SOEN 390

Team 15 - Deliverable 2

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 Fares	40162541
Oussama Cherifi	40212275
Zakaria El Manar El Bouanani	40190432

Table Of Contents

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

Figure 4.1 Rently System Block Diagram

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 2 User Stories Backlog

REN-4 As a user I want to login to the system. 30 USP

- Setup an authentication system with JWT and basic roles
- Setup frontend connection to AUTH
- Create authentication unit tests
- Implement OAuth2 with Google

REN-53 As a public user, I want to link my profile to my unit to have access to the services associated to my unit (become an occupant). 8 USP

- BE implementation of key generation per condo for management company
- BE implement key activation for user to become occupant or renter
- FE implementation of the related forms for activation keys

REN-54 As a public user, I want to create a profile to have access to the website 18 USP

- FE Connect setting user profile page to the developed endpoints
- FE Make a profile page
- BE Setup image storage system for profile pictures for public users (S3 Bucket/Flickr)
- BE Add relevant attributes to user class and implement relevant services

REN-55 As an owner I want to have a dashboard of my properties to see everything related to them, like fees, status of requests and more. 16 USP

- BE Implement the relevant services and endpoints
- FE Implement user dashboard and connect it to the endpoints

REN-86 As a management company, I want to add, modify and delete my buildings. 13 USP

- FE Create form related to adding new buildings
- FE Make form to modify building
- BE Implement related endpoints for adding a building
- BE Implement related endpoints to modify/delete buildings

REN-91 As a management company, I want to add, modify and delete my condos in my buildings. 13 USP

- BE Implement related endpoints for adding condos
- BE Implement related endpoints to modify/delete condos
- FE Create form related to adding condos in a building
- FE Make form to modify condo

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

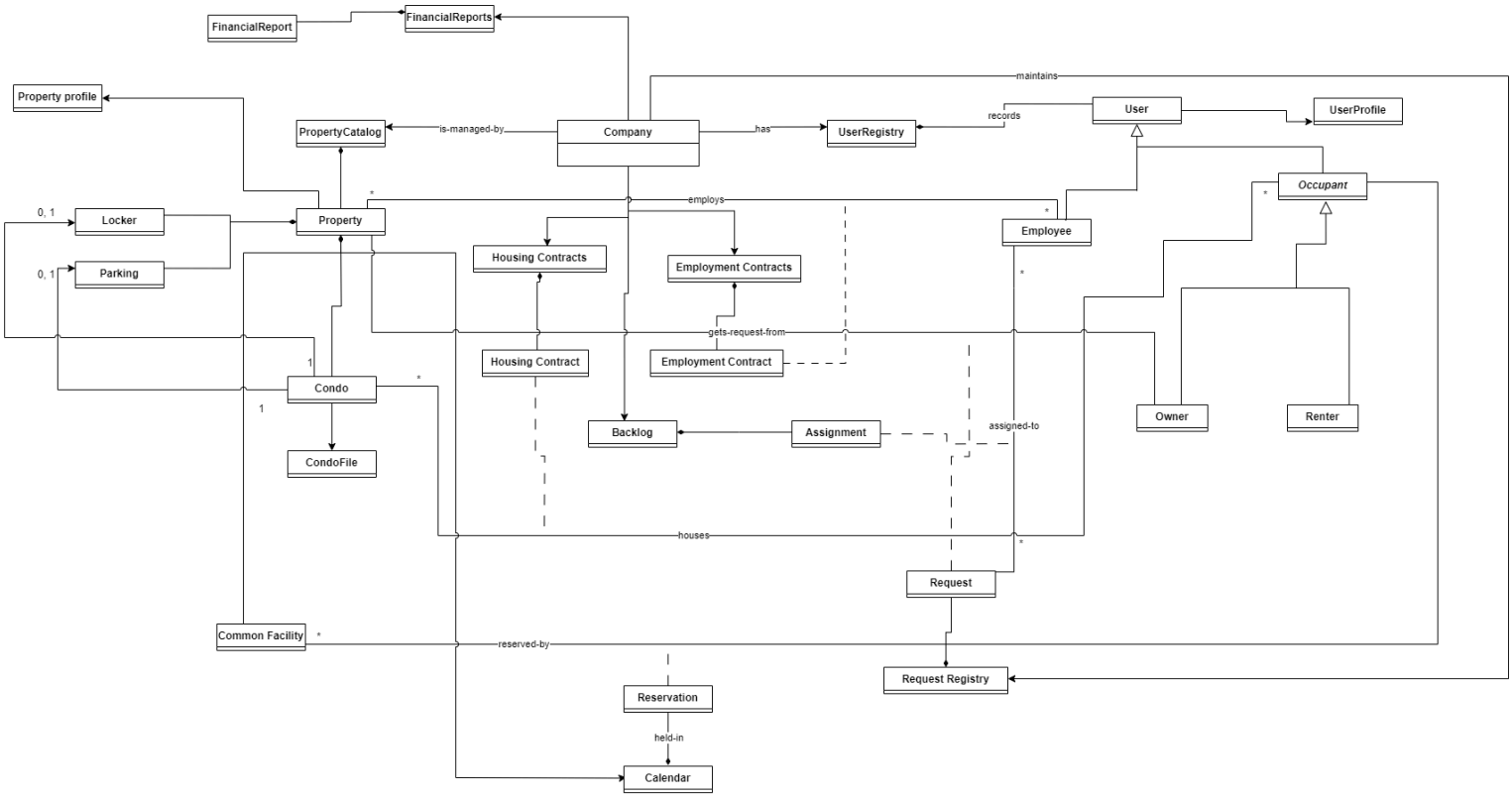


Figure 1: Domain model

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

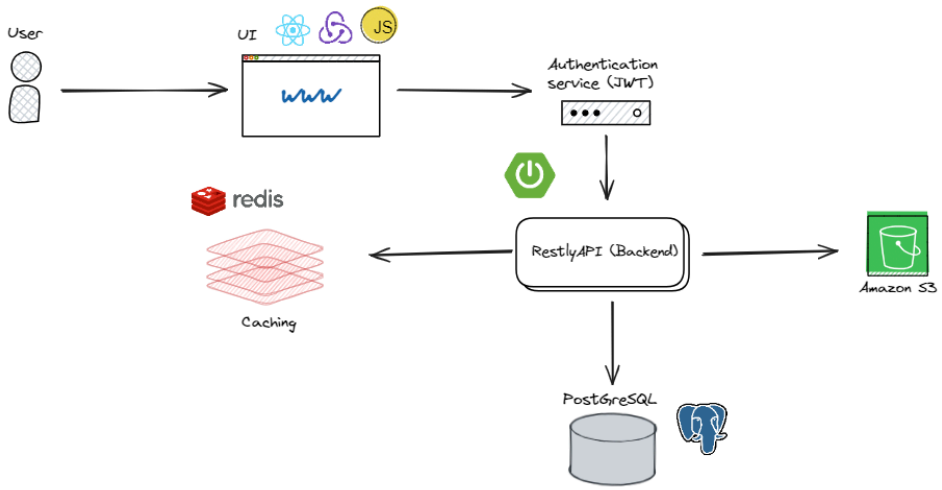


Figure 2: Component and architecture diagram

4.1.3 Use Case diagram

Governing model kind: Use case diagrams

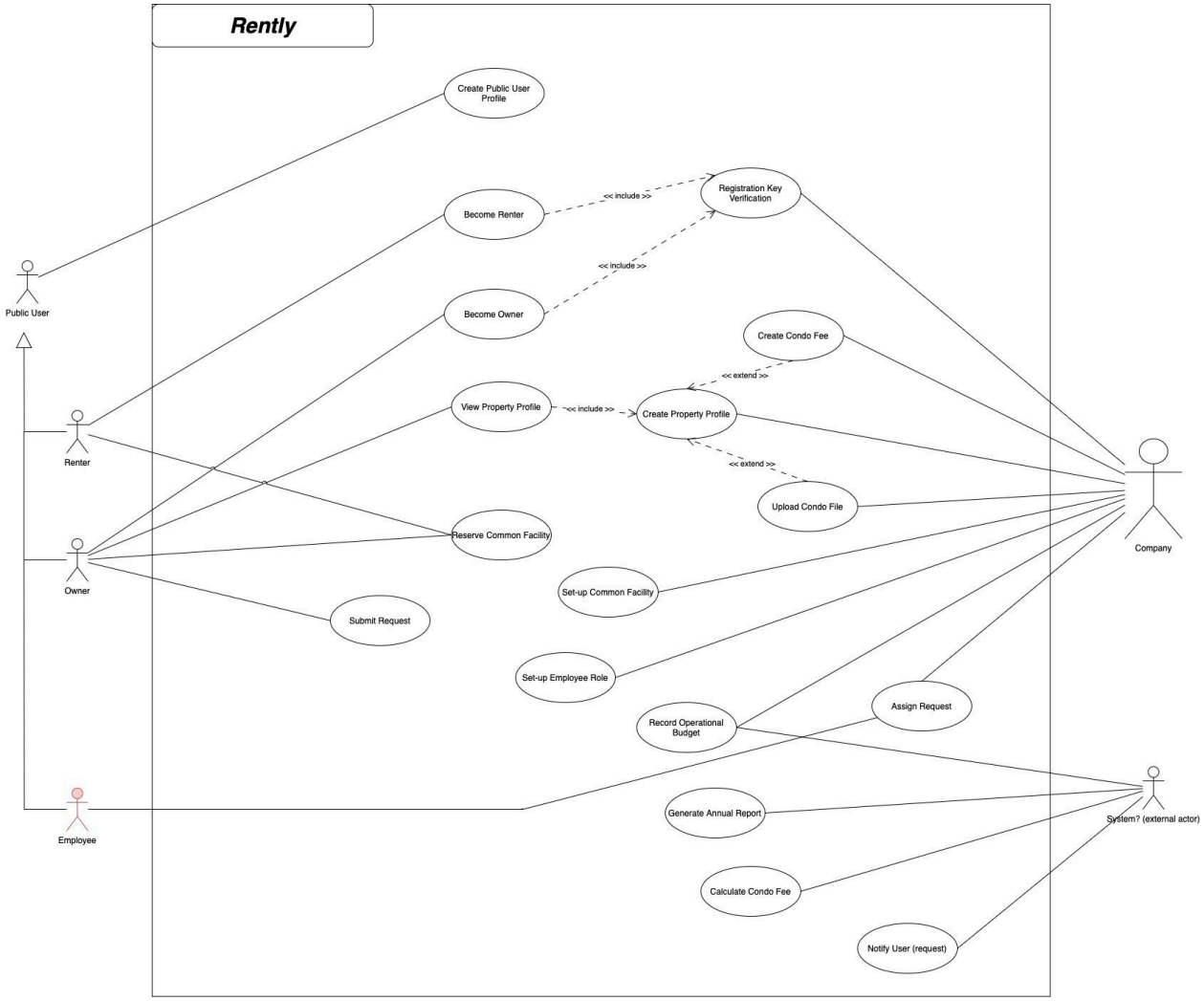
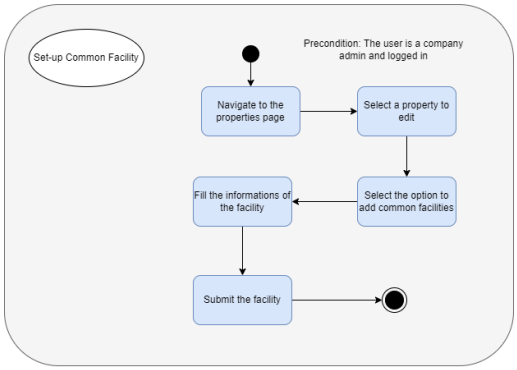


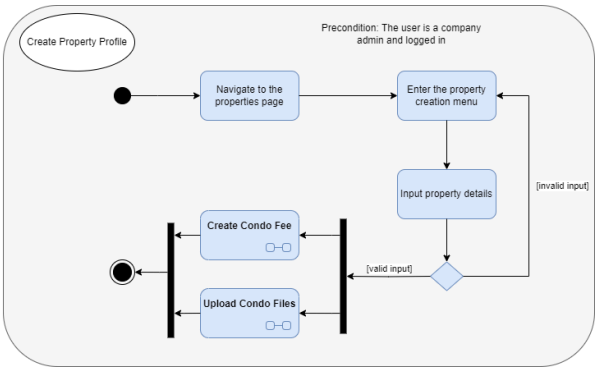
Figure 3: Use case diagram

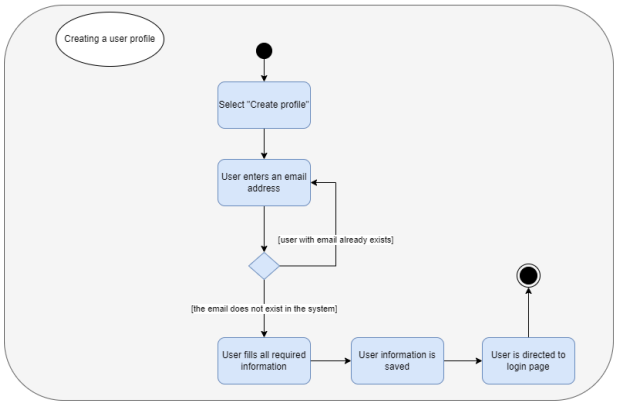
The use case diagram shows all the actors that will interact with the system upon completion of the development. Three external types of actors, the renter, owner and employee fulfill various use cases, where some also involve the action of a system employee.

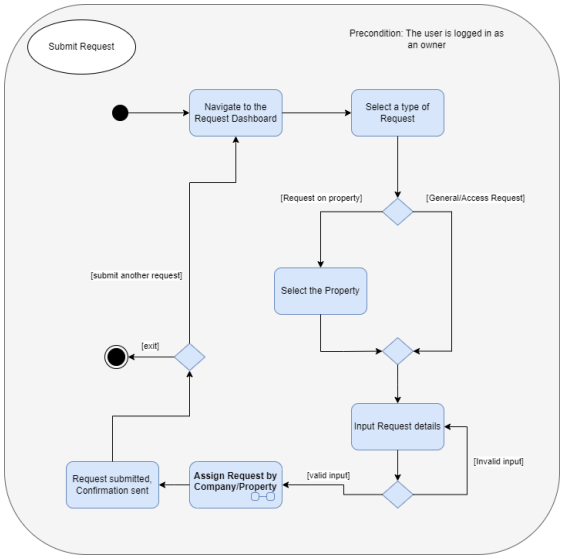
4.1.4 Activity diagram

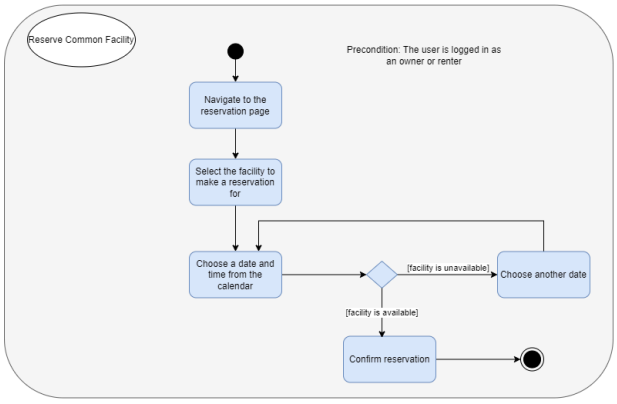
Governing model kind: Activity diagrams

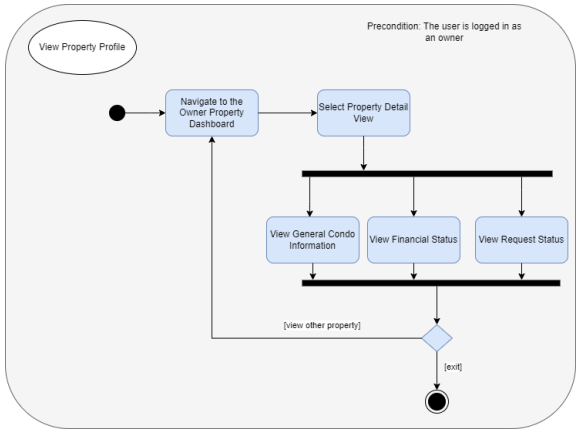












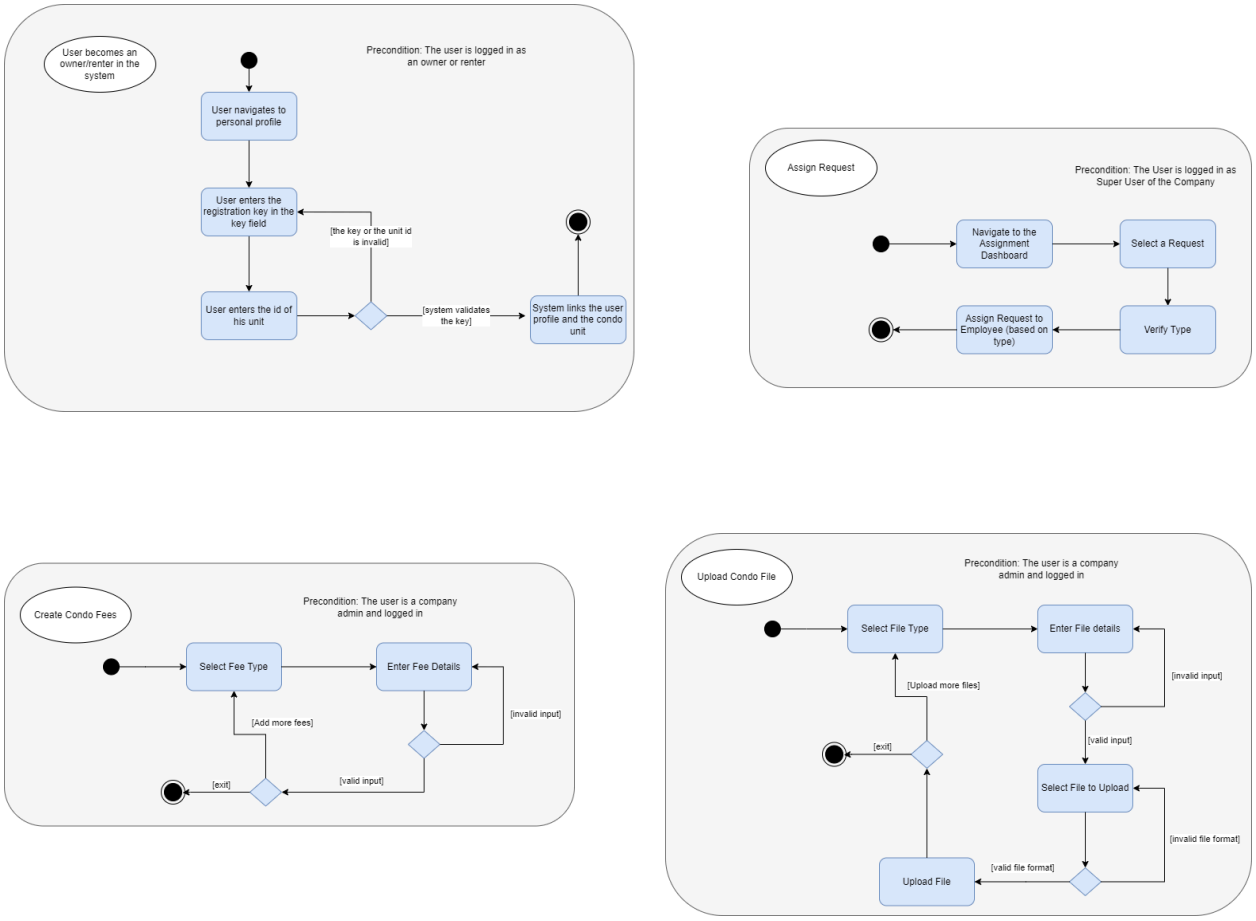


Figure 4: Activity diagrams

Figure 4 represents 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

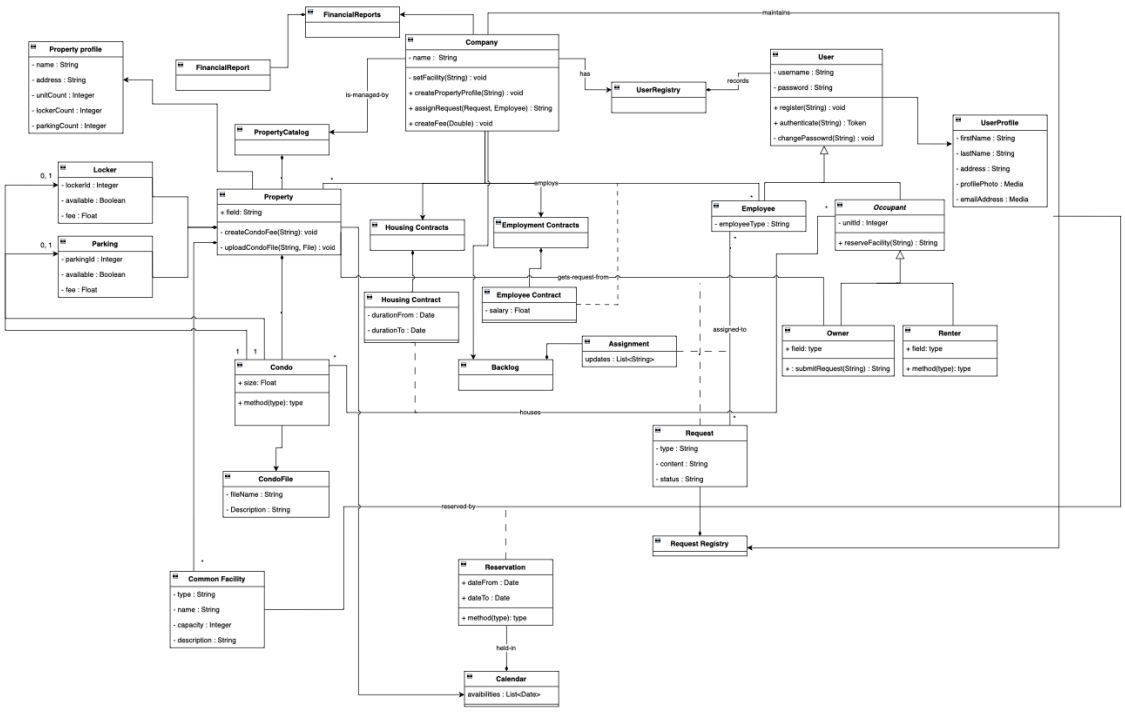


Figure 5: Class diagram

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

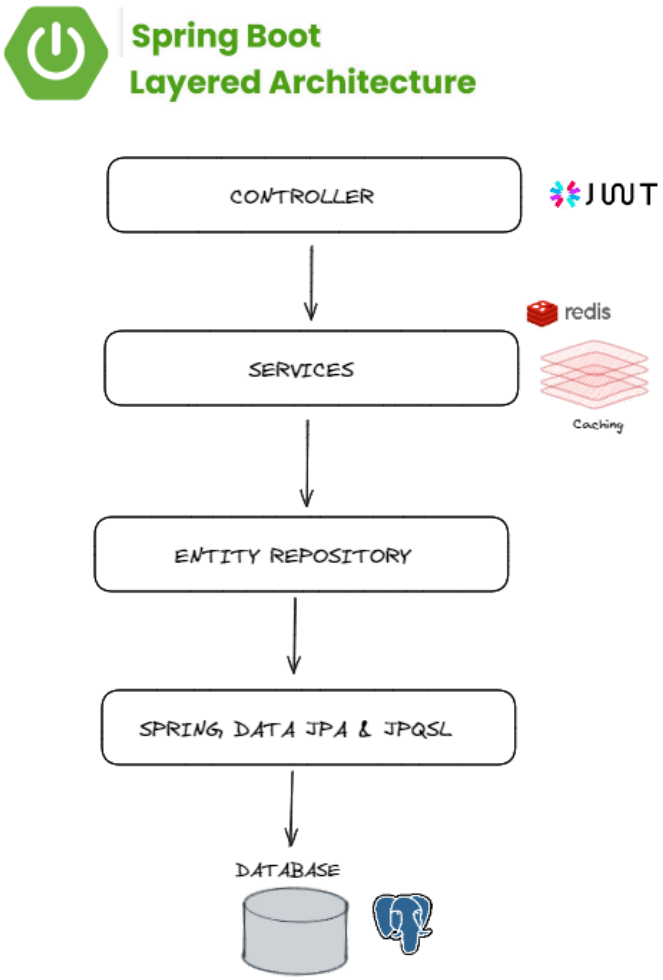


Figure 6: Backend architecture diagram

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

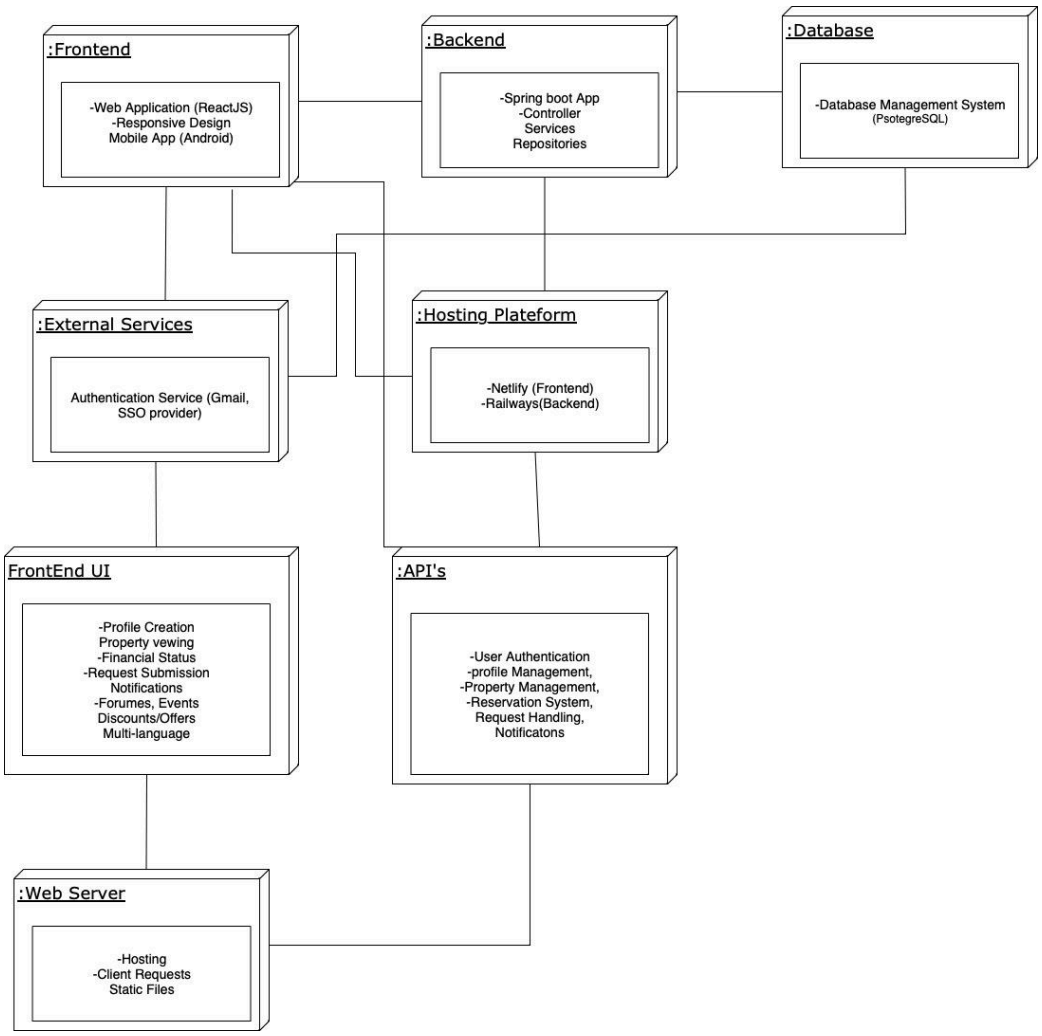


Figure 7: Deployment diagram

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.

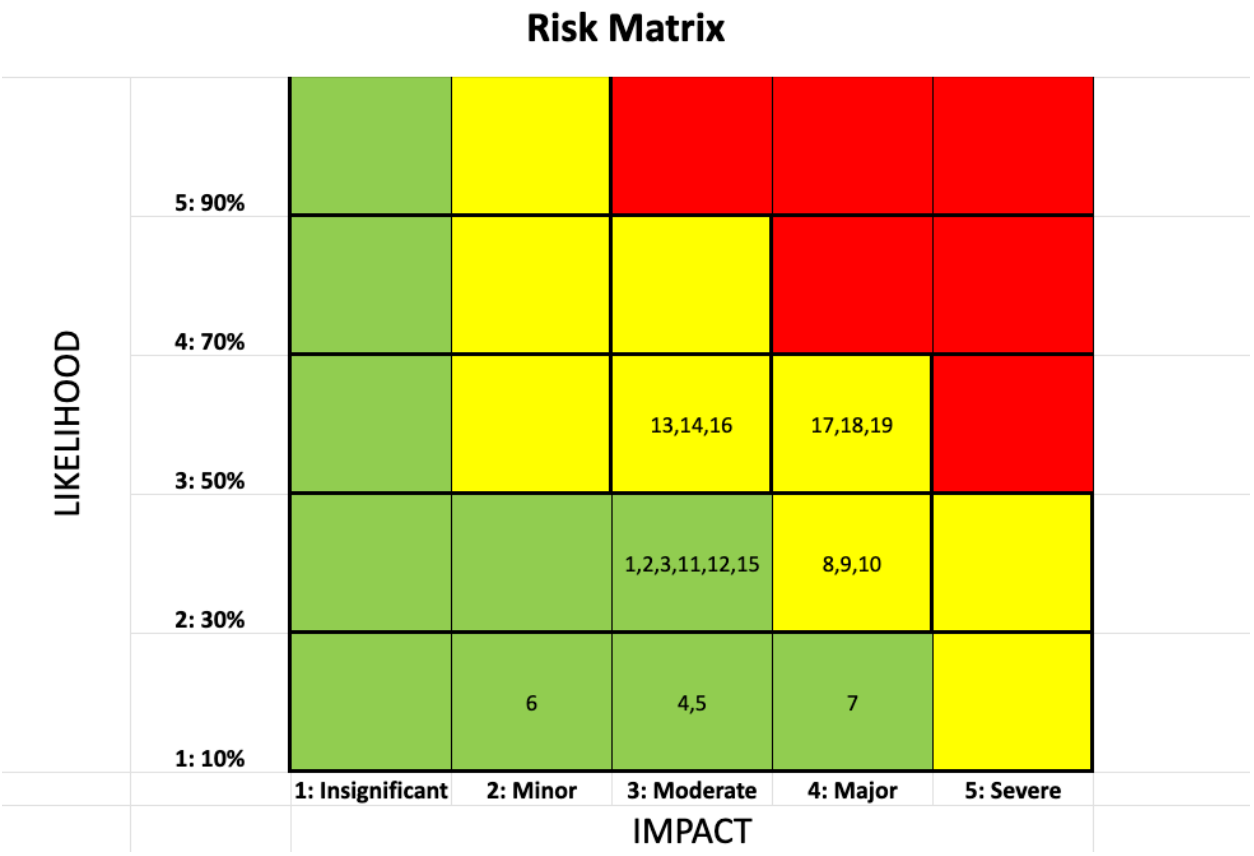


Figure 2: Rently Risk Matrix with risk IDs

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

Testing plan & Testing Coverage

The current test cases are built using Junit. For the next sprint, we plan to integrate JaCoCo, HemCrest and other libraries to improve and facilitate the testing process.

Current overall coverage summary : 80%+ code coverage

Current scope: all classes

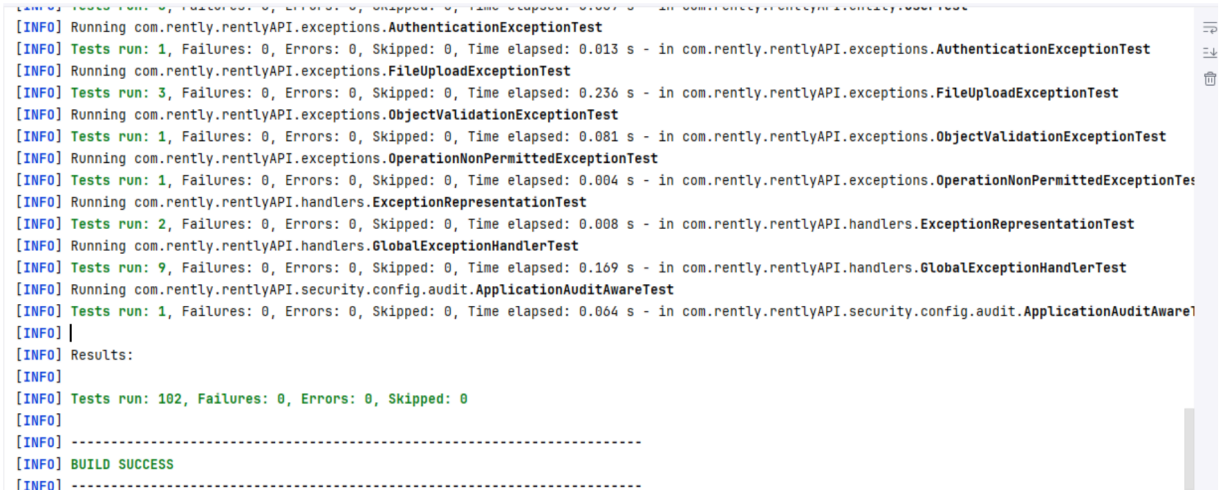
Overall Coverage Summary

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)

Sample test build :



Rently Sprint 2 Retrospective

Introduction

This postmortem is written in a context where the second sprint is completed, with features already functional. We are using Spring Boot in the backend and React in the frontend to develop Rently, our condo management platform. We deviated from our initial plan for sprint 2, and had to implement features different from those we said we would. We also realized at the end of the sprint our approach is not very efficient on the backend side, and that we will have some major refactoring to do in the 3rd sprint. Specifically, we did not implement the class diagram and then from there worked on our features, but rather we chose features and worked on implementing the class diagram as we went. Some things went well, like the team collaboration and a good separation of tasks.

What went wrong

1 - Setting up the AWS S3 bucket to store files

We faced a few problems when setting up the AWS S3 bucket to store files. Firstly, we had issues creating an MAI account, which is a more secure sub account with particular privileges. Then after creating the account, we had to find the appropriate service (S3 bucket) and then find the access key and the secret key. Basically, the setup for this service was problematic and took a lot of time for us to be done with it. After the setup, we had another issue to allow external users to read and write to the storage bucket. There were also a few problems on the backend when trying to use the AWS dependency within Spring Boot. The online tutorial Spring gave was outdated, so we had to use the AWS documentation, which was not very clear.

2 - Problems using online shareable resources

We encountered problems related to online resources that are free but with restrictions. For example, when using Postman to test our endpoints, we made a collection of endpoints with the bodies of requests so that for future tests we just run them without wasting time. We wanted to share the collection with other teammates, but when we reached 5 people in the Postman workspace, we got put into a free trial for 2 weeks and we do not know yet what will happen after. We will likely have to manually share the collections after any endpoint is modified, which is very inconvenient, keeping track of how 9 people are using them. Similarly, we needed a database that would allow us to work collaboratively, but it was hard to find. We decided that for those working on the backend, they would each use their own separate database, so they need to create their account on Neon, and those working on the frontend would use the same database, since they do not need it as extensively as the backend developers. Overall, it was a bit hard to navigate the resources and their limitations.

3 - Implementation of the class diagram

One major problem we realized we had caused was related to the class diagram. We did not start the coding process by implementing the class diagram and then working our way up from there. Instead, we created the basic classes, like condo, building and user, and ignored a lot of other components of the class diagram. For example, lockers and parkings are associated to building and condos, but they appeared nowhere in our code. We thought we could update the structure as we go, but we realized later that it increases the complexity by a lot, due to the amount of refactoring that was required whenever we needed to introduce a new component. We realized our approach was wrong and decided to refactor our code and implement the full diagram to have our structure ready at the beginning of sprint 3.

4 - Understanding of user roles

A problem we faced was about the different roles that are given to Rently users. For example, some members of the backend team understood a specific set of users, while others understood a different set. It was striking when the time came to make endpoints for companies to create buildings and condos. The user class was initially coded with lists of buildings and employees as attributes, to accommodate for company users. The problem with that is that regular users, like owners or renters are given attributes that are irrelevant to them. We basically considered the "company" entity to be a user. We decided that we would change it in the refactoring process at the beginning of sprint 3. This implementation made us realize that we needed to communicate more closely with each other to be sure of how things work for the most crucial parts of the software.

5 - Unclear set of user stories to be implemented

When we made the plan for sprint 2, we had decided on a few user stories to work on during sprint 2. When we started working, we realized that some of those stories needed to be pushed to later sprints because they were dependent on other parts being fully functional. For example, we saw that it does not really make sense to start thinking about common facilities and reservations before buildings, condos and basic operations with them are fully settled. We also had to change a lot of our user stories because we realized they did not cover the full set of functionalities of the project.

What went right

1 - Good frontend development process

The frontend development part was better structured than the backend part. There were not that many changes that were made, only additions to existing code. There also wasn't a major problem that was encountered that forced the frontend team to rethink their entire process, unlike

in the backend. There was close supervision of the subteam by their team leader, which meant that any concerns were quickly addressed.

2 - Clearing up expectations for future sprints
The problems we faced in this sprint allowed us to discuss more seriously how we would structure the rest of the project. We decided how the classes would be coded, their responsibilities and we decided to adhere as closely to our design as we could. This was a good thing because the discussion we had forced everyone to seriously think about the project in practical terms and gave everyone a better understanding of what needs to be done. If we did not face those problems mentioned above, they might have arisen in later sprints.

3 - Development flow
Everyone respected the rules we had set for how we use JIRA and how git actions are performed. We did not face any problems related to major commits not being linked to a branch or being untracked on JIRA. This showed us that the communication at the beginning of sprint 1 related to JIRA was clear and understood by everyone. We know that we will not have any problems related to this in the future, and that we can focus our attention on more time-consuming tasks.

4 - Sprint extension
We greatly benefitted from the unexpected extension to sprint 2 at the end of the sprint. We were able to allocate more time on the documents and really plan how to handle sprint 3 so that we do not face the same problems we did. We realized that because we were being squeezed by the deadline, we were not able to take a step back and look at all the problems we had. We were working and navigating problems at the same time, which was not done in an efficient way. The extension helped us clear everything and reassess our progress to correct ourselves.

Conclusion

Our takeaway from this sprint is that poor communication in certain areas can really quickly derail a project. We experienced it in the backend and took the necessary steps and decisions to correct ourselves in the future sprints. We also realized that staying true to a design is crucial to stay on the right track so that everyone can refer to the same thing and work coherently, instead of each person having a different view of different parts of the system, and then introducing incoherence when everyone’s work is brought together. For future sprints, we now know that the plan needs to be as clear as possible, and very thought out so that the proper stories can be implemented by members of the team.

Release Plan - Sprint 3 User Stories

As an occupant, I want to be able to manage my reservations and see other people's reservations.

- FE implementation a calendar interface
- BE reservations logic

As a management company, I want to be able to manage my buildings as well as individual units within them.

- FE simplified interface to manage buildings
- BE building management logic

As a management company, I want to be able to upload multiple files to specific buildings ie: id cards, personnel info of the tenant .

- FE simplified interface to upload files
- BE file support system for uploading and storing.

As a management company, I want to be able to manage owners, ie: add new owners, view details of existing ones,...etc.

- FE interface to view owner with options to manage
- BE logic to manage owners and keep track of changes

As a management company, I want to have a general idea of the buildings, owners and employees on a single dashboard.

- FE dashboard with all the information displayed in a simple manner
- BE logic to display information

As an occupant, I want to see the availability of the common facilities in the calendar so that I know when to make my reservation.

- FE display of the facilities's availability
- BE logic for retrieving availabilities

As an management company, I want to be able to register operational costs and keep track of certain info like the amount, a description, property, category…etc

- FE field to input info about the operational cost
- BE logic to store the info in the system

As an management company, I want to manage the roles for my employees

- FE dashboard of employees as well as option to manage their roles
- BE logic to manage roles and permission of employees

UI Prototypes for Sprint 3

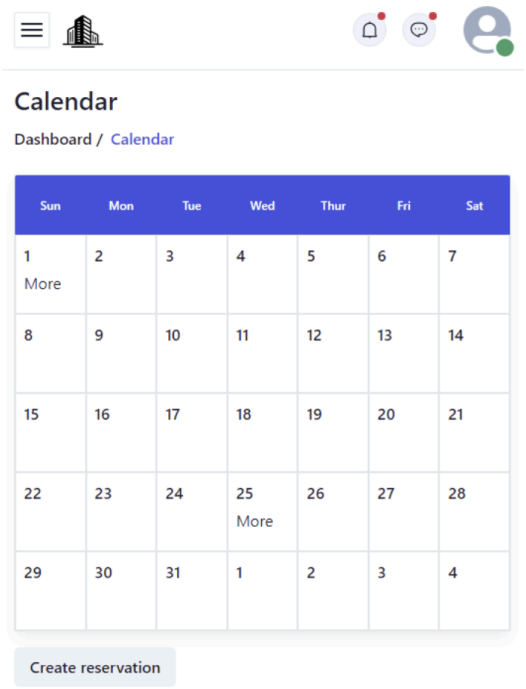
Overview

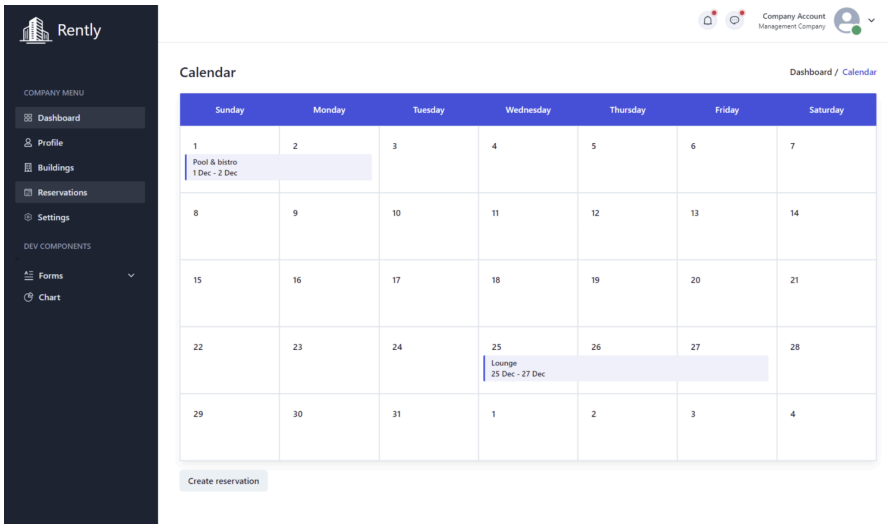
Sprint 3 will focus on enhancing the user experience by implementing critical features within our property management system. Below is the list of features to be developed, their status, and the assignment of UI prototypes for further development and integration.

Feature List and Status

1. REN-59: Calendar View

- Description: A calendar interface that allows users to view and manage reservations or bookings.
- UI Prototype:

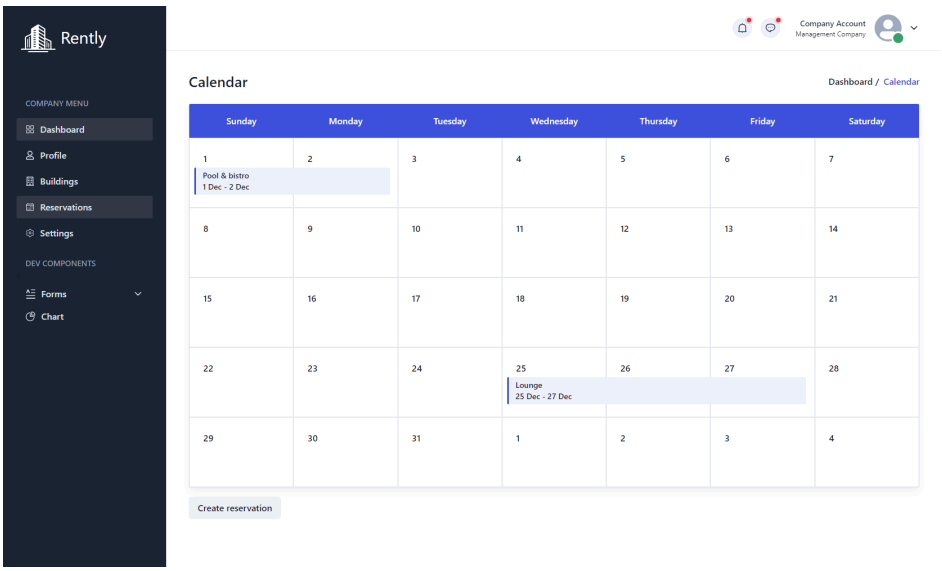




- Assigned to: Chems

2. REN-61: Manage Buildings

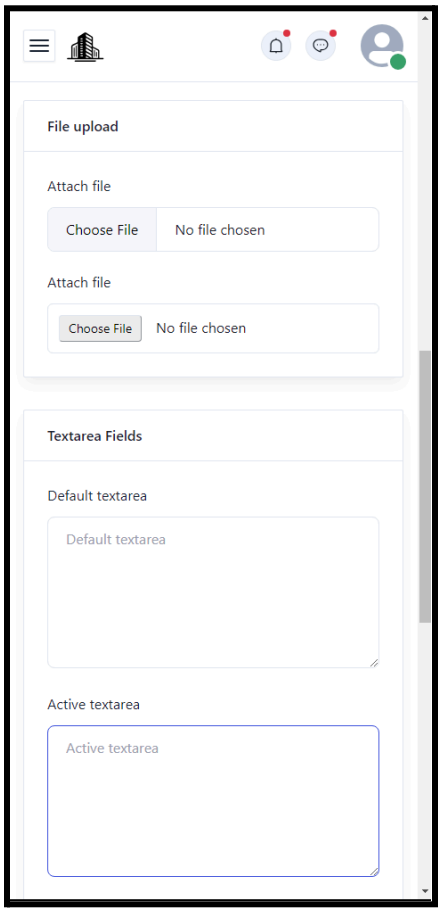
- Description: Interface for adding and managing multiple buildings, with a feature to manage individual units within each building.
- UI Prototype:



- Assigned to: Chems

3. REN-61: File Upload

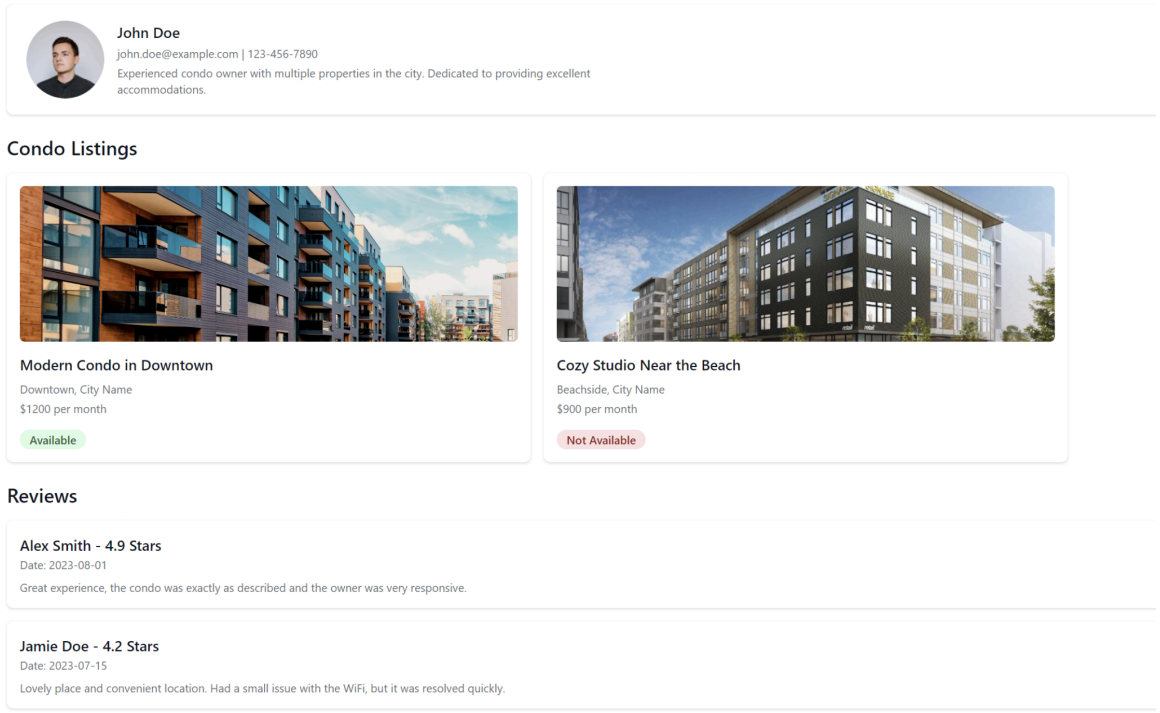
- Description: When managing a building, we need a section for users to upload files, with multiple file selection support.
- UI Prototype:



- Assigned to: Chems

4. REN-78 Owner Details Admin View

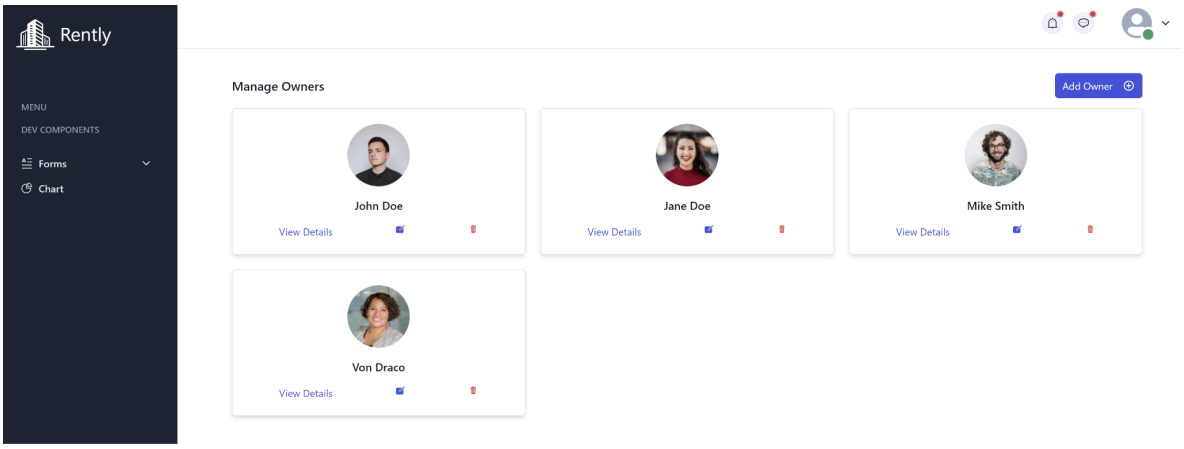
- Description: Detailed view for property owners, including personal information, property listings, and reviews.
- UI Prototype:



- Assigned to: Adel

5. REN-78 Manage Owners as Admin

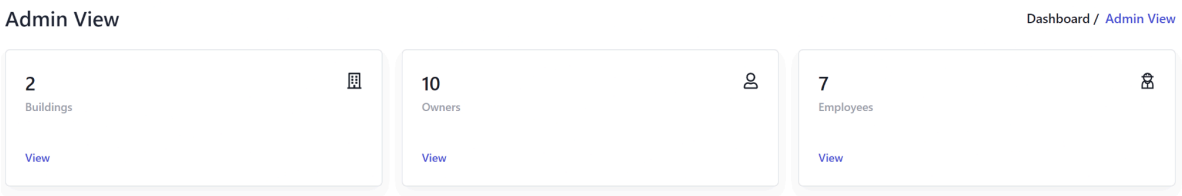
- Description: Interface to manage property owners, with options to add new owners and view details for existing ones.
- UI Prototype:



- Assigned to: Adel

6. REN-78 Admin Dashboard

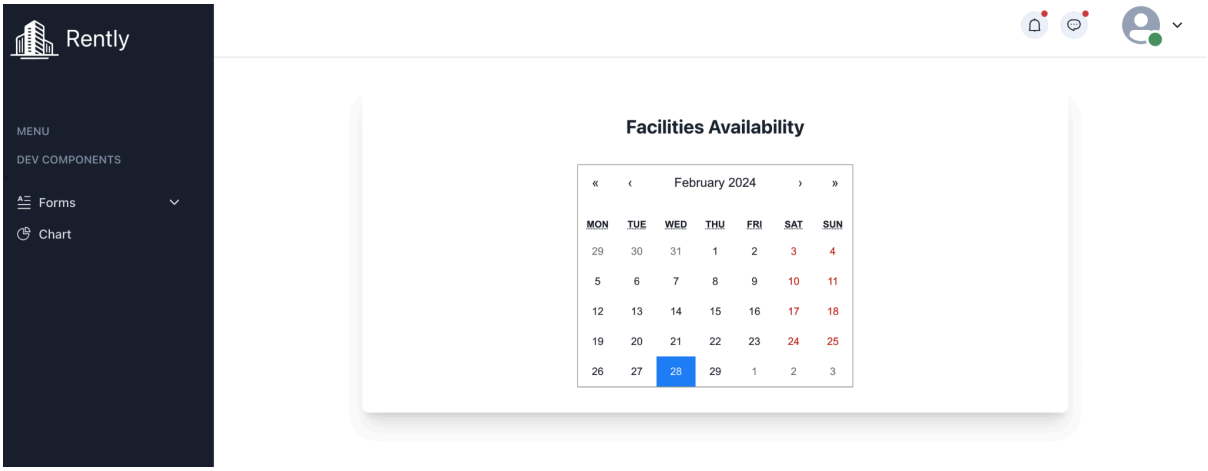
- Description: Dashboard for admins displaying an overview of buildings, owners, and employees.
- UI Prototype:



- Assigned to: Adel

7. REN-60 See the availabilities of common facilities

- Description: As an occupant, I want to see the availabilities of the common facilities in the calendar so that I know when to make my reservation
- UI Prototype:



- Assigned to: Omar

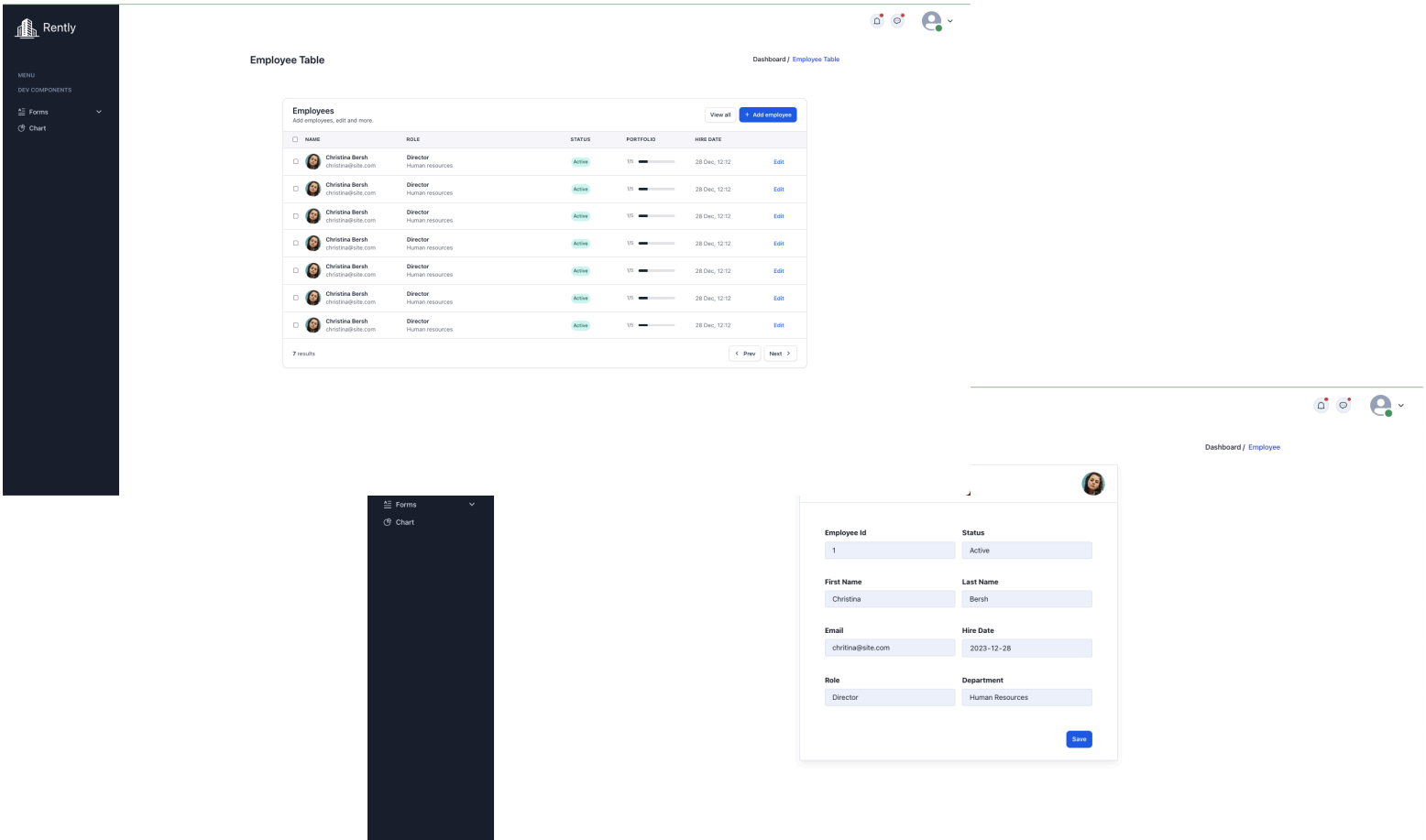
8. REN-62 Enter operational costs

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

- UI Prototype:
- Assigned to: Omar

9. Set roles for employees

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



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 3. 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 **February 29th**, with a buffer period for testing and refinements until **March 21st**.

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.