SOEN 390 - Software Engineering Team Design Project
Final Report

Due Date: May 1st, 2024

Written By: Team 5

Student Name	Student ID
Ann-Marie Czuboka	40209452
Isabelle Czuboka	40209525
Karina Sanchez-Duran	40189860
Guillaume Lachapelle	40203325
Nicholas Piperni	40207764
Oliver Vilney	40214948
Poula Farid	40208791
Ziad Elsharkawi	40213438
Carla Shawi	40190561
Tharsipa Nadarajah	40190597

Department of Computer Science & Software Engineering
Concordia University

Winter 2024

1. Table of Contents:

2. Vision Statement:

The stakeholder is the Product Owner, who holds the responsibility of overseeing every aspect of the project's development. This entails close collaboration with various teams, including the Project Manager, Quality Assurance Team, UX/UI Designers, Web Developers, and the Legal Department. The Product Owner's duties encompass defining project requirements, communicating with all involved teams, establishing acceptance criteria, and actively participating in key development activities such as sprint planning and product backlog management. This concentrated involvement ensures that the project progresses in alignment with the Product Owner's vision and objectives, ultimately leading to its successful realization.

As for the users of the condo management system, they encompass various categories, each with distinct roles and responsibilities. Public users, aiming to become condo owners or renters, undertake tasks such as profile creation, registration key submission, and facility reservation. Condo owners, as property owners, utilize the system to access property information, submit requests, and reserve common facilities. Similarly, condo renters engage in common facility reservations through the platform. Condo management companies, responsible for property administration, handle tasks including property profile creation, unit details entry, registration key distribution, common facility setup, and employee role assignments. Meanwhile, employees associated with these buildings interact with the system by creating profiles, associating with buildings, responding to requests, and providing task completion notifications. Each user category contributes to the smooth functioning of the condo management system through their dedicated roles.

3. Requirements Documentation:

For each requirement listed in the traceability matrix, comprehensive documentation is provided in this section, encompassing user stories, screenshots, acceptance tests, and implementation details.

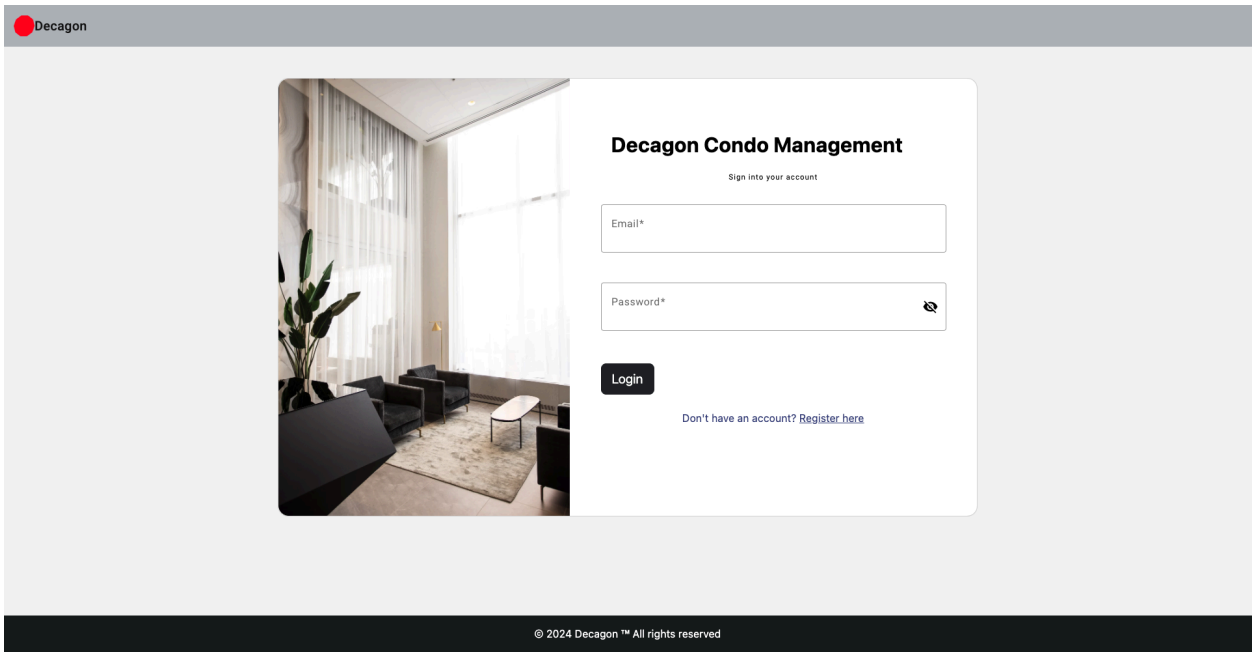
3.1 Requirement #1

As a user, I would like to Login/Sign-Up and be authenticated in order to safely access my account

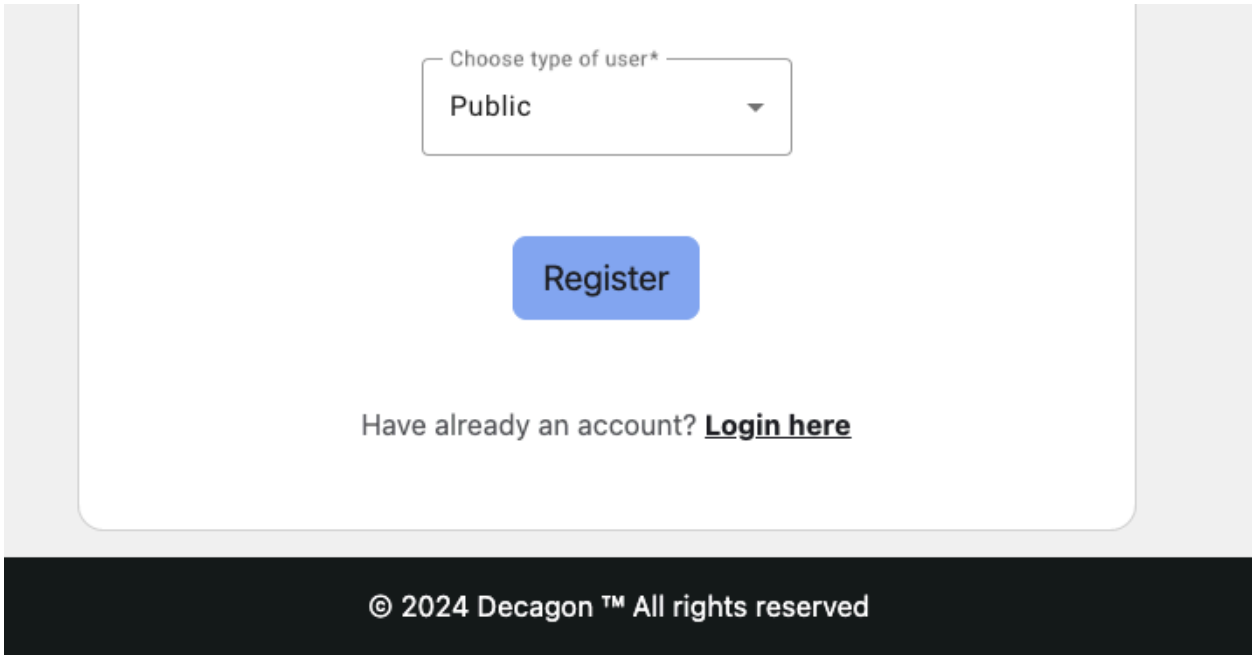
The login/sign-up requirement is part of user story #11. This requirement was implemented at the very beginning of the project as a Sprint 1 deliverable. It allows the user to log in and be authenticated, as well as enable the user to log out of their account.

This implementation includes several UI pages: (1) login, (2) register and (3) verify-email.

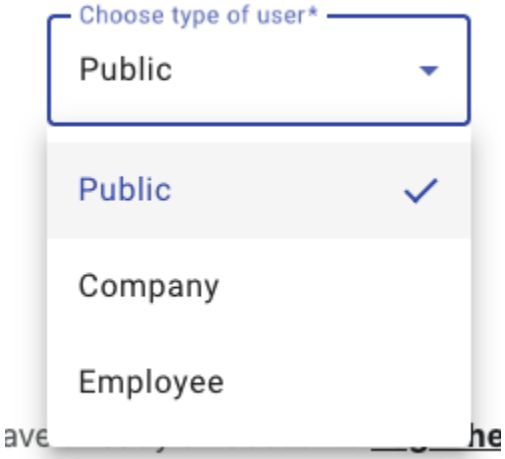
Login page



Register page

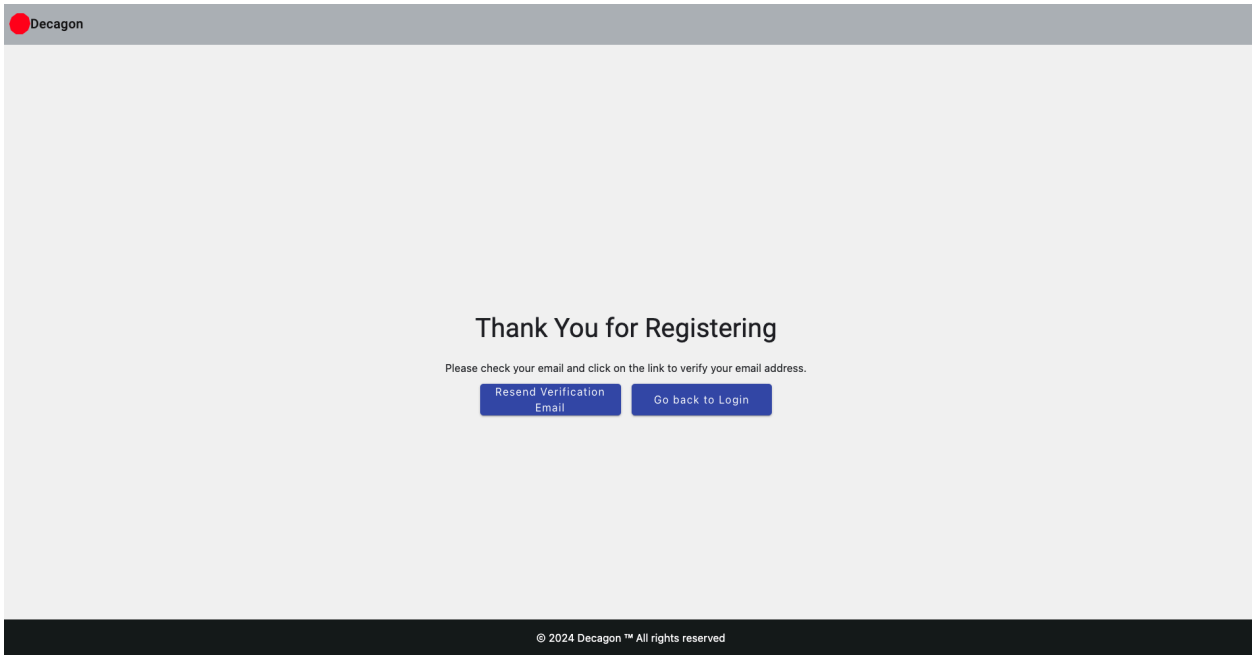


Of course, we included a way to know what type of user is registering an account in the system. We decided on a simple dropdown included in the register page seen below.



Verify-email Page

Once a new user registers an account in the system, the UI navigates them to this page. In the meantime, an email is sent to the email the user provided



In order to verify this requirement, the team has created a user acceptance test that simulates real-world scenarios and evaluates the system's functionality of logging in with proper and improper credentials.

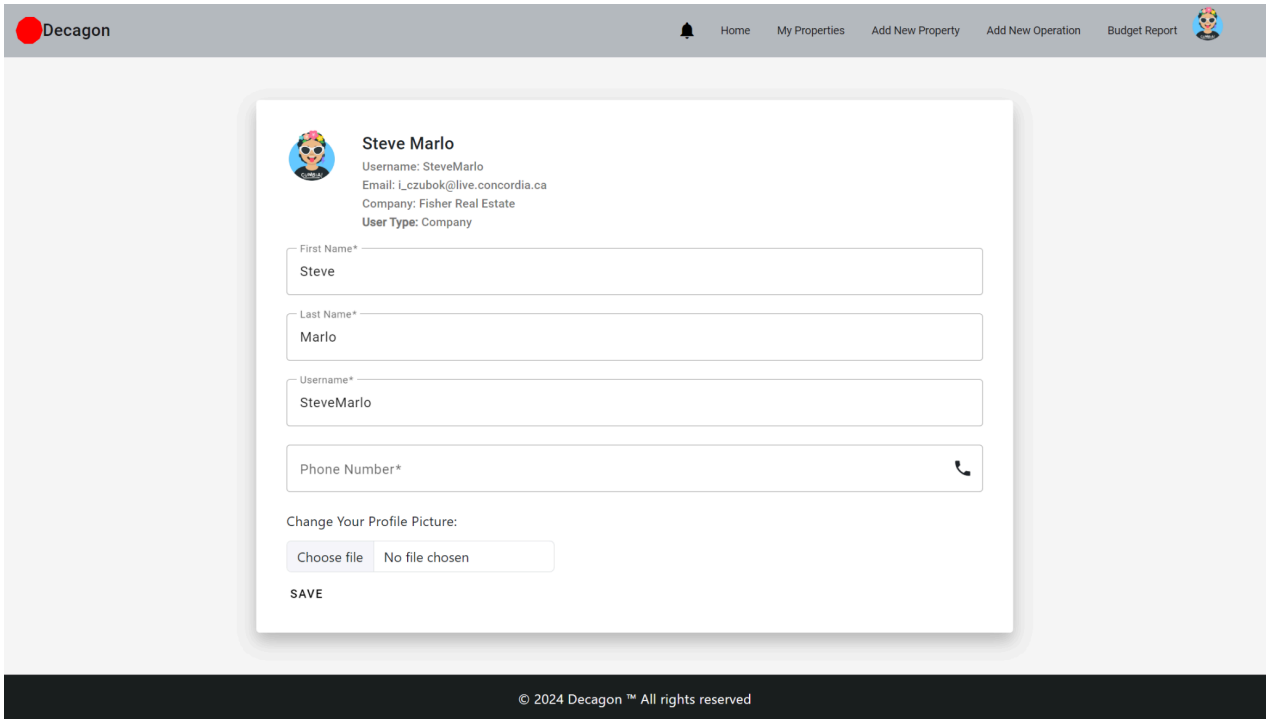
This user acceptance test can be accessed in the GitHub repository. The cypress test is named LoginAndProfilePage.cy.ts.

3.2 Requirement #2

As a user, I would like to personalize my own profile in order to provide more detail about me specifically.

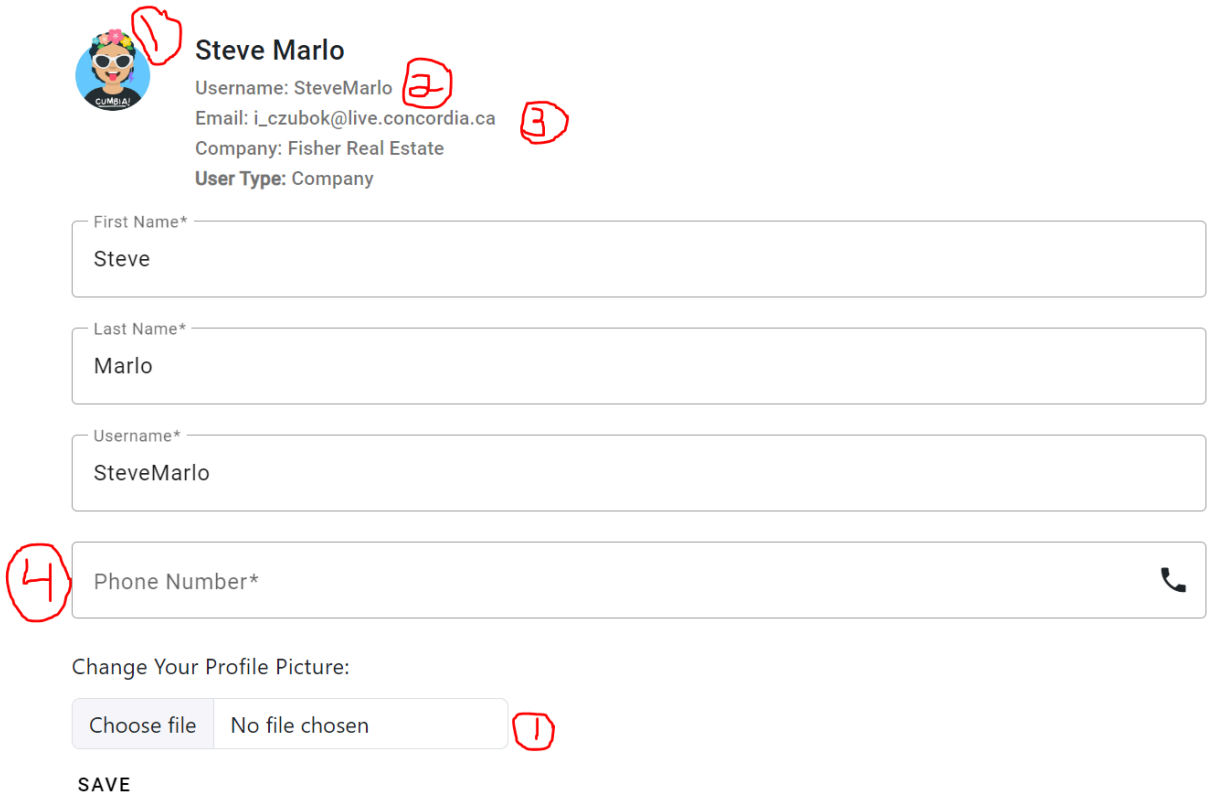
The Profile Page requirement is part of user story #12. This requirement was done at the very beginning of the project as a Sprint 1 deliverable. It allows the user to personalize their profile on their account. This implementation includes only one page: the profile page whose icon is at the top right of the header of every page. As per requirements, the page includes these features: profile picture (1), user name (2), contact email (3), and phone number (4).

Profile Page



Icon and Link to Profile Page

Features



This user acceptance test can be accessed in the GitHub repository. The cypress test is named LoginAndProfilePage.cy.ts.

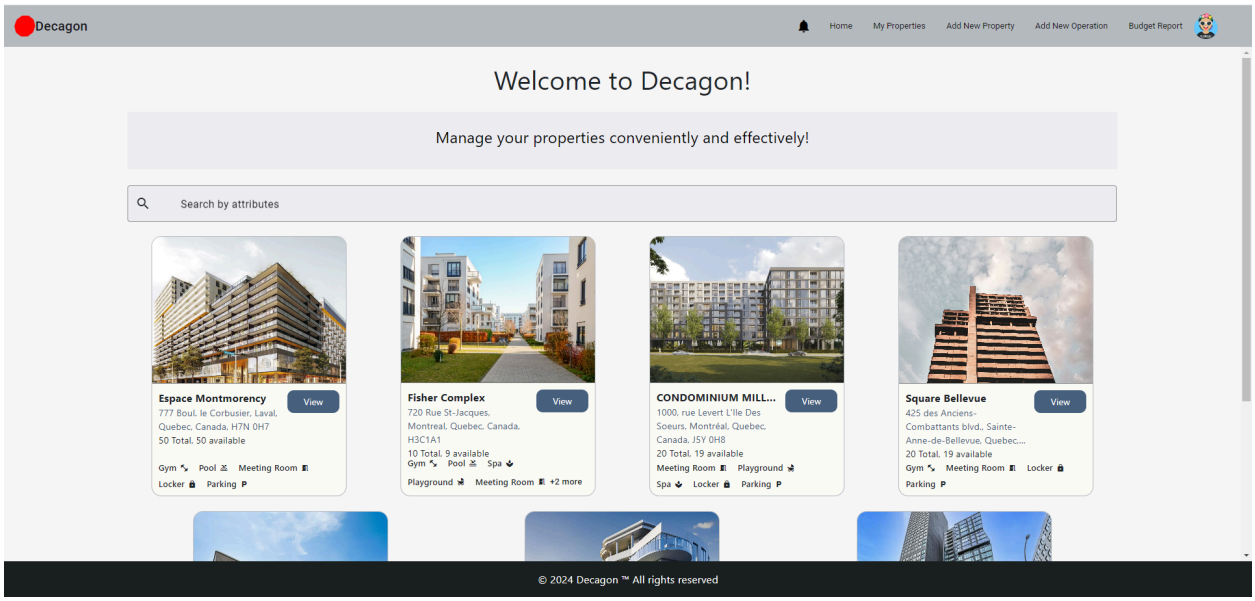
3.3 Requirement #3

As a user, I would like to have a good view (dashboard) of my properties to quickly access my portfolio units.

This user story encompassed a comprehensive effort spanning multiple sprints from Sprint 2 through Sprint 4, to enhance the user experience by providing a robust dashboard view of properties. This involved the creation and implementation of various backend functionalities and frontend components to ensure seamless access to portfolio units for both managers and users. The tasks were meticulously broken down to cover a spectrum of requirements, from user acceptance tests to frontend development and backend integration. Below will be a detailed breakdown of the tasks composing this user story.

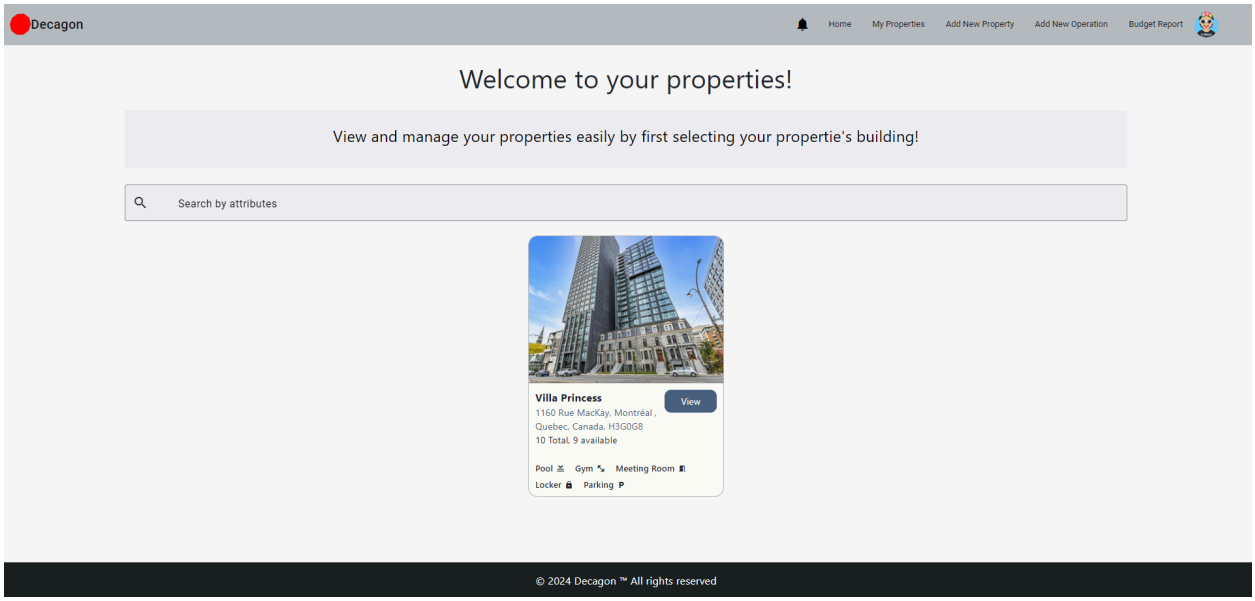
Landing Page: This page provides users with a dashboard that displays a portfolio of properties. Each property is presented in a card format with pictures, addresses, total

number of units with how many are available for rent or sale and icons indicating the amenities.



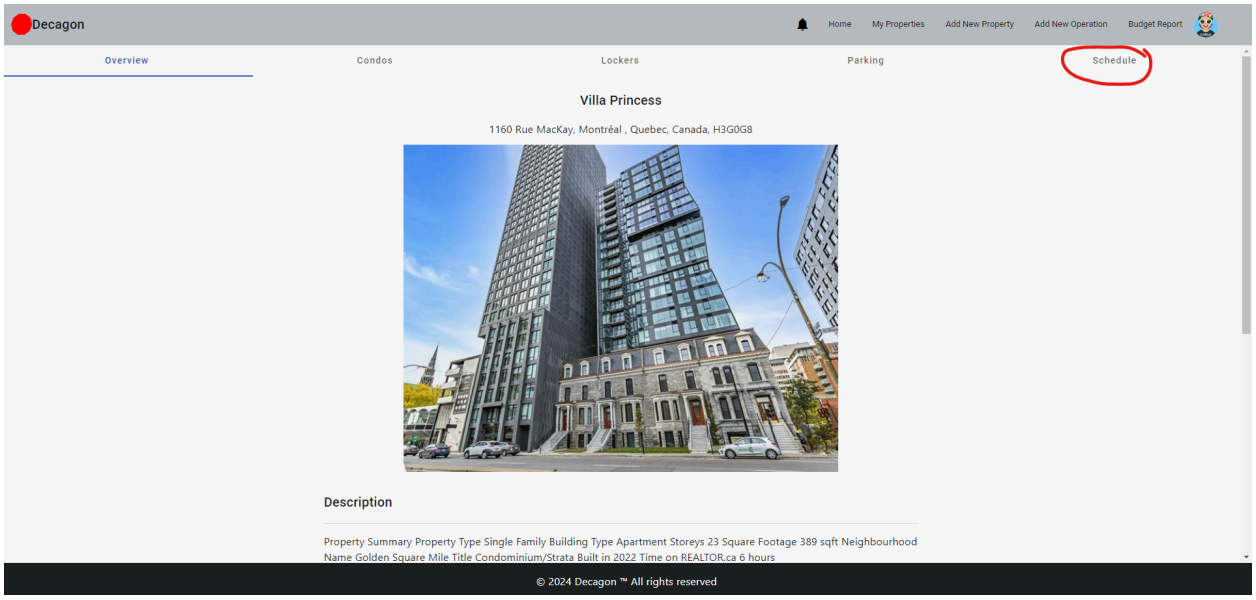
User Acceptance test can be found under the name: [AvailableProperties.cy.ts]

My Properties Page: This interface is designed to give users a straightforward and efficient way to view and manage their properties. Upon clicking "My Properties," the user is greeted with a welcoming message and prompted to manage their properties by first selecting a building from their portfolio.

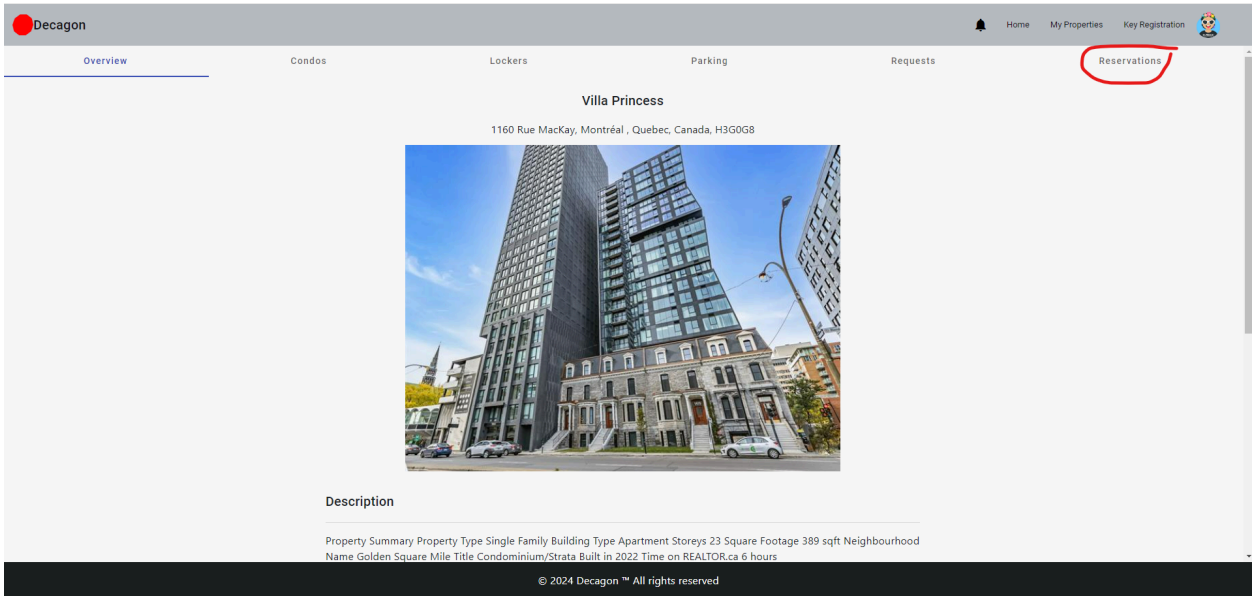


User Acceptance test can be found under the name: [MyPropertiesCompany.cy.ts] and [MyPropertiesPublic.cy.ts]

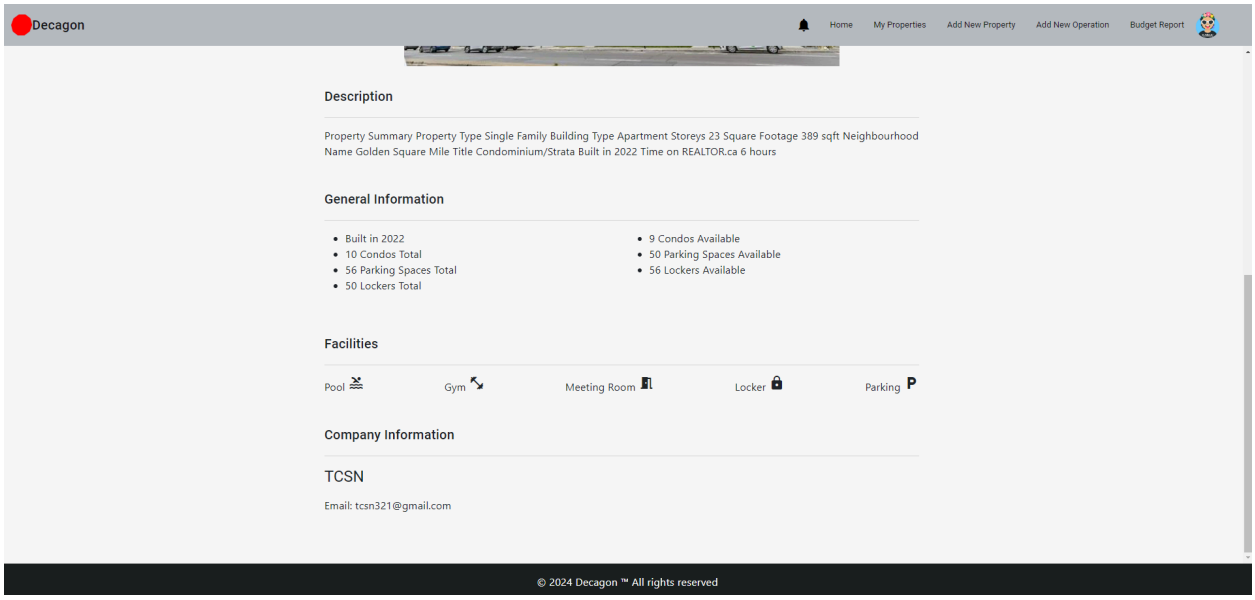
Building-info Page: Upon clicking the "View" button in the "My Properties" section of the Decagon property management application, the user is presented with a detailed overview page for a specific property. This page is designed with a versatile tabbed interface, which caters to the distinct needs of different user types. For company users, such as property managers or administrative staff, the tabs available include Overview, Condos, Lockers, Parking, and Schedule. These sections provide extensive management tools and detailed property insights.



In contrast, renters accessing the property information are presented with a slightly modified set of tabs tailored to their experience, which might include Overview, Condos, Lockers, Parking, Requests, and Reservations, enabling them to manage aspects of their rental, report issues, and utilize shared building amenities.

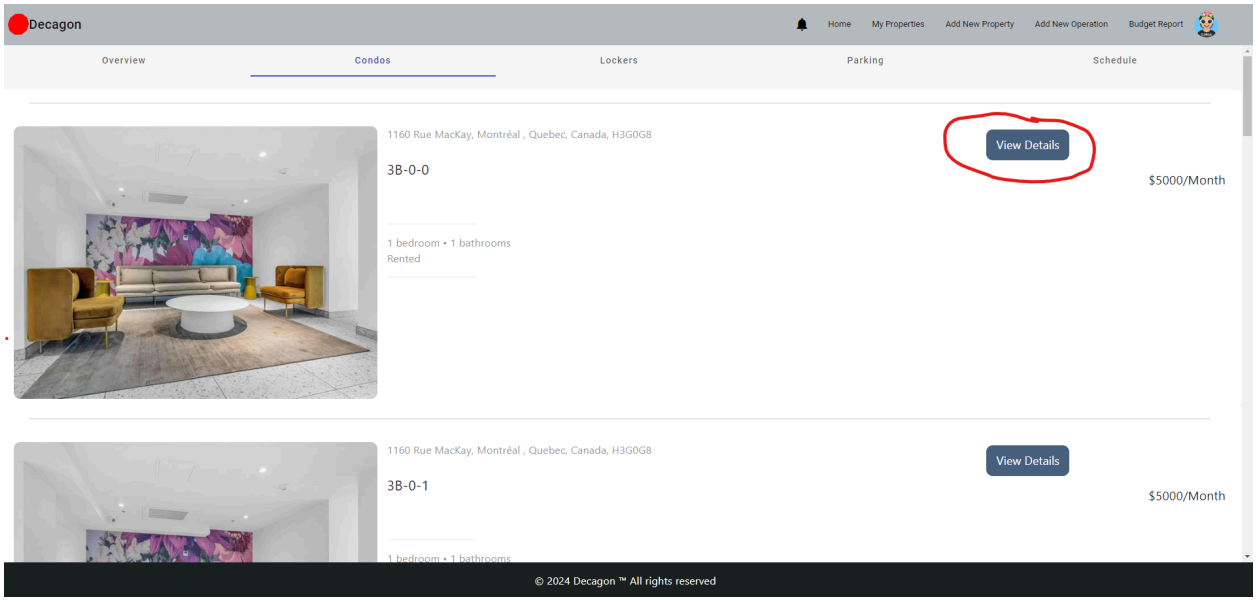


Overview Tab Page: The detailed overview includes a description section that provides a summary of the property, a general information section which includes the year it was built, the total number of condos, available condos, parking spaces, and lockers, along with the available facilities and finally a section that has the company information.

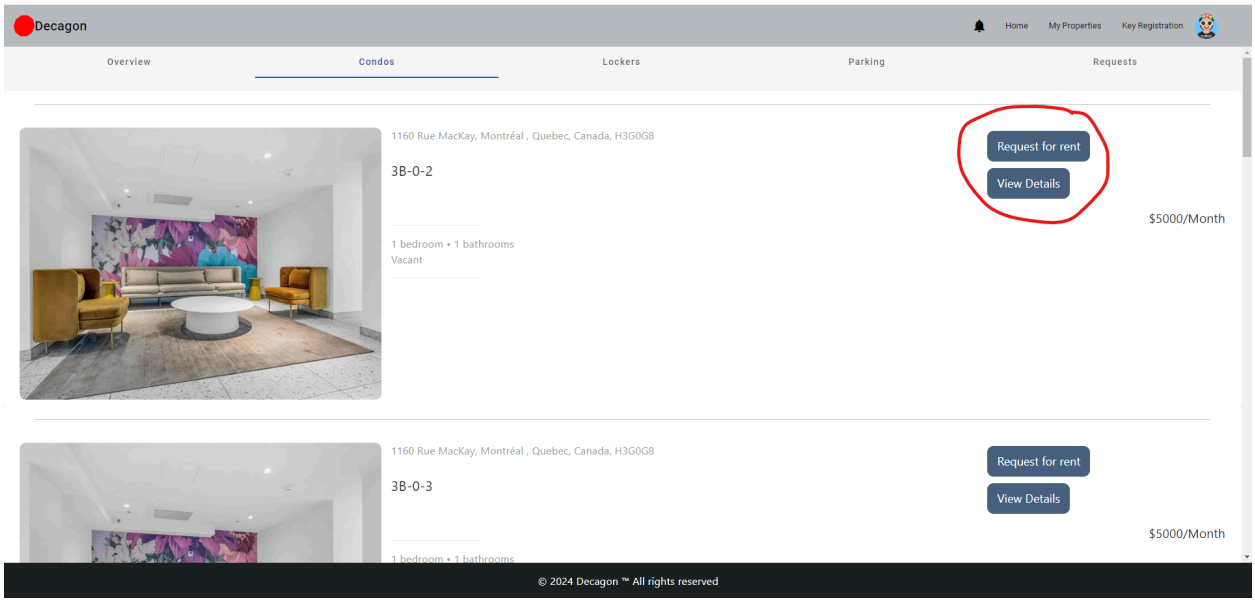


Condo Tab Page: As users navigate to the Condos Tab within the Decagon property management application, they encounter a user-centric design that displays a comprehensive list of all condos under that building. This layout is optimized to provide detailed information on each unit, facilitating an in-depth review for those interested in the specifics of each property.

Company users, can keep track of the available condos and see their details

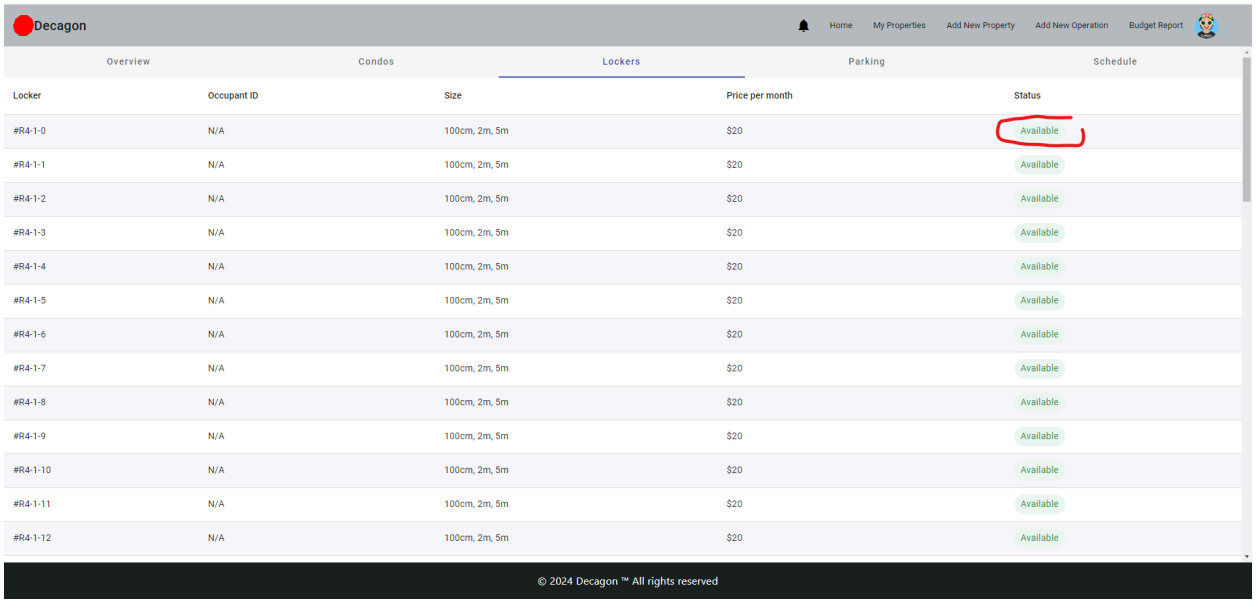


Public users since they can be potential renters or buyers will have the capability of requesting a specific condo for rent or ownership.

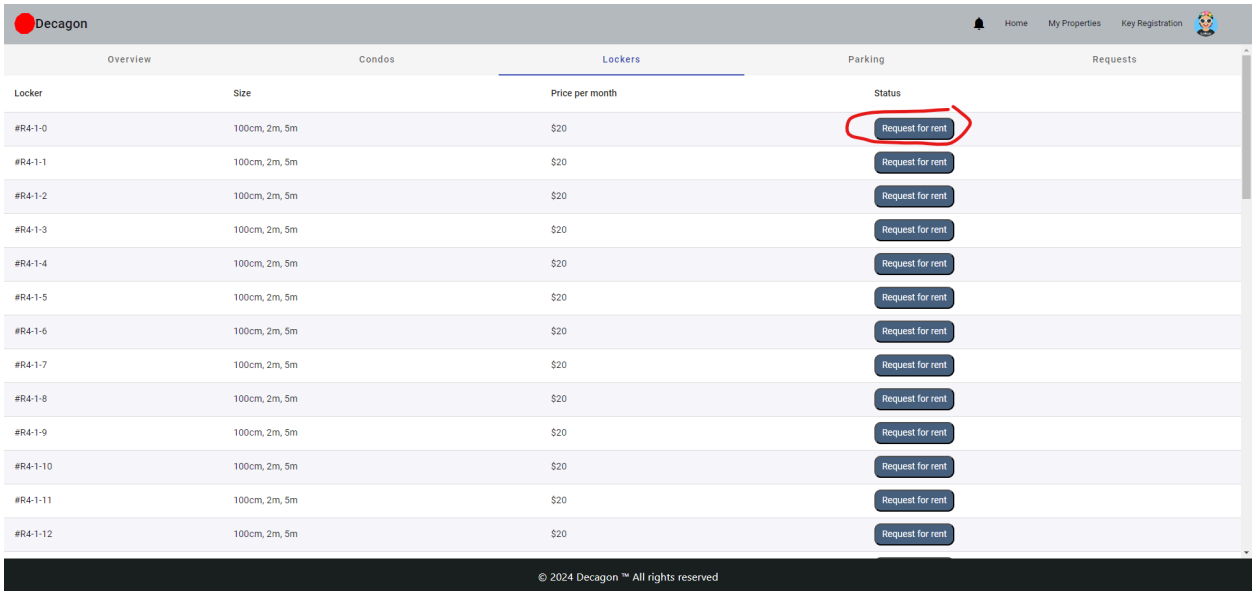


Lockers Tab Page: When users visit the Lockers Tab in the Decagon property management application, they are met with an intuitive and organized display showcasing a complete list of lockers associated with a particular building. This interface is designed for ease of use, presenting all necessary details about each locker, including dimensions, location within the building, and rental price.

The company's view focuses on locker management. It shows the locker numbers, their dimensions and the occupancy status marked as “Available” for each unrented locker.



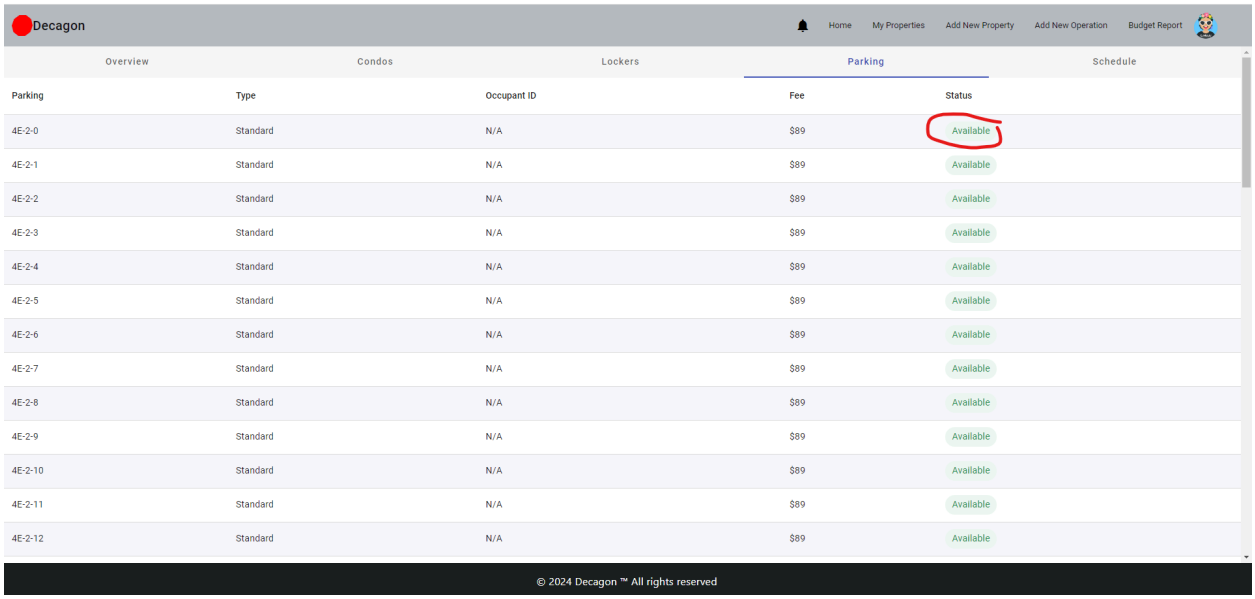
The public user's view provides users with a "Request for rent" button, enabling users to easily express their interest in renting a locker. This feature simplifies the process for individuals seeking additional storage space in the building, allowing them to secure a locker without needing to navigate away from the page.



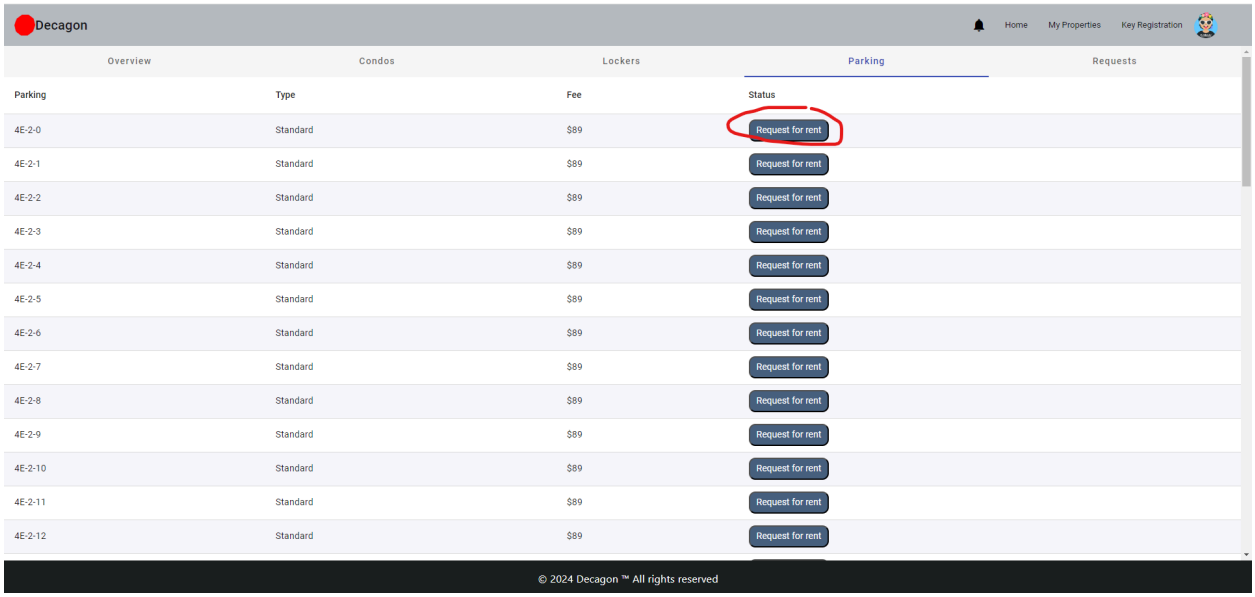
Parking Tab Page: As users access the Parking Tab in the Decagon property management application, they are introduced to a clear and structured overview of parking spaces

within a specific building. This interface is meticulously crafted to display all necessary details, which include the parking spot number, the type of parking space, and the associated fee.

For company users, this tab is a valuable resource for parking space administration. It enumerates each spot, the type—whether it's standard or reserved—and the status, denoting “Available” for spots that are currently unoccupied. This view is essential for tracking the utilization of parking spaces and for updating the status as they are rented out or become available.

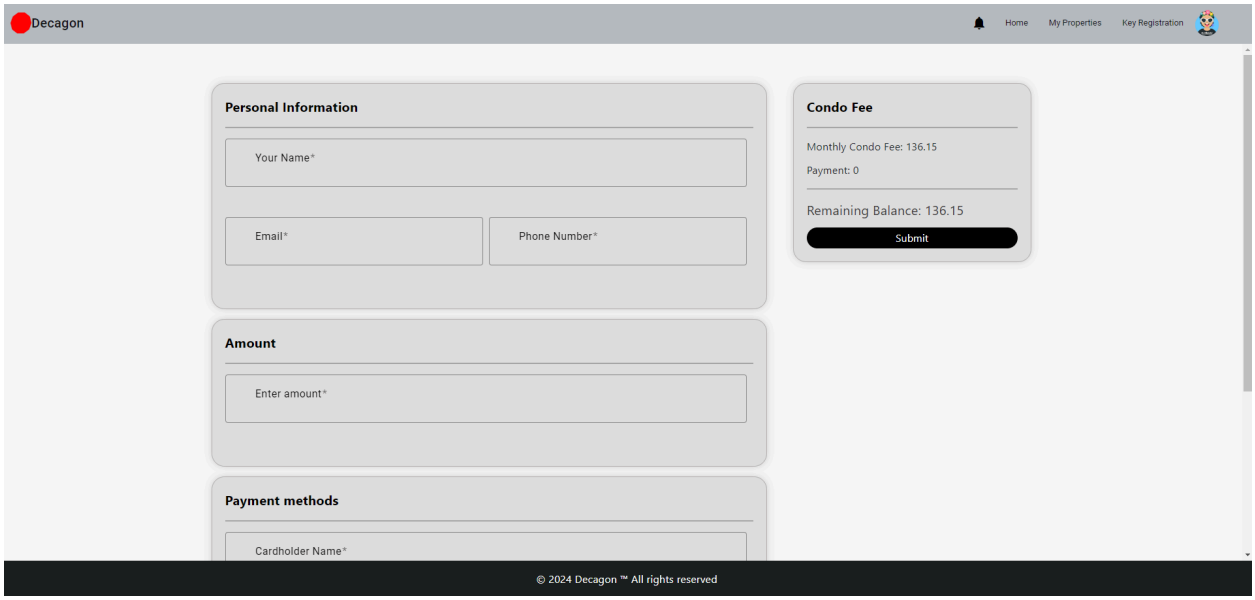


For public users, the interface accommodates straightforward interaction through a “Request for rent” option associated with each parking space. This functionality streamlines the rental process, enabling individuals to express interest and begin the rental process with a single click, directly from this tab. This user-friendly feature is aimed at simplifying how prospective tenants or visitors secure parking within the building.

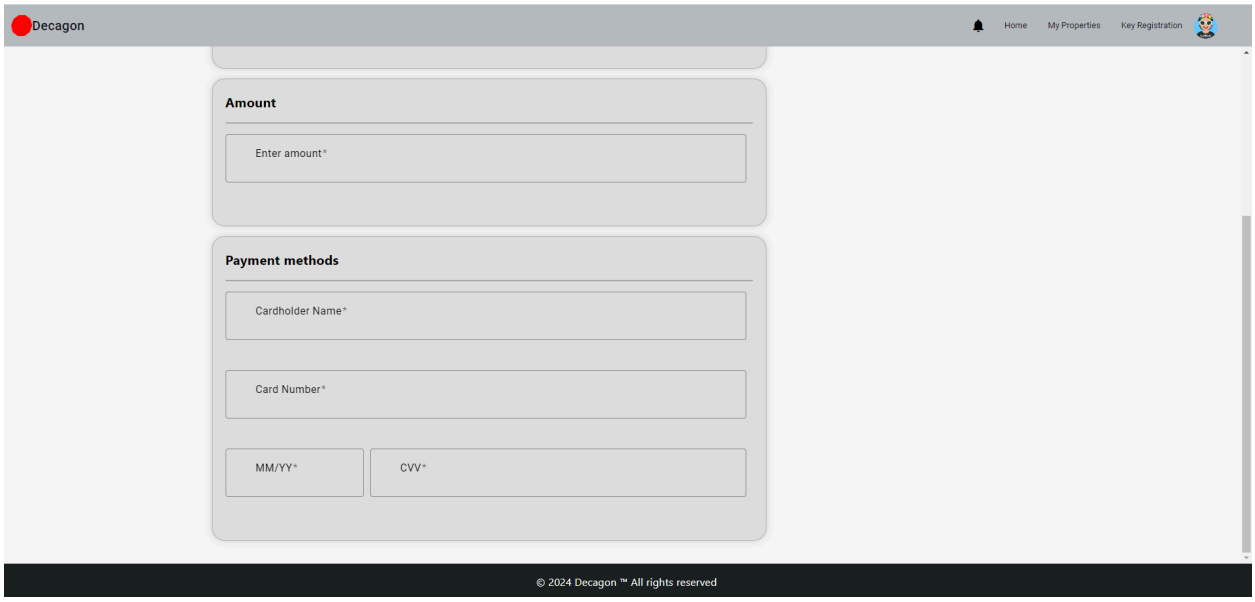


User Acceptance test can be found under the name: [Building-Overview.cy.ts]

Paybalance Page: This page is designed to offer users a streamlined and secure way to manage and submit payments to their condo fees. It is divided into distinct sections for clarity and ease of use. It starts with a "Personal Information" section where users are prompted to enter their name, email, and phone number, ensuring that the payment is correctly associated with the right tenant or owner. Adjacent to the personal details, there's a "Condo Fee" section that displays the monthly condo fee, any previous payments made, and the remaining balance due. This module provides users with a clear understanding of their financial status. users find an "Amount" field where they can enter the specific amount they wish to pay. This could be the total balance due or a partial payment, depending on their preference or financial situation.



The last section is dedicated to "Payment methods," which includes fields for the cardholder's name, card number, expiration date, and CVV. These details are necessary to process the payment transaction securely.



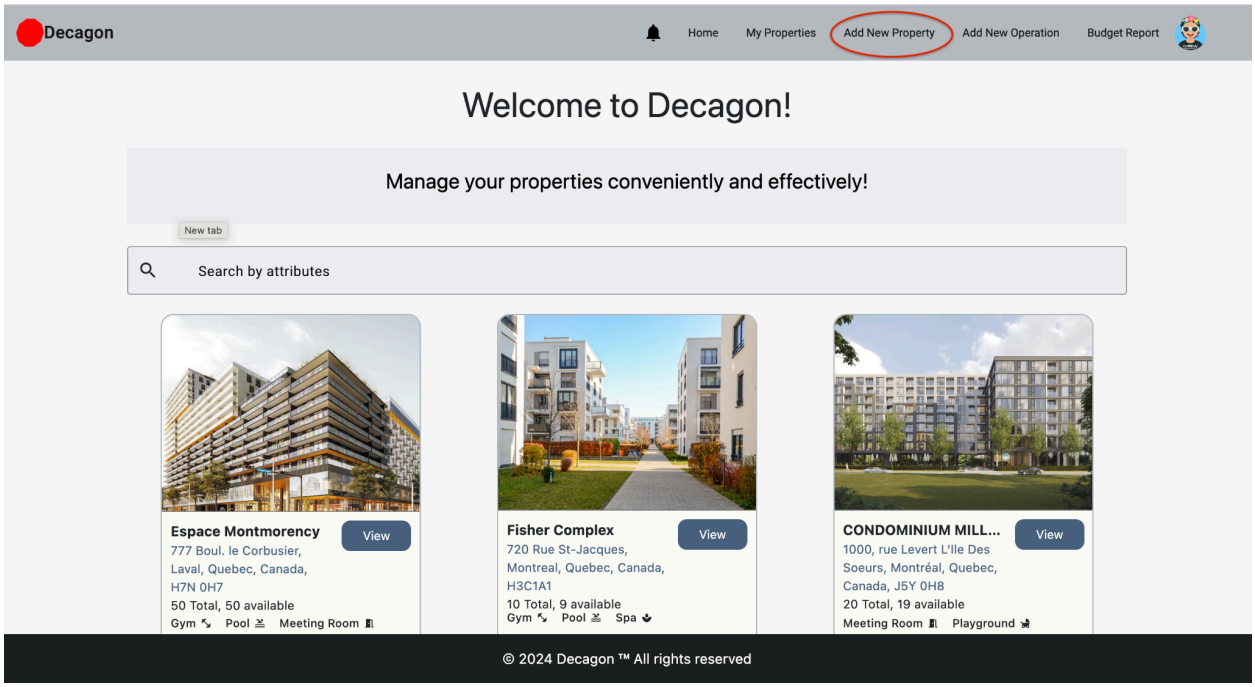
User Acceptance test can be found under the name: [PayFees.cy.ts]

3.4 Requirement #4

As a condo manager, I would like to create a profile for a property under my management to provide more detail on a specific unit.

This user story unfolds across a series of dedicated sprints, spanning from Sprint 2 through Sprint 4. The objective is to enrich the user experience for condo managers by empowering them to create detailed profiles for properties under their management. This involved the creation and implementation of various backend functionalities and frontend components to ensure a seamless process for completing the creation of a profile for a property. The tasks were meticulously broken down to cover a spectrum of requirements, from user acceptance tests to frontend development and backend integration. Below will be a detailed breakdown of the tasks composing this user story.

Landing Page: This page offers users a dashboard showcasing a portfolio of properties. Within the navigation bar, users will find the option "Add New Property," which, with a single click, redirects them to the Add New Property page, initiating the property creation process seamlessly.



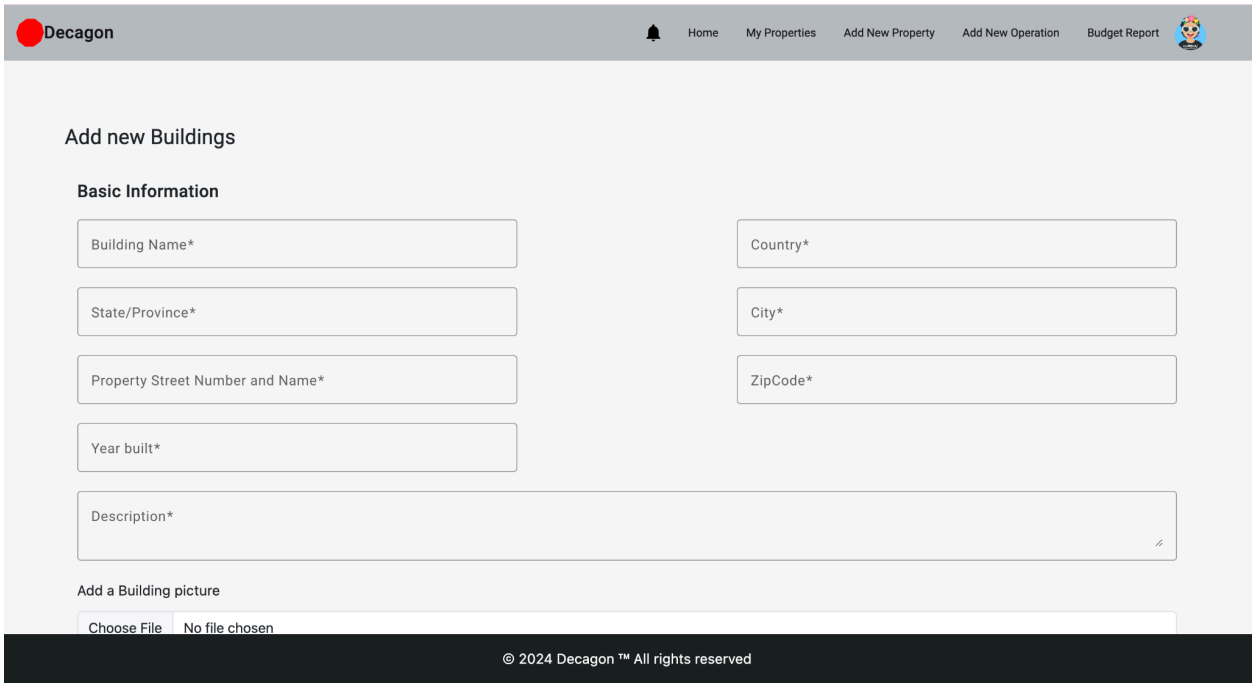
Add New Property Page: This page guides users through the process of adding details about the property they intend to include. Users must input information such as the building name, state/province, country, city, zip code, property street number and name,

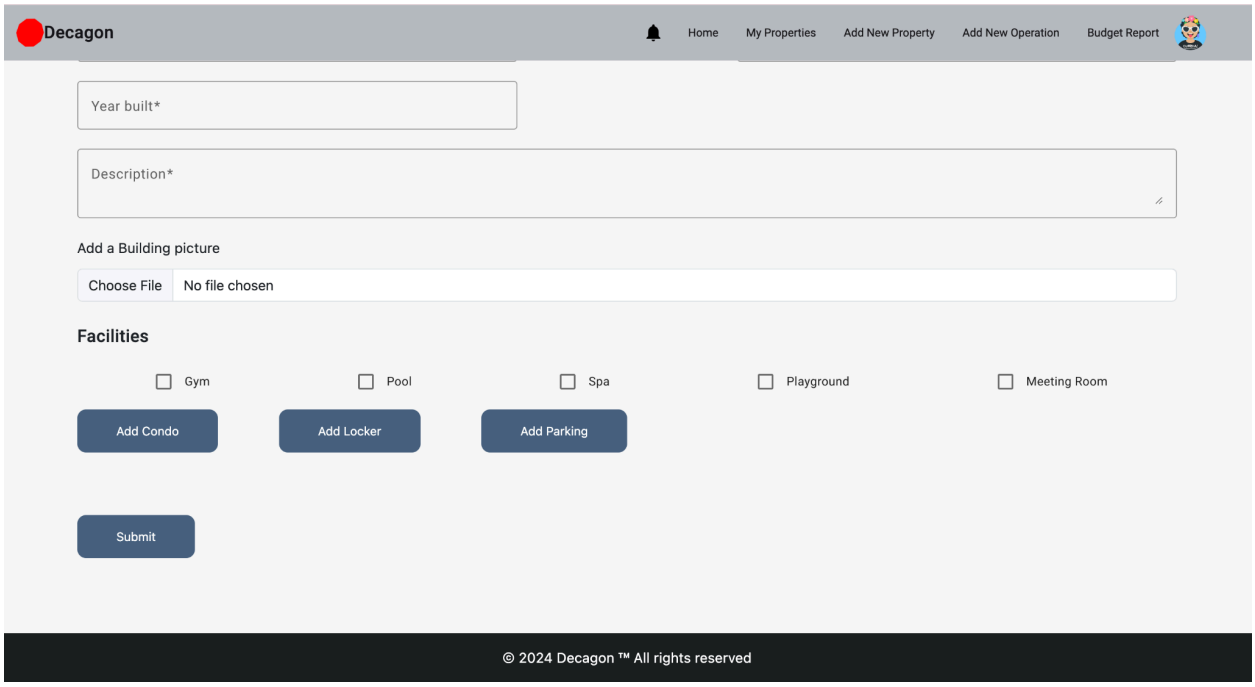
year built, and a description. Additionally, users have the option to upload a picture of the building.

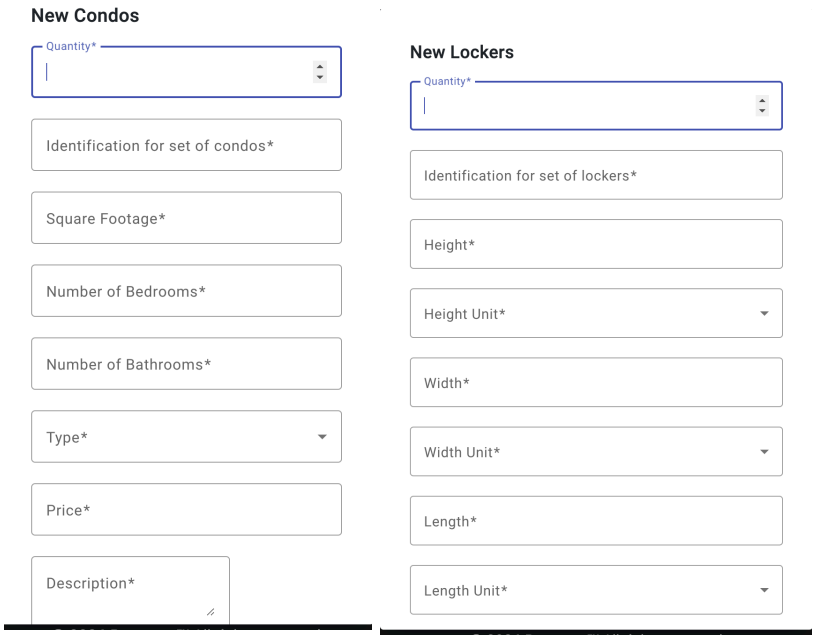
Furthermore, users can specify facilities available within the building, selecting from options such as Gym, Pool, Spa, Playground, and Meeting Room. In addition, users can add specific options like Condo, Locker, or Parking by clicking on the respective buttons.

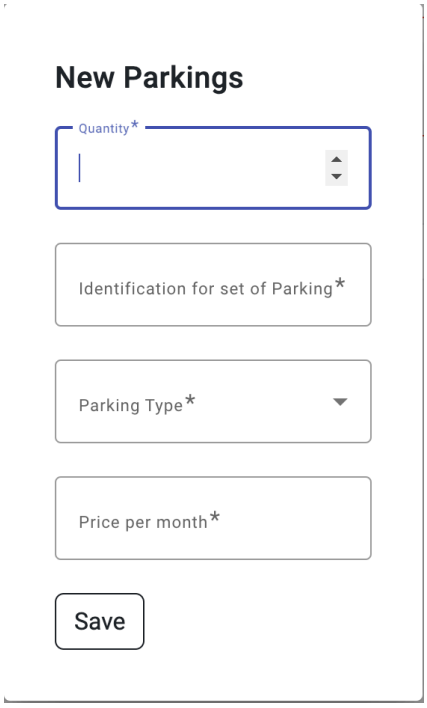
Upon clicking the buttons labelled "Add Condo," "Add Locker," or "Add Parking," users are presented with a form to input the necessary information for each specific option (condo, locker, or parking). After entering the required details, users can save the information.

Finally, after providing all necessary details, users can submit the information by clicking the "Submit" button.




Decagon
Home
My Properties
Add New Property
Add New Operation
Budget Report
Year built*
Description*
Add a Building picture
Choose File
No file chosen
Facilities
Gym
Pool
Spa
Playground
Meeting Room
Add Condo
Add Locker
Add Parking
Submit
© 2024 Decagon ™ All rights reserved



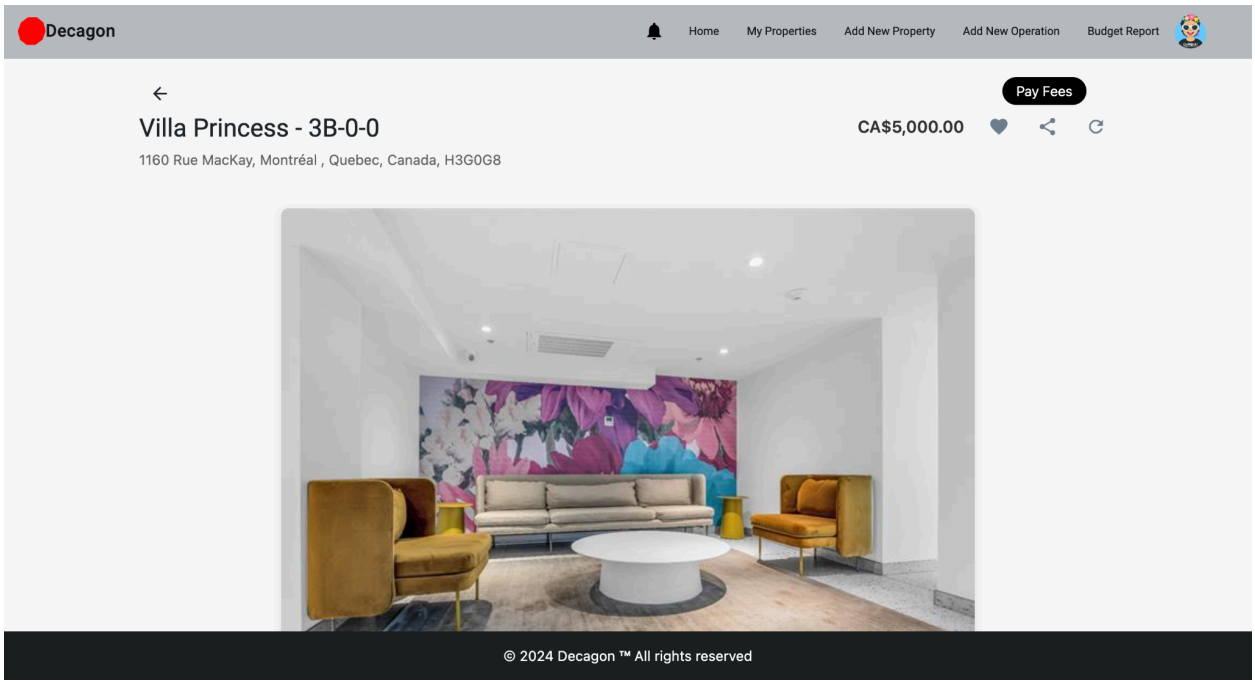


User Acceptance test can be found under the name: CreateBuilding.cy.ts

Upon successfully adding a new property, a user can access the individual condo page by clicking the My Properties option in the navigation bar and clicking on the Condo Tab which provides detailed information on each unit, facilitating an in-depth review for those interested in the specifics of each property. To view the individual condo page, the

user should click the “View Details” button which will redirect to the Individual Condo Page.

Individual Condo Page: The individual condo page serves as a comprehensive platform displaying detailed information about the specific condo unit. Users can observe key details such as the unit's address, displayed prominently at the top, and additional information such as the price along with the features such as the number of rooms, bedrooms, bathrooms, square footage, and the year built, providing a thorough understanding of the property's specifications. Furthermore, the description section offers insights into the unique aspects of the condo. Additionally, the page showcases the location of the condo through an embedded map, enhancing user understanding of its surroundings. Moreover, users can find the contact information, including details about the owner or tenant of the condo. Importantly, if the condo is not currently rented or owned and the condo manager owns the property, the "Edit" button will be visible, allowing them to update the information and ensure that it remains accurate and up-to-date.



Features

Rooms
2

Bedrooms
1

Bathrooms
1

Square Footage
389

Year Built
2022

Description

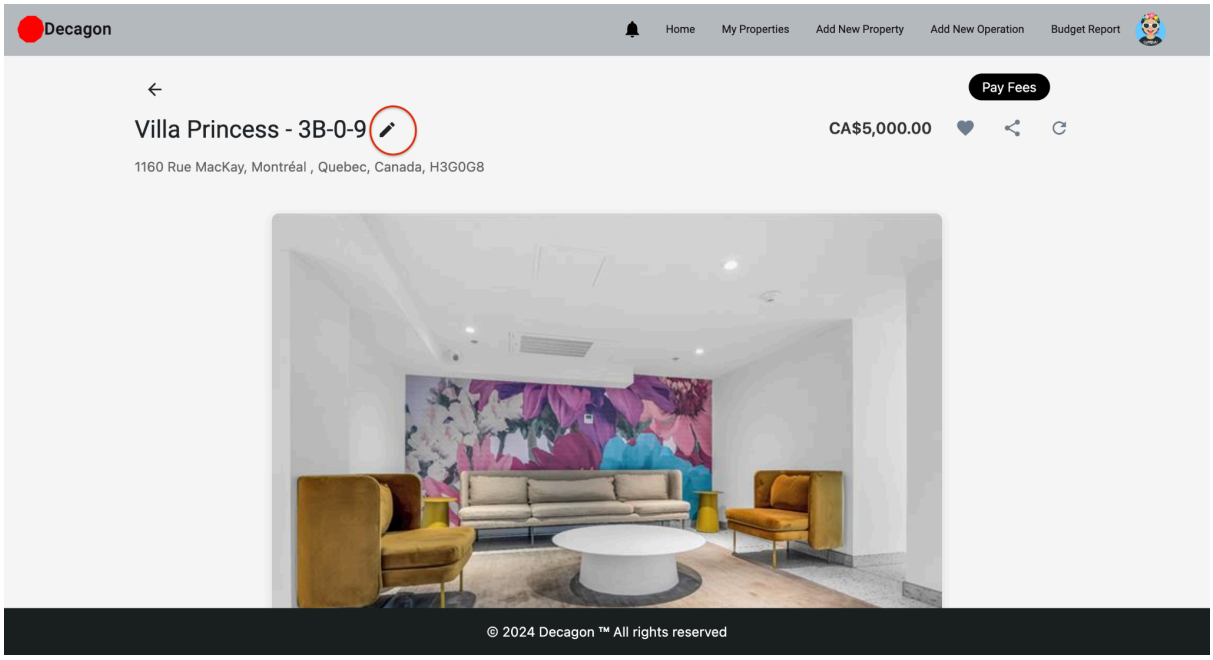
Absolutely unique chance for the investors or first time buyers! compact one closed bedroom apartment , Ideal possibility to combine business and residential space for young or already established. Strategically located at the corner of Rene Levesque and Mackay in prestigious, newly reconstructed multi-use building with gym and swimming pool. Beautiful main lobby off Mackay. High walking score from universities, shopping and transportation. Building still has a warranty ! No need to pay for rent anymore!

Location

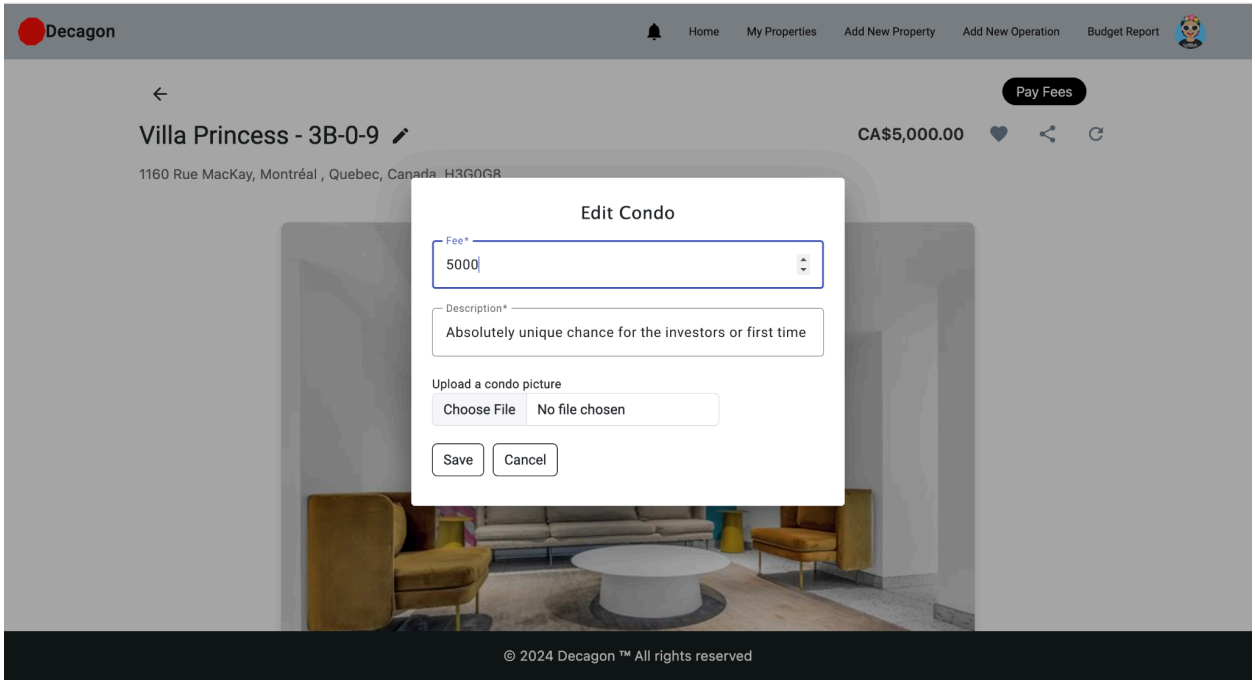
Contact Information

Isabelle Czuboka
Public
Condo status: Rented
isaczu15@gmail.com
514-653-9322

Individual Condo Edit Page: The "Edit" button is visible under specific conditions, ensuring that only authorized users can access it. These conditions include the scenario where the condo is not currently rented or owned, and the condo manager owns the property. By meeting these criteria, the condo manager gains access to the edit functionality, allowing them to modify the condo information as needed, ensuring accuracy and relevance.



Upon clicking the "Edit" button, a form will be displayed, allowing the condo manager to modify the existing condo information. The form includes fields to update details such as the fee, description, and the option to upload a new condo picture. After making the desired changes, the condo manager can save the modifications using the "Save" button. Once saved, the condo information will be updated accordingly, ensuring that the details remain accurate and up-to-date.



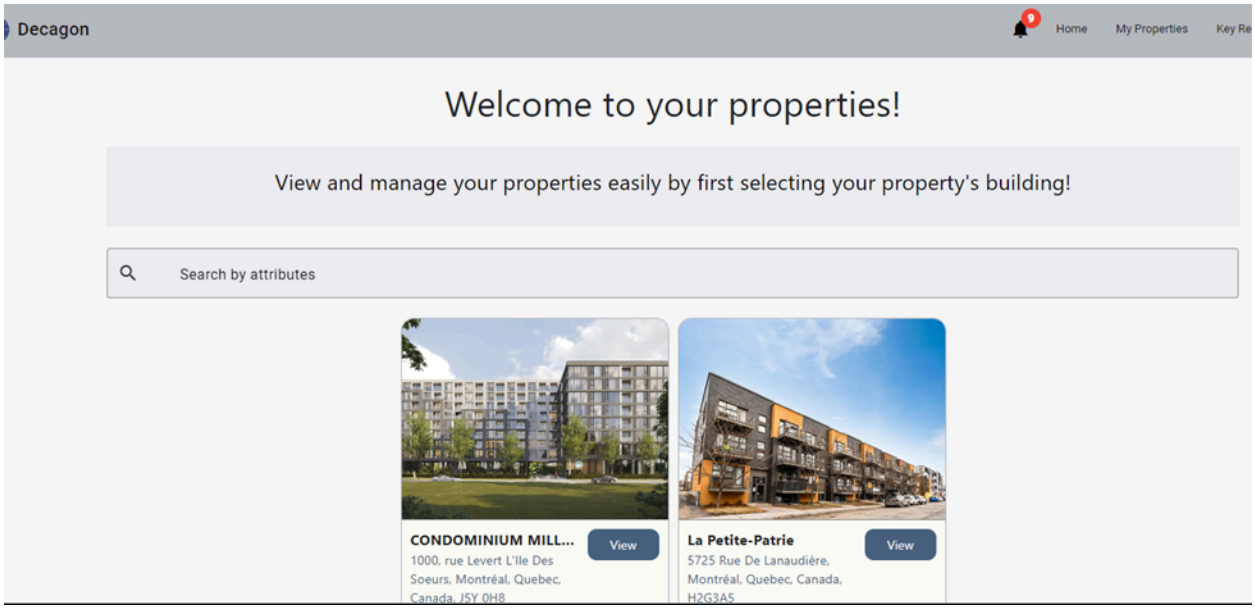
User Acceptance test can be found under the name:[IndividualCondoPageTest.cy.ts]

3.5 Requirement #5

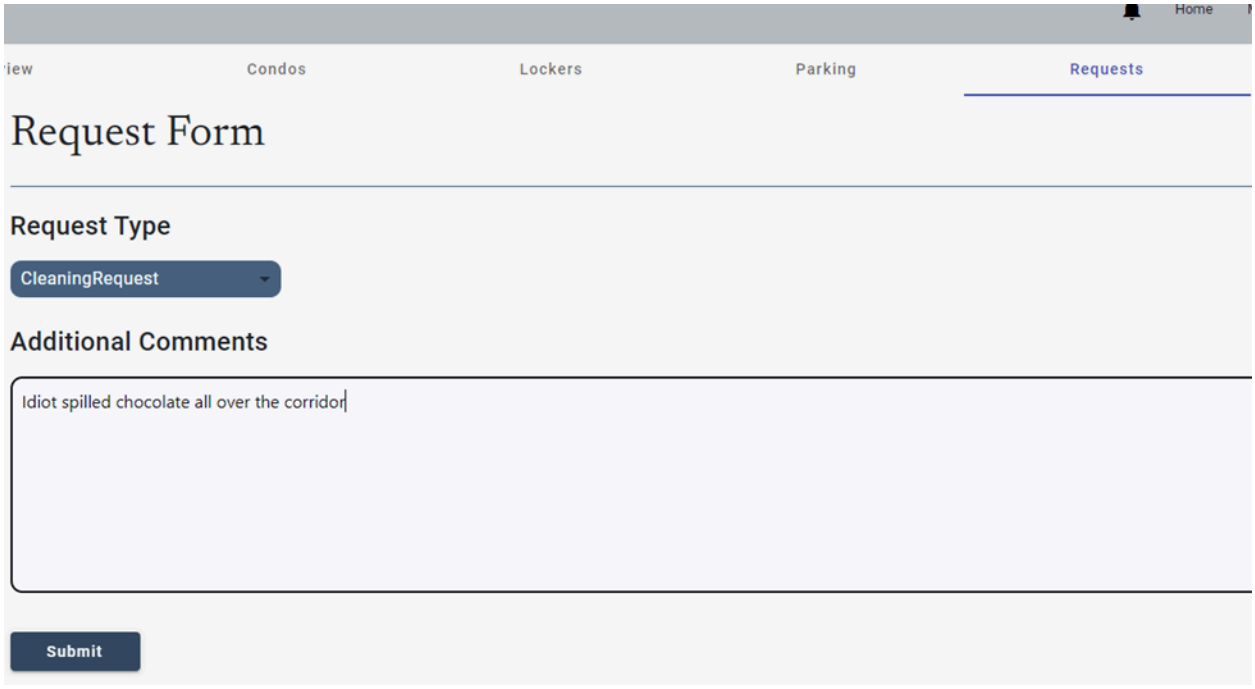
[As a condo owner or renter, I would like to submit requests so I can easily be catered to.]

This user story was developed during sprints 3 and 4. The front end was done during sprint 3. In this sprint, this task was mainly put to satisfy its visual requirements. During sprint 4 the back end was implemented. The functionalities were made as well as the tests. Both acceptance tests and unit tests were done in that sprint. Also since it was the last time the task was worked on it was generally refined in the 4th sprint. The function works from the assumption that someone is an owner and a renter so the page only appears in the info building page where a public user accesses it from the “my properties” page. The owner/renter can describe their request according to a corresponding type. The system will then send a notification (explained in requirement number 6) to the company or employee account letting them know of the request. The company or employee account can then set the status of the request. Upon change of status, the owner/renter will be notified of the change of status. The notification will be sent to the appropriate employee or else the company will receive the notification.

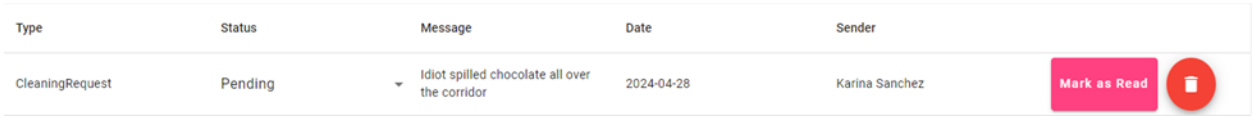
My properties page



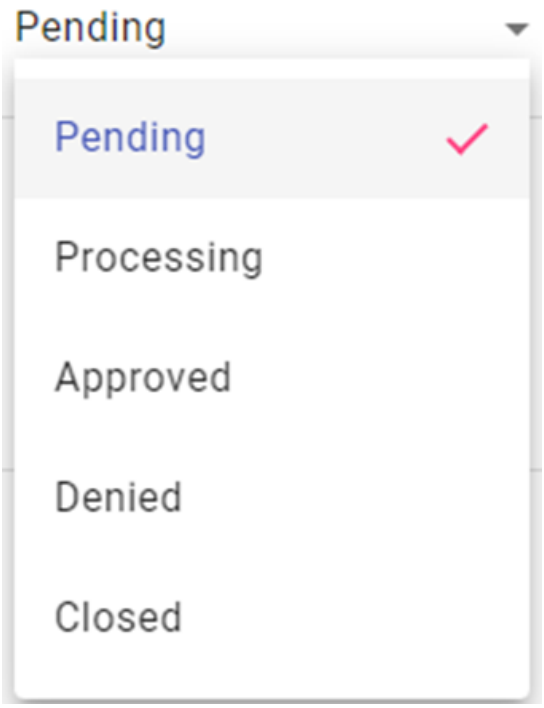
Building info page and the Request tab (aka request page)



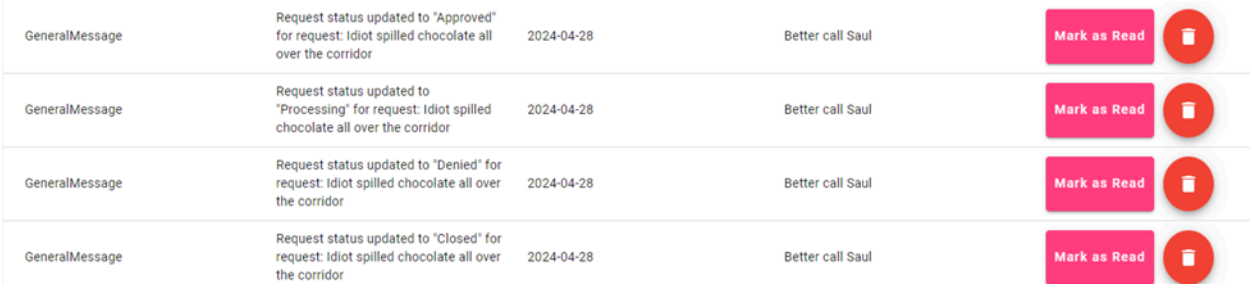
Corresponding Company notification



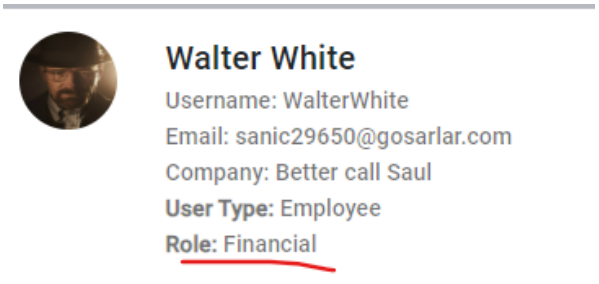
Status change possibilities

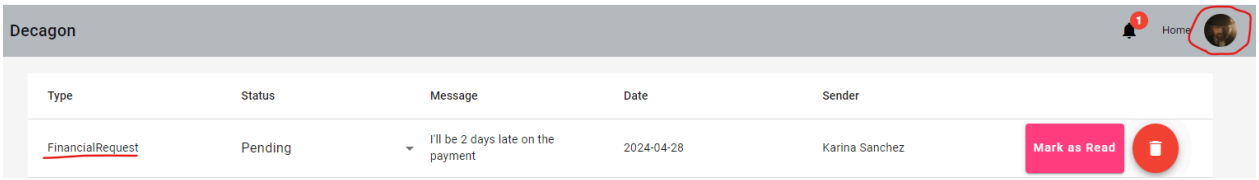


Corresponding public user notification



Notification with corresponding employee (in this case a financial employee received a financial request)





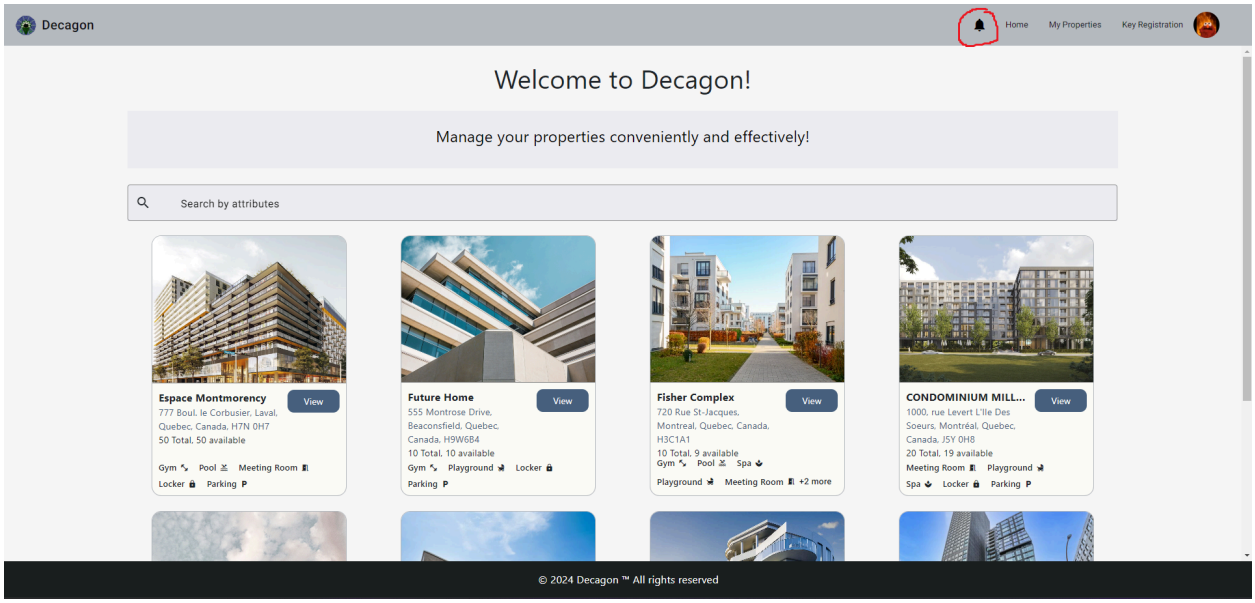
User Acceptance test can be found under the name:[RequestsAndNotifications.cy.ts]

3.6 Requirement #6

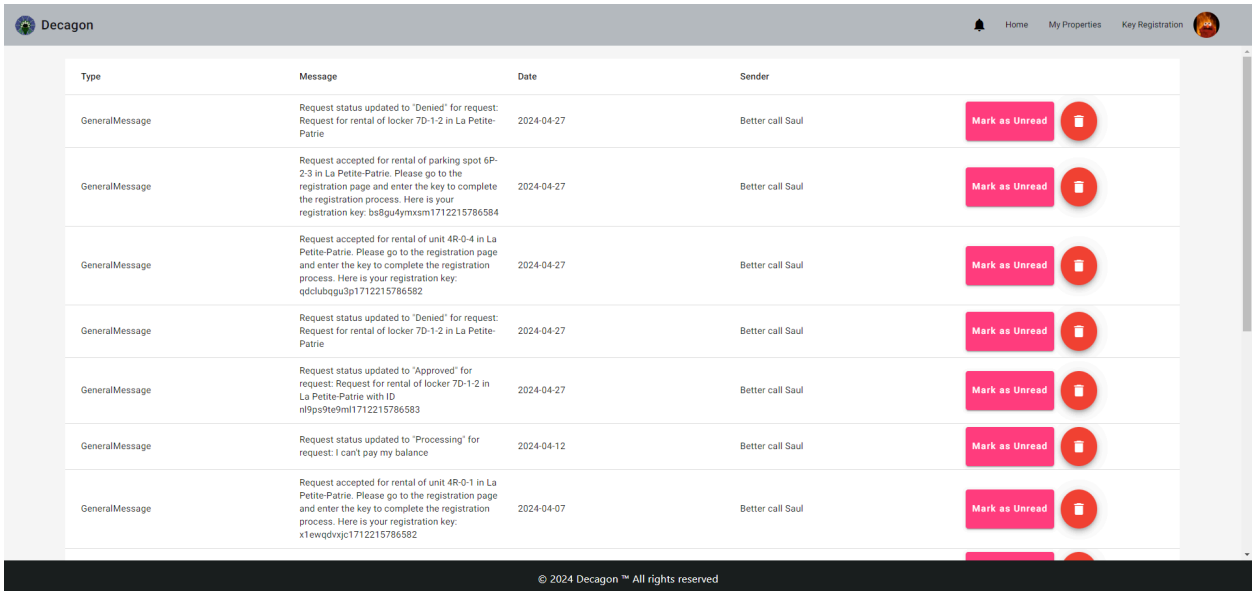
As a user, I would like to view notifications personalized to me on my account for quick updates.

This user story was developed during sprints 2 and 3 and was refined in sprint 4. In sprint 2, the main goal was to focus on the front end for the notification icon in the header, as well as the notifications page. Both frontend and backend were implemented during sprint 2. Sprint 3 focused on user acceptance tests, which test the notification icon in the header as well as the notification page itself. The icon also needed to be tested since it contains a badge indicating the number of unread notifications. Sprint 3 also added frontend and backend functionality to request a condo unit, a parking spot, or a locker. Such requests for rent/ownership are sent via the notification system to the company that manages those buildings. During sprint 4, the main goal was to refine the notification system. First, we linked it to the Requests page, adding a type to every notification. Therefore, any request sent would have a type. This request type would enable us to complete another refinement we had in mind, which is ensuring that the requests are sent to the employee with that role in that building, instead of straight to the company user that manages that building. Here's an example. Let's say I am a public user making a rent request on a condo unit, and then that request is sent straight to the company that manages that building. However, if the request is more specific, through the Requests page, such as a financial request, the system will look through the employees of that building and try to find an employee with the "financial" role assigned to that building. If it cannot find such an employee, then the request is sent directly to the company. This is a way to ensure that employees receive requests with types that match their employee role. Finally, we added a notification status. This status updates the public users of their requests' status. For example, if I am a company accepting a condo ownership request, I can set the status of the request to "Accepted" and the public user who made that request will get notified of that change and receive the registration key for the condo. If I set the status to "Denied", then they also get notified of that change but do not receive the registration key for the condo. This is a way to improve the user experience by notifying public users when their requests get processed.

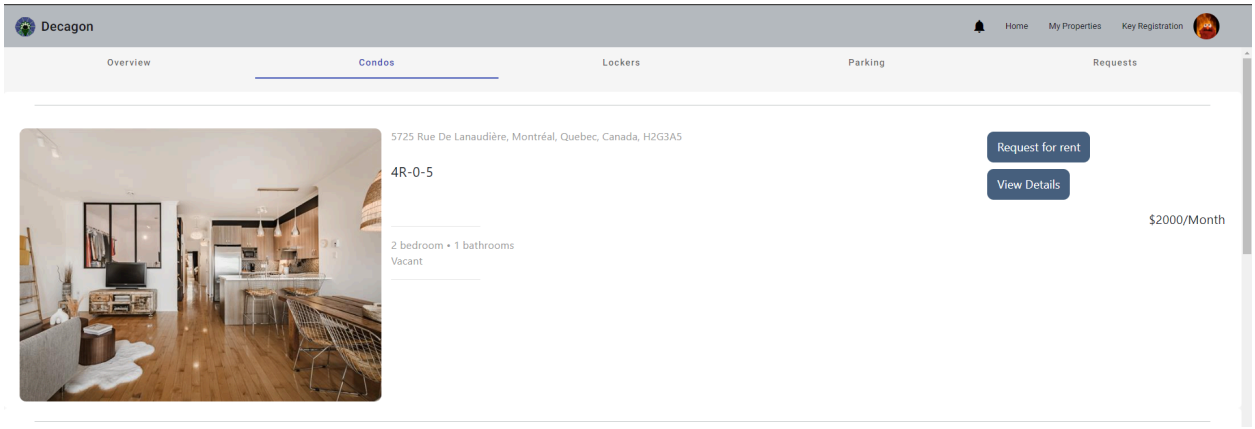
Location of notification icon in header



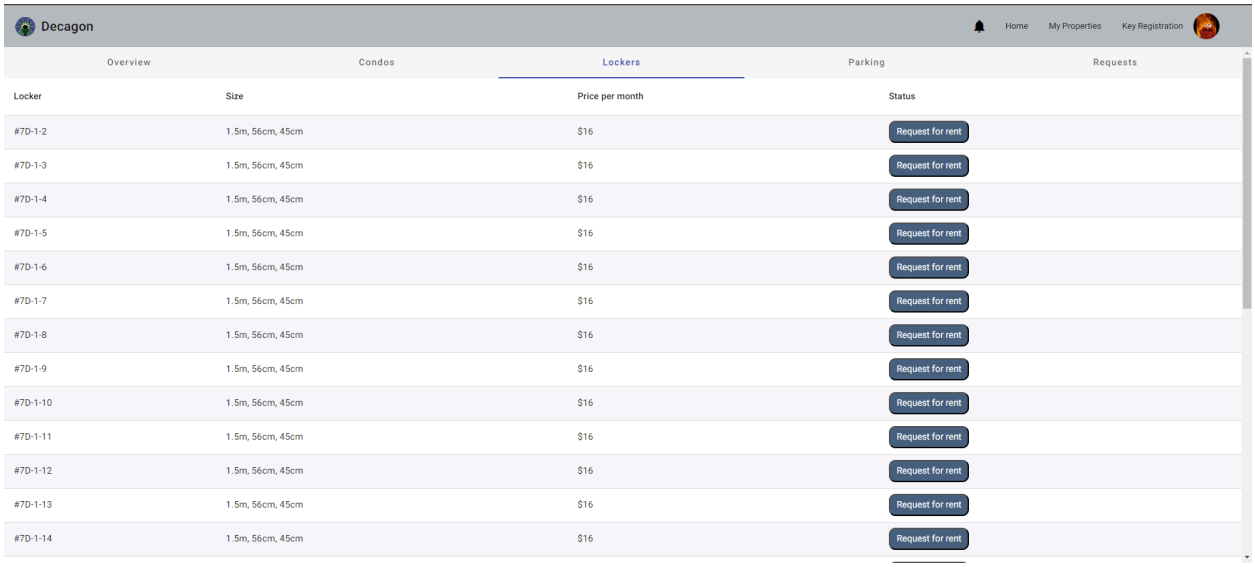
The notifications page



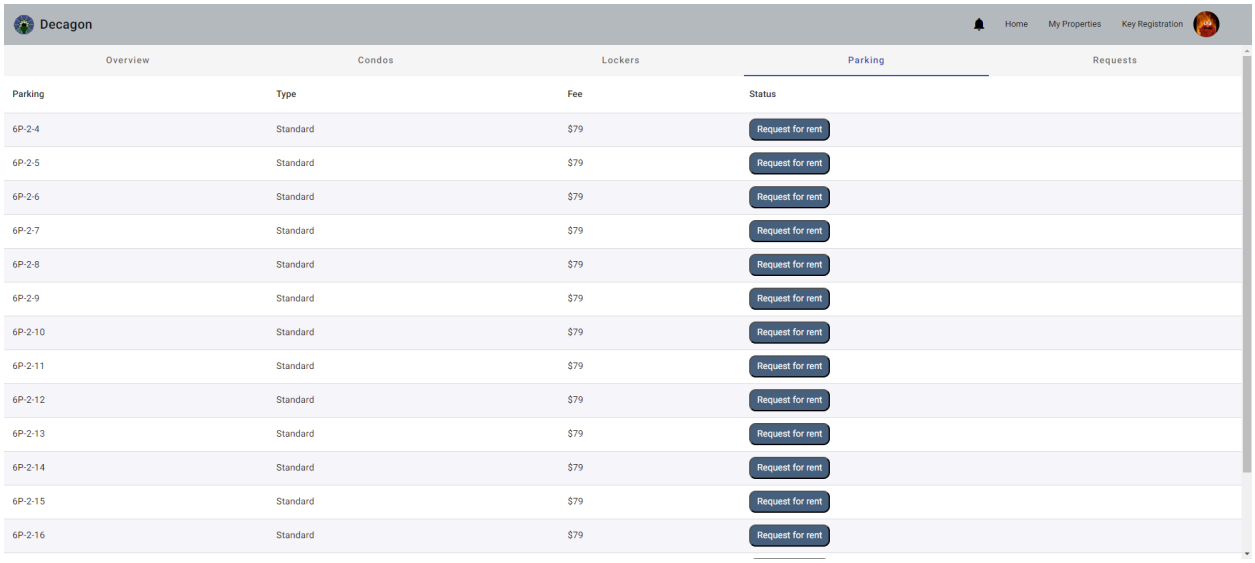
Requesting a condo unit



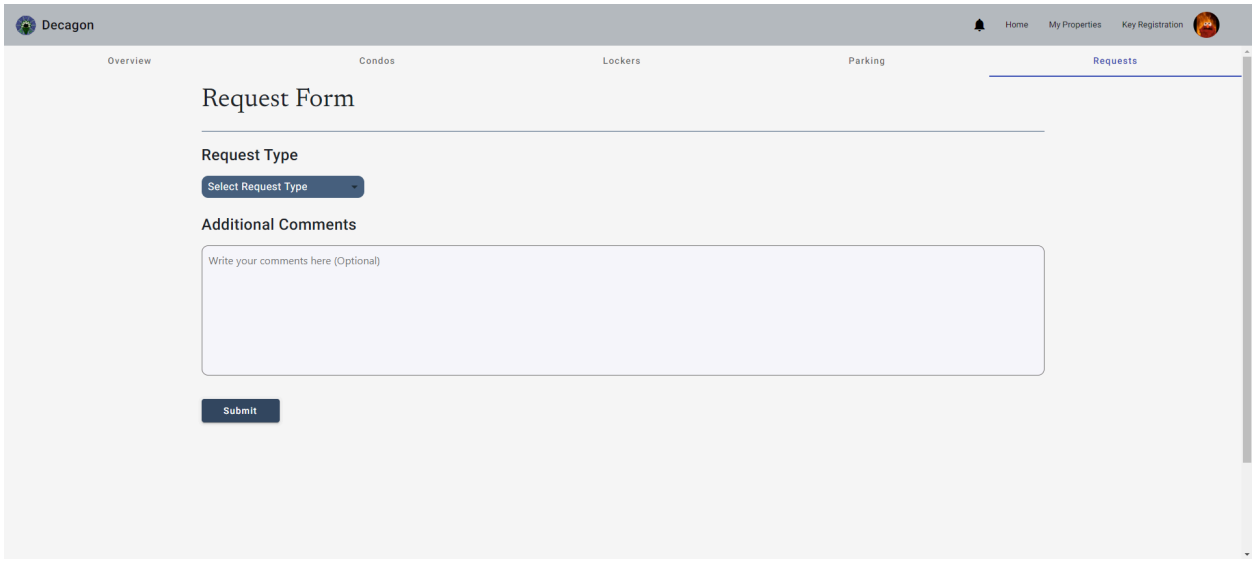
Requesting a locker



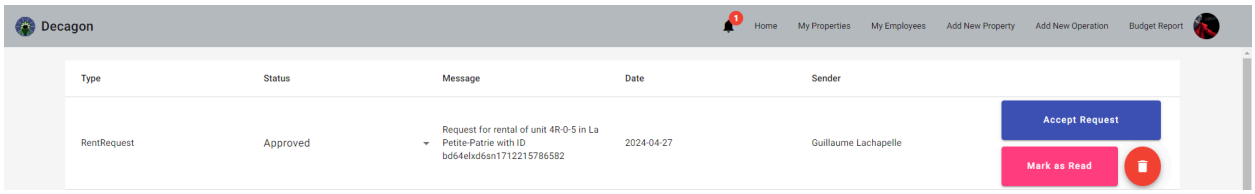
Requesting a parking spot



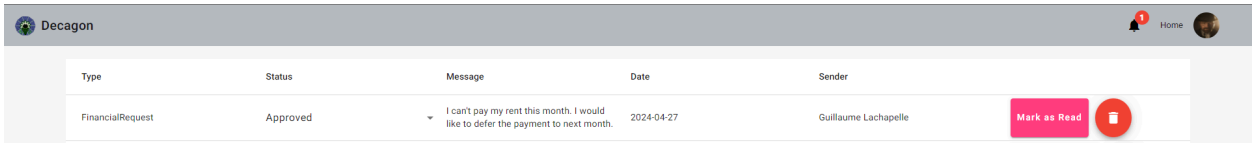
The requests page



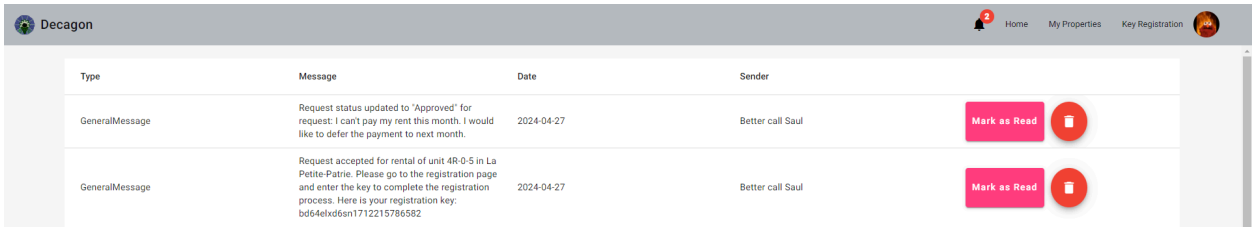
Company user receiving requests from public users



Employees getting requests from public users



Public users getting status updates on their requests



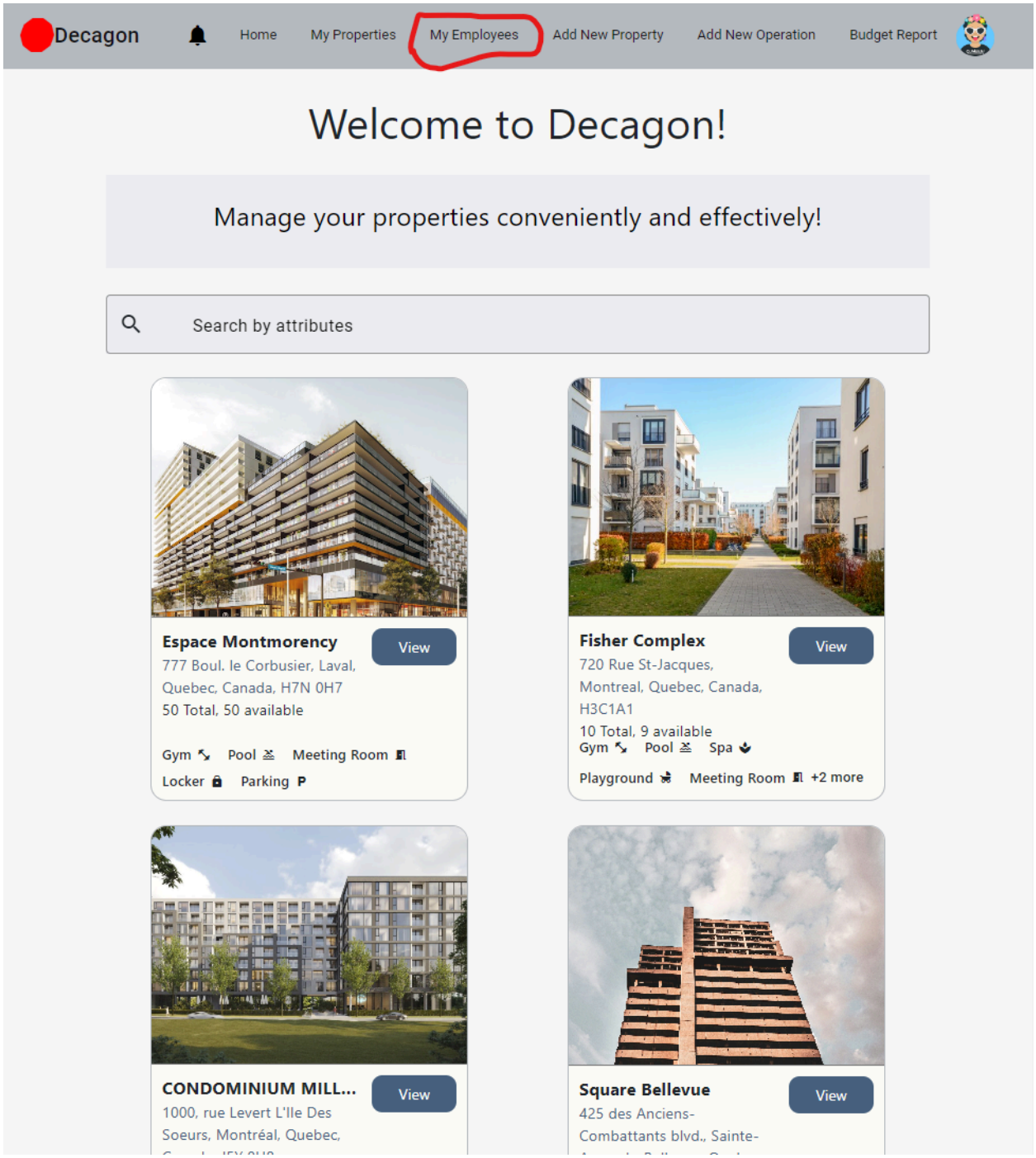
User Acceptance test can be found under the name:RequestsAndNotifications.cy.ts

3.7 Requirement #7

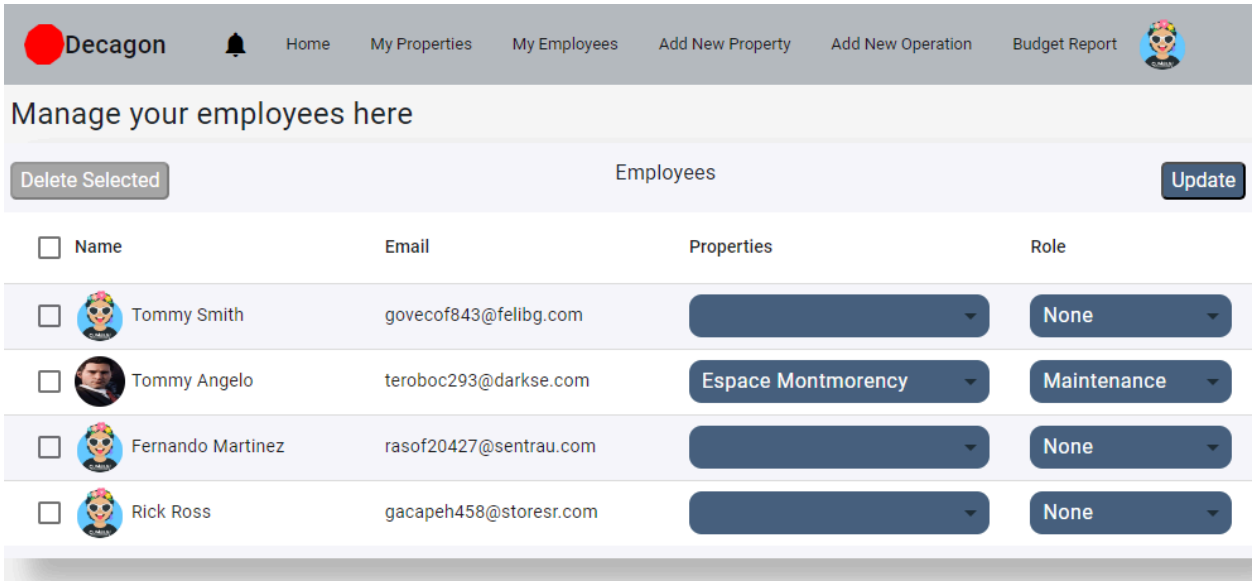
As a condo management company, I would like to set up different roles for different employees in order to detail my staff.

The employee page requirement is part of user story #54. This requirement was started in Sprint 2 with the front end developed. It was finished in Sprint 3 with the backend functionality implemented. It allows a company owner to assign different roles to their employees and assign them to one or more buildings. They can also remove employees. To access this page, the user must be logged in as a company user type (company owner).

Location of Employee Page button in the header

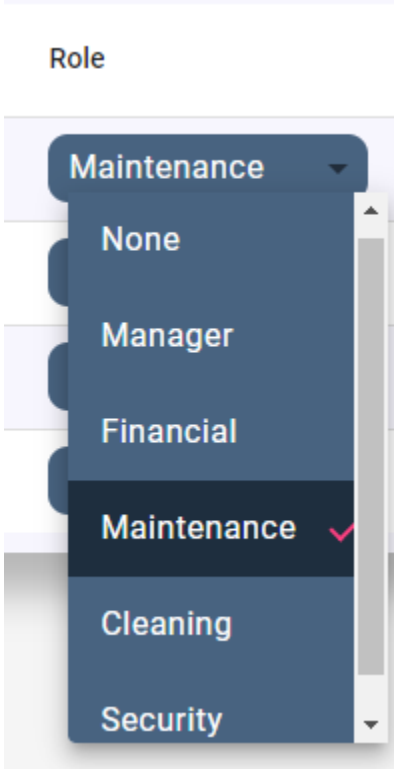


The My Employees Page

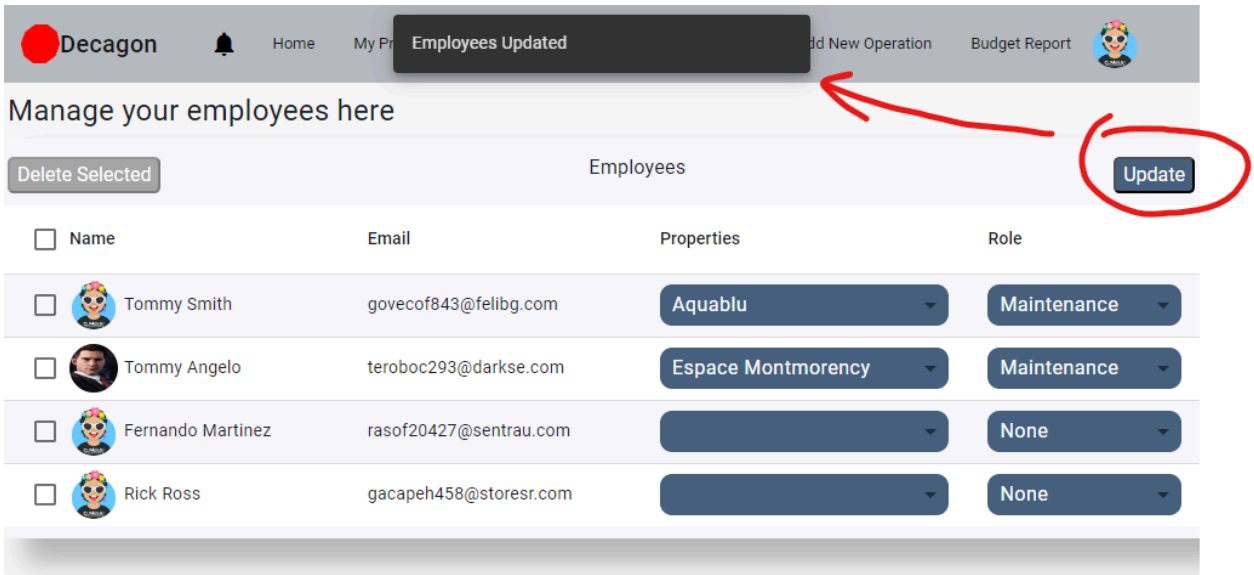


Shows list of buildings owned by the company to assign to employee

Assign Employee Role

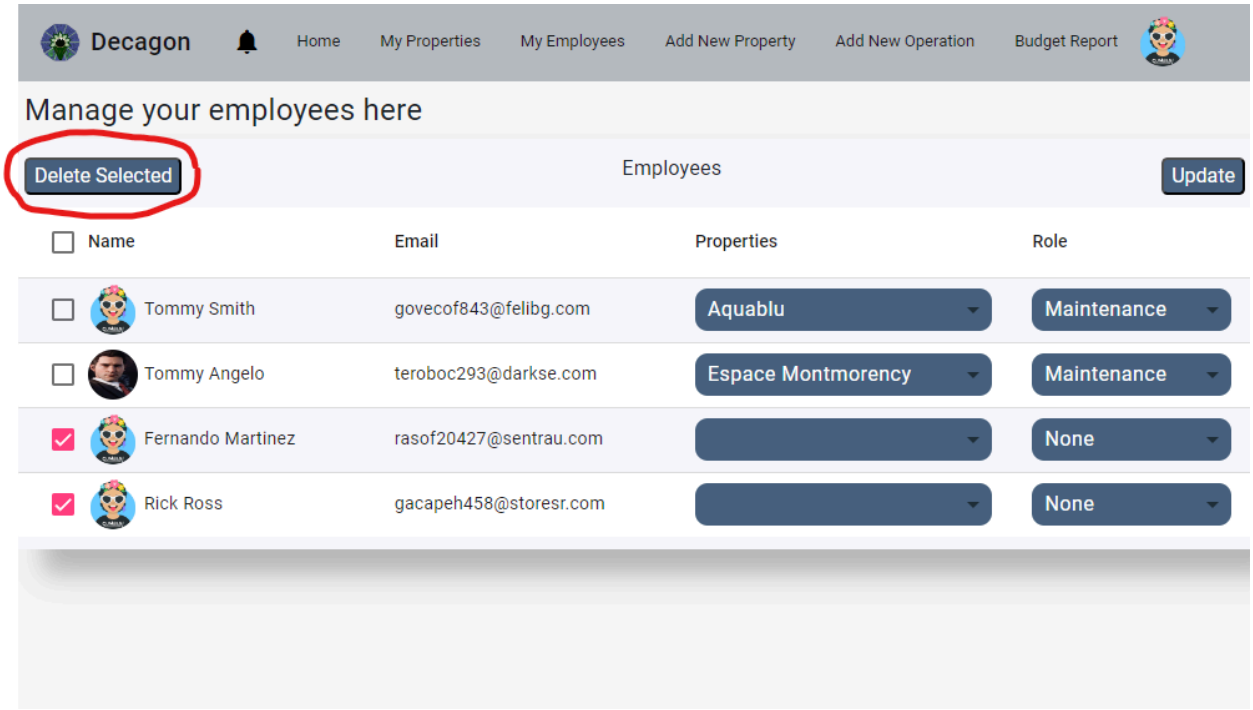


Click Update to save changes



Employees will now receive user requests based on their assigned roles and properties.

Can also select employees to remove



User Acceptance test can be found under the name: MyEmployees.cy.ts

3.8 Requirement #8

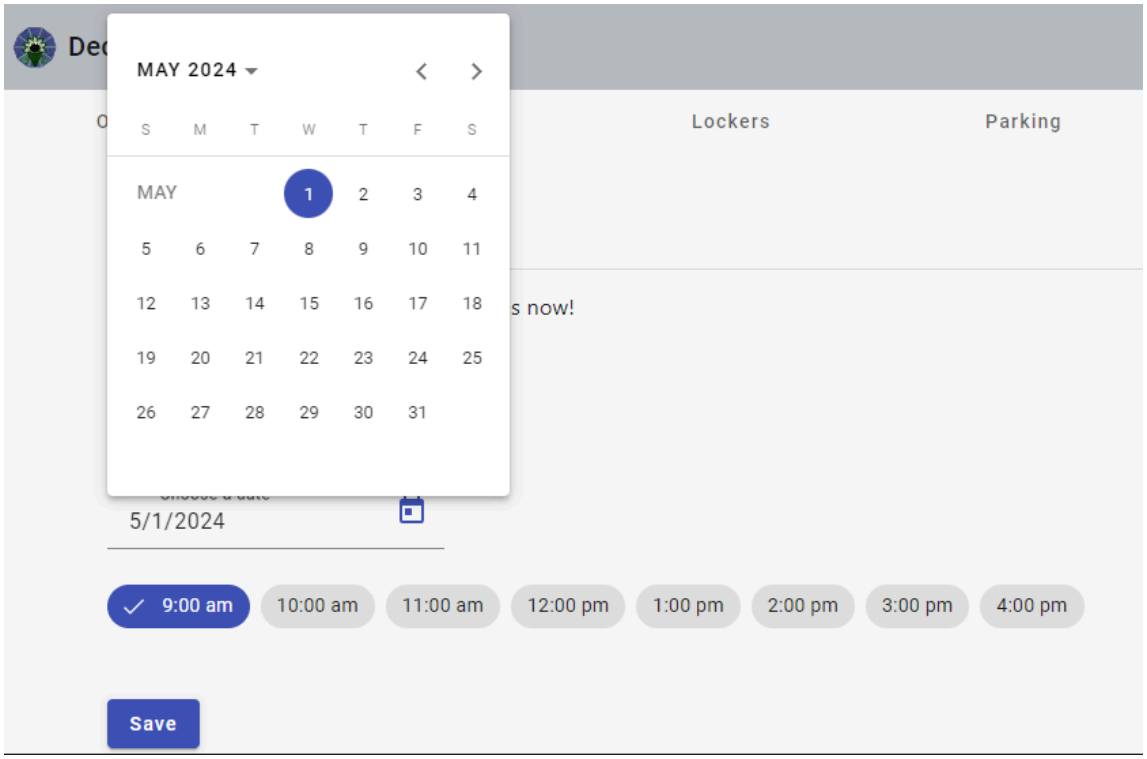
As a condo owner or renter, I would like to book rooms available in the different complex building I own or rent to easily access the building facilities.

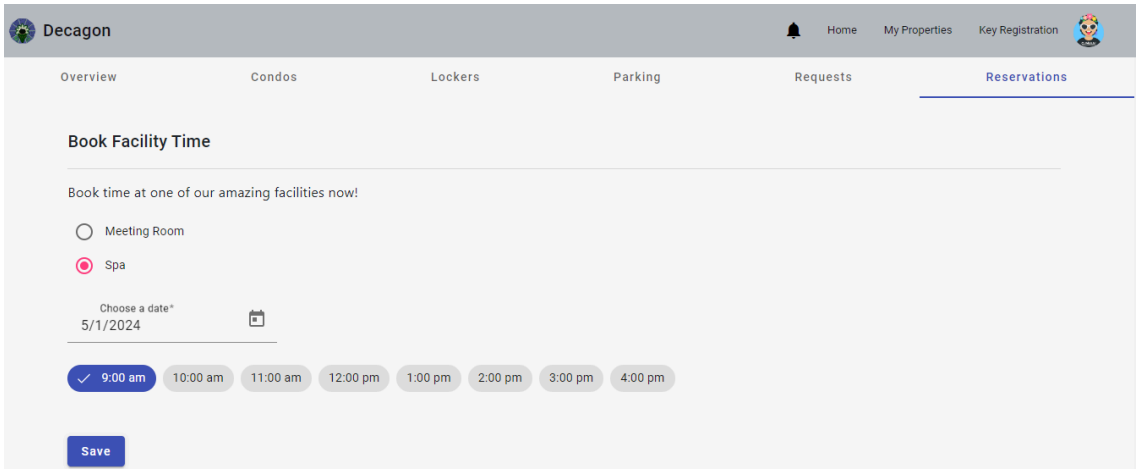
The Reservations tab is only accessible to public users who own or rent a condo in a given building. Inside the Reservations tab, a condo owner or renter is able to book time at a facility in the building. If another condo owner or renter has already booked a time at a given facility in a building, no other condo owner or renter will be able to book the facility at that time. Given a building, the Schedule tab is only accessible to company users who manage that particular building. Once a condo owner or renter books time at a facility in a building, the booking will appear on the page and companies will be able to see an updated schedule (with the new booking) where they can see all bookings.

The Reservations tab and the Schedule tab on the building information page are part of user story #18. The front-end of the booking form (inside the Reservations tab) was originally placed in the building overview tab in Sprint 2. However, the decision to move the booking form to its own tab was made in Sprint 3. Thus, the front-end of the Reservations tab was officially completed in Sprint 3. The front-end of the Schedule tab

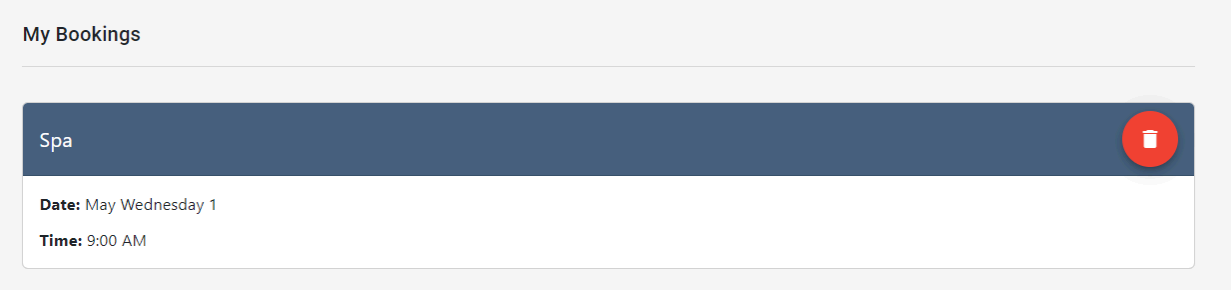
was also completed in Sprint 3. The back-end of the Reservations tab and the Schedule tab were both completed at the end of Sprint 4. The test for the Reservations tab (i.e.: to book time at a facility) can be found here: ReservationsAndBookings.cy.ts. The test to view the schedule can be found here: SchedulePageCompany.cy.ts. Both Cypress tests pass.

In the two screenshots below, a condo owner or renter is making a booking at the Spa on May 1, 2024, at 9:00 am.

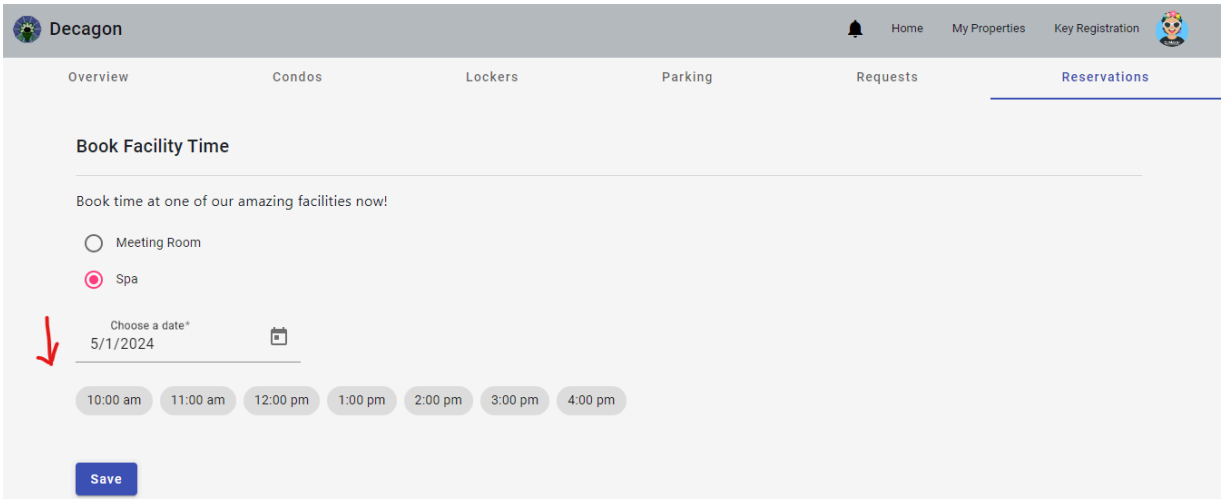




After the condo owner or renter has made the booking, the booking appears beneath the form in a section called “My Bookings” (seen in the screenshot below).

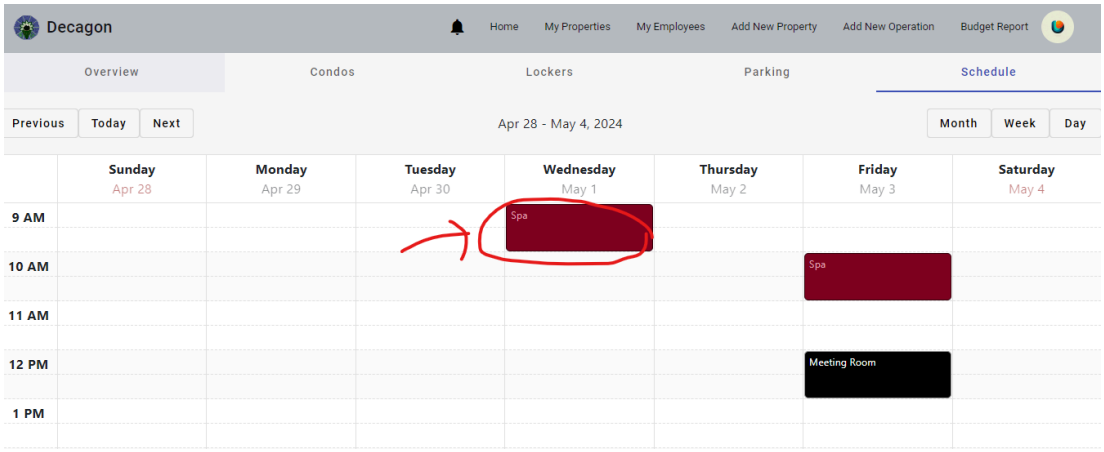


Notice in the screenshot below how the 9:00 am time slot is not there anymore because someone already booked time at the Spa on May 1, 2024, at 9:00am.

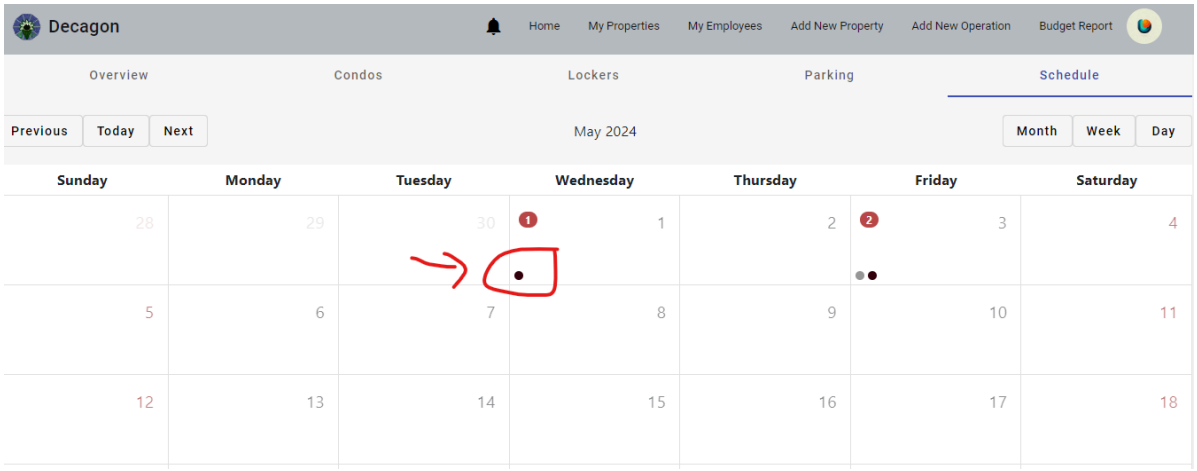


In the screenshots below, a company user who manages that building can see the new booking (circled in red) and see other bookings as well:

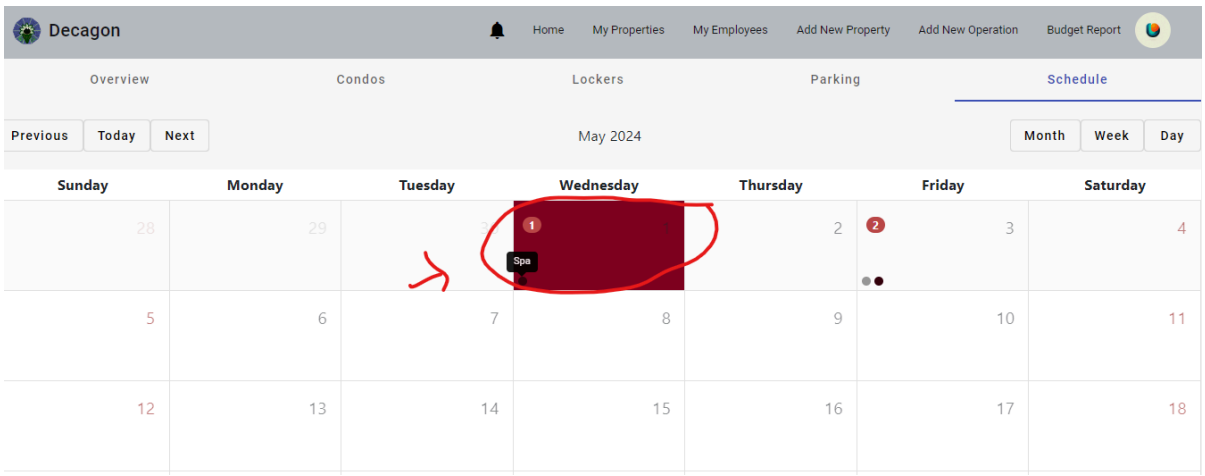
Week View of Schedule:



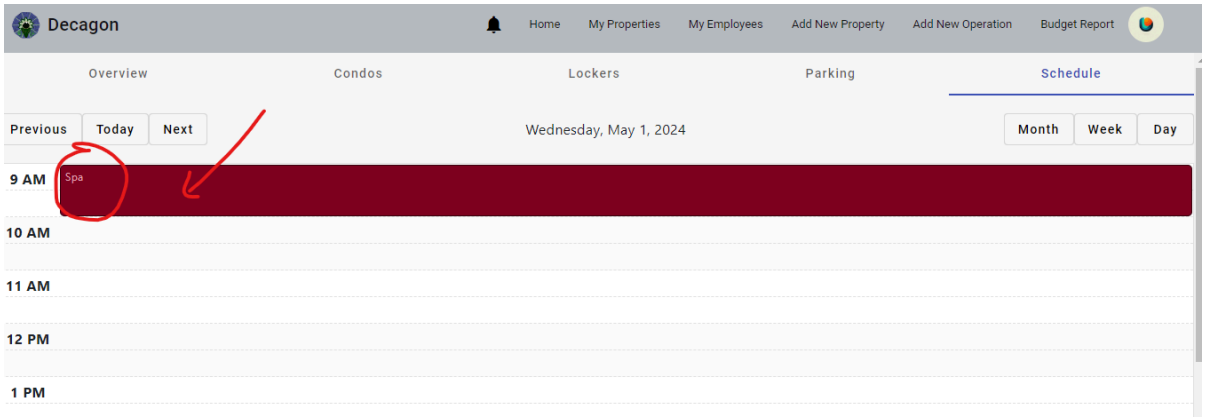
Calendar View (without hovering over booking) of Schedule:



Calendar View (whilst hovering overbooking/dot) of Schedule:



Day View of Schedule:



3.9 Requirement #9

As a condo manager, I would like to view a simplified financial system to easily generate important information.

During Sprint 3, our team dedicated efforts towards implementing a pivotal component of the Decagon application—a simplified Financial System, with a primary focus on the Budget Report component. This endeavour aimed to encapsulate essential financial data into an easily digestible format for condo managers.

User Acceptance test can be found under the name: NewBuildingOperation.cy.ts, and BudgetReport.cy.ts

Budget Report Page and Accessibility

The Budget Report component, as seen in the image below, boasts a pristine and uncluttered interface, inviting immediate attention to the critical financial metrics. It displays a table with columns for 'Building', 'Condo Fee Revenue', 'Operation Costs', and 'Profit', offering a transparent view of financial standings across different properties. Each row corresponds to a specific property, presenting the condo fee revenue against it. This revenue is reflective of the occupancy status of the condos within each building.

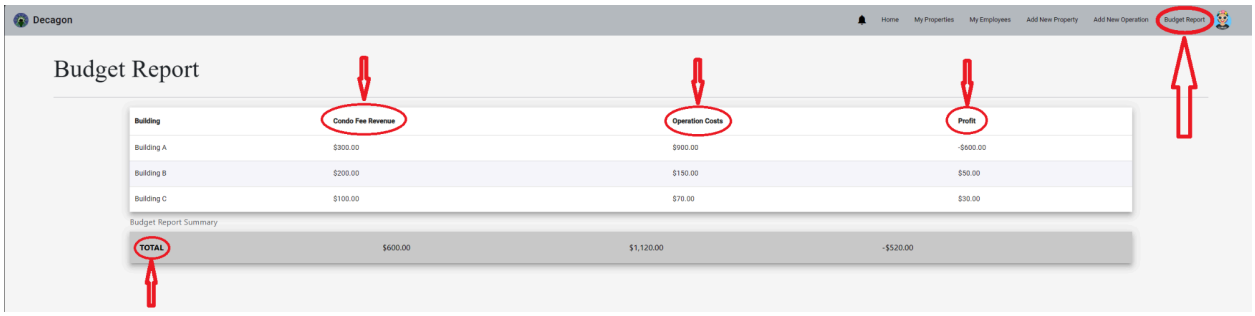
Operational Insight: The Operation Costs column enumerates the expenses incurred for maintenance and other activities reported within the sprint duration, ensuring a comprehensive reflection of the buildings' upkeep expenditures.

Profit Calculation: The component features an automated calculation where the Operation Costs are subtracted from the Condo Fee Revenue to indicate the net Profit for each building, aiding managers in financial assessment and strategic decision-making.

Consolidation: At the foot of the component lies the Budget Report Summary, tallying the total Condo Fee Revenue, Operation Costs, and overall Profit, thus providing a quick, at-a-glance financial performance summary of all managed properties.

Ease of Use and Accessibility:

The layout of the Budget Report component is crafted with a user-centric approach, ensuring that all financial data is immediately accessible without the need to navigate through multiple pages or layers. Profit margins are distinctively highlighted, allowing managers to quickly discern financial health and operational effectiveness. The Budget Report component is strategically positioned for optimal accessibility. As the rightmost tab in the application's header, indicated in the provided screenshot, it is designed to be conveniently accessible, standing as a testament to our user-centric development approach. Reflecting our understanding of user roles and privacy concerns, this crucial financial tool is exclusive to condo managers, ensuring they have the dedicated resources needed to perform their roles effectively.



The Budget Report component underwent thorough testing to validate its functionality and usability including the NewBuildingOperation.cy.ts and the BudgetReport.cy.ts cypress tests (that ran and that were fully passed). This ensures that the tool not only meets the initial requirements but also provides an error-free experience for the end user.

Streamlined Financial Management: This component stands as a central tool in Decagon's suite, empowering condo managers with the ability to monitor and manage financial operations efficiently.

Supports Informed Decisions: By providing an easily interpretable financial snapshot, the Budget Report component aids managers in making informed decisions that affect the profitability and operational sustainability of their properties.

3.10 Requirement #10

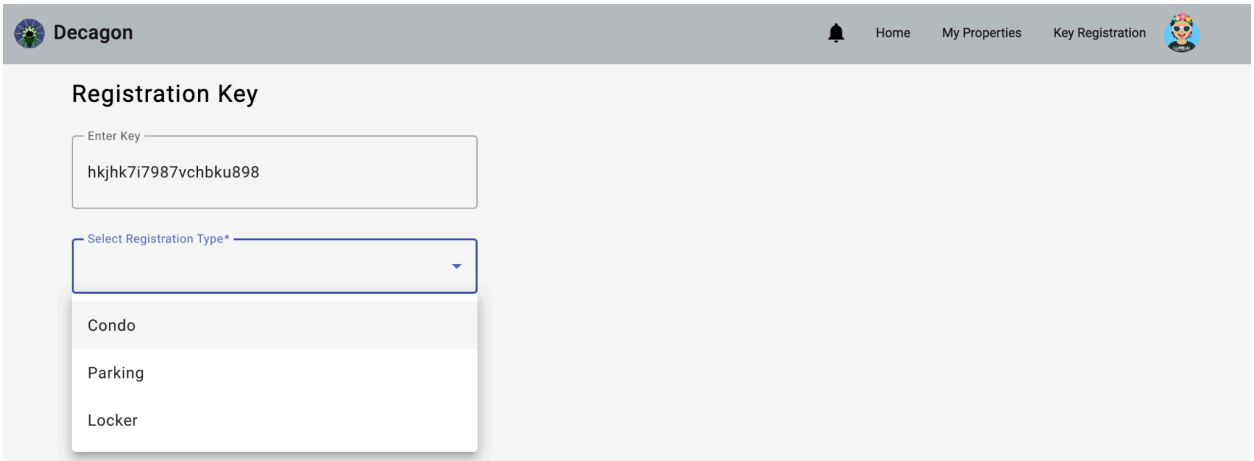
As a condo owner or renter, I would like register my account with units, lockers and parkings so that my account can be properly associated.

The implementation of this feature marks a significant milestone in enhancing the user experience for condo owners or renters within our system. This user story focused on enabling users to register their accounts seamlessly with units, lockers, and parking spaces, ensuring a comprehensive association between their accounts and the physical amenities they utilize. Through a series of front-end and back-end tasks, coupled with rigorous user acceptance testing, our team successfully delivered a robust solution that enhances user interaction and account management.

User Interaction: Upon accessing the registration page, users are prompted to provide a unique key issued by the building owner or manager, ensuring authentication and authorization. This key serves as a secure gateway to associating their account with specific units, lockers, or parking spaces within the condominium complex. The user interface simplifies this process by offering a dropdown menu where users can conveniently select the type of amenity they wish to register, whether it be a unit, parking space, or locker. Once the user selects the desired type, they proceed to submit the information, initiating the registration process.

Step 1

The user provides the key to the desired unit.



Step 2

The user chooses the type of the unit and registers.

Backend Implementation: Behind the scenes, the back-end infrastructure handles the processing and validation of user inputs, ensuring accuracy and integrity throughout the registration workflow. The implementation of registering units, parking spaces, or lockers to the user's account securely involved data validation, authentication checks, and database operations to establish the association accurately. Similarly, the back-end implementation focused on handling requests related to unit, parking, or locker registration, ensuring seamless communication between the front-end and back-end systems.

User Acceptance Testing: To guarantee the reliability and functionality of the registration process, our team meticulously crafted user acceptance tests (UATs) to validate each aspect of the user story. The creation of UATs for registering units, parking spaces, and lockers with an account, covering various scenarios and edge cases to simulate real-world usage accurately, was paramount. This testing framework extended to encompass the user's ability to request units, lockers, or parking spaces, ensuring that the entire registration and request workflow met user expectations and system requirements.

.

User Acceptance test can be found under the name: Key-Registration.cy.ts

The successful implementation of this feature demonstrates our commitment to delivering user-centric solutions that streamline account management for condo owners and renters. By integrating seamless user interactions, robust backend functionality, and rigorous testing practices, we have enhanced the usability and reliability of our platform, setting a strong foundation for future enhancements and improvements.

4. User Interface demonstration:

For a visual walkthrough of the user interface meeting the above requirements, please refer to the following link **https://youtu.be/ODVT2vB41_w**

5. Testing:

In the Testing section, we will outline the testing strategies and activities for each sprint, including unit tests, and cypress tests (covering acceptance, integration, and system tests). Moreover, the code coverage percentage and the code quality.

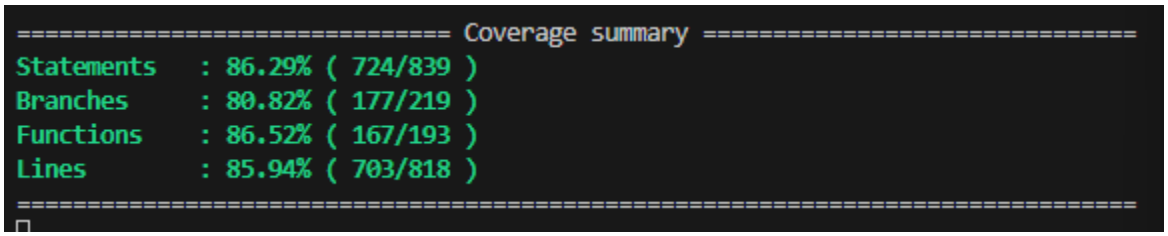
For our project, our team made use of the Jasmine testing framework installed by Angular for unit testing. Cypress was used throughout the project for integration testing, system testing and user acceptance testing.

5.1 Sprint 1:

In Sprint 1, user acceptance tests and unit tests were done for two tasks: Create a Profile Page and Login.

The unit tests and cypress tests were configured to run on GitHub automatically after every push to a branch and every pull request. The test run is configured to fail if the statement coverage is less than 80% or if one of the Cypress tests fails.

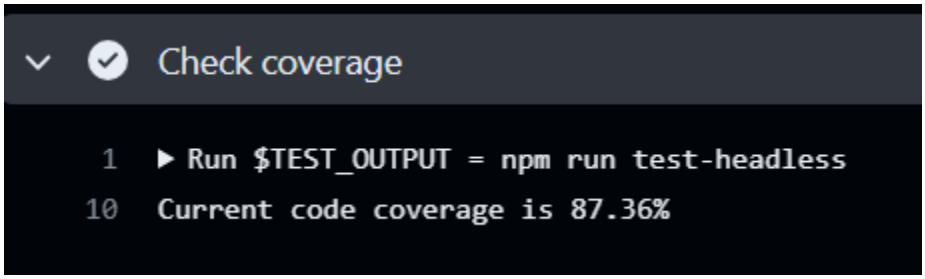
Coverage Example:



5.2 Sprint 2:

The focus of Sprint 2 was to implement the front-end of pages which were crucial for the efficient development of the project. As a result, only a few pages also implemented some back-end functionality. Unit tests were done for the pages that incorporated back-end functionality. However, no user acceptance tests were done in Sprint 2 since the focus of Sprint 2 was on front-end web development and many of the necessary back-end functionalities which were necessary to adequately execute user acceptance tests were still missing.

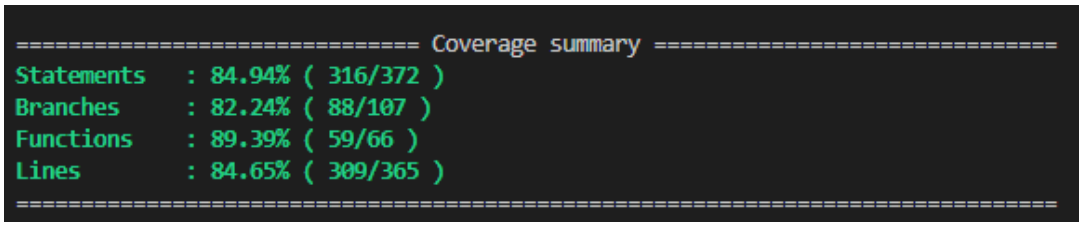
Coverage at the end of sprint 2:



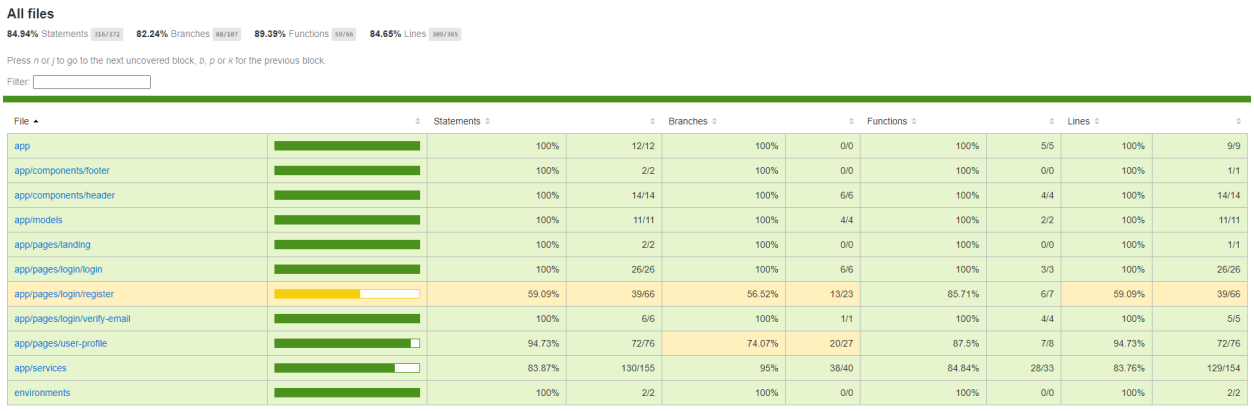
5.3 Sprint 3:

In Sprint 3, unit tests were done for all back-end tasks. Additionally, user acceptance tests were done for several tasks. Namely, for the display of properties (buildings, condo units, parking and lockers), for the editing of the profile page, for the employees' page and the notifications page.

Code coverage at the end of Sprint 3:



Visualization of coverage:



All files

84.94% Statements 316/372 82.24% Branches 88/107 89.39% Functions 59/66 84.65% Lines 309/365

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

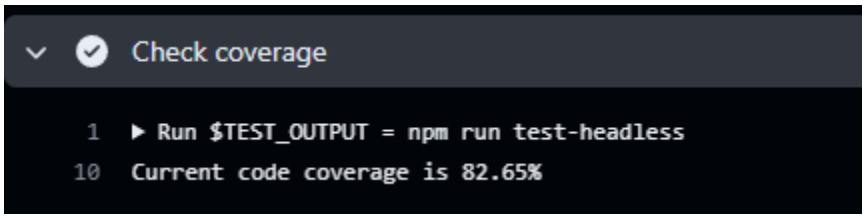
Filter:

File		Statements		Branches		Functions		Lines	
app		100%	12/12	100%	0/0	100%	5/5	100%	9/9
app/components/footer		100%	2/2	100%	0/0	100%	0/0	100%	1/1
app/components/header		100%	14/14	100%	6/6	100%	4/4	100%	14/14
app/models		100%	11/11	100%	4/4	100%	2/2	100%	11/11
app/pages/landing		100%	2/2	100%	0/0	100%	0/0	100%	1/1
app/pages/login/login		100%	26/26	100%	6/6	100%	3/3	100%	26/26
app/pages/login/register		59.09%	39/66	56.52%	13/23	85.71%	6/7	59.09%	39/66
app/pages/login/verify-email		100%	6/6	100%	1/1	100%	4/4	100%	5/5
app/pages/user-profile		94.73%	72/76	74.07%	20/27	87.5%	7/8	94.73%	72/76
app/services		83.87%	130/155	95%	38/40	84.84%	28/33	83.76%	129/154
environments		100%	2/2	100%	0/0	100%	0/0	100%	2/2

5.4 Sprint 4:

In Sprint 4, unit tests were done for all back-end tasks (as per usual). User acceptance tests relating to the requests page, finances, the individual condo page and registering a property to a unit/parking/locker were completed.

Code coverage at the end of Sprint 4:

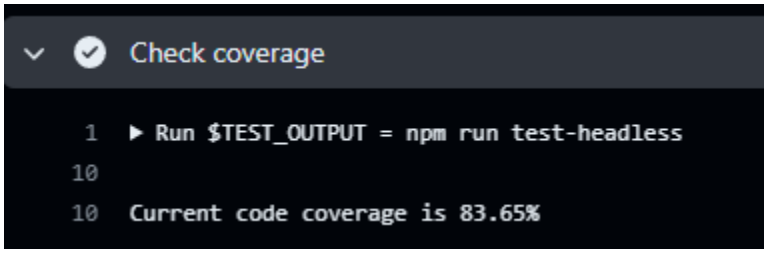


Code Duplication at the end of Sprint 4:

5.5 Sprint 5:

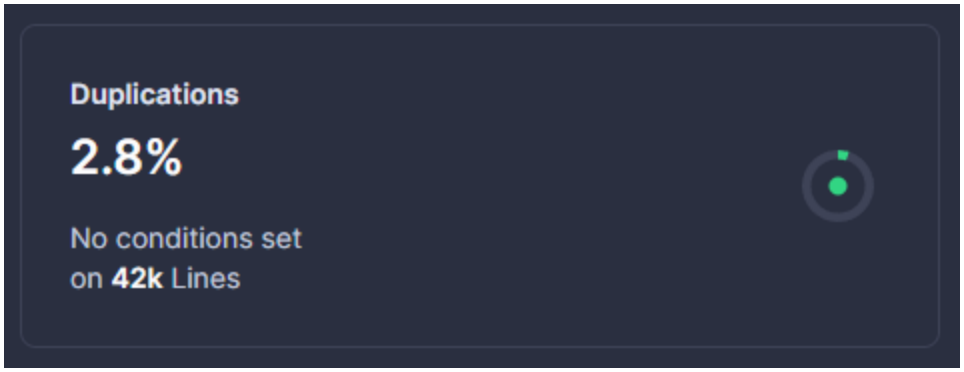
Most of the testing was completed in previous sprints. In Sprint 5, only two user acceptance tests remained. Both user acceptance tests related to the reservations/schedule page which had to do with requirement #10. All other forms of testing were completed prior to Sprint 5.

Unit test code coverage is **83.65%** on the most recent commit



Cypress tests (Acceptance / Integration / System tests)

Sonarcloud (Code Quality)
Code duplication on the main branch



6. Retrospective:

In the retrospective section, we will provide a summary of the key outcomes and learnings from each sprint, highlighting successes, challenges, and areas for improvement.

6.1 Sprint 1:

In the retrospective, several challenges and successes were identified:

6.1.1 Challenges:

1. **Testing Tool Confusion**: Initial attempts to utilize SonarQube/Cloud for test coverage feedback were confusing and misleading, affecting the team's understanding of testing gaps. This was addressed by integrating coverage into the pipeline.
2. **Access Issues with Firebase:** Access problems with Firebase hindered backend development progress, leading to confusion among team members. A resolution was agreed upon to contact the project owner promptly.
3. **Task Documentation:** Uncertainty surrounding task and milestone documentation slowed progress, but clarification from the TA improved understanding and streamlined the process.
4. **Risk Assessment and Analysis:** Confusion around the required documentation for risk assessment prompted clarifications from the professor, highlighting the importance of earlier communication within the team.

6.1.2 Successes:

1. **Division of Tasks:** Early task delegation facilitated immediate work commencement, with plans for regular task division discussions post-demo.

2. **Collaborative Team Effort:** A culture of collaboration and open communication enhanced problem-solving and team satisfaction, supported by regular team-building activities and discussions.
3. **Software Architecture:** Clear software architecture diagrams aided understanding and will guide implementation, with the potential for more frequent reviews to ensure alignment as development progresses.
4. **Completion of Sprint 1 Tasks**: All Sprint 1 tasks were completed on time due to timely and efficient task execution, supported by effective task division and teamwork.

6.2 Sprint 2:

In the retrospective, several challenges and successes were identified:

6.2.1 Challenges:

1. **Task Focus:** Initially, the team aimed to focus on specific user stories from start to finish, leading to difficulties in managing frontend and backend tasks simultaneously within the sprint timeframe. This resulted in some tasks being rushed and impacted their effectiveness.
2. **Implementing an Effective Map:** Developing an interactive map posed challenges, particularly in converting addresses into coordinates for map markers. However, persistent effort and exploration of geocoding techniques eventually led to successful implementation.
3. **Redefining Planned Tickets:** Towards the end of the sprint, it became evident that certain user stories were overestimated, leading to time-consuming reevaluation and redefinition of tasks to align with requirements and scope. Creating UI prototypes proved instrumental in this process.

6.2.2 Successes:

1. **Feedback:** Recognizing the importance of understanding previous sprint challenges, the team organized a meeting with the TA (product owner) to review and address documentation process improvements. This proactive approach mitigated potential delays in subsequent sprints.
2. **Prototyping:** The creation of high-quality prototypes significantly facilitated frontend development, streamlining the process and saving time and effort. Detailed prototypes enhanced visualization and decision-making, contributing to faster development cycles.

6.3 Sprint 3:

In the retrospective, several challenges and successes were identified:

6.3.1 Challenges:

1. **Ambitious Workload:** The team found the sprint's workload to be overly ambitious, resulting in tasks being pushed to the backlog, particularly user acceptance tests that required backend functionality not yet implemented.
2. **Merge Conflicts:** The sprint saw a high volume of changes and pull requests simultaneously, leading to numerous merge conflicts that consumed significant time to resolve. Mitigation strategies included team meetings, code review, and post-merge testing.
3. **Code Management Documentation:** There was a lack of clarity regarding the requirement for producing documentation for code management, communicated late in the sprint. While information had been collected for a PowerPoint presentation, separating and organizing this documentation proved challenging due to time constraints.

6.3.2 Successes:

1. **Good Task Distribution:** Recognizing the issue of task dependencies and overwhelming workload, adjustments were made in Sprint 3 to prioritize front-end components first and tackle back-end tasks only after front-end issues were resolved. This approach improved efficiency and reduced wait times between tasks.
2. **Good Team Collaboration and Support:** Strong collaboration and support among team members characterized by regular meetings, progress updates, and willingness to assist with code-related challenges. Team members actively supported each other and provided valuable insights to solve problems.

6.4 Sprint 4:

In the retrospective, several challenges and successes were identified:

6.4.1 Challenges:

1. **Longer Development for Spec Tests:** Despite fewer issues in this sprint, the development of specification tests for some components proved time-consuming due to the complexity of logic in TypeScript files. With over 300 tests, running

and verifying these tests became a significant time investment, especially towards the end of the project.

2. **Code Refactoring due to Duplications:** Completion of acceptance tests for individual condo pages revealed over 60% code duplications, necessitating code refactoring to address potential maintainability, scalability, and performance issues. This process required additional time and resources to execute effectively.
3. **Data Loss in Firebase:** An unfortunate incident occurred during this phase wherein an important part of the database was lost, likely during the execution of tests. This setback highlighted the importance of careful testing procedures to prevent data loss and required considerable time to rectify.

6.4.2 Successes:

1. **Lighter Workload:** This sprint was perceived as having a lighter workload compared to previous sprints, making it easier for team members to complete their tasks. Deliberate efforts were made to reduce the workload, particularly considering the heavy workload from other classes during this period.

6.5 Sprint 5:

In the retrospective, several challenges and successes were identified:

6.5.1 Challenges:

1. **Changes in Format when Using Capacitor:** During Sprint 5, transitioning the project into a mobile app using Capacitor introduced slight CSS adjustments. While these changes were not critical, they added extra tasks that felt challenging to complete amidst finals and deadline pressures.
2. **Report Complexities:** Composing the final report proved challenging due to unclear requirements and the absence of a sample document for reference. Determining the report's content without redundancy posed difficulties for the team.

6.5.2 Successes:

1. **Ease of Use with Capacitor:** Despite minor format issues, Capacitor proved to be an easy-to-use tool for mobile app development. Following a few directives, the app could seamlessly run on mobile devices, saving significant time compared to alternative techniques.
2. **Completion of Front/Back-End Development:** With all programming tasks completed besides testing, the workload for Sprint 5 was eased, allowing the team

to focus on documentation and finalizing deployment. This achievement streamlined the project's progress towards completion.