SOEN390 - Software Engineering Team Design Project
Team 6 - Deliverable 5

Compiled Report

Winter 2024

Done by:
Hoang Minh Khoi Pham 40162551
Michaël Gugliandolo 40213419
Jessey Thach 40210440
Mahanaim Rubin Yo 40178119
Vanessa DiPietrantonio 40189938
Ahmad Elmahallawy 40193418
Clara Gagnon 40208598
Khanh Huy Nguyen 40125396
Jean-Nicolas Sabatini-Ouellet 40207926
Mohamad Mounir Yassin 40198854

Professor Junqiu Yang
Department of Computer Science and Software Engineering
Gina Cody School of Engineering and Computer Science

Concordia University

Table of Contents

1. Final Product Vision Statement **10**
1.1. Introduction 10
1.1.1. References 10
1.2. Positioning 10
1.2.1. Problem Statement 10
1.2.2. Product Position Statement 10
1.3. Stakeholder and User Descriptions 11
1.3.1. Stakeholder Summary 11
1.3.2. User Summary 13
1.3.3. User Environment 14
1.3.4. Key Stakeholder or User Needs 14
1.3.5. Alternatives and Competition 15
1.4. Product Overview 15
1.4.1. Product Perspective 15
1.4.2. Assumptions and Dependencies 15
1.5. Product Features 15
1.5.1. Login / Sign In 15
1.5.2. View Property Information 15
1.5.3. Manage Property under management 16
1.5.4. Financial System 16
1.5.5. Reservation System 16
1.5.6. Request System 16
1.5.7. Notification Page 16
1.6. Other Product Requirements 16
2. Requirements and User Stories Backlog **17**
2.1. Requirements 17
2.1.1. Non-Functional Requirements 17
2.1.2. Functional Requirements 18
2.2. User Stories Backlog 20
2.2.1. Epic 1: User Account Management 20
2.2.2. Epic 2: User Profile 21
2.2.3. Epic 3: Company Account Management 22
2.2.4. Epic 4: Registration Management 23
2.2.5. Epic 5: Condo Property Management 24
2.2.6. Epic 6: Requests System 28
2.2.7. Epic 7: Reservation System 29
2.2.8. Epic 8: Financial System 30
2.2.9. Epic 9: User Navigation System 32

2.2.10. Employee Management 32
2.2.11. Connections (Optional) 33
2.2.12. Promotions (Optional) 34
2.2.13. User Stories by Sprint 35
2.2.13.1. Sprint 1 35
2.2.13.2. Sprint 2 35
2.2.13.3. Sprint 3 36
2.2.13.4. Sprint 4 36
2.2.13.5. Sprint 5 36
3. Final Software Architecture 37
3.1. Introduction 37
3.1.1. Identifying Information 37
3.1.2. Supplementary Information 37
3.1.3. Other Information 38
3.1.3.1. Overview (optional) 38
3.1.3.2. Architecture Evaluations 38
3.1.3.3. Rationale for key decisions 38
3.2. Stakeholders and concerns 39
3.2.1. Stakeholders 39
3.2.1.1. User 39
3.2.1.2. Product Owner 40
3.2.2. Concerns 41
3.2.3. Concern-Stakeholder Traceability 41
3.3. Viewpoints 42
3.3.1. Logical View 42
3.3.1.1. Overview 42
3.3.1.2. Concerns and stakeholders 42
3.3.1.2.1. Concerns 42
3.3.1.2.2. Typical Stakeholders 43
3.3.1.2.3. “Anti-concerns” (optional) 43
3.3.1.3. Model kinds 43
3.3.1.3.1. Domain model 43
3.3.1.3.1.1. Domain model conventions 43
3.3.1.3.1.2. Domain model operations (optional) 43
3.3.1.3.1.3. Domain model correspondence rules 43
3.3.1.4. Class Diagram 43
3.3.1.4.1.1. Class Diagram conventions 43
3.3.1.4.1.2. Class Diagram operations (optional) 44
3.3.1.4.1.3. Class Diagram correspondence rules 44
3.3.2. Process View 44
3.3.2.1. Overview 44

3.3.2.2. Concerns and stakeholders 44
3.3.2.2.1. Concerns 44
3.3.2.2.2. Typical Stakeholders 44
3.3.2.2.3. “Anti-concerns” (optional) 44
3.3.2.3. Model kinds 44
3.3.2.3.1. Activity diagram 44
3.3.2.3.1.1. Activity Diagram conventions 45
3.3.2.3.1.2. Activity Diagram operations (optional) 45
3.3.2.3.1.3. Activity Diagram correspondence rules 45
3.3.2.3.2. Sequence diagram 45
3.3.2.3.2.1. Sequence Diagram conventions 45
3.3.2.3.2.2. Sequence Diagram operations (optional) 45
3.3.2.3.2.3. Sequence Diagram correspondence rules 45
3.3.3. Physical View 45
3.3.3.1. Overview 45
3.3.3.2. Concerns and stakeholders 45
3.3.3.2.1. Concerns 45
3.3.3.2.2. Typical Stakeholders 46
3.3.3.2.3. “Anti-concerns” (optional) 46
3.3.3.3. Model kinds 46
3.3.3.3.1. Deployment diagram 46
3.3.3.3.1.1. Deployment Diagram conventions 46
3.3.3.3.1.2. Deployment Diagram operations (optional) 47
3.3.3.3.1.3. Deployment Diagram correspondence rules 47
3.3.4. Development View 47
3.3.4.1. Overview 47
3.3.4.2. Concerns and stakeholders 47
3.3.4.2.1. Concerns 47
3.3.4.2.2. Typical Stakeholders 47
3.3.4.2.3. “Anti-concerns” (optional) 48
3.3.4.3. Model kinds 48
3.3.4.3.1. Component Diagram 48
3.3.4.3.1.1. Component Diagram conventions 48
3.3.4.3.1.2. Component Diagram operations (optional) 48
3.3.4.3.1.3. Component Diagram correspondence rules 48
3.3.5. Use Case View 48
3.3.5.1. Overview 48
3.3.5.2. Concerns and stakeholders 48
3.3.5.2.1. Concerns 48
3.3.5.2.2. Typical Stakeholders 49
3.3.5.2.3. “Anti-concerns” (optional) 49

3.3.5.3. Model kinds 49
3.3.5.3.1. Use Case Diagram 49
3.3.5.3.1.1. Use Case Diagram conventions 49
3.3.5.3.1.2. Use Case Diagram operations (optional) 50
3.3.5.3.1.3. Use Case Diagram correspondence rules 50
3.3.6. Operations on views 50
3.3.7. Correspondence rules 50
3.3.8. Examples (optional) 50
3.3.9. Notes (optional) 50
3.3.10. Sources 50
3.4. Views 51
3.4.1. View: Logical 51
3.4.1.1. Models 51
3.4.1.1.1. Domain Model 51
3.4.1.1.2. Class Diagram 51
3.4.1.2. Known Issues with View 52
3.4.2. View: Process 52
3.4.2.1. Models 52
3.4.2.1.1. Activity Diagram 52
3.4.2.1.1.1. Public User Registration 53
3.4.2.1.1.2. Public User Login 54
3.4.2.1.1.3. Simplified Registration System 54
3.4.2.1.1.4. Simplified Financial System 55
3.4.2.1.1.5. Company Employee Management 56
3.4.2.1.1.6. Request System 57
3.4.2.1.2. Sequence Diagram 58
3.4.2.1.2.1. Update User Profile 58
3.4.2.1.2.2. Simplified Reservation System 59
3.4.2.1.2.3. Simplified Financial System 59
3.4.2.2. Known Issues with View 60
3.4.3. View: Physical 60
3.4.3.1. Models 60
3.4.3.1.1. Deployment Diagram 60
3.4.3.2. Known Issues with View 62
3.4.4. View: Development 62
3.4.4.1. Models 62
3.4.4.1.1. Component Diagram 62
3.4.4.2. Known Issues with View 63
3.4.5. View: Use Case 64
3.4.5.1. Models 64
3.4.5.1.1. Use Case Diagram 64

3.4.5.2. Known Issues with View 69
3.5. Consistency an Correspondences 70
3.5.1. Known inconsistencies 70
3.5.2. Correspondences in the AD 70
3.5.3. Correspondence rules 71
4. Overall Retrospective 71
4.1. Introduction 71
4.2. What Went Wrong 72
4.2.1. Sprint 1 72
4.2.2. Sprint 2 73
4.2.3. Sprint 3 74
4.2.4. Sprint 4 75
4.2.5. Sprint 5 77
4.3. What Went Right 78
4.3.1. Sprint 1 78
4.3.2. Sprint 2 79
4.3.3. Sprint 3 80
4.3.4. Sprint 4 81
4.3.5. Sprint 5 82
4.4. Conclusion 83
4.4.1. Sprint 1 83
4.4.2. Sprint 2 83
4.4.3. Sprint 3 83
4.4.4. Sprint 4 84
4.4.5. Sprint 5 84
4.4.6. Overall Conclusion 84
5. Overall Testing 84
5.1. Testing Tool 84
5.1.1. Platform 85
5.1.1.1. Github 85
5.1.1.2. Sonarqube 85
5.1.2. Testing Module 85
5.2. Approach 85
5.2.1. Test Planning and Design 85
5.2.2. Unit Test 85
5.2.3. Integration Test 86
5.2.4. End-to-End (E2E) 86
5.3. Sprint 1 86
5.3.1. Metrics 86
5.3.1.1. Reliability 87
5.3.1.2. Maintainability 87

5.3.1.3. Security 87
5.3.1.4. Coverage 87
5.3.1.5. Duplication 87
5.3.1.6. Size 88
5.3.1.7. Complexity 88
5.3.1.8. Issues 88
5.3.2. Coverage 89
5.3.3. Acceptance test 92
5.3.4. Test Results 95
5.4. Sprint 2 96
5.4.1. Metrics 96
5.4.1.1. Reliability 96
5.4.1.2. Maintainability 97
5.4.1.3. Security 98
5.4.1.4. Coverage 98
5.4.1.5. Duplication 98
5.4.1.6. Size 99
5.4.1.7. Complexity 100
5.4.1.8. Issues 101
5.4.2. Coverage 102
5.4.3. Acceptance Tests 107
5.4.4. Test Results 111
5.5. Sprint 3 111
5.5.1. Metrics 111
5.5.1.1. Reliability 112
5.5.1.2. Maintainability 112
5.5.1.3. Security 113
5.5.1.4. Coverage 114
5.5.1.5. Duplication 115
5.5.1.6. Size 116
5.5.1.7. Complexity 116
5.5.1.8. Issues 117
5.5.2. Coverage 117
5.5.3. Acceptance Test 121
5.5.4. Test Result 126
5.6. Sprint 4 126
5.6.1. Metrics 126
5.6.1.1. Reliability 126
5.6.1.2. Maintainability 127
5.6.1.3. Security 128
5.6.1.4. Coverage 129

5.6.1.5. Duplication 130
5.6.1.6. Size 131
5.6.1.7. Complexity 131
5.6.1.8. Issues 132
5.6.2. Coverage 132
5.6.3. Acceptance Tests 136
5.6.4. Test Results 141
5.7. Sprint 5 141
5.7.1. Metrics 141
5.7.1.1. Reliability 141
5.7.1.2. Maintainability 142
5.7.1.3. Security 143
5.7.1.4. Coverage 144
5.7.1.5. Duplication 145
5.7.1.6. Size 146
5.7.1.7. Complexity 146
5.7.1.8. Issues 147
5.7.2. Coverage 147
5.7.3. Acceptance Test 151
5.7.4. Test Result 155
6. Final Risk Assessment and Management Plan 156
6.1. Introduction 156
6.2. Risk Assessment 158
7. Overall Release Plan 162
7.1. Sprint 1 Release Plan 162
7.2. Sprint 2 Release Plan 163
7.3. Sprint 3 Release Plan 164
7.4. Sprint 4 Release Plan 164
7.5. Sprint 5 Release Plan 165
8. UI Prototypes 166
8.1. UI Prototypes for Sprint 2 User stories 166
8.1.1. US-04 166
8.1.2. US-05 167
8.1.3. US-012 167
8.1.4. US-013 167
8.1.5. US-014 168
8.1.6. US-015 168
8.1.7. US-016 169
8.1.8. US-018 169
8.1.8.1. Scenario 1: User wants to upload new condo files to a new property. 169
8.1.8.2. Scenario 2: User wants to upload new condo files to an existing property 170

8.1.9. US-019 172
8.1.9.1. Scenario 1: User wants to enter detailed information for a new condo unit 172
8.1.9.2. Scenario 2: User wants to enter detailed information to an existing condo 173
8.2. UI Prototypes for Sprint 3 User stories 174
8.2.1. US-017 174
8.2.2. US-023 175
8.2.3. US-024 176
8.2.4. US-025 177
8.2.4.1. Condo Owner's Notification Page 177
8.2.4.2. Company Employee's Notification Page 178
8.2.4.3. Company Manager's Notification Page 178
8.2.5. US-038 179
8.2.6. US-039 179
8.2.7. US-053 181
8.2.8. US-054 182
8.3. UI Prototypes for Sprint 4 User stories 183
8.3.1. US-026 183
8.3.2. US-027 183
8.3.3. US-028 185
8.3.4. US-029 186
8.3.4.1. Modify Existing Reservation for a Common Facility 186
8.3.4.2. Delete Existing Reservation for a Common Facility 188
8.3.5. US-030 188
8.3.6. US-031 189
8.3.7. US-032 191
8.3.8. US-033 192
8.3.9. US-034 193
8.3.10. US-035 193

1. Final Product Vision Statement

1.1. Introduction

The purpose of this document is to collect, analyze, and define high-level needs and features of the Condo Management System. It focuses on the capabilities needed by the stakeholders, and the target users, and **why** these needs exist. The details of how the Condo Management System fulfills these needs are detailed in the use-case and supplementary specifications.

1.1.1. References

N/A

1.2. Positioning

1.2.1. Problem Statement

The problem of	inefficient and fragmented condo management
affects	condo owners, rental users, and management companies
the impact of which is	delayed responses to requests, lack of transparency in financial transactions, and difficulty in accessing essential condo-related information
a successful solution would be	to improve communication, streamline management processes, and enhance user experience by giving users a uniformed product where all of them can perform all the property-related activities in one place.

1.2.2. Product Position Statement

For	condo owners, rental users, and management companies
Who	want to perform condo management tasks
The (product name)	Condo Management System is a software product
That	centralizes all the condo-related activities into one application

Unlike	the current inefficient and fragmented condo management that needs users to go to different places/applications to do their tasks
Our product	group all condo-related tasks into one single application, simplifying the process of condo management and saving time for all users.

1.3. Stakeholder and User Descriptions

1.3.1. Stakeholder Summary

Name	Description	Responsibilities
Condo owners	Users who own the condo unit	● Provides regular payment of condo fees ● Provides submission of requests.
Rental Users	Users who is renting the condo unit	● Provide their own unique profile with their username, contact email, phone number,registration key
Condo Management Company	The company who is managing the properties which contain multiple condos.	● Registers their employees in the system and assigns different roles to them
Condo Management Employees	The employees who are managing and working on features for the property which contains multiple condos. This includes the manager, financial manager, and operators.	● Provide the system with the rate of the condo fee per square foot/parking spot. Provide the cost of each operation ● Provide information to the system

		concerning the facilities available
Property Owners/Investors	Individuals or entities who own the condominium properties managed by the management company.	● Provide oversight and direction to the management company ● Ensure the property is well-maintained and financially viable ● Approve major decisions such as budgeting, capital expenditures, and changes to property management contracts ● Collaborate with the management company to address concerns and achieve shared goals
Condo Association Boards	Boards or committees representing the collective interests of condo owners within a property, which may work closely with the management company.	● Develop and enforce rules and regulations for the condominium community ● Manage the condominium association's budget and finances ● Interface with the management company to address resident concerns, property maintenance issues, and other matters affecting the community ● Plan and oversee community events and initiatives

Regulatory Authorities (e.g. government bodies or agencies)	Government bodies or agencies responsible for overseeing and regulating condominium management practices.	● Enforce laws and regulations governing condominium management and operations ● Conduct inspections and audits to ensure compliance with applicable regulations ● Provide guidance and assistance to condominium associations and management companies on regulatory requirements
Developers	Developers of the condo management system	● Develop and maintain the website ● Ensure the functionality and usability of the system ● Address any technical issues or updates related to the system

1.3.2. User Summary

Name	Description	Responsibilities	Stakeholder
Condo management	Organizations responsible for managing condo properties.	- Ensure efficient daily operations - Financial stability - Adherence to regulations	- Property owners/investors - Condo association boards - Regulatory authorities (e.g. government bodies or agencies) - Condo management employees
Condo Owners	Individuals who own	- Access property information - Monitor Financial Status	self-represented.

	condominium units.	- Submits request	
Rental Users	Individuals renting condo units	- Profile management - Financial tracking - Property Access - Request submission - Reservation Participation - Communication	self-represented.

1.3.3. User Environment

- This system encompasses many tasks that all vary in duration. For example condo owners spend time reviewing their property dashboards, checking their financial information and submitting requests. Whilst a condo management company will spend time doing financial tracking and addressing user requests.
- The number of people involved can change depending on if new rental users join the system and as condo management companies expand their portfolios. On the same token, as the system's reach expands, the time to complete tasks such as responding to user requests could become longer.
- The user environment is primarily indoor and may involve both desktop and mobile usage. Condo owners and rental users may access the system from the comfort of their homes or on the go, emphasizing the need for a responsive and user-friendly interface.
- Currently, the CMS is designed to operate on both desktop and mobile platforms, ensuring accessibility for users across various devices. Future platform considerations may include updates for emerging technologies and changes in user preferences.
- Users within the condominium environment may be accustomed to using communication tools, document management systems, and financial software. The CMS should be designed with the capability to integrate with these applications, streamlining user interactions and ensuring a seamless experience.

1.3.4. Key Stakeholder or User Needs

Need	Priority	Concerns	Current Solution	Proposed Solutions
Accessible Information	High	Lack of transparency	Manual records	Centralized
Financial Transparency	High	Unclear financial status	Manual Tracking	Automated financial system

Efficient Requests	Medium	Delayed responses	Manual submission	Streamlined request process

1.3.5. Alternatives and Competition

Existing alternatives include manual condo management processes, potentially leading to inefficiencies and delays. Competitive alternatives may offer partial solutions but lack the comprehensive features proposed by the Condo Management System.

1.4. Product Overview

1.4.1. Product Perspective

The Condo Management System is an independent, self-contained product that integrates seamlessly with existing condo management practices. It provides a unified platform for users to manage profiles, financials, and property-related activities.

1.4.2. Assumptions and Dependencies

The successful implementation of the Condo Management System assumes the availability of stable system platforms, adherence to specified standards, and seamless integration with external applications. Dependencies include the timely provision of necessary information by stakeholders.

1.5. Product Features

1.5.1. Login / Sign In

Public Users (owners and rental users) will be able to create their own profile after providing a username, contact email, phone number with an option of adding a profile picture. Public users can then be registered as a condo owner or a rental user by providing a registration key given by their condo management company. Users will be able to sign out and login using their credentials.

1.5.2. View Property Information

Users will be presented with a dashboard of their properties, including their general information. ex: personal profile, condo information, financial status, status of the submitted request, etc.

1.5.3. Manage Property under management

Condo Management companies shall create their own property profiles providing the system with property name, unit count, parking count, locker count, address. They shall also upload condo files for each property. Condo management companies shall enter detailed information for each condo unit, each parking spot, and each locker in a building. Condo management companies shall send registration keys to unit owners or rental users for their dedicated units. Such registration keys shall be used by unit owners or rental users to link a condo unit with their profiles.

1.5.4. Financial System

The financial system is within the condo management system and it shall allow condo management companies to enter condo fees per square foot and per parking spot. It shall also calculate and present the condo fees to each unit owner. The financial system shall record operational budget and prompt companies to enter the cost for each operation. Finally, it shall produce an annual report.

1.5.5. Reservation System

The condo management system shall contain a simplified reservation system. Condo management companies shall be able to set up common facilities that require reservations. Condo owners and rental users shall be able to reserve these common facilities.

1.5.6. Request System

Condo owners shall submit requests such as moving in/out, intercom changes, requesting access and so on. Each request shall be assigned to the appropriate employee.

1.5.7. Notification Page

All users shall have a notification page, where they can view the latest activities in their submitted or assigned requests.

1.6. Other Product Requirements

N/A

2. Requirements and User Stories Backlog

2.1. Requirements

2.1.1. Non-Functional Requirements

Requirement 1: The Condo Management System shall ensure mobile accessibility on Android, iOS, Linux, MacOS, and Windows devices, promoting usability for users across diverse platforms.

- Quality Attribute: Usability
- Sub-Quality Attribute: Accessibility
- Subject: Condo Management System
- Action: Accessible on multiple platforms
- Constraints of Action: Accessible on Android, iOS, Linux, MacOS, and Windows

Requirement 2: The Condo Management System shall support English and at least one additional language, enhancing usability for a multilingual user base.

- Quality Attribute: Usability
- Sub-Quality Attribute: Accessibility
- Subject: Condo Management System
- Action: Multilingual support
- Constraints of Action: Available in English and at least one additional language

Requirement 3: Users shall have the flexibility to log in using their Gmail account or other Single Sign-On options.

- Quality Attribute: Usability
- Sub-Quality Attribute: Authenticity
- Subject: User
- Action: Login
- Constraints of Action: Valid login through Gmail or other Single Sign-On options

Requirement 4: The system shall have a response time of less than 5 seconds, ensuring quick interactions for users.

- Quality Attribute: Performability
- Sub-Quality Attribute: Time Behavior
- Subject: Condo Management System
- Action: Response
- Constraints of Action: Within 5 seconds

Requirement 5: Financial data, including condo fees and costs, shall be encrypted to ensure privacy, and only authorized users can decrypt the information.

- Quality Attribute: Privacy
- Sub-Quality Attribute: Confidentiality
- Subject: Financial Data
- Action: Encryption
- Constraints of Action: Decryption for authorized users only

Requirement 6: At least 10 users should be able to access the application concurrently, ensuring optimal performance.

- Quality Attribute: Performability
- Sub-Quality Attribute: Capacity
- Subject: Condo Management System
- Action: Access the system

- Constraints of Action: At least 3 Users

2.1.2. Functional Requirements

Requirement 1: Public users must provide a unique registration key from their condo management company to become condo owners or rental users in the system.

- Quality Attribute: Security
- Sub-Quality Attribute: Authenticity
- Subject: Public User
- Action: Register as condo owner or rental user
- Constraints of Action: Valid registration key required

Requirement 2: Condo owners should have access to a comprehensive dashboard displaying property information, including personal profile, condo details, and financial status.

- Quality Attribute: Usability
- Sub-Quality Attribute: Accessibility
- Subject: Condo Owner
- Action: View dashboard
- Constraints of Action: Accessible property details and financial status

Requirement 3: The system shall allow the user to view and edit their profile information.

- Quality Attribute: Usability
- Sub-Quality Attribute: Accessibility
- Subject: Condo Management System
- Action: View & edit profile
- Constraints of Action: Must be logged in

Requirement 4: The system shall allow registered users to log in to the condo management system.

- Quality Attribute: Security
- Sub-Quality Attribute: Authenticity
- Subject: Condo Management System
- Action: Login
- Constraints of Action: Allow valid users

Requirement 5: A user can try to login 5 times before getting blocked.

- Quality Attribute: Security
- Sub-Quality Attribute: Authenticity
- Subject: User
- Action: Login

Requirement 6: The system shall allow valid users to recover/change their password when the user forgets or desires to change it.

- Quality Attribute: Usability
- Sub-Quality Attribute: Operability
- Subject: Condo Management System

- Action: Recover Password
- Constraints of Action: Valid user

Requirement 7: Condo management companies must be able to upload condo files for each property, ensuring easy access for condo owners.

- Quality Attribute: Usability
- Sub-Quality Attribute: Manageability
- Subject: Condo Management System
- Action: Upload condo files
- Constraints of Action: Secure access for authorized users

Requirement 8: The system shall support the entry of detailed information for each condo unit, parking spot, and locker, including owner details and associated fees.

- Quality Attribute: Completeness
- Sub-Quality Attribute: Data Accuracy
- Subject: Condo Management System
- Action: Enter detailed information for condo units, parking spots, and lockers
- Constraints of Action: Accurate and up-to-date data entry

Requirement 9: Condo management companies should be able to generate and distribute registration keys to unit owners or rental users for linking their profiles to specific units.

- Quality Attribute: Security
- Sub-Quality Attribute: Authorization
- Subject: Condo Management System
- Action: Generate and distribute registration keys
- Constraints of Action: Authorized access to key generation and distribution

Requirement 10: The system shall calculate condo fees for each unit based on factors such as square footage and parking spot allocation.

- Quality Attribute: Accuracy
- Sub-Quality Attribute: Efficiency
- Subject: Condo Management System
- Action: Calculate condo fees
- Constraints of Action: Accurate calculation algorithms

Requirement 11: Condo owners and rental users should be able to reserve common facilities using a calendar-like interface.

- Quality Attribute: Usability
- Sub-Quality Attribute: Interactivity
- Subject: Condo Owner/Rental User
- Action: Reserve common facilities
- Constraints of Action: Clear and intuitive reservation process

2.2. User Stories Backlog

The table below represents the legend for the User Stories Backlog throughout sprints 1-5. Section 2.2.13 will specify the user stories that were completed for each sprint.

Table background color	**Color Meaning**
Purple	User story is in the process of being implemented
Blue	User story has been implemented and its implementation is completed
Green	User story has been created during this sprint
Yellow	User story has been completed, but it's description has been edited during this sprint
Red	User story has been removed from the scope of the project and their respective story points have been subtracted from the total

2.2.1. Epic 1: User Account Management

US-01	User Registration		
As a public user, I want to sign up with my first name, last name, username, email, phone number and password so that I can create a unique profile.			
User Account Management			
M	5	L	1

US-02	User Login		
As a public user, I want to login so that I can access personalized information and utilize features exclusive to registered users.			
User Account Management			
M	8	L	3

US-03	User Log Out		
As a public user, I want to be able to log out of my account so that I can prevent unauthorized access to my account once my session is complete.			
User Account Management			
M	8	M	1

US-04	Forgotten Password Reset		
As a public user, I want to be able to request a password reset so that I can regain access to my account if I forget my password.			
User Account Management			
M	8	M	2

US-05	Change Password		
As a public user, I want to be able to change my password whenever I'd like, so that I can enhance the security of my account.			
User Account Management			
M	5	M	2

2.2.2. Epic 2: User Profile

US-06	Make User Profile		
As a public user, I want to have my own user profile, so that I can personalize my experience within the condo management system.			
User Profile			

M	8	M	5

US-07	Edit User Profile
As a public user, I want to be able to modify my user profile details within the condo management system, so that I can keep my information accurate and up-to-date.	
User Profile	

M	5	L	2

US-08	View User Profile
As a public user, I want to view my user profile, so that I can maintain accurate and up-to-date personal details within the condo management system.	
User Profile	

M	5	L	5

2.2.3. Epic 3: Company Account Management

US-09	Create company account
As a Condo Management Company, I want to create an account with company details so that I can establish a presence within the condo management system and manage properties under my care efficiently.	
Company Account Management	

M	5	L	5

US-010	Company Log in
As a Condo Management Company, I want to be able to log into my existing account so that I can manage my existing properties	

Company Account Management			
M	5	L	3

US-011	Company Log Out		
As a Condo Management Company, I want to be able to log out of my account so that I can prevent unauthorized access to my account once my session is complete.			
Company Account Management			
M	8	M	1

US-012	Forgotten Password Reset		
As a Condo Management Company, I want to be able to request a password reset so that I can regain access to my account if I forget my password.			
Company Account Management			
M	8	M	2

US-013	Change Password		
As a Condo Management Company, I want to change my password when I'd like so that I can enhance the security of my account.			
Company Account Management			
M	5	M	2

2.2.4. Epic 4: Registration Management

US-014	Obtain registration key for condo owner

As a public user, I want to have a registration key from the condo company to become an owner in the system so that I can authenticate my ownership and access the owner-specific features.			
Registration Management			
M	3	L	2

US-015	Obtain registration key for rental user
As a public user, I want to have a registration key from the condo company to become a rental user in the system so that I can authenticate my identity and access the tenant-specific features.	
Registration Management	

M	3	L	2

US-016	Send registration key
As a condo management company, I want to be able to send registration keys to unit owners or rental users for their units so that they can link their profiles to their units and access relevant information and features.	
Registration Management	

M	3	L	2

2.2.5. Epic 5: Condo Property Management

US-017	Have a dashboard
As a condo owner, I want to have a dashboard of properties (general information, personal profile, condo information, financial status, remaining balance of condo fee payments, status of submitted request) so that I can easily manage and keep track of my properties.	
Condo and Property Management	

M	8	L	5

US-018	Upload condo file		
As a condo management company, I want to be able to upload a condo file for each property so that all condo owners of that property can access important documents and information.			
Condo and Property Management			
M	3	M	3

US-019	Enter detailed information for condo		
As a condo management company, I want to be able to enter detailed information for each condo unit (condo unit, parking spot, locker) so that I can keep track of all assets, amenities and their details.			
Condo and Property Management			
M	8	L	5

US-020	Edit detailed information for condo		
As a condo management company, I want to edit the contents of the condo, so that the information can be corrected if it is wrong or updated if I have new information to add to it.			
Condo and Property Management			
M	8	M	3

US-021	Add an Entry in Dashboard		
As a condo owner, I want to add an entry of a property in the contents of the dashboard page, so that the information on my dashboard can be updated to add new properties in my ownership.			
Condo and Property Management			
M	8	M	2

US-022	Delete an Entry in Dashboard

As a condo owner, I want to delete an entry made in the contents of the dashboard page, so that I can remove properties that are incorrect or no longer owned by me if ever I've sold them.			
Condo and Property Management			
M	5	M	2

US-047	Company Action Options		
As a condo management company, I want to be able to view all my action options on a dashboard page, so that I can efficiently manage various tasks and responsibilities related to the properties under my management.			
Condo and Property Management			
M	5	M	2

US-048	Company Properties List		
As a condo management company, I want to be able to view a list of my properties so that I can organize the properties under my management			
Condo and Property Management			
M	5	M	2

US-049	Create Property Profile		
As a condo management company, I want to be able to create a profile for a property under my management, which should have at least property name, unit count, parking count, locker count, address, so that I can maintain accurate and organized records of each property's essential details and efficiently manage their resources and operations.			
Condo and Property Management			
M	8	M	3

US-050	Edit detailed information for property

As a condo management company, I want to be able to edit the profile for a property under their management (property name, unit count, parking spot count, locker count, address), so that I can keep my property information accurate and up-to-date.			
Condo and Property Management			
M	8	M	3

US-051	View Property Profile
As a condo management company, I want to be able to view the profile for a property under my management (property name, unit count, parking count, locker count, address, condo list) so that I can have a comprehensive overview of all essential details pertaining to the property.	
Condo and Property Management	

M	8	M	3

US-052	Create Condo Unit
As a condo management system, I want to be able to create a condo for my property under my management, which should have at least the condo id, size, parking spot id, locker id, so that I can accurately categorize and manage each condo unit's assets and amenities within the property.	
Condo and Property Management	

M	8	M	3

US-053	Delete a Property
As a condo management company, I want to delete an entry of a property made in the contents of my properties list, so that I can remove properties that are incorrect or no longer under my ownership.	
Condo and Property Management	

M	5	M	2

US-054	View my Condo

As a condo owner, I want to be able to view the profile for a condo that I own (size, parking spot id, locker id, condo fee) so that I can have a comprehensive overview of all essential details pertaining to the condo.			
Condo and Property Management			
M	5	M	2

US-057	Condo Owner Condo List		
As a condo owner, I want to be able to view a list of my condos so that I can organize the condos under my ownership.			
Condo and Property Management			
M	8	M	3

2.2.6. Epic 6: Requests System

US-023	Submit requests		
As a condo owner, I want to be able to submit requests, so that I can communicate my needs and issues to the management. Note: The types of requests consist of: ● Moving in/out ● Intercom changes ● Request access ● Report a violation ● Report deficiency found in common area ● Question			
Requests System			
M	3	M	5

US-024	See assigned requests
As a condo management employee, I want to be able to see the assigned requests so that I can promptly address and resolve them and stay updated on the status of my assigned requests.	

Requests System			
M	8	L	2

US-025	Notification Page for Request Tracking
As a user, I want to be able to have a notification page so that I can see and stay updated on the latest activities in my submitted or assigned requests.	
Requests System	

M	5	L	2

US-055	Submitted Request Inbox
As a condo owner, I want to have an inbox page for my submitted requests so that I can track my past inquiries, receive updates, and compose new requests with the management team.	
Requests System	

M	8	M	2

2.2.7. Epic 7: Reservation System

US-026	Reserve common facilities
As a condo owner, I want to be able to reserve common facilities in a calendar-like interface so that I can easily plan and schedule my use of these facilities.	
Reservation System	

M	5	H	3

US-027	Set up a common facility
As a condo management company, I want to be able to set up a common facility which requires reservations so that occupants can utilize shared amenities fairly and in an organized manner.	

Reservation System			
M	5	M	5

US-028	See availabilities
As a rental user, I want to be able to see the availability of common facilities so that I can see if any availability aligns with my schedule.	
Reservation System	

M	5	L	1

US-029	Modify Reservation
As a rental user, I want to be able to modify or cancel my existing reservation for a common facility, so that I am no longer committed to a reservation that I cannot go to in the case of changes in my plans or scheduling conflicts.	
Reservation System	

M	5	L	3

US-030	Disable Reservation for Unavailable Facilities
As a condo management company, I want to be able to disable the availability of a common facility, so that occupants cannot reserve the facility if it is already reserved and are aware of its unavailability.	
Reservation System	

S	5	L	5

2.2.8. Epic 8: Financial System

US-031	Enter condo fees
As a condo management company, I want to be able to enter the condo fee per square foot, per parking spot so that accurate calculations can be made and presented to condo owners.	

Condo Fee Management			
M	5	L	1

US-032	Enter Operation Cost		
As a condo management company, I want to be able to enter the cost for each operation so that I can keep track of financial expenses and provide transparency to owners.			
Financial System			
M	8	M	5

US-033	See the calculated fee		
As a condo owner, I want to be able to see the calculated condo fee for each unit so that I am aware of the financial obligation and can plan accordingly.			
Financial System			
M	5	L	1

US-034	Record Operational Budget		
As a condo management company, I want to be able to see a record of the operational budget, which includes the total condo fees collected from condo owners, so that I can manage the financial resources of the company.			
Financial System			
M	8	M	3

US-035	Annual report		
As a condo management company, I want to be able to see an annual report so that I can assess the financial performance and overall status of the property.			
Financial System			
M	5	M	3

2.2.9. Epic 9: User Navigation System

US-036	Navigate pages		
As a public user, I want to easily navigate between different pages so that I can effortlessly explore various features and sections of the condo management app and website.			
User Navigation Experience			
M	5	L	5

2.2.10. Employee Management

US-037	Create Employee Profile		
As a condo management company, I want to create employee profiles, so that I can see and manage new hires within the company's system.			
Employee Management			
M	8	M	5

US-038	Assign roles		
As a condo management company, I want to be able to set up different roles for different employees so that responsibilities can be assigned to the appropriate employee and property management tasks can be efficiently handled.			
Employee Management			
M	8	M	3

US-039	Delete Employee Profile
As a condo management company, I want to delete employee profiles, so that I can remove people that are no longer employees within the company's system.	

Employee Management			
M	8	M	5

US-056	View Employees List		
As a condo management company, I want to access a list of registered employees at the company so that I can view and manage employee profiles.			
Employee Management			
M	5	M	3

2.2.11. Connections (Optional)

US-040	Post on a forum		
As a user, I want to be able to post on a forum so that I can share information or opinions.			
Connections			
C	2	L	5

US-041	Reply to Posts		
As a user, I want to be able to reply to posts on a forum so that I can actively engage with the community.			
Connections			
C	2	L	5

US-042	Organize events with occupants
As a user, I want to be able to organize events so that I can encourage community engagement and social interaction with other occupants.	

Connections			
C	2	L	8

US-043	Invite occupants to events		
As a user, I want to be able to invite occupants to available events so that we can foster a sense of community and engagement among residents.			
Connections			
C	2	L	5

2.2.12. Promotions (Optional)

US-044	Add coupons/offers		
As a condo management company, I want to be able to add coupons and offers for a property so that condo owners and rental users of that property can access exclusive deals and promotions			
Promotions			
C	2	L	5

US-045	Edit coupons/offers		
As a condo management company, I want to be able to edit coupons and offers for a property, so that the information can be corrected if it is wrong or updated if I have new information to add to it.			
Promotions			
C	2	L	3

US-046	Delete coupons/offers
As a condo management company, I want to be able to delete coupons and offers for a property so that I can remove coupons and offers that are incorrect or no longer applicable to the property.	

Promotions			
C	2	L	5

2.2.13. User Stories by Sprint

2.2.13.1. Sprint 1

The following table represents the user stories that were implemented in sprint 1. All user stories set out for Sprint 1 were completed.

User Story ID	User Story Points (USP)	Priority	Status
US-01	5	High	DONE
US-02	5	High	DONE
US-03	3	Medium	DONE
US-07	2	Medium	DONE
US-09	5	High	DONE
US-08	5	High	DONE
US-36	2	High	DONE
US-10	3	High	DONE
US-11	1	Medium	DONE
Total USP	31		

2.2.13.2. Sprint 2

The following table represents the user stories that were implemented in sprint 2. All user stories set out for Sprint 2 were completed.

User Story ID	User Story Points (USP)	Priority	Status
US-04	2	Low	DONE
US-05	2	Low	DONE
US-012	2	Low	DONE
US-013	2	Low	DONE
US-015	2	High	DONE
US-014	2	High	DONE
US-016	3	High	DONE
US-018	3	High	DONE
US-019	2	High	DONE
Total USP	20		

2.2.13.3. Sprint 3

The following table represents the user stories that were implemented in sprint 3. All user stories set out for Sprint 3 except US-039, which was removed due to out of scope, were completed. We also removed US-021 and US-022 for out of scope.

User Story ID	User Story Points (USP)	Priority	Status
US-017	8	High	DONE
US-023	5	Medium	DONE
US-024	3	High	DONE
US-025	3	High	DONE
US-038	5	High	DONE
US-039	4	Low	REMOVED
US-053	5	Medium	DONE
US-054	2	Medium	DONE
Total USP	35		

2.2.13.4. Sprint 4

The following table represents the user stories that were implemented in sprint 4. All user stories set out for Sprint 1 were completed.

User Story ID	User Story Points (USP)	Priority	Status
US-026	3	High	DONE
US-027	5	High	DONE
US-028	2	High	DONE
US-029	3	Medium	DONE
US-030	5	Medium	DONE
US-031	2	High	DONE
US-032	2	High	DONE
US-033	1	Low	DONE
US-034	3	Medium	DONE
US-035	3	Medium	DONE
Total USP	29		

2.2.13.5. Sprint 5

All the user stories for the mandatory requirements have been completed in Sprints 1 through 4. User stories for the optional requirements, namely posts and replies to a common forum, were implemented and completed this sprint.

User Story ID	User Story Points (USP)	Priority	Status

US-040	5	Low	DONE
US-041	5	Low	DONE
Total USP	10		

3. Final Software Architecture

3.1. Introduction

3.1.1. Identifying Information

The Software Architecture Document details the software architecture of the Condo Management System, a web application designed to streamline the management of condominium properties, facilitating tasks such as facility reservations, financial management, resident registration, and communication amongst users and between users and management. The condo management system will follow the model-view-controller (MVC) architecture. This architecture pattern organizes the system's logic into three distinct layers, each carrying specific tasks. The model layer represents our database and how our data will be stored. The view layer is the UI of the application; either our webpage or the view of the mobile application. The controller layer is our backend, handling requests and triggered events from the view and modifying the model's data. Our controller layer will be separated into microservices for increased separation of concerns. Microservices are small loosely coupled independent services that communicate with APIs.

3.1.2. Supplementary Information

Date of Issue and Status: Wednesday, May 1st Draft (This is a document that will be worked on continuously until the end of sprint 5)

Authors, Reviewers, Approving Authority, Issuing Organization: The authors, reviewers, and approving authority consists of the members of team 6. Every member has responsibilities as an author, reviewer, and approving authority.

Change History: The first issue of this document contains the following sections, which were:

1. Introduction
2. Stakeholders and Concerns
3. Viewpoints
4. Views
5. Consistency and Correspondence

Within the 5 sections, the following requirements were taken by the members of team 6:

Identifying Information: Vanessa & Jessey & Michaël

Supplementary Information: Vanessa & Jessey & Michaël

Architecture Evaluations and Rationale for Key Decisions: Jessey & Michaël

Stakeholders: Jessey & Michaël

Concerns: Jessey & Michaël

Concern-Stakeholder Traceability: Jessey & Michaël

Logical View Description: Jessey & Michaël

Physical View Description: Jessey & Michaël

Process View Description: Jessey & Michaël

Use Case View Description: Jessey & Michaël

Domain Model: Vanessa & Ahmad

Class Diagram: Vanessa

Activity Diagram: Vanessa

Sequence Diagram: Mahan & Jessey

Use Case Diagram: Jessey & Michaël & Khoi & Vanessa

Component Diagram: Jessey & Michaël & Khoi

Deployment Diagram: Jessey & Michaël & Khoi

Version Control: Version number v.1.5 (This is the final version for the SAD documentation, composing of the parts for sprint 1 through sprint 5)

References:

- Requirements and User Story Backlog Sprints 1 through 4

3.1.3. Other Information

3.1.3.1. Overview (optional)

3.1.3.2. Architecture Evaluations

3.1.3.3. Rationale for key decisions

The model-view-controller (MVC) architecture is the most appropriate architecture design for our system. Since we must develop the system on a website and a mobile app version, we can easily swap the view component. The controller and the model components will remain the same for both the website and the mobile app, only the view component will differ between the two.

We chose this model over the layered architecture model because the latter is too complex for the project's scope. Moreover, since the layers are dependent from one to an adjacent one, a change in a single layer can affect the entire system because it operates as a single unit. Therefore, scalability and maintenance is difficult.

The Condo Management System's controller layer will be separated into independent microservices. Separating these concerns allows for improved scalability, better fault isolation and increased transfer speed. Moreover, microservices allow for better data security, because each service is responsible for a specific task, thus it makes it easier to implement security.

3.2. Stakeholders and concerns

3.2.1. Stakeholders

3.2.1.1. User

1. Concerns common to all users

Concern	Quality Attribute
Will my private information be leaked?	Security
Will my email be spammed with junk mail?	Security
Does the system protect from all unauthorized access?	Security
Is it easy to operate the system?	Usability

2. Public User: A new user who isn't registered and doesn't rent any condo unit.

Concern	Quality Attribute
Is this the right product for me?	Relevance

3. Tenant: A registered user who is renting a condo unit.

Concern	Quality Attribute
Is my information correct?	Correctness
Does the system give a quick response?	Usability

4. Condo Owner: A registered user who owns a condo.

Concern	Quality Attribute
Is my information correct?	Correctness
Does the system give a quick response?	Usability

5. Property Owner: A registered user who owns properties.

Concern	Quality Attribute
Is the system available all the time?	Availability
Will my condo and property lists load quickly?	Accessibility
Will my dashboard be often updated?	Accessibility

6. Condo Management Staff: A registered condo management staff

Concern	Quality Attribute
Is the system available all the time?	Availability
Does the system give a quick response?	Usability

3.2.1.2. Product Owner

1. Project Manager: Software project manager who plans, organizes and directs the project

Concern	Quality Attribute
Will we have enough time to finish?	Timeliness
Will we follow the budget?	Affordability
Have we identified all risks?	Risk identifiability
Will my team stay organized and motivated?	Team Motivation and Accountability
Which development process will we use?	Process Adaptability

2. Software Architect: Engineer who designs and ensures system requirements are met

Concern	Quality Attribute
Will our architecture be able to handle many users?	Scalability
Is our architecture secure?	Security

Will our architecture be easy to maintain?	Maintainability

3. Developer: Follow the development process to build the system

Concern	Quality Attribute
Will we collaborate well as a team?	Collaboration
Are the requirements precise and realistic?	Requirement Accuracy
Will the requirements change a lot during development?	Adaptability

4. Tester: Test the system through various checks

Concern	Quality Attribute
Will we collaborate well as a team?	Collaboration
Is testing planned from the beginning?	Testability
Will we automate tests?	Testability
Can we accept some small bugs?	Fault Tolerance

3.2.2. Concerns

#1 Concern: How feasible is it to construct and deploy the Condo Management System?

#2 Concern: What are the potential risks and impacts of the Condo Management System to its stakeholders throughout its life cycle?

#3 Concern: How is the Condo Management System to be maintained and evolved?

#4 Concern: How will the database be organized? How will the tables be related?

#5 Concern: How does the system deal with security (authentication)?

#6 Concern: How will the system perform under extreme conditions/system congestion (e.g. many requests are sent at the same time)?

#7 Concern: Will our controller layer adapt easily to multiple views?

3.2.3. Concern-Stakeholder Traceability

This association can be recorded via a simple table or other depiction.

Table 2.1: Association of stakeholders to concerns in an AD

	Concern 1	Concern 2	Concern 3	Concern 4	Concern 5	Concern 6	Concern 7
Public user	-	-	-	-	X	X	-
Tenant	-	X	X	-	X	X	-
Condo owner	-	X	X	-	X	X	-
Property owner	-	X	X	-	X	X	-
Condo management staff	-	X	X	-	X	X	-
Project manager	X	X	X	X	X	X	X
Software architect	X	X	X	X	X	X	X
Developer	X	X	X	X	X	X	X
Tester	X	X	-	-	X	X	X

3.3. Viewpoints

3.3.1. Logical View

Also known as the structural view.

3.3.1.1. Overview

This view provides a structured representation of the key concepts, entities and relationships within a problem domain. The logical view concerns the system's overall functionality provided to the end-users. It focuses on the static structure of the system, therefore only dealing with essential components such as classes, objects, attributes and relationships. Some examples of this view are the domain model and the class diagram. The class diagram visually describes the dependencies and how different components of the system interact with each other through classes and associations. The logical view provides a perception of the overall design and organizational structure of the system. [1]

3.3.1.2. Concerns and stakeholders

3.3.1.2.1. Concerns

#4 Concern: How will the database be organized? How will the tables be related?

The domain model, which falls under the logical view, loosely represents how the data will be organized in the database. Some classes are equivalent to tables, such as the User class where its attributes are going to be mapped to columns in the database.

3.3.1.2.2. Typical Stakeholders

- Project manager
- Software architect
- Developer

This view sets up the layout of the system, which concerns people directly working on the construction of the software, except testers. Testers aren't included because they check that everything is working fine but the minute details of the system's structure aren't relevant to them. General users, such as tenants and staff, aren't included because they only care about the end product.

3.3.1.2.3. "Anti-concerns" (optional)

3.3.1.3. Model kinds

3.3.1.3.1. Domain model

The domain model is a visual representation of meaningful conceptual classes from the real-world objects in the domain of interest. The domain model illustrates the domain objects, the associations between the objects and their attributes. At the level of the domain model, it is still not possible to develop or write code based on this representation. [2] [3]

3.3.1.3.1.1. Domain model conventions

I) Model kind languages or notations (optional)
II) Model kind metamodel (optional)
III) Model kind templates (optional)

3.3.1.3.1.2. Domain model operations (optional)

3.3.1.3.1.3. Domain model correspondence rules

3.3.1.4. Class Diagram

As explained in the introduction, our system isn't object-oriented, so we will do a sequence diagram instead of a class diagram. However, here's the description of a class diagram.

A class diagram is similar to the domain model, but the class diagram also shows the attributes, operations and relationships among objects from the system's conceptual classes. In other words, this UML (Unified Modeling Language) diagram is a type of static structure diagram which describes the structure of a system. It is the most important part of designing for an object-oriented system because the class diagram illustrates the classes like objects to be programmed. Therefore, translating the model into executable code is not too complex to do. [4] [5]

3.3.1.4.1.1. Class Diagram conventions

I) Model kind languages or notations (optional)
II) Model kind metamodel (optional)
III) Model kind templates (optional)

3.3.1.4.1.2. Class Diagram operations (optional)

3.3.1.4.1.3. Class Diagram correspondence rules

3.3.2. Process View

Also known as the behaviour view.

3.3.2.1. Overview

Contrary to the logical view which focuses on a static view, the process view focuses on the dynamic view of the system. This means that it details the system's behaviour by dissecting the flow of control between components during a process. This is a detailed breakdown of the steps needed to accomplish a use case inside the system. Any sequential flow of activities, interactions and dependencies are taken into account in this view. In this view, stakeholders gain a better insight into resource utilization, task sequencing and latent bottleneck, resulting in potential efficiency and optimized performance.An example of this view are activity diagrams. [1]

3.3.2.2. Concerns and stakeholders

3.3.2.2.1. Concerns

#1 Concern: How feasible is it to construct and deploy the Condo Management System?

Activity diagrams illustrate system behaviour and how different components collaborate. They are built by system designers, and they can be used to demonstrate the feasibility of the Condo Management System.

3.3.2.2.2. Typical Stakeholders

- Project manager
- Software architect
- Developer
- Tester

This view describes the behaviours of the system, which concerns people directly working on the development of the software. The project manager and software architect design the various processes, and developers and testers develop them and ensure full functionality in all possible scenarios. General users aren't included because they only care about the end product.

3.3.2.2.3. "Anti-concerns" (optional)

3.3.2.3. Model kinds

3.3.2.3.1. Activity diagram

In our condominium management system, we have adopted a microservices architecture, wherein each backend system corresponds with a distinct process. This design philosophy allows for granular control and scalability, ensuring efficient handling of various user interactions. Through meticulous activity diagram representations, we elucidate the intricacies of four pivotal user interactions: user registration, user login, reservation creation, and annual report generation.

3.3.2.3.1.1. Activity Diagram conventions

I) Model kind languages or notations (optional)

II) Model kind metamodel (optional)

III) Model kind templates (optional)

3.3.2.3.1.2. Activity Diagram operations (optional)

3.3.2.3.1.3. Activity Diagram correspondence rules

3.3.2.3.2. Sequence diagram

A sequence diagram is an interaction diagram that illustrates how the operations unfold. These diagrams depict the interaction between objects in collaboration. Sequence diagrams are arranged in a time sequence, visually portraying the order of interactions. [13]

3.3.2.3.2.1. Sequence Diagram conventions

I) Model kind languages or notations (optional)

II) Model kind metamodel (optional)

III) Model kind templates (optional)

3.3.2.3.2.2. Sequence Diagram operations (optional)

3.3.2.3.2.3. Sequence Diagram correspondence rules

3.3.3. Physical View

Also known as the deployment view.

3.3.3.1. Overview

The physical view is related to the hardware infrastructure needed for our system. For example, since we will use a server to host our website and database, the physical view will illustrate connections between each layer, which are nodes for this view. Furthermore, this view tackles scalability and resource allocation, since they depend on the server specifications, availability and plans for its future. An example of the physical view is the deployment diagram. [1]

3.3.3.2. Concerns and stakeholders

3.3.3.2.1. Concerns

#1 Concern: How feasible is it to construct and deploy the Condo Management System?

Deployment diagrams illustrate where and how the components are used in the system's deployment. They are built by system designers, and they demonstrate the physical infrastructure needed to deploy the system.

#3 Concern: How is the Condo Management System to be maintained and evolved?

Once again, deployment diagrams illustrate the detailed configuration of the run time system from the components. Therefore, this view exemplifies how the components interact through the APIs and servers, thus making it possible to plan the supervision, maintenance and evolution of the Condo Management System.

#6 Concern: How will the system perform under extreme conditions/system congestion (e.g. many requests are sent at the same time)?

The physical view contains specifications about servers and how they are connected together. To answer questions related to high traffic and resource allocation, developers use the physical view, since it's the only view containing the concrete information needed to answer those questions.

3.3.3.2.2. Typical Stakeholders

- Public user
- Tenant
- Condo owner
- Property owner
- Condo management staff
- Project manager
- Software architect
- Developer
- Tester

This view shows the physical infrastructure of the system, which concerns all 9 stakeholders related to this project. The development team must know what they are working with and how their components connect. General users are concerned with the system's performance and maintainability for the future.

3.3.3.2.3. "Anti-concerns" (optional)

3.3.3.3. Model kinds

3.3.3.3.1. Deployment diagram

Deployment diagrams portray the physical architecture of a system, like how nodes in a distributed system are arranged. They reveal how software and hardware elements interact, showcasing the physical distribution of processing. They also display the artifacts stored on each node and the components they implement. Nodes represent hardware devices; some examples are computers, sensors, printers and any other device that supports the system's runtime environment. Communication paths and deployment relationships represent the connections in the system. [8]

3.3.3.3.1.1. Deployment Diagram conventions

I) Model kind languages or notations (optional)

II) Model kind metamodel (optional)

III) Model kind templates (optional)

3.3.3.3.1.2. Deployment Diagram operations (optional)

3.3.3.3.1.3. Deployment Diagram correspondence rules

3.3.4. Development View

Also known as the implementation view.

3.3.4.1. Overview

The development view shows how each component is interconnected when developing our system. It will demonstrate our environments and development classes, not present in the logical view. In addition, it will specify the interfaces connecting the different modules to ensure that the whole development team is on the same page. An example of the development view is the component diagram. [1]

3.3.4.2. Concerns and stakeholders

3.3.4.2.1. Concerns

#5 Concern: How does the system deal with security (authentication)?

The development view identifies all the software components in the system and their distribution. It identifies layers, what goes in each one, and how the layers are intertwined. This is crucial to plan precisely and in advance since architectures with components separated into independent layers or modules have increased security. For example, in our MVC architecture, the view doesn't have access to the model (database), it has to go through the controller and its user authentication.

#7 Concern: Will our controller layer adapt easily to multiple views?

To address this concern, it is key to identify all links between the controller and the view. These are represented best in the development view.

3.3.4.2.2. Typical Stakeholders

- Public user
- Tenant
- Condo owner
- Property owner
- Condo management staff
- Project manager
- Software architect
- Developer
- Tester

This view is crucial for the development team, as it shows what they have to do during implementation rather than a more general design. It also lays out how to achieve the needed security, which concerns the general public and all users of the system.

3.3.4.2.3. "Anti-concerns" (optional)

3.3.4.3. Model kinds

3.3.4.3.1. Component Diagram

A component diagram is a UML diagram which outlines how the physical components in a system are organized and interconnected. These diagrams are created to model implementation details and to ensure that the planned development adequately covers all the requirements of the system. Some examples of components that can be found in a component diagram are documents, database table, files, executables and any physical elements with a location in a storage. Moreover, these diagrams are almost identical to class diagrams but they focus on a system's components, providing a static implementation view of the system. [9] [10]

3.3.4.3.1.1. Component Diagram conventions

I) Model kind languages or notations (optional)

II) Model kind metamodel (optional)

III) Model kind templates (optional)

3.3.4.3.1.2. Component Diagram operations (optional)

3.3.4.3.1.3. Component Diagram correspondence rules

3.3.5. Use Case View

Also known as the scenario view.

3.3.5.1. Overview

The use case view is the only view focusing on an external view of the system. It elaborates on all the actions a user can perform from their perspective. It models their interactions and how the system responds under all scenarios; this process allows us to verify the requirements and to identify possible shortcomings because of its different perspective from the other views. An example of the use case view is the use case diagram. [1]

3.3.5.2. Concerns and stakeholders

3.3.5.2.1. Concerns

#2 Concern: What are the potential risks and impacts of the Condo Management System to its stakeholders throughout its life cycle?

To view our system from the stakeholder's perspective, we utilize the use case view. Since it explains interactions between users and the system, it highlights the validity of the requirements and any risks related. As explained above, use cases cover how the system should respond under any scenario, so they can be used to identify limitations.

3.3.5.2.2. Typical Stakeholders

- Tenant
- Condo owner
- Property owner
- Condo management staff
- Project manager
- Software architect
- Developer
- Tester

This view describes the behaviours of the system and how to interact with it. This concerns the development team since they have to consider every possible action that users can take to ensure full functionality in all scenarios. General users are concerned because they want the least amount of issues and risks. This view is also ideal for all stakeholders because it's the easiest to understand due to its high-level nature.

3.3.5.2.3. "Anti-concerns" (optional)

3.3.5.3. Model kinds

3.3.5.3.1. Use Case Diagram

A use case diagram falls under the category of dynamic or behaviour diagrams in UML. These diagrams capture a high-level overview of functionality, scope and requirements of a system through interactions between the system and its actors. Use cases encompass a series of actions, services, and functions that the system must execute. The actors are individuals or entities fulfilling defined roles within the system. While use case diagrams describe what the system does and how actors engage with it, they do not delve into the internal workings of the system. [11] [12]

3.3.5.3.1.1. Use Case Diagram conventions

I) Model kind languages or notations (optional)

II) Model kind metamodel (optional)

III) Model kind templates (optional)

3.3.5.3.1.2. Use Case Diagram operations (optional)

3.3.5.3.1.3. Use Case Diagram correspondence rules

3.3.6. Operations on views

3.3.7. Correspondence rules

3.3.8. Examples (optional)

3.3.9. Notes (optional)

3.3.10. Sources

[1] vpadmin, "4 + 1 Views in Modeling System Architecture with UML," *Visual Paradigm Guides*, Sep. 12, 2023. https://guides.visual-paradigm.com/4-1-views-in-modeling-system-architecture-with-uml/ (accessed Feb. 01, 2024).
[2] "Domain model," *Wikipedia*, Jan. 05, 2024. https://en.wikipedia.org/wiki/Domain_model (accessed Feb. 01, 2024).
[3] O. Chursin, "A Brief Introduction to Domain Modeling," *Medium*, Dec. 27, 2017. https://olegchursin.medium.com/a-brief-introduction-to-domain-modeling-862a30b38353 (accessed Feb. 01, 2024).
[4] "Class diagram," *Wikipedia*, Jul. 28, 2022. https://en.wikipedia.org/wiki/Class_diagram (accessed Feb. 01, 2024).
[5] Visual Paradigm, "What is Class Diagram?," *Visual-paradigm.com*, 2019. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/ (accessed Feb. 01, 2024).
[6] "Activity diagram," *Wikipedia*, May 09, 2020. https://en.wikipedia.org/wiki/Activity_diagram (accessed Feb. 01, 2024).
[7] Visual Paradigm, "What is Activity Diagram?," *Visual-paradigm.com*, 2019. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/ (accessed Feb. 01, 2024).
[8] "Deployment diagrams," *www.ibm.com*, Sep. 25, 2020. https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-deployment (accessed Feb. 01, 2024).
[9] "What is Component Diagram?," *Visual-paradigm.com*, 2019. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/ (accessed Feb. 01, 2024).
[10] "Component Diagrams - See Examples, Learn What They Are," *www.smartdraw.com*. https://www.smartdraw.com/component-diagram/ (accessed Feb. 01, 2024).
[11] smartdraw, "Use Case Diagrams - Use Case Diagrams Online, Examples, and Tools," *Smartdraw.com*, 2019. https://www.smartdraw.com/use-case-diagram/ (accessed Feb. 01, 2024).
[12] IBM, "Use-case diagrams," *www.ibm.com*, Mar. 04, 2021. https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case (accessed Feb. 01, 2024).
[13] Visual Paradigm, "What is Sequence Diagram?," *Visual-paradigm.com*, 2019. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/ (accessed Feb. 01, 2024).

3.4. Views

3.4.1. View: Logical

3.4.1.1. Models

3.4.1.1.1. Domain Model

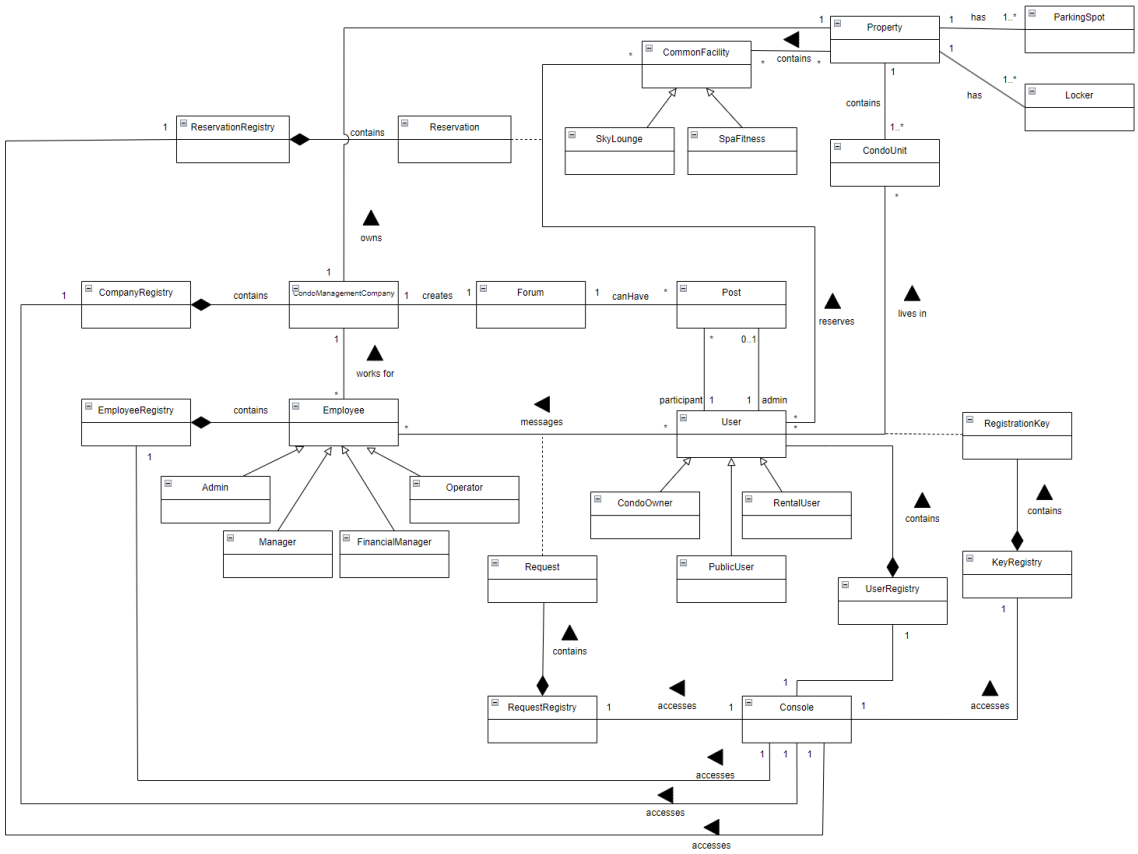


Figure 1: Domain model of the Condo Management System

The domain model organizes the user types, employee types and common facility types. Moreover, the console acts as the controller and is associated with 6 registries: key registry, employee registry, user registry, reservation registry, company registry, and request registry. The relationships between property, company, condo, users, and many more are modeled in this diagram.

3.4.1.1.2. Class Diagram

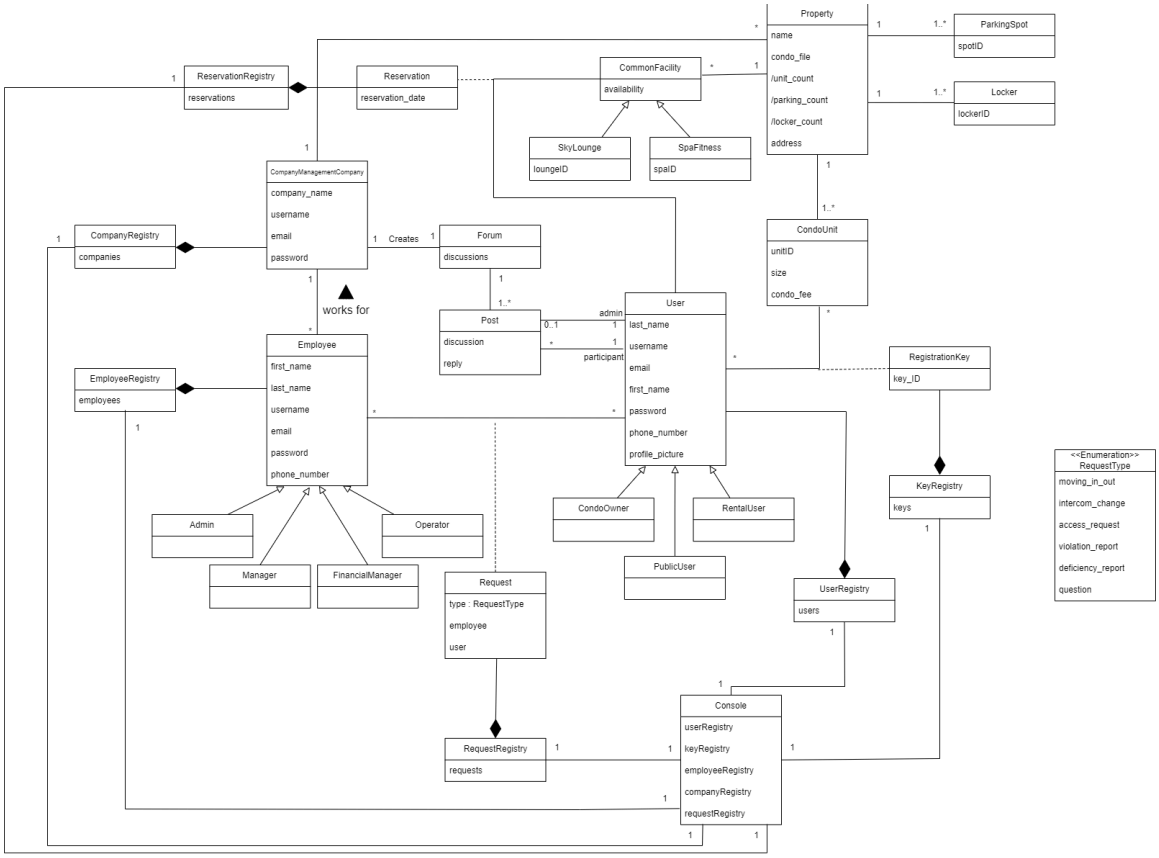


Figure 2: Class Diagram of the Condo Management System

The class diagram encapsulates the structure of the Condo Management system, represented by classes. It builds off the domain model, providing the static organization of user, employee, and common facility types as well as the relationships among all the entities within the system. Additionally, their attributes are listed in this diagram.

3.4.1.2. Known Issues with View

It's difficult to represent data flow in this view because it's a static view of the system and there is little definition of attribute dependencies between classes. Also, naming inconsistencies create confusion.

3.4.2. View: Process

3.4.2.1. Models

3.4.2.1.1. Activity Diagram

Each activity diagram addressed a group of related use cases or a simplified system within the Condo Management System. This serves to provide a clear visual representation of the steps involved in fulfilling a specific requirement. Each diagram is divided into swimlanes to represent the different actors/entities involved in the process, and the series of actions and interactions within that particular entity.

Note that company registration and login follow the same flow of activities as for the public users, as illustrated in Figure 3 and Figure 4 below.

3.4.2.1.1.1. Public User Registration

This diagram depicts the process of actions which is taken by a public user when they would like to register for a new account.

Precondition: User is not logged in.

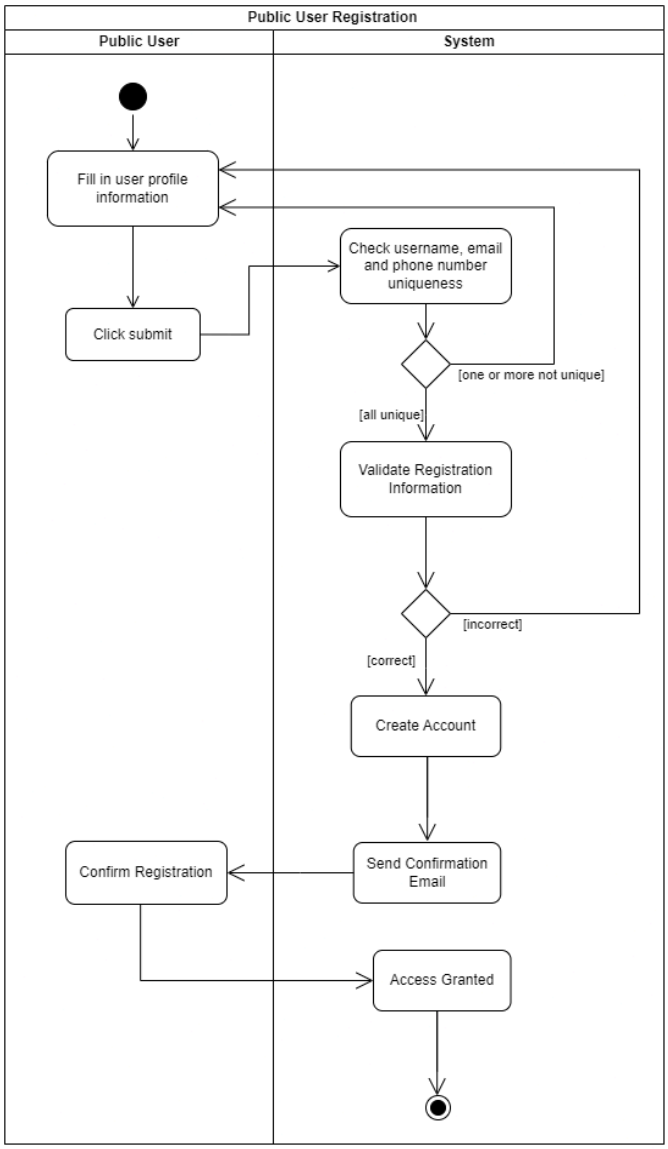


Figure 3: Activity Diagram for Public User Registration

3.4.2.1.1.2. Public User Login

This diagram depicts the process of actions which is taken by a public user when they would like to log in to their account.

Precondition: User is not logged in.

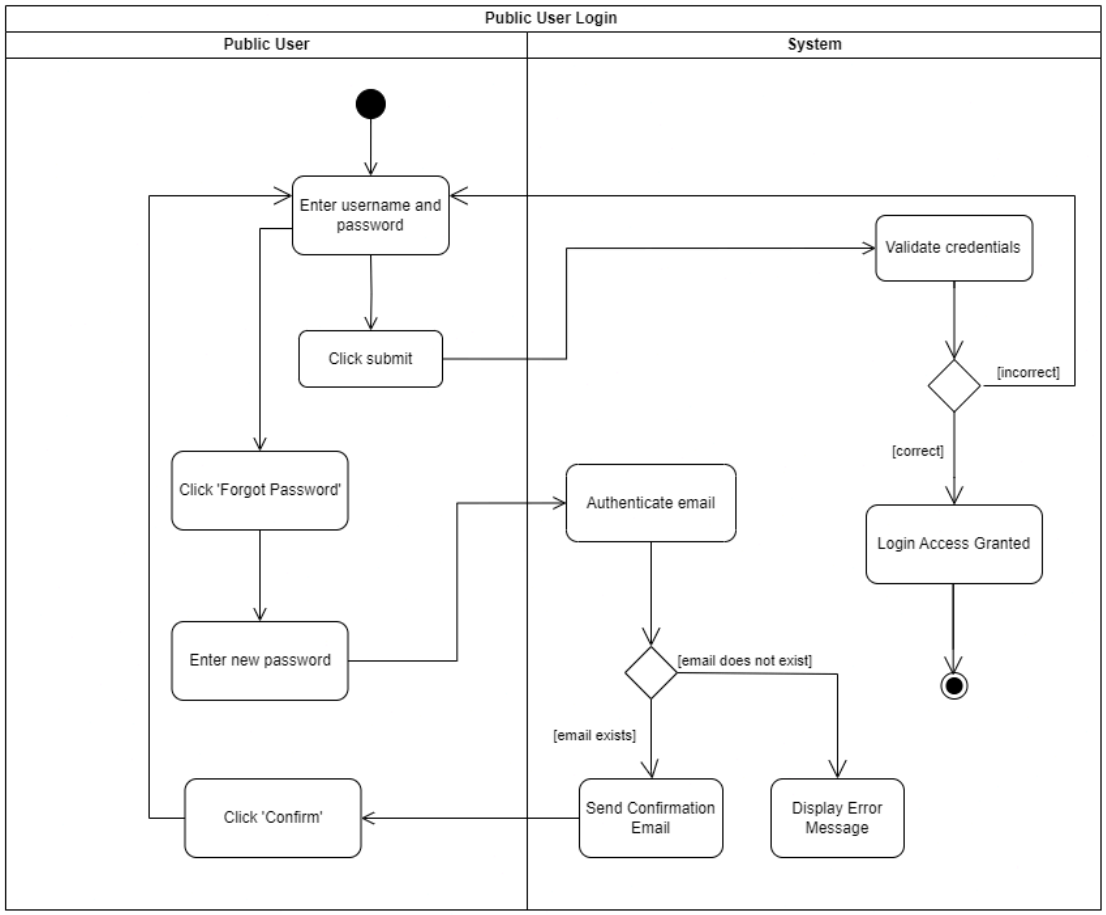


Figure 4: Activity Diagram for Public User Login

3.4.2.1.1.3. Simplified Registration System

This diagram depicts the process of actions which is taken by a registered user (rental user or condo owner) when they would like to make a reservation for one of the common facilities of the property. The user can view and manage the common facilities on a calendar-like interface. The user is assumed to be logged in.

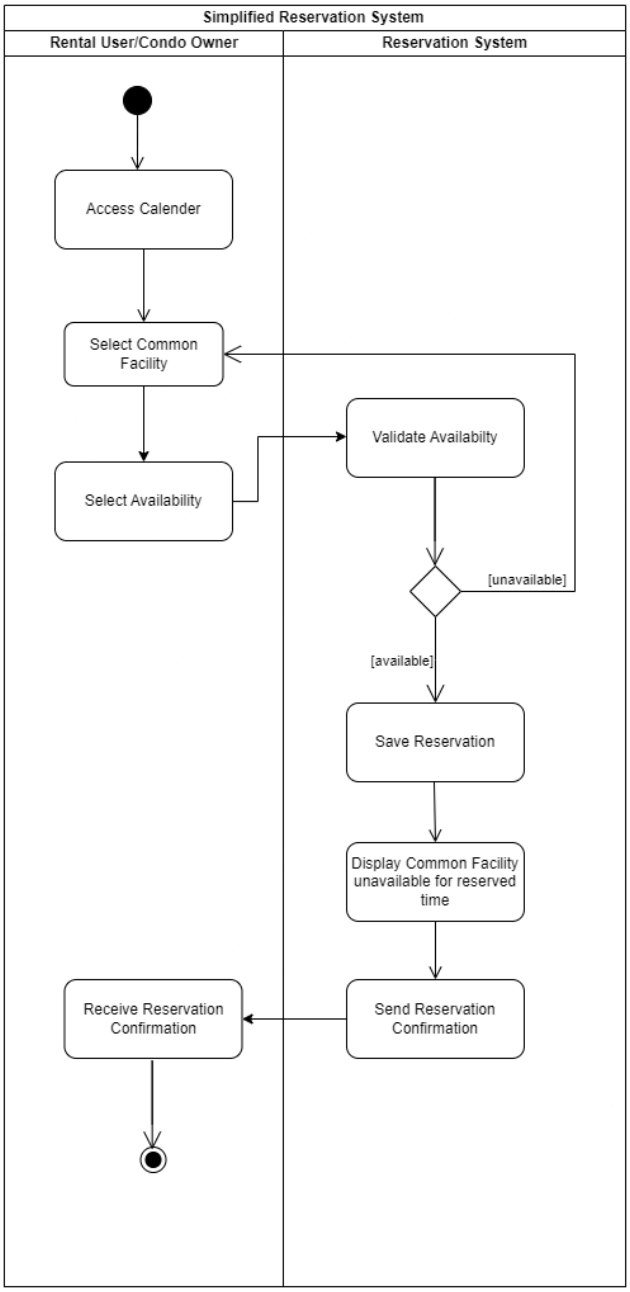


Figure 5: Activity Diagram for Simplified Reservation System

3.4.2.1.1.4. Simplified Financial System

This diagram depicts the process of actions which is taken by the company admin using the financial system within the Condo Management System. A user can enter a condo fee per square foot and parking spot, and enter operational cost. The financial system will record the operational budget, and generate an annual report. The condo fee of each unit will be calculated and presented to the unit owner.

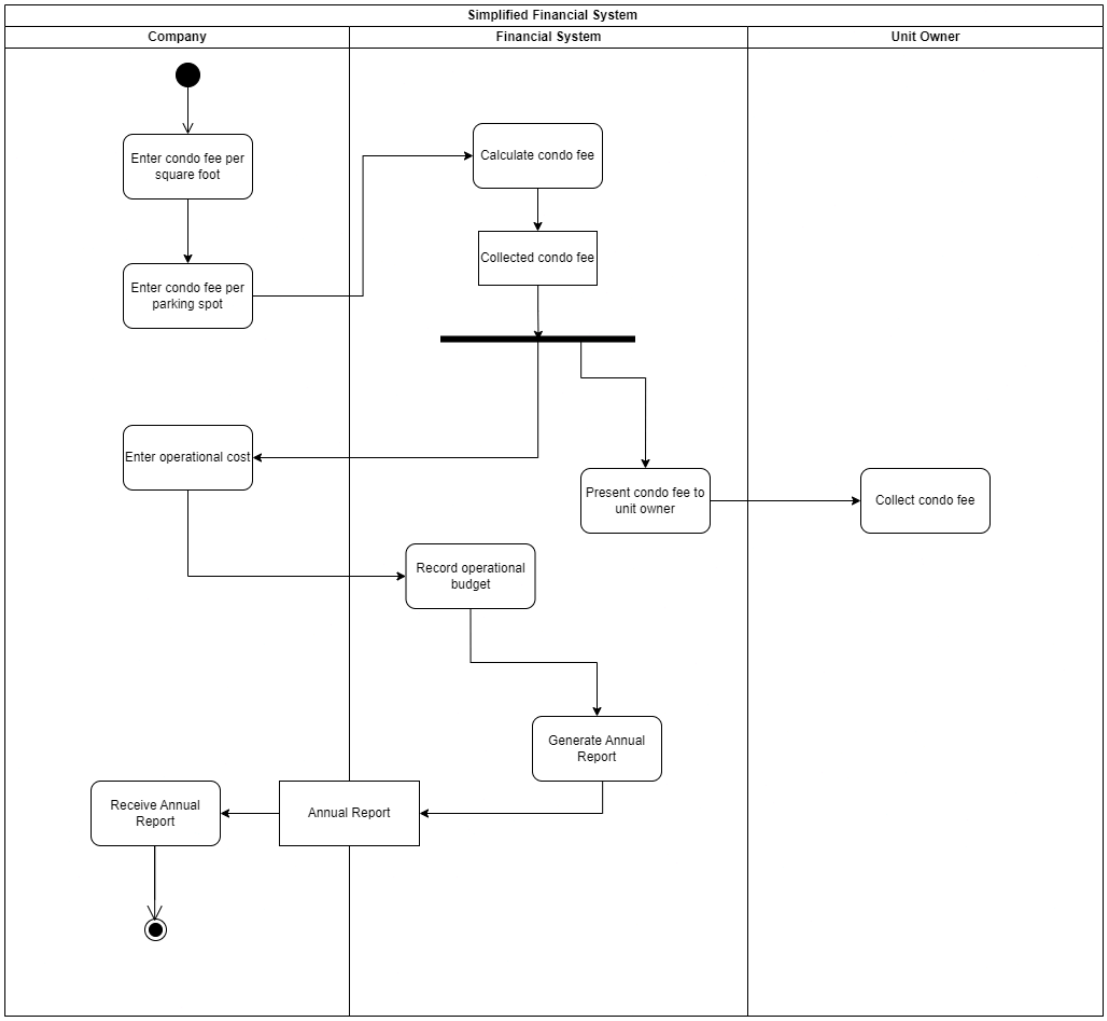


Figure 6: Activity Diagram for Simplified Financial System

3.4.2.1.1.5. Company Employee Management

This diagram depicts the process of actions which is taken by the company admin to create employee profiles and assign their roles in the system. The company admin can view and manage the employee profiles.

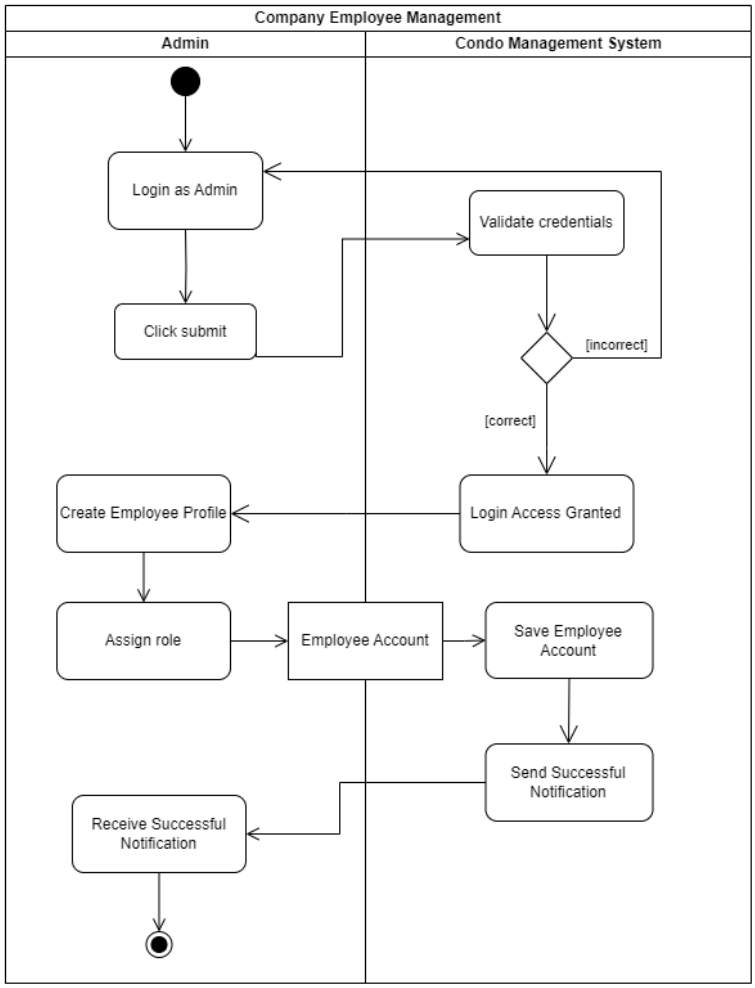


Figure 7: Activity Diagram for Company Employee Management

3.4.2.1.1.6. Request System

This diagram depicts the process of actions which is taken by the condo owner and company employee within the simplified request system. The condo owner can submit requests to communicate with the Condo Management Company. The appropriate employee type is assigned a request based on the type of request and their corresponding role. Once the request is replied to and its status is updated, the condo owner receives a notification on their page on their request's updated status.

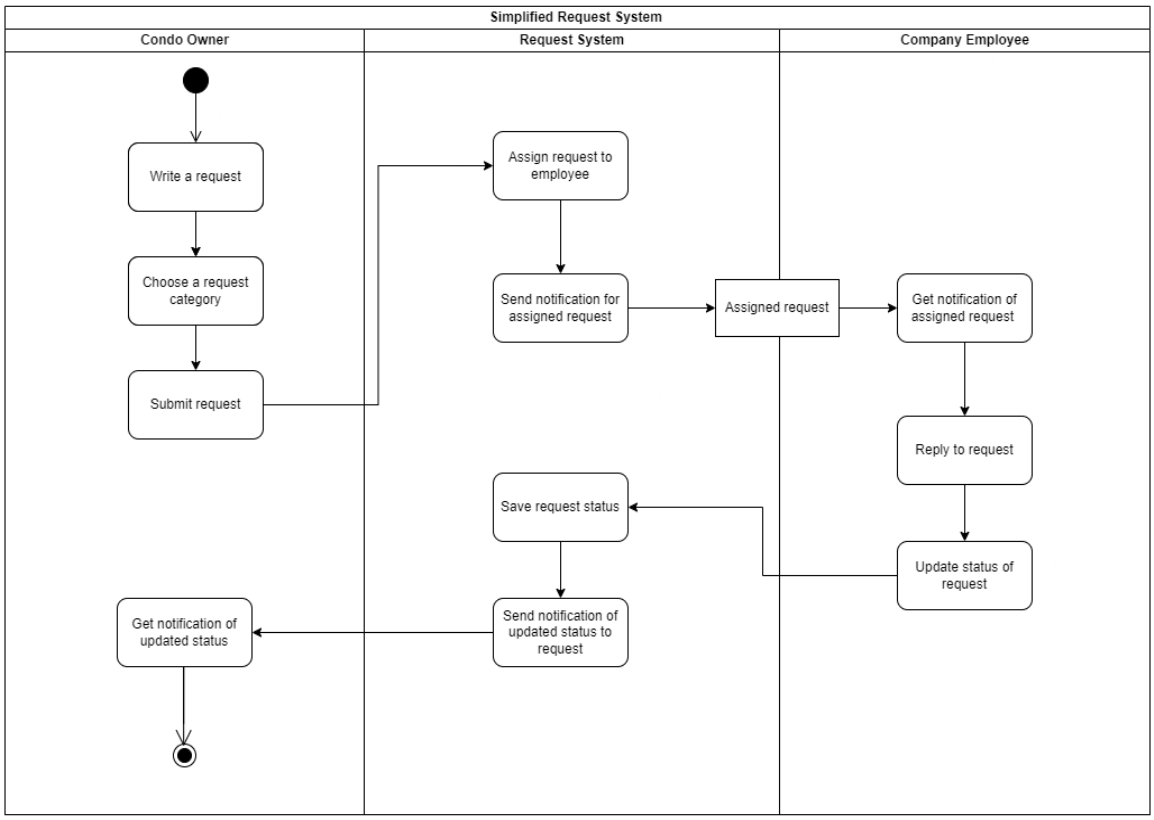


Figure 8: Activity Diagram for the Simplified Request System

3.4.2.1.2. Sequence Diagram

3.4.2.1.2.1. Update User Profile

This diagram sequentially represents the calls to modify the user's info on the profile page. The calls are made from the User to the UserController and Database.

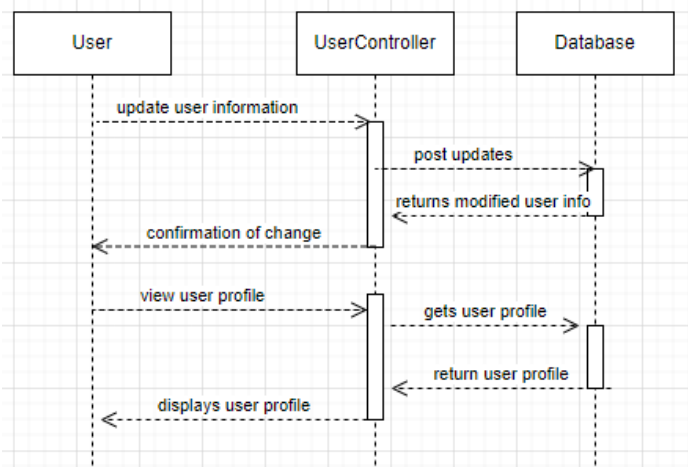


Figure 9: Updating existing user profile

• Step 1: The client sends a request to UserController to modify the profile.

• Step 2: ProfileController updates user details in the database. If successful, the database returns the modified data to the ProfileController

• Step 3: ProfileController sends confirmation of update success back to the user.

3.4.2.1.2.2. Simplified Reservation System

This diagram sequentially illustrates the calls to reserve a service or an amenity based on the available slots. The calls are made from the User to the AppointmentController and Database.

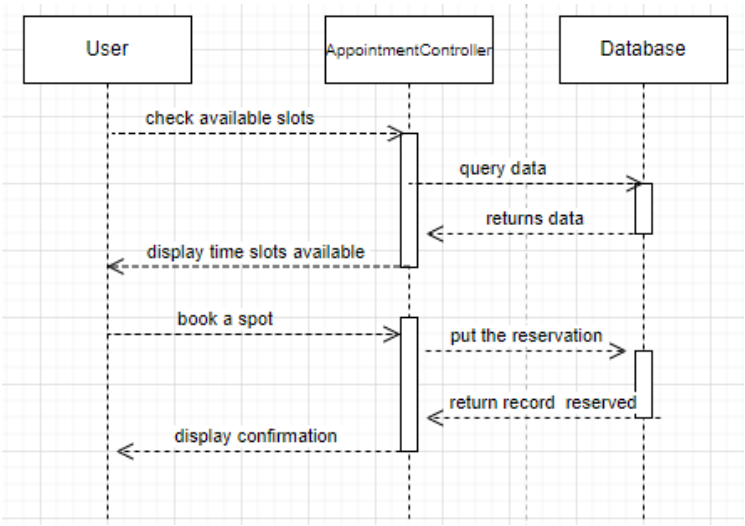


Figure 10: Simplified reservation system

Preconditions: User must be logged in to his/her profile.

• Step 1: The user sends a request to AppointmentController to check for available slots for the service for that given week(ex: spa, fitness lounge)

• Step 2: AppointmentController sends a query to the database and fetches all the available time slots for that service for the selected week. After retrieving the available time slots, the user will be shown the different time slots available.

• Step 3: The user will then select one of the timeslots displayed, the appointment controller will then post an update to the database to set the time slot as taken with the user ID associated with the timeslot.

• Step 4: The appointment controller waits for the database return and sends a confirmation to the client of the booking time and service chosen.

3.4.2.1.2.3. Simplified Financial System

This diagram sequentially depicts the calls to calculate the predicted revenue of a property to generate an annual report. The calls are made from the CompanyAdmin to the FinanceController and to the Database.

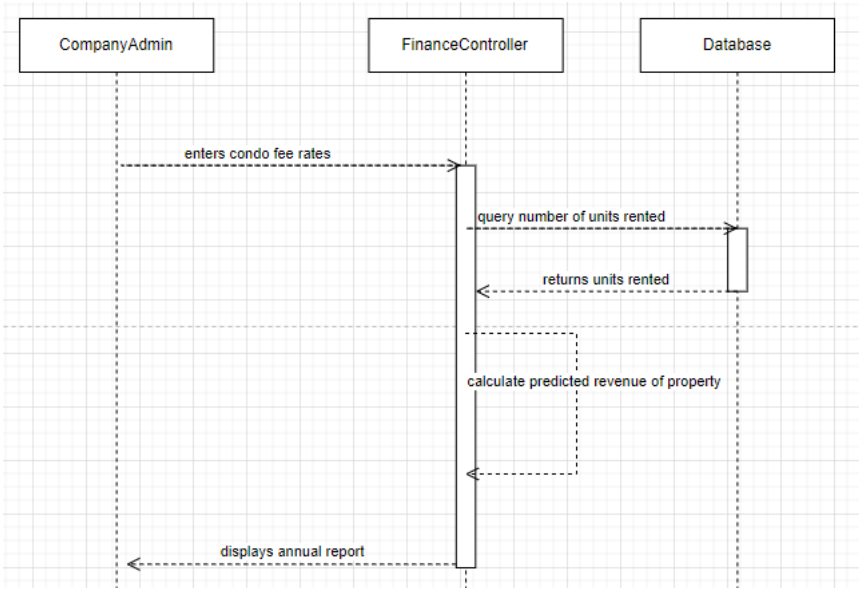


Figure 11: Simplified financial system

Preconditions: Company admin must be logged in to his/her profile.

• Step 1: The user with authorized access enters the rate of their condo fee based on the square footage of the property, with extra charges for amenities(ex: parking spot)

• Step 2: FinanceController sends a query to the database and fetches all units and amenities rented at that time along with their square footage.

• Step 3: The FinanceController will then calculate the predicted revenue based on the rate entered by the admin and the state of the rentals at that moment. The FinanceController then displays the estimated revenue to the company admin.

3.4.2.2. Known Issues with View

Some known issues that may arise in this view in the context of the Condo Management System, include incomplete activities, ambiguous flow and overly complex diagrams. These diagrams may lack items from the stakeholder requirements, leading to inaccuracies in depicting the system's behaviour.

3.4.3. View: Physical

3.4.3.1. Models

3.4.3.1.1. Deployment Diagram

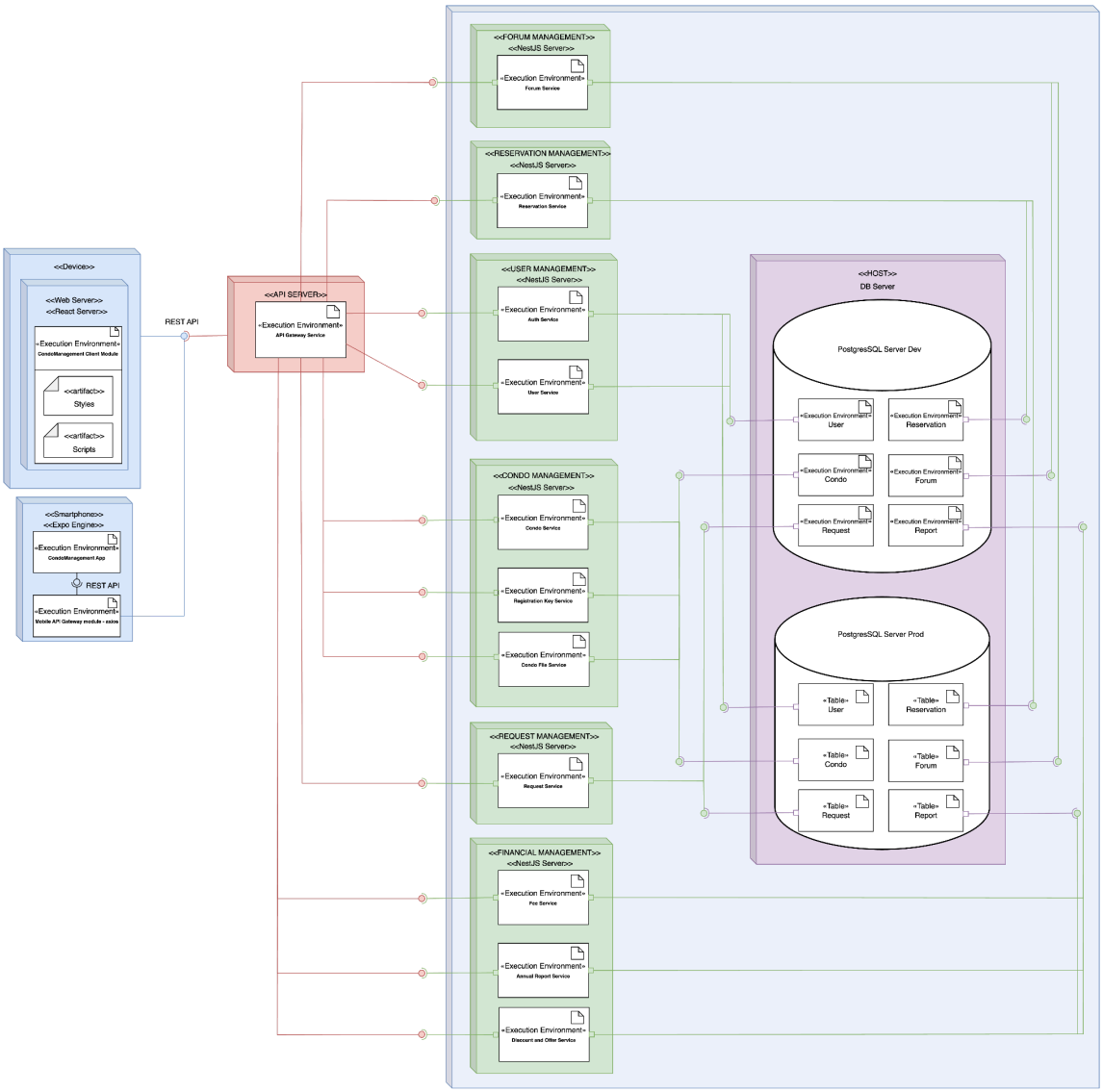
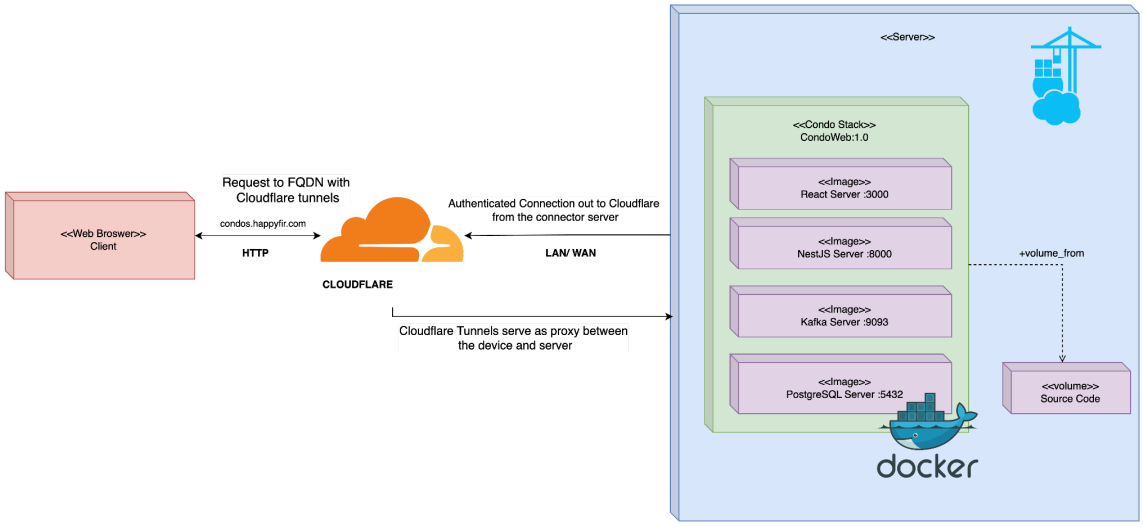


Figure 12: Deployment Diagram of the Condo Management System

The deployment diagram defines all our microservices and all the containers and tables in our server. It represents how our view makes API calls to the controller layer, which is hosted on a physical server. The server contains two independent environments: dev and prod.



Technology for the deployment:

Our deployment infrastructure revolves around a hosted website where we leverage Portainer as our central management tool for orchestrating multiple stacks housing Docker images. Specifically, for our condo management website, we have meticulously organized four distinct stacks to ensure seamless operation. These encompass a frontend stack powered by React on port 3000, a backend stack on port 8000, a Kafka stack on port 9093, and a PostgreSQL database stack on port 5432. Each stack encapsulates the requisite Docker images necessary for their respective functionalities, ensuring efficient resource allocation and scalability. To optimize user accessibility and enhance performance, we utilize Cloudflare to manage HTTP requests and streamline domain name resolution, thereby ensuring a smooth and reliable browsing experience for our clients.

3.4.3.2. Known Issues with View

Even though this view addresses scalability and resource allocation, it's challenging to well align diagrams with real-world infrastructure. Hence, it's complicated to plan solutions and convey those ideas in the diagrams. As the system scales, the diagrams change because of evolving connections between system layers. Similarly, it's hard to accurately represent network configurations and communication protocols.

3.4.4. View: Development

3.4.4.1. Models

3.4.4.1.1. Component Diagram

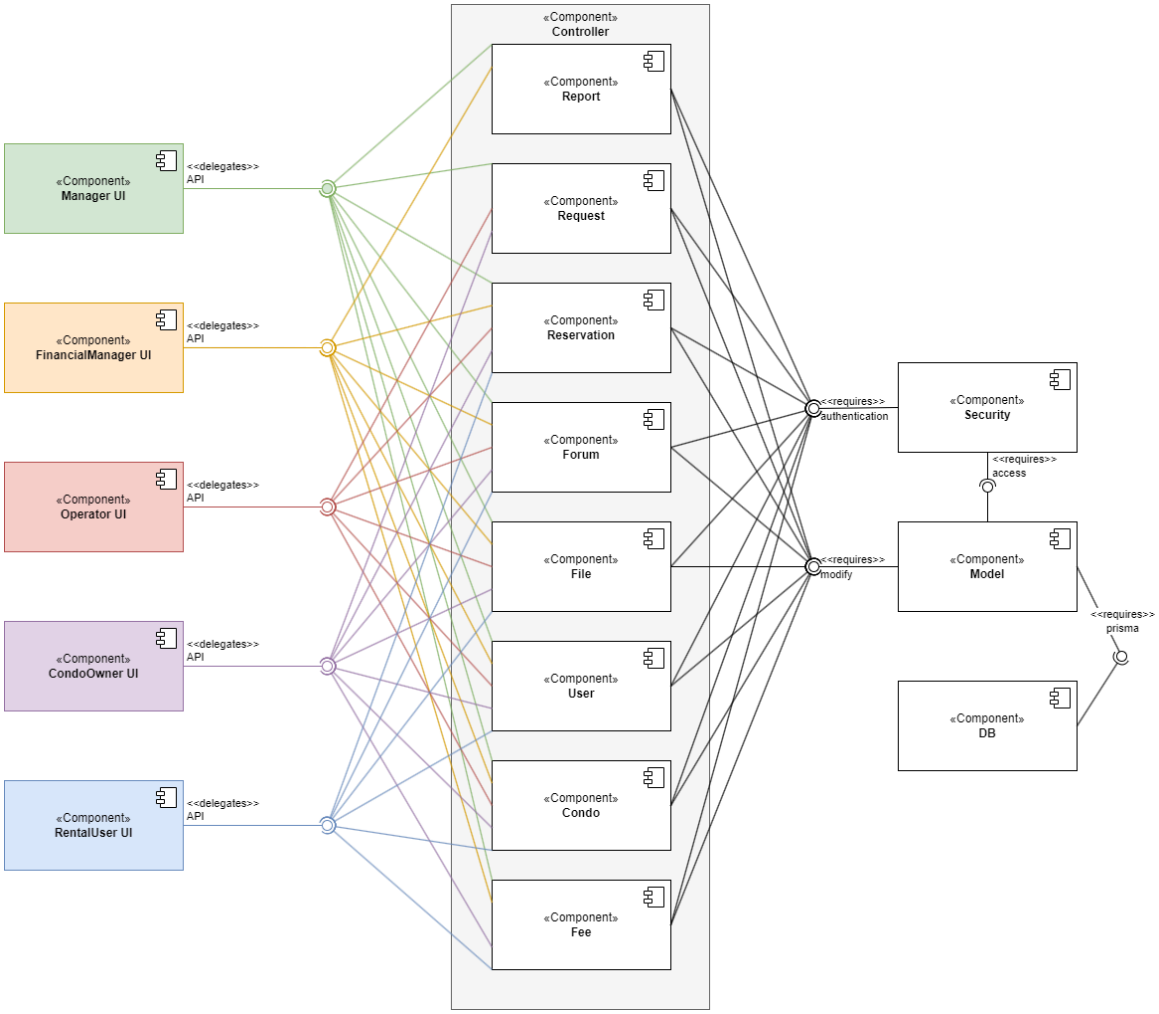


Figure 13: Component Diagram of the Condo Management System

Each UI component is connected to all the controller components it has access to, which varies depending on the user's access rights. Each controller components require authentication from the security component. They also require modification from the model component, which requires access to the security component. Moreover, the model component is required to access the database (DB) component with Prisma.

3.4.4.2. Known Issues with View

Lack of documentation and varying representations of component dependencies result in vague conclusions. Diagrams might show components interacting in a certain way, but they may not work well in real life, possibly due to problems with how information is passed through different parts.

3.4.5. View: Use Case

3.4.5.1. Models

3.4.5.1.1. Use Case Diagram

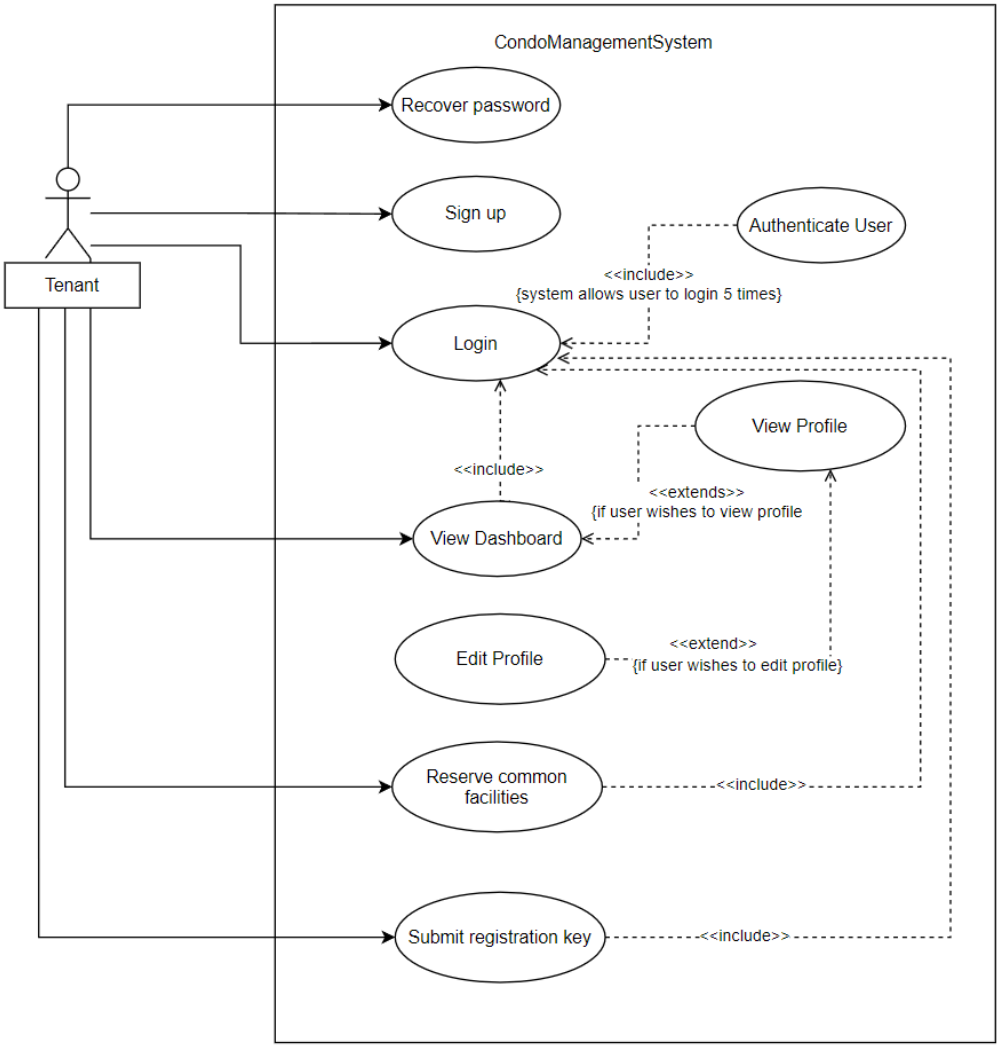


Figure 14: Use case diagram of the Condo Management System from a tenant's point of view

Tenants can create an account, then log in or recover their password if they forget. Login is associated with authentication, which is needed for the other functionalities: accessing their profile and submitting a registration key to link their account to their condo. Once registered, they can view their dashboard and reserve common facilities. Tenants have the same functionality as condo owners, except that they can't submit requests.

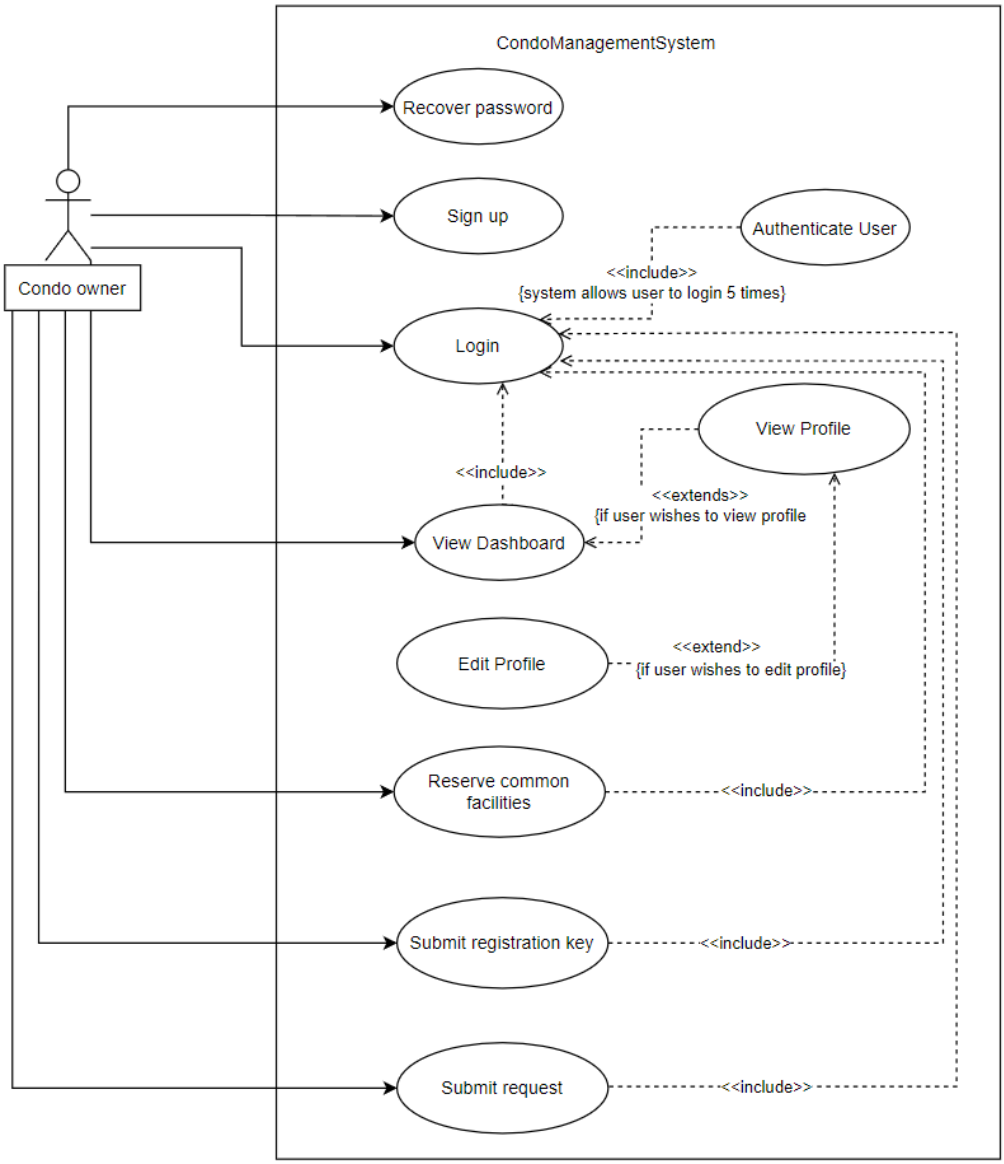


Figure 15: Use case diagram of the Condo Management System from a condo owner's point of view

Condo owners, unlike tenants, can submit requests. They can authenticate themselves and use a registration key to link their account with their condo. They have access to the facilities and a dashboard and can submit requests if they want to.

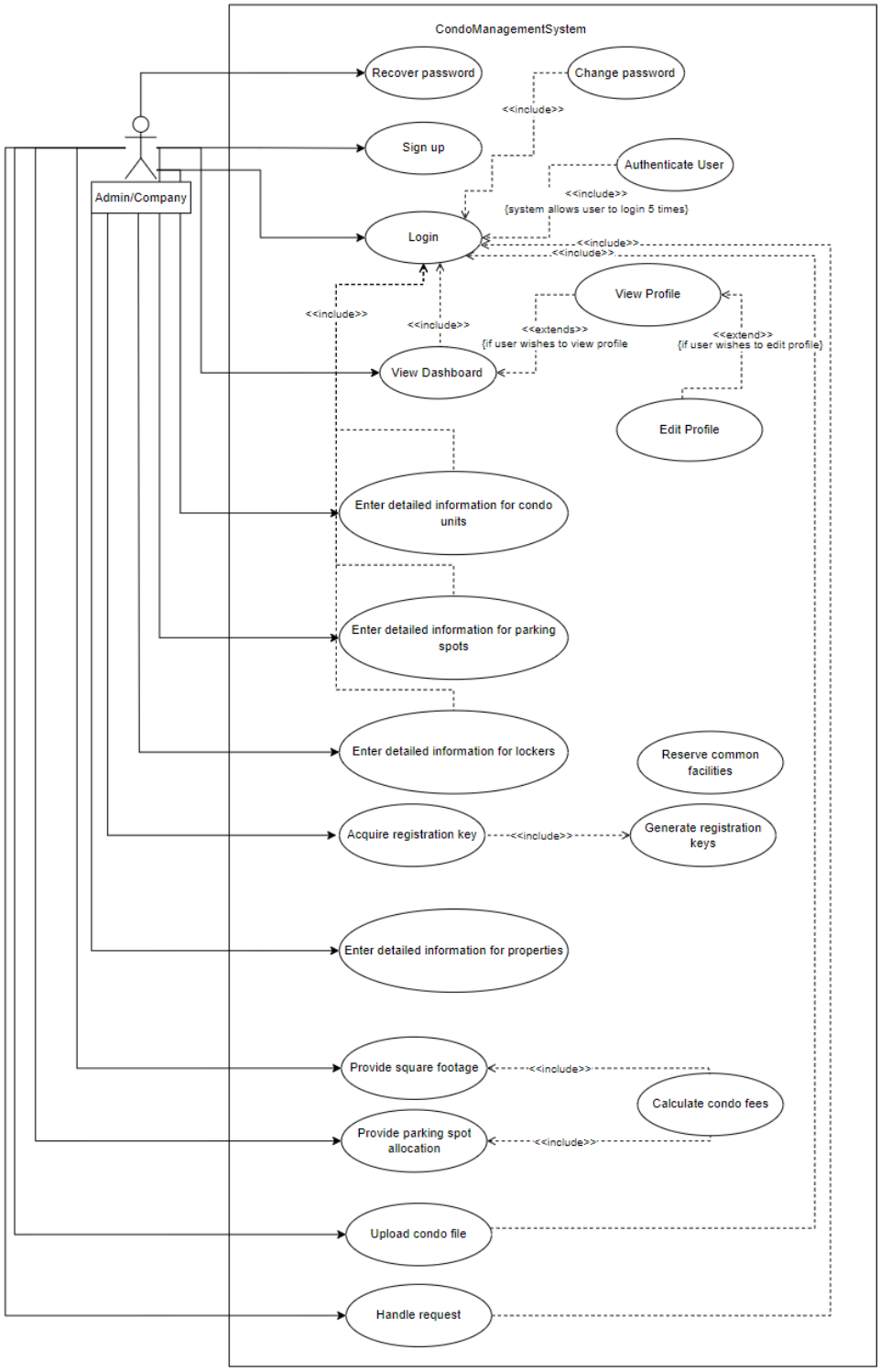


Figure 16: Use case diagram of the Condo Management System from admin's point of view

An admin represents a company, they manage many properties. They can view their profile through their dashboard after successfully logging in. Their dashboard is a hub to access all their information: edit their

profile, view active requests, upload condo files, view properties and view financial reports. When selecting a single property, they can enter the information for condo units, parking spots and lockers.

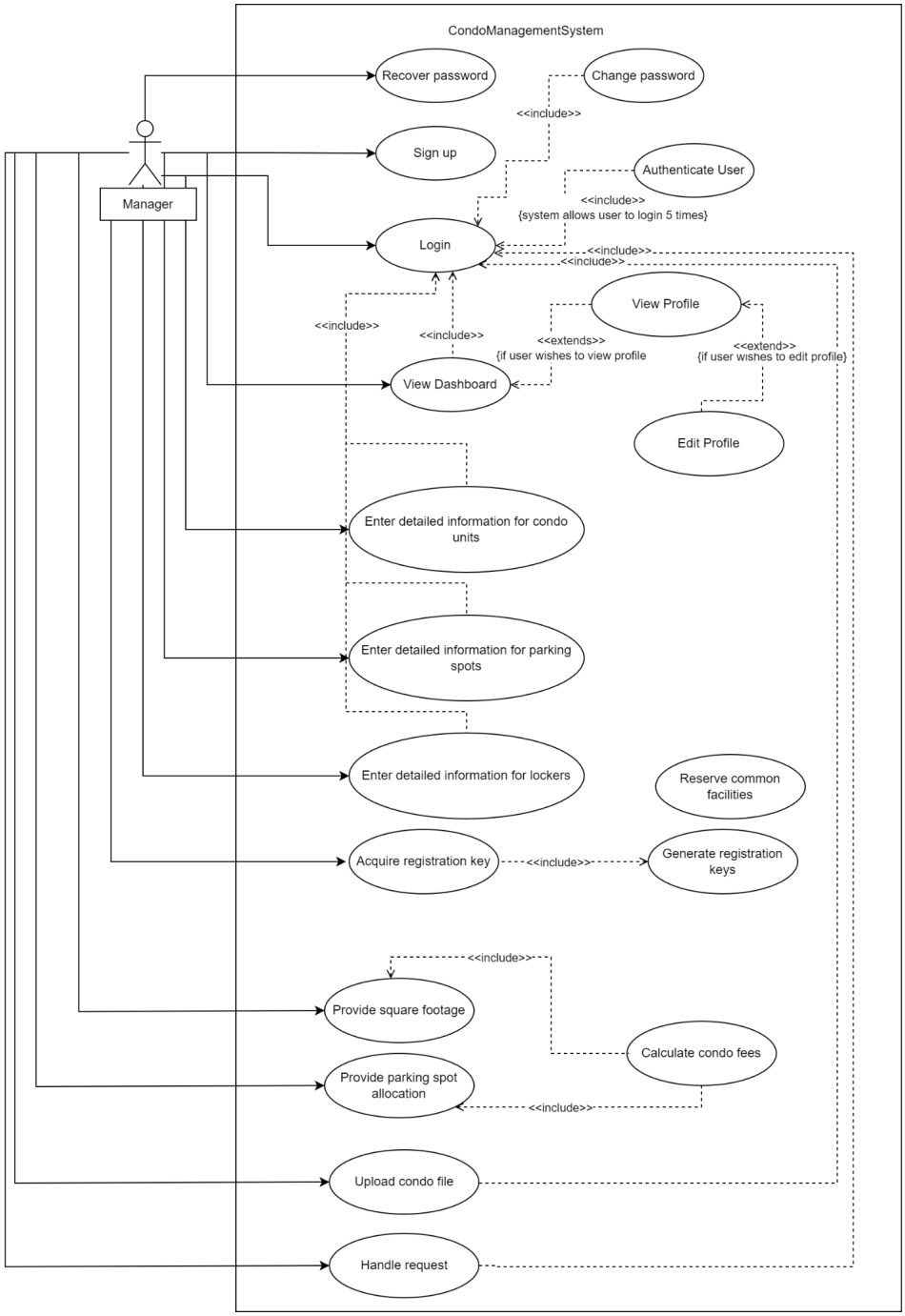


Figure 16: Use case diagram of the Condo Management System from a manager's point of view

Managers take care of a single property. They have almost the same features as admins except that they only have access to their property.

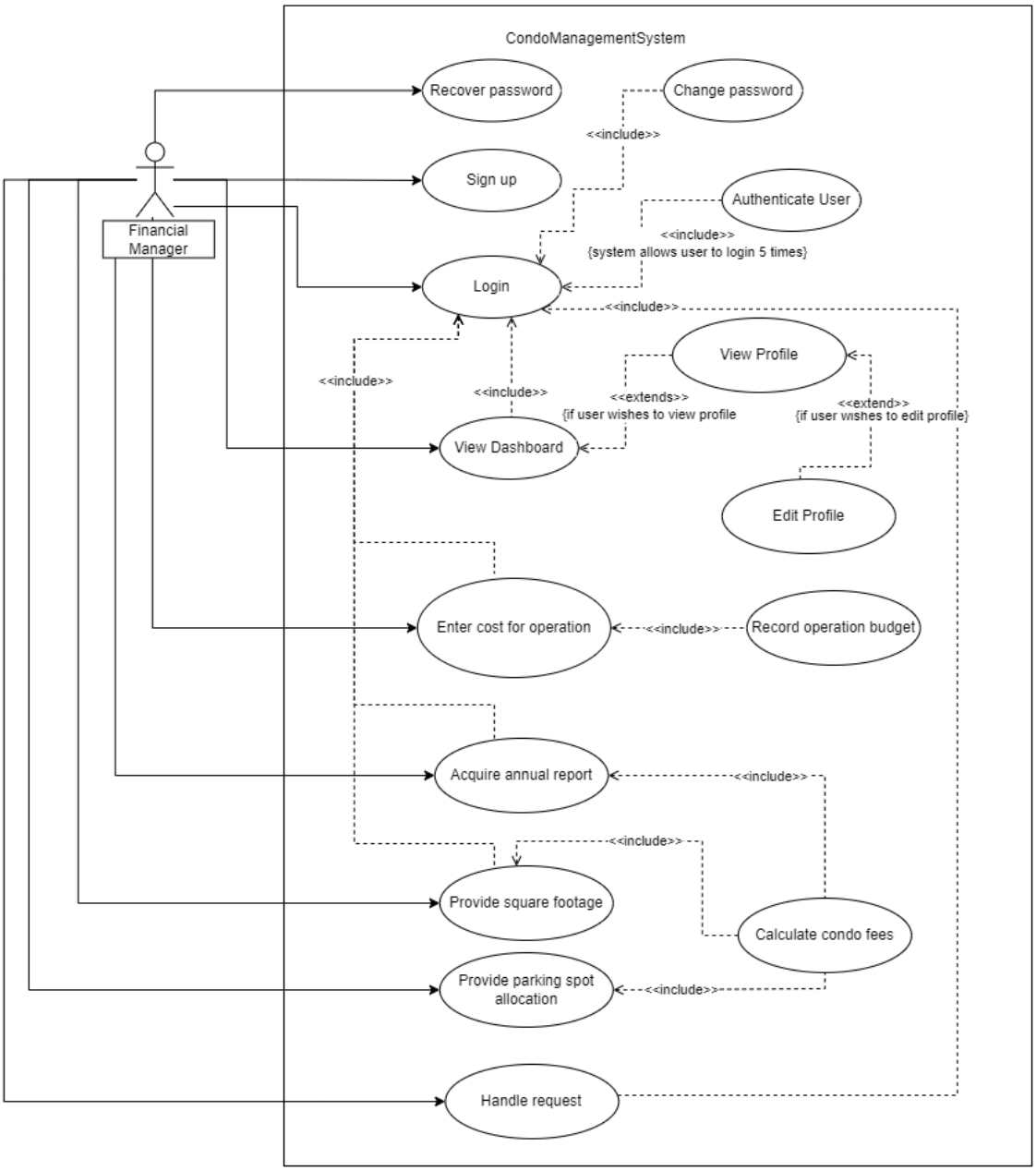


Figure 17: Use case diagram of the Condo Management System from a financial manager's point of view

Financial Managers can view their profile through their dashboard after successfully logging in. Their dashboard is a hub to access all their pertaining information: edit their profile, acquire financial reports, record operational budgets, and handle financial inquiries and issues. They work under their company/admin, hence they have acces to financial data to all their properties. Financial managers and managers are distinct roles within a company. Managers primarily handle a property, like staffing, organization, etc, while financial managers focus on the accounting aspects of the company.

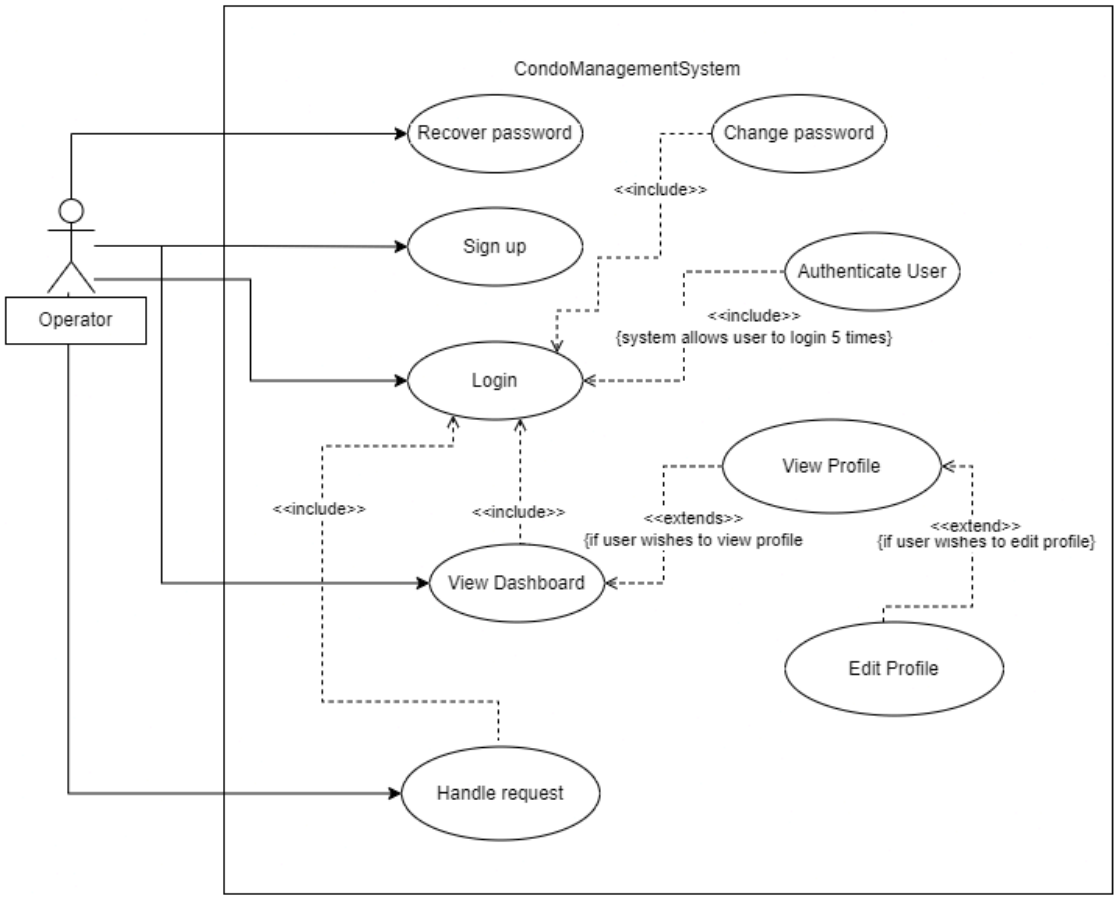


Figure 18: Use case diagram of the Condo Management System from a operator's point of view

Operators and all the other employees working in the property such as janitors, custodians, receptionists, etc. Operators can view their profile through their dashboard after successfully logging in. Their dashboard is a hub to access all their pertaining information: edit their profile, and handle requests for daily operational inquiries or issues in the property.

3.4.5.2. Known Issues with View

Use cases can overlook edge cases and exceptions. Due to incomplete coverage of user interactions, the system can produce errors or inconsistent and vague responses. Furthermore, if there are a large number of use cases, it's challenging to have the same style of response in every scenario. When responses vary, it leads to a worse user experience. Lastly, the use case view fails to address potential conflicts between different user roles and their respective interactions with the system. For example, a tenant may want to pay the fee shown on their screen while the manager increases the fee on their side at the same time.

3.5. Consistency an Correspondences

3.5.1. Known inconsistencies

Inconsistencies between the use case diagram and the activity diagram:

Inconsistencies may arise between the interactions in the use case diagram and the detailed activities in the activity diagrams, resulting from their different levels of abstraction. Focusing on the external view, use case diagrams depict key high-level interactions between actors and the system. On the other hand, activity diagrams show the precise flow of activities within a particular use case. Stakeholders have varying priorities and technical expertise; hence they may understand requirements or design decisions differently. Resolving inconsistencies requires more negotiation and collaboration between stakeholders, which isn't always feasible.

Inconsistencies between the development view and the physical view:

The development view describes the organization of software components such as modules, subsystems, and their dependencies. This view captures concerns related to software development and maintenance. On the opposite side, the physical view mainly focuses on the connections between software components and physical hardware, addressing concerns such as deployment, scalability, and performance. Inconsistencies can arise when the real-life deployment differs from the planned development view, increasing the risk that the software doesn't operate as expected.

Inconsistencies between the physical view and the process view:

Inconsistencies may occur between the physical view and the process view when the runtime system is limited by physical constraints such as network latency or resource limitations. To realistically achieve the expected performance, we might have to adjust the process view, such as optimizing communication protocols, load balancing, or fault tolerance mechanisms.

Inconsistencies between the logical view and the process view:

The logical view emphasizes the system's static structure (components, modules and relationships). In contrast, the process view highlights its dynamic behaviour, illustrating the runtime interactions among components. Inconsistencies may arise on how the logical structure aligns with the runtime behaviour, especially when changes happen in one view but not in the other.

Inconsistencies between the logical view and the physical view:

A common inconsistency between these views is translating the logical designs into physical infrastructure. For instance, a logical view can portray a distributed system architecture, while the physical view can reveal limitations in the deployment environment. In complex systems, it's challenging to ensure consistency across all views and diagrams, especially with frequent changes. As the system progresses, new requirements and constraints may come up, making it harder to maintain consistency.

3.5.2. Correspondences in the AD

Correspondences between the logical view and the development view:

The logical view and the development view associate the system's logical architecture and its organization into development modules, subsystems, or components. Changes in one view should

be reflected in the other, thus ensuring that the system's structure stays aligned with its planned architecture during development.

Correspondences between the logical view and the physical view:

The logical view and the use-case view are related to the system's functionality and its logical components or modules. This ensures that the logical architecture supports the functionality required by the identified use cases and that changes in use cases are reflected appropriately in the logical structure of the system.

Correspondences between the process view and the physical view:

The runtime behaviour of the system must align with its physical configuration and constraints. This is ensured by the correlation between the process and physical views, where the processes and communication among components map to their physical deployment.

3.5.3. Correspondence rules

For the separation of concerns, our diagrams are well-defined and don't take elements from other views. For example, our class diagram focuses on our backend structure, which isn't present at all in the process view. Similarly, the separation of concerns of our controller is identified in the physical view, under the context of API calls, and the development view, under the context of their connections to the different views important to consider during development.

For completeness, all relevant architectural elements and relationships are represented in their corresponding diagrams, allowing the understanding of the system from an architectural point of view.

For contextual integration, as we are operating in an agile methodology, iterative cycles of development allow us to make incremental improvements to our web application, while making time for frequent feedback. This has allowed us to continuously adapt our design to the initial requirements of our stakeholders.

Lastly, using the principles of modularity and reusability, the website is designed as a collection of reusable modules used in all the pages of our website; this facilitates the scalability of the project, as well as regulating the format of our code.

4. Overall Retrospective

4.1. Introduction

In our ongoing venture, we're actively developing a Condo Management System, a digital solution crafted to streamline and integrate various facets of condominium living. Leveraging the SERN (SQL, Express.js, React, Node.js) stack and integrating Docker for deployment, our team is dedicated to forging a seamless platform for efficient condo management. Our primary objective remains to deliver a user-friendly and interconnected system that elevates the overall condo living experience.

Throughout the project, we encountered a mix of successes and challenges, and this document will outline what worked well, difficulties, and opportunities for refinement in the final outcome. In this document, we will discuss what went wrong (why it happened, its impact, how we addressed the issue and improved). We will also discuss the same aspects about what went right.

4.2. What Went Wrong

4.2.1. Sprint 1

4.2.1.1. Lack of Division of Responsibilities

- **Why did it happen?** The lack of explicitly defined roles and responsibilities within the team led to confusion regarding who was accountable for specific tasks.
- **Impact:** Ambiguities in responsibilities resulted in delays and hindered overall project progress.
- **Addressing the Issue:** Initiating a structured review of team roles and responsibilities, along with individual discussions to clarify expectations.
- **What can be improved:** Implement a clear and well-documented division of responsibilities at the beginning of each project phase to avoid confusion and streamline workflow.

4.2.1.2. Absence of Setting Deadlines

- **Why did it happen?** The absence of deadlines for certain tasks led to a lack of urgency, causing delays and impacting the overall project timeline.
- **Impact:** Tasks were prolonged as team members worked without defined time constraints, affecting the sprint's efficiency.
- **Addressing the Issue:** Introducing well-defined deadlines for each task within the sprint to ensure a structured and timely workflow.
- **What can be improved:** Implement a comprehensive timeline for all tasks, fostering a proactive work environment and preventing unnecessary delays.

4.2.1.3. Not Writing Tests with Tasks

- **Why did it happen?** The team overlooked the integration of testing procedures with individual tasks, possibly due to time constraints or oversight.
- **Impact:** Insufficient test coverage led to undetected bugs and challenges in ensuring the system's reliability.
- **Addressing the Issue:** Implementing a testing-first approach, ensuring that tests are written in tandem with task completion, promoting a more robust and error-resistant codebase.
- **What can be improved:** Emphasize the importance of writing tests concurrently with task development to maintain a high standard of code quality and not accept Pull Requests without test documents.

4.2.1.4. Inconsistent Naming Conventions

- **Why did it happen?** The absence of a standardized naming convention resulted in team members adopting different styles, leading to confusion and potential errors.
- **Impact:** Code readability suffered, making it challenging for team members to understand and collaborate effectively.
- **Addressing the Issue:** Establishing and enforcing a unified naming convention for code elements to enhance consistency and readability.
- **What can be improved:** Conduct regular code reviews, emphasizing adherence to the agreed-upon naming convention to ensure uniformity across the entire codebase

4.2.1.5. Difficult to organize a meeting where every single member is present

- **Why did it happen?** Scheduling meetings proved challenging due to varying personal commitments, and diverse schedules among team members.

- **Impact:** The inability to have every member present hindered effective communication and decision-making during crucial project discussions.
- **Addressing the Issue:** Implementing a collaborative scheduling tool to identify overlapping availability and streamline meeting coordination.
- **What can be improved:** Establishing a standardized meeting schedule at the project's outset to accommodate diverse time zones and commitments, ensuring optimal attendance.

4.2.2. Sprint 2

4.2.2.1. Backend Overhaul

- **Issue:** The backend team decided to revamp the entire codebase early in the project.
- **Why did it happen?** Lack of proper planning and communication regarding the initial architecture.
- **Impact:** Delayed development progress and increased workload for both frontend and backend teams.
- **Addressing the Issue:** Implemented stricter planning protocols for major architectural decisions and enhanced communication channels between teams.

4.2.2.2. Inflexible Task Prioritization

- **Issue:** Task prioritization was rigid and did not allow for flexibility in addressing emerging issues or opportunities.
- **Why did it happen?** The Lack of a dynamic task prioritization process and adherence to a fixed roadmap.
- **Impact:** Hindered the team's ability to adapt to changing requirements or user feedback promptly.
- **Addressing the Issue:** Introduced a more agile approach to task prioritization, allowing for frequent reassessment and reprioritization based on evolving project needs.
- **What can be improved:** Implement a continuous feedback loop and embrace a more iterative approach to task prioritization to enhance responsiveness and agility in project execution.

4.2.2.3. Communication Gap

- **Issue:** Insufficient communication between frontend and backend teams.
- **Why did it happen?** Lack of regular sync-up meetings and coordination efforts.
- Impact: Inconsistent integration of frontend and backend components, leading to compatibility issues.
- **Impact:** Introduced regular cross-functional meetings and established communication channels for real-time collaboration.
- **Addressing the Issue:** Introduced regular cross-functional meetings and established communication channels for real-time collaboration.
- **What can be improved:** Implement automated integration testing to identify compatibility issues early in the development cycle.

4.2.2.4. Task Division Imbalance

- **Issue:** Imbalance in the division of tasks within the web application development**.**
- **Why did it happen?** Inadequate assessment of individual skill sets and workload distribution.

- **Impact:** Uneven progress across different components of the system, leading to bottlenecks.
- **Addressing the Issue:** Conducted regular retrospective meetings to reassess task allocation and redistribute workload accordingly.
- **What can be improved:** Implement a skill-based task allocation system to ensure equitable distribution of responsibilities.

4.2.2.5. Difficult to organize a meeting where every single member is present

- **Issue:** We could not get everyone in 1 meeting to make sure everyone is on the same page in sprint 2.
- **Why did it happen?** Scheduling meetings proved challenging due to varying personal commitments, and diverse schedules among team members.
- **Impact:** The inability to have every member present hindered effective communication and decision-making during crucial project discussions.
- **Addressing the Issue:** Implementing a collaborative scheduling tool to identify overlapping availability and streamline meeting coordination.
- **What can be improved:** Establishing a standardized meeting schedule at the project's outset to accommodate diverse time zones and commitments, ensuring optimal attendance.

4.2.3. Sprint 3

4.2.3.1. Frontend Manual Testing Challenges

- **Issue:** Difficulties encountered during frontend testing due to backend changes impacting database state.
- **Why did it happen?** Insufficient coordination between frontend and backend teams regarding database management during testing.
- **Impact:** Hindered effective frontend testing, leading to delays in identifying and resolving issues.
- **Addressing the Issue:** Implemented stricter protocols for coordinating database changes between frontend and backend teams during testing phases.
- **What can be improved:** Enhance communication and synchronization between frontend and backend teams to ensure a stable testing environment.

4.2.3.2. Inflexible Task Prioritization

- **Issue:** Task prioritization was rigid and did not allow for flexibility in addressing emerging issues or opportunities.
- **Why did it happen?** The Lack of a dynamic task prioritization process and adherence to a fixed roadmap.
- **Impact:** Hindered the team's ability to adapt to changing requirements or user feedback promptly.
- **Addressing the Issue:** Introduced a more agile approach to task prioritization, allowing for frequent reassessment and reprioritization based on evolving project needs.
- **What can be improved:** Implement a continuous feedback loop and embrace a more iterative approach to task prioritization to enhance responsiveness and agility in project execution.

4.2.3.3. Communication Gap

- **Issue:** Insufficient communication between frontend and backend teams.

- **Why did it happen?** Lack of regular sync-up meetings and coordination efforts.
- Impact: Inconsistent integration of frontend and backend components, leading to compatibility issues.
- **Impact:** Introduced regular cross-functional meetings and established communication channels for real-time collaboration.
- **Addressing the Issue:** Introduced regular cross-functional meetings and established communication channels for real-time collaboration.
- **What can be improved:** Implement automated integration testing to identify compatibility issues early in the development cycle.

4.2.3.4. Task Division Imbalance

- **Issue:** Imbalance in the division of tasks within the web application development**.**
- **Why did it happen?** Inadequate assessment of individual skill sets and workload distribution.
- **Impact:** Uneven progress across different components of the system, leading to bottlenecks.
- **Addressing the Issue:** Conducted regular retrospective meetings to reassess task allocation and redistribute workload accordingly.
- **What can be improved:** Implement a skill-based task allocation system to ensure equitable distribution of responsibilities.

4.2.3.5. Difficult to organize a meeting where every single member is present

- **Issue:** We could not get everyone in 1 meeting to make sure everyone is on the same page in sprint 2.
- **Why did it happen?** Scheduling meetings proved challenging due to varying personal commitments, and diverse schedules among team members.
- **Impact:** The inability to have every member present hindered effective communication and decision-making during crucial project discussions.
- **Addressing the Issue:** Implementing a collaborative scheduling tool to identify overlapping availability and streamline meeting coordination.
- **What can be improved:** Establishing a standardized meeting schedule at the project's outset to accommodate diverse time zones and commitments, ensuring optimal attendance.

4.2.4. Sprint 4

4.2.4.1. Frontend Manual Testing Challenges

- **Issue:** Difficulties encountered during frontend testing due to backend changes impacting database state.
- **Why did it happen?** Insufficient coordination between frontend and backend teams regarding database management during testing.
- **Impact:** Hindered effective frontend testing, leading to delays in identifying and resolving issues.
- **Addressing the Issue:** Implemented stricter protocols for coordinating database changes between frontend and backend teams during testing phases.
- **What can be improved:** Enhance communication and synchronization between frontend and backend teams to ensure a stable testing environment.

4.2.4.2. Inflexible Task Prioritization

- **Issue:** Task prioritization was rigid and did not allow for flexibility in addressing emerging issues or opportunities.
- **Why did it happen?** The Lack of a dynamic task prioritization process and adherence to a fixed roadmap.
- **Impact:** Hindered the team's ability to adapt to changing requirements or user feedback promptly.
- **Addressing the Issue:** Introduced a more agile approach to task prioritization, allowing for frequent reassessment and reprioritization based on evolving project needs.
- **What can be improved:** Implement a continuous feedback loop and embrace a more iterative approach to task prioritization to enhance responsiveness and agility in project execution.

4.2.4.3. Communication Gap

- **Issue:** Insufficient communication between frontend and backend teams.
- **Why did it happen?** Lack of regular sync-up meetings and coordination efforts.
- Impact: Inconsistent integration of frontend and backend components, leading to compatibility issues.
- **Impact:** Introduced regular cross-functional meetings and established communication channels for real-time collaboration.
- **Addressing the Issue:** Introduced regular cross-functional meetings and established communication channels for real-time collaboration.
- **What can be improved:** Implement automated integration testing to identify compatibility issues early in the development cycle.

4.2.4.4. Task Division Imbalance

- **Issue:** Imbalance in the division of tasks within the web application development**.**
- **Why did it happen?** Inadequate assessment of individual skill sets and workload distribution.
- **Impact:** Uneven progress across different components of the system, leading to bottlenecks.
- **Addressing the Issue:** Conducted regular retrospective meetings to reassess task allocation and redistribute workload accordingly.
- **What can be improved:** Implement a skill-based task allocation system to ensure equitable distribution of responsibilities.

4.2.4.5. Difficult to organize a meeting where every single member is present

- **Issue:** We could not get everyone in 1 meeting to make sure everyone is on the same page in sprint 2.
- **Why did it happen?** Scheduling meetings proved challenging due to varying personal commitments, and diverse schedules among team members.
- **Impact:** The inability to have every member present hindered effective communication and decision-making during crucial project discussions.
- **Addressing the Issue:** Implementing a collaborative scheduling tool to identify overlapping availability and streamline meeting coordination.
- **What can be improved:** Establishing a standardized meeting schedule at the project's outset to accommodate diverse time zones and commitments, ensuring optimal attendance.

4.2.5. Sprint 5

4.2.5.1. Frontend Manual Testing Challenges

- **Issue:** Difficulties encountered during frontend testing due to backend changes impacting database state.
- **Why did it happen?** Insufficient coordination between frontend and backend teams regarding database management during testing.
- **Impact:** Hindered effective frontend testing, leading to delays in identifying and resolving issues.
- **Addressing the Issue:** Implemented stricter protocols for coordinating database changes between frontend and backend teams during testing phases.
- **What can be improved:** Enhance communication and synchronization between frontend and backend teams to ensure a stable testing environment.

4.2.5.2. Inconsistent performance of tools used in backend

- **Issue:** Inconsistent performance of Kafka in the backend.
- **Why did it happen?** The backend tool, Kafka, exhibited irregular failures without clear reasons, possibly stemming from configuration issues or compatibility issues within the backend environment.
- **Impact:** Unpredictable failures in Kafka hindered data processing and communication flow, potentially causing delays and disruptions in system operations.
- **Addressing the Issue:** Conduct a thorough investigation into Kafka's configuration and deployment, collaborate with backend engineers and Kafka experts to identify root causes, implement necessary adjustments to enhance Kafka's stability, and establish robust monitoring mechanisms to promptly detect and address any anomalies.
- **What can be improved:** Strengthen proactive monitoring and troubleshooting capabilities, optimize Kafka's configuration for better compatibility and performance, and foster collaboration among backend engineers to collectively address and prevent similar issues in the future.

4.2.5.3. Communication Gap

- **Issue:** Insufficient communication between frontend and backend teams.
- **Why did it happen?** Lack of regular sync-up meetings and coordination efforts.
- Impact: Inconsistent integration of frontend and backend components, leading to compatibility issues.
- **Impact:** Introduced regular cross-functional meetings and established communication channels for real-time collaboration.
- **Addressing the Issue:** Introduced regular cross-functional meetings and established communication channels for real-time collaboration.
- **What can be improved:** Implement automated integration testing to identify compatibility issues early in the development cycle.

4.2.5.4. Difficult to organize a meeting where every single member is present

- **Issue:** We could not get everyone in 1 meeting to make sure everyone is on the same page in sprint 2.
- **Why did it happen?** Scheduling meetings proved challenging due to varying personal commitments, and diverse schedules among team members.

- **Impact:** The inability to have every member present hindered effective communication and decision-making during crucial project discussions.
- **Addressing the Issue:** Implementing a collaborative scheduling tool to identify overlapping availability and streamline meeting coordination.
- **What can be improved:** Establishing a standardized meeting schedule at the project's outset to accommodate diverse time zones and commitments, ensuring optimal attendance.

4.3. What Went Right

4.3.1. Sprint 1

4.3.1.1 Proactive Team Collaboration

- **Why did it happen?** The team valued collaboration for its positive impact on learning and project outcomes. We recognized that diverse contributions lead to better solutions.
- **Impact:** Improved problem-solving, knowledge sharing, and a positive team dynamic.
- **Addressing it:** Encouraged regular collaborative sessions, utilized online platforms, and emphasized the value of individual contributions.
- **Improvement:** Introduce structured team-building activities within the academic context to further strengthen collaboration.

4.3.1.2 Good Continuous Work

- **Why did it happen?** Understanding the academic nature, we aimed for steady effort, reducing stress and ensuring a thorough understanding of course concepts.
- **Impact:** Reduced stress, improved comprehension, and a steady project progression.
- **Addressing it:** Implemented milestones, time management, and a disciplined approach to tasks.
- **Improvement:** Implement periodic self-assessments to gauge individual contributions and ensure adherence to project timelines.

4.3.1.3 Effective Communication between Frontend and Backend Teams

- **Why did it happen?** Recognizing communication's role, the team aimed for shared learning and successful project execution.
- **Impact:** Enhanced understanding, improved collaboration, and streamlined project outcomes.
- **Addressing it:** Utilized university tools, scheduled cross-disciplinary meetings, and emphasized collaborative learning.
- **Improvement:** Consider academic-focused workshops on effective communication strategies for interdisciplinary projects.

4.3.1.4 Well-Conducted Team Meetings

- **Why did it happen?** Well-structured meetings were deemed essential for shared understanding and coordination.
- **Impact:** Improved cohesion, clarity on goals, and individual accountability.
- **Addressing it:** Structured meetings with clear agendas, university tools, and documentation of key decisions.

- **Improvement**: Meet in person to make sure that everyone is active and focusing.

4.3.1.5 Quick Decision-Making as a Team

- **Why did it happen?** Swift decision-making was adopted due to academic constraints and project timelines.
- **Impact:** Minimized delays, maintained momentum, and increased overall productivity.
- **Addressing it:** Established clear decision-making protocols, empowered team members, and maintained transparent communication.
- **Improvement:** Make sure that we fully understand the requirements as soon as the sprint starts to make decisions quickly.

4.3.2. Sprint 2

4.3.2.1 Consistent Naming Conventions

- **Why did it happen?** Adhered to consistent naming conventions across the project
- **Impact:** Enhanced code readability and maintainability, facilitating smoother collaboration among team members.
- **Addressing it:** Enforced naming conventions through code reviews and automated linting tools.
- **Improvement:** Continuously update and refine naming conventions to align with evolving best practices.

4.3.2.2 Test-Inclusive Pull Requests (PR)

- **Why did it happen?** Ensured that all PRs included comprehensive test coverage with a coverage above 80% in all metrics
- **Impact:** Reduced the likelihood of introducing regressions and improved overall code quality.
- **Addressing it:** Implemented automated code quality checks and mandated test coverage thresholds for PR approvals.
- **Improvement:** Enhance test coverage metrics and provide additional training on writing effective unit and integration tests.

4.3.2.3 Sprint Goal Achievement

- **Why did it happen?** Successfully covered all user stories outlined for the sprint
- **Impact:** Demonstrated team efficiency and progress toward project milestones.
- **Addressing it:** Maintained a prioritized backlog and conducted regular sprint planning sessions.
- **Improvement:** Refine estimation techniques and conduct more granular user story breakdowns for improved sprint planning accuracy.

4.3.2.4 Good Progression of Work from the Beginning

- **Why did it happen?** The project exhibited a consistent and steady pace of progress from its inception.
- **Impact:** Ensured timely completion of milestones and maintained momentum throughout the development cycle.

- **Addressing it:** Regularly monitored progress against project timelines and adjusted resource allocation as needed to maintain productivity.
- **Improvement**: Continue to emphasize the importance of pacing and consistency in work progression, ensuring that momentum is sustained across all stages of development.

4.3.2.5 Quick Decision-Making as a Team

- **Why did it happen?** Swift decision-making was adopted due to academic constraints and project timelines.
- **Impact:** Minimized delays, maintained momentum, and increased overall productivity.
- **Addressing it:** Established clear decision-making protocols, empowered team members, and maintained transparent communication.
- **Improvement:** Make sure that we fully understand the requirements as soon as the sprint starts to make decisions quickly.

4.3.3. Sprint 3

4.3.3.1 Consistent Naming Conventions

- **Why did it happen?** Adhered to consistent naming conventions across the project
- **Impact:** Enhanced code readability and maintainability, facilitating smoother collaboration among team members.
- **Addressing it:** Enforced naming conventions through code reviews and automated linting tools.
- **Improvement:** Continuously update and refine naming conventions to align with evolving best practices.

4.3.3.2 Test-Inclusive Pull Requests (PR)

- **Why did it happen?** Ensured that all PRs included comprehensive test coverage with a coverage above 80% in all metrics
- **Impact:** Reduced the likelihood of introducing regressions and improved overall code quality.
- **Addressing it:** Implemented automated code quality checks and mandated test coverage thresholds for PR approvals.
- **Improvement:** Enhance test coverage metrics and provide additional training on writing effective unit and integration tests.

4.3.3.3 Sprint Goal Achievement

- **Why did it happen?** Successfully covered all user stories outlined for the sprint
- **Impact:** Demonstrated team efficiency and progress toward project milestones.
- **Addressing it:** Maintained a prioritized backlog and conducted regular sprint planning sessions.
- **Improvement:** Refine estimation techniques and conduct more granular user story breakdowns for improved sprint planning accuracy.

4.3.3.4 Early Completion of Essential Tasks

- **Why did it happen?** Prioritization of essential tasks at the project's outset led to their early completion.

- **Impact:** Reduced wait time for dependent tasks and ensured a smoother workflow progression.
- **Addressing it:** Maintained a focus on critical tasks during sprint planning sessions and emphasized their early completion.
- **Improvement:** Further refine task prioritization techniques to identify and prioritize essential tasks more effectively.

4.3.3.5 Timely Execution and Good Time Management

- **Why did it happen?** Effective time management practices were employed throughout the sprint.
- **Impact:** Facilitated timely completion of tasks and maintained project momentum.
- **Addressing it:** Maintained regular monitoring of project timelines and conducted periodic assessments of task progress.
- **Improvement:** Continuously refine time management strategies to optimize productivity and task completion efficiency.

4.3.4. Sprint 4

4.3.4.1 Consistent Naming Conventions

- **Why did it happen?** Adhered to consistent naming conventions across the project
- **Impact:** Enhanced code readability and maintainability, facilitating smoother collaboration among team members.
- **Addressing it:** Enforced naming conventions through code reviews and automated linting tools.
- **Improvement:** Continuously update and refine naming conventions to align with evolving best practices.

4.3.4.2 Test-Inclusive Pull Requests (PR)

- **Why did it happen?** Ensured that all PRs included comprehensive test coverage with a coverage above 80% in all metrics
- **Impact:** Reduced the likelihood of introducing regressions and improved overall code quality.
- **Addressing it:** Implemented automated code quality checks and mandated test coverage thresholds for PR approvals.
- **Improvement:** Enhance test coverage metrics and provide additional training on writing effective unit and integration tests.

4.3.4.3 Sprint Goal Achievement

- **Why did it happen?** Successfully covered all user stories outlined for the sprint
- **Impact:** Demonstrated team efficiency and progress toward project milestones.
- **Addressing it:** Maintained a prioritized backlog and conducted regular sprint planning sessions.
- **Improvement:** Refine estimation techniques and conduct more granular user story breakdowns for improved sprint planning accuracy.

4.3.4.4 Early Completion of Essential Tasks

- **Why did it happen?** Prioritization of essential tasks at the project's outset led to their early completion.

- **Impact:** Reduced wait time for dependent tasks and ensured a smoother workflow progression.
- **Addressing it:** Maintained a focus on critical tasks during sprint planning sessions and emphasized their early completion.
- **Improvement:** Further refine task prioritization techniques to identify and prioritize essential tasks more effectively.

4.3.4.5 Timely Execution and Good Time Management

- **Why did it happen?** Effective time management practices were employed throughout the sprint.
- **Impact:** Facilitated timely completion of tasks and maintained project momentum.
- **Addressing it:** Maintained regular monitoring of project timelines and conducted periodic assessments of task progress.
- **Improvement:** Continuously refine time management strategies to optimize productivity and task completion efficiency.

4.3.5. Sprint 5

4.3.5.1 Consistent Naming Conventions

- **Why did it happen?** Adhered to consistent naming conventions across the project
- **Impact:** Enhanced code readability and maintainability, facilitating smoother collaboration among team members.
- **Addressing it:** Enforced naming conventions through code reviews and automated linting tools.
- **Improvement:** Continuously update and refine naming conventions to align with evolving best practices.

4.3.5.2 Test-Inclusive Pull Requests (PR)

- **Why did it happen?** Ensured that all PRs included comprehensive test coverage with a coverage above 80% in all metrics
- **Impact:** Reduced the likelihood of introducing regressions and improved overall code quality.
- **Addressing it:** Implemented automated code quality checks and mandated test coverage thresholds for PR approvals.
- **Improvement:** Enhance test coverage metrics and provide additional training on writing effective unit and integration tests.

4.3.5.3 Sprint Goal Achievement

- **Why did it happen?** Successfully covered all user stories outlined for the sprint
- **Impact:** Demonstrated team efficiency and progress toward project milestones.
- **Addressing it:** Maintained a prioritized backlog and conducted regular sprint planning sessions.
- **Improvement:** Refine estimation techniques and conduct more granular user story breakdowns for improved sprint planning accuracy.

4.3.5.4 Early Completion of Essential Tasks

- **Why did it happen?** Prioritization of essential tasks at the project's outset led to their early completion.

- **Impact:** Reduced wait time for dependent tasks and ensured a smoother workflow progression.
- **Addressing it:** Maintained a focus on critical tasks during sprint planning sessions and emphasized their early completion.
- **Improvement:** Further refine task prioritization techniques to identify and prioritize essential tasks more effectively.

4.3.5.5 Timely Execution and Good Time Management

- **Why did it happen?** Effective time management practices were employed throughout the sprint.
- **Impact:** Facilitated timely completion of tasks and maintained project momentum.
- **Addressing it:** Maintained regular monitoring of project timelines and conducted periodic assessments of task progress.
- **Improvement:** Continuously refine time management strategies to optimize productivity and task completion efficiency.

4.4. Conclusion

4.4.1. Sprint 1

Despite encountering challenges such as a lack of division of responsibilities and difficulties in organizing meetings, Sprint 1 yielded positive results. The team demonstrated proactive collaboration, maintained continuous work, ensured effective communication between frontend and backend teams, conducted well-structured team meetings and exhibited quick decision-making. These successes highlighted the importance of teamwork, communication, and disciplined work habits.

4.4.2. Sprint 2

Sprint 2 saw improvements in consistent naming conventions, test-inclusive pull requests, and the achievement of sprint goals. However, challenges persisted in task prioritization, communication gaps between frontend and backend teams, and task division imbalance. The team recognized the need for more agile approaches to task prioritization and enhanced communication channels to address these challenges effectively.

4.4.3. Sprint 3

In Sprint 3, the team continued to adhere to consistent naming conventions, prioritize test-inclusive pull requests, and achieve sprint goals. Early completion of essential tasks and effective time management were notable successes. However, communication gaps and difficulties in frontend testing persisted. The team acknowledged the importance of refined task prioritization and enhanced coordination between frontend and backend teams to overcome these challenges.

4.4.4. Sprint 4

Sprint 4 showcased continued adherence to consistent naming conventions, test-inclusive pull requests, and timely execution of tasks. However, challenges in task prioritization and communication gaps remained, indicating the need for further refinement in these areas. The team recognized the importance of early completion of essential tasks and effective time management to maintain project momentum and overcome obstacles.

4.4.5. Sprint 5

In the final sprint, Sprint 5, the team maintained consistency in naming conventions, test-inclusive pull requests, and timely execution of tasks. Challenges in frontend testing and inconsistent performance of backend tools were addressed through enhanced communication and synchronization between frontend and backend teams. The team emphasized the importance of continuous improvement in communication and coordination to ensure project success.

4.4.6. Overall Conclusion

In reflecting on our overall journey throughout the development of the Condo Management System (CMS), our team has undergone a steep learning curve, marked by both successes and challenges. Our retrospective analysis serves as a helpful tool for not only understanding what went wrong and what went right but also for paving the way toward continuous development.

As we moved through our development process, we experienced a variety of issues, including unclear assignments, and communication gaps between teams. While these challenges caused some delays, they also taught us valuable lessons about the need of clear communication (assigning tasks to correct people, clear instructions, detailed PRs), systematic planning (GitHub project tool to define tasks, respect deadlines), and the flexibility to react to changing project requirements (new feature types added along the way i.e enhancement, debug). We addressed these difficulties by discussing between frontend and backend immediately to quickly make the required modifications, which has paved the road for more efficient project execution over the past 4 months.

5. Overall Testing

This test approach document describes the appropriate strategies, process, workflows and methodologies used to plan, organize, execute and manage testing of Condo Management System.

5.1. Testing Tool

As a part of the developing cycle, testing plays a key role in maintainability and reliability for existing software, including the Condo Management System. In 2 parts, backend and frontend, tests will be conducted under types of unit tests, integration test and system test.

5.1.1. Platform

2 platforms that we chose to implement for this project: Github platform and Sonarqube.

5.1.1.1. Github

We implemented 2 workflows as GitHub actions for CI in our repository, one for dev branch and one for main branch. The CI GitHub action is responsible for building and deploying the frontend and backend, and for monitoring the status of the deployment on the server, not to mention the compatibility of the node_modules implemented throughout the project.

5.1.1.2. Sonarqube

We implemented Sonarqube to validate the code quality and maintainability. Not only is it a powerful static code analysis tool that automatically reviews codes for bugs, vulnerabilities, code smells and quality issues, but also it can indicate the latent issues early on, resulting in faster resolution and reducing risks in development.

5.1.2. Testing Module

Jest is a testing framework for unit and integration tests with the benefits of being simple, speedy and built-in features (mocks and snapshots). In the frontend, we implemented Jest with a configuration for Typescript, on the other hand, the backend is implemented with JavaScript. Jest, with its functions, can provide a quick test report with insightful data, telling the % coverage of statements, branches, functions and lines.

5.2. Approach

During the development process, the systematic approach is adopted to use Jest for testing. The process includes several important steps to ensure our tests' efficiency and the imminent resolution of issues may encounter:

5.2.1. Test Planning and Design

Supposedly, before any implementation, test cases and scenarios of each functional requirement, user story or acceptance criterion will be conveyed. All possible cases of functionality (positive, negative, edge, error) will have to be covered by test cases.

5.2.2. Unit Test

Unit testing evaluates individual components or "units" of software independently. It is a part of white box testing, an approach that examines the internal modules of the code. These units are the smallest testable pieces in the software. Unit testing is an essential component of software development since it ensures that each unit is executed correctly and detects any flaws early on. Following the TDD approach, each component, function or module will be put as top priority to write unit tests. After each

implementation or code change, we execute the unit test to ensure the maintainability and quality of the updated codes. Notwithstanding, it results in high chances of catching regression earlier in the process and ensuring the new code still meets the expected behavior. A sample of a JEST unit testing on the Frontend side is shown below.

```
test("initially renders default request description", () => {
  render(<Requests />);
  const defaultDescription = screen.getByText(/I hope you're well/i);
  expect(defaultDescription).toBeInTheDocument();
});
```

Figure 1: Unit Testing Sample with JEST

5.2.3. Integration Test

Integration testing aims to evaluate the collective performance of various system components by examining their interactions rather than scrutinizing the entire system as a single entity. The focus is on verifying if these components communicate effectively. For instance, when component A transmits message X to component B, integration testing assesses whether their interactions result in the expected behaviors and functionalities. For instance, the landing page for the user profile is composed of several components. Testing these components separately would be unit testing, but testing them together and how they interact is integration testing.
Verification between frontend and backend components, API endpoints and data flow will be ensured by integration tests. We will be writing integration tests to ensure the quality of our code.

5.2.4. End-to-End (E2E)

Condo Management website's workflow from frontend UI and backend services will be tested by E2E with real user interaction, stimulated by Cypress UI. The reason we decided to go with Cypress is because it provides an intuitive UI which makes it easier to learn and use. Furthermore, Cypress provides a powerful debugging tool, developers can easily inspect the DOM element if a test assertion fails and find out what went wrong. Supposedly, E2E is less frequently run compared to unit and integration tests, only if there are some major deployments such as pushing from dev to main. Broken user flows and critical bugs are more likely to be seen in this test as it provides confidence in overall functionality.

5.3. Sprint 1

5.3.1. Metrics

With SonarQube it is easy to keep track of our quality metrics. The metrics are separated in two distinct sections. The first section is overview metrics, which combine other metrics to give an overall perspective of the code quality. As such, we have selected reliability, maintainability and issue metrics. The second section is singular metrics, which are security, test coverage, code duplication, code size, code complexity and issues.

5.3.1.1. Reliability

SonarQube assesses the reliability of code by identifying potential bugs, code smells, and other issues that could lead to unexpected behavior or system failures. By analyzing reliability metrics, such as the number of bugs detected and their severity, SonarQube helps ensure that the codebase is robust and dependable. Furthermore, the use of tests ensures that the functionalities work consistently according to the requirements. Finally, as part of our core development structure, each function also goes through manual testing in our dev environment. This acts as a final safety net to ensure that the code does what it is supposed to do consistently.

5.3.1.2. Maintainability

SonarQube assesses code maintainability based on readability, complexity, and adherence to coding standards and best practices. Improving maintainability measures such as code duplication, cyclomatic complexity, and code coverage simplifies future maintenance, refactoring, and code review procedures.

5.3.1.3. Security

SonarQube contains security-focused rules and tests that help uncover vulnerabilities, security hotspots, and possible security hazards in code. In addition, SonarQube reduces security risks and strengthens the application's overall security posture by assessing security metrics such as the number and severity of security vulnerabilities.

5.3.1.4. Coverage

SonarQube assesses code coverage by calculating the proportion of code covered by automated tests, such as unit tests, integration tests, and end-to-end tests.
boosting code coverage measures assures that the codebase has been properly tested, lowering the possibility of undetected bugs and boosting overall product quality.

5.3.1.5. Duplication

Duplication is a metric that needs to be monitored correctly since multiple people work on the same project. For example, two developers work on two different features that have similarities. If there are no checks in place, the code can get bloated, and hard to read and maintain. Thus, SonarQube is used to help us reduce duplication by finding repeating code patterns or similar code parts throughout the source. Addressing code duplication improves maintainability, readability, and consistency, resulting in more efficient development and simpler issue fixes.

5.3.1.6. Size

SonarQube gives information on the size of the codebase, including the number of files, lines of code, and code churn over time. Monitoring code size metrics allows us to track project growth, identify areas of code bloat or excessive complexity, and improve resource allocation and project planning.

5.3.1.7. Complexity

SonarQube evaluates code complexity by measures such as cyclomatic complexity, nesting depth, and method length. Analyzing complexity metrics aids in identifying unnecessarily complicated code segments, potential failure spots, and places for rewriting or optimization to enhance code quality and maintenance.

5.3.1.8. Issues

SonarQube keeps track of issues found in the code, including but not limited to: code smells, potential bugs, bad practices, unnecessary hard coding etc…
This process is called linting and is a good tool to ensure consistent good practices across developers. As a consequence, issue tracking can increase the maintainability, readability, complexity and other software qualities. The number of issues should be kept as low as possible

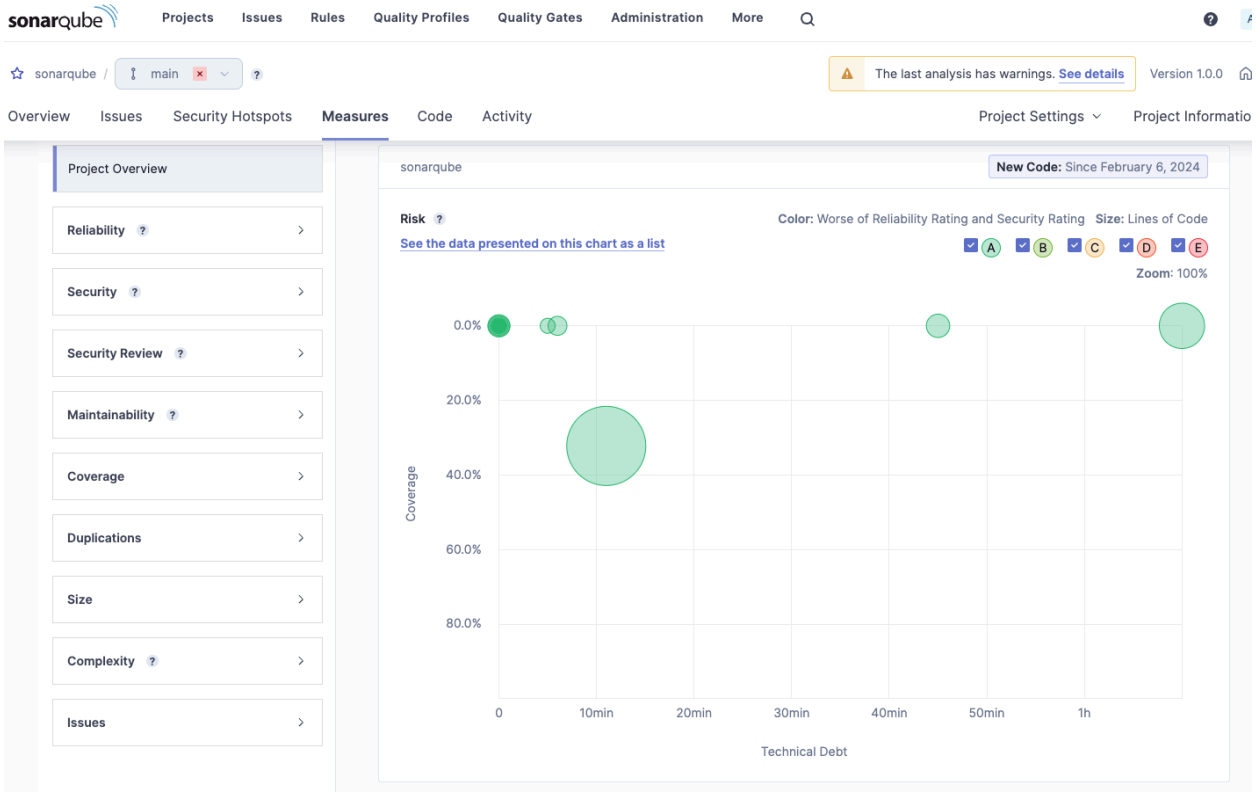


Figure 2: Measures of our code inside Sonarqube localhost server

5.3.2. Coverage

Software testing involves various metrics to assess the effectiveness of test cases in exercising the code. Four commonly used metrics are statement coverage, branch coverage, function coverage, and line coverage.

Statement coverage

measures the percentage of executable statements that have been executed at least once by the test cases. This metric provides insight into how well individual statements in the code have been exercised during testing.

Branch coverage

focuses on decision points in the code, such as if statements or loops. It assesses the percentage of decision branches that have been executed at least once, giving a more granular view of how well the code paths have been tested.

Function coverage

evaluates the percentage of functions or methods that have been executed by the test cases. This metric helps ensure that the different functional components of the software are adequately tested.

Line coverage

measures the percentage of lines of code that have been executed at least once. It provides a comprehensive overview of how well the entire codebase has been tested in terms of individual lines of code.

As asked in the requirements. We were able to pass 80% code coverage. On the frontend side, the statement coverage overall was 90.63%, the branch coverage was 80.76%, function coverage was 81.25% and lines were at 89.68% (as shown in the figure below. Therefore we succeeded at achieving over 80% for everything. We tested 13 different components and we had 49 different scenarios which all passed without any failure

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	90.63	80.76	81.25	89.68	
src	100	100	100	100	
App.tsx	100	100	100	100	
src/Components	96.55	83.33	80	100	
NavBar.tsx	96.55	83.33	80	100	44,76
src/Components/Authentication	94.78	83	100	93.75	
EmployeeRegistration.tsx	100	71.92	100	100	30-212
LogIn.tsx	100	91.66	100	100	50,115
LogOut.tsx	100	100	100	100	
SignUp.tsx	86.04	88.88	100	83.33	71-72,77-82
src/Components/Common	100	100	100	100	
AuthUtil.tsx	100	100	100	100	
InitialValues.ts	100	100	100	100	
LoadingScreen.tsx	100	100	100	100	
ValidationSchema.ts	100	100	100	100	
src/Components/UserProfile	80.61	70.73	65.21	76.31	
UserInformation.tsx	73.23	67.56	55.55	67.85	47,55-56,70-72,79-80,102-111,175-243
UserProperties.tsx	100	100	100	100	
UserRequests.tsx	100	100	100	100	
src/Pages	93.33	100	100	93.33	
EmployeeRegistrationLandingPage.tsx	100	100	100	100	

```
Test Suites: 13 passed, 13 total
Tests:       49 passed, 49 total
Snapshots:   0 total
Time:        20.045 s
```

Figure 3: Sample Frontend Percentage Coverage on the Terminal

One of JEST's advantages is that it can provide a report in .html format which creates a nice view for the user to see the coverage. A few snapshots of the report were taken below. This provides more details and insights to the person interested.

For the future, we plan to maintain the overall code coverage of at least 80%. This will reduce the number of undetected bugs and will give us more confidence in our application. Code coverage

enforcement can be done by configuring JEST to fail every time coverage drops below 80% which will prevent the PR from being merged.

Our goal for the next sprint is to keep improving the different types of coverage to cover more test and edge cases. We are aiming to be at around 90% and ensure that all of them lie within the green color.

On the backend side, we were able to have 94.99% statement coverage, 80.82% branch coverage,94.99% line coverage, with 100% coverage on the functions percentage. For the next sprint, on top of improving our file controller testing, we will try to maintain coverage above 80% as we add more features to our application.

```
---------------------|---------|----------|---------|---------|-------------------------
File                 | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------------|---------|----------|---------|---------|-------------------------
All files            |   94.99 |    80.82 |     100 |   94.99 |
 backend             |   96.29 |        0 |     100 |   96.29 |
  index.js           |   96.29 |        0 |     100 |   96.29 | 20
 backend/controller  |   94.22 |     80.7 |     100 |   94.22 |
  fileController.js  |   74.54 |    66.66 |     100 |   74.54 | 22-33,35-36
  userController.js  |   97.37 |    82.35 |     100 |   97.37 | 146-149,294-295,329-331
 backend/middleware  |   96.55 |    86.66 |     100 |   96.55 |
  authMiddleware.js  |   97.29 |       80 |     100 |   97.29 | 23
  errorMiddleware.js |   85.71 |       75 |     100 |   85.71 | 4-5
  minioMiddleware.js |     100 |      100 |     100 |     100 |
 backend/routes      |     100 |      100 |     100 |     100 |
  fileRoutes.js      |     100 |      100 |     100 |     100 |
  userRoutes.js      |     100 |      100 |     100 |     100 |
---------------------|---------|----------|---------|---------|-------------------------
```

Figure 4: Sample backend Percentage Coverage on the Terminal

All files

90.63% Statements 271/299 **80.76%** Branches 168/208 **81.25%** Functions 39/48 **89.68%** Lines 226/252

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
src		100%	10/10	100%	0/0	100%	1/1	100%	10/10
src/Components		96.55%	28/29	83.33%	10/12	80%	4/5	100%	24/24
src/Components/Authentication		94.78%	109/115	83%	127/153	100%	12/12	93.75%	90/96
src/Components/Common		100%	17/17	100%	1/1	100%	2/2	100%	16/16
src/Components/UserProfile		80.61%	79/98	70.73%	29/41	65.21%	15/23	76.31%	58/76
src/Pages		93.33%	28/30	100%	1/1	100%	5/5	93.33%	28/30

Figure 5: Main Page Report with JEST

All files src/Components/Authentication

94.78% Statements 109/115 **83%** Branches 127/153 **100%** Functions 12/12 **93.75%** Lines 90/96

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
EmployeeRegistration.tsx	100%	35/35	71.92%	41/57	100%	4/4	100%	28/28
LogIn.tsx	100%	31/31	91.66%	22/24	100%	3/3	100%	26/26
LogOut.tsx	100%	6/6	100%	0/0	100%	1/1	100%	6/6
SignUp.tsx	86.04%	37/43	88.88%	64/72	100%	4/4	83.33%	30/36

Figure 6: Detailed Statistics of the Authentication Folder and its Components

All files src/Components/Common

100% Statements 17/17 **100%** Branches 1/1 **100%** Functions 2/2 **100%** Lines 16/16

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
AuthUtil.tsx	100%	7/7	100%	1/1	100%	1/1	100%	6/6
InitialValues.ts	100%	2/2	100%	0/0	100%	0/0	100%	2/2
LoadingScreen.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
ValidationSchema.ts	100%	3/3	100%	0/0	100%	0/0	100%	3/3

Figure 7: Detailed Statistics of the Common Folder with Reusable files

All files src/Components/UserProfile

80.61% Statements 79/98 **70.73%** Branches 29/41 **65.21%** Functions 15/23 **76.31%** Lines 58/76

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
UserInformation.tsx	73.23%	52/71	67.56%	25/37	55.55%	10/18	67.85%	38/56
UserProperties.tsx	100%	18/18	100%	4/4	100%	4/4	100%	13/13
UserRequests.tsx	100%	9/9	100%	0/0	100%	1/1	100%	7/7

Figure 8: Detailed Statistics of the Common Folder with Reusable files

All files src/Pages

93.33% Statements 28/30 **100%** Branches 1/1 **100%** Functions 5/5 **93.33%** Lines 28/30

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
EmployeeRegistrationLandingPage.tsx	100%	4/4	100%	0/0	100%	1/1	100%	4/4
Hero.tsx	100%	4/4	100%	0/0	100%	1/1	100%	4/4
LogInLandingPage.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
RegistrationLandingPage.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
UserProfileLandingPage.tsx	83.33%	10/12	100%	1/1	100%	1/1	83.33%	10/12

Figure 9: Detailed Statistics of the Pages Folder with Landing Pages

All files src

100% Statements 10/10 **100%** Branches 0/0 **100%** Functions 1/1 **100%** Lines 10/10

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
App.tsx		100%	10/10	100%	0/0	100%	1/1	100%	10/10

Figure 10: Detailed Statistics of the Src folder (we tested the app itself)

5.3.3. Acceptance test

Sprint 1 Acceptance Tests:

Test ID	AT-1
Requirement	As a public user, I want to be able to create my profile with a profile picture, user name, contact email and phone number.
Acceptance Criteria	The system provides a user interface for public users to create their profiles.
Test Case	Expected Results
TC-1 Users sign up with unique username, email, phone number	The system will save the user record and direct users to the home page
TC-2 Users sign up with an existing username, email or phone number	The system will return a pop-up to indicate that users can't create a new account since the data are already in use
TC-3 Users sign up as an admin for a new company with a unique username, email, phone number	The system will save the user record and company record, establish the relationship and direct users to the home page
TC-3 Users sign up as an admin for a new company with existing username, email, phone number	The system will return a pop-up to indicate that users can't create a new account since the data are already in use

Test ID	AT-2

Requirement Tested	As a public user, I want to be able to log in and log out of my account.
Acceptance Criteria	The user has an existing account. The system provides a user interface for public users to log in and log out.
Test Case:	Expected Results
TC-1 User logs in with correct credentials	The system successfully authenticates the user's credentials and redirects the user to the designated landing page.
TC-2 User logs in with incorrect credentials	The system displays an error message indicating that the login credentials are incorrect.
TC-3 User logs out from account	The system securely logs the user out of their account and redirects the user to a designated logout confirmation page (or the login page).

Test ID	AT-3
Requirement Tested	As a public user, I want to be able to view and modify my account.
Acceptance Criteria	The user has an existing account. The system provides a user interface to display the user's profile.
Test Case:	Expected Results
TC-1 User edits profile and clicks "Save Changes"	The system successfully saves and displays the changes made by the user to their profile.
TC-2 User edits profile and clicks "Cancel"	The system discards the changes made by the user without saving and displays the existing profile information.

Sprint 2 Acceptance Tests:

Test ID	AT-4
Requirement Tested	As a public user, I want to be able to change my password.
Acceptance Criteria	The user has an existing account.

Test Case:	Expected Results
TC-1 User enters a new password	The system updates the user's password with the new one provided and invalidates the old password.
TC-2 User enters the same password	The system displays an error message indicating that the new password cannot be the same as the current password.
TC-3 User enters a password less than 8 characters long	The system displays an error message indicating the password requirement for minimum length was not met.

Test ID	AT-5
Requirement Tested	The system should be able to generate and send registration keys to unit owners and tenants.
Acceptance Criteria	Registration keys are unique.
Test Case:	Expected Results
TC-1 Generate Registration Key for Unit Owner	A unique registration key is generated for a unit owner.
TC-2 Generate Registration Key for Tenant	A unique registration key is generated for a tenant.
TC-3 Send Key to Registered User	The user receives the registration key via a communication channel in the system.
TC-4 Link Profile to Unit using Registration Key	The user gains access to relevant information and features to edit their unit.

Test ID	AT-6
Requirement Tested	As a condo management company, I want to be able to enter detailed information for each condo unit.
Acceptance Criteria	Given the company admin is logged in, they should be able to enter detailed information for each condo unit of their properties.
Test Case:	Expected Results

TC-1 Enter detailed information for Condo Unit	The system saves the detailed information of the new condo unit and displays a “Confirmation” message.

Test ID	AT-7
Requirement Tested	The system should allow the company admin to upload condo files to a property.
Acceptance Criteria	The condo file is in .pdf format.
Test Case:	Expected Results
TC-1 The user drags and drops the files in the dropbox.	The system displays the condo files in the dropbox.
TC-2 The user clicks the upload button in the dropbox and selects the files.	The system displays the condo files in the dropbox.

5.3.4. Test Results

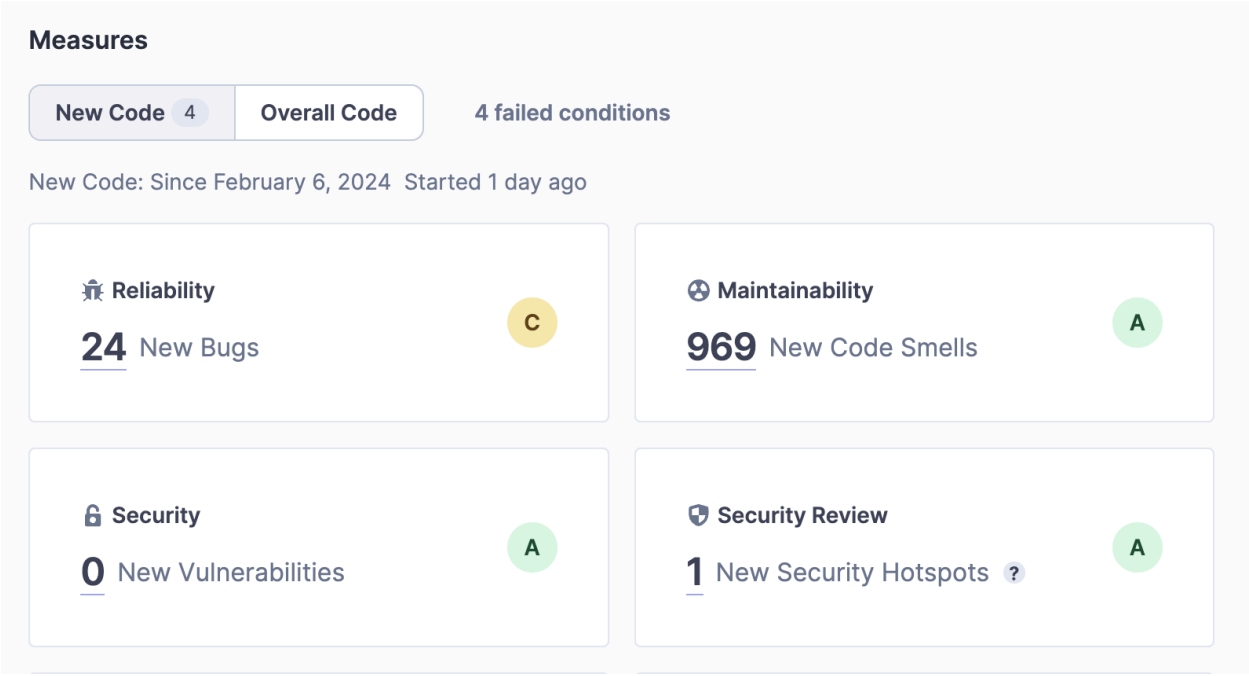


Figure 10: SonarQube Analytics Result for new Code from yesterday

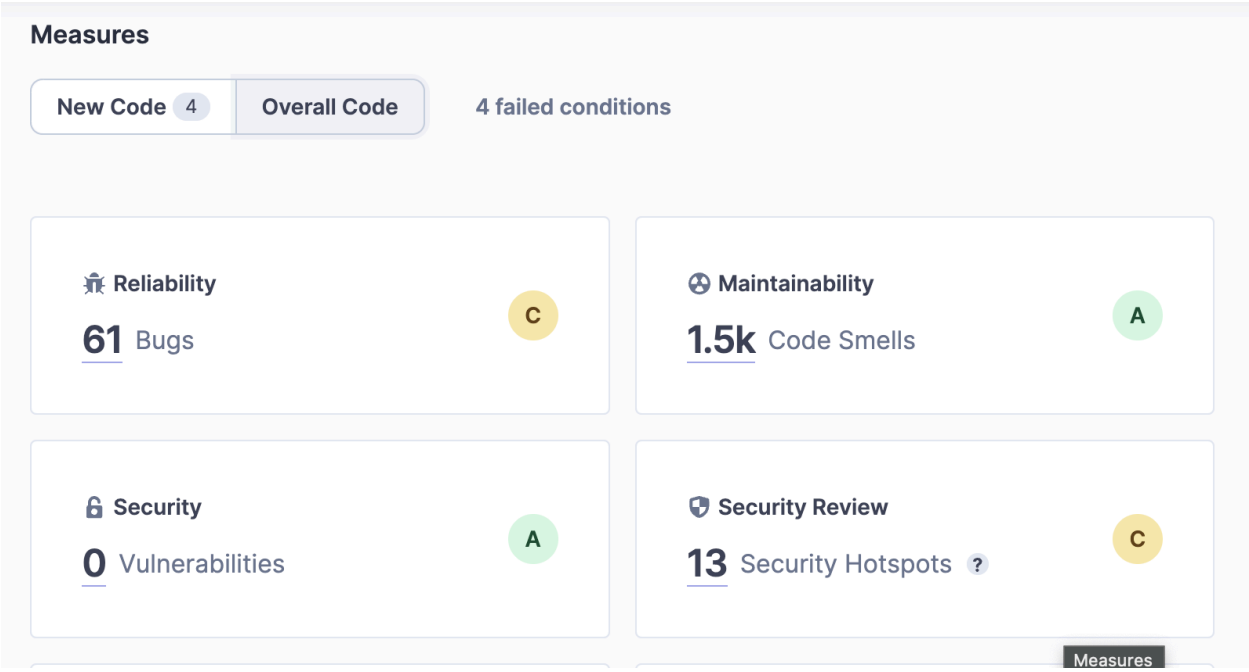


Figure 11: SonarQube Analytics Result for overall Code

5.4. Sprint 2

5.4.1. Metrics

With SonarQube it is easy to keep track of our quality metrics. The metrics are separated in two distinct sections. The first section is overview metrics, which combine other metrics to give an overall perspective of the code quality. As such, we have selected reliability, maintainability and issue metrics. The second section is singular metrics, which are security, test coverage, code duplication, code size, code complexity and issues.

5.4.1.1. Reliability

SonarQube assesses the reliability of code by identifying potential bugs, code smells, and other issues that could lead to unexpected behavior or system failures. By analyzing reliability metrics, such as the number of bugs detected and their severity, SonarQube helps ensure that the codebase is robust and dependable. Furthermore, the use of tests ensures that the functionalities work consistently according to the requirements. Finally, as part of our core development structure, each function also goes through manual testing in our dev environment. This acts as a final safety net to ensure that the code does what it is supposed to do consistently.

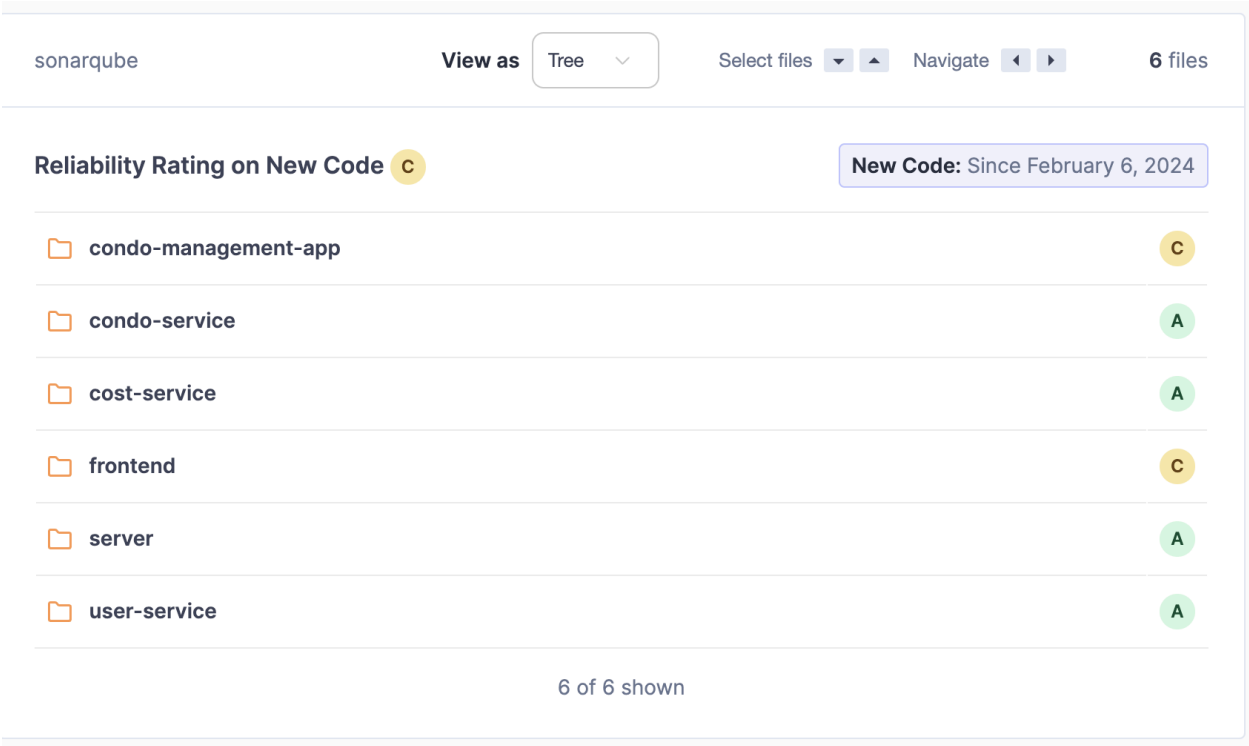


Figure 2: Reliability Report on Sprint 2

5.4.1.2. Maintainability

SonarQube assesses code maintainability based on readability, complexity, and adherence to coding standards and best practices. Improving maintainability measures such as code duplication, cyclomatic complexity, and code coverage simplifies future maintenance, refactoring, and code review procedures.

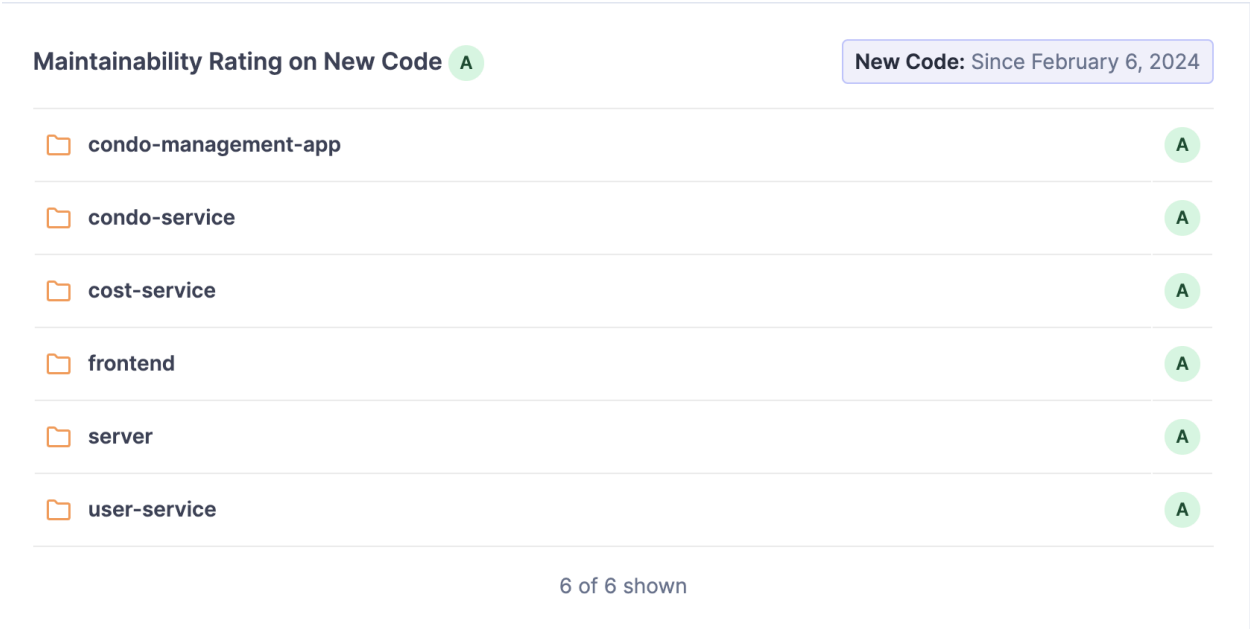


Figure 3: Maintainability report for sprint 2

5.4.1.3. Security

SonarQube contains security-focused rules and tests that help uncover vulnerabilities, security hotspots, and possible security hazards in code. In addition, SonarQube reduces security risks and strengthens the application's overall security posture by assessing security metrics such as the number and severity of security vulnerabilities.

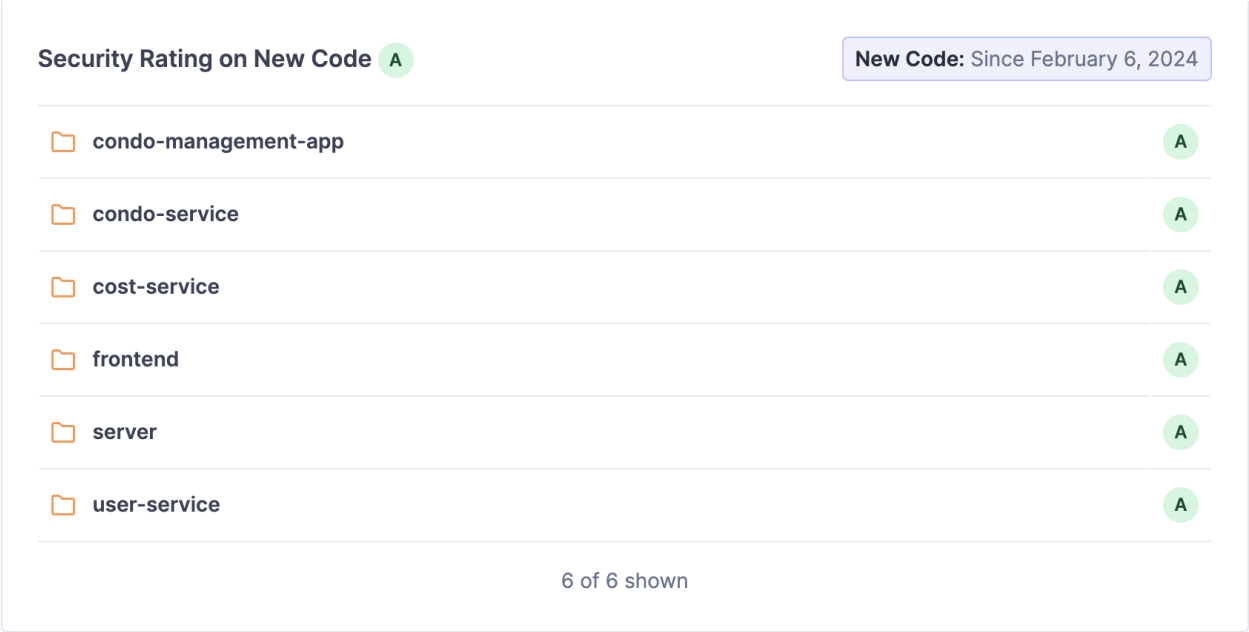


Figure 4: Security Report on Sprint 2

5.4.1.4. Coverage

SonarQube assesses code coverage by calculating the proportion of code covered by automated tests, such as unit tests, integration tests, and end-to-end tests.
boosting code coverage measures assures that the codebase has been properly tested, lowering the possibility of undetected bugs and boosting overall product quality.

5.4.1.5. Duplication

Duplication is a metric that needs to be monitored correctly since multiple people work on the same project. For example, two developers work on two different features that have similarities. If there are no checks in place, the code can get bloated, and hard to read and maintain. Thus, SonarQube is used to help us reduce duplication by finding repeating code patterns or similar code parts throughout the source. Addressing code duplication improves maintainability, readability, and consistency, resulting in more efficient development and simpler issue fixes.

Duplicated Lines (%) 16.8% See history

New Code: Since February 6, 2024

	Duplicated Lines (%)	Duplicated Lines
condo-management-app	0.0%	0
condo-service	86.5%	83
cost-service	86.5%	83
frontend	12.3%	1,126
server	20.2%	4,144
user-service	11.4%	237

6 of 6 shown

Figure 5: Duplication Report for sprint 2

5.4.1.6. Size

SonarQube gives information on the size of the codebase, including the number of files, lines of code, and code churn over time. Monitoring code size metrics allows us to track project growth, identify areas of code bloat or excessive complexity, and improve resource allocation and project planning.

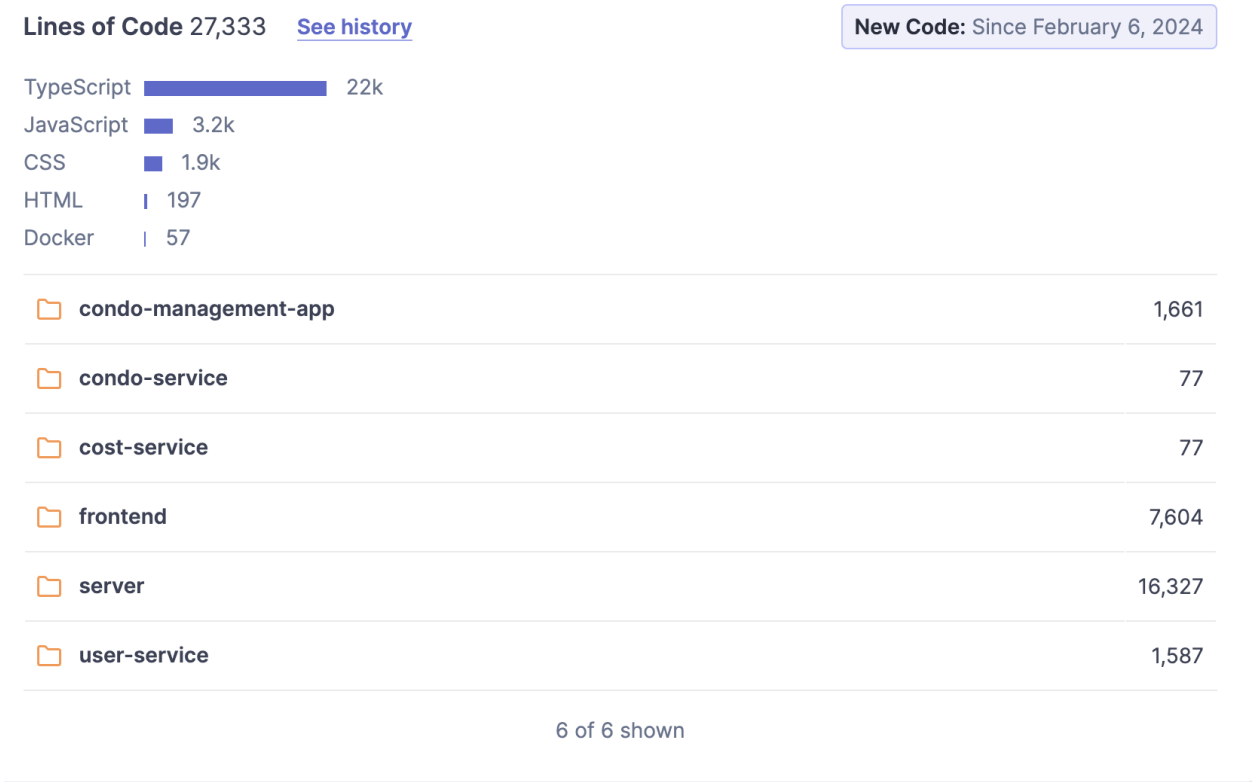


Figure 6: Line Of Code Report for Sprint 2

5.4.1.7. Complexity

SonarQube evaluates code complexity by measures such as cyclomatic complexity, nesting depth, and method length. Analyzing complexity metrics aids in identifying unnecessarily complicated code segments, potential failure spots, and places for rewriting or optimization to enhance code quality and maintenance.

sonarqube | View as Tree | Select files | Navigate | 6 files

Cyclomatic Complexity 3,031 See history | **New Code:** Since February 6, 2024

condo-management-app	163
condo-service	10
cost-service	10
frontend	666
server	2,006
user-service	176

6 of 6 shown

Figure 7: Cyclomatic Complexity Report for sprint 2

5.4.1.8. Issues

SonarQube keeps track of issues found in the code, including but not limited to: code smells, potential bugs, bad practices, unnecessary hard coding etc…
This process is called linting and is a good tool to ensure consistent good practices across developers. As a consequence, issue tracking can increase the maintainability, readability, complexity and other software qualities. The number of issues should be kept as low as possible

New Issues 405 | **New Code:** Since February 6, 2024

condo-management-app	122
condo-service	1
cost-service	1
frontend	87
server	186
user-service	8

6 of 6 shown

Figure 8: Issue Report for sprint 2

5.4.2. Coverage

Software testing involves various metrics to assess the effectiveness of test cases in exercising the code. Four commonly used metrics are statement coverage, branch coverage, function coverage, and line coverage.

Statement coverage

measures the percentage of executable statements that have been executed at least once by the test cases. This metric provides insight into how well individual statements in the code have been exercised during testing.

Branch coverage

focuses on decision points in the code, such as if statements or loops. It assesses the percentage of decision branches that have been executed at least once, giving a more granular view of how well the code paths have been tested.

Function coverage

evaluates the percentage of functions or methods that have been executed by the test cases. This metric helps ensure that the different functional components of the software are adequately tested.

Line coverage

measures the percentage of lines of code that have been executed at least once. It provides a comprehensive overview of how well the entire codebase has been tested in terms of individual lines of code.

As asked in the requirements. We were able to pass 80% code coverage in both sprint 1 and 2. On the frontend side, the statement coverage overall was 88.3%, the branch coverage was 81.67%, function coverage was 80..89% and lines were at 87.69% (as shown in the figure below. Therefore we succeeded at achieving over 80% for everything. We had 35 test suites (22 additional test cases from sprint 1) and we had 150 test cases(101 additional test cases from sprint 1) which all passed without any failure. These numbers are significant compared to sprint 1 and we managed to increase the branch coverage by around 1% which was hard to achieve. Although the other test metrics fell by 2-3%, we were still above 80%.

On the backend side, the statement coverage overall was 89.11%, the branch coverage was 81.98%, function coverage was 93.38% and lines were at 87.88%. We were able to achieve over 80% coverage for all of these criterias. In total, there were 19 test suites and 87 tests. All of the tests passed without exception. Compared to sprint 1, sprint 2's statement coverage decreased by around 6% which is due to the abundant lines of code added in this sprint for additional functionalities. Our branch coverage has improved by 1%, while function and line coverage decreased by 2% and 7 % respectively. That being said, all our test metrics exceeds the 80% mark despite the additional functionalities added in sprint 2.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	88.3	81.67	80.89	87.69	
src	97.29	80	83.33	97.05	
App.tsx	100	100	100	100	
api.ts	93.33	80	80	91.66	21
urls.ts	100	100	100	100	
src/Components	96.66	85.71	80	100	
NavBar.tsx	96.66	85.71	80	100	45,82
src/Components/Authentication	93.33	83.95	100	93.47	
EmployeeRegistration.tsx	100	71.92	100	100	33-216
LogIn.tsx	85.71	86.2	100	86.11	57-68,80
LogOut.tsx	100	100	100	100	
ResetPassword.tsx	100	96.42	100	100	125
SignUp.tsx	91.07	87.67	100	91.3	118-123
src/Components/Common	100	71.42	100	100	
AuthUtil.tsx	100	100	100	100	
InitialValues.ts	100	100	100	100	
List.tsx	100	66.66	100	100	17-20
LoadingScreen.tsx	100	100	100	100	
ValidationSchema.ts	100	100	100	100	
src/Components/CompanyDashboard	100	100	100	100	
IconCard.tsx	100	100	100	100	
src/Components/Condo	100	100	100	100	
Condo.tsx	100	100	100	100	
src/Components/CondoProfile	90.24	91.17	90.9	88.73	
CondoCreation.tsx	82.05	87.5	100	80.55	25-35,62-63
CondoInfoField.tsx	100	100	100	100	
CondoInfoForm.tsx	97.05	100	85.71	96.15	50
src/Components/Property	78.94	75	61.11	79.41	
CreateProperty.tsx	74.6	75	56.25	75	57,68-76,88,108-109,113-114,118-123,197
Property.tsx	100	100	100	100	
src/Components/PropertyProfile	81.03	90.47	65.51	79.79	
CondoFilesModal.tsx	73.43	81.81	58.82	72.72	54-63,89-115,170-199
PropertyInfoField.tsx	100	100	100	100	
PropertyInfoForm.tsx	87.5	100	66.66	84.37	49-50,54-55,110
src/Components/Requests	97.77	92.3	100	97.43	
KeyGeneration.tsx	97.77	92.3	100	97.43	96
src/Components/UserProfile	88	70.73	85.71	85.71	
UserInfoFields.tsx	100	100	100	100	
UserInformation.tsx	78.46	52	75	74	44,52-53,63-65,72-73,95-104
UserKeyRegister.tsx	95	100	75	93.33	38
UserProperties.tsx	100	100	100	100	
UserRequests.tsx	100	100	100	100	
src/Pages	82.18	67.85	76.92	81.69	
CompanyDashboardLandingPage.tsx	93.75	66.66	100	93.75	20
CondoCreationLandingPage.tsx	100	100	100	100	
CondoProfileLandingPage.tsx	100	100	100	100	
CreatePropertyLandingPage.tsx	100	100	100	100	
EmployeeRegistrationLandingPage.tsx	100	100	100	100	
EmployeeRequestsLandingPage.tsx	100	100	100	100	
Hero.tsx	100	100	100	100	
LogInLandingPage.tsx	100	100	100	100	
PropertyListLandingPage.tsx	96.66	75	100	96	50
PropertyProfileLandingPage.tsx	58.46	57.14	40	52.94	52-53,60-87,95-127,131-132,157-161
RegistrationLandingPage.tsx	100	100	100	100	
UserProfileLandingPage.tsx	84.61	100	100	84.61	19-21

Test Suites: 35 passed, 35 total
Tests: 150 passed, 150 total

Figure 9: Sample Frontend Percentage Coverage on the Terminal

One of JEST's advantages is that it can provide a report in .html format which creates a nice view for the user to see the coverage. A few snapshots of the report were taken below. This provides more details and insights to the person interested.

For the future, we plan to maintain the overall code coverage of at least 80%. This will reduce the number of undetected bugs and will give us more confidence in our application. Code coverage enforcement can be done by configuring JEST to fail every time coverage drops below 80% which will prevent the PR from being merged.

Sprint 3 Testing Plan for Code Coverage

Our plan for Sprint 3 is to achieve over 90% coverage in statement as well as line coverage. For Branch and Function coverage, since these are the hardest, we aim to achieve 85%. We will continue to test every component we work on and enforce each team member to include tests in the Pull Requests to ensure that we will always be above 80% coverage in all metrics. We will continue to use JEST as a testing library as it is effective and detailed.

1

Frontend Reports:

88.3% Statements 800/906 **81.67%** Branches 312/382 **80.89%** Functions 127/157 **87.69%** Lines 677/772

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
src	97.29%	36/37	80%	4/5	83.33%	5/6	97.05%	33/34
src/Components	96.66%	29/30	85.71%	12/14	80%	4/5	100%	25/25
src/Components/Authentication	93.33%	154/165	83.95%	157/187	100%	15/15	93.47%	129/138
src/Components/Common	100%	43/43	71.42%	5/7	100%	9/9	100%	38/38
src/Components/CompanyDashboard	100%	8/8	100%	0/0	100%	1/1	100%	5/5
src/Components/Condo	100%	5/5	100%	0/0	100%	1/1	100%	4/4
src/Components/CondoProfile	90.24%	74/82	91.17%	31/34	90.9%	10/11	88.73%	63/71
src/Components/Property	78.94%	60/76	75%	24/32	61.11%	11/18	79.41%	54/68
src/Components/PropertyProfile	81.03%	94/116	90.47%	19/21	65.51%	19/29	79.79%	79/99
src/Components/Requests	97.77%	44/45	92.3%	12/13	100%	8/8	97.43%	38/39
src/Components/UserProfile	88%	110/125	70.73%	29/41	85.71%	24/28	85.71%	84/98
src/Pages	82.18%	143/174	67.85%	19/28	76.92%	20/26	81.69%	125/153

Figure 10: Main Page Report with JEST

93.33% Statements 154/165 **83.95%** Branches 157/187 **100%** Functions 15/15 **93.47%** Lines 129/138

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
EmployeeRegistration.tsx	100%	35/35	71.92%	41/57	100%	4/4	100%	28/28
LogIn.tsx	85.71%	36/42	86.2%	25/29	100%	3/3	86.11%	31/36
LogOut.tsx	100%	6/6	100%	0/0	100%	1/1	100%	6/6
ResetPassword.tsx	100%	26/26	96.42%	27/28	100%	3/3	100%	22/22
SignUp.tsx	91.07%	51/56	87.67%	64/73	100%	4/4	91.3%	42/46

Figure 11: Detailed Statistics of the Authentication Folder and its Components

100% Statements 43/43 **71.42%** Branches 5/7 **100%** Functions 9/9 **100%** Lines 38/38

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
AuthUtil.tsx	100%	7/7	100%	1/1	100%	1/1	100%	6/6
InitialValues.ts	100%	5/5	100%	0/0	100%	0/0	100%	5/5
List.tsx	100%	20/20	66.66%	4/6	100%	7/7	100%	16/16
LoadingScreen.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
ValidationSchema.ts	100%	6/6	100%	0/0	100%	0/0	100%	6/6

Figure 12: Detailed Statistics of the Common Folder with Reusable files

File	Statements		Branches		Functions		Lines	
UserInfoFields.tsx	100%	13/13	100%	12/12	100%	7/7	100%	13/13
UserInformation.tsx	78.46%	51/65	52%	13/25	75%	9/12	74%	37/50
UserKeyRegister.tsx	95%	19/20	100%	0/0	75%	3/4	93.33%	14/15
UserProperties.tsx	100%	18/18	100%	4/4	100%	4/4	100%	13/13
UserRequests.tsx	100%	9/9	100%	0/0	100%	1/1	100%	7/7

Figure 13: Detailed Statistics of the UserProfile

File	Statements		Branches		Functions		Lines	
CompanyDashboardLandingPage.tsx	93.75%	15/16	66.66%	2/3	100%	1/1	93.75%	15/16
CondoCreationLandingPage.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
CondoProfileLandingPage.tsx	100%	17/17	100%	2/2	100%	2/2	100%	15/15
CreatePropertyLandingPage.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
EmployeeRegistrationLandingPage.tsx	100%	4/4	100%	0/0	100%	1/1	100%	4/4
EmployeeRequestsLandingPage.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
Hero.tsx	100%	4/4	100%	0/0	100%	1/1	100%	4/4
LogInLandingPage.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
PropertyListLandingPage.tsx	96.66%	29/30	75%	6/8	100%	5/5	96%	24/25
PropertyProfileLandingPage.tsx	58.46%	38/65	57.14%	8/14	40%	4/10	52.94%	27/51
RegistrationLandingPage.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
UserProfileLandingPage.tsx	84.61%	11/13	100%	1/1	100%	1/1	84.61%	11/13

Figure 14: Detailed Statistics of the Pages Folder with Landing Pages

97.29% Statements 36/37 **80%** Branches 4/5 **83.33%** Functions 5/6 **97.05%** Lines 33/34

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
App.tsx	100%	18/18	100%	0/0	100%	1/1	100%	18/18
api.ts	93.33%	14/15	80%	4/5	80%	4/5	91.66%	11/12
urls.ts	100%	4/4	100%	0/0	100%	0/0	100%	4/4

Figure 15: Detailed Statistics of the Src Folder(we tested the app itself)

90.24% Statements 74/82 **91.17%** Branches 31/34 **90.9%** Functions 10/11 **88.73%** Lines 63/71

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
CondoCreation.tsx		82.05%	32/39	87.5%	21/24	100%	3/3	80.55%	29/36
CondoInfoField.tsx		100%	9/9	100%	4/4	100%	1/1	100%	9/9
CondoInfoForm.tsx		97.05%	33/34	100%	6/6	85.71%	6/7	96.15%	25/26

Figure 16: Detailed Statistics of the CondoProfile

File		Statements		Branches		Functions		Lines	
CreateProperty.tsx		74.6%	47/63	75%	24/32	56.25%	9/16	75%	42/56
Property.tsx		100%	13/13	100%	0/0	100%	2/2	100%	12/12

Figure 17: Detailed Statistics of the Property

File		Statements		Branches		Functions		Lines	
CondoFilesModal.tsx		73.43%	47/64	81.81%	9/11	58.82%	10/17	72.72%	40/55
PropertyInfoField.tsx		100%	12/12	100%	2/2	100%	3/3	100%	12/12
PropertyInfoForm.tsx		87.5%	35/40	100%	8/8	66.66%	6/9	84.37%	27/32

Figure 18: Detailed Statistics of the PropertyProfile

The rest of the folders with no details only have 1 file, so it is already represented in the “All Files” section.

Backend Reports:

All files

89.11% Statements 630/707 **81.98%** Branches 91/111 **93.38%** Functions 111/121 **87.88%** Lines 602/685

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
company/base		90.21%	83/92	90.66%	5/6	83.33%	16/18	90%	81/90
companyEmployee/base		95.45%	42/44	85.45%	9/11	100%	6/6	95.23%	40/42
condoUnit/base		85.71%	78/91	80.88%	17/18	88.88%	16/18	85.39%	76/89
file/base		82.14%	53/56	86.31%	17/19	87.76%	4/6	81.48%	44/54
locker/base		86.74%	38/43	87.27%	11/11	100%	6/6	100%	41/41
parkingSpot/base		86.74%	40/43	87.27%	11/11	100%	6/6	100%	41/41
property/base		88.78%	95/107	96.27%	10/11	87.27%	19/22	86.19%	98/105
registrationKey/base		90.69%	39/43	100%	3/3	100%	6/6	85.6%	38/41
role/base		86.74%	39/43	100%	3/3	100%	4/6	95.6%	39/41
user/base		80.26%	49/92	16.66%	5/6	88.88%	18/18	100%	90/90
userCondo/base		86.74%	38/43	92.2%	9/11	100%	6/6	100%	41/41

Figure 19: Main Page Report of the Backend with JEST

File		Statements		Branches		Functions		Lines	
company/base		90.21%	83/92	90.66%	5/6	83.33%	16/18	90%	81/90
companyEmployee/base		95.45%	42/44	85.45%	9/11	100%	6/6	95.23%	40/42

Figure 20: Detailed Statistics of company

File		Statements		Branches		Functions		Lines	
condoUnit/base		85.71%	78/91	80.88%	17/18	88.88%	16/18	85.39%	76/89
userCondo/base		86.74%	38/43	92.2%	9/11	100%	6/6	100%	41/41

Figure 21: Detailed Statistics of condo

File		Statements		Branches		Functions		Lines	
user/base		80.26%	49/92	16.66%	5/6	88.88%	18/18	100%	90/90
userCondo/base		86.74%	38/43	92.2%	9/11	100%	6/6	100%	41/41

Figure 22: Detailed Statistics of User

The rest of the folders with no details only have 1 file, so it is already represented in the "All Files" section.

5.4.3. Acceptance Tests

<u>Sprint 3 Acceptance Tests</u>

Test ID	AT-1
Requirement	As a condo owner, I want to have a dashboard of properties (general information, personal profile, condo information, financial status, remaining balance of condo fee payments, status of submitted request)
Acceptance Criteria	User is logged into their account.
Test Case	Expected Results
TC-1 User clicks on "My Profile" from the dashboard page.	The user navigates to their personal profile.
TC-2 User clicks on "My Condos" from the dashboard page.	The user navigates to the list of condos under their ownership.
TC-3 User clicks on "My Finances" from the dashboard page.	The user navigates to the financial status and payments section of their account.
TC-4 User clicks on "My Requests" from the dashboard page.	The user navigates to the list of their submitted requests.

Test ID	AT-2
Requirement Tested	As a condo owner, I want to be able to submit requests.
Acceptance Criteria	The user is logged into their account. The system provides a user interface to compose a new request.
Test Case:	Expected Results
TC-1 Condo owner selects a request type and submits a request.	Request is successfully submitted, and the details are stored in the system. Assigned personnel receive notification of the new request for it to be addressed promptly.
TC-2 Condo owner does not select a request type and submits a request.	The request is not successfully submitted and the user is notified to select a request type from the dropdown for a valid submission.

Test ID	AT-3
Requirement Tested	As a condo management employee, I want to be able to see the assigned requests.
Acceptance Criteria	The user is logged into their account. The system provides a user interface to display all assigned requests.
Test Case:	Expected Results
TC-1 Employee accesses the assigned requests page	The system displays a list of all requests assigned to them by type with relevant details available.
TC-2 Employee updates the status of a request	The request status is successfully updated in the system and notifications are sent to the corresponding condo owner.

Test ID	AT-4
Requirement Tested	Users should have a notification page.
Acceptance Criteria	Users can access a notification page from the navigation bar. The notification page should

	display a list of recent activities related to submitted or assigned requests.
Test Case:	Expected Results
TC-1 Condo owner navigates to the notification page.	The system displays a list of recent activities related to their submitted requests with updated status.
TC-2 Condo employee navigates to the notification page.	The system displays a list of recent activities related to assigned requests.
TC-3 Manager navigates to the notification page.	The system displays a list of recent activities related to assigned requests.

Test ID	AT-5
Requirement Tested	Condo management companies should be able to set up different roles for employees
Acceptance Criteria	The user is logged into their account. The system provides a user interface to set up role management for employees.
Test Case:	Expected Results
TC-1 Company admin assigns the role “Manager” for an employee.	The role is successfully assigned to the corresponding employee, and permissions are configured to allow access to relevant features and data.
TC-2 Company admin assigns the role “Financial Manager” for an employee.	The role is successfully assigned to the corresponding employee, and permissions are configured to allow access to relevant features and data.
TC-3 Company admin assigns the role “Operator” for an employee.	The role is successfully assigned to the corresponding employee, and permissions are configured to allow access to relevant features and data.

Test ID	AT-6

Requirement Tested	The system should allow condo management companies to be able to delete employee profiles from the system to remove individuals who are no longer employed within the company.
Acceptance Criteria	User is logged into their account. The system provides a list of employee profiles registered in the system.
Test Case:	Expected Results
TC-1 Company searches for and selects profile they wish to delete	The system locates and displays the desired employee profile for deletion.
TC-2 Company initiates the deletion action and confirms the action when prompted.	The system displays a confirmation prompt to verify the deletion action. Upon confirmation, the selected employee profile is permanently deleted from the system.
TC-3 Company verifies the deletion by confirming that the employee profile is no longer accessible in the system	The selected employee's profile is successfully removed from the system and it is no longer listed in the employee registry list. All associated data and permissions are also deleted.

Test ID	AT-7
Requirement Tested	The system should allow companies to delete a property profile in the contents of their property list.
Acceptance Criteria	The user is logged into their account. The system displays the user interface for the list of properties under the company's ownership.
Test Case:	Expected Results
TC-1 Company selects a property from the properties list and initiates the deletion action.	The system displays a confirmation prompt to confirm the deletion. Upon confirmation, the property entry is successfully removed from the system.

Test ID	AT-8
Requirement Tested	The system should allow condo users to view condo profiles of the condos they own.

Acceptance Criteria	The user is logged into their account. The system displays the condo owner's dashboard.
Test Case:	Expected Results
TC-1 Condo owner navigates to the "My Condos" section from the dashboard.	The system displays the list of condos under the condo owner's ownership.
TC-2 Condo owner selects a specific condo from the condos list to view its profile.	The system navigates to and displays the selected condo profile and displays its details, such as size, occupant name, and condo fee.

5.4.4. Test Results

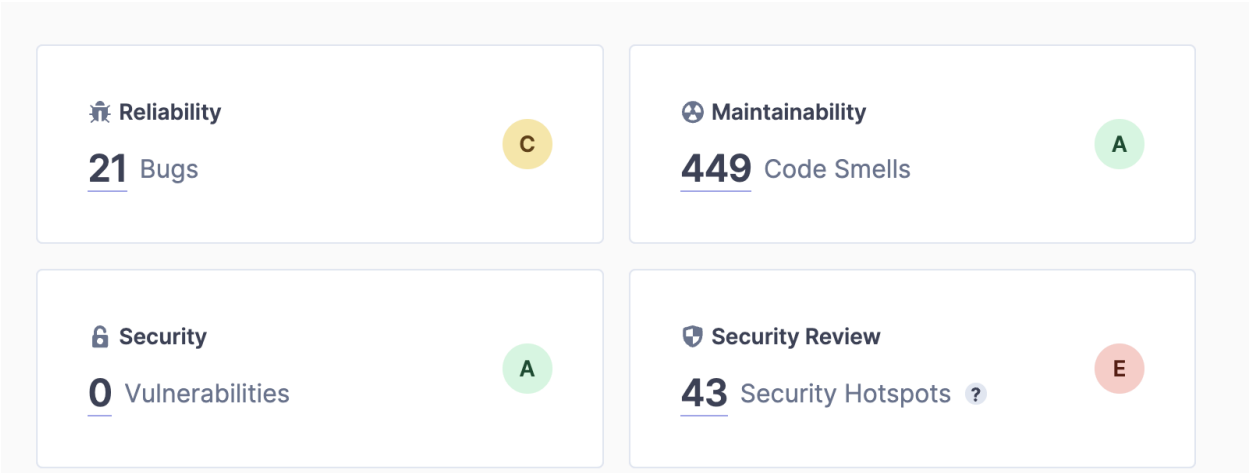


Figure 23: SonarQube Analytics Result for overall Code

5.5. Sprint 3

5.5.1. Metrics

With SonarQube it is easy to keep track of our quality metrics. The metrics are separated in two distinct sections. The first section is overview metrics, which combine other metrics to give an overall perspective of the code quality. As such, we have selected reliability, maintainability and issue metrics. The second section is singular metrics, which are security, test coverage, code duplication, code size, code complexity and issues.

5.5.1.1. Reliability

SonarQube assesses the reliability of code by identifying potential bugs, code smells, and other issues that could lead to unexpected behavior or system failures. By analyzing reliability metrics, such as the number of bugs detected and their severity, SonarQube helps ensure that the codebase is robust and dependable. Furthermore, the use of tests ensures that the functionalities work consistently according to the requirements. Finally, as part of our core development structure, each function also goes through manual testing in our dev environment. This acts as a final safety net to ensure that the code does what it is supposed to do consistently.

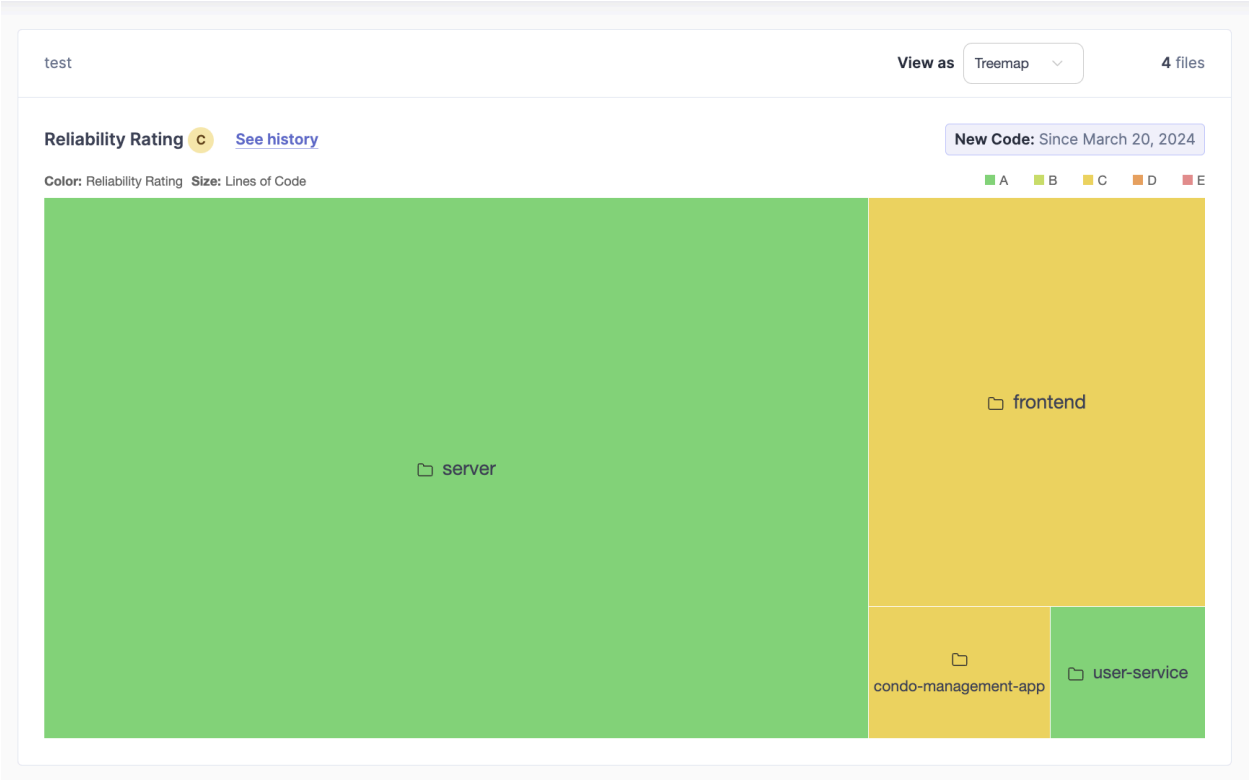


Figure 2: Reliability Report on Sprint 3

5.5.1.2. Maintainability

SonarQube assesses code maintainability based on readability, complexity, and adherence to coding standards and best practices. Improving maintainability measures such as code duplication, cyclomatic complexity, and code coverage simplifies future maintenance, refactoring, and code review procedures.

Figure 3: Maintainability report for sprint 3

5.5.1.3. Security

SonarQube contains security-focused rules and tests that help uncover vulnerabilities, security hotspots, and possible security hazards in code. In addition, SonarQube reduces security risks and strengthens the application's overall security posture by assessing security metrics such as the number and severity of security vulnerabilities.

Figure 4: Security Report on Sprint 3

5.5.1.4. Coverage

SonarQube assesses code coverage by calculating the proportion of code covered by automated tests, such as unit tests, integration tests, and end-to-end tests.
boosting code coverage measures assures that the codebase has been properly tested, lowering the possibility of undetected bugs and boosting overall product quality.

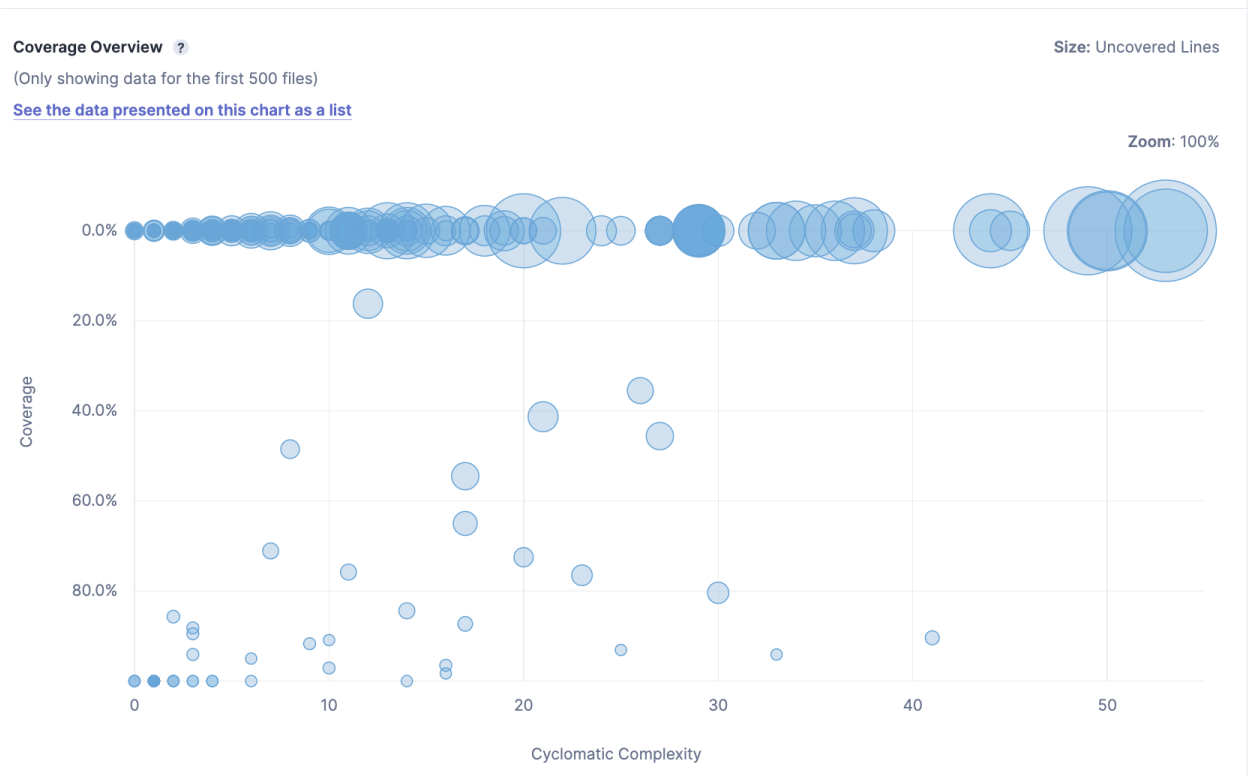


Figure 5: Coverage Report on Sprint 3

5.5.1.5. Duplication

Duplication is a metric that needs to be monitored correctly since multiple people work on the same project. For example, two developers work on two different features that have similarities. If there are no checks in place, the code can get bloated, and hard to read and maintain. Thus, SonarQube is used to help us reduce duplication by finding repeating code patterns or similar code parts throughout the source. Addressing code duplication improves maintainability, readability, and consistency, resulting in more efficient development and simpler issue fixes.

	Duplicated Lines (%)	Duplicated Lines
condo-management-app	0.0%	0
frontend	9.5%	1,076
server	19.1%	7,010
user-service	9.2%	174

Figure 6: Duplication Report for sprint 3

5.5.1.6. Size

SonarQube gives information on the size of the codebase, including the number of files, lines of code, and code churn over time. Monitoring code size metrics allows us to track project growth, identify areas of code bloat or excessive complexity, and improve resource allocation and project planning.

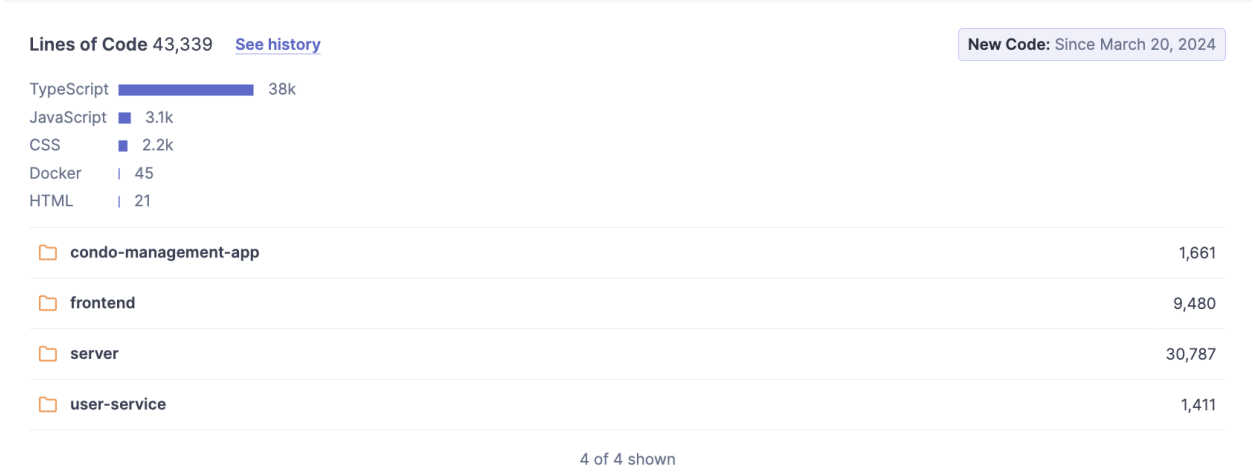


Figure 7: Line Of Code Report for Sprint 3

5.5.1.7. Complexity

SonarQube evaluates code complexity by measures such as cyclomatic complexity, nesting depth, and method length. Analyzing complexity metrics aids in identifying unnecessarily complicated code segments, potential failure spots, and places for rewriting or optimization to enhance code quality and maintenance.

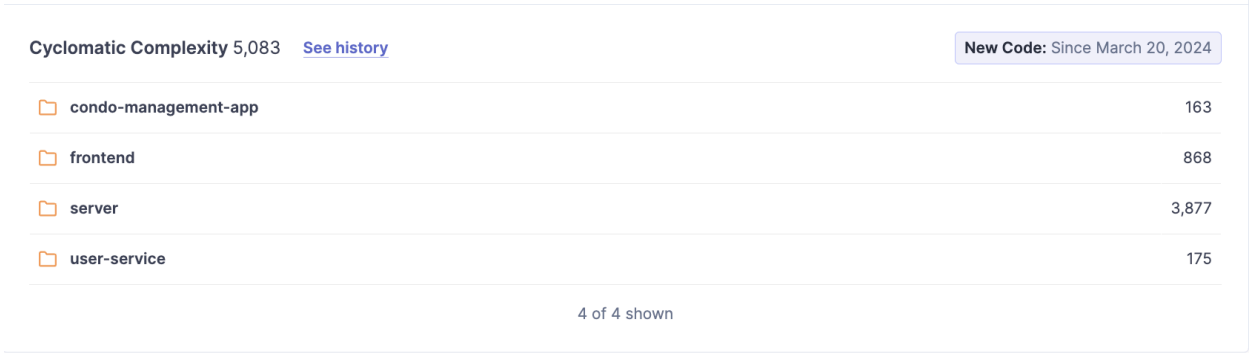


Figure 8: Cyclomatic Complexity Report for sprint 3

Cognitive Complexity 962 See history	New Code: Since March 20, 2024
condo-management-app	56
frontend	304
server	543
user-service	59

4 of 4 shown

Figure 9: Cognitive Complexity Report for sprint 3

5.5.1.8. Issues

SonarQube keeps track of issues found in the code, including but not limited to: code smells, potential bugs, bad practices, unnecessary hard coding etc…
This process is called linting and is a good tool to ensure consistent good practices across developers. As a consequence, issue tracking can increase the maintainability, readability, complexity and other software qualities. The number of issues should be kept as low as possible

Issues 655 See history	New Code: Since March 20, 2024
condo-management-app	122
frontend	211
server	312
user-service	10

4 of 4 shown

Figure 10: Issue Report for sprint 3

5.5.2. Coverage

Software testing involves various metrics to assess the effectiveness of test cases in exercising the code. Four commonly used metrics are statement coverage, branch coverage, function coverage, and line coverage.

Statement coverage

measures the percentage of executable statements that have been executed at least once by the test cases. This metric provides insight into how well individual statements in the code have been exercised during testing.

Branch coverage

focuses on decision points in the code, such as if statements or loops. It assesses the percentage of decision branches that have been executed at least once, giving a more granular view of how well the code paths have been tested.

Function coverage
evaluates the percentage of functions or methods that have been executed by the test cases. This metric helps ensure that the different functional components of the software are adequately tested.
Line coverage
measures the percentage of lines of code that have been executed at least once. It provides a comprehensive overview of how well the entire codebase has been tested in terms of individual lines of code.

As asked in the requirements. We were able to pass 80% code coverage in both sprint 1, 2 and 3. On the frontend side, the statement coverage overall was 87.06%, the branch coverage was 84.3%, function coverage was 85% and lines were at 88% (as shown in the figure below. Therefore we succeeded at achieving over 80% for everything. We had 45 test suites (10 additional test cases from sprint 2) and we had 182 test cases(32 additional test cases from sprint 2) which all passed without any failure. These numbers are significant compared to sprint 2 and we managed to increase the branch coverage by around 2% which was hard to achieve. Although the other test metrics fell by 2-3%, we were still above 80%.

On the backend side, the statement coverage overall was 90.09%, the branch coverage was 88.59%, function coverage was 95.02% and lines were at 92.41%. We were able to achieve over 80% coverage for all of these criterias. In total, there were 28 test suites and 251 tests. We have added a significant amount of tests compared to previous sprints to account for adding the notification feature on our website. All of the tests passed without exception. Compared to sprint 2, sprint 3's statement coverage increased by around 2% which is due to the rigorous testing methods executed in this sprint for the additional functionalities. Our branch coverage has improved by 6%, while function and line coverage increased by 11% and 4 % respectively. That being said, all our test metrics exceed the 80% mark despite the additional functionalities added in sprint 3.

One of JEST's advantages is that it can provide a report in .html format which creates a nice view for the user to see the coverage. A few snapshots of the report were taken below. This provides more details and insights to the person interested.
For the future, we plan to maintain the overall code coverage of at least 80%. This will reduce the number of undetected bugs and will give us more confidence in our application. Code coverage enforcement can be done by configuring JEST to fail every time coverage drops below 80% which will prevent the PR from being merged.

Sprint 4 Testing Plan for Code Coverage
Our plan for Sprint 4 is to achieve over 90% coverage in statement as well as line coverage. For Branch and Function coverage, since these are the hardest, we aim to achieve 85%. We will continue to test every component we work on and enforce each team member to include tests in the Pull Requests to ensure that we will always be above 80% coverage in all metrics. We will continue to use JEST as a testing library as it is effective and detailed.

1

Frontend Reports:

All files

87.06% Statements 104479/1200 **84.3%** Branches 16871/200 **85%** Functions 85/100 **88%** Lines 88/100

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
src/Components/Common		100%	27/45	100%	6/7	100%	9/9	100%	40/40
src/Components/CompanyDashboard		100%	8/8	100%	0/0	100%	1/1	100%	5/5
src/Components/Condo		100%	5/5	100%	0/0	100%	1/1	100%	4/4
src/Components		96.96%	32/33	85.71%	30/35	80%	4/5	100%	28/28
src/Components/Authentication		93.52%	159/170	89.13%	164/184	100%	9/15	93.7%	134/143
src/Components/CondoProfile		90.24%	74/82	91.17%	31/34	90.9%	10/11	88.73%	63/71
src/Components/UserProfile		84.82%	123/145	71.92%	41/57	88.88%	24/27	83.18%	94/113
src/Components/Property		82%	59/76	73.33%	22/30	61.11%	11/18	77.94%	53/68
src/Components/Requests		81.48%	132/162	50%	28/56	71.87%	23/32	80.85%	114/141
src		80%	89/93	84%	12/15	90%	21/22	62.66%	47/75
src/Components/PropertyProfile		69.69%	92/132	84%	21/25	83%	17/31	69.91%	79/113
src/Pages		66.08%	226/342	83%	69/72	84%	51/64	68.51%	198/289

Figure 11: Main Page Report with JEST

All files **src/Components/Authentication**

93.52% Statements 159/170 **89.13%** Branches 164/184 **100%** Functions 15/15 **93.7%** Lines 134/143

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
EmployeeRegistration.tsx		100%	40/40	88.88%	48/54	100%	4/4	100%	33/33
LogIn.tsx		85.71%	36/42	86.2%	25/29	100%	3/3	86.11%	31/36
LogOut.tsx		100%	6/6	100%	0/0	100%	1/1	100%	6/6
ResetPassword.tsx		100%	26/26	96.42%	27/28	100%	3/3	100%	22/22
SignUp.tsx		91.07%	51/56	87.67%	64/73	100%	4/4	91.3%	42/46

Figure 12: Detailed Statistics of the Authentication Folder and its Components

All files **src/Components/Common**

100% Statements 45/45 **71.42%** Branches 5/7 **100%** Functions 9/9 **100%** Lines 40/40

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
AuthUtil.tsx		100%	7/7	100%	1/1	100%	1/1	100%	6/6
InitialValues.ts		100%	6/6	100%	0/0	100%	0/0	100%	6/6
List.tsx		100%	20/20	66.66%	4/6	100%	7/7	100%	16/16
LoadingScreen.tsx		100%	5/5	100%	0/0	100%	1/1	100%	5/5
ValidationSchema.ts		100%	7/7	100%	0/0	100%	0/0	100%	7/7

Figure 13: Detailed Statistics of the Common Folder with Reusable files

All files **src/Components/UserProfile**

84.82% Statements 123/145 **71.92%** Branches 41/57 **88.88%** Functions 24/27 **83.18%** Lines 94/113

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements			Branches			Functions			Lines	
UserInfoFields.tsx		100%	14/14		100%	14/14		100%	7/7		100%	14/14
UserInformation.tsx		79.68%	51/64		58.33%	14/24		72.72%	8/11		76%	38/50
UserKeyRegister.tsx		77.5%	31/40		60%	9/15		100%	4/4		75.86%	22/29
UserProperties.tsx		100%	18/18		100%	4/4		100%	4/4		100%	13/13
UserRequests.tsx		100%	9/9		100%	0/0		100%	1/1		100%	7/7

Figure 14: Detailed Statistics of the UserProfile

90.24% Statements 74/82 **91.17%** Branches 31/34 **90.9%** Functions 10/11 **88.73%** Lines 63/71

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements			Branches			Functions			Lines	
CondoCreation.tsx		82.05%	32/39		87.5%	21/24		100%	3/3		80.55%	29/36
CondoInfoField.tsx		100%	9/9		100%	4/4		100%	1/1		100%	9/9
CondoInfoForm.tsx		97.05%	33/34		100%	6/6		85.71%	6/7		96.15%	25/26

Figure 15: Detailed Statistics of the CondoProfile

The rest of the folders with no details have very few files, so it is already represented in the “All Files” section.

Backend Reports:

All files

90.09% Statements 1028/1141 **88.59%** Branches 163/184 **95.02%** Functions 191/201 **92.41%** Lines 1023/1107

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
commonFacility/base	91.52%	54/59	100%	4/4	100%	10/10	91.22%	52/57
company/base	81.48%	88/108	100%	7/7	81.81%	18/22	81.13%	86/106
companyEmployee/base	88.33%	53/60	91.66%	11/12	90%	9/10	86.21%	50/58
condoUnit/base	93.46%	100/107	84.21%	16/19	86.36%	19/22	88.57%	93/105
file/base	96.42%	54/56	89.47%	17/19	100%	6/6	96.29%	52/54
forum/base	91.52%	54/59	100%	4/4	100%	10/10	91.22%	52/57
health/base	100%	10/10	100%	1/1	100%	3/3	100%	10/10
locker/base	95.34%	41/43	90.91%	10/11	100%	6/6	95.12%	39/41
notification/base	93.02%	40/43	90.91%	10/11	100%	6/6	97.56%	40/41
parkingSpot/base	95.34%	41/43	100%	11/11	100%	6/6	95.12%	39/41
post/base	95.34%	41/43	100%	11/11	100%	6/6	95.12%	39/41
property/base	80.67%	101/123	83.33%	10/12	84.61%	22/26	79.33%	96/121
registrationKey/base	95.45%	42/44	100%	3/3	100%	6/6	95.23%	40/42
request/base	89.65%	52/58	91.66%	22/24	100%	10/10	76.78%	43/56
reservation/base	95.34%	41/43	100%	11/11	100%	6/6	95.12%	39/41
role/base	95.34%	41/43	100%	3/3	100%	6/6	95.12%	39/41
user/base	80.12%	125/156	100%	10/10	88.23%	30/34	79.87%	123/154
userCondo/base	95.34%	41/43	100%	11/11	100%	6/6	95.12%	39/41

Figure 16: Main Page Report of the Backend with JEST

File	Statements		Branches		Functions		Lines	
company/base	90.21%	83/92	90.66%	5/6	83.33%	16/18	90%	81/90
companyEmployee/base	95.45%	42/44	85.45%	9/11	100%	6/6	95.23%	40/42

Figure 17: Detailed Statistics of company

File	Statements		Branches		Functions		Lines	
condoUnit/base	85.71%	78/91	80.88%	17/18	88.88%	16/18	85.39%	76/89
userCondo/base	86.74%	38/43	92.2%	9/11	100%	6/6	100%	41/41

Figure 18: Detailed Statistics of condo

File	Statements		Branches		Functions		Lines	
user/base	80.26%	49/92	16.66%	5/6	88.88%	18/18	100%	90/90
userCondo/base	86.74%	38/43	92.2%	9/11	100%	6/6	100%	41/41

Figure 19: Detailed Statistics of User

The rest of the folders with no details only have 1 file, so it is already represented in the “All Files” section.

5.5.3. Acceptance Test

<u>Sprint 4 Acceptance Tests</u>

Test ID	AT-1
Requirement	As a condo owner, I want to be able to reserve common facilities in a calendar-like interface.
Acceptance Criteria	User is logged into their account.
Test Case	Expected Results
TC-1 User clicks on “Common Facilities Calendar” from the dashboard page.	The user navigates to a calendar-like interface for common facilities.

Test ID	AT-2
Requirement Tested	As a condo management company, I want to be able to set up a common facility which requires reservations.
Acceptance Criteria	The user is logged into their account. The system provides a user interface to reserve a common facility.
Test Case:	Expected Results
TC-1 Company selects a date and time for a common facility that is available.	The reservation is successfully submitted, and the details are stored in the system.
TC-2 Company selects a date and time for a common facility that is unavailable.	The reservation is not successfully submitted and the user is notified to select a date and time from the dropdown for a valid submission.

Test ID	AT-3
Requirement Tested	As a rental user, I want to be able to see the availability of common facilities.
Acceptance Criteria	The user is logged into their account.
Test Case:	Expected Results
TC-1 User clicks on “Common Facilities Calendar” from the dashboard page.	The user navigates to a calendar-like interface for common facilities.

Test ID	AT-4
Requirement Tested	As a rental user, I want to be able to modify or cancel my existing reservation for a common facility.
Acceptance Criteria	Users can access the list of their reservations from the Common Facilities Calendar Page. The Reservations List page should display a list of all reservations related to the user's submissions.
Test Case:	Expected Results
TC-1 User deletes an existing reservation.	The system displays a confirmation prompt to confirm the deletion. Upon confirmation, the reservation entry is successfully removed from the system.
TC-2 User modifies an existing reservation with valid availability.	The system displays a confirmation prompt to confirm the modification. Upon confirmation, the reservation entry is successfully modified in the system.
TC-3 User deletes an existing reservation with invalid availability.	The modification is not successfully submitted and the user is notified to select a date and time from the dropdown for a valid submission.

Test ID	AT-5
Requirement Tested	As a condo management company, I want to be able to disable the availability of a common facility.
Acceptance Criteria	The user is logged into their account. The system provides a user interface to set the status of common facilities.
Test Case:	Expected Results
TC-1 Company assigns the status "Mark as Closed" to a common facility.	The status of the common facility is successfully updated in the system as closed, and permissions are configured to disallow users access to this facility for reservations.

TC-1 Company assigns the status “Mark as Open” to a common facility.	The status of the common facility is successfully updated in the system as open, and permissions are configured to allow users access to this facility for reservations.

Test ID	AT-6
Requirement Tested	As a condo management company, I want to be able to enter the condo fee per square foot, per parking spot for a condo unit.
Acceptance Criteria	The user is logged into their account. The system displays the user interface for setting condo fees for a condo unit.
Test Case:	Expected Results
TC-1 Company enters the fee per square foot.	The fee per square foot entry is successfully entered in the system.
TC-1 Company enters the fee per parking spot.	The fee per parking spot entry is successfully entered in the system.

Test ID	AT-7
Requirement Tested	As a condo management company, I want to be able to enter the cost for each operation.
Acceptance Criteria	User is logged into their account. The system provides a user interface to add cost for operations.
Test Case:	Expected Results
TC-1 Company selects an operation and assigns a cost to it.	The cost of the operation is successfully submitted and stored in the system.

Test ID	AT-8
Requirement Tested	As a condo owner, I want to be able to see the calculated condo fee for each unit.

Acceptance Criteria	The user is logged into their account. The system displays the condo owner's dashboard.
Test Case:	Expected Results
TC-1 Condo owner navigates to the "My Condos" section from the dashboard.	The system displays the list of condos under the condo owner's ownership.
TC-2 Condo owner selects a specific condo from the condos list to view its profile.	The system navigates to and displays the selected condo profile and displays its details, such as size, occupant name, and condo fee.

Test ID	AT-9
Requirement Tested	As a condo management company, I want to be able to see a record of the operational budget, which includes the total condo fees collected from condo owners.
Acceptance Criteria	The user is logged into their account. The system displays the company's dashboard dashboard.
Test Case:	Expected Results
TC-1 Company navigates to the "Operational Budget Report" section from the dashboard.	The system displays the report for the condo fees collected from condo owners.

Test ID	AT-10
Requirement Tested	As a condo management company, I want to be able to see an annual report.
Acceptance Criteria	The user is logged into their account. The system displays the company's dashboard.
Test Case:	Expected Results
TC-1 Company navigates to the "Annual Report" section from the dashboard.	The system displays the annual report, including details of the revenue, expenses and profits of the company.

5.5.4. Test Result

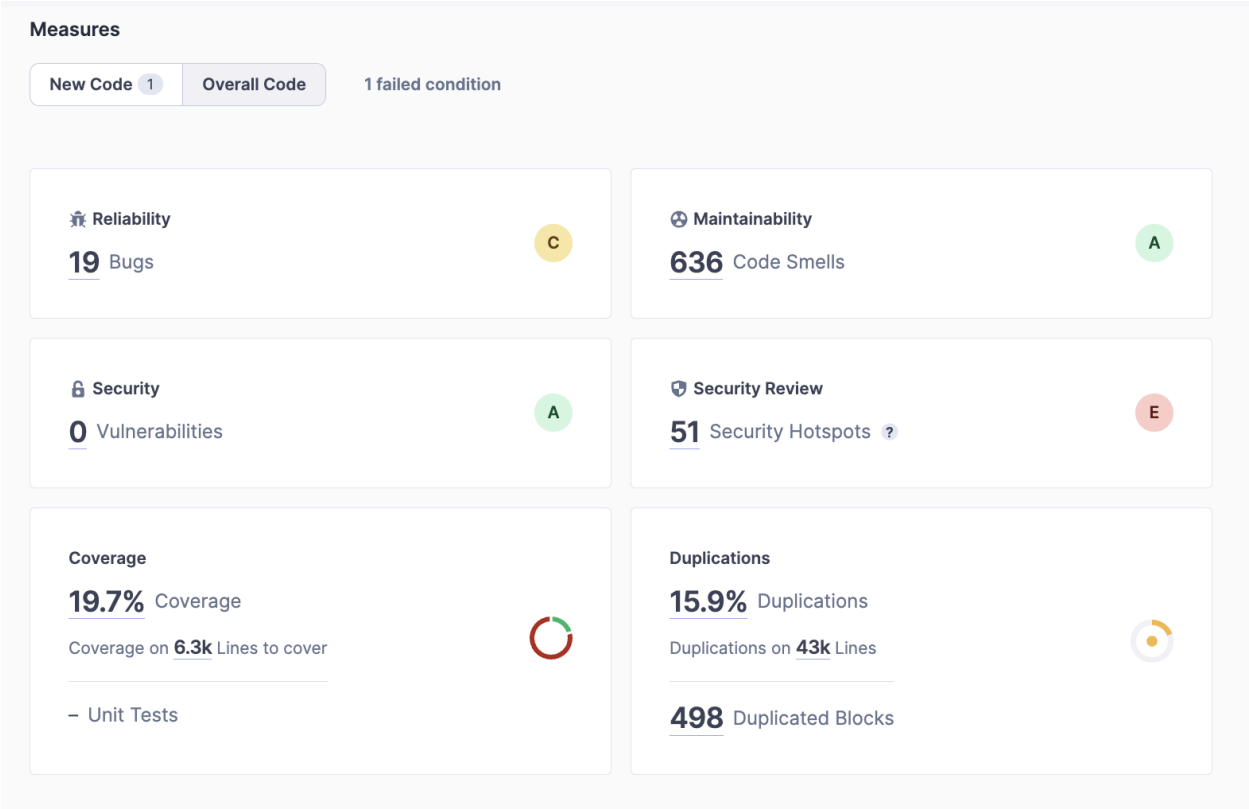


Figure 20: SonarQube Analytics Result for overall Code

5.6. Sprint 4

5.6.1. Metrics

With SonarQube it is easy to keep track of our quality metrics. The metrics are separated in two distinct sections. The first section is overview metrics, which combine other metrics to give an overall perspective of the code quality. As such, we have selected reliability, maintainability and issue metrics. The second section is singular metrics, which are security, test coverage, code duplication, code size, code complexity and issues.

5.6.1.1. Reliability

SonarQube assesses the reliability of code by identifying potential bugs, code smells, and other issues that could lead to unexpected behavior or system failures. By analyzing reliability metrics, such as the number of bugs detected and their severity, SonarQube helps ensure that the codebase is robust and dependable. Furthermore, the use of tests ensures that the functionalities work consistently according to the requirements. Finally, as part of our core development structure, each function also goes through manual

testing in our dev environment. This acts as a final safety net to ensure that the code does what it is supposed to do consistently.

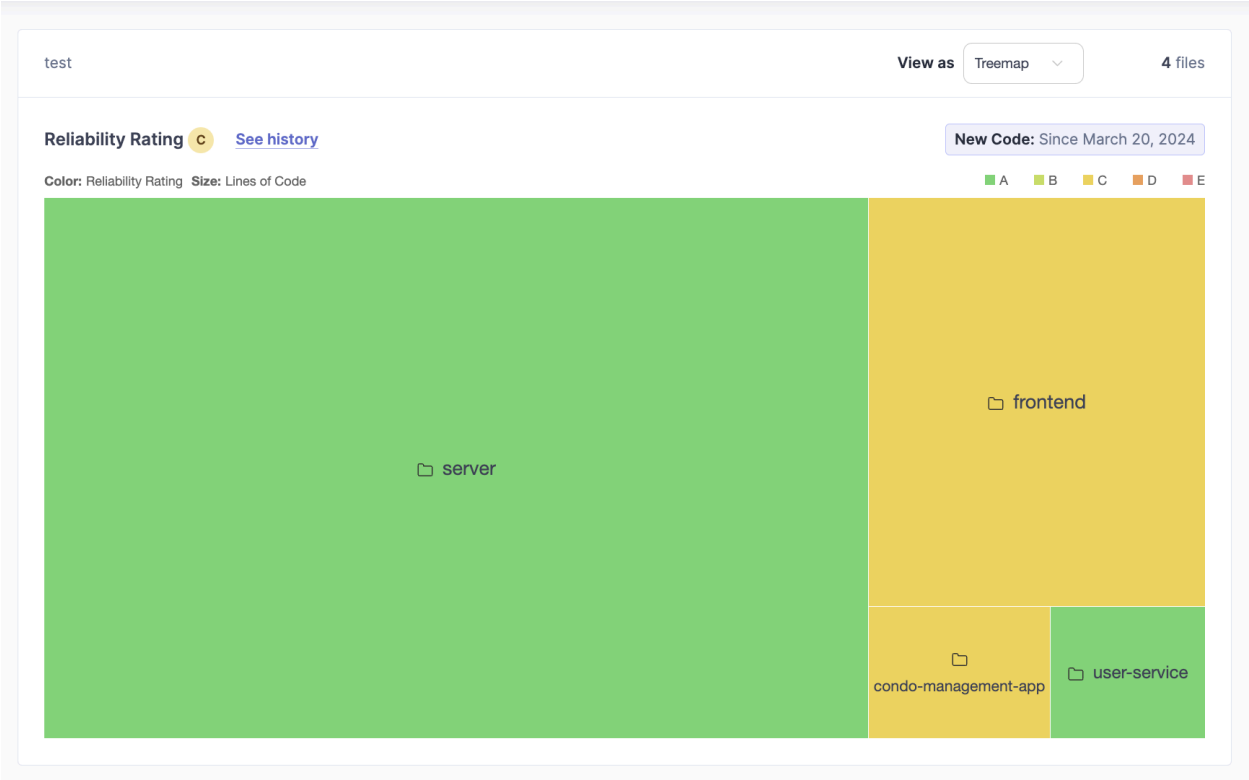


Figure 2: Reliability Report on Sprint 4

5.6.1.2. Maintainability

SonarQube assesses code maintainability based on readability, complexity, and adherence to coding standards and best practices. Improving maintainability measures such as code duplication, cyclomatic complexity, and code coverage simplifies future maintenance, refactoring, and code review procedures.

Figure 3: Maintainability report for sprint 4

5.6.1.3. Security

SonarQube contains security-focused rules and tests that help uncover vulnerabilities, security hotspots, and possible security hazards in code. In addition, SonarQube reduces security risks and strengthens the application's overall security posture by assessing security metrics such as the number and severity of security vulnerabilities.

Figure 4: Security Report on Sprint 4

5.6.1.4. Coverage

SonarQube assesses code coverage by calculating the proportion of code covered by automated tests, such as unit tests, integration tests, and end-to-end tests.
boosting code coverage measures assures that the codebase has been properly tested, lowering the possibility of undetected bugs and boosting overall product quality.

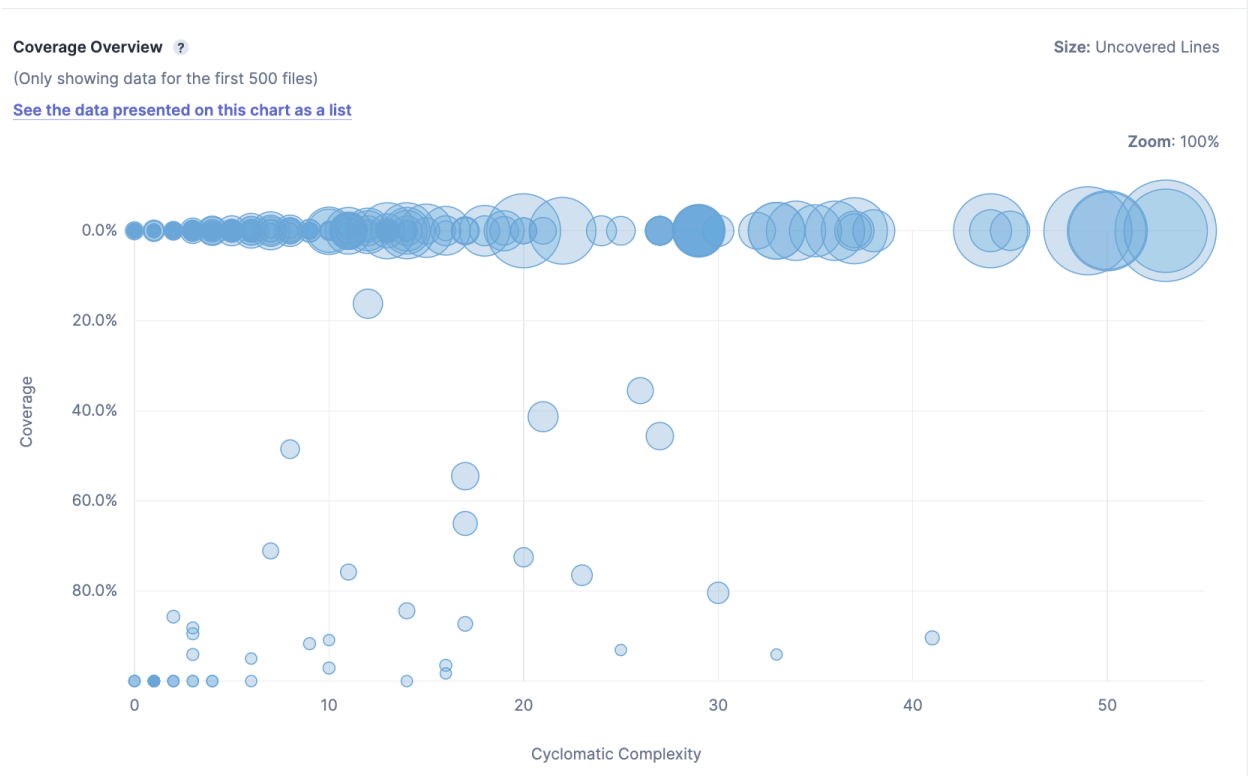


Figure 5: Coverage Report on Sprint 4

5.6.1.5. Duplication

Duplication is a metric that needs to be monitored correctly since multiple people work on the same project. For example, two developers work on two different features that have similarities. If there are no checks in place, the code can get bloated, and hard to read and maintain. Thus, SonarQube is used to help us reduce duplication by finding repeating code patterns or similar code parts throughout the source. Addressing code duplication improves maintainability, readability, and consistency, resulting in more efficient development and simpler issue fixes.

	Duplicated Lines (%)	Duplicated Lines
condo-management-app	0.0%	0
frontend	9.5%	1,076
server	19.1%	7,010
user-service	9.2%	174

Figure 6: Duplication Report for sprint 4

5.6.1.6. Size

SonarQube gives information on the size of the codebase, including the number of files, lines of code, and code churn over time. Monitoring code size metrics allows us to track project growth, identify areas of code bloat or excessive complexity, and improve resource allocation and project planning.

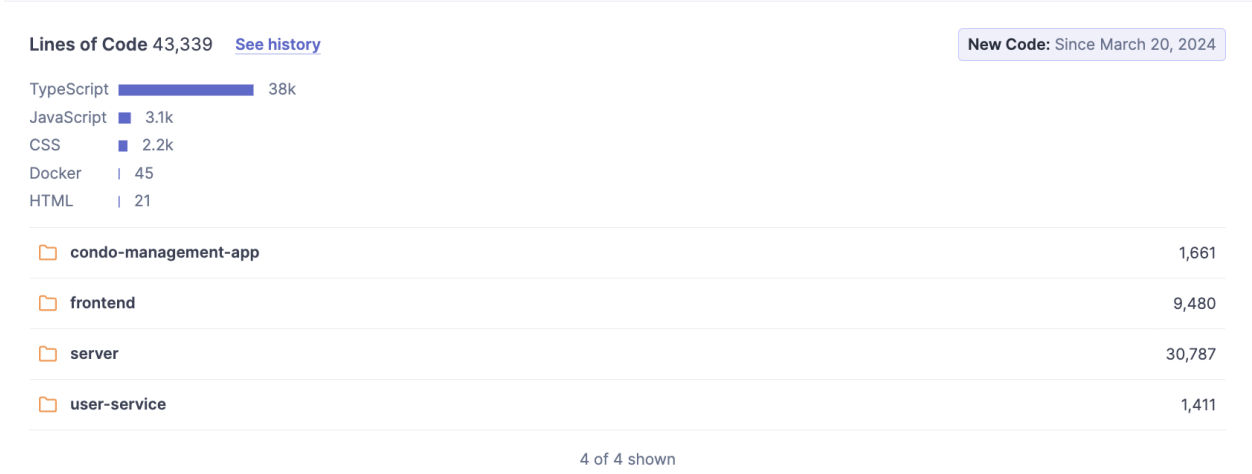


Figure 7: Line Of Code Report for Sprint 4

5.6.1.7. Complexity

SonarQube evaluates code complexity by measures such as cyclomatic complexity, nesting depth, and method length. Analyzing complexity metrics aids in identifying unnecessarily complicated code segments, potential failure spots, and places for rewriting or optimization to enhance code quality and maintenance.

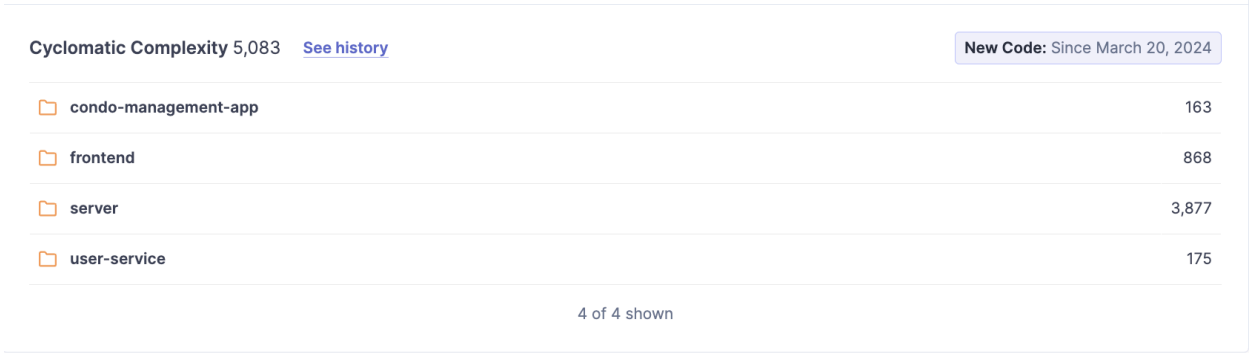


Figure 8: Cyclomatic Complexity Report for sprint 4

Cognitive Complexity 962 See history	New Code: Since March 20, 2024
condo-management-app	56
frontend	304
server	543
user-service	59

4 of 4 shown

Figure 9: Cognitive Complexity Report for sprint 4

5.6.1.8. Issues

SonarQube keeps track of issues found in the code, including but not limited to: code smells, potential bugs, bad practices, unnecessary hard coding etc…
This process is called linting and is a good tool to ensure consistent good practices across developers. As a consequence, issue tracking can increase the maintainability, readability, complexity and other software qualities. The number of issues should be kept as low as possible

Issues 655 See history	New Code: Since March 20, 2024
condo-management-app	122
frontend	211
server	312
user-service	10

4 of 4 shown

Figure 10: Issue Report for sprint 4

5.6.2. Coverage

Software testing involves various metrics to assess the effectiveness of test cases in exercising the code. Four commonly used metrics are statement coverage, branch coverage, function coverage, and line coverage.

Statement coverage

measures the percentage of executable statements that have been executed at least once by the test cases. This metric provides insight into how well individual statements in the code have been exercised during testing.

Branch coverage

focuses on decision points in the code, such as if statements or loops. It assesses the percentage of decision branches that have been executed at least once, giving a more granular view of how well the code paths have been tested.

Function coverage
evaluates the percentage of functions or methods that have been executed by the test cases. This metric helps ensure that the different functional components of the software are adequately tested.
Line coverage
measures the percentage of lines of code that have been executed at least once. It provides a comprehensive overview of how well the entire codebase has been tested in terms of individual lines of code.

As asked in the requirements. We were able to pass 80% code coverage in both sprint 1, 2 and 3. On the frontend side, the statement coverage overall was 81.14%, the branch coverage was 82.82%, function coverage was 84.62% and lines were at 80.85% (as shown in the figure below. Therefore we succeeded at achieving over 80% for everything. We had 59 test suites (15 additional test cases from sprint 3) and we had 209 test cases(27 additional test cases from sprint 3) which all passed without any failure. These numbers are significant compared to sprint 3. However, we can see that percentages fell due to how complicated the system started to get, but we are still above 80% in all metrics.

On the backend side, the statement coverage overall was 97.48%, the branch coverage was 92.07%, function coverage was 92.69% and lines were at 94.64%. We were able to achieve over 80% coverage for all of these criterias. In total, there were 31 test suites compared to the 28 test suites from sprint 4 and 259 test suites this sprint compared to 251 testsuites last sprint. We have added a significant amount of tests compared to previous sprints to account for adding the financial system on our website. All of the test suites passed without exception. Compared to sprint 3, sprint 4's statement coverage increased by around 2% which is due to the rigorous testing methods executed in this sprint for the additional functionalities. Our branch coverage has improved by 7%, while function and line coverage increased by 3% and 2 % respectively. That being said, all our test metrics exceed the 80% mark despite the additional functionalities added in sprint 4.

One of JEST's advantages is that it can provide a report in .html format which creates a nice view for the user to see the coverage. A few snapshots of the report were taken below. This provides more details and insights to the person interested.
For the future, we plan to maintain the overall code coverage of at least 80%. This will reduce the number of undetected bugs and will give us more confidence in our application. Code coverage enforcement can be done by configuring JEST to fail every time coverage drops below 80% which will prevent the PR from being merged.

Sprint 5 Testing Plan for Code Coverage
Our plan for Sprint 5 is to maintain over 80% coverage in statement as well as line coverage. For Branch and Function coverage, since these are the hardest, we aim to achieve 85%. We will continue to test every component we work on and enforce each team member to include tests in the Pull Requests to ensure that we will always be above 80% coverage in all metrics. We will continue to use JEST as a testing library as it is effective and detailed.

1

Frontend Reports:

All files

81.14% Statements 1368/1686 **82.82%** Branches 535/646 **84.62%** Functions 275/325 **80.85%** Lines 1136/1405

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
src	84.7%	84/110	81.61%	22/26	80.35%	29/34	83.33%	61/81
src/Components	97.05%	33/34	84.61%	33/39	80%	4/5	100%	29/29
src/Components/Authentication	93.52%	159/170	89.13%	164/184	100%	15/15	93.7%	134/143
src/Components/Common	100%	48/48	71.42%	5/7	100%	9/9	100%	43/43
src/Components/CompanyDashboard	100%	8/8	100%	0/0	100%	1/1	100%	5/5
src/Components/Condo	100%	5/5	100%	0/0	100%	1/1	100%	4/4
src/Components/CondoProfile	83.83%	99/124	84.04%	32/41	85%	17/20	88.88%	84/108
src/Components/Costs	51.72%	60/116	88.88%	35/45	82%	15/25	74.82%	64/92
src/Components/Property	87.63%	59/76	90%	27/30	81.11%	16/18	87.94%	53/68
src/Components/PropertyProfile	70.45%	93/132	84%	21/25	74.83%	17/31	70.79%	80/113
src/Components/Requests	82.09%	133/162	78.78%	47/56	75%	28/32	81.56%	115/141
src/Components/Reservations	82.82%	176/198	72.58%	51/58	81.81%	41/44	73.13%	155/166
src/Components/UserProfile	83.56%	122/146	68.42%	39/57	77.77%	21/27	82.45%	94/114
src/Pages	70.86%	253/357	84.71%	73/78	52.38%	33/63	72.14%	215/298

Figure 11: Main Page Report with JEST

All files src/Components/Authentication

93.52% Statements 159/170 **89.13%** Branches 164/184 **100%** Functions 15/15 **93.7%** Lines 134/143

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
EmployeeRegistration.tsx	100%	40/40	88.88%	48/54	100%	4/4	100%	33/33
LogIn.tsx	85.71%	36/42	86.2%	25/29	100%	3/3	86.11%	31/36
LogOut.tsx	100%	6/6	100%	0/0	100%	1/1	100%	6/6
ResetPassword.tsx	100%	26/26	96.42%	27/28	100%	3/3	100%	22/22
SignUp.tsx	91.07%	51/56	87.67%	64/73	100%	4/4	91.3%	42/46

Figure 12: Detailed Statistics of the Authentication Folder and its Components

All files src/Components/Common

100% Statements 45/45 **71.42%** Branches 5/7 **100%** Functions 9/9 **100%** Lines 40/40

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
AuthUtil.tsx	100%	7/7	100%	1/1	100%	1/1	100%	6/6
InitialValues.ts	100%	6/6	100%	0/0	100%	0/0	100%	6/6
List.tsx	100%	20/20	66.66%	4/6	100%	7/7	100%	16/16
LoadingScreen.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
ValidationSchema.ts	100%	7/7	100%	0/0	100%	0/0	100%	7/7

Figure 13: Detailed Statistics of the Common Folder with Reusable files

All files **src/Components/UserProfile**

84.82% Statements 123/145 **71.92%** Branches 41/57 **88.88%** Functions 24/27 **83.18%** Lines 94/113

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements			Branches			Functions			Lines	
UserInfoFields.tsx		100%	14/14		100%	14/14		100%	7/7		100%	14/14
UserInformation.tsx		79.68%	51/64		58.33%	14/24		72.72%	8/11		76%	38/50
UserKeyRegister.tsx		77.5%	31/40		60%	9/15		100%	4/4		75.86%	22/29
UserProperties.tsx		100%	18/18		100%	4/4		100%	4/4		100%	13/13
UserRequests.tsx		100%	9/9		100%	0/0		100%	1/1		100%	7/7

Figure 14: Detailed Statistics of the UserProfile

90.24% Statements 74/82 **91.17%** Branches 31/34 **90.9%** Functions 10/11 **88.73%** Lines 63/71

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements			Branches			Functions			Lines	
CondoCreation.tsx		82.05%	32/39		87.5%	21/24		100%	3/3		80.55%	29/36
CondoInfoField.tsx		100%	9/9		100%	4/4		100%	1/1		100%	9/9
CondoInfoForm.tsx		97.05%	33/34		100%	6/6		85.71%	6/7		96.15%	25/26

Figure 15: Detailed Statistics of the CondoProfile

The rest of the folders with no details have very few files, so it is already represented in the “All Files” section.

Backend Reports:

All files

97.48% Statements 1201/1232 **92.07%** Branches 186/202 **92.69%** Functions 203/219 **94.64%** Lines 1132/1196

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
commonFacility/base	91.52%	54/59	87.5%	7/8	100%	10/10	91.22%	52/57
company/base	92.85%	130/140	100%	9/9	93.33%	28/30	94.92%	131/138
companyEmployee/base	98.33%	59/60	100%	12/12	100%	10/10	94.82%	55/58
condoUnit/base	97.57%	101/107	100%	19/19	90.92%	20/22	98.14%	101/105
cost/base	93.02%	40/43	100%	7/7	100%	6/6	100%	41/41
file/base	96.42%	54/56	100%	19/19	100%	6/6	96.29%	52/54
forum/base	91.52%	54/59	100%	8/8	100%	10/10	91.22%	52/57
health/base	100%	10/10	100%	1/1	100%	3/3	100%	10/10
locker/base	95.34%	41/43	100%	11/11	100%	6/6	95.12%	39/41
notification/base	93.04%	40/43	81.81%	9/11	100%	6/6	100%	41/41
parkingSpot/base	95.34%	41/43	90.91%	10/11	100%	6/6	95.12%	39/41
post/base	95.34%	41/43	90.91%	10/11	100%	6/6	95.12%	39/41
property/base	90.1%	118/139	100%	13/13	93.33%	28/30	93.72%	130/137
registrationKey/base	95.45%	42/44	100%	3/3	100%	6/6	95.23%	40/42
request/base	87.58%	50/58	93.83%	22/24	100%	10/10	100%	56/56
reservation/base	95.34%	41/43	95.45%	10/11	100%	6/6	95.12%	39/41
role/base	95.34%	41/43	100%	3/3	100%	6/6	95.12%	39/41
user/base	80.12%	125/156	100%	10/10	88.23%	30/34	79.87%	123/154
userCondo/base	95.34%	41/43	100%	11/11	100%	6/6	95.12%	39/41

Figure 16: Main Page Report of the Backend with JEST

File	Statements		Branches		Functions		Lines	
company/base	90.21%	83/92	90.66%	5/6	83.33%	16/18	90%	81/90
companyEmployee/base	95.45%	42/44	85.45%	9/11	100%	6/6	95.23%	40/42

Figure 17: Detailed Statistics of company

File	Statements		Branches		Functions		Lines	
condoUnit/base	85.71%	78/91	80.88%	17/18	88.88%	16/18	85.39%	76/89
userCondo/base	86.74%	38/43	92.2%	9/11	100%	6/6	100%	41/41

Figure 18: Detailed Statistics of condo

File	Statements		Branches		Functions		Lines	
user/base	80.26%	49/92	16.66%	5/6	88.88%	18/18	100%	90/90
userCondo/base	86.74%	38/43	92.2%	9/11	100%	6/6	100%	41/41

Figure 19: Detailed Statistics of User

The rest of the folders with no details only have 1 file, so it is already represented in the “All Files” section.

5.6.3. Acceptance Tests

<u>Sprint 4 Acceptance Tests</u>

Test ID	AT-1
Requirement	As a condo owner, I want to be able to reserve common facilities in a calendar-like interface.
Acceptance Criteria	User is logged into their account.
Test Case	Expected Results
TC-1 User clicks on “Common Facilities Calendar” from the dashboard page.	The user navigates to a calendar-like interface for common facilities.

Test ID	AT-2
Requirement Tested	As a condo management company, I want to be able to set up a common facility which requires reservations.
Acceptance Criteria	The user is logged into their account. The system provides a user interface to reserve a common facility.
Test Case:	Expected Results
TC-1 Company selects a date and time for a common facility that is available.	The reservation is successfully submitted, and the details are stored in the system.
TC-2 Company selects a date and time for a common facility that is unavailable.	The reservation is not successfully submitted and the user is notified to select a date and time from the dropdown for a valid submission.

Test ID	AT-3
Requirement Tested	As a rental user, I want to be able to see the availability of common facilities.
Acceptance Criteria	The user is logged into their account.
Test Case:	Expected Results
TC-1 User clicks on “Common Facilities Calendar” from the dashboard page.	The user navigates to a calendar-like interface for common facilities.

Test ID	AT-4
Requirement Tested	As a rental user, I want to be able to modify or cancel my existing reservation for a common facility.
Acceptance Criteria	Users can access the list of their reservations from the Common Facilities Calendar Page. The Reservations List page should display a list of all reservations related to the user's submissions.
Test Case:	Expected Results
TC-1 User deletes an existing reservation.	The system displays a confirmation prompt to confirm the deletion. Upon confirmation, the reservation entry is successfully removed from the system.
TC-2 User modifies an existing reservation with valid availability.	The system displays a confirmation prompt to confirm the modification. Upon confirmation, the reservation entry is successfully modified in the system.
TC-3 User deletes an existing reservation with invalid availability.	The modification is not successfully submitted and the user is notified to select a date and time from the dropdown for a valid submission.

Test ID	AT-5
Requirement Tested	As a condo management company, I want to be able to disable the availability of a common facility.
Acceptance Criteria	The user is logged into their account. The system provides a user interface to set the status of common facilities.
Test Case:	Expected Results
TC-1 Company assigns the status "Mark as Closed" to a common facility.	The status of the common facility is successfully updated in the system as closed, and permissions are configured to disallow users access to this facility for reservations.

TC-1 Company assigns the status “Mark as Open” to a common facility.	The status of the common facility is successfully updated in the system as open, and permissions are configured to allow users access to this facility for reservations.

Test ID	AT-6
Requirement Tested	As a condo management company, I want to be able to enter the condo fee per square foot, per parking spot for a condo unit.
Acceptance Criteria	The user is logged into their account. The system displays the user interface for setting condo fees for a condo unit.
Test Case:	Expected Results
TC-1 Company enters the fee per square foot.	The fee per square foot entry is successfully entered in the system.
TC-1 Company enters the fee per parking spot.	The fee per parking spot entry is successfully entered in the system.

Test ID	AT-7
Requirement Tested	As a condo management company, I want to be able to enter the cost for each operation.
Acceptance Criteria	User is logged into their account. The system provides a user interface to add cost for operations.
Test Case:	Expected Results
TC-1 Company selects an operation and assigns a cost to it.	The cost of the operation is successfully submitted and stored in the system.

Test ID	AT-8
Requirement Tested	As a condo owner, I want to be able to see the calculated condo fee for each unit.

Acceptance Criteria	The user is logged into their account. The system displays the condo owner's dashboard.
Test Case:	Expected Results
TC-1 Condo owner navigates to the "My Condos" section from the dashboard.	The system displays the list of condos under the condo owner's ownership.
TC-2 Condo owner selects a specific condo from the condos list to view its profile.	The system navigates to and displays the selected condo profile and displays its details, such as size, occupant name, and condo fee.

Test ID	AT-9
Requirement Tested	As a condo management company, I want to be able to see a record of the operational budget, which includes the total condo fees collected from condo owners.
Acceptance Criteria	The user is logged into their account. The system displays the company's dashboard dashboard.
Test Case:	Expected Results
TC-1 Company navigates to the "Operational Budget Report" section from the dashboard.	The system displays the report for the condo fees collected from condo owners.

Test ID	AT-10
Requirement Tested	As a condo management company, I want to be able to see an annual report.
Acceptance Criteria	The user is logged into their account. The system displays the company's dashboard.
Test Case:	Expected Results
TC-1 Company navigates to the "Annual Report" section from the dashboard.	The system displays the annual report, including details of the revenue, expenses and profits of the company.

5.6.4. Test Results

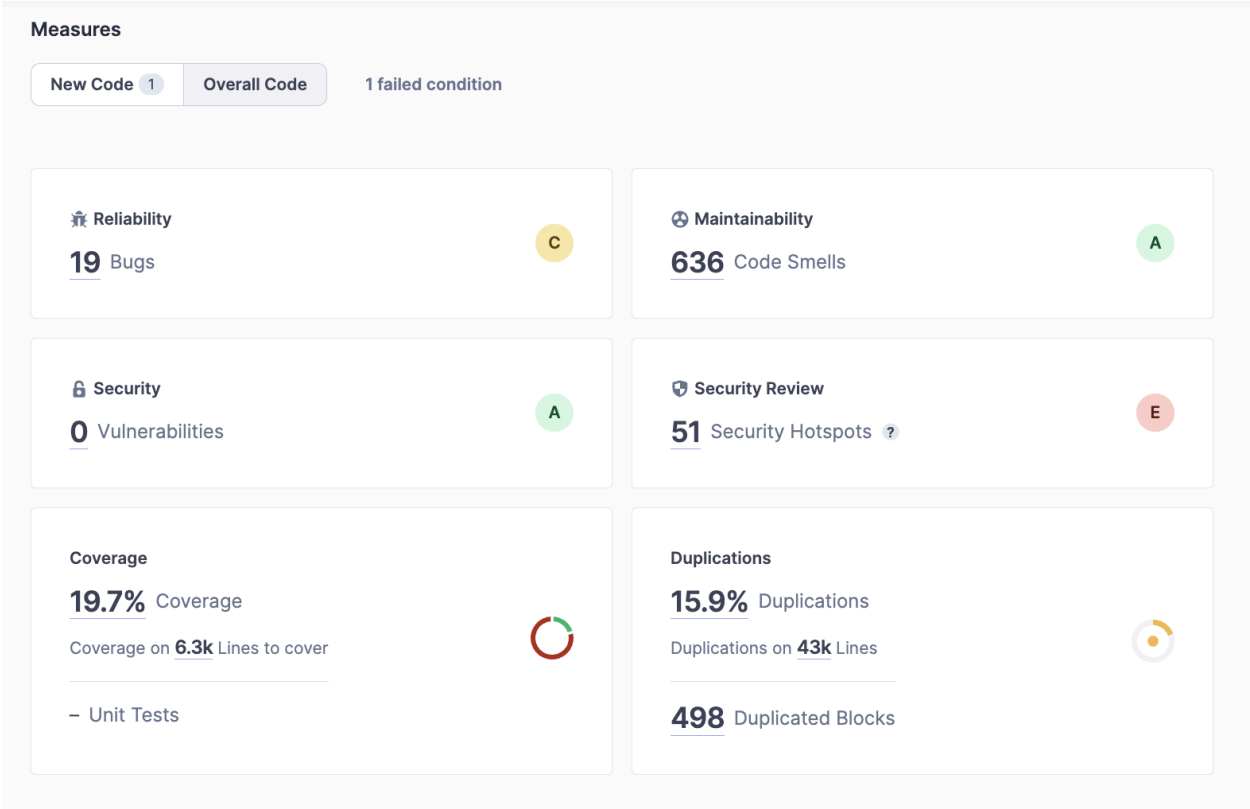


Figure 20: SonarQube Analytics Result for overall Code

5.7. Sprint 5

5.7.1. Metrics

With SonarQube it is easy to keep track of our quality metrics. The metrics are separated in two distinct sections. The first section is overview metrics, which combine other metrics to give an overall perspective of the code quality. As such, we have selected reliability, maintainability and issue metrics. The second section is singular metrics, which are security, test coverage, code duplication, code size, code complexity and issues.

5.7.1.1. Reliability

SonarQube assesses the reliability of code by identifying potential bugs, code smells, and other issues that could lead to unexpected behavior or system failures. By analyzing reliability metrics, such as the number of bugs detected and their severity, SonarQube helps ensure that the codebase is robust and dependable.

Furthermore, the use of tests ensures that the functionalities work consistently according to the requirements. Finally, as part of our core development structure, each function also goes through manual testing in our dev environment. This acts as a final safety net to ensure that the code does what it is supposed to do consistently.

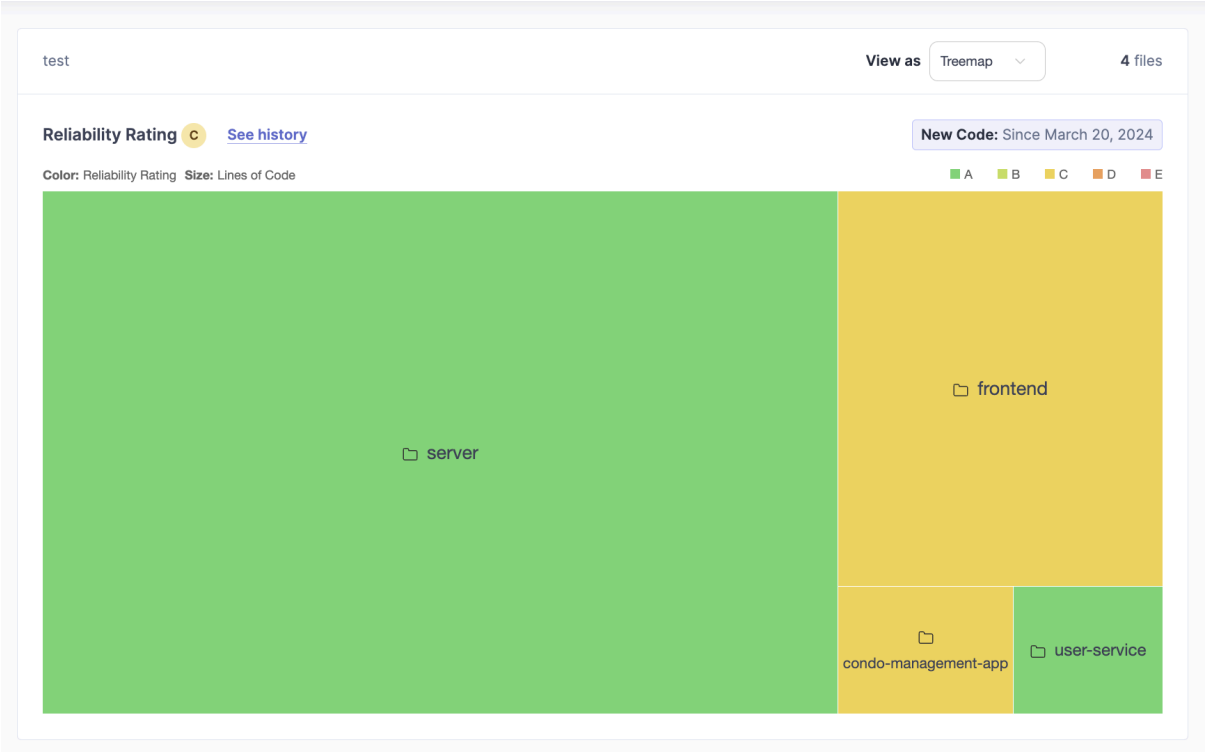


Figure 2: Reliability Report on Sprint 5

5.7.1.2. Maintainability

SonarQube assesses code maintainability based on readability, complexity, and adherence to coding standards and best practices. Improving maintainability measures such as code duplication, cyclomatic complexity, and code coverage simplifies future maintenance, refactoring, and code review procedures.

Figure 3: Maintainability report for sprint 5

5.7.1.3. Security

SonarQube contains security-focused rules and tests that help uncover vulnerabilities, security hotspots, and possible security hazards in code. In addition, SonarQube reduces security risks and strengthens the application's overall security posture by assessing security metrics such as the number and severity of security vulnerabilities.

Figure 4: Security Report on Sprint 5

5.7.1.4. Coverage

SonarQube assesses code coverage by calculating the proportion of code covered by automated tests, such as unit tests, integration tests, and end-to-end tests.
boosting code coverage measures assures that the codebase has been properly tested, lowering the possibility of undetected bugs and boosting overall product quality.

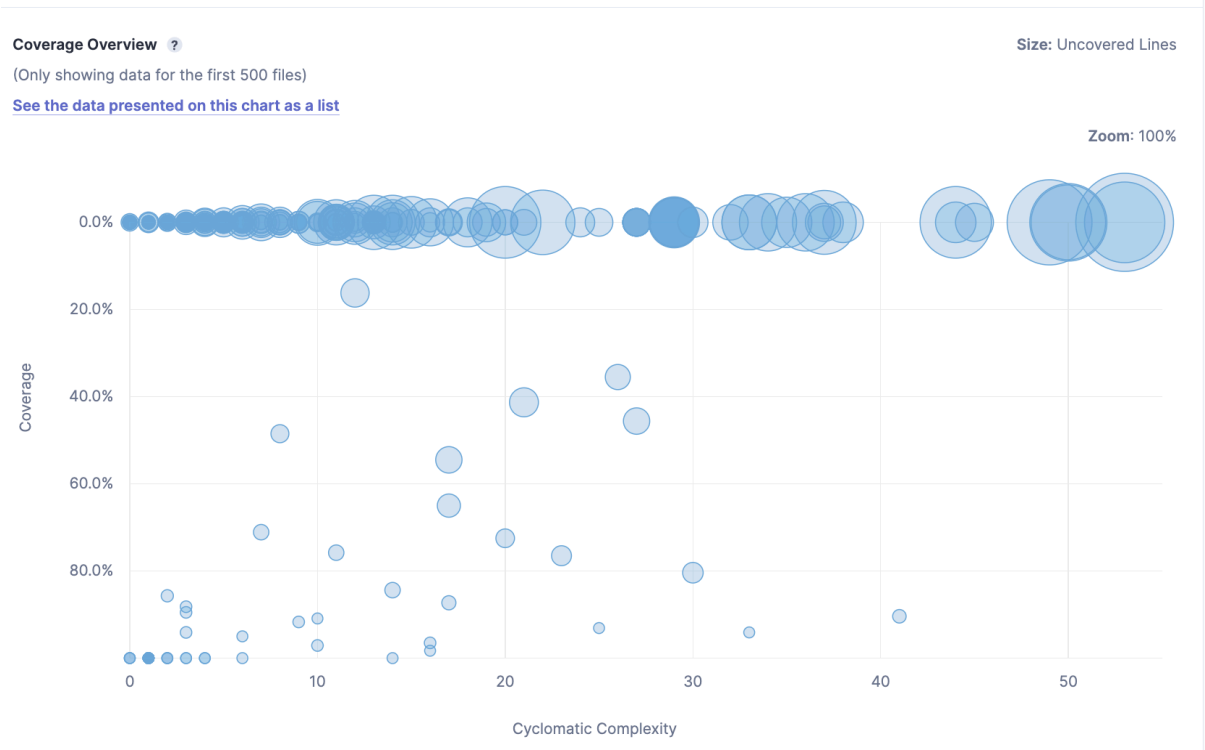
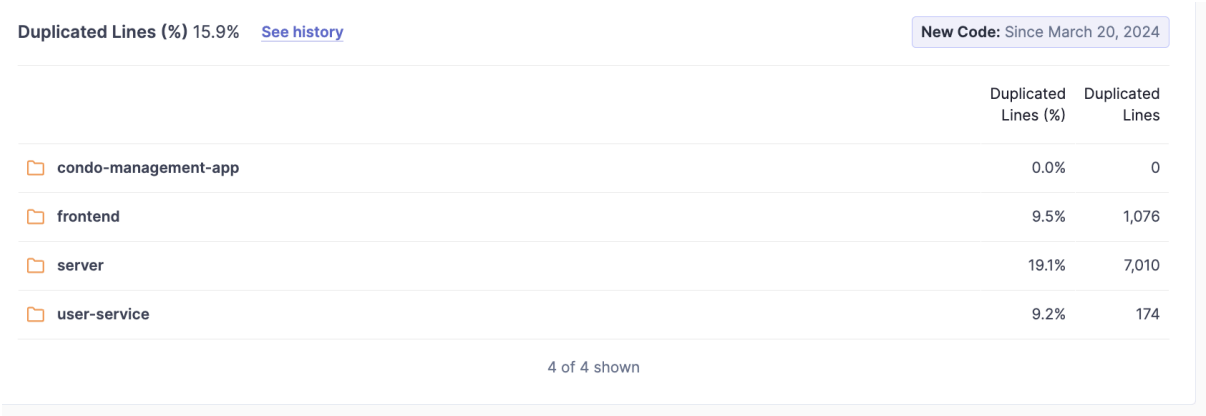


Figure 5: Coverage Report on Sprint 5

5.7.1.5. Duplication

Duplication is a metric that needs to be monitored correctly since multiple people work on the same project. For example, two developers work on two different features that have similarities. If there are no checks in place, the code can get bloated, and hard to read and maintain. Thus, SonarQube is used to help us reduce duplication by finding repeating code patterns or similar code parts throughout the source. Addressing code duplication improves maintainability, readability, and consistency, resulting in more efficient development and simpler issue fixes.



Duplicated Lines (%) 15.9% See history

New Code: Since March 20, 2024

	Duplicated Lines (%)	Duplicated Lines
condo-management-app	0.0%	0
frontend	9.5%	1,076
server	19.1%	7,010
user-service	9.2%	174

4 of 4 shown

Figure 6: Duplication Report for sprint 5

5.7.1.6. Size

SonarQube gives information on the size of the codebase, including the number of files, lines of code, and code churn over time. Monitoring code size metrics allows us to track project growth, identify areas of code bloat or excessive complexity, and improve resource allocation and project planning.

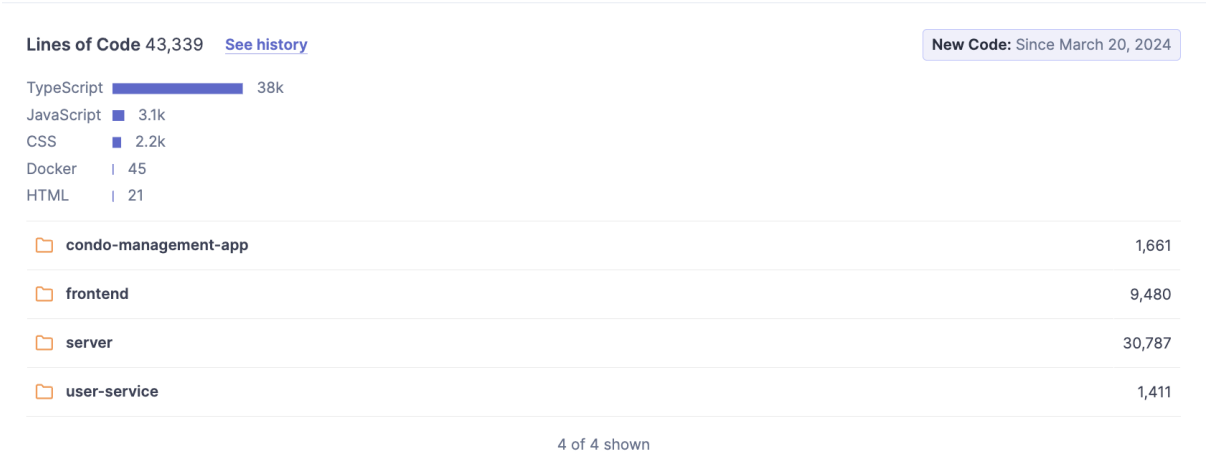


Figure 7: Line Of Code Report for Sprint 5

5.7.1.7. Complexity

SonarQube evaluates code complexity by measures such as cyclomatic complexity, nesting depth, and method length. Analyzing complexity metrics aids in identifying unnecessarily complicated code segments, potential failure spots, and places for rewriting or optimization to enhance code quality and maintenance.

Figure 8: Cyclomatic Complexity Report for sprint 5

Cognitive Complexity 962 See history	New Code: Since March 20, 2024
condo-management-app	56
frontend	304
server	543
user-service	59

4 of 4 shown

Figure 9: Cognitive Complexity Report for sprint 5

5.7.1.8. Issues

SonarQube keeps track of issues found in the code, including but not limited to: code smells, potential bugs, bad practices, unnecessary hard coding etc…
This process is called linting and is a good tool to ensure consistent good practices across developers. As a consequence, issue tracking can increase the maintainability, readability, complexity and other software qualities. The number of issues should be kept as low as possible

Issues 655 See history	New Code: Since March 20, 2024
condo-management-app	122
frontend	211
server	312
user-service	10

4 of 4 shown

Figure 10: Issue Report for sprint 5

5.7.2. Coverage

Software testing involves various metrics to assess the effectiveness of test cases in exercising the code. Four commonly used metrics are statement coverage, branch coverage, function coverage, and line coverage.

Statement coverage

measures the percentage of executable statements that have been executed at least once by the test cases. This metric provides insight into how well individual statements in the code have been exercised during testing.

Branch coverage

focuses on decision points in the code, such as if statements or loops. It assesses the percentage of decision branches that have been executed at least once, giving a more granular view of how well the code paths have been tested.

Function coverage
evaluates the percentage of functions or methods that have been executed by the test cases. This metric helps ensure that the different functional components of the software are adequately tested.
Line coverage
measures the percentage of lines of code that have been executed at least once. It provides a comprehensive overview of how well the entire codebase has been tested in terms of individual lines of code.

As asked in the requirements. We were able to pass 80% code coverage in sprint 1, 2, 3,4 and 5. On the frontend side, the statement coverage overall was 91.99 %, the branch coverage was 87.64%, function coverage was 84.62% and lines were at 86.42% (as shown in the figure below. Therefore we succeeded at achieving over 80% for everything and the coverage is better than any other sprint. We had 59 test suites which is the same as the previous sprint since we removed some test suites and added new ones instead and we had 192 test cases(17 less test cases from sprint 4) which all passed without any failure. These numbers are significant compared to sprint 4.

On the backend side, the testing coverage did not change as no new features were added since everything was finished in the last sprint. the statement coverage overall was 97.48%, the branch coverage was 92.07%, function coverage was 92.69% and lines were at 94.64%. We were able to achieve over 80% coverage for all of these criterias. In total, there were 31 test suites compared to the 28 test suites from sprint 3 and 259 test suites this sprint compared to 251 testsuites last sprint. We have added a significant amount of tests compared to previous sprints to account for adding the financial system on our website. All of the test suites passed without exception. Compared to sprint 3, sprint 4's and sprint 5's statement coverage increased by around 2% which is due to the rigorous testing methods executed in this sprint for the additional functionalities. Our branch coverage has improved by 7%, while function and line coverage increased by 3% and 2 % respectively. That being said, all our test metrics exceed the 80% mark despite the additional functionalities added in sprint 4.

One of JEST's advantages is that it can provide a report in .html format which creates a nice view for the user to see the coverage. A few snapshots of the report were taken below. This provides more details and insights to the person interested.
For the future, we plan to maintain the overall code coverage of at least 80%. This will reduce the number of undetected bugs and will give us more confidence in our application. Code coverage enforcement can be done by configuring JEST to fail every time coverage drops below 80% which will prevent the PR from being merged.

l

Frontend Reports:

All files

91.99% Statements 1551/1686 **87.64%** Branches 566/646 **84.62%** Functions 275/325 **86.42%** Lines 1214/1405

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
src	84.7%	84/110	81.61%	22/26	80.35%	29/34	83.33%	61/81
src/Components	97.05%	33/34	84.61%	33/39	80%	4/5	100%	29/29
src/Components/Authentication	93.52%	159/170	89.13%	164/184	100%	15/15	93.7%	134/143
src/Components/Common	100%	48/48	71.42%	5/7	100%	9/9	100%	43/43
src/Components/CompanyDashboard	100%	8/8	100%	0/0	100%	1/1	100%	5/5
src/Components/Condo	100%	5/5	100%	0/0	100%	1/1	100%	4/4
src/Components/CondoProfile	83.83%	99/124	84.04%	32/41	85%	17/20	88.88%	84/108
src/Components/Costs	51.72%	60/116	88.88%	35/45	82%	15/25	74.82%	64/92
src/Components/Property	87.63%	59/76	90%	27/30	81.11%	16/18	87.94%	53/68
src/Components/PropertyProfile	70.45%	93/132	84%	21/25	74.83%	17/31	70.79%	80/113
src/Components/Requests	82.09%	133/162	78.78%	47/56	75%	28/32	81.56%	115/141
src/Components/Reservations	82.82%	176/198	72.58%	51/58	81.81%	41/44	73.13%	155/166
src/Components/UserProfile	83.56%	122/146	68.42%	39/57	77.77%	21/27	82.45%	94/114
src/Pages	86.86%	307/357	84.71%	73/78	83.38%	59/63	72.14%	215/298

Figure 11: Main Page Report with JEST

All files src/Components/Authentication

93.52% Statements 159/170 **89.13%** Branches 164/184 **100%** Functions 15/15 **93.7%** Lines 134/143

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
EmployeeRegistration.tsx	100%	40/40	88.88%	48/54	100%	4/4	100%	33/33
LogIn.tsx	85.71%	36/42	86.2%	25/29	100%	3/3	86.11%	31/36
LogOut.tsx	100%	6/6	100%	0/0	100%	1/1	100%	6/6
ResetPassword.tsx	100%	26/26	96.42%	27/28	100%	3/3	100%	22/22
SignUp.tsx	91.07%	51/56	87.67%	64/73	100%	4/4	91.3%	42/46

Figure 12: Detailed Statistics of the Authentication Folder and its Components

All files src/Components/Common

100% Statements 45/45 **71.42%** Branches 5/7 **100%** Functions 9/9 **100%** Lines 40/40

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
AuthUtil.tsx	100%	7/7	100%	1/1	100%	1/1	100%	6/6
InitialValues.ts	100%	6/6	100%	0/0	100%	0/0	100%	6/6
List.tsx	100%	20/20	66.66%	4/6	100%	7/7	100%	16/16
LoadingScreen.tsx	100%	5/5	100%	0/0	100%	1/1	100%	5/5
ValidationSchema.ts	100%	7/7	100%	0/0	100%	0/0	100%	7/7

Figure 13: Detailed Statistics of the Common Folder with Reusable files

All files **src/Components/UserProfile**

84.82% Statements 123/145 **71.92%** Branches 41/57 **88.88%** Functions 24/27 **83.18%** Lines 94/113

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
UserInfoFields.tsx	100%	14/14	100%	14/14	100%	7/7	100%	14/14
UserInformation.tsx	79.68%	51/64	58.33%	14/24	72.72%	8/11	76%	38/50
UserKeyRegister.tsx	77.5%	31/40	60%	9/15	100%	4/4	75.86%	22/29
UserProperties.tsx	100%	18/18	100%	4/4	100%	4/4	100%	13/13
UserRequests.tsx	100%	9/9	100%	0/0	100%	1/1	100%	7/7

Figure 14: Detailed Statistics of the UserProfile

90.24% Statements 74/82 **91.17%** Branches 31/34 **90.9%** Functions 10/11 **88.73%** Lines 63/71

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
CondoCreation.tsx	82.05%	32/39	87.5%	21/24	100%	3/3	80.55%	29/36
CondoInfoField.tsx	100%	9/9	100%	4/4	100%	1/1	100%	9/9
CondoInfoForm.tsx	97.05%	33/34	100%	6/6	85.71%	6/7	96.15%	25/26

Figure 15: Detailed Statistics of the CondoProfile

The rest of the folders with no details have very few files, so it is already represented in the “All Files” section.

Backend Reports:

All files

97.48% Statements 1201/1232 **92.07%** Branches 186/202 **92.69%** Functions 203/219 **94.64%** Lines 1132/1196

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
commonFacility/base	91.52%	54/59	87.5%	7/8	100%	10/10	91.22%	52/57
company/base	92.85%	130/140	100%	9/9	93.33%	28/30	94.92%	131/138
companyEmployee/base	98.33%	59/60	100%	12/12	100%	10/10	94.82%	55/58
condoUnit/base	97.57%	101/107	100%	19/19	90.92%	20/22	98.14%	101/105
cost/base	93.02%	40/43	100%	7/7	100%	6/6	100%	41/41
file/base	96.42%	54/56	100%	19/19	100%	6/6	96.29%	52/54
forum/base	91.52%	54/59	100%	8/8	100%	10/10	91.22%	52/57
health/base	100%	10/10	100%	1/1	100%	3/3	100%	10/10
locker/base	95.34%	41/43	100%	11/11	100%	6/6	95.12%	39/41
notification/base	93.04%	40/43	81.81%	9/11	100%	6/6	100%	41/41
parkingSpot/base	95.34%	41/43	90.91%	10/11	100%	6/6	95.12%	39/41
post/base	95.34%	41/43	90.91%	10/11	100%	6/6	95.12%	39/41
property/base	90.1%	118/139	100%	13/13	93.33%	28/30	93.72%	130/137
registrationKey/base	95.45%	42/44	100%	3/3	100%	6/6	95.23%	40/42
request/base	87.58%	50/58	93.83%	22/24	100%	10/10	100%	56/56
reservation/base	95.34%	41/43	95.45%	10/11	100%	6/6	95.12%	39/41
role/base	95.34%	41/43	100%	3/3	100%	6/6	95.12%	39/41
user/base	80.12%	125/156	100%	10/10	88.23%	30/34	79.87%	123/154
userCondo/base	95.34%	41/43	100%	11/11	100%	6/6	95.12%	39/41

Figure 16: Main Page Report of the Backend with JEST

File		Statements		Branches		Functions		Lines	
company/base		90.21%	83/92	90.66%	5/6	83.33%	16/18	90%	81/90
companyEmployee/base		95.45%	42/44	85.45%	9/11	100%	6/6	95.23%	40/42

Figure 17: Detailed Statistics of company

File		Statements		Branches		Functions		Lines	
condoUnit/base		85.71%	78/91	80.88%	17/18	88.88%	16/18	85.39%	76/89
userCondo/base		86.74%	38/43	92.2%	9/11	100%	6/6	100%	41/41

Figure 18: Detailed Statistics of condo

File		Statements		Branches		Functions		Lines	
user/base		80.26%	49/92	16.66%	5/6	88.88%	18/18	100%	90/90
userCondo/base		86.74%	38/43	92.2%	9/11	100%	6/6	100%	41/41

Figure 19: Detailed Statistics of User

The rest of the folders with no details only have 1 file, so it is already represented in the “All Files” section.

5.7.3. Acceptance Test

<u>Sprint 4 Acceptance Tests</u>

Test ID	AT-1
Requirement	As a condo owner, I want to be able to reserve common facilities in a calendar-like interface.
Acceptance Criteria	User is logged into their account.
Test Case	Expected Results
TC-1 User clicks on “Common Facilities Calendar” from the dashboard page.	The user navigates to a calendar-like interface for common facilities.

Test ID	AT-2

Requirement Tested	As a condo management company, I want to be able to set up a common facility which requires reservations.
Acceptance Criteria	The user is logged into their account. The system provides a user interface to reserve a common facility.
Test Case:	Expected Results
TC-1 Company selects a date and time for a common facility that is available.	The reservation is successfully submitted, and the details are stored in the system.
TC-2 Company selects a date and time for a common facility that is unavailable.	The reservation is not successfully submitted and the user is notified to select a date and time from the dropdown for a valid submission.

Test ID	AT-3
Requirement Tested	As a rental user, I want to be able to see the availability of common facilities.
Acceptance Criteria	The user is logged into their account.
Test Case:	Expected Results
TC-1 User clicks on “Common Facilities Calendar” from the dashboard page.	The user navigates to a calendar-like interface for common facilities.

Test ID	AT-4
Requirement Tested	As a rental user, I want to be able to modify or cancel my existing reservation for a common facility.
Acceptance Criteria	Users can access the list of their reservations from the Common Facilities Calendar Page. The Reservations List page should display a list of all reservations related to the user’s submissions.
Test Case:	Expected Results

TC-1 User deletes an existing reservation.	The system displays a confirmation prompt to confirm the deletion. Upon confirmation, the reservation entry is successfully removed from the system.
TC-2 User modifies an existing reservation with valid availability.	The system displays a confirmation prompt to confirm the modification. Upon confirmation, the reservation entry is successfully modified in the system.
TC-3 User deletes an existing reservation with invalid availability.	The modification is not successfully submitted and the user is notified to select a date and time from the dropdown for a valid submission.

Test ID	AT-5
Requirement Tested	As a condo management company, I want to be able to disable the availability of a common facility.
Acceptance Criteria	The user is logged into their account. The system provides a user interface to set the status of common facilities.
Test Case:	Expected Results
TC-1 Company assigns the status “Mark as Closed” to a common facility.	The status of the common facility is successfully updated in the system as closed, and permissions are configured to disallow users access to this facility for reservations.
TC-1 Company assigns the status “Mark as Open” to a common facility.	The status of the common facility is successfully updated in the system as open, and permissions are configured to allow users access to this facility for reservations.

Test ID	AT-6
Requirement Tested	As a condo management company, I want to be able to enter the condo fee per square foot, per parking spot for a condo unit.

Acceptance Criteria	The user is logged into their account. The system displays the user interface for setting condo fees for a condo unit.
Test Case:	Expected Results
TC-1 Company enters the fee per square foot.	The fee per square foot entry is successfully entered in the system.
TC-1 Company enters the fee per parking spot.	The fee per parking spot entry is successfully entered in the system.

Test ID	AT-7
Requirement Tested	As a condo management company, I want to be able to enter the cost for each operation.
Acceptance Criteria	User is logged into their account. The system provides a user interface to add cost for operations.
Test Case:	Expected Results
TC-1 Company selects an operation and assigns a cost to it.	The cost of the operation is successfully submitted and stored in the system.

Test ID	AT-8
Requirement Tested	As a condo owner, I want to be able to see the calculated condo fee for each unit.
Acceptance Criteria	The user is logged into their account. The system displays the condo owner's dashboard.
Test Case:	Expected Results
TC-1 Condo owner navigates to the "My Condos" section from the dashboard.	The system displays the list of condos under the condo owner's ownership.
TC-2 Condo owner selects a specific condo from the condos list to view its profile.	The system navigates to and displays the selected condo profile and displays its details, such as size, occupant name, and condo fee.

Test ID	AT-9
Requirement Tested	As a condo management company, I want to be able to see a record of the operational budget, which includes the total condo fees collected from condo owners.
Acceptance Criteria	The user is logged into their account. The system displays the company's dashboard dashboard.
Test Case:	Expected Results
TC-1 Company navigates to the "Operational Budget Report" section from the dashboard.	The system displays the report for the condo fees collected from condo owners.

Test ID	AT-10
Requirement Tested	As a condo management company, I want to be able to see an annual report.
Acceptance Criteria	The user is logged into their account. The system displays the company's dashboard.
Test Case:	Expected Results
TC-1 Company navigates to the "Annual Report" section from the dashboard.	The system displays the annual report, including details of the revenue, expenses and profits of the company.

5.7.4. Test Result

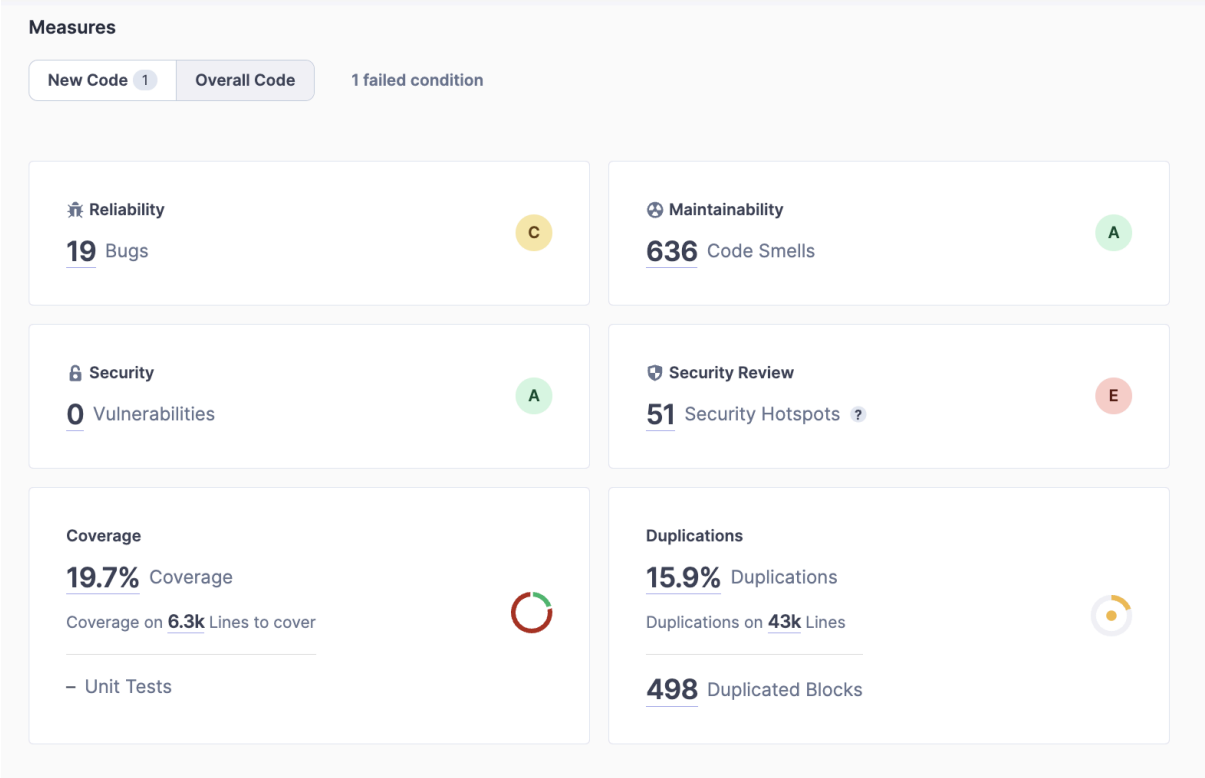


Figure 20: SonarQube Analytics Result for overall Code

6. Final Risk Assessment and Management Plan

6.1. Introduction

A risk is an event or condition that, if it occurs, could have a positive or negative effect on a project's objectives. Risk Management is the process of identifying, assessing, responding to, monitoring, and reporting risks. This Risk Management Plan defines how risks associated with the Condo Management software project will be identified, analyzed, and managed. It outlines how risk management activities will be performed, recorded, and monitored throughout the lifecycle of the project and provides templates and practices for recording and prioritizing risks.

The Risk Management Plan is created by the project manager in the Planning Phase of the CDC Unified Process and is monitored and updated throughout the project. The intended audience of this document is the project team, project sponsor and management.

Risks and assessments were identified by looking at the distribution of the workload, staffing, process, design, supplier, resources, and dependencies. Additionally, the team monitors test results as it pertains to failures and keeps an eye on the progress rate of different user stories and tasks. Reviewing potential shortfalls against expectations, and analyzing negative trends concerning the different technologies used.

Quantification of Impact

Impact	Quantification
Low	Low impact represents risks that would take 0-15 hours to resolve if it occurred. It would have minimal impact on project objectives or resources.
Medium	Medium impact represents risks that would take 16-30 hours to resolve if it occurred. It would have a Moderate impact on project objectives or resources, manageable with some effort.
High	High Impact represents risks that would take over 30 hours to resolve if it occurred. Significant impact on project objectives or resources, requiring immediate attention and extensive effort to mitigate.

Quantification of Probability

Probability	Quantification
Low	Low probability means the risk has a 0 -25% of occurring. Unlikely to occur or rare occurrence.
Medium	Medium probability means the risk has a 26-50% chance of occurring. Reasonable chance of occurrence, not uncommon.
High	High probability means the risk has over a 50% chance of occurring.

In order to determine the total risk score of a risk we chose to multiply the metrics for both impact and probability. For example: RiskExample has a probability of 51% and an impact of 24 hours. Its risk score would be 20 x 51 = 1020. 1020 is above 900 so that risk would be categorized as high. We used this risk score to help us determine which category to place the overall risk (Low-green, Medium-yellow, High-red).

Overall Risk Scores:

Low: 0-450

Medium: 451-900

High: over 900

Quantification of Figure 1

Impact	Probability	Quantification of each category
Low	Low	The risk impact is below 15 hours, the probability is below 25% and the risk score is below 450.
Low	Medium	The risk impact is below 15 hours, the probability is between 26% and 50% and the risk score is below 450.
Medium	Low	The risk impact is between 16 and 30 hours, the probability is below 25% and the risk score is below 450.
Low	High	The risk impact is below 15 hours, the probability is above 50% and the risk score is between 451 and 900.
Medium	Medium	The risk impact is between 16 and 30 hours, the probability is between 26% - 50% and the risk score is between 451 and 900.
High	Low	The risk impact is above 30 hours, the probability is below 25% and the risk score is between 451 and 900.
Medium	High	The risk impact is between 16-30 hours, the probability is above 50 % and the risk score is above 900.
High	Medium	The risk impact is above 30 hours, the probability is between 26%-50% and the risk score is above 900.
High	High	The risk impact is above 30 hours, the probability is above 50 % and the risk score is above 900.

6.2. Risk Assessment

1. **Schedule Risk**

- Probability: 30%
- Impact: If it occurs, it could delay the project by approximately 10 hours.
- Explanation: We arrived at this evaluation by analyzing past project schedules and considering potential factors that could lead to delays.
- Risk Score: 30 x 10 = 300. LOW

2. **Scope Creep**

- Probability: 40%

- Impact: Scope creep could result in additional development efforts, leading to well over 40 hours of extra effort.
- Explanation: Scope creep was identified based on the tendency of requirements to evolve during the development process, leading to increased complexity and workload.
- Risk Score: 40 x 40 = 1600 HIGH

3. Narrow Knowledge of the Technology Used

- Probability: 55%
- Impact: Insufficient knowledge of the technology could lead to an approximate delay of 20 hours due to learning curves and potential errors in implementation.
- Explanation: The probability is based on the level of familiarity team members have with the technology stack. Impact assessment considers the time required for on-the-job learning and potential setbacks in development.
- Risk Score: 55 x 15 = 1100 HIGH

4. Ambiguous Requirements

- Probability: 35%
- Impact: Ambiguous requirements may necessitate an additional 20 hours of project duration for clarification and adjustments.
- Explanation: The probability is derived from historical data on requirement clarification requests. Impact assessment accounts for the time needed to resolve ambiguities and potential rework caused by misunderstandings.
- Risk Score: 35 x 20 = 680 MEDIUM

5. Privacy Leaks

- Probability: 20%
- Impact: A privacy leak incident could require approximately 30 hours to investigate, mitigate, and address legal implications.
- Explanation: Probability assessment considers the frequency of similar incidents in the industry. Impact assessment accounts for the time and resources needed to address the breach and its consequences.
- Risk Score: 20 x 30 = 600 MEDIUM

6. Poor Code Quality

- Probability: 15%

- Impact: Poor code quality might lead to an additional 12 hours of debugging and refactoring efforts.
- Explanation: Probability is based on past experiences with code quality issues. Impact assessment considers the time required to identify and rectify defects, as well as the potential impact on project timelines.
- Risk Score: 15 x 12 = 180 LOW

7. Lack of Communication Among Team Members

- Probability: 20%
- Impact: Ineffective communication could result in a 35 hours in project duration due to misunderstandings and coordination issues.
- Explanation: Probability assessment considers team dynamics and the complexity of communication channels. Impact assessment accounts for the time lost in clarifications and rework caused by miscommunication.
- Risk Score: 20 x 35 = 700. MEDIUM

8. Database Performance

- Probability: 15%
- Impact: Database performance issues may lead to an approximate delay of 8 hours for troubleshooting and optimization.
- Explanation: Probability assessment is based on the complexity of the database architecture. Impact assessment considers the time needed to diagnose and address performance bottlenecks.
- Risk Score: 15 x 8 = 120. LOW

9. Low Productivity from Team Members

- Probability: 35%
- Impact: Low productivity could result in an approximate delay of 20 hours due to missed deadlines and increased workload on other team members.
- Explanation: Probability assessment considers factors affecting team motivation and morale. Impact assessment accounts for the time lost in addressing productivity issues and redistributing tasks.
- Risk Score: 35 x 20 = 500. MEDIUM

From the above metrics we can place the risks in this chart as such:

Impact Probability	Low (0-15h)	Medium (16-30h)	High (>30h)
Low(<25%)	R6, R8	R5	R7
Medium (26-50%)	R1	R4, R9	R2
High (>50%)		R3	

Figure 1: Risk management chart

Risk ID	Risk Type and Description	Risk Score	Resolved in Sprint	Strategy and Effectiveness
R1	Schedule: Schedule Risk	LOW	1	Avoidance: Develop Project Plan and break the team into smaller teams to allow easier scheduling.
R2	Management: Scope Creep	HIGH	1	Acceptance: Breakdown the project into smaller manageable tasks.
R3	Technical: Narrow knowledge of the technology used	HIGH	1	Mitigation: Have the team members familiarize themselves with the technology used outside the scope of the project.
R4	Management: Ambiguous Requirements	MEDIUM	1	Mitigation: Incremental development
R5	External: Privacy leaks	LOW	5	Avoidance: Encrypting users credentials and conducting security tests for the software.
R6	Technical: Poor code quality	LOW	2	Avoidance: Continuously reviewing, testing, and refactoring the code

R7	Management: Lack of communication among team members	MEDIUM	1	Acceptance: Develop daily communication channels
R8	Technical: Database performance	LOW	5	Mitigation: Use a more reliable option
R9	Management: Low productivity from team members	MEDIUM	4	Acceptance: Setting achievable deadlines would help prevent burnout and increase motivation.

Table 1: List of identified risks

7. Overall Release Plan

7.1. Sprint 1 Release Plan

In Sprint 1, we expected to accomplish all tasks regarding setting up the technology for our web application, Public User and Company login and sign-up, Viewing user profile information, Editing user profile information, and being able to navigate through the pages of the web application.

We've made significant progress towards our deployment goals. Currently, our web application is successfully deployed on a remote server, providing accessibility and scalability. Throughout this sprint, we focused on refining our deployment strategy to ensure seamless integration with cloud services or devices in the future. While API calls are currently directed to localhost, a temporary measure for development purposes, our plan includes dynamically redirecting these calls to the appropriate location for hosted deployment, ensuring full functionality of the web application. Leveraging Docker technology, we've containerized our website and implemented Docker Image CI for continuous integration, streamlining the deployment process. This CI system, triggered by pull requests, automatically builds and deploys both frontend and backend components remotely, adhering to their respective Dockerfile specifications. Additionally, we've established two independent environments; dev and prod, each housed within its own virtual container, facilitating efficient testing and deployment workflows. With these foundations laid in Sprint 1, we are well-positioned to seamlessly transition to cloud services or devices in subsequent phases of development.

User Story ID	User Story Points (USP)	Priority	Status
US-01	5	High	DONE
US-02	5	High	DONE

US-03	3	Medium	DONE
US-07	2	Medium	DONE
US-09	5	High	DONE
US-08	5	High	DONE
US-36	2	High	DONE
US-10	3	High	DONE
US-11	1	Medium	DONE
Total USP	31		

Table 1: Sprint 1 Release Plan

7.2. Sprint 2 Release Plan

In Sprint 2 we are prioritizing the implementation of the registration key system. We also expect to finish all tasks regarding condo management companies being able to create properties along with uploading files. We also plan to accomplish any leftover tasks relating to the authentication system (adding a forget password option and reset password option).

Our condo management system achieved significant enhancements by incorporating a dashboard for owners with comprehensive property and financial data, a facility to delete outdated property listings, and detailed profiles for individual condos. Owners can now submit various requests to management, while employees have a system to view and manage these requests efficiently. A new notifications page keeps all users informed of request statuses and updates. The system's flexibility has been bolstered by the addition of role-specific access for employees and the ability to delete former employee profiles, streamlining operations and security for the management company.

User Story ID	User Story Points (USP)	Priority	Status
US-04	2	Low	DONE
US-05	2	Low	DONE
US-012	2	Low	DONE
US-013	2	Low	DONE
US-015	2	High	DONE
US-014	2	High	DONE
US-016	3	High	DONE
US-018	3	High	DONE
US-019	2	High	DONE
Total USP	20		

Table 2: Sprint 2 Release Plan

7.3. Sprint 3 Release Plan

In Sprint 3, we aim to complete key features for both condo owners and management. This includes developing a property management dashboard, enabling request submissions, and providing detailed condo profiles for owners. Management tasks involve setting up roles for staff, deleting properties, and managing employee profiles. A notification page will also be implemented to keep all users updated. The sprint's effort is estimated at 35 user story points, indicating the anticipated workload for the team.

User Story ID	User Story Points (USP)	Priority	Status
US-017	8	High	DONE
US-023	5	Medium	DONE
US-024	3	High	DONE
US-025	3	High	DONE
US-038	5	High	DONE
US-039	4	Low	REMOVED
US-053	5	Medium	DONE
US-054	2	Medium	DONE
Total USP	35		

Table 3: Sprint 3 Release Plan

7.4. Sprint 4 Release Plan

In Sprint 4, we aim to prioritize the implementation of both the Reservation and Financial systems. We will implement the key features for condo owners and company management regarding these systems. Reservation system includes tasks for set up, reservation, disabling, modifying and canceling all in regards to common facilities. Financial system includes tasks such as entering and calculating condo fees and operations cost, and generating annual financial reports.

User Story ID	User Story Points (USP)	Priority	Status
US-026	3	High	DONE
US-027	5	High	DONE
US-028	2	High	DONE
US-029	3	Medium	DONE
US-030	5	Medium	DONE
US-031	2	High	DONE
US-032	2	High	DONE
US-033	1	Low	DONE
US-034	3	Medium	DONE
US-035	3	Medium	DONE
Total USP	29		

Table 4: Sprint 4 Release Plan

7.5. Sprint 5 Release Plan

In Sprint 5, we will prioritize implementing additional features for our application, namely the forums. Forums include tasks for posting, replying, and viewing other posts.

User Story ID	User Story Points (USP)	Priority	Status
US-040	5	Low	DONE
US-041	5	Low	DONE
Total USP	10		

Table 5: Sprint 5 Release Plan

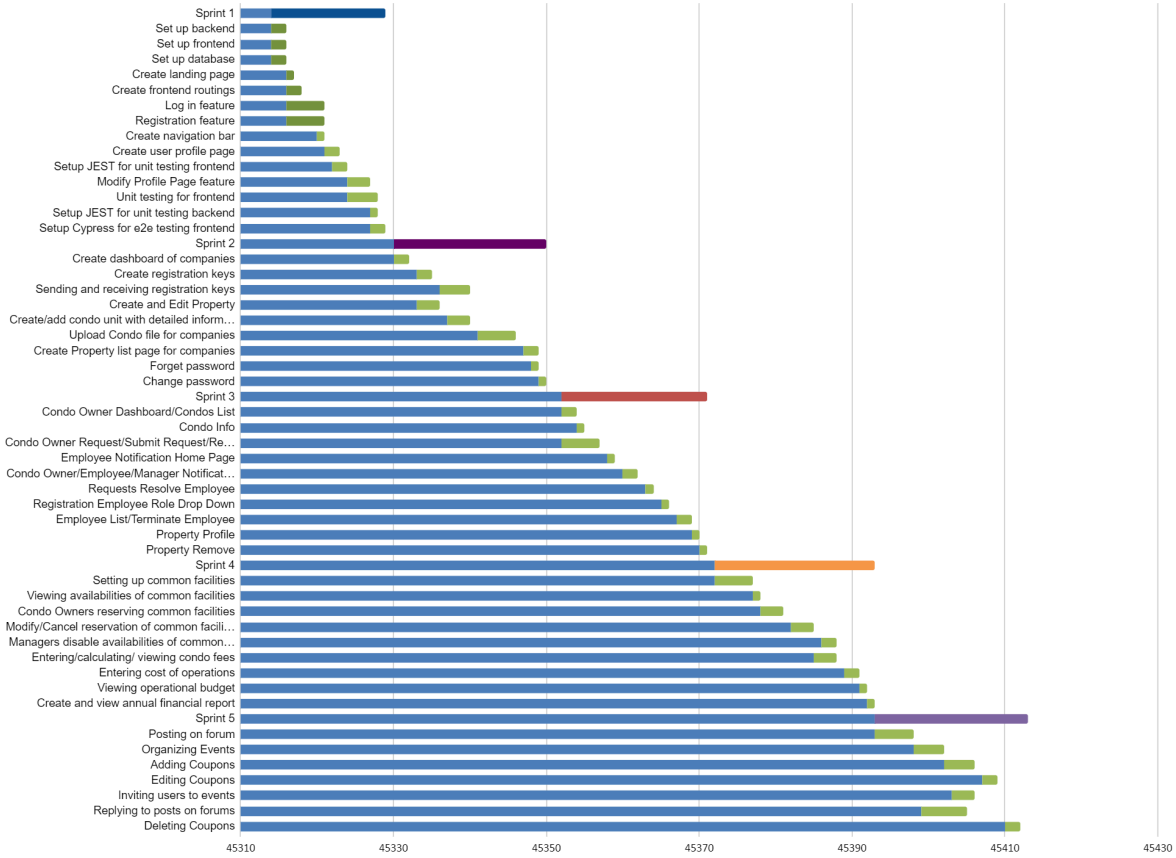


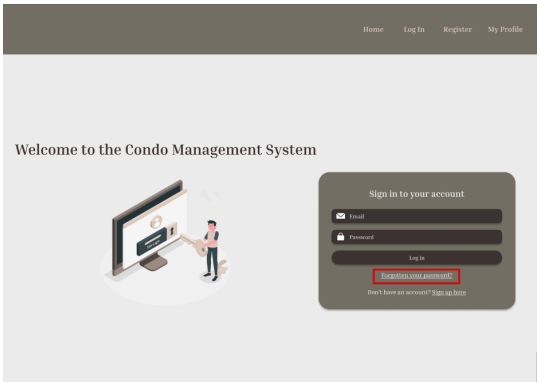
Figure 1:Project Plan Bar Chart

8. UI Prototypes

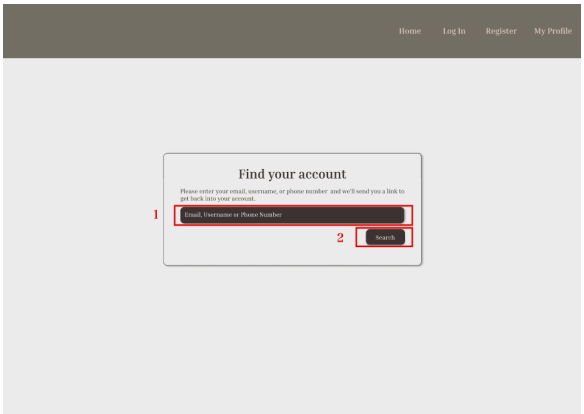
8.1. UI Prototypes for Sprint 2 User stories

8.1.1. US-04

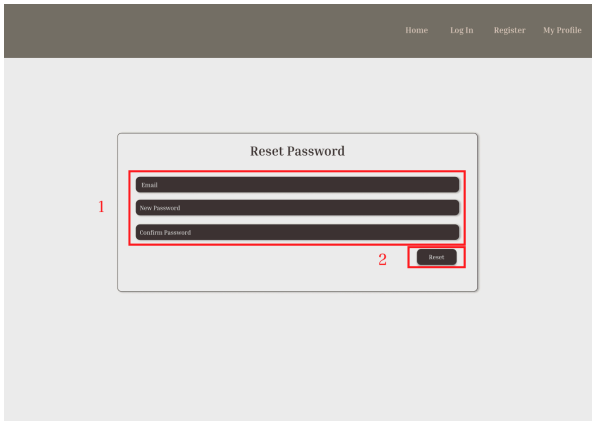
As a public user, I want to be able to request a password reset so that I can regain access to my account if I forget my password.



Login Page



Find Your Account Page



Reset Password Page

Prerequisite(s):

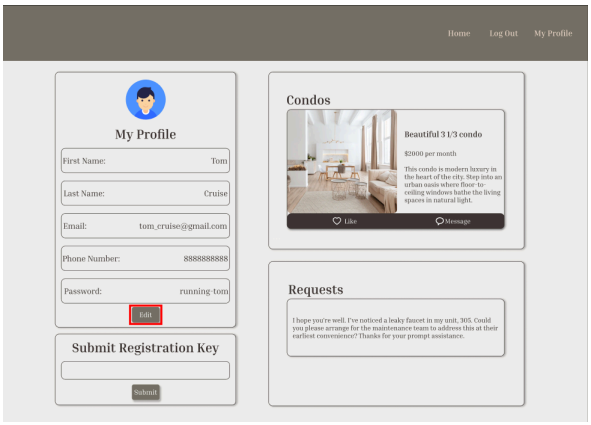
- The user has an existing account in the system.

Steps:

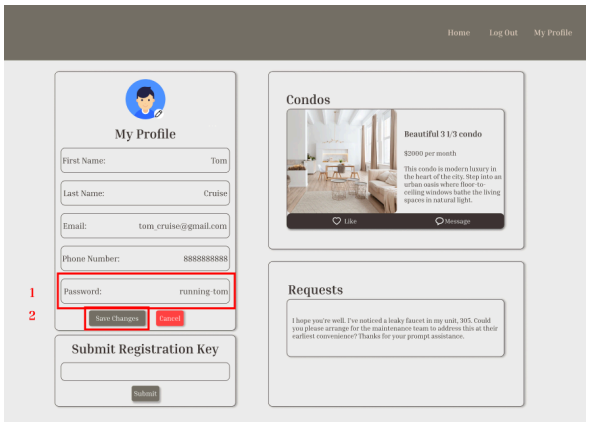
1. The user clicks on the “Forgotten your password?” button on the login page.
2. The user enters their email, username or phone number on the “Find your account” page.
3. The user clicks on the “Search” button.
4. An email will be sent to the user with the link to the “Reset Password” Page
5. The user fills in the form with his email and his new password twice to confirm.
6. The user clicks on the “Reset” button.

8.1.2. US-05

As a public user, I want to be able to change my password whenever I'd like, so that I can enhance the security of my account.



Profile Page



Profile Page

Prerequisite(s):

- The user is logged into his account.

Steps:

1. The user clicks on the "My Profile" button in the navigation bar.
2. The user clicks on the "Edit" button on the profile page.
3. The user selects the "Password" textbox and types his new password.
4. The user clicks on the "Save Changes" button.

8.1.3. US-012

As a Condo Management Company, I want to be able to request a password reset so that I can regain access to my account if I forget my password.

Please refer back to US-04 for steps.

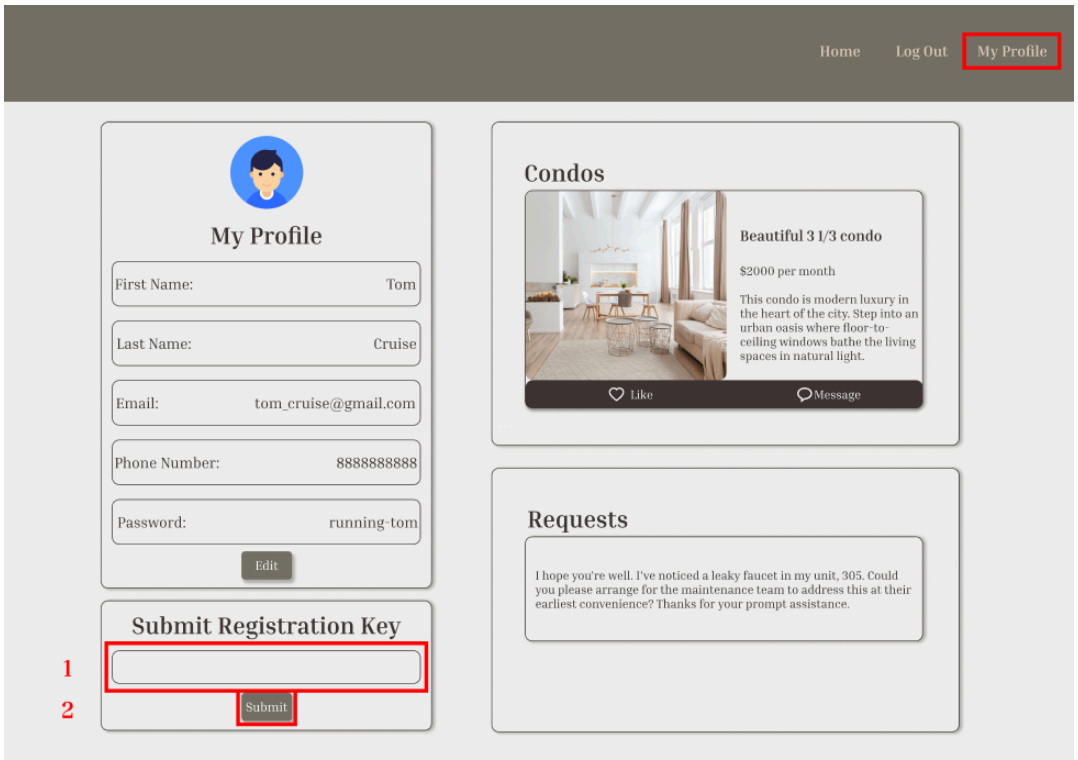
8.1.4. US-013

As a Condo Management Company, I want to change my password when I'd like so that I can enhance the security of my account.

Please refer back to US-05 for steps.

8.1.5. US-014

As a public user, I want to have a registration key from the condo company to become an owner in the system so that I can authenticate my ownership and access the owner-specific features.



Profile Page

Prerequisite(s):

- The user is logged into their account.

Steps:

1. The user clicks on the “My Profile” button in the navigation bar.
2. The user types his registration key in the textbox.
3. The user clicks on the “Submit” button.

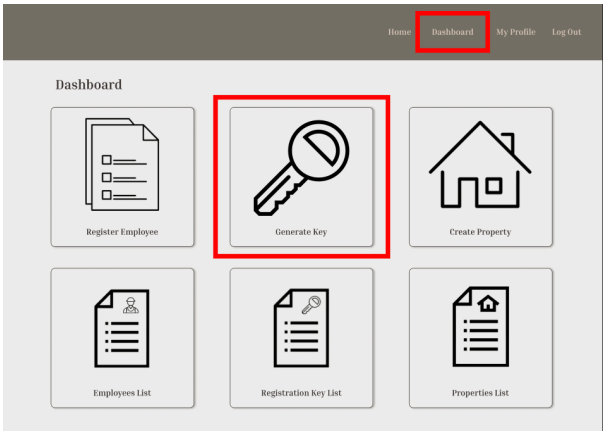
8.1.6. US-015

As a public user, I want to have a registration key from the condo company to become a rental user in the system so that I can authenticate my identity and access the tenant-specific features.

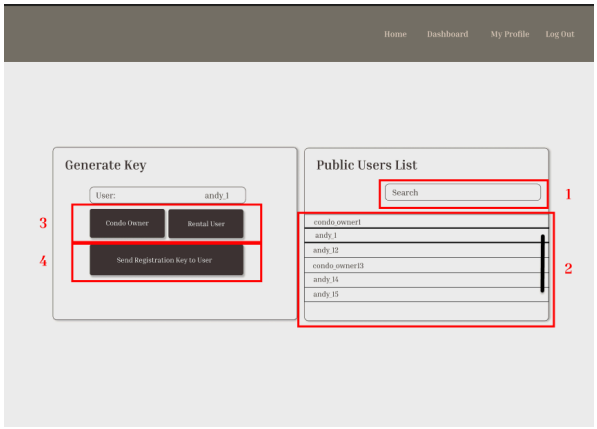
Please refer back to US-14 for steps.

8.1.7. US-016

As a condo management company, I want to be able to send registration keys to unit owners or rental users for their units so that they can link their profiles to their units and access relevant information and features.



Dashboard



Generate Key Page

Prerequisite(s):

- The user is logged into his account.

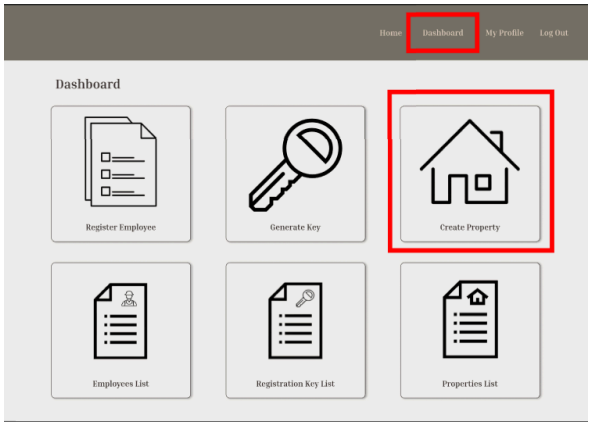
Steps:

1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks on the “Generate Key” box.
3. The user types a username in the search box.
4. The user scrolls in the list of public users and clicks on the user once found.
5. The user selects either the “Condo Owner” button or the “Rental User” button depending on the public user’s role.
6. The user clicks on the “Send Registration Key to User” button.

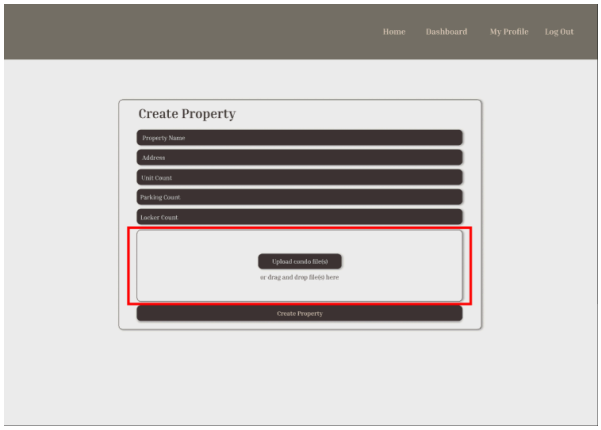
8.1.8. US-018

As a condo management company, I want to be able to upload condo file for each property so that all condo owners of that property can access important documents and information.

8.1.8.1. Scenario 1: User wants to upload new condo files to a new property.



Dashboard Page



Create Property Page

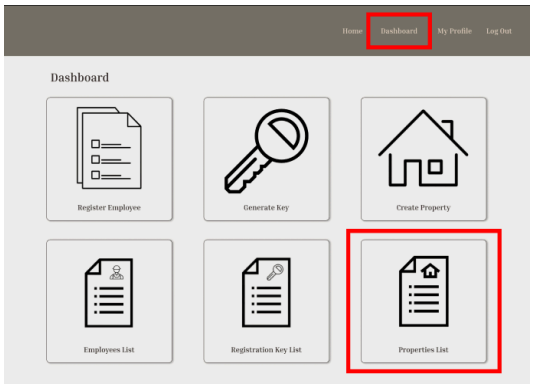
Prerequisite(s):

- The user is logged into his account.
- Condo files are in .pdf format.

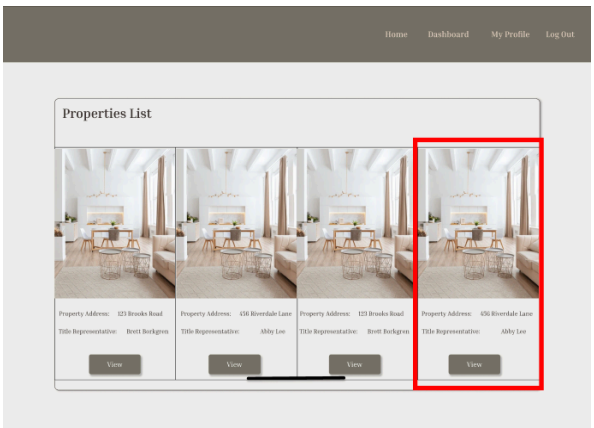
Steps:

1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks the “Create Property” box.
3. The user clicks on the “Upload condo file(s)” button or drag and drops condo files in the box.

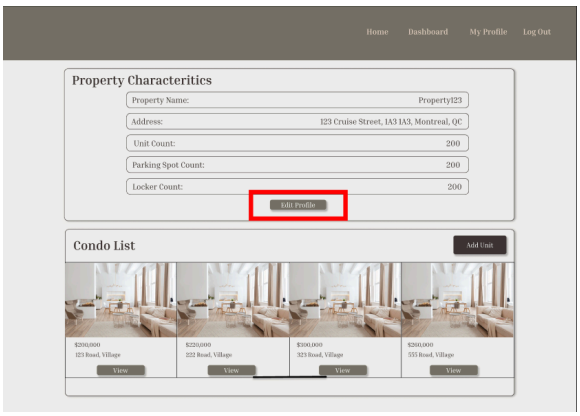
8.1.8.2. Scenario 2: User wants to upload new condo files to an existing property



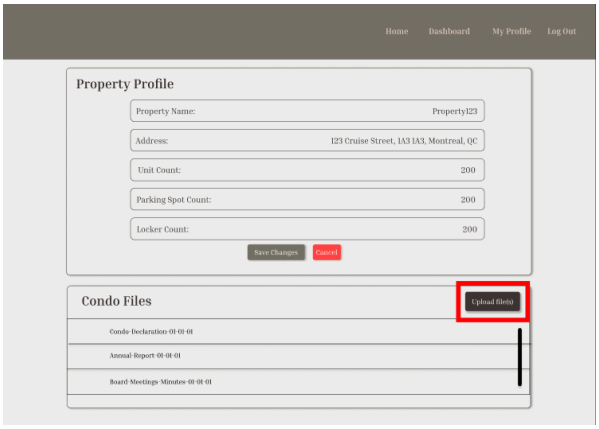
Dashboard Page



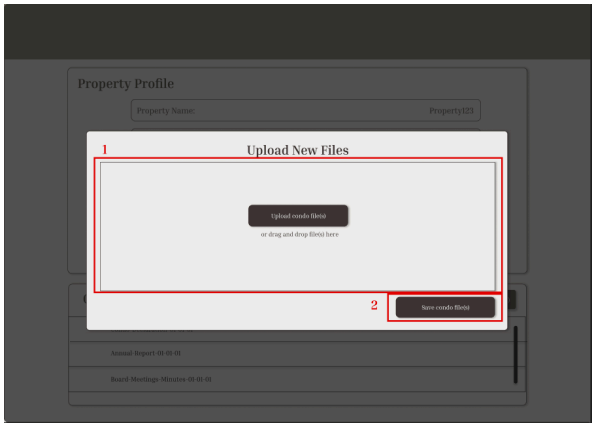
Properties List Page



General Property Page



Property Profile Edit Page



Upload New Files Pop-up

Prerequisite(s):

- The user is logged into his account.
- Condo files are in .pdf format.
- The new condo files are not duplicates of existing condo files for the property.

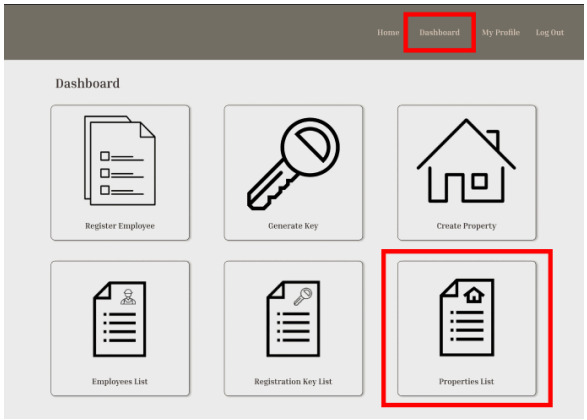
Steps:

1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks the “Properties List” box.
3. The user scrolls through the list of properties and clicks on the property they desire to upload a condo file for.
4. The user clicks the “Edit Profile” button in the General Property page.
5. The user clicks on the “Upload file(s)” button in the Condo Files section of the Property Profile Edit page.
6. The user uploads new condo files or drag and drops condo files in the “Upload New Files” Pop-up.
7. The user clicks the “Save condo file(s)” button.

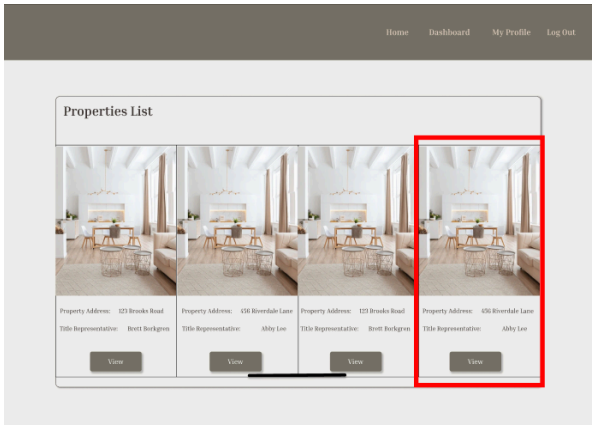
8.1.9. US-019

As a condo management company, I want to be able to enter detailed information for each condo unit (condo unit, parking spot, locker) so that I can keep track of all assets, amenities and their details.

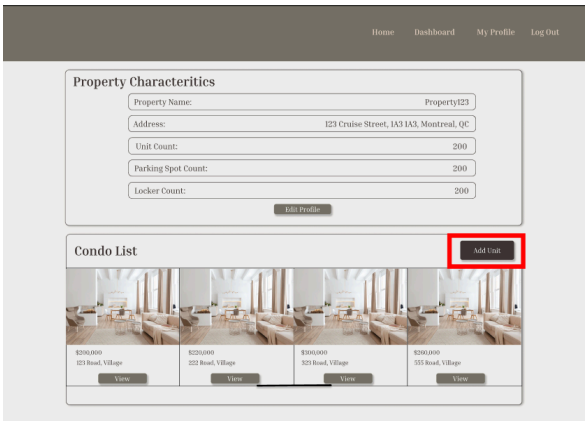
8.1.9.1. Scenario 1: User wants to enter detailed information for a new condo unit



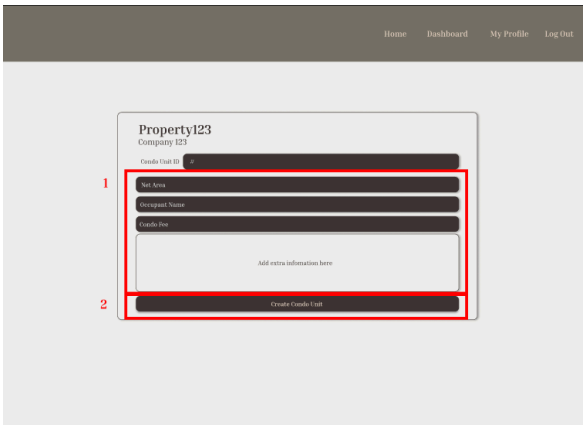
Dashboard Page



Properties List Page



General Property Page



Condo Unit Page

Prerequisite(s):

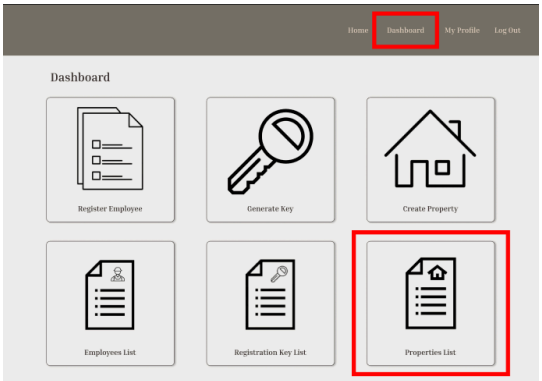
- The user is logged into his account.

Steps:

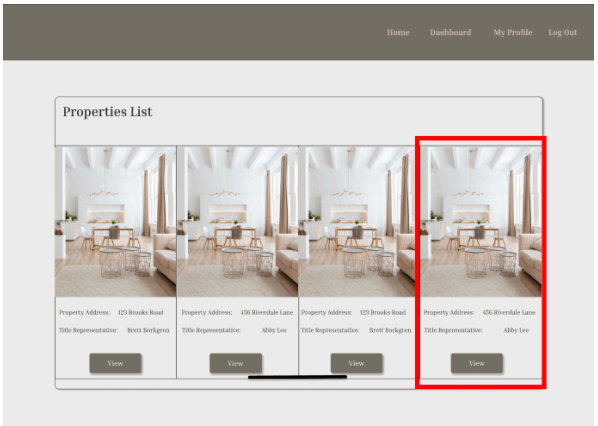
1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks the “Properties List” box.
3. The user scrolls through the list of properties and clicks on the property once found.
4. The user clicks the “Add Unit” button in the Condo List section of the General Property page.

5. The user fills the form for required information and any additional detailed information about the condo unit.
6. The user clicks the “Create Condo Unit” button to save the information.

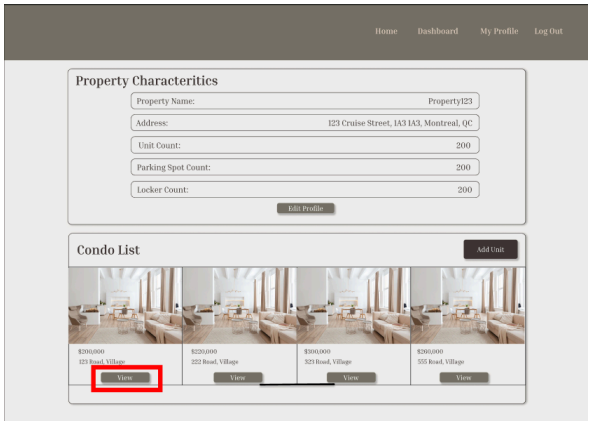
8.1.9.2. Scenario 2: User wants to enter detailed information to an existing condo



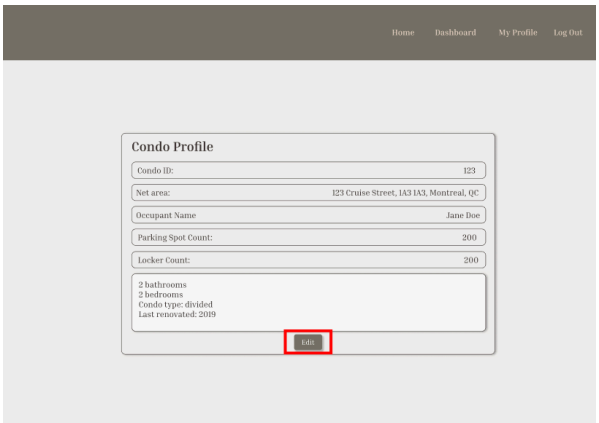
Dashboard Page

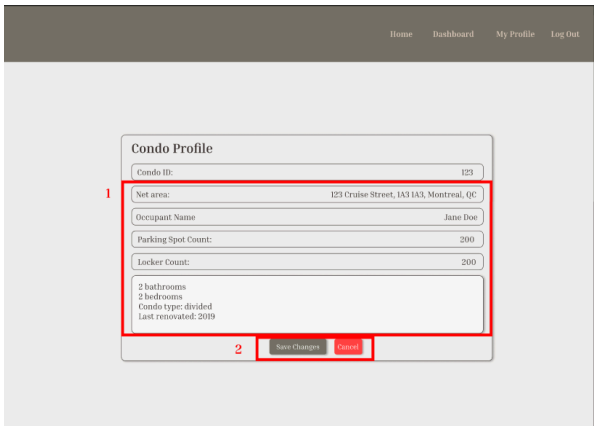


Properties List Page



General Property Page





Condo Profile Page	Condo Profile Page

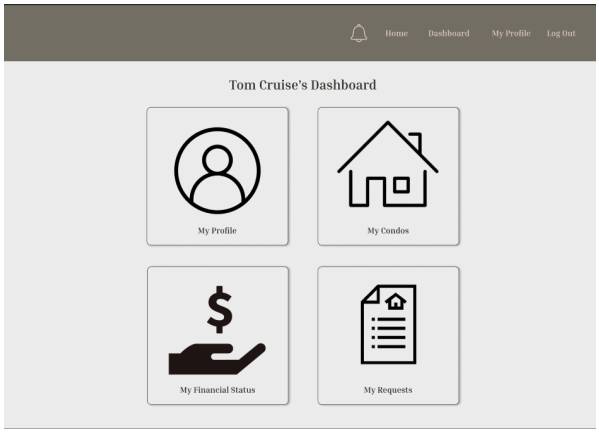
Prerequisite(s):
- The user is logged into his account.

Steps:
1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks the “Properties List” box.
3. The user scrolls through the list of properties and clicks on the property once found.
4. The user clicks the “View” button in the Condo List section of the General Property page.
5. The user clicks the “Edit” button in the Condo Profile Page.
6. The user fills the form for required information and any additional detailed information about the condo unit.
7. The user clicks the “Save Changes” button to save the information.

8.2. UI Prototypes for Sprint 3 User stories

8.2.1. US-017

As a condo owner, I want to have a dashboard of properties (general information, personal profile, condo information, financial status, remaining balance of condo fee payments, status of submitted request) so that I can easily manage and keep track of my properties.



Condo Owner Dashboard

Prerequisite(s):
- User is logged in to their account
- User is registered as a condo owner

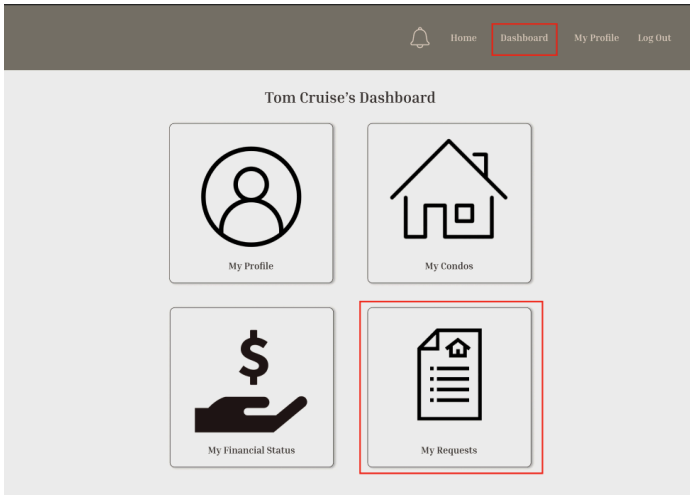
Steps:
1. The user clicks on “Dashboard” in the navigation bar.

8.2.2. US-023

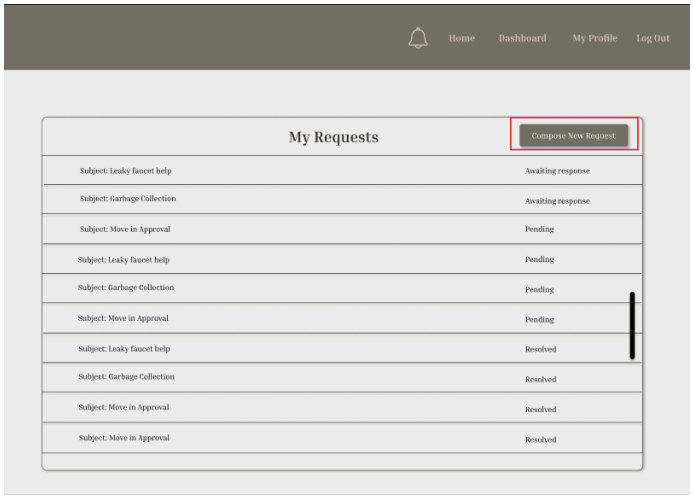
As a condo owner, I want to be able to submit requests, so that I can communicate my needs and issues to the management.

Note: The types of requests consist of:

- *Moving in/out*
- *Intercom changes*
- *Request access*
- *Report a violation*
- *Report deficiency found in common area*
- *Question*

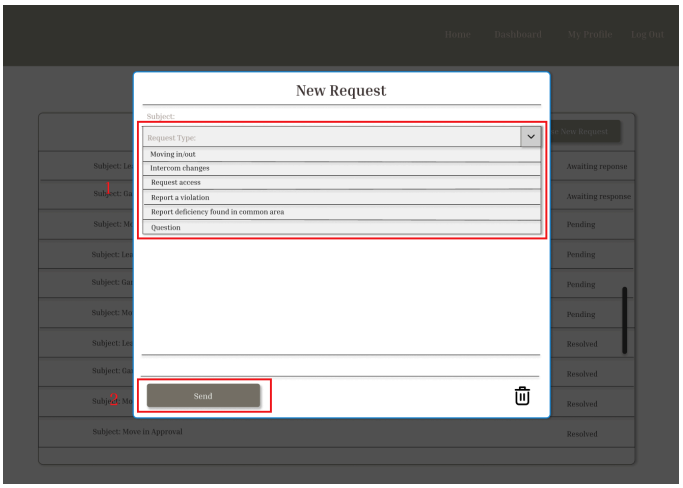


Condo Owner Dashboard



My Requests Page

New Request Pop-up



New Request Pop-up

Prerequisite(s):

- User is logged in to their account

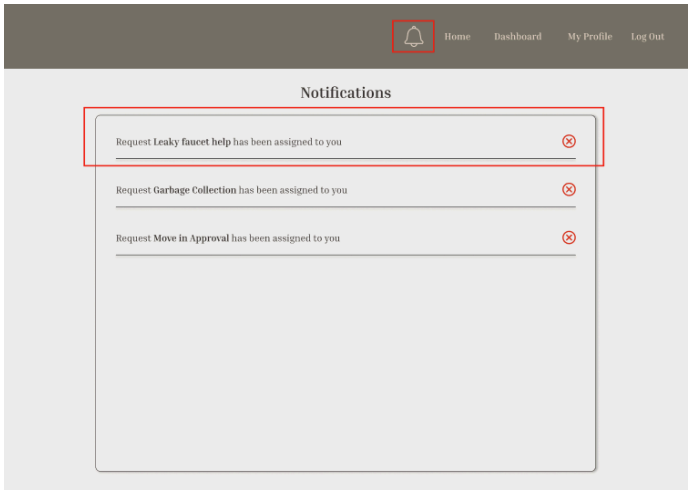
- User is registered as a condo owner

Steps:

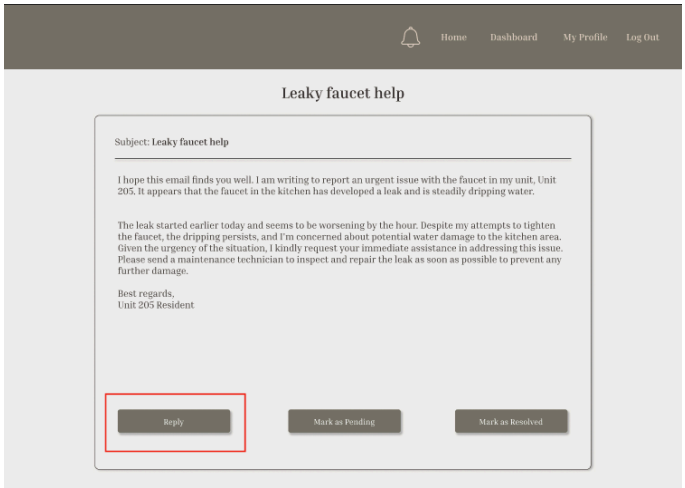
1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks on the “My Requests” box.
3. The user clicks on the “Compose new request” button.
4. The user enters the subject of their request, chooses a request type from the dropdown menu, composes their request and sends it to the condo management company.

8.2.3. US-024

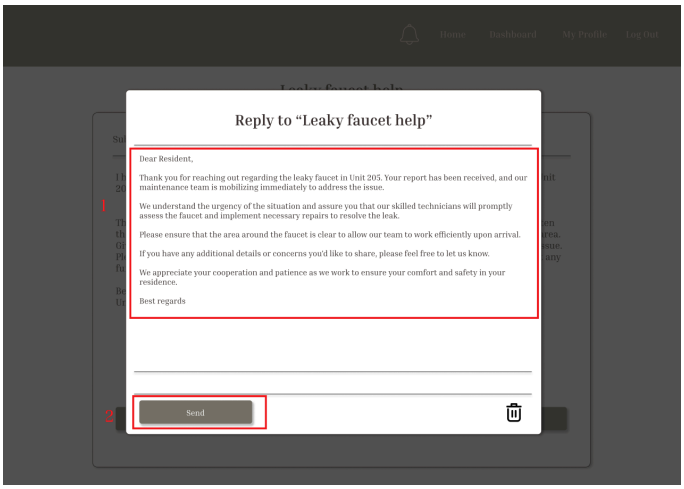
As a condo management employee, I want to be able to see the assigned requests so that I can promptly address and resolve them and stay updated on the status of my assigned requests.



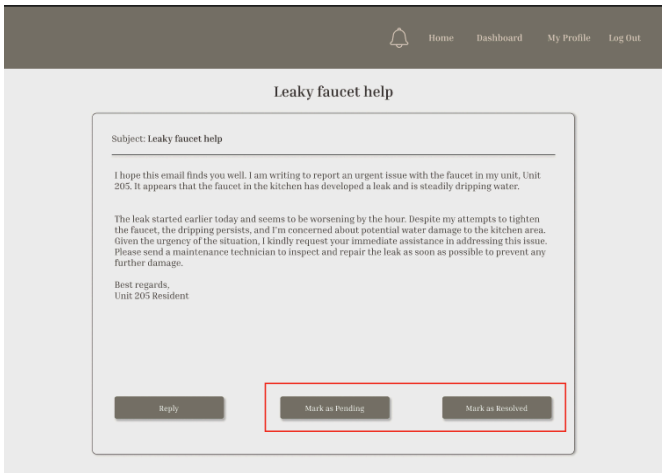
Employee Notification Page



Assigned Requests Page



Compose Reply Pop-up



Assigned Requests Page

Prerequisite(s):

- User is logged into their account
- User is a registered employee of the condo management company

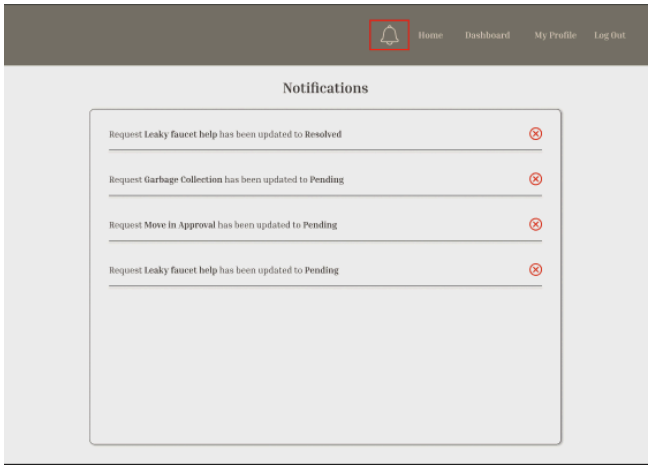
Steps:

1. The user navigates to the Notification page by clicking on the notification icon in the navigation bar.
2. The user clicks on the request that was assigned to them.
3. The user clicks on the “Reply” button in the Requests page of their assigned request.
4. The user composes a response to the condo owner’s request and clicks “Send.”
5. The user navigates to the Requests page of their assigned request and clicks “Pending” or “Resolved” to indicate to the condo user the status of their request.

8.2.4. US-025

As a user, I want to be able to have a notification page so that I can see and stay updated on the latest activities in my submitted or assigned requests.

8.2.4.1. **Condo Owner’s Notification Page**



Condo Owner Notification Page

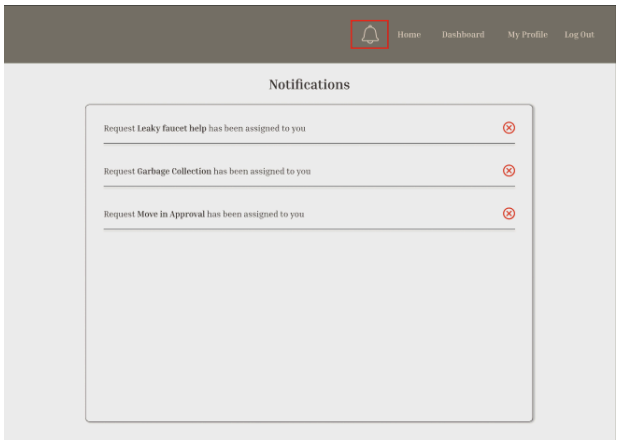
Prerequisite(s):

- User is logged into their account.

Steps:

1. The user navigates to the Notification page by clicking on the notification icon in the navigation bar.
2. The user sees all the updates of their submitted requests.

8.2.4.2. **Company Employee's Notification Page**



Employee Notification Page

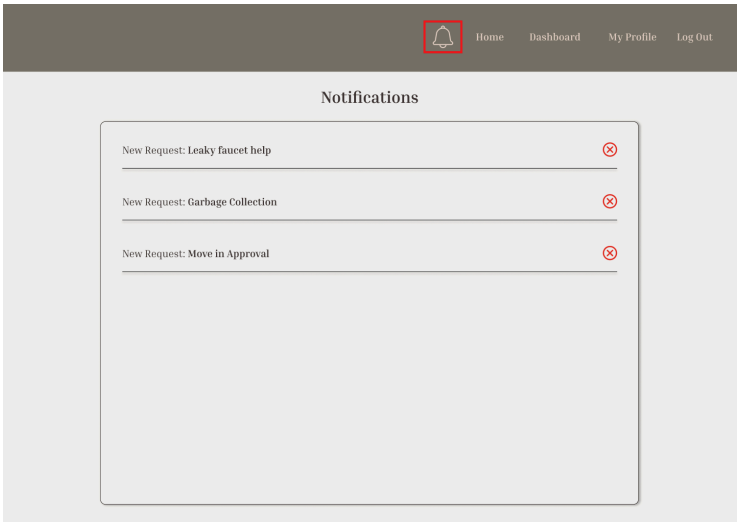
Prerequisite(s):

- User is logged into their account.

Steps:

1. The user navigates to the Notification page by clicking on the notification icon in the navigation bar.
2. The user sees all their assigned requests.

8.2.4.3. **Company Manager's Notification Page**



Manager Notification Page

Prerequisite(s):

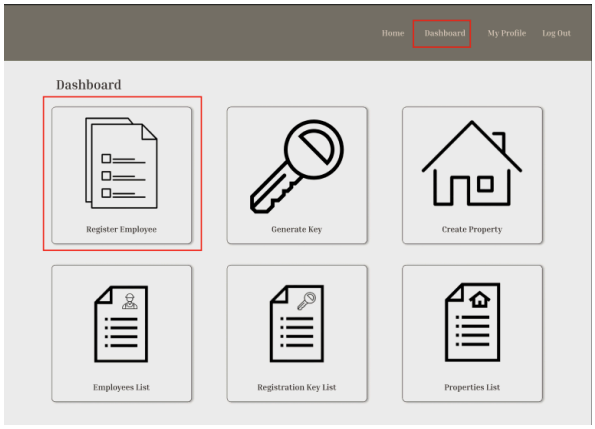
- User is logged into their account.

Steps:

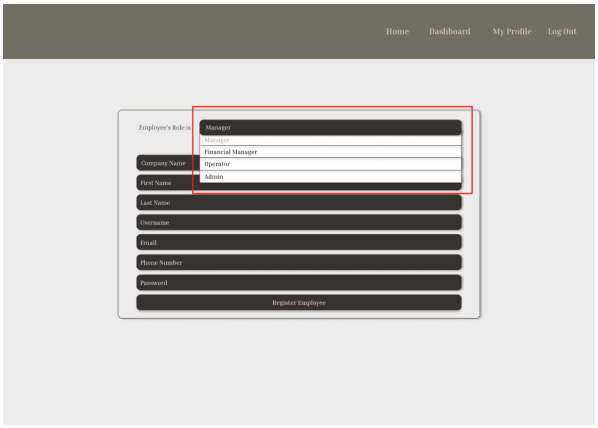
1. The user navigates to the Notification page by clicking on the notification icon in the navigation bar.
2. The user sees all the new requests.

8.2.5. US-038

As a condo management company, I want to be able to set up different roles for different employees so that responsibilities can be assigned to the appropriate employee and property management tasks can be efficiently handled.



Company Dashboard page



Register Employee Page

Prerequisite(s):

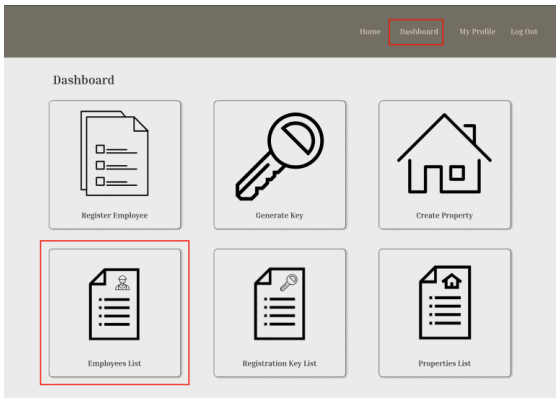
- User is logged into their account

Steps:

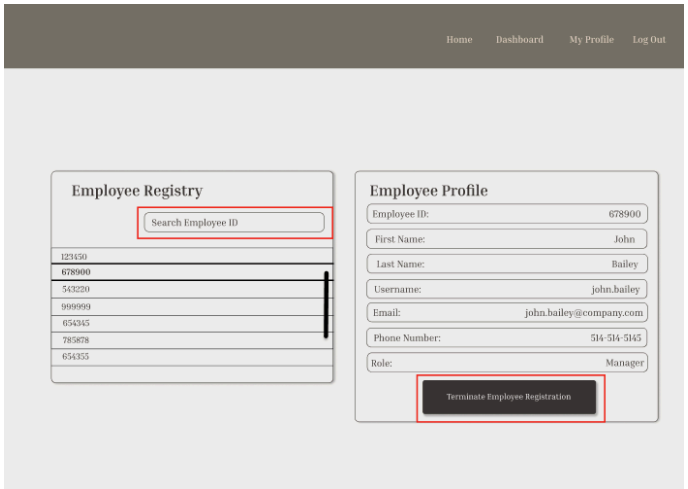
1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks the “Register Employee” box.
3. The user clicks on the dropdown menu to assign a role for the new employee.

8.2.6. US-039

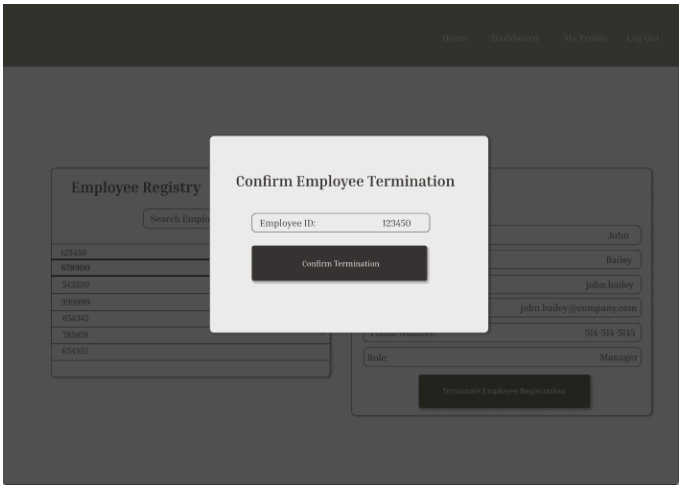
As a condo management company, I want to delete employee profiles, so that I can remove people that are no longer employees within the company’s system.



Dashboard Page



Employees List Page



Confirm Employee Termination

Prerequisite(s):

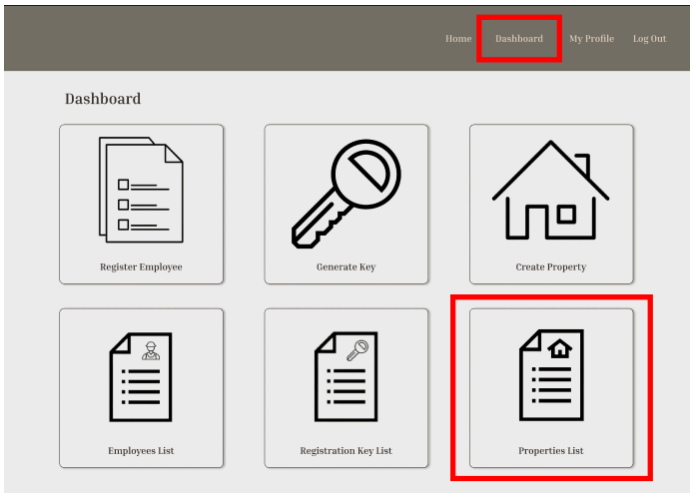
- User is logged into their account

Steps:

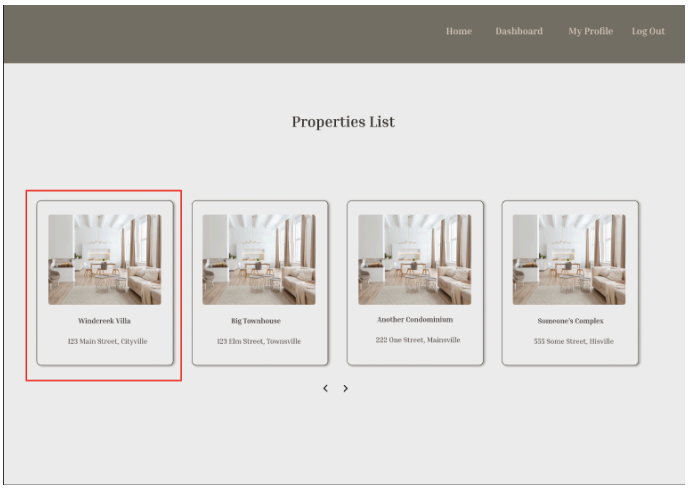
1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks the “Employees List” box.
3. The user scrolls through the list of employee ID’s and clicks on the employee they desire, which displays the employee’s profile.
4. The user clicked on the “Terminate Employee Registration” button in the Employee Profile section of the page.
5. The user clicks the “Confirm Termination” button in the Confirm Employee Termination pop-up.

8.2.7. US-053

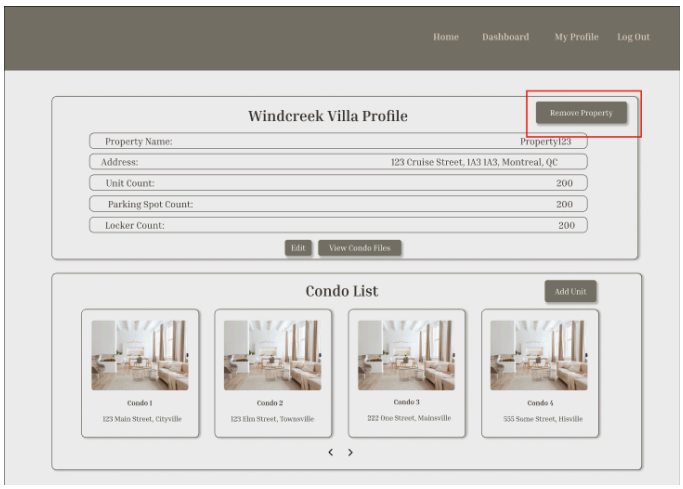
As a condo management company, I want to delete an entry of a property made in the contents of my properties list, so that I can remove properties that are incorrect or no longer under my ownership.



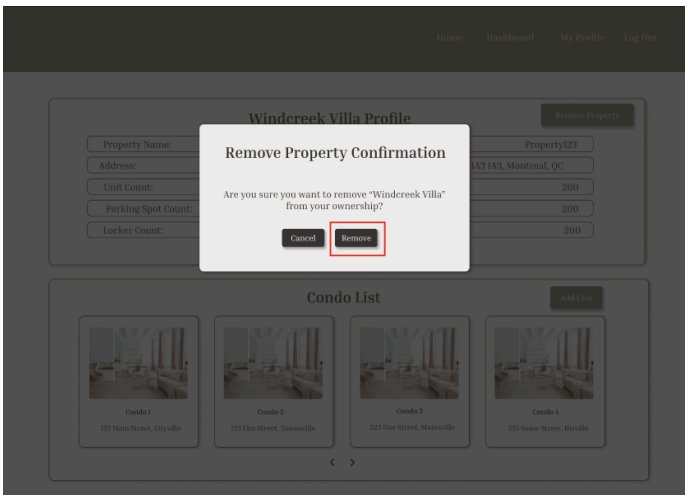
Dashboard Page



Properties List Page



Property Profile Page



Remove Property Confirmation

Prerequisite(s):

- User is logged into their account

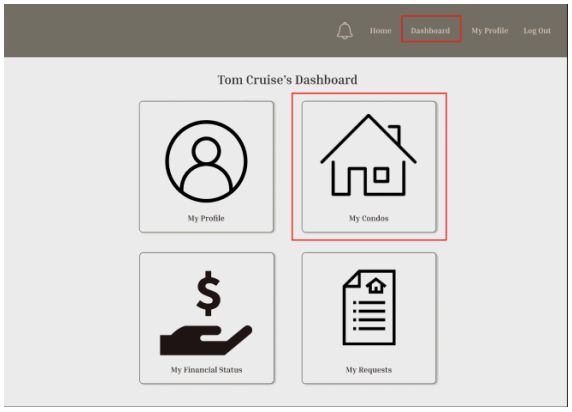
Steps:

1. The user navigates to the Dashboard by clicking on the "Dashboard" tab in the navigation bar.
2. The user clicks the "Properties List" box.
3. The user scrolls through the list of properties and clicks on the property once found.
4. The user clicks the "Remove Property" button in the Property Profile page.

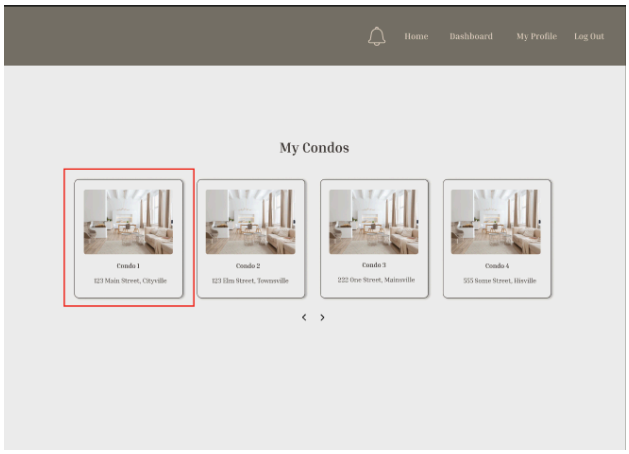
5. The user clicks the “Remove” button to confirm removing the property from their ownership.

8.2.8. US-054

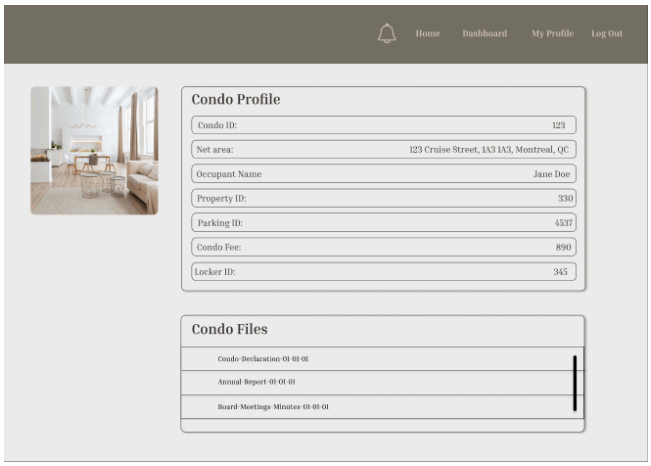
As a condo owner, I want to be able to view the profile for a condo that I own (size, parking spot id, locker id, condo fee) so that I can have a comprehensive overview of all essential details pertaining to the condo.



Condo Owner Dashboard Page



My Condos Page



Condo Profile Page

Prerequisite(s):

- User is logged into their account

Steps:

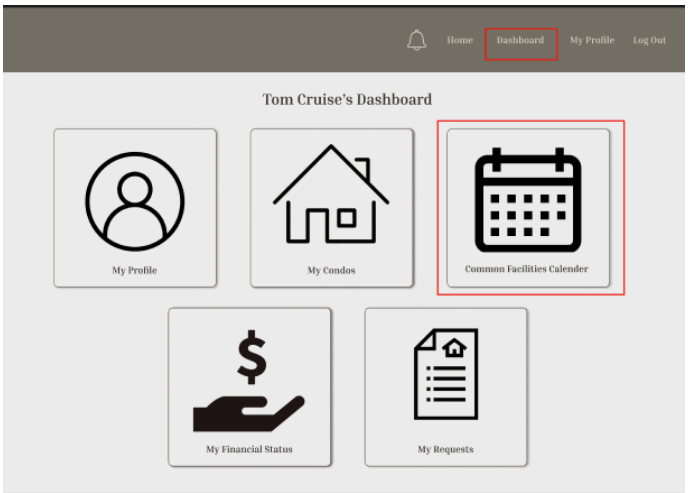
1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks the “My Condos” box.
3. The user scrolls through the list of condos and clicks on the condo once found.

4. The user views the condo profile with all the relevant information pertaining to it.

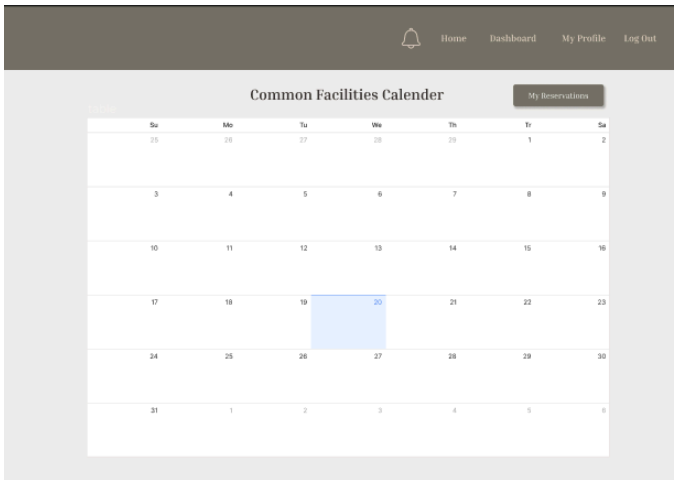
8.3. UI Prototypes for Sprint 4 User stories

8.3.1. US-026

As a condo owner, I want to be able to reserve common facilities in a calendar-like interface so that I can easily plan and schedule my use of these facilities.



Condo Owner Dashboard Page



Common Facilities Calendar Page

Prerequisite(s):

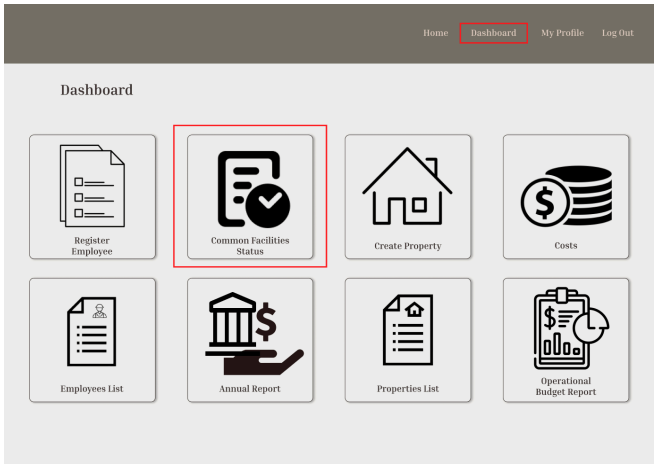
- User is logged in to their account.
- User is registered as a condo owner.

Steps:

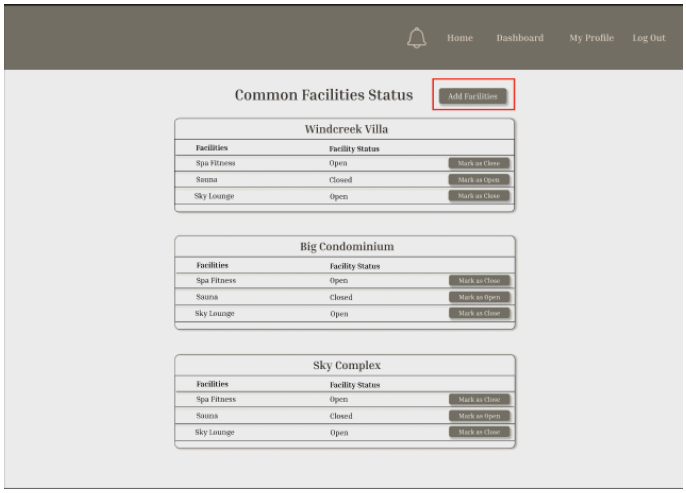
1. The user navigates to the Dashboard by clicking on the "Dashboard" tab in the navigation bar.
2. The user clicks on the "Common Facilities Calendar" box.
3. The user navigates to the Common Facilities Calendar page.

8.3.2. US-027

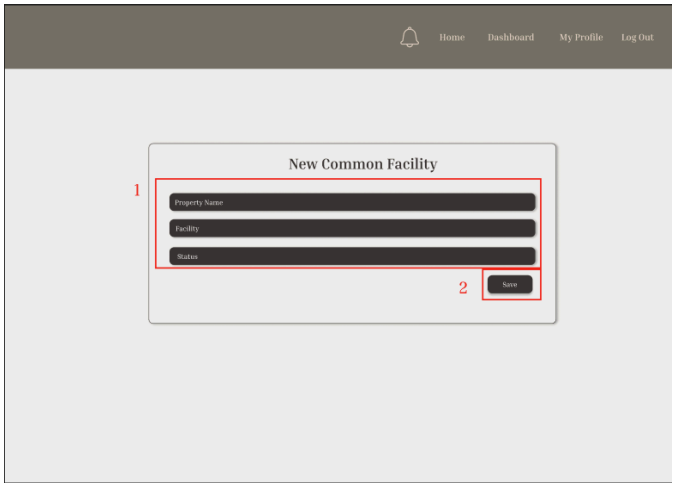
As a condo management company, I want to be able to set up a common facility which requires reservations so that occupants can utilize shared amenities fairly and in an organized manner.



Company Dashboard Page



Common Facilities Status Page



New Common Facility Page

Prerequisite(s):

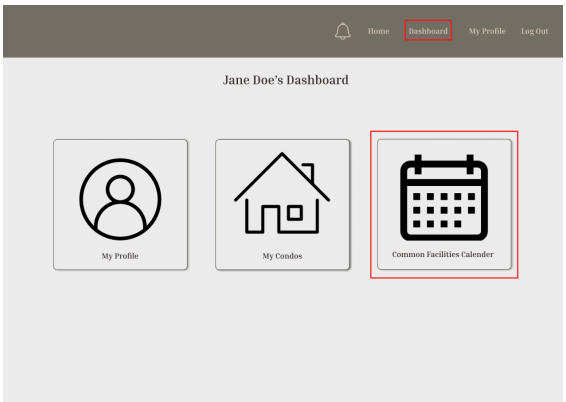
- User is logged in to their account.
- User is registered as a condo management company.

Steps:

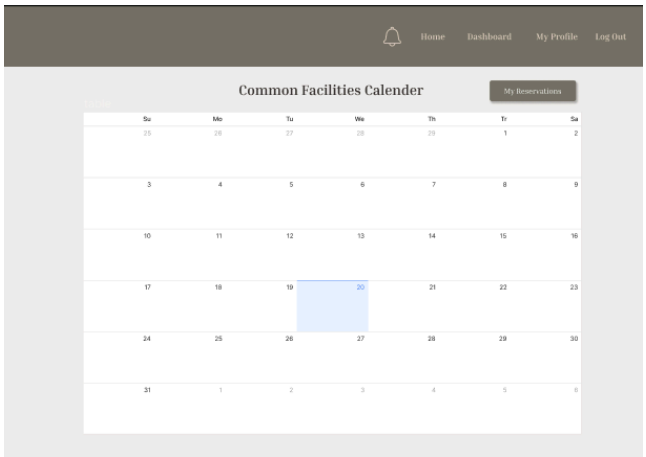
1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks on the “Common Facilities Status” box.
3. The user clicks on the “Add Facility” button.
4. The user fills in the form with the property name, new facility and status.
5. The user clicks on the “Save” button.

8.3.3. US-028

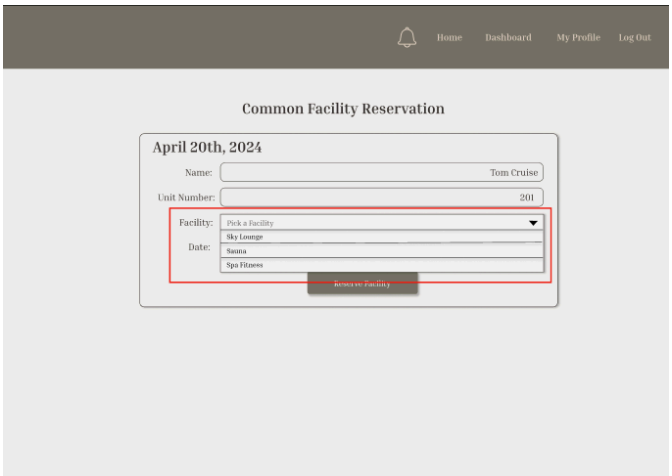
As a rental user, I want to be able to see the availability of common facilities so that I can see if any availability aligns with my schedule.



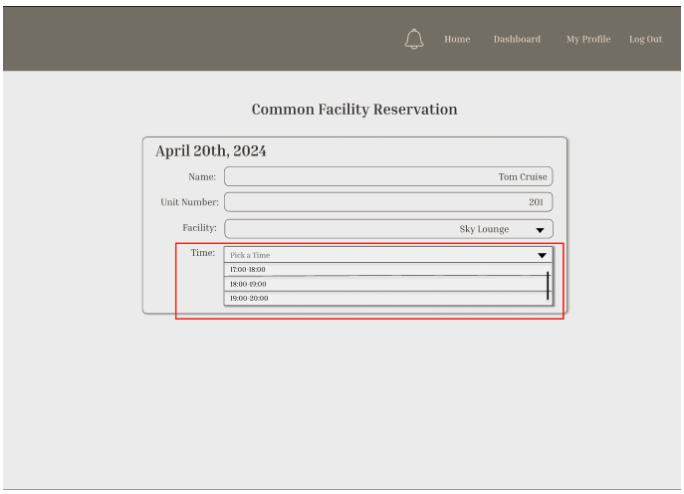
Rental User Dashboard Page



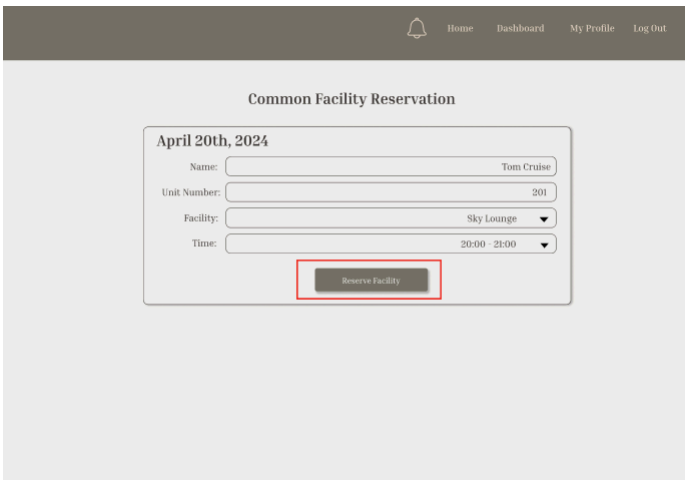
Common Facilities Calendar Page



Common Facility Reservation Page



Common Facility Reservation Page



Common Facility Reservation Page

Prerequisite(s):

- User is logged into their account.
- User is registered as a rental user.

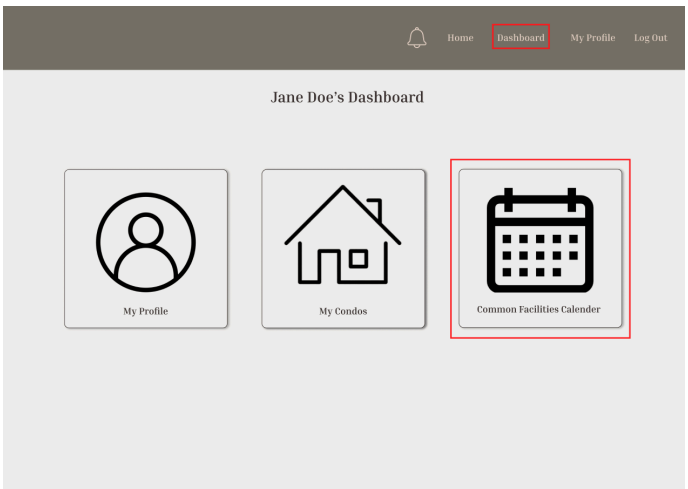
Steps:

1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks on the “Common Facilities Calendar” box.
3. The user navigates to the Common Facilities Calendar page and clicks the date of their choice on the calendar to reserve.
4. The user selects the Facility of their choice from the dropdown.
5. The user selects the available times associated with that facility on this particular date.
6. The user clicks the “Reserve Facility” button.

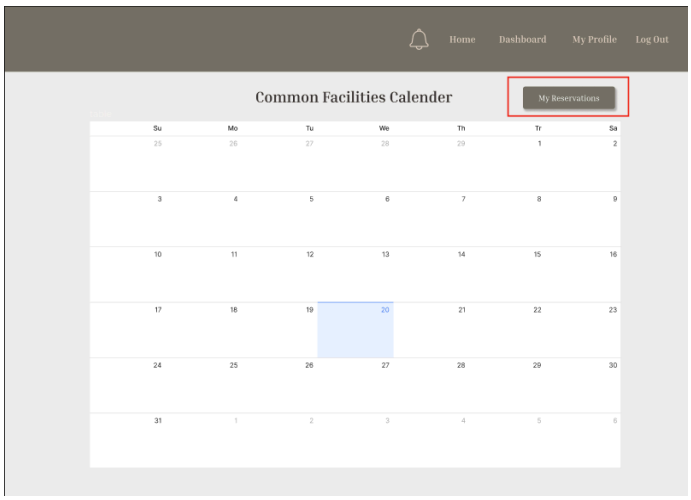
8.3.4. US-029

As a rental user, I want to be able to modify or cancel my existing reservation for a common facility, so that I am no longer committed to a reservation that I cannot go to in the case of changes in my plans or scheduling conflicts.

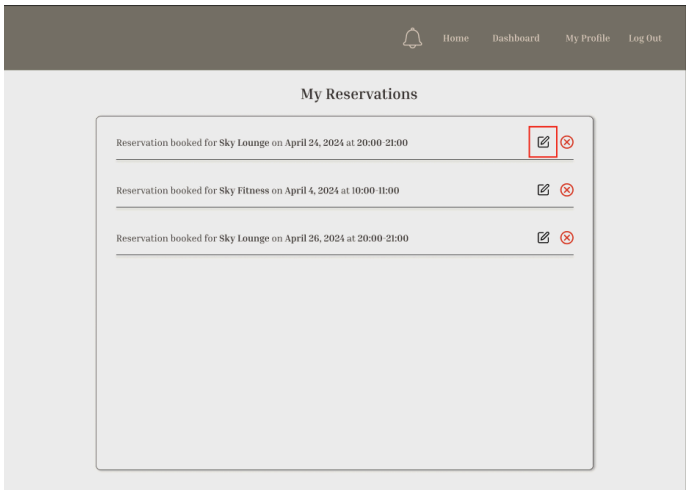
8.3.4.1. **Modify Existing Reservation for a Common Facility**



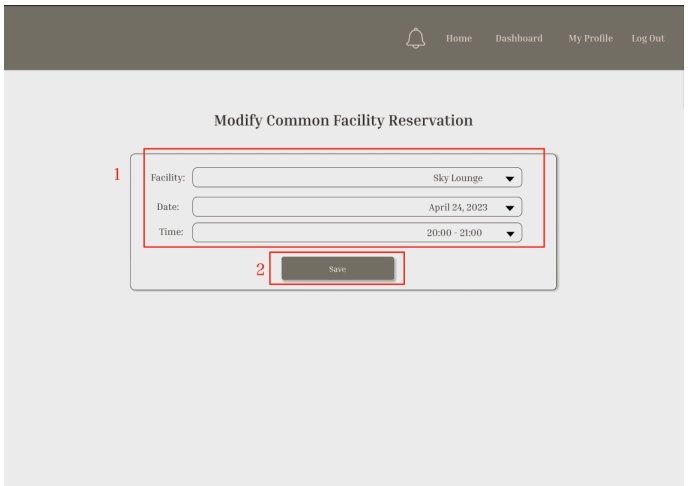
Rental User Dashboard Page



Common Facilities Calendar Page



My Reservations Page



Modify Common Facility Reservation Page

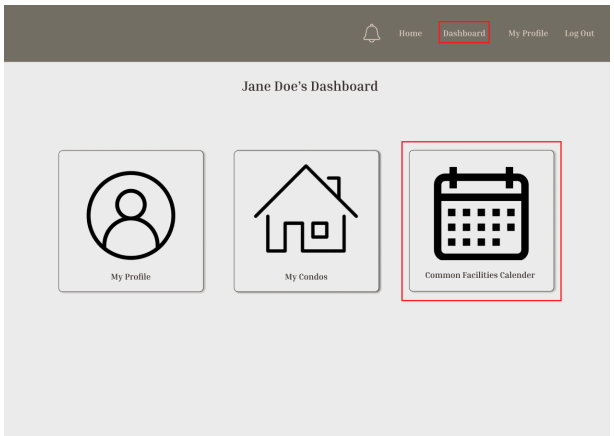
Prerequisite(s):

- User is logged into their account.
- User is registered as a rental user.

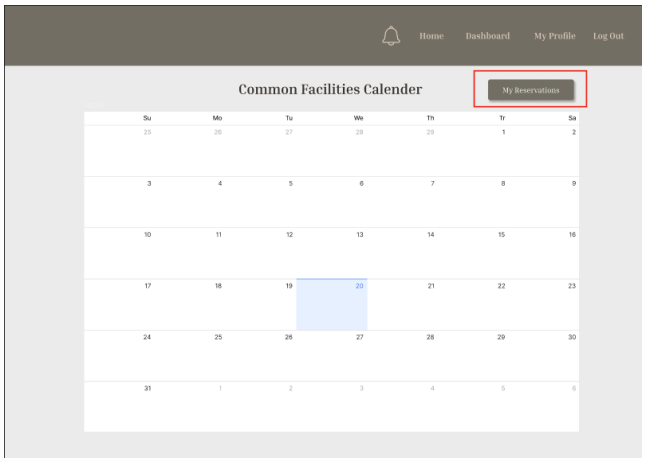
Steps:

1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks on the “Common Facilities Calendar” box.
3. The user clicks on the “My Reservations” button.
4. The user clicks on the edit icon.
5. The user fills out the form to modify their reservation.
6. The user clicks the “Save” button.

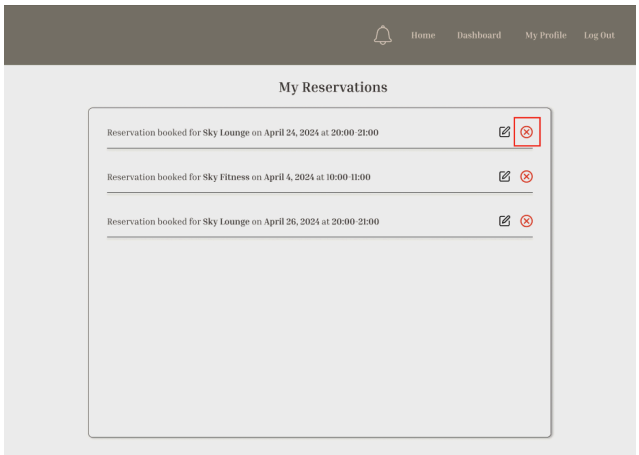
8.3.4.2. **Delete Existing Reservation for a Common Facility**



Rental User Dashboard



Common Facilities Calendar Page



My Reservations Page

Prerequisite(s):

- User is logged into their account.
- User is registered as a rental user.

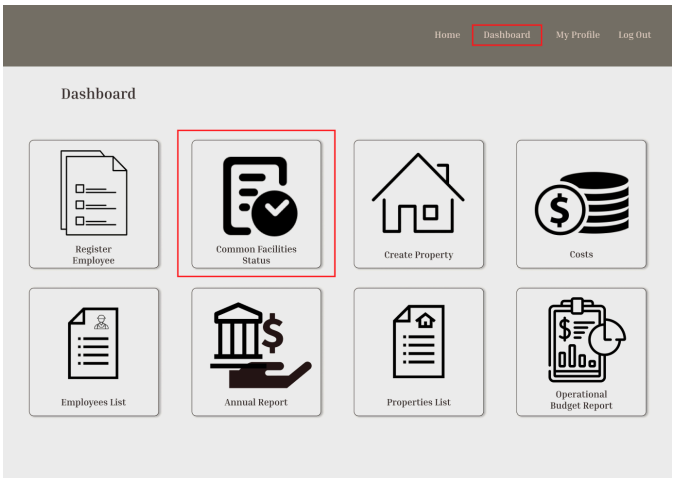
Steps:

1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks on the “Common Facilities Calendar” box.
3. The user clicks on the “My Reservations” button.
4. The user clicks on the delete icon.

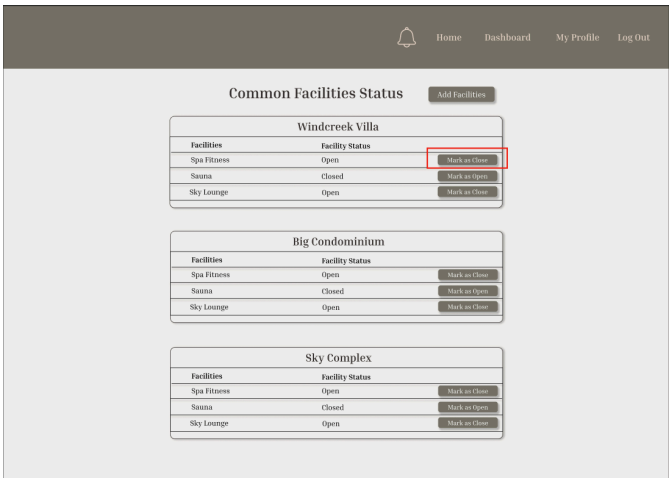
8.3.5. US-030

As a condo management company, I want to be able to disable the availability of a common

facility, so that occupants cannot reserve the facility if it is already reserved and are aware of its unavailability.



Company Dashboard Page



Common Facilities Status Page

Prerequisite(s):

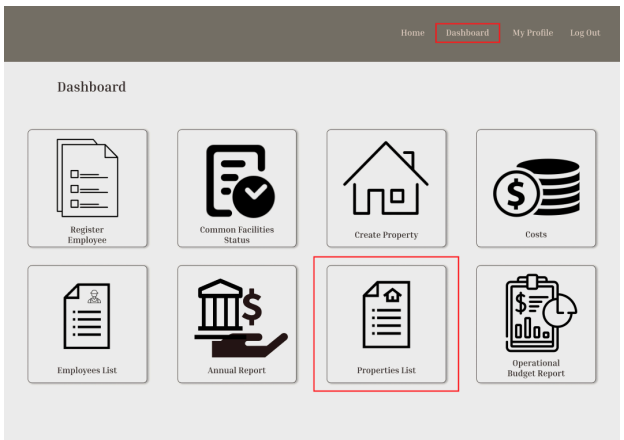
- User is logged into their account.
- User is registered as a condo management company.

Steps:

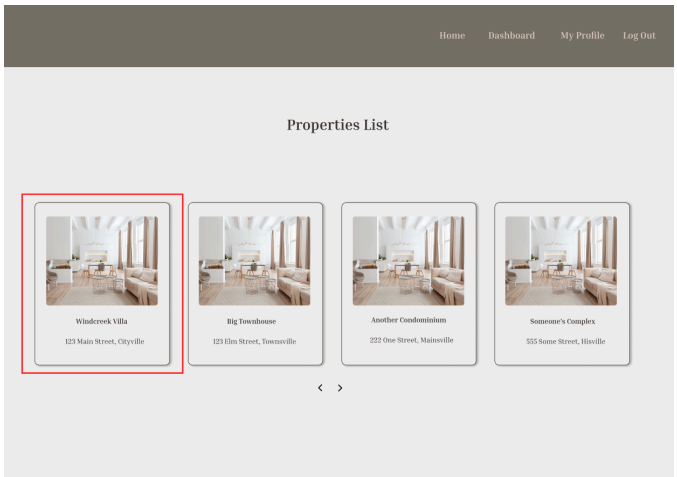
1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks on the “Common Facilities Status” box.
3. The user clicks on the “Mark as Closed” button on the appropriate facility.

8.3.6. US-031

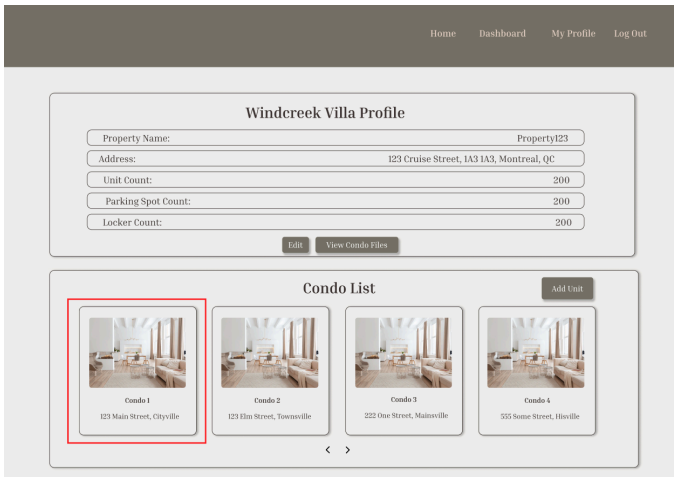
As a condo management company, I want to be able to enter the condo fee per square foot, per parking spot so that accurate calculations can be made and presented to condo owners.



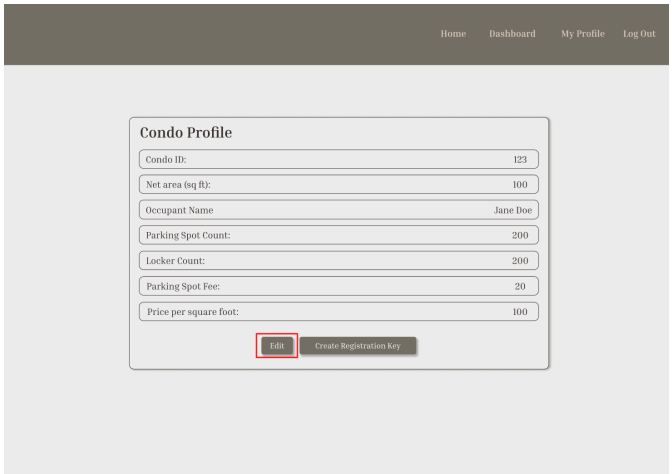
Company Dashboard



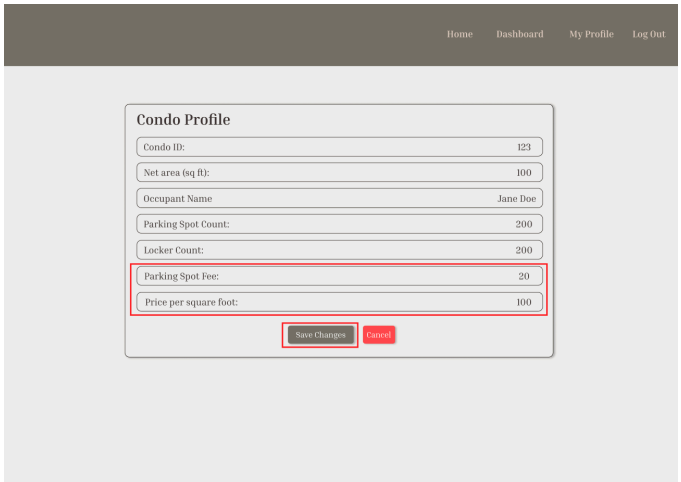
Properties List Page



Property Profile Page



Condo Profile Page



Edit Condo Profile Page

Prerequisite(s):

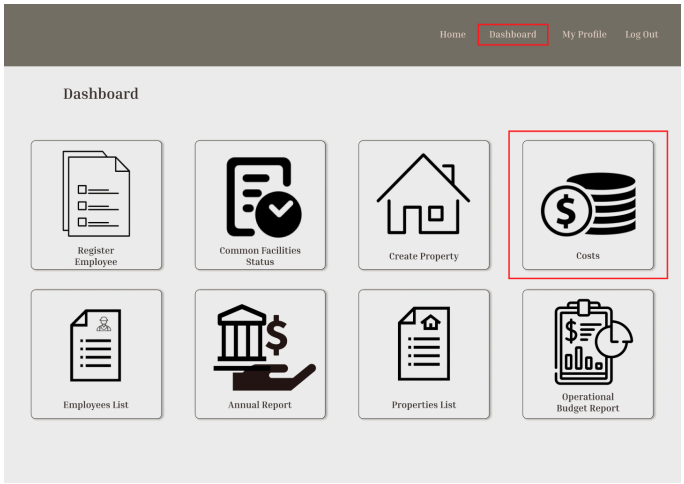
- User is logged into their account.
- User is registered as a condo management company.

Steps:

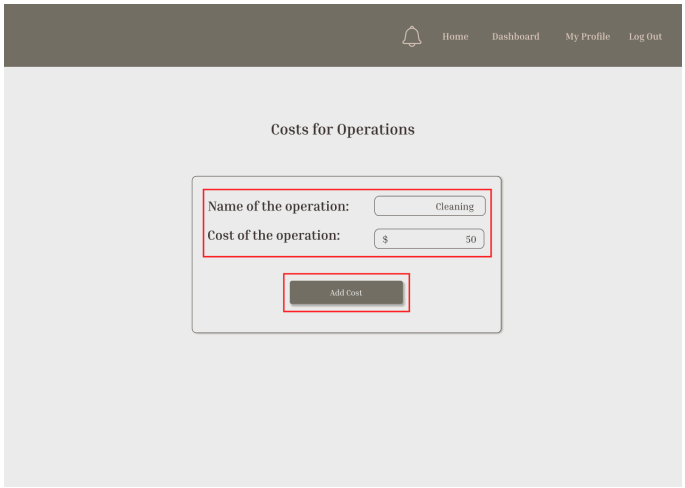
1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks on the Properties List box on the Dashboard.
3. The user clicks on the desired property in the Properties List.
4. The user clicks on the desired condo under the condo list on the Property Profile Page.
5. The user clicks on the Edit button on the Condo Profile Page.
6. The user enters the parking spot fee and the price per square foot.
7. The user clicks on the Save Changes button.

8.3.7. US-032

As a condo management company, I want to be able to enter the cost for each operation so that I can keep track of financial expenses and provide transparency to owners.



Company Dashboard



Costs Page

Prerequisite(s):

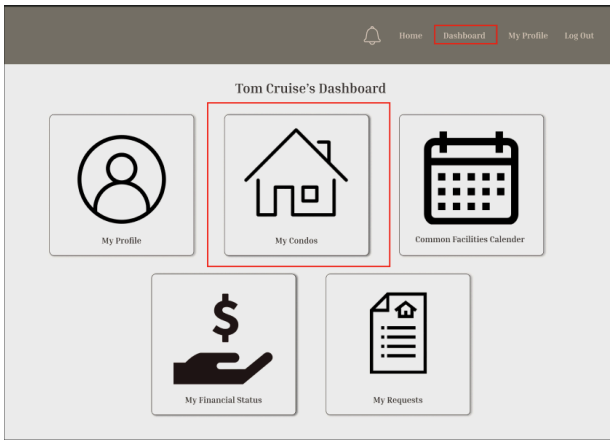
- User is logged into their account.
- User is registered as a condo management company.

Steps:

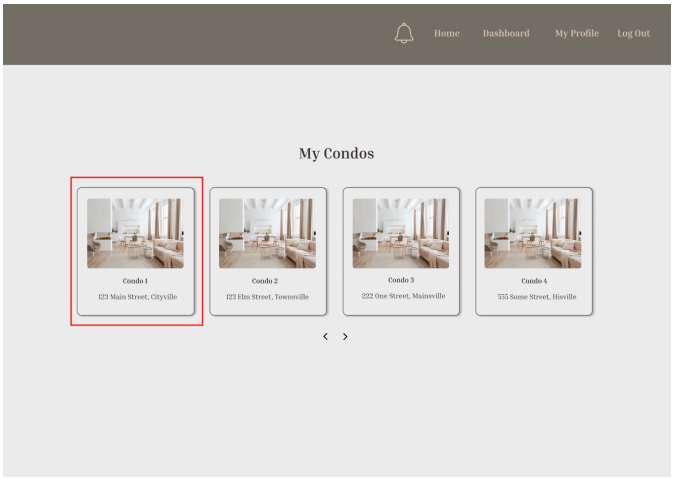
1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks on the Costs box on the Dashboard.
3. The user enters the name and the cost of the operation, and clicks on “Add Cost”.

8.3.8. US-033

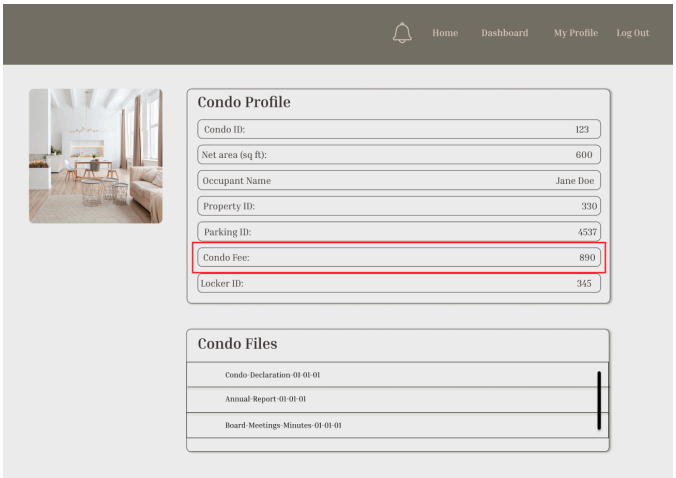
As a condo owner, I want to be able to see the calculated condo fee for each unit so that I am aware of the financial obligation and can plan accordingly.



Condo Owner Dashboard



My Condos Page



Condo Profile Page

Prerequisite(s):

- User is logged into their account.
- User is registered as a condo owner.

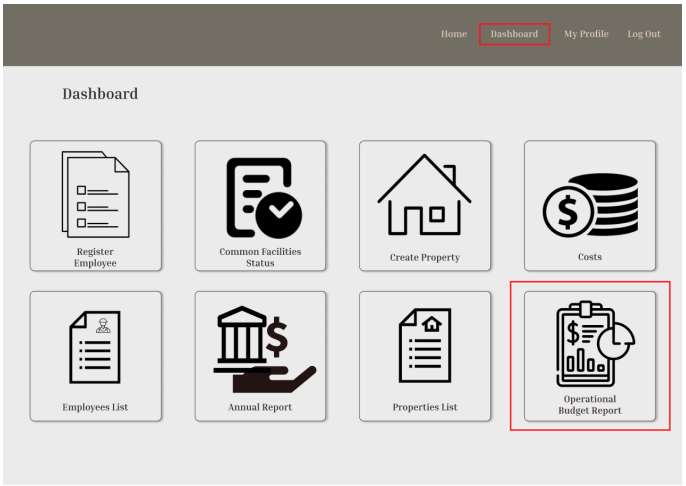
Steps:

1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks on the My Condos box on the Dashboard.

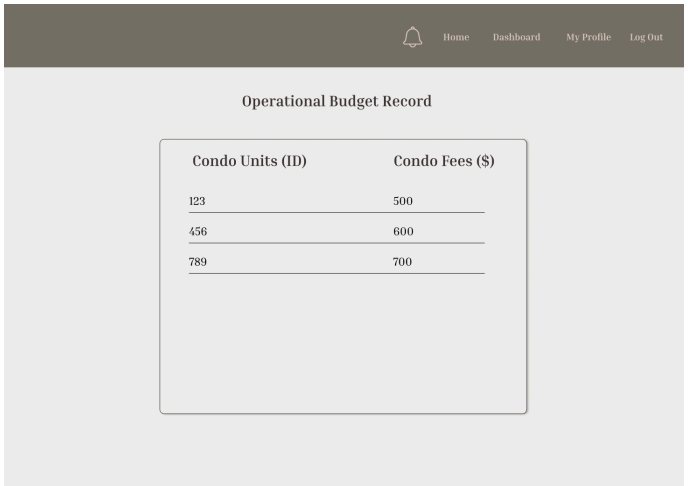
3. The user clicks on the desired condo in the Condos List.
4. The user can see the condo fee next to the Condo Fee tag.

8.3.9. US-034

As a condo management company, I want to be able to see a record of the operational budget, which includes the total condo fees collected from condo owners, so that I can manage the financial resources of the company.



Company Dashboard



Operational Budget Report Page

Prerequisite(s):

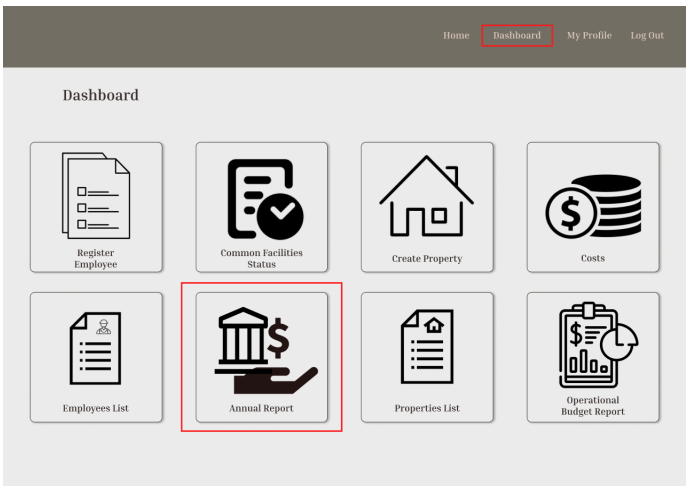
- User is logged into their account.
- User is registered as a condo management company.

Steps:

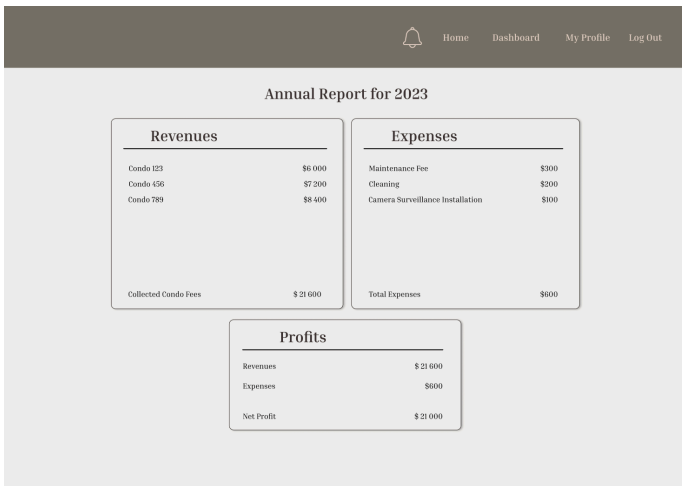
1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks on the Operational Budget Report box on the Dashboard.
3. The user can see the report on the Operational Budget Report page.

8.3.10. US-035

As a condo management company, I want to be able to see an annual report so that I can assess the financial performance and overall status of the property.



Company Dashboard



Annual Report Page

Prerequisite(s):

- User is logged into their account.
- User is registered as a condo management company.

Steps:

1. The user navigates to the Dashboard by clicking on the “Dashboard” tab in the navigation bar.
2. The user clicks on the Annual Report box on the Dashboard.
3. The user can see the generated annual report on the Annual Report page.