Group 7

Virtual Chat 3D
Software Architecture Document

Version 2.0

Revision History

Date	Version	Description	Author
18/11/2024	1.0	Introduction and Architectural Goals and Constraints	Lý Nghị Hoằng
21/11/2024	1.1	Logical View	Lý Nghị Hoằng Châu Vĩnh Đạt Trần Đăng Tuấn
1/12/2024	1.2	Deployment view and Implement view	Lý Nghị Hoằng
28/12/2024	1.3	Database description	Trần Đăng Tuấn
29/12/2024	2.0	Rewrite Logical View: - High level - Server - Client - Database Rewrite Server Components	Châu Vĩnh Đạt Lý Nghị Hoằng

Table of Contents

Software Architecture Document

1. Introduction

- **Purpose**

This Software Architecture Document (SAD) outlines the high-level design of the Virtual 3D Chat application. It defines the architectural principles, components, and their interactions, as well as the technical decisions made to achieve the system's functional and non-functional requirements.

- **Scope**

This document covers the following aspects of the application architecture:

- System Overview: A high-level description of the system and its components.
- Architectural Patterns: The patterns used to design the system.
- Technology Stack: The programming languages, frameworks, and tools used to implement the system.
- Security Architecture: The security measures implemented to protect the system and user data.
- Deployment Architecture: The deployment strategy and infrastructure requirements.

2. Architectural Goals and Constraints

Architectural Goals:

1. **Performance:**
   - Ensure low-latency communication between clients and the server for smooth and responsive interactions.
2. **Usability:**
   - **Intuitive Interface:** Create a user-friendly interface that is easy to use without learning.
   - **Immersive Experience:** Provide a visually appealing and engaging 3D environment.
3. **Portability:**
   - **Cross-Platform Compatibility:** Ensure the application can run on various platforms: Android, iOS.
4. **Security:**
   - **Secure Authentication:** Implement robust authentication mechanisms to prevent unauthorized access.
   - **Encryption:** Encrypt sensitive data both in transit.

Architectural Constraints:

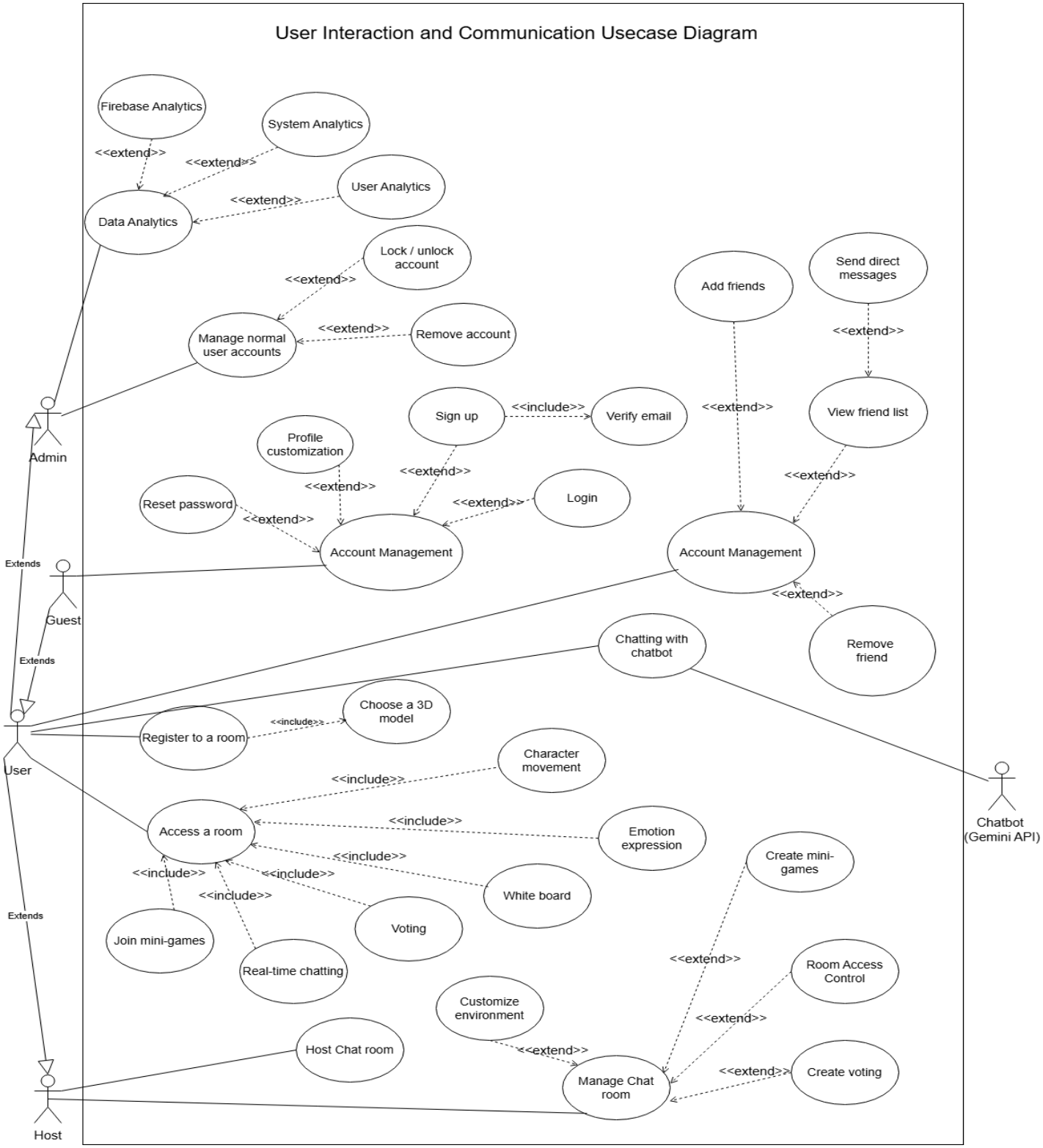
1. **Real-time Communication:**
   - **SocketIO:** Use SocketIO for real-time communication between clients and the server.
   - **LAN Hosting Online Room**: Use Unity Netcode for hosting online rooms in LAN.
2. **3D Rendering Engine:**
   - **Unity:** Leverage Unity's powerful 3D rendering capabilities to create immersive environments with cross-Platform compatibility.
3. **User Experience:**
   - **User Interface:** Design an intuitive and user-friendly UI.
   - **Input Methods:** Support various input methods, including keyboard, mouse, and touch.
   - **Social Features:** Implement social features like friend lists, groups, and private messaging.
4. **Security**
   - **Encryption:** Build custom packet encryption in transit based on RSA and AES encryption algorithm.
   - **Authentication**: Use the firebase authentication to verify users and manage user accounts.
5. **Development Tools and Frameworks:**
   - **Client-Side Development**

- Unity:
  - Core Engine: Used for 3D game development, rendering, physics, and animation.
  - Visual Scripting: For creating game logic without writing code.
  - Asset Store: A marketplace for purchasing or downloading free 3D models, scripts, and other assets.
- JetBrains Rider:
  - C# IDE: For writing and debugging C# scripts.
  - Code Completion and Analysis: Provides intelligent code suggestions and error detection.
  - Integration with Unity: Seamlessly integrates with Unity for efficient development.

- **Server-Side Development**
  - Node.js:
    - JavaScript Runtime: For building scalable and efficient server-side applications.
    - Asynchronous Programming: Enables handling multiple requests concurrently.
    - Large Ecosystem of Modules: Access to a wide range of libraries and frameworks.
  - Visual Studio Code:
    - Lightweight Code Editor: For writing and debugging JavaScript and TypeScript code.
    - Extensions: Supports various extensions for syntax highlighting, code completion, debugging, and version control.
    - Integration with Node.js: Provides tools for debugging, testing, and deploying Node.js applications.
- **Database**
  - Firebase:
    - Real-time Database: For storing and synchronizing data in real-time.
    - Authentication: For user authentication and authorization.
    - Cloud Storage: For storing user-generated content like avatars.
- **Version Control**
  - Git:
    - Distributed Version Control System: For tracking changes to source code and collaborating with other developers.
    - Popular Hosting Platforms: GitHub for hosting and managing code repositories.
- **Additional Tools**
  - Blender: For modify 3D models and animations if necessary.

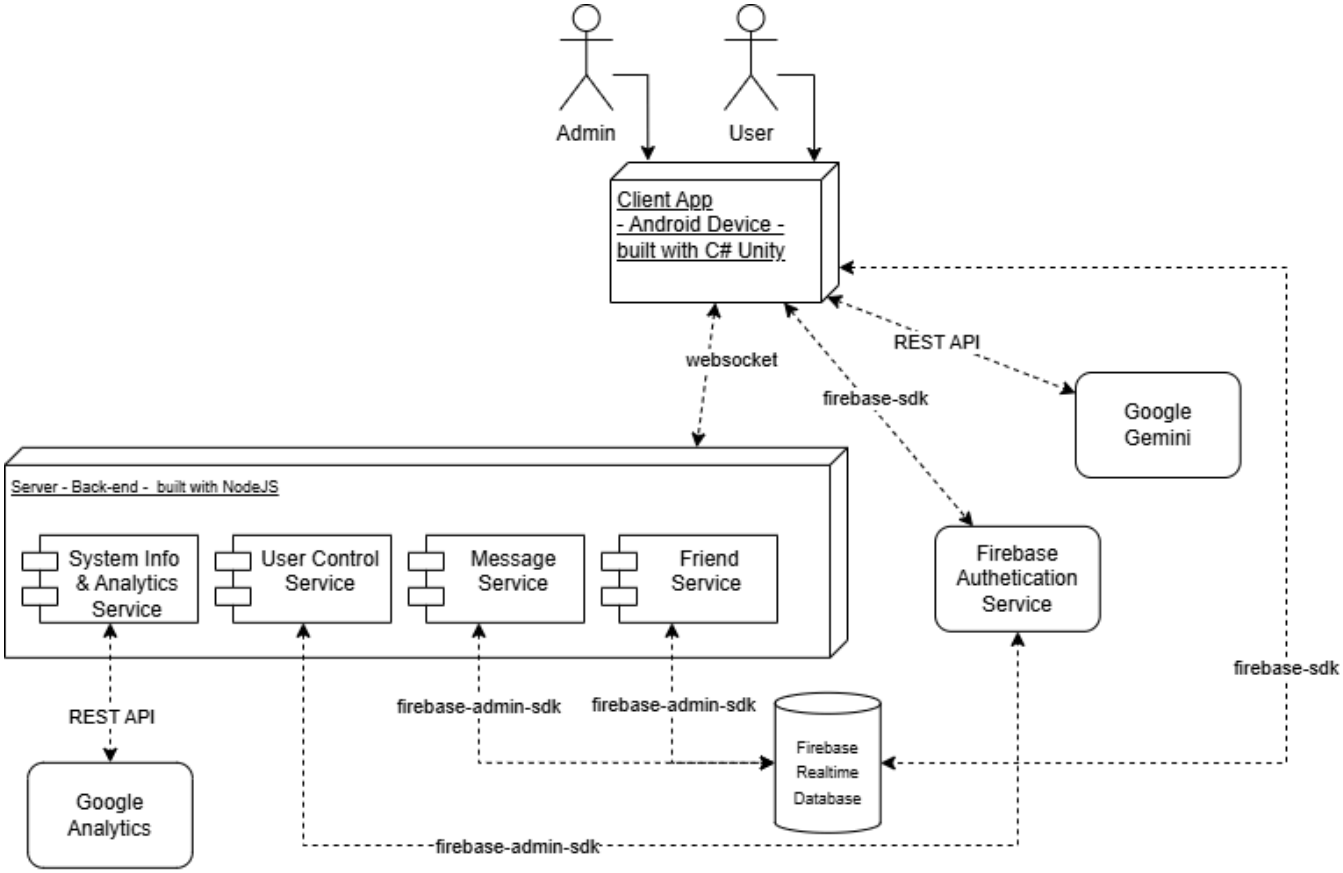
By carefully considering these architectural goals and constraints, you can build a robust, scalable, and secure virtual 3D chat application.

3. Use-Case Model

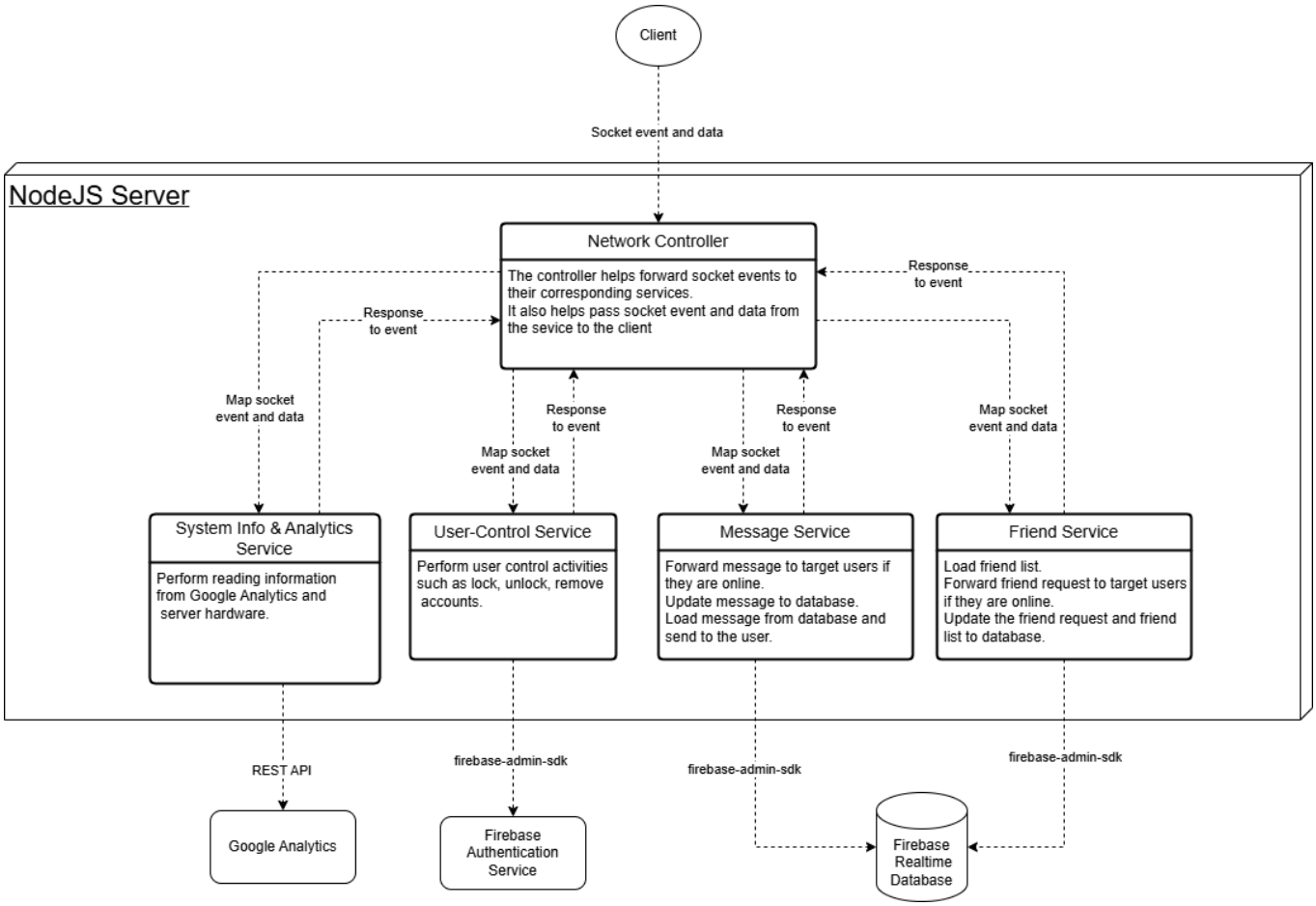


4. Logical View

Software High Level Architecture



Server Side

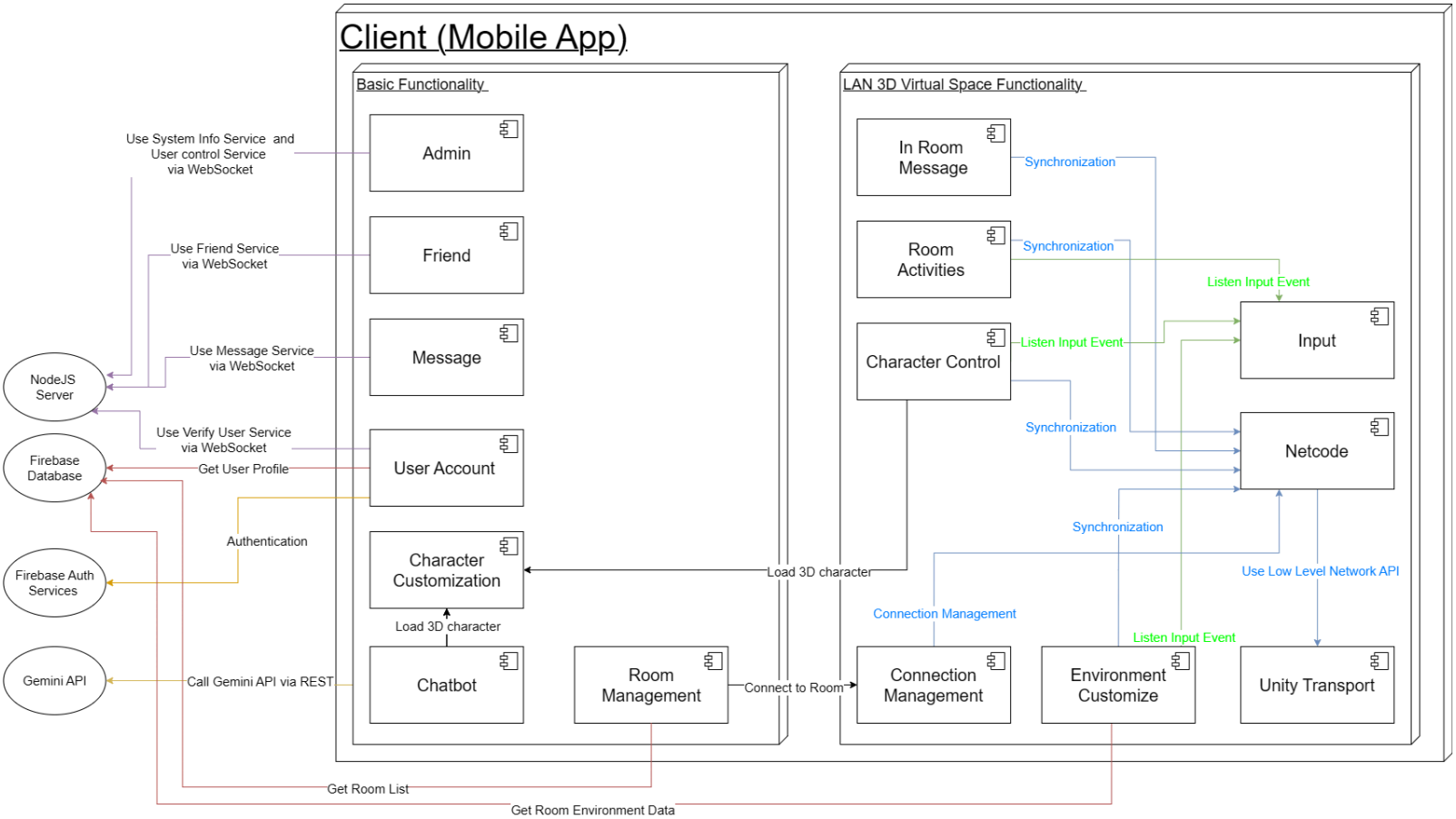


- Since our project mainly focuses on the functionalities built in the Client for Online Room, the server only acts as an intermediary between clients for direct messaging, storing information into the database, and serving administrative purposes (such as managing users and reading server reports).
- The Server utilizes the Network Controller to receive and transfer socket data packages to the corresponding services to handle and then return results.
- The services in the system can be described as follows:

Server Component	**Description**
NetworkController	Manage and distribute socket events to different services, provide functions for other services to respond to events properly.
Friend Service	Provide friend management functions: adding, removing friends. And write into the server.
Message Service	Provide real-time messaging functions between clients without having to connect to any online chat room.
User-Control Service	Provide administrative user management: view list of users, lock/unlock

	users with firebase-admin-sdk.
System Info & Analytics Service	Provide information about the computer running the server (System Info) and analytics created from information retrieved from firebase and google analytic APIs (Analytics).

Client Side



- System was designed in **Modular Monoliths** style, and organized the codebase into well-defined modules within a single application. These modules implement functionality of the system. Modular monoliths strike a balance between flexibility, scalability, and maintainability.
- The Components (Modules) are designed in the **MVC design pattern**:
  - **Model**: a class to store all data for the Component.
  - **View**: a class to interact with the Object provided by Unity and handle displaying.
  - **Controller**: a class to handle and control all the logic of the Component.
- Each component includes both a view and logic controller.
- The system is divided into 2 parts (subsystem):

a. The Basic Functionality subsystem:

- This subsystem provides the following features:
  + User authentication and account management.
  + Social functionality:
    + Friend management.

        + Message with friend.
    + 3D interactive Chatbot.
    + Admin functionality for admin account to monitor and manage system:
        + View system info and analytic.
        + User management (unlock/lock, delete user).
- These functionalities are mainly provided by service on server (by sending requests to the server).
- Here is a brief description of the components in the subsystem:

Component	**Description**
UserAccount	Provide user authentication and profile management for the user. Send user login token to server to verify user for Friend, Message, Admin component, then receive a response from the server to check whether it is a valid user and check whether it is an admin account. Use Firebase Auth Service for user authentication. Use Firebase Realtime Database service for profile storing.
Friend	Provide friend management functions. Friend requests and friend management are handled by using Friend Service on the server via WebSocket.
Message	Provide real-time messaging functions. Send/Receive messages are handled by using Message Service on the server via WebSocket.
Admin	Provide administrator functionalities: - User management: view list of users, lock/unlock users - System Info and Analytics Functionalities are handled by using Admin User-control, System Info, Analytics Service on the server via WebSocket.
Chatbot	Provide virtual 3D character chatting functionalities: - Chat with LLM and express expressions on the 3D character based on chat content. - Customize 3D characters by importing your owner model. Interactive chat and emotion analytic are provided by Gemini APIs, and customization of 3D character model using interface provided by CharacterCustomization component.
CharacterCustomization	Handle 3D models importing, manage imported models, load model, provide an interface for the Chatbot Component and the Character Control Component (in the Online Runtime subsystem) for loading the model. Using VRoid SDK for VRM model file loading (mesh loading, armature rigging, texture and material loading).

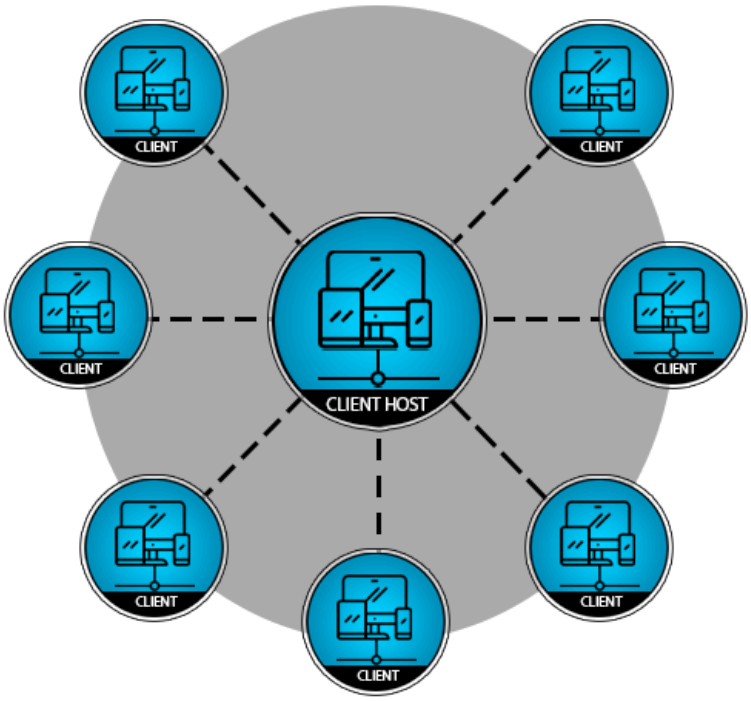
b. **Online Runtime subsystem**:

- This subsystem provides the core feature of the system: a 3D virtual space hosted on a LAN, where users can interact with others in this space. This space is named after the Online Room. This subsystem provides features:
    + Room management
    + Environment customization
    + Character in room control

    + Messaging/Chatting
    + Room Activities.
- These functionalities (including connection management, data synchronization,...) are mainly provided by Unity Netcode.
- This subsystem uses Client-hosted listen server Network topologies.
- Here is a brief description of the components of the subsystem:

Component	**Description**
RoomManagement	Provide functions for hosting or joining rooms, managing created rooms, room access management, and guests management (view joined users list and kick unwanted users). Use Firebase Realtime Database service for room data storing. Use NetworkManager (provided by Unity Netcode) to manage connection and user in room.
EnvironmentCustomize	Provide functions for managing objects in the room: - Provide a view to managing objects in the room (add, edit, delete,...) - Use Firebase Realtime Database service for object data storing. - Synchronize objects in the online room via NetworkObject (provided by Unity Netcode).
Message	Provide functions for chatting in a chat room. Message synchronize via RPC (provided by Unity Netcode).
Input	Provide the UI control component for users to control their 3D avatars in a chat room. (such as virtual joysticks, touch pad, etc.) Provide universal input event Interface. Map Unity Standard Input System (New Input System) for keyboard or physic joystick and custom UI control component to this interface. Other components will register input events of this interface to read input events.
CharacterControl	Main Control character on room: - Synchronize user 3D avatar via RPC (provided by Unity Netcode). - Control the avatar's movement, animation, and emotion. Synchronize via NetworkTransform, NetworkAnimator (provided by Unity Netcode).
Activities	Provide functions for other activities in the chat room such as voting, whiteboard, mini games.
Netcode (provided by Unity)	Unity Netcode component provides a framework for building multiplayer games with Unity. It includes tools, components, and patterns to manage synchronization, state replication, and communication between clients and servers. Use Unity Transport as Transport Layer
Unity Transport (provided by Unity)	Network Transport Layer provided by Unity. Provide Low-level network communication (sending/receiving data packets). Using UDP protocol.

Network topologies for online room

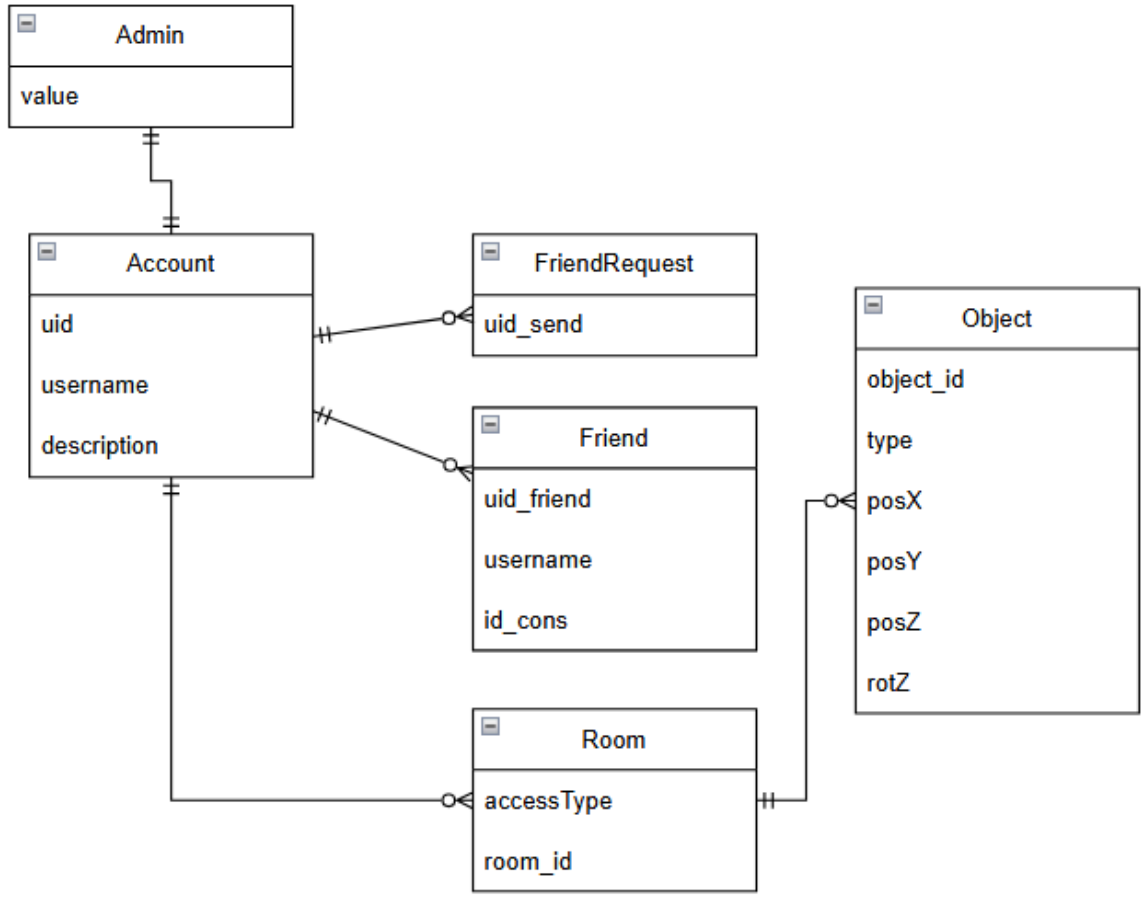


Reference:
https://docs-multiplayer.unity3d.com/netcode/current/terms-concepts/network-topologies/#client-hosted-listen-server
A client-hosted listen server operates by running the game server within the same process as a game client. This approach is cost-effective and ensures greater privacy since the host machine directly manages the server without relying on third-party infrastructure. However, this setup comes with some trade-offs:

- **Residential Internet Connection**: The hosting client must handle all game updates and data communication through a residential internet connection, which may lead to increased latency or limited bandwidth.
- **Performance Impact**: Since the server and the game client share the same machine, server performance is affected by the demands of generating an output image for the host player. This can result in a suboptimal experience for both the host and other players connected to the server.

Client-hosted listen servers are ideal for smaller-scale games or private sessions where cost and simplicity take precedence over scalability and performance.

Database design



The database is stored in Firebase Realtime Database, with the collections relationships illustrated by the diagram above. We define an Account entity to manage account information. Each account is linked to the following:

- FriendRequest: Stores incoming friend requests and the corresponding sender IDs.
- Friend: Tracks the user's friends and includes the ID of their associated conversations.
- Room: Identifies the rooms where the user is the host.

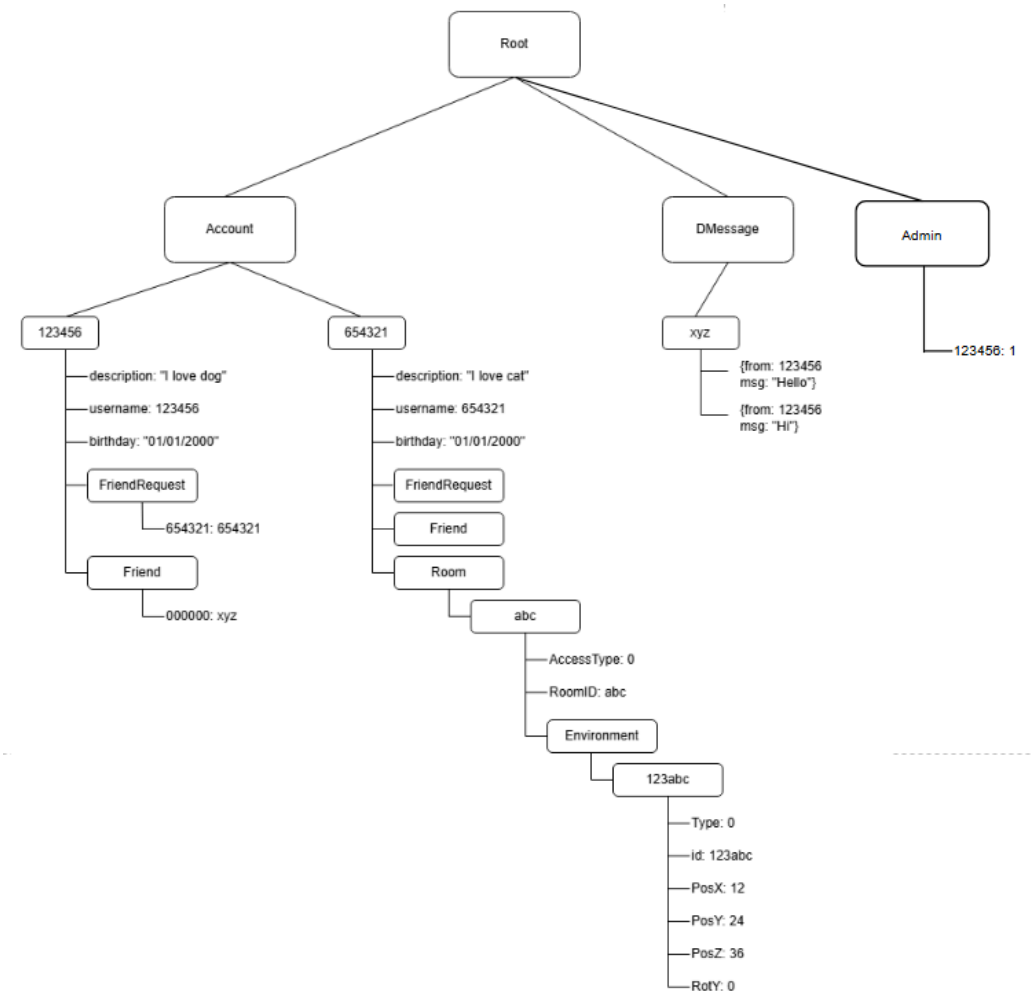
Within each Room, we need to store objects, represented as the Object entity, which helps locate and manage the environment’s elements.

In this design:

- Account Collection:
    - Each document within the Account collection represents a user and contains key user information such as:
        - description: User bio or status.
        - username: Unique identifier for the user.
        - birthday: User's date of birth.
    - Sub-collections within each account include:

        - FriendRequest: Tracks incoming friend requests.
        - Friend: Stores the user's friends and their associated conversation IDs.
        - Room: Represents rooms where the user is the host.
- Room and Environment:
    - Each room document has additional attributes:
        - AccessType: Defines permissions
        - RoomID: A unique identifier for the room.
    - The Environment sub-collection within each room stores objects as documents. Each object includes:
        - Type: The object's category.
        - Position attributes (PosX, PosY, PosZ) for object location.
        - RotY: The object's rotation along the Y-axis.
- DMessage Collection:
    - The DMessage collection contains message data. Each document (e.g., xyz) includes:
        - from: Sender's user ID.
        - msg: Message content.

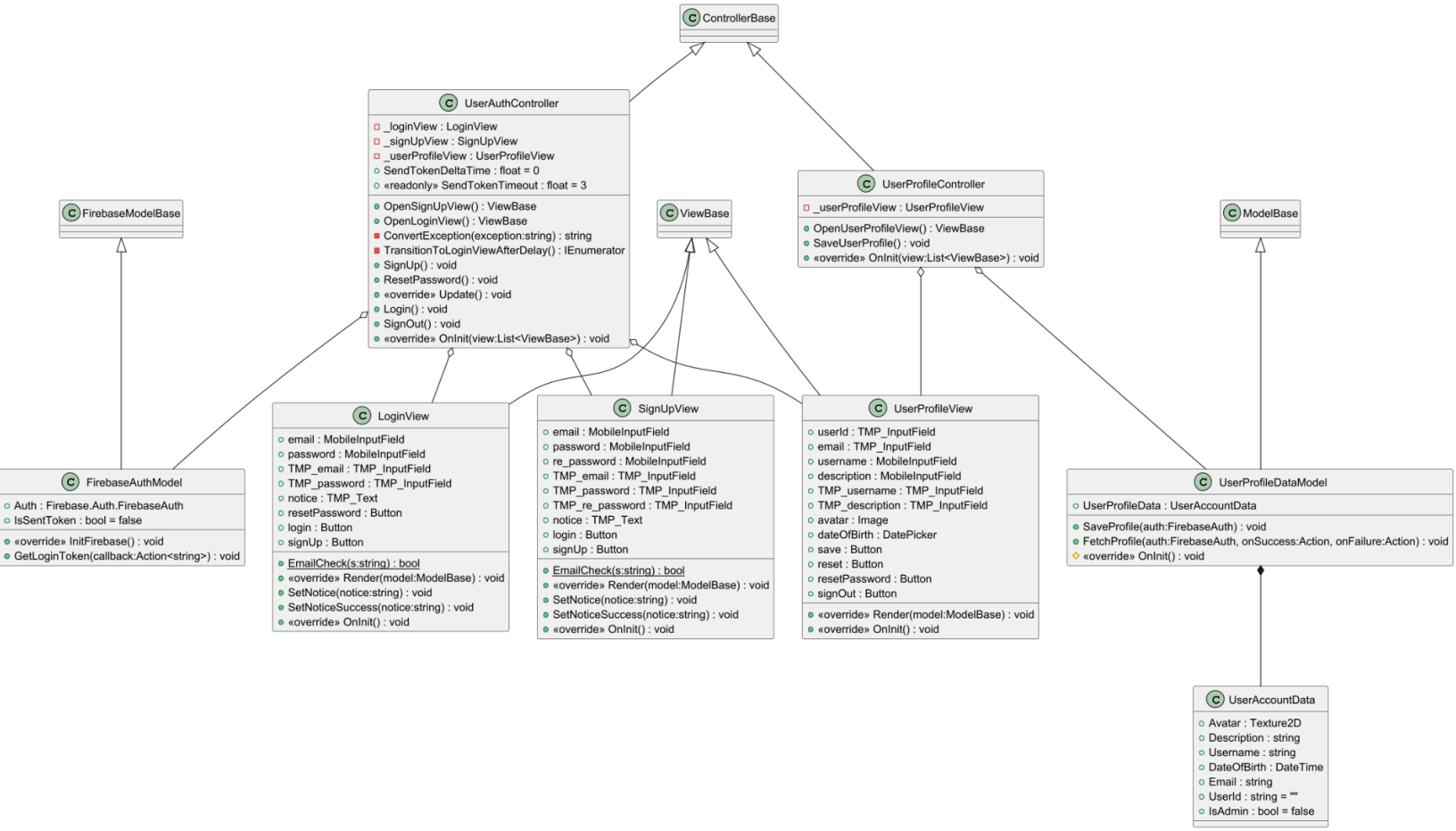
Here is a sample of how the data is stored in the tree-like style in Firebase:



4.1 Component: Client/UserAccount

This component provides Firebase authentication and user profile management functionality, including:

- Sign up, login, sign out, reset password using Firebase Auth service provided by Firebase SDK.
- View and edit profile: Avatar, Description, Username, Date of birth using Firebase Realtime Database service provided by Firebase SDK.
- Send user login token to server to verify user for Friend, Message, Admin component using WebSocket, then receive a response from the server to check whether it is a valid user and check whether it is an admin account.



Controllers:

- UserAuthController:
  - Controls the logic for login, sign up, logout, reset password functions by using Firebase SDK.
  - This class also gives access to the LoginView, SignUpView.
  - Send user login token to server to verify user using WebSocket then receive response from server then receive a response from the server to check whether it is a valid user and check whether it is an admin account.
- UserProfileController:
  - Reading/Writing user profile information from/to Firebase realtime database using Firebase SDK
  - Gives access to the UserProfileView.

Views:

- LoginView:
    - handles the Login View for users and reads user's inputs for credentials.
- SignUpView:
    - handles the Sign up View for users and reads user's inputs for sign up information.
- UserProfileView:
    - handles the View for user's profile management, provides editing and displaying user's profile information.

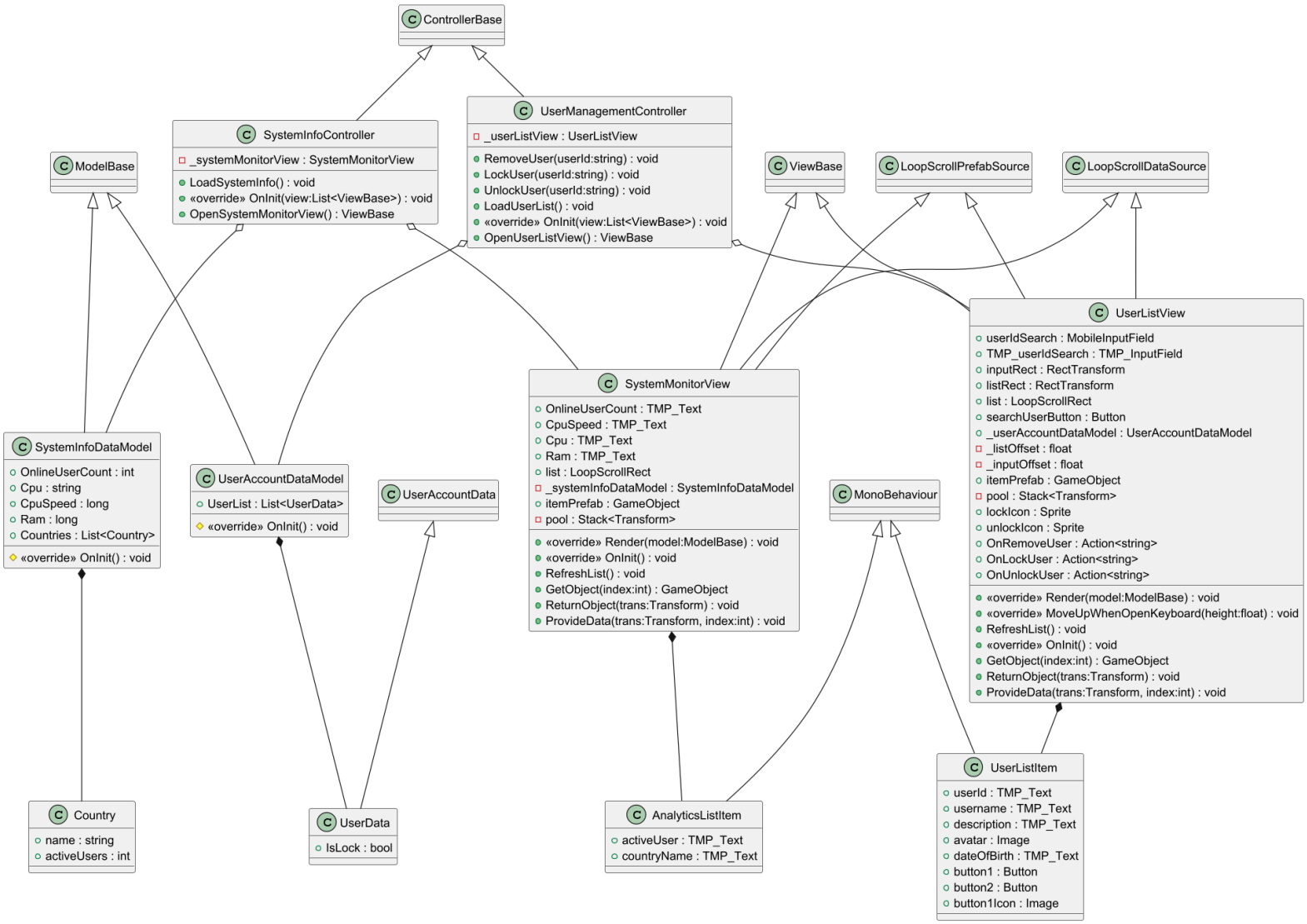
Models:

- FirebaseAuthModel:
    - Store the user's Firebase Auth instance, and relevant information.
- UserProfileDataModel:
    - Store the user's profile information after reading from the Firebase.

4.2 Component: Client/Admin

This component provides admin functionality. Including:

- User management functionalities are provided by UserControl service in server via sending requests by WebSocket:
    - view list of users
    - lock/unlock users using WebSocket.
    - delete users using WebSocket.
- System Info and Analytics functionalities are provided by System Info and Analytics service in server via sending requests by WebSocket.



Controllers:

- UserManagementController:
  - Controls the logic for managing users: removing user, locking/unlocking user, loading user list. These functions are executed by calling the server side APIs.
  - Get the user list by fetching from the server.
  - Access to the OpenUserListView.
- SystemInfoController:
  - Controls the logic for viewing System information and Analytic by fetching from the server.
  - Access to the OpenUserListView.

Views:

- UserListView:
  - Handles the interface for viewing users list functions.
  - Read the Admin's command (lock/unlock, delete) for execution.

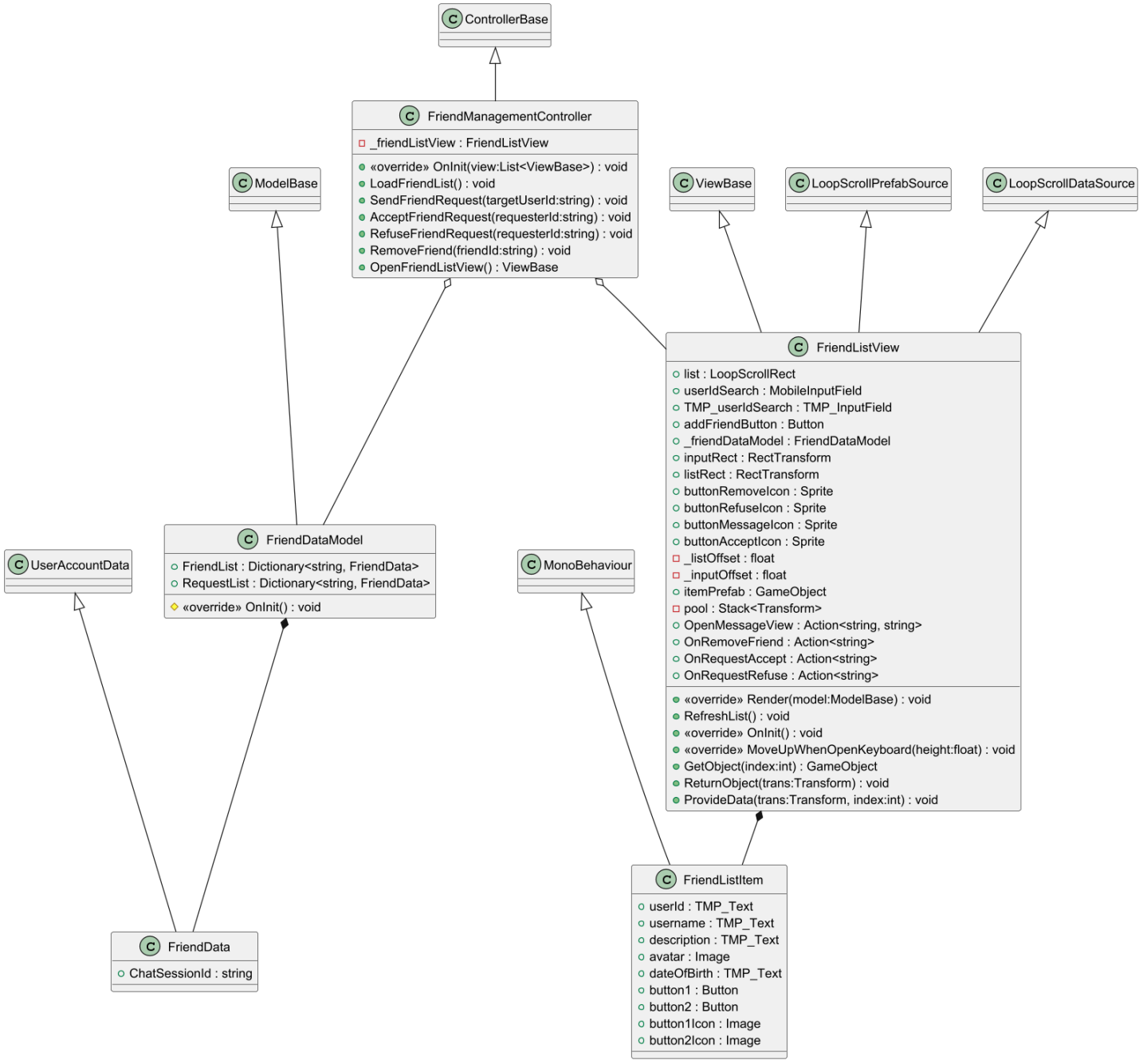
- SystemMonitorView:
    - View statistical information about the System and Server.

Models:

- UserAccountDataModel:
    - Stores users' data read from the server.
- SystemInfoDataModel:
    - Stores the data about the system read from the server.

4.3 Component: Client/Friend

This component provides friend management functionality.
Including: view friend list, send new friend request, accept/refuse friend request, remove friend.
These functionalities are provided by Friend service in server via sending requests by WebSocket



Controllers:

- FriendManagementController:
  - Controls the logic for managing a user's friends list, including adding, accepting, and removing friends. These functions are executed by calling the server side APIs.
  - Get the friend list by fetching from the server.
  - Provides access to the FriendListView for rendering friend-related data.

Views:

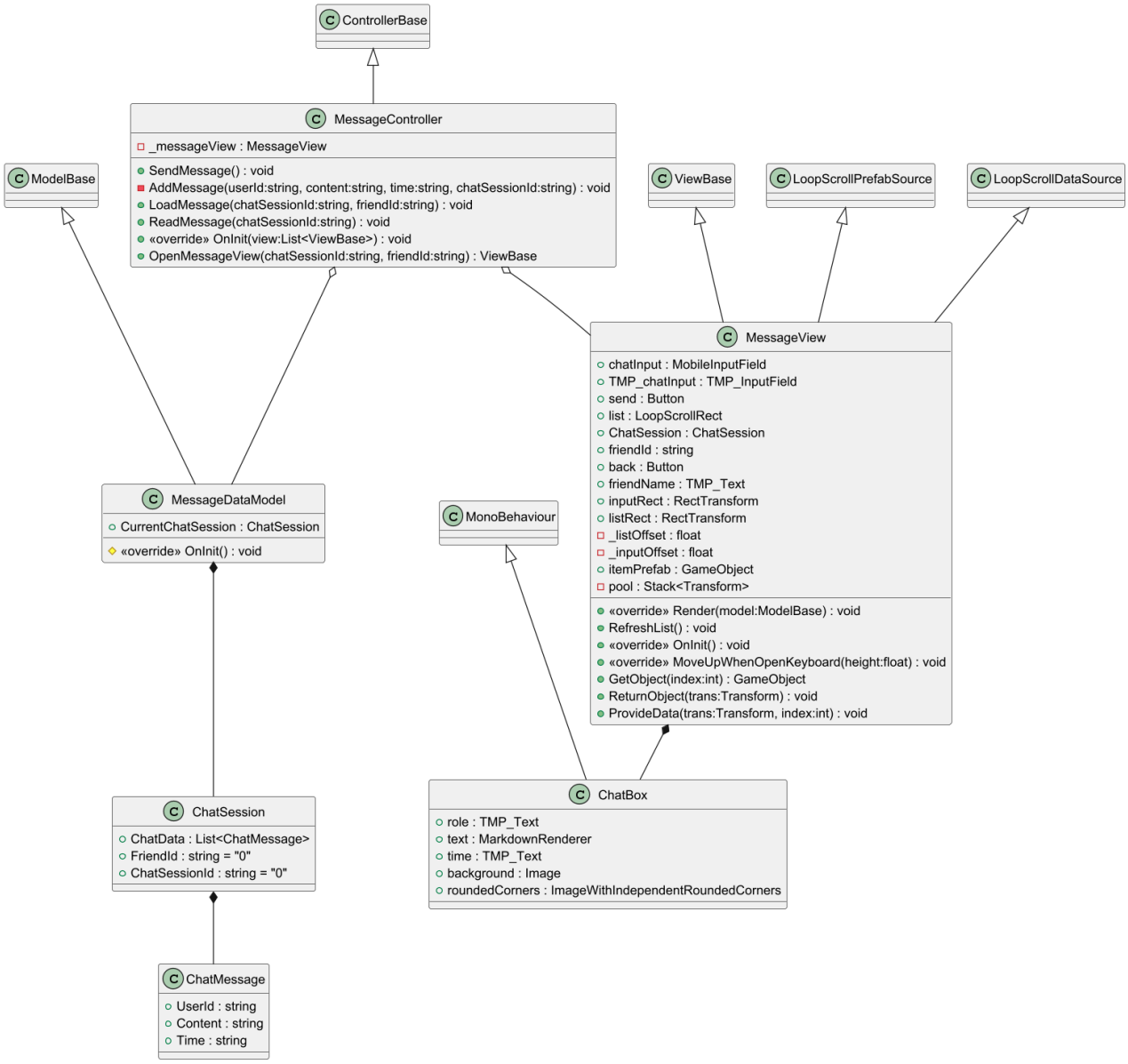
- FriendListView:
    - Handles the interface for managing a user's friends, displaying the friend list, and providing actions like adding or removing friends.
    - Reads user input for adding friends, accepting friend requests, removing friends and interacting with the friend list.

Models:

- FriendDataModel:
    - Stores the user's friend data, including details like friend IDs, statuses, and other related information read from the server..

4.4 Component: Client/Message

This component provides message functionality.
Including sending and receiving messages between users.
These functionalities are provided by Message service in server via sending requests by WebSocket



Controllers:

- MessageController:
  - Manages the logic for handling messages between users, including reading, sending, and receiving new messages.
  - Handles message loading and checks if there are new messages from specific users.
  - These functions are executed by calling the server side APIs.
  - Provides access to the MessageView to display and interact with chat messages.

Views:

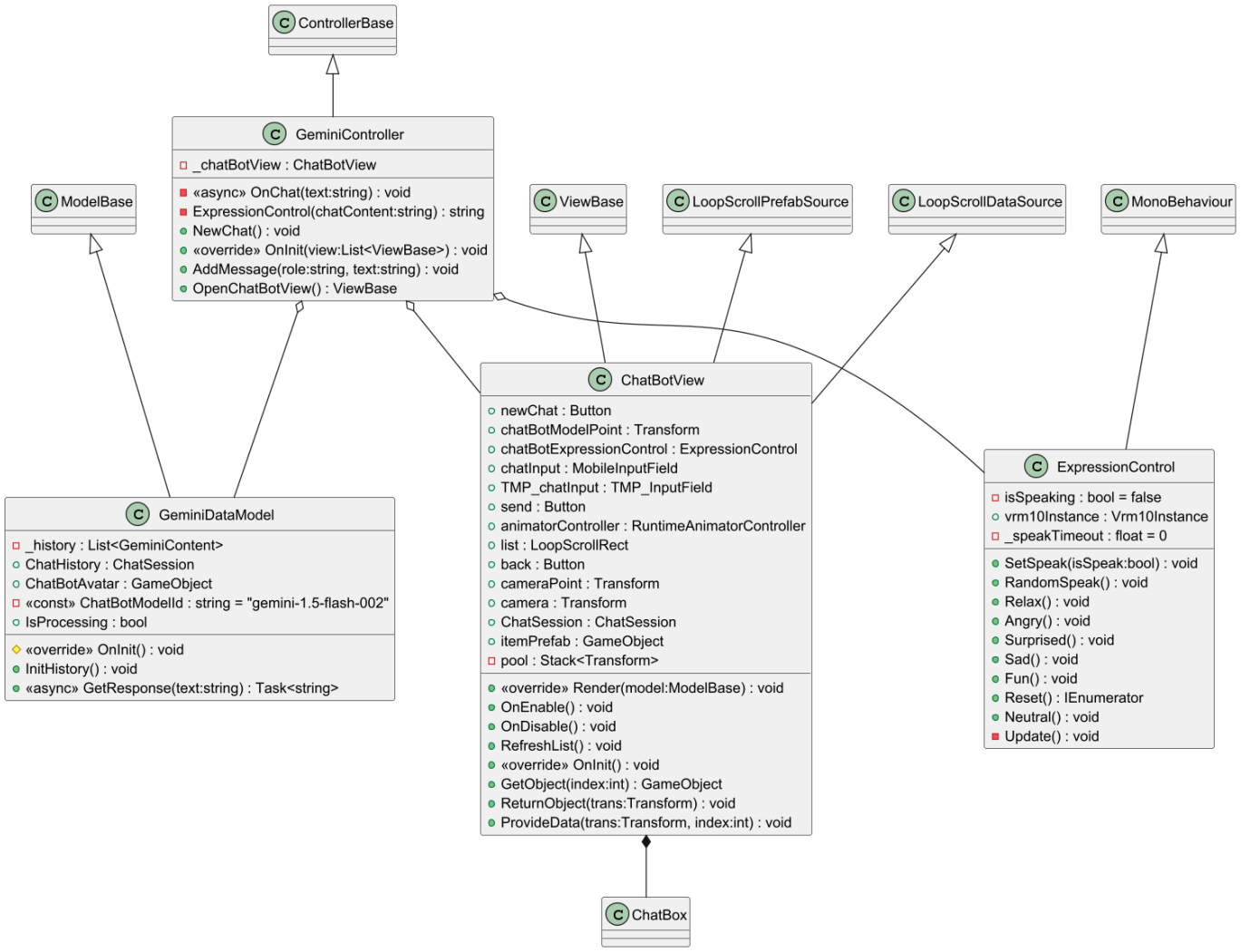
- MessageView:
  - Manages the user interface for chat interactions, including sending and viewing messages.
  - Displays a scrollable list of messages and allows user input through a text field and send button.
  - Dynamically renders chat items using a template for individual chat messages.

Models:

- MessageDataModel:
    - Stores the user's chat session fetching from the server.
    - Provides message data to the MessageController for operations and updates.

4.5 Component: Client/ChatBot

This component provides ChatBot functionality based on Gemini API.
Including interactive chat with Gemini, change Chatbot emotion according to chat content, customize chatbot 3D character.
LLM interactive chat and emotion analytics are provided by Gemini API via sending requests by http.
3D character emotion is provided by the Vroid SDK API.
3D character customization is provided by CharacterCustomize Component



Controllers:

- GeminiController:
    - Manages the interaction logic for the Gemini chatbot, including handling user inputs, controlling expressions, and starting new chat sessions.
    - Provides access to the ChatBotView for rendering the chatbot's responses and interactions.

    - Handles chat asynchronously and manages the emotional states of the chatbot through ExpressionControl.
    - Use interactive chat provided by Gemini Service by sending http requests.
- ExpressionControl:
    - Unity Component script to control the chatbot's visual expressions and speaking animations.
    - Manages various emotional states (e.g., angry, sad, happy) and updates the expression dynamically during interactions.

Views:

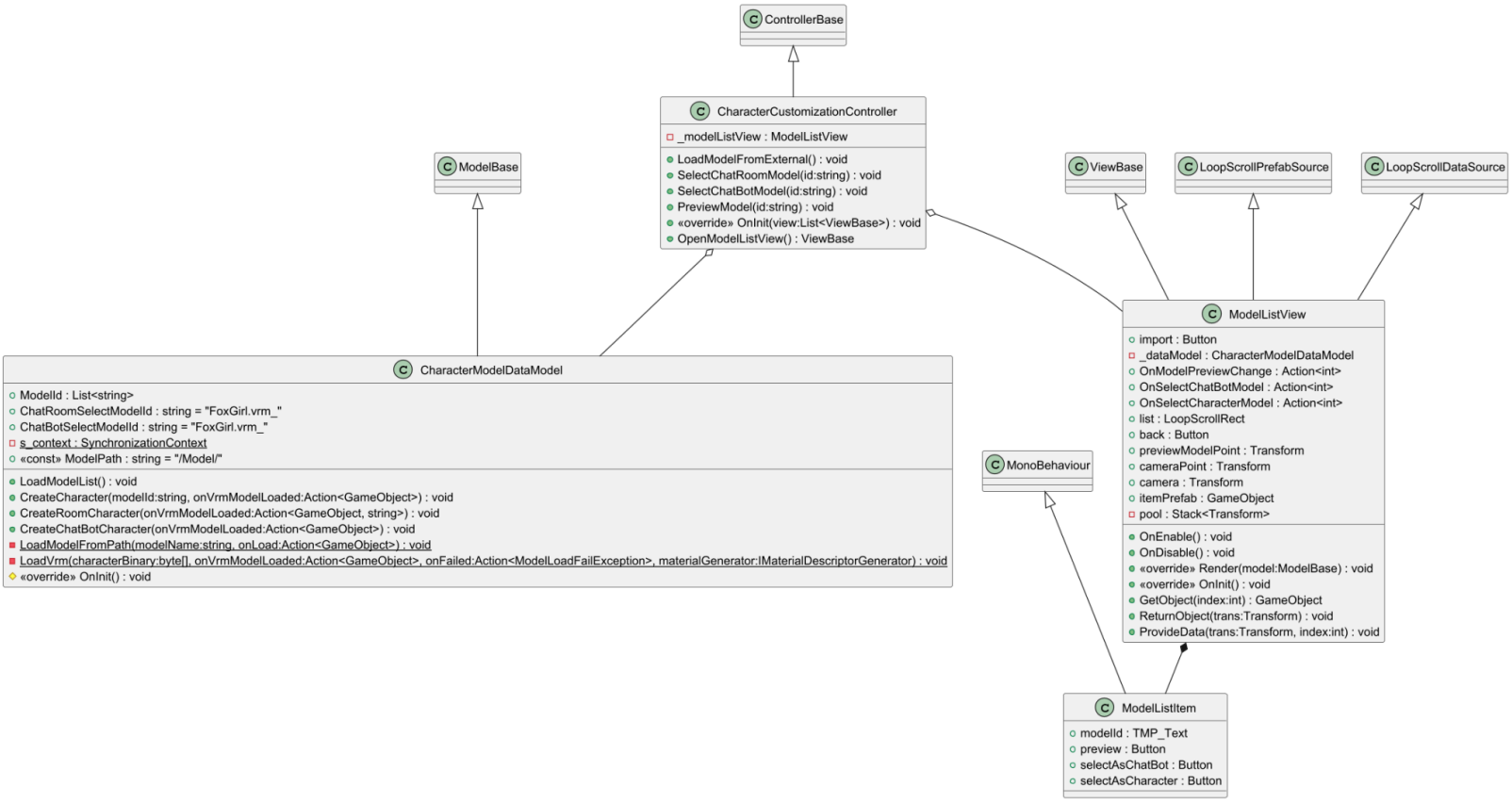
- ChatBotView:
    - Manages the user interface for the Gemini chatbot, including input/output for chat interactions.
    - Displays the chatbot's visual representation and updates its emotional expressions.
    - Dynamically renders user messages and chatbot responses in separate chat boxes.

Models:

- GeminiDataModel:
    - Stores the chatbot's data, including a history of chats and the name of the currently active gemini model id.
    - Supplies chatting related data to GeminiController for processing and UI updates.

4.6 Component: CharacterCustomize

This component provides character customization functionality.
Including: import/preview/select model for user’s character in online room and for chatbot avatar.
Model loading from VRM model file functionality is provided Vroid SDK.
Model loading included: mesh loading, armature rigging, texture and material loading and finally creating GameObject with rigged mesh.



Controller:

- CharacterCustomizationController:
  - Manages logic for character customization, including importing models from external memory of mobile devices and selecting specific models for chat rooms or chatbots.
  - Handles interaction between the user and the ModelListView, enabling model selection and updates.

View:

- ModelListView:
  - Displays a scrollable list of character models using a dynamic template.
  - Includes functionality for selecting models and previewing them in a designated area.
  - Allows user interaction via buttons and dynamically renders model-related data.

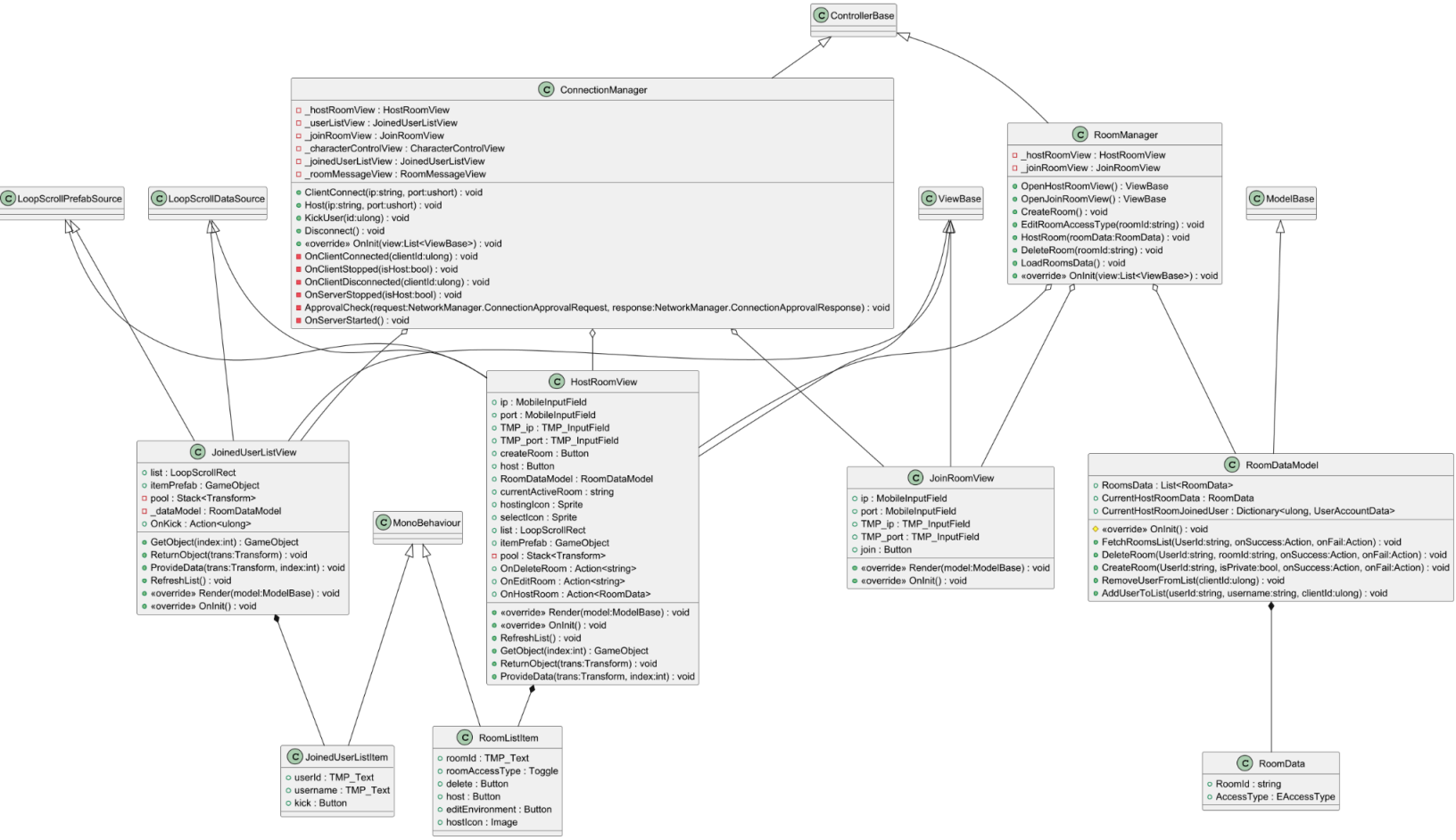
Model:

- CharacterModelDataModel:
  - Load list of available models from device app internal storage.
  - Asynchronously loading VRM models from files, creating 3D characters using Vroid SDK API.
  - Store selected model id for Online Room and Chatbot.

4.7 Component: OnlineRuntime/RoomManagement

This component provides connection and room management functionality. Including:

- Management connection: host room/join to room by using Unity Netcode API.
- View list of joined users for host, kick user for host by using Unity Netcode API.
- Manage room: create/delete room, set room access scope: anyone/only friend. These functionalities are provided by Firebase realtime database.



Controller:

- RoomManager:
  - Manages the logic for creating rooms, editing rooms' information and deleting rooms.
  - Handles interaction with the HostRoomView.
- ConnectionManager:
  - Handles the connection between host and client users.
  - Provide a function for the host to view the list of joined users and kick other joined users.

View:

- HostRoomView:
  - Handles the host user's input for managing chat rooms.
  - Displays created chat rooms list and rooms' information.
- JoinRoomView:
  - Provides an interface for the user to access chat rooms via IP and Port.
- JoinedUserView:
  - Provides an interface for the host to view the list of users who joined the room.

Model:

- RoomDataModel:
  - Contains created room data.

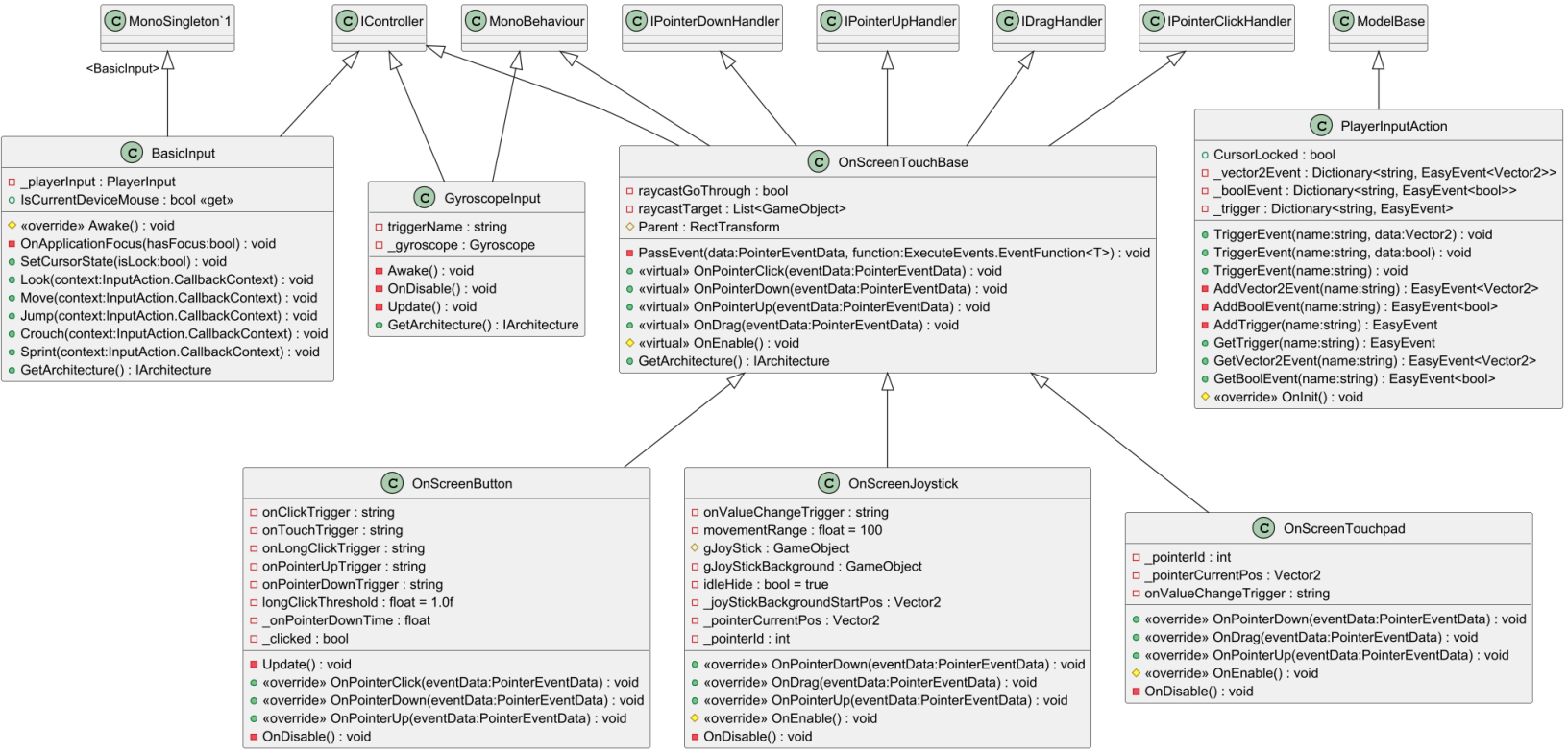
- Get room data by fetching from firebase realtime database.

4.8 Component: OnlineRuntime/Input

This component provides keyboard and mouse input to control the character.
Including:

- Keyboard and Mouse: move (wasd), look (mouse), jump (space), crouch (c), sprint(shift).
- Touch screen: move (on screen virtual joystick), look (on screen touchpad), jump (on screen button), crouch (on screen button), sprint(on screen button).
- sensor input: look (Gyroscope)

Provide custom UI control component for mobile device: on screen virtual joystick, touchpad, button and gyroscope.
Provide universal input event Interface: Map Unity Standard Input System (New Input System) for keyboard or physic joystick and custom UI control component to this interface.
Other components will register input events of this interface to read input events.

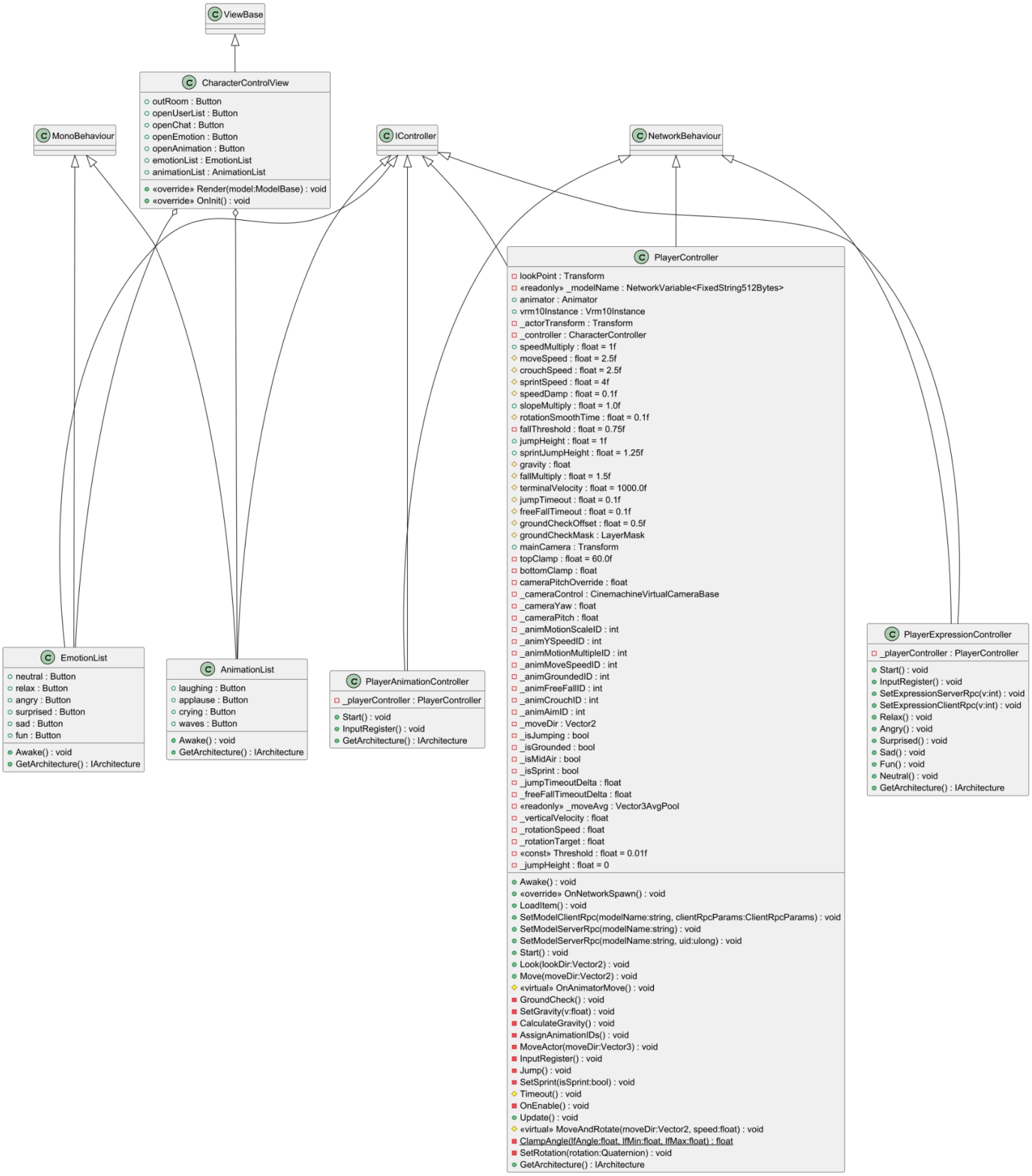


- PlayerInputAction:
  - Container to store input events.
  - Trigger specific event by name.
  - Get event by name (used to register event callback).
  - Provide universal input event interface.
- OnScreenButton:
  - Unity component script provides a button component with additional functionality.
- OnScreenJoystick:
  - Unity component script provides a virtual joystick component for the touchscreen.
- OnScreenTouchPad:
  - Unity component script provides a touchpad component for the touchscreen.
- GyroscopeInput:
  - Wrap Unity Gyroscope API to provide input from Gyroscope.

- BasicInput:
    - Wrap Unity input system for keyboard and mouse input to universal input event interface.

4.9 Component: OnlineRuntime/CharacterControl

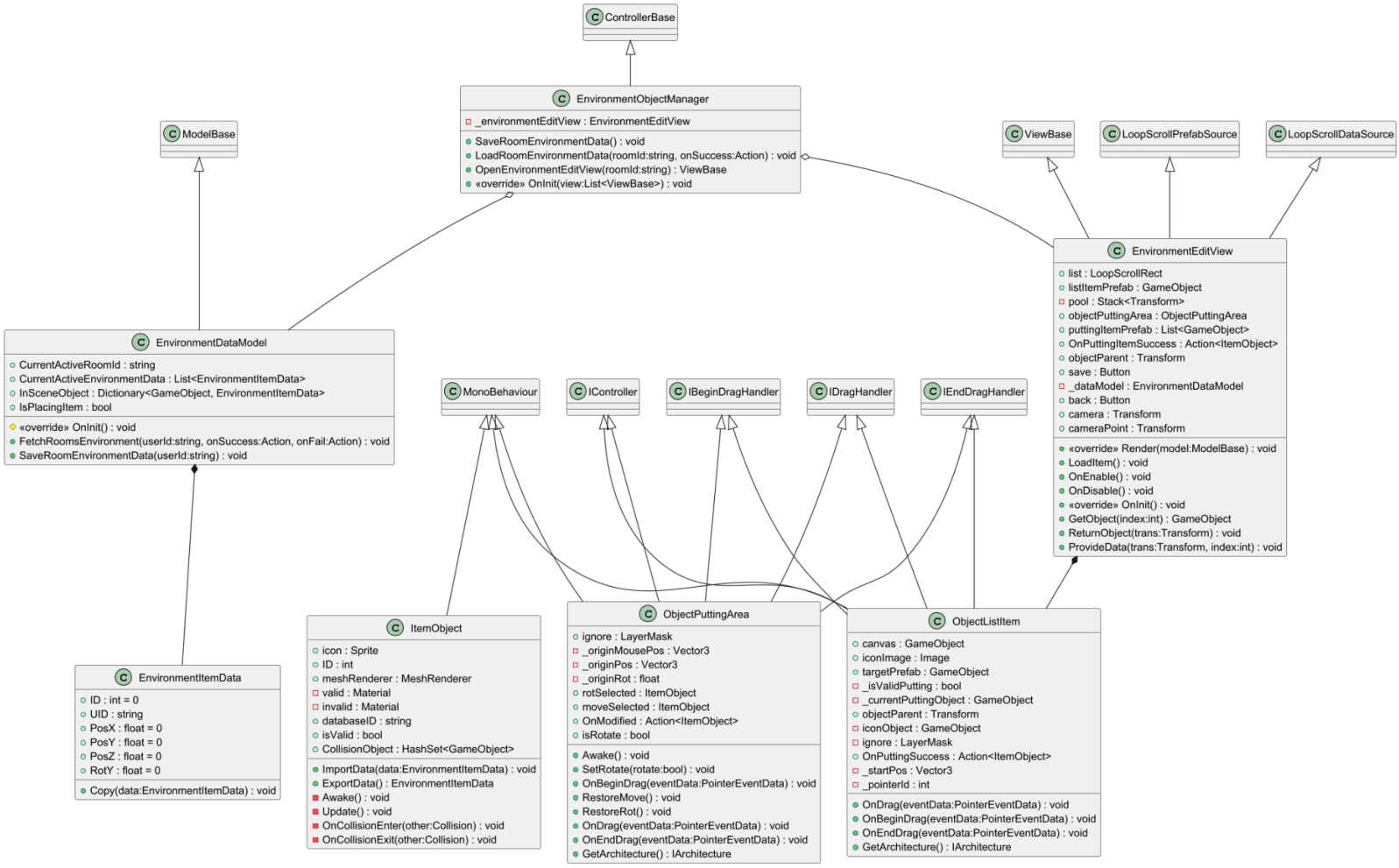
This component provides control to the character on the online room based on Unity netcode.
Including: Move, look around, jump, crouch, sprint, play animation, play expression.
Use the Network Transform and Network Animator component of Unity Netcode to synchronize characters position, rotation and animation in the room.
Use RPC to manually synchronize character emotion in the room.
Use Vroid SDK API to control character emotion.
Register universal input event interface provided by Input Component to get user input.


ViewBase
CharacterControlView
outRoom : Button
openUserList : Button
openChat : Button
openEmotion : Button
openAnimation : Button
emotionList : EmotionList
animationList : AnimationList
«override» Render(model:ModelBase) : void
«override» OnInit() : void
MonoBehaviour
IController
NetworkBehaviour
PlayerController
lookPoint : Transform
«readonly» _modelName : NetworkVariable<FixedString512Bytes>
animator : Animator
vrm10Instance : Vrm10Instance
_actorTransform : Transform
_controller : CharacterController
speedMultiply : float = 1f
moveSpeed : float = 2.5f
crouchSpeed : float = 2.5f
sprintSpeed : float = 4f
speedDamp : float = 0.1f
slopeMultiply : float = 1.0f
rotationSmoothTime : float = 0.1f
fallThreshold : float = 0.75f
jumpHeight : float = 1f
sprintJumpHeight : float = 1.25f
gravity : float
fallMultiply : float = 1.5f
terminalVelocity : float = 1000.0f
jumpTimeout : float = 0.1f
freeFallTimeout : float = 0.1f
groundCheckOffset : float = 0.5f
groundCheckMask : LayerMask
mainCamera : Transform
topClamp : float = 60.0f
bottomClamp : float
cameraPitchOverride : float
_cameraControl : CinemachineVirtualCameraBase
_cameraYaw : float
_cameraPitch : float
_animMotionScaleID : int
_animYSpeedID : int
_animMotionMultipleID : int
_animMoveSpeedID : int
_animGroundedID : int
_animFreeFallID : int
_animCrouchID : int
_animAimID : int
_moveDir : Vector2
_isJumping : bool
_isGrounded : bool
_isMidAir : bool
_isSprint : bool
_jumpTimeoutDelta : float
_freeFallTimeoutDelta : float
«readonly» _moveAvg : Vector3AvgPool
_verticalVelocity : float
_rotationSpeed : float
_rotationTarget : float
«const» Threshold : float = 0.01f
_jumpHeight : float = 0
Awake() : void
«override» OnNetworkSpawn() : void
LoadItem() : void
SetModelClientRpc(modelName:string, clientRpcParams:ClientRpcParams) : void
SetModelServerRpc(modelName:string) : void
SetModelServerRpc(modelName:string, uid:ulong) : void
Start() : void
Look(lookDir:Vector2) : void
Move(moveDir:Vector2) : void
«virtual» OnAnimatorMove() : void
GroundCheck() : void
SetGravity(v:float) : void
CalculateGravity() : void
AssignAnimationIDs() : void
MoveActor(moveDir:Vector3) : void
InputRegister() : void
Jump() : void
SetSprint(isSprint:bool) : void
Timeout() : void
OnEnable() : void
Update() : void
«virtual» MoveAndRotate(moveDir:Vector2, speed:float) : void
ClampAngle(lfAngle:float, lfMin:float, lfMax:float) : float
SetRotation(rotation:Quaternion) : void
GetArchitecture() : IArchitecture
EmotionList
neutral : Button
relax : Button
angry : Button
surprised : Button
sad : Button
fun : Button
Awake() : void
GetArchitecture() : IArchitecture
AnimationList
laughing : Button
applause : Button
crying : Button
waves : Button
Awake() : void
GetArchitecture() : IArchitecture
PlayerAnimationController
_playerController : PlayerController
Start() : void
InputRegister() : void
GetArchitecture() : IArchitecture
PlayerExpressionController
_playerController : PlayerController
Start() : void
InputRegister() : void
SetExpressionServerRpc(v:int) : void
SetExpressionClientRpc(v:int) : void
Relax() : void
Angry() : void
Surprised() : void
Sad() : void
Fun() : void
Neutral() : void
GetArchitecture() : IArchitecture

- PlayerController:
    - Unity Network object component script provides controlling character action in the room.
    - Handle input events to control character.
    - Use the Network Transform and Network Animator component of Unity Netcode to synchronize characters position, rotation and moving animation in the room.
    - Character motion implemented by using root motion and Unity build-in CharacterController component.
    - Use RPC to synchronize character 3D models.
- PlayerExpressionController:
    - Unity Network object component script provides controlling character expression in the room.
    - Handle input events to control character.
    - Use RPC to synchronize.
    - Use Vroid API to control character emotion.
- PlayerAnimationController:
    - Play specify animation: Applause, Waves, Crying and Laughing.
    - Handle input events to control character.
    - Use the Network Animator component of Unity Netcode to synchronize characters moving animation in the room.
- CharacterControlView:
    - Map on screen UI input to universal input event interface.

4.10 Component: OnlineRuntime/EnvironmentCustomize

This component provides Environment customization based on Unity netcode.
Including: Add/remove pre-build objects, move objects, rotate objects. save/load room environment data to firebase.
When start host room, spawn custom object and NetworkObject of Unity Netcode to automatically synchronize custom objects in the room.

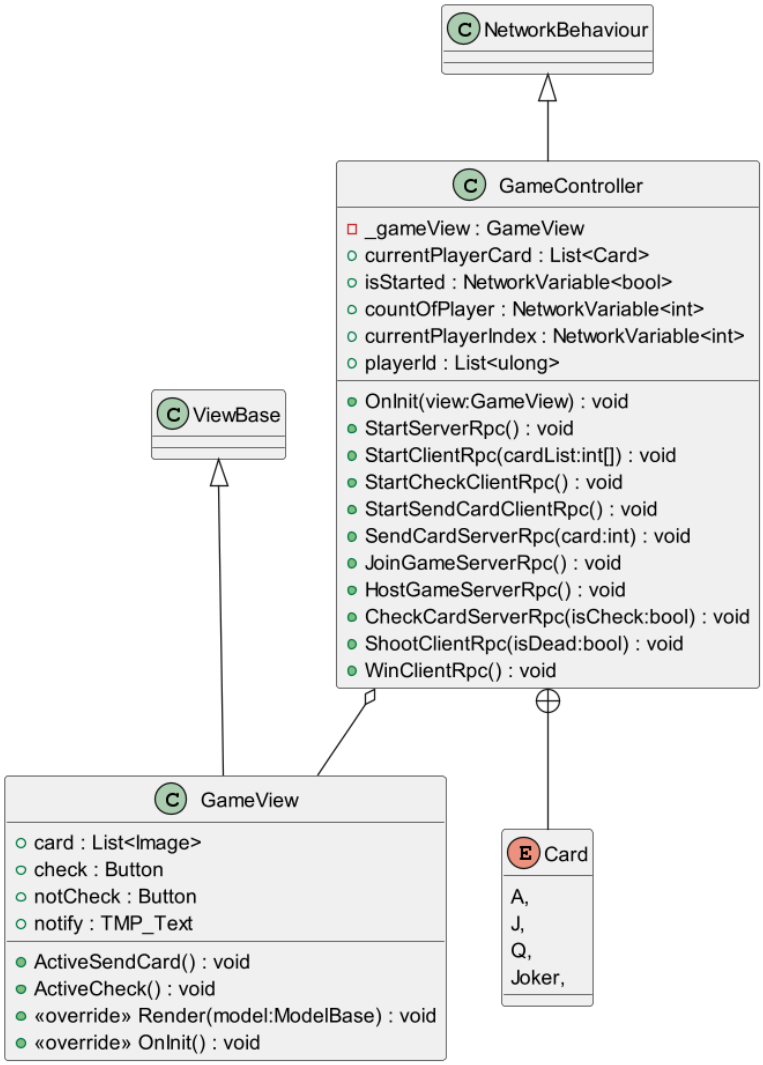


- EnvironmentObjectManager:
  - Provides functions for editing objects in chat rooms.
  - Provides methods to save/load data to/from firebase database.
- EnvironmentEditView:
  - Provides interfaces for the host to use functionalities in EnvironmentObjectManager.
- ObjectPuttingArea:
  - Unity component script provides interface to putting/editing objects on scene by drag and drop.
- RoomEnvironmentDataModel:
  - Store information about the objects in the chat room.
  - Save/load data to/from firebase realtime database.
- ItemObject:
  - Unity component script provides render object model of 3D object in the scene.
  - Check Collision in the scene.

4.11 Component: OnlineRuntime/Activities/MiniGame

This component provides Minigame functionality based on Unity netcode. This Component provides a card game like

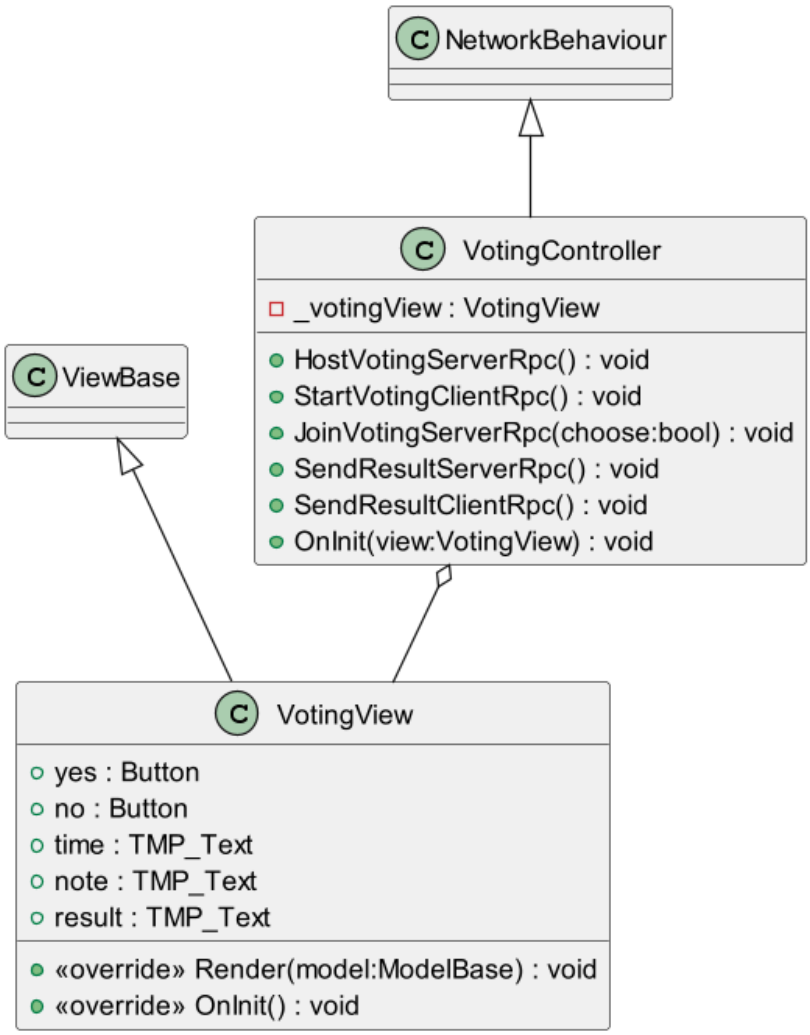
Liar's Bar (https://store.steampowered.com/app/3097560/Liars_Bar/). This Component will be further designed in later phases of the development process.



- GameController:
  - Unity network object component script management game rule, game logic, game state, synchronize using RPC.
- GameView:
  - View of Game.

4.12 Component: OnlineRuntime/Activities/Voting

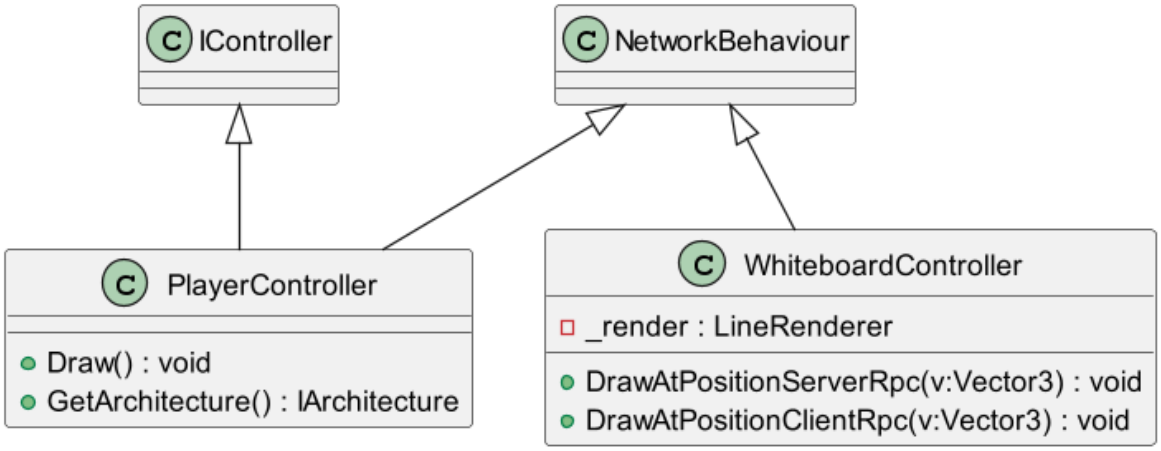
This component provides hosting voting based on Unity netcode, including: hosting a voting, voting settings (vote duration), voting, display result to all users joined in room.



- VotingController:
  - Unity network object component script management Voting process, synchronize using RPC.
- VotingView:
  - View of voting

4.13 Component: OnlineRuntime/Activities/Whiteboard

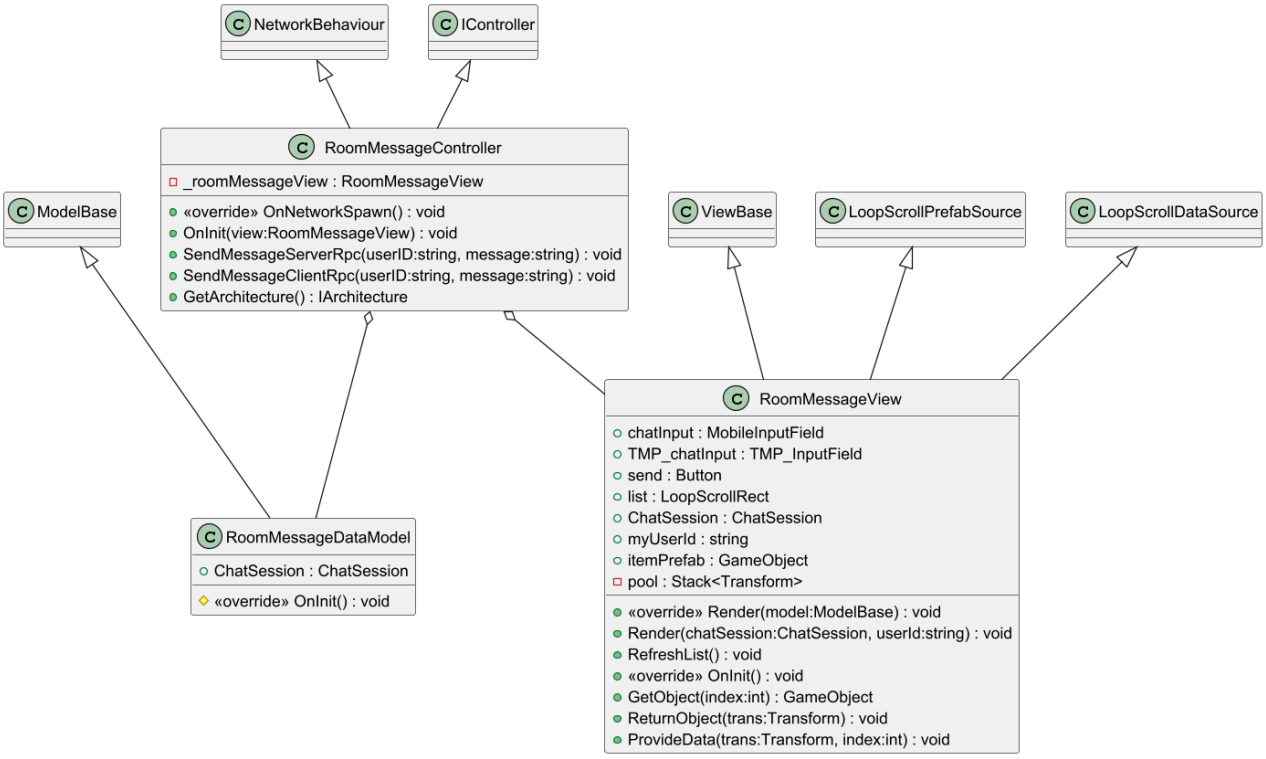
This component provides a whiteboard based on Unity netcode. Any users joined in the room can draw/erase on the whiteboard.



- PlayerController:
  - Unity network object component script to provide whiteboard drawing.
- Whiteboard controller:
  - Render lines on the whiteboard and synchronize to all clients using RPC.

4.14 Component: OnlineRuntime/Message

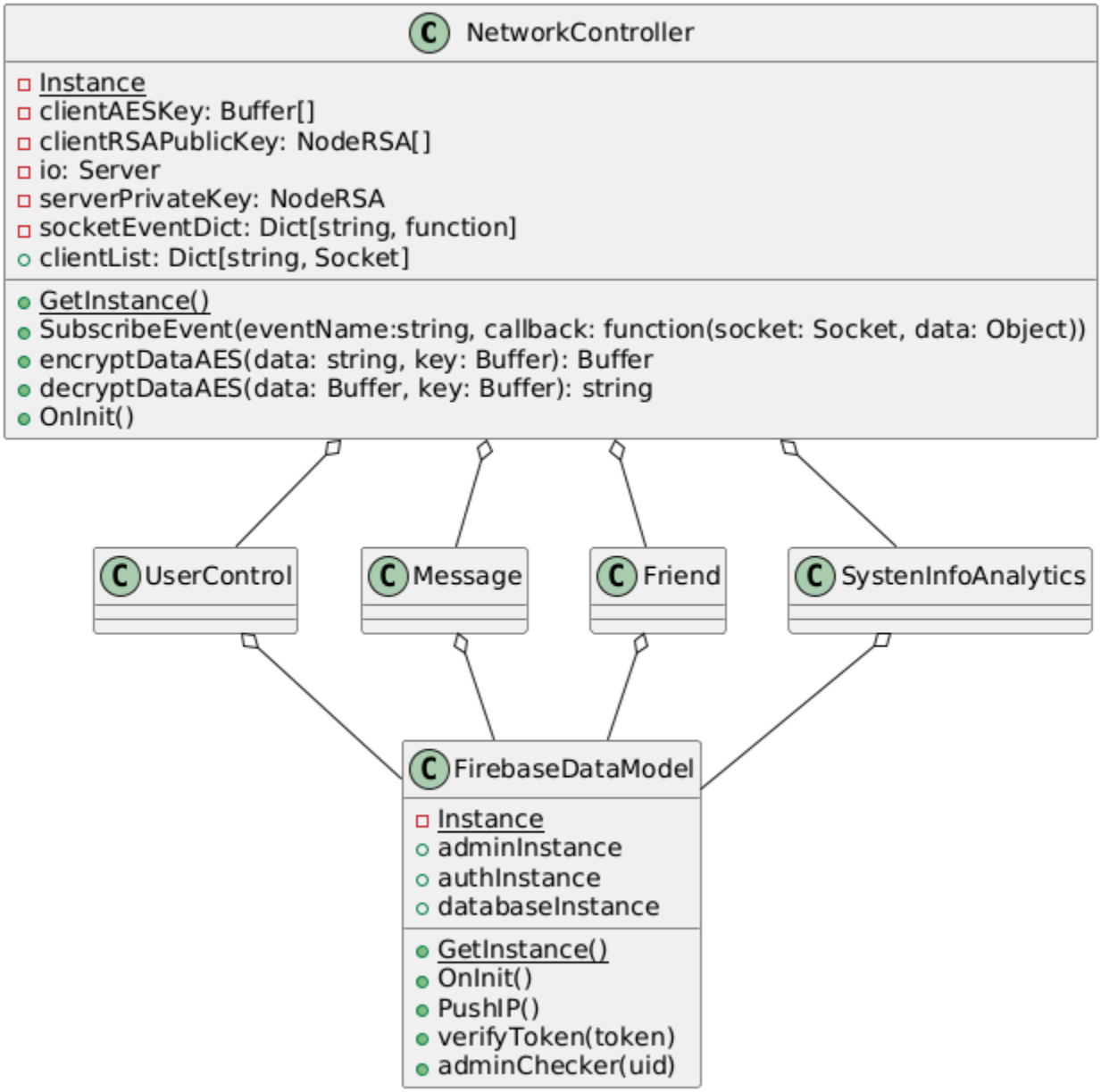
This component provides message communication based on Unity netcode
Including send/receive messages on chatbox.
Use RPC to synchronize messages.



- RoomMessageController:
  - Provides send/receive real time messages functionality in the room.
  - Use RPC to synchronize messages.

- RoomMessageView:
  - View to display messages on chat box.
- RoomMessageDataModel:
  - Container to store chat messages in the room.

4.15 Component: Server/NetworkController and FirebaseData/Model



NetworkController:

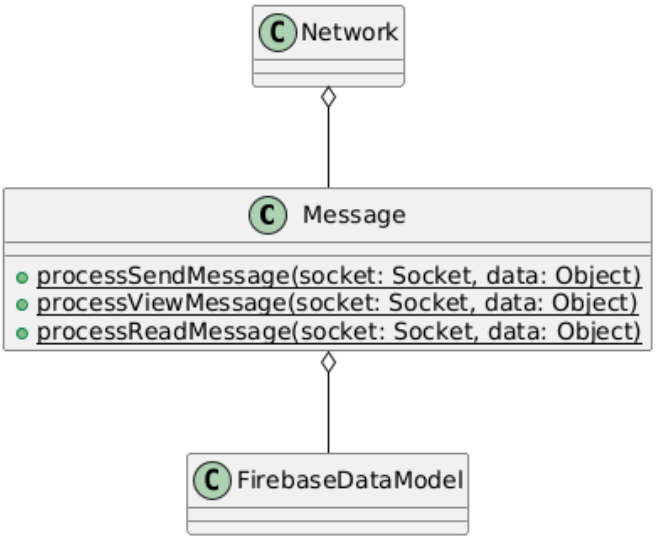
- The data sent between the two sides is encrypted with RSA and AES.
- The component is built in the Singleton design pattern to allow synchronization and prevent reinitialization of the socket server.
- Each service component uses the SubscribeEvent method to establish a socket event and the callback function

when the event is called from the client.

FirebaseDataModel:

- The class is a singleton of firebase services object provided by the firebase-admin-sdk.
- It helps other components communicate with the firebase.

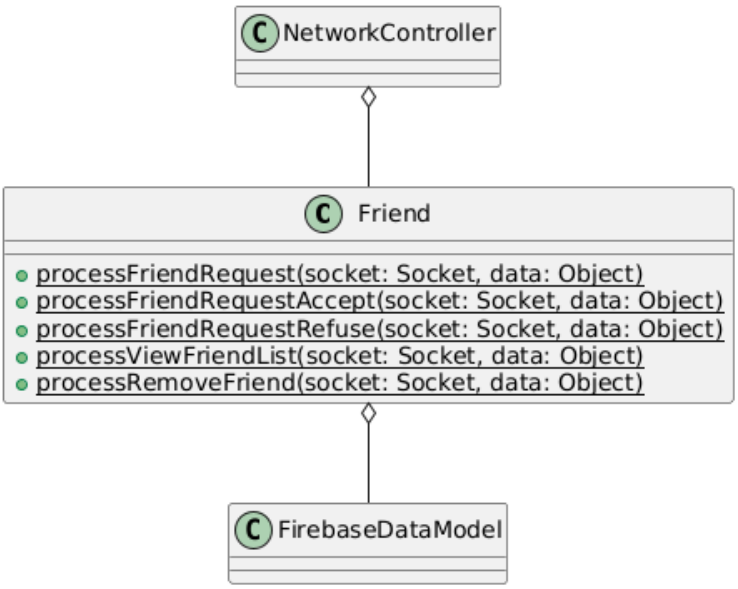
4.16 Component: Server/Message



The component provides functions to handle Direct Message related events.
It allows:

- Forwarding messages to other sockets.
- Save messages into the database.
- View messages read from the database.

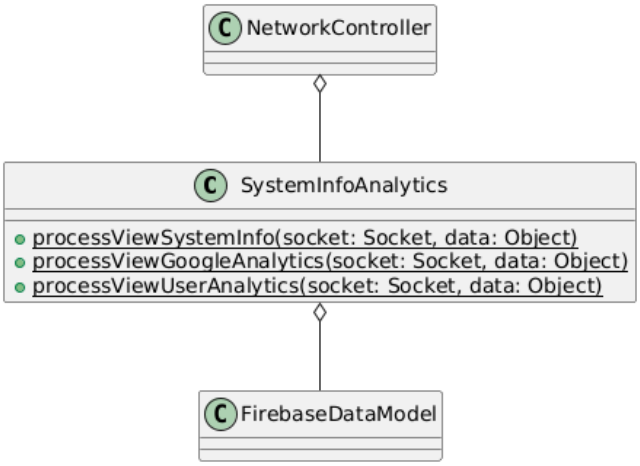
4.17 Component: Server/Friend



The component provides functions to handle Friend related events.
It allows:

- Forwarding friend requests to other sockets.
- Accept or refuse requests into the database.
- View friend requests read from the database.
- View the friend list of a specific user.

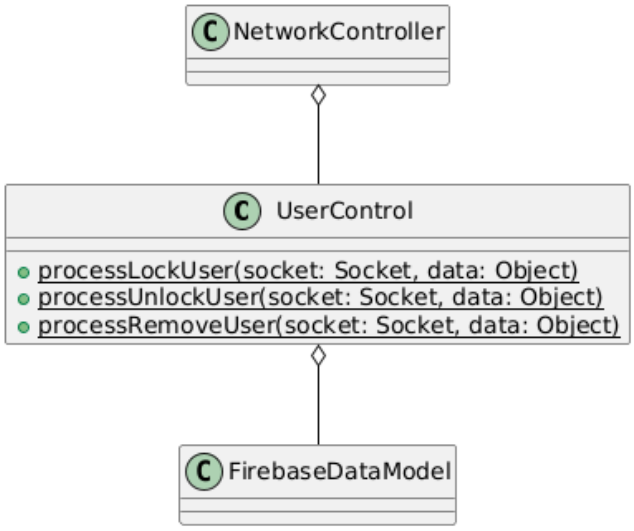
4.18 Component: Server/SystemInfoAnalytic



The component provides functions to create reports and send to admin related events.
It allows:

- Read the current status of the PC.
- Call Google Analytic API to create reports.
- Analyze a user from gathered information.

4.19 Component: Server/UserControl

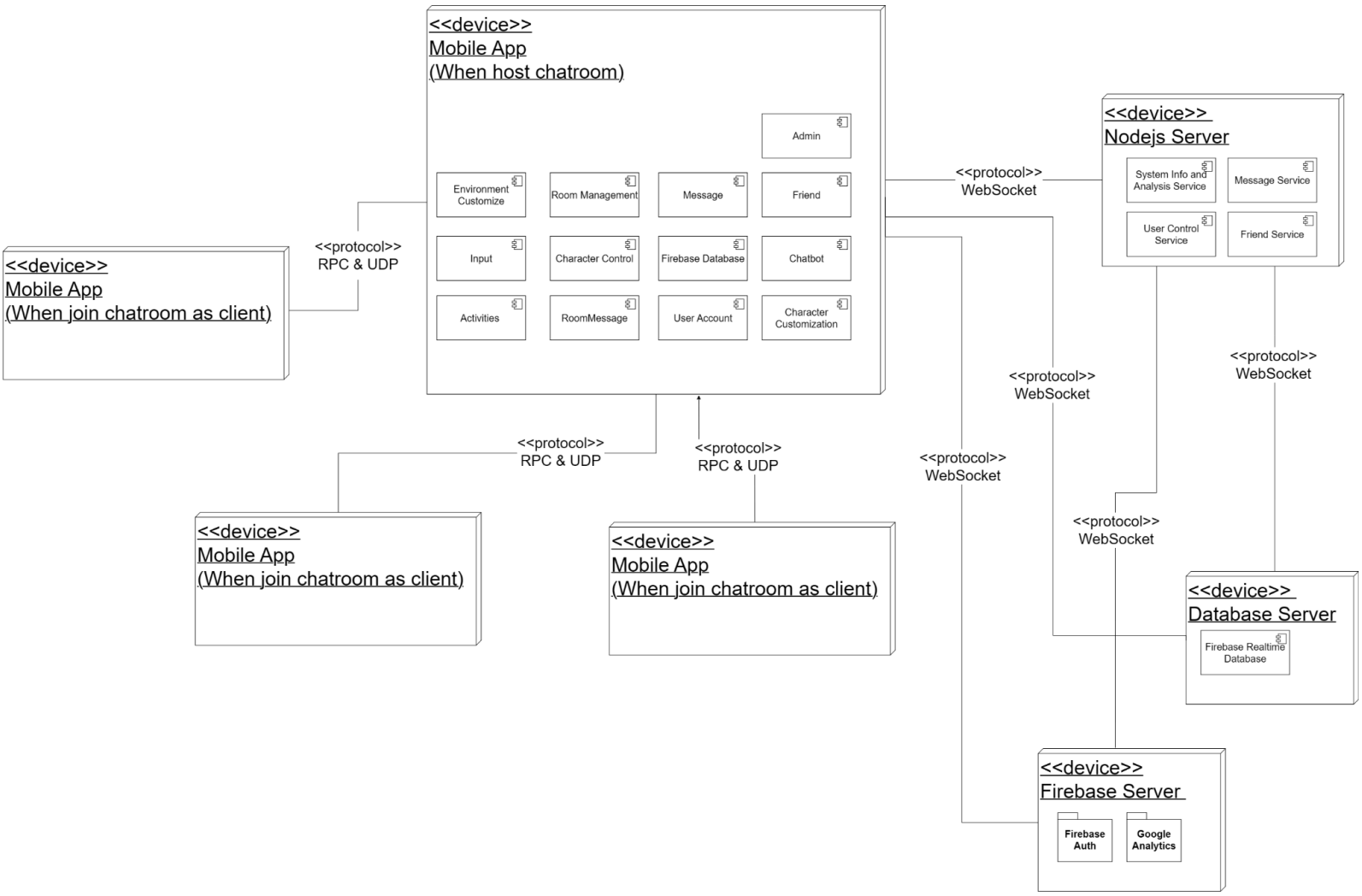


The component provides functions for admins to manage the user.
It allows:

- Locking a user.
- Unlocking a user.

- Remove an account.

5. Deployment



Node: Mobile App (Device)

This node represents the user's and admin's client device, which could be an android or ios device. This app is used to communicate with servers using basic functionality (user account, friend message,...) by using Websocket. This app is also used to host an online chatroom as server or join an online chatroom as client (Client-hosted listen server (https://docs-multiplayer.unity3d.com/netcode/current/terms-concepts/network-topologies/#client-hosted-listen-server). Host and client communications are provided by Unity Netcode. It also connects to the firebase realtime database (save some client side data) and firebase authentication (user authentication).

Node: Nodejs server (Device)

This node represents the server where the application's backend logic resides (for basic functionality). It handles processing requests (Websocket events) from the client, executing business logic, and interacting with the database server (firebase realtime database) and external firebase services.

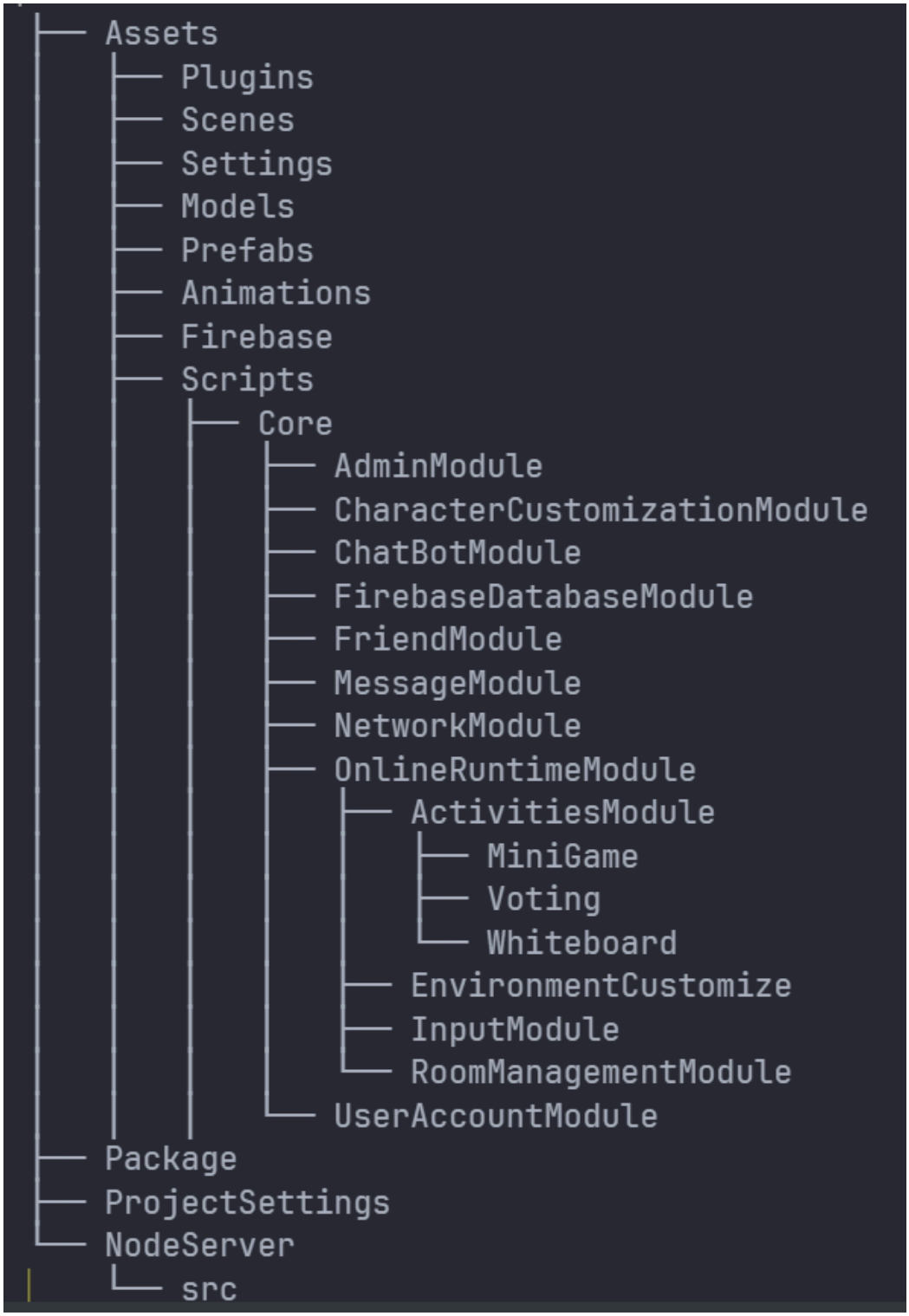
Node: Database Server (Device)

This node represents the database server that stores and manages the application's data. The database we are using here is firebase realtime database.

Node: Firebase Server (Device)

This node represents the firebase server that provides firebase authentication and firebase analytics services.

6. Implementation View



```
├── Assets: Unity project main folder.
│   ├── Plugins: Additions third-party plugins folder.
│   ├── Scenes: Folder store scenes.
│   ├── Settings: Unity URP settings and some other configuration folder.
│   ├── Models: Folder store 3D character models
│   ├── Prefabs: Folder store prefabs
│   ├── Animations: Folder store 3D character animations
│   ├── Firebase: Firebase sdk folder.
│   ├── Scripts: Folder store C# source code
│   │   ├── Core: Core implementation code.
│   │   │   ├── AdminModule: Folder for Admin Component code
│   │   │   ├── CharacterCustomizationModule
│   │   │   ├── ChatBotModule: Folder for ChatBot Component code
│   │   │   ├── FirebaseDatabaseModule: Folder for Firebase Component code
│   │   │   ├── FriendModule: Folder for Friend Component code
│   │   │   ├── MessageModule: Folder for Message Component code
│   │   │   ├── NetworkModule: Folder for Network Component code
│   │   │   ├── OnlineRuntimeModule: Folder for OnlineRuntime Component code
│   │   │   │   ├── ActivitiesModule: Folder for Activity Component code
│   │   │   │   │   ├── MiniGame: Folder for Room Mini Game Component code
│   │   │   │   │   ├── Voting: Folder for Room Voting Component code
│   │   │   │   │   └── Whiteboard: Folder for Room Whiteboard Component code
│   │   │   │   ├── EnvironmentCustomize:  Folder for Room Environment Customize Component code
│   │   │   │   ├── InputModule: Folder for Input Component code
│   │   │   │   └── RoomManagementModule: Folder for Room Component code
│   │   │   └── UserAccountModule:  Folder for Account Component code
│   │   └── Utilities: Some Utilities functions help implementation.
├── Package: Define Module install by Unity Package Manager.
├── ProjectSettings: Settings of Unity project.
└── NodeServer: nodejs application folder, also include RSA key, firebase SDK key of server.
    └── src: Folder store js code for server-side.
```