residential Maintenance Management System. Retrieved from https://github.com/qperkins33/Residential-Maintenance-Management-System

13.0 APPENDIX

Appendix A: Glossary of Terms

- **DAO (Data Access Object)**: A design pattern that provides an abstract interface to database operations, separating business logic from data access logic.
- **FXML**: An XML-based markup language used to define JavaFX user interfaces declaratively.
- **JavaFX**: A software platform for creating rich internet applications using Java.
- **JDBC**: Java Database Connectivity, an API for connecting Java applications to databases.
- **Maintenance Request**: A formal request submitted by a tenant for repair or maintenance work.
- **UML**: Unified Modeling Language, a standardized modeling language for visualizing system design.
- **Work Order**: A task assigned to maintenance staff to complete a specific maintenance request.

Appendix B: Database Schema DDL

Complete database schema creation scripts are available in the project repository at: https://github.com/qperkins33/Residential-Maintenance-Management-System/tree/main/diagrams

Appendix C: User Manual Excerpts

For Tenants:

To submit a maintenance request, follow these steps:

1. Log in to the system using your username and password.
2. Click on "Submit Maintenance Request" from the dashboard.
3. Fill in the request form including description and category.
4. Upload photos if available (optional but recommended).
5. Click "Submit" to create the request.
6. Note your Request ID for future reference.

For Building Managers:

To assign a maintenance request to staff:

1. Log in and view pending requests from your dashboard.
2. Select a request to review the details.
3. Assess or adjust the priority level if needed.
4. Click "Assign to Staff" to see available maintenance personnel.
5. Review staff specializations and current workload.
6. Select the appropriate staff member and confirm assignment.
7. The system will create a work order and notify the staff member.

For Maintenance Staff:

To update work status:

1. Log in and view your assigned work orders.
2. Select a work order to see full details.
3. Click "Update Status" to change from "Assigned" to "In Progress".
4. Add progress notes as you work on the repair.
5. When completed, enter actual costs and upload completion photos.
6. Change status to "Completed" to close the work order.

Appendix D: System Configuration Guide

Setting up the Development Environment:

1. Install Java Development Kit (JDK) 21 or higher.
2. Download and install IntelliJ IDEA.
3. Clone the project repository from GitHub.
4. Open the project in IntelliJ IDEA.
5. Configure the JavaFX module path in run configurations.
6. Set up the database connection parameters.
7. Run the database initialization scripts.
8. Build and run the application using the "RMM JavaFX" configuration.

Appendix E: Common Issues and Troubleshooting

Issue: "JavaFX runtime components are missing"
Solution: Ensure JavaFX libraries are properly configured in the module path. Use the provided "RMM JavaFX" run configuration.

Issue: "Database connection failed"
Solution: Check database server is running and connection parameters (host, port, username, password) are correct.

Issue: "Photo upload fails"
Solution: Verify the upload directory exists and has write permissions. Check file size is under 5MB.

Appendix F: Contact Information and Support

For questions, issues, or contributions to the project, please visit:
https://github.com/qperkins33/Residential-Maintenance-Management-System

Project Team:
Available through the GitHub repository contacts information.

Project Repository:
https://github.com/qperkins33/Residential-Maintenance-Management-System

Documentation:
Complete technical documentation and code comments are available in the repository.