Architecture Description of Client-Server for Decagon

Template prepared by:
Rich Hilliard
r.hilliard@computer.org

Table of Contents

1 Introduction

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

1.1 Identifying information

The architecture being expressed is a client-server architecture with a decoupled front-end and back-end. The front-end is implemented using Angular, a TypeScript-based web application framework, while the back-end is implemented using Firebase services, including Firebase Authentication and Firebase Realtime Database. We describe the decoupling of the front-end and back-end because of the concealment of Firebase implementation details within the service files, distinct from the TypeScript files.

The system is a web-app based Condo Management System. The system is designed to manage condominium-related information such as adding properties and condo units by condo management while owner and rental users can access information and make requests to the management. Its architecture encapsulates both the client-side, Angular, and server-side, Firebase, components.

The code for the project is organized in a single directory. The front-end Angular files are stored in this directory and are responsible for presenting the user interface and managing user interactions. Additionally, the back-end service files, used by the front-end, are also located in the same directory. These service files interact with Firebase services for authentication and database operations on the Realtime database..

This architecture is designed to deliver a scalable, real-time solution for efficiently managing condominium-related data. It establishes a clear separation of concerns between front-end and back-end components, employing service files for each entity as controller classes. This implementation follows the principles of high cohesion and low coupling through effective refactoring and design methods.

1.2 Supplementary information

1. Date of Issue: February 1st, 2024
2. Status:

   Document is a draft and will be under review on February 7th, 2024.

3. Authors, Reviewers, Approving Authority, Issuing Organization:

   Authors: Ann-Marie Czuboka and Karina Duran-Sanchez

   Reviewers: Isabelle Czuboka, Guillaume Lachapelle, Nicholas Piperni, Oliver Vilney, Carla Shawi, Tharsipa Nadarajah, Poula Farid, Ziad Elsharkawi

   Approving Authority - Project Owner: Triet Pham

   Issuing Organization - Concordia University, SOEN 390, Professor: Jinqiu Yang

4. Change History:

   Revision 1 - First Draft on February 2nd, 2024.

5. Summary:

The System Architecture (SA) document outlines the architecture of a system, providing a good view of its structure, components, modules, and their interactions. It includes details about

hardware and software through the use of multiple diagrams. This document serves as a resource for developers, architects, and other stakeholders involved in the design, development, and maintenance of the system.

1.3 Other information

1.3.1 Overview

The architecture described in this document aims to build a robust and scalable design for a Condo Management System. The purpose is to efficiently manage condominium-related data, including user authentication, real-time database interactions, and great user experiences. The architecture uses Angular, a TypeScript-based front-end framework, and Firebase, a serverless platform, for authentication and data storage.

The scope of this architecture spans both the client-side and server-side components. On the client side, Angular is utilized to create a responsive user interface, while the server side utilizes the Firebase services, including Firebase Authentication and Firebase Realtime Database for storing and retrieving real-time data.

Client-Side (Angular):

- Responsibility: User interface, interactions, and presentation logic.
- Technology Stack: Angular framework, TypeScript, HTML, SCSS, BootStrap.
- Interaction: Communicates with the server for authentication and data operations.

Server-Side (Firebase):

- Responsibility: User authentication, real-time data storage, and server-side logic.
- Technology Stack: Firebase Authentication, Firebase Realtime Database.
- Interaction: Listens for client requests, authenticates users, and manages real-time data.
- Scalability: Firebase services provide scalability and real-time capabilities (hosting).

This document serves as a guide to the architecture of the Condo Management System. It provides insights into the design decisions, key components, and interactions between the client and server.

1.3.2 Architecture evaluations

We evaluated the architecture by producing several design diagrams. These include class diagram, component diagram, domain model and deployment diagram.

For a better view of all the software architecture diagrams, please follow this link:

https://drive.google.com/file/d/1XJi_ItVIW9J63RhIwWlPqFVHPzfRN7F1/view?usp=sharing

1. Domain Model:

A domain model represents the key concepts, entities, and their relationships within the domain of Condo Management. It provides a conceptual overview of the real-world entities and how they relate to each other.

The domain model is essential for verifying if the system aligns with the user requirements of the system. It mainly helps stakeholders, including developers, have an understanding of the core entities and their interactions. [1]

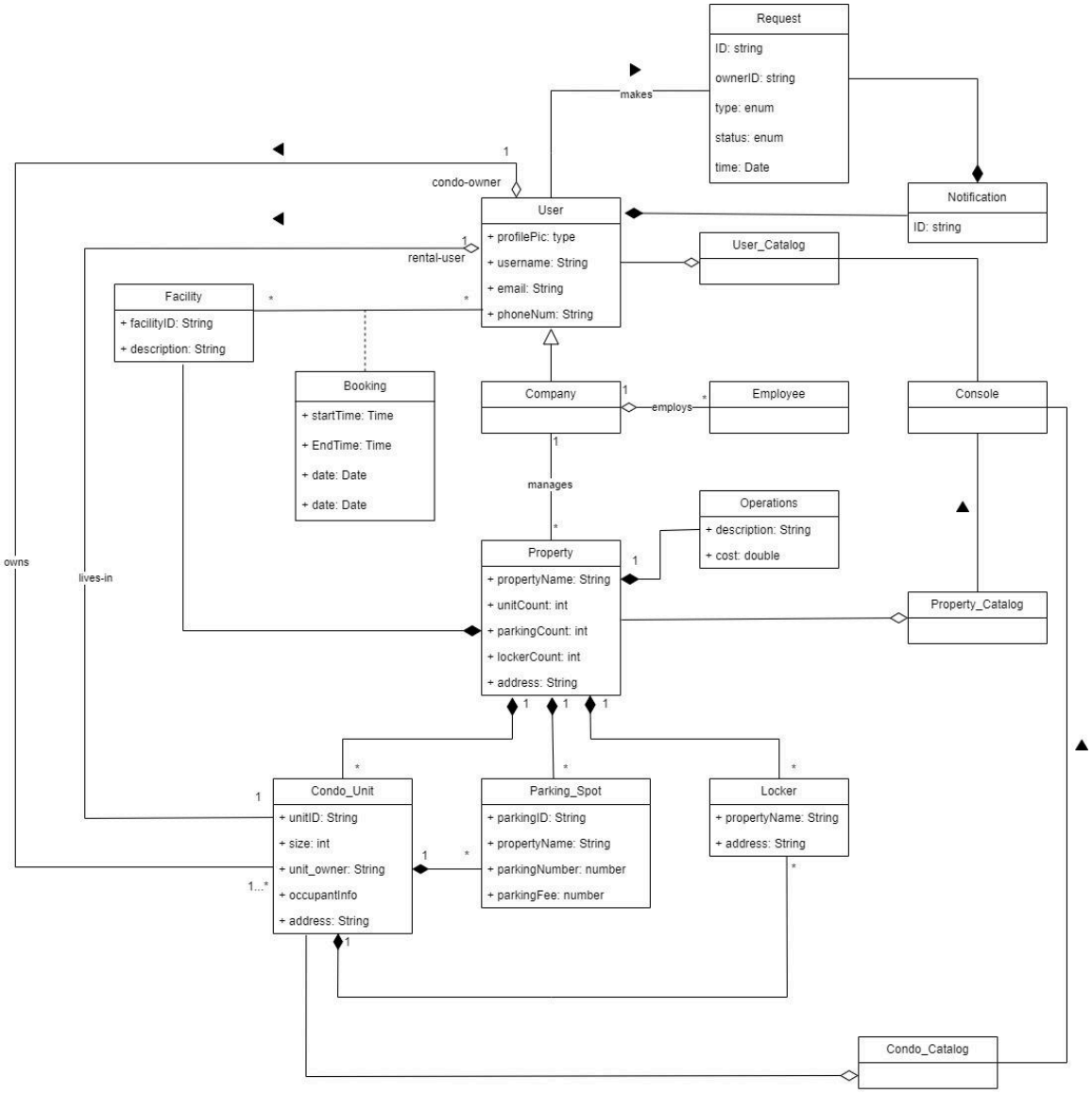

Figure 1: Domain Model Diagram

2. Class Diagram:

A class diagram is static and illustrates the classes, their attributes, and the relationships between classes in the Condo Management System. It provides a visual representation of the object-oriented structure, highlighting key entities, their properties, and how they interact.

Class diagrams are crucial for understanding the organization of code, and facilitating communication among development teams. A well developed class diagram helps developers comprehend the structure of the system, helps to organize the code, and ensures that key entities and their relationships are understood before implementation.[2]

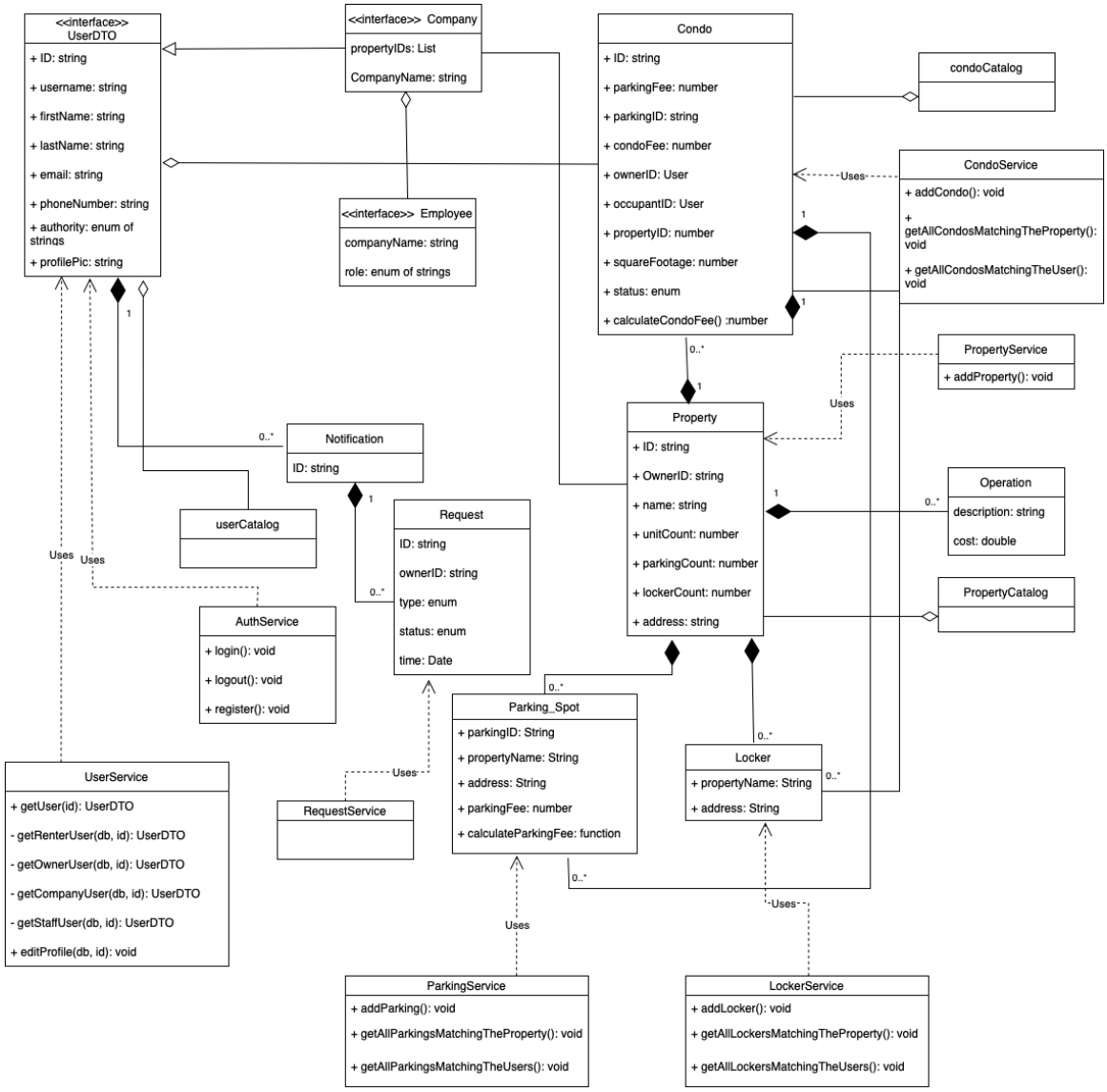

Figure 2: Class Diagram

3. Component Diagram:

A component diagram displays, in high-level, the components of the Condo Management System and their relationships with each other. It outlines the structure of the system, showcasing how components collaborate and form the entire system. [3]

Component diagrams are essential for showing the component architecture of the system, which helps in understanding how different parts of the application work together.

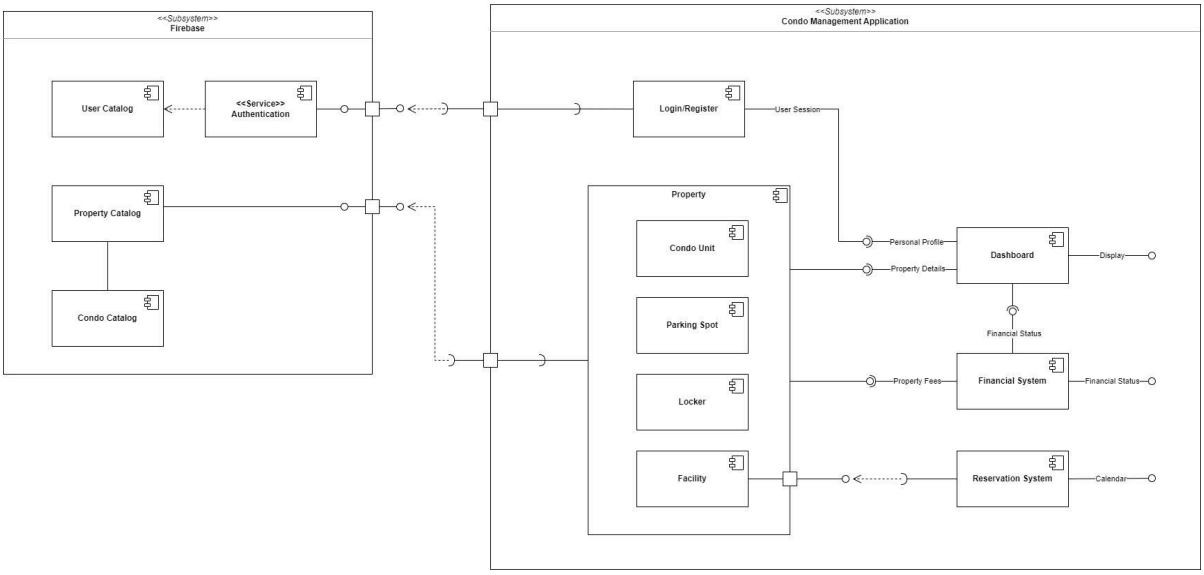

Figure 3: Component Diagram

5. Deployment Diagram:

A deployment diagram illustrates the physical arrangement of software components and their interactions. It shows how components are deployed across hardware, servers, and other infrastructure. [4]

These diagrams play an important role in the understanding and communication among project stakeholders, including developers, architects, and project managers.

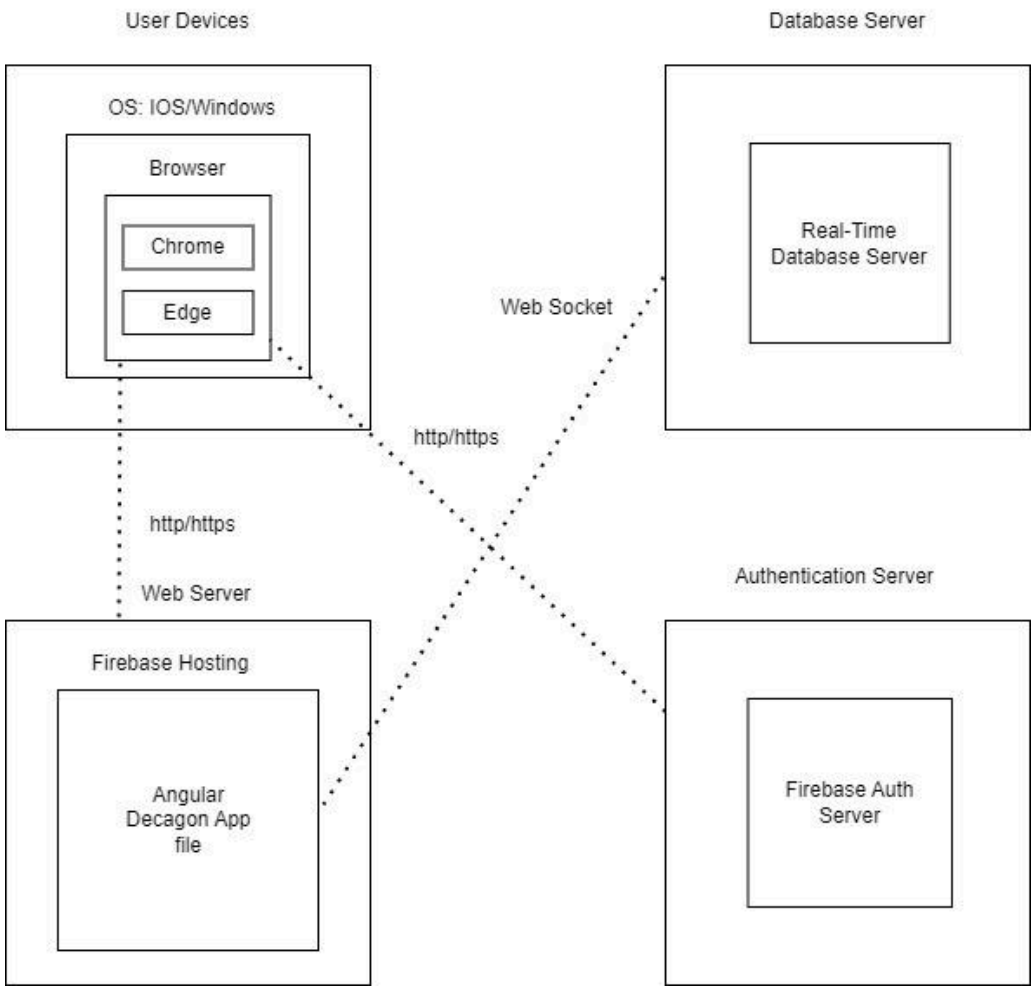

Figure 5: Deployment Diagram

6. Activity Diagram

An activity diagram is a behavioral diagram that shows the flow of control of a system. Additionally, an activity diagram helps describe how different use cases of a system are related at varying levels of abstraction. [5]

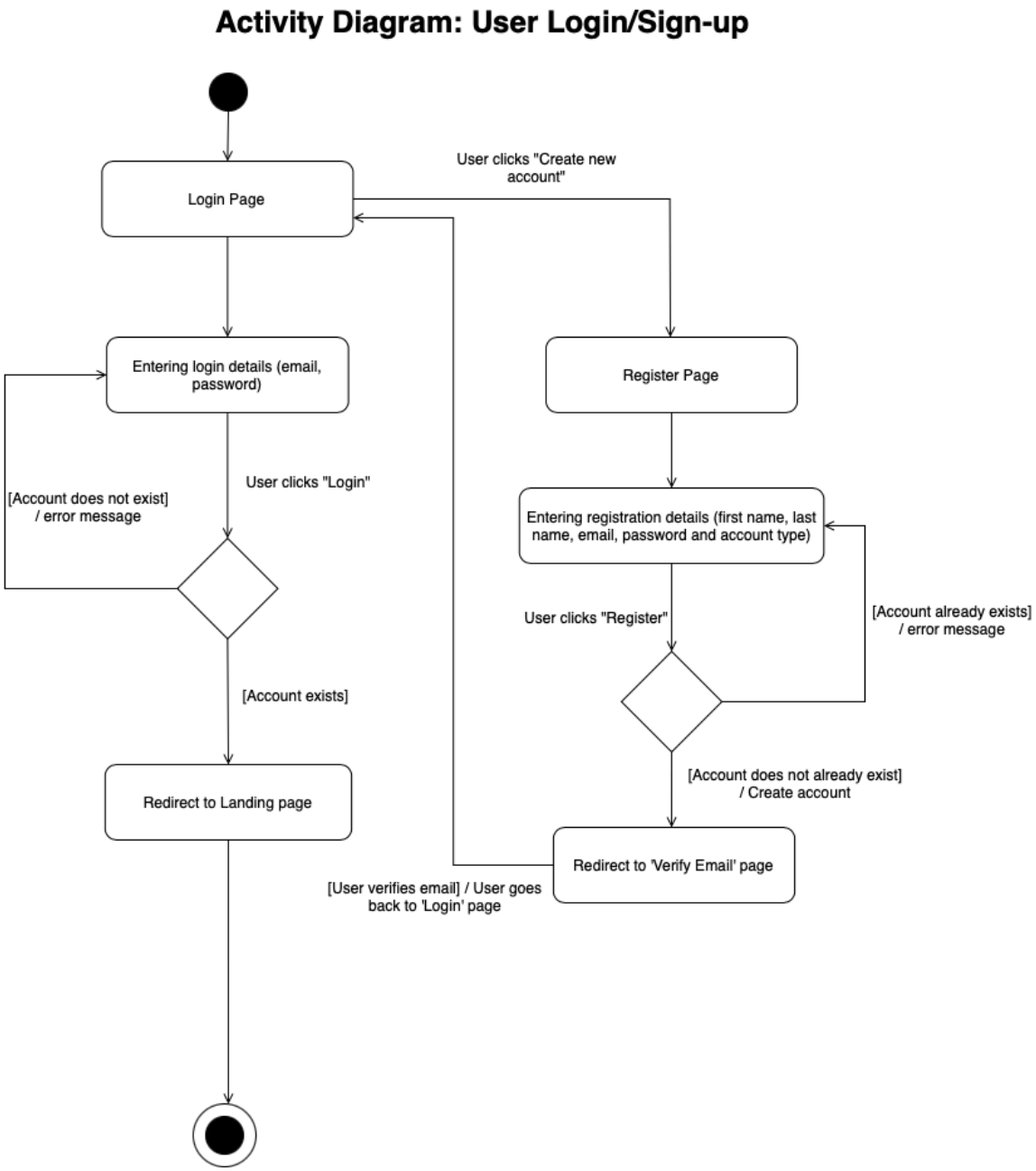

Figure 6: Activity Diagram - User Login/Sign-up

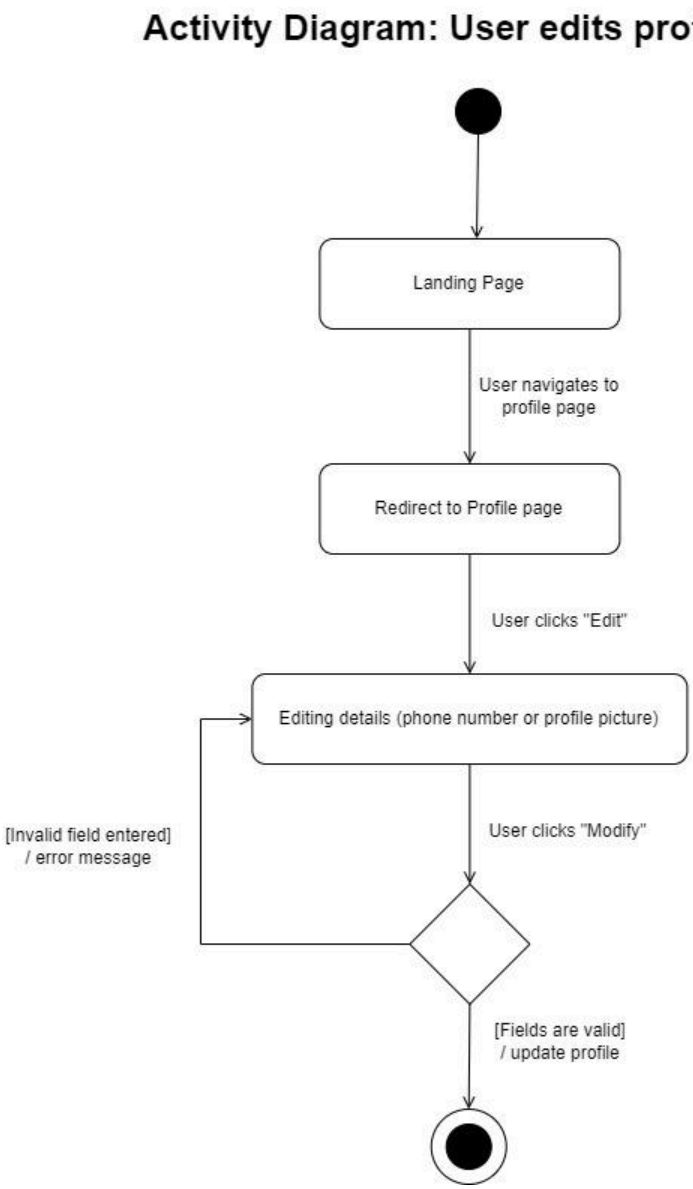

Figure 7: Activity Diagram - User Edits Profile

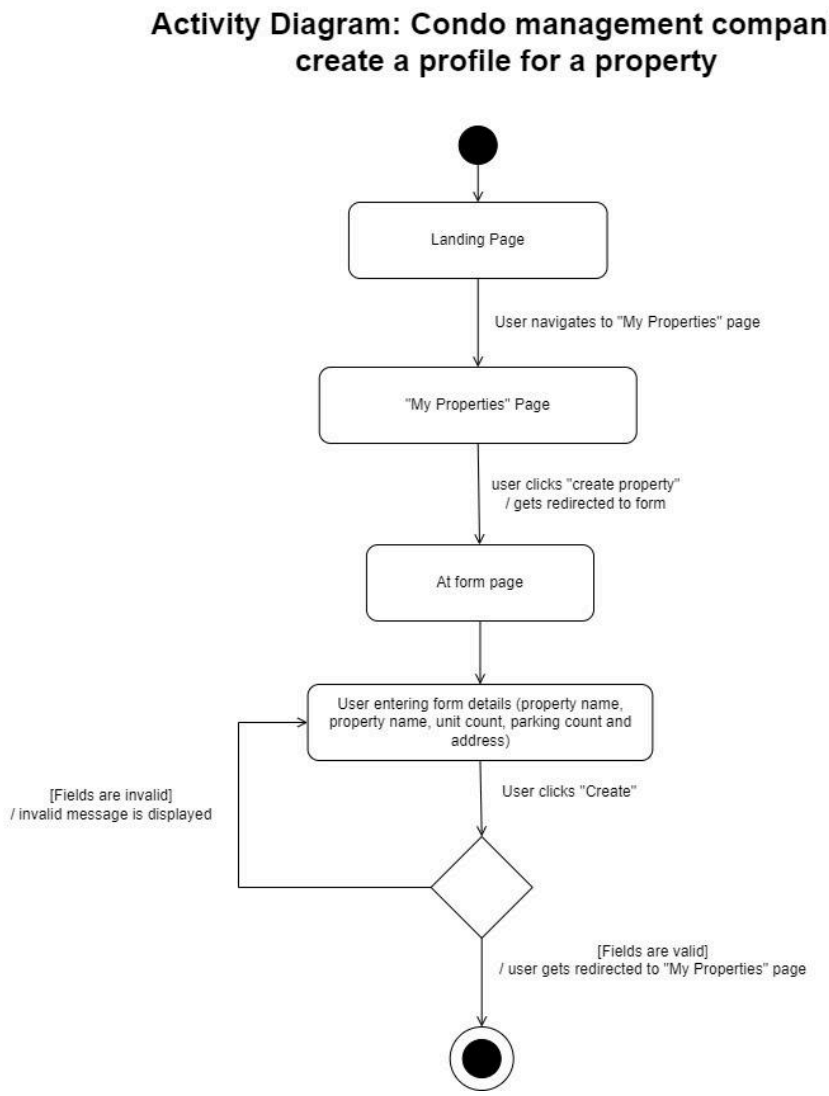

Figure 8: Activity Diagram - Property Profile Creation

7. Use-case Diagram

The purpose of a use-case diagram is to capture system requirements. A use-case diagram is a type of behavioral diagram that illustrates system interactions with an actor and possibly other systems. A use-case diagram is limited in that it doesn't show the steps a system needs to complete a function. [6]-[7]

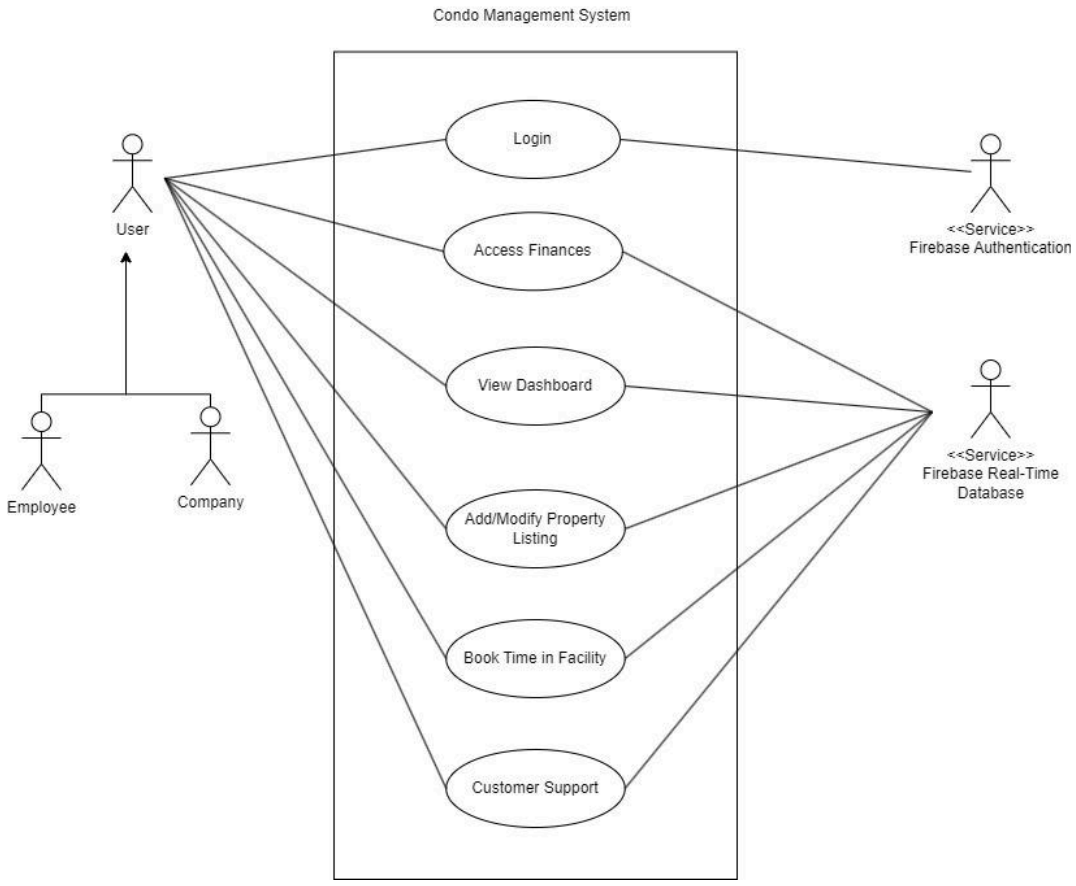

Figure 9: Use-case Diagram

1.3.3 Rationale for key decisions

Decision 1: Choosing Angular as the Front-End Framework

We selected Angular as the front-end framework for the Condo Management System.[8]

Rationale: Angular was chosen due to extensive library and built in components, and adherence to the MVC (Model-View-Controller) architecture. This decision aligns with our team's expertise (from previous school and work projects), ensuring efficient development, maintainability, and scalability of the front-end. Additionally, Angular's component based structure facilitates code organization, facilitating collaboration among developers.

Decision 2: Utilizing Firebase for Authentication and Realtime Database

Firebase was chosen as the backend solution for user authentication and realtime database storage.

Rationale: Firebase provides a serverless architecture, minimizing development time and boiler plate code. Its authentication services offer robust security measures, including multi-factor authentication such as Google Sign In. The integration of Firebase Realtime Database ensures seamless real-time data updates, aligning with the dynamic nature of condominium management system requirements. Additionally, most of the development team has experience with Firebase services.

Decision 3: Integration of GitHub Continuous Integration/Continuous Deployment (CI/CD)

A CI/CD pipeline will be implemented to automate the build, testing, and deployment processes. [9]

Rationale: Implementing the GitHub CI/CD will reduce manual errors, and ensure a consistent and reliable release process. This decision supports the Agile development methodology, enabling frequent releases, rapid feedback, and continuous improvement. Furthermore, using the CI/CD pipeline of GitHub just makes sense since the team will be using GitHub as a task management platform.

Decision 4: Utilizing Cypress as automation tests (end-to-end)

Cypress tests will be implemented for every user story as a testing tool. [10]

Rationale: Cypress offers a user friendly testing framework that allows the quality assurance team to write automated tests quickly and efficiently. Also, it offers a modern approach to testing web applications by running the web-app in another window and viewing the tests run in real time.

2 Stakeholders and concerns

This chapter contains information items for stakeholders of the architecture, the stakeholders' concerns for that architecture, and the traceability of concerns to stakeholders. See also: ISO/IEC/IEEE 42010, 5.3

2.1 Stakeholders

1. Users

Condo residents (owners and renters), property managers, and employees who interact with the Condo Management System.

2. Operators

Condo staff responsible for operating and managing the Condo Management System on a day-to-day basis.

3. Acquirers

The individual or group responsible for acquiring the Condo Management System, such as condo management associations or property developers.

4. Owners

Individuals or companies that own the Condo Management System as a product.

5. Suppliers

External entities providing software components or third-party services to the Condo Management System.

6. Developers

The team responsible for designing and implementing the Condo Management System.

7. QA Team

Individuals or teams involved in the testing and maintenance of the Condo Management System.

2.2 Concerns

Functional Concerns

1. User-friendly interface for everyday tasks.

Description: Ensuring that the user interface is intuitive and efficient for users to perform their day-to-day tasks.

2. Access to real-time information related to the condo management system.

Description: Providing users with real-time access to relevant information within the Condo Management System.

3. Alignment of the system with the specific needs of the condo management domain.

Description: Ensuring that the system's functionalities meet the user's requirements of condo management.

4. Compatibility and integration with the condo management system.

Description: Ensuring that the Condo Management System components can work together and integrate with other systems.

5. Adherence to coding standards and best practices.

Description: Ensuring that the development team follows established coding standards and best practices in software development.

Non-Functional Concerns

1. Security and privacy of personal data.

Description: Addressing the security and privacy aspects to protect users' personal data.

2. System reliability.

Description: Ensuring that the system operates reliably and consistently without unexpected downtime.

3. Intuitive controls for effective operation.

Description: Focusing on the ease of use and efficiency of the system's operational controls.

4. Monitoring tools for identifying and addressing issues.

Description: Implementing tools and mechanisms to monitor the system for issues and address them proactively.

5. Cost-effectiveness of the system's acquisition and maintenance.

Description: Evaluating and optimizing the overall cost efficiency of acquiring and maintaining the Condo Management System.

6. Return on investment and financial viability.
Description: Assessing the financial benefits and sustainability of the system over time.

7. Long-term maintainability and adaptability to evolving requirements.
Description: Ensuring that the system is designed and built for long-term maintainability and adaptability to future changes and requirements.

8. Clarity in architectural decisions for effective implementation.
Description: Ensuring that architectural decisions are clear and well-defined to facilitate effective implementation by the development team.

9. Collaboration tools and environments for development.
Description: Providing tools and environments that facilitate collaboration and communication within the development team.

10. Clear documentation of requirements.
Description: Ensuring that project requirements are documented comprehensively for reference and understanding.

11. Effective debugging tools.
Description: Providing tools that aid in debugging and troubleshooting during development and maintenance.

12. Training and knowledge transfer for new team members.
Description: Focusing on knowledge transfer processes to ensure that new team members can effectively contribute to the project.

2.3 Concern–Stakeholder Traceability

The association of concerns to stakeholders can be depicted using a table.

Table 1: Association of stakeholders to concerns

	Users	Operators	Acquirers	Owners	Suppliers	Developers	QA Team
Functional Concerns							
User-friendly interface for everyday tasks.	X	X					
Access to real-time information related to the condo management system.	X	X					
Alignment of the system with the specific needs of the condo management domain.			X				
Adherence to coding standards and best practices.						X	X
Compatibility and integration with the condo management system.					X		
Non-Functional Concerns							
System reliability.		X					
Intuitive controls for effective operation.		X					
Long-term maintainability and adaptability to evolving requirements.				X			
Return on investment and financial viability.				X			
Cost-effectiveness of the system's acquisition and maintenance.			X				
Monitoring tools for identifying and addressing issues.		X					

Security and privacy of personal data.	X						
Collaboration tools and environments for development.						X	
Clarity in architectural decisions for effective implementation.						X	
Clear documentation of requirements.							X
Effective debugging tools.							X
Training and knowledge transfer for new team members.							X

3 Viewpoints+

Viewpoint 1: Functional Viewpoint

The Functional Viewpoint concerns the system's functionality and features, emphasizing both end-user interactions and the implementation itself. [11]

Concerns:

- User-friendly interface for everyday tasks.
- Access to real-time information related to the condo management system.
- Alignment of the system with the specific needs of the condo management domain.

Typical Stakeholders:

- Condo residents
- Property managers
- Staff
- Development team

Anti-concerns:

- Inefficient or confusing user interface design.
- Delayed or inaccurate real-time data updates.

- Mismatch between system functionalities and condo management needs.

Model Kinds:

- Use Case Diagrams: Illustrate interactions between users and the system.
- Activity Diagrams: Represent workflows related to system requirements.

This viewpoint is essential for ensuring that the Condo Management System not only meets user expectations in terms of usability but also fulfills the functional requirements necessary for effective condo management operations i.e. the user stories.

Viewpoint 2: Operational Viewpoint

This viewpoint focuses on the operational aspects, including system reliability, controls, and monitoring. [11]

Concerns:

- System reliability.
- Intuitive controls for effective operation.
- Monitoring tools for identifying and addressing issues.

Typical Stakeholders:

- Condo staff responsible for day-to-day operations.

Anti-concerns:

- Unreliable system leading to downtime.
- Complex or non-intuitive controls hindering efficient operation.
- Lack of monitoring tools leading to delayed issue resolution.

Model Kinds:

- Deployment Diagrams
- Activity Diagram

Viewpoint 3: Development Viewpoint

This viewpoint focuses on the architecture, development, and maintenance aspects of the Condo Management System. [11]

Concerns:

- Alignment of the system with the specific needs of the condo management domain.
- Cost-effectiveness of the system's acquisition and maintenance.
- Clarity in architectural decisions for effective implementation.

Typical Stakeholders:

- Development team
- Project managers
- QA team

Anti-concerns:

- Mismatch between system functionalities and condo management needs.
- Unforeseen high costs in the acquisition and maintenance phase.
- Ambiguous or poorly communicated architectural decisions.

Model Kinds:

- Domain Model
- Class Diagrams
- Component Diagrams

Viewpoint 4: Information Viewpoint

The Information Viewpoint is concerned with the way that the system stores and manages system information. The Condo Management System uses Firebase and its services to store and manage the user catalog, property catalog and condo catalog. [11]

Concerns:

- Information storage
- Information flow

Typical Stakeholders:

- Development team
- QA team

Anti-concerns:

- Information loss
- Inability to retrieve information

Model Kinds:

- Component diagram
- Class diagram
- Deployment diagram

4 Views+

View 1: Design View

The design view is concerned with organizing conceptual entities and addressing the basic overall structure of a system. Therefore, the development team is concerned with developing all the models associated with this view. The Quality Assurance team is concerned with reviewing the models and ensuring they are correct and robust. Lastly, the project managers are concerned with whether these models can be implemented in a time-effective and cost-effective manner. [11]-[12]

Viewpoint (s) governing this view:

- Development Viewpoint
- Information Viewpoint

Model Kinds:

- Domain model diagram
- Class Diagram
- Component Diagram

The domain model diagram, class diagram and component diagram can be seen in *Figure 1*, *Figure 2* and *Figure 3*, respectively.

Known Issues with View:

The model kinds associated with this view (i.e: domain model diagram, class diagram and component diagram) are imperative to understanding the structure of a system. However, they provide no information regarding the flow of control or the sequence of events needed to complete a task.

View 2: Use case view

The use case view is concerned with satisfying user requirements and functionality. Therefore, the use-case view concerns condo residents, property managers and staff members because they are the actors who will directly interact with the system. The development team is also concerned with the use-case view because they are in charge of ensuring the system meets all user requirements (i.e: use cases). [11]-[12]

Viewpoint(s) governing this view:

- Functional Viewpoint

Model Kinds:

- Use-case diagram

The use-case diagram can be seen in *Figure 9*.

Known Issues with View:

A use-case diagram is limited in that it doesn't show the steps a system needs to complete a function nor does it give any indication as to the structure of the system. The use-case diagram simply depicts the use-cases expected to be fulfilled in a system to ensure that user requirements are satisfied.

View 3: Process View

The process view describes the dynamic aspects and run-time behavior of a system. Therefore, the development team is concerned with the process view in order to correctly manage the flow of control of the system. The process view also pertains to all users in order to understand the behavior of the system and how it is meant to be used. [11]-[12]

Viewpoint (s) governing this view:

- Operational viewpoint
- Functional viewpoint

Model Kinds:

- Activity diagrams

Several activity diagrams were produced for Sprint 1 and they can be seen in *Figure 6*, *Figure 7* and *Figure 8*.

Known Issues with View:

Whilst activity diagrams provide a detailed visual representation of a system's workflow, activity diagrams are difficult to maintain (require regular updates) and are not able to accurately represent all the details of real-world complex scenarios. [13]

View 4: Deployment View

Deployment diagrams are a type of structural diagram that illustrate how software components are deployed by one or many different pieces of hardware. The development team is concerned with producing these models in order to organize how their system will be implemented. Company staff members are also concerned with the deployment view in order to understand how to manage the system in their day-to-day operations.

Viewpoint governing this view(s):

- Operational viewpoint
- Information viewpoint

Model Kinds:

- Deployment diagram

The deployment diagram for the Condo Management Application system can be seen in *Figure 5*.

Known Issues with View:

Structural diagrams like deployment diagrams are important to understand the structure of a system. However, they are limited in the fact that they provide no information on the sequence of events needed to complete a task nor object interaction.

5 Consistency and correspondences

5.1 Known inconsistencies

There exists some known inconsistencies between the domain model and the class diagram. Namely, the domain model does not include all attributes present in the class diagram. This inconsistency was done purposefully in order to keep the domain model as simple as possible. Only the most important attributes were included in the domain model and the rest were excluded to present a clear conceptual diagram (one that requires no real technical background) to all stakeholders.

Additionally, the domain model includes role names, association names and additionally association lines that are not present in the class diagram. Once again, this was done purposefully in order to clearly represent how classes conceptually relate to each other.

Another known inconsistency exists in activity diagrams seen in Figure 7 and Figure 8. In order to avoid redundancy, it was decided not to include the entire system flow by excluding the login/sign-in part which , in reality, would be necessary in order to edit a personal profile or add a profile for a property. Therefore, in these activity diagrams, one assumes the user has already logged in and has been successfully authenticated by the system.

5.2 Correspondences in the AD

In general, all software architecture diagrams produced and seen in this report are consistent with each other. Oftentimes, diagrams were used as a foundation for other models. Other times, models were produced in parallel.

For instance, the domain model and the class diagram have the same basic structure and contain the same classes. The type of association (e.g: aggregation or composition) and multiplicity between the classes also remains the same in both the domain model and the class diagram. The use-case diagram contains several use cases which are consistent with the user requirements specification and with the user stories seen in the backlog of our GitHub repository. The deployment diagram and component diagram is consistent with the current implementation of the system during Sprint 1 and both will continue being used as a base for all future sprints. Finally, the activity diagrams are consistent with the current flow of control with the system for Sprint 1. The way system components interact with each other through various interfaces (seen in the component diagram) is also represented in the way one moves from one state of the system to another in the activity diagrams.

The concerns of the stakeholders were taken into account throughout the entire software architecture design process. Every view and its governing viewpoint were associated with one or more specific models to address the concerns of the listed stakeholders.

5.3 Correspondence rules

The software architecture design process followed several correspondence rules:

1) User Requirements

   The user requirements were thoroughly analyzed and used to first create a domain model to provide a conceptual overview of the system. The domain model was used as a base to create the class diagram which contains much more detailed information about the structure of the Condo Management Application. Additionally, given user requirements, the Angular framework was chosen because it is efficient, provides a user-friendly UI and it can be used to convert our web app into a native mobile app (which is an additional feature). All user requirements were successfully taken into account.

2) Software viewpoints and views

   Four software viewpoints were identified during the architecture design process: functional, operational, development and information. These four viewpoints directly affected the four views identified: design view, use-case view, process view and deployment view. These views and their governing viewpoints successfully address the concerns of stakeholders and how they will be handled.

3) Modeling Rules

   During the software architecture design process, several models were produced to represent the structure and behavior of the system including the domain model, class diagram, component diagram, activity diagrams, use-case diagram and deployment diagram. All these models have pre-defined rules, notations and conventions and all were followed to produce correct and robust software documentation of the Condo Management Application. Additionally, the software models were created using GRASP Pattern concepts. There is a clear separation of concerns among modules to promote high cohesion and low coupling. The models also assign a controller (e.g: The Console in the domain model) to handle input system events which can be seen in the domain model. There are also several creators in the system that are assigned the responsibility of creating new instances of classes (e.g: the User Catalog creates new instances of users).

6 Architecture decisions and rationale

6.1 Decisions

Decision	ID-1: Selection of Angular as the Front-End Framework
Statement of the decision	Angular was chosen as the front end framework for the Condo Management System
Correspondences or linkages concerns	Affects users and developers. Influences the system's user interface and overall user experience.
Owner of the decision	Development Leader
Correspondences or linkages to affected AD elements	Linked to the class diagram by representing Angular components. Reflected in the Deployment Diagram.
Rationale linked to the decision	Angular was chosen due to its extensive library and built in components, and adherence to the MVC (Model-View-Controller) architecture. This decision aligns with our team's expertise (from previous school and work projects), ensuring efficient development, maintainability, and scalability of the front-end. Additionally, Angular's component based structure facilitates code organization, facilitating collaboration among developers.
Forces and constraints on the decision	Time constraints required a framework with a short learning curve for the development team. Angular's component-based structure and command line integration facilitates maintainability and cuts development time.

Assumptions influencing the decision	Most of the development team is familiar with Angular and TypeScript.
Considered alternatives and their potential consequences	**React:** React is popular but has a steeper learning curve and more boilerplate code. The team has less experience with React. **Vue.js:** - Vue.js is known for its simplicity but has a smaller online community. [14]

Decision	ID-2: Selection of Firebase for Authentication and Realtime Database
Statement of the decision	Firebase was chosen for user authentication and as the backend storage solution for the Condo Management System.
Correspondences or linkages concerns	Affects users and operators. Impacts user data and system functionality.
Owner of the decision	Development Leader
Correspondences or linkages to affected AD elements	Reflected in Component Diagram by showing the integration of Firebase components as well as the deployment diagram represented as a couple of servers.
Rationale linked to the decision	Firebase provides a serverless architecture, minimizing development time and boiler plate code. Its authentication services offer robust security measures, including multi-factor authentication such as Google Sign In. The integration of Firebase Realtime Database ensures seamless real-time data updates, aligning with the dynamic nature of condominium management system

	requirements. Additionally, most of the development team has experience with Firebase services.
Forces and constraints on the decision	Security concerns required a solution with robust authentication implementation.
Assumptions influencing the decision	Most of the development team is familiar with Firebase services. Firebase's pricing aligns with the project budget as it is free.
Considered alternatives and their potential consequences	**MongoDB:** MongoDB isn't as popular as Firebase and doesn't have as much documentation. **AWS Amplify:** AWS Amplify provides similar services but has more complexities. [15]

Bibliography

[1] “Advanced topic - domain modeling,” Scaled Agile Framework, https://scaledagileframework.com/domain-modeling/ (accessed Feb. 3, 2024).

[2] What is class diagram?, https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/ (accessed Feb. 5, 2024).

[3] What is component diagram?, https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/ (accessed Feb. 5, 2024).

[4] What is deployment diagram?, https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/ (accessed Feb. 5, 2024).

[5] What is activity diagram?, https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/ (accessed Feb. 5, 2024).

[6] “Use-case diagrams,” in UML modeling, https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case (accessed Feb. 5, 2024).

[7] What is use case diagram?, https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (accessed Feb. 5, 2024).

[8] Angular, https://angular.io/guide/libraries (accessed Feb. 5, 2024).

[9] “Continuous integration and continuous delivery (CI/CD) fundamentals,” GitHub Resources, https://resources.github.com/ci-cd/ (accessed Feb. 5, 2024).

[10] “Cypress Automation: 7 key benefits to devops-driven businesses,” TestingXperts, https://www.testingxperts.com/blog/cypress-automation#:~:text=Cypress%20offers%20a%20powerful%20and,React%2C%20Angular%2C%20and%20Vue. (accessed Feb. 5, 2024).

[11] “Viewpoints,” Software Systems Architecture Viewpoints Comments, https://www.viewpoints-and-perspectives.info/home/viewpoints/ (accessed Feb. 5, 2024).

[12] “Architecture in technical perspective view,” GeeksforGeeks, https://www.geeksforgeeks.org/architecture-in-technical-perspective-view/ (accessed Feb. 5, 2024).

[13] “Activity diagram: What is it and how to draw one?,” What Is Activity Diagram, https://boardmix.com/tips/uml-activity-diagram/ (accessed Feb. 5, 2024).

[14] P. Theodosiou, “Comparing vue and react in 2023: Pros and cons,” DEV Community, https://dev.to/ptheodosiou/comparing-vue-and-react-in-2023-pros-and-cons-10nl (accessed Feb. 5, 2024).

[15] D. Team, “Firebase vs mongodb stitch vs aws amplify vs azure mobile apps,” Devathon, https://devathon.com/blog/firebase-vs-mongodb-stitch-vs-aws-amplify-vs-azure-mobile-apps/ (accessed Feb. 5, 2024).