SOEN 390

Team 15 - Deliverable 5

Rently
https://rently-management.netlify.app/

Winter 2024

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

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 5 User Stories Backlog

Green: DONE

User Story	JIRA ID	USP	Priority	Status
As a public user, I want to create an account and login	REN-78	5	HIGH	DONE
As a system administrator, I want to create a company	REN-78	4	MEDIUM	DONE
As a system administrator, I want to create company admins and link them to a company	REN-78	5	MEDIUM	DONE
As a company administrator, I want to login	REN-78	3	HIGH	DONE
As a company administrator, I want to register employees	REN-78	5	HIGH	DONE
As an employee, I want to login	REN-78	3	HIGH	DONE
As a public user, I want to use an activation key to become an owner/renter	REN-53	5	HIGH	DONE
As a company administrator, I want to create a building	REN-86	5	HIGH	DONE
As a company administrator, I want to add condos to the buildings	REN-91	3	HIGH	DONE
As a company administrator, I want to set up common facilities in my buildings for users to reserve	REN-58	3	LOW	DONE

As an owner/renter, I want to make a reservation on a common facility in a calendar-like interface	REN-59	6	MEDIUM	DONE
As a company administrator, I want to send a registration key to public users to link them to a unit	REN-78	6	HIGH	DONE
As an owner/renter, I want to create a profile	REN-54	5	HIGH	DONE
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-55	4	MEDIUM	DONE
As an owner/renter, I want to see availabilities of common facilities	REN-60	6	MEDIUM	DONE
As an employee, I want to update assignments I am working on, like changing the status or adding comments	REN-136	5	MEDIUM	DONE
As a company administrator, I want to have access to a financial annual report to know what my expenses/budget were	REN-70	5	MEDIUM	DONE
As a company administrator, I want to enter operational costs for all operations made to know my expenses and budget	REN-62	4	LOW	DONE
As an owner, I want to access the condo files my management company made available for all units in my building	REN-65	4	LOW	DONE

As a company administrator, I want to add lockers and parking to my building	REN-134	4	LOW	DONE
As a company administrator, I want to upload condo files for my properties	REN-61	7	High	DONE
As a company administrator, I want to enter condo fees to units	REN-135	5	High	DONE
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	DONE
As a company administrator, I want to see the status of all the requests owners made.	REN-64	4	Medium	DONE
As an owner, I want to make special requests to be treated by employees	REN-66	4	Medium	DONE
As an owner, I want to see the status of my requests in a notification page	REN-67	5	High	DONE
As an employee, I want to see the owner requests that are assigned to me	REN-68	6	High	DONE
As an employee, I want to see the status of my assigned requests on a notifications page	REN-69	4	Medium	DONE
Total USP		130		

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

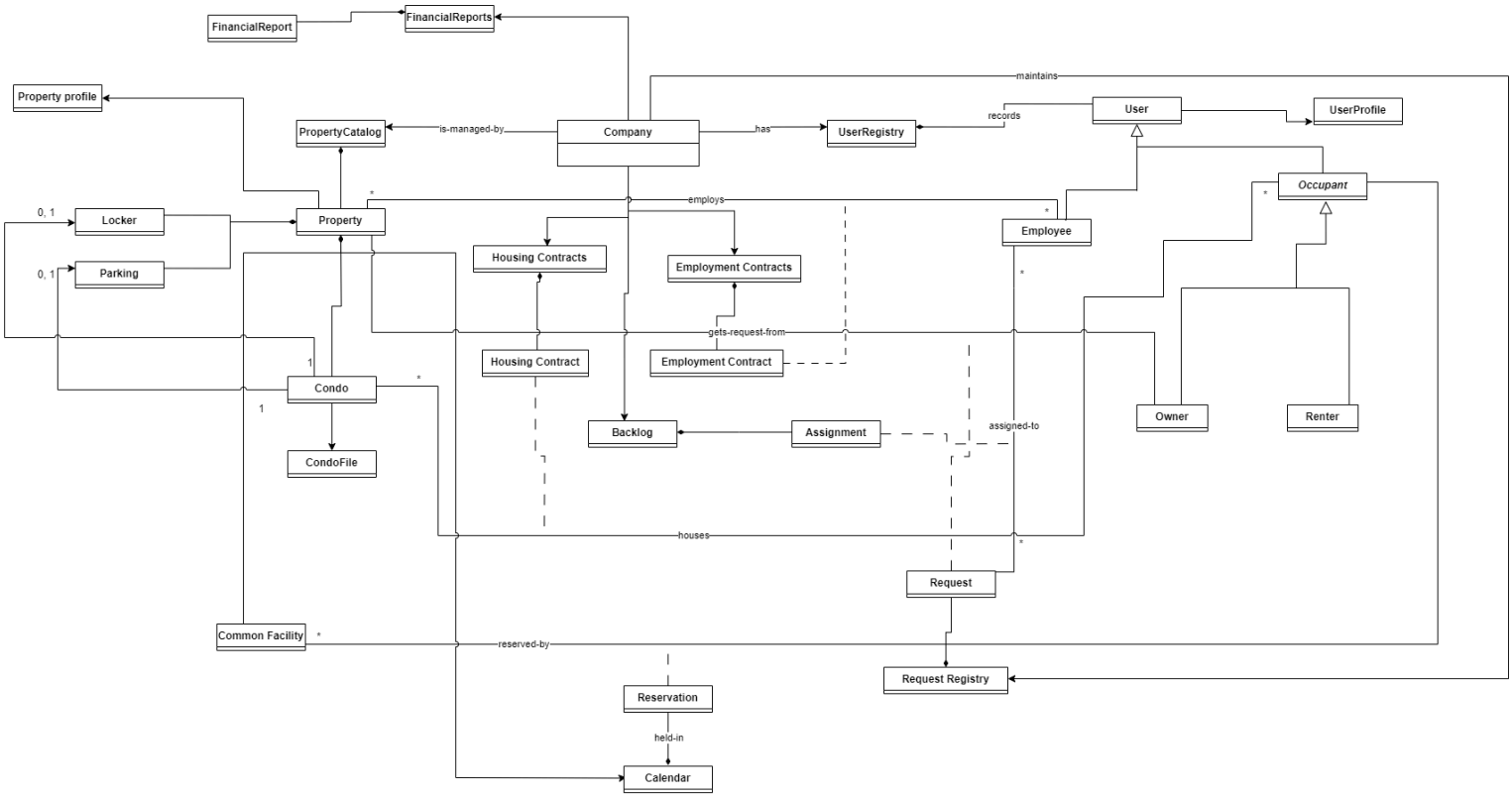


Figure 1: Domain model

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

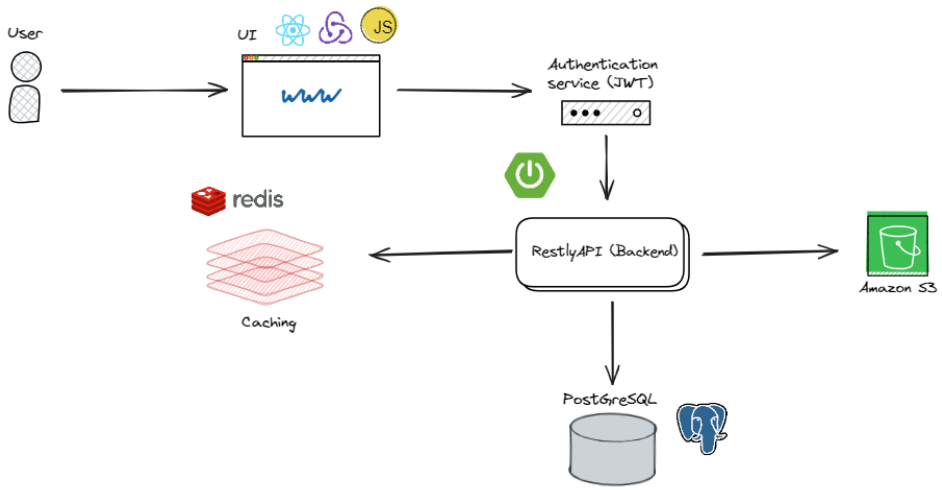


Figure 2: Component and architecture diagram

4.1.3 Use Case diagram

Governing model kind: Use case diagrams

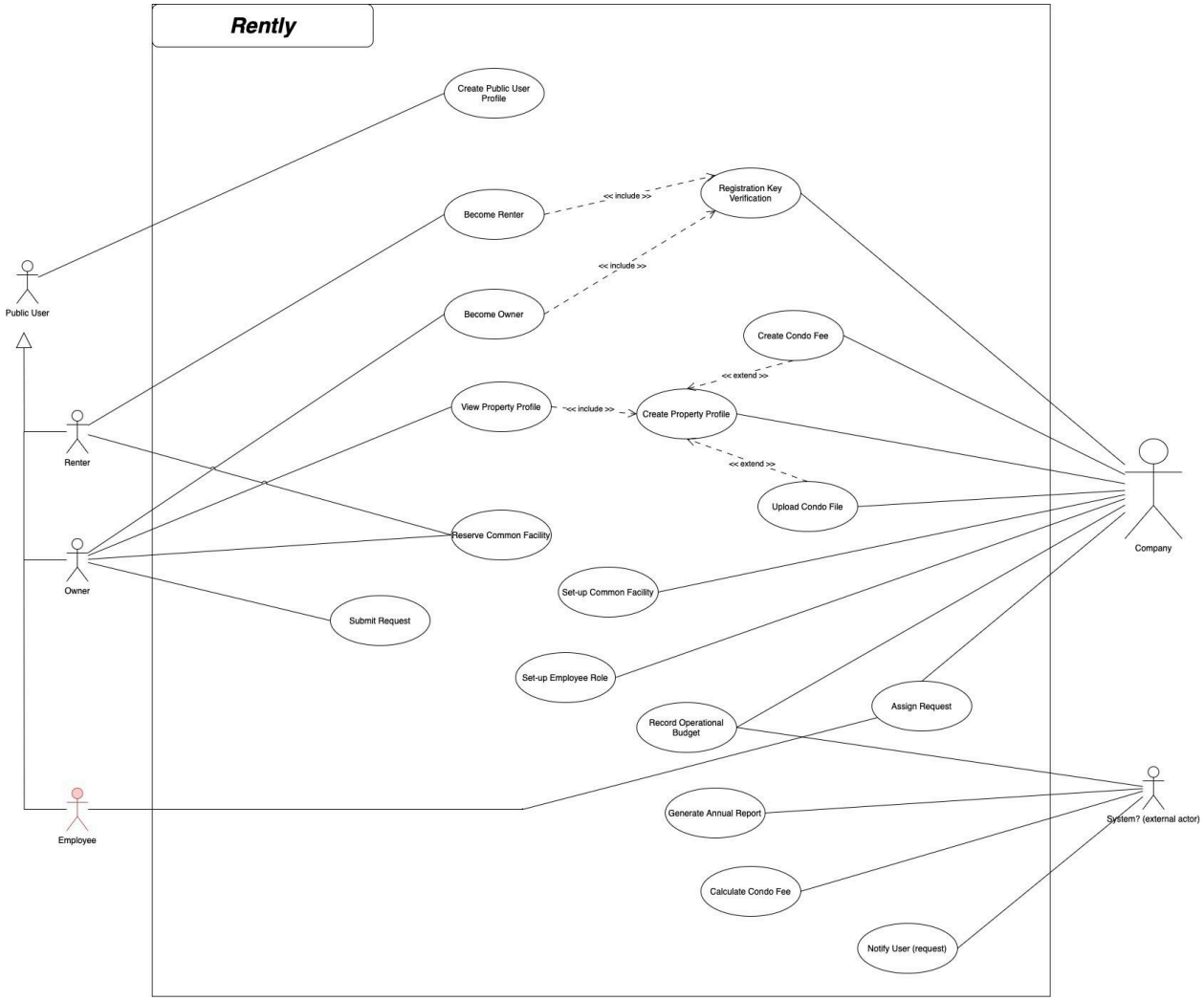


Figure 3: Use case diagram

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

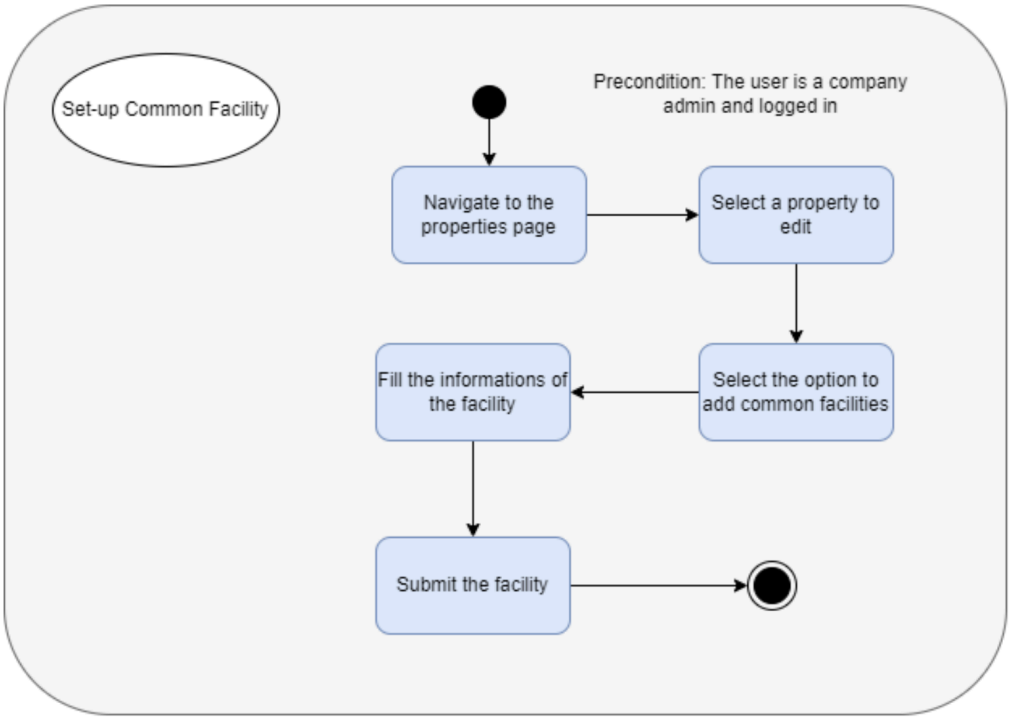


Figure 4: Activity diagram for setting up common facilities

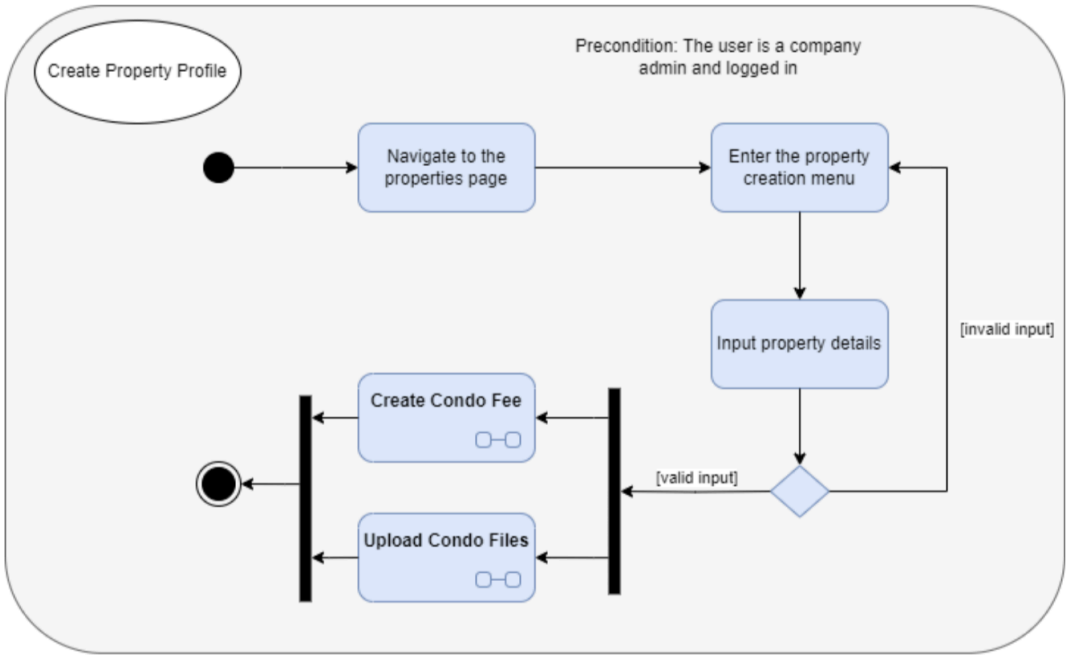


Figure 5: Activity diagram for creating a property profile

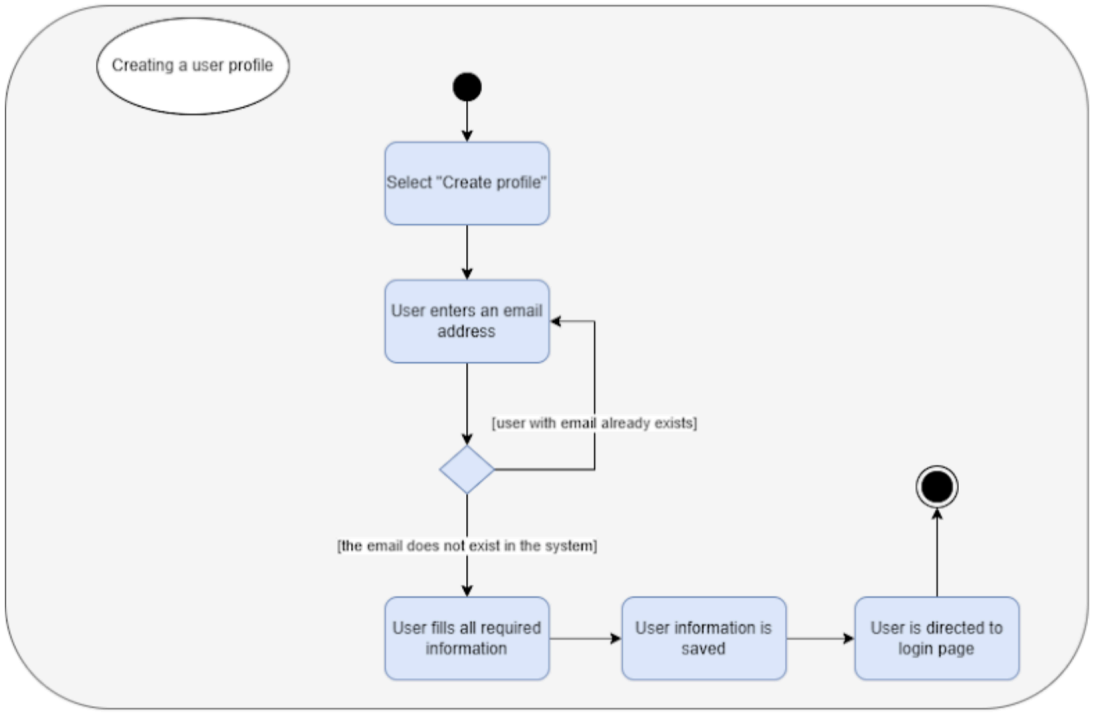


Figure 6: Activity diagram for creating a user profile

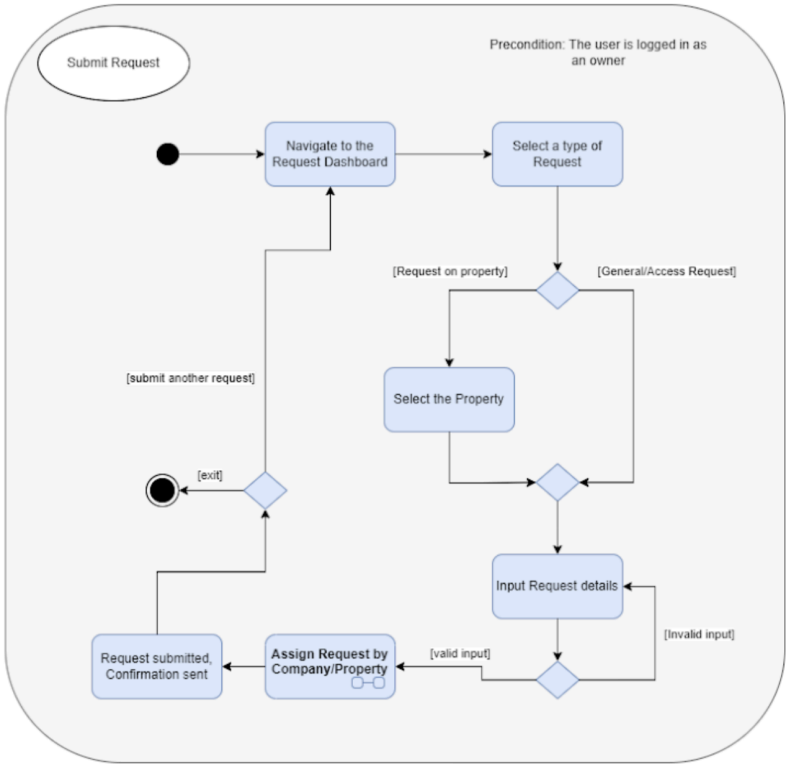


Figure 7: Activity diagram for submitting a request

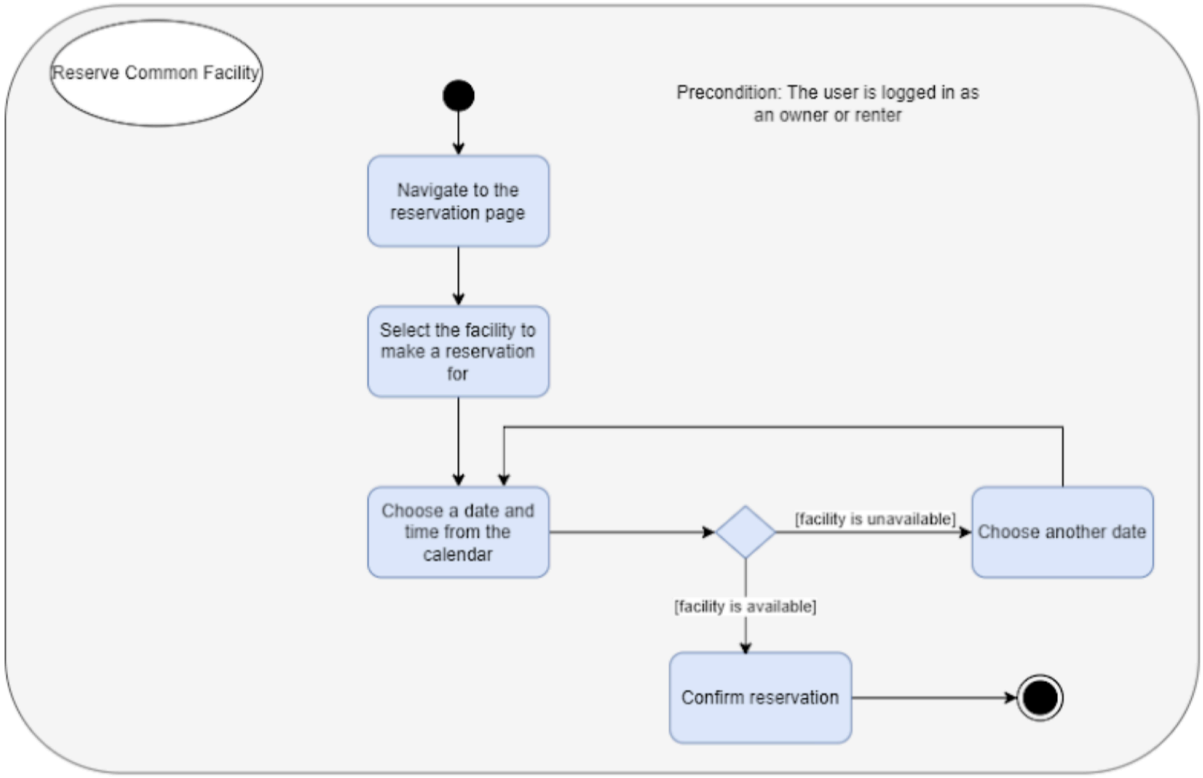


Figure 8: Activity diagram for reserving common facilities

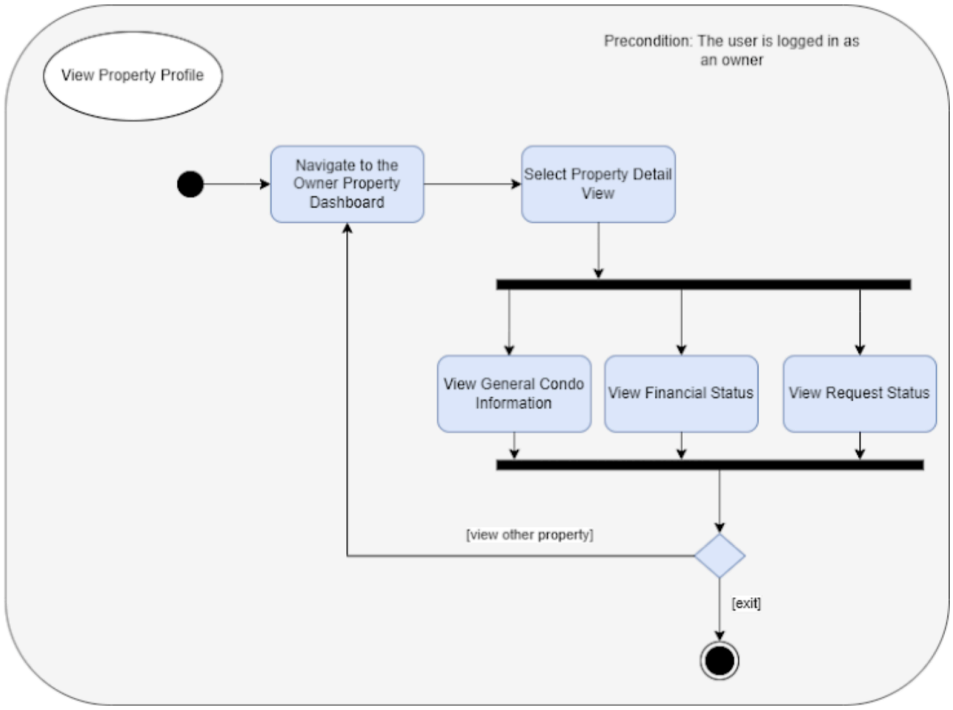


Figure 9: Activity diagram for viewing property profile

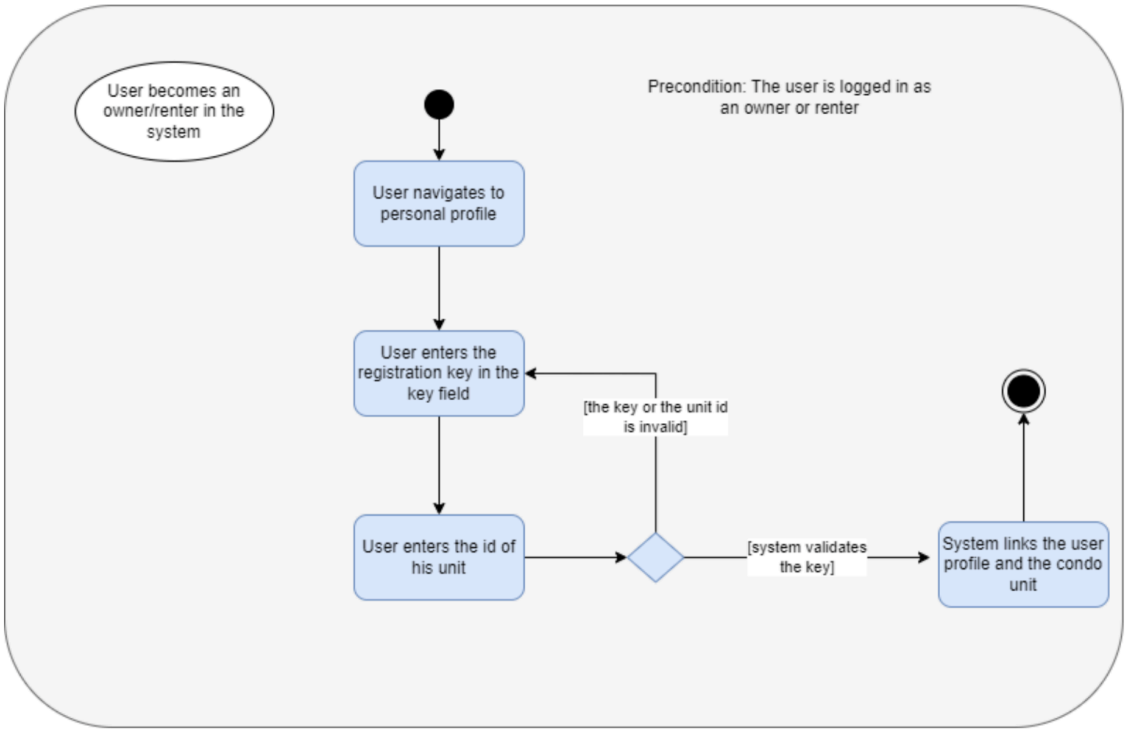


Figure 10: Activity diagram for when a public user becomes an occupant in the system

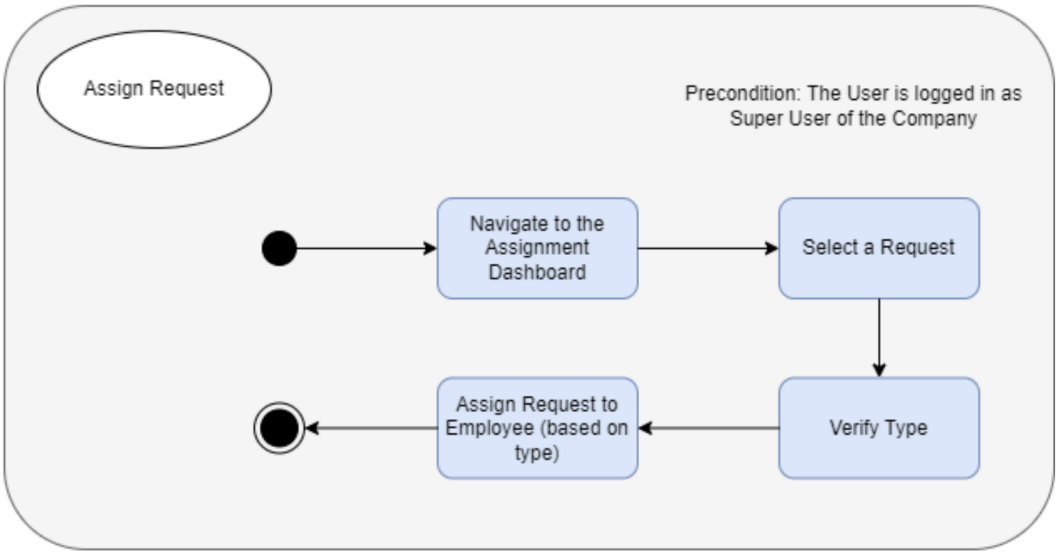


Figure 11: Activity diagram for assigning a request to an employee

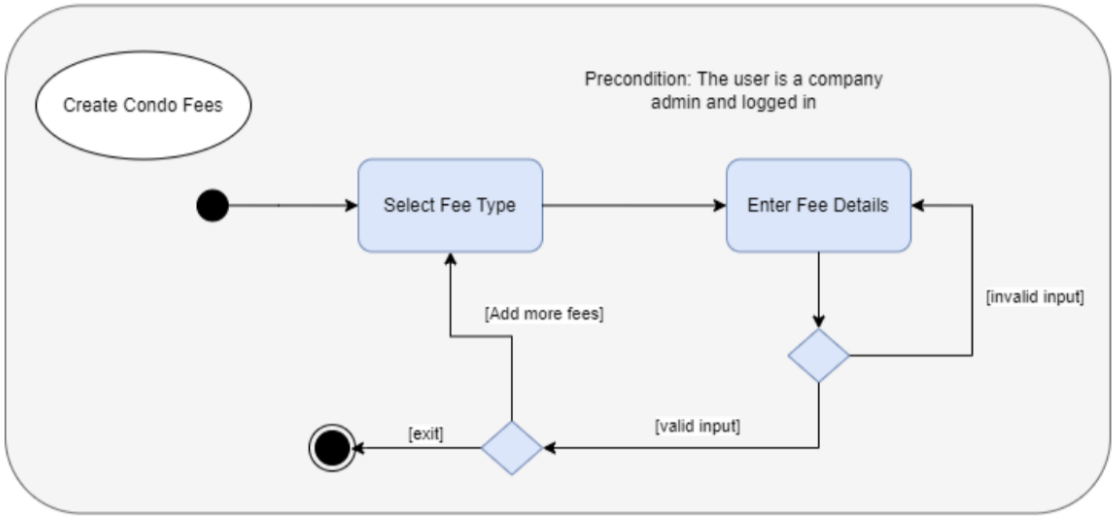


Figure 12: Activity diagram for creating condo fees

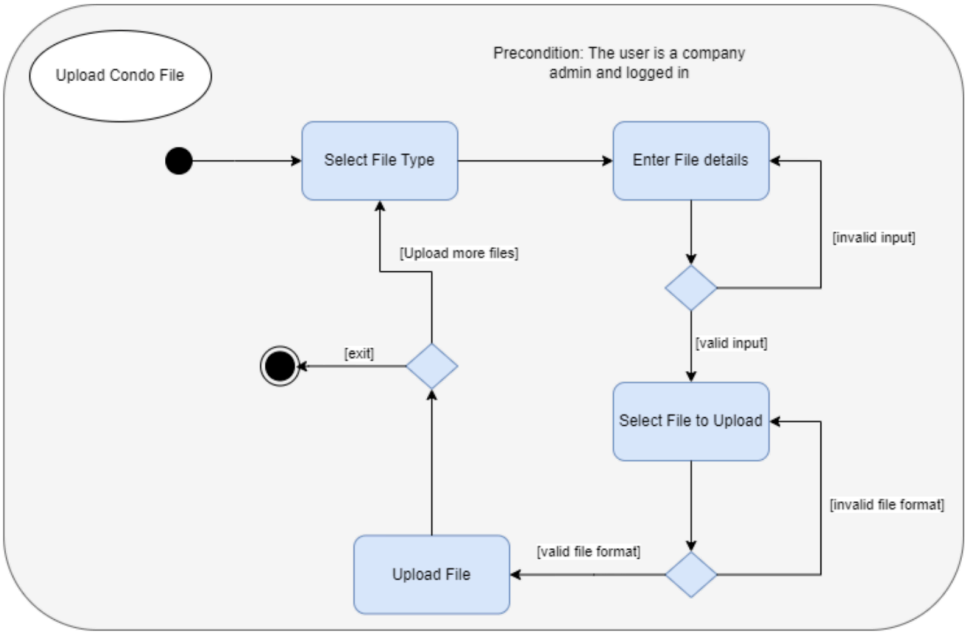


Figure 13: Activity diagram for uploading condo files

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

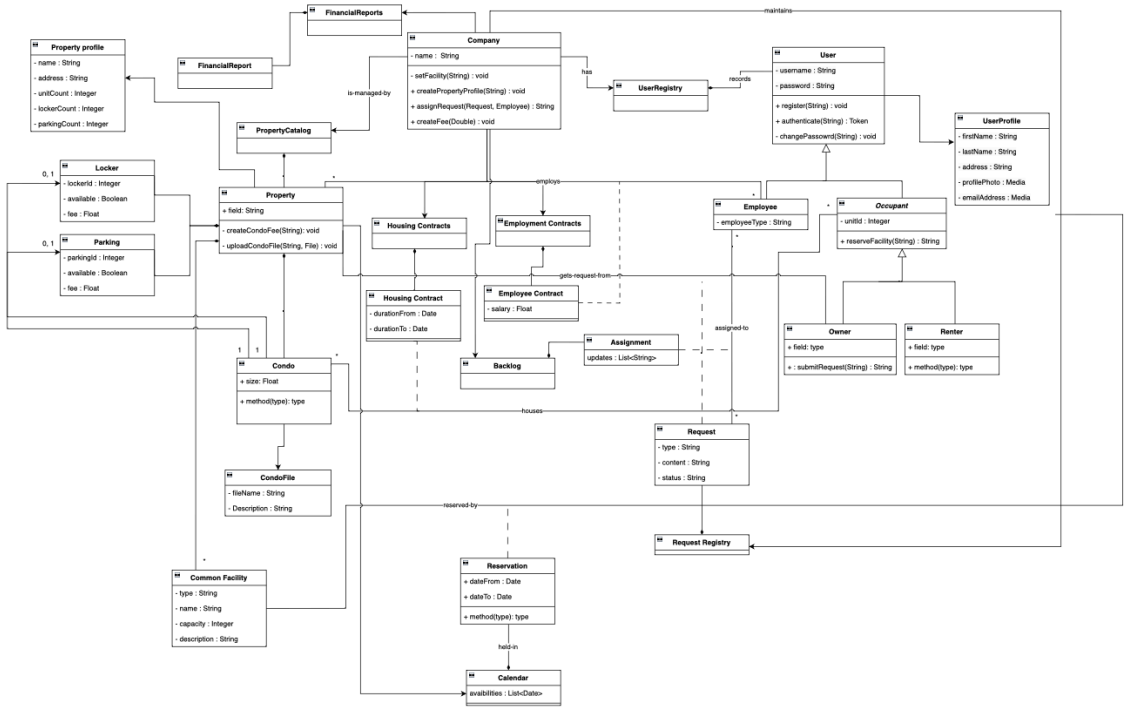


Figure 14: Class diagram

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

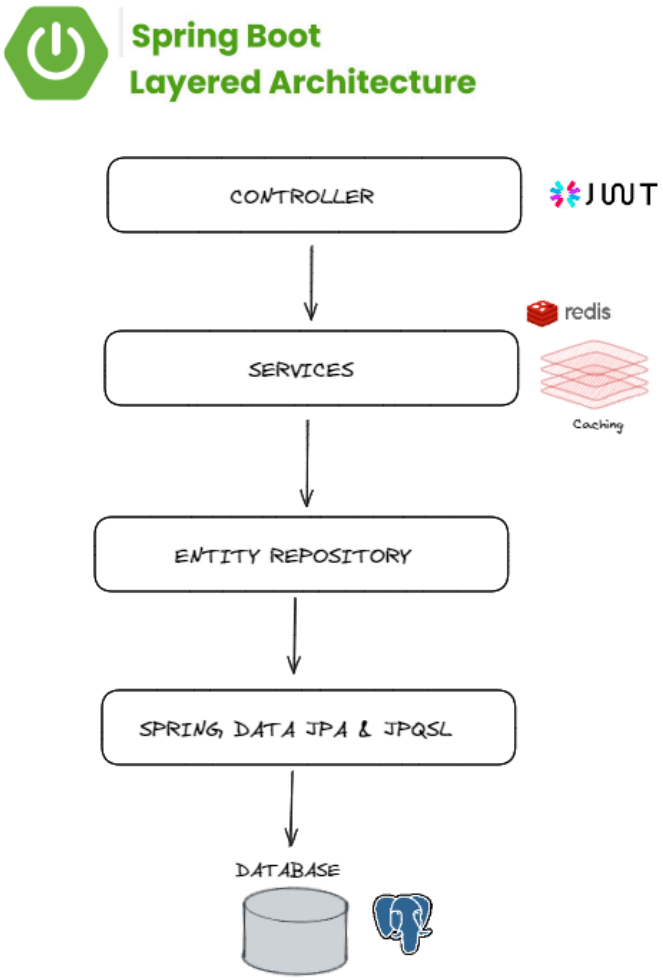


Figure 15: Backend architecture diagram

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

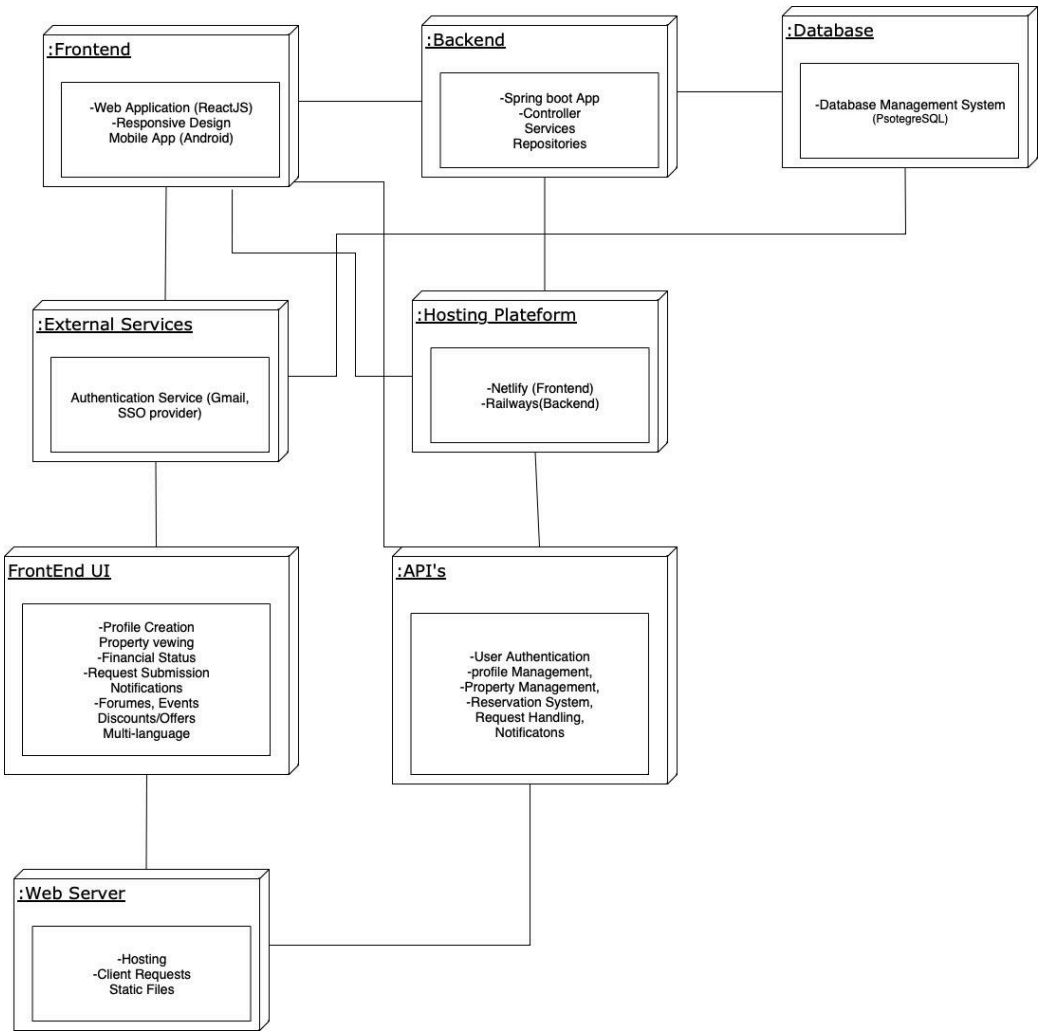


Figure 16: Deployment diagram

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 Scoring

Risk analysis and scoring 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.

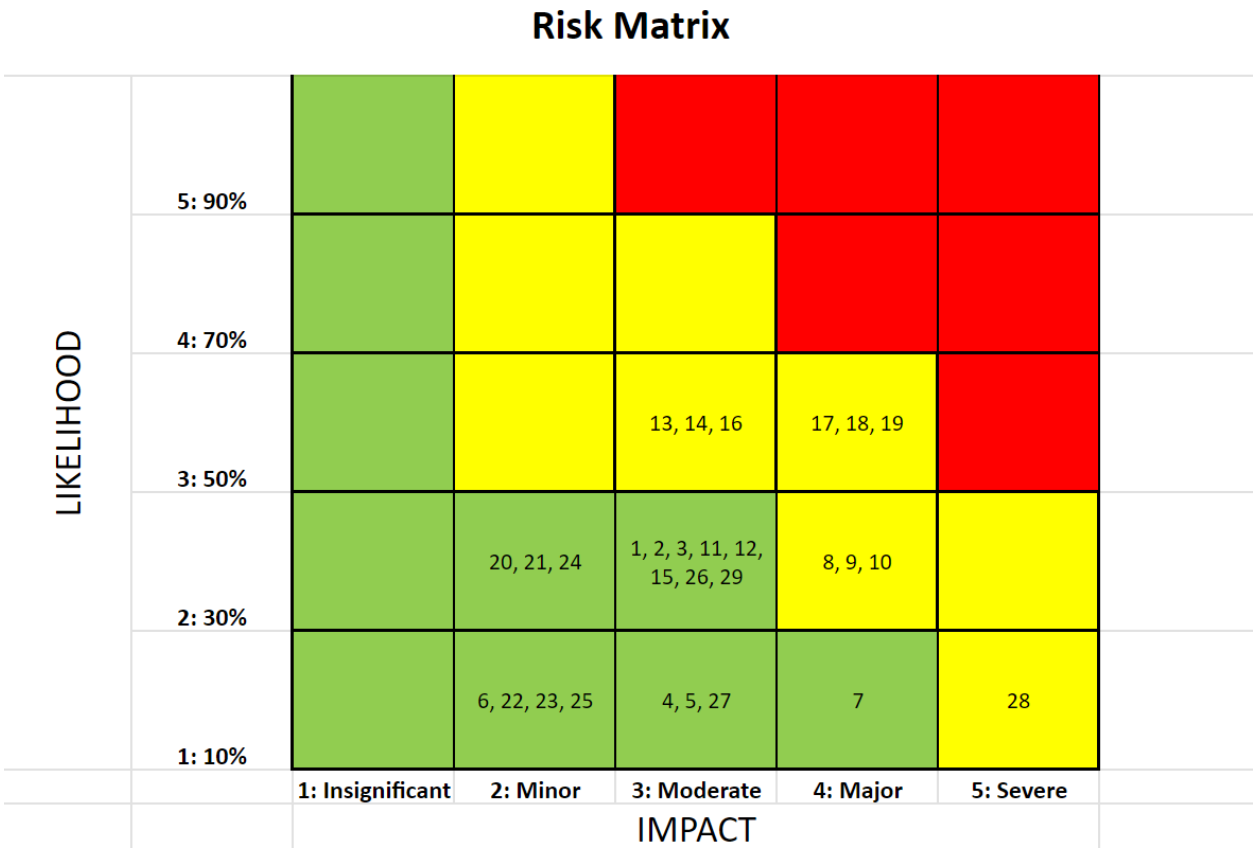


Figure 2: Rently Risk Matrix, Sprint 5 risks included with risk IDs

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 5 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
26	**Management** Future Sprints affected due to student workload	3	2	Medium	4/11/2024 (1,2,3,4,5)	Mitigate	Team members remain in contact, and can move tasks around to better accommodate other academic responsibilities
27	**Technical** Site not dynamic	3	1	Low	4/11/2024 (1)	Avoid	Tailwind CSS supports dynamic sites, allowing for the system to be viewed well on all types of devices
28	**Management** Repository lost	5	1	Medium	4/11/2024 (1,2,3,4,5)	Accept	**Transfer** as well, GitHub has a low likelihood of losing information. The team must accept the risks

							associated with this
29	**Technical** Backend Endpoints not developed for frontend	3	2	Medium	4/11/2024 (2,3,4)	Mitigate	When a frontend developer realizes they need specific data, they let the backend team know that they need that information built into and endpoint to be accessed

6.0 Changes

The fifth sprint of the project is where the team deploys the system to be hosted externally. This means that most of the features are already implemented and that the only work remaining is testing for reliability on the hosted site.
Due to this, it was determined that the majority of the risks had already been discovered and analyzed. Half (2 of 4) of the discovered risks in this sprint were deemed to be management/programmatic risks, with the other half being technical risks. Below is a quick list of the added risks for this sprint

- 26: Future Sprints affected due to student workload **Management**
- 27: Site not dynamic **Technical**
- 28: Repository lost **Management**
- 29: Backend Endpoints not developed for frontend **Technical**

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.

JUnit was chosen as the primary testing framework for the backend because of its support of Java systems and its robustness and large options for testing Java objects. Also, as mentioned above, it is often used in the industry for testing. Using JUnit has also been taught (in another course) to many of the team members previously or concurrently with this project.

Using JUnit, we use the three A's testing approach, Arrange, Act, and Assert. When this type of approach is used, it becomes clear what is being tested and what the expected outcomes should be. Below, in Figure 19 below, we show an example test case using this structure.

```java
@Test
public void testFromEntity() {
    // Arrange
    when(mockedCondo.getUser()).thenReturn(mockedUser);
    when(mockedCondo.getUser()).thenReturn(mockedUser);
    when(mockedUser.getId()).thenReturn( t: 1);
    when(mockedCondo.getBuilding()).thenReturn(mockedBuilding);

    // Act
    CondoDto testCondoDto = CondoDto.fromEntity(mockedCondo);

    // Assert
    assertEquals(mockedCondo.getId(), testCondoDto.getId());
    assertEquals(mockedCondo.getName(), testCondoDto.getName());
    assertEquals(mockedCondo.getAddress(), testCondoDto.getAddress());
    assertEquals(mockedCondo.getCondoNumber(), testCondoDto.getCondoNumber());
    assertEquals(mockedCondo.getCondoType(), testCondoDto.getCondoType());
    assertEquals(mockedCondo.getDescription(), testCondoDto.getDescription());
    assertEquals(mockedCondo.getStatus(), testCondoDto.getStatus());
    assertEquals(mockedCondo.getUser().getId(), testCondoDto.getUserId());
    assertEquals(mockedCondo.getBuilding().getId(), testCondoDto.getBuildingId());
}
```

Figure 19: Example test case, showcasing Arrange, Act and Assert sections of the unit test

Current scope: all classes

Overall Coverage Summary

Package	Class, %
all classes	82.7% (134/162)

Coverage Breakdown

Package	Class, %
com.rently.rentlyAPI	0% (0/1)
com.rently.rentlyAPI.auth.controller	100% (1/1)
com.rently.rentlyAPI.auth.dto	62.5% (5/8)
com.rently.rentlyAPI.dto	88.7% (47/53)
com.rently.rentlyAPI.entity	93.8% (45/48)
com.rently.rentlyAPI.entity.enums	100% (4/4)
com.rently.rentlyAPI.entity.user	100% (22/22)
com.rently.rentlyAPI.exceptions	100% (4/4)
com.rently.rentlyAPI.handlers	100% (3/3)
com.rently.rentlyAPI.services.impl	16.7% (3/18)

Figure 20: Run coverage of JUnit tests

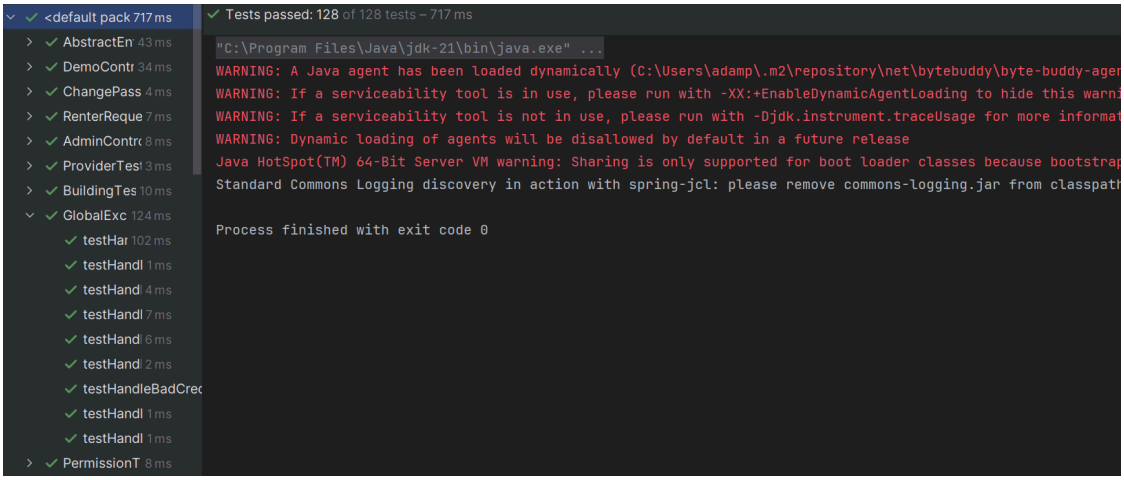


Figure 21: Run JUnit tests in IntelliJ, showcasing 128 of 128 tests passing

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.

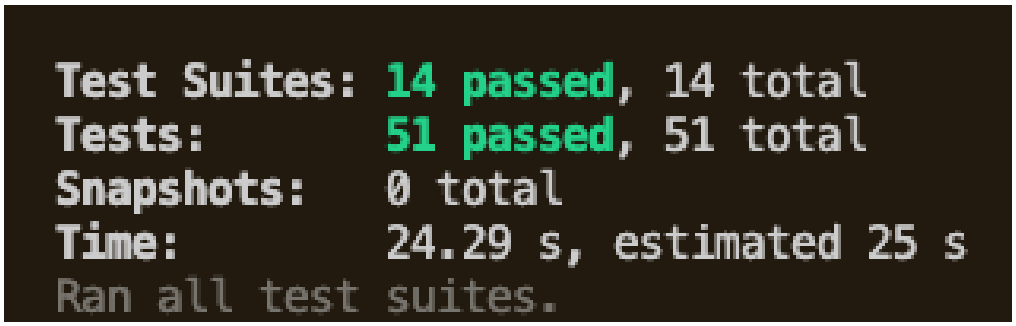


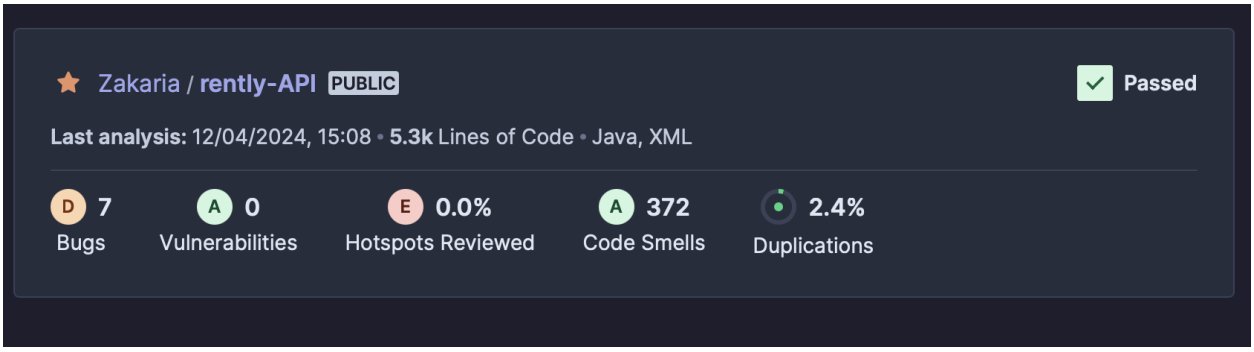
Figure 22: Run of Jest tests developed

Figure 23: Example of test built using Jest

Figure 24: Another example of test built using Jest

SonarCloud

SonarCloud was integrated in sprint 4 to work with the backend to provide static code analysis of the system as it changes. This provides active feedback on development efforts and ensures that each pull request made does not cause build issues in the system. This can also provide acceptance test standards. The scan is done manually when a specific maven command is run. It is done manually so that there is no need to do it in a pipeline and build the entire project as it increases the complexity of the task as well as increasing the time before getting results.



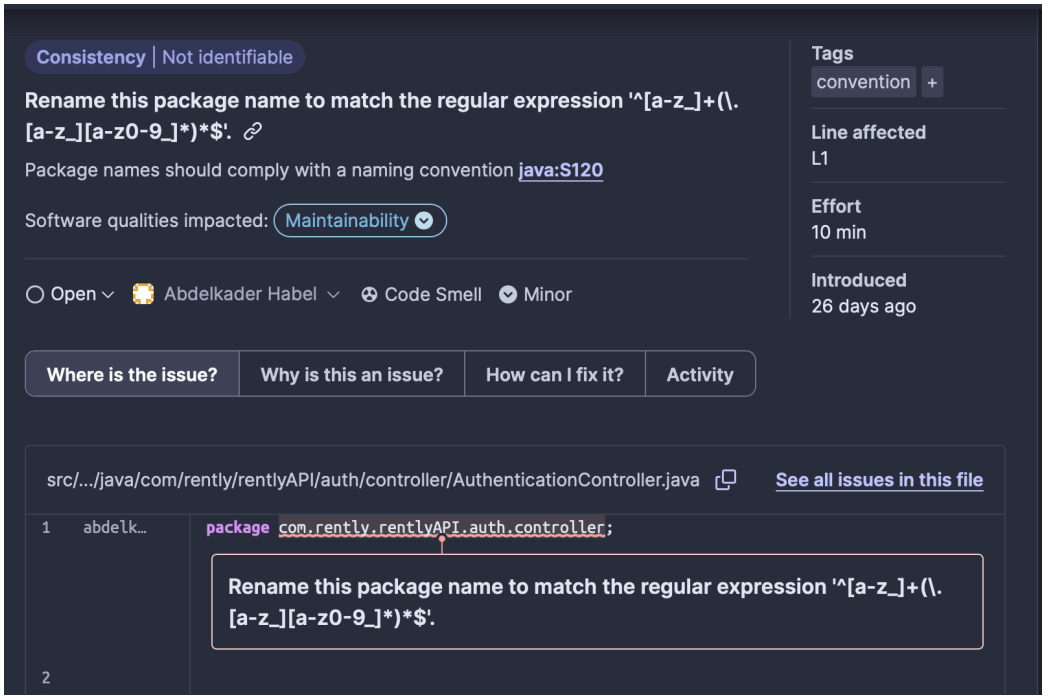
The above image shows the results of a scan. It shows bugs as well as code smells. When diving deeper into the bugs, we notice that there are limitations to what Sonar can do in terms of understanding the code. For example, see the image below:

```
        if (renterService.findByEmail(email).isPresent()) {
            return renterService.findByEmail(email).get();
```

Call "Optional#isPresent()" or "!Optional#isEmpty()" before accessing the value.

```
        }
        throw new AuthenticationException("User with email " + email + " not found");
    }
```

The bug says that we need to call isPresent() or isEmpty() on an Optional<> object before calling get(). The problem with the "bug" is it does not see the check is being done in the if statement when checking if there is a renter with the given email. Therefore, this bug is not in fact a bug, and thus we choose to ignore it. Similarly for the code smells, a lot of them are misidentifications. Indeed, it is not aware of Spring Boot naming conventions so there are many times when it identifies a smell that shouldn't be there, such as in the below image:



The name of the package is correct, but Sonar identifies it as wrong.

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

Rently Sprint 5 Retrospective

Introduction

This postmortem is written in a context where the fifth and final sprint is completed, with features functional. This sprint was generally positive.

What went wrong

1 - API Endpoints that had not been updated in the frontend
One problem we faced was that the website was not fully working when we released it in sprint 4. We only realized that after the end of the sprint and had to look for what it was. More precisely, when registering and logging in, the system was showing error messages even though the given credentials were correct. We looked at the network section of the develop feature on our browser and saw that we were calling a specific endpoint. Then when we looked at the frontend code, we realized we had not updated the endpoint at that level. Changing that fixed the problem, but it took a lot of time to pinpoint.

2 - We were a little late on the frontend development
We realized we had a few things to change and fix in the frontend, a task that was more challenging and required more work than needed in the backend. It was especially bothersome because it was at the end of the project and there was no possibility to move anything to a future sprint. It was not an easy task but we still got it in time.

What went right

1 - Backend development
The work that needed to be done in the backend was minimal, so we did not have to worry much about code there. Most of the work had been done in previous sprints in a fast manner thanks to our approach to the development. We were able to focus on other tasks while the frontend was being completed.

2 - Team collaboration
Given this is the end of the project and it happened during final exams, the collaboration between members needed to be very good, and it was. We held a meeting where we made clear the responsibilities of everyone as well as where we discussed our progress. We fixed a few problems during that meeting and any confusion anyone had was cleared.

3 - Deployment ease

Deploying the project was very easy and straightforward. In sprint 4, we deployed the website on railways and netlify. The link to the website we had was a randomly generated one and we needed to change it to something recognizable and related to our platform. We thought it would be complicated to rename it and would probably require us to redeploy and play around a lot in the settings, but none of that happened, we just needed to update a field in netlify settings.

Conclusion

In conclusion, Sprint 5 was overall positive. We faced some problems related to endpoints and status of the frontend code, but it was resolved well enough thanks to the good collaboration between everyone. Most delays were in the frontend because of the supplemental amount of work there is to do compared to the backend. However we also had a few good points in this sprint, like the backend development, which was fast and easy. The collaboration and communication between team members was also very good and helped us get the project through completion. Lastly, the ease of deployment was a very good advantage to us. It confirmed to us that we chose the right services to host our backend and frontend, as opposed to other services that would have required much more configurations.

Code Management

1. Quality of source code reviews

To ensure high standards in our codebase and foster a collaborative development environment, our team adheres to a rigorous source code review process:

- **Reviewers:** Each pull request must be reviewed by at least two peers who are not the author. This ensures diverse perspectives and reduces the likelihood of bias or oversight.
- **Teams:** The development team is divided into two groups—Frontend and Backend. This specialization allows for more focused and effective reviews.
- **Weekly Meetings:** Both teams hold weekly meetings to discuss the quality of the source code. These meetings serve as a platform for:
- **Code Smell Identification:** Detect and plan the resolution of any code smells that could degrade code quality over time.
- **Cohesion and Complexity:** Review and improve code cohesion and minimize complexity, making the system easier to maintain and extend.
- **Documentation:** Emphasize the importance of comments and documentation to ensure that the code is understandable and maintainable by anyone in the team, not just the original author.
- **Approval Process:** A pull request can only be merged after receiving approval from at least two reviewers. This rule is strictly enforced to maintain code quality and ensure adherence to project standards.
- **Continuous Improvement:** Feedback from code reviews is used not only for immediate corrections but also to continuously refine our coding standards and review processes.

This structured approach to source code reviews helps prevent bugs, improves the understanding of the codebase across the team, and fosters a culture of collective code ownership and continuous improvement.

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

Our team maintains high standards of code quality by adhering to a well-documented set of coding conventions. These conventions cover various aspects of coding and are tailored to our specific technology stack. Key areas include:

- **Naming Conventions:** We use camelCase for naming variables and functions, and PascalCase for classes. This consistency is crucial for readability and maintainability.
- **File and Folder Organization:** Files and folders are named and structured systematically according to their functionality and scope, which helps in navigating the codebase efficiently.
- **Programming Practices:** We follow best practices such as DRY (Don't Repeat Yourself) and SOLID principles to enhance code modularity and reduce redundancy.

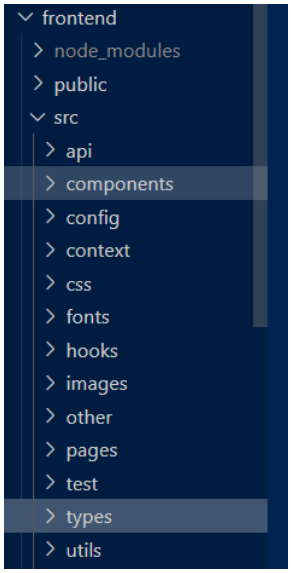
Examples from Our Codebase:

1. **Naming Convention:**

```
const axiosPrivate = useAxiosPrivate();
const navigate = useNavigate();


const handleProfileUpdate = async (e: { preventDefault: () => void; }) => {
    e.preventDefault();
```

2. **File Organization:**



Ensuring Compliance:

- **Tool-Assisted Audits:** We regularly use linters and formatters like ESLint and Prettier integrated into our development environment to automatically ensure that the code adheres to our conventions.
- **Leadership Oversight:** The heads of the front-end and back-end teams periodically review the code to ensure that all conventions are being followed. This oversight helps maintain discipline and catch any deviations early.
- **Peer Reviews:** During code reviews, reviewers specifically check for adherence to these coding conventions along with other quality parameters.

By strictly following these conventions and continuously monitoring our adherence through both automated tools and manual reviews, we ensure our codebase remains clean, organized, and easy for any team member to understand and work with.

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.

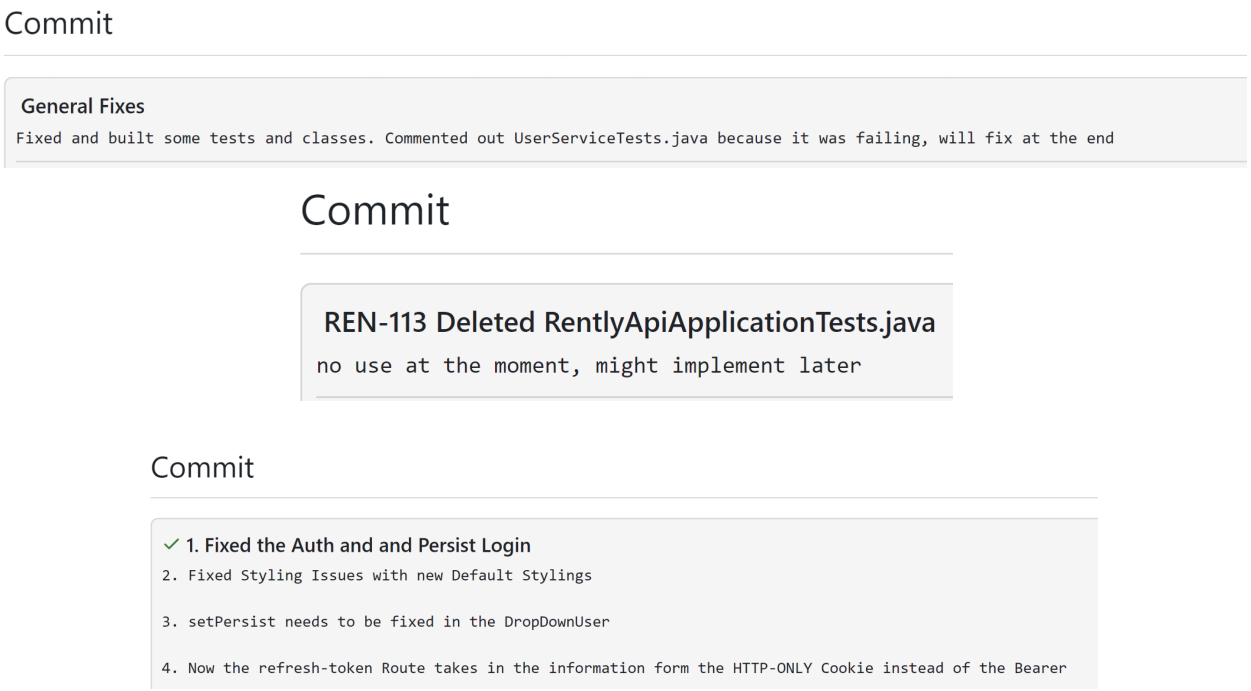
5. Quality of source code documentation

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

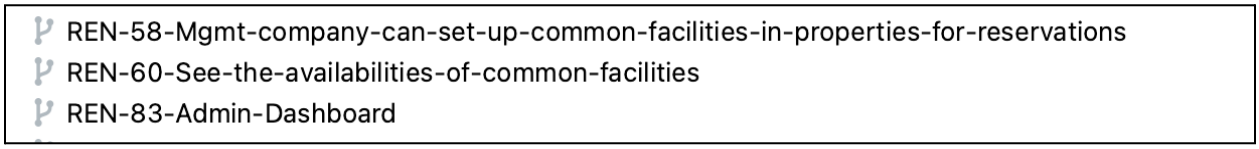
We enforce a standard for detailed and informative commit messages, comprising a succinct title

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.



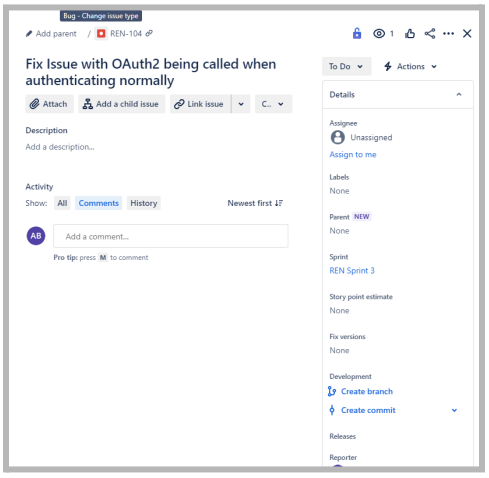
It can be seen in the image that separate branches are used for different features. For instance, the handling of the admin dashboard is handled on a branch separate from the one that will allow management companies to set up common facilities.

9. Atomic commits

The next thing we made sure to take care of is the Atomicity of the commits. In other words, we make sure that every change, no matter how minor, is recorded in a manner that precisely captures the developer's interest at that moment. For example, in our project, if we're fixing a login issue, instead of combining this fix with other unrelated improvements, we make a dedicated commit just for the fix. This commit will clearly state what the problem was and how it was solved, making it easy for anyone to understand the change without having to go over unrelated code. An example from our codebase is as you can see in the image on the left side, where we tackled a specific bug in ticket REN-127 with a commit labeled "REN-127 minor fixes", merged into the main branch (#43) by Abdelh17 last week. The commit was as minor as forgetting a letter in the word "/Sytem". This isolated fix showcases our approach to atomic commits, making it easier for the team to track and understand changes without wading through unrelated updates. This method enhances our code's clarity and maintenance, ensuring each commit is a clear, standalone change. A link for the image can be found here.

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.

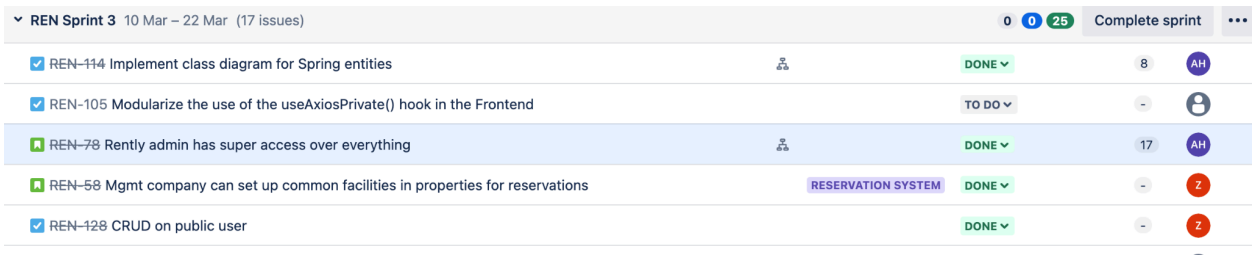
Testers will often verify the bug, and if it can be reproduced, they will mark down the steps to reproduce it. These steps will allow future validators to verify if the bug has been appropriately fixed. They will mark these

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. A link for the image can be found here.

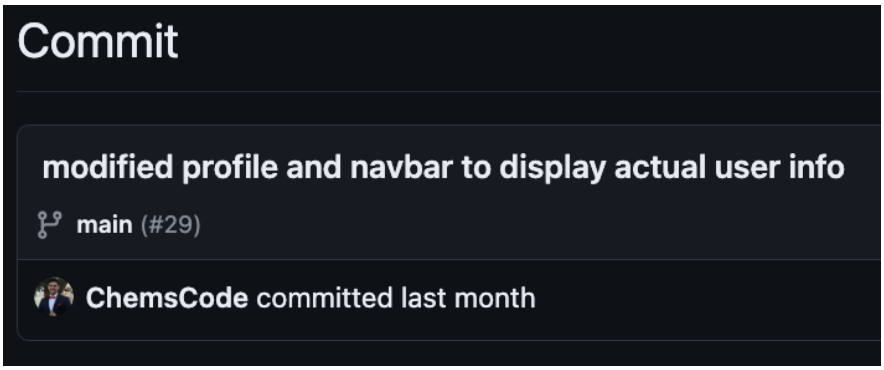
11. Use of issue labels for tracking and filtering

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

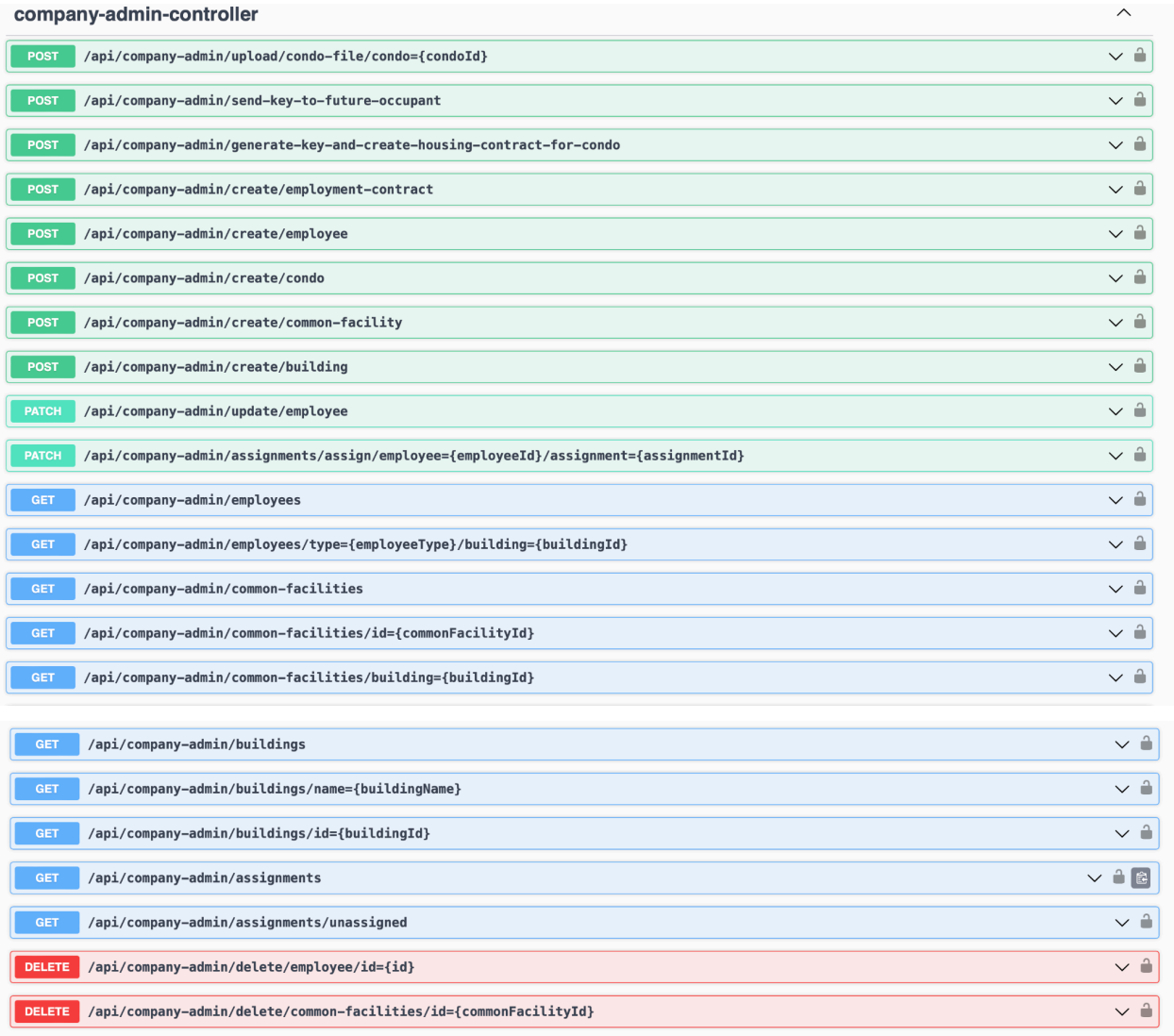
We prioritize linking commits directly to bug reports or feature requests, ensuring every code change is traceable to a specific task in our project management tools. Whether it is on Jira or Github, all of our tasks / commits have meaningful names that link to bug issues specifically. This ensures every commit clearly refers to an issue number or a unique identifier, making it straightforward to follow the change's purpose and its contribution to the project. For instance, displaying the user info to the screen, an issue might be captured in a commit titled "Fixed / Modified #29 - modified profile and navbar to display actual user info", directly tying the work to issue #29 in our tracker. An example from our workflow of the previous example mentioned is shown in the image attached with the following link to the commit, where we addressed a specific enhancement in a ticket, merged into the development branch (#29) by Chems for sprint 3. The modification was as detailed as adjusting the layout for better user experience. This approach of linking commits with their corresponding tickets simplifies navigating through our project's history, improving our code's maintainability and ensuring each commit represents a transparent, targeted change.



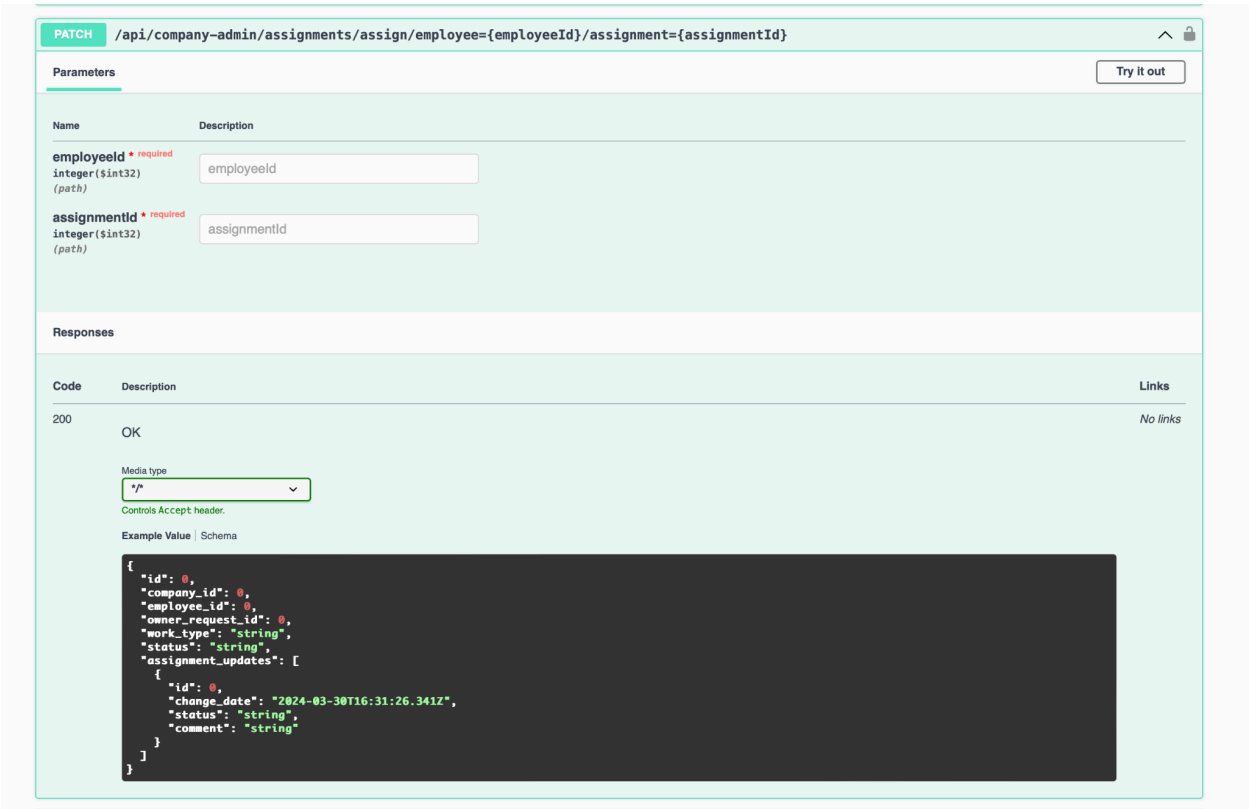
13. External tools used for project management

A. SwaggerUI

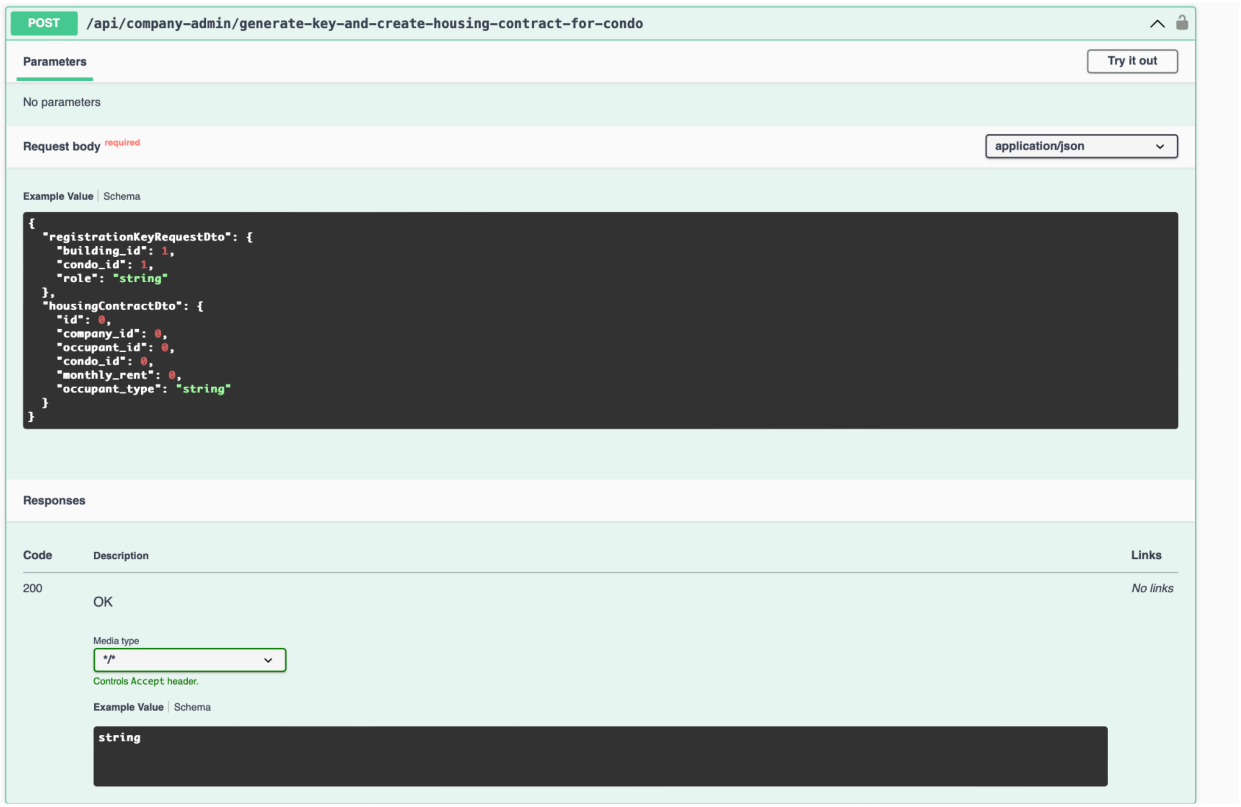
SwaggerUI is a tool that allows us to visualize all our endpoints in a single page, with all the expected inputs and sample outputs. It separates the different endpoints by type, which for us corresponds to the different controllers. We will describe our use of Swagger by focusing on the Company admin controller, but the same goes with the other endpoints.



In the above picture, we see all the various endpoints a company admin has access to.It shows which endpoint allows a GET, POST, PATCH or DELETE request. When expanding an endpoint like shown below, more details like inputs and output samples are available.

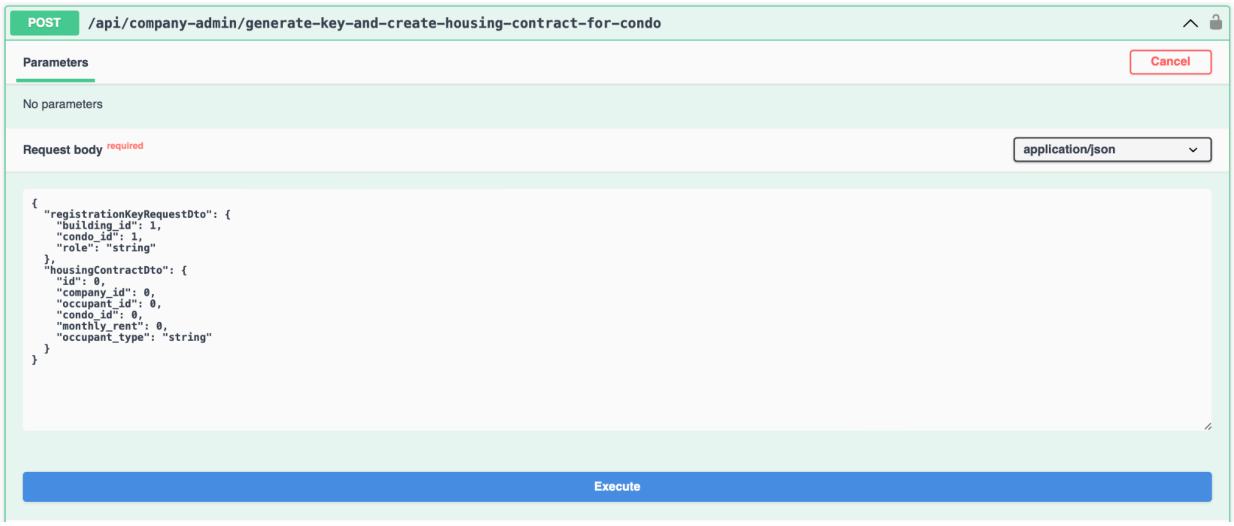


In this section, we see that the url takes an employeeId and an assignmentId, which are available in the parameters section, with the input type. Under that, the response section shows the output to expect once the request is treated. The example above deals with parameters passed in the URL, the one below will describe the scenario when a JSON body is sent to the backend.

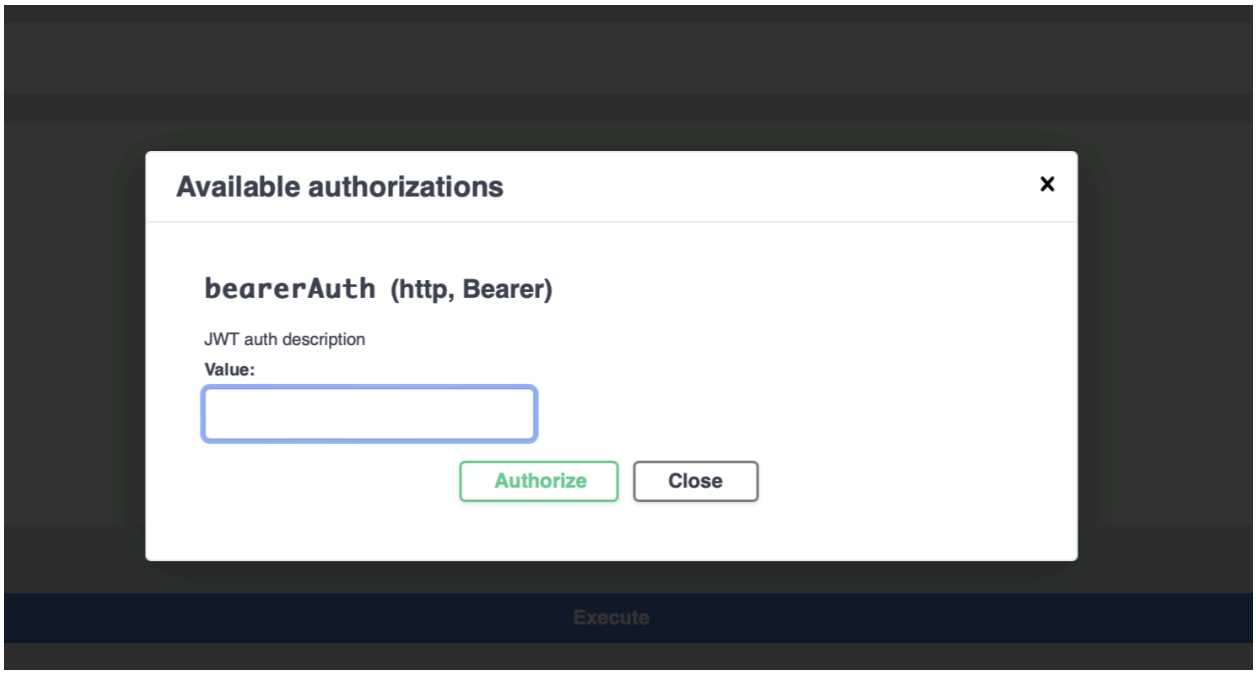


In this above case, when the company admin wants to generate a key to associate it to a housing unit for the public user to link it to his profile, the admin sends a request body that is in JSON format. This time, no parameters are sent in the URL. The request body section shows an example of what a request should look like, with the data types included too. Like above, a response sample is shown.

Swagger allows us to test these endpoints directly on the interface, without using other tools like Postman. By clicking “Try it out”, a new segment will appear and allow to write a request, as shown below. The request can be executed and a response will be returned in the UI.



Most of our endpoints are protected and require authentication, which is also something Swagger allows. By clicking on the small lock at the corner of the request, the user is prompted to input the bearer token to authenticate him.

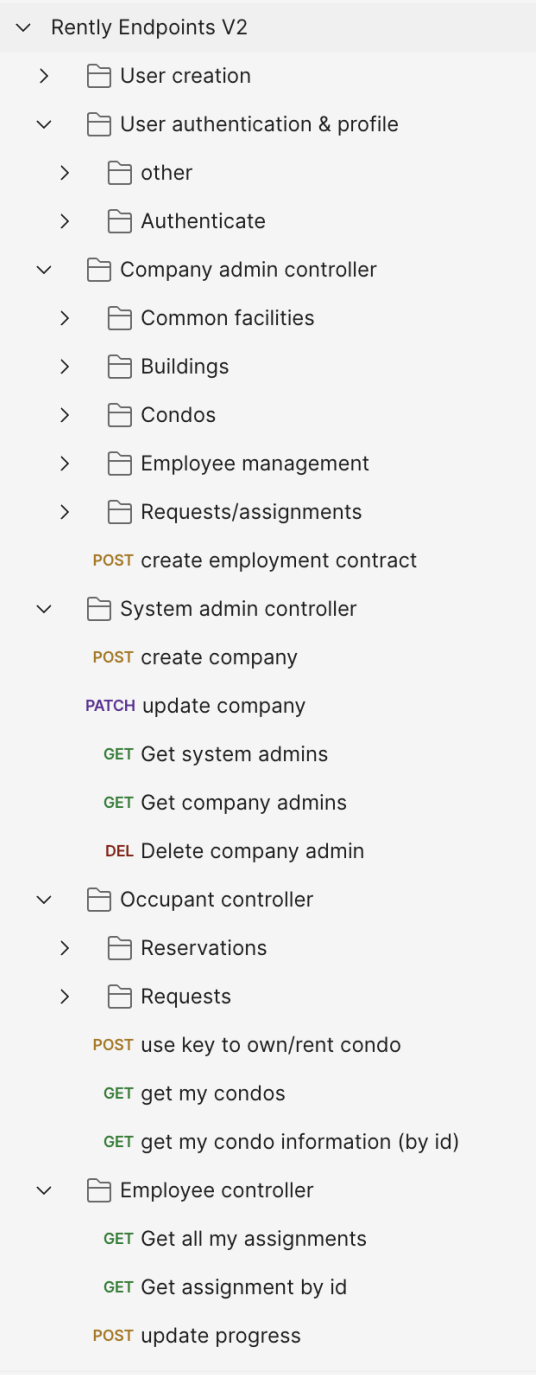


Although SwaggerUI has the ability to test our endpoints, we do not use it for that purpose. We use it as a documentation tool to see which endpoints require what inputs. It helps all the team members understand how we are structured and how to make everything work. It prevents us from making mistakes as we only need to copy the request body there and replace the dummy values with our test values in a separate software, Postman. The reason for not using SwaggerUI

to actually send requests is that it does not allow us to save the requests that have been made previously to speed up our processes, as will be explained for Postman.

B. Postman

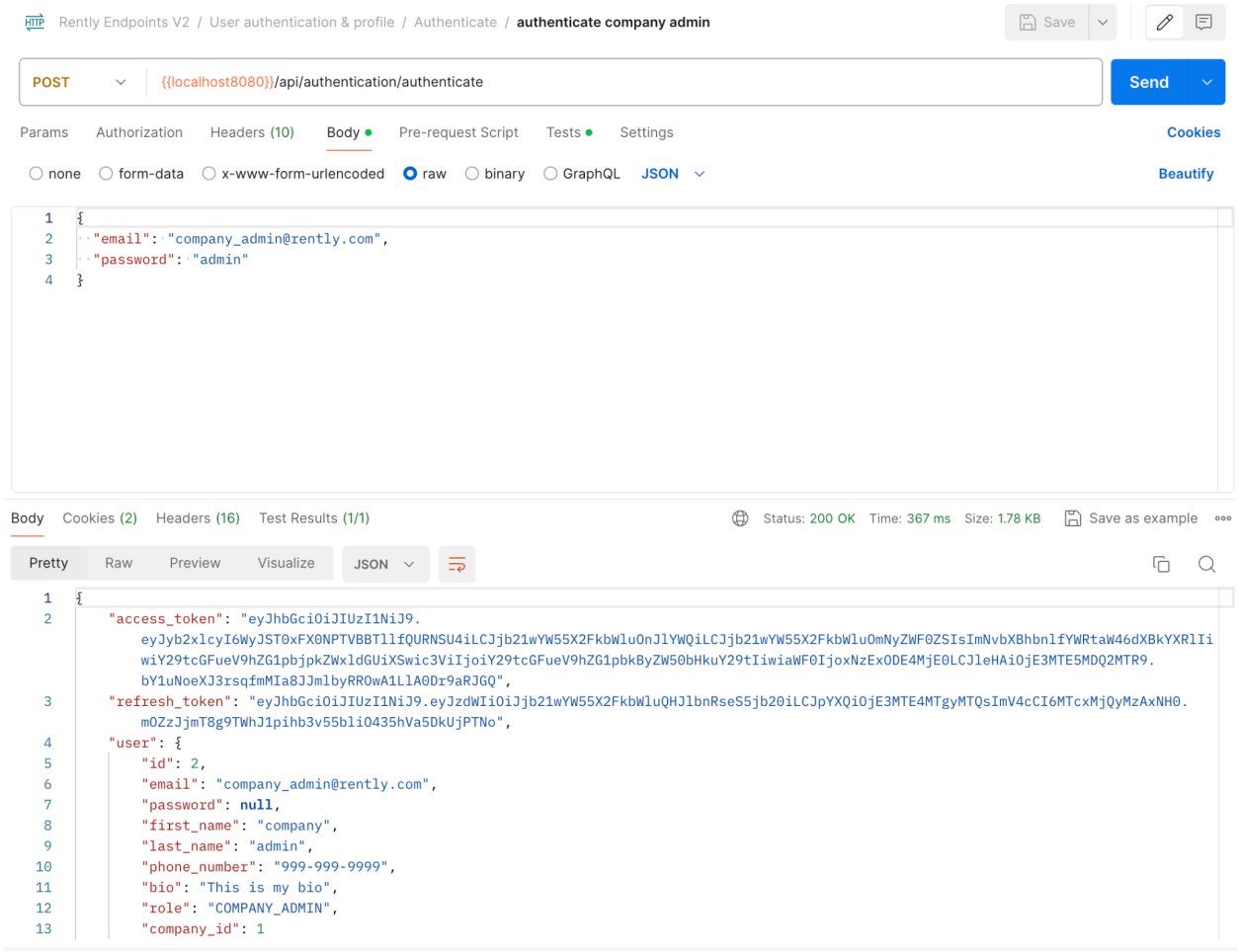
We use Postman to test our endpoints as it is a standard and well established software that is used for this type of requests. We did not want to reinvent the wheel and went with what is already well documented.



Our Postman workspace is separated into the different controllers and then into the subgroups of tasks that need to be accomplished. So for example, Anything an owner can do about requests is under the request folder in the occupant folder. When clicking on any request, we can see a JSON body with values that are filled and are not generic. That is because Postman allows us to save our requests, unlike Swagger. This means that each time we need to authenticate a user, we just need to send the request that contains the email and password, instead of each time typing them out. This considerably speeds up our work when we need to work on tasks that need several requests to be made before.

For example, for a user to create a request, there are many steps to follow. First a system admin needs to be created, then authenticated. Then the admin needs to create a company and a company admin. Then the company admin needs to create a building, a condo unit and a housing contract to generate a key. Then a public user needs to be created, authenticated and needs to submit a request to become owner with the key the company admin generated. Only at this point can we actually test whether our endpoint works or not. For this reason, all our endpoints already have some values that are pre filled such that any team member only needs to run them instead of completing all the steps mentioned, which could also lead to errors and more time wasted. SwaggerUI

would be very difficult for tasks that depend on several others before.



This above image shows what a request looks like in Postman, with the URL, the request type on the left, the JSON body and the send button. The response is also shown, with the status number (200 OK in this case). Notice there is an access token returned in this response. This is because we are at the authentication controller. This token needs to be copied and pasted in the adequate field in the authorization tab under the request url in the requests that need authentication.

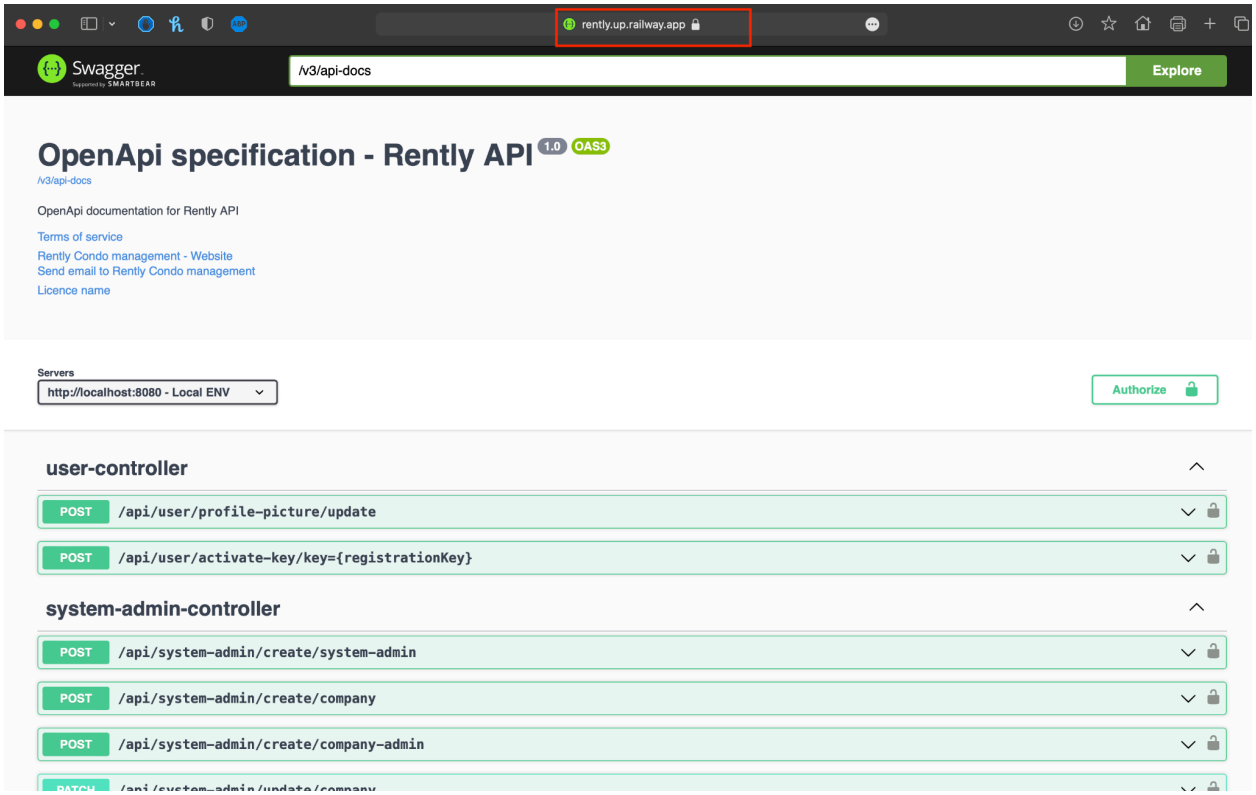
Overall, Postman is an easy and collaborative way to test our endpoints, with inputs already filled. The workspace allows multiple users to work on the same endpoints simultaneously like in a Google Doc. The workspace can also be downloaded in a file and uploaded by another user to use if the maximum number of users in the workspace has been reached.

Project deployment

Backend

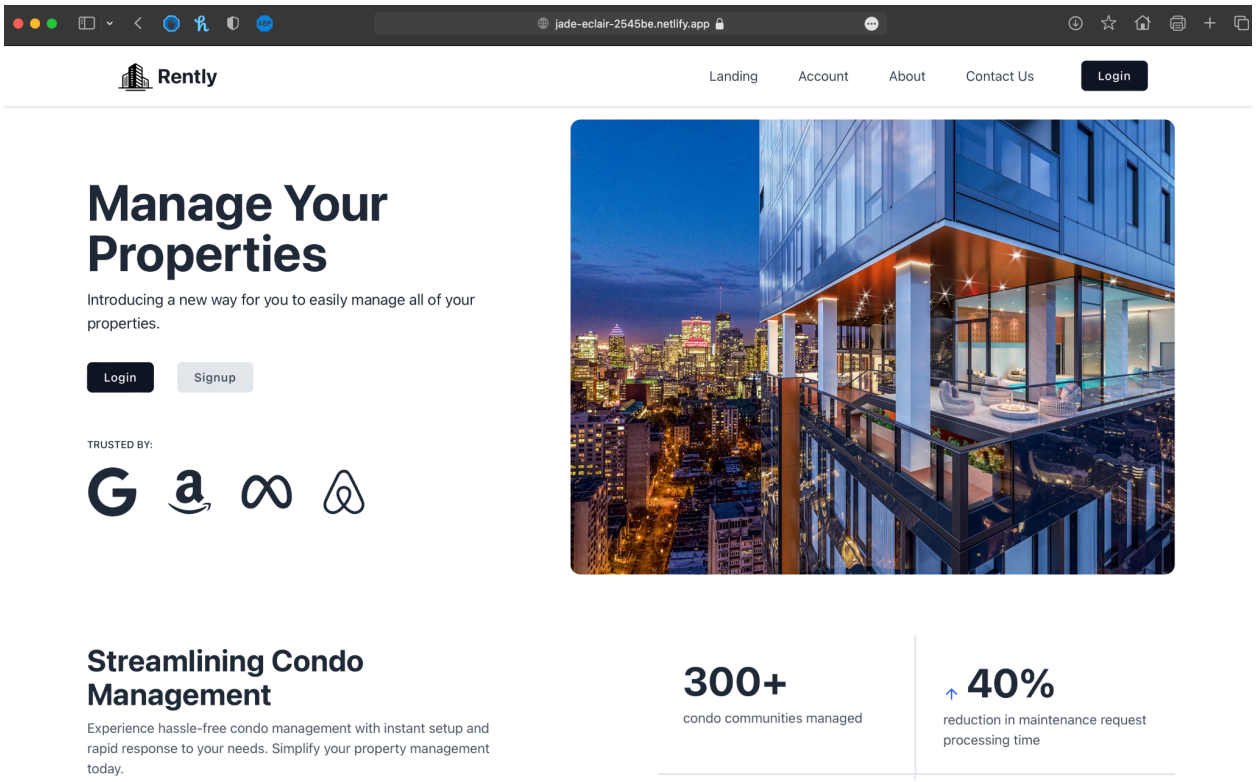
The backend was deployed on Railways, at the following address:
https://rently.up.railway.app/

To look at the API documentation, click on the following link:
https://rently.up.railway.app/swagger-ui/index.html



Frontend

The frontend is deployed at the following address.
https://rently-management.netlify.app/



Traceability matrix

Requirements	Jira ID	Pull Requests	Test Cases
Create an account and login as public user	REN-15,REN-78	#12	PublicUserDtoTest.java
Use an activation key to become an owner/renter as public user	REN-53	#25	KeyTest.java
Create a profile as owner/renter	REN-54,REN-75	#27	OwnerDtoTest.java, RenterDtoTest.java
View a dashboard of properties as an owner	REN-16,REN-55,REN-75	#16, #27	
View a dashboard of properties as an owner	REN-16,REN-55,REN-75	#16, #27	
Make reservation on a common facility in a calendar-like interface as owner/renter	REN-14,REN-59,REN-152	#13	CommonFacilityReservationDtoTest.java,CommonFacilityReservationTest.java
Make special requests to be treated by employees as an owner	REN-66	#58	
View status of my requests in a notification page as an owner	REN-67	#64	
View availabilities of common facilities as owner/renter	REN-60	#33	
Have access to the condo files my management company made available for all units in my building as owner	REN-65		
Create a company as system admin	REN-78, REN-158	#39,#40,#65	CompanyTest.java, CompanyDtoTest.java
Create company admins and link them to a company as system admin	REN-78, REN-159	#40,#66	CompanyAdminDtoTest.java

Login as a company admin	REN-78	#40	
Add,modify, and delete buildings as company admin	REN-86	#34	BuildingTest.java
Add, modify, and delete condo units for buildings as company admin	REN-91	#25	CondoTest.java
Register employees as company admin	REN-78, REN-124,REN-154	#40,#62	EmploymentContractDtoTest.java
Set up common facilities in buildings for users to reserve as company admin	REN-58	#43	CommonFacilityTest.java, CommonFacilityDtoTest.java
Enter condo fees to units as company admin	REN-135		HousingContractDtoTest.java
Enter operational costs for all operations made to know my expenses and budget as company admin	REN-62		
Have access to financial annual report for expenses and budget as company admin	REN-70		
To send a registration key to public user to link them to a condo unit as company admin	REN-78		
Add lockers and parking to a building as a company admin	REN-134		
Upload condo files for properties as company admin	REN-61, REN-146, REN-149	#50,#53	FileUploadExceptionTest.java
Set different roles for employees as company admin	REN-63, REN-144	#49	EmployeeAssignmentTest.java
View the status of all the requests owners	REN-64, REN-153	#61	

made as company admin			
Can generate a key for owner/renter as company admin	REN-157		
Create an account and login as an employee	REN-78,REN-124	#40	EmployeeDtoTest.java
View the owner requests assigned to me as an employee	REN-68	#55	
View the status of my assigned requests on a notification page as an employee	REN-69	#60	
Update assignments I am working on like changing status or adding comments as employee	REN-136	#56	

Rently Overall Project Retrospective

Introduction

This project postmortem is written in the context of the end of the development of Rently. The 5 sprints of the project are now behind us and the project is completed.

What went wrong

1 - Lack of proper code understanding from the beginning
At the beginning of the project, during the first two sprints, there was a lot of development that was done that we had to completely redo in the third sprint. That happened because some people were focused on the documentation and some people were focused on the code. The problem was the disconnect between those two mentalities. Once the documentation was advanced enough and the design was approved by the team, there was already a lot of code that was written that did not adhere to what we had decided on paper. We had to redo the work of 3 sprints in one.

2 - Immense refactoring challenge
As mentioned before, the lack of understanding of our platform caused us to perform massive refactoring activities in the third sprint. The backend had to be practically completely redone to work with the design and to be coherent. The refactoring was extremely tiring and made us lose a lot of valuable time that could've been spent developing features instead of redoing everything.

What went right

1 - Team work
Throughout the project, the team worked very well together. It was split in two, for the frontend and backend. One team lead was selected for each subteam and was in charge of managing the members that were with him. This made our meetings much easier as responsibilities were clearly defined. Meetings were held often to discuss next steps and clear up any confusion. Everyone was showing up and participating, which made the experience more enjoyable. Everyone was also working on something they felt comfortable with. From the very beginning we established the preferences of the team members on whether they want to be in the frontend or the backend. This made the development process faster as there were no steep learning curves for this project.

2 - Technologies used

We are happy with the choices we made regarding technologies. We used Spring Boot and React for the development of the backend and frontend. Those choices were made because of the amount of resources online, but also because of the familiarity of the developers with these frameworks. Beyond frameworks, we also used tools to help us with the development.

We used postman to test our endpoints, both for the people in the frontend and the backend. When the backend was being developed, the Postman suite was used to test the different actions we were doing, and when it was the turn of the frontend to be developed, the developers used postman to see the responses they were getting, as well as the format of the requests that had to be sent. The Postman suite was split into different categories so that it is easy to find the correct endpoint according to the action we want to perform. We split them into controllers, from the end user point of view. So for example, there was a folder with the endpoints related to the actions an employee can perform, another folder for a company admin, one for public users and so on and so forth.

We also used SwaggerUI to document our API. It was a great tool to visually see the request and response samples that need to be sent and that will be received. It could have been used as a tool to actually test our endpoints, but we found that Postman was a better tool for that. Swagger was mostly for documentation and for helping the people in the frontend know at a glance what they need to send to different endpoints. It was also a good addition because it effectively sorted all the requests according to the controller in which they belonged.

3 - Project planning

The project planning was well executed once we got the hang of it. We started to use JIRA more effectively and started grouping subtasks below a greater task. We set up automations that would change the status of tasks and send emails when a commit was pushed or when a merge request was approved. The only problem we faced with JIRA was at the end of the project when we ran out of free automations and had to manually do updates, which was not very precise. We also held meeting regularly to see what tasks would be completed next and to assess the progress of our work.

Lessons learned

1 - It is very important to put effort into the design to avoid implementation problems later on.

This was confirmed to us when we reached sprint 3 and had to reimplement the entire backend. We had to redo that because the design was not complete and was not followed. Had we followed the design from the beginning, we would have not spent days redoing everything almost from scratch.

2 - It is important to put team members where they are comfortable.
This makes it easier on everyone in the team. Members will not hate what they are doing if they do not understand a particular technology. It also speeds up the development process when we have people that are comfortable with what they are doing instead of people that are learning to use a technology.

3 - Not everyone will contribute the same amount.
We realized that it is not possible for everyone to contribute the same amount to the project. That is if there are 9 people in a team, it is not realistic to split the coding into 9 parts. It will not work and we lived it. What ends up happening is some people are better at coding and some people are better at writing documentation. So even though the contribution of each member is not equal in all aspects, when we look at the overall project, everyone's work is different. One writes more documentation so that others can focus on developing faster and better.

4 - It is important to have people assigned to lead a project.
We assigned team leads in the frontend and backend as well as a project manager, which greatly helped us in our work. The leads were in charge of monitoring their teams and assigning tasks and the project manager's role was to ensure the project was on track. If we did not assign these roles, it would have been difficult for 9 people with different ideas to efficiently work on this project without causing problems down the road.