Software Architecture Description of the Condo Management System

Sprint 2 Document

Software Architecture

Table of Contents

Software Architecture **2**
1 - Introduction **3**
1.1 - Identifying information 3
1.2 - Rationale for key decisions 3
2 - Stakeholders and Concerns **5**
3 - Views **7**
3.1 - Scenarios 7
3.1.1 - Use cases for “Public User” and “Condo Owner” 7
3.1.2 - Use cases for “Condo Management Company” 8
3.2 - Logical View 9
3.2.1 - Domain Model 9
3.3 - Process View 10
3.3.1 - Activity Diagram for Reservation Process 10
3.4 - Development View 11
3.4.1 - Class Diagram for User Management 11
3.4.2 - Component Diagram 12
3.5 - Deployment View 12
3.5.1 - Deployment Diagram 12
3.6 - Known Issues with Views 13
4 - Architectural Decisions **13**
4.1 - Front-end Technologies 13
4.1.1 - React 13
4.1.2 - NodeJS 13
4.2 - Back-end Technologies 14
4.2.1 - C# 14
4.2.2 - .NET 14
4.2.3 - Entity Framework Core 14
4.2.4 - Microsoft SQL Server 14
References **15**

1 - Introduction

1.1 - Identifying information

The architecture being expressed in our case is a condo management system (CMS). Our tech stack includes programming languages, frameworks, a database, as well as front-end and back-end tools.

For the front-end, we are using React to build a progressive web app, along with NodeJS. Our back-end is an ASP.NET Web API written in C#, using Entity Framework Core and Microsoft SQL Server to compose the database. All of these technologies are part of the overall .NET coding environment.

The system of interest for which this document is written is our condo management system. Indeed, we are building a condo management app as well as its corresponding website. An important feature that our application possesses is its accessibility on major operating systems as a PWA, either as a website or downloadable web app.

1.2 - Rationale for key decisions

The specific technologies used for this architecture were chosen based on all the team members' experiences from previous projects. Most importantly, they were chosen for their compatibility.

Front-End Technologies	**Reasoning**
React	React was selected for its component-based architecture, which enables modular and reusable code. This framework simplifies the development of dynamic and interactive user interfaces, making it well-suited for creating the progressive web app aspect of our condo management system. Its wide adoption and robust community support ensure an abundance of resources and tools, facilitating development and problem-solving.
NodeJS	NodeJS was chosen for its efficiency and scalability, thanks to its non-blocking, event-driven architecture. It serves as the backbone for our React-based front-end, providing a lightweight and flexible server-side platform that can handle multiple connections simultaneously without

	compromising performance. NodeJS's extensive npm ecosystem allows easy integration of packages and tools, enhancing our development workflow and capabilities in building a progressive web app.

Back-End Technologies	**Reasoning**
C#	This programming language was chosen because it can be easily run on the .NET Framework, being its main language.
.NET	This framework is the main dependency for all other back-end technologies used. It provides the environment that allows us to develop using the Microsoft suite of tools.
Entity Framework Core	EF Core was chosen to handle .NET entities. This object-database mapper connects with a database such as Microsoft SQL Server and handles data transactions while minimizing the need to write in the SQL language.
Microsoft SQL Server	Microsoft SQL Server was chosen for its integration with most operating systems, which avoids the possibility of any compatibility issues.

2 - Stakeholders and Concerns

- **Public User**: A new user who registers and makes use of the condo management app to obtain data, submit requests, and engage with the system.

Category	Concerns
Purpose	Creating profiles, and personal data is secure and easily registered using the condo management company's registration key.
Suitability	User-friendly interface for profile management.
Feasibility	The app's accessibility and availability across multiple platforms.
Risks/Impacts	Security and privacy issues about profile data.
Maintenance/Evolution	Updating and providing user support.

- **Condo Owner**: The owner of a condo unit, uses the app to manage their properties, monitor financial data, make reservations, and submit requests.

Category	Concerns
Purpose	Easy request submission, accurate financial information, and effective management of condo units.
Suitability	A user-friendly dashboard of the property with general information and a property overview.
Feasibility	Availability across a range of platforms and devices.
Risks/Impacts	Accuracy of data and security of financial information.
Maintenance/Evolution	Regular upgrades for new features and refinements.

- **Condo Management Company**: The organization in charge of managing several properties. It creates property profiles, uploads files, manages financial aspects, and sets up common facilities and roles for employees.

Category	**Concerns**
Purpose	Appropriate personnel work assignment, financial tracking, and property management.
Suitability	Financial tools and a comprehensive property profile management system.
Feasibility	Compatibility across several platforms and integration with banking systems.
Risks/Impacts	Possible financial inequalities and data security.
Maintenance/Evolution	Ongoing assistance with system updates and personnel responsibility management.

- **Rental User**: The tenant of a rented condo unit. They can review their financial information, make reservations, submit requests, and manage their rental properties using the app.

Category	**Concerns**
Purpose	Easy request submission, accurate financial information, and effective management of rented units.
Suitability	Easy-to-use dashboard with financial information and a summary of the rental unit.
Feasibility	Availability across a range of platforms and devices.
Risks/Impacts	Reliability of data and financial information security.
Maintenance/Evolution	Regular updates for new features and improvements.

3 - Views

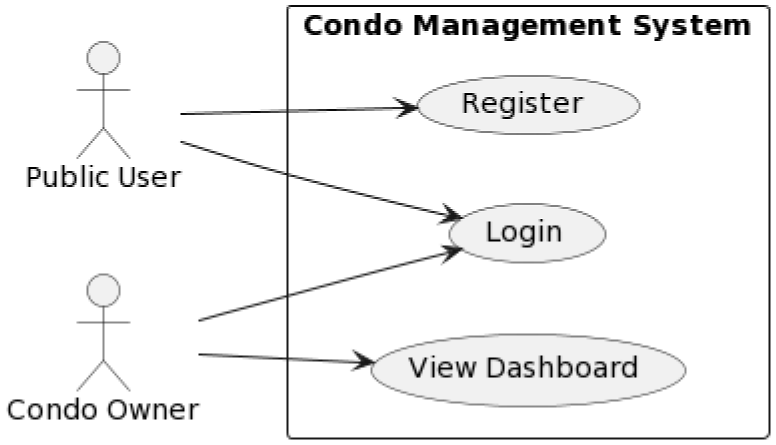
The Condo Management System (CMS) is described here using the 4+1 architectural view model [4]. This model contains five viewpoints: scenarios (use-case view), the logical view, the process view, the development view, and the deployment view.

3.1 - Scenarios

The Use Case View is important to capture the functional requirements of the CMS and demonstrate the interactions between the system and its users.

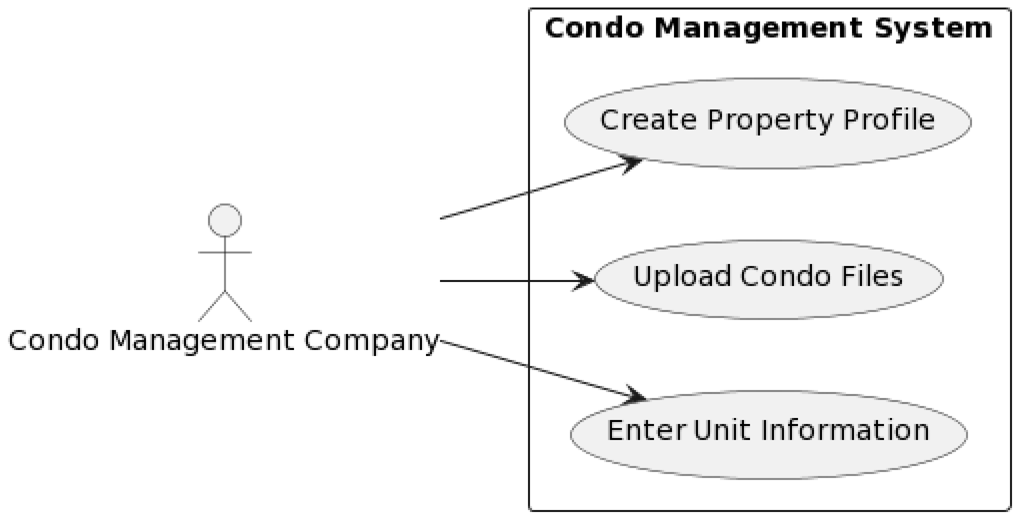
3.1.1 - Use cases for “Public User” and “Condo Owner”

- Registration: Allows a Public User to create a unique profile using a registration key provided by their condo management company.
- Login: Users can log in to the system using their credentials or Single Sign-On (SSO) with Gmail or other accounts.
- Dashboard: Condo Owners can access a dashboard displaying their profile, property details, financial status, and request statuses.



3.1.2 - Use cases for “Condo Management Company”

- Property Profile Creation: Allows the creation of a property profile with essential details.
- Condo File Upload: Management can upload files accessible to condo owners.
- Unit Information Entry: Detailed information for each unit, parking spot, and locker can be entered.

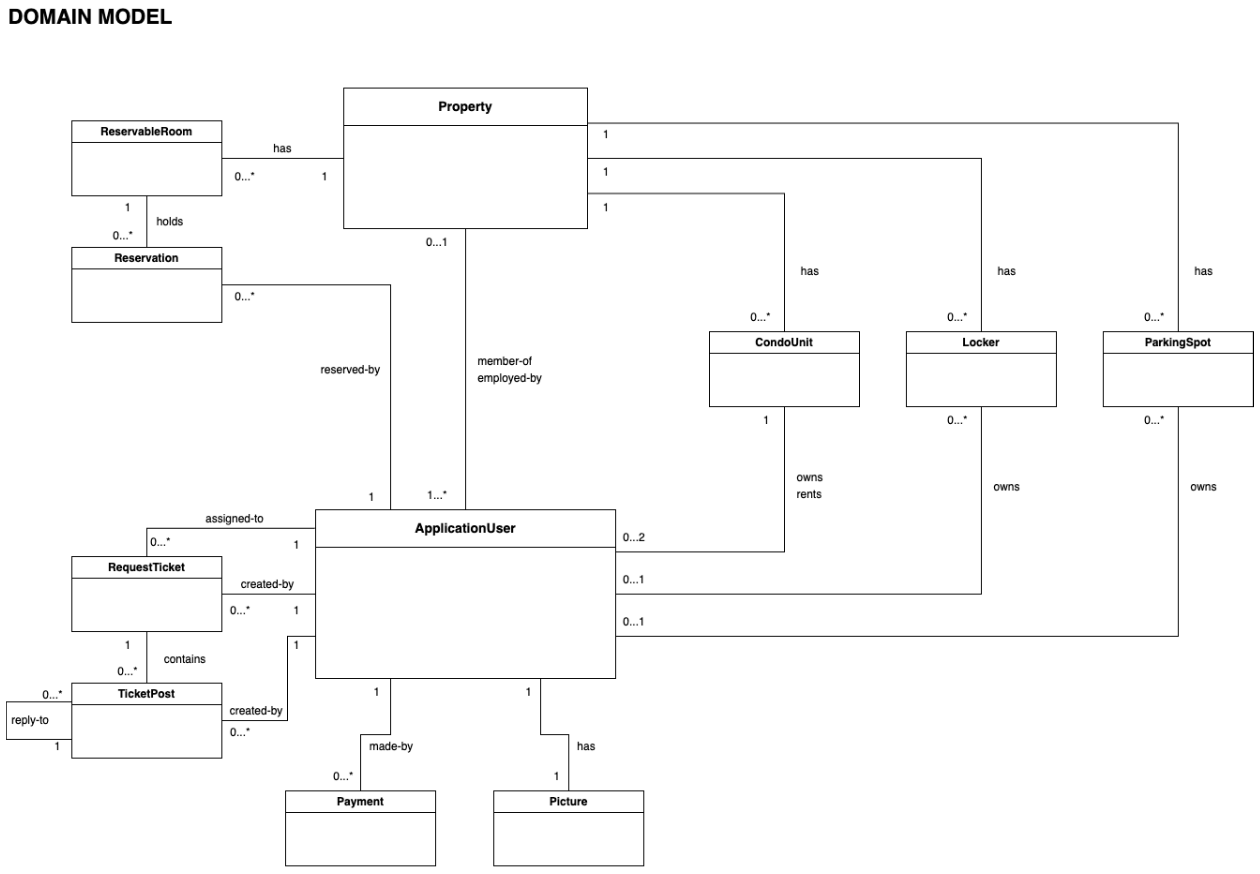


3.2 - Logical View

The Logical View focuses on the system's functionality and the conceptual organization of the system.

3.2.1 - Domain Model

The domain model provides a high-level overview of the main entities and their relationships with the CMS.

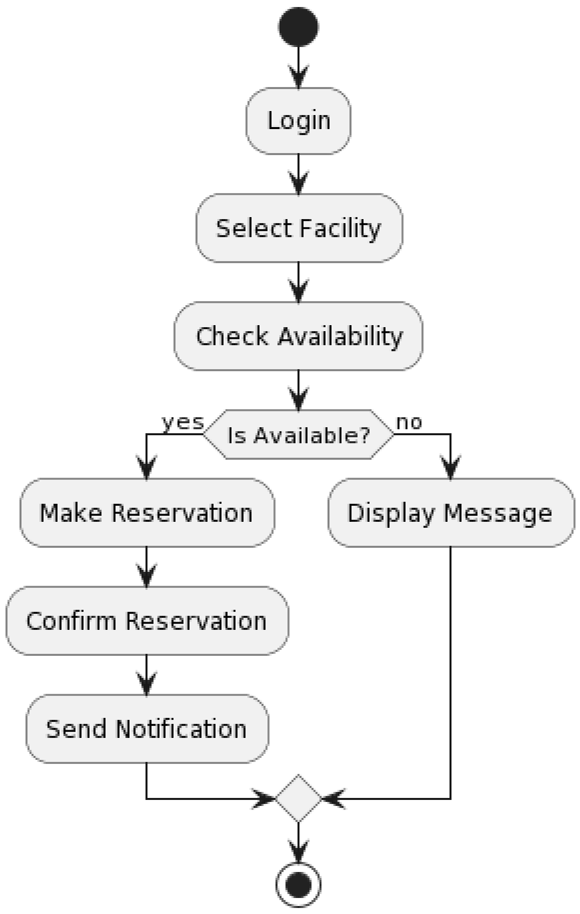


3.3 - Process View

The Process View focuses on the dynamic aspects of the system, illustrating how the system responds to events.

3.3.1 - Activity Diagram for Reservation Process

This diagram shows the steps involved when a user makes a reservation for a common facility.



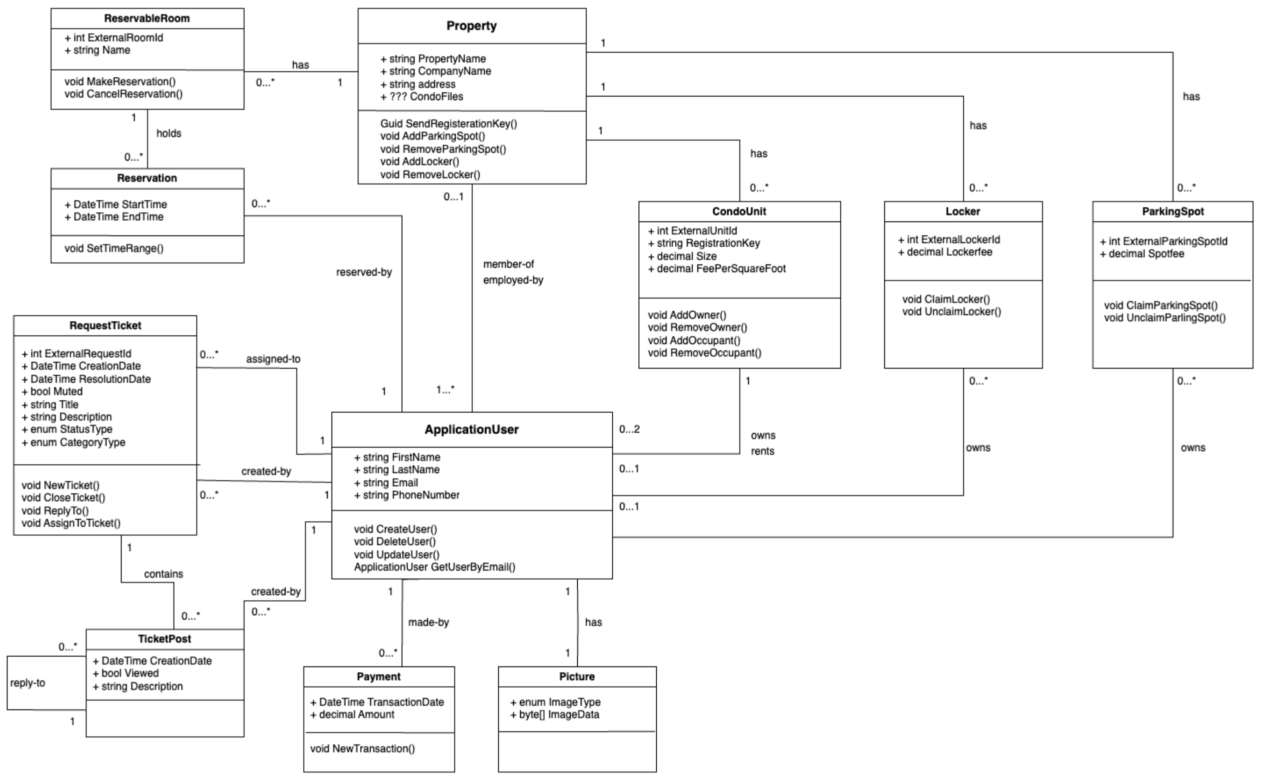
3.4 - Development View

The Development View is concerned with the actual realization of the system in source code.

3.4.1 - Class Diagram for User Management

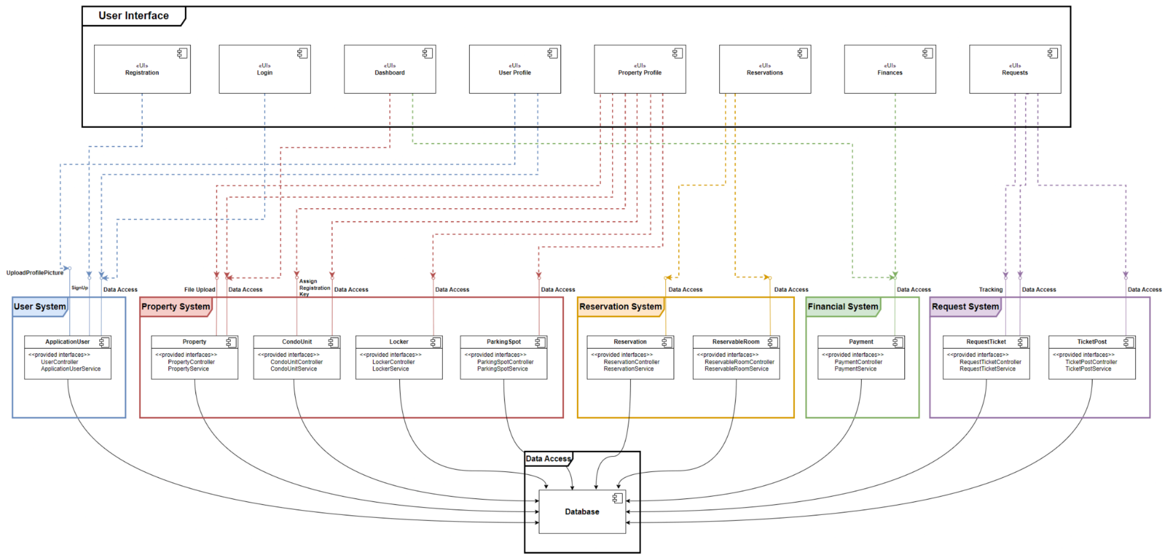
This diagram represents the classes involved in managing user information and their relationships.

CLASS DIAGRAM



3.4.2 - Component Diagram

This diagram illustrates the main software components of the CMS and their interactions.

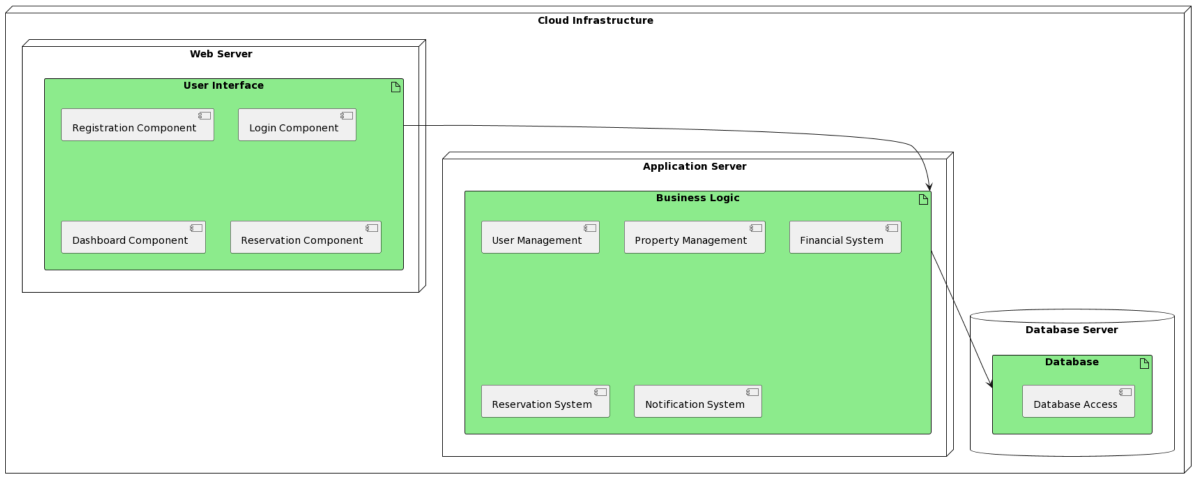

User Interface
«UI» Registration
«UI» Login
«UI» Dashboard
«UI» User Profile
«UI» Property Profile
«UI» Reservations
«UI» Finances
«UI» Requests
UploadProfilePicture
SignUp
Data Access
File Upload
Data Access
Assign Registration Key
Data Access
Data Access
Data Access
Data Access
Data Access
Data Access
Tracking
Data Access
Data Access
User System
ApplicationUser
<<provided interfaces>> UserController ApplicationUserService
Property System
Property
<<provided interfaces>> PropertyController PropertyService
CondoUnit
<<provided interfaces>> CondoUnitController CondoUnitService
Locker
<<provided interfaces>> LockerController LockerService
ParkingSpot
<<provided interfaces>> ParkingSpotController ParkingSpotService
Reservation System
Reservation
<<provided interfaces>> ReservationController ReservationService
ReservableRoom
<<provided interfaces>> ReservableRoomController ReservableRoomService
Financial System
Payment
<<provided interfaces>> PaymentController PaymentService
Request System
RequestTicket
<<provided interfaces>> RequestTicketController RequestTicketService
TicketPost
<<provided interfaces>> TicketPostController TicketPostService
Data Access
Database

3.5 - Deployment View

The Deployment View shows the physical distribution of software components across the hardware infrastructure.

3.5.1 - Deployment Diagram

This diagram illustrates the deployment of the CMS components on the cloud infrastructure.



3.6 - Known Issues with Views

At this stage, there are no known discrepancies between the views and their viewpoint conventions. However, as the project evolves, we will document any inconsistencies, incomplete items, open or unresolved issues, and deviations from the conventions established by the viewpoints. These will be addressed through decision-making processes and documented as outcomes and rationales.

4 - Architectural Decisions

4.1 - Front-end Technologies

The front-end of our Condo Management System is engineered to provide a seamless, user-friendly experience across various platforms and devices. To achieve this, we have opted for technologies that are both versatile and powerful, ensuring our application is accessible, responsive, and easy to use. The choice of React and NodeJS specifically caters to these requirements, facilitating the development of a progressive web app that meets the modern standards of web applications.

4.1.1 - React

React was chosen for its exceptional capability to build dynamic and complex user interfaces with efficiency and ease [3]. This JavaScript library offers a component-based architecture, enabling our team to develop reusable UI components that can manage their state, leading to more manageable code and faster development cycles. React's virtual DOM feature optimizes rendering, making the user experience smooth and responsive. Furthermore, its vast ecosystem, including tools, libraries, and community support, provides valuable resources for solving development challenges. React's popularity and wide adoption also mean that it stays at the forefront of web development technologies, ensuring our application remains current and robust.

4.1.2 - NodeJS

The decision to utilize NodeJS as part of our front-end technology stack is grounded in its efficiency and scalability, which are crucial for handling the server-side aspects of our progressive web app [5]. NodeJS's event-driven, non-blocking I/O model makes it particularly suited for data-intensive, real-time applications that run across distributed devices. It enables our application to handle numerous simultaneous connections with high throughput, which is essential for a condo management system expected to serve a large number of users. Additionally, NodeJS's compatibility with JavaScript allows for a unified development experience across both client and server sides, streamlining the development process and reducing the learning curve for developers familiar with JavaScript. The extensive npm package ecosystem further enhances NodeJS's functionality, offering a wealth of modules and tools that can be easily integrated to extend the application's capabilities and accelerate development.

4.2 - Back-end Technologies

The majority of the technologies that the Condo Management System’s back-end will be built upon are part of Microsoft’s developer environment. This will help the team avoid any compatibility issues between components as we build the back-end web API, as all of these are designed to work together. The following sections describe the reasoning behind our choices in more detail.

4.2.1 - C#

The decision to use C# as the main language for this project was made largely to be able to use the other parts of the tech stack listed in the next sections as they are designed for this language in particular. Indeed, the choice of these frameworks is what will determine how well the concerns of the stakeholders are ultimately addressed, not the coding language itself. Nonetheless, choosing C# still provides certain benefits over other languages for our team for two main reasons. First, it is quite similar to Java, a language that the majority of the team has extensively worked with throughout our engineering programs. Second, certain members of the team have experience with C# itself. Both of these factors indicate that C# will be easy for the team to work with during this project.

4.2.2 - .NET

The .NET platform [1], built by Microsoft, is the ideal infrastructure for the Condo Management System. The large community support that .NET has will help the team more easily find solutions to any problems that may arise. Indeed, the ASP.NET Web API we are building has great documentation available, providing us with guidance as to how we should structure our files and follow code conventions.

4.2.3 - Entity Framework Core

A framework used to allow .NET to easily access a database and store entities, Entity Framework Core is the obvious choice for a system using Microsoft technologies. One of the best benefits it provides is for developers, allowing us to almost completely avoid writing SQL queries [2].

4.2.4 - Microsoft SQL Server

In keeping with the theme of the back-end tech stack, Microsoft SQL Server allows us to easily implement our domain model while remaining as conflict-free as possible due to being part of the Microsoft environment.

References

[1] https://www.digiteum.com/net-advantages-software-development/
[2] https://learn.microsoft.com/en-us/ef/core/
[3] https://react.dev/
[4] https://en.wikipedia.org/wiki/4%2B1_architectural_view_model
[5] https://nodejs.org/en