SOEN 390

Team 15 - Deliverable 3

Rently

Winter 2023

Student	Student ID
Abdelkader Habel	40209153
Adam Boucher	40165035
Adel Bouchatta	40175598
Anes Khadiri	40159080
Chems-Eddine Saidi	40192094
Francesco Ferrato	26642152
Omar Mohammad	40162541
Oussama Cherifi	40212275
Zakaria El Manar El Bouanani	40190432

Table Of Contents

Table Of Contents **2**
Product Vision Statement **5**
1. Introduction 5
2. Positioning 6
2.1. Problem Statement 6
2.2. Product Position Statement 7
3. Stakeholder and User Descriptions 8
3.1. Stakeholder Summary 8
3.2. User Summary 9
3.3. User Environment 9
3.4. Key Stakeholder or User Needs 11
3.5. Alternatives and Competition 13
4. Product Overview 14
4.1. Product Perspective 14
4.2. Assumptions and Dependencies 14
5. Product Features 15
Core Functionalities: 15
Additional Features: 17
Future Enhancements: 17
6. Other Product Requirements 17
Standards, Hardware, or Platform Requirements 17
Performance Requirements 18
Environmental Requirements 18
Quality Attributes 18
Design and External Constraints 19
Documentation Requirements 19
Priority of Requirements 19
Sprint 3 User Stories Backlog **20**
Software Architecture Document **22**
1. Introduction 22
1.1 Identifying information 22
1.2 Overview 22
2. Stakeholders and concerns 22
2.1 Stakeholders 22
2.2 Concerns 22

3. Viewpoint specification 23
3.1 Overview 23
3.2 Model kinds 23
4. Views 23
4.1 View: Condo management system architecture 23
4.1.1 Domain model 23
4.1.2 Component and architecture diagram 25
4.1.3 Use Case diagram 26
4.1.4 Activity diagram 27
4.1.5 Class diagram 32
4.1.6 Backend architecture diagram 33
4.1.7 Deployment diagram 34
Risk Management Plan **35**
Scope 35
Disclaimer 35
1.0 Risk Identification 35
2.0 Risk Analysis 36
3.0 Risk Matrix 37
4.0 Risks & Mitigation Strategies 38
5.0 Risk Analysis 39
Testing plan **46**
Methodology 46
Tools used 46
JUnit 46
Jest 46
SonarQube 47
Metrics/Success Criteria 47
Test Results and Coverage: 48
Backend - JUnit & Mockito 48
Frontend - Jest 49
Moving Forward/Sprint 4 49
Rently Sprint 3 Retrospective **50**
Introduction 50
What went wrong 50
What went right 51
Conclusion 52
Release Plan - Sprint 4 User Stories **53**
Table of stories in sprint 4 53
Progression chart 54

UI Prototypes for Sprint 4 **55**
Overview 55
Feature List and Status 55
1. REN-70 Company has access to Annual Financial Report 55
2. REN-78 Owner Details Admin View 56
3. REN-63 Set roles for employees 57
4. REN-64 Company Admin can View Status of Requests Made 58
4. REN-66 Owner Can Submit Requests 59
5. REN-66 can View Status of Requests Made 60
6. REN-78 Manage Owners as Admin 60
7. REN-61: File Upload 61
8. REN-62 Enter operational costs 62
Next Steps 62
Deadlines 62
Additional Notes 62
Code Management **63**
1. Quality of source code reviews 63
2. Correct use of design patterns 64
A. Facade design pattern: 64
B. Repository design pattern : 65
C. Data transfer Object (DTO) design pattern: 67
3. Respect to code conventions 68
4. Design quality (number of classes/packages, size, coupling, cohesion) 69
5. Quality of source code documentation 69
6. Refactoring activity documented in commit messages 70
7. Quality/detail of commit messages 70
8. Use of feature branches 71
9. Atomic commits 72
10. Bug reporting 72
11. Use of issue labels for tracking and filtering 73
12. Links between commits and bug reports/features 74

Product Vision Statement

1. Introduction

This Vision Document outlines the strategic framework and operational blueprint for the Rently System, an innovative solution designed to revolutionize condominium management. Addressing the critical gap in current management practices, the Rently System integrates property and user profiles, financial records, facility reservations, service requests, and notifications into a single, user-friendly platform. By providing real-time updates and comprehensive management capabilities, it aims to significantly enhance efficiency, accuracy, and user satisfaction across the condominium management ecosystem.

The document details the system’s positioning, highlighting its unique approach to solving the fragmented management landscape, and delineates the key features and functionalities that set it apart from traditional management solutions. It identifies the primary stakeholders and users, including condo management companies, their employees, and the condo residents, and articulates how the Rently System meets their varied needs.

Further, it provides a succinct overview of the product architecture, anticipated benefits, and the competitive advantage it offers. Assumptions, dependencies, and a comparative analysis with existing alternatives are also presented, underscoring the system's potential to redefine condominium management.

Through this introduction, we aim to provide stakeholders with a clear understanding of the Rently System's objectives, its value proposition, and the transformative impact it promises for condominium management operations.

2. Positioning

2.1. Problem Statement

The problem of	Providing an all-in-one solution for managing all the condominium operations. This includes maintaining user and property profiles, financial records, facility reservations, service requests, and notifications.
affects	The condo management company, the condo management company employees, the condo owners, and the condo renters.
the impact of which is	The absence of an all-in-one solution causes unreliable management of all condominium operations. This leads to inaccuracies in user and property profiles, errors in financial records, difficulties in securing facility reservations, delays in service requests, and inconsistent notifications.
a successful solution would be	The implementation of a comprehensive solution that facilitates efficient management of property and user profiles, financial records, facility reservations, service requests, and notifications. This all-in-one solution should be user friendly, accessible across various devices and supporting a minimum of two different languages.

2.2. Product Position Statement

For	The condo management company employees, the condo owners, and the condo renters.
Who	All users have access to their assigned profiles, their financial records, the facility reservation calendar, the service request page, and notification page.
The Rently System	is a software product.
That	Updates all users in real time about all condominium operations. This includes their assigned profiles, their financial records, the facility reservation calendar, the service request page, and notification page.
Unlike	The traditional condo management systems that don't have all operations in an all-in-one user-friendly solution. This lacks real time cohesion between all condominium operations.
Our product	Updates all users in real time about all condominium operations. This includes their assigned profiles, their financial records, the facility reservation calendar, the service request page, and notification page. This provides users with a sense of security.

3. Stakeholder and User Descriptions

3.1. Stakeholder Summary

Name	Description	Responsibilities
Condo Company	Oversees management and operations of condominium properties.	Guaranteeing that the software system works as intended to facilitate efficient condominium management.
IT Department	Management of technical infrastructure and support.	Responsible for the system’s technical stability, security, and ongoing maintenance, including updates and support.
Government Agency	Regulatory body for condominium management.	Ensures the system’s compliance with local regulations.
Investors/Board	Financial stakeholders in the condominium management company.	Invests in the system and expects it to yield operational efficiency and profit. Influences strategic direction and decision-making.
Owner/Renter Association	Represents the collective interests of condo owners and renters.	Advocates for owners' and renters' rights, ensures their needs are met by the management system, and influences policy.

3.2. User Summary

Name	Description	Responsibilities	Stakeholder
Administrative Employee	Handles administrative tasks within condo management.	This involves creating property profiles, distributing registration keys, and setting up different roles for different employees.	Condo Company
Operational Employee	Manages daily operations and finance within the condo complexes.	Utilizes the system to streamline maintenance scheduling, handle financial transactions, and manage daily condominium operations.	Condo company
Public User	Individuals that use the condominium management system.	Engages with the system to view and update personal profiles, register as owners/renters, and receive notifications.	Owner/Renter association
Owner or Renter	Owners or renters of condominium units.	Utilizes the system to view property information, make facility reservations, and make service requests.	Owner/Renter association

3.3. User Environment

Administrative Employee: Administrative employees independently manage a variety of detailed tasks, including creating comprehensive property profiles, inputting condo unit details, and uploading relevant documents. The duration of these tasks range from minutes to several hours based on complexity. They are also responsible for distributing registration keys to link units with owner or renter profiles and assigning roles to operational employees. Work is performed in an office setting, requiring effective multitasking skills and a stable internet connection on platforms like Windows or macOS. Anticipated system updates, such as the introduction of forums, event organization tools, Single Sign-On functionalities, and support for multiple

languages, are set to streamline administrative workflows. This will help integrate the system with existing accounting and CRM platforms.

Operational Employee: Operational employee's involvement in tasks ranges widely, tackling immediate fixes to longer-term projects, often on an individual basis or through teamwork. Their reliance on iOS and Android mobile devices stems from the need to stay connected while frequently on the move, addressing tasks in various locations, some of which may suffer from poor connectivity. This mobility underlines the importance of a system optimized for mobile access, allowing them to efficiently manage maintenance and operations regardless of their physical location. Anticipated system updates, such as the introduction of forums, event organization tools, Single Sign-On functionalities, and support for multiple languages, are set to streamline operational workflows.

Public User: Individual public users utilize the system for personal activities, like setting up their profiles with essential details such as a profile picture, username, email, and phone number, a quick process usually done in a few minutes. To become recognized as condo owners or renters, they're required to input a registration key given by the condo management company, which is similarly a brief procedure. They need system access across different settings, using web browsers and mobile applications on existing platforms, with plans to broaden access to more platforms and include additional languages. Future enhancements of the system are set to introduce features like Single Sign-On, community forums, event planning capabilities, and special promotions.

Owner or Renter: Owners and renters use the system personally for a range of tasks, from viewing detailed dashboards of their properties to submitting various requests. The dashboard provides a comprehensive overview, including personal profiles, condo details, financial status , and the status of submitted requests. Submitting requests, such as elevator reservations for moving, intercom changes, access needs (fobs, keys), reporting violations or deficiencies in common areas, or general inquiries, is streamlined for efficiency. Each request is promptly assigned to the appropriate operational employee based on its nature, ensuring a responsive and personalized management experience. Future system enhancements will focus on fostering community engagement through forums, event planning, and exclusive offers. The introduction of Single Sign-On functionality is expected to simplify access across various environments, supported on current web and mobile platforms with plans to expand to additional platforms and languages, enhancing the user experience.

3.4. Key Stakeholder or User Needs

Administrative Employee:

Need	Priority	Concerns	Current Solution	Proposed Solutions
Creating Property Profiles	High	User-friendly	Standard web forms	Streamlined profile setup with guided steps
Registration Key Distribution	High	Secure distribution and record-keeping	Separate system for key distribution	Integrate feature within the application.
Employee Role Setup	High	Ease of assigning and adjusting roles	Separate system for assigning roles	Integrate feature within the application.

Operational Employee:

Need	Priority	Concerns	Current Solution	Proposed Solutions
Mobile Responsiveness	High	Access to system features while on the move	Unresponsive design on mobile	Works on IOS and Android
Notification page for requests	High	Immediate update on requests	Separate system for notifications	Integrate feature within the application.

Public User:

Need	Priority	Concerns	Current Solution	Proposed Solutions
Creating user profile	High	User-friendly	Standard web forms	Streamlined profile setup with guided steps
Viewing user profile	High	User-friendly	unresponsive design	Responsive, user-friendly design with easy navigation
Notification page for requests	High	Need for prompt and accurate updates	Separate system for notifications	Integrate feature within the application.

Owner or Renter:

Need	Priority	Concerns	Current Solution	Proposed Solutions
Viewing Property Info	High	User-friendly	unresponsive design	Responsive, user-friendly design with easy navigation
Submitting Requests	High	Timely submission and tracking	Separate request submission system	Incorporate a direct in-app request submission and tracking feature

3.5. Alternatives and Competition

Competitor's Product:

- **Strengths:** Competitor products are often well-established, featuring a comprehensive set of functionalities, backed by professional support and consistent updates. They typically offer scalability that can support growth and adapt to increasing demands.
- **Weaknesses:** These solutions may come with high costs and could include unnecessary features that complicate usage. They might not offer the specific customization needed to perfectly fit unique condo management requirements.

Homegrown Solution:

- **Strengths:** A custom-built solution can be precisely tailored to match the specific needs of condo management, potentially providing a more intuitive user experience and seamless integration with current operational workflows. This approach allows for greater flexibility in feature development and prioritization.
- **Weaknesses:** Developing a solution in-house can be resource-intensive, requiring significant time and financial investment. There's also the risk of encountering technical issues, and such systems may lack the comprehensive features and reliability found in established products.

Maintaining the Status Quo:

- **Strengths:** Opting to maintain existing processes avoids the costs and challenges associated with implementing a new system. It allows management and users to continue with familiar procedures without the need for retraining.
- **Weaknesses:** This approach may lead to operational inefficiencies and a lack of competitive edge in the long run. It fails to address the growing demand for digital convenience, potentially resulting in user dissatisfaction and challenges in managing modern condo properties effectively.

4. Product Overview

4.1. Product Perspective

The Rently system is self contained, except the database systems that are implemented to manage accounts and properties. The frontend system will interact with backend logic, which will connect to these external databases to show values of interest to users, renters/owners and condominium management companies.

The system will have many subsystems, that connect to relevant databases, the subsystems are shown below

- Profile Subsystem
    - Users component
    - Property component
- Financial Subsystem
- Reservation Subsystem
- Notification Subsystem
- Forum Subsystem

Their interconnections are shown in the block diagram below

4.2. Assumptions and Dependencies

This document makes assumptions on the availability of resources and the dependencies that will be used for development and deployment.

Firstly, this vision document assumes that the target system will be deployable and hosted on a 3rd-party system (either an aaS-type system, or hosted on a personal server). We further assume that the hosting service supports the type of application being used (ReactJS frontend,

SpringBoot backend). Inability to find a system that supports these technologies can result in deployment delays and missed deliverables, especially in the final sprint of the project.

It is also assumed that stakeholders may not change their designs significantly, and that the given requirements are understood correctly without ambiguity. Any major changes encountered will be evaluated per the risk management plan and its associated components. Minor changes may be resolved quickly by way of meeting with stakeholders to achieve a middle-ground understanding.

Further, it is assumed that the external systems being used are fault tolerant and can handle exceptional situations. This is simply to help mitigate risks related to loss of data.

5. Product Features

The Rently system is designed to streamline condominium management by providing a comprehensive suite of features that apply to the needs of condo management companies, their employees, condo owners, and renters. By focusing on user-needs design and functionality, Rently aims to address the current gaps in the market, offering an all-in-one solution for efficient property management, financial oversight, and community engagement.

Core Functionalities:

1. User Profile Management
    - **Description**: Enables users to create and manage personal profiles, including uploading pictures and updating contact details.
    - **Why**: Essential for personalizing the user experience and ensuring clear communication.
    - **Priority**: High - foundational for user engagement and security.
    - **Attributes**: Stability (High), Benefit (High), Effort (Medium), Risk (Low).

2. Property and User Association
    - **Description**: Uses registration keys to link owners and renters with their properties, ensuring secure property-user associations.
    - **Why**: Critical for accurate property management and user verification.
    - **Priority**: High - impacts system integrity and user trust.
    - **Attributes**: Stability (High), Benefit (High), Effort (Medium), Risk (Moderate).

3. Dashboard for Property Overview
    - **Description**: Offers detailed dashboards for property information, financial status, and management requests.
    - **Why**: Provides users with a comprehensive view of their property-related transactions and statuses.
    - **Priority**: High - enhances user experience and system usability.
    - **Attributes**: Stability (High), Benefit (High), Effort (High), Risk (Low).

4. Property Profile Creation
    - **Description**: Allows condo management to create detailed property profiles with essential information and documents.
    - **Why**: Centralizes property information for easy access and management.
    - **Priority**: Medium - important for data organization and access.
    - **Attributes**: Stability (Medium), Benefit (High), Effort (Medium), Risk (Low).

5. Financial Management System
    - **Description**: Simplifies managing condo fees, budgets, and financial reporting.
    - **Why**: Ensures financial transparency and operational efficiency.
    - **Priority**: High - vital for financial oversight and planning.
    - **Attributes**: Stability (High), Benefit (High), Effort (High), Risk (Moderate).

6. Reservation System
    - **Description**: Enables booking of common facilities with a view of availability.
    - **Why**: Simplifies the reservation process and facility management.
    - **Priority**: Medium - adds value through convenience and utility.
    - **Attributes**: Stability (Medium), Benefit (Medium), Effort (Medium), Risk (Low).

7. Request Submission and Management
    - **Description**: Streamlines the process for submitting and tracking management requests.
    - **Why**: Facilitates efficient communication and timely response to resident needs.
    - **Priority**: High - crucial for operational efficiency and resident satisfaction.
    - **Attributes**: Stability (High), Benefit (High), Effort (High), Risk (Moderate).

8. Notification System
    - **Description**: Provides real-time updates on request statuses and property announcements.
    - **Why**: Keeps stakeholders informed and engaged with the latest information.
    - **Priority**: High - essential for communication and user engagement.
    - **Attributes**: Stability (High), Benefit (High), Effort (Medium), Risk (Low).

Additional Features:

1. Multilingual Support
    - **Description**: Offers system usability in English and at least one other language.
    - **Why**: Enhances accessibility and inclusivity for a diverse user base.
    - **Priority**: Medium - broadens user accessibility and market reach.
    - **Attributes**: Stability (Medium), Benefit (Medium), Effort (High), Risk (Low).

2. Single Sign-On (SSO)
    - **Description**: Allows login using Gmail or other accounts, streamlining access.
    - **Why**: Simplifies the login process, enhancing user convenience.
    - **Priority**: Medium - improves user experience without compromising security.
    - **Attributes**: Stability (High), Benefit (Medium), Effort (Medium), Risk (Low).

Future Enhancements:

1. Community Engagement Tools
    - **Description**: Introduces forums, event organization tools, and special promotions.
    - **Why**: Fosters a sense of community and engagement among condo residents.
    - **Priority**: Low - offers additional value but not essential to core functionality.
    - **Attributes**: Stability (Medium), Benefit (Low), Effort (High), Risk (Moderate).

6. Other Product Requirements

Standards, Hardware, or Platform Requirements

- **Compatibility:** The application must be compatible with Android, iOS, Linux, macOS, and Windows platforms to ensure accessibility across a broad range of devices.
- **Web and Mobile Application Standards:** Ensure the application follows basic web development best practices, such as responsive design to accommodate various screen sizes and devices. Or functional UI on every browser. For mobile apps, adhere to platform-specific guidelines (Android's Material Design or iOS Human Interface Guidelines).

- **Security Basics:** Apply fundamental security practices, such as secure storage of user credentials and data encryption for sensitive information.
- **Hardware Requirements:** Minimal hardware requirements don't need to be specifically defined with information such as desired processor, ram, or operating system. Our browser app should be accessible on mobile and desktop platforms. Our browser app will be accessible for even lower-end platforms.

Performance Requirements

- **Response Time:** The application should respond to user inputs within 2 seconds under normal conditions.
- **Availability:** The service must be available at all times, excluding scheduled maintenance windows.
- **Scalability:** Must support up to at least 1000 concurrent users without degradation of performance. This target aligns with the anticipated user base and ensures a quality user experience while staying within the constraints of available free-tier infrastructure and services. Eventually, if the need for a higher objective of concurrent users, upgrading tiers will be a possibility.

Environmental Requirements

- **Energy Efficiency:** Mobile versions should be optimized for low power consumption options to preserve battery life. With features such as a UI dark mode.

Quality Attributes

- **Robustness:** The system should handle invalid inputs or unexpected user behavior gracefully without crashing. Automated testing will be implemented such that, potential sudden bugs will be easy to track early on.
- **Fault Tolerance:** Having multiple server clouds or databases would help having a crashing system replaced immediately in the case of a problem. However, implementing such features might prove a little time costly. To be determined.
- **Usability:**
    - **Simplicity:** Use a clean, uncluttered interface with straightforward fonts and colors to highlight essential information, such as condo listings.
    - **Consistent Design:** Maintain uniformity in color schemes, button styles, and typography across the app to facilitate familiarity and ease of use.
    - **Navigation:** Implement a logical flow that allows users to quickly access their dashboard, property listings, and service requests with minimal effort.
    - **Responsive Design:** Ensure the app is adaptable across devices, providing a seamless experience on desktops, tablets, and smartphones.

- **Security:** Must implement strong encryption for data storage and transmission.

Design and External Constraints

- **Design Simplicity:** Prioritize core functionalities like user registration and property listings, with a design that's straightforward and manageable within the project's scope.
- **Resource Limitations:** Utilize free and open-source tools and platforms suitable for student projects, focusing on essential features over advanced functionalities due to budget and time constraints.
- **Third-Party Services:** Choose cost-effective, possibly free-tier, third-party APIs such as Gmail for Single Sign-on and services for features such as notifications, emphasizing ease of integration. These APIs must be maintained for compatibility with no versioning issues.

Documentation Requirements

- **Documentation:** Focus on clear documentation of design choices and code, which is crucial for educational projects and mimics professional software development practices.
- **Online Help:** Interactive help and FAQ sections within the app.

Priority of Requirements

- **Security and Privacy:** Highest priority due to the sensitive nature of user and property information.
- **Usability and Accessibility:** Critical for user ease of use and satisfaction.
- **Performance and Reliability:** Key to demonstrating the app's effectiveness in handling condo management tasks smoothly within the scope of the project.
- **Compliance and Integration:** Aim to understand basic project considerations and ensure the app can successfully connect with external systems with limited budget and time.

Sprint 3 User Stories Backlog

Green: Acceptance tests pass/Minor changes needed but functional feature
Yellow: User stories in sprint 4
Blue: User stories in sprint 5

- REN-78 As a public user, I want to create an account and login
- REN-78 As a system administrator, I want to create a company
- REN-78 As a system administrator, I want to create company admins and link them to a company
- REN-78 As a company administrator, I want to login
- REN-78 As a company administrator, I want to register employees
- REN-78 As an employee, I want to login
- REN-53 As a public user, I want to use an activation key to become an owner/renter
- REN-86 As a company administrator, I want to create a building
- REN-91 As a company administrator, I want to add condos to the buildings
- REN-58 As a company administrator, I want to set up common facilities in my buildings for users to reserve
- REN-59 As an owner/renter, I want to make a reservation on a common facility in a calendar-like interface
- REN-78 As a company administrator, I want to send a registration key to public users to link them to a unit
- REN-54 As an owner/renter, I want to create a profile
- REN-55 As an owner, I want to see a dashboard of my properties to see everything relating to them, like fees and status of requests.
- REN-60 As an owner/renter, I want to see availabilities of common facilities
- REN-134 As a company administrator, I want to add lockers and parking to my building
- REN-61 As a company administrator, I want to upload condo files for my properties
- REN-135 As a company administrator, I want to enter condo fees to units
- REN-63 As a company administrator, I want to set different roles for my employees so they can do the work they are supposed to do
- REN-64 As a company administrator, I want to see the status of all the requests owners made.
- REN-66 As an owner, I want to make special requests to be treated by employees
- REN-67 As an owner, I want to see the status of my requests in a notification page
- REN-68 As an employee, I want to see the owner requests that are assigned to me
- REN-69 As an employee, I want to see the status of my assigned requests on a notifications page
- REN-136 As an employee, I want to update assignments I am working on, like changing the status or adding comments

- REN-62 As a company administrator, I want to enter operational costs for all operations made to know my expenses and budget
- REN-70 As a company administrator, I want to have access to a financial annual report to know what my expenses/budget were
- REN-65 As an owner, I want to access the condo files my management company made available for all units in my building

Software Architecture Document

1. Introduction

1.1 Identifying information

Throughout this report, the project will be referred to as “the project”, “the product” and “the platform”. It all refers to Rently, a condo management platform which will help management companies manage their properties and give access to their users to perform various actions on the platform.

1.2 Overview

The architecture for the system is separated into 3 main components. The frontend, the backend and the database. The database is a PostgreSQL database, the frontend is made with React and the backend in Spring Boot. The backend is to be split into several layers, going from a controller layer, a service layer and several layers up until the database.

2. Stakeholders and concerns

2.1 Stakeholders

The set of stakeholders that are impacted by this project contains owners and renters of units in buildings managed by companies who will use the product to track the operations of their properties. It also includes the management companies, for whom this product is mainly made for. It includes the employees of said companies, who will interact with the system in their day-to-day operations. The stakeholders also include the members of the development team, who meet regularly and discuss with another stakeholder, the project owner – TA in this case – to advance this project.

2.2 Concerns

The project has several concerns.

- The end product should allow management companies to perform various operations on their properties, such as setting up common facilities for their occupants, answering their requests, welcoming new occupants and more.
- The end product should allow a renter or owner to have access to basic information about the unit they live in, give them access to an interface to make requests about their unit or even make reservations for a common facility.
- The deployment of the system is expected to be straightforward, with various parts of the architecture hosted on different providers’ platforms. The frontend is to be hosted on Netlify, the backend on Railways and the database on Neon.

- The system of interest is built at the backend with Spring Boot, allowing for a great separation of concerns and good modularity, thus allowing for good prospects for maintenance and scalability. Spring Boot is a framework that is known and has been proven to work well due to various integrations that are available, especially with authentication and user-defined roles.
- The system is expected to be easily maintainable, due to how the backend is structured.

3. Viewpoint specification

3.1 Overview

This viewpoint focuses on the architectural, structural and behavioral aspects of the system, encompassing user interactions, property management, structure and deployment of the system.

3.2 Model kinds

- Domain model
- Component diagram
- Use case diagram
- Activity diagram
- Class diagram
- Backend Architecture diagram
- Deployment diagram

4. Views

4.1 View: Condo management system architecture

4.1.1 Domain model

Governing model kind: Domain models

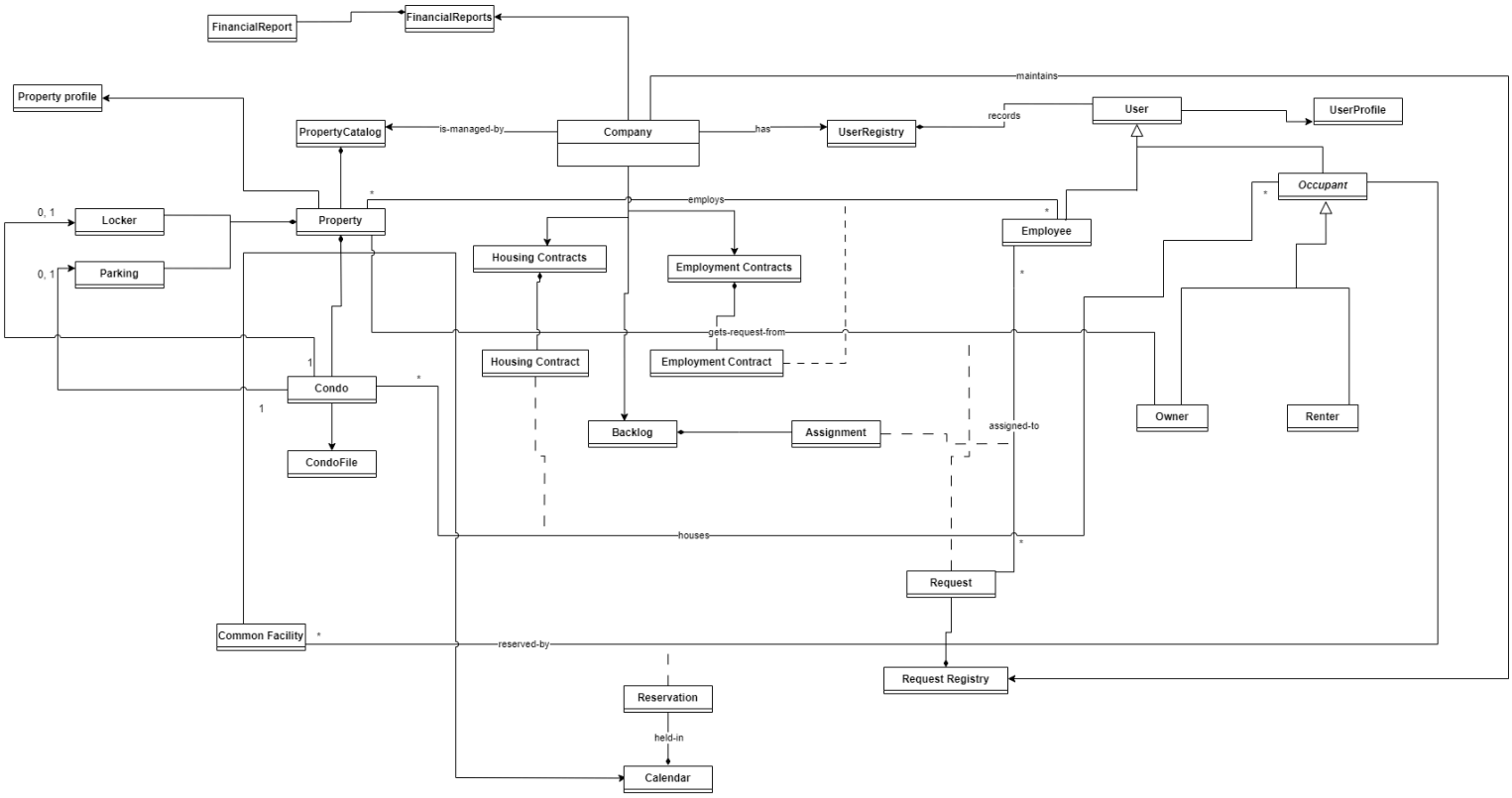


Figure 1: Domain model

This domain model represents the classes that represent real-life elements of the system. The company handles a few registries, making it a controller class and an entry point into the system. Association classes like “Assignment” or “Reservation” are connected to relations between other classes where the relation has a M-N multiplicity. The property class maintains several data structures for various objects, like condos, lockers or parkings. Several types of users require the use of inheritance to simplify development.

4.1.2 Component and architecture diagram

Governing model kind: Component Diagrams

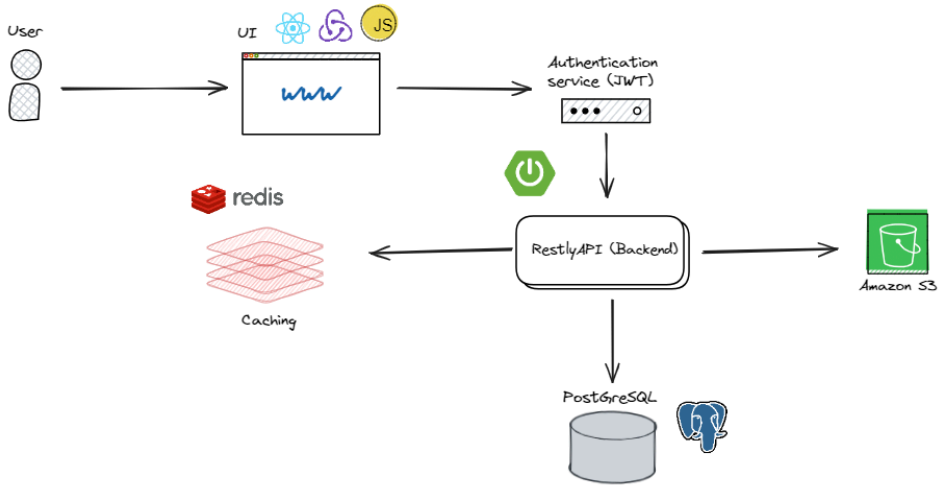


Figure 2: Component and architecture diagram

4.1.3 Use Case diagram

Governing model kind: Use case diagrams

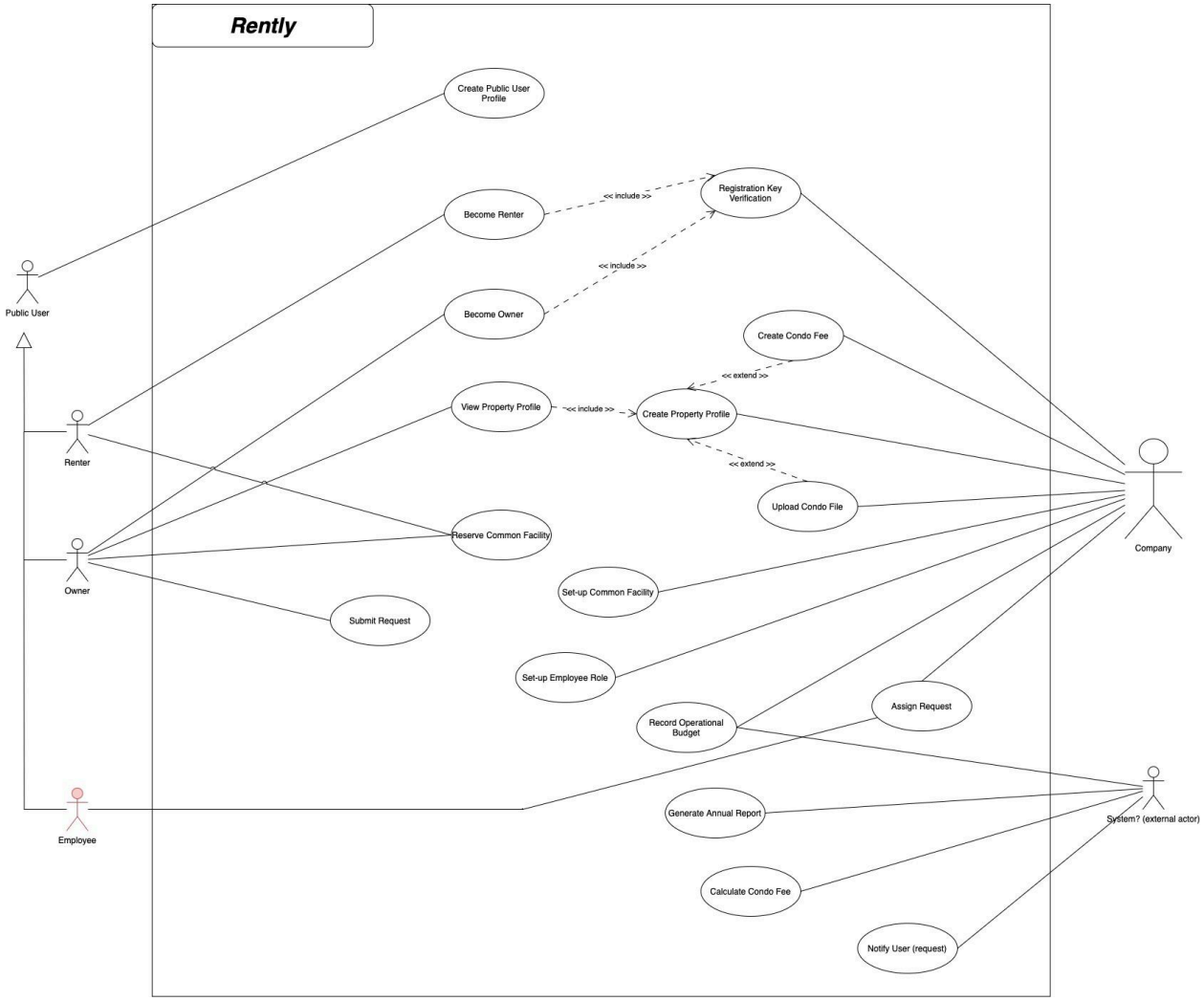
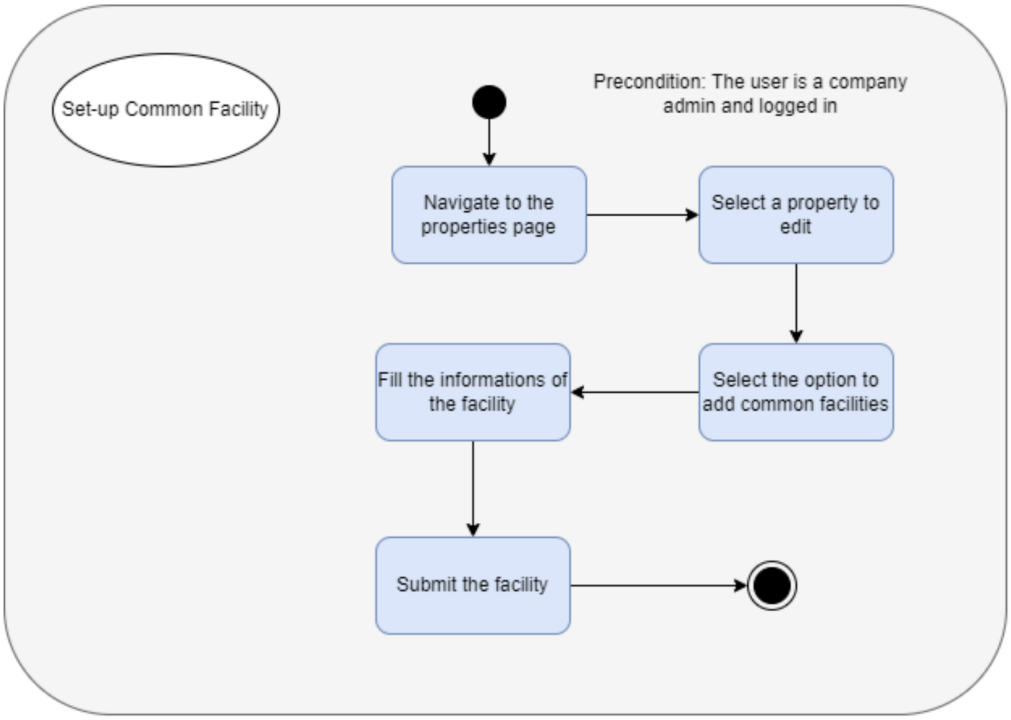


Figure 3: Use case diagram

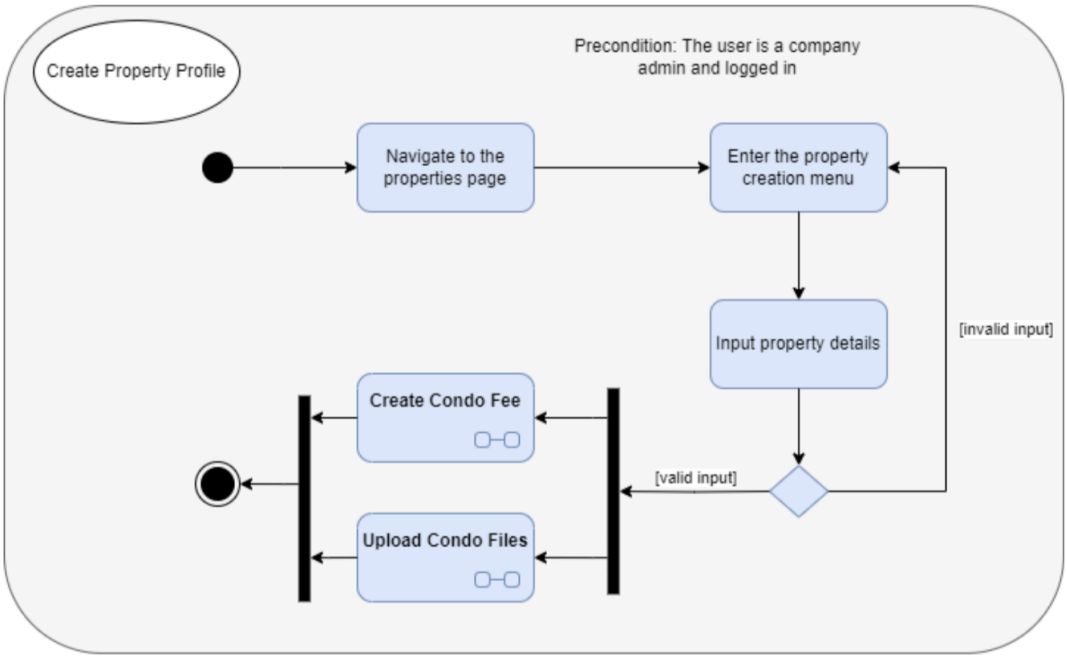
The use case diagram shows all the actors that will interact with the system upon completion of the development. Three external types of actors, the renter, owner and employee fulfill various use cases, where some also involve the action of a system employee.

4.1.4 Activity diagram

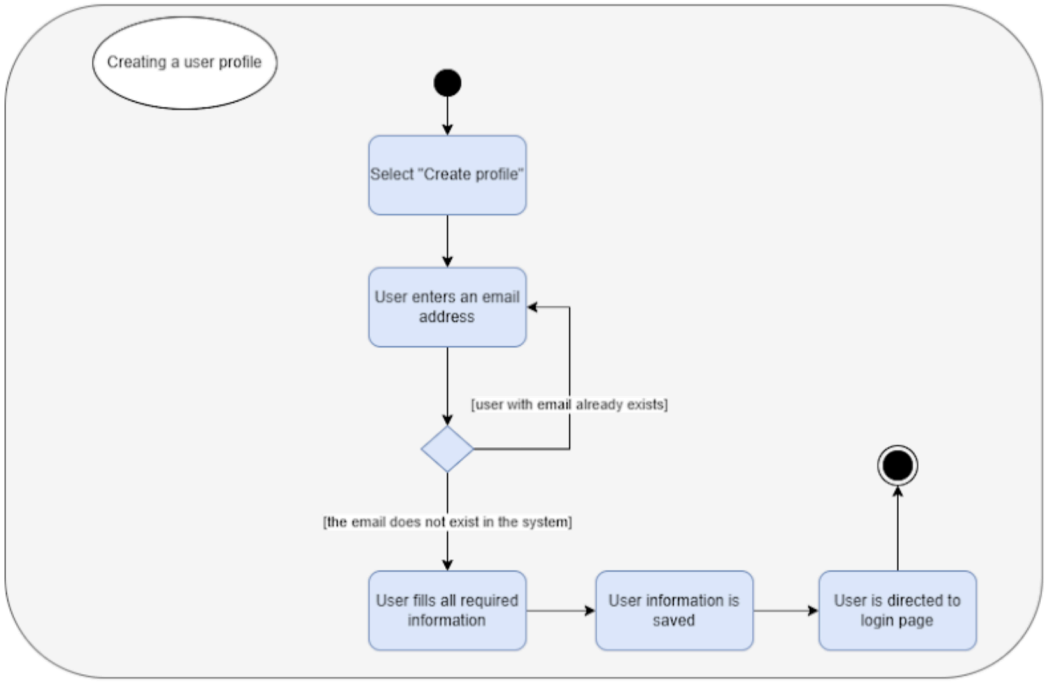
Governing model kind: Activity diagrams



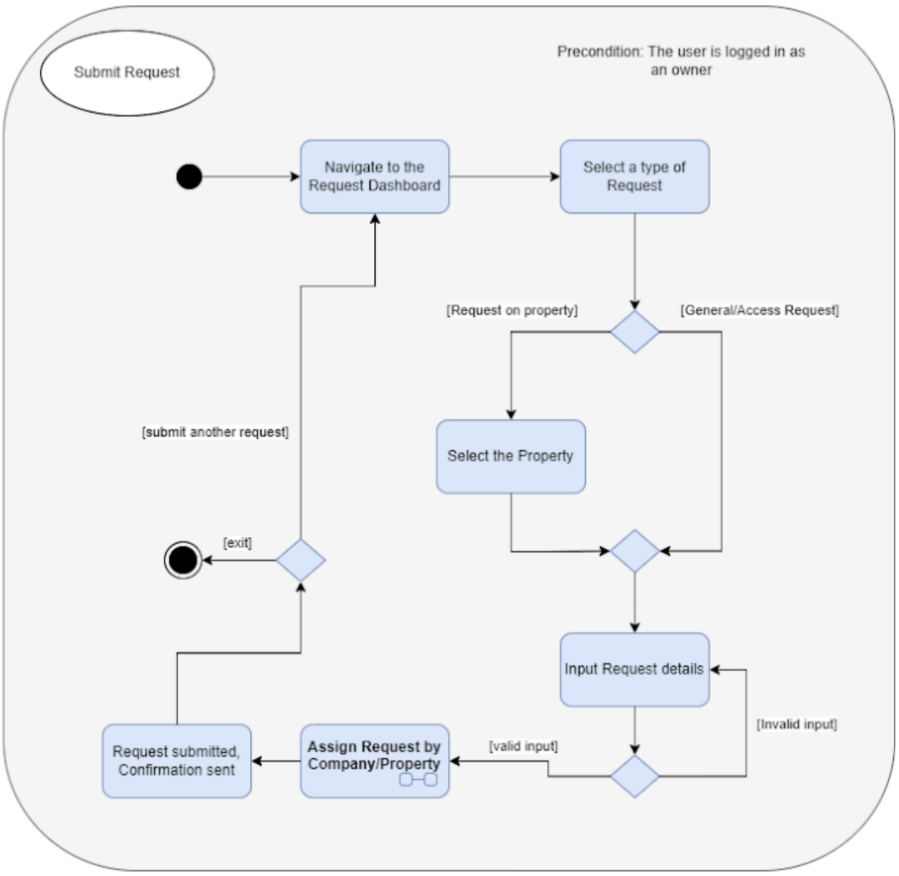
Activity diagram for setting up common facilities



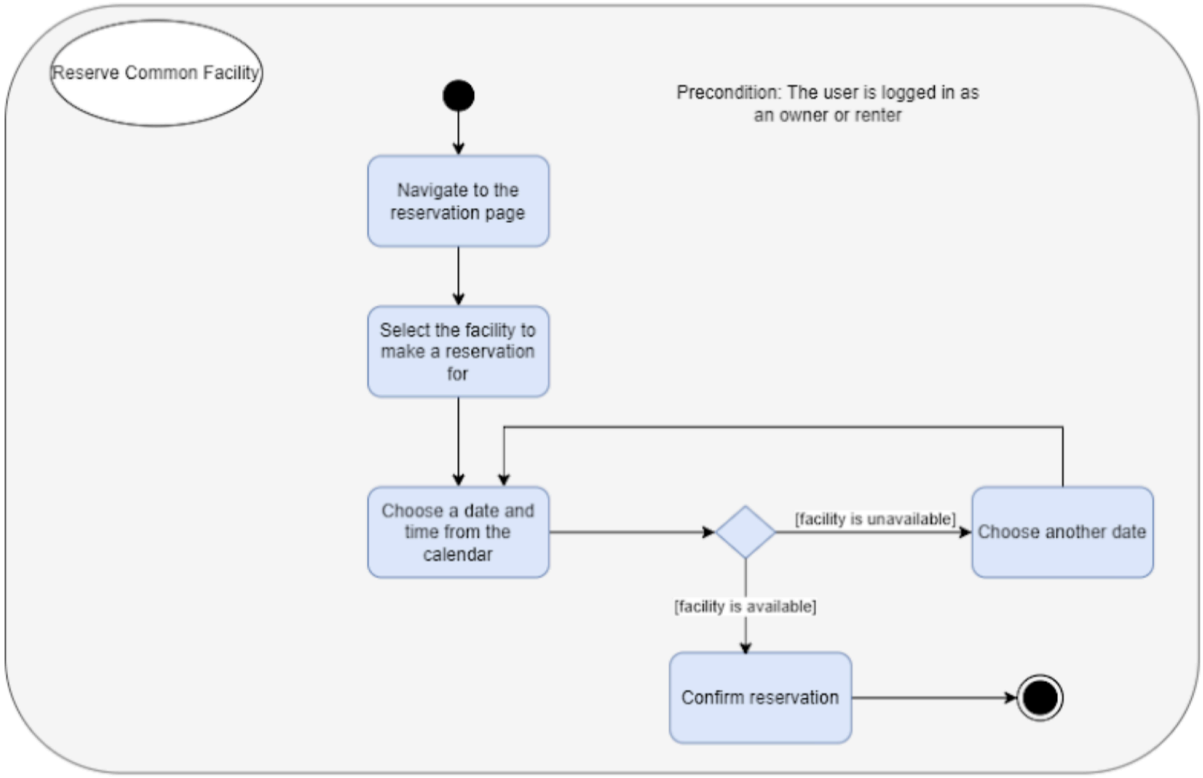
Activity diagram for creating a property profile



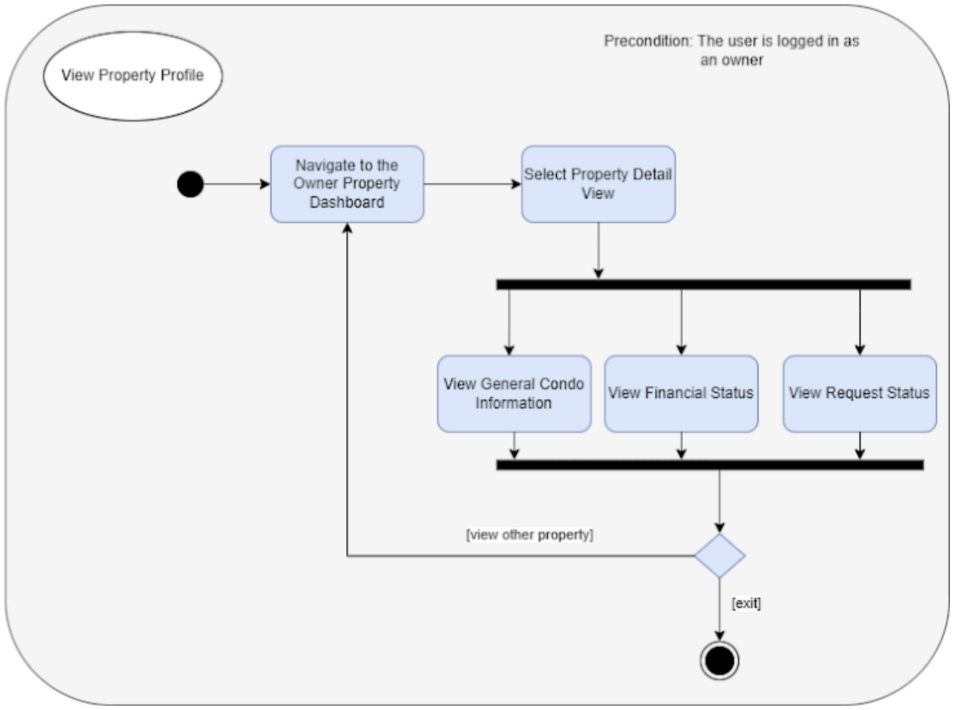
Activity diagram for creating a user profile



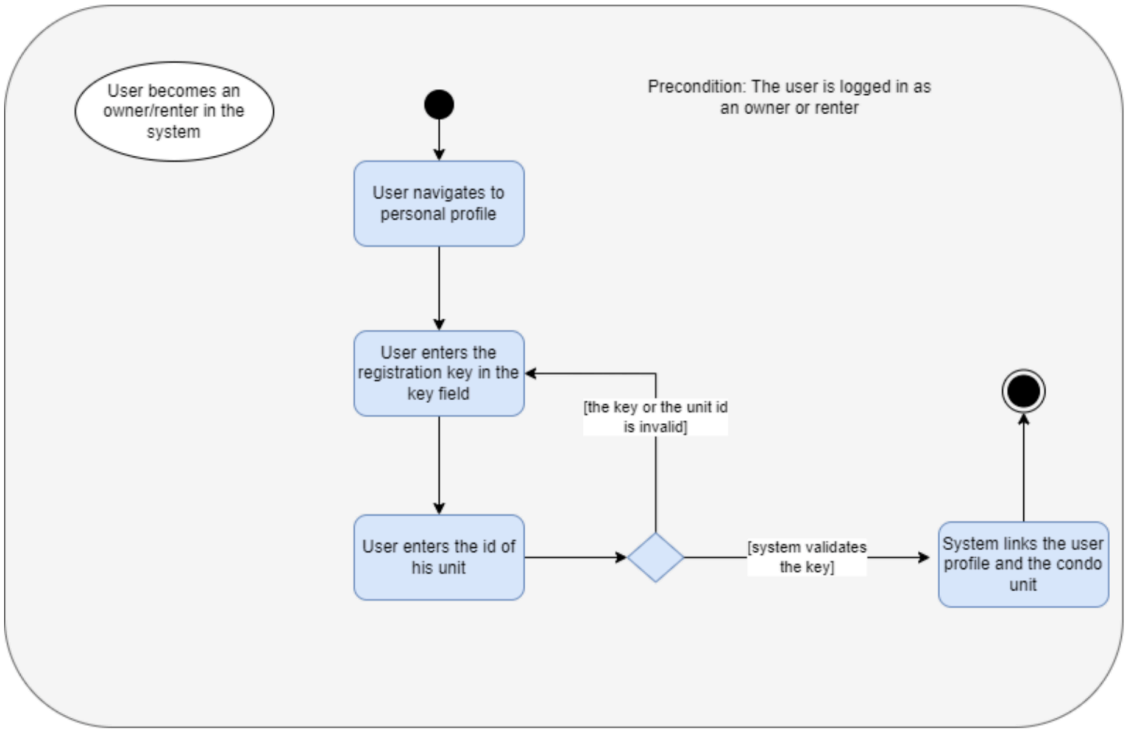
Activity diagram for submitting a request



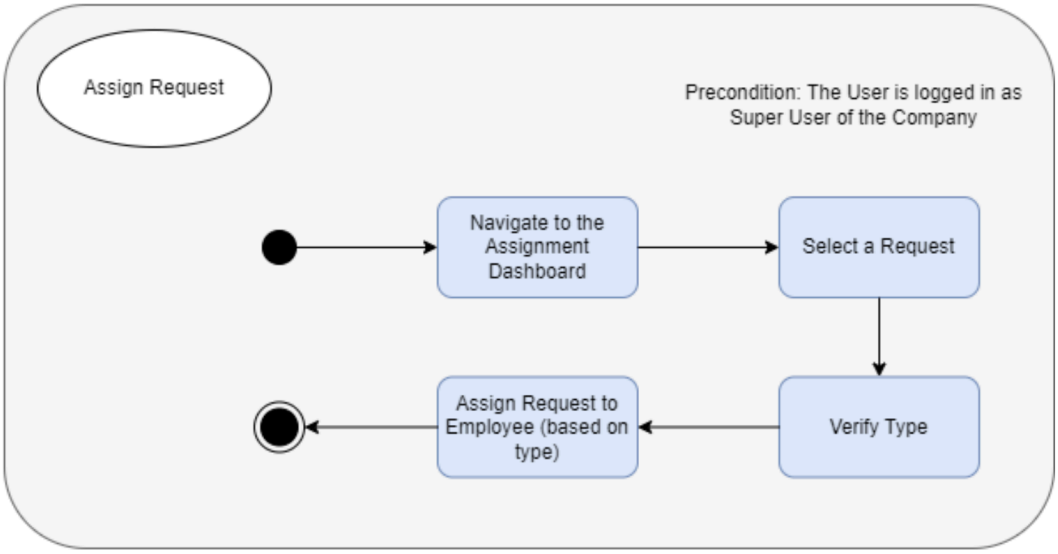
Activity diagram for reserving common facilities



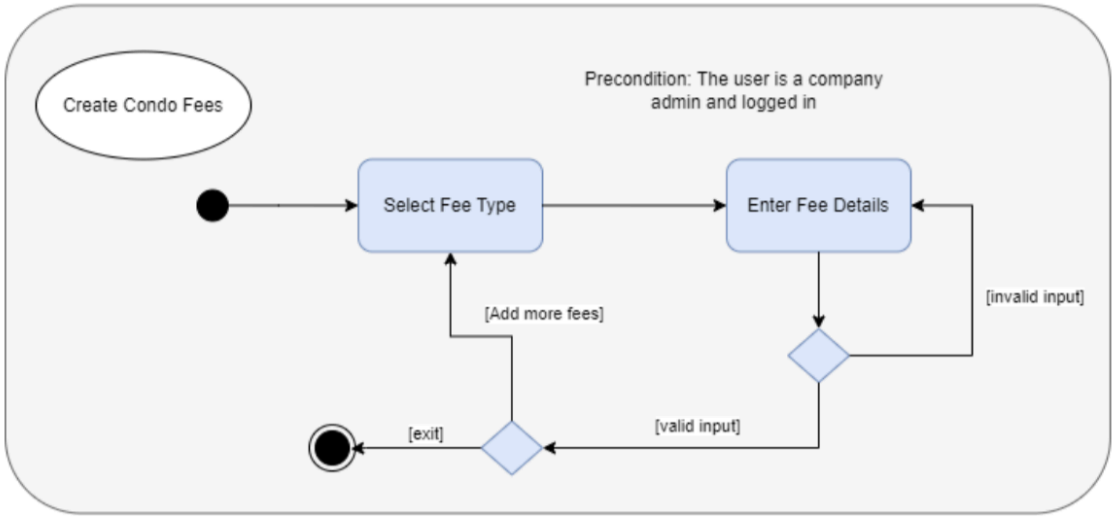
Activity diagram for viewing property profile



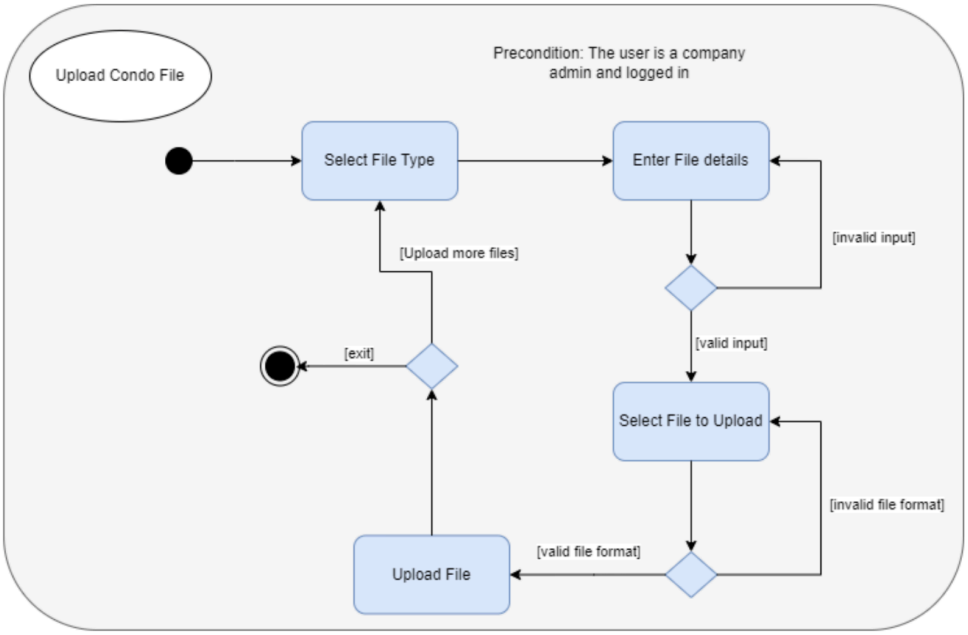
Activity diagram for when a public user becomes an occupant in the system



Activity diagram for assigning a request to an employee



Activity diagram for creating condo fees



Activity diagram for uploading condo files

The above figures represent the activity diagrams for a few use cases in the system. Several of them have preconditions about the existence and authentication status of the actor. They serve the purpose of explaining the flow of events for a given use case.

4.1.5 Class diagram

Governing model kind: Class diagrams

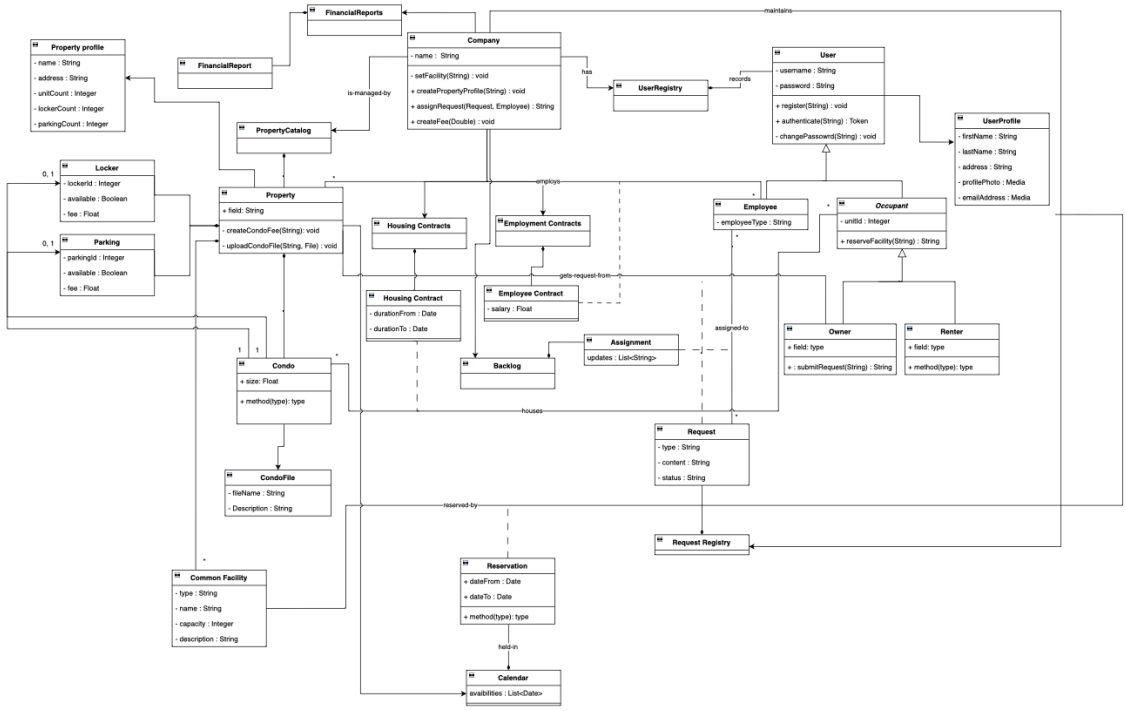


Figure 5: Class diagram

The above diagram is the class diagram, which has the functions that the classes will implement, as well as more attributes of the classes.

4.1.6 Backend architecture diagram

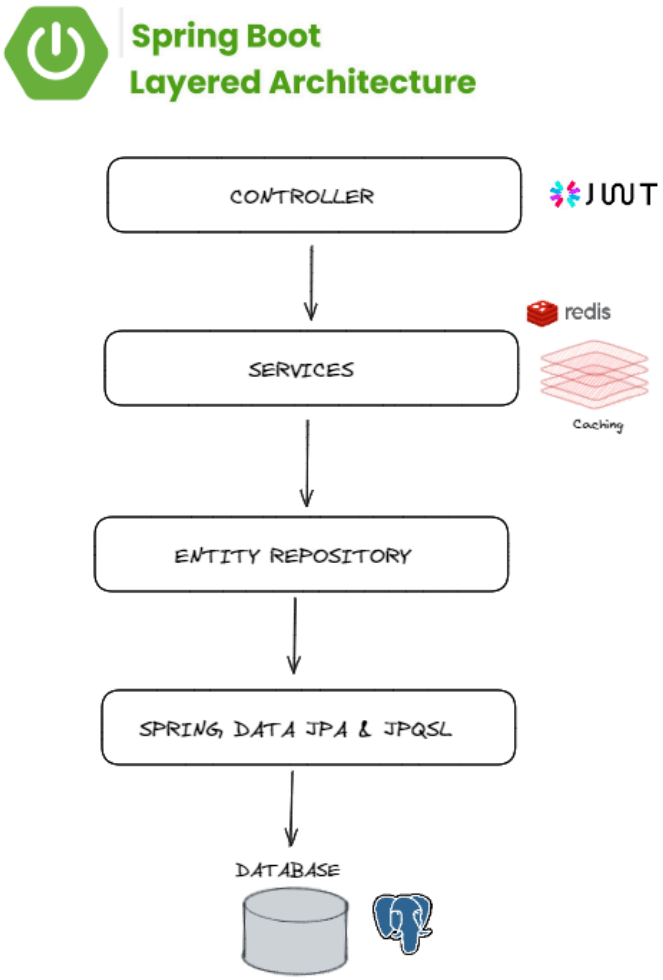


Figure 6: Backend architecture diagram

Figure 6 is a representation of how a Spring Boot backend is structured, going from the controller up to the Postgres database.

4.1.7 Deployment diagram

Governing model kind: Deployment diagrams

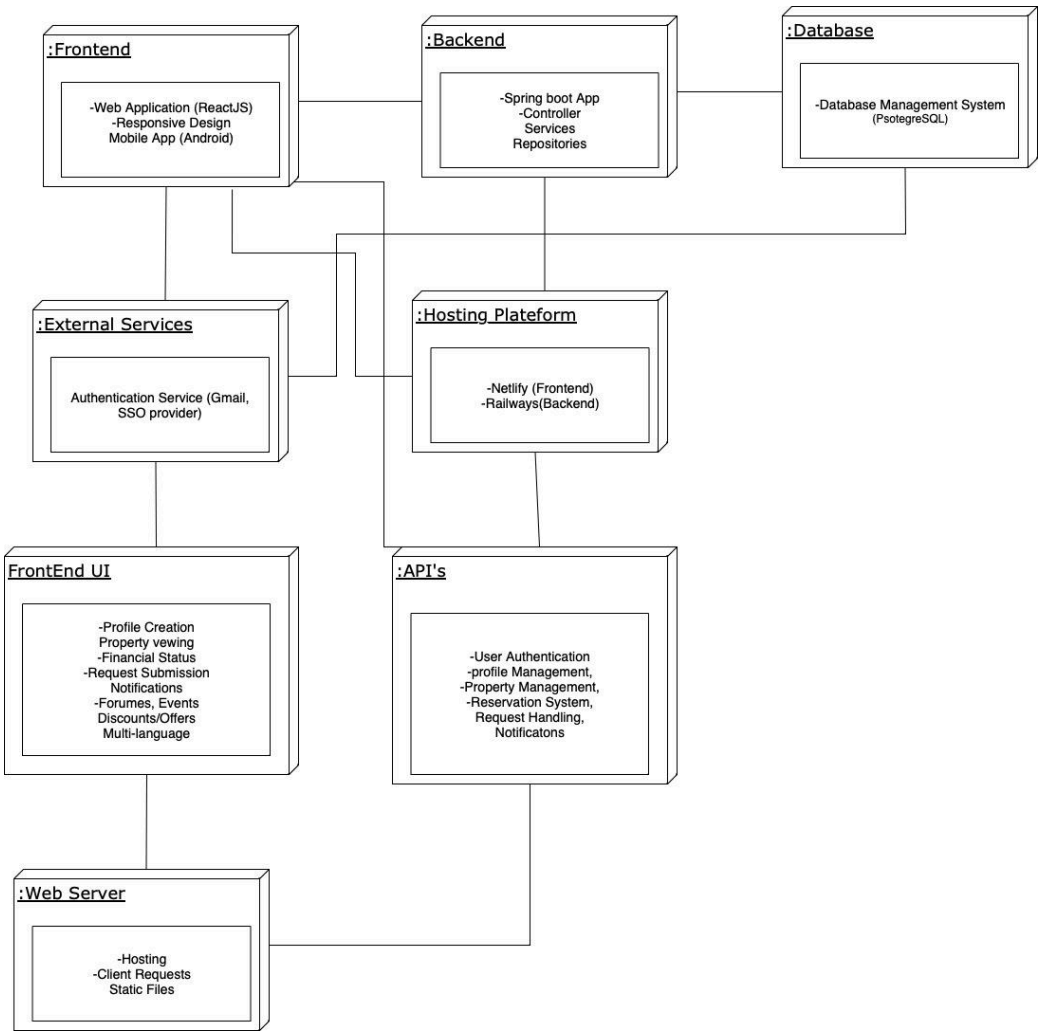


Figure 7: Deployment diagram

Figure 7 is a representation of how various parts of the platform are interconnected once they are hosted. It depicts the relationship between the hosting platform for the backend and frontend is linked to the backend for example, or how the external SSO login service is connected to the database.

Risk Management Plan

This management plan is designed to track, analyze and manage the identified risks associated with development of the Rently condominium management system.

Scope

This RMP document will outline the risks identified by the Rently development team, their likelihood, severity, and chosen response strategy.
Due to lack of time and allocated budget, this document will show the aforementioned information in the form of tables and matrices.
This document will not mention the specifics of how and who will manage the risks identified.

Disclaimer

All risks are assumed to be distributed evenly among the members of the respective subteam (frontend, backend). Program risks are taken on by the team in its entirety. Delegation of risk management is done at the development team's discretion.
Any questions regarding the risks may be asked at the stakeholder meeting directly preceding the addition or adjustment of the risks

1.0 Risk Identification

The first step in analyzing and managing risks associated with a project is to identify them correctly. Similar to requirements management, misidentification of risks can lead to, at best, ambiguity and a partial understanding of the risk and its chosen mitigation technique. Usually, risk management is best done by a subset of the team, typically those in positions of authority. This is because full understanding of the product being developed is essential to identifying the risks with the least amount of misunderstandings. It is also done by managers and leads so developers and technicians need not concern themselves with the project as a whole.
In the case of a typical student-led software development project, it is not uncommon for all students to have an understanding of the entirety of the system being developed. In this case, choosing 2 or more teammates to perform the risk analysis may be sufficient, since the students can discuss and include their differing points of view.
In the case of the Rently system, two students of the 9-person team identified the risks for following sprints.

2.0 Risk Analysis

Risk analysis on the Rently system was done with a clear understanding of the values being chosen.
The “Likelihood” and “Impact” values associated with each risk were done according to the image shown in Figure 1 below.

Risk Matrix & Scales		
LIKELIHOOD SCALE		
Level	Definition	Likelihood
1	It would be surprising if this happened	10%
2	Less likely to happen than not	30%
3	Just as likely to happen as not	50%
4	More likely to happen than not	70%
5	It would be surprising if this did not happen	90%
IMPACT SCALE		
Level	Definition	Cost (% of WBS element value)
1	• Schedule: Insignificant or no schedule slippage. • Functionnality: “Functionality” decrease barely noticeable or no impact. • Programmatic: Only secondary project objectives could be impacted. • Quality: “Quality” degradation barely noticeable	10%
2	• Schedule: 5% slippage. Additional activities required. Able to meet need dates. • Functionnality: Minor areas of “Functionality” are affected. Same approach retained. • Programmatic: Some main project objectives could not be met. Threat can most likely be eliminated with workarounds. • Quality: Only very demanding applications are affected	30%
3	• Schedule: Overall project 5-10% slippage. Some milestone slips. • Functionnality: Moderate areas of “Functionality” are affected but workarounds available. • Programmatic: Main project objectives could not be met. Threat can likely be eliminated with workarounds. • Quality: “Quality” reduction requires client approval.	50%
4	• Schedule: Overall project 10-20% slippage. Possible project critical path impacted. • Functionnality: Major areas of “Functionality” are affected but workarounds available. • Programmatic: Main project objectives most likely will not be met. Workarounds still available. • Quality: “Quality” reduction most likely unacceptable to the client approval.	70%
5	• Schedule: Overall project >20% slippage. Very likely major project milestones can't be met. • Functionnality: “Functionality” reduction unacceptable to client. Project end item is effectively useless. • Programmatic: Main project objectives can't be met without major changes. • Quality: “Quality” reduction is unacceptable. Project end item is effectively unusable.	90%

Figure 1: Values associated with risk likelihood and impact

3.0 Risk Matrix

This section is going to cover the uses and the development of the Risk Assessment Matrix that will be used and updated throughout the project development life cycle.
A risk matrix is typically used to give a quick view of the number of risks, how impactful they are and how likely each risk is. This can give a good overview of the system as a whole as well, for example if all risks were located in high-likelihood, high-impact cells, it could warn project managers and stakeholders about the project as a whole.
Figure 2 shows the risk matrix, along with the identified risks placed in their respective cells.

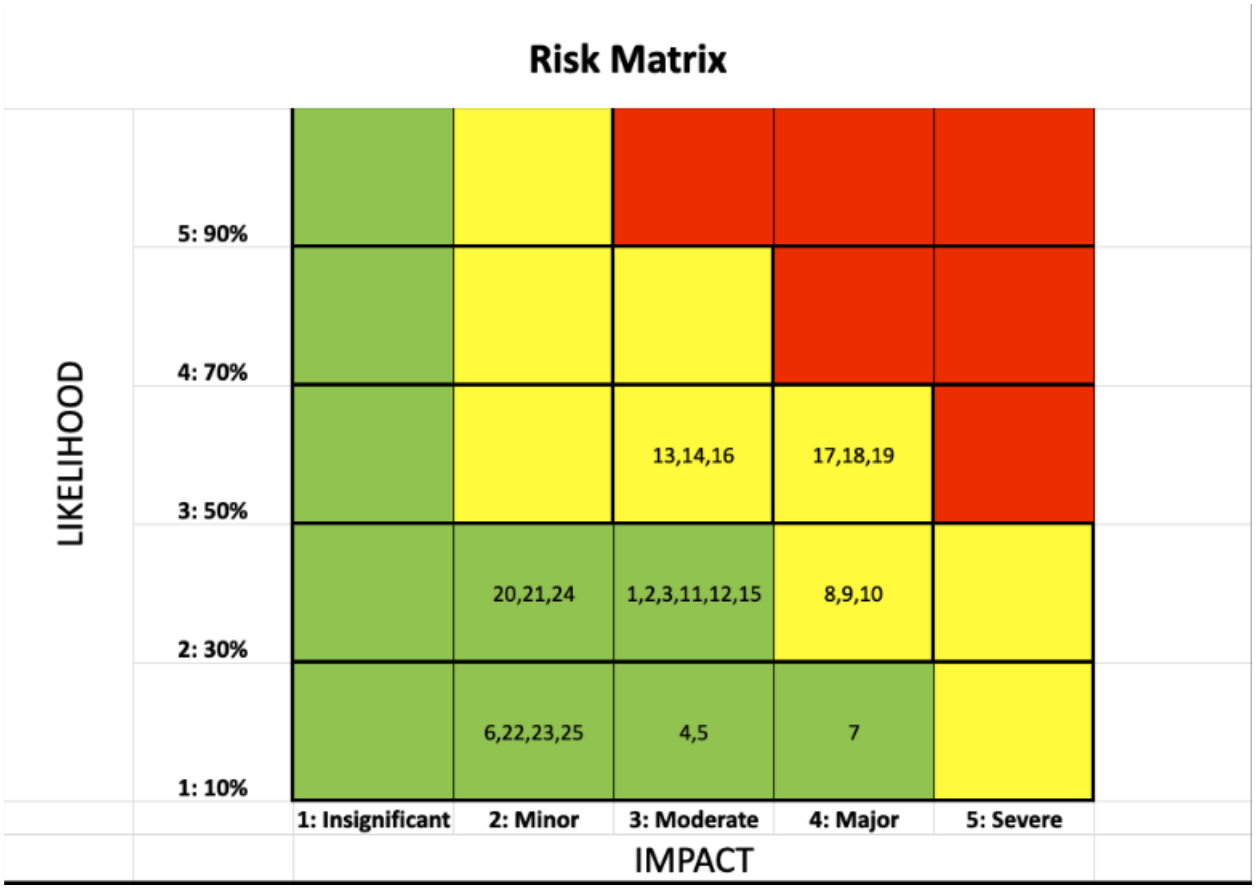


Figure 2: Rently Risk Matrix with risk IDs

All risks are located below “Severe” in impact and all have a likelihood of 50% and below of occurring. This is mainly due to the close communication with the team, and the experience that some students have in web development.
The identified risks each have a unique identifier, this can help clear issues when the team discusses mitigation techniques, preventing ambiguities between risks that may be worded the same.

4.0 Risks & Mitigation Strategies

Typically, there are 4 risk mitigation strategies used involved in project management:

- Accept
- Avoid
- Mitigate
- Transfer

Accept: Acknowledge the risk and choose not to resolve, transfer or mitigate. Typically if the risk is of low impact or can simply not be managed
Avoid: Completely eliminate or forego risk, typically by not engaging in activities that can lead to the occurrence of the risk
Mitigate: Reduce the likelihood or impact of the risk
Transfer: Assign or move the risk to a third-party

These are the mitigation strategies that are going to be used in the risk analysis process.

5.0 Risk Analysis

Here we will cover the identified risks, their likelihood, impact and the associated mitigation strategy.

Low Risk (Green): Low risk scenarios are identified with an impact score ranging from 1(Insignificant) to 4(Major) and a likelihood score from 1(10%) to 5(90%).This leads to a product risk score between 2 to 6 which represents the lower left diagonal of the matrix. For example the product of a likelihood 2 and impact 3 would give 6. These risks are considered to have minimal effects on project goals and are typically unlikely to occur. They are generally addressed with standard procedures and do not necessitate significant resource allocation. The management strategy for low-risk items involves regular monitoring to implement basic risk mitigation tactics should these risks become a reality.

Medium Risk (Yellow): Medium risk scenarios are identified with an impact score ranging from 2(Minor) to 5 and a likelihood score from 1 to 5. This leads to a product risk score between 8 to 12 which represents the middle diagonal of the matrix. These risks have the potential to noticeably impact the project and have a moderate probability of happening. Addressing these risks requires increased attention and may call for specific mitigation strategies to manage their impact effectively. The typical response involves proactive risk management approaches, such as contingency planning or implementing risk reduction techniques.

High Risk (Red): High risk scenarios are identified with an impact score ranging from 3(Moderate) to 5 and a likelihood score from 3(50%) to 5. This leads to a product risk score between 15 to 25 which represents the upper right diagonal of the matrix. Such risks present a significant threat to the project, with a high chance of occurrence and the capability of greatly affecting the project. These risks demand immediate and prioritized action. Management strategies could include crafting detailed response plans, reallocating resources, or applying risk avoidance tactics to safeguard the project’s success.

Table 1: Sprint 1 Risk Analysis

Risk ID	Description	Impact	Probability	Severity	Entry Date (Sprint)	Response Strategy	Response Plan
1	**Management** Lack of team communication	3	2	Low	2/3/2024 (1)	Mitigate	Communication by slack and discord. Multiple meetings per week if needed
2	**Technical** Features not implemented	3	2	Low	2/3/2024 (2,3,4)	Mitigate	Most important features are implemented first
3	**Management** Team members not completing tasks	3	2	Low	2/3/2024 (1,2,3,4,5)	Mitigate	Team members will ask for help if a task is too time consuming
4	**Technical** Team members don't know the technologies being used	3	1	Low	2/3/2024 (1,2,3)	Avoid	Team members have discussed the technologies they know before beginning sprint 1
5	**Management** Development becomes out-of-scope of stakeholder needs	3	1	Low	2/3/2024 (1,2,3,4,5)	Avoid	Project scope is set and is to be followed to avoid development runoff
6	**Technical** UI not consistent site-wide	2	1	Low	2/5/2024 (2,3,4,5)	Avoid	Regular code reviews, developers have access to entire codebase.

							Sanity checks. Shared React components
7	**Management** Not understanding stakeholder requirements	4	1	Low	2/6/2024 (2,3,4,5)	Avoid	Multiple sprints and meetings with stakeholders throughout project development
8	**Technical** UI not connected to backend logic	4	2	Medium	2/6/2024 (2,3,4)	Mitigate	Performing tests on integrated system
9	**Technical** Backend logic not connected to database	4	2	Medium	2/6/2024 (2,3,4)	Mitigate	Performing tests on integrated system
10	**Technical External** Site hosting issues	4	2	Medium	2/6/2024 (4)	Mitigate	**Transfer** as well Performing tests on hosted system to ensure site reliability
11	**Technical External** Integration with 3rd party systems	3	2	Low	2/6/2024 (2,3,4)	Mitigate	**Transfer** as well Site hosting, 3rd party APIs, database management. Performing tests and checks throughout development

12	**Technical External** Setting up S3 Store	3	2	Low	2/27/2024 (2,3)	Mitigate	Research best practices, follow AWS documentation
13	**Technical External** Implementing Google Auth and SSO	3	3	Medium	2/27/2024 (2,3)	Mitigate	Use official Google Auth documentation and perform sufficient testing
14	**Technical** Increasing Backend Complexity	3	3	Medium	2/27/2024 (2,3,4,5)	Mitigate	Modularize code,document thoroughly
15	**Technical** Increasing Frontend Complexity	3	2	Low	2/27/2024 (2,3,4,5)	Mitigate	Modularize code,document thoroughly
16	**Technical** CORS issues	3	3	Medium	2/27/2024 (2,3)	Mitigate	specifying permitted domains, restricting credential use,frequently updating security measures, and educating the team on CORS protocol to avoid unauthorized access.

17	**Technical** JWT setup	4	3	Medium	2/27/2024 (2,3)	Mitigate	Secure JWTs with a robust key, use HTTPS for safe tranmission, and rigorously test token management to confirm correct authentication and token expiry handling
18	**Technical** Refresh token setup	4	3	Medium	2/27/2024 (2,3)	Mitigate	Store refresh tokens in HTTP-only cookies marked as secure for transmission over HTTPs, and regularly update these tokens for enhanced security
19	**Technical** HTTP Only Cookie setup	4	3	Medium	2/27/2024 (2,3)	Mitigate	Use secure attributes for cookies, implement same-site policies, and ensure proper CSRF protection
20	**Technical** Inadequate Test Coverage of frontend	2	2	Low	3/19/2024 (2,5)	Avoid	Team members have experience with Jest

21	**Technical** Inadequate Test Coverage of backend	2	2	Low	3/19/2024 (2,5)	Avoid	Team members have experience with JUnit and Mockito
22	**Technical** Inconsistent Coding Standards frontend	2	1	Low	3/19/2024 (1,5)	Avoid	Team members have decided on project structure, component design, state management, and hook usage
23	**Technical** Inconsistent Coding Standards backend	2	1	Low	3/19/2024 (1,5)	Avoid	Team members have decided on project structure, RESTful endpoint design, database interaction, and use of annotations
24	**Technical** Inefficiency in Handling SonarQube identified non critical issues	2	2	Low	3/19/2024 (3,5)	Mitigate	Team members will prioritize critical SonarQube findings and adjust the tool's settings to align with project priorities ensuring a focus on high-impact issues

25	**Management** Documentation out of date	2	1	Low	3/19/2024 (1,5)	Mitigate	Team members are assigned the same sections for each sprint and are responsible for updates

Testing plan

This section will outline the testing philosophy and the approach taken for proper testing of the system. Here, tools and methodologies will be presented to give an idea of a testing framework that will be used.

Methodology

In general, unit testing is conducted on all methods that allow for appropriate testing within a respectable time frame. This will be done to ensure that the system is built in an appropriate manner, and that all methods give an expected output when given an expected input. Due to the size of the codebase, rigorous modeling (such as Input Domain Modeling with an Interface- or Functionality- based approach) will be avoided because the majority of the team has done little testing, and these concepts are just now being introduced to many team members as the system is being built. Further, different testing planning methods (such as use of formal control follow graphs for test planning) will be avoided due to its verbose documentation and lack of time of implementation.
In place, the methodology being used is planning to cover as many lines of code with the unit tests being built. This ensures coverage of the most amount of code, to find issues if correct input is given.

Tools used

JUnit

The unit tests for the backend are being developed using JUnit, a unit testing framework developed for codebases built in Java. It provides large functionality and an easy-to-use syntax to allow all members of the team to contribute in testing the system.
Further, it is often used in the industry in Test-Driven Development (TDD), this will give students exposure to software and tools that they may encounter in the workforce following graduation from university.

Jest

The tests being developed for the frontend system are being done through Jest, a software testing framework designed for JavaScript systems. It has been used by members of the team in previous instances, so it is a good starting point for JavaScript testing. Keeping to frameworks that members are familiar with is key to ensuring adequate development of the system in a short period of time.

SonarQube

SonorQube will be integrated into the GitHub repository to provide continuous integration and static code analysis of the system as changes are made. This will provide active feedback on development efforts and ensure that each pull request made does not cause build issues in the system. This can also provide acceptance test standards.
Implementation of SonarQube is to be pushed to Sprint 4, this is because SonarQube scans compiled classes, these are not pushed to the GitHub repository. A CI/CD pipeline needs to be built to correctly implement SonarQube, which is going to be done in Sprint 4.

Metrics/Success Criteria

The system will be defined as adequately tested when the coverage using IntelliJ/JUnit's testing coverage software passes with a coverage above 80%. Doing this allows for the team to forgo that last few percent of coverage, something that can be difficult to achieve in general, especially for a codebase with such a short-lived life-cycle.

Current overall coverage summary : 80%+ code coverage

Test Results and Coverage:

Backend - JUnit & Mockito

Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	80.2% (65/81)	84.4% (331/394)	82.05% (750/914)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
com.rently.rentlyAPI	0% (0/1)	0% (0/2)	0% (0/2)
com.rently.rentlyAPI.config	0% (0/1)	0% (0/1)	0% (0/1)
com.rently.rentlyAPI.controller	75% (3/4)	92% (23/25)	87.5% (28/32)
com.rently.rentlyAPI.controller.auth	0% (0/1)	0% (0/4)	100% (4/4)
com.rently.rentlyAPI.dto	86.6% (13/15)	85.8% (73/85)	87.34% (138/158)
com.rently.rentlyAPI.dto.auth	12.5% (1/8)	11.1% (4/36)	11.1% (4/36)
com.rently.rentlyAPI.entity	94.11% (16/17)	85.5% (68/80)	84.42% (75/89)
com.rently.rentlyAPI.entity.auth	0% (0/7)	89.7% (35/39)	0% (0/44)
com.rently.rentlyAPI.entity.enums	0% (0/1)	0% (0/2)	0% (0/4)
com.rently.rentlyAPI.exceptions	0% (0/4)	0% (0/9)	0% (0/11)
com.rently.rentlyAPI.handlers	0% (0/3)	100% (16/16)	100% (47/47)
com.rently.rentlyAPI.security	0% (0/2)	0% (0/9)	100% (37/37)
com.rently.rentlyAPI.security.config	0% (0/2)	0% (0/13)	0% (0/47)
com.rently.rentlyAPI.security.config.audit	0% (0/1)	0% (0/2)	0% (0/8)
com.rently.rentlyAPI.security.filter	0% (0/1)	0% (0/2)	0% (0/20)
com.rently.rentlyAPI.security.utils	0% (0/2)	0% (0/16)	0% (0/70)
com.rently.rentlyAPI.services.auth	0% (0/4)	86.6% (13/15)	86.6% (78/90)
com.rently.rentlyAPI.services.impl	16.7% (1/6)	88.8% (32/36)	90.2% (185/205)
com.rently.rentlyAPI.validators	0% (0/1)	0% (0/2)	0% (0/9)

```
[INFO] Running com.rently.rentlyAPI.exceptions.AuthenticationExceptionTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.013 s - in com.rently.rentlyAPI.exceptions.AuthenticationExceptionTest
[INFO] Running com.rently.rentlyAPI.exceptions.FileUploadExceptionTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.236 s - in com.rently.rentlyAPI.exceptions.FileUploadExceptionTest
[INFO] Running com.rently.rentlyAPI.exceptions.ObjectValidationExceptionTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.081 s - in com.rently.rentlyAPI.exceptions.ObjectValidationExceptionTest
[INFO] Running com.rently.rentlyAPI.exceptions.OperationNonPermittedExceptionTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 s - in com.rently.rentlyAPI.exceptions.OperationNonPermittedExceptionTes
[INFO] Running com.rently.rentlyAPI.handlers.ExceptionRepresentationTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.008 s - in com.rently.rentlyAPI.handlers.ExceptionRepresentationTest
[INFO] Running com.rently.rentlyAPI.handlers.GlobalExceptionHandlerTest
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.169 s - in com.rently.rentlyAPI.handlers.GlobalExceptionHandlerTest
[INFO] Running com.rently.rentlyAPI.security.config.audit.ApplicationAuditAwareTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.064 s - in com.rently.rentlyAPI.security.config.audit.ApplicationAuditAwareT
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 102, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

Frontend - Jest

```
Test Suites: 14 passed, 14 total
Tests:       51 passed, 51 total
Snapshots:   0 total
Time:        24.29 s, estimated 25 s
Ran all test suites.
```

```
import { render, screen } from "@testing-library/react";
import CreatePropertyForm from "../components/CreatePropertyForm";
import userEvent from "@testing-library/user-event";

describe("CreatePropertyForm Component", () => {
    const mockOnFormSubmit = jest.fn();

    beforeEach(() => {
        mockOnFormSubmit.mockClear();

        render(<CreatePropertyForm onFormSubmit={mockOnFormSubmit} />);
    });

    it("renders input fields for property attributes", () => {
        expect(screen.getByText("Property name")).toBeInTheDocument();
        expect(screen.getByText("Unit count")).toBeInTheDocument();
        expect(screen.getByText("Parking count")).toBeInTheDocument();
        expect(screen.getByText("Locker count")).toBeInTheDocument();
        expect(screen.getByText("Street address")).toBeInTheDocument();
        expect(screen.getByText("City")).toBeInTheDocument();
        expect(screen.getByText("Province")).toBeInTheDocument();
        expect(screen.getByText("Postal code")).toBeInTheDocument();
    });

    it("renders the 'Create Property' button", () => {
        expect(
            screen.getByRole("button", { name: "Create Property" })
        ).toBeInTheDocument();
    });

    it("calls onFormSubmit when form is submitted", async () => {
        const submitButton = screen.getByRole("button", {
            name: "Create Property",
        });
        await userEvent.click(submitButton);

        expect(mockOnFormSubmit).toHaveBeenCalled();
    });
});
```

```
import { render, screen } from "@testing-library/react";
import Register from "../pages/Register";
import { BrowserRouter } from "react-router-dom";

describe("Register Component", () => {
    beforeEach(() => {
        render(
            <BrowserRouter>
                <Register />
            </BrowserRouter>
        );
    });

    it("renders the 'First name' label", () => {
        expect(screen.getByText("First name")).toBeInTheDocument();
    });

    it("renders the 'Last Name' label", () => {
        expect(screen.getByText("Last Name")).toBeInTheDocument();
    });

    it("renders the 'Email' label", () => {
        expect(screen.getByText("Email")).toBeInTheDocument();
    });

    it("renders the 'Phone Number' label", () => {
        expect(screen.getByText("Phone Number")).toBeInTheDocument();
    });

    it("renders the 'Password' label", () => {
        expect(screen.getByText("Password")).toBeInTheDocument();
    });

    it("renders the 'Confirm Password' label", () => {
        expect(screen.getByText("Confirm Password")).toBeInTheDocument();
    });
});
```

Moving Forward/Sprint 4

For sprint 4, the team will continue to work with JUnit and Jest for unit testing on the system. As mentioned, SonarQube will also be implemented into the pipeline to provide static analysis on the system prior to deployment. This is an important task to complete to ensure proper deployment of the system, without this there may be possible build issues that are not encountered until demo time to the stakeholders. These processes and tools will help immensely with all the moving parts making the project a reality.

Rently Sprint 3 Retrospective

Introduction

This postmortem is written in a context where the third sprint is completed, with features already functional. As mentioned in the retrospective for sprint 2, this sprint involved a great amount of refactoring. The entire backend system was redone, going from the authentication services to the persistence layer. The code that was written in the first two sprints did not fully adhere to the design we had made and also did not meet code quality standards, especially in terms of cohesion and coupling.

What went wrong

1 - Immense refactoring challenge
We had to refactor the entire backend code that was written in the first two sprints, which was a very tedious task. There were a lot of problems in terms of code quality and respect of coding conventions that we needed to address. Some classes were doing many different things, and we needed to break them down to reduce the levels of coupling and make them more cohesive.

2 - Problems using SonarCloud
We set up SonarCloud to find code issues and help us improve the quality of our code. We thought the set up was going to be easy and not tedious. We had already used SonarCloud in previous courses and the setup was easy, so our expectations were not met. When we ran the analysis the first time, we realized it did not work on .java classes, and it only takes compiled classes. The compiled classes are not present in the repository, and we did not want to push them. What this means is we need to build the project “online” before running the scan. This forces us to use a CI/CD pipeline, like CircleCI.

3 - Refactoring broke our tests
Something that went very bad was linked to the tests. In doing our refactoring, we broke all the tests that were for the backend. Our top priority was to get the system working to catch up on the activities of the 3rd sprint. Not having these tests makes us vulnerable to problems we might have missed.

4 - One branch held most changes
We initially started working on the common facility feature before we decided to refactor. We made the mistake of making the changes on the same branch as the feature and were not in a position to merge it to the main branch because it would have broken everything, with a long time before we fix the problems. We were forced to always make changes on different branches that diverge from the feature branch instead of the main branch. This meant that for a good

while, the branch that was initially for the feature effectively became our main branch. It was bad because it forced us to finish all our changes before merging to main and starting other features.

What went right

1 - Application of design patterns

We used the facade design pattern, which made our code clearer and made it so that the right services were being called from a few endpoints. For example, when registering an employee or a company admin, the controller calls the user service, which then retrieves the type of user and calls the appropriate constructor. So the data would go to the user service who then calls the employeeService or the adminService. This was done for several functionalities. It allowed us to easily structure the code and makes development easier.

2 - Backend development speed

Since we had the design patterns, it was easy to implement the features once a few of them were done. It was like a template for us to follow and helped us avoid spending too much time thinking about where to place certain functions. Also, following the GRASP patterns gave us a lot of cases of reusability, which in turn saves us a lot of time.

3 - Better use of Postman CI/CD

We also improved our use of Postman. With the growing number of endpoints, it was becoming difficult to perform a specific action without performing a series of other actions beforehand. For example, to create a common facility, a system admin must first be created, then create a company and add to it a system admin, then authenticate the admin, make it create a building and then from there create a common facility. Needless to say it is difficult to test a growing system if all these requests depend on a lot of them. We wrote scripts to help us perform these actions so we do not have to do them.

4 - Smaller number of merge conflicts

Due to the great communication between team members, a more coherent code and a better organization, the merge conflicts were lesser and of small complexity to solve than in the first sprints. This was true even with 3-4 members working on the same branch. This was a great advantage to us because we did not spend time solving the conflicts. It was also a sign for us that our new approach is working.

Conclusion

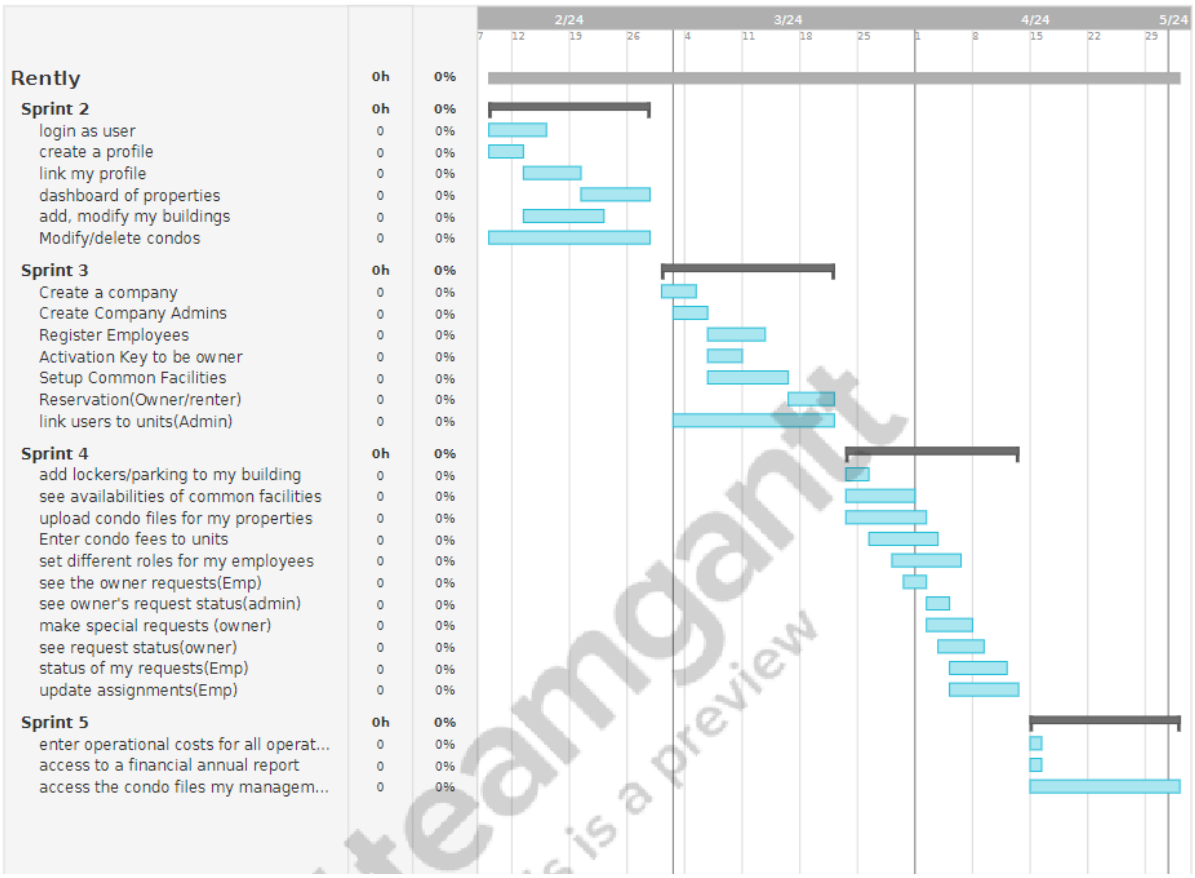
In conclusion, Sprint 3 had its ups and downs. Despite the tough task of refactoring and some hiccups with SonarCloud and broken tests, we made it through. Using design patterns like the facade pattern really helped clean up our code and speed up development, although we were stuck working from a feature branch that was acting like the main branch in the backend. Postman was a lifesaver for testing our endpoints, especially with complex sequences. Plus, our improved teamwork led to fewer merge conflicts, making everything smoother.

Release Plan - Sprint 4 User Stories

Table of stories in sprint 4

User Story	JIRA ID	(USP)	Priority	Status
As a company administrator, I want to add lockers and parking to my building	REN-134	4	Medium	TODO
As a company administrator, I want to upload condo files for my properties	REN-61	7	High	TODO
As a company administrator, I want to enter condo fees to units	REN-135	5	High	TODO
As a company administrator, I want to set different roles for my employees so they can do the work they are supposed to do	REN-63	5	High	TODO
As a company administrator, I want to see the status of all the requests owners made.	REN-64	4	Medium	TODO
As an owner, I want to make special requests to be treated by employees	REN-66	4	Medium	TODO
As an owner, I want to see the status of my requests in a notification page	REN-67	5	High	TODO
As an employee, I want to see the owner requests that are assigned to me	REN-68	6	High	TODO
As an employee, I want to see the status of my assigned requests on a notifications page	REN-69	4	Medium	TODO
Total USP		47		

Progression chart



UI Prototypes for Sprint 4

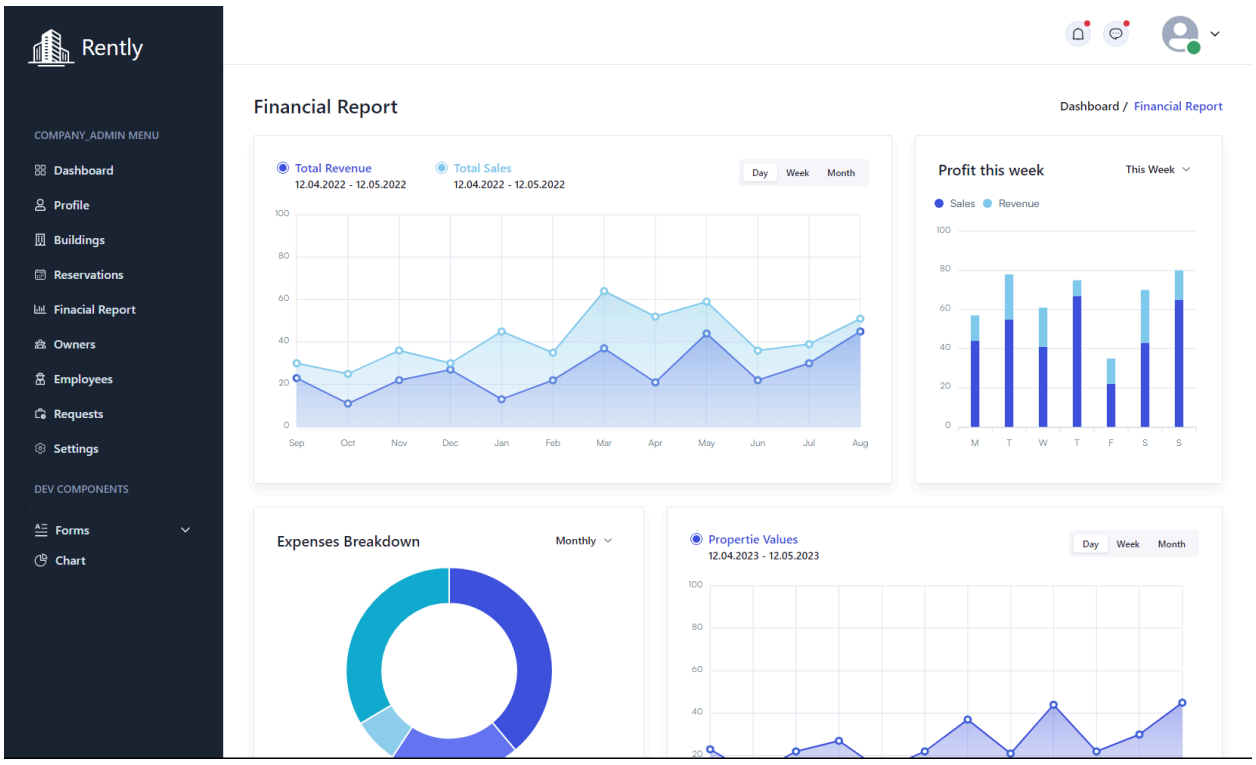
Overview

Sprint 4 will focus on enhancing the user experience by implementing critical features within our property management system. Below is the list of features to be developed, their status, and the assignment of UI prototypes for further development and integration.

Feature List and Status

1. REN-70 Company has access to Annual Financial Report

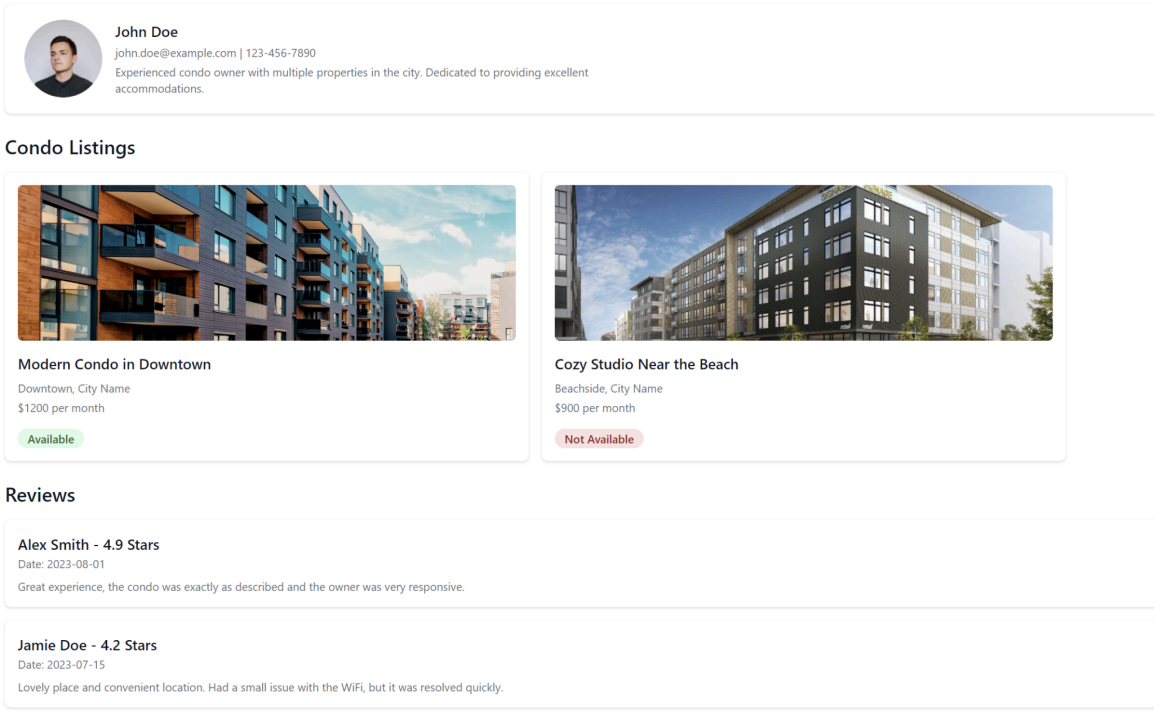
- Description: Page including the annual financial report for a company so that they can track their expenses, see the trends in their profits and manage their finances.
- UI Prototype:



Assigned to: Chems

2. REN-78 Owner Details Admin View

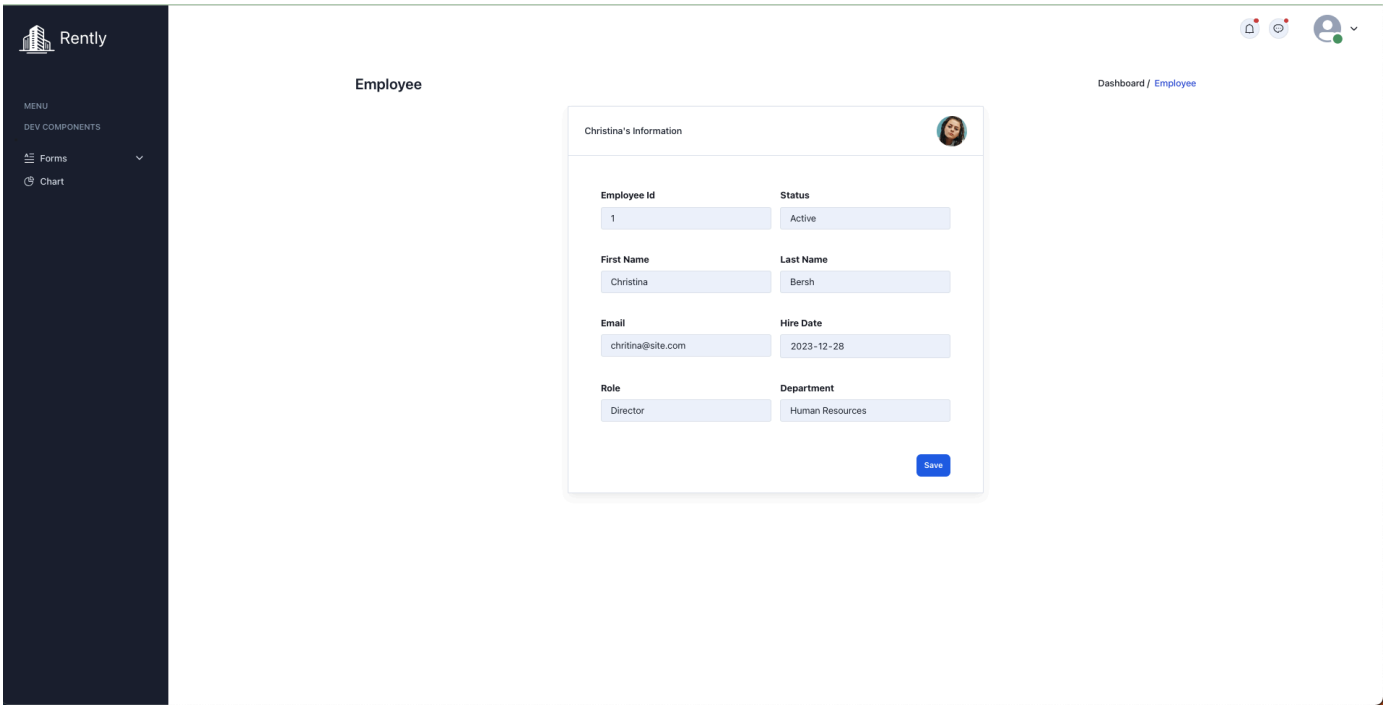
- Description: Detailed view for property owners, including personal information, property listings, and reviews.
- UI Prototype:



Assigned to: Adel

3. REN-63 Set roles for employees

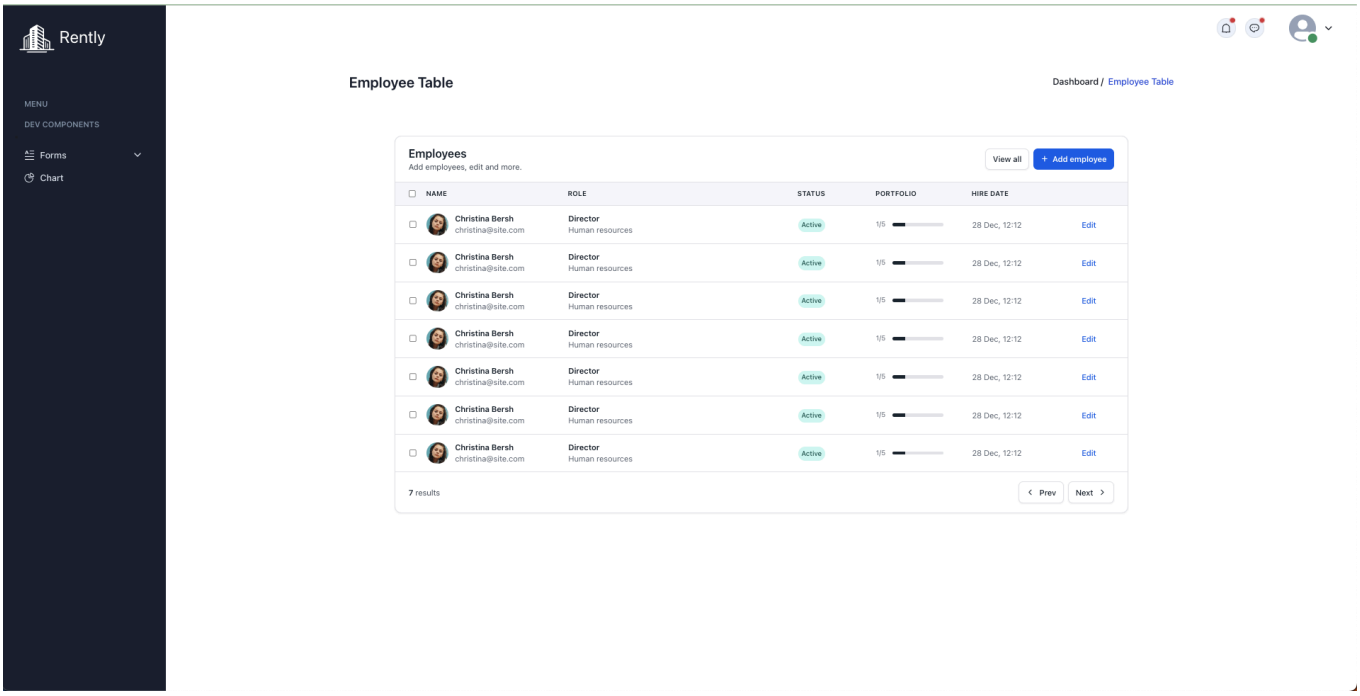
- Description: Dashboard for admins displaying an overview of buildings, owners, and employees.
- UI Prototype:

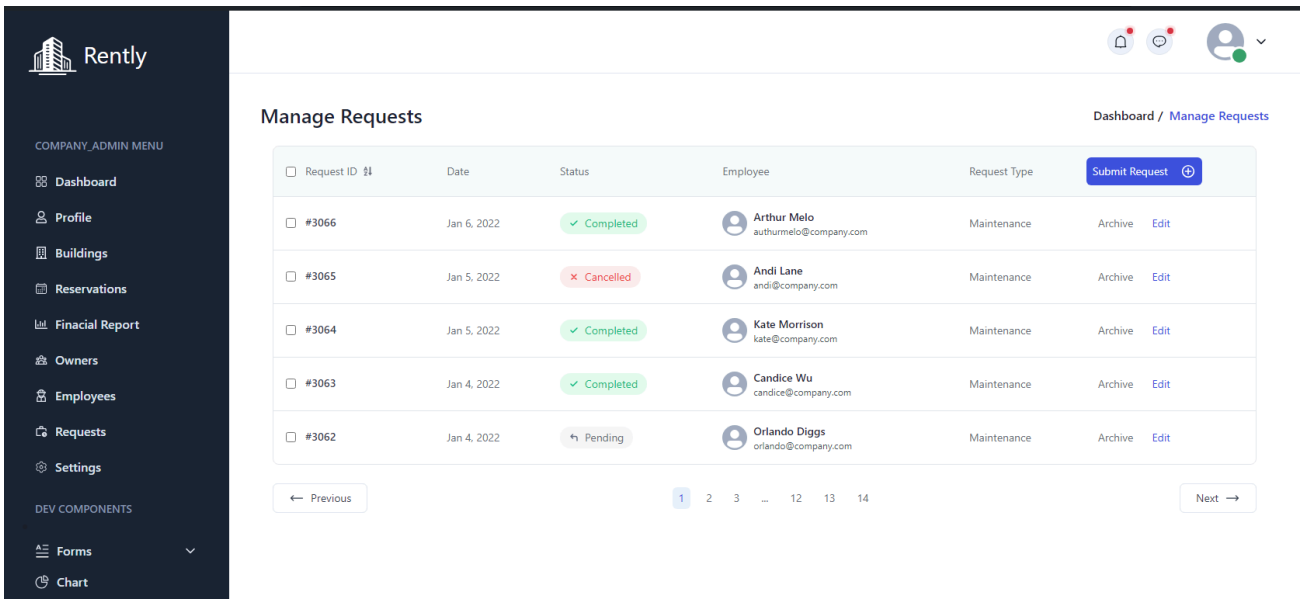


Assigned to: Francesco

4. REN-64 Company Admin can View Status of Requests Made

- Description: Interface to manage and be able to view the status of the requests made by the company.
- UI Prototype:

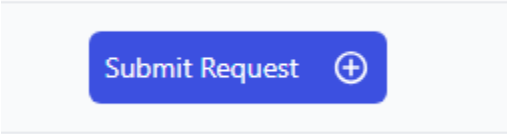


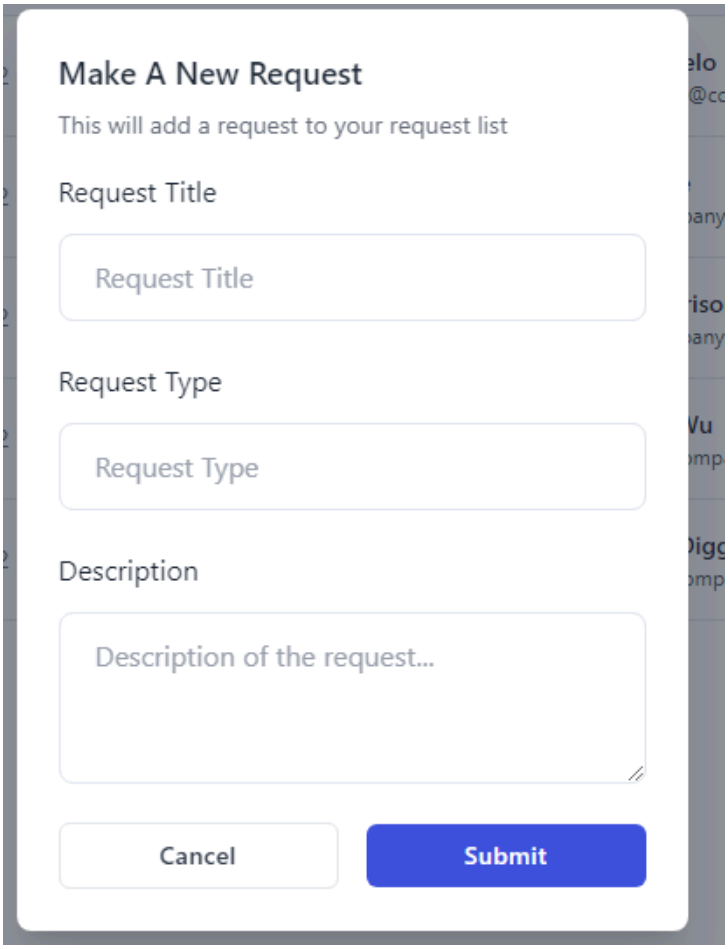


Assigned to: Chems

4. REN-66 Owner Can Submit Requests

- Description: Interface to allow for the owner to submit a request.
- UI Prototype:

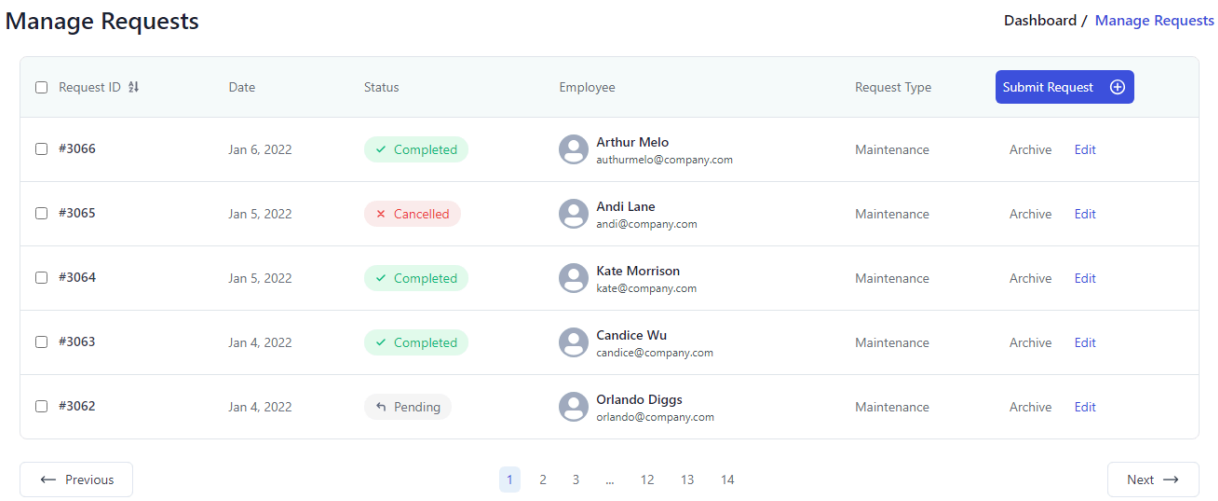




Assigned to: Adel

5. REN-66 can View Status of Requests Made

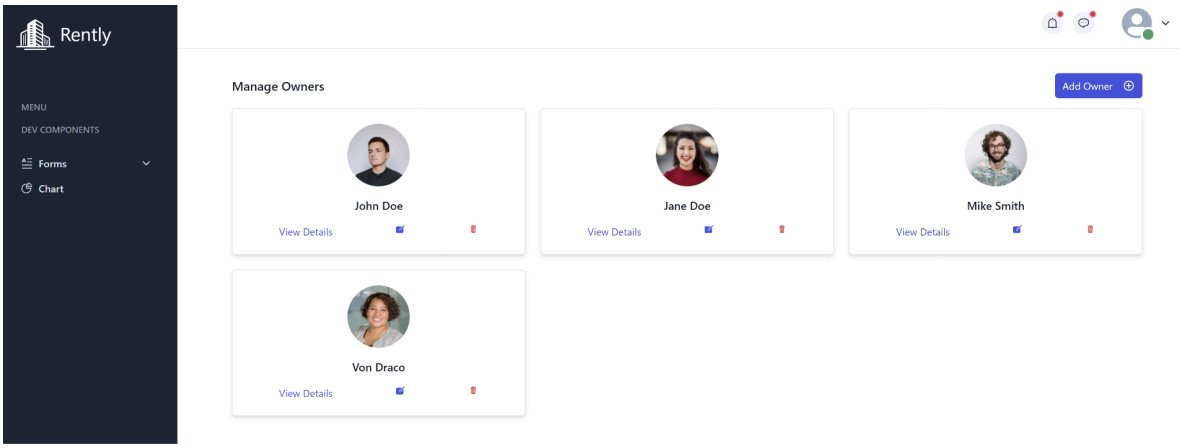
- Description: Interface to manage property owners, with options to add new owners and view details for existing ones.
- UI Prototype:



Assigned to: Chems

6. REN-78 Manage Owners as Admin

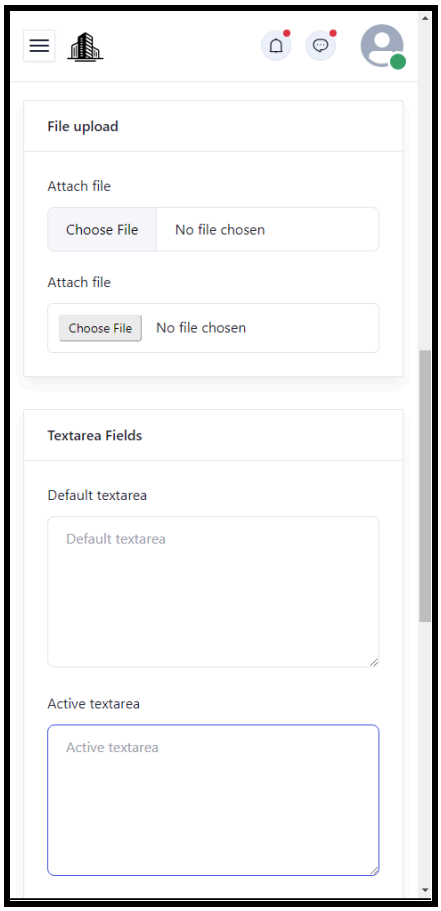
- Description: Interface to manage property owners, with options to add new owners and view details for existing ones.
- UI Prototype:



Assigned to: Adel

7. REN-61: File Upload

- Description: When managing a building, we need a section for users to upload files, with multiple file selection support.
- UI Prototype:



Assigned to: Chems

8. REN-62 Enter operational costs

- Description: Dashboard for admins displaying an overview of buildings, owners, and employees.
- UI Prototype:

Assigned to: Omar

Next Steps

The frontend team is to take the provided UI prototypes and will integrate them with the backend services to be developed in Sprint 4. Each team member will report progress in the weekly stand-ups and raise any issues for immediate resolution.

Deadlines

The integration of these features into the main development branch is scheduled for **March 28th**, with a buffer period for testing and refinements until **April 2nd**.

Additional Notes

- All UI prototypes are to be made responsive to accommodate various device sizes.
- The frontend team is to ensure that accessibility standards are met for all new features.
- Performance optimization and testing is a priority before the features are deployed to production.

Code Management

1. Quality of source code reviews

Source code reviews are conducted rigorously to maintain high code quality, facilitate knowledge sharing, and catch potential issues early. Reviews are performed by at least two team members other than the author, focusing on code correctness, maintainability, performance implications, and adherence to project standards. This practice helps prevent bugs, improve codebase understanding, and foster a culture of collective code ownership. We also separate the team in two, frontend and backend, which both have weekly meetings on the quality of source code, amongst other factors. In which they have the opportunity to clean any code smells, code cohesion, complexity as well as adding comments and documentation.

2. Correct use of design patterns

Since we are using Java spring boot as the backend of our application, we are constrained to use various design patterns to achieve our goals. In this example, we showcase 3 of the many design patterns we are using: Facade pattern, Repository design pattern and Data transfer Object design pattern.

A. **Facade** design pattern:

```
1 usage   Abdel
@Override
public SystemAdminDto registerSystemAdmin(SystemAdminDto systemAdminDto) {
    if (userExistsForRegistration(systemAdminDto.getEmail()) == false) {
        return systemAdminService.registerSystemAdmin(systemAdminDto);
    }
    throw new AuthenticationException("User with email " + systemAdminDto.getEmail() + " already exists.");

}

1 usage   Abdel +1
@Override
public CompanyAdminDto registerCompanyAdmin(CompanyAdminDto companyAdminDto) {
    if (userExistsForRegistration(companyAdminDto.getEmail()) == false) {
        return companyAdminService.registerCompanyAdminAndLinkToCompany(companyAdminDto);
    }
    throw new AuthenticationException("User with email " + companyAdminDto.getEmail() + " already exists.");
}

1 usage   Abdel
@Override
public EmployeeDto registerEmployee(EmployeeDto employeeDto) {
    if (userExistsForRegistration(employeeDto.getEmail()) == false) {
        return employeeService.registerEmployee(employeeDto);
    }
    throw new AuthenticationException("User with email " + employeeDto.getEmail() + " already exists.");
}

1 usage   Zakaria +1
@Override
public PublicUserDto registerPublicUser(PublicUserDto publicUserDto) {
    if (userExistsForRegistration(publicUserDto.getEmail()) == false) {
        return publicUserService.registerPublicUser(publicUserDto);
    }
    throw new AuthenticationException("User with email " + publicUserDto.getEmail() + " already exists.");
}
```

The userService class acts as a facade to the other concrete user services. When registering a user, the controller uses the user service and it then delegates the creation to the right service. In the above image, it can be seen that in the userService class, several services are called according to the type of user that needs to be registered.

```
no usages   Zakaria +1
@PostMapping(path = "/create/company-admin")
public ResponseEntity<CompanyAdminDto> createCompanyAdmin(@RequestBody CompanyAdminDto companyAdminDto) {
    return ResponseEntity.ok(userService.registerCompanyAdmin(companyAdminDto));
}
```

In this image, we can see the controller using the user service class to register the user.

B. **Repository** design pattern :

```
package com.rently.rentlyAPI.services;

import com.rently.rentlyAPI.dto.UpdateProfileRequestDto;
import com.rently.rentlyAPI.dto.UserProfileDto;
import com.rently.rentlyAPI.entity.Key;
import com.rently.rentlyAPI.entity.User;
import org.springframework.data.jpa.repository.Query;

import java.security.Principal;
import java.util.Optional;

public interface UserService {
    Optional<User> findByEmail(String email);

    public User updateProfile(UpdateProfileRequestDto request, Integer userId);

    public UserProfileDto viewProfile(Integer userId);

    User activateKeyToChangeRole(String key);
```

```
import org.springframework.web.multipart.MultipartFile;

import java.security.Principal;

@RestController
@RequestMapping("/api/v1/users")
@RequiredArgsConstructor
public class UserController {

    private final UserServiceImpl userService;
    private final S3ServiceImpl s3ServiceImpl;

    //sample url : http://localhost:8080/api/v1/users/activateKeyToChangeRole?key=abc
    @PostMapping("/activateKeyToChangeRole")
    public ResponseEntity<KeyDto> activateKeyToChangeRole(@RequestParam String key) {
        userService.activateKeyToChangeRole(key);
        return ResponseEntity.ok().build();
    }

    @PatchMapping("/change-password")
    public ResponseEntity<?> changePassword(
          @RequestBody ChangePasswordRequestDto request,
          Principal connectedUser
    ) {
```

```
package com.rently.rentlyAPI.repository;

import com.rently.rentlyAPI.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Integer> {

  Optional<User> findByEmail(String email);

  @Query(value = """
        select u from User u inner join Key k
        on u.id = k.user.id
        where k.key = :key
        """)
    User findUserByKey(String key);

}
```

- The UserRepository interface acts as a repository for performing database operations related to the User entity.
- The UserService class provides methods to interact with the UserRepository. It encapsulates the business logic.
- The UserController class handles HTTP requests and responses. It interacts with the UserService to perform CRUD operations on User entities.

C. **Data transfer Object (DTO)** design pattern:

(Note - To avoid redundant information about the code, please refer to the previous images)

```
public static UserDto fromEntity(User user) {
    return UserDto.builder()
        .id(user.getId())
        .firstname(user.getFirstname())
        .lastname(user.getLastname())
        .email(user.getEmail())
        .role(user.getRole().name())
        .phoneNumber(user.getPhoneNumber())
        .bio(user.getBio())
        .build();
}

public static User toEntity(UserDto userDTO) {
    return User.builder()
        .id(userDTO.getId())
        .firstname(userDTO.getFirstname())
        .lastname(userDTO.getLastname())
        .phoneNumber(userDTO.getPhoneNumber())
        .bio(userDTO.getBio())
        .email(userDTO.getEmail())
        .role(Role.valueOf(userDTO.getRole()))
        .build();
}
```

```java
public class UserDto {

    private Integer id;

    @NotBlank(message = "The first name is required")
    private String firstname;

    @NotBlank(message = "The last name is required")
    private String lastname;

    @NotBlank(message = "The phone number is required")
    private String phoneNumber;

    private String bio;

    @NotBlank(message = "The email is required")
    private String email;

    @Builder.Default
    private String role = Role.USER.name();

    public static UserDto fromEntity(User user) {
        return UserDto.builder()
            .id(user.getId())
```

- We've introduced UserDTO to transfer data between the controller and service layers.
- The UserService class now works with UserDTO objects instead of User entities, converting them back and forth as needed.
- The UserController similarly operates with UserDTO objects instead of User entities.

3. Respect to code conventions

We adhere to a predefined set of coding conventions and style guides tailored for our tech stack, including naming conventions, file and folder organization, and programming practices. These conventions are documented and accessible to all team members, ensuring consistency and readability across the codebase. Regular tool-assisted code audits are performed to ensure compliance, using linters and formatters integrated into our development workflow. Head of front-end and back-end also occasionally ensure that every convention put in place is being respected by their respective team members.

4. Design quality (number of classes/packages, size, coupling, cohesion)

In our Spring Boot application, we've chosen to organize our code based on layers like controllers, DTOs (Data Transfer Objects), and services, rather than by individual features. This decision is rooted in simplicity and clarity, as it allows us to focus on one aspect of functionality at a time within each layer. By breaking down our application into distinct layers, we ensure that each component has a clear and well-defined purpose, making it easier to understand and maintain.

Moreover, this approach promotes scalability and testability. We can easily add new features or modify existing ones without affecting other parts of the application. Additionally, writing unit tests for individual layers becomes more straightforward, leading to more robust and reliable code. In terms of structure, we have ~25 packages, ~112 classes and ~320 methods. In average, every method contains 10-15 LOC.

Furthermore, organizing our code in this manner encourages reusability, as components within each layer can be utilized across different parts of the application or even in other projects. It also facilitates collaboration within our development team, as everyone can easily grasp the overall structure of the application and contribute effectively.

By adhering to common design patterns like MVC (Model-View-Controller), we ensure consistency and coherence throughout our codebase. This ultimately results in a more maintainable, scalable, and high-quality Spring Boot application that meets both developer and user needs effectively.

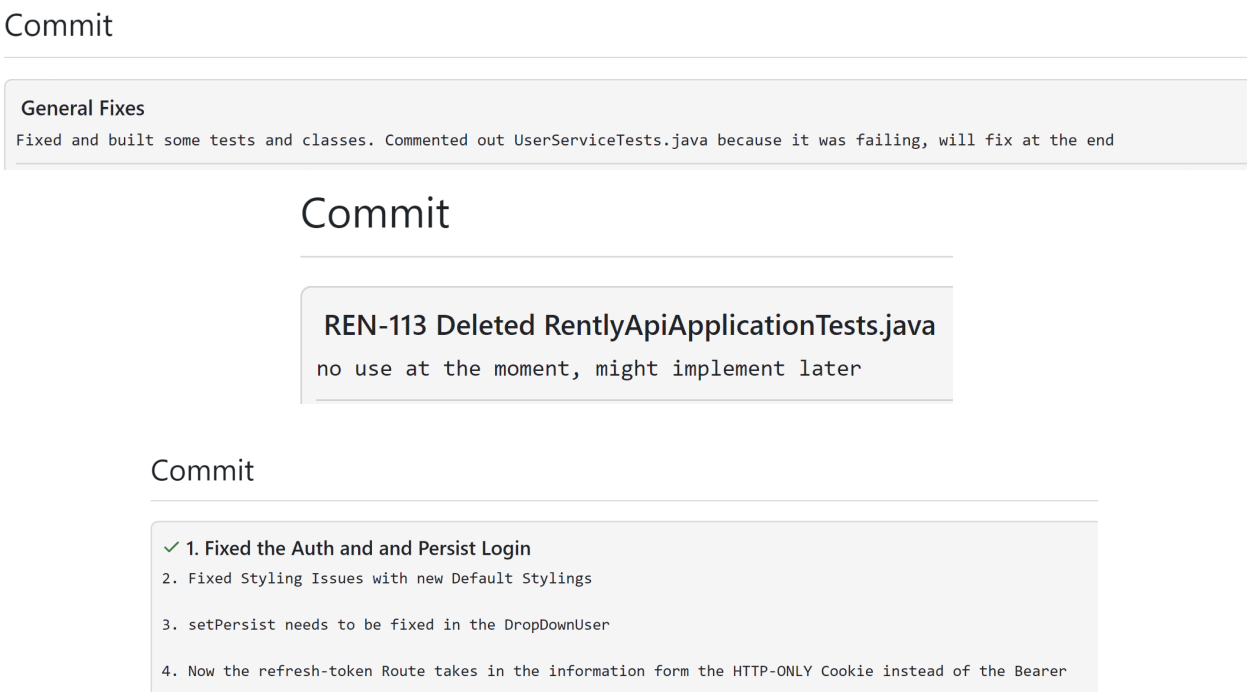
5. Quality of source code documentation

High-quality source code documentation is paramount for ongoing development and future maintenance. We document key components, functions, and classes, explaining their purpose, usage, and any peculiarities. Inline comments are used judiciously to clarify complex code sections. Additionally, high-level system architecture and API documentation are maintained to aid in system understanding and onboarding new team members.

6. Refactoring activity documented in commit messages

Refactoring is an integral part of our development process to improve the codebase's structure and readability without changing its behavior. Refactoring activities are clearly documented in commit messages, outlining the reasons for changes, the nature of the refactor, and any potential impact. This transparency aids in historical understanding and review processes.

Examples of Refactoring Commits



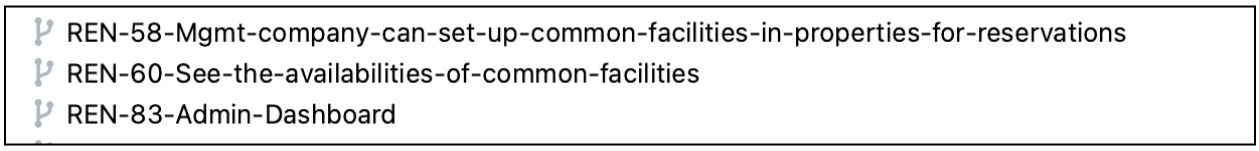
7. Quality/detail of commit messages

We enforce a standard for detailed and informative commit messages, comprising a succinct title followed by a comprehensive description of the changes made, the rationale behind them, and any relevant notes or acknowledgments. This practice facilitates easier code reviews, project tracking, and future reference, contributing significantly to the project's historical documentation.

In the example below, we can see that the commit mentions a problem that was fixed, as well as the breaking of a circular dependency and moving the responsibility of an action to another class.

8. Use of feature branches

The adoption of feature branches is a key strategy in our development workflow, allowing our team to encapsulate new developments and ensure the stability of the main branch and to make sure only working code will be pushed to it and won't affect other components. For instance, if we're working on a new authentication feature, we would create a branch named feature/authentication-enhancement. This branch will enable the team to develop, test, and refine the new feature in isolation from the ongoing activities in the main codebase. Another example is a new user interface for our application, developed in a branch called feature/new-ui-design. These branches are then rigorously reviewed and tested before being merged into the main branch, ensuring that each new feature integrates smoothly without disrupting the existing functionality. This branching strategy not only facilitates parallel development efforts across multiple features but also ensures that the main branch remains a stable and reliable base for the application. An example of our work in dealing with branches is in the image below.



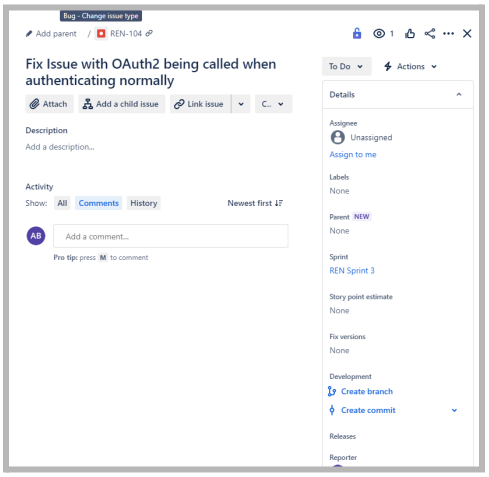
It can be seen in the image that separate branches are used for different features. For instance, the handling of the admin dashboard is handled on a branch separate from the one that will allow management companies to set up common facilities.

9. Atomic commits

Following the principle of atomic commits is a fundamental of our version control strategy, emphasizing the importance of each commit as a singular, coherent change within the project's codebase. This disciplined approach to committing ensures that every change, no matter how minor, is recorded in a manner that precisely captures the developer's intention at that moment. Atomic commits facilitate a granular level of change management, allowing for targeted revisions, rollbacks, and feature adjustments without the risk of unintended consequences that can arise from more complex, compounded commits. The methodology behind atomic commits involves breaking down work into the smallest possible units of change that remain functional and logical on their own. This practice enhances the project's history, making it not only more understandable for current team members but also accessible and decipherable for future contributors. Debugging becomes significantly more straightforward under this system, as developers can pinpoint the exact commit that introduced an issue, reducing the time and complexity involved in tracing bugs. Moreover, atomic commits serve as the foundation for an efficient and effective code review process. Reviewers can assess each change in isolation, ensuring a thorough evaluation without the distraction of unrelated modifications. This focused review process leads to more insightful feedback and, ultimately, a higher quality codebase.

10. Bug reporting

The next step in the life-cycle of a bug is validating and fixing it. Validation means the process by which the testing team or the person assigned can ensure that the bug is a legitimate issue in the system, and not one that may be caused by the device state of the user.



Once a bug report has been received, a JIRA ticket (with the “bug” tag) can be created to keep track of the issue, as seen to the left. This is a JIRA ticket that was created with a bug pertaining to OAuth2 being called when normal authentication is needed.

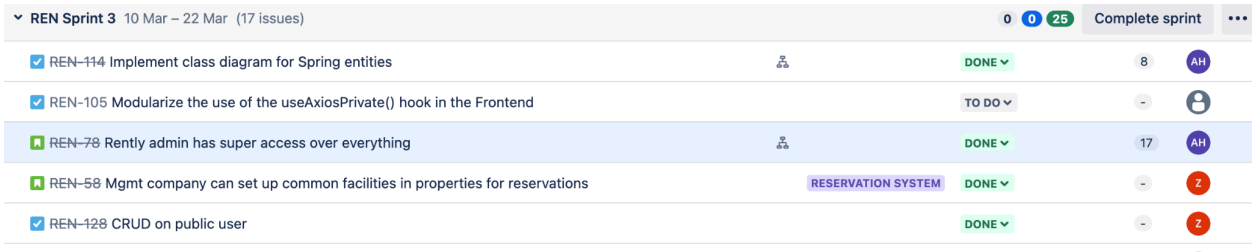
Testers will often verify the bug, and if it can be reproduced, they will mark down the steps to reproduce it. These steps will allow future validators to verify if the bug has been appropriately fixed. They will mark these steps in the JIRA ticket, and move the ticket to “In Progress”

Then, a developer will begin fixing the issue, often done by verifying the build of the system and checking the module that they believe is causing the issue. From there, they may determine

which section of the code is where the issue first arises. Afterwards, they will attempt to fix the bug at hand. Once the bug has been fixed, the developer will put forth comments in the JIRA ticket explaining the issue and what fix was put forward. It can then be marked as “Ready for Review” in which case a tester will verify that the bug has been fixed and that other issues are not caused as a result of the fix. Once the bug has been determined to be fixed, a Pull Request can be generated by the Developer and await approval.

11. Use of issue labels for tracking and filtering

The strategic use of issue labels in our project management workflow plays a big role in elevating the efficiency and clarity of our development process. By efficiently categorizing every task, bug, and enhancement with descriptive labels, we've established a robust framework that not only streamlines task management but also significantly enhances team productivity. These labels, representing diverse dimensions such as task type ('bug', 'enhancement'), priority ('high', 'low'), and status ('in progress', 'review'), enable precise filtering and quick access to relevant issues, facilitating targeted focus and efficient allocation of resources. This nuanced approach to issue labeling goes beyond mere organizational benefit. It fosters a culture of transparency and enhances collective understanding of project dynamics. Moreover, it plays a role in allowing team members to easily navigate through the project's structure, prioritize actions based on immediate needs or strategic importance, and align their efforts with the project’s overarching goals. On top of that, the adoption of a consistent labeling strategy ensures that all stakeholders, from developers to project managers, share a common understanding of the project's current state, upcoming priorities, and potential tasks, making it a basis of our project's success.



12. Links between commits and bug reports/features

Integrating commits with bug reports and feature requests is a basis of our approach to maintaining high levels of traceability and accountability throughout the development cycle. This integration is achieved by carefully ensuring that every commit message includes a direct reference, such as an issue number or a unique identifier, to the corresponding task or bug report in our project management system. For example, a commit fixing a login issue might be documented as “Fixes #123 - resolve login failure for edge cases,” where #123 is the issue number in our tracker. Similarly, a commit introducing a new user profile feature might be noted as “Implements feature #321 - add user profile page,” clearly linking the code change to its feature request. This methodical approach not only enhances the workflow efficiency but also significantly boosts the documentation quality of our project without any disadvantages to it. It enables both developers and stakeholders to seamlessly track the evolution of features and the resolution process for identified bugs. By fostering a disciplined connection between code commits and their respective issues or features, we establish a transparent step by step record that enhances project visibility and aids in the review process. Contributors / Developers take greater ownership of their work, knowing that their contributions are part of a structured effort to achieve project milestones. Moreover, this strategy promotes collaborative review and discussion, as each commit is contextualized within the broader project goals, ensuring that our development efforts are not just aligned but also well-documented and easily navigable. Through this practice, our project progresses in a well and systematically documented direction, driving towards our collective objectives with clarity and accountability.