**Sprint 1**

**PewPew Squad**

**Condo Management System Project:**

Condofy

Gevorg Alaverdyan #40202177

Ramy Attalla #40205642

Khashayar Azad #40211574

Sophie Hajjar #40210405

Yeprem Antranik #40204291

Mario El Shaer #40210124

Omar Shehata #40164193

Tarek Elalfi #40197527

Jay Patel #40203705

Khaled Saleh #40210125

07-02-2024

**Table of Contents**

[***1***](#_heading=h.30j0zll) ***Introduction 4***

[**1.1**](#_heading=h.1fob9te) **Identifying information. 4**

[**1.2**](#_heading=h.2et92p0) **Supplementary information 4**

[**1.3**](#_heading=h.49x2ik5) **Other information 6**

[**1.3.1**](#_heading=h.2p2csry) **Overview** 6

[**1.3.2**](#_heading=h.23ckvvd) **Architecture evaluations** 6

[**1.3.3**](#_heading=h.32hioqz) **Rationale for key decisions** 6

[***2***](#_heading=h.41mghml) ***Stakeholders and concerns 6***

[**2.1**](#_heading=h.vx1227) **Stakeholders 6**

[**2.2**](#_heading=h.2u6wntf) **Concerns 7**

[**2.3**](#_heading=h.19c6y18) **Concern–Stakeholder Traceability 9**

[***3***](#_heading=h.nmf14n) ***Viewpoints+ 11***

[**3.1**](#_heading=h.2lwamvv) **Typical stakeholders 12**

[**3.2**](#_heading=h.3l18frh) **Model kinds+ 14**

[**3.3**](#_heading=h.4k668n3) **Correspondence Rules 15**

[**3.4**](#_heading=h.sqyw64) **Operations on views 16**

[**3.6**](#_heading=h.1rvwp1q) **Correspondence rules 18**

[***4***](#_heading=h.3hv69ve) ***Views 19***

[**4.1**](#_heading=h.2afmg28) **Scenarios View 20**

[**4.1.1 Use Cases Related to all Users** 20](#_heading=h.pkwqa1)

[**4.1.2 Use Cases Related to Owners** 22](#_heading=h.39kk8xu)

[**4.1.3 Use Cases Related to Managers** 23](#_heading=h.1opuj5n)

[**4.1.4 Knowns issues** 24](#_heading=h.48pi1tg)

[**4.2 Logical View 24**](#_heading=h.2nusc19)

[**4.2.1 Components** 24](#_heading=h.1302m92)

[**4.2.2 Known issues** 25](#_heading=h.3mzq4wv)

[**4.3 Process View 26**](#_heading=h.2250f4o)

[**4.3.1 Activity diagram** 26](#_heading=h.haapch)

[**4.3.2 Known issues** 26](#_heading=h.319y80a)

[**4.4 Domain Model and Class Diagram 27**](#_heading=h.1gf8i83)

[**29**](#_heading=h.40ew0vw)

[**4.5 Deployment View 30**](#_heading=h.2fk6b3p)

[**4.5.2 Known issues** 31](#_heading=h.upglbi)

[***5***](#_heading=h.3ep43zb) ***Consistency and correspondences 32***

[**5.1**](#_heading=h.2szc72q) **Known inconsistencies 32**

[**5.2**](#_heading=h.1yyy98l) **Correspondences in the AD 33**

# **1 Introduction**

## **1.1 Identifying information.**

The purpose of the Condofy Condo Management System Architect (CCMSAD) is to offer a comprehensive insight into the architecture design of the Condofy Condo Management System, emphasizing key quality attributes: usability, availability, maintainability, and testability. These attributes have been selected due to their critical importance in shaping the design and construction of the system.

## **1.2 Supplementary information**

* **Date of Issue and Status**:
  + February 28th, 2024.
  + Status: v0.5,
* **Authors, Reviewers, Approving Authority, Issuing Organization:** 
  + Authors:
  + Reviewers:
  + Approving Authority:
  + Issuing Organization
* **Change History:** 
  + February 9th: Initial writing of this document awaiting review for further changes and improvements.
* **Summary:**
  + The architecture document for the Condofy Condo Management System (CCMS) provides a detailed overview of its design, emphasizing key attributes such as usability, availability, maintainability, and testability. Covering essential functional requirements, quality attribute scenarios, and architectural insights, it offers a comprehensive understanding of CCMS's architecture.
* **Scope:**
  + Due to the relatively short duration of the project, the team has decided to add some constraints on the project.
    - No need for large scalability as the product will serve customers living in Montréal.
    - No localization, the only language in the application is English.
* **Context**:
  + The Condofy Condo Management System operates within the context of condominium management, addressing the needs and challenges faced by property managers, residents, and other stakeholders. External factors such as regulatory requirements, technological advancements, and market trends influence the system's design and evolution.

* **Version Control Information:**
  + Version Control Mechanism: Git
  + Web Repository Location: <https://github.com/Condofy24/CondoManagement>
  + Mobile Repository Location: <https://github.com/Condofy24/CondofyMobile>
* **References:**

[1] https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

## **1.3 Other information**

### **1.3.1 Overview**

The Condofy Condo Management System Architecture Description (CCMSAD) presents a comprehensive overview of the architecture governing the Condofy Condo Management System. It serves to encapsulate essential points regarding the system's design, structure, and functionality. The overview includes sections addressing the purpose, scope, and context of the architecture, providing readers with a holistic understanding of its significance and implications.

The remainder of this AD is organized to facilitate navigation based on stakeholders and concerns, ensuring that readers can easily access relevant information based on their interests and objectives.

### **1.3.2 Architecture evaluations**

Any evaluations conducted on the Condofy Condo Management System Architecture will be documented, highlighting the findings, insights, and recommendations derived from the evaluation process. These evaluations serve to assess the effectiveness, efficiency, and suitability of the architecture in meeting its intended objectives and addressing stakeholder needs.

### **1.3.3 Rationale for key decisions**

The Condofy Condo Management System Architecture Description (CCMSAD) includes rationale for each decision considered pivotal in shaping the architecture. These rationales provide insights into the thought process, considerations, and trade-offs underlying key architectural decisions, offering transparency and justification for the chosen architectural direction. Adhering to ISO/IEC/IEEE 42010 standards, the inclusion of rationale enhances the comprehensibility and traceability of the architecture description, fostering confidence in the architectural decisions made throughout the development process.

# **2 Stakeholders and concerns**

Identifying stakeholders for the "Condo Management Systems" project is vital. Stakeholders are those with a vested interest or impact on the project. In this section, we pinpoint potential stakeholders, examine their concerns about the project's architecture, and establish a direct link between those concerns and the stakeholders involved. This helps ensure the project addresses everyone's needs, increasing the likelihood of success.

## **2.1 Stakeholders**

1. **Public Users (Condo Owners and condo renters**) **(PU)**: Individuals owning or renting condos, utilizing the system for personal profiles, property information, and community engagement.

2. **Condo Management Companies (CM)**: Organizations managing properties, using the system for streamlined property management, financial records, and communication.

3. **Project Managers (PM):**  is responsible for the planning, execution and the supervision of the project while assuring effective goal-achieving within the limitations of scope, time, and money.

4. **Employees with Different Roles (Staff):** Employees with specific responsibilities within a property, using the system for role-based tasks.

5. **Developers**: Professionals responsible for system design, development, and maintenance across platforms.

6. **Software architect (SA):** designs high-level structures and creates technical specifications for platforms, frameworks, and technology.

7. **Tester:** checks the artifacts and behaviors of the system under validation and verification.

8. **Regulatory Authorities (RA)**: Government bodies oversee compliance with regulations related to the system.

9. **System Administrators (Sysadmin)**: Individuals managing system configuration, maintenance, and monitoring.

10. **Accountants (ACCT)**: Professionals reviewing financial reports for accuracy and compliance.

## **2.2 Concerns**

1. **Purpose(s) of the System-of-Interest:**

**1.1 User Management and Engagement:**

* Ensuring users can easily create and manage their profiles.
* Facilitating communication and interaction between condo owners, rental users, and condo management companies.

**1.2 Property and Financial Management:**

* Streamlining the management of condo properties, including financial aspects.
* Providing tools for condo management companies to handle property-related tasks efficiently.

**1.3 Operational Efficiency:**

* Improving the efficiency of daily operations for condo management companies through features like reservation systems, request management, and role-based access controls.

**1.4 Request Management:**

* Provide a streamlined and user-friendly mechanism for condo owners to submit various types of requests.

1. **Feasibility of Construction and Deployment:**

**2.1 Technical Feasibility:**

* Ensuring the technical aspects of building the system, including the integration with external services, are feasible.

**2.2 Financial Feasibility:**

* Evaluating the cost-effectiveness of constructing and deploying the system within budget constraints.

**2.3 Resource Feasibility:**

* Assessing the availability of skilled resources and technologies required for development.

**2.4 Time Feasibility:**

* Developing a realistic timeline for construction and deployment, considering potential challenges and dependencies.

1. **Potential Risks and Impacts:**

**3.1 Security Risks:**

* Identifying potential security vulnerabilities in user data, financial information, and files.
* Technical: Inadequate security measures can lead to data leaks.
* Technical: Nonprotected routes in backend can lead unauthorized access.

**3.2 Operational Risks:**

* Schedule: Ambiguous activity diagrams may result in delays due to workflow misunderstanding.

**3.3 Privacy Concerns:**

* Ensuring compliance with privacy regulations to avoid legal and reputational risks.

**3.4 Management risks:**

* Unclear vision in product statement can lead to misalignment among team members.

1. **Maintenance and Evolution:**

**4.1 System Maintenance:**

* Establishing a plan for ongoing system maintenance, including bug fixes, updates, and security patches.

**4.2 Evolution and Upgrades:**

* Planning for future enhancements and upgrades to meet changing user needs and technology advancements.

**4.3 Scalability:**

* Ensuring the system architecture allows for scalability to accommodate future growth.

**4.4 Regulatory Compliance:**

* Staying informed about and adapting to changes in regulations affecting condo management systems.

## **2.3 Concern–Stakeholder Traceability**

This association between potential Stakeholders and the potential concerns is recorded through the following simple table. Showing every Stakeholder and their concerns.

Table 2.1: showing association of stakeholders to concerns in an AD. The upper row represents Stakeholders from section 2.1 (labeled from 1 to 10 with their abbreviations) and the Leftmost column represents the Concerns from section 2.2(labeled from 1.1 to 4.4)

| 2.2/2.1 | PU | CM | PM | STAFF | DEV | SA | Tester | RA | Sysadmin | ACCT |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1.1 | X | X |  |  |  | X | X |  |  |  |
| 1.2 | X | X |  |  |  |  |  |  |  | X |
| 1.3 | X | X | X | X | X | X | X |  | X | X |
| 1.4 | X | X | X | X | X | X | X |  | X |  |
| 2.1 |  |  | X |  | X | X |  |  |  |  |
| 2.2 |  |  | X |  |  |  |  | X | X |  |
| 2.3 |  |  | X |  |  | X | X |  | X |  |
| 2.4 |  | X | X |  | X |  |  |  |  |  |
| 3.1 | X | X | X |  | X | X | X |  |  |  |
| 3.2 |  | X | X |  | X | X | X |  |  | X |
| 3.3 | X |  |  |  | X | X | X | X |  |  |
| 3.4 |  | X | X |  | X |  |  | X |  |  |
| 4.1 |  |  |  |  | X | X |  |  | X |  |
| 4.2 |  |  |  |  | X | X | X |  | X |  |
| 4.3 |  | X |  |  | X | X | X | X | X |  |
| 4.4 | X | X | X |  |  |  |  | X | X | X |

# 

# **3 Viewpoints+**

* **Functional Viewpoint**
  + **Concerns Addressed**: User Management and Engagement (1.1), Property and Financial Management (1.2), Request Management (1.4).
  + **Rationale**: This viewpoint focuses on the functionality of the system and how it meets user needs. It's relevant for stakeholders like condo owners, rental users, and condo management companies. The models can include use case diagrams and activity diagrams, which are essential for understanding user interactions and process flows.

**Information Viewpoint**

* + **Concerns Addressed**: Operational Efficiency (1.3), Technical Feasibility (2.1).
  + **Rationale**: This viewpoint addresses how data is managed and utilized within the system. It's crucial for stakeholders concerned with data integrity, such as condo management companies. Models like Activity diagrams will help in understanding data relationships and flow.

**Concurrency Viewpoint**

* + **Concerns Addressed**: Operational Efficiency (1.3).
  + **Rationale**: It focuses on the system's performance in terms of handling multiple operations simultaneously, which is critical for features like reservation systems and request management. This viewpoint is relevant for technical stakeholders concerned with system performance.

**Development Viewpoint**

* + **Concerns Addressed**: Technical Feasibility (2.1), Resource Feasibility (2.3), Time Feasibility (2.4).
  + **Rationale**: It covers the system's software architecture, tools, and practices, essential for assessing the feasibility of construction and deployment. Stakeholders include developers and project managers. Models like component diagrams can be used.

**Security Viewpoint**

* + **Concerns Addressed**: Security Risks (3.1).
  + **Rationale**: This viewpoint is critical for identifying and mitigating security vulnerabilities. It's essential for stakeholders concerned with data protection, including users and legal teams. Models like threat models and security protocols are relevant.

**Operational Viewpoint**

* + **Concerns Addressed**: Operational Risks (3.2), Privacy Concerns (3.2), System Maintenance (4.1).
  + **Rationale**: It deals with the operational aspects of the system, including deployment, monitoring, and maintenance. Stakeholders include operations staff and IT support teams. This viewpoint can utilize models like deployment diagrams.

**Economic Viewpoint**

* + **Concerns Addressed**: Financial Feasibility (2.2).
  + **Rationale**: This viewpoint focuses on the economic aspects of the project, such as budgeting and cost-effectiveness. It's vital for financial stakeholders and project sponsors. Models like cost-benefit analysis and financial projections are used.

**Scalability and Evolution Viewpoint**

* + **Concerns Addressed**: Evolution and Upgrades (4.2), Scalability (4.3), Regulatory Compliance (4.4).
  + **Rationale**: It addresses the system's ability to evolve and scale over time. This viewpoint is crucial for long-term planning and is of interest to strategic planners and regulatory compliance teams. Models like scalability architecture and roadmap plans can be used.

## **3.1 Typical stakeholders**

* **End Users**
  + **Condo Owners**: Interested in features related to property management, financial transactions, and request submissions.
  + **Rental Users**: Concerned with user engagement features, ease of access, and interaction with condo management.

**Condo Management Companies**

* + **Property Managers**: Focus on operational efficiency, property management features, and financial management tools.
  + **IT Staff**: Concerned with the technical feasibility, security aspects, and system maintenance.

**Developers and Technical Team**

* + **Software Developers**: Interested in the development viewpoint, focusing on system architecture, technology stack, and integration with external services.
  + **System Architects**: Focus on the overall system design, scalability, and evolution.

**Project Management**

* + **Project Managers**: Concerned with resource, time, and financial feasibility, as well as risk management.
  + **Quality Assurance (QA) Team**: Interested in system reliability, security, and compliance with requirements.

**Security Experts**

* + **Cybersecurity Professionals**: Focus on security risks, data protection, and compliance with privacy regulations.

**Financial Stakeholders**

* + **Investors and Budget Managers**: Interested in the economic viewpoint, particularly financial feasibility, and cost-effectiveness.

**External Partners and Service Providers**

* + **Third-Party Service Providers**: Engage with integration points and external interfaces of the system.
  + **Regulatory Bodies**: Interested in how the system complies with industry standards and regulations.

## **3.2 Model kinds+**

* **Functional Viewpoint**
  + **Model Kind**: Use Case Diagrams, Activity Diagrams.
  + **Conventions**:
    - **Use Case Diagrams**: Illustrates system functionality in terms of actors (users) and use cases (actions).
    - **Activity Diagrams**: Shows the flow of activities and actions. They include activity states, transitions, decision nodes, and parallel processing (fork/join).

**Information Viewpoint**

* + **Model Kind**: Class Diagram
  + **Conventions**:
    - **Class Diagram**: Displays the data model of the system, focusing on entities, attributes, and relationships.

**Concurrency Viewpoint**

* + **Model Kind**: Activity Diagrams.
  + **Conventions**:
    - **State Diagrams**: Represents the states of a component or system and transitions based on events.
    - **Sequence Diagrams**: Illustrates how processes operate with one another and in what order.

**Development Viewpoint**

* + **Model Kind**: Component Diagrams.
  + **Conventions**:
    - **Component Diagram**: Shows the organization and dependencies among a set of components.

**Security Viewpoint**

* + **Model Kind**: Threat Models, Security Protocols.
  + **Conventions**:
    - **Threat Models**: Identifies potential threats and vulnerabilities.
    - **Security Protocols**: Diagrams showing the steps and processes for secure communications.

**Operational Viewpoint**

* + **Model Kind**: Deployment Diagrams.
  + **Conventions**:
    - **Deployment Diagrams**: Show the physical deployment of artifacts on nodes.

**Economic Viewpoint**

* + **Model Kind**: Cost-Benefit Analysis Diagrams.
  + **Conventions**:
    - **Cost-Benefit Analysis Diagrams**: Compare the costs and benefits of different approaches. They include costs, benefits, and timelines.

**Scalability and Evolution Viewpoint**

* + **Model Kind**: Roadmap Plans.
  + **Conventions**:
    - **Roadmap Plans**: Illustrates the planned evolution of the system. They include milestones, feature enhancements, and timelines.

## **3.3 Correspondence Rules**

**Use Case Diagrams & Activity Diagrams**

* **Correspondence Rules**:
  + Activities in an activity diagram should correspond to actions within a use case in the use case diagram.
  + Actors in use case diagrams should align with external entities or roles represented in activity diagrams.

**State Diagrams & Sequence Diagrams**

* + **Correspondence Rules**:
    - Objects in sequence diagrams should match the entities whose states are depicted in state diagrams.
    - Transitions in state diagrams should reflect the sequence of messages exchanged in sequence diagrams.

## **3.4 Operations on views**

* **Functional Viewpoint**
  + **Construction Methods**: Use templates for use case and activity diagrams. Identify key functionalities and actors. Apply user-centered design principles.
  + **Interpretation Methods**: Understand how user requirements are translated into system functions. Look for completeness in coverage of functionalities.
  + **Analysis Methods**: Evaluate the alignment of use cases with user needs. Check for consistency between activity and use case diagrams.
  + **Implementation Methods**: Translate use cases and activities into system requirements and design specifications.

**Information Viewpoint**

* + **Construction Methods**: Utilize Activity diagram templates. Identify key data entities and their relationships. Apply data normalization principles.
  + **Interpretation Methods**: Analyze data flow and entity relationships to understand information management.
  + **Analysis Methods**: Check for data integrity and consistency. Validate against data requirements.
  + **Implementation Methods**: Design database schemas and data processing algorithms.

**Concurrency Viewpoint**

* + **Construction Methods**: Develop state and sequence diagrams based on system operations. Identify concurrent processes.
  + **Interpretation Methods**: Understand the interaction and life cycle of system components.
  + **Analysis Methods**: Evaluate system performance and identify potential bottlenecks or deadlocks.
  + **Implementation Methods**: Implement concurrent processing mechanisms in system design.

**Development Viewpoint**

* + **Construction Methods**: Create component diagrams and Gantt charts. Identify system components and dependencies. Plan development phases.
  + **Interpretation Methods**: Understand the architectural structure and project timeline.
  + **Analysis Methods**: Assess component dependencies and project risks. Ensure alignment with project timelines.
  + **Implementation Methods**: Guide system development and project management based on planned components and schedules.

**Security Viewpoint**

* + **Construction Methods**: Develop threat models and security protocols. Identify potential vulnerabilities and mitigation strategies.
  + **Interpretation Methods**: Understand security requirements and measures.
  + **Analysis Methods**: Evaluate security coverage and potential risks. Ensure compliance with security standards.
  + **Implementation Methods**: Implement security measures in system design and operations.

**Operational Viewpoint**

* + **Construction Methods**: Develop deployment diagrams and operational workflows. Identify deployment configurations and operational processes.
  + **Interpretation Methods**: Understand system deployment and operational procedures.
  + **Analysis Methods**: Assess operational efficiency and system scalability.
  + **Implementation Methods**: Guide system deployment and operational management.

**Economic Viewpoint**

* + **Construction Methods**: Perform cost-benefit analysis and financial projections. Identify financial constraints and opportunities.
  + **Interpretation Methods**: Understand the economic feasibility and financial implications of the system.
  + **Analysis Methods**: Evaluate financial viability and return on investment.
  + **Implementation Methods**: Use economic insights to inform decision-making and prioritization.

**Scalability and Evolution Viewpoint**

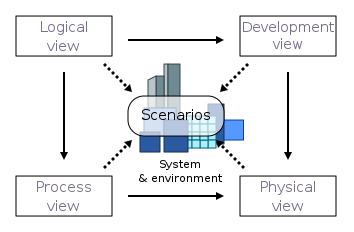
* + **Construction Methods**: Develop scalability architecture diagrams and roadmap plans. Identify scalability requirements and future enhancements.
  + **Interpretation Methods**: Understand the scalability and evolution strategy of the system.
  + **Analysis Methods**: Assess the system's ability to scale and adapt to future needs.
  + **Implementation Methods**: Guide the design of scalable systems and plan for future upgrades.

## **3.6 Correspondence rules**

Correspondence rules across architectural viewpoints ensure coherence within a system's architecture by linking use cases to activities, entities to data flows, sequence interactions to state changes, component dependencies to project plans, threat models to security measures, deployment configurations to operational processes, financial assumptions to economic evaluations, and scalability plans to future enhancements. These rules guarantee that different architectural perspectives are consistently aligned, supporting the system's comprehensive and integrated design.

# **4 Views**

For this product we have decided to go with the 4+1 View Model of Architecture [1] to describe the views of Condofy. Each view relates back to the previously mentioned viewpoints and the stakeholders’ concerns.



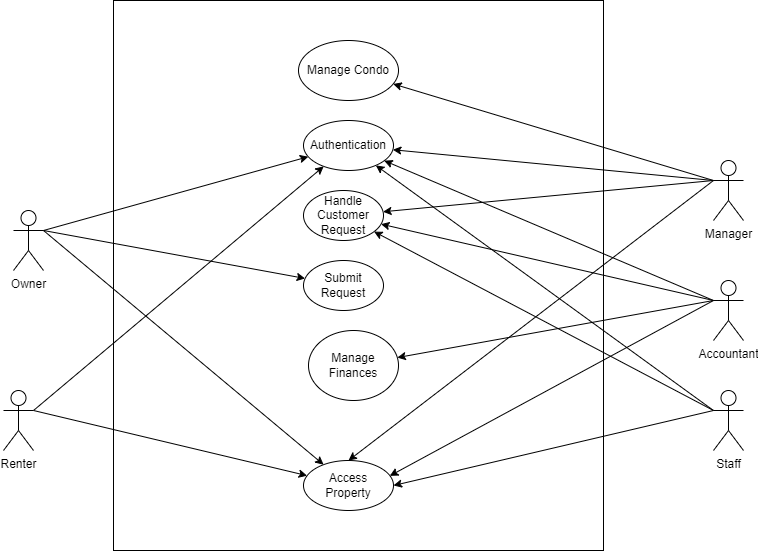
*Figure 1: 4+1 View Model of Architecture*

## 

## 

## **4.1 Scenarios View**

In the scenarios view we cover various use cases that exist in the application. The entire system can be represented in a use case diagram that represents all the actions taken by different types of users. The scenarios’ view is related to the functional viewpoint, which represents the different actors and the actions they can perform within the context of the application.



### **4.1.1 Use Cases Related to all Users**

For the following uses cases the “User” is the public user (condo owner/renter), employees with different roles (including managers) and accountants. These users make use of all the functionalities in the application, some have different requirements than others. However, the following use cases relate to all of them.

* **Authentication**: All users of Condofy must be able to sign in to their accounts securely. Since all users have confidential data in their accounts, security is a main concern being addressed here. Recalling the security viewpoint, authentication has been included in our system design.
* **Registration**: All users must be able to sign up for Condofy or be signed up for the application, in the case of employees. Recalling the security viewpoint, registration has been included in our system design.

A diagram of a authentication process

Description automatically generated

*Figure 2: Use Case Diagram for Authentication*

### **4.1.2 Use Cases Related to Owners**

For these use cases “User” refers to a condo owner and “employee” refers to a user that works for a management company.

**User Request:** Users must be able to submit requests that have to be handled by an employee.

A diagram of a customer request

Description automatically generated

*Figure 3: Use Case Diagram for Customer Request*

**Access condo:** Users must be able to view their properties and their relevant information.

A diagram of a access property

Description automatically generated

*Figure 4: Use Case Diagram for Access Property*

### **4.1.3 Use Cases Related to Managers**

For these use cases “Manager” refers to a manager at a condo management company.

**Manage condos:** Managers must have access to delete, add and update condos. This information is confidential, and the data must not be accessible by anyone else, which represents the security viewpoint.

A diagram of a diagram

Description automatically generated

*Figure 5: Use Case Diagram for Managing Condo*

### **4.1.4 Knowns issues**

Various issues exist for the scenarios view:

**Incomplete**: Some scenarios lack important information due to the diagrams being very high level. For instance, the registration does not accurately represent how employees get registered differently than public users.

**Ambiguous**: Some scenarios could also be unclear for certain stakeholders, and instead of offering a bridge between development and stakeholders it could cause confusion.

## **4.2 Logical View**

This view focuses on the functionalities of the system on a high level. The logical view represents all the components of the system. The development and information viewpoints are being represented here, as we focus on the flow of data between components and how components interact together.

The following component diagram represents the entire system.

### **4.2.1 Components**

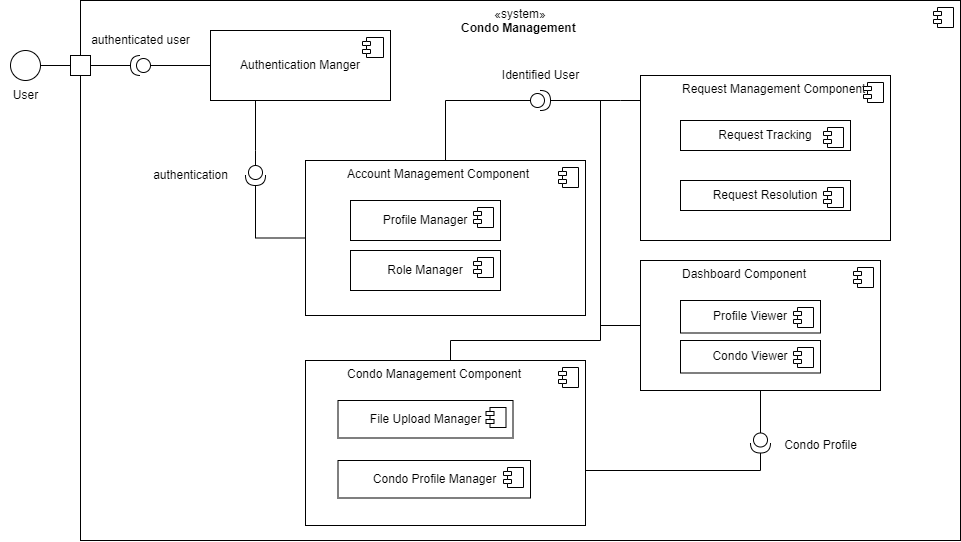
**Authentication Manager:** This component is responsible for handling all the requests relating to signing in, out and registering as any type of user. The authentication manager is also responsible for sending tokens that are going to be used in other components to ensure that the user has the correct privileges or is properly logged in.

**Account Management Component:** This component is responsible for handling all profiles and roles for all types of users.

**Condo Management Component:** This component is responsible for handling file uploading and the profiles for every condo.

**Request Management Component:** This component is responsible for handling the requests sent my owners. The component is also responsible for showing employees the requests and handling their responses.

**Dashboard Component:** This component is responsible for viewing the user’s profile and all their condos.



*Figure 6: Component Diagram*

### **4.2.2 Known issues**

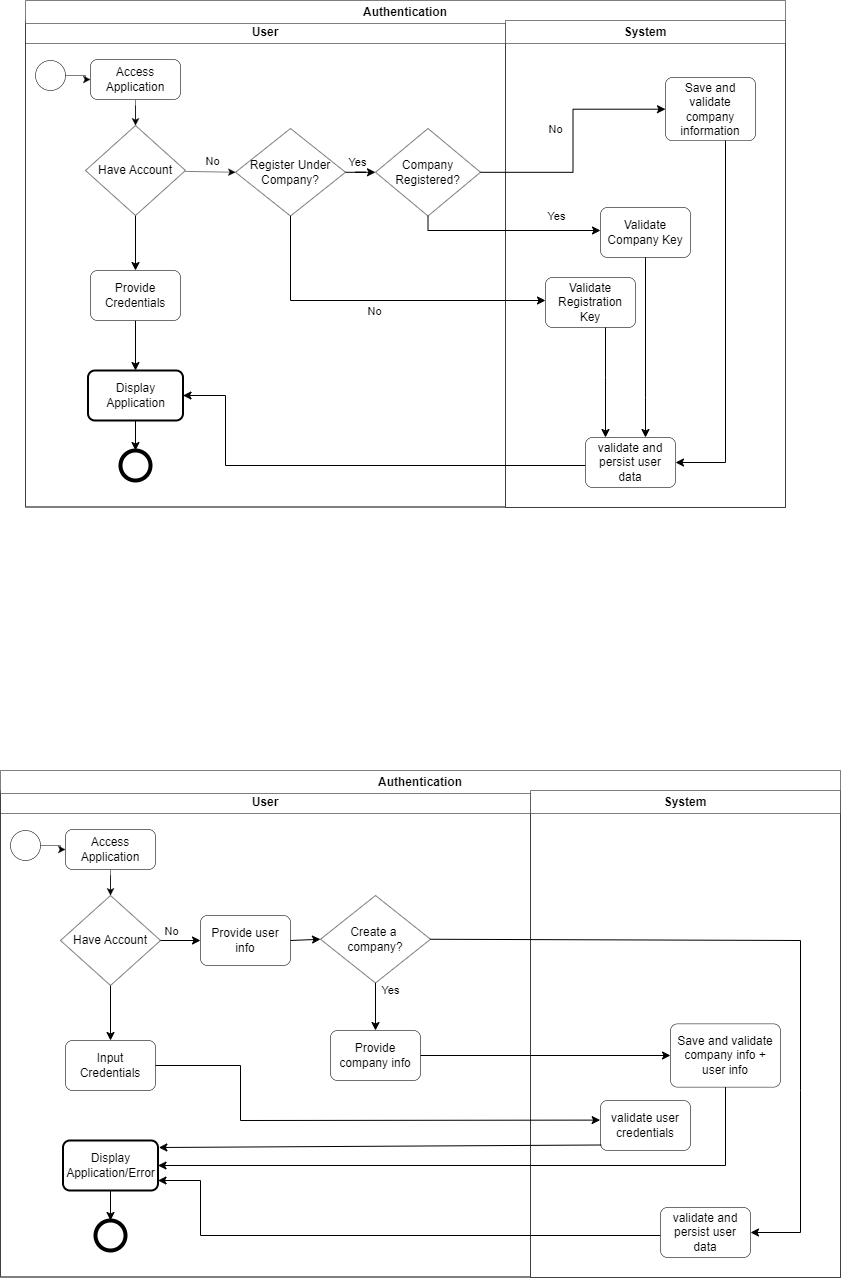
**Incomplete**: Component diagrams are high level and as such there is always some information that is not represented. For instance, the dashboard component should somehow show that different users will have different versions of this component. The UI will be different on the frontend and the backend handles different requests.

**No accurate division**: Component diagrams do not accurately represent how components are divided. For example, there is no clear division between what gets handled on the server side and what gets handled on the client side.

## **4.3 Process View**

This view focuses on the processes performed by the system in run-time. Since this view focuses on concurrent task, we can see that it represents the concurrency viewpoint. The process view can be represented using an activity diagram, which shows the overall flow of different scenarios.

### **4.3.1 Activity diagram**



*Figure 7: Activity Diagram*

### **4.3.2 Known issues**

**Unrealistic flow:** It is possible that sometimes the flow of a certain process is unrealistic and cannot be implemented the same way due to technical constraints.

**Complex**: Complex process can lead to complex activity diagrams, which end up being confusing instead of helpful design tools.

## **4.4 Domain Model and Class Diagram**

To develop a high-level understanding of the system, a domain model is used to represent entities and how they interact with one another. The class diagram gives a more detailed design for the system, showing attributes for entities and how they interact with one another.

Some important standard notations followed in this design:

* Arrows show visibility; where an arrow from A points to B implies A sees B. (e.g. Building has the id of the company it is managed by)
* \* Means 0 to many multiplicity
* The domain model does not show attributes. The class diagram shows attributes.

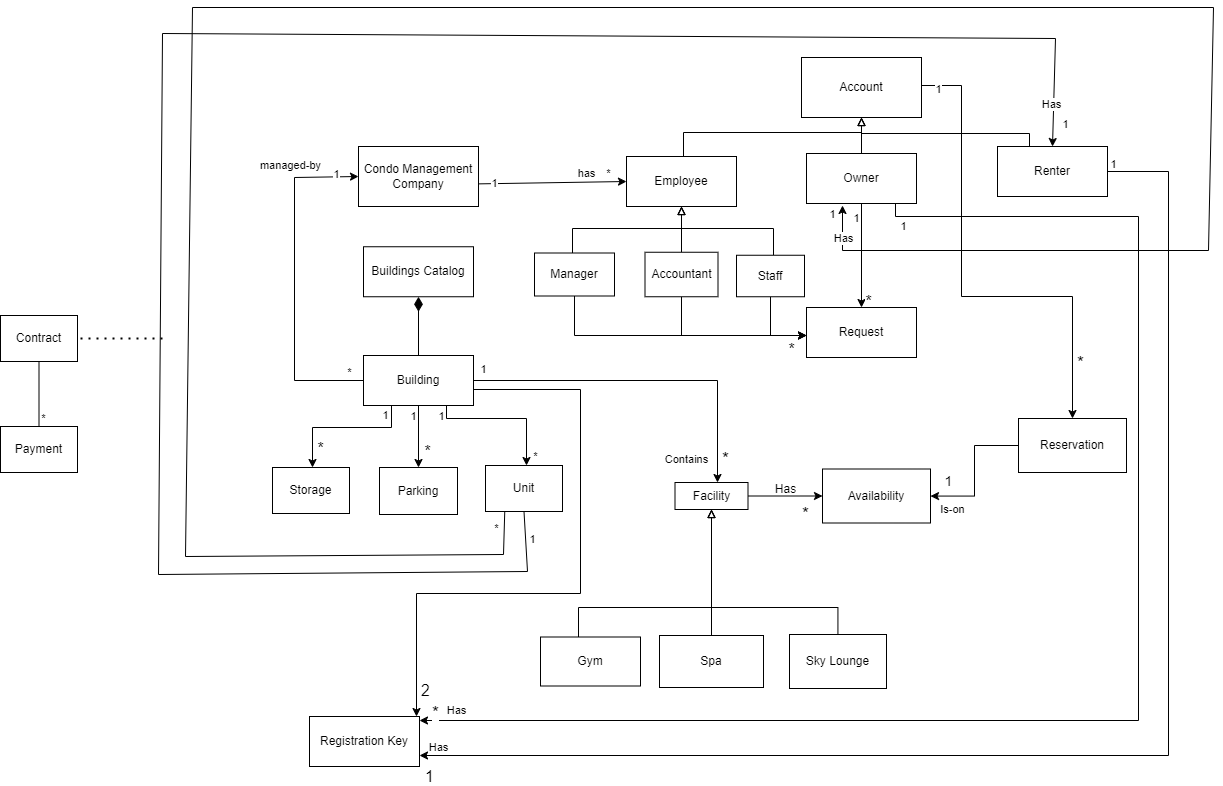
The domain model identifies 3 main types of users, owners, renters, and employees. Employees (Staff, Accountant and Manager as defined in the Class Diagram) are linked to condo management companies. Renters are bound by contracts.

Companies are created by managers. During the signing up process of the first manager, they create a company. At this point the company is being managed by one manager, who can create other employees (including managers). This allows for better security and avoids having to send token to employees to join their company.

Owners and renters can only sign up when they have a registration key provided to them from the condo management company. The registration key is a key that connects the user’s profile to a unit. Essentially, a public user of the application should always be a renter or owner. This allows the system to be more secure, as it prevents non-residents and spammers from signing up to the website. The registration key also tells the system the role of the user, as the manager gives renters and owners different keys that can be recognized by the system. Employees have access to requests so they can address owners’ concerns.

The companies can manage many buildings (initially none) and each building has many storages, parking spots and units (initially none). The reason everything is initially defaulted to none is to allow the manager to fully customize the company and manage the properties as they see fit. The class diagram has unitCount, parkingCount, storageCount so that it can track how many assets a building has.

Buildings have facilities (Gym, Spa, and Sky lounge), which have many availabilities. Reservations have one availability taken each.



## 

## **4.5 Deployment View**

This view is responsible for presenting the physical deployment of the application and the different nodes on which it exists and how they interact.

A diagram of a computer

Description automatically generated

*Figure 10: Deployment Diagram*

Condofy is a restful web application that serves dynamic content to clients, which primarily include web browsers on various devices capable of processing JavaScript. It operates on a robust server configured with the latest Node.js environment to manage high-volume data transactions and computational tasks efficiently.

The server, hosted in a centralized location with a dedicated network setup, is optimized for the application's operational demands without the immediate need for load balancing, given its current geographic and user-load constraints. The NestJS application on the server is meticulously structured with organized business logic, carefully selected middleware for authentication and logging, and efficient route handlers tailored for quick processing of HTTP requests and responses.

Direct interaction with the NoSQL database, MongoDB, is facilitated by the MongoDB Node.js Driver, allowing for agile development and flexible data schema modifications. The choice of a NoSQL database underpins the system's capacity for rapid scalability and cost-effective data management, ready to adapt to growing user demands.

The server architecture is designed with future growth in mind, allowing for seamless horizontal scaling by integrating additional servers as needed. Additionally, the system incorporates strategic data backup solutions to maintain data integrity and provides a framework for regular security updates and audits to safeguard against potential threats.

### **4.5.2 Known issues**

**Incomplete**: deployment view is very high level and as such omits details that could be used to certain architectures.

**No alignment with other diagrams**: the deployment diagram used for this view does not go with any other diagram used for the user views and so it’s difficult to see how it fits in.

# **5 Consistency and correspondences**

As the software development industry continues to progress, a system's architecture serves as the fundamental framework for its present capabilities and future expansion. In this segment, we carefully examine the consistency requirements of Condofy, clearly stating any existing inconsistencies and outlining correspondences with precision. By doing so, we guarantee that the architecture description is precise, reliable, and consistent, all of which are essential for the successful development of the system.

## **5.1 Known inconsistencies**

**Verification Process**

The absence of a verification process in the class diagram is a notable inconsistency, given the importance of this feature in the system's requirements. A verification process is critical to ensure that only authorized users, such as condo owners and renters, can access sensitive information and functionalities within the software. The class diagram should ideally reflect this process, possibly with a dedicated class or association that handles the verification logic, including the use of registration keys provided by the condo management company. This would typically involve classes or methods that validate the registration keys against the user profiles, linking them to their respective units, parking spots, or storage units. Without this explicit representation, the diagram lacks a clear depiction of a fundamental security feature, which could lead to ambiguity in the implementation phase and potential security vulnerabilities in the system.

**Employee Roles**

The class diagram's depiction of a generic "Employee" class presents an inconsistency by not differentiating between the specific roles of Manager, Accountant, and Staff as outlined in the system requirements. This lack of distinction is significant because each role likely has different permissions, responsibilities, and interactions with the system's features. For instance, an Accountant may need access to financial records and the ability to generate reports, whereas a Manager might require broader system access for overseeing operations. Without clearly defined classes for these roles in the diagram, developers may not implement the necessary role-based access control within the system, potentially leading to inadequate security measures and inefficient workflow management. This could result in unauthorized access to sensitive information and functions, compromising the system's integrity and operational efficiency.

**Request Assignment**

The class diagram's lack of a clear mechanism for assigning requests to specific employees based on the request type is a significant oversight. This functionality is crucial for ensuring that workflow within the condo management system is efficient and that requests are handled by the most qualified personnel. Without this, there could be confusion and delays in addressing requests, as well as a potential mismatch of tasks and employee expertise. The importance of this mechanism lies in its impact on service quality, response time, and overall user satisfaction, as well as the internal organization of the condo management company's operations. Implementing a clear assignment process in the system's design would enable a more streamlined, accurate, and accountable approach to managing and fulfilling user requests.

## **5.2 Correspondences in the AD**

In the class diagram, there is a clear correspondence between the "Account" class and the "Owner" and "Renter" subclasses. This architectural relationship captures the essential link between a user account and the role it plays within the system. The rule here dictates that every account must be classified either as an owner or a renter, which is a crucial distinction for access control and feature availability within the system. This rule holds in the architecture description, as the connection is explicitly drawn, ensuring that user accounts are properly associated with their respective ownership or rental status.

The property management aspect of the system is well-represented through the relationship between the "Condo" class and the "Company" class, further branching into "Unit," "Parking," and "Storage" classes. The correspondence rule in this context ensures that each condo is managed by the company and is linked to its constituent units and amenities, which is clearly maintained in the architecture description. This ensures that the condo management company has the necessary oversight and data organization structure to manage property details effectively.

As observed in the class diagram, the "Condo" and "Account" linkage through the "Owner" subclass establishes a direct relationship between the property and its owner, reflecting the ownership status within the system. The correspondence rule here mandates that every condo must be associated with an owner account, and this rule is observed in the architecture description, providing a reliable reference for ownership details and related functionalities. However, to align with financial transaction requirements, it would be prudent to integrate financial attributes directly into this relationship, enhancing the system's financial tracking capabilities.