Group 7

Virtual Chat 3D
Software Architecture Document

Version 1.2

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

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.

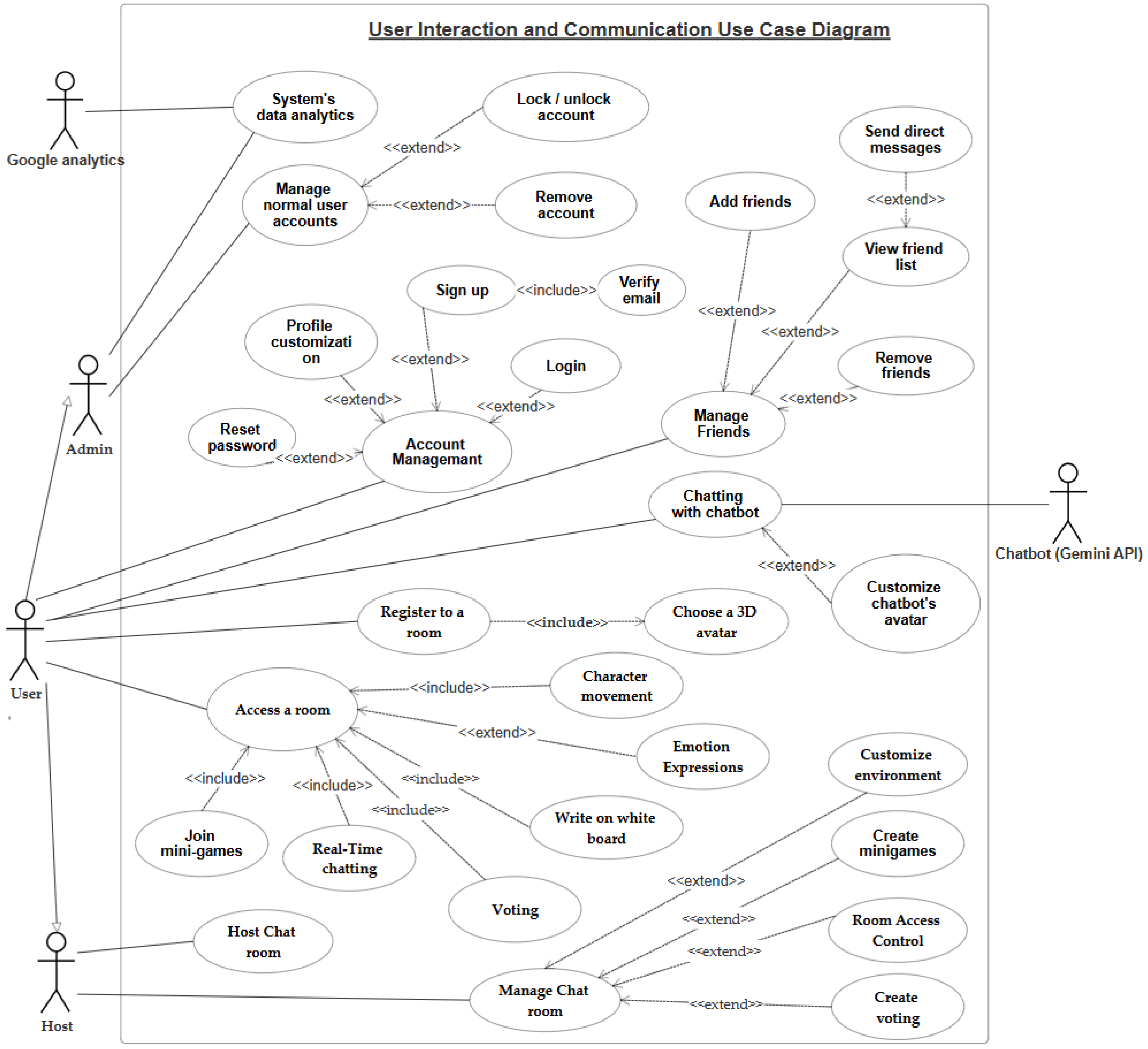
Architectural Constraints:

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:

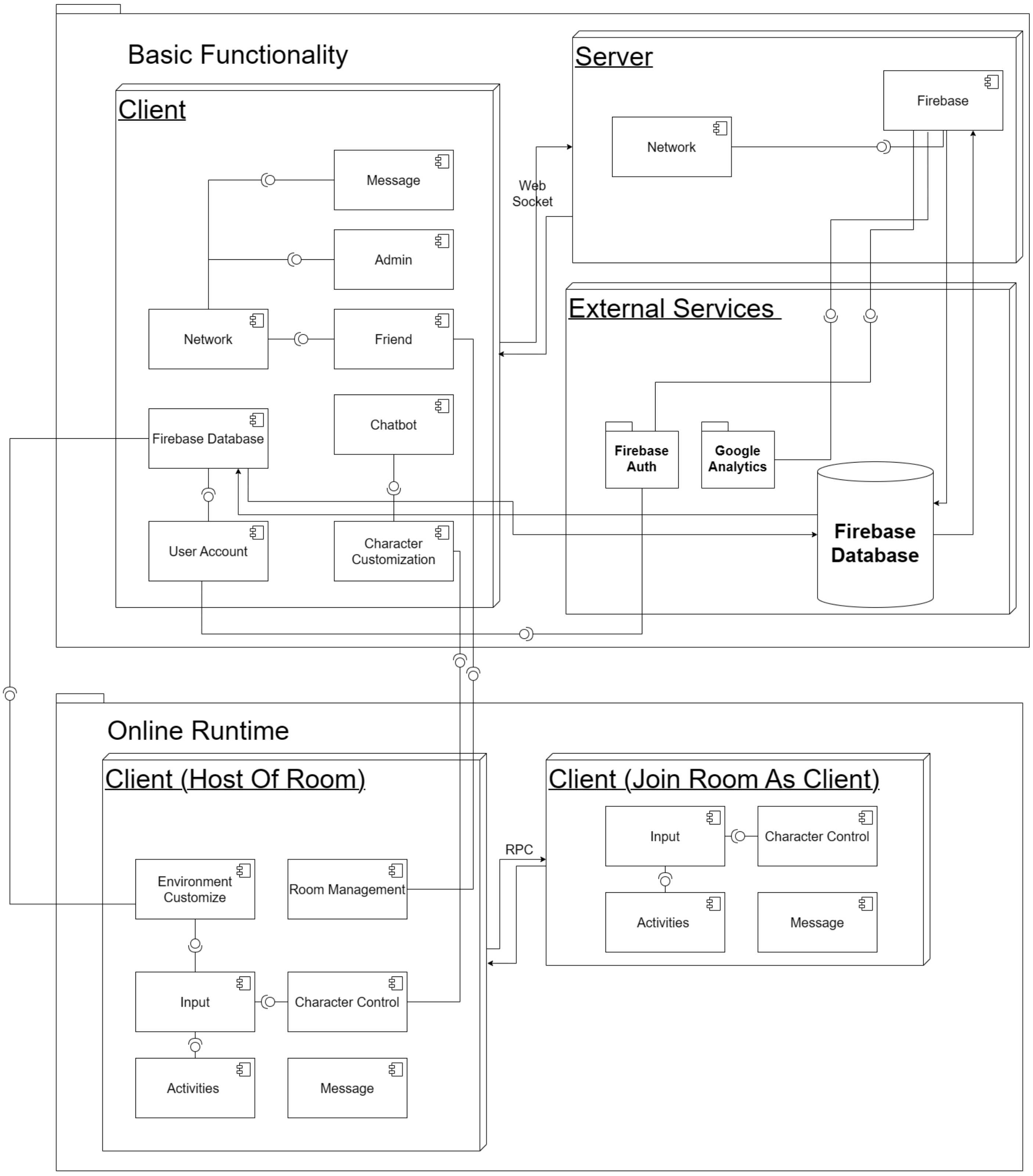
        - Core Engine: Used for 3D game development, rendering, physics, and animation.
        - Visual Scripting: For creating game logic without writing code.
        - Asset Store: A marketplace for purchasing or downloading free 3D models, scripts, and other assets.
    - JetBrains Rider:
        - C# IDE: 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



There are 2 subsystem in the project:

a. The Basic Functionality subsystem:

- This subsystem provides the following features:
  - Account authentication.
  - User account profile management
  - Social functionality: friend management, message with friend.
  - 3D interactive Chatbot.
  - It also provides firebase database access on client-side.
- The subsystem contains 2 parts:
  - The Client - implemented with Unity.
  - The Server - implemented using NodeJS.
  - These parts interact with each other through the Interface provided by the Network Component using the Web Socket protocol provided by SocketIO.
- The Components in the Client Side 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.
  - Control: a class to handle and control all the logic of the Component.
- Here is a brief description of the components in the Client Side of the subsystem:

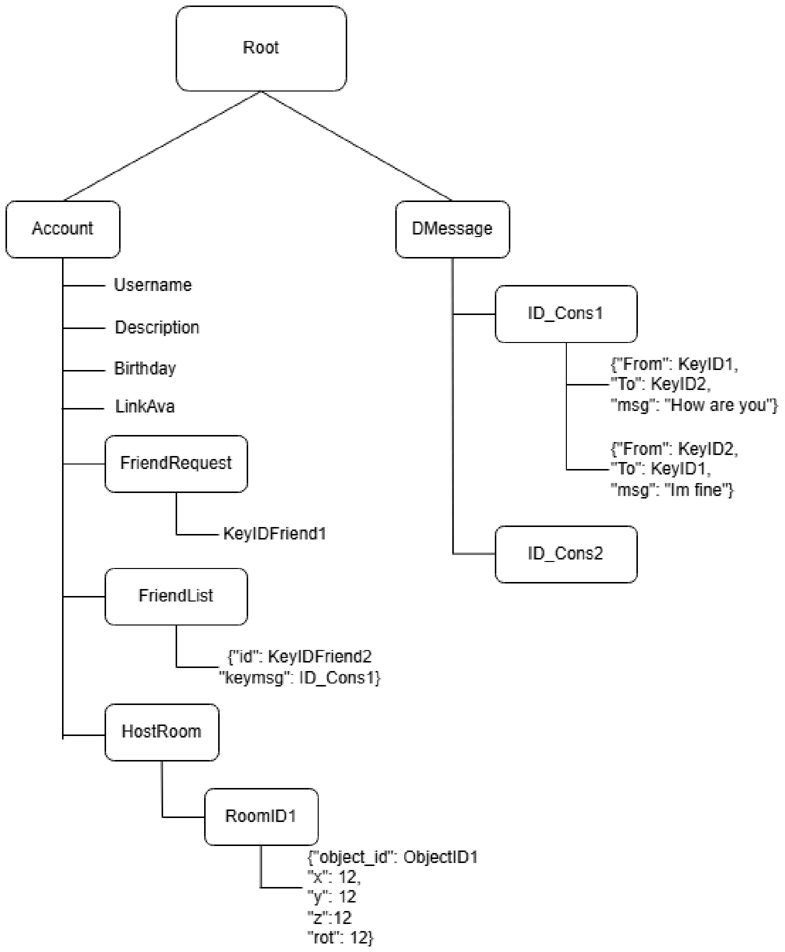
Component (Client Side)	**Description**
UserAccount	Provide authentication and profile management for the user.
Friend	Provide friend management functions by reading data from the Firebase and display on the view. Real-time friend requests are handled by sending requests to the server.
Message	Provide real-time messaging functions by sending and receiving messages from the server. It also read past messages from the Firebase.
Admin	Provide administrator functionalities: lock/unlock users, reading statistics from Firebase.
FirebaseDatabase	Wraps up Firebase SDK for the Client to read directly from Firebase.
Chatbot	Provide interactive chat using Gemini APIs, and enable customization for the Chatbot model.
CharacterCustomization	Handle 3D models importing, manage imported models, provide an interface for the Chatbot Component and the Character Control Component (in the Online Runtime subsystem).
Network	Provide functions for Clients to communicate with the Server in cryptic manner.

- The Server Side utilizes the web socket with the SocketIO library to communicate with the Clients and use the Firebase Admin SDK to manage the Firebase's data, along with the built-in Google Analytics for Firebase to further extract information..

Component (Server Side)	**Description**

Network	Provide functions for the Server to process request from Clients in cryptic manner: - Direct Messages. - Friend Management. - Admin functionality.
FirebaseDataModel	Provide: - Functions to read/write the database and execute admin tasks - Functions to interact with the Google Analytics APIs and to read information on the server PC..

Here is the structure of data stored in Firebase:



b. **Online Runtime subsystem**:
- This subsystem runs only on the Client Side.
- It provides the following online room features:
    + Room management
    + Environment customization
    + Character in room control
    + Messaging/Chatting
    + Other Activities.
- The Client hosts or joins the chat room by using the RPC provided by Unity Netcode.

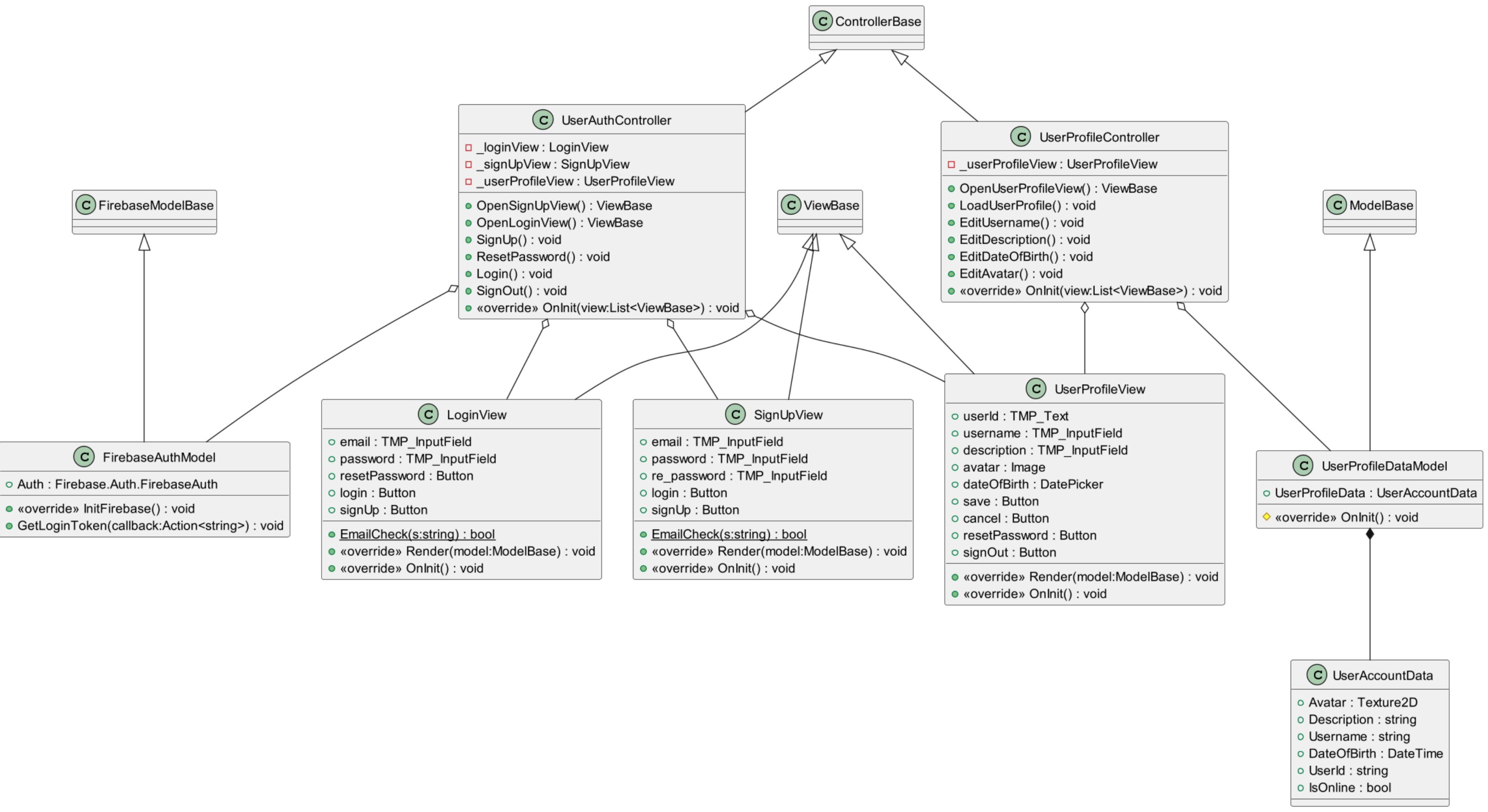
- Here is a brief description of the components of the subsystem:

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).
EnvironmentCustomize	Provide functions for managing, editing objects in the room. It also utilizes the FirebaseDatabase Component in the Basic functionalities subsystem for storing room details in Firebase.
Message	Provide functions for chatting in a chat room.
Input	Provide the UI for users to control their 3D avatars in a chat room. (such as joystick, touch pad, etc.)
CharacterControl	Control the avatar's movement, animation, and emotion.
Activities	Provide functions for other activities in the chat room such as voting, whiteboard, mini games.

4.1 Component: UserAccount

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

- Sign up, login, sign out, reset password and get login token (use to send login state to server)
- View and edit profile: Avatar, Description, Username, Date of birth.


ControllerBase
UserAuthController
_loginView : LoginView
_signUpView : SignUpView
_userProfileView : UserProfileView
OpenSignUpView() : ViewBase
OpenLoginView() : ViewBase
SignUp() : void
ResetPassword() : void
Login() : void
SignOut() : void
«override» OnInit(view:List<ViewBase>) : void
UserProfileController
_userProfileView : UserProfileView
OpenUserProfileView() : ViewBase
LoadUserProfile() : void
EditUsername() : void
EditDescription() : void
EditDateOfBirth() : void
EditAvatar() : void
«override» OnInit(view:List<ViewBase>) : void
FirebaseModelBase
ViewBase
ModelBase
FirebaseAuthModel
Auth : Firebase.Auth.FirebaseAuth
«override» InitFirebase() : void
GetLoginToken(callback:Action<string>) : void
LoginView
email : TMP_InputField
password : TMP_InputField
resetPassword : Button
login : Button
signUp : Button
EmailCheck(s:string) : bool
«override» Render(model:ModelBase) : void
«override» OnInit() : void
SignUpView
email : TMP_InputField
password : TMP_InputField
re_password : TMP_InputField
login : Button
signUp : Button
EmailCheck(s:string) : bool
«override» Render(model:ModelBase) : void
«override» OnInit() : void
UserProfileView
userId : TMP_Text
username : TMP_InputField
description : TMP_InputField
avatar : Image
dateOfBirth : DatePicker
save : Button
cancel : Button
resetPassword : Button
signOut : Button
«override» Render(model:ModelBase) : void
«override» OnInit() : void
UserProfileDataModel
UserProfileData : UserAccountData
«override» OnInit() : void
UserAccountData
Avatar : Texture2D
Description : string
Username : string
DateOfBirth : DateTime
UserId : string
IsOnline : bool

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.
- UserProfileController:
  - Reading and editing information from Firebase using 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.

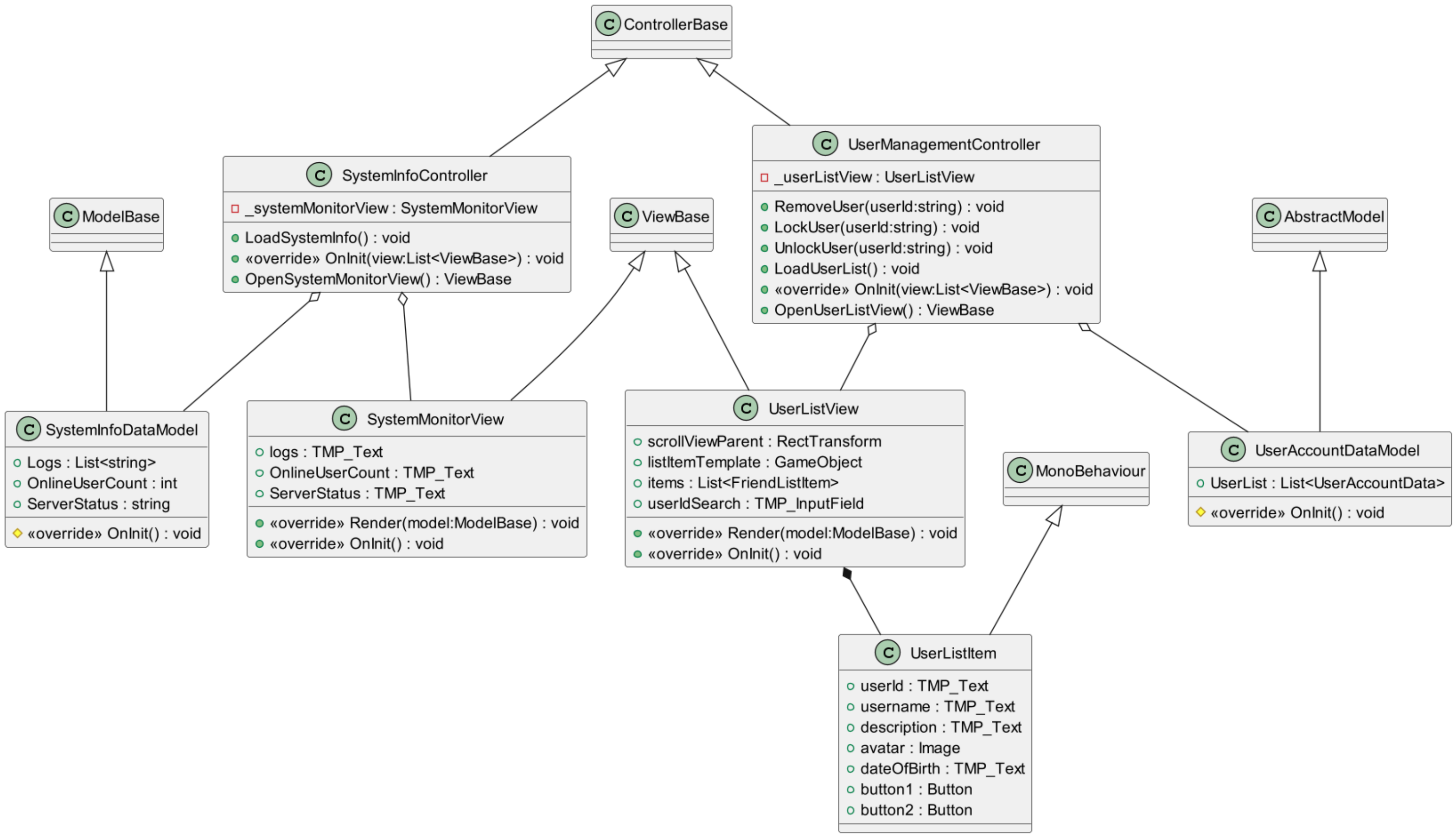
Models:

- 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: Admin

This component provides user management functionality. Including:

- View user list, lock/unlock user, remove user.
- System state monitoring



ControllerBase
SystemInfoController
_systemMonitorView : SystemMonitorView
LoadSystemInfo() : void
«override» OnInit(view:List<ViewBase>) : void
OpenSystemMonitorView() : ViewBase
UserManagementController
_userListView : UserListView
RemoveUser(userId:string) : void
LockUser(userId:string) : void
UnlockUser(userId:string) : void
LoadUserList() : void
«override» OnInit(view:List<ViewBase>) : void
OpenUserListView() : ViewBase
ModelBase
ViewBase
AbstractModel
SystemInfoDataModel
Logs : List<string>
OnlineUserCount : int
ServerStatus : string
«override» OnInit() : void
SystemMonitorView
logs : TMP_Text
OnlineUserCount : TMP_Text
ServerStatus : TMP_Text
«override» Render(model:ModelBase) : void
«override» OnInit() : void
UserListView
scrollViewParent : RectTransform
listItemTemplate : GameObject
items : List<FriendListItem>
userIdSearch : TMP_InputField
«override» Render(model:ModelBase) : void
«override» OnInit() : void
MonoBehaviour
UserAccountDataModel
UserList : List<UserAccountData>
«override» OnInit() : void
UserListItem
userId : TMP_Text
username : TMP_Text
description : TMP_Text
avatar : Image
dateOfBirth : TMP_Text
button1 : Button
button2 : Button

Controllers:

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

Views:

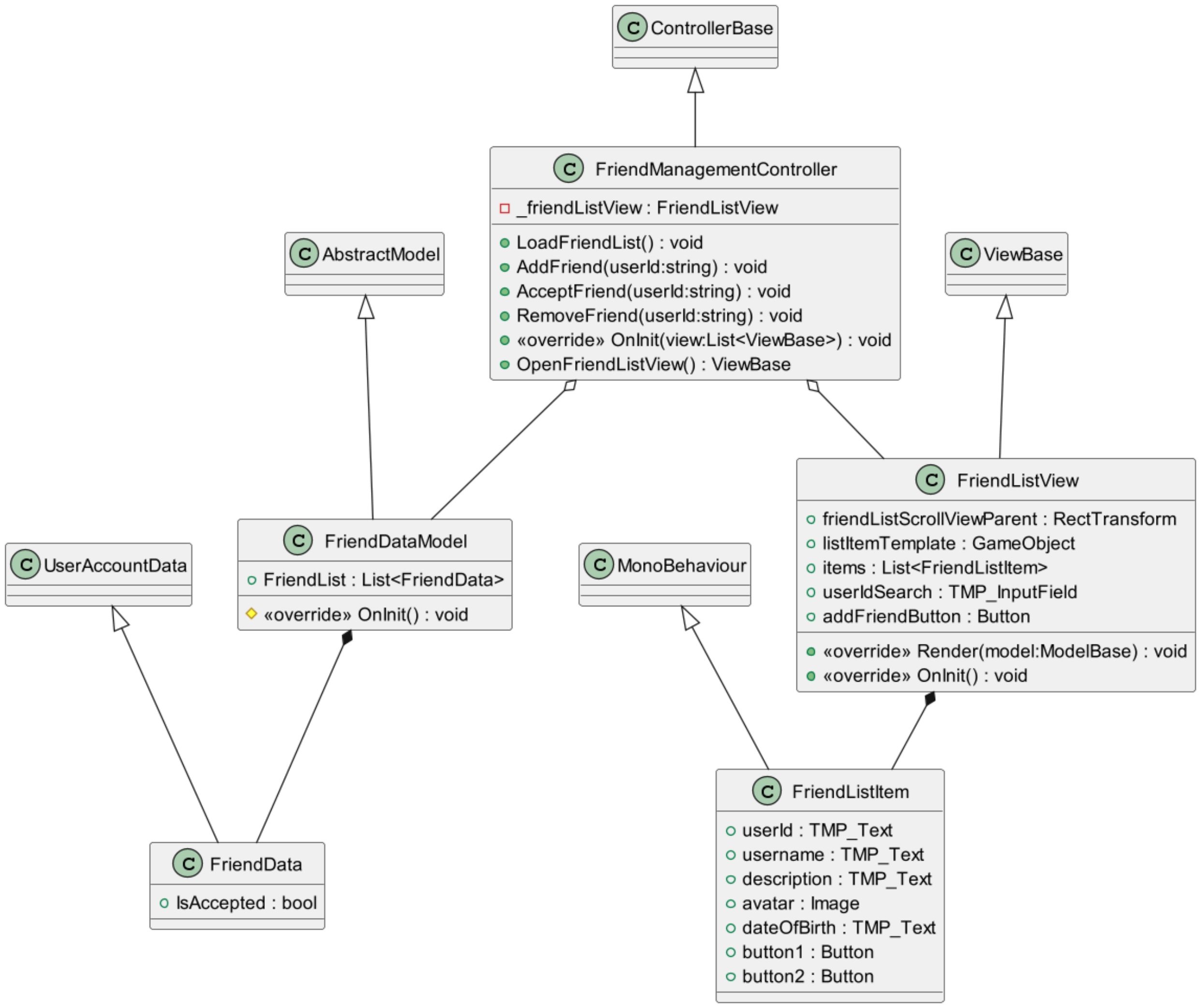
- UserListView:
  - Handles the interface for viewing users list functions.
  - Read the Admin's command for execution.
- SystemMonitorView:
  - View statistical information about the System and server.

Models:

- UserAccountDataModel:
  - Store normal users' data read from the server or Firebase.
- SystemInfoModel:
  - Stores the data about the system read from the server or Google Analytics.

4.3 Component: Friend

This component provides friend management functionality. Including: view friend list, send new friend request, accept/refuse friend request, remove friend.


ControllerBase
FriendManagementController
_friendListView : FriendListView
LoadFriendList() : void
AddFriend(userId:string) : void
AcceptFriend(userId:string) : void
RemoveFriend(userId:string) : void
«override» OnInit(view:List<ViewBase>) : void
OpenFriendListView() : ViewBase
AbstractModel
ViewBase
FriendListView
friendListScrollViewParent : RectTransform
listItemTemplate : GameObject
items : List<FriendListItem>
userIdSearch : TMP_InputField
addFriendButton : Button
«override» Render(model:ModelBase) : void
«override» OnInit() : void
UserAccountData
FriendDataModel
FriendList : List<FriendData>
«override» OnInit() : void
MonoBehaviour
FriendListItem
userId : TMP_Text
username : TMP_Text
description : TMP_Text
avatar : Image
dateOfBirth : TMP_Text
button1 : Button
button2 : Button
FriendData
IsAccepted : bool

Controllers:

- FriendManagementController:
  - Controls the logic for managing a user's friends list, including loading, adding, accepting, and removing friends.
  - 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.

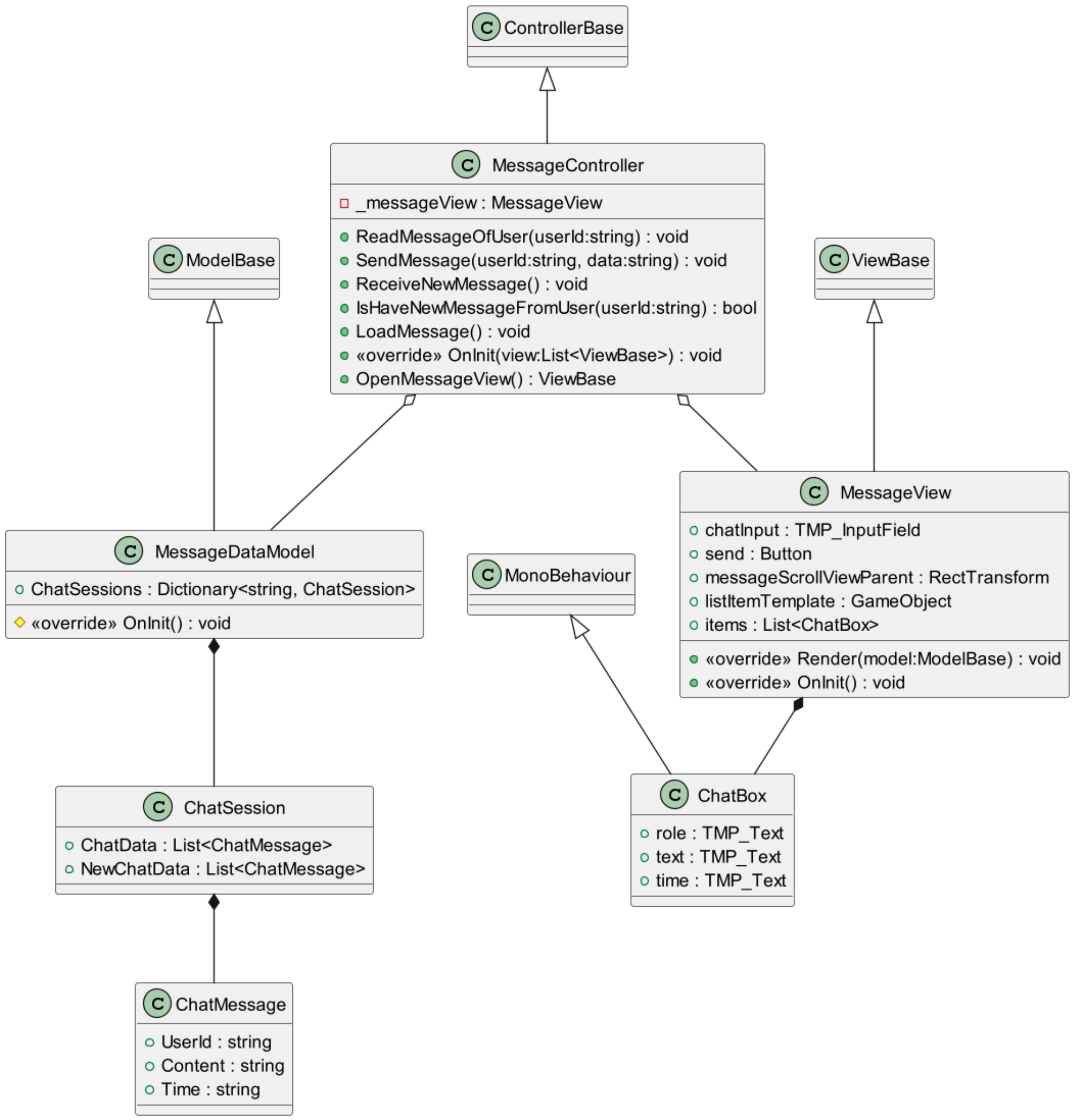
- ○ Reads user input for adding friends and interacting with the friend list.

Models:

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

4.4 Component: Message

This component provides message functionality. Including sending and receiving messages between users.


ControllerBase
MessageController
_messageView : MessageView
ReadMessageOfUser(userId:string) : void
SendMessage(userId:string, data:string) : void
ReceiveNewMessage() : void
IsHaveNewMessageFromUser(userId:string) : bool
LoadMessage() : void
«override» OnInit(view:List<ViewBase>) : void
OpenMessageView() : ViewBase
ModelBase
ViewBase
MessageDataModel
ChatSessions : Dictionary<string, ChatSession>
«override» OnInit() : void
MonoBehaviour
MessageView
chatInput : TMP_InputField
send : Button
messageScrollViewParent : RectTransform
listItemTemplate : GameObject
items : List<ChatBox>
«override» Render(model:ModelBase) : void
«override» OnInit() : void
ChatSession
ChatData : List<ChatMessage>
NewChatData : List<ChatMessage>
ChatBox
role : TMP_Text
text : TMP_Text
time : TMP_Text
ChatMessage
UserId : string
Content : string
Time : string

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.
  - Provides access to the MessageView to display and interact with chat messages.

Views:

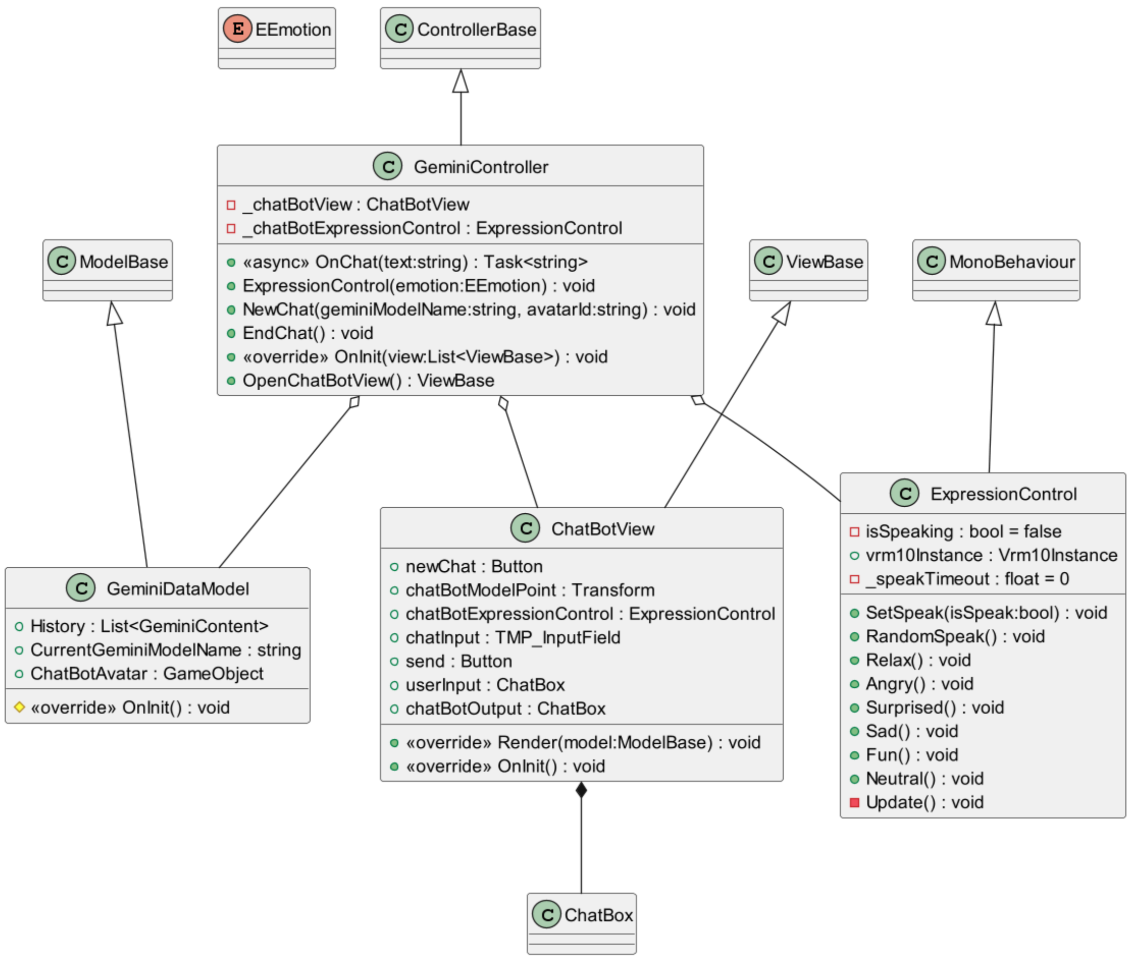
- MessageView:
  - Manages the user interface for chat interactions, including sending and viewing messages.
  - Displays a scrollable list of messages and allows user input through a text field and send button.
  - Dynamically renders chat items using a template for individual chat messages.

Models:

- MessageDataModel:
  - Stores the user's chat sessions as a dictionary, mapping user IDs to chat sessions (ChatSession).
  - Provides message data to the MessageController for operations and updates.

4.5 Component: ChatBot

This component provides ChatBot functionality based on Gemini API. Including interactive chat with Gemini, change Chatbot emotion according to chat content.


EEmotion
ControllerBase
GeminiController
_chatBotView : ChatBotView
_chatBotExpressionControl : ExpressionControl
«async» OnChat(text:string) : Task<string>
ExpressionControl(emotion:EEmotion) : void
NewChat(geminiModelName:string, avatarId:string) : void
EndChat() : void
«override» OnInit(view:List<ViewBase>) : void
OpenChatBotView() : ViewBase
ModelBase
ViewBase
MonoBehaviour
GeminiDataModel
History : List<GeminiContent>
CurrentGeminiModelName : string
ChatBotAvatar : GameObject
«override» OnInit() : void
ChatBotView
newChat : Button
chatBotModelPoint : Transform
chatBotExpressionControl : ExpressionControl
chatInput : TMP_InputField
send : Button
userInput : ChatBox
chatBotOutput : ChatBox
«override» Render(model:ModelBase) : void
«override» OnInit() : void
ExpressionControl
isSpeaking : bool = false
vrm10Instance : Vrm10Instance
_speakTimeout : float = 0
SetSpeak(isSpeak:bool) : void
RandomSpeak() : void
Relax() : void
Angry() : void
Surprised() : void
Sad() : void
Fun() : void
Neutral() : void
Update() : void
ChatBox

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.
- ExpressionControl:

- Unity Component script to controls 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.

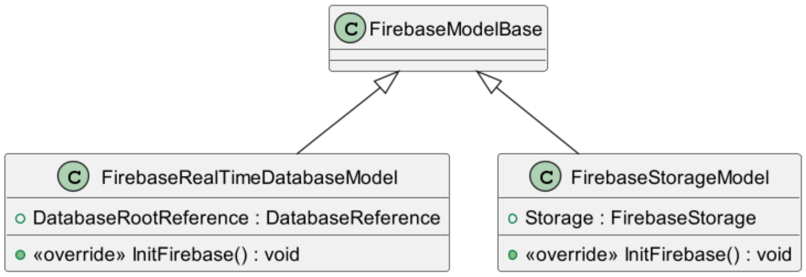
Models:

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

4.6 Component: FirebaseDatabase

This component provides firebase realtime database functionality. This component wraps Firebase realtime database and firebase storage api provides:

- Read/write/update/delete operation to firebase realtime database (used to store client-side data of the users)
- Read/write/update/delete operation to firebase storage (used to store image/file of the users)



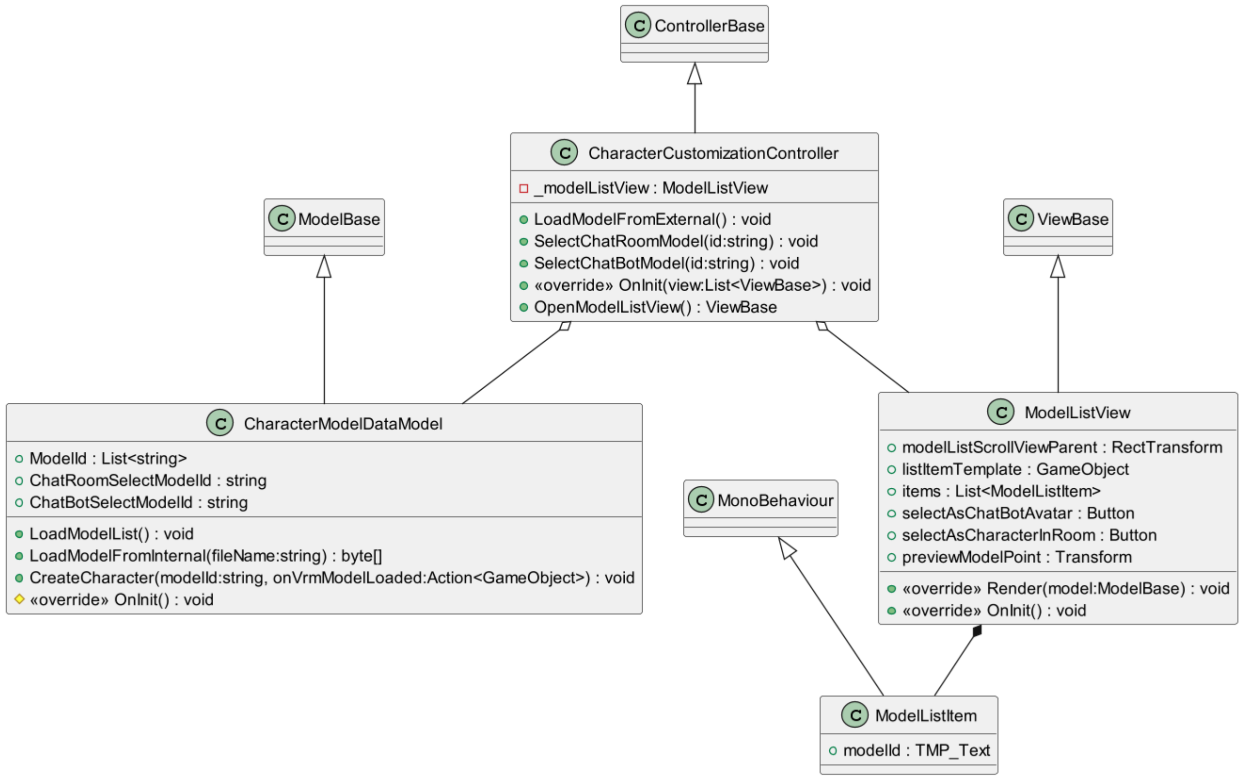
FirebaseModelBase
FirebaseRealTimeDatabaseModel
DatabaseRootReference : DatabaseReference
«override» InitFirebase() : void
FirebaseStorageModel
Storage : FirebaseStorage
«override» InitFirebase() : void

Models:

- FirebaseRealTimeDatabaseModel:
  - Provides access to the root reference of Firebase Realtime Database for interacting with structured data in real-time.
  - Initializes the database instance for use in the application.
- FirebaseStorageModel:
  - Manages Firebase Storage for handling file uploads and downloads.
  - Initializes the storage instance, allowing the application to store and retrieve large files such as images or videos.

4.7 Component: CharacterCustomize

This component provides character customization functionality. Including: import/preview/select model for user's character in online room and for chatbot avatar.



ControllerBase
CharacterCustomizationController
_modelListView : ModelListView
LoadModelFromExternal() : void
SelectChatRoomModel(id:string) : void
SelectChatBotModel(id:string) : void
«override» OnInit(view:List<ViewBase>) : void
OpenModelListView() : ViewBase
ModelBase
ViewBase
CharacterModelDataModel
ModelId : List<string>
ChatRoomSelectModelId : string
ChatBotSelectModelId : string
LoadModelList() : void
LoadModelFromInternal(fileName:string) : byte[]
CreateCharacter(modelId:string, onVrmModelLoaded:Action<GameObject>) : void
«override» OnInit() : void
MonoBehaviour
ModelListView
modelListScrollViewParent : RectTransform
listItemTemplate : GameObject
items : List<ModelListItem>
selectAsChatBotAvatar : Button
selectAsCharacterInRoom : Button
previewModelPoint : Transform
«override» Render(model:ModelBase) : void
«override» OnInit() : void
ModelListItem
modelId : TMP_Text

Controller:

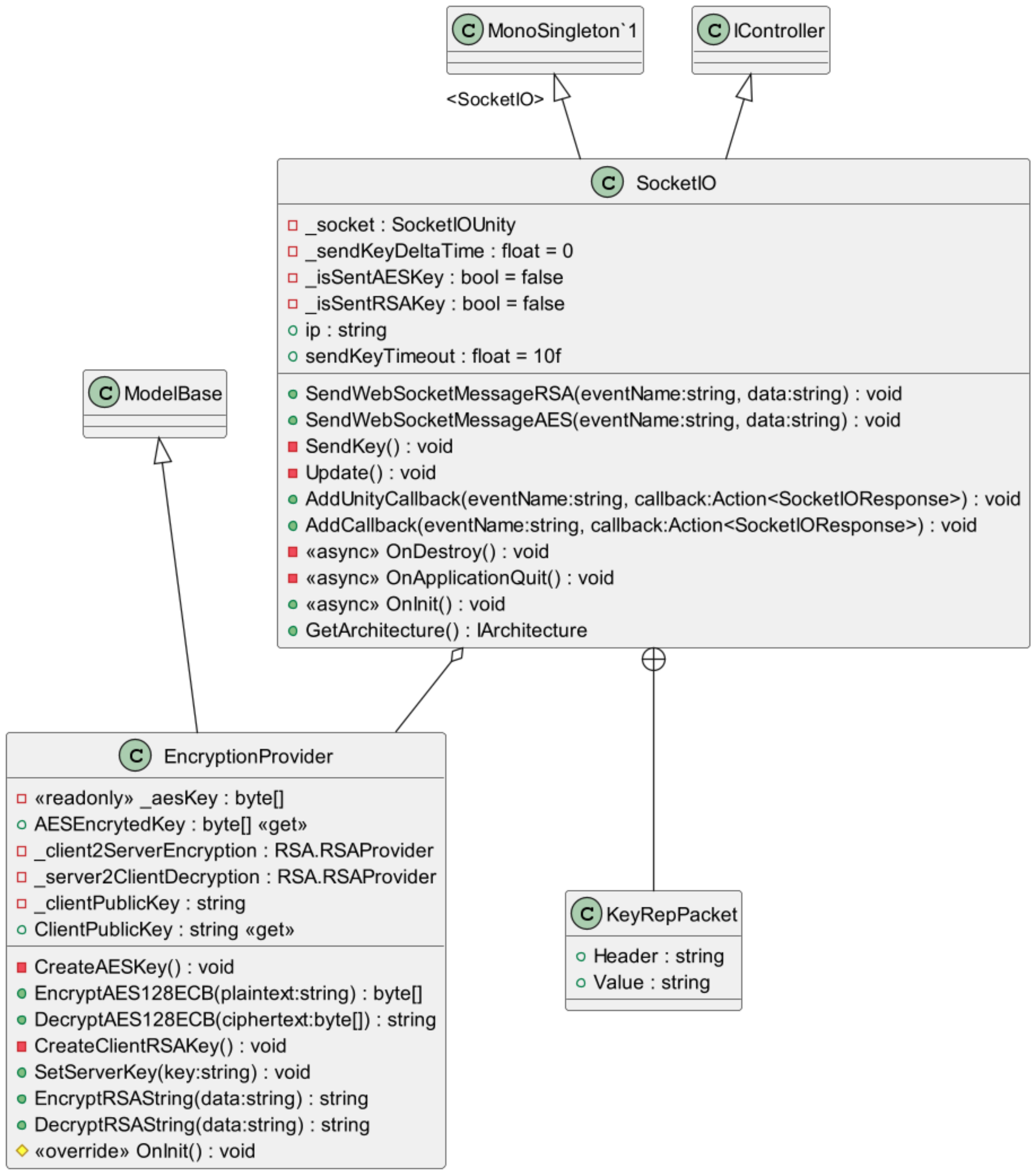
- CharacterCustomizationController:
  - Manages logic for character customization, including importing models from external memory of mobile devices and selecting specific models for chat rooms or chatbots.
  - Handles interaction between the user and the ModelListView, enabling model selection and updates.

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

4.8 Component: Network

This component provides security network communication between client and server functionality based on SocketIO. Include connection management, encrypt when sending data, decrypt when receiving data.



MonoSingleton`1
IController
<SocketIO>
SocketIO
_socket : SocketIOUnity
_sendKeyDeltaTime : float = 0
_isSentAESKey : bool = false
_isSentRSAKey : bool = false
ip : string
sendKeyTimeout : float = 10f
SendWebSocketMessageRSA(eventName:string, data:string) : void
SendWebSocketMessageAES(eventName:string, data:string) : void
SendKey() : void
Update() : void
AddUnityCallback(eventName:string, callback:Action<SocketIOResponse>) : void
AddCallback(eventName:string, callback:Action<SocketIOResponse>) : void
«async» OnDestroy() : void
«async» OnApplicationQuit() : void
«async» OnInit() : void
GetArchitecture() : IArchitecture
ModelBase
EncryptionProvider
«readonly» _aesKey : byte[]
AESEncrytedKey : byte[] «get»
_client2ServerEncryption : RSA.RSAProvider
_server2ClientDecryption : RSA.RSAProvider
_clientPublicKey : string
ClientPublicKey : string «get»
CreateAESKey() : void
EncryptAES128ECB(plaintext:string) : byte[]
DecryptAES128ECB(ciphertext:byte[]) : string
CreateClientRSAKey() : void
SetServerKey(key:string) : void
EncryptRSAString(data:string) : string
DecryptRSAString(data:string) : string
«override» OnInit() : void
KeyRepPacket
Header : string
Value : string

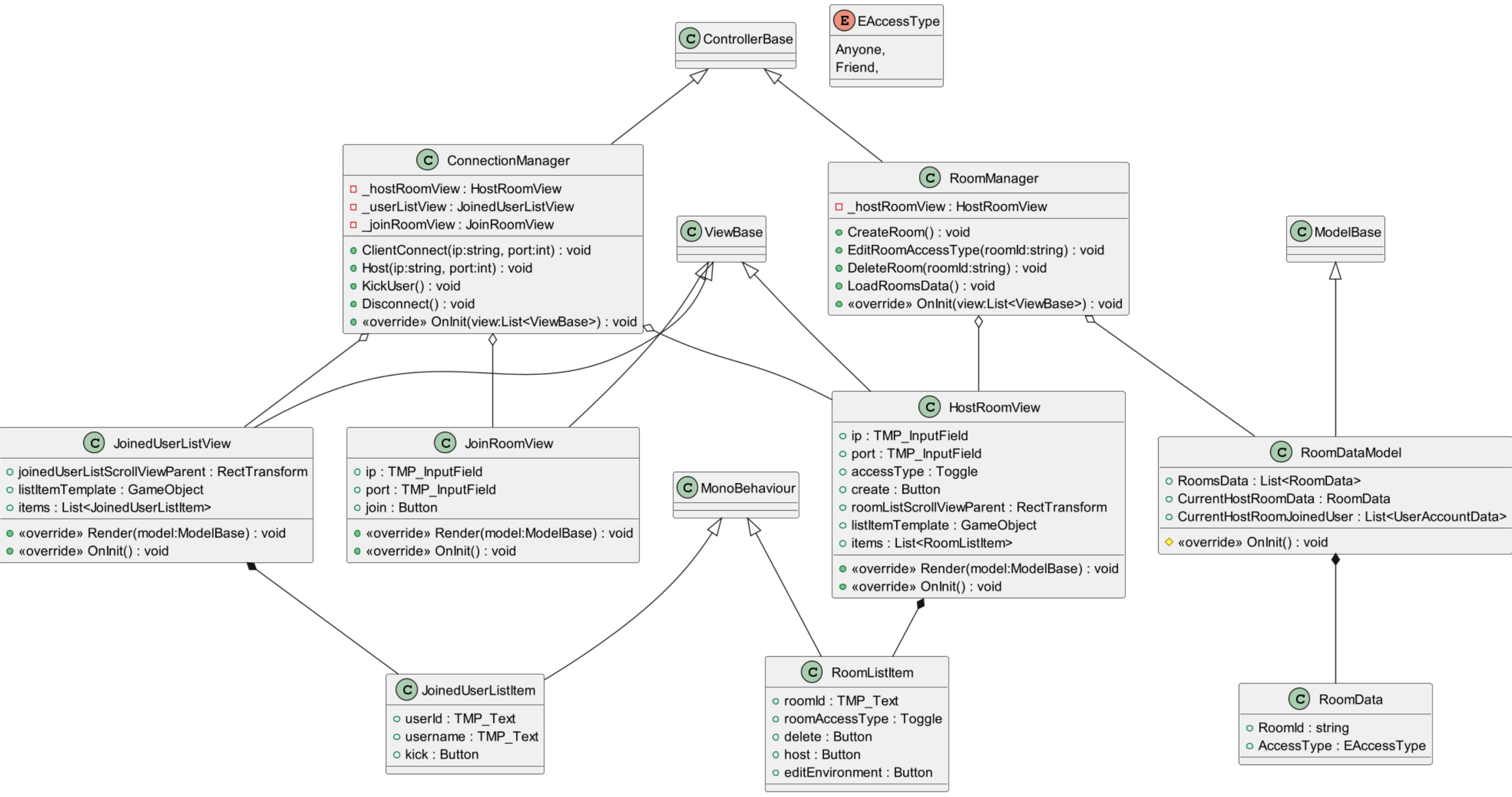
- EncryptionProvider:
  - Manages AES and RSA encryption for secure communication.
  - Creates and stores AES and RSA keys, encrypts and decrypts data using these keys.
  - Supports bidirectional encryption between client and server by setting up RSA keys for secure transmission and AES for encrypted communication.
  - Provides functionalities like EncryptAES128ECB, DecryptAES128ECB, EncryptRSAString, and DecryptRSAString for handling encryption tasks.
  - Manages server-side public key setup and ensures encryption protocol initialization.
- SocketIO:
  - Manages WebSocket communication using encryption to ensure secure data transfer.
  - Handles sending encrypted messages using RSA or AES (SendWebSocketMessageRSA and SendWebSocketMessageAES).

- Establishes WebSocket callbacks for event-driven communication.
- Handles encryption key exchange (SendKey) for secure messaging over the network.
- Manages WebSocket connection lifecycle, including initialization (OnInit) and cleanup (OnDestroy, OnApplicationQuit).

4.9 Component: OnlineRuntime/RoomManagement

This component provides connection and room management functionality. Including:

- Management connection: host room/join to room
- View list of joined users for host, kick user for host.
- Manage room: create/delete room, set room access scope: anyone/only friend.


ControllerBase
EAccessType
Anyone,
Friend,
ConnectionManager
_hostRoomView : HostRoomView
_userListView : JoinedUserListView
_joinRoomView : JoinRoomView
ClientConnect(ip:string, port:int) : void
Host(ip:string, port:int) : void
KickUser() : void
Disconnect() : void
«override» OnInit(view:List<ViewBase>) : void
RoomManager
_hostRoomView : HostRoomView
CreateRoom() : void
EditRoomAccessType(roomId:string) : void
DeleteRoom(roomId:string) : void
LoadRoomsData() : void
«override» OnInit(view:List<ViewBase>) : void
ViewBase
ModelBase
JoinedUserListView
joinedUserListScrollViewParent : RectTransform
listItemTemplate : GameObject
items : List<JoinedUserListItem>
«override» Render(model:ModelBase) : void
«override» OnInit() : void
JoinRoomView
ip : TMP_InputField
port : TMP_InputField
join : Button
«override» Render(model:ModelBase) : void
«override» OnInit() : void
MonoBehaviour
HostRoomView
ip : TMP_InputField
port : TMP_InputField
accessType : Toggle
create : Button
roomListScrollViewParent : RectTransform
listItemTemplate : GameObject
items : List<RoomListItem>
«override» Render(model:ModelBase) : void
«override» OnInit() : void
RoomDataModel
RoomsData : List<RoomData>
CurrentHostRoomData : RoomData
CurrentHostRoomJoinedUser : List<UserAccountData>
«override» OnInit() : void
JoinedUserListItem
userId : TMP_Text
username : TMP_Text
kick : Button
RoomListItem
roomId : TMP_Text
roomAccessType : Toggle
delete : Button
host : Button
editEnvironment : Button
RoomData
RoomId : string
AccessType : EAccessType

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.

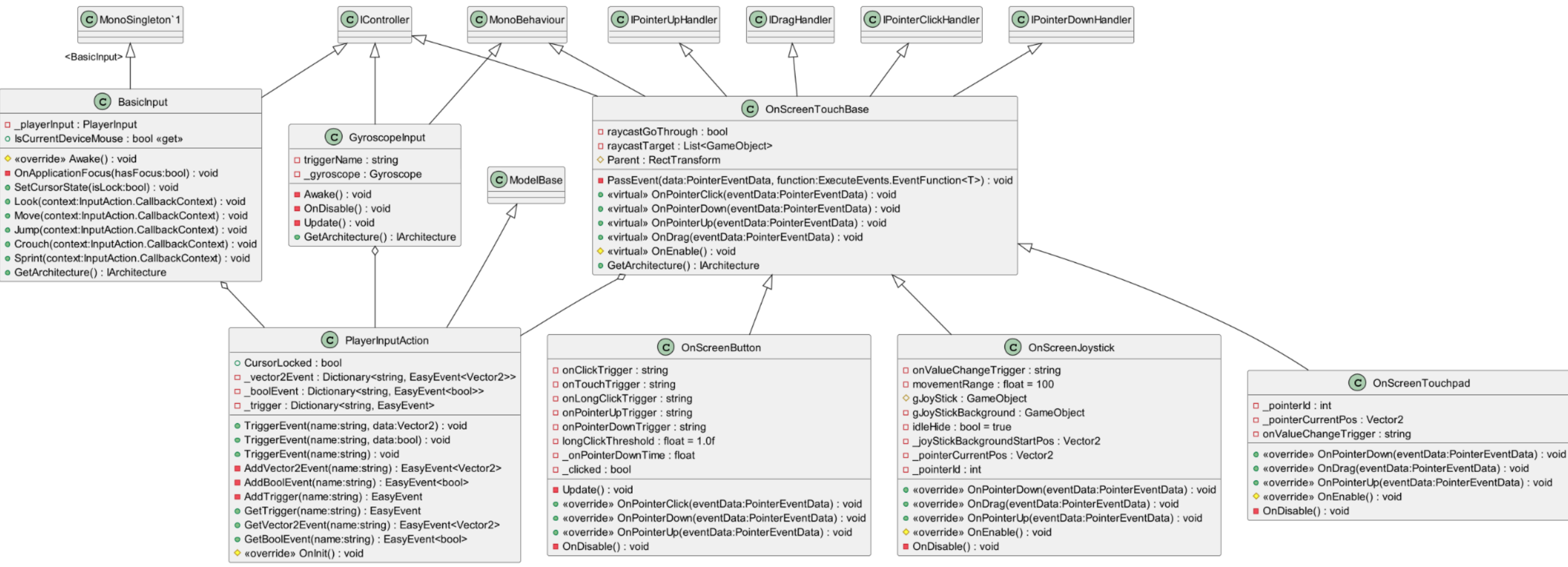
Model:

- RoomDataModel:
    - Contains created room data.

4.10 Component: OnlineRuntime/Input

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

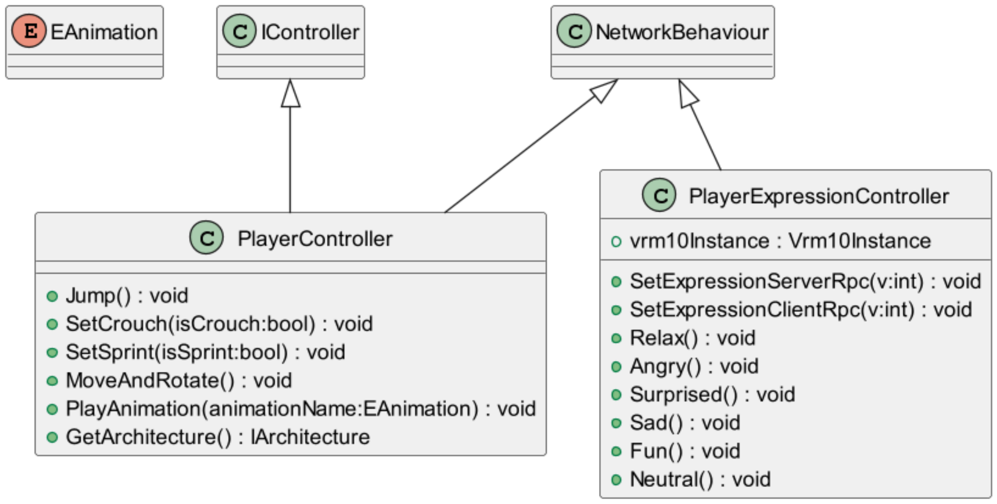
- Keyboard and Mouse: move (wasd), look (mouse), jump (space), crouch (c), sprint(shift).
- Touch screen: move (on screen virtual joystick), look (on screen touchpad), jump (on screen button), crouch (on screen button), sprint(on screen button).
- sensor input: look (Gyroscope)


MonoSingleton`1
IController
MonoBehaviour
IPointerUpHandler
IDragHandler
IPointerClickHandler
IPointerDownHandler
<BasicInput>
BasicInput
_playerInput : PlayerInput
IsCurrentDeviceMouse : bool «get»
«override» Awake() : void
OnApplicationFocus(hasFocus:bool) : void
SetCursorState(isLock:bool) : void
Look(context:InputAction.CallbackContext) : void
Move(context:InputAction.CallbackContext) : void
Jump(context:InputAction.CallbackContext) : void
Crouch(context:InputAction.CallbackContext) : void
Sprint(context:InputAction.CallbackContext) : void
GetArchitecture() : IArchitecture
GyroscopeInput
triggerName : string
_gyroscope : Gyroscope
Awake() : void
OnDisable() : void
Update() : void
GetArchitecture() : IArchitecture
ModelBase
OnScreenTouchBase
raycastGoThrough : bool
raycastTarget : List<GameObject>
Parent : RectTransform
PassEvent(data:PointerEventData, function:ExecuteEvents.EventFunction<T>) : void
«virtual» OnPointerClick(eventData:PointerEventData) : void
«virtual» OnPointerDown(eventData:PointerEventData) : void
«virtual» OnPointerUp(eventData:PointerEventData) : void
«virtual» OnDrag(eventData:PointerEventData) : void
«virtual» OnEnable() : void
GetArchitecture() : IArchitecture
PlayerInputAction
CursorLocked : bool
_vector2Event : Dictionary<string, EasyEvent<Vector2>>
_boolEvent : Dictionary<string, EasyEvent<bool>>
_trigger : Dictionary<string, EasyEvent>
TriggerEvent(name:string, data:Vector2) : void
TriggerEvent(name:string, data:bool) : void
TriggerEvent(name:string) : void
AddVector2Event(name:string) : EasyEvent<Vector2>
AddBoolEvent(name:string) : EasyEvent<bool>
AddTrigger(name:string) : EasyEvent
GetTrigger(name:string) : EasyEvent
GetVector2Event(name:string) : EasyEvent<Vector2>
GetBoolEvent(name:string) : EasyEvent<bool>
«override» OnInit() : void
OnScreenButton
onClickTrigger : string
onTouchTrigger : string
onLongClickTrigger : string
onPointerUpTrigger : string
onPointerDownTrigger : string
longClickThreshold : float = 1.0f
_onPointerDownTime : float
_clicked : bool
Update() : void
«override» OnPointerClick(eventData:PointerEventData) : void
«override» OnPointerDown(eventData:PointerEventData) : void
«override» OnPointerUp(eventData:PointerEventData) : void
OnDisable() : void
OnScreenJoystick
onValueChangeTrigger : string
movementRange : float = 100
gJoyStick : GameObject
gJoyStickBackground : GameObject
idleHide : bool = true
_joyStickBackgroundStartPos : Vector2
_pointerCurrentPos : Vector2
_pointerId : int
«override» OnPointerDown(eventData:PointerEventData) : void
«override» OnDrag(eventData:PointerEventData) : void
«override» OnPointerUp(eventData:PointerEventData) : void
«override» OnEnable() : void
OnDisable() : void
OnScreenTouchpad
_pointerId : int
_pointerCurrentPos : Vector2
onValueChangeTrigger : string
«override» OnPointerDown(eventData:PointerEventData) : void
«override» OnDrag(eventData:PointerEventData) : void
«override» OnPointerUp(eventData:PointerEventData) : void
«override» OnEnable() : void
OnDisable() : void

- PlayerInputAction:
    - Container to store input events.
    - Trigger specific event by name.
    - Get event by name (used to register event callback).
- 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.

4.11 Component: OnlineRuntime/CharacterControl

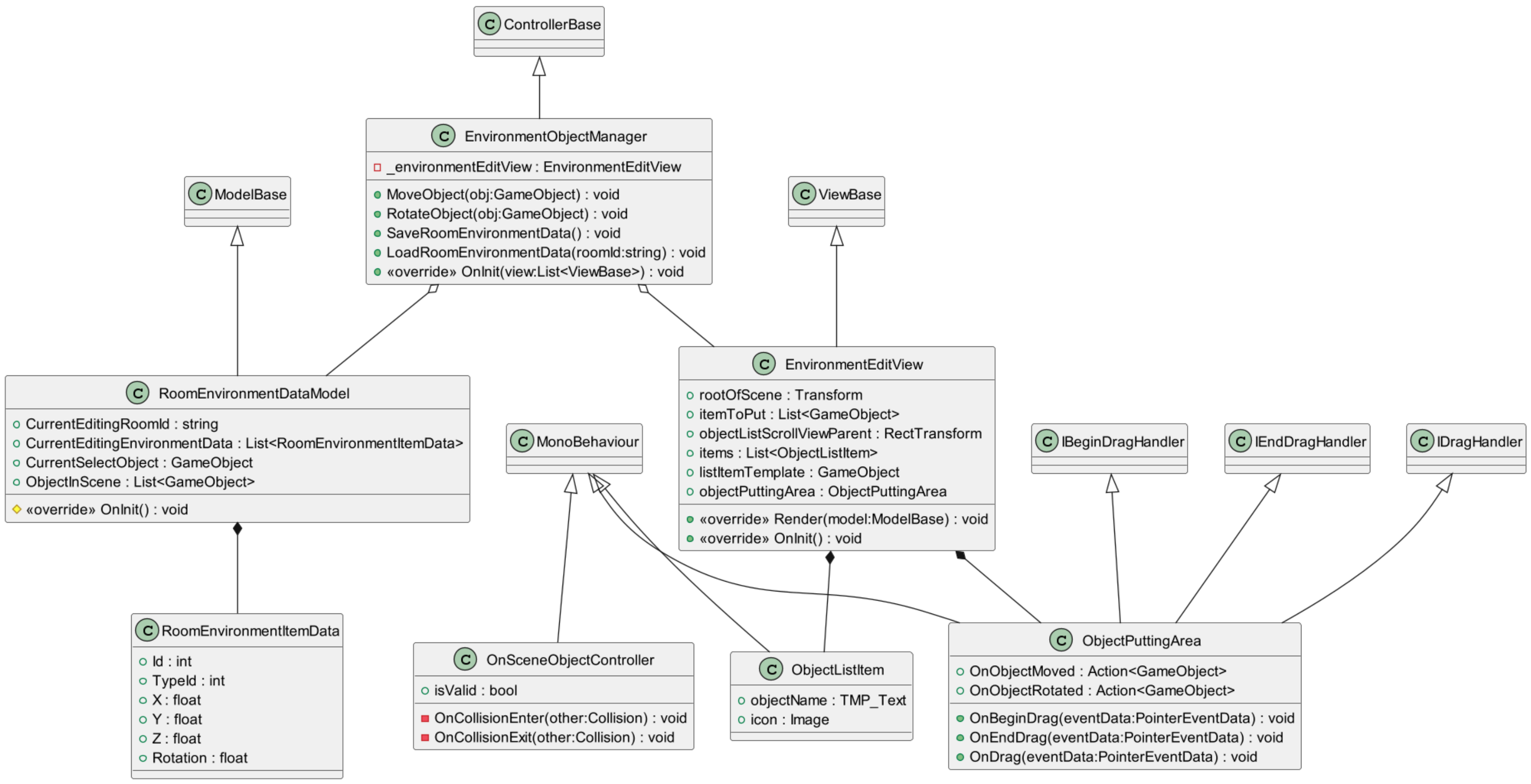
This component provides control to the character on the online room based on Unity netcode. Including: Move, look around, jump, crouch, sprint, play animation, play expression.


E EAnimation
C IController
C NetworkBehaviour
C PlayerController
Jump() : void
SetCrouch(isCrouch:bool) : void
SetSprint(isSprint:bool) : void
MoveAndRotate() : void
PlayAnimation(animationName:EAnimation) : void
GetArchitecture() : IArchitecture
C PlayerExpressionController
vrm10Instance : Vrm10Instance
SetExpressionServerRpc(v:int) : void
SetExpressionClientRpc(v:int) : void
Relax() : void
Angry() : void
Surprised() : void
Sad() : void
Fun() : void
Neutral() : void

- PlayerController:
  - Unity Network object component script provides controlling character action in the room.
  - Use Network Transform and Network Animator component of Unity Netcode to synchronize character in room).
- PlayerExpressionController:
  - Unity Network object component script provides controlling character expression in the room.
  - Use RPC to synchronize.

4.12 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.



ControllerBase
EnvironmentObjectManager
_environmentEditView : EnvironmentEditView
MoveObject(obj:GameObject) : void
RotateObject(obj:GameObject) : void
SaveRoomEnvironmentData() : void
LoadRoomEnvironmentData(roomId:string) : void
«override» OnInit(view:List<ViewBase>) : void
ModelBase
ViewBase
RoomEnvironmentDataModel
CurrentEditingRoomId : string
CurrentEditingEnvironmentData : List<RoomEnvironmentItemData>
CurrentSelectObject : GameObject
ObjectInScene : List<GameObject>
«override» OnInit() : void
EnvironmentEditView
rootOfScene : Transform
itemToPut : List<GameObject>
objectListScrollViewParent : RectTransform
items : List<ObjectListItem>
listItemTemplate : GameObject
objectPuttingArea : ObjectPuttingArea
«override» Render(model:ModelBase) : void
«override» OnInit() : void
MonoBehaviour
IBeginDragHandler
IEndDragHandler
IDragHandler
RoomEnvironmentItemData
Id : int
TypeId : int
X : float
Y : float
Z : float
Rotation : float
OnSceneObjectController
isValid : bool
OnCollisionEnter(other:Collision) : void
OnCollisionExit(other:Collision) : void
ObjectListItem
objectName : TMP_Text
icon : Image
ObjectPuttingArea
OnObjectMoved : Action<GameObject>
OnObjectRotated : Action<GameObject>
OnBeginDrag(eventData:PointerEventData) : void
OnEndDrag(eventData:PointerEventData) : void
OnDrag(eventData:PointerEventData) : void

Controllers:

- EnvironmentObjectManager:
    - Provides functions for editing objects in chat rooms.
    - Provides save/load data to/from firebase database.

Views:

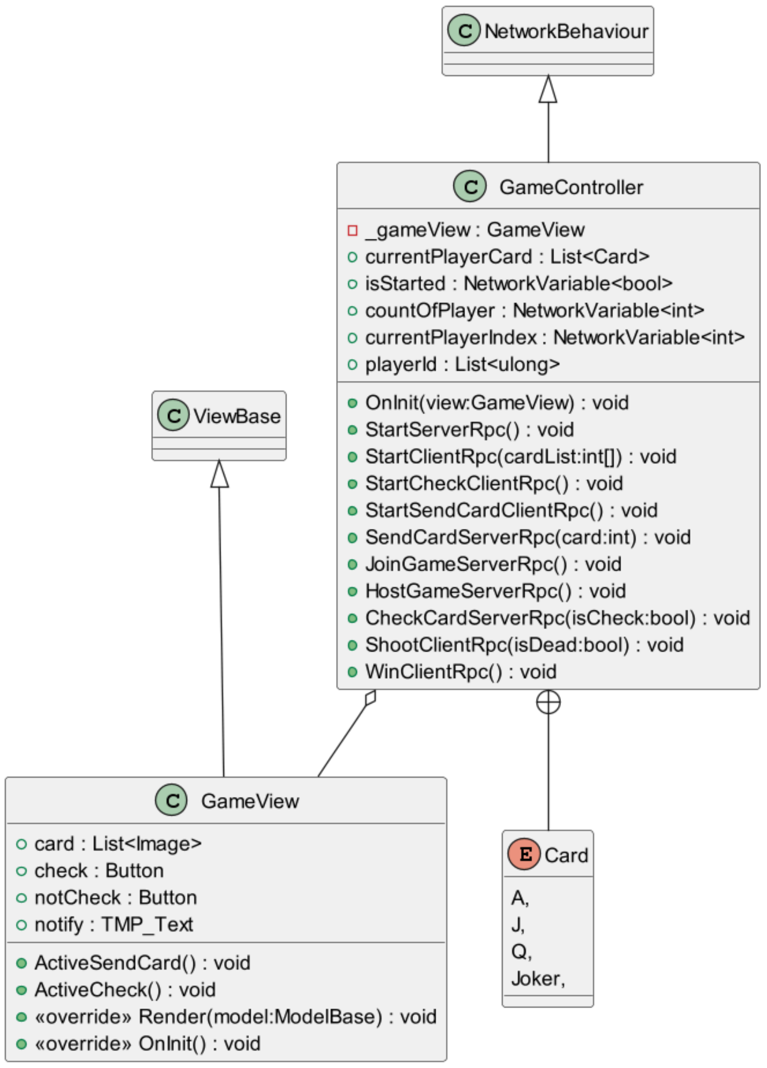
- EnvironmentEditView:
    - Provides interfaces for the host to use functionalities in EnvironmentObjectManager.
- ObjectPuttingArea:
    - Unity component script provides interface to putting/editing object on scene.

Models:

- RoomEnvironmentDataModel:
    - Store information about the objects in the chat room.

4.13 Component: OnlineRuntime/Activities/MiniGame

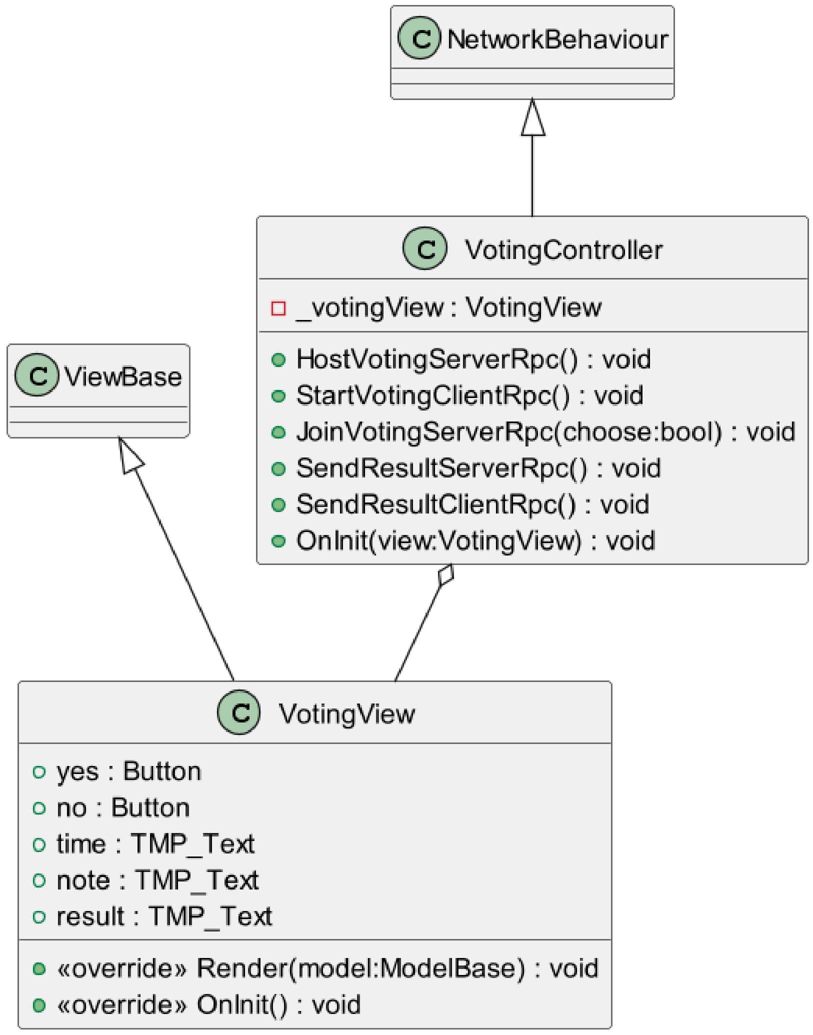
This component provides Minigame functionality based on Unity netcode. This Component provides a card game like Liar's Bar (https://store.steampowered.com/app/3097560/Liars_Bar/). This Component will be further designed in later phases of the development process.


NetworkBehaviour
GameController
_gameView : GameView
currentPlayerCard : List<Card>
isStarted : NetworkVariable<bool>
countOfPlayer : NetworkVariable<int>
currentPlayerIndex : NetworkVariable<int>
playerId : List<ulong>
OnInit(view:GameView) : void
StartServerRpc() : void
StartClientRpc(cardList:int[]) : void
StartCheckClientRpc() : void
StartSendCardClientRpc() : void
SendCardServerRpc(card:int) : void
JoinGameServerRpc() : void
HostGameServerRpc() : void
CheckCardServerRpc(isCheck:bool) : void
ShootClientRpc(isDead:bool) : void
WinClientRpc() : void
ViewBase
GameView
card : List<Image>
check : Button
notCheck : Button
notify : TMP_Text
ActiveSendCard() : void
ActiveCheck() : void
«override» Render(model:ModelBase) : void
«override» OnInit() : void
Card
A,
J,
Q,
Joker,

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

4.14 Component: OnlineRuntime/Activities/Voting

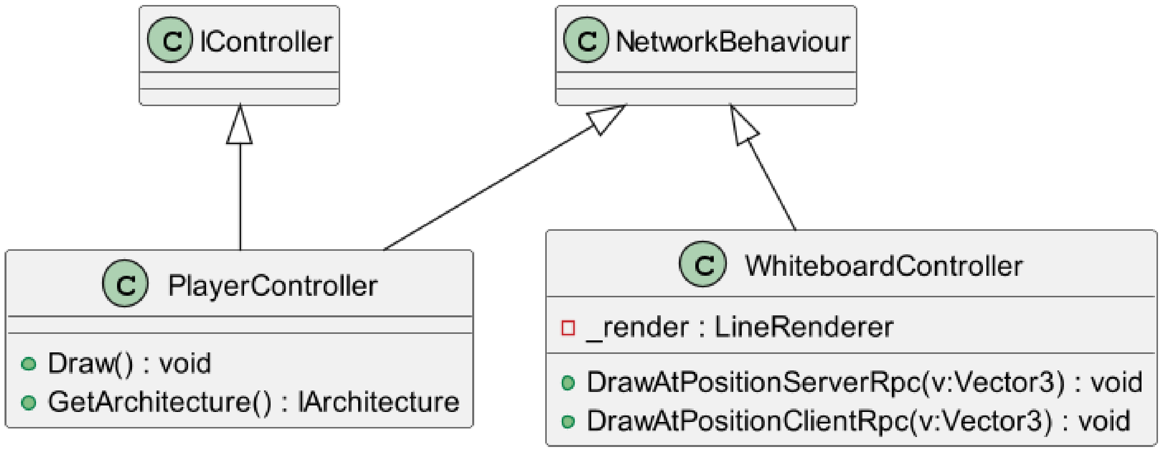
This component provides hosting voting based on Unity netcode, including: hosting a voting, voting settings (vote duration), voting, display result to all users joined in room.


NetworkBehaviour
VotingController
_votingView : VotingView
HostVotingServerRpc() : void
StartVotingClientRpc() : void
JoinVotingServerRpc(choose:bool) : void
SendResultServerRpc() : void
SendResultClientRpc() : void
OnInit(view:VotingView) : void
ViewBase
VotingView
yes : Button
no : Button
time : TMP_Text
note : TMP_Text
result : TMP_Text
«override» Render(model:ModelBase) : void
«override» OnInit() : void

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

4.15 Component: OnlineRuntime/Activities/Whiteboard

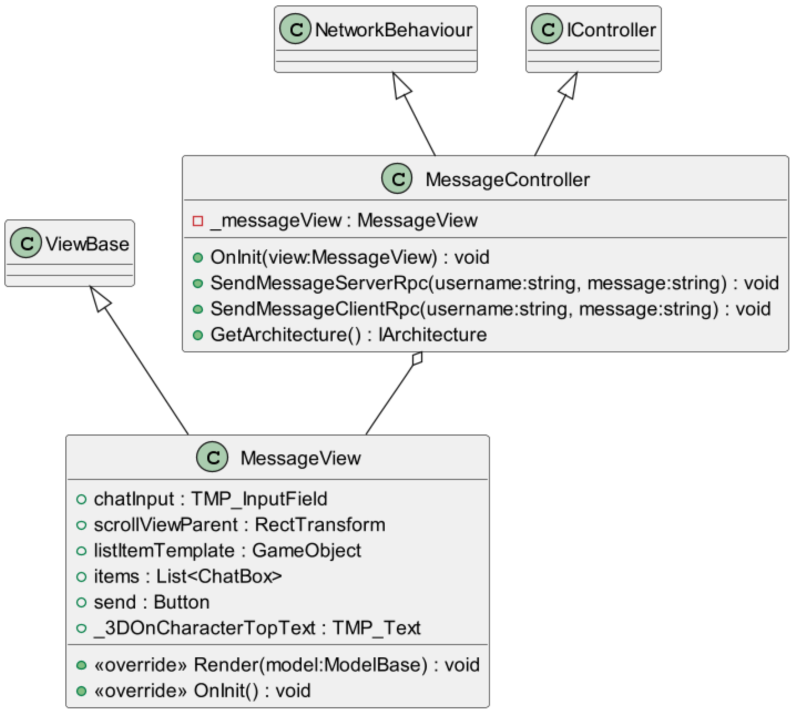
This component provides a whiteboard based on Unity netcode. Any users joined in the room can draw/erase on the whiteboard.


IController
NetworkBehaviour
PlayerController
Draw() : void
GetArchitecture() : IArchitecture
WhiteboardController
_render : LineRenderer
DrawAtPositionServerRpc(v:Vector3) : void
DrawAtPositionClientRpc(v:Vector3) : void

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

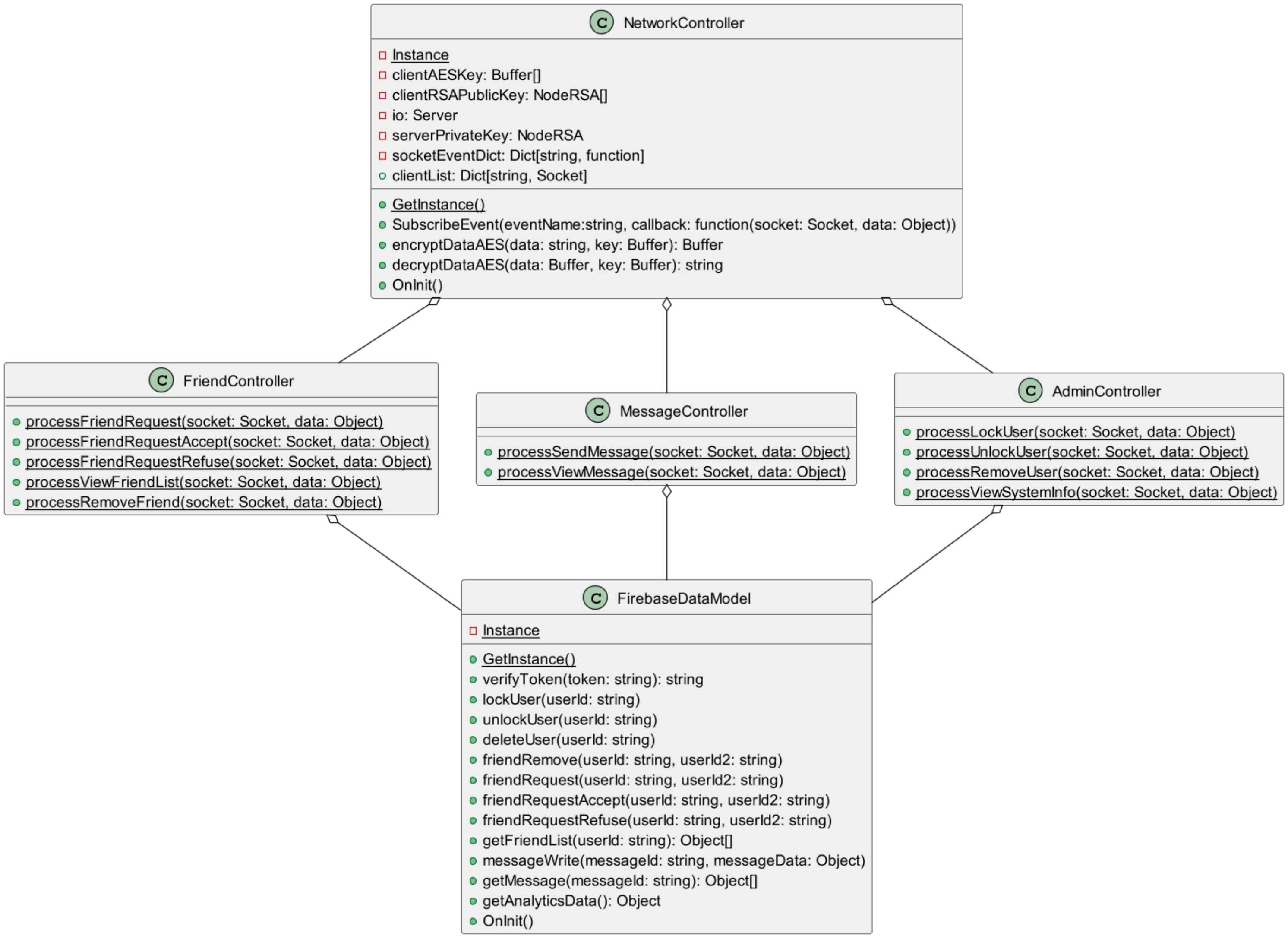
4.16 Component: OnlineRuntime/Message

This component provides message communication based on Unity netcode, including send/receive messages on chatbox. Messages of users are also displayed on the head of 3D characters who sent the message.


NetworkBehaviour
IController
MessageController
_messageView : MessageView
OnInit(view:MessageView) : void
SendMessageServerRpc(username:string, message:string) : void
SendMessageClientRpc(username:string, message:string) : void
GetArchitecture() : IArchitecture
ViewBase
MessageView
chatInput : TMP_InputField
scrollViewParent : RectTransform
listItemTemplate : GameObject
items : List<ChatBox>
send : Button
_3DOnCharacterTopText : TMP_Text
«override» Render(model:ModelBase) : void
«override» OnInit() : void

- MessageController:
    - Provides send/receive real time messages functionality in the room.
    - Use RPC to synchronize messages.
- MessageView:
    - View to display messages on chat box and display messages on top of character.

4.17 Component: Server/Network and Server/FirebaseDataModel



NetworkController
Instance
clientAESKey: Buffer[]
clientRSAPublicKey: NodeRSA[]
io: Server
serverPrivateKey: NodeRSA
socketEventDict: Dict[string, function]
clientList: Dict[string, Socket]
GetInstance()
SubscribeEvent(eventName:string, callback: function(socket: Socket, data: Object))
encryptDataAES(data: string, key: Buffer): Buffer
decryptDataAES(data: Buffer, key: Buffer): string
OnInit()
FriendController
processFriendRequest(socket: Socket, data: Object)
processFriendRequestAccept(socket: Socket, data: Object)
processFriendRequestRefuse(socket: Socket, data: Object)
processViewFriendList(socket: Socket, data: Object)
processRemoveFriend(socket: Socket, data: Object)
MessageController
processSendMessage(socket: Socket, data: Object)
processViewMessage(socket: Socket, data: Object)
AdminController
processLockUser(socket: Socket, data: Object)
processUnlockUser(socket: Socket, data: Object)
processRemoveUser(socket: Socket, data: Object)
processViewSystemInfo(socket: Socket, data: Object)
FirebaseDataModel
Instance
GetInstance()
verifyToken(token: string): string
lockUser(userId: string)
unlockUser(userId: string)
deleteUser(userId: string)
friendRemove(userId: string, userId2: string)
friendRequest(userId: string, userId2: string)
friendRequestAccept(userId: string, userId2: string)
friendRequestRefuse(userId: string, userId2: string)
getFriendList(userId: string): Object[]
messageWrite(messageId: string, messageData: Object)
getMessage(messageId: string): Object[]
getAnalyticsData(): Object
OnInit()

Server/Network:
This component is used for the server to process requests from the clients via SocketIO:
- Direct Messages.
- Friend Management.
- Admin functionality

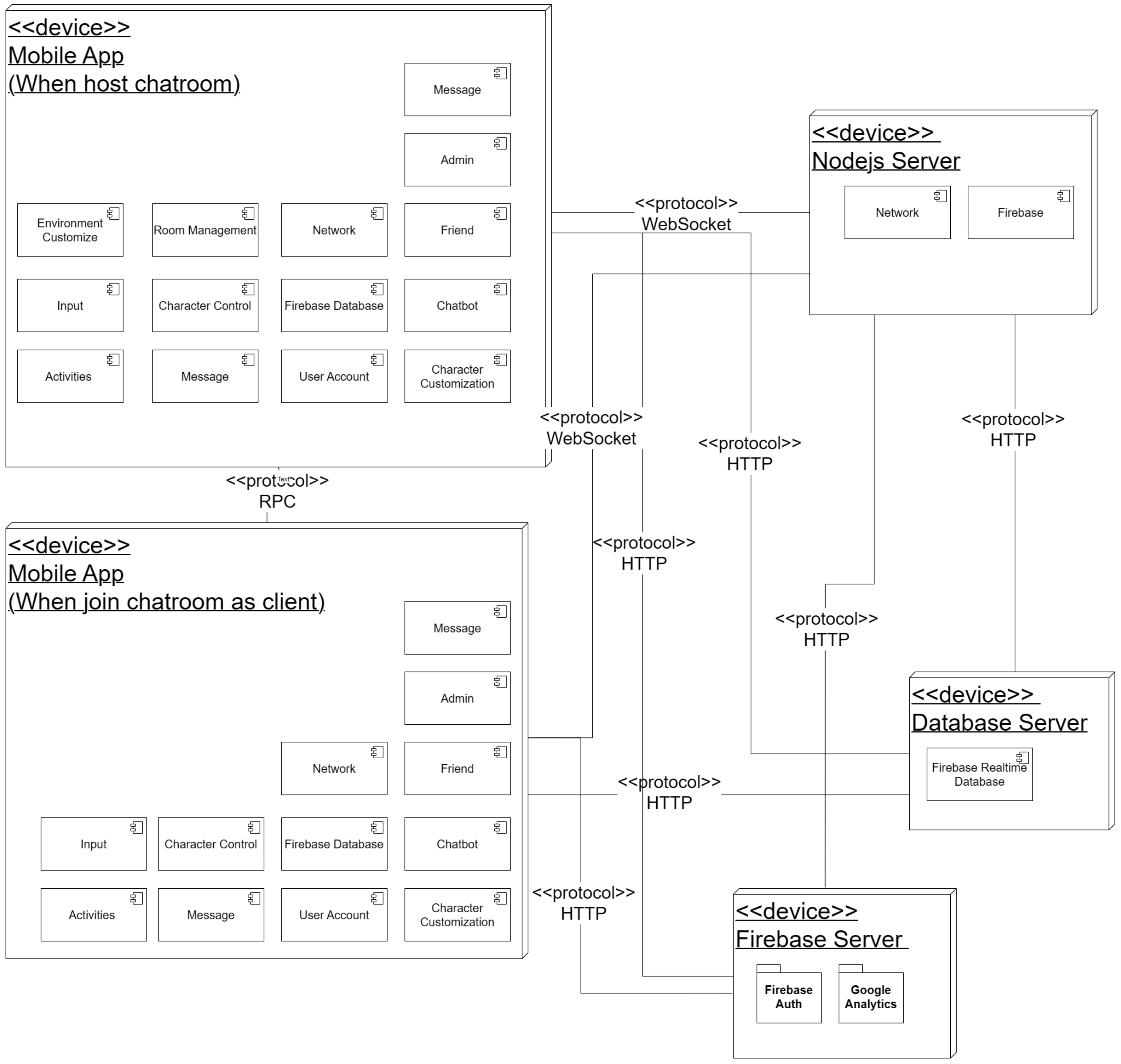
The data sent between the two sides is encrypted with RSA and AES.
Explain:
- NetworkController:
    - Management connection with clients.
    - Provides data encryption/decryption.
    - Interface to process comes in request.
- FriendController:
    - Process Friend functionality request.
- MessageController:
    - Process Message functionality request.
- AdminController:

- Process Admin functionality request.

Server/FirebaseDataModel:
This component is used to manage data in the firebase database and get data from google analytics.
Explain:

- FirebaseDataModel:
    - Manage data in the firebase database for Message, Friend, Admin functionality.
    - Get data from google analytics.

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 use RPC to communicate. 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 by using HTTP.

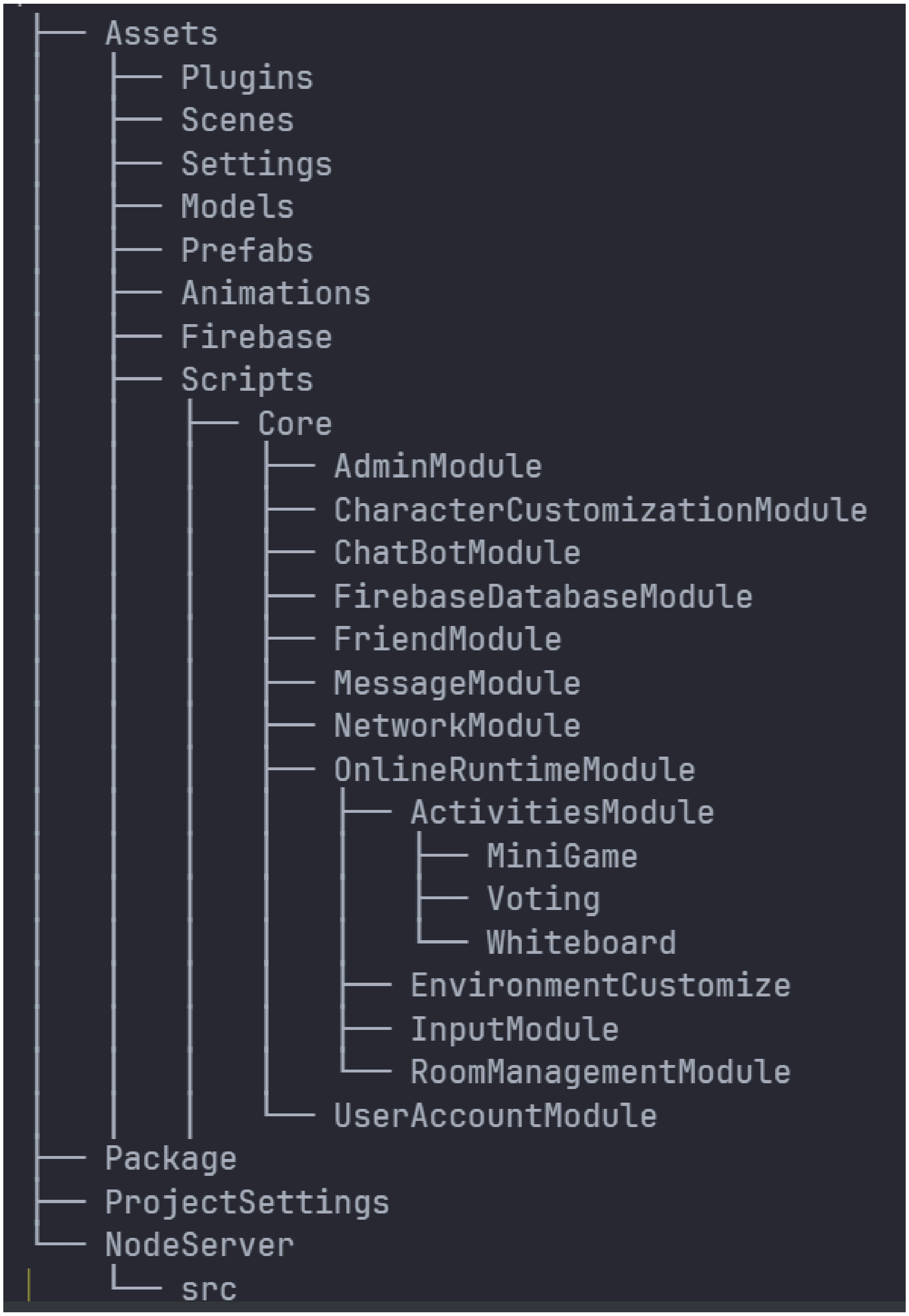
Node: Database Server (Device)

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
│   ├── Plugins
│   ├── Scenes
│   ├── Settings
│   ├── Models
│   ├── Prefabs
│   ├── Animations
│   ├── Firebase
│   ├── Scripts
│   │   ├── Core
│   │   │   ├── AdminModule
│   │   │   ├── CharacterCustomizationModule
│   │   │   ├── ChatBotModule
│   │   │   ├── FirebaseDatabaseModule
│   │   │   ├── FriendModule
│   │   │   ├── MessageModule
│   │   │   ├── NetworkModule
│   │   │   ├── OnlineRuntimeModule
│   │   │   │   ├── ActivitiesModule
│   │   │   │   │   ├── MiniGame
│   │   │   │   │   ├── Voting
│   │   │   │   │   └── Whiteboard
│   │   │   │   ├── EnvironmentCustomize
│   │   │   │   ├── InputModule
│   │   │   │   └── RoomManagementModule
│   │   │   └── UserAccountModule
├── Package
├── ProjectSettings
└── NodeServer
    └── src
```

```
├── 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 of server.
    └── src: Folder store js code for server-side.
```