Real Estate Management System (REMS)

Presented By

Sienna Markham, 220274536
Mahjabin Mollah, 220416038
Eamon Ryan, 219447192

GitHub: https://github.com/eamontryan/digt3101_team3

Team 3

February 1st, 2026

Table of Contents

Section 1:

Introduction

The field of software engineering increasingly relies on structured methodologies, disciplined design practices, and iterative development cycles to deliver reliable and scalable systems. Modern organizations depend on comprehensive software solutions to automate processes, improve operational efficiency, and support decision-making. Real estate management, in particular, requires sophisticated digital systems capable of tracking property data, supporting client interactions, handling transactions securely, and ensuring lifecycle visibility across large portfolios.

The Real Estate Management System (REMS) project provides an opportunity to apply the full Software Development Lifecycle (SDLC) to a real-world problem domain. By combining Agile iterative development with formal specification and design artifacts, this project replicates modern industry practices and reinforces the practical skills of requirement analysis, architectural modelling, implementation, and testing.

Motivation

Current Situation

Large real estate development companies manage numerous commercial properties, often containing hundreds of retail units across multiple mall locations. Without an integrated software system, tasks such as tracking rental units, scheduling viewings, collecting rent, reviewing lease agreements, and responding to maintenance requests become error-prone, fragmented, and inefficient. Manual or semi-digital processes frequently result in double-booked appointments, delayed reporting, missing invoices, and inconsistent record-keeping.

Existing Solutions

While generic property management platforms exist, they rarely provide the level of customization required for commercial mall management, including complex rental cycles, multi-unit discounts, utility billing, digital lease renewals, and automated emergency-level maintenance escalation. Existing systems also may not integrate with the company's workflow, lack tailored modules, or fail to support simultaneous Agile and documentation-heavy development processes required in regulated environments.

Why this Project is Needed

This project aims to bridge these shortcomings by developing a robust, end-to-end software solution specifically tailored for large-scale commercial real estate operations. Creating a custom REMS allows complete alignment with business rules while providing students hands-on experience applying SDLC principles.

Project Description

The Real Estate Management System (REMS) is a Java-based software solution designed to automate and streamline commercial property management. The system will support:

Main Requirements

- **Store Unit Management:**
  - Track store size, location, rental tier, classification, and tenant purpose
- **Search & Availability:**
  - Allow tenants to filter and search for rentable units
- **Appointment Scheduling:**
  - Enable tenants to book viewing appointments while preventing double-bookings
- **Rental Applications & Digital Leases:**
  - Electronic submissions, approval workflows, and lease renewal logic
- **Automated Rent Collection:**
  - Support multiple billing cycles (monthly/quarterly/annual), discounts, invoices, and overdue notifications
- **Utility Billing:**
  - Combine electricity, water, and waste charges into unified monthly statements
- **Maintenance Portal:**
  - Log, prioritize, and escalate maintenance requests; apply charges for misuse
- **Administrative Tools:**
  - Reporting, data entry, management dashboards (future enhancement)

Benefits of the System

- Reduces administrative workload and manual errors
- Improves transparency for tenants and leasing agents
- Offers predictable and automated billing workflows
- Enables better planning through structured data and reporting
- Provides a modern digital experience for a traditionally paperwork-heavy industry
- Scales across multiple mall properties

Organization of the Report

This report is organized into nine main sections.

- **Section 1** introduces the project, including its motivation, description, and overall structure
- **Section 2** outlines the problem statement, the proposed solution, the design objectives, and the project's limitations
- **Section 3** presents the system specification, detailing all functional, nonfunctional, and optional pseudo requirements
- **Section 4** describes the system's use cases, including the use case diagram and corresponding descriptions
- **Section 5** provides the user stories associated with each major system feature

- **Section 6** explains the system analysis and design models, including the domain model, sequence diagrams, and architecture
- **Section 7** presents the architecture and detailed design of the system, including component and class diagrams and applied design patterns
- **Section 8** presents the project management plan, including the Gantt chart, updated product and sprint backlogs, and group meeting logs
- **Section 9** outlines the test-driven development approach, including functional, non-functional, and performance test cases used to validate the system

A references section is included at the end of the report.

Section 2:

Problem Statement and Design Objectives

Problem Statement and Proposed Solution

Problem Statement

The leasing division of a commercial real estate development company currently lacks an integrated system to manage retail units, schedule appointments, handle rental applications, collect rent, track utility consumption, and coordinate maintenance operations across multiple mall properties. Fragmented tools and manual processes cause inefficiencies, double-bookings, delayed payments, inconsistent maintenance handling, and a poor tenant experience.

What the Project Aims to Do

This project aims to design and implement a full-featured Real Estate Management System that centralizes all property management workflows into a unified platform. The system will support both staff-facing and tenant-facing interfaces, ensuring accuracy, automation, and scalability.

Main Solution

The proposed solution is a Java-based software system developed using Agile methods and formal SDLC documentation. It will integrate modules for unit management, appointment scheduling, rental processing, automated rent and utility billing, and maintenance request handling. The solution will be supported by a relational or non-relational database, UML-based architectural modeling, and a full suite of tests.

Design Objectives

Objective 1 - Requirements Documentation

Due Date: November 23, 2025 (Deliverable 1)

Deliverables:

- Complete Software Requirements Specification (SRS)
- Full list of Functional Requirements (e.g., unit management, rent billing, scheduling)
- Full list of Non-Functional Requirements (e.g., reliability, maintainability, usability)
- Detailed User Stories and Acceptance Criteria
- Use Case List, Use Case Descriptions, and Use Case Diagram

Measurement Criteria:

- All requirements approved by team and instructor
- Requirements traceable to system features

Purpose:
Establishes a precise, shared understanding of what the system must do.

Objective 2 - System Architecture & Design Models

Due Date: February 1, 2026 (Deliverable 2)

Deliverables:

- Complete Software Design Document (SDD)
- Defined System Architecture (Layered or MVC)
- UML Class Diagram showing major classes and relationships
- UML Sequence Diagrams for core workflows (e.g., renewal, booking, payment)
- UML Activity Diagrams showing process flows
- Full Database Schema / ERD

Measurement Criteria:

- Diagrams accurately reflect requirements
- Architecture supports scalability, maintainability, and required performance

Purpose:
Provides a blueprint for implementation and ensures the design supports all REMS features.

Objective 3 - Core Feature Implementation

Due Date: March 22, 2026 (Deliverable 3 – Implementation)

Functional Modules to Deliver:

- Store Unit Management (CRUD, classification, rental tiering)
- Appointment Scheduling with double-booking prevention
- Rental Application & Digital Lease Module
- Automated Rent Billing (cycles, invoices, discounts)
- Utility Billing (water, electricity, waste consolidation)
- Maintenance Portal with prioritization & escalation logic

Measurement Criteria:

- All modules implemented in Java
- All modules integrated with the database
- All modules verified against the SRS

Purpose:
Implements the tangible, functional components that define the system.

Objective 4 - Testing & Quality Assurance

Due Date: March 22, 2026 (Deliverable 3 – Testing)

Deliverables:

- Complete Test Plan (unit, integration, acceptance tests)
- JUnit Test Suites with measurable coverage
- Test Report documenting results, defects, and verification of each requirement
- Evidence of continuous integration and code quality checks

Measurement Criteria:

- Meets code coverage expectations (set by instructor/team)
- All tests map to functional + non-functional requirements

Purpose:
Ensures the system meets reliability, usability, and performance expectations.

Objective 5 - Project Management & Documentation

Due Date: Throughout the Project

Deliverables:

- Fully maintained GitHub/GitLab Repository
  - Branching strategy, commits, pull requests, code reviews
- Sprint Backlogs, Burndown Charts, Retrospectives
- Final documentation package delivered by **March 22, 2026**:
  - Project Management Plan
  - SRS

- SDD
- Test Plan
- Test Report
- Implementation Notes

Measurement Criteria:

- Repository is up-to-date and reflects real collaborative work
- All documentation meets SDLC standards

Purpose:
Guarantees professional-level workflow and traceability across all project phases.

Objective 6 - Enhancement Sprint & Portfolio Update

Due Date: End of Summer 2026

Deliverables:

- Enhanced Version of REMS (bug fixes, optimizations, minor feature additions)
- Completed E-Portfolio documenting architecture, implementation, testing, and learning outcomes

Measurement Criteria:

- Enhancements successfully integrated and tested
- Portfolio meets program requirements

Purpose:
Demonstrates continuous improvement and professional reflection.

Limitations

Due to time, scope, and resource constraints, the project will not include:

- Full production-level security hardening (only basic authentication/authorization)
- Integration with real payment processors (simulated or mocked payments only)
- Real hardware-based utility metering (only manual or simulated data inputs)
- Multi-language localization or internationalization

These limitations ensure the project remains achievable within the course schedule while still meeting core learning objectives.

Section 3:

System Specification

Functional Requirements

Property & Store Unit Management

- **FR1.** The system shall allow authorized staff to add, update, view, and delete store unit records.
- **FR2.** The system shall store and manage details for each store unit, including location, size, rental rate, classification tier, intended business purpose, and availability.
- **FR3.** The system shall support management of multiple mall properties, each with its own inventory of store units.

Tenant Store Search

- **FR4.** The system shall allow potential tenants to search for available store units using filters such as location, size, rental rate, classification tier, business purpose, and availability.
- **FR5.** The system shall display search results with standardized store information and contact or scheduling options.

Appointment Scheduling

- **FR6.** The system shall allow tenants to schedule viewing appointments with a leasing agent based on agent availability.
- **FR7.** The system shall prevent double-bookings by ensuring no overlapping appointments for the same leasing agent or store unit.
- **FR8.** The system shall send automated notifications confirming appointments and any subsequent updates.

Rental Applications & Lease Management

- **FR9.** The system shall allow tenants to submit digital rental applications, including file uploads for required documents.
- **FR10.** The system shall automatically generate digital lease agreements based on property and rental information.
- **FR11.** The system shall support electronic signing of lease agreements.
- **FR12.** The system shall track lease start and end dates for each store unit.
- **FR13.** The system shall automatically process lease renewals according to configurable company policies.

Rent Collection & Billing

- **FR14.** The system shall allow tenants to choose a payment cycle, including monthly, quarterly, bi-annual, or annual options and calculate rental charges dynamically based on the selected payment cycle.

- **FR15.** The system shall apply discount logic for tenants leasing multiple store units.
- **FR16.** The system shall generate invoices for rent, utilities, and applicable fees.
- **FR17.** The system shall track tenant payments and send notifications for overdue payments to tenants and alerts to administrative staff.

Utility Billing

- **FR18.** The system shall allow authorized staff to input manually or import utility consumption data for electricity, water, and waste management.
- **FR19.** The system shall automatically consolidate rent and utility charges into a unified monthly bill.

Maintenance Requests

- **FR20.** The system shall allow tenants to submit maintenance requests through an online portal.
- **FR21.** The system shall categorize maintenance requests by priority, automatically escalating emergency requests.
- **FR22.** The system shall apply charges to tenant accounts for misuse-related maintenance requests.

User Access & Roles

- **FR23.** The system shall support user authentication for tenants, leasing agents, and administrative staff.
- **FR24.** The system shall enforce role-based access control, restricting sensitive operations to authorized users.

Nonfunctional Requirements

1. **Performance:**
    - **NFR1.** The system shall process search queries and return available unit results within 2 seconds under normal load.
    - **NFR2.** The system shall maintain efficient data retrieval for large property portfolios (25+ malls, hundreds of units).
2. **Quality:**
    - **NFR3.** The system shall maintain 99% uptime, excluding scheduled maintenance.
    - **NFR4.** The system shall ensure that no lease, appointment, or billing transaction is partially completed (atomicity).
3. **Safety and Security:**
    - **NFR5.** The system shall validate user input to prevent malformed or invalid data from corrupting stored information.
    - **NFR6.** The system shall provide confirmation dialogs for critical actions (e.g., deleting property records or terminating leases).
4. **Interface:**
    - **NFR7.** The system shall present a user-friendly, responsive, and mobile-friendly interface for tenants and staff.

- **NFR8.** The system shall be compatible with standard web browsers (Chrome, Firefox, Edge, Safari).
- **NFR9.** The system shall support integration with external data sources for utility consumption (simulated or API-based).

Pseudo Requirements (Optional)

- **C1.** The system must use a relational database (e.g., MySQL or PostgreSQL)
- **C2.** The system must be compatible with legacy property data exported from an old record-keeping system.
- **C3.** The system shall be implemented using Java
- **C4.** The GUI must be built using a framework that is compatible with Java

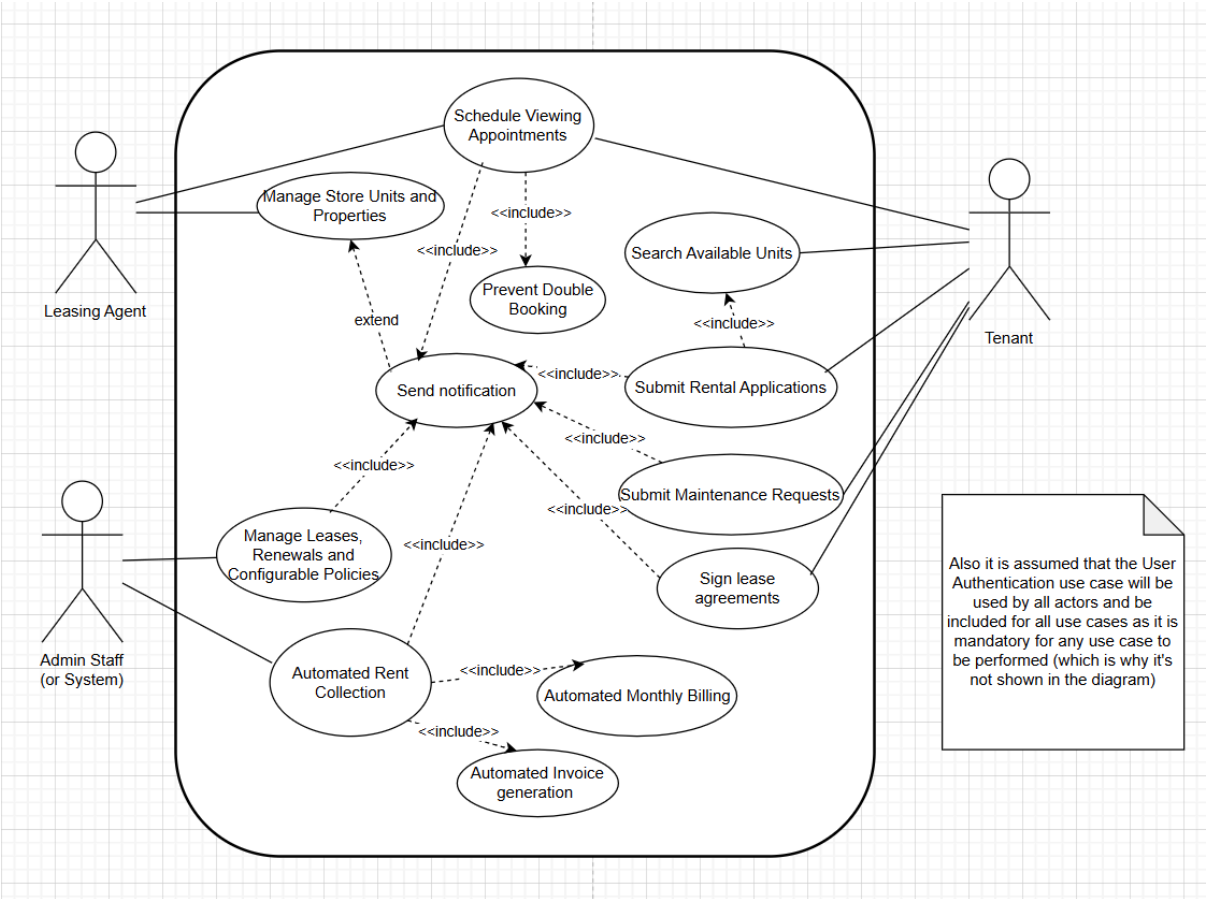
Section 4:

Use Cases

Use cases provide a structured approach to understanding the interactions between users and the system.

Use Case Diagram

Include a UML use case diagram that shows the actors (e.g., Admin, Regular User) and the main use cases (e.g., Manage Inventory, Generate Reports, User Authentication).



Use Case Descriptions

Detail 4 key use cases with the following structure:

1. **Use Case Name**: Manage Inventory
   Actors: Admin, Regular User
   Description: This use case describes how users can add, update, and delete inventory items.
   Preconditions: User must be logged in.
   Main Flow:

- The user selects "Manage Inventory" from the main menu.
- The system displays a list of current inventory items.
- The user selects an item to edit or adds a new item.
- The system saves the changes and updates the database.

Postconditions: The inventory is updated, and changes are visible to all users.

2. **Use Case Name**: Schedule Viewing Appointments
   Actors: Leasing Agent, Tenant
   Description: This use case allows tenants or leasing agents to request, schedule, or update property viewing appointments while preventing double-bookings.
   Preconditions:
   - Leasing Agent and Tenant must be authenticated.
   - Units being scheduled to view are available

   Main Flow:
   - Tenant selects “Schedule Viewing Appointments.”
   - The system shows available dates/times based on unit and agent availability.
   - Tenant selects a date and time.
   - The system performs Prevent Double Booking (included).
   - The system confirms the appointment and saves the booking.
   - The system triggers Send Notification (included) to confirm the appointment to all parties.

   Postconditions:
   - A viewing appointment is saved in the system.
   - Notifications are sent to the tenant and leasing agent.
   - Time slot becomes unavailable to others.

3. **Use Case Name**: Automated Rent Collection
   Actors: Admin Staff (System as Internal Actor)
   Description:This use case automatically processes recurring rent payments according to each tenant’s lease terms.
   Preconditions:
   - System must have valid tenant lease and billing information.
   - Admin Staff must be authenticated to view or configure settings.

   Main Flow:
   - The system identifies all tenants with rent due.
   - The system calculates the amount owed based on lease terms.
   - The system triggers Automated Monthly Billing (included).
   - The system attempts to collect payment automatically.
   - If successful, the system updates the rent balance.
   - The system triggers Automated Invoice Generation (included) to create a receipt.
   - The system triggers Send Notification (included) to update the tenant and staff on payment status.

   Postconditions:
   - Rent is collected or marked as failed.
   - Notifications and invoices are generated.

4. **Use Case Name**: Submit Rental Applications
   Actors: Tenant
   Description: This use case allows tenants to fill out and submit rental applications for

available units.
Preconditions:
- Tenant must be authenticated.
- Unit information that tenant searched for must be available in the system

Main Flow:
- The tenant selects “Submit Rental Applications.”
- The tenant searches for a unit (included)
- The system displays a list of available units.
- The tenant selects a unit and fills in application details.
- The system validates the submitted application.
- The system saves the application.
- The system triggers Send Notification (included) to inform staff that a new application is received.

Postconditions:
- Application is saved and marked as “Submitted.”
- Notifications are sent to leasing/administrative staff.

5. **Use Case Name**: Manage Leases, Renewals and Configurable Policies
Actors: Admin Staff (System as internal actor)
Description: This use case allows administrative staff to create new leases, update existing lease terms, process renewals, and manage configurable policy settings (late fees, deposit rules, renewal options).
Preconditions:
   - Admin Staff must be authenticated.
   - Property/unit records must exist in the system.

Main Flow:
   - The Admin Staff selects “Manage Leases, Renewals and Configurable Policies.”
   - The system displays all existing leases and policy configurations.
   - The Admin chooses to create a new lease, edit an existing lease, process a renewal, or update policies.
   - The Admin enters or edits the required information.
   - The system validates lease parameters and policy rules.
   - The system saves the updated or new lease/policy information.
   - The system triggers Send Notification (included) to inform tenants or staff of changes.

Postconditions:
   - Lease or policy information is updated in the system.
   - Notifications are sent to relevant parties.

6. **Use Case Name**: Sign Lease Agreements
Actors: Tenant
Description: This use case describes how a tenant reviews and electronically signs a lease agreement after receiving approval for their rental application.
Preconditions:
   - Tenant must be authenticated.
   - An approved lease must exist for the tenant.
   - System must have the digital lease document generated and available.

Main Flow:
   - The tenant selects “Sign Lease Agreements.”
   - The system displays the lease terms and agreement document.

- The tenant reviews the document.
- The tenant electronically signs the lease.
- The system verifies the signature and stores the signed document.
- The system updates the tenant's status to "Lease Signed."
- The system triggers Send Notification (included) to inform staff of the completed signing.

Postconditions:

- Lease is marked as signed and stored in the database.
- Staff are notified for next steps (e.g., move-in scheduling).

7. **Use Case Name**: Submit Maintenance Requests
   Actors: Tenant
   Description: This use case allows tenants to submit maintenance or repair requests for issues in their rented units.
   Preconditions:
   - Tenant must be authenticated.
   - Tenant must have an active lease.

   Main Flow:
   - The tenant selects "Submit Maintenance Requests."
   - The system displays categories of maintenance issues (e.g., plumbing, electrical, appliance).
   - The tenant enters the problem description and optionally uploads photos.
   - The system validates the request information.
   - The system saves the maintenance request and assigns a ticket number.
   - The system triggers Send Notification (included) to notify maintenance staff or administrators.
   - The system updates the request status to "Submitted."

   Postconditions:
   3. Maintenance request is logged and visible to maintenance/admin staff.
   4. Notification is sent to the staff for action.

Section 5:

User Stories

- Format: As a [user role], I want to [goal] so that [reason]
- Map each user story to functional requirements where applicable
- Indicate story points (effort estimate) and acceptance criteria

Feature 1: Store Unit Management & Search

User Story 1 — Create Software Requirements Specification (SRS)

Functional Requirement(s): *All FRs*
Story:
As a project manager, I want a complete SRS so that all stakeholders have a clear understanding of functional and non-functional requirements.

Acceptance Criteria:

- SRS includes FRs, NFRs, Use Cases & User Stories
- All requirements mapped to user needs

- Structure follows SDLC format
- SRS is reviewed and approved by team/instructor

Priority: High
Story Points: 5

User Story 2 — Develop System Architecture & UML Design

Functional Requirement(s): *All FRs*

Story:
As a system architect, I want UML diagrams, system architecture, and ERD so the development team has a clear blueprint for implementation.

Acceptance Criteria:
- High-level architecture (MVC / layered) created
- UML Class, Sequence & Activity Diagrams completed
- ERD reflects all FRs
- Review meeting held with developers

Priority: High
Story Points: 8

User Story 3 — Implement Role-Based Access Control (RBAC)

Functional Requirement(s): FR23–FR24

Story:
As an Admin, I want to assign roles (Tenant, Agent, Admin) so only authorized users can access sensitive features.

Acceptance Criteria:
- Users must login/have an account to be assigned a role
- Roles are stored in DB
- Permission logic enforced on UI/API
- Unauthorized access is blocked
- Authentication tested with sample users

Priority: High
Story Points: 5

User Story 4 — Digital Lease Agreement Generation & Electronic Signing

Functional Requirement(s): FR9–FR11

Story:
As a Tenant, I want the system to generate digital lease agreements automatically that I can electronically sign so I don't need physical paperwork.

Acceptance Criteria:
- Lease template generated automatically
- Lease stored in database
- Electronic signing enabled when lease is approved
- Status changes to "Active" after signing

Priority: High
Story Points: 8

User Story 5 — Automated Late Payment & Overdue Alerts
Functional Requirement(s): FR17

Story:
As an Admin, I want automated overdue alerts so reminders don't need to be manually sent.

Acceptance Criteria:
- Email/SMS alert logic implemented
- Alerts stored in DB / logs
- Triggered automatically after due date
- Can be disabled/configured by admin

Priority: Medium
Story Points: 3

User Story 6 — Bulk Discount & Multiple Unit Billing Logic
Functional Requirement(s): FR15

Story:
As a Tenant leasing multiple units, I want bulk discount logic so I get accurate billing.

Acceptance Criteria:
- Billing formula applies discount correctly
- Discount shown on invoice
- Can handle multiple scenarios
- Logs recorded for audit purposes

Priority: Medium
Story Points: 5

User Story 7 — Multi-Mall Property Management
Functional Requirement(s): FR1–FR3

Story:
As an Admin, I want to manage multiple malls with separate store inventories so that each mall can operate independently and be tracked accurately.

Acceptance Criteria:
- Mall entity created in database
- Stores linked to specific mall
- UI supports mall switching/filtering
- View per-mall inventory page
- Add/Update/Delete mall inventory(units)

Priority: Medium
Story Points: 4

User Story 8 — Mobile-Friendly & Responsive UI
Functional Requirement(s): NFR7–NFR8

Story:
As a Tenant, I want a responsive UI so I can manage leases and requests on any device.

Acceptance Criteria:

- UI tested on mobile, tablet, desktop
- Works on Chrome, Firefox, Edge, Safari
- Menu/navigation adapts per device
- Layout meets accessibility guidelines

Priority: Medium
Story Points: 5

User Story 9 — System Recovery from Partial Failures
Functional Requirement(s): NFR4

Story:
As a system user, I want transactions to fully complete or roll back so that no data is corrupted.

Acceptance Criteria:
- Failure detection logic implemented
- Rollback tested on sample transactions
- DB state remains consistent after failure
- Failure log recorded per event

Priority: High
Story Points: 5

User Story 10 — Lease Tracking & Automatic Renewal
Functional Requirement(s): FR12–FR13
Story:
As a Tenant I want to be able to track lease expiries and have automatic renewals

Acceptance Criteria:
- Lease start & end date stored in DB
- Email alert **X days before expiry and renewal**
- Summary dashboard shows soon-to-expire leases
- Leases can be automatically renewed
- User notification triggered automatically

Priority: Medium
Story Points: 4

User Story 11 — Testing & Code Quality (JUnit, CI/CD)
Functional Requirement(s): NFR3–NFR4

Story:
As a QA Engineer, I want automated testing & CI/CD so that system reliability is verified.

Acceptance Criteria:
- JUnit test suite created
- CI/CD pipeline runs on push
- Code coverage reviewed & documented
- Minimum threshold set for pass/fail

Priority: High
Story Points: 5

User Story 12 — 2-Second Search Property Performance
Functional Requirement(s): NFR1 & FR4–FR5

Story:
As a Tenant, I want to be able to search properties for viewing and search results to load within 2 seconds so the system feels fast.

Acceptance Criteria:
- User can search for properties/units
- Properties and Units are displayed to the user
- Search optimized for speed
- Indexing added to DB
- Time logged for performance testing
- Performance report generated

Priority: High
Story Points: 3

User Story 13 — Transaction Atomicity (Full Rollback)
Functional Requirement(s): NFR4

Story:
As a user, I want full rollbacks so database safety is guaranteed.

Acceptance Criteria:
- Rollback rules defined
- Failure scenarios tested
- DB remains consistent after crashes
- Rollback logs recorded

Priority: High
Story Points: 3

User Story 14 — Import & Manage Utility Usage Data for Monthly Bill
Functional Requirement(s): FR14–FR19

Story:
As Admin Staff, I want to input/import utility data so monthly bills reflect real consumption.

Acceptance Criteria:
- CSV/manual input supported
- Data stored in utility table
- Calculate monthly bill based on rent, payment cycle and utility charges
- Generate invoice

Priority: High
Story Points: 5

User Story 15 — Misuse Billing for Maintenance
Functional Requirement(s): FR22

Story:
As a Tenant, I want misuse-related maintenance charges applied fairly so that I'm only billed

when I'm actually responsible for the damage.

Acceptance Criteria:
- Misuse billing logic added
- Admin override available
- Cost shown on invoice
- Logged in DB for traceability

Priority: Medium
Story Points: 4

User Story 16 — Enhancement & E-Portfolio Update
Functional Requirement(s): *General – Final Sprint*

Story:
As a student developer, I want to fix bugs and polish UI so I can present the project professionally.

Acceptance Criteria:
- Bugs from Sprint 4 fixed
- UI reviewed & improved
- Portfolio prepared for submission
- Final documentation updated

Priority: Low
Story Points: 3

User Story 17 — Schedule Viewing Appointments
Functional Requirement(s): FR6–FR8

Story:
As a tenant, I would like to schedule viewing appointments with a leasing agent to view a property.

Acceptance Criteria:
- Choose a property and schedule an appointment to view it in an available time slot
- Prevent any double booking
- Get a confirmation notification

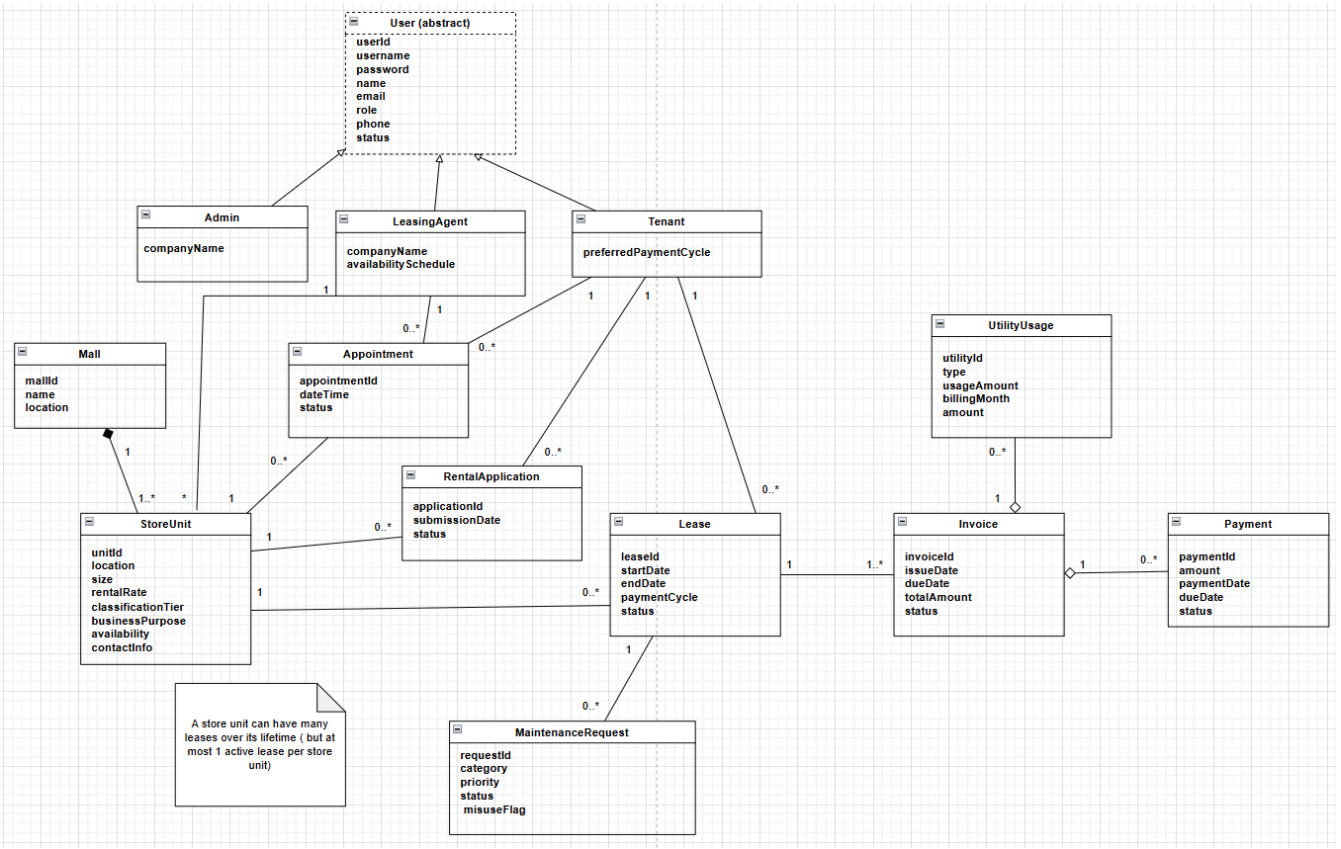
Priority: High
Story Points: 6

Section 6:

System Analysis and Domain Model

Domain Model

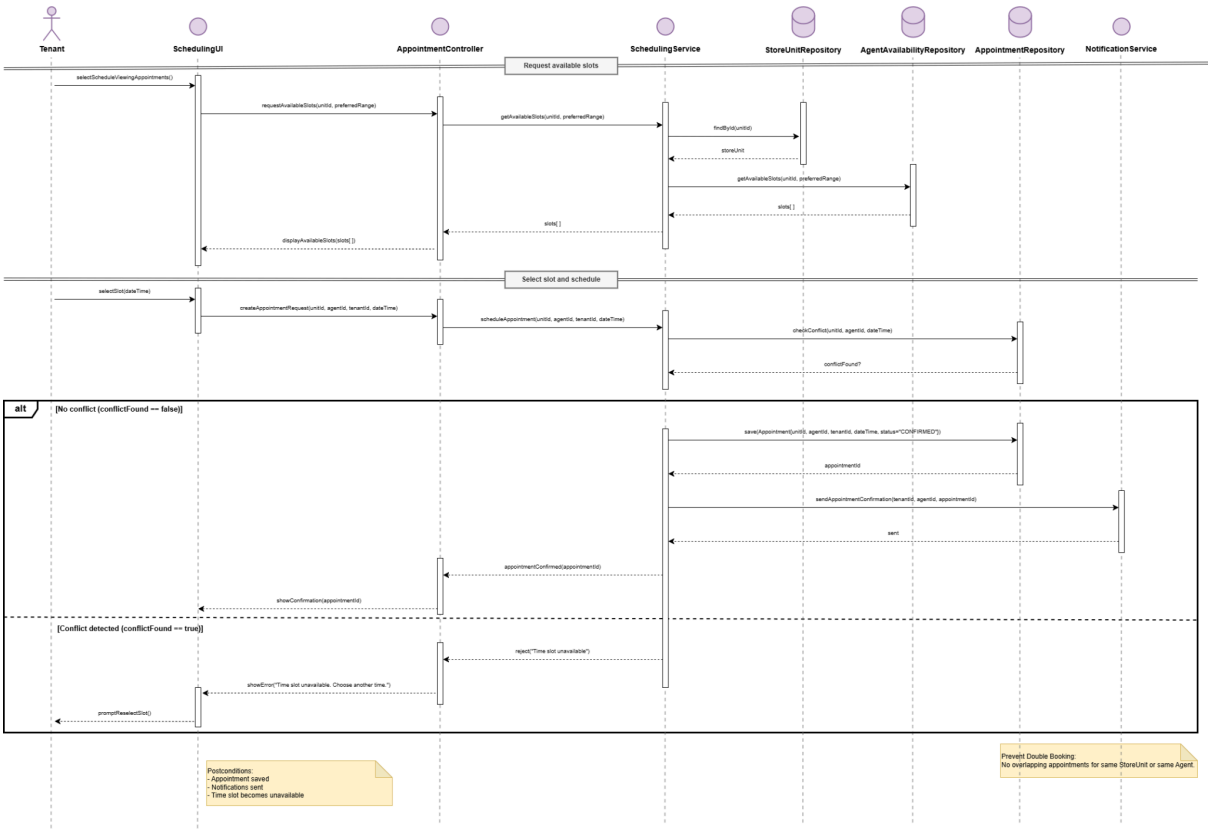
Provide the initial class diagram that reflects the domain model of your system.



Sequence Diagrams

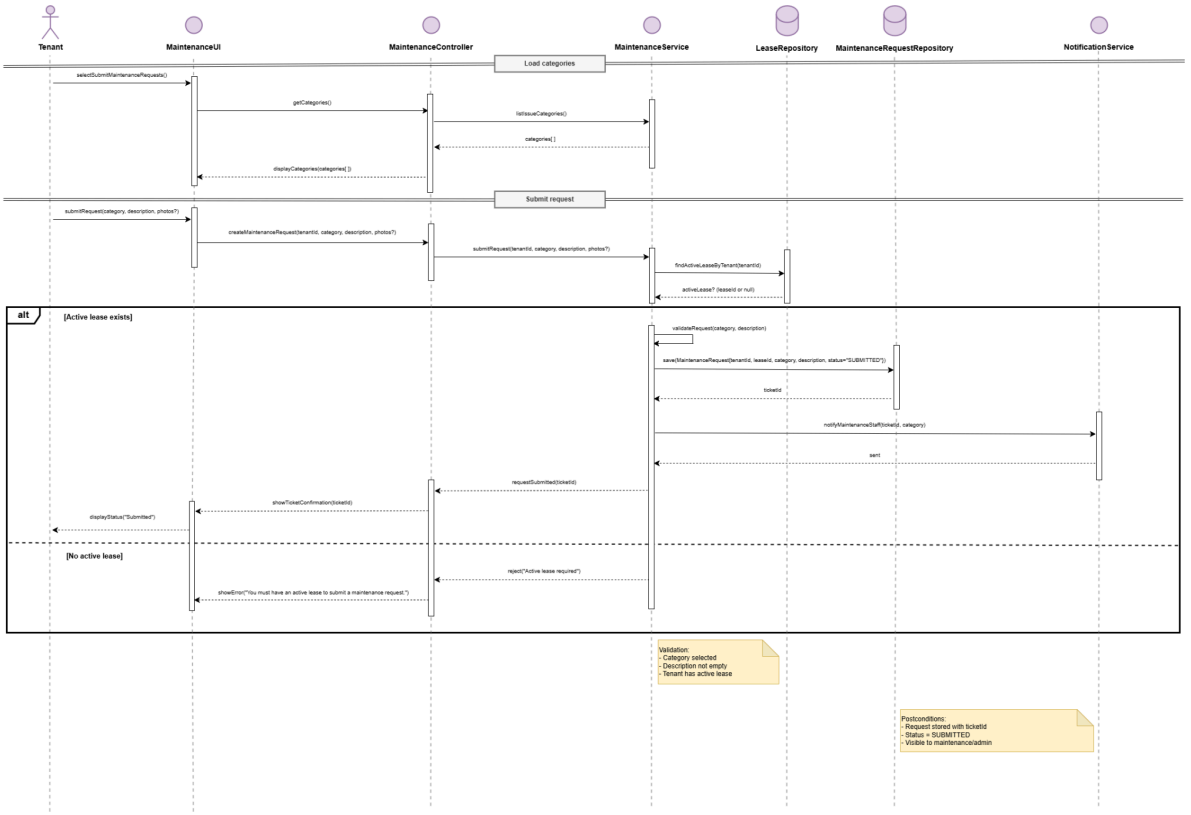
For each of the four fully described cases, draw the corresponding sequence diagram. Make sure you identify and associate each sequence diagram with the proper use case by maintaining unique Identifiers for use cases. Refer to the project description for which of the scenario you need to write collaboration diagrams.

1) UC-02: Schedule Viewing Appointments



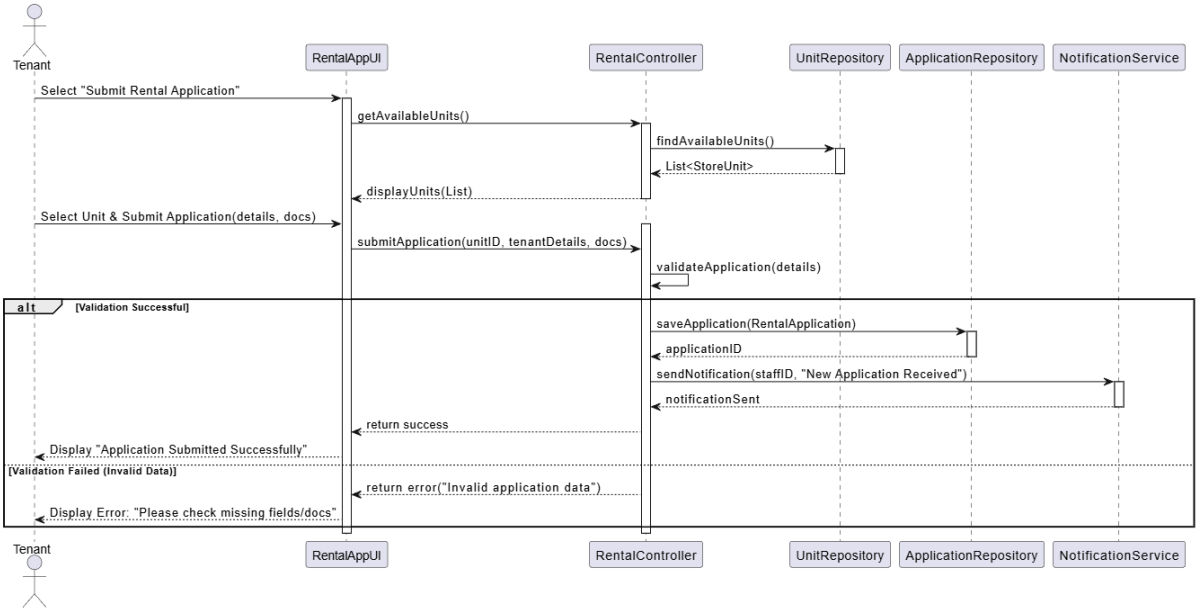
https://drive.google.com/file/d/1Th1-qoAAt72ItERqizcO3aIEEqX46JSv/view?usp=sharing (better visibility - also available in repo)

2) UC-07: Submit Maintenance Requests

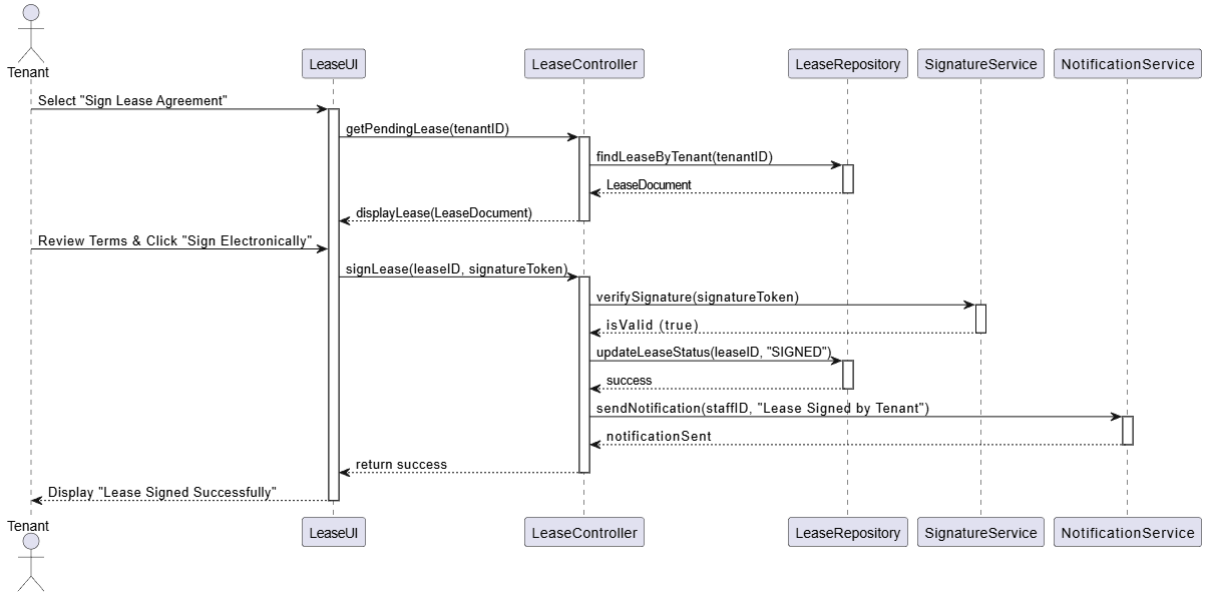


https://drive.google.com/file/d/1_9eO8yxWoWkVVdYlGJo_3Rg7sbaEg0cL/view?usp=sharing (better visibility - also available in repo)

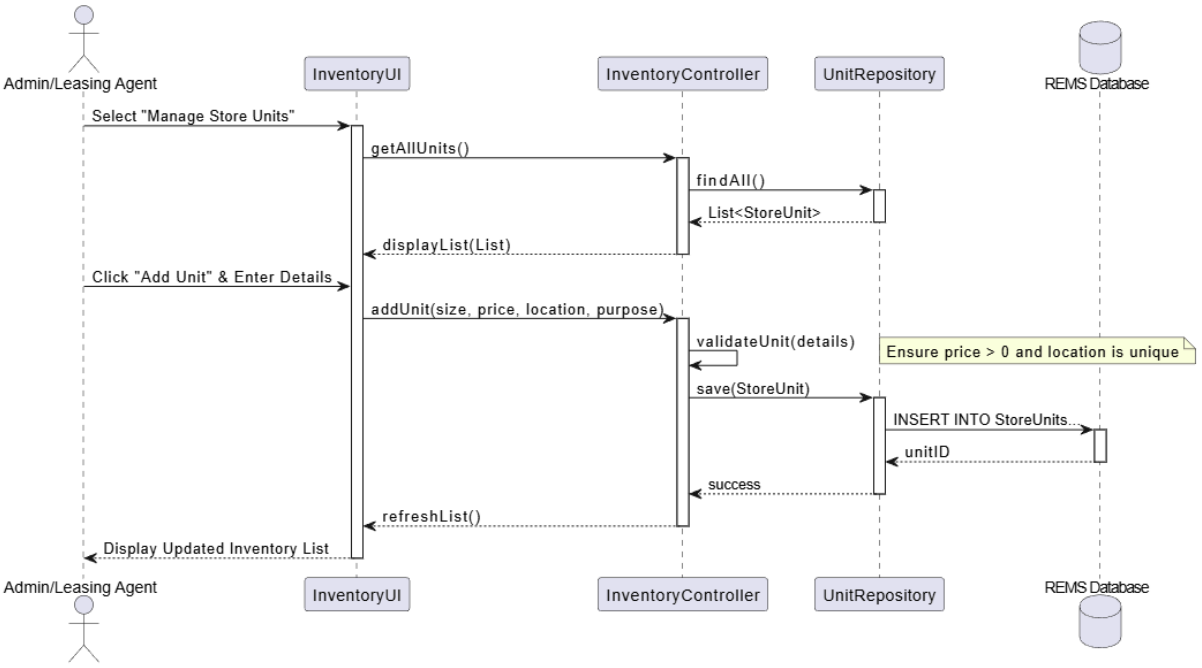
3) UC-04: Submit Rental Applications



4) UC-06: Sign Lease Agreements



5) UC-01: Manage Inventory



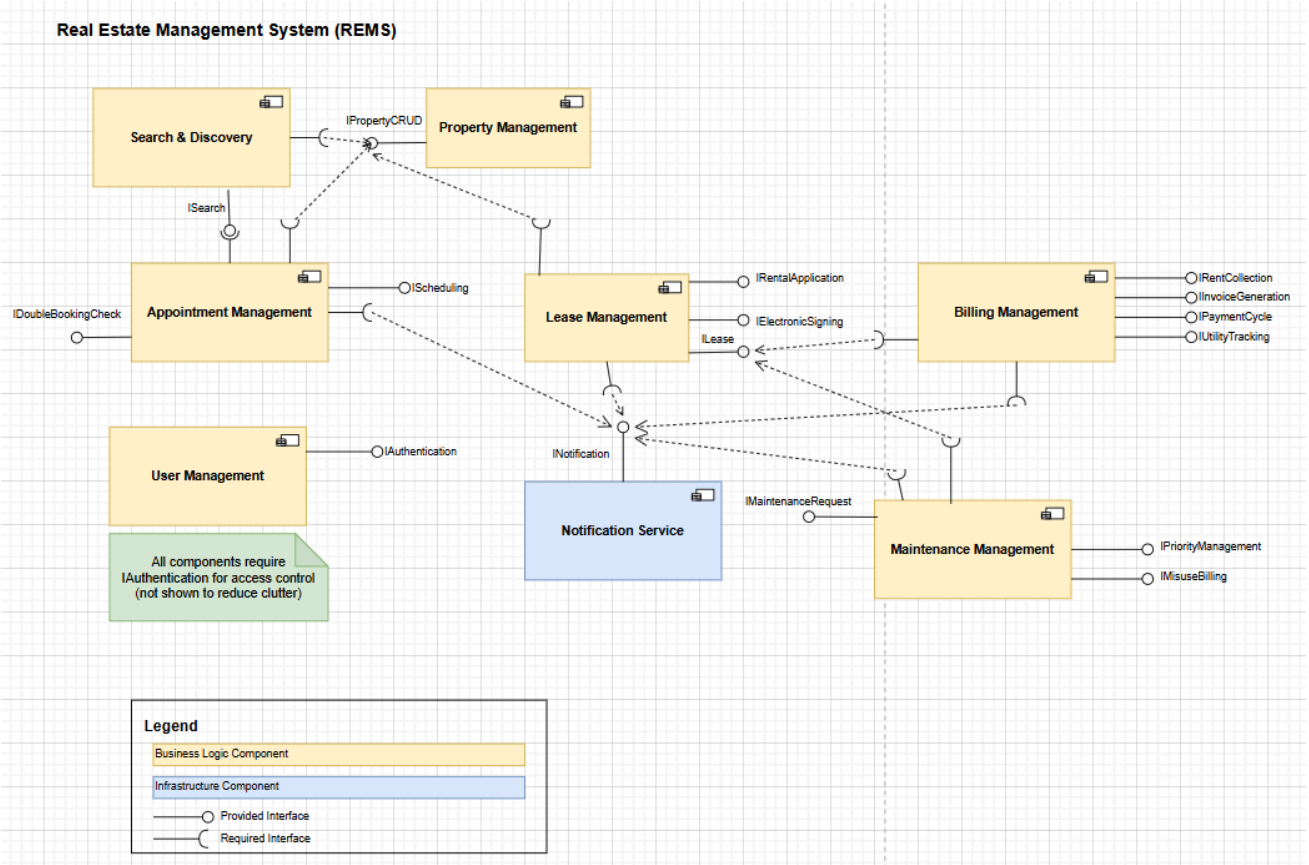
Section 7:

Architecture and Design

Provide detailed up-to-date architecture and design specifications of your system as it stands now. More specifically you will provide:

Component Diagram

1. A component diagram of your system clearly labelling each exposed interface. You may want to use more than one component diagram in case of complex nested components so that the clarity and readability of your diagrams are increased. Make sure you clearly show the client components (i.e. the components using the exposed interfaces as per UML Component diagram notation).



2. A table with a description of each component as per the table below.

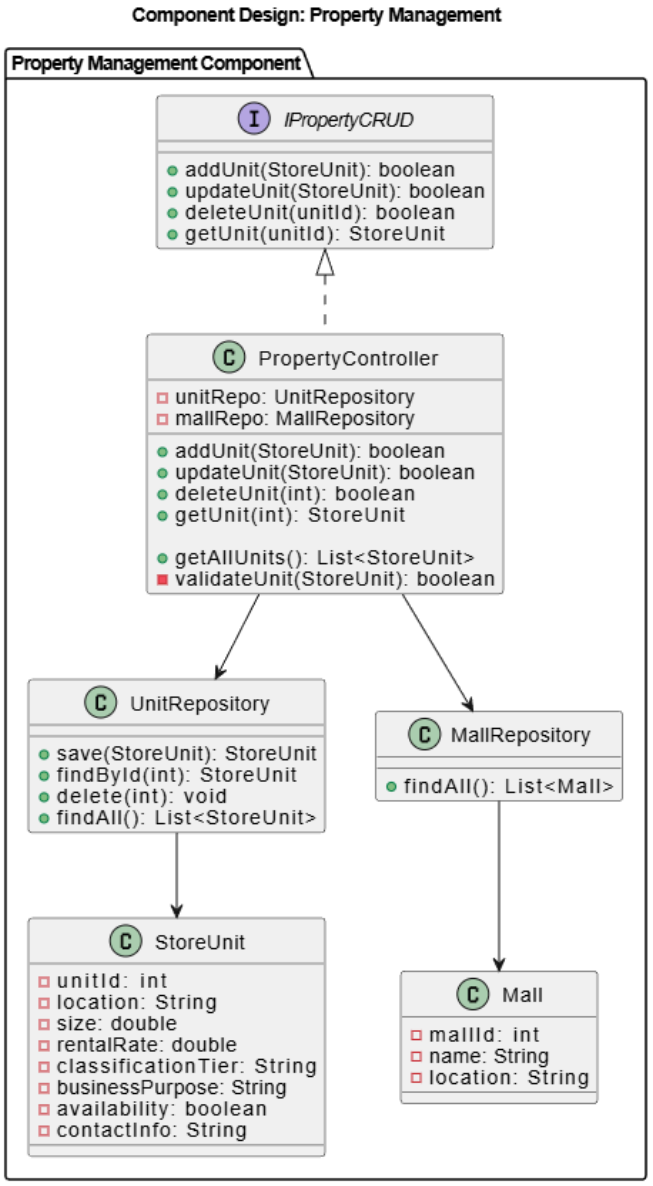
Component Name	*Description*
User Management	Handles authentication, authorization, and role-based access control for all system users (Tenants, Leasing Agents, Admin Staff)
Property Management	Manages the property and unit inventory including CRUD operations, unit details, availability, and property records
Search & Discovery	Provides property search and filtering capabilities for prospective tenants to find available units
Appointment Management	Manages scheduling of property viewings with double-booking prevention and appointment confirmations
Lease Management	Handles the complete tenant lifecycle: rental applications, lease generation, electronic signing, and lease renewal processes
Billing Management	Manages all financial transactions including rent collection, invoice generation, payment cycles, utility tracking, and utility billing
Maintenance Management	Processes maintenance requests with priority management and handles billing for tenant misuse-related damages

Notification Service	Sends notifications (email) for appointments, lease applications, payments, and maintenance updates

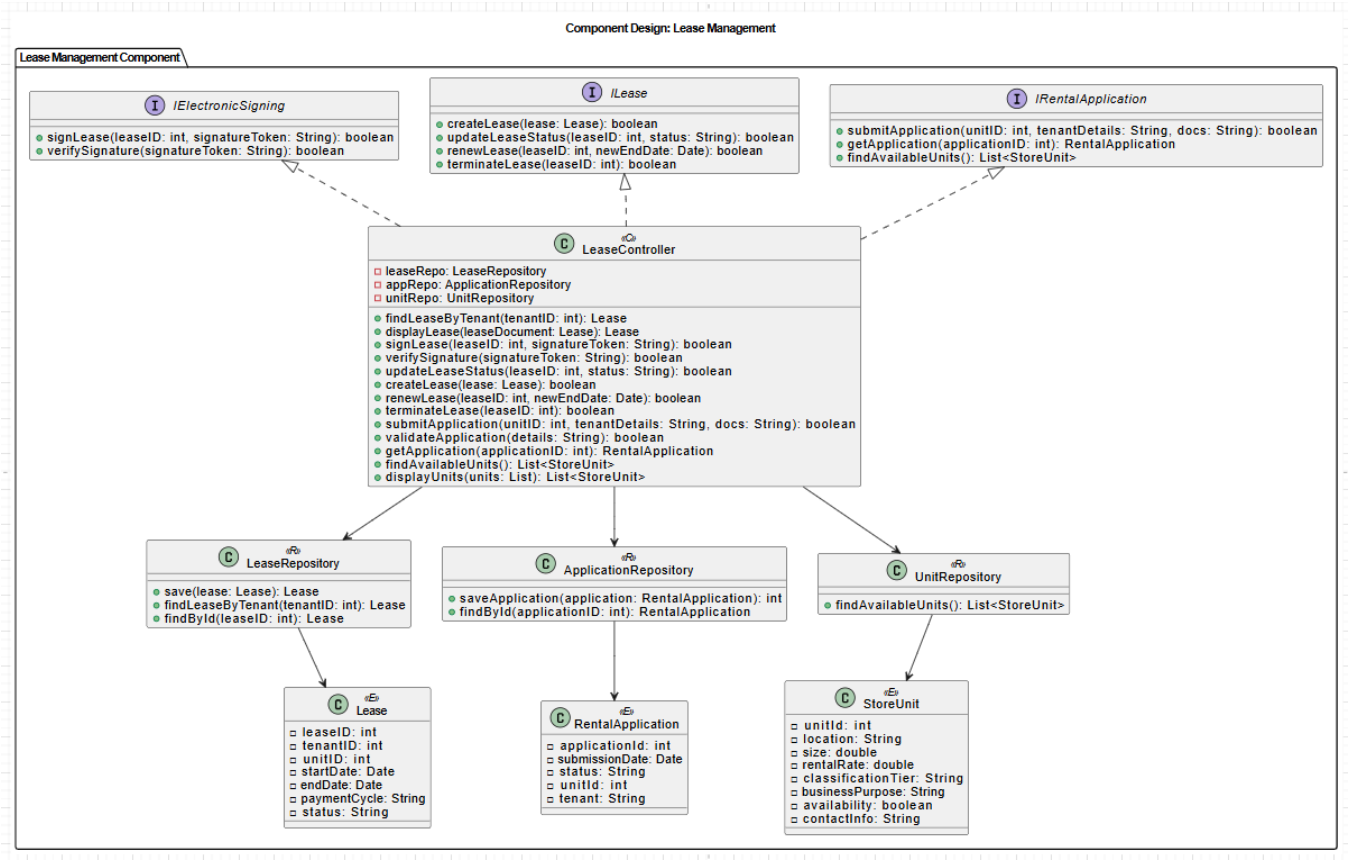
Detailed Class Diagrams

Detailed class diagram for each component in your system. You can have a separate class diagram per component (such as packages)

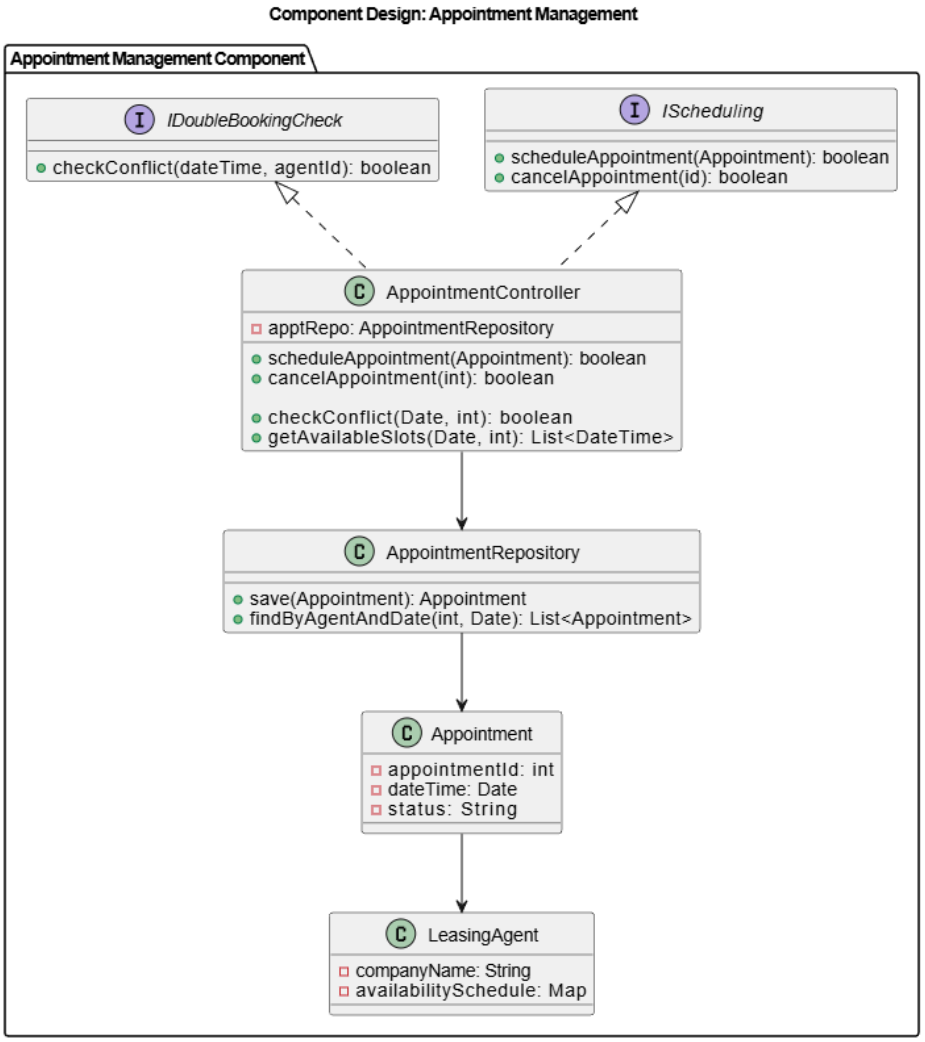
1) Property Management Component



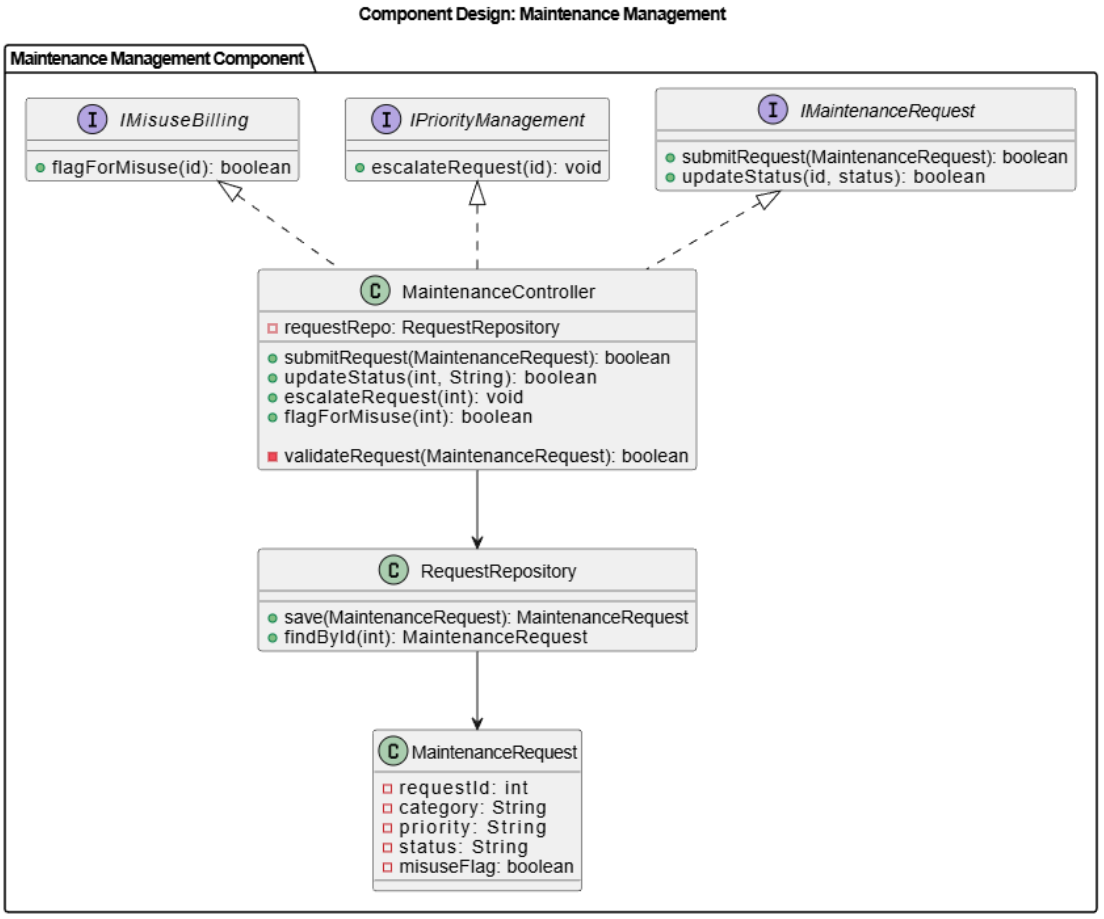
2) Lease Management Component



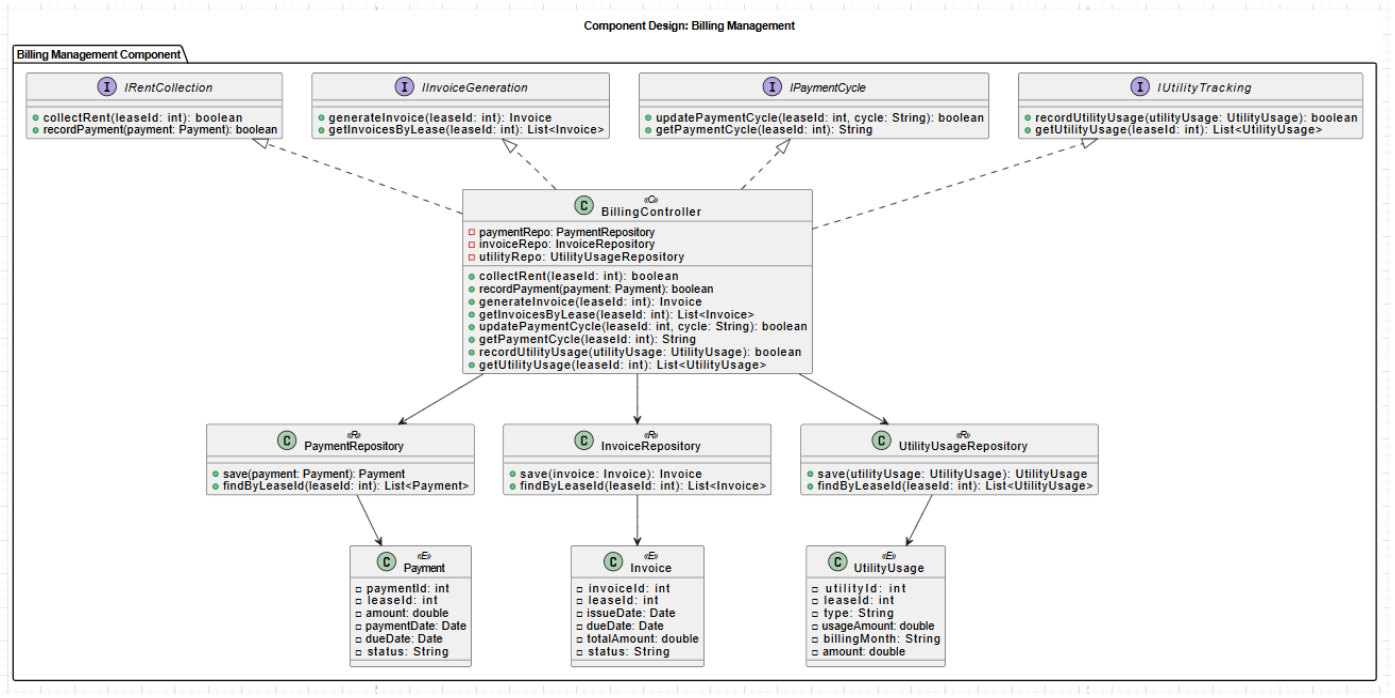
3) Appointment Management Component



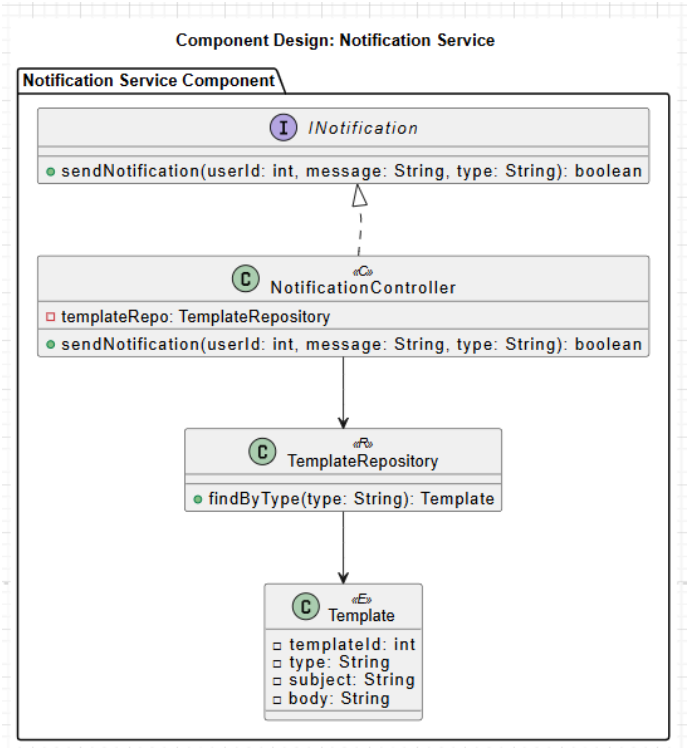
4) Maintenance Management Component



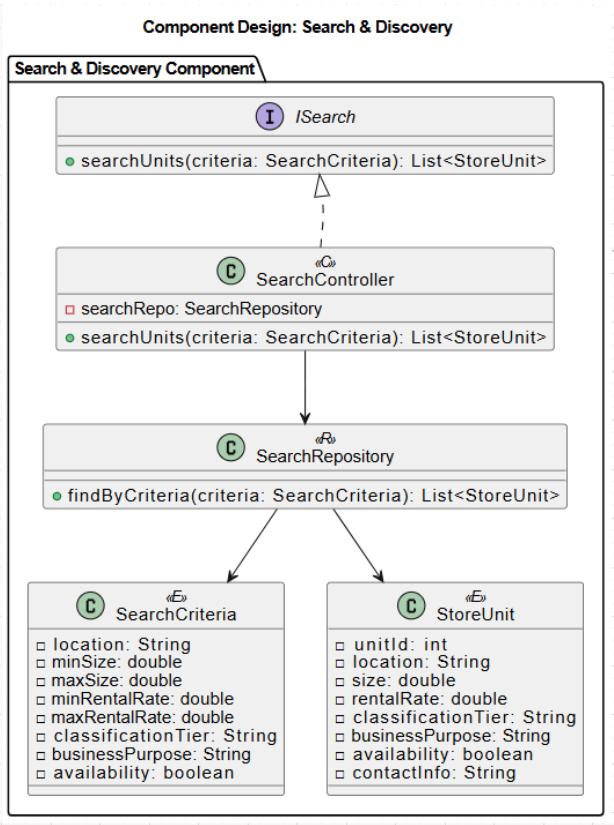
5) Billing Management Component



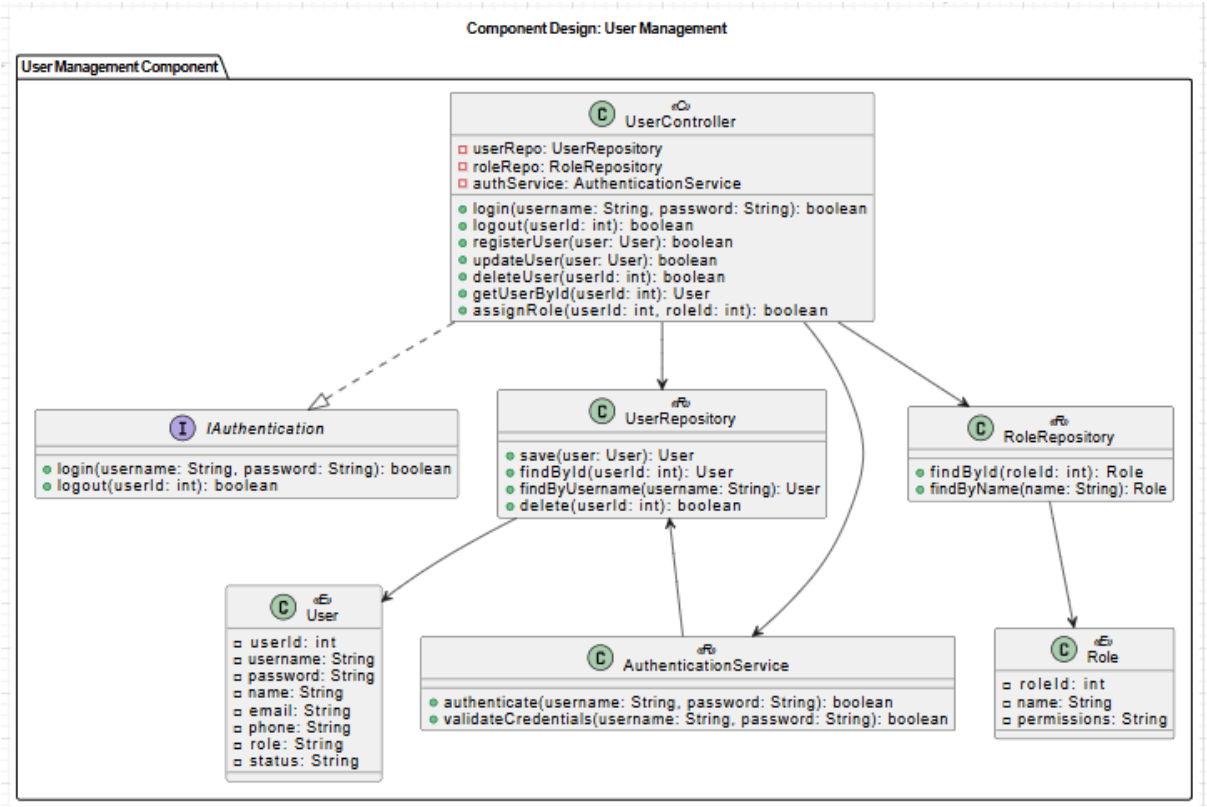
6) Notification Service



7) Search & Discovery Component



8) User Management Component



Use of Design Patterns

In this section you will:

1. List each design pattern used
2. Provide the rationale for each design pattern you used
3. Point where in your code the design pattern is implemented

Component Diagram	Design Pattern(s) Used	How the Pattern Appears in the Diagram / Rationale
Property Management Component	MVC, Repository, Facade	• **MVC:** PropertyController acts as the controller that receives requests from the UI layer (not shown) and coordinates property and unit operations. Domain objects (StoreUnit, Mall) encapsulate business data, while workflow and validation logic is handled in the controller rather than embedded in the entities. • **Repository:** UnitRepository and MallRepository encapsulate all persistence logic (e.g., save, findById, findAll, delete). This prevents PropertyController from containing

		SQL or database-specific code and keeps data access interchangeable (e.g., switching database implementations later). ● **Facade (via Interfaces):** IPropertyCRUD exposes a simplified, high-level entry point for property and unit CRUD operations. Other components interact with this interface instead of directly accessing controllers or repositories, reducing coupling and limiting ripple effects if the internal implementation changes.
Lease Management Component	MVC, Repository, Facade	● **MVC:** LeaseController coordinates end-to-end lease workflows (submit application, create lease, sign lease, renew/terminate). Entities like Lease and RentalApplication represent the business state, while the controller handles the sequence and validation of steps. ● **Repository:** LeaseRepository and ApplicationRepository isolate all data operations (save, findById). This keeps lease logic independent of storage details and supports unit testing using mocks/fakes. ● **Facade (via Interfaces):** ILease, IRentalApplication, and IElectronicSigning provide stable contracts for lease operations. This allows other components (Billing, Notifications) to interact with lease actions without needing access to how leases/applications are persisted or how signing is validated.
Appointment Management Component	MVC, Repository, Strategy, Facade	● **MVC:** AppointmentController manages scheduling requests from the UI (not shown), coordinates availability lookups, triggers conflict checks, and finalizes appointment creation/cancellation. The Appointment entity holds appointment state (appointmentId, dateTime, status) while the controller owns the workflow. ● **Repository:** AppointmentRepository persists appointments and supports conflict queries (e.g., findByAgentAndDate(...)). This ensures conflict detection can be performed reliably using persisted data rather than UI state.

		• **Strategy:** IDoubleBookingCheck encapsulates conflict-checking rules so the scheduling logic can evolve (e.g., check conflicts by agent only, by unit only, or by both) without rewriting the controller. The controller/service calls the strategy and doesn't hardcode the rules. • **Facade (via Interface):** IScheduling acts as the public scheduling contract to the rest of the system. It exposes scheduling operations while hiding internal collaboration between controller, repository, and the double-booking strategy.
Maintenance Management Component	MVC, Repository, Facade	• **MVC:** MaintenanceController coordinates maintenance workflows: submitting requests, escalation, and misuse-flagging/status updates. The MaintenanceRequest entity represents request data (category, priority, misuseFlag) while the controller enforces the process rules and state transitions. • **Repository:** RequestRepository handles persistence and retrieval of maintenance requests (save, findById). This keeps the controller free of DB logic and allows request workflows to be tested independently of the database. • **Facade (via Interfaces):** IMaintenanceRequest, IPriorityManagement, and IMisuseBilling provide clear, role-focused entry points into the component. They reduce coupling by letting other components call "maintenance operations" without needing to know how escalation rules or misuse billing are implemented internally.
Billing Management Component	MVC, Repository, Facade, Strategy	• **MVC:** BillingController is the Controller coordinating billing workflows (rent collection, invoice generation, payment cycle updates, utility tracking). It exposes high-level operations like collectRent(leaseId), generateInvoice(leaseId), recordPayment(payment), and delegates persistence to repositories and business rules to the component interfaces. Domain objects: Payment, Invoice, UtilityUsage.

		• **Repository:** Persistence is separated into dedicated repositories: PaymentRepository (save, findByLeaseId), InvoiceRepository (save, findByLeaseId), and UtilityUsageRepository (save, findByLeaseId). This keeps DB logic out of the controller and allows swapping storage/DB implementations without changing the controller logic. • **Facade (via Interfaces):** The interfaces IRentCollection, IInvoiceGeneration, IPaymentCycle, and IUtilityTracking act as clear service entry points for billing-related capabilities. Other components (Lease Management, Maintenance, etc.) can depend on these interfaces rather than internal classes, reducing coupling and keeping billing functionality "modular." • **Strategy (Payment Cycle):** IPaymentCycle represents a pluggable policy for how billing is computed/scheduled (e.g., monthly vs weekly vs custom cycles). The controller uses updatePaymentCycle(leaseId, cycle) and getPaymentCycle(leaseId) so payment-cycle logic can evolve (or be replaced) without rewriting invoice/rent collection flows.
Notification Service (Infrastructure Component)	MVC, Repository, Facade	• **MVC:** NotificationController acts as the Controller that coordinates sending notifications. It receives sendNotification(userId, message, type) from callers/UI (not shown), applies business rules (e.g., pick template by type, set status), and orchestrates persistence + sending. Domain objects used: Notification, Template. • **Repository:** NotificationRepository encapsulates persistence for notifications (save(notification), findByUserId(userId)), so controllers/services don't contain DB logic. TemplateRepository encapsulates template lookup (findByType(type)), keeping template retrieval separated and reusable. • **Facade (via Interface):** INotification is the single entry point other components call.

		Instead of exposing internal repositories, the system depends on INotification.sendNotification(...). This reduces coupling and keeps notifications consistent across flows (appointments, lease signing, rental applications, billing, maintenance).
User Management Component	MVC, Repository, Facade, Strategy (optional)	• **MVC:** UserController is the Controller coordinating user actions (login/logout/register/update/delete/assignRole). Domain entities are User and Role. UI is not shown, but the controller is the clear "middle layer" between UI and data/services. • **Repository:** UserRepository and RoleRepository encapsulate all data access (save, findById, findByUsername, findByName, delete). This keeps UserController from directly calling the DB. • **Facade (via Interface):** IAuthentication exposes a clean entry point (login, logout) so other components can call authentication without knowing internal details (repositories, authService, etc.). It reduces coupling across the system. • **Strategy (Optional):** AuthenticationService can be treated as a pluggable authentication strategy (e.g., local DB auth now, OAuth/SSO later) because the controller depends on the service, not the mechanism.
Search & Discovery Component	MVC, Repository, Facade	• **MVC:** SearchController is the Controller responsible for the search workflow. It receives the search request (criteria) from the UI (not shown), coordinates the search operation, and returns a List<StoreUnit> to be displayed. SearchCriteria and StoreUnit act as the domain/data objects used in the search flow. • **Repository:** SearchRepository encapsulates the query logic (findByCriteria(criteria)) so search/filtering rules and DB query details are not embedded in the controller. This keeps the controller clean and makes search easier to test or optimize (e.g., adding indexes, changing query strategy) without changing the controller contract. • **Facade (via Interface):** ISearch exposes a single high-level operation

		(searchUnits(criteria): List<StoreUnit>) that other components can call without knowing internal structure (controller + repository). This reduces coupling and makes it easy to reuse search functionality across the system (Tenant search, Leasing Agent search, Admin search) while keeping a consistent entry point.
Additional / Implied Design Patterns		
Property Management Component + Billing Management Component	Factory / Builder	Object creation for complex domain entities (e.g., StoreUnit creation in Property Management and Invoice generation in Billing Management) follows a structured construction process. While not modeled as explicit Factory/Builder classes in the diagrams, the design supports introducing these patterns during implementation to centralize and standardize object creation logic.
Notification Service Component	Observer	The Notification Service currently operates as a shared service/facade invoked directly by components. The design naturally supports evolving into an Observer pattern in future iterations, where domain events (e.g., appointment booked, lease signed, payment overdue) could publish notifications to subscribed listeners without increasing coupling.

Data Storage

Discuss the data management options that you will use in your implementation (SQL DB, Files, NoSql, etc.)

The Real Estate Management System (REMS) will primarily use a relational SQL database to store core system data. An SQL database is well suited for this system because the data is highly structured and contains many relationships, such as tenants linked to leases, leases linked to invoices, and invoices linked to payments. SQL also supports transactions and constraints, which are important for maintaining data integrity in workflows like rent collection, lease signing, and appointment scheduling.

Entities such as users, roles, store units, appointments, rental applications, leases, invoices, payments, and maintenance requests will be stored in the SQL database. Constraints and validation logic will be used to enforce rules such as preventing double-booked appointments and ensuring tenants have an active lease before submitting maintenance requests.

For data that does not fit well into relational tables, such as signed lease documents and uploaded maintenance photos, file or object storage will be used. Only metadata (e.g., file path, upload date, associated record ID) will be stored in the database, keeping the system efficient while maintaining traceability.

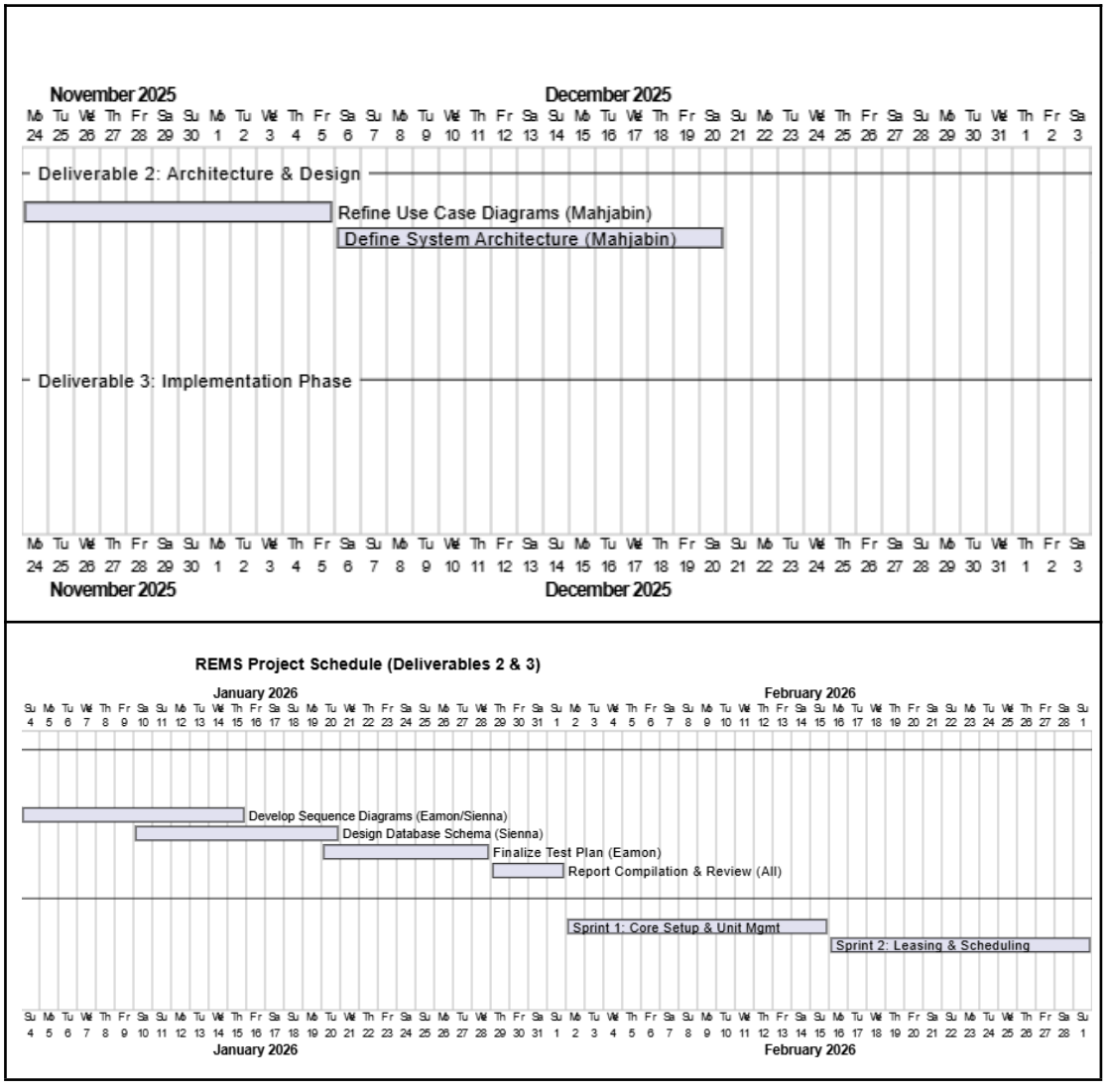
This combination of SQL storage for structured data and file storage for documents provides strong consistency, reliability, and scalability for the REMS application.

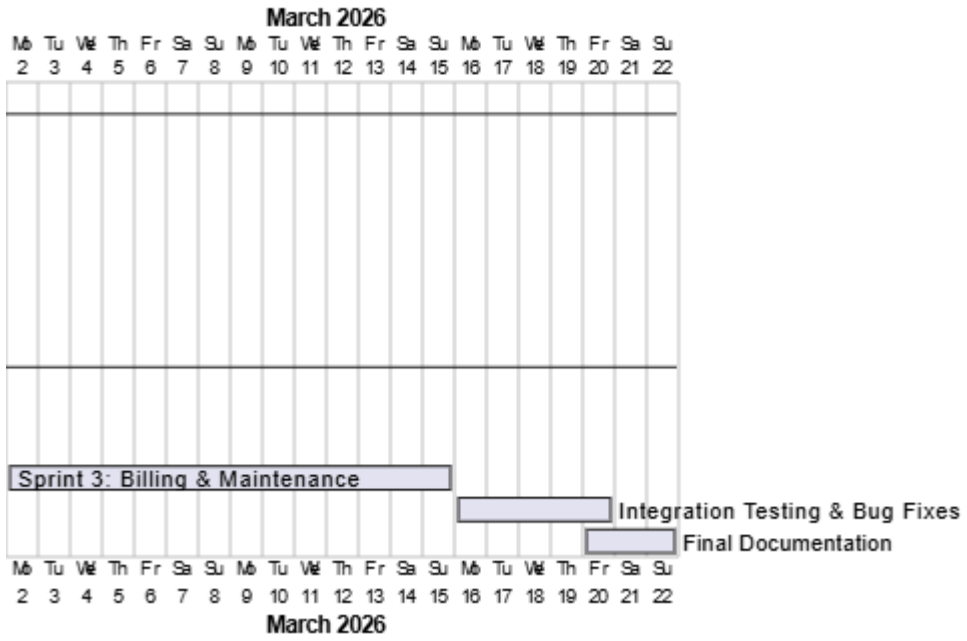
Section 8:

Project Management Plan

Gantt Chart

In this section you will provide a revised Gantt Chart of your activities. You can use as a base the one submitted in Deliverable 1.





Project Backlog and Sprint Backlog

Assuming you follow a Scrum process model, provide a revised list (with respect to Deliverable 1) of product backlog items so that you can select items for your Sprint backlog. Make sure the product backlog list and the tasks in each product backlog item are consistent with the Gantt Chart in Section 8.1. Follow the templates provided for you on eClass and include their picture here. You may also add them to your repo for reference.

- agile-sprint-backlog-Team3.xlsx
- agile-product-backlog-Team3.xlsx

Sprint Backlog updates for Deliverable 2			
Item / User Story	**Task / Area**	**Update Status**	**Notes (Sprint Review–Ready)**
US-02: Develop System Architecture & UML Design	High-level Architecture / Component Diagram	**Done**	Component-based layered architecture completed and aligned with use cases
	Domain Model (Class Diagram)	**Done**	Domain model completed with entities, attributes, associations, and multiplicities

	Sequence Diagrams	**Done**	5 sequence diagrams completed (exceeds minimum of 4)
	ERD / Database Schema	**In-Progress / Deferred**	Logical data design covered via domain model and data storage discussion; full ERD deferred to implementation
US-17: Schedule Viewing Appointments	Scheduling Flow Design	**Done**	Sequence diagram completed showing availability lookup and booking
	Double-Booking Prevention	**Done**	Conflict detection modeled using alternate paths
	Notification Handling	**Done**	Confirmation notifications modeled via Notification Service
Sprint-level Update	Backlog Accuracy	**Updated**	Tasks moved from Not Started → Done/In-Progress to reflect Deliverable 2 design work

Product Backlog updates for Deliverable 2			
Item / User Story	**Task / Area**	**Update Status**	**Notes (Sprint Review–Ready)**
US-02: Develop System Architecture & UML Design	High-level Architecture / Component Diagram	**In-Progress**	Domain model completed, component diagram completed, 5 sequence diagrams completed (exceeds minimum), data storage design documented; ERD / Database schema deferred
US-17: Schedule Viewing Appointments	Scheduling Flow Design	**In-Progress**	Design artifacts completed (sequence diagram incl. double-booking + notifications); implementation planned for Sprint 2

Group Meeting Logs

Deliverable 1 Meeting Logs

Add the minutes of each meeting, listing the attendance, what the topics of discussion in the meeting were, any decisions that were made, and which team members were assigned which tasks. These minutes must be submitted with the project report in each deliverable and will provide input to be used for the overall assessment of the project.

Present Group Members	Meeting Date	Issues Discussed / Resolved
Sienna, Mahjabin, Eamon	Oct 18, 2025	Defined project scope and assigned sections. Sienna to start on Introduction and Problem Statement. Mahjabin to draft System Specs. Eamon to begin User Stories and Project Management Plan.
Sienna, Mahjabin, Eamon	Nov 01, 2025	Discussion: Alignment of Requirements and Stories. Resolved: Mahjabin presented the draft Functional Requirements (Section 3). Eamon reviewed them to ensure they mapped 1-to-1 with the User Stories (Section 5) and created the initial backlog items based on them. Sienna finalized the "Existing Solutions" and "Motivation" text to ensure the scope wasn't too broad.
Sienna, Mahjabin, Eamon	Nov 15, 2025	Discussion: Diagrams and Timeline. Resolved: Reviewed Mahjabin's Use Case Diagram (Section 4) to ensure no actors were missing. Eamon presented the Gantt Chart (Section 6) to confirm the sprint dates with the team. Sienna completed the Design Objectives and Limitations sections.
Sienna, Mahjabin, Eamon	Nov 22, 2025	Discussion: Final Report Assembly. Resolved: Merged Eamon's Testing Plan and Project Management sections with Sienna's Intro and Mahjabin's Specs. Sienna generated the Table of Contents and completed both product and sprint

		backlogs. Eamon verified that the "Testing Plan" (Section 7) covered the high-priority requirements listed in Section 3. Submitted final PDF.

Deliverable 2 Meeting Logs

Present Group Members	Meeting Date	Issues Discussed / Resolved
Sienna, Mahjabin, Eamon	Jan 08, 2026	Discussion: Kick-off for Deliverable 2 Design Phase. Resolved: Reviewed feedback from Deliverable 1 and updated test cases. Mahjabin took ownership of the Domain Model and Architecture. Sienna handled Data Storage, Design Patterns, Sequence Diagrams and update Product + Sprint Backlogs. Eamon assigned to create Sequence Diagrams for the "Tenant" workflows (Applications & Leases) and update the Gantt chart.
Sienna, Mahjabin, Eamon	Jan 22, 2026	Discussion: Connecting Diagrams to Requirements. Resolved: Eamon and Sienna reviewed the Sequence Diagrams to ensure they matched the Class Diagrams Mahjabin created. Identified a missing "Notification Service" component in the original design and agreed to add it to the Component Diagram (Section 7.1) and Sequence flows. Noticed some attributes were inconsistent between domain model and class diagrams and aligned them.

		Recognized additional design patterns that were missing and needed to be mentioned.
Sienna, Mahjabin, Eamon	Jan 31, 2026	Discussion: Final Review and Formatting. Resolved: Merged all sections into the final PDF. Eamon confirmed that the Test Cases in Section 9 covered the new "Lease Signing" logic. Sienna updated the Table of Contents and organized doc plus tweaked Design Patterns Rationale. Mahjabin reviewed consistency between domain model, component diagram and class diagrams. Validated that the Gantt Chart (Section 8.1) accurately reflects the upcoming Sprint schedule for Deliverable 3.

Section 9:

Test Driven Development

In this section you will extend/revise the test cases you have submitted in Deliverable 1. The idea is by Deliverable 3 you will have a comprehensive list of functional-level test cases. The information will be provided in the form of a table as follows:

Highlight in yellow or add in a separate section the new and/or revised test cases.

Field	Details
Test Case ID	TC-001
Title	Search Store Unit - Valid Price Filter
Requirement	FR4: The system shall allow tenants to search using filters.
Preconditions	Database contains units with rents: $1000, $2000, $3000. User is on Search Page.
Steps	1. Enter "$2500" in the "Max Price" filter.

	2. Click "Search".
Expected Result	The system displays the units costing $1000 and $2000 only.
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	High

Field	Details
Test Case ID	TC-002
Title	Appointment Scheduling - Conflict Check
Requirement	FR7: The system shall prevent double-booking.
Preconditions	Agent A is already booked for Nov 20th at 2:00 PM.
Steps	1. User B attempts to book Agent A for Nov 20th at 2:00 PM. 2. User submits booking request.
Expected Result	System rejects the request and displays an error: "Time slot unavailable."
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	High

Field	Details
Test Case ID	TC-003
Title	Submit Rental Application – Valid Information
Requirement	FR9: Allow tenants to submit digital rental applications
Preconditions	Tenant is logged in
Steps	1. Navigate to "Submit Rental Application" 2. Complete all mandatory fields 3. Upload required documents 4. Click "Submit"

Expected Result	● Application is stored ● Leasing agent receives notification ● Tenant sees confirmation message
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	Medium

Field	Details
Test Case ID	TC-004
Title	Manage Store Units – Add New Unit
Requirement	FR1: Add, update, view, and delete store unit records.
Preconditions	Leasing Agent is logged in
Steps	1. Select "Manage Store Units" 2. Click "Add Unit" 3. Enter valid unit info (size, rate, classification, purpose) 4. Save
Expected Result	New unit stored and unit appears in the unit list
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	High

Field	Details
Test Case ID	TC-005
Title	Calculate Bill
Requirement	FR:19: Consolidate rent and utility charges into monthly bill
Preconditions	● Lease exists with monthly rent ● Billing date reached
Steps	1. System calculates rent 2. System applies discounts if applicable 3. System generates invoice

Expected Result	● A monthly bill gets created with rent and utility charges included and the user is notified. ● Invoice is created
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	High

Field	**Details**
Test Case ID	TC-006
Title	User can successfully login
Requirement	FR23: Support user authentication
Preconditions	User is logged out
Steps	1. User clicks “Login” 2. User enters username and password 3. If credentials are correct they are logged in and brought to home page 4. If credentials are incorrect they are prompted to try again
Expected Result	Upon successful login user is redirected to home page
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	High

Field	**Details**
Test Case ID	TC-007
Title	Attempt to Sign Unapproved Lease Agreement
Requirement	FR11: Allows user to electronically sign lease agreement
Preconditions	● Tenant is logged in ● Tenant has no approved lease to sign
Steps	1. Click on “Leases” 2. Click on an unapproved lease to sign 3. Attempt to sign it

Expected Result	• System prevents access to lease signing. • Message displayed: "Your application must be approved before signing."
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	Medium

Field	Details
Test Case ID	TC-008
Title	Role-Based Access Control - Units/Property
Requirement	FR24: Enforce role-based access control
Preconditions	• Tenant is logged in and has the role of tenant
Steps	1. Tenant clicks on "Properties"
Expected Result	• Tenant does not have any options to add, update or delete any properties • They can only search for properties
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	High

NFR: Performance Analysis, Security, Compatibility & Responsiveness

Field	Details
Test Case ID	TC-009 ****UPDATED****
Title	Search Response Time
Requirement	NFR1: System shall process search queries within 2 seconds.
Preconditions	Database populated with ~~100+~~ 1,000+ mock unit records.
Steps	1. Initiate a search query with multiple filters (price, size, location). 2. Measure time from submission to result rendering.
Expected Result	Results load in < 2.0 seconds under increased dataset size.
Actual Result	*(To be filled during testing)*

Status	*(To be filled during testing)*
Priority	Medium

Field	Details
Test Case ID	**TC-010**
Title	Input Validation - Special Characters
Requirement	**NFR5**: The system shall validate user input to prevent malformed or invalid data.
Preconditions	• Admin is logged in. • User is on the "Add Store Unit" or "Edit Profile" page.
Steps	1. Navigate to a text input field (e.g., "Unit Name"). 2. Enter a script tag string: <script>alert('test')</script>. 3. Click "Save".
Expected Result	• The system rejects the input or sanitizes it (removing the script tags). • An error message "Invalid characters detected" is displayed. • The script does **not** execute.
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	High

Field	Details
Test Case ID	**TC-011**
Title	Delete Confirmation Dialog
Requirement	**NFR6**: The system shall provide confirmation dialogs for critical actions.
Preconditions	• Admin is logged in. • "Manage Store Units" list is visible.

Steps	1. Locate an existing store unit. 2. Click the "Delete" button. 3. Wait for the popup. 4. Click "Cancel".
Expected Result	• A confirmation dialog appears asking "Are you sure?". • Upon clicking "Cancel", the dialog closes and the unit is **not** deleted.
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	Medium

Field	Details
Test Case ID	**TC-012**
Title	Browser Layout Consistency
Requirement	**NFR8**: The system shall be compatible with standard web browsers (Chrome, Firefox, Edge).
Preconditions	• The system is deployed and accessible via URL.
Steps	1. Open the "Tenant Dashboard" in Google Chrome and verify layout. 2. Open the same page in Mozilla Firefox and verify layout. 3. Compare the navigation bar and button alignment.
Expected Result	• The layout, fonts, and button positions appear identical and functional across both browsers.
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	Low

Field	Details
Test Case ID	**TC-013**
Title	Mobile Interface Adaptation
Requirement	**NFR7**: The system shall present a user-friendly, responsive interface for mobile devices.
Preconditions	• User is accessing the system via a desktop browser.
Steps	1. Open the application in a browser. 2. Resize the browser window width to 375px (mobile view). 3. Observe the navigation menu and data tables.
Expected Result	• The navigation menu collapses into a "hamburger" icon. • Data tables stack or become scrollable without breaking the layout. • No horizontal scrolling is required for main content.
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	Medium

Field	Details
Test Case ID	TC-014 ****NEW****
Title	Search Store Units - Response Time Under Load
Requirement	NFR1: Search queries must load within 2 seconds.
Preconditions	Database contains 1,000+ store units. System deployed locally or staging. User is on Search Page.
Steps	1. Apply filters: mall="Square One", maxRent=2500, available=true, businessPurpose=Retail. 2. Click Search. 3. Repeat 10 times (same filters).

	4. Record response time each run (start = request sent, end = results displayed).
Expected Result	Average response time ≤ 2.0s and no single run > 3.0s. Results list is correct and UI does not freeze.
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	High

Field	**Details**
Test Case ID	TC-015 ****NEW****
Title	Schedule Viewing Appointment - Performance (Slot Lookup)
Requirement	NFR1: UI actions should respond within 2 seconds.
Preconditions	Agent has availability schedule; system contains 200+ existing appointments; Tenant is logged in.
Steps	1. Go to Schedule Viewing Appointments. 2. Select a unit. 3. Request available times for a 7-day window. 4. Measure time from “Request slots” click to slot list display.
Expected Result	Available slots display in ≤ 2.0 seconds. No duplicate slots shown.
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	Medium

Field	**Details**
Test Case ID	TC-016 ****NEW****
Title	Appointment Scheduling - Conflict Check Performance
Requirement	NFR1, FR7 (prevent double booking)
Preconditions	Appointment repository contains 500+ appointments across multiple agents/units. Tenant logged in.

Steps	1. Attempt to schedule a slot that requires a conflict check (same agent + same time). 2. Measure time from “Confirm booking” to conflict result message. 3. Repeat 10 times with different agents.
Expected Result	Conflict decision returned in ≤ 1.0 second (recommended threshold) and never exceeds 2.0 seconds. Correctly blocks conflicts.
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	High

Field	**Details**
Test Case ID	TC-017 ****NEW****
Title	Generate Monthly Bill - Processing Time
Requirement	NFR1 + FR19 (consolidated billing)
Preconditions	Tenant lease exists. UtilityUsage exists for billing month. Database contains 100 tenants with leases (mock).
Steps	1. Trigger monthly billing job (manual admin trigger or scheduled run). 2. Measure runtime from job start to invoice creation completion for all tenants.
Expected Result	Batch billing completes within X seconds (set target, e.g., ≤ 10 seconds for 100 tenants in dev/staging). All invoices created with correct totals.
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	Medium

Field	**Details**
Test Case ID	TC-018 ****NEW****
Title	Database Query Optimization - Search Uses Index
Requirement	NFR1 (performance)

Preconditions	SQL DB in use. Index exists (or planned) on StoreUnit(mallId, availability, rentalRate) or similar.
Steps	1. Run an EXPLAIN (or DB query plan output) for the search query with filters. 2. Verify index usage. 3. Execute query and measure time.
Expected Result	Query plan shows index used (not full table scan) and execution time supports ≤2 second target at 1,000+ units.
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	Medium

Field	**Details**
Test Case ID	TC-019 ****NEW****
Title	UI Responsiveness - No Freeze During Large Result Set
Requirement	NFR1 (performance) + NFR7 (responsive UI)
Preconditions	Database contains 1,000+ units. Tenant logged in.
Steps	1. Search with broad filters (returns 200+ results). 2. Scroll results list. 3. Click into a unit detail page.
Expected Result	UI remains responsive: scroll works smoothly, page interaction remains usable, no crash, and navigation occurs without long blocking delay (>2 seconds).
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	Low

Field	**Details**
Test Case ID	TC-020 ****NEW****
Title	Notification of Overdue Payments

Requirement	FR17: The system shall track tenant payments and send notifications for overdue payments to tenants and alerts to administrative staff.
Preconditions	A tenant has an overdue payment
Steps	1. System checks payments that have been past their due date and have not been paid 2. System sends email notification to tenant of overdue payment with warning
Expected Result	Tenant receives overdue payment email
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	High

Field	**Details**
Test Case ID	TC-021 ****NEW****
Title	Apply Charges for Misused Maintenance Requests
Requirement	FR22: The system shall apply charges to tenant accounts for misuse-related maintenance requests.
Preconditions	A tenant has submitted a misused maintenance request
Steps	1. Admin checks maintenance request 2. Admin marks it as misused 3. An email is sent to tenant with a notification of the charges placed on their account due to the misuse-related maintenance request
Expected Result	Tenant receives misuse maintenance request notification email and charge on their account
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	Medium

Field	Details
Test Case ID	TC-022 ****NEW****
Title	Escalation of Maintenance Requests
Requirement	FR21. The system shall categorize maintenance requests by priority, automatically escalating emergency requests.
Preconditions	There is an emergency maintenance request
Steps	1. System organized maintenance requests by priority 2. Admin reviews and escalates the emergency request
Expected Result	Tenant's maintenance request is escalated
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	Medium

Field	Details
Test Case ID	TC-023 ****NEW****
Title	Bill is Created based on Payment Cycle
Requirement	FR14. The system shall allow tenants to choose a payment cycle, including monthly, quarterly, bi-annual, or annual options and calculate rental charges dynamically based on the selected payment cycle.
Preconditions	A tenant has a lease
Steps	1. Tenant chooses their payment cycle (monthly, quarterly, bi-annual, or annual) 2. Bill is calculated according to that cycle 3. Invoice is generated and sent based on that cycle

Expected Result	Tenant receives invoice based on payment cycle
Actual Result	*(To be filled during testing)*
Status	*(To be filled during testing)*
Priority	High