SOEN 390 - Software Engineering Team Design Project

Team 18

Deliverable 3 (Sprint 1)

Architecture Description of Concordia Condo Management System (CCMA) for Condominium Administrative and Community Engagement Platform

Winter 2024

Student Name	Student ID
Jeremy Rimokh	40110746
Ishaï Cohen	40188880
Eden Oliel	40211989
Alexander El Ghawi	40200062
Joud Babik	40031039
Minh Duc Vu	40166824
Nicholas Werugia	40131956
Yash Patel	40175454
Patrick MacEachen	40209790
Vithujanan Vigneswaran	40157822

Contents

List of Figures

List of Tables

1 Introduction

This chapter describes introductory information items of the AD, including identifying and supplementary information.

1.1 Identifying information.

The Architecture Description of Concordia Condo Management System (CCMA) for Condominium Administrative and Community Engagement Platform is a condominium management system, which encompasses an interactive web platform and a complementary mobile application. This digital ecosystem is designed to serve the administrative needs and community engagement of condominium residents and management, facilitating various functions such as facility booking, maintenance requests, and communication. As a software-intensive system, it integrates various software components and services to provide a seamless user experience and support the operational processes of condo management. The system is developed as an academic project under Concordia University, demonstrating the application of software architecture principles in real-world scenarios.

1.2 Supplementary information

Project Summary

This document outlines the development and implementation of a comprehensive condominium management website, designed to facilitate the efficient management of condo-related activities and services. The project is initiated under the auspices of Concordia University, supervised by Dr. Jinqiu Yang. The website aims to provide a centralized platform for condo residents, management, and staff to communicate, manage facilities, and access important information with ease.

Date of Issue: January 18th, 2023

Status: In Development

Authors: Listed in table 1

Reviewers: Lin Ling (Teaching Assistant) at Concordia university

Approving Authority: Lin Ling (Teaching Assistant) and Dr. Jinqiu Yang

Issuing Organization: Concordia University

Change History: This is the first version of the document submitted for sprint 1

Scope: the scope of the project is detailed below

Context: The project is developed as part of the coursework for SOEN 390, aiming to address the practical needs of condo management through a digital solution.

Version Control Information: Managed through GitHub repository "THE-390"

Configuration Management Information: Document versioning and updates are tracked via GitHub to ensure consistency and collaboration among development team members.

Project Scope

The scope of the condo management website includes:

- **User Managemen**t: Secure registration and login mechanisms for residents, condo management, and staff. Profiles for each user category with relevant permissions and access levels.
- **Communication Platform:** A forum or bulletin board for announcements, queries, and feedback between residents and the management.
- **Facility Booking and Management:** An interface for booking common facilities, such as meeting rooms, gym slots, and event spaces. Management can update facility status and availability.
- **Maintenance Requests:** A portal for residents to submit maintenance requests and track their status. Integration with management's workflow for efficient handling and resolution.
- **Document Repository:** A secure location for storing and accessing condominium documents, policies, and meeting minutes.
- **Mobile App:** To ensure accessibility and convenience, a mobile application will complement the website, providing users with on-the-go access to features and notifications.
- **Integration and Scalability:** The platform will be designed for easy integration with existing management tools and scalable to accommodate future requirements.
- **Version Control:** Development is managed through the GitHub repository "THE-390", accessible at https://github.com/JRB958/THE-390.git, ensuring a collaborative and transparent development process.

This project is designed to enhance the living experience of condo residents by streamlining management processes and fostering a connected community. Through careful planning and execution, it aims to set a standard for digital condo management solutions.

Student Name	Student ID
Jeremy Rimokh	40110746
Ishaï Cohen	40188880
Eden Oliel	40211989
Alexander El Ghawi	40200062
Joud Babik	40031039
Minh Duc Vu	40166824
Nicholas Werugia	40131956
Yash Patel	40175454
Patrick MacEachen	40209790
Vithujanan Vigneswaran	40157822

Table 1: List of all student authors.

1.3 Other information

The MVC-architected Condo Management System centralizes condominium operations, enabling user profile creation and property management through a streamlined dashboard interface. This AD serves as a roadmap for navigating the system's architecture, highlighting its core functionalities and the rationale behind key design choices.

Evaluations of the architecture affirm its capability to support the system's robust feature set while maintaining scalability and performance. Decisions made throughout the design process, particularly the adoption of MVC, were anchored in ensuring a modular, maintainable, and secure framework for the Condo Management System.

Interrelationships between the architecture's various views and models are clearly defined, ensuring a comprehensive understanding of the system's structure and operation for all stakeholders involved in the project.

Related requirements	The condominium management system shall provide a seamless interface for users to manage their property, communicate with management, book facilities, and submit maintenance requests.
Alternative	Standalone desktop application.

Constraint	1. The system must be easily accessible without extensive installation procedures. 2. It needs to be compatible with various devices used by the condo residents and management staff.
Assumption	Most users have access to the internet and are more accustomed to using web and mobile applications rather than desktop applications.
Architecture Decision	
Identifier	#1
Description	The system will be developed as a cross-platform web application that is complemented by a mobile application for both Android and iOS devices.
Rationale	1. A web application does not require users to install software, making it instantly accessible via a web browser. 2. A mobile application ensures accessibility for users on-the-go and provides notifications directly to their smartphones. 3. Cross-platform compatibility ensures that regardless of the device (PC, tablet, smartphone), users can access the system with their preferred device. 4. The development team is proficient in web and mobile application development, which aligns with the skill set required for the project.

Table 2: Architecture Decision 1 - Overall Architecture

Related requirements	The system must be highly available, scalable, and capable of handling varying loads without service degradation.
Alternative	On-premises hosting solution.
Constraint	1. Limited physical infrastructure resources. 2. Need for a scalable solution that can adapt to fluctuating usage patterns.

Assumption	The user base will grow, and the system will require scaling in response to the increased demand.
Architecture Decision	
Identifier	#2
Description	Implement the web application on a cloud-based infrastructure using a provider such as AWS, Azure, or Google Cloud.
Rationale	1. Cloud providers offer on-demand resource scaling to handle increased traffic without physical hardware constraints. 2. High availability configurations provided by cloud services ensure minimal downtime and reliable user access. 3. Cloud services include disaster recovery capabilities, securing data and services against potential incidents.

Table 3: Architecture Decision 2 - Cloud-Based Infrastructure

Related requirements	The system must separate the data model, user interface, and control logic to facilitate maintainability and scalability.
Alternative	Monolithic architecture.
Constraint	1. The team's familiarity with the MVC pattern. 2. The need for a clear separation of concerns to simplify future updates and maintenance.
Assumption	Development using MVC will improve code quality and make future enhancements easier and more manageable.
Architecture Decision	
Identifier	#3
Description	Employ an MVC (Model-View-Controller) architecture to structure the application into logical chunks for better manageability and separation of concerns.

Rationale	1. MVC architecture enhances the ability to test components in isolation, improving overall system reliability. 2. It provides a clear framework for development, making it easier for new developers to understand the codebase. 3. Separation of concerns facilitates easier maintenance and scalability of the application over time.

Table 4: Architecture Decision 3 - MVC Architecture

Related requirements	The user interface must provide a fluid and interactive experience, minimizing page reloads and delay.
Alternative	Traditional multi-page web application.
Constraint	1. The need for an application-like user experience within a web browser. 2. Ensuring compatibility across all modern browsers.
Assumption	Users expect a fast and seamless interaction with the web application, akin to native desktop or mobile applications.
Architecture Decision	
Identifier	#4
Description	Develop the web interface as a Single Page Application (SPA) using a JavaScript framework like React.
Rationale	1. SPAs offer a more dynamic and responsive user experience by only reloading parts of the page in response to user actions. 2. This approach minimizes server round trips, reducing the overall latency of the application. 3. It allows for the application's UI to be more responsive and provides the end-user with a smooth and uninterrupted experience.

Table 5: Architecture Decision 4 - Single Page Application (SPA)

Related	The system must enable integration with various

requirements	services and client applications while adhering to industry standards.
Alternative	SOAP-based web services.
Constraint	1. The need for a stateless architecture that can handle numerous clients simultaneously. 2. Requirement for a scalable and flexible backend service that can evolve over time.
Assumption	RESTful services are widely accepted and understood, making them a default choice for web API development.
Architecture Decision	
Identifier	#5
Description	Implement a RESTful API design for backend services, facilitating ease of integration and adherence to web standards.
Rationale	1. RESTful APIs use standard HTTP methods, making them easy to implement and consume by various clients. 2. They support a stateless communication mechanism, which is ideal for web-based services where numerous clients may interact with the system concurrently. 3. RESTful APIs provide a flexible way to deliver data in multiple formats, such as JSON or XML, accommodating different client needs.

Table 6: Architecture Decision 5 - RESTful API Design

Related requirements	The development process must be efficient, with a robust framework for both frontend and backend services, and the database must handle document-oriented operations effectively.
Alternative	Use of alternative frameworks such as Angular for the frontend, Flask for the backend, and SQL databases for data storage.
Constraint	1. The need for a modern, component-based frontend framework. 2. Requirement for a high-performance, scalable

	backend framework that complements the frontend. 3. The database should support rapid development and complex hierarchies.
Assumption	The team has experience with React and Django, and the system requirements include handling flexible, schema-less data.
Architecture Decision	
Identifier	#6
Description	The web application will utilize React for the frontend to build a component-based user interface, Django as the backend framework for robust and scalable web services, and MongoDB for a flexible, schema-less database.
Rationale	1. React's component-based architecture allows for reusable UI components, making the frontend scalable and easier to maintain. 2. Django offers a comprehensive set of tools for rapid

Table 7: Architecture Decision 6 - Use of Frameworks (React, Django, MongoDB)

1.3.2 Architecture evaluations

Tables 2 to 7 highlight evaluations and descriptions for Architectural decisions for sprint 1

1.3.3 Rationale for key decisions

Tables 2 to 7 highlight rationale for Architectural decisions for sprint 2

2 Stakeholders and concerns

2.1 Stakeholders and their Concerns

1. **Project manager**: A person that oversees the development of the system.
a. Scalability: Can the system handle a growing number of users and new updates ?

b. Feasibility: Are the resources (developers and budget) allocated to develop the system sufficient?
c. Feasibility: Can the system be deployed in a reasonable time?

2. **Product Owner**: A person that provides the directions about the system's purpose based on the users' needs and business goals.
a. Scalability: Can the system handle a growing number of users and new updates ?
b. Security: Can the system protect users from malicious users and threats ?
c. Flexibility: Can the system tolerate customization to meet end users needs?
d. Interoperability: Can the system support integration with other systems and services?

3. **Development Team**: A team consisting of people that will code and program the application.
a. Performance: Will the system be able to handle heavy traffic and offer a smooth experience?
b. Flexibility: Can the system tolerate customization to meet end users' needs?
c. Scalability: Can the system handle a growing number of users and new updates?
d. Testability: Can the system be tested without much effort?
e. Security: Can the system protect users from malicious users and threats?
f. Flexibility: Does the system run similarly on any device?

4. **Marketing Team**: A team consisting of people that will come up with a strategy to promote the app on the market.
a. Interoperability: Can the system support integration with other systems and services?
b. Scalability: Can the system handle a growing number of users and new updates?
c. Security: Can the system protect users from malicious users and threats ?

5. **Investors**: A person that is funding the creation of this application.
a. Scalability: Can the system survive on the market against its competitors and quickly growing trends?
b. Usability: Is there a demand on the market for the features offered by the system?

6. **IT Department**: Team consisting of people that maintain the software and hardware of the application.
a. Performance: Will the system be able to handle heavy traffic and offer a smooth experience?

b. Portability: Can the system run on numerous platforms?
c. Reliability: Can the system still run some features when there is no network connectivity?
d. Flexibility: Does the system run similarly on any device?

7. **Condo Management Company**: A company that owns condos and that is trying to put them on sale or rent.
a. Security: Can the system protect users from malicious users and threats ?
b. Usability: Is the sign up process and property profile creation simple?
c. Interoperability: Can the system support integration with other systems and services?
d. Usability: Is it easy to customize a profile that was created ?

8. **Future Condo Owner**: A person that is looking to buy a new home.
a. Security: Can the system protect users from malicious users and threats ?
b. Usability: Is the sign up process and user profile creation simple?
c. Interoperability: Can the system support integration with other systems and services?

9. **Current Condo Owner**: A person that has purchased units of a condo.
a. Security: Can the system protect users from malicious users and threats ?
b. Usability: Is it easy to customize a profile that was initially created ?

10. **Future Condo Renter**: A person that is looking to rent a new home.
a. Security: Can the system protect users from malicious users and threats ?
b. Usability: Is the sign up process and user profile creation simple?
c. Interoperability: Can the system support integration with other systems and services?

11. **Current Condo Renters**: A person that is currently renting a unit of a condo.
a. Security: Can the system protect users from malicious users and threats ?
b. Usability: Is it easy to customize a profile that was initially created ?

12. **Employee of the Condo Management Company**: A person that was assigned by the condo management company to treat their customers' requests.
a. Security: Can the system protect users from malicious users and threats ?
b. Usability: Is the sign up process and user profile creation simple?
c. Interoperability: Can the system support integration with other systems and services?

d. Usability: How can the system be used to fulfill requests submitted by a condo owner?

13. **Real Estate Brokers**: A person that helps either the condo management company or the client to sell or buy a property.

a. Security: Can the system protect users from malicious users and threats ?
b. Usability: Is it easy to get in contact with clients?

14. **Hacker**: A person that is trying to take advantage of the app by obtaining access to unauthorized personal information.

a. Security: Can the system protect users from malicious users and threats ?

15. **Competitors**: A company that views the app as a threat for their business.

a. Security: Which information should be available to the public ?
b. Security: Can the system be reverse engineered easily?

16. **Data Brokers**: An organization that collects personal data to sell them later.

a. Security: Can the system protect users from malicious users and threats ?
b. Security: Which information should be available to the public ?

3 Viewpoints

3.1 Functional Viewpoint

3.1.1 Overview

The functional viewpoint describes what the system does from the viewpoint of the user. It defines how the system will execute the required functions and how the exposed interfaces interact with each other.

3.1.2 Concerns and Stakeholders

3.1.2.1 Concerns

From a functional viewpoint, the concerns are:

- How does the user interact with the system to accomplish a task?
- What are the critical use cases of the system?
- What are the main modules or subsystems of the system and how do they interact with each other?

3.1.2.2 Stakeholders

The typical stakeholders of this view are:

- Product owner
- Development team
- Future and current condo owners/renters
- Condo Management Company
- Employees

3.1.3 Model Kinds

For this viewpoint, the kind of diagrams used are the use case diagram and the activity diagram. Both are classified into the behavioral model..

3.1.4 Behavioural Model

3.1.4.1 Behavioural Model Conventions

The UML conventions were followed to create the diagrams..

3.1.4.2 Behavioural Model Correspondence Rules

- An actor should be associated with at least one use case.
- Actors are properly identified. They should have different roles.
- Avoid redundant use cases.
- Actions are well defined.

3.1.5 Operation on Views

- Define actors that need to be included in the diagram.

- Define use cases.
- Associate actors with use cases

3.1.6 Correspondence Rules

- The scenario view will be described by the use case diagram.
- The process view will be described by the activity diagram.

3.1.7 Sources

- https://www.viewpoints-and-perspectives.info/home/viewpoints/functional-viewpoint/
- https://www.javatpoint.com/uml-diagrams

3.2 Development Viewpoint

3.2.1 Overview

The development viewpoint describes how the system has been developed. It focuses on the development process, the methodologies used and the software artifacts created during the process.

3.2.2 Concerns and Stakeholders

3.2.2.1 Concerns

From a development viewpoint, the concerns are:

- What are the main components or modules of the system?
- How are components and modules connected and organized?
- What is the model or pattern used for the system structure?
- Can the system be tested without much effort?
- Can the system protect users from malicious users and threats?
- Does the system run similarly on any device?

3.2.2.2 Stakeholders

From a development viewpoint, the stakeholders are:

- Development team
- IT department
- Condo Management Company
- Product Owner

3.2.3 Model Kinds

For this viewpoint, the diagrams used are the domain model, component diagram and the class diagram. They are classified as structural diagrams, which means the only model kind for this viewpoint is the structural model.

3.2.4 Structural Model

3.2.4.1 Structural Model Conventions

The structural diagrams all follow the UML convention. They describe the structure of the system.

3.2.4.2 Structural Model Correspondence Rules

- Classes are well defined.
- Classes have associations with other classes.
- Associations should have a multiplicity.
- Activity diagram should be thorough.
- Conceptual classes in the domain model come from the requirements.

3.2.5 Operation on Views

- Define classes.
- Define the associations and multiplicity between classes.
- Make a list of possible users actions
- Define key concepts and relationships.

3.2.6 Correspondence Rules

- The class diagram will define the logical view.
- The domain model will define the logical view.
- The component diagram will describe the logical view.

3.2.7 Sources

- https://www.viewpoints-and-perspectives.info/home/viewpoints/development/
- https://www.javatpoint.com/uml-diagrams

3.3 Deployment Viewpoint

3.3.1 Overview

The deployment viewpoint focuses on how the system is deployed, and what type of environment the system runs on. It also describes any third-party requirements and physical constraints for the deployment process.

3.3.2 Concerns and Stakeholders

3.3.2.1 Concerns

From a deployment viewpoint, the concerns are:

- How scalable is the deployment architecture?
- How is the performance of the deployment architecture?
- How is load balancing handled?

- What are the security measures put in place to protect data?

3.3.2.2 Stakeholders

From a deployment viewpoint, the stakeholders are:

- Development Team
- IT Department

3.3.3 Model Kinds

The model used to describe this viewpoint is the deployment diagram.

3.3.4 Deployment Diagram

3.3.4.1 Deployment Diagram Conventions

To construct the deployment diagram, the UML conventions must be followed.

3.3.4.2 Deployment Diagram Correspondence Rules

- Relate software elements and the corresponding hardware elements they are deployed to.
- Associate nodes with each other to describe their interactions.
- Each software element should belong to only one node.
- Indicate how load balancing is handled.

3.3.5 Operations on Views

- Identify key software and hardware elements.
- Associate nodes and indicate messages they pass.
- Show the artifacts in the nodes.

3.3.6 Correspondence Rules

- The physical view will be described by the deployment diagram.

3.3.7 Sources

- https://www.viewpoints-and-perspectives.info/home/viewpoints/deployment/
- https://www.javatpoint.com/uml-diagrams

4 Views

Much of the material in an AD is presented through its architecture views. Each view follows the conventions of its governing viewpoint. A view is made up of architectural models.

The Logical View in the 4+1 View Model primarily addresses the functional requirements of the system, providing a high-level view of the application's core structure and its behavior from a user's perspective. It is typically modeled using class diagrams, which showcase the system's key classes, interfaces, and the relationships between them, encapsulating the system's principal functionality. This view is crucial for understanding the object model of the application and is often the focus for application developers as it guides the application's development in terms of key abstractions and their interactions.

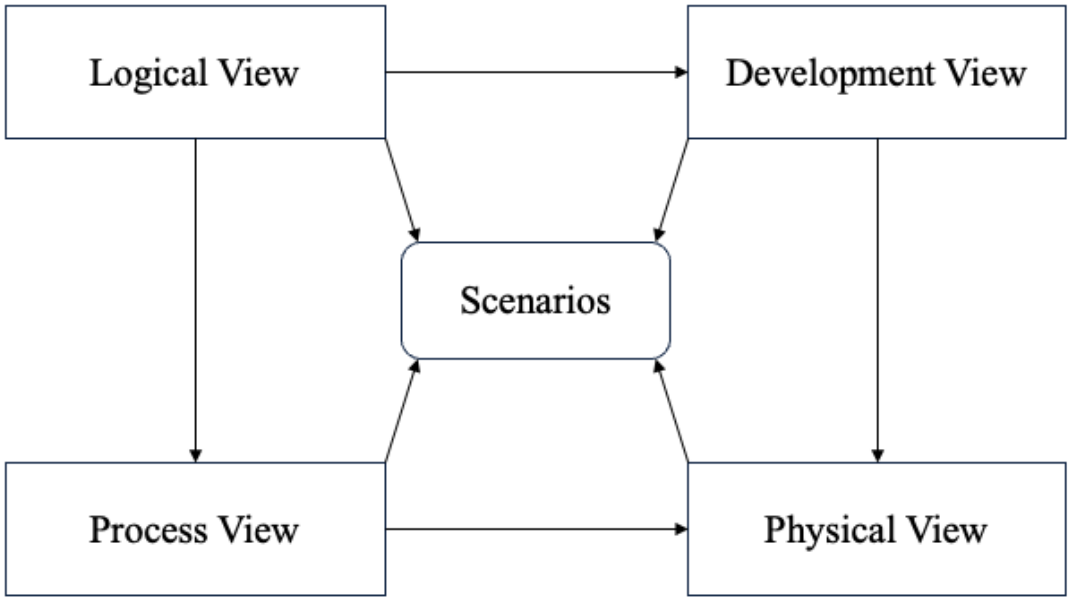

Figure 1 The components of the 4 + 1 view model - Sprint 1

4.1 View: Logical View

The Logical View in the 4+1 Model governed by the functional viewpoint details the system's structure through classes, interfaces, and their interactions, focusing on end-user functionality and object responsibilities. It maps out the object-oriented design, highlighting aspects like inheritance and object relationships, which are crucial for developers.

4.1.1 Models+

The logical view can be represented by the multiple models, including the Class model, the sequence model, or State model.

4.1.2 Class Model and Domain Model

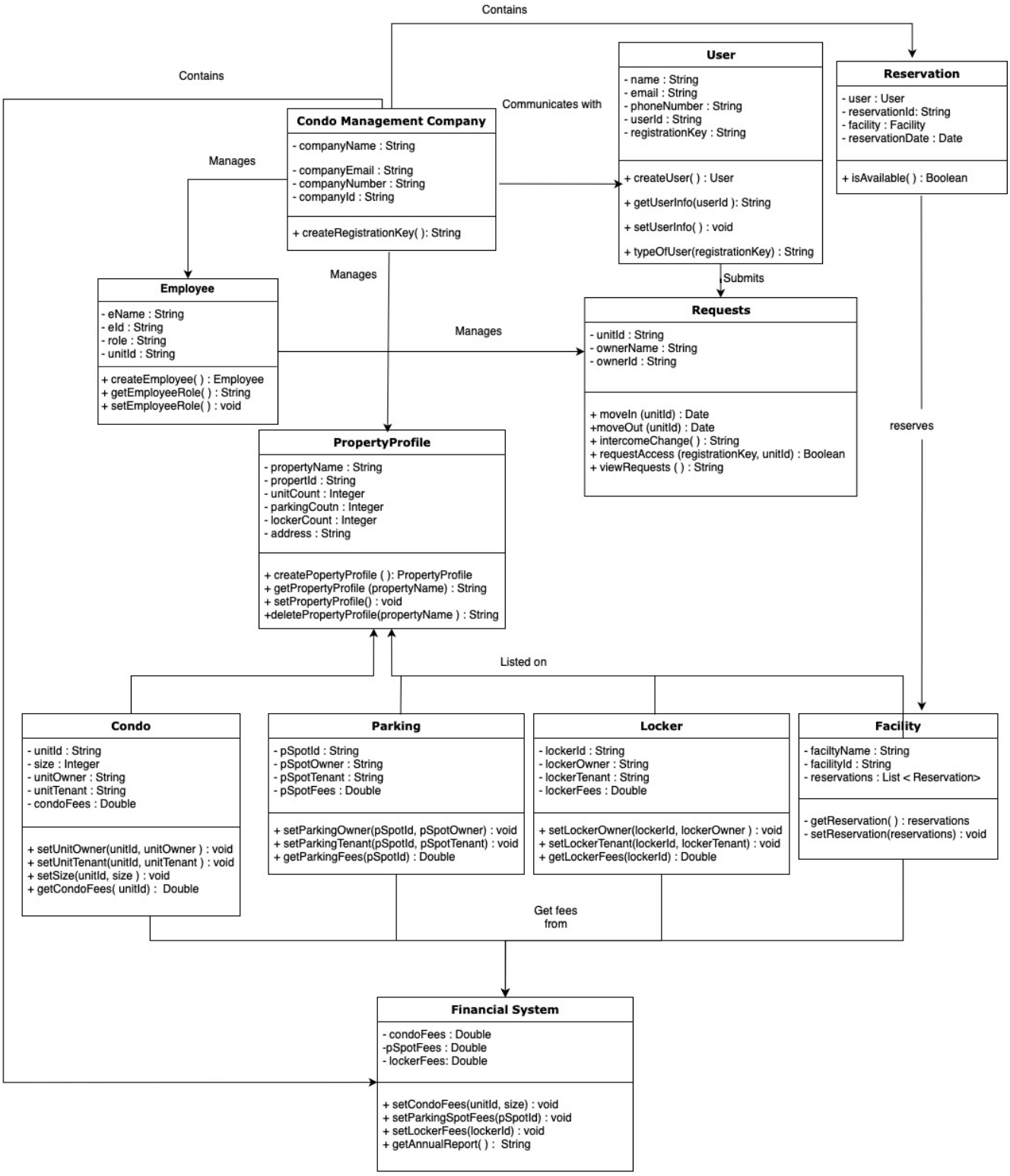

Figure 2: Class diagram for the project - Sprint 1

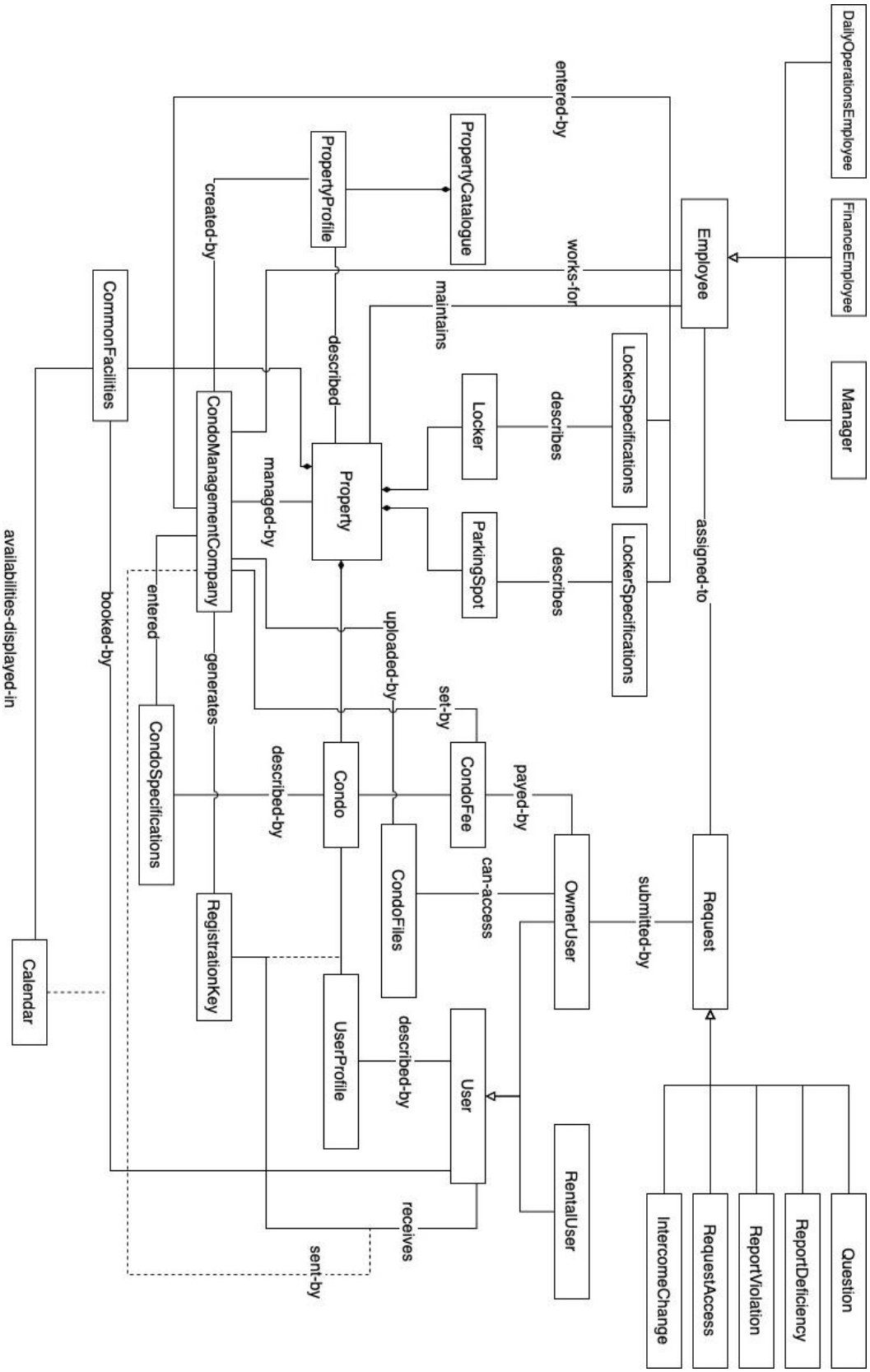

Figure 3: Domain model for the project from - Sprint 1

4.1.3 Known Issues with View

Multiplicities of the classes in the domain and class models need to be added as well as the

4.2 View: Development View

The Development View in the 4+1 View Model which is governed by the development viewpoint delineates the system’s structure from the developer's perspective, focusing on software management aspects like module organization, source code layering, and component partitioning. It uses diagrams such as component and package diagrams to provide a roadmap for the system's software build, its dependencies, and its development environment setup.

4.2.1 Models+

The development view can be represented with the package model which outlines how code and components are organized into packages or modules and the component model which will be presented.

4.2.2 Component Model

The Component Model within the Development View represents the modular parts of a system's software architecture, detailing how each component functions, interacts with others, and its dependencies. It serves as a blueprint for developers, illustrating the breakdown of the system into manageable, reusable, and interchangeable parts that encapsulate specific functionality.

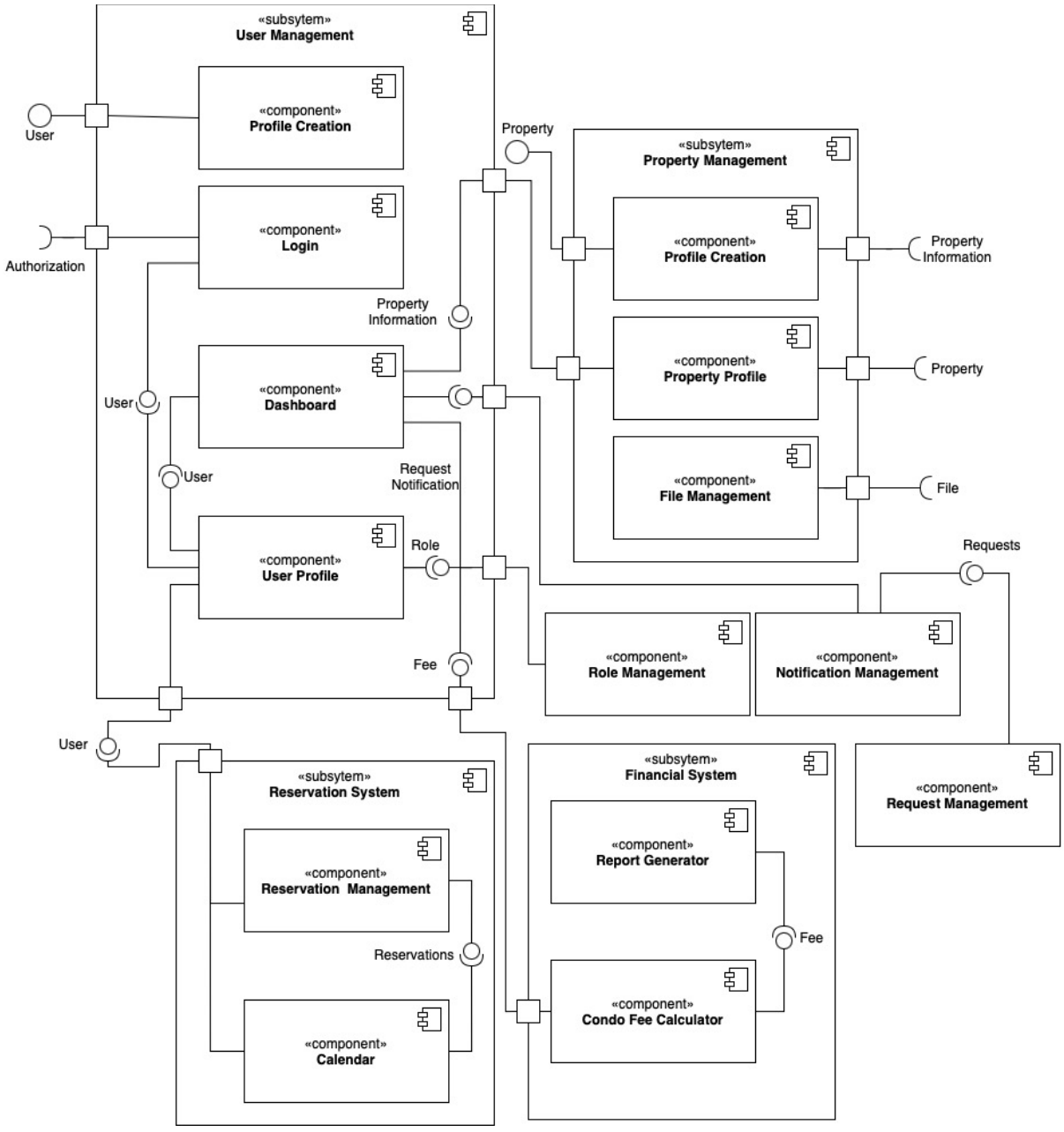

Figure 4 component diagram for the project submitted - Sprint 1

4.2.3 Known Issues with View

nothing for now

4.3 View: Process view

The Process View of the 4+1 View Model which is governed by the behavioral viewpoint captures the dynamic aspects of the system, illustrating how runtime processes interact and manage data flow. It focuses on the concurrency, synchronization, and communication between various components, often represented through activity and sequence diagrams, to ensure the system's scalability and performance.

4.3.1 Models+

The process view can be represented by the concurrency model which details the concurrency mechanisms and synchronization aspects of a system. In addition, the activity model also represent the view.

4.3.2 Activity Model

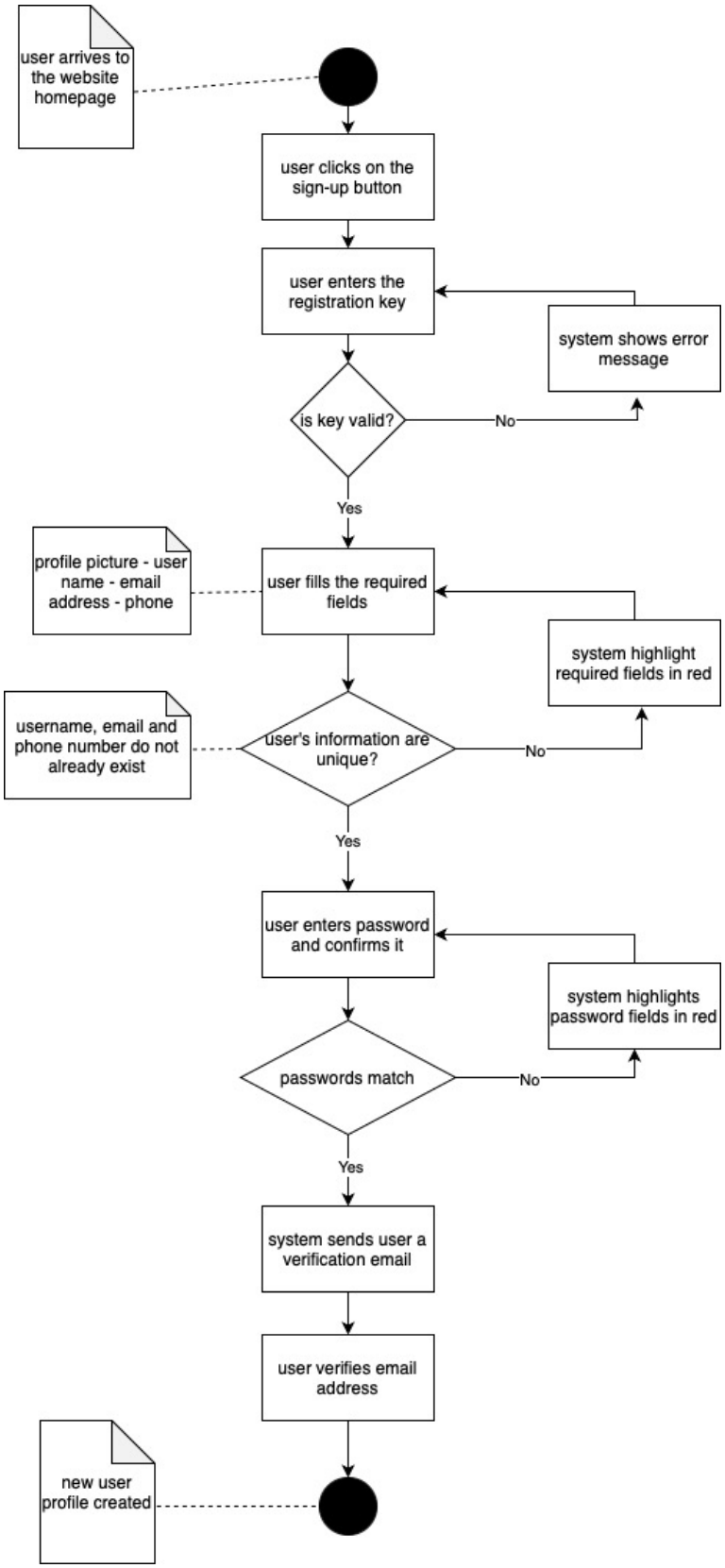

Figure 5 Activity diagram for the Sign Up Process - Sprint 1

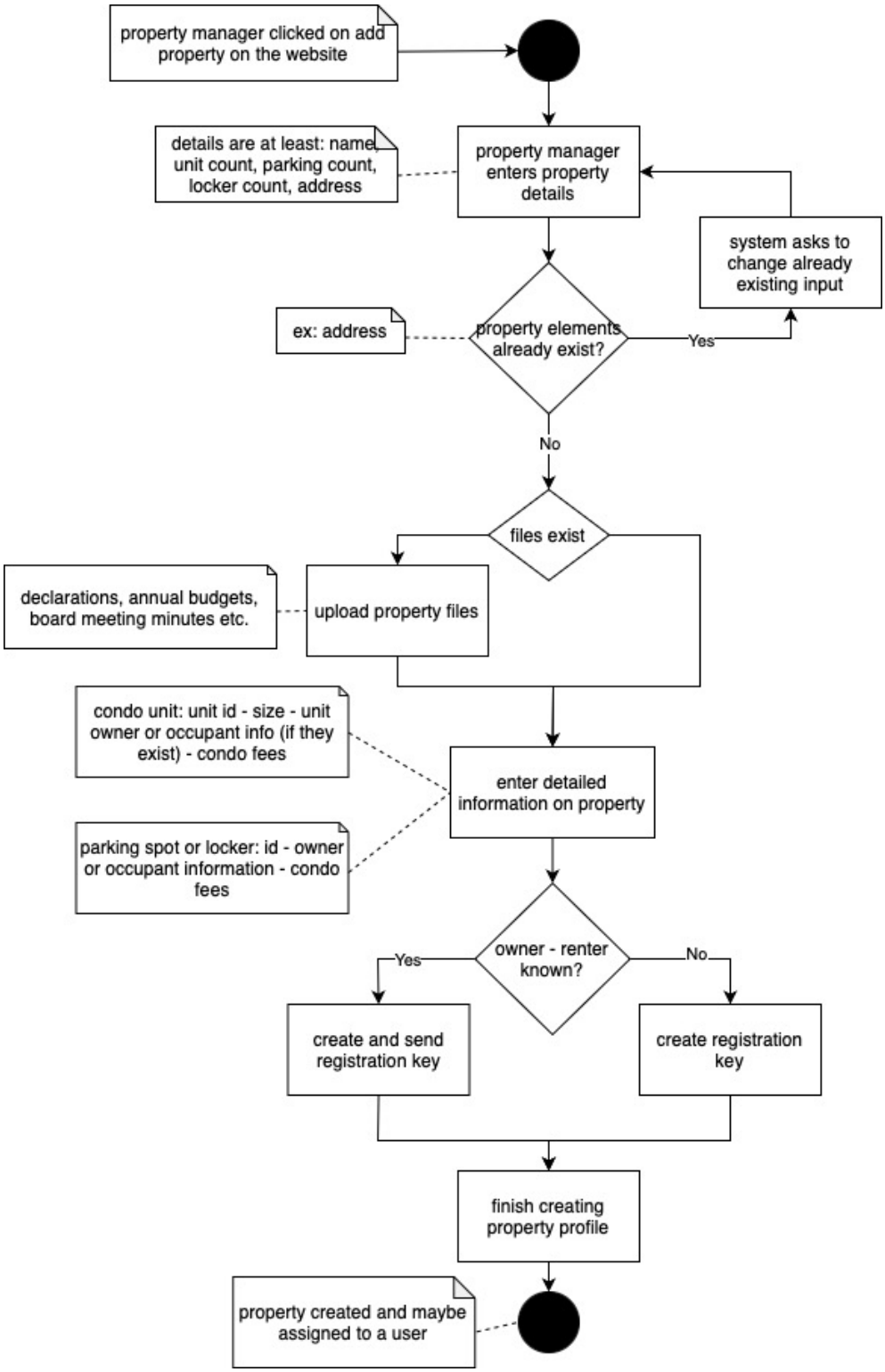

Figure 6: Activity diagram for Property Profile Creation - Sprint 1

4.3.3 Known Issues with View

There’s more diagrams to be added for later stages of the project

4.4 View: Physical view

The details of this information will be as specified by the organization and/or project. See §1 for examples of identifying and supplementary information.

Views have their own identifying and supplementary information distinct from ADs because they may be developed and evolve separately over the lifetime of a project.

4.4.1 Models+

The deployment model represents the physical view which is in turn governed by the deployment viewpoint.

4.4.2 Deployment model

to be added

4.4.3 Known Issues with View

the model is yet to be created to fit the physical view.

4.5 View: Scenario view

The Scenario View, governed by the usability viewpoint is central to the 4+1 View Model, provides a practical illustration of the system's architecture through real-world use cases, detailing how various elements interact to fulfill specific user-driven tasks. It serves to validate the architecture, ensuring that it meets the user requirements and stakeholder expectations. This view often uses use case diagrams and sequence diagrams to depict the interaction between the users and the system, highlighting the system's behavior in various scenarios.

4.5.1 Models+

The scenario view can be represented by the interaction model which illustrates the flow of activities and communication between system components during user interactions, using diagrams to detail the sequence of events. It validates system behavior, ensuring alignment with user requirements and operational objectives in real-world scenarios. as well as the use case model discussed below.

4.5.2 Use Case model

The Use Case Model captures and organizes the functional requirements of a system by detailing the interactions between actors (users or external systems) and the system itself to achieve specific goals. It is expressed through use case diagrams,

which provide a high-level overview of the system's functionality and clarify the context and scope of each interaction.

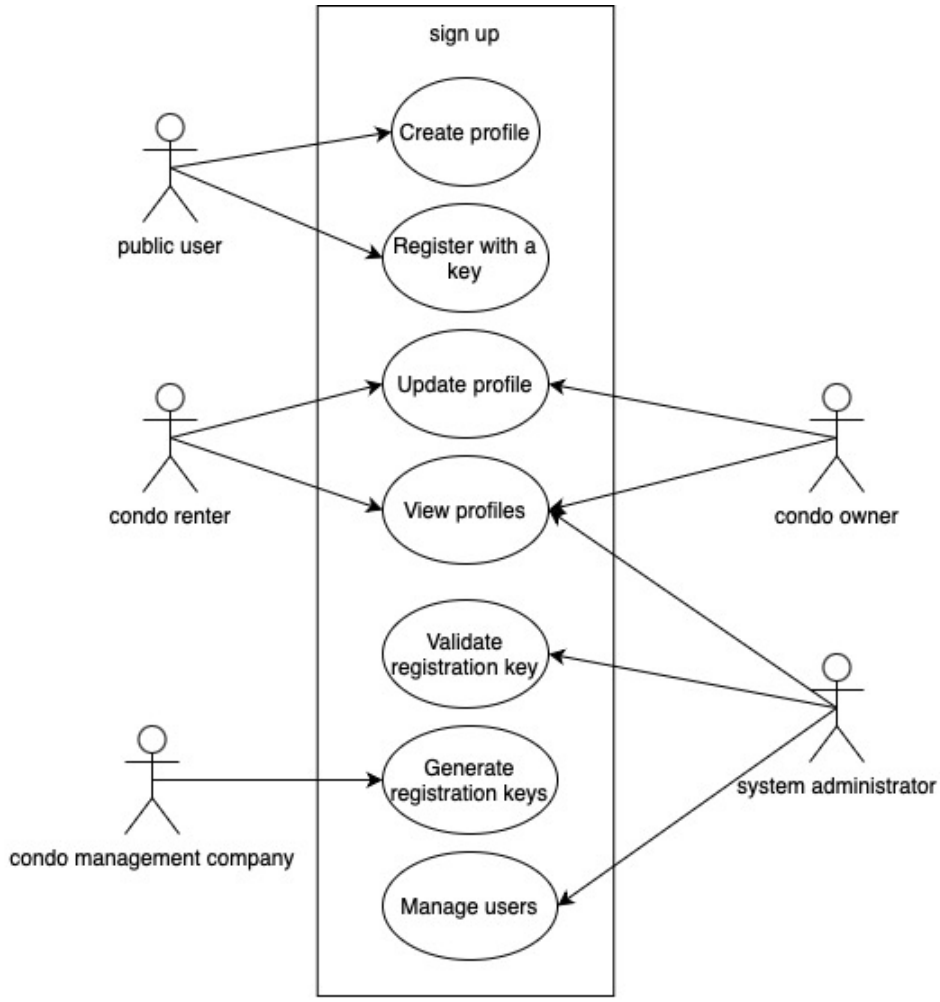

Figure 7: Use case scenario for user sign up - sprint 1

Scenario for sign up use case:

1. A **Public User** accesses the system and chooses to create a profile.
2. The user provides the necessary details (profile picture, username, email, phone number) and creates their profile.
3. The user then receives a registration key from their **Condo Management Company**.
4. The user enters this key into the system to register either as a **Condo Owner** or a **Rental User**.
5. The system validates the key.
6. Upon successful validation, the user's status is updated to either Condo Owner or Rental User, and they can now access additional functionalities like updating their profile.

7. The **System Administrator** oversees these operations, ensuring everything runs smoothly and managing user accounts as needed.

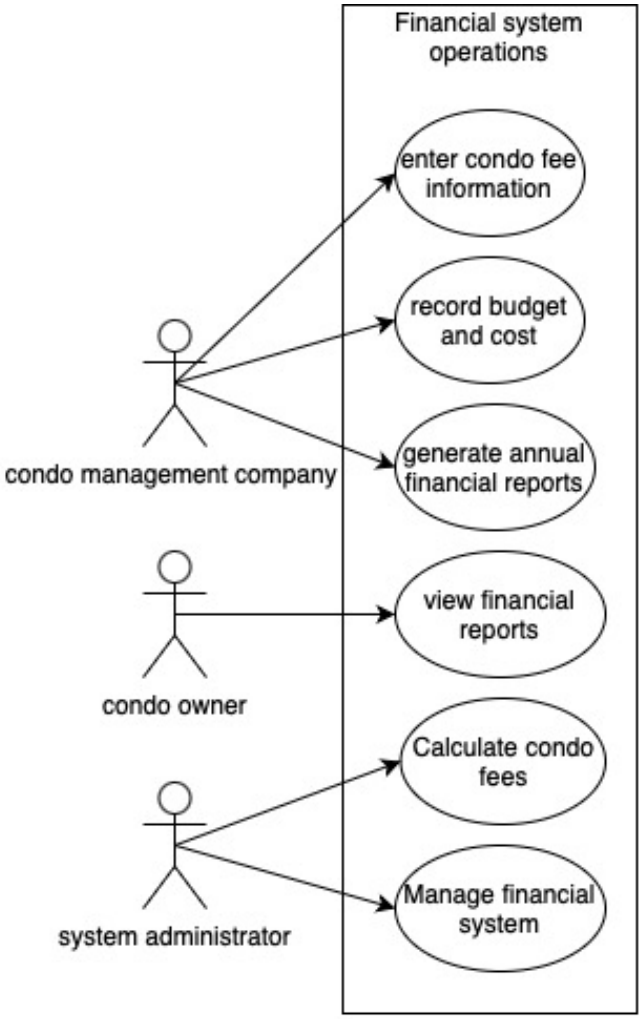

Figure 8: Use case scenario for Financial system operations- sprint 1

Scenario for financial system operations:

1. The **Condo Management Company** enters the condo fees per square foot and per parking spot into the system.
2. The system calculates the total condo fee for each unit based on these rates and the specific details of each unit.
3. The management company also records and enters the operational budget, which includes the total collected condo fees, and inputs the cost for each operation.
4. At the end of the year, the company uses the system to generate an annual financial report, which includes details of all the condo fees collected and other financial data.

5. **Condo Owners** can access this report to understand the financial status of their property, including the total fees paid and how these funds are being utilized.
6. The **System Administrator** oversees the system to ensure that all financial data is correctly processed, and that the system remains secure and functional.

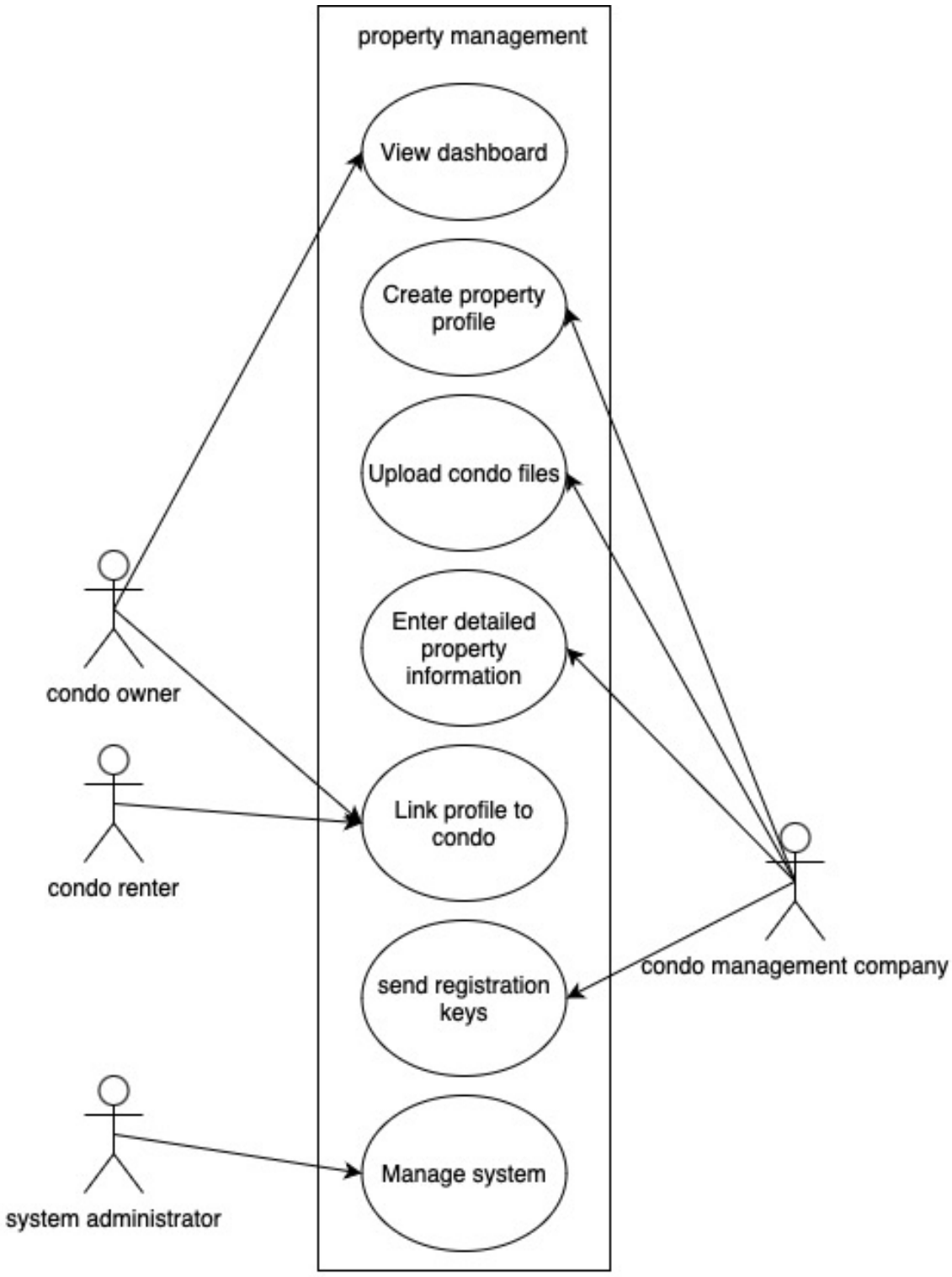

Figure 9: Use case diagram for property management operations - sprint 1

Scenario for property management use case:

1. The **Condo Management Company** creates profiles for each property, including all relevant details.
2. They upload necessary condo files to the system, ensuring they are accessible to condo owners.
3. Detailed information about each condo unit, parking spot, and locker is entered into the system.
4. The management company sends registration keys to **Condo Owners** and **Rental Users**.
5. **Condo Owners** and **Rental Users** receive these keys and use them to link their specific units with their profiles on the system.
6. **Condo Owners** can view their dashboard, which provides comprehensive information about their properties, including financial status and other personal details.
7. The **System Administrator** oversees the entire process, ensuring the system's integrity and smooth operation.

4.5.3 Known Issues with View

more associations between operations might be possible and more use case diagrams can be implemented in the next phases of the project.

5 Consistency and correspondences

This chapter describes consistency requirements, recording of known inconsistencies in an AD, and the use and documentation of correspondences and correspondence rules.

5.1 Known inconsistencies.

- In the activity diagrams, there is a slight inconsistency about when the registration key is provided to the client. It's given both during the sign-up process and during the purchase process.
- There is a slight consistency in the naming convention between certain diagram but it's not major because it's easily understandable that both names represent the same thing.

5.2 Correspondences in the AD

- There is a GUI representing a User Profile that users can interact with, which corresponds to the User Profile class in the Class Diagram and Domain Model as initially planned.

5.3 Correspondence rules

- There needs to be a clear and consistent naming convention between the database tables and the user profiles they are related to.
- Any changes in the profile schema in the database should reflect on the profile page view in the website.
- The user interface elements need to align with their corresponding system functionality (The Reservation page should be linked to the functionality that oversees handling reservation requests)
- Any interface definition needs to have a corresponding GUI element. (The reservation interface needs to be linked to a reservation page a user can interact with)