COMP4920 Senior Design Project II, Spring 2021
Advisor: Mutlu Beyazıt

DRONOMATRIX: Cyber Drone Design Specifications Document

Revision Final
23.05.2021

By:
EGE ERBERK USLU,
BORA GÜZEL,
SİMGE BİNNAZ ÖZDEMİR

Revision History

Revision	Date	Explanation
1.0	18.01.2021	Initial high-level design
2.0	26.03.2021	- High level design in Section 2 has been refined - Detailed design of all classes are revised in Section 3 - Testing design in Section 4 has been reviewed and corrected for typographical errors.
2.1	05.05.2021	-Section 4 is detailed due to the growing system
Final	23.05.2021	-Final version of DSD

List of Figure

Table of Contents

1. Introduction

Cyber Drone project is a simulation program that allows those who want to use drones to experience about how to use a drone. The project is to develop 3D simulation in a unity game engine environment in C# programming language to do mainly the following:

1. Users should be able to orient the drones like up, down, right, left direction and observe the flight of drones in different situations.
2. Multiple users would like to use the software simultaneously.
3. Users would like to control and customize the drones.
4. Simulation system administrators should be able to access information about events which have occurred in the simulation on the web.

The design is based on Cyber Drone Requirements Specification Document, in version 1 in file Cyber Drone -RSD-2020-11-18-Rev-2.0.doc [1] . The design process used produce the design conforms to organizational specifications given in [2] . The notation used in this document to describe the design of Cyber Drone is mainly UML and conforms to organizational specifications.

The software architecture and overall high-level structure of Cyber Drone are given in Section 2, 3 and 4 and design details of all application functions and the testing part in terms of methods of all classes are given in Section 5 of this document.

2. Cyber Drone System Design

Cyber Drone project is a desktop application project that can be made in a Unity Game engine. The system consists of only software subsystems. Detailed information about the software subsystem design of the Cyber Drone project is given in the next section.
Also, the UML of the project is shown below generally.
The UML design of the Cyber Drone project has two main sections which are desktop part and web part. Desktop part includes two subsystems 'Unity subsystem'' and 'Unity Canvas subsystem' part which are shown in figure 1 and 2. Web part includes three subsystems 'Web Server subsystem', 'Client Server subsystem' and 'Online User Server subsystem' which are shown in figure 3, 4 and 5.

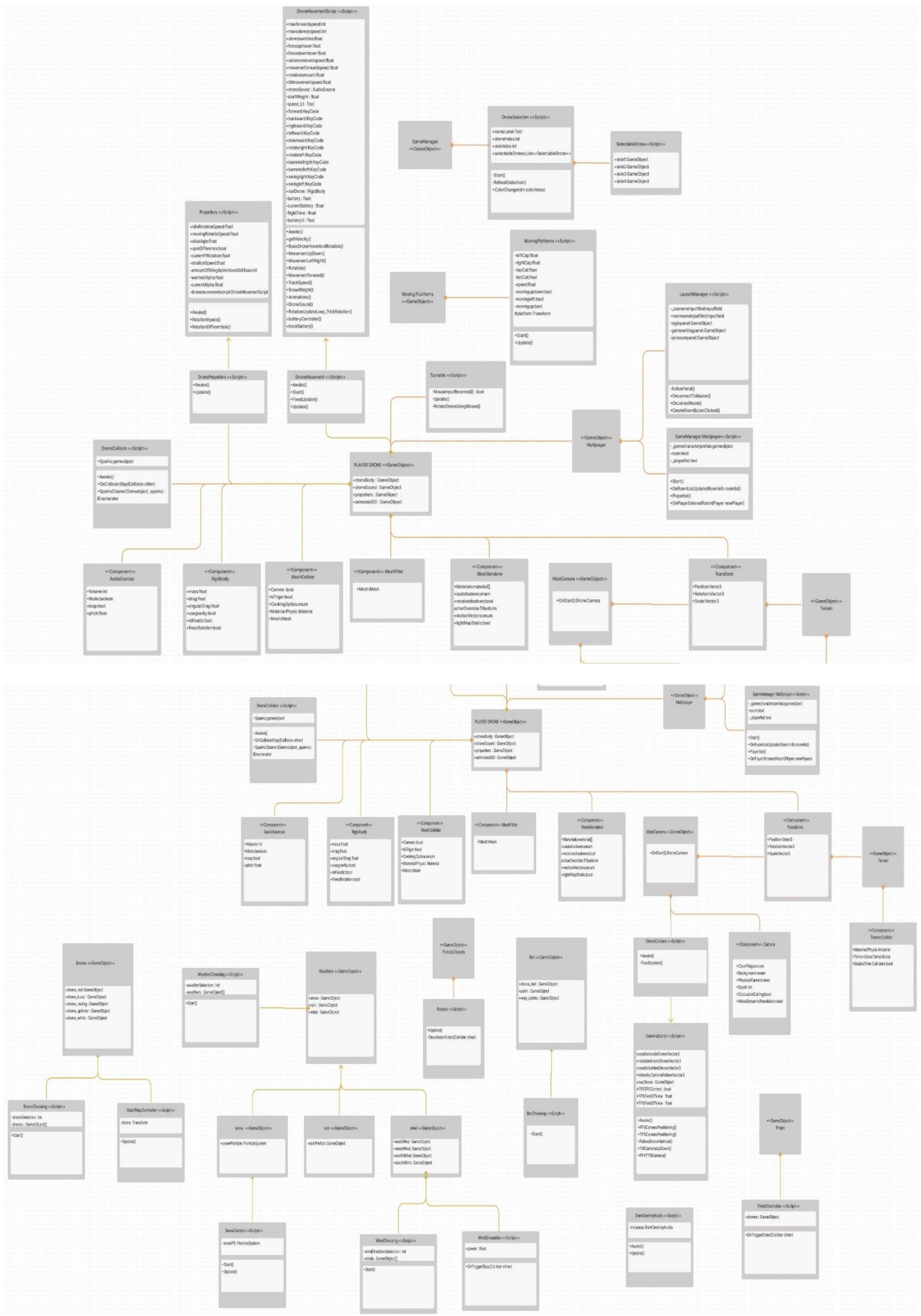

DroneMovementScript <<Script>>
DroneSelection <<Script>>
GameManager <<GameObject>>
SelectableDrone<<Script>>
Propellers <<Script>>
MovingPlatforms <<Script>>
Moving PLatforms <<GameObject>>
LaunchManager <<Script>>
DronePropellers <<Script>>
DroneMovement <<Script>>
Turnable <<Script>>
<<GameObject>> Multiplayer
GameManager Multiplayer<<Script>>
DroneCollision <<Script>>
PLAYER DRONE <<GameObject>>
<<Component>> AudioSources
<<Component>> Rigidbody
<<Component>> MeshCollider
<<Component>> MeshFilter
<<Component>> MeshRenderer
MainCamera <<GameObject>>
<<Component>> Transform
<<GameObject>> Terrain
Drones <<GameObject>>
WeatherChoosing <<Script>>
Weathers <<GameObject>>
<<GameObject>> PickUpObjects
Bot <<GameObject>>
DroneCamera <<Script>>
<<Component>> Camera
<<Component>> TerrainCollider
Rotator <<Script>>
CameraScript <<Script>>
DroneChoosing <<Script>>
OutoMapController <<Script>>
snow <<GameObject>>
rain <<GameObject>>
wind <<GameObject>>
BotChoosing <<Script>>
<<GameObject>> Props
SnowControl <<Script>>
WindChoosing <<Script>>
WindSimulation <<Script>>
DontDestroyAudio <<Script>>
FinishController <<Script>>

Figure 1:Cyber Drone Unity subsystem part.

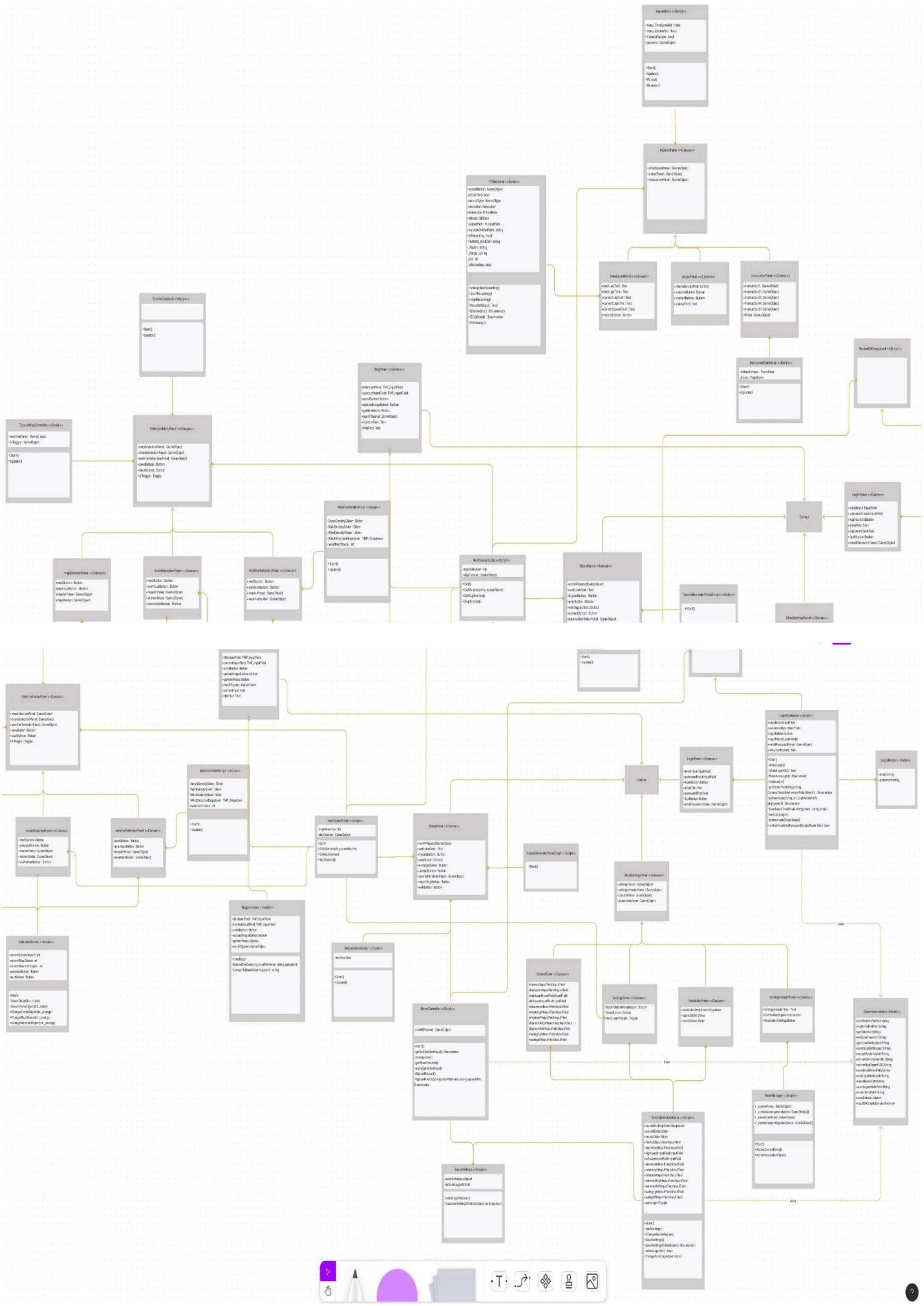


Figure 2: Cyber Drone Unity Canvas subsystem part.

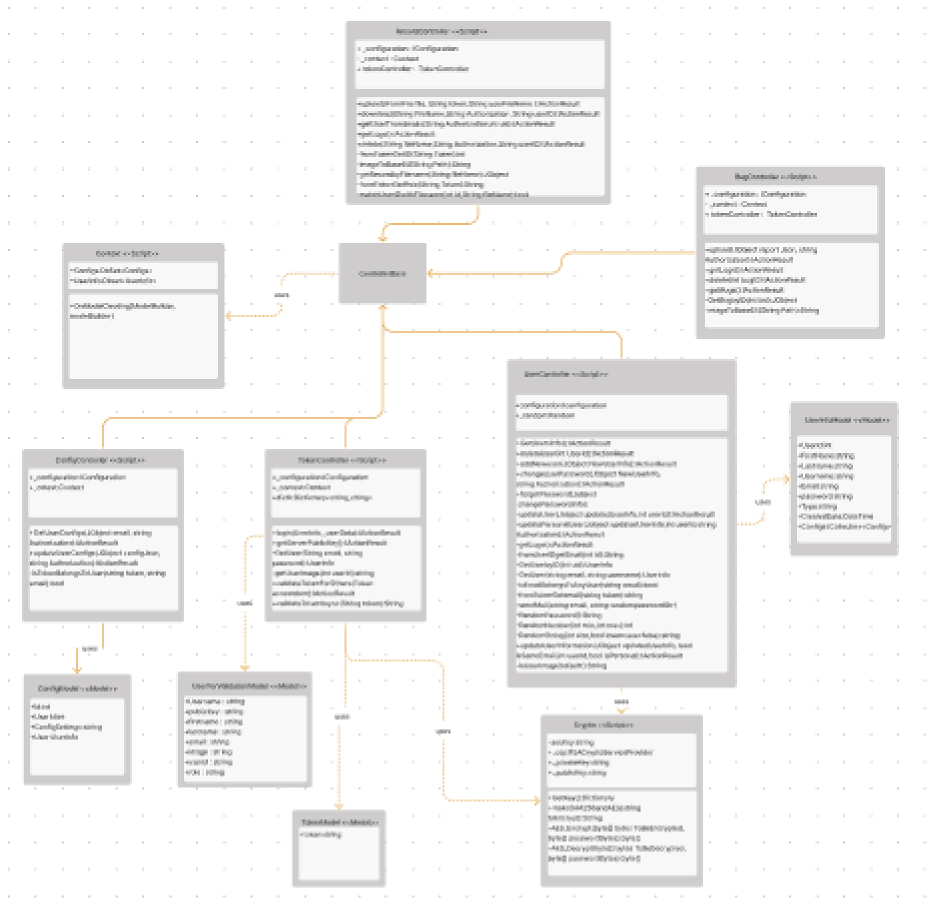


Figure 3:Cyber Drone Web Server Subsystem part.

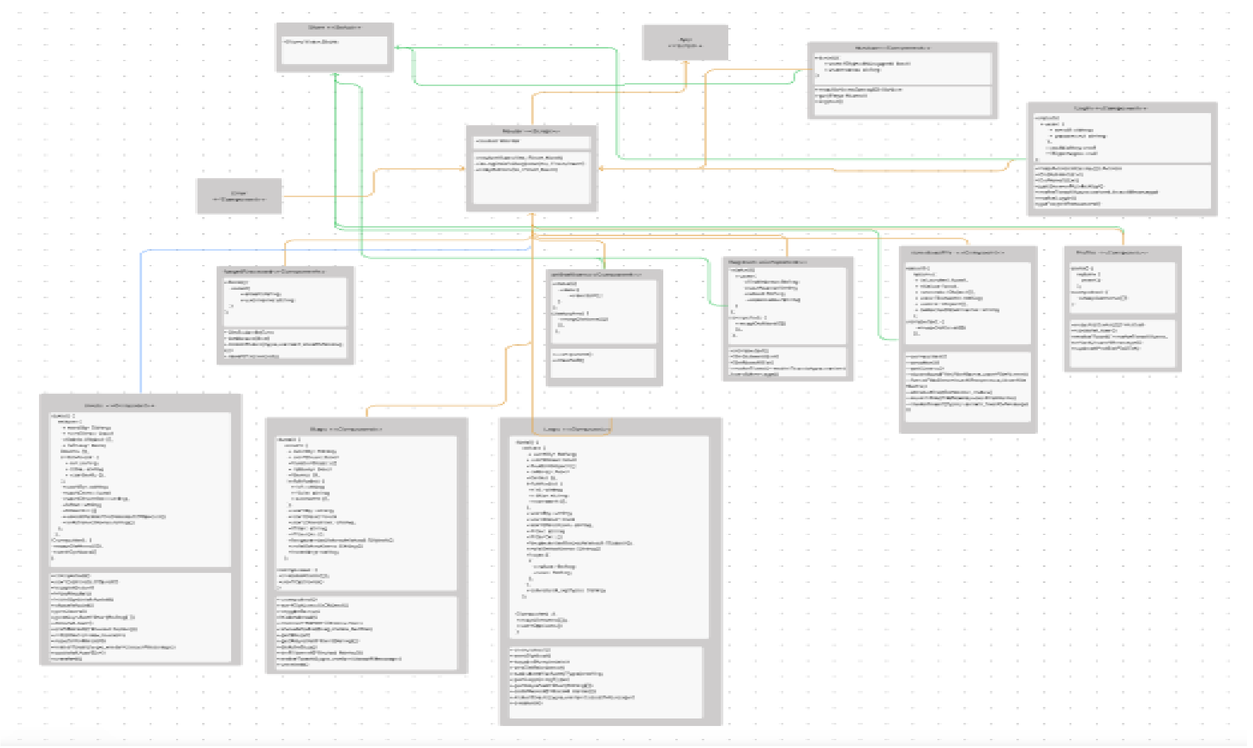


Figure 4:Client Server Subsystem part.

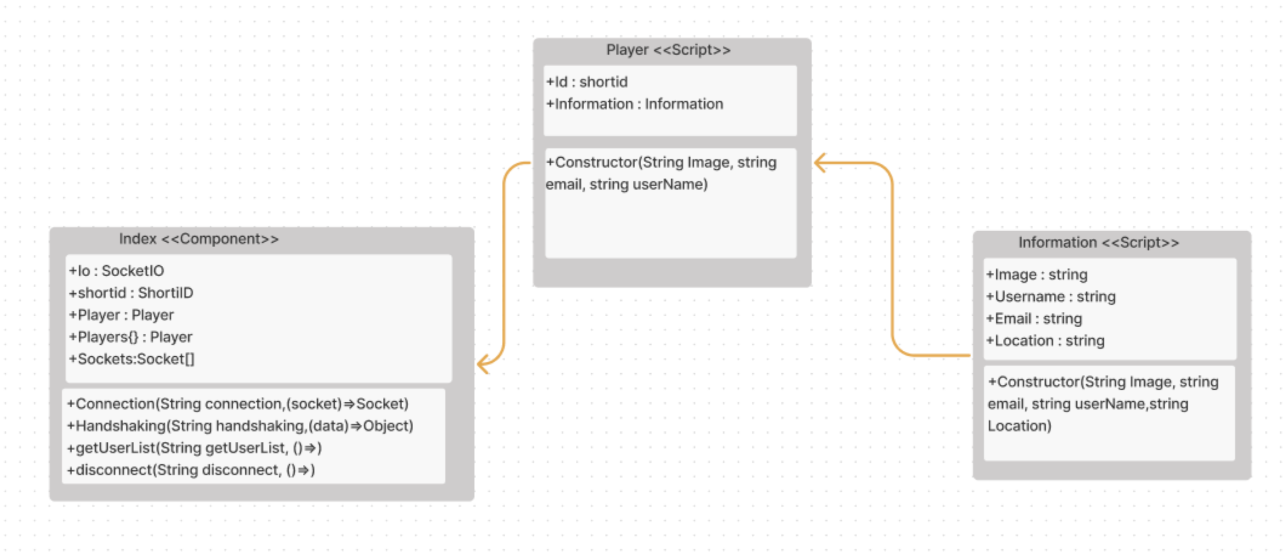

Player <<Script>>
+Id : shortid
+Information : Information
+Constructor(String Image, string email, string userName)
Index <<Component>>
+Io : SocketIO
+shortid : ShortilD
+Player : Player
+Players() : Player
+Sockets:Socket[]
+Connection(String connection,(socket)⇒Socket)
+Handshaking(String handshaking,(data)⇒Object)
+getUserList(String getUserList, ()⇒)
+disconnect(String disconnect, ()⇒)
Information <<Script>>
+Image : string
+Username : string
+Email : string
+Location : string
+Constructor(String Image, string email, string userName,string Location)

Figure 5:Online User Server Subsystem part.

3.Cyber Drone Software Subsystem Design

Cyber Drone project is a desktop application software developed with the Unity game engine. The users will enter the software with their account simultaneously. Details regarding the architecture and design of our system are covered in the following sections.

3.1. Cyber Drone Software System Architecture

Cyber drone project system architecture is client-server. Also, the project software design pattern is Model-View-Controller (MVC).
The project has a single player and multiplayer simulation program. Therefore, it has a client-server system architecture [3].

Client-server architecture consists of two parties which are a server and multiple clients. The server will provide services to multiple clients. Clients request services from the server and the server provides relevant services to those clients while the server continues to listen to client requests [4].

In the Cyber Drone simulations that have this client-server architecture, the actor plays her/his character by seeing herself/himself with own side. The client, who sees it through her/his own eyes, sends inputs such as pressing keys, mouse movement or clicks to the server. In response the server updates the state of your drone in the world and replies with a packet containing the state of your drone [5].

Whether single or multiplayer, the software design pattern is Model-View-Controller (MVC). In the figure below we showed an example MVC model according to our project. The controller is our drone movement script. The user sends a request to the controller then the controller requests information from the model. Model responds to the information to the controller. After the controller receives the information, it sends the requested data to the view. By this way, the user sees the movements of his/her drone on the screen. [6] The MVC design of the Cyber Drone project is shown in figure 4 below.

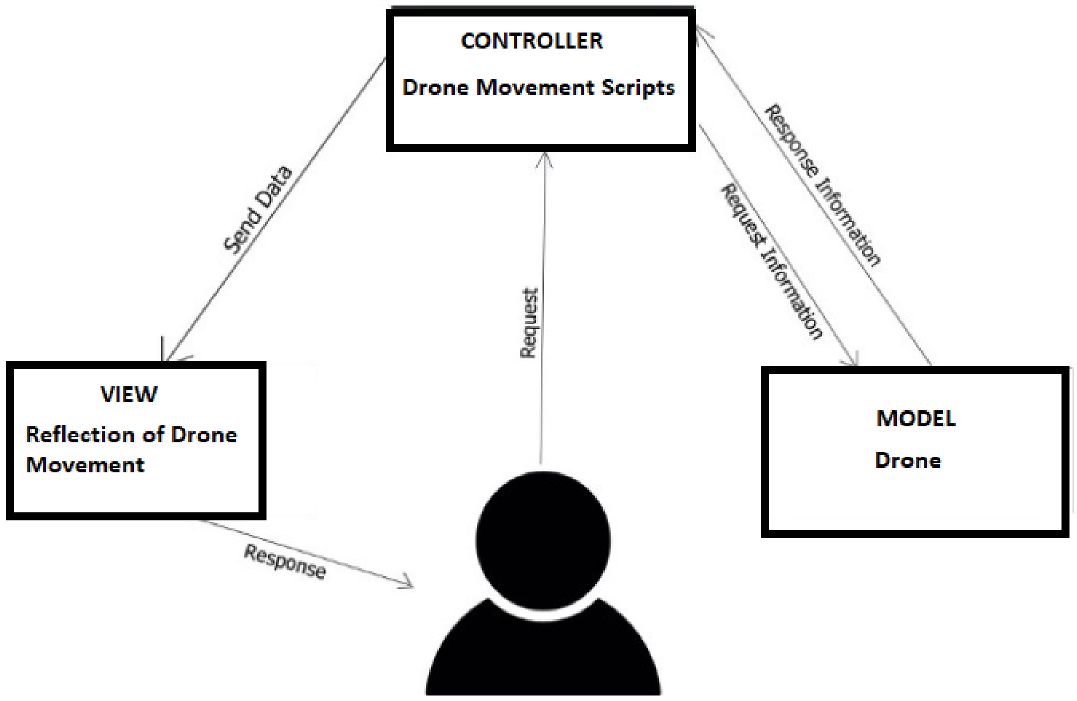


Figure 6:MVC Design.

3.2. Cyber Drone Software System Structure

Cyber Drone simulation starts with the registration of the user. Entries to the system are managed with unique numbers created for each account which is password generated systems. People who are approved to access the system can see the main menu screen. Drone and map selection is made from the main menu screen. Also, the program has a tutorial map for users to practice in the beginning. With the start of the simulation, the user accesses the drone and starts to control it on some map. Users can control all drone movements like up, down, turn with 360-degree rotating follower camera.
Developers, who are the other actors of the system, can be able to access system source code and the detailed information about the system with the web subsystem.

3.3. Cyber Drone Software System Environment

Cyber Drone desktop application works on Windows (7 and above), Linux (10.04 and above) and MacOs operating systems which are supported by Unity Game Engine version. This system will be made with Unity game engine, which is a platform developed by Unity Technology, used to develop video games and simulations. Also, Cyber Drone is written using the C# programming language. In addition, this project has a web part. Client web subsystems that contain management operations will be implemented with Vue Javascript and Vue Bootstrap Cascading Style Sheets. On the other hand, server subsystems will be implemented with C# ASP.net Core.

4. Cyber Drone Software System Detailed Design

Cyber Drone simulation systems consist of 2 subsystems which are desktop part and web part. Details of the systems and classes are shown in the headings below.

4.1. Cyber Drone Main Module/Class

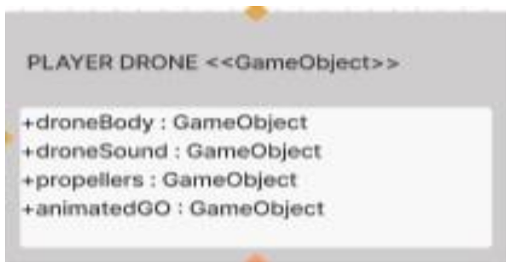
The main class of the Cyber Drone simulation is written in C# language. This main class is a common module in both developer and client. All interactions are carried out through this module.

4.2. Cyber Drone Desktop Subsystem Classes

4.2.1. Unity Subsystem Classes

4.2.1.1 PLAYER DRONE <<Game Object>> Class

This class symbolize the game object in unity game engine. The class object belonging to the UML design is shown in the **Figure 7**.

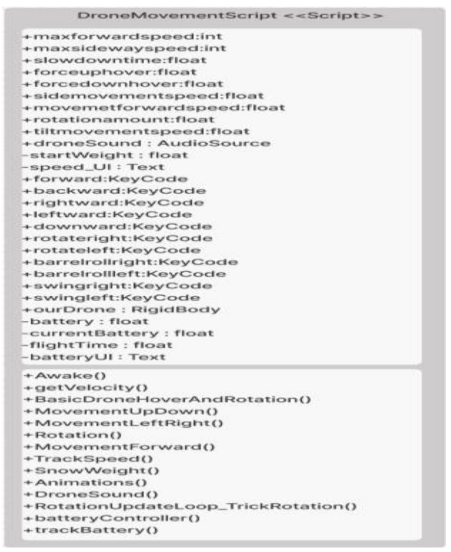


```
PLAYER DRONE <<GameObject>>
+droneBody : GameObject
+droneSound : GameObject
+propellers : GameObject
+animatedGO : GameObject
```

Figure 7 PLAYER DRONE Game Object class.

4.2.1.2 DroneMovementScript <<Script>> Class

This class symbolize the drone movement script in unity game engine. The class object belonging to the UML design is shown in the **Figure 8**.



```
DroneMovementScript <<Script>>
+maxforwardspeed:int
+maxsidewayspeed:int
+slowdowntime:float
+forceuphover:float
+forcedownhover:float
+sidemovementspeed:float
+movemetforwardspeed:float
+rotationamount:float
+tiltmovementspeed:float
+droneSound : AudioSource
-startWeight : float
-speed_UI : Text
+forward:KeyCode
+backward:KeyCode
+rightward:KeyCode
+leftward:KeyCode
+downward:KeyCode
+rotateright:KeyCode
+rotateleft:KeyCode
+barrelrollright:KeyCode
+barrelrollleft:KeyCode
+swingright:KeyCode
+swingleft:KeyCode
+ourDrone : RigidBody
-battery : float
-currentBattery : float
-flightTime : float
-batteryUI : Text
+Awake()
+getVelocity()
+BasicDroneHoverAndRotation()
+MovementUpDown()
+MovementLeftRight()
+Rotation()
+MovementForward()
+TrackSpeed()
+SnowWeight()
+Animations()
+DroneSound()
+RotationUpdateLoop_TrickRotation()
+batteryController()
+trackBattery()
```

Figure 8: Drone MovementScript Script Class

- **Class DroneMovementScript -Method Awake**

  Awake method is called either when an active Game Object that contains the script is initialized when a Scene loads, or when a previously inactive Game Object is set to active, or after a Game Object created with Object. It gets the drone component for movement operations and other operations like drone’s sound.

- **Class DroneMovementScript -Method getVelocity**

  This method gets the magnitude of drone’s velocity.

- **Class DroneMovementScript -Method BasicDroneHoverAndRotation**

  This method calculates the rotation and hover forces to apply to drone.

- **Class DroneMovementScript -Method MovementUpDown**
  This method provides the drone up lift and down move.

- **Class DroneMovementScript -Method MovementLeftRight**
  This method provides the drone left and right move.

- **Class DroneMovementScript -Method Rotation**
  This method provides the drone rotation move like right up.

- **Class DroneMovementScript -Method MovementForward**
  This method provides the forward and backward motion.

- **Class DroneMovementScript -Method TrackSpeed**
  This method finds the Current Speed text object then it assigns the current velocity to this text for showing the current speed to user.

- **Class DroneMovementScript -Method SnowWeight**
  This method calculates the drone mass according to snow that effects the drone’s mass.

- **Class DroneMovementScript -Method Animations**
  This method activates the specific animation of drone according to related stimulus.

- **Class DroneMovementScript -Method DroneSound**
  This method calculates the drone’s sound according to velocity.

- **Class DroneMovementScript -Method RotationUpdateLoop_TrickRotation**
  This method calculates the rotation and camera perspective of drone while rolling is happening.

- **Class DroneMovementScript -Method batteryController**
  This method calculates the rotation and camera perspective of drone while rolling is happening.

- **Class DroneMovementScript -Method trackBattery**
  This method calculates the rotation and camera perspective of drone while rolling is happening.

4.2.1.2 Drone Movement <<Script>> Class

This class symbolize the drone movement script in unity game engine.This class is the sub class of DroneMovementScript class. The class object belonging to the UML design is shown in the **Figure 9**.

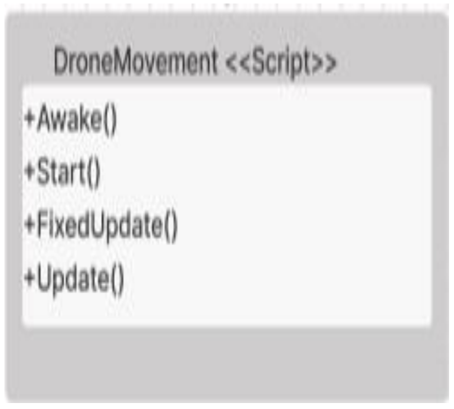


Figure 9: Drone Movement Script Class

- **Class Drone Movement -Method Awake**

  Awake method is called either when an active Game Object that contains the script is initialized when a Scene loads, or when a previously inactive Game Object is set to active, or after a Game Object created with Object. It triggers the DroneMovementsScript's awake method.

- **Class Drone Movement -Method Start**

  Start is called on the frame when a script is enabled just before any of the Update methods are called the first time. It sets the hotkeys of drone using coming object that from database about user's config. It affects the drone movement hotkeys.

- **Class Drone Movement -Method FixedUpdate**

  FixedUpdate can run once, zero, or several times per frame, depending on how many physics frames per second are set in the time settings and how fast/slow the framerate is. So this method calls SnowWeight, GetVelocity, TrackSpeed, MovementUpDown, MovementLeftRight, MovementForward, Rotation and BasicDroneHoverAndRotation methods for calculating in each frame.

- **Class Drone Movement -Method Update**

  Update is called every frame, if the MonoBehaviour is enabled. Update is the most used function to implement any kind of game script. It calls the RotationUpdateLoop-TrickRotations, Animations, DroneSound for activating the mentioned methods for calculating in each frame.

4.2.1.3. Propellers <<Script>> Class

This class includes some methods about the drone propellers movement in unity game engine. The class object belonging to the UML design is shown in the **Figure 10.**

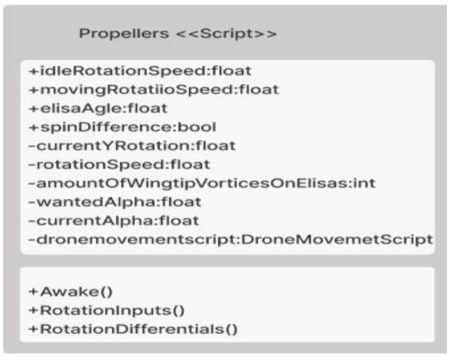


Figure 10: Propellers Script Class

- **Class Drone Movement -Method Awake**
  Awake method is called either when an active Game Object that contains the script is initialized when a Scene loads, or when a previously inactive Game Object is set to active, or after a Game Object created with Object. Instantiate is initialized.

- **Class Drone Movement -Method RotationInputs**
  This method takes the rotation inputs according to drone movement.

- **Class Drone Movement -Method RotationDifferentials**
  This method uses propeller differentials.

4.2.1.4. DronePropellers <<Script>> Class

This class is the sub class of Propellers class. This class includes some methods about the drone propellers movement in unity game engine. The class object belonging to the UML design is shown in the **Figure 11.**

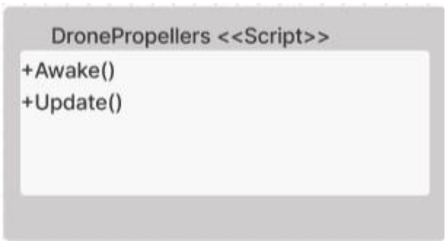


Figure 11: DronePropellers Script Class

- **Class DronePropellers -Method Awake**
  Awake method is called either when an active Game Object that contains the script is initialized when a Scene loads, or when a previously inactive Game Object is set to active, or after a Game Object created with Object. Instantiate is initialized. It triggers the Propellers class’s awake method.

- **Class DronePropellers -Method Update**

  Update is called every frame, if the MonoBehaviour is enabled. Update is the most used function to implement any kind of game script. This method calls the RotationInputs and RotationsDifferentials for managing rotation operations.

4.2.1.4. DroneCollision<<Script>> Class

This class includes method which is handling collision with drone and other obstacles. The class object belonging to the UML design is shown in the Figure **12**.

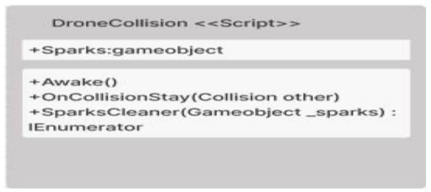

DroneCollision <<Script>>
+Sparks:gameobject
+Awake()
+OnCollisionStay(Collision other)
+SparksCleaner(Gameobject _sparks) : IEnumerator

Figure 12: DroneCollision Script Class

- **Class DroneCollision-Method Awake**

  Awake method is called either when an active Game Object that contains the script is initialized when a Scene loads, or when a previously inactive Game Object is set to active, or after a Game Object created with Object.Instantiate is initialized. This method checks the particle prefab.

- **Class DroneCollision -Method OnCollisionStay**

  It realizes that there is a collision between the drone and the object. When this event happens, it gives the collision effect to the screen which is sparks game object above figure. By this method if a collision happens between a drone and the surface the sparkle effect will show up.

- **Class DroneCollision -Method SparksCleaner**

  It destroys the sparks that exists after one second of game time.

4.2.1.5. MainCamera<<GameObject>> Class

This class represent the main camera game object. The class object belonging to the UML design is shown in the **Figure 13.**

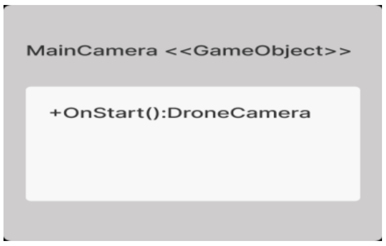


Figure 13: Main Camera Game Object class.

- **Class MainCamera-Method OnStart**

  This method is activated when the scene opens. Calls the Drone Movement script required for the drone's movements.

4.2.1.6. CameraScript<<Script>> Class

This class represent the drone follow camera script. The class object belonging to the UML design is shown in the **Figure 14**.

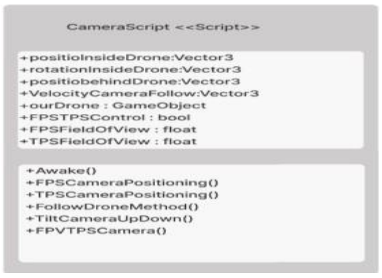


Figure 14: CameraScript Script Class

- **Class CameraScript -Method Awake**

  Awake method is called either when an active Game Object that contains the script is initialized when a Scene loads, or when a previously inactive Game Object is set to active, or after a Game Object created with Object. It sets the audio’s volume.

- **Class CameraScript -Method FPVTSCamera**

  This method checks which view format will be used. First person view or third person view.

- **Class CameraScript -Method FPSCameraPositioning**

  This method gets the fps camera component then it sets the position of first person camera to new values that comes from drone’s location to track drone.

- **Class CameraScript -Method TPSCameraPositioning**

  This method gets the fps camera component then it sets the position of third person camera to new values that comes from drone’s location to track drone. For achieving this it calls FollowDroneMethod and TiltCameraUpDown methods.

- **Class CameraScript -Method FollowDroneMethod**

  This method calculates and tracks the new position of camera according to drone speed and height.

- **Class CameraScript -Method TiltCameraUpDown**

  This method calculates tilt of camera according to previous and next positions.

4.2.1.6. DroneCamera<<Script>> Class

This class is the sub class of CameraScript script. It is responsible for calling methods that are for managing camera. The class object belonging to the UML design is shown in the **Figure 15**.

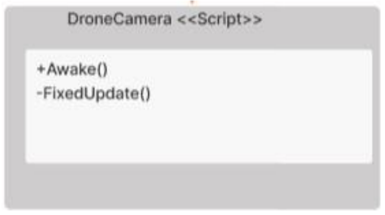


Figure 15: Drone Camera Script Class

- **Class DroneCamera-Method Awake**

  Awake method is called either when an active Game Object that contains the script is initialized when a Scene loads, or when a previously inactive Game Object is set to active, or after a Game Object created with Object.Instantiate is initialized. It triggers the awake method in CameraScript Class.

- **Class DroneCamera -Method FixedUpdate**

  FixedUpdate can run once, zero, or several times per frame, depending on how many physics frames per second are set in the time settings and how fast/slow the framerate is. This method is responsible for understanding of user’s choice about first person camera or third person camera. In addition, This method calls the FPVTPSCamera method for managing camera angle and locations.

4.2.1.7. Rigidbody<<Component>> Class

Rigidbodies are components that allow a Game Object to react to real-time physics. This includes reactions to forces and gravity, mass, drag and momentum. The class object belonging to the UML design is shown in the **Figure 16**.

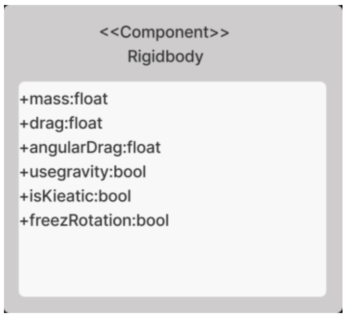


Figure 16: Rigidbody Component

4.2.1.8. Mesh Collider<<Component>> Class

The Mesh Collider takes a **Mesh Asset** and builds its Collider based on that Mesh. It is more accurate for collision detection than using primitives for complicated Meshes. Mesh Colliders that are marked as Convex can collide with other Mesh Colliders. The class object belonging to the UML design is shown in the **Figure 17**.

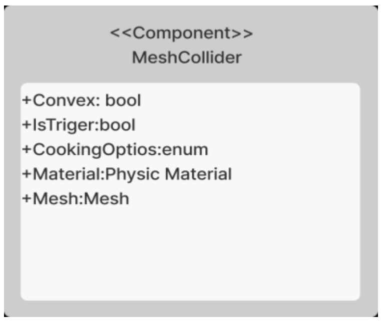


Figure 17: Mesh Collider Component

4.2.1.9. Mesh Renderer<<Component>> Class

The Mesh Renderer takes the geometry from the Mesh Filter and renders it at the position defined by the Game Object’s Transform component. The class object belonging to the UML design is shown in the **Figure 18.**

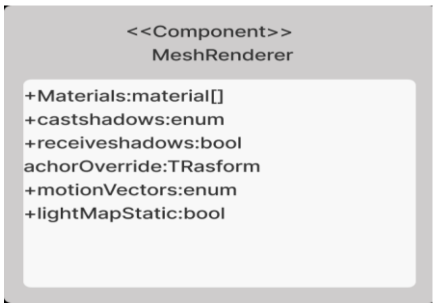


Figure 18: Mesh Renderer Component

4.2.1.10. Mesh Filter<<Component>> Class

The Mesh Filter takes a mesh from your assets and passes it to the Mesh Renderer for rendering on the screen. The class object belonging to the UML design is shown in the **Figure 19**.

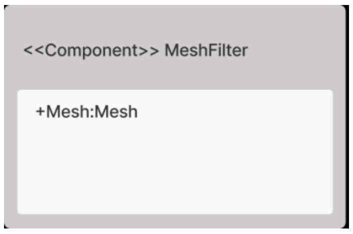


Figure 19: Mesh Filter Component

4.2.1.11. Audio Source<<Component>> Class

The Audio Source plays back an Audio Clip in the scene. The class object belonging to the UML design is shown in the **Figure 20**.

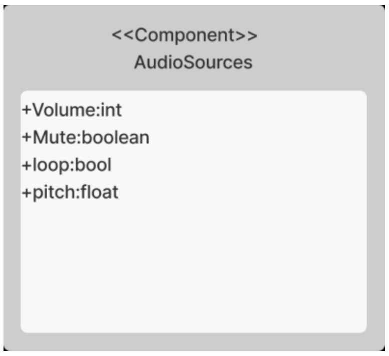


Figure 20: Audio Source Component

4.2.1.12. Transform <<Component>> Class

The Transform component determines the Position**,** Rotation, and Scale of each object in the scene. Every Game Object has a Transform. The class object belonging to the UML design is shown in the **Figure 21**.

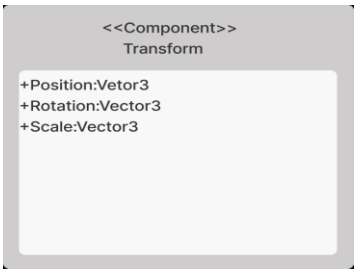


Figure 21: Transform Component

4.2.1.13. Camera <<Component>> Class

Cameras are the devices that capture and display the world to the player. By customizing and manipulating cameras, you can make the presentation of your game truly unique. The class object belonging to the UML design is shown in the **Figure 22**.

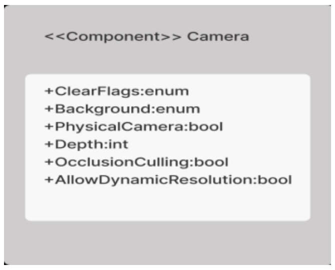


Figure 22: Camera Component

4.2.1.14. Terrain <<GameObject>> Class

Terrain game object provides the ground and obstacles belongs to terrain in scene. The class object belonging to the UML design is shown in the **Figure 23**.

Figure 23: Terrain Game Object Class

4.2.1.15. Terrain Collider <<Component>>Class

The Terrain Collider implements a collision surface with the same shape as the **Terrain** object it is attached to. The class object belonging to the UML design is shown in the **Figure 24**.

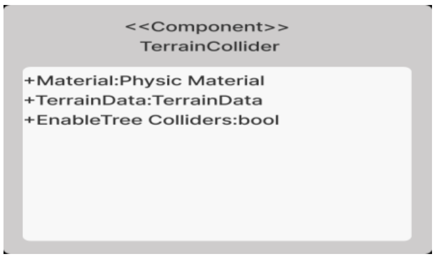


Figure 24: Terrain Collider Component

4.2.1.16. MovingPlatforms <<Script>>Class

This script provides the motion of active obstacles on map. The class object belonging to the UML design is shown in the **Figure 25**.

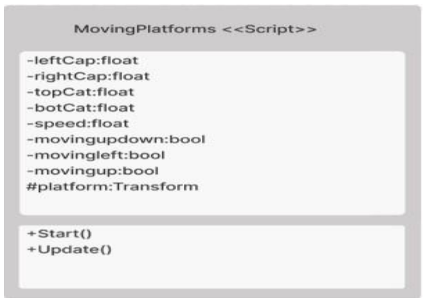


Figure 25: Moving Platforms Script Class

- **Class MovingPlatform-Method Start**

  Start is called on the frame when a script is enabled just before any of the Update methods are called the first time. By this method we are getting the transform knowledge of the object.

- **Class MovingPlatform -Method Update**

  Update is called every frame, if the MonoBehaviour is enabled. Update is the most used function to implement any kind of game script. This method performs the movement of the platforms.

4.2.1.17. DroneSelection <<Script>>Class

This script provides the drone selection methods on the customization scene. The class object belonging to the UML design is shown in the **Figure 26**.

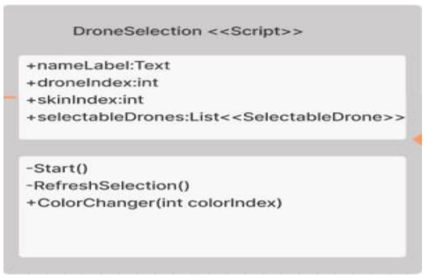

DroneSelection <<Script>>
+nameLabel:Text
+droneIndex:int
+skinIndex:int
+selectableDrones:List<<SelectableDrone>>
-Start()
-RefreshSelection()
+ColorChanger(int colorIndex)

Figure 26: Drone Selection Script Class

- **Class DroneSelection -Method Start**

  Start is called on the frame when a script is enabled just before any of the Update methods are called the first time. It calls the RefreshSelection script.

- **Class DroneSelection -Method RefreshSelection**

  This method refreshes the selection and brings the first drone model on the scene.

- **Class DroneSelection -Method ColorChanger**

  This method gets the color index and set the skin index to this value. After that, it sets the player’s skin reference to this then it calls the RefreshSelection methods to update drone.

4.2.1.18. SelectableDrone <<Script>>Class

This script provides the objects for drone skins. The class object belonging to the UML design is shown in the **Figure 27**.

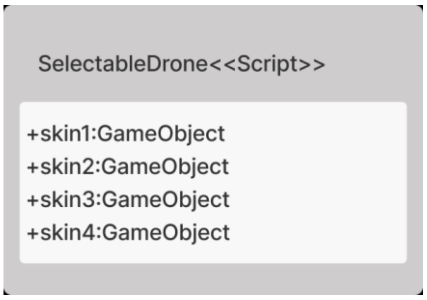


Figure 27: SelectableDrone Script Class

4.2.1.19. Turnable <<Script>>Class

This script provides the ability of rotate to the drone models on the customization scene. The class object belonging to the UML design is shown in the **Figure 28**.

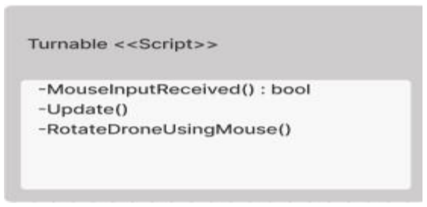


Figure 28: Turnable Script Class

- **Class Turnable -Method MouseInputReceived**
  This method provides us to get the input values of the mouse click.

- **Class Turnable -Method Update**
  Update is called every frame, if the MonoBehaviour is enabled. Update is the most used function to implement any kind of game script.

- **Class Turnable -Method RotateDronesUsingMouse**
  This method provides us to transform the drone model at the “y” axis by our mouse.

4.2.1.20. Rotator <<Script>>Class

This script provides the ability of rotate to the pick-up objects on the map and provides the player to collect these objects. The class object belonging to the UML design is shown in the **Figure 29**.

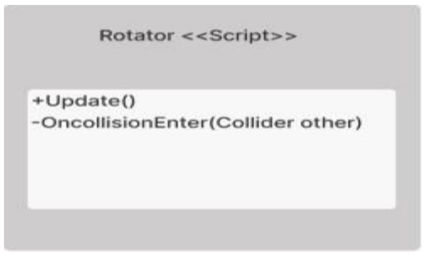


Figure 29: Rotator Script Class

- **Class Rotator -Method OnCollisionEnter**
  OnCollisionEnter is called when this collider/rigidbody has begun touching another rigidbody/collider. By using this method, the player can collect the pick-up objects. This method checks if this object collides with the "Player" tagged object and if it is, it destroys the object himself.

- **Class Rotator -Method Update**
  Update is called every frame, if the MonoBehaviour is enabled. Update is the most used function to implement any kind of game script. By using this method, the objects rotate around themselves. This method executes the process of rotation.

4.2.1.21. GameManager <<GameObject>>Class

This class symbolize the game object in unity game engine. The class object belonging to the UML design is shown in the **Figure 30**.

Figure 30: GameManager form Script Class

4.2.1.22. PickUpObjects << GameObject >>Class

This class symbolize the game object in unity game engine. It symbolizes the pickup objects on the map. The class object belonging to the UML design is shown in the **Figure 31**.

Figure 31: Moving Platform Script Class

4.2.1.23. MovingPlatforms << GameObject >>Class

This class symbolize the game object in unity game engine. It symbolizes the platforms on the map. The class object belonging to the UML design is shown in the **Figure 32**.

Moving PLatforms
<<GameObject>>

Figure 32: Moving Platform Script Class

4.2.1.24. Multiplayer<<GameObject>>Class

This class involve all actions dependent on Multiplayer game part. Users who choose the multiplayer option their game with the direction of this class. Multiplayer parts are managed by this game object. The class object belonging to the UML design is shown in the **Figure 33.**

<<GameObject>>
Multiplayer

Figure 33: Multiplayer Game Object Class

4.2.1.25 LaunchManager<<Script>>Class

This class manages the passing of scenes and panels such as login panel, game scene. It provides management according to the button events coming from the user. The class object belonging to the UML design is shown in the **Figure 34.**

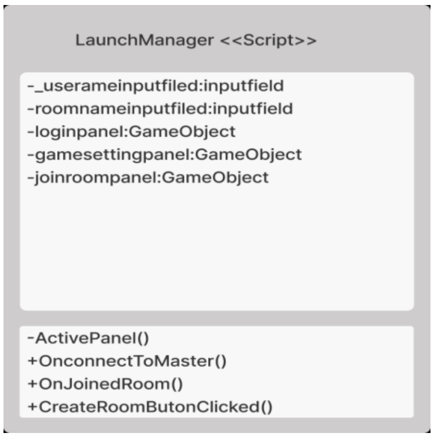

LaunchManager <<Script>>
-_userameinputfiled:inputfield
-roomnameinputfiled:inputfield
-loginpanel:GameObject
-gamesettingpanel:GameObject
-joinroompanel:GameObject
-ActivePanel()
+OnconnectToMaster()
+OnJoinedRoom()
+CreateRoomButonClicked()

Figure 34: Launch Manager Script Class

- **Class LaunchManager-Method ActivePanel**

  This method provides the passing of panels each other. It opens and closes panels according to button events from the user.

- **Class LaunchManager-Method OnConnectedToMaster**

  This method manages all connection between user, application, and Photon network server. If connection is true, then all server methods start, and lobbies opens to the user. All set up process which are provides by Photon Network is done.

- **Class LaunchManager-Method OnJoinedRoom**

  It allows users to join the room. When all photon connections are done, lobbies are open and all room options set.

- **Class LaunchManager-Method CreateroomButonClicked**

  This method set the new creating room settings. Users choose the number of player and room name with this method. Then system set the other options automatically come with the user information. After that, user can share the room name with other and they can play with each other.

4.2.1.26 GameManager Multiplayer<<Script>>Class

This class provides instantiation the game object on specific map. The class object belonging to the UML design is shown in the **Figure 35.**

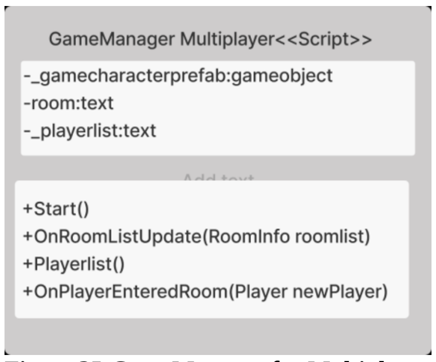

GameManager Multiplayer<<Script>>
-_gamecharacterprefab:gameobject
-room:text
-_playerlist:text
+Start()
+OnRoomListUpdate(RoomInfo roomlist)
+Playerlist()
+OnPlayerEnteredRoom(Player newPlayer)

Figure 35:GameManager for Multiplayer Part Class

- **Class GameManager-Method Start**

  This method called by unity game engine at the beginning of class creation. In here game object which is user's drone calls in instantiate method. Object exists on specific start point on the user screen.

- **Class GameManager -Method OnRoomListUpdate**

  This method provides to see existing room player list on screen. Every updated list in game, prints on the user screen panel.

- **Class GameManager-Method PlayerList**

  All connected users in room keeps in player list. When a new player entered the room, this player list updates. This method keeps these updates and send the changes to OnRoomListUpdate method.

- **Class GameManager -Method OnplayerEnteredRoom**

  When every player enters the room, this method calls. User and Photon server connection checks and determines the which room player in. These coming data player list updates with using this method.

4.2.1.27 BotChoosing<<Script>>Class

This class provides to understand user's choice about enabling or disabling AI. The class object belonging to the UML design is shown in the **Figure 36.**

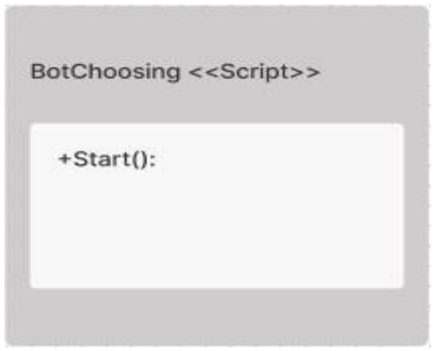


Figure 36: BotChoosing Script Class

- **Class BotChoosing -Method Start**

  This method called by unity game engine at the beginning of class creation. In here game object, which is ai drone, is created or not according user's choice of about AI bot.

4.2.1.28 SnowControl<<Script>>Class

This class provides to manage snow particle system such as changing density of snow. The class object belonging to the UML design is shown in the **Figure 37.**

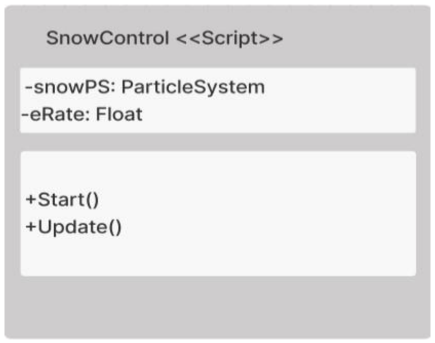


Figure 37: SnowControl Script Class

- **Class SnowControl -Method Start**

  This method is activated when the scene opens. It gets the snow particle system to manage it and assigns it to snowPS.

- **Class SnowControl -Method Update**

  This method is triggered when every frames changes. It changes the snow density according to player's preference.

4.2.1.29 WindChoosing<<Script>>Class

This class provides to manage wind's direction. The class object belonging to the UML design is shown in the **Figure 38.**

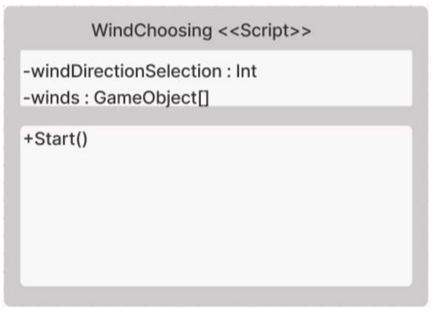


Figure 38: WindChoosing Script Class

- **Class WindChoosing -Method Start**

  This method is activated when the scene opens. It finds the wind direction from the player's preference and activate the selected wind direction and disables the other directions.

4.2.1.30 DontDestroyAudio<<Script>>Class

This class provides to not destroying audio when screen changes. The class object belonging to the UML design is shown in the **Figure 39.**

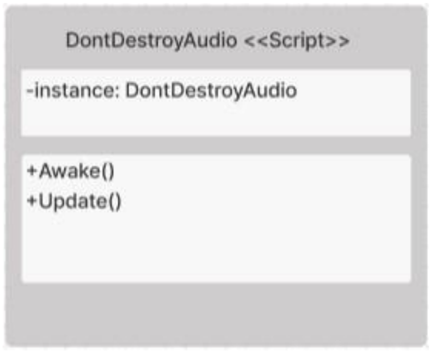


Figure 39: DontDestroyAudio Script Class

- **Class DontDestroyAudio -Method Awake**

  Awake method is called either when an active Game Object that contains the script is initialized when a Scene loads, or when a previously inactive Game Object is set to active, or after a Game Object created with Object. This method checks the audio instance to don't destroy while screen changing.

- **Class DontDestroyAudio -Method Update**

This method is triggered when every frames changes. It updates the audio's volume to desired volume which is assigned by according to user's preference.

4.2.1.31 OutofMapController<<Script>>Class

This class controls that the drone does not go beyond the boundaries of the map. The class object belonging to the UML design is shown in the **Figure 40.**

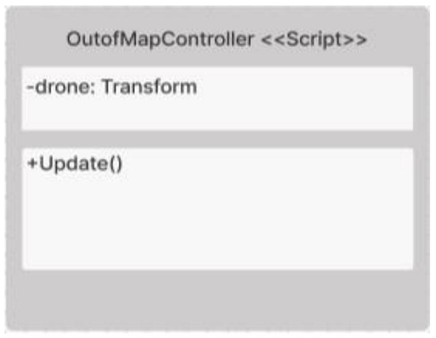


Figure 40: OutofMapController Script Class

- **Class OutofMapController -Method Update**

  This method is triggered when every frames changes. It controls that the drone does not go beyond the boundaries of the map according to selected by user. If drone is out of map, scene is loaded for teleporting user to beginning point of map.

4.2.1.32 DroneChoosing<<Script>>Class

This class controls enabling selected drone and skin of drone. The class object belonging to the UML design is shown in the **Figure 41.**

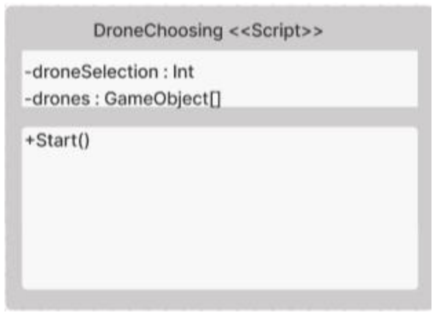


Figure 41: DroneChoosing Script Class

- **Class DroneChoosing -Method Start**

  This method is activated when the scene opens. It gets the drone selection from user's preference and activate selected drone and selected skin.

4.2.1.33 FinishController<<Script>>Class

This class controls whether a drone enters the finish area or not. The class object belonging to the UML design is shown in the **Figure 42.**

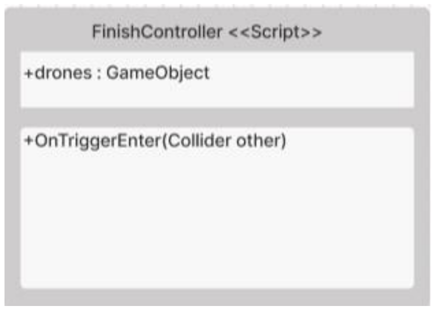


Figure 42: Finish Controller Script Class

- **Class FinishController -Method OnTriggerEnter**

  This method is triggered when a drone enters the finish area. It shows a popup to user that gives the message of ending.

4.2.1.35 WeatherChoosing<<Script>>Class

This class activates the selected weather conditions. The class object belonging to the UML design is shown in the **Figure 43.**

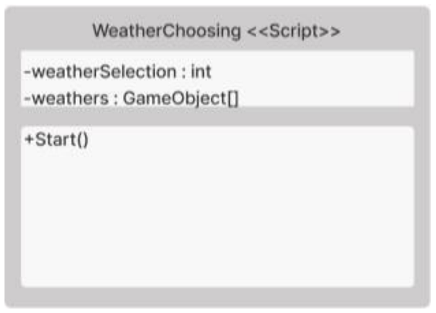


Figure 43: Weather Choosing Script Class

- **Class WeatherChoosing -Method Start**

  This method is activated when the scene opens. It gets the selected weather from player's preference and set the weatherselection to this. Then it activates the selected weather in simulation.

4.2.1.36 WindSimulation<<Script>>Class

This class understands when drone is in area that affected by wind then effects the wind to drone. The class object belonging to the UML design is shown in the **Figure 44.**

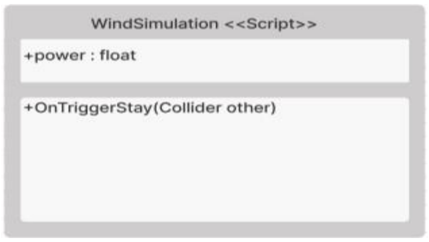


Figure 44: Wind Simulation Script Class

- **Class WeatherChoosing -Method OnTriggerStay**
  This method is triggered while drone is in area that affected by wind. It finds the drone object and it adds the force of wind to drone. Thanks to that, drone movement is effected by wind.

4.2.1.37 Drones<<GameObject>>Class

This class is responsible for storing the drones that exist in the simulation. It uses the DroneChoosing and OutofMapController for managing drone's selections and controlling boundaries of them. The class object belonging to the UML design is shown in the **Figure 45.**

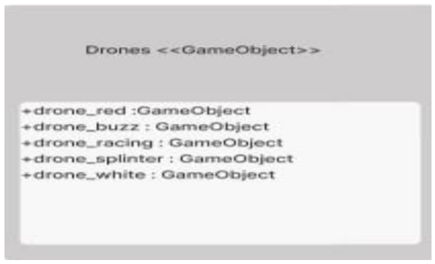

Drones <<GameObject>>
+drone_red :GameObject
+drone_buzz : GameObject
+drone_racing : GameObject
+drone_splinter : GameObject
+drone_white : GameObject

Figure 45: Drones GameObject Class

4.2.1.38 Bot<<GameObject>>Class

This class is responsible for storing bot object. In addition, it calculates the bot drone's movement path and way points. These are managed by BotChoosing script. The class object belonging to the UML design is shown in the **Figure 46.**

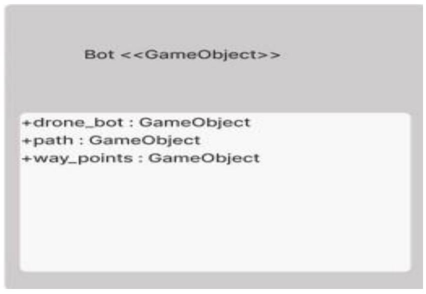


Figure 46: Bot GameObject Class

4.2.1.39 Weathers<<GameObject>>Class

This class is responsible for storing different weathers such as snow, rain and wind. The class object belonging to the UML design is shown in the **Figure 47.**

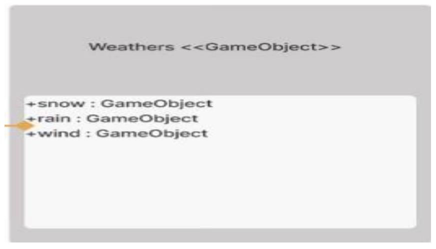


Figure 47: Weathers GameObject Class

4.2.1.40 Snow<<GameObject>>Class

This class is for storing and controlling Snow Particle System. This system is controlled by SnowControl script. The class object belonging to the UML design is shown in the **Figure 48.**

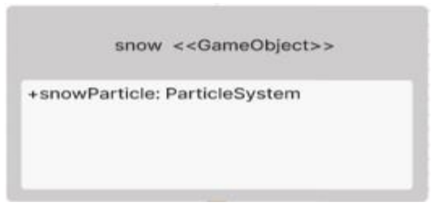


Figure 48: Snow GameObject Class

4.2.1.41 Rain<<GameObject>>Class

This class is for storing and controlling rain Particle System. The class object belonging to the UML design is shown in the **Figure 49.**

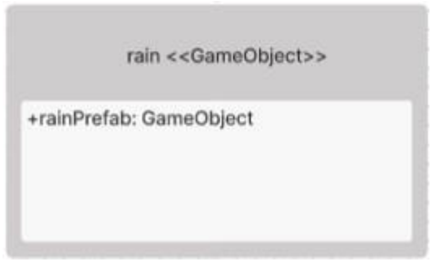


Figure 49: Rain GameObject Class

4.2.1.42 Wind<<GameObject>>Class

This class is for storing and controlling wind. It stores the wind gameobject which are the four main directions. Wind system is controlled by WindChoosing and WindSimulation scripts. The class object belonging to the UML design is shown in the **Figure 50.**

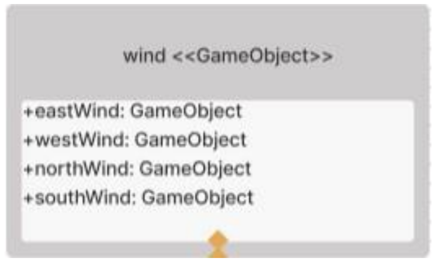


Figure 50: Wind GameObject Class

4.2.1.43 Props<<GameObject>>Class

This class symbolize the game object in unity game engine. It symbolizes the props on the map. The class object belonging to the UML design is shown in the **Figure 51**.

Figure 51: Props GameObject Class

4.2.2 Unity Canvas Part Classes

4.2.2.1 LoginController <<Script>>Class

Login Controller class is for managing the user's login procedure. This class communicates with the Login Panel to get proper values for authentication process such as email, password.
The class object belonging to the UML design is shown in the **Figure 52**.

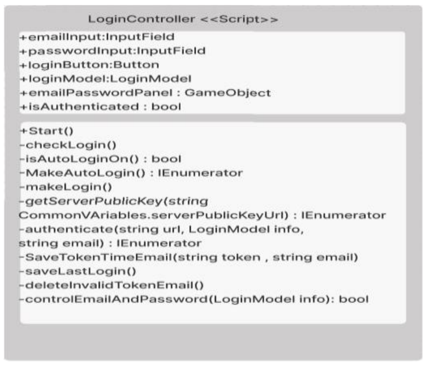

LoginController <<Script>>
+emailInput:InputField
+passwordInput:InputField
+loginButton:Button
+loginModel:LoginModel
+emailPasswordPanel : GameObject
+isAuthenticated : bool
+Start()
-checkLogin()
-isAutoLoginOn() : bool
-MakeAutoLogin() : IEnumerator
-makeLogin()
-getServerPublicKey(string CommonVAriables.serverPublicKeyUrl) : IEnumerator
-authenticate(string url, LoginModel info, string email) : IEnumerator
-SaveTokenTimeEmail(string token , string email)
-saveLastLogin()
-deleteInvalidTokenEmail()
-controlEmailAndPassword(LoginModel info): bool

Figure 52: Login Controller Script Class

- **Class LoginController-Method Start**

  This method called by unity game engine at the beginning of class creation. So checkLogin and AddListener methods that in that method are called initially. These methods are generally for starting other methods that are wanted to be start at initially. In this way, login button is assigned to call makeLogin method when a click event is thrown. On the other hand, it starts the login procedure for authentication by calling checkLogin method. At the end, it creates a new socket to connect online user server.

- **Class LoginController-Method checkLogin**

  This method controls and starts the user's authentication. It calls the isAutoLoginOn method for making the login automatically due to user's preference. According to the result of isAutoLoginOn, MakeAutoLogin will be started if result is true in boolean form.

Additionally, this procedure checks the RSA variable is assigned to any value. If not, it calls the getServerPublicKey method for getting server's public key.

- **Class LoginController-Method isAutoLoginOn**

  This method reads a proper value for making login automatically which is in xml file in application folder. This proper values name comes from the CommonVariables class which is called autoLoginNodePath. It reads the value at this path then it parses the value to bool value. After that it returns that value.

- **Class LoginController-Method MakeAutoLogin**

  This method reads the token which is saved in xml file when any login procedure happened successfully before. Token Node path comes from CommonVariables class which is saved in tokenNodePath variable for reading procedure. After the reading procedure, this method checks the token value. If token is empty string, then procedure ends itself. If not, web request is going to be started.Token is attached into the request. After that request sends to the validateTokenUrl which comes from CommonVariables class. Then response of the server will be checked. If any error occurs such as bad request and invalid token, this method calls the deleteInvalidTokenEmail procedure for removing token and email that saved in xml file. Otherwise, it reads a server response. The response is a token and user's details. These details and tokens are going to be saved in userDetails variable which is in CommonVariable class. Then it calls the saveLastLogin method for saving last login information. After that, it calls another procedure called getServerPublicKey for saving server's public key for encryption of password that will be used in other scenarios. At the end it loads a Menu Scene and it emits the new object called player into online users server by using handshaking methods in socket.

- **Class LoginController-Method getServerPublicKey**

  This method creates a Web request then sends a get request to serverPublicKeyURL. Request' url comes from CommonVariables class. Then response of the server will be checked. If any error occurs such as bad request, this method shows error log. If not, it reads the server's response which is public key information. And it updates the RSA (For Encryption) information which is stored in CommonVariables.

- **Class LoginController-Method makeLogin**

  This method gets user's email and password from UI element. Then it creates Login model, and it assigns email and password. After that, it calls the authenticate method and gives the getTokenUrl which comes from CommonVariable class and loginInfo to this method.

- **Class LoginController-Method authenticate**

  This method checks the email and password validity by calling controlEmailAndPassword method. According to that, this procedure continues the functionality or gives a popup message about wrong email and password format. Then this method converts the user's password to byteArray. Then It encrypts the user's password according to server public key. It uses a RSA encryption. This byte array will be converted to string form and assign to user's password. After that, this method creates a Web request then sends a post request to getTokenUrl which comes from CommonVariables class to server. Then response of the server will be checked. If any error occurs such as bad request and invalid credentials, this method shows error log. If not, it reads a server response. The response is a token and user's details. These details and tokens are going to be saved in userDetails variable which is in

CommonVariables class. Then it calls SaveTokenTimeEmail for saving related token, email, and time in xml file. At the end it loads a Menu Scene.

- **Class LoginController-Method SaveTokenTimeEmail**

  This method takes token value and email value from authenticate method. It writes these values in proper paths (loginInfoFilePath,tokenNodePath, lastLoginNodePath, userEmailNodePath) which are comes from CommonVariables class. This method is created for saving token and other information that mentioned above for auto login procedure.

- **Class LoginController-Method SaveLastLogin**

  This method is created for auto logged user's login time. If user has a valid token and be authenticated, then it saves the last login time to xml file. It writes to time to proper path(lastLoginNodePath) which comes from CommonVariables class.

- **Class LoginController-Method deleteInvalidTokenEmail**

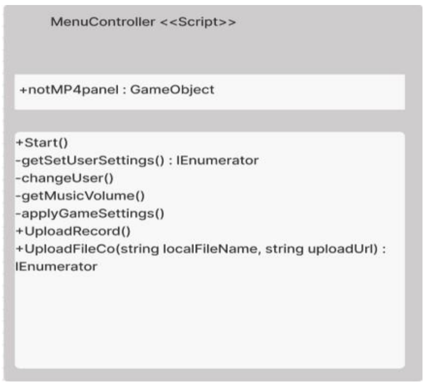
  This method is for deleting invalid token and email from loginInfoFile xml file. Due to wrong login process, it deletes the previous token from xml file.

- **Class LoginController-Method controlEmailAndPassword**

  This method is created for checking username and password fields are empty or wrong format (not email address format). It is controlled by regular expression. It returns a true if email and passwords are not empty or valid email format. Otherwise, it returns false.

4.2.2.2. MenuController <<Script>>Class

Menu Controller class is for managing the user's menu procedure. This class communicates with the Menu Panel to start proper actions for starting simulation, opening settings menu, showing welcome message, or changing the user. The class object belonging to the UML design is shown in the **Figure 53**.



```
MenuController <<Script>>

+notMP4panel : GameObject

+Start()
-getSetUserSettings() : IEnumerator
-changeUser()
-getMusicVolume()
-applyGameSettings()
+UploadRecord()
+UploadFileCo(string localFileName, string uploadUrl) :
IEnumerator
```

Figure 53: Menu Controller Script Class

- **Class MenuController-Method Start**

  This method called by unity game engine at the beginning of class creation. These methods are generally for starting other methods that are wanted to be start at initially. It checks the user is logged in from GameSettings. Initially this value assigned to false. If user is not logged in it starts the getUserSettings method. Otherwise, it calls applyGameSettings to apply Games settings according to user configuration which comes from database.

- **Class MenuController-Method getSetUserSettings**

  This method creates web request for getting user settings from database. It sends a request to getUserSettingsUrl which comes from CommonVariables class. It inserts a user email which comes from userDetails in CommonVariables class and Authorization token into header and body part of request. Then it sends a request. After that, response of the server will be checked. If any error occurs such as bad request, invalid token, email not found or authorization problem, it shows an error message. Otherwise, it sets the isUserLoggedIn in CommonVariables to true and calls the loadUserSettingsToFile methos which is in GameSettings class for saving coming user settings to file and script. Then it calls to appyGameSettings for applying these settings.

- **Class MenuController-Method applyGameSettings**

  This method sets the game settings of coming settings from database.

- **Class MenuController-Method getMusicVolume**

  This method gets the music volume value from userSettings Object which is stored in GameSettings class.

- **Class MenuController-Method uploadRecord**

  This method starts the uploading mp4 file to web server process. It gets the path of selected file and it checks it is empty or mp4 file, if result is true then starts the uploading procedure otherwise it shows and error message to user.

- **Class MenuController-Method uploadFileCo**

  This method is created for uploading mp4 file to web server. It reads the given file by using its path. If file is not read properly, system shows an error message to user. Otherwise, it adds the file to www form. Then it adds user’s token and file name to web request of body. Then it starts the uploading procedure to web server by using related api. If everything is okey, it shows a success message to user otherwise it shows an error message to user.

- **Class MenuController-Method ChangeUser**

  This method deletes the token of user that logged in previously from the loginInfo.xml file. Then clears the usersettings and details in script and set the logged in to false. On the other hand, it loads the login scene. At the end it closes the socket that connected to online user server.

4.2.2.3 SettingsMenuController <<Script>>Class

Settings Menu Controller class is for managing the user’s settings This class communicates with the Settings Panel to start proper actions for handling settings such as changing resolution, applying settings, change settings such as music volume, hotkeys of drone movement. The class object belonging to the UML design is shown in the **Figure 54**.

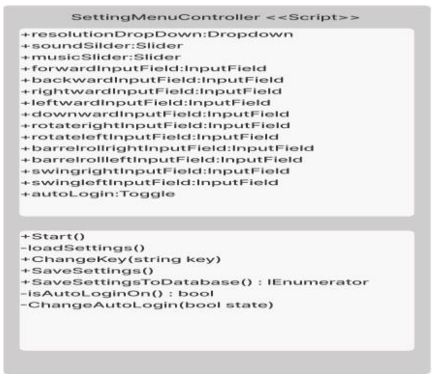

SettingMenuController <<Script>>
+resolutionDropDown:Dropdown
+soundSlider:Slider
+musicSlider:Slider
+forwardInputField:InputField
+backwardInputField:InputField
+rightwardInputField:InputField
+leftwardInputField:InputField
+downwardInputField:InputField
+rotaterightInputField:InputField
+rotateleftInputField:InputField
+barrelrollrightInputField:InputField
+barrelrollleftInputField:InputField
+swingrightInputField:InputField
+swingleftInputField:InputField
+autoLogin:Toggle
+Start()
-loadSettings()
+ChangeKey(string key)
+SaveSettings()
+SaveSettingsToDatabase() : IEnumerator
-isAutoLoginOn() : bool
-ChangeAutoLogin(bool state)

Figure 54: Settings Menu Controller Class.

- **Class SettingsMenuController-Method Start**

  This method called by unity game engine at the beginning of class creation. These methods are generally for starting other methods that are wanted to be start at initially. In this way related fields(input,dropdown) are assigned to components. At the end it calls the loadSettings method to ui for showing hotkeys, music,sound volumes and autlogin states.

- **Class SettingsMenuController-Method loadSettings**

  This method loads user settings. It assigns components to related value that comes from userSetting’s value which comes from GameSettings class.

- **Class SettingsMenuController-Method changeKey**

  This method gets the key value for understanding which key is changed. According to that, it assigns the new value to related key. It can change hotkeys, music, sounds, resolution and autologin states. It gets the value of field that changed, then it sets to related field of userSettings object which comes from GameSettings class.

- **Class SettingsMenuController-Method saveSettings**

  This method calls the loadUsetSettingsToFile methos which is in GameSettings for saving settings to json config file. Then it starts the saveSettingsToDatabase for saving current user configs to database.

- **Class SettingsMenuController-Method saveSettingsToDatabase**

  This method creates a web request to send user config to server. Request is put request. It inserts user’s email and config to request body as json form. Additionally, it inserts Authorization token to Request header. Then sends a put request to saveSettingUrl which comes from CommonVariables class. Then response of the server will be checked. If any error occurs such as bad request and invalid token and non-valid email, it shows an error message to user. Otherwise, it shows a success message to user.

- **Class SettingsMenuController-Method isAutoLoginOn**

  This method checks the autologinpath in loginInfo xml file. Then it parses the string to Boolean for returning is auto login or not.

- **Class SettingsMenuController-Method changeAutoLogin**

  This method is triggered when user wants to change autologin state, it gets the value of related toggle. Then it converts the bool value to string for saving the state of autologin to xml file. Then It changes the autologin node path to given state in autologin xml file.

4.2.2.4 GameSettings <<Script>>Class

Settings Menu Controller class is for managing the user's settings This class communicates with the Settings Panel to start proper actions for handling settings such as changing resolution, going back to Main Menu, applying settings, settings music volume, and changing hotkeys of drone movement. The class object belonging to the UML design is shown in the **Figure 55**.

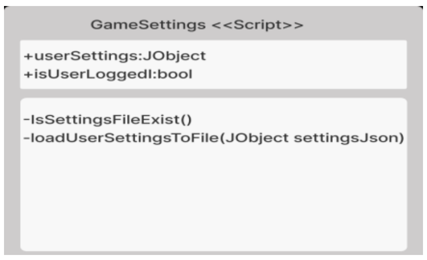


Figure 55: Game Settings Script Class.

- **Class GameSettings-Method isSettingsFileExist**

  This method checks are their related xml file is created or not. json file path comes from userSettingsFilePath which is in CommonVariable class. If related file in this path it returns a Boolean form true, otherwise it returns Boolean form false.

- **Class GameSetting-Method loadUserSettingsToFile**

  This method is for writing user settings to related json file. It tooks a user setting as a parameter. Firstly, it calls IsSettingsFileExist methos for checking is there related json file is here. If not, it creates a new config file. At the end, it writes to user settings to related json file.

4.2.2.5 Canvas <<Canvas>>Class

Canvas class is used for screen rendering. It represents the abstract space in which the UI is laid out and rendered all UI elements. The class object belonging to the UML design is shown in the **Figure 56**.

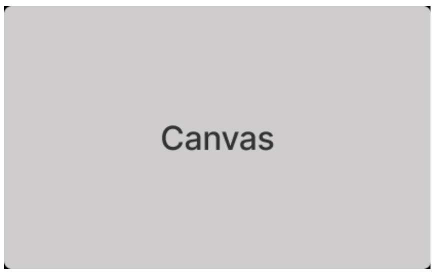
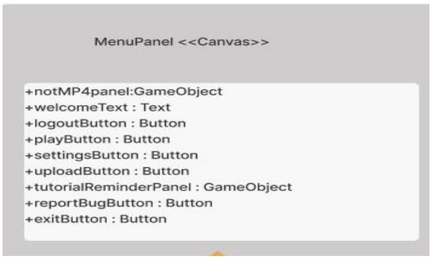


Figure 56: Canvas Class

4.2.2.6 MenuPanel <<Canvas>>Class

Menu Panel class is a class member of Panel class which extends to Canvas Class. It is responsible for showing elements which is for menu operations to user via user interface. The class object belonging to the UML design is shown in the **Figure 57**.

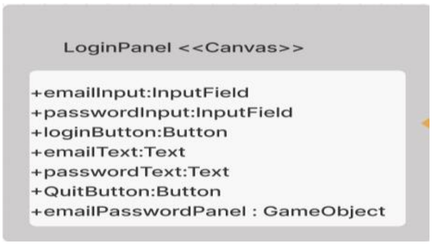

MenuPanel <<Canvas>>

+notMP4panel:GameObject
+welcomeText : Text
+logoutButton : Button
+playButton : Button
+settingsButton : Button
+uploadButton : Button
+tutorialReminderPanel : GameObject
+reportBugButton : Button
+exitButton : Button

Figure 57: Menu Panel Canvas Class

4.2.2.7 LoginPanel <<Canvas>>Class

Login Panel class is a class member of Panel class which extends to Canvas Class. It is responsible for showing elements which is for login operations to user via user interface. The class object belonging to the UML design is shown in the **Figure 58**.

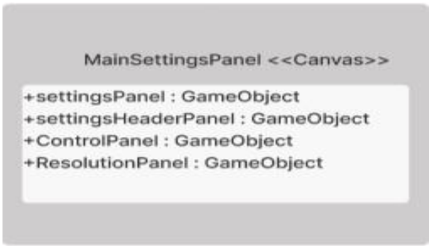

LoginPanel <<Canvas>>

+emailInput:InputField
+passwordInput:InputField
+loginButton:Button
+emailText:Text
+passwordText:Text
+QuitButton:Button
+emailPasswordPanel : GameObject

Figure 58: Login Panel Canvas Class

4.2.2.8 SettingsPanel <<Canvas>>Class

Settings Panel class is a class member of Panel class which extends to Canvas Class. It is responsible for showing sub panels which is for showing sub panels which are settings, resolution, and control panels to user via user interface. The class object belonging to the UML design is shown in the **Figure 59**.


MainSettingsPanel <<Canvas>>

+settingsPanel : GameObject
+settingsHeaderPanel : GameObject
+ControlPanel : GameObject
+ResolutionPanel : GameObject

Figure 59: Main Settings Panel Canvas Class

4.2.2.9 SettingsHeaderPanel <<Canvas>>Class

The SettingsHeader Panel class is a sub-panel of the Main Settings Panel that controls the control settings and resolution settings panels to show the relevant panel to the user via the user interface. The class object belonging to the UML design is shown in the **Figure 60**.

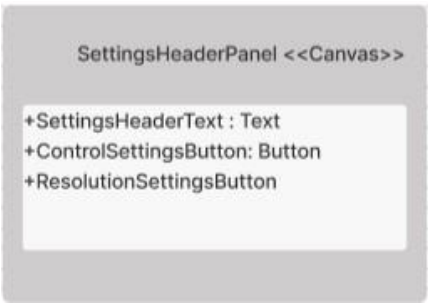

SettingsHeaderPanel <<Canvas>>
+SettingsHeaderText : Text
+ControlSettingsButton: Button
+ResolutionSettingsButton

Figure 60: Setting's Header Panel Canvas Class

4.2.2.10 ControlPanel <<Canvas>>Class

The ControlPanel Panel class is a sub-panel of the Main Settings Panel that show the hotkeys of the user via the user interface. The class object belonging to the UML design is shown in the **Figure 61**.

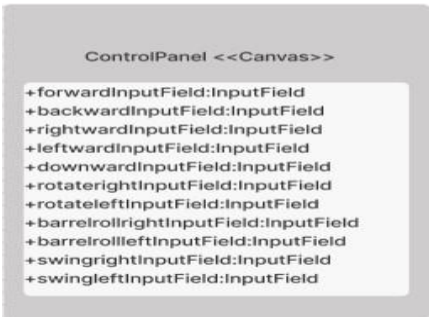

ControlPanel <<Canvas>>
+forwardInputField:InputField
+backwardInputField:InputField
+rightwardInputField:InputField
+leftwardInputField:InputField
+downwardInputField:InputField
+rotaterightInputField:InputField
+rotateleftInputField:InputField
+barrelrollrightInputField:InputField
+barrelrollleftInputField:InputField
+swingrightInputField:InputField
+swingleftInputField:InputField

Figure 61: Control Panel Canvas Class

4.2.2.10 ControlPanel <<Canvas>>Class

The ControlPanel Panel class is a sub-panel of the Main Settings Panel that show the hotkeys of the user via the user interface. The class object belonging to the UML design is shown in the **Figure 62**.

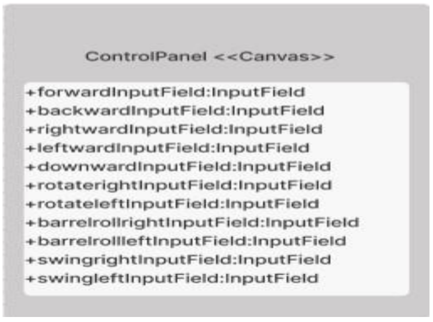

ControlPanel <<Canvas>>
+forwardInputField:InputField
+backwardInputField:InputField
+rightwardInputField:InputField
+leftwardInputField:InputField
+downwardInputField:InputField
+rotaterightInputField:InputField
+rotateleftInputField:InputField
+barrelrollrightInputField:InputField
+barrelrollleftInputField:InputField
+swingrightInputField:InputField
+swingleftInputField:InputField

Figure 62: Control Panel Canvas Class

4.2.2.11 ResolutionPanel <<Canvas>>Class

The Resolution Panel class is a sub-panel of the Main Settings Panel that show the resolution, music and sound sliders of the user via the user interface. The class object belonging to the UML design is shown in the **Figure 63**.

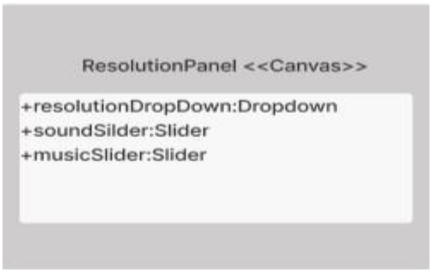


Figure 63: Resolution Panel Canvas Class

4.2.2.12 SettingsPanel <<Canvas>>Class

The SettingsPanel Panel class is a sub-panel of the Main Settings Panel that show the save and back menu buttons and the auto login toggle to the user via the user interface. The class object belonging to the UML design is shown in the **Figure 64**.

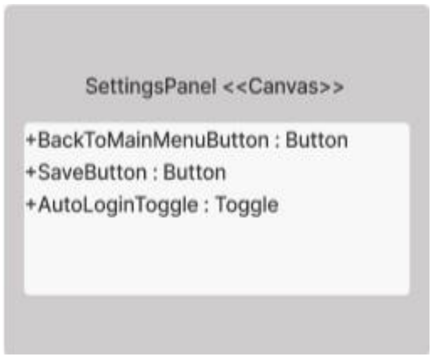


Figure 64: Settings Panel Canvas Class

4.2.2.13 LoginModel <<Model>>Class

Login Model is created for managing login inputs as a class form. It stores login information, and it is transferred to server for authentication. The class object belonging to the UML design is shown in the **Figure 65**.

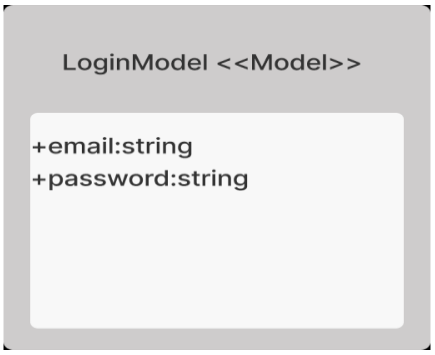


Figure 65: Login Model Class

4.2.2.14 Common Variables<<Model>>Class

Common Variables class is created for storing constant values which is rarely changes. Additionally, all other class members of simulation sub system take variables from this class in order to reach URLs, paths, etc. When any value changes in that class, the other classes are affected from it. It gives an advantage of when anything changes in class, programmer does not change other values in different classes. The class object belonging to the UML design is shown in the **Figure 66.**

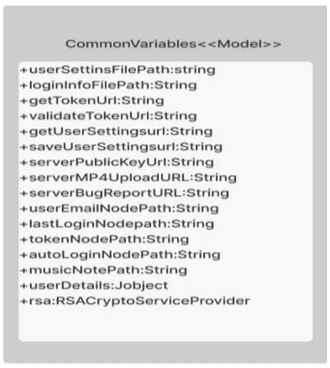

CommonVariables<<Model>>
+userSettinsFilePath:string
+loginInfoFilePath:String
+getTokenUrl:String
+validateTokenUrl:String
+getUserSettingsurl:String
+saveUserSettingsurl:String
+serverPublicKeyUrl:String
+serverMP4UploadURL:String
+serverBugReportURL:String
+userEmailNodePath:String
+lastLoginNodepath:String
+tokenNodePath:String
+autoLoginNodePath:String
+musicNotePath:String
+userDetails:Jobject
+rsa:RSACryptoServiceProvider

Figure 66: Common Variables Class

4.2.2.15 EnableDisableAI<<Script >> Class

This class is responsible for managing ui part of enabling and disabling AI bot. The class object belonging to the UML design is shown in the **Figure 67**

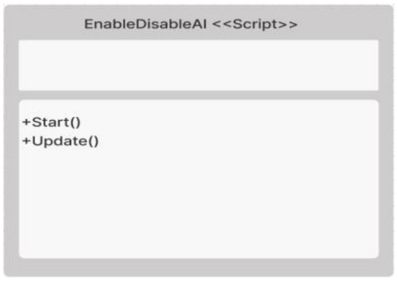


Figure 67: Enable Disable Ai control UI

- **Class EnableDisableAI -Method Start**

  This method is activated when the scene opens. It finds AIToggle component to understand enabling and disabling.

- **Class EnableDisableAI -Method Update**

  This method is triggered when every frames changes. It checks the AIToggle component's value to set player's ai preference.

4.2.2.16 TutorialMapController<<Script >> Class

This class is responsible for managing ui part of tutorial map selection. The class object belonging to the UML design is shown in the **Figure 68.**

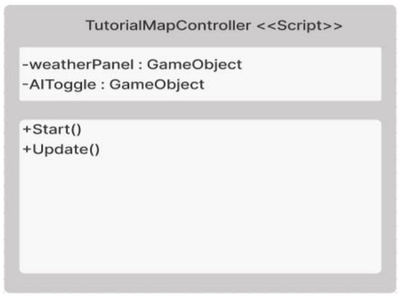


Figure 68: TutorialMap Controller Script Class

- **Class EnableDisableAI -Method Start**

  This method is activated when the scene opens. It finds weatherSelectionPanel and AIToggle components to show view of these.

- **Class EnableDisableAI -Method Update**

  This method is triggered when every frames changes. User cannot change weathers and cannot use ai in tutorials part. In that case, this method checks the user's preference that selecting tutorial map or not to showing AIToggle and Weather Panel.

4.2.2.17 PanelManager<<Script >> Class

This class is responsible for managing ui part of settings panel such as control panel and resolution panel selection.It loads the selected panel to show components to user. The class object belonging to the UML design is shown in the **Figure 69.**

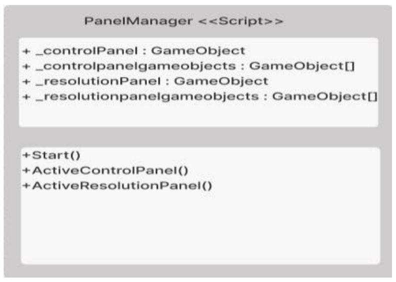

PanelManager <<Script>>
+ _controlPanel : GameObject
+ _controlpanelgameobjects : GameObject[]
+ _resolutionPanel : GameObject
+ _resolutionpanelgameobjects : GameObject[]
+Start()
+ActiveControlPanel()
+ActiveResolutionPanel()

Figure 69: PanelManager Script Class

- **Class PanelManager -Method Start**

  This method is activated when the scene opens. Initially, it calls the ActiveControlPanel to load components of control panel.

- **Class PanelManager -Method ActiveControlPanel**

  It sets the resolution panel to be inactive and it sets the control panel to be active. In addition, it activates the other game objects that in control panel.

- **Class PanelManager -Method ActiveResolutionPanel**

  It sets the control panel to be inactive and it sets the resolution panel to be active. In addition, it activates the other game objects that in resolution panel.

4.2.2.18 BugController<<Script >> Class

This class is responsible for managing ui part of bug scene. It gets the values and understands the click event of buttons for uploading file and sending bug to server. The class object belonging to the UML design is shown in the **Figure 70**

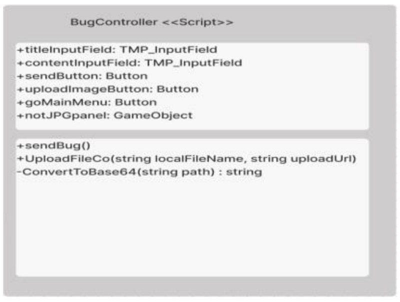

BugController <<Script>>
+titleInputField: TMP_InputField
+contentInputField: TMP_InputField
+sendButton: Button
+uploadImageButton: Button
+goMainMenu: Button
+notJPGpanel: GameObject
+sendBug()
+UploadFileCo(string localFileName, string uploadUrl)
-ConvertToBase64(string path) : string

Figure 70: Bug Controller Script Class

- **Class BugController -Method sendBug**

  This method starts the uploading procedure for sending bug report to server. It calls the UploadFileCo method to achieve this.

- **Class BugController -Method UploadFileCo**

  It loads the given file and calls the ConvertToBase64 methods to convert file to string and adds in the created post web request in json form. It gets the proper values from the ui part and add into request in json form. Then it starts the upload procedure.

- **Class BugController -Method ConvertToBase64**

  This method reads the file that selected by user and it converts the bytes of file to base64 string for valid json.

4.2.2.19 BugPanel<<Canvas >> Class

Bug Panel class is a class member of Panel class which extends to Canvas Class. It is responsible for showing elements which is for changing and updating system and bug operations to user via user interface. The class object belonging to the UML design is shown in the **Figure 71**.

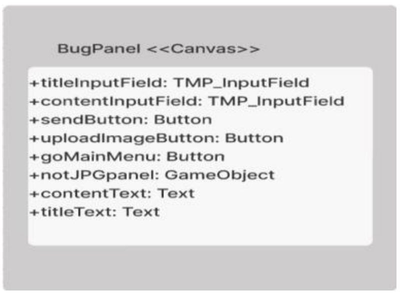


Figure 71: Bug Panel Canvas Class

4.2.2.20 FFRecorder<<Script >> Class

This class is responsible for managing screen recording. It starts and ends the recording procedure. At the ends it saves the record to records folder. The class object belonging to the UML design is shown in the **Figure 72.**

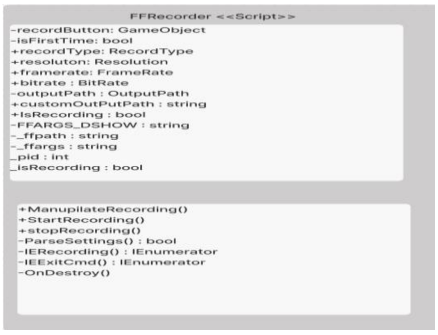


Figure 72: FFRecorder Script Class

- **Class FFRecorder -Method ManupilateRecording**

  This method is controlled by using click event of Record Button. It starts or ends the recording according to state. User can not start the new record while record has already begun. According to isRecording state, view of record icon is changed to red or white.

- **Class FFRecorder -Method StartRecording**

  This method starts the recording procedure if there is no current recording process. It kills the ffmpeg process which is started in windows in the previous time by using other programs. Otherwise, it calls the ParseSettings methods to change the recording variables. Then it calls the IERecording method.

- **Class FFRecorder -Method StopRecording**

  This method ends the recording procedure by calling IEExitCmd method. Then it saves the recorded mp4 file to desired file path.

- **Class FFRecorder -Method ParseSettings**

  This method sets the related variables for getting mp4 record such as output file path, bitrate, resolution, process arguments.

- **Class FFRecorder -Method IERecording**

  This method creates a new process to start the recording process. It gives the arguments which is stored in -ffargs. It creates a new cmd process. Then it assigns the isRecording to true.

- **Class FFRecorder -Method IEExitCmd**

  This method closes cmd process.

- **Class FFRecorder -Method OnDestroy**

  This method is triggered when the script file is closed for closing the ffmpeg process.

4.2.2.21 WeatherHolderScript<<Script >> Class

This class is responsible for controlling weather choise in ui. Also it understand the density of related weather in ui.The class object belonging to the UML design is shown in the **Figure 73**

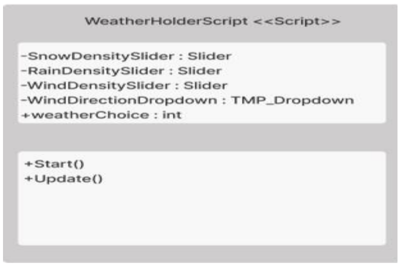

WeatherHolderScript <<Script>>
-SnowDensitySlider : Slider
-RainDensitySlider : Slider
-WindDensitySlider : Slider
-WindDirectionDropdown : TMP_Dropdown
+weatherChoice : int
+Start()
+Update()

Figure 73: Weather Holder Script Class

- **Class WeatherHolderScript -Method Start**

  This method is activated when the scene opens. It gets the selected weather choice and give the related sliders and directions to related variables to use in simulation.

- **Class WeatherHolderScript -Method Update**

  This method is triggered when every frames changes. It gets the selected weather choice and give the related sliders and directions to related player’s preference to use in simulation.

4.2.2.22 SelectionButton<<Script >> Class

This class is responsible for controlling selection of drones, weathers and maps. in ui. The class object belonging to the UML design is shown in the **Figure 74.**

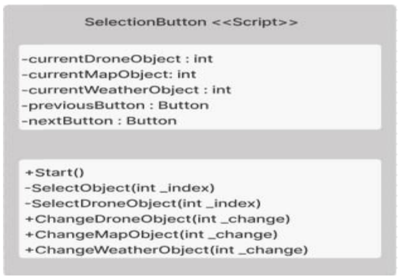

SelectionButton <<Script>>
-currentDroneObject : int
-currentMapObject: int
-currentWeatherObject : int
-previousButton : Button
-nextButton : Button
+Start()
-SelectObject(int _index)
-SelectDroneObject(int _index)
+ChangeDroneObject(int _change)
+ChangeMapObject(int _change)
+ChangeWeatherObject(int _change)

Figure 74: SelectionButton Script Class

- **Class SelectionButton -Method Start**

  This method is activated when the scene opens. It calls SelectObject and SelectDroneObject methods to start the scene with first items.

- **Class SelectionButton -Method SelectObject**

  This method is activated when the scene opens. It sets the selected object by giving index of it.

- **Class SelectionButton -Method SelectDroneObject**

  This method is activated when the scene opens. It sets the selected drone by giving index of it.

- **Class SelectionButton -Method ChangeDroneObject**

  This method is activated when the scene opens. It sets the selected drone by understanding the index of drone. Then it sets the player's drone preference to current drone. Then it calls the SelectDroneObject methods to apply it.

- **Class SelectionButton -Method ChangeMapObject**

  This method is activated when the scene opens. It sets the selected map by understanding the index of map. Then it sets the player's map preference to current map. Then it calls the SelectObject methods to apply it.

- **Class SelectionButton -Method WeatherMapObject**

  This method is activated when the scene opens. It sets the selected map by understanding the index of weather. Then it sets the player's weather preference to current weather. Then it calls the SelectObject methods to apply it.

4.2.2.23 NextSceneScript<<Script>>Class

This class controls the scenes changes like loading or closing. In addition, it manages the player's reference like clearing. The class object belonging to the UML design is shown in the **Figure 75.**

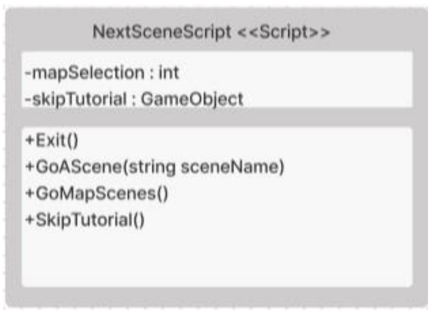

NextSceneScript <<Script>>
-mapSelection : int
-skipTutorial : GameObject
+Exit()
+GoAScene(string sceneName)
+GoMapScenes()
+SkipTutorial()

Figure 75: Next Scene Script Class

- **Class NextSceneScript -Method Exit**

  This method closes the main application.

- **Class NextSceneScript -Method GoAScene**

  This method clears the player's preferences when it is needed. It clears the specific preferences of user before loads the related scene.

- **Class NextSceneScript -Method GoMapScene**

  This method gets the preference of user about selected map and loads the selected map.

- **Class NextSceneScript -Method SkipTutorial**

  This method gets the toggle of skipping tutorial and gets the value of toggle and sets the player's preference about skipping tutorial according to that.

4.2.2.24 TutorialReminderPanelScript<<Script>>Class

This class controls skipping the tutorial or not. The class object belonging to the UML design is shown in the **Figure 76.**

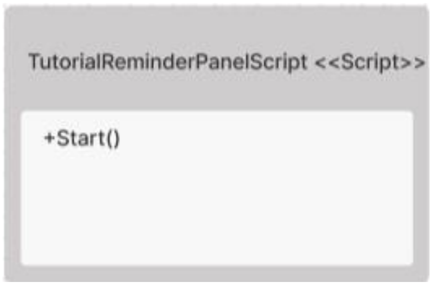


Figure 76: Tutorial Reminder Panel Script Class

- **Class TutorialReminderPanelScript -Method Start**

  This method is activated when the scene opens. According to player's preference about skipping tutorial, it activates the tutorial canvas or deactivates it.

4.2.2.25 PauseMenu<<Script>>Class

This class controls pausing and resuming of simulation. The class object belonging to the UML design is shown in the **Figure 77.**

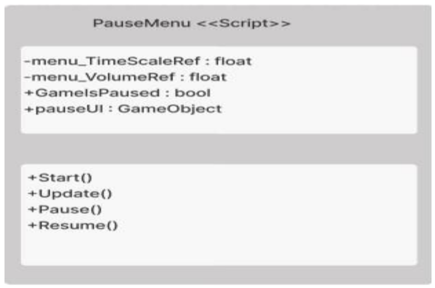


Figure 77: Pause Menu Script Class

- **Class PauseMenu -Method Start**

  This method is activated when the scene opens. It sets the timescale to given float value.

- **Class PauseMenu -Method Update**

  This method is triggered when every frames changes. It gets the pause key input and according to gameispaused variable, it pauses or resumes the simulation by calling pause or resume methods.

- **Class PauseMenu -Method Pause**

  This method pauses the game. It activates the pause menu and show the pause menu to user. Then it changes the gameispaused to false for future operations.

- **Class PauseMenu -Method Resume**

  This method resumes the game. It deactivates the pause menu and closes the pause menu.Then it changes the gameispaused to true for future operations.

4.2.2.26 InstructionController<<Script>>Class

This class controls drone position and gives the related instruction for tutorial map. The class object belonging to the UML design is shown in the **Figure 78.**

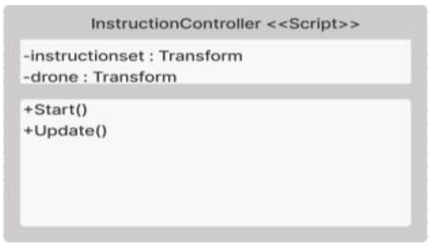


Figure 78: Instruction Controller Script Class

- **Class InstructionController -Method Start**

  This method is activated when the scene opens. It gets the transform component and sets the instructionset to that for showing instructions to user via popup.

- **Class InstructionController -Method Update**

  This method is triggered when every frames changes. It gets the player's position in each frame and shows the related instruction message when player in specific area.

4.2.2.27 SelectionMenuPanel<<Canvas>>Class

SelectionMenu Panel class is for understanding and showing of maps, drones, weathers, enabling ai and button events to user via user interface. The class object belonging to the UML design is shown in the **Figure 79**.

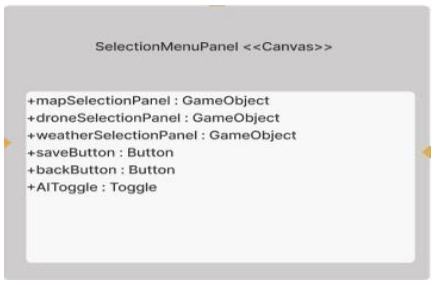

SelectionMenuPanel <<Canvas>>
+mapSelectionPanel : GameObject
+droneSelectionPanel : GameObject
+weatherSelectionPanel : GameObject
+saveButton : Button
+backButton : Button
+AIToggle : Toggle

Figure 79: Selection Menu Panel Canvas Class

4.2.2.28 MapSelectionPanel<<Canvas>>Class

MapSelection Panel is sub class if SelectionMenuPanel class.This class is responsible of showing and understanding user choice about map selection via user interface. The class object belonging to the UML design is shown in the **Figure 80**.

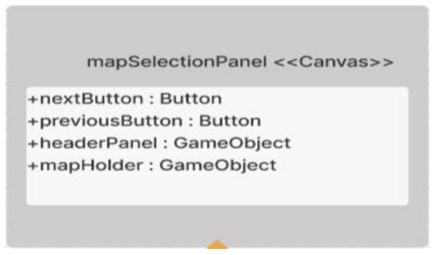


Figure 80: Map Selection Panel Canvas Class

4.2.2.29 DroneSelectionPanel<<Canvas>>Class

DroneSelection Panel is sub class if SelectionMenuPanel class. This class is responsible of showing and understanding user choice about drone selection via user interface. The class object belonging to the UML design is shown in the **Figure 81**.

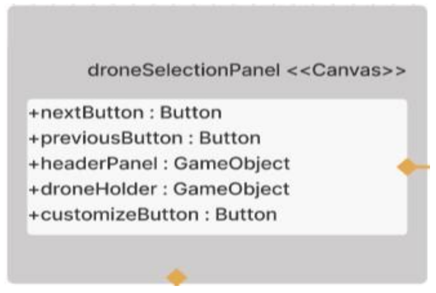

droneSelectionPanel <<Canvas>>
+nextButton : Button
+previousButton : Button
+headerPanel : GameObject
+droneHolder : GameObject
+customizeButton : Button

Figure 81: Drone Selection Panel Canvas Class

4.2.2.30 WeatherSelectionPanel<<Canvas>>Class

WeatherSelection Panel is sub class if SelectionMenuPanel class.This class is responsible of showing and understanding user choice about weather selection via user interface. The class object belonging to the UML design is shown in the **Figure 82**.

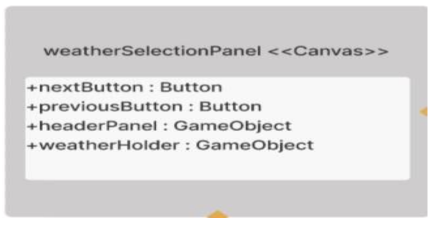

weatherSelectionPanel <<Canvas>>
+nextButton : Button
+previousButton : Button
+headerPanel : GameObject
+weatherHolder : GameObject

Figure 82: Weather Selection Panel Canvas Class

4.2.2.31 InGamePanel<<Canvas>>Class

InGame Panel class is for understanding and showing of components which are in the simulation to user via user interface. The class object belonging to the UML design is shown in the **Figure 83**.

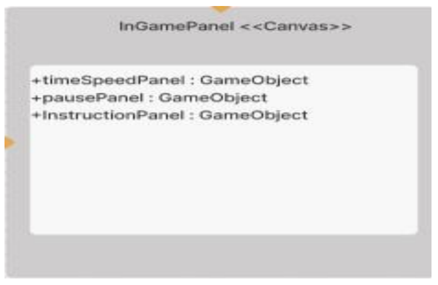


Figure 83: InGame Panel Canvas Class

4.2.2.32 timeSpeedPanel<<Canvas>>Class

timeSpeed Panel is sub class if InGamePanel class. This class is responsible for showing time, speed and lap times to user via user interface. The class object belonging to the UML design is shown in the **Figure 84**.

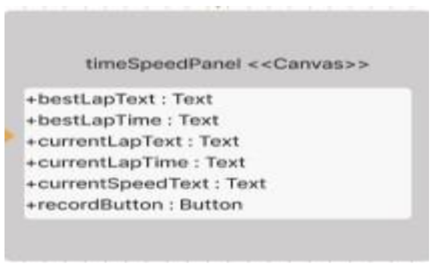


Figure 84: timeSpeed Panel Canvas Class

4.2.2.33 pausePanel<<Canvas>>Class

pause Panel is sub class if InGamePanel class. This class is responsible for showing pause text and understanding the click events of related buttons which are in pause panel to user via user interface. The class object belonging to the UML design is shown in the **Figure 85**.

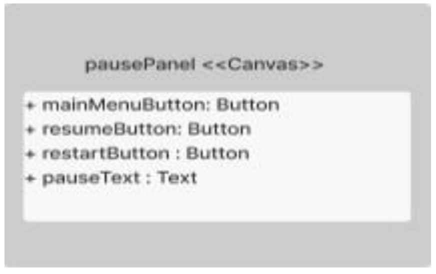
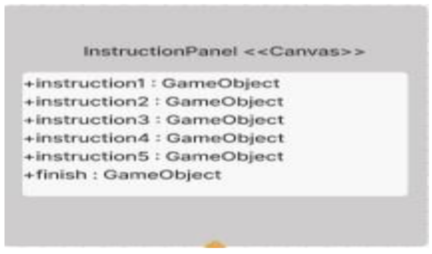


Figure 85: Pause Panel Canvas Class

4.2.2.34 InstructionPanel<<Canvas>>Class

Instruction Panel is sub class if InGamePanel class. This class is responsible for showing instructions to user via user interface. The class object belonging to the UML design is shown in the **Figure 86**.



InstructionPanel <<Canvas>>
+instruction1 : GameObject
+instruction2 : GameObject
+instruction3 : GameObject
+instruction4 : GameObject
+instruction5 : GameObject
+finish : GameObject

Figure 86: Instruction Panel Canvas Class

4.2.2.35 WelcomeTextScript<<Script>>Class

This class is responsible for showing username of logged in server. The class object belonging to the UML design is shown in the **Figure 87.**

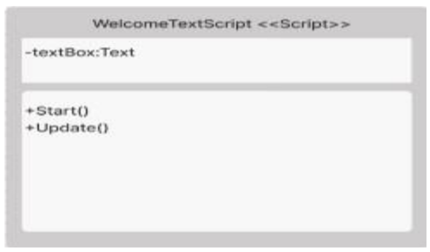


Figure 87: Welcome Text Script Class

- **Class PauseMenu -Method Start**

  This method is activated when the scene opens. It finds the welcome text from canvas.

- **Class PauseMenu -Method Update**

  This method is triggered when every frames changes. It assigns the text of welcomeText to username of logged in server by using player's information.

4.2.2.36 SocketIOComponent<<Script>>Class

SocketIOComponent is a script of where the socket programming happens. It also subclass of unity's monobehaviour class. Sub classes of SocketIOComponent are Login Controller and Menu Controller. The class object belonging to the UML design is shown in the **Figure 88**.

Figure 88: SocketIO Component Script Class

4.3. Cyber Drone Web Subsystem Classes

Cyber Drone project has 3 subsystem which are in web subsystem. These three subsystems are explained below.

4.3.1 Web Server Subsystem Classes

4.3.1.1 TokenController <<Script>> Class

Token Controller class is for managing the user's login procedure. It checks user credentials to make user authenticate and validate exist token. This class communicates with UserForValidationModel, TokenModel and Crypto class for managing authentication methodologies. The class object belonging to the UML design is shown in the **Figure 89**.

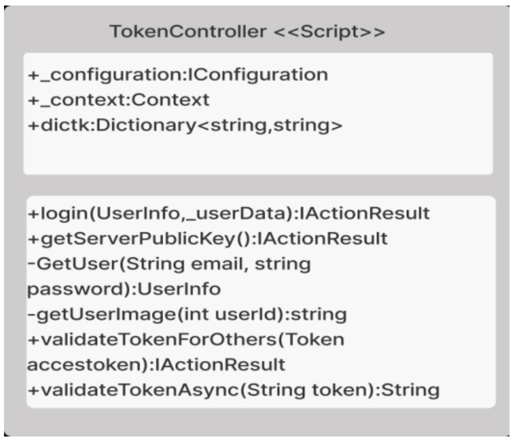

TokenController <<Script>>
+_configuration:IConfiguration
+_context:Context
+dictk:Dictionary<string,string>
+login(UserInfo,_userData):IActionResult
+getServerPublicKey():IActionResult
-GetUser(String email, string password):UserInfo
-getUserImage(int userId):string
+validateTokenForOthers(Token accestoken):IActionResult
+validateTokenAsync(String token):String

Figure 89: Token Controller Script Class

- **Class TokenController-Method login**

  This method is for making user authenticate. It takes UserInfo information from web request. Firstly, it decrypts user's password which is encrypted by client which uses server public key by using server's private key. These methods take a hash value of password by using sha256. Then it encrypts it by using AES. Purpose of this procedure is for comparing user's password with password field in database which is stored as hashed, and AES encrypted. After that, it checks user's email and password from by calling and giving request information to getUser method. If there are valid credentials it sends a token and username to users as a response. Also, it saves user's information and token in server for other operations that user wants. If user's credentials are invalid, it sends an invalid credentials error to user as a web response.

- **Class TokenController-Method getUserImage**

  This method finds the user image path by using user id which comes from api, then opens the file with the path of related user after that, image of user is converted to base64 string format for json.

- **Class TokenController-Method getServerPublicKey**

  This method takes the server's public key from Crypto class. Then it sends server's public key to client for client's encryptions processes.

- **Class TokenController-Method getUser**

  This method takes the users' email and password then checks is there any user belongs to these credentials and return user's credentials if there is.

- **Class TokenController-Method validateTokenForOthers**

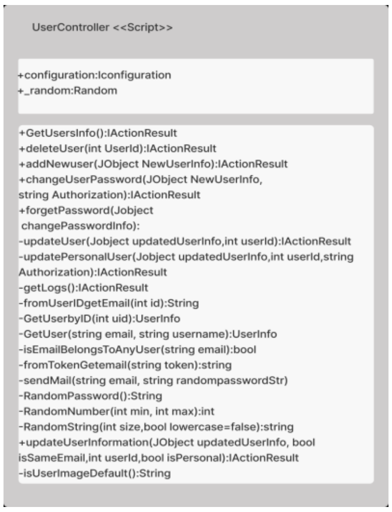
  This method takes the users' token, then finds the related user id with using that token.

- **Class TokenController-Method validateTokenAsync**

  This method takes the token from web request. Then it checks does system contains this token. If it does, it returns users credentials as UserForValidation form which includes username,role,firstname,lastname,email,userid and image. Additionally, it adds the server public key for future processes. Otherwise, it returns invalid token response to client.

4.3.1.2. UserController <<Script>> Class

User Controller class is for managing the user's information procedure. It is responsible for create, read, update and deleting operations of users. This class communicates with UserInfoModel for managing user operations.
The class object belonging to the UML design is shown in the **Figure 90**.



```
UserController <<Script>>

+configuration:Iconfiguration
+_random:Random

+GetUsersInfo():IActionResult
+deleteUser(int UserId):IActionResult
+addNewuser(JObject NewUserInfo):IActionResult
+changeUserPassword(JObject NewUserInfo,
string Authorization):IActionResult
+forgetPassword(Jobject
 changePasswordInfo):
-updateUser(Jobject updatedUserInfo,int userId):IActionResult
-updatePersonalUser(Jobject updatedUserInfo,int userId,string
Authorization):IActionResult
-getLogs():IActionResult
-fromUserIDgetEmail(int id):String
-GetUserbyID(int uid):UserInfo
-GetUser(string email, string username):UserInfo
-isEmailBelongsToAnyUser(string email):bool
-fromTokenGetemail(string token):string
-sendMail(string email, string randompasswordStr)
-RandomPassword():String
-RandomNumber(int min, int max):int
-RandomString(int size,bool lowercase=false):string
+updateUserInformation(JObject updatedUserInfo, bool
isSameEmail,int userId,bool isPersonal):IActionResult
-isUserImageDefault():String
```

Figure 90: User Controller Script Class

- **Class UserController-Method GetUsersInfo**

  This method collects all user's information which contains user id, first name, last name, email, type and creating date from database. It returns this information as a web response.

- **Class UserController-Method deleteUser**

  This method takes user id from request url. Then it deletes the related user from database via user's id. At the end, it returns success message as a web response.

- **Class UserController-Method addNewUser**

  This method takes user's related information which contains first name, last name and email from body of request. Then it calls the isEmailBelongsToAnyUser for checking any user is registered with this email. If not, then it calls randomPassword method for creating random password to user. Generated password will be hashed and encrypted by using sha and AES. New user role is assigned to user role by initially. Then these methods execute database query for adding new users. After that it calls sendMail method for sending user password for accessing system. Then it returns success message to client as web response. If any user is registered with email that client sends, it returns error message to client as web response.

- **Class UserController-Method changeUserPassword**

  This method takes old and new password information from client request's body which is encrypted by server's public key. Additionally, it takes token from request header. This method calls fromTokenGetEmail method by giving token. Purpose getting user's email from this token. If there is email that belongs to token. From request, it takes old password, then it is decrypted by Crypto Class by using server private key. Old password will be hashed and encrypted by using makeSHA256andAes method from Crypto Class. Then from database, password of that user will be returned by using user's email. After that, old passwords are checked by this method. If they are not matched this method return invalid credentials error to client as web response. If they matched, new password is decrypted by Crypto Class by using server private key. New password will be hashed and encrypted by using makeSHA256andAes method from Crypto Class. Then it updates related user's password part by executing database query. Then it returns success message to client as web response.

- **Class UserController-Method forgetPassword**

  This method takes user's related information which contains email and username from body of request. Then it calls the isEmailBelongsToAnyUser for checking any user is registered with this email. If, then it calls randomPassword method for creating random password to user. Generated password will be hashed and encrypted by using SHA and AES. Then these methods execute database query for updating pass of related user. After that it calls sendMail method for sending user password for accessing system. Then it returns success message to client as web response. If user is not registered with email that client sends, it returns error message to client as web response.

- **Class UserController-Method updateUser**

  This method takes 2 arguments which are updated user information and user id, firstly, according to user id, fromUserIDgetEmail methods is run. Return of that methods brings to email address of user. Gotten email address is compared with the updated information, Goal of adding that to that process is finding the email is new or not. If it is new, server must check other email address. Because in system, email addresses are unique values. After comparing updateUserInformation methods will be run according to updated email is new or not. updateUserInformation methods takes a special argument for personal updating. In that case this argument takes false Boolean, because admins want to change user information. According to result of that methods, server sends error or success message to client.

- **Class UserController-Method updatePersonalUser**

  This method takes 2 arguments which are updated user information and user id, firstly, according to user id, fromUserIDgetEmail methods is run. Return of that methods brings to email address of user. Gotten email address is compared with the updated information, Goal of adding that to that process is finding the email is new or not. If it is new, server must check other email address. Because in system, email addresses are unique values. After comparing updateUserInformation methods will be run according to updated email is new or not. updateUserInformation methods takes a special argument for personal updating. In that case this argument takes true Boolean, because user want to change her/his own user information. According to result of that methods, server sends error or success message to client.

- **Class UserController-Method getLogs**

  This method returns user logs. Server finds all logs of user operation such as create,update,delete. Also web server returns user log information such as logID,operationType, UserId, Firstname, Lastname, Username, Email, Type, CreatedDate, ImagePath and logDate.

- **Class UserController-Method fromUserIDgetEmail**

  This method takes one argument from web request which is user id, according to this id, server executes database query for finding related user email address then returns it.

- **Class UserController-Method GetUserbyID**

  This method takes one argument which is user id, according to this id, server executes database query for finding related user then returns it

- **Class UserController-Method getUser**

  This method takes 2 arguments which are email and username. Then By using database, it returns user's information which is matched with username and email.

- **Class UserController- Method isEmailBelongToAnyUser**

  This method takes the argument which is email. Then By using database, it returns true which is matched with this email. If it is not return false.

- **Class UserController-Method fromTokenGetEmail**

  This method takes the argument which is token. Then By using server, it checks the token is stored or not. If token is stored it finds related user which is matched with this token. Then it returns related user's email.

- **Class UserController-Method sendMail**

  This method takes 2 arguments which are email and random password. It sends the randomly generated password to user's email by using email service.

- **Class UserController-Method randomPassword**

  This method generates random password by using randomNumber and RandomString methods. Then it returns random password in string form.

- **Class UserController-Method randomNumber**

  This method takes 2 arguments which are max in min values for detecting interval of random integers. Then it generates random integer according to this interval. Then it returns this randomly generated integer.

- **Class UserController-Method randomString**

  This method takes 2 arguments which are size(integer) and isLowercase(boolean). According to size, length of string is set. Additionally, according to lowercase bool, it generates lower case string or uppercase string. It generates random string and returns this string.

- **Class UserController-Method updateUserInformation**

  This method takes four argument which are updatedUserInfo, isSameEmail, userId and isPersonal. UpdatedUserInfo contains user's new information. IsSameEmail is for checking updated email is same with the old one. userId is for finding related user in database. IsPersonal is for if user wants to change own credentials, or any admin want to change other user's information. Firtly, methods check isSameEmail Boolean for understanding client want to change email or not. If client wants to change, server calls isEmailBelongsToAnyUser methods for whether email belongs to someone or not. If email belongs to someone server return error message to client. Else, server checks user wants to change own credentials or admin want to change user's information, according to that related codes are executed. It just updates user credentials which does not includes image field. Otherwise, if user want to change his/her own credentials, server executes related part of code which includes bot user information and image field. This part includes checking image field for default image. Server calls for isUserImageDefault method for not overwriting or overwriting image for server having not so many images. If user want to change profile picture and user image is not default, then new image overwrites the old image, but if user image is default image and user want to change it, so server creates a new image in related folder and includes in database by executing related query. At the end, according to processes server return related error message or success message to client.

- **Class UserController-Method isUserImageDefault**

  This method takes one argument which is userid for finding user image path, server runs database query and return image path of user, it compares the image path of related user with default image path, if they are matched, methods return null, else it returns the image path of related user in string form.

4.3.1.3. ConfigController <<Script>> Class

Config Controller class is for managing the user's configuration procedure. It is responsible for creating, updating, and reading operations of user's config. This class communicates with ConfigModel for managing this procedure. The class object belonging to the UML design is shown in the **Figure 91**.

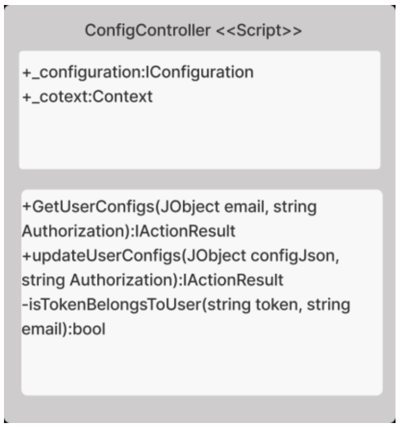

ConfigController <<Script>>
+_configuration:IConfiguration
+_cotext:Context
+GetUserConfigs(JObject email, string Authorization):IActionResult
+updateUserConfigs(JObject configJson, string Authorization):IActionResult
-isTokenBelongsToUser(string token, string email):bool

Figure 91: Config Cotroller Script Class

- **Class ConfigController-Method getUserConfigs**

  This method takes email information from client request's body. Additionally, it takes token from request header. According to token and email, it calls isTokenBelongsToUser method for checking this token related with this email or not. If they are matched, it executes read query for getting user's configs by using email that comes from request. Then configs are sent to client in web response. If token and email are not matched, then server sends error messages in response form.

- **Class ConfigController-Method updateUserConfigs**

  This method takes config and email information from client request's body. Additionally, it takes token from request header. According to token and email, it calls isTokenBelongsToUser method for checking this token related with this email or not. If they are matched, it executes update query for updating user's configs by using email that comes from request. Then success message is sent to client in web response. If token and email are not matched, then server sends error messages in response form.

- **Class ConfigController-Method isTokenBelongsToUser**

  This method takes 2 arguments which are token and email in string form. Then it checks is there any token is matched with token that comes from arguments. If the token is found, it returns related user's email, then if emails are matched, these methods return true otherwise it returns false.

4.3.1.4. BugController <<Script>> Class

Bug Controller class is for managing the bugs procedure. It is responsible for creating, updating and reading operations of bug information. The class object belonging to the UML design is shown in the **Figure 92**.

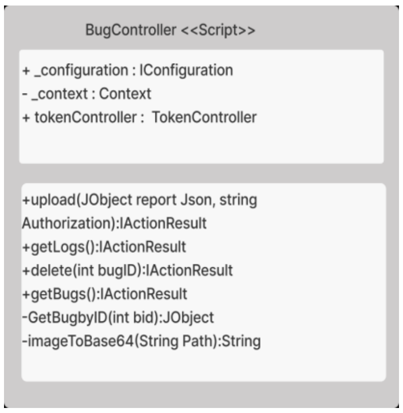

BugController <<Script>>
+ _configuration : IConfiguration
- _context : Context
+ tokenController : TokenController
+upload(JObject report Json, string Authorization):IActionResult
+getLogs():IActionResult
+delete(int bugID):IActionResult
+getBugs():IActionResult
-GetBugbyID(int bid):JObject
-imageToBase64(String Path):String

Figure 92: BugController Script Class

- **Class BugController-Method upload**

  This method takes two arguments from web request which are report json and Authorization token, firstly, by using using tokenController ValidateTokenForOther method is run by giving token to method.It returns userid of related token. From report json, method takes title, description and image of report. Image is in base64 format. So firstly, image is converted to byte array then image is saved in related folder. Then related database query for adding new report is executed. Then server return error or success message depends on the status of process.

- **Class BugController-Method getLogs**

  This method returns bug logs. Server finds all logs of bug operation such as create,update,delete. Also web server returns user log information such as logID,operationType, userId, title,description, image,date,logDate,userFullName.

- **Class BugController-Method delete**

  This method takes two arguments from web request which is bugID, according to bugId server calls getBugbyID, it returns the bug information because for deleting image from image path. Then server executes delete operation in database by using bugID, then servers use image path from related bug for deleting image in related folder. Then server return error or success messages according to status of process.

- **Class BugController-Method getBugs**

  This method returns all bugs in server. Server finds all logs information such as create,update,delete. Also web server returns bugs log information such as bugId,bugTitle, bugDescription, Bugimage, BugDate.

- **Class BugController-Method GetBugbyID**

  This method takes on argument which is bug id. It execetues read operation in database by using bugid. Then it returns bugId and BugImage information as JObject form.

- **Class BugController-Method imageToBase64**

  This method takes on argument, which is image Path, it reads the related file, and gets bytes of this file. Then byte array is converted to base64 string for, and that string is returned.

4.3.1.5 RecordController <<Script>> Class

Record Controller class is for managing the records procedure. It is responsible for creating, updating and reading operations of record information. The class object belonging to the UML design is shown in the **Figure 93**.

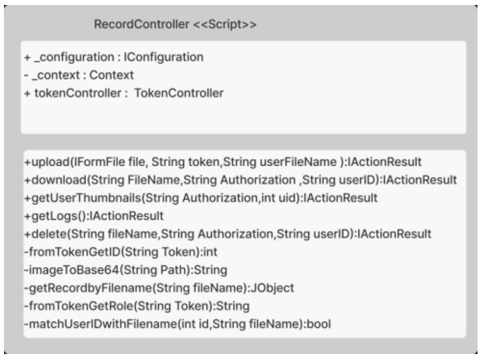

RecordController <<Script>>
+ _configuration : IConfiguration
- _context : Context
+ tokenController : TokenController
+upload(IFormFile file, String token,String userFileName):IActionResult
+download(String FileName,String Authorization ,String userID):IActionResult
+getUserThumbnails(String Authorization,int uid):IActionResult
+getLogs():IActionResult
+delete(String fileName,String Authorization,String userID):IActionResult
-fromTokenGetID(String Token):int
-imageToBase64(String Path):String
-getRecordbyFilename(String fileName):JObject
-fromTokenGetRole(String Token):String
-matchUserIDwithFilename(int id,String fileName):bool

Figure 93: RecordController Script Class

- **Class RecordController-Method upload**

  This method takes three arguments from web request form which are mp4 file of record, token and userfilename. Firstly, by using using tokenController ValidateTokenForOther method is run by giving token to method.It returns userid of related token. It creates random file name for saving coming mp4 file. Then it executues insert query for saving new coming record. Record information includes userId,path, thumbPath and name. Then new coming mp4 file is saved in file and name is assigned by random file name generator. After that, ffmpeg program is executed by creating new task. It extracts a thumbnail(jpeg) of mp4 record and saves in thumbnails folder with same name which is assigned to mp4 record. According to status of process, server returns error message or success message.

- **Class RecordController-Method download**

  This method takes one argument which is filename, servers open relate file by using that argument. Then it reads the bytes of this file. After that it returns the related file in mp4 format.

- **Class RecordController-Method getUserThumbnails**

  This method takes two arguments which are Authorization token and user id. Server executes database query for reading thumbnail information. It returns thumnail image in base64 form by calling imageToBase64 method, userFilename and filename properties.

- **Class RecordController-Method getLogs**

  This method record bug logs. Server finds all logs of record operation such as create,update,delete. Also web server returns record log information such as logID,operationType, userId, path, thumbpath,logDate,userFullName.

- **Class RecordController -Method delete**

  This method takes one argument which is filename. From filename it finds record informations by calling GetRecordbyFileName methos. It returns record information of related record. Firstly, server executes delete query by using recordID property. Then moves the file that wanted to be deleted to another folder called deletedRecords for saving in future. Then thumbnails of related records is deleted permanently. Then server returns error or success message according to status of process.

- **Class RecordController -Method fromTokenGetID**

  This method takes one argument called token. From that token, relates user is found. Then related user id is returned by this method.

- **Class RecordController -Method imageToBase64**

  This method takes one argument, which is image Path, it reads the related file, and gets bytes of this file. Then byte array is converted to base64 string for, and that string is returned.

- **Class RecordController -Method getRecordbyFilename**
  This method takes one argument, which is filename, then servers execute database read query by using filename property. It returns the id, mp4Path and thumbPath of record.

- **Class RecordController -Method fromTokenGetRole**
  This method takes one argument, which is token, it finds the user role from the token.

- **Class RecordController -Method matchUserIDwithFilename**
  This method takes two arguments, which are user id and file name, it gets the id of given file name with using database query then checks the ids are matched or not. According to that, it returns Boolean value.

4.3.1.6 ControllerBaseController <<Script>> Class

This class is the main class of all controllers. Controllers that mentioned above extends to this class. It is responsible for perceiving request URL. According to that, related controller starts to process. The class object belonging to the UML design is shown in the **Figure 94**.

Figure 94: Controller Base Class

4.3.1.7 Crypto <<Script>> Class

Crypto class creates RSA, SHA and AES methods for encryption, decryption and hashing methods. It is responsible for creating public and private keys and making them encrypted, decrypted, and hashed. Initially, it creates RSA, sha, private and public keys when this class initialized. The class object belonging to the UML design is shown in the **Figure 95**.

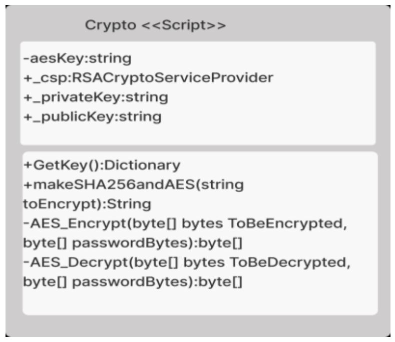

Crypto <<Script>>
-aesKey:string
+_csp:RSACryptoServiceProvider
+_privateKey:string
+_publicKey:string
+GetKey():Dictionary
+makeSHA256andAES(string toEncrypt):String
-AES_Encrypt(byte[] bytes ToBeEncrypted, byte[] passwordBytes):byte[]
-AES_Decrypt(byte[] bytes ToBeDecrypted, byte[] passwordBytes):byte[]

Figure 95: Crypto Script Class

- **Class Crypto-Method GetKey**

  This method returns the public key of server for encryption processes.

- **Class Crypto-Method makeSHA256andAES**

  This method takes argument which is string that is wanted to be encrypted. Then this method converts the string to byte array. Then it applies same procedure for AES key. Then it computes hash of AES key which in form of bytes. After that, this method calls AES_Encrypt method to encrypt. At the end of AES encryption, encrypted result byte array is converted to string form and this string is returned.

- **Class Crypto-Method AES_Encrypt**

  This method takes 2 arguments which are byte array that wanted to be encrypt and byte array that is AES key. It encrypts the wanted value according to AES encryption then it returns the encrypted value in byte array form.

- **Class Crypto-Method AES_Decrypt**

  This method takes 2 arguments which are byte array that wanted to be decrypt and byte array that is AES key. It decrypts the wanted value according to AES decryption then it returns the decrypted value in byte array form

4.3.1.8 Context <<Script>> Class

This class is responsible for managing all models. In creation, it adds and matches these models to database to take actions in database easily. The class object belonging to the UML design is shown in the **Figure 96**.

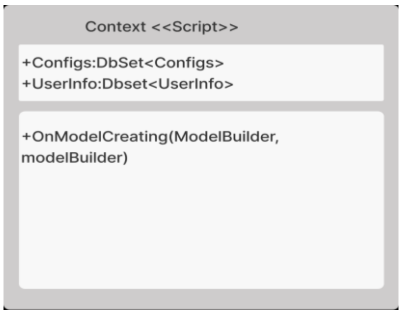

Context <<Script>>
+Configs:DbSet<Configs>
+UserInfo:Dbset<UserInfo>
+OnModelCreating(ModelBuilder, modelBuilder)

Figure 96: Context Script Class

- **Class Context-Method OnModelCreating**

This method creates database models for taking actions in database.

4.3.1.9 ConfigModel <<Model>> Class

Config Model is created for managing config information as a class form. It stores configs information. This model is exactly same with database form of config. It works for ConfigController. The class object belonging to the UML design is shown in the **Figure 97**.

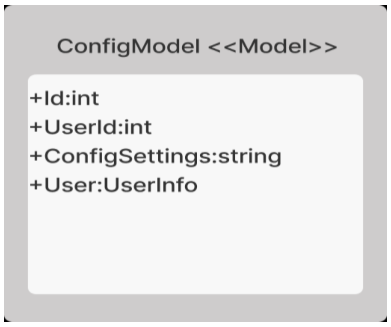


Figure 97: Config Model Script Clas.

4.3.1.10 UserInfoModel <<Model>> Class

UserInfo Model is created for managing user information as a class form. It stores user information. This model is exactly same with database form of user. It works for UserController. The class object belonging to the UML design is shown in the **Figure 98.**

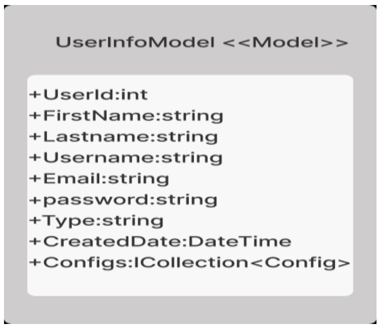


Figure 98: User Info Model Script Class

4.3.1.11 TokenModel <<Model>> Class

Token Model is created for managing and storing token information as a class form. It works for TokenController. The class object belonging to the UML design is shown in the **Figure 99.**

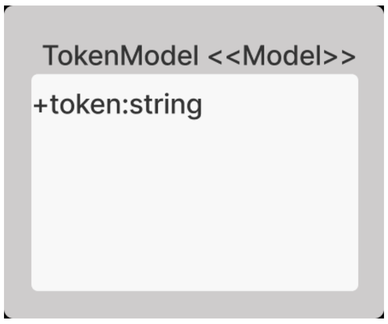


Figure 99: Token Model Script Class

4.3.1.12 UserForValidationModel<<Model>> Class

UserForValidation is created for managing and storing user information. Difference between user model it stores 8 attributes which are email, username, and role. It works for TokenController. The class object belonging to the UML design is shown in the **Figure 100**.

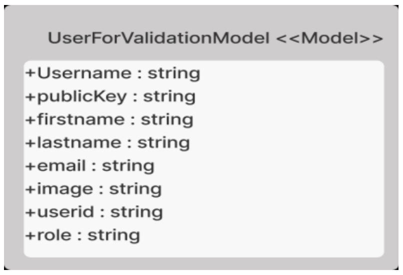


Figure 100: User For Validation Model Script Class

4.3.2 Client Server Subsystem Classes

4.3.2.1. Store <<Script>> Class

Store.js is created for managing state information by using Vuex. It includes state, getters, mutations and actions. It exports the store as vuex store format.
The class object belonging to the UML design is shown in the **Figure 101**.

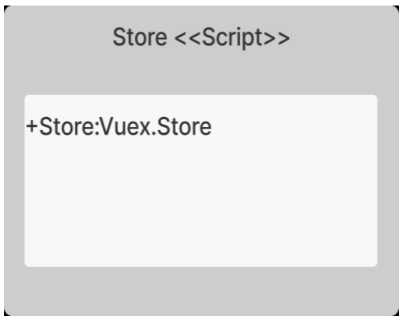


Figure 101: Store Script Class

- **Class Store-Method state**

  This method contains the state of desired information such as user. User object contains username, isAuthenticated, token, role, firstname, lastname, email, image, userID and socket properties. This method is for managing these kinds of information in everywhere in vue js components globally.

- **Class Store-Method getters**

  This method returns the state that stores in Vuex Management System which is mentioned in state part. These methods return the state object. In project it returns user object properties such as username, isAuthenticated, token, etc.

- **Class Store-Method mutations**

  This method is for changing state of user which stored in Vuex Management System. Mutations are triggered by actions. Then changes the related value of state(user).

- **Class Store-Method actions**

  This method is for triggering mutations. Firstly, it takes commit object for triggering and desired value. Then it triggers the desired mutations for changing states.

4.3.2.2 Router <<Script>> Class

Router.js is created for managing routing operations. It checks the desired page that user wants to access it, then manipulates the components according to this manipulation. Then renders the new pages. The class object belonging to the UML design is shown in the **Figure 102**.

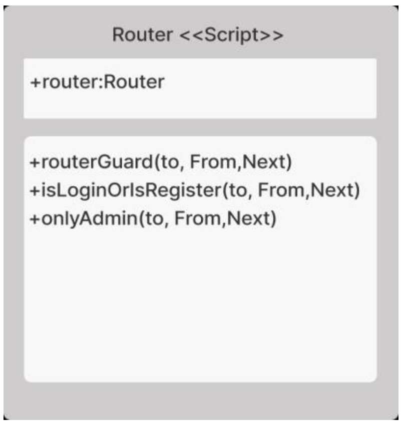

Router <<Script>>
+router:Router
+routerGuard(to, From,Next)
+isLoginOrIsRegister(to, From,Next)
+onlyAdmin(to, From,Next)

Figure 102: Router Script Class

- **Class Store-Method routerGuard**

  This method takes their arguments which are to, from and next. Next is for using the next page of user wants to open. Firstly, it checks the user is authenticated or not for accessing pages. If user is already authenticated, then it allows to open new page. Otherwise, it checks the token in local storage that created in login process. If there is a token here, it sends this token to validateToken api, servers return a message whether error message or validation properties. These properties are mentioned in store state part in this document (4.4.1) It assigns the coming values to user state, then creates a new socket for handling connection to online user server. Then calls handshaking method in there and sends related information which are email, image, username and location(web) properties. Then user is directed to next page.

- **Class Store-Method isLoginOrIsRegister**

  This method controls the page is login or register page for accessing pages.

- **Class Store-Method onlyAdmin**

  This method is for controlling admin pages. Only admin users can this page. Register and users are admin pages. Only admin is allowed to use these pages. Also, operations are same with routerGuard that mentioned in this section.

4.3.2.3. App <<Script>> Class

App.vue is created for managing html rendering operations. It renders the vue components in html tag called router-view. It updates html component that comes from router.js that every change that occurred.

The class object belonging to the UML design is shown in the **Figure 103**.

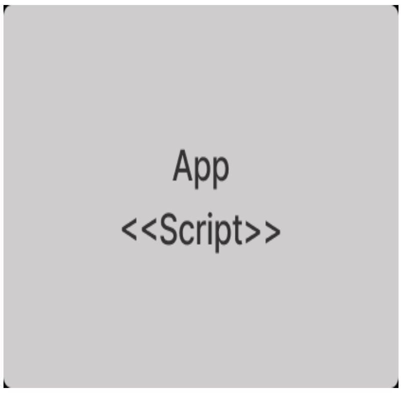
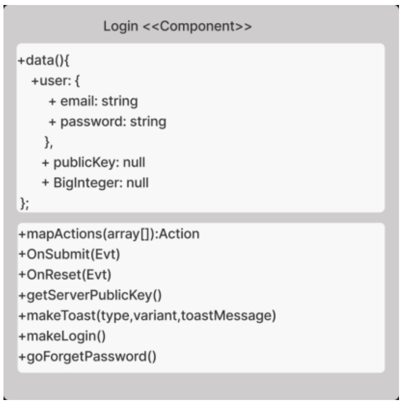


Figure 103: App Script Class

4.3.2.4 Login <<Component>> Class

Login.vue is created for handling login operations with javascript and rendering related html in that part. The class object belonging to the UML design is shown in the **Figure 104**.



```
Login <<Component>>
+data(){
  +user: {
     + email: string
     + password: string
    },
   + publicKey: null
   + BigInteger: null
};
+mapActions(array[]):Action
+OnSubmit(Evt)
+OnReset(Evt)
+getServerPublicKey()
+makeToast(type,variant,toastMessage)
+makeLogin()
+goForgetPassword()
```

Figure 104:Login Component Class

- **Class Store-Method mapActions**
  This method gets the given actions in array. Format in vuex actions.

- **Class Store-Method OnSubmit**
  This method is triggered when client clicks on the submit button and it calls getServerPublicKey methods for preparing login operation.

- **Class Store-Method OnReset**
  This method is triggered when client clicks on the reset button and it clears the user object's properties.

- **Class Store-Method getServerPublicKey**

  This method creates a get request. Then it makes a request to related api which is "getServerPublicKey" then gets the server's public key from response of api message. Message format is xml format, so it calls parseXmlToJson method for assigning public key for RSA. Then It encrypts the user information with using server public key and RSA encryption system. Then it calls makelogin methods for completing login operation.

- **Class Store-Method makeToast**

  This method creates a toaster according to arguments that are called type, variant and toastmessage. It renders a toaster component with error or success variant. And it gives a given message to toaster.

- **Class Store-Method makeLogin**

  It creates a post request to getToken URL. It puts a user object in string form in request. Then it catches the response of server, if message is success request, it takes a message of data that in response then it adds token username and role information to local storage in browser. Then it calls the mapActions to assign state of user in vuex management system. then creates a new socket for handling connection to online user server. Then calls handshaking method in there and sends related information which are email, image, username and location(web) properties. Then user is directed to next page.

- **Class Store-Method goForgetPassword**

  This method redirects client to forget password page.

4.3.2.5 Navbar <<Component>> Class

Navbar.vue is created for handling navigation var operations with javascript and rendering related html in that part. It renders show special elements according to user's authentication state and role.The class object belonging to the UML design is shown in the **Figure 105**.

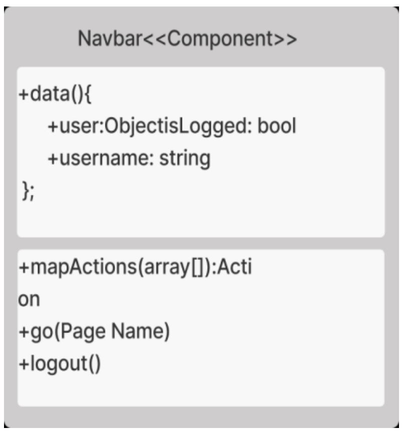


Figure 105: Navbar Component Class

- **Class Navbar-Method mapActions**

  This method gets the given actions in array.Format in vuex actions.

- **Class Navbar-Method go (Page Name)**

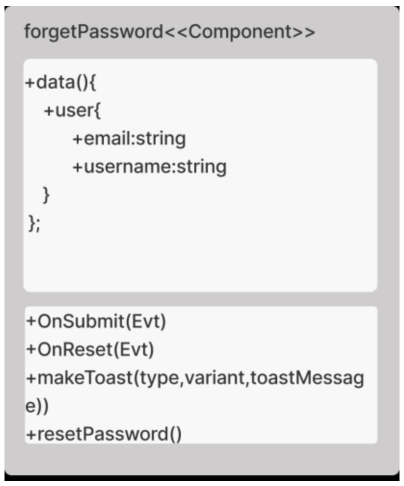
  There are several methods that start with go such as goRegister, goLogs,goAdd, etc… This method is responsible for redirecting client to page that her/his desired.

- **Class Navbar-Method logout**

  This method is responsible for logging the user out. In this method, user state that stored in vuex store management (store js) is cleared. Then user socket that connects the user to online user system is disconnected. After that, user's validation token is removed from local storage which is in browser. At the ends, user is directed to login page.

4.3.2.6 forgetPassword <<Component>> Class

forgetPassword.vue is created for handling re-passwording operations with javascript and rendering related html in that part. The class object belonging to the UML design is shown in the **Figure 106**.



```
forgetPassword<<Component>>

+data(){
  +user{
     +email:string
     +username:string
  }
};

+OnSubmit(Evt)
+OnReset(Evt)
+makeToast(type,variant,toastMessag
e))
+resetPassword()
```

Figure 106: forgetPassword Component Class

- **Class forgetPassword -Method OnSubmit**

  This method is triggered when client clicks on the submit button and it calls resetPassword methods for preparing re-passwording operation.

- **Class forgetPassword -Method OnReset**

  This method is triggered when client clicks on the reset button and it clears the user object's properties.

- **Class forgetPassword -Method makeToast**

  This method creates a toaster according to arguments that are called type, variant and toastmessage. It renders a toaster component with error or success variant. And it gives a given message to toaster.

- **Class forgetPassword -Method resetPassword**

  It creates a post request to forget-password url. It puts a user object in string form in request which contains user email and username. Then it catches the response of server, if message is success request, makeToast methods is called by giving success message of server. Then user is directed to user page. Web server sends a new password to user's email address. Otherwise, if server message is error response, makeToast methods is called by giving error message of server.

4.3.2.7 onlineUsers <<Component>> Class

onlineUsers.vue is created for handling showing online users that in system with javascript and rendering related html in that part. The class object belonging to the UML design is shown in the **Figure 107**.

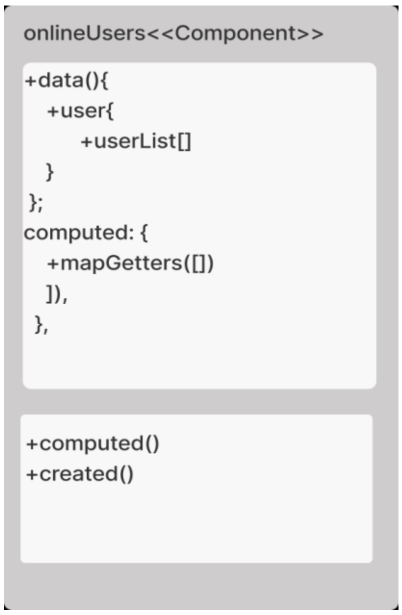


Figure 107: onlineUsers Component Class

- **Class onlineUsers -Method computed:**

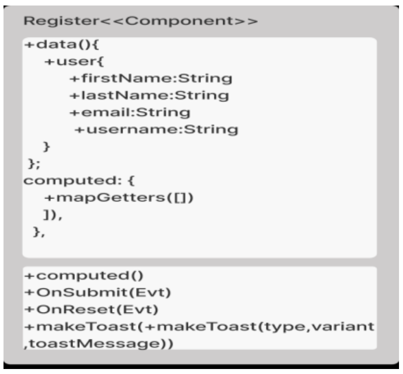
  This method is triggered on the computed part in vue system. With the help of that, vuex store management system's states get the related getters about user states.

- **Class onlineUsers -Method created:**

  This method is triggered on the created part in vue system. Firstly, by using computed property which is getSocket is called. After that socket creates a listener such as, userList, connected and disconnected. userList is for collecting online users that is already in system. Connected is understanding that a new user is connected to online user system. Then it collects the new user's information and adds to user list. Disconnected is the reverse procedure of connected. It understands an user is disconnected from system. Then it removes the user from user list.

4.3.2.8 Register <<Component>> Class

register.vue is created for handling adding new user to syste with javascript and rendering related html in that part. The class object belonging to the UML design is shown in the **Figure 108**.



```
Register<<Component>>
+data(){
   +user{
       +firstName:String
       +lastName:String
       +email:String
        +username:String
   }
 };
computed: {
   +mapGetters([])
   ]),
 },

+computed()
+OnSubmit(Evt)
+OnReset(Evt)
+makeToast(+makeToast(type,variant
,toastMessage))
```

Figure 108: Register Component Class

- **Class register -Method computed**

  This method is triggered on the computed part in vue system. With the help of that, vuex store management system’s states get the related getters about user states.

- **Class register -Method makeToast**

  This method creates a toaster according to arguments that are called type, variant and toastmessage. It renders a toaster component with error or success variant. And it gives a given message to toaster.

- **Class register -Method onReset**

  This method is triggered when client clicks on the reset button and it clears the user object’s properties.

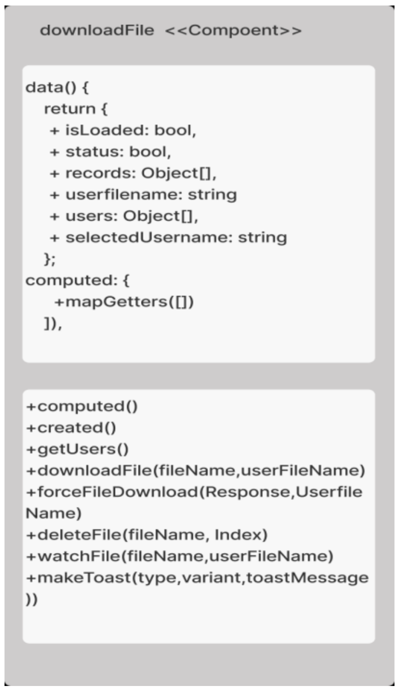
- **Class register -Method onSubmit**

  It creates a put request to user/add url. It puts a user object in string form in request which contains user firstname ,lastname, email and username. Then it catches the response of server, if message is success request, makeToast methods is called by giving success message of server. Then user is directed to main page. Web server sends a new password to added user’s email address. Otherwise, if server message is error response, makeToast methods is called by giving error message of server.

4.3.2.9 downloadFile <<Component>> Class

downloadFile.vue is created for handling records that in system with javascript and rendering related html in that part. It manipulates the user records like deleting, watching, and downloading.

The class object belonging to the UML design is shown in the **Figure 109**.



```
downloadFile <<Compoent>>

data() {
  return {
   + isLoaded: bool,
   + status: bool,
   + records: Object[],
   + userfilename: string
   + users: Object[],
   + selectedUsername: string
  };
computed: {
    +mapGetters([])
  ]),

+computed()
+created()
+getUsers()
+downloadFile(fileName,userFileName)
+forceFileDownload(Response,Userfile
Name)
+deleteFile(fileName, Index)
+watchFile(fileName,userFileName)
+makeToast(type,variant,toastMessage
))
```

Figure 109:downloadFile Component Class

- **Class downloadFile -Method computed**

  This method is triggered on the computed part in vue system. With the help of that, vuex store management system's states get the related getters about user states.

- **Class downloadFile -Method created**

  This method is triggered on the created part in vue system. Firstly, it calls getRecords method for getting single user's records. Then checks the role of user to get users that in the system for displaying other user's records.

- **Class getUsers -Method getUsers**

  It creates a get request to user/squad url. It gives a user's token in string form in request. Then it catches the response of server, if message is success request, it gets the user list in json form which is came from response of server. Then it assigns the users list to response of server which is userlist.

- **Class getRecords -Method getRecords**

  This method takes three arguments which are userid, firstname and lastname. It creates a get request to record/thumbnails/[userid] url. It gives a user's token in string form in request. Then it catches the response of server, if message is success request, it gets the record list in json form which is came from response of server. Single record object contains thumbnails image in base64 string form, userFileName and filename. It assigns the record list to response of server.

- **Class getRecords -Method downloadFile**

  This method takes two arguments which are filename, userfilename. It creates a get request to record/download/[filename]url. It gives a user's token in string form in request. Then it catches the response of server, if message is success request, it gets the record in array buffer in bytes form which comes from web server. Then it calls forceFileDownload method for downloading operation.

- **Class getRecords -Method forceFileDownload**

  This method takes two arguments which are response of server in mp4 form, userfilename. It creates a html which is for downloading the coming file.

- **Class getRecords -Method deleteFile**

  This method takes two arguments which are filename and index of record that in record list in client system. It creates a delete request to record/thumbnails/ [filename]. It gives a user's token in string form in request. Then it catches the response of server, if message is success request, given index recorded is popped from record list in client system, then makeToast methods is called by giving success message of server. Otherwise, if server message is error response, makeToast methods is called by giving error message of server.

- **Class getRecords -Method watchFile**

  This method takes two arguments which are filename, userfilename. It creates a get request to record/download/[filename]url. It gives a user's token in string form in request. Then it catches the response of server, if message is success request, it gets the record in array buffer in bytes form which comes from web server. Then it creates a modal in html, then it gives the coming byte array to video tag. With the help of that, user is able to watch her/his on record in client system.

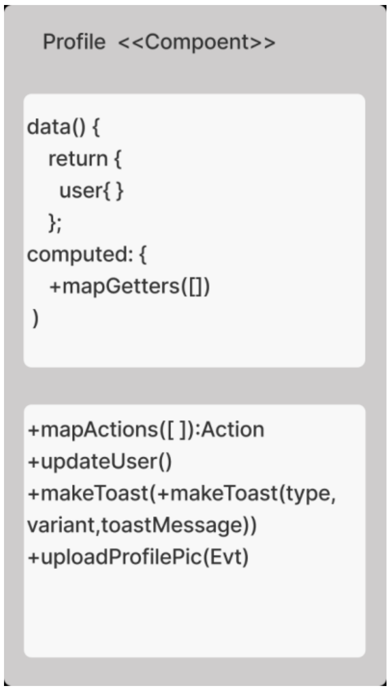
- **Class getRecords -Method makeToast**

  This method creates a toaster according to arguments that are called type, variant and toastmessage. It renders a toaster component with error or success variant. And it gives a given message to toaster.

4.3.2.10 Profile <<Component>> Class

profile.vue is created for handling user's information that in system with javascript and rendering related html in that part. It manipulates the user's information by updating them.

The class object belonging to the UML design is shown in the **Figure 110**.



```
data() {
  return {
    user{ }
  };
computed: {
  +mapGetters([])
)
```

```
+mapActions([ ]):Action
+updateUser()
+makeToast(+makeToast(type,
variant,toastMessage))
+uploadProfilePic(Evt)
```

Figure 110:Profile Component Class

- **Class profile -Method computed**

  This method is triggered on the computed part in vuex system. With the help of that, vuex store management system's states get the related getters about user states.

- **Class profile -Method mapActions**

  This method is for collection actions that are part of vuex system. With the help of that, vuex store management system's changes the states of user by calling mutations of user's state.

- **Class profile -Method updateUser**

  It creates a put request to user/personal/update/[userID] url. It puts a user object in string form in request which contains user firstname, lastname, email and username and profile image. Then it catches the response of server, if message is success request, makeToast methods is called by giving success message of server. Then calls the action in vuex statement for changing state of users (information) such as updateUsername,updateLastname, etc. Otherwise if server message is error response, makeToast methods is called by giving error message of server.

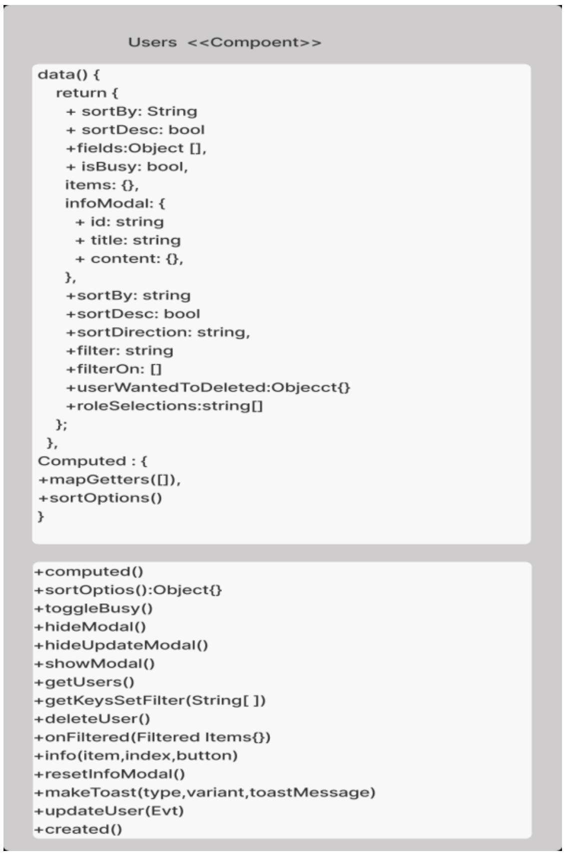
- **Class profile -Method mapActions**

  This method is for uploading file from desktop that user want to change picture. It creates a filereader and creates a html for uploading picture. Then it assigns the new picture in html tags.

4.3.2.11 Users <<Component>> Class

users.vue is created for handling all user’s information that in system with javascript and rendering related html in that part. It manipulates the user’s information by updating reading and deleting them.

The class object belonging to the UML design is shown in the **Figure 111**.



```
Users <<Compoent>>

data() {
   return {
     + sortBy: String
     + sortDesc: bool
     +fields:Object [],
     + isBusy: bool,
     items: {},
     infoModal: {
       + id: string
       + title: string
       + content: {},
     },
     +sortBy: string
     +sortDesc: bool
     +sortDirection: string,
     +filter: string
     +filterOn: []
     +userWantedToDeleted:Objecct{}
     +roleSelections:string[]
   };
 },
Computed : {
+mapGetters([]),
+sortOptions()
}

+computed()
+sortOptios():Object{}
+toggleBusy()
+hideModal()
+hideUpdateModal()
+showModal()
+getUsers()
+getKeysSetFilter(String[ ])
+deleteUser()
+onFiltered(Filtered Items{})
+info(item,index,button)
+resetInfoModal()
+makeToast(type,variant,toastMessage)
+updateUser(Evt)
+created()
```

Figure 111: Users Component Class

- **Class Users -Method computed**

  This method is triggered on the computed part in vuex system. With the help of that, vuex store management system’s states get the related getters about user states. And calls sortOptions methods while in computing state of vue.

- **Class Users -Method sortOptions**

  This method creates the sort labels and their key values for understanding any change occurs in fields. When data are changed which means that computing is occurred, this method is triggered.

- **Class Users -Method toggleBusy**

  This method is created for showing loading icon or not. It takes the reverse of Boolean of isBusy state when it is triggered.

- **Class Users -Method hideModal**

  This method is for clearing the object which is userWantToDeleted and closing the modal called delete user modal.

- **Class Users -Method hideUpdateModal**

  This method is for clearing the object which is infomodal and closing the modal called update user modal.

- **Class Users -Method showModal**

  This method takes two argument which is user that wanted to be deleted and index of user in userlist, then assigns the object called userWantedToDeleted to this user that mentions before. Then it shows a delete user modal for showing to delete warning modal to user.

- **Class Users -Method getUsers**

  It creates a get request to user/squad URL. It puts a user’s token in request. Then it catches the response of server, response of web server is a list of all users that saved in systems. Then it assigns the items to this user list in json form. Then it extracts the keys of json form. Then it gives to this value to getKeysSetFilter methods by calling it. Then it toggles the busy state by calling toggleBusy method. Otherwise, if server message is error response, makeToast methods is called by giving error message of server.

- **Class Users -Method getKeysSetFilter**

  It takes the one argument called keys then it adds the key values to fields list in object form. This is for dynamic rendering of filtering options.

- **Class Users -Method deleteUser**

  This method takes the userid and index of user that in user list in client from userwantedtodeleted object. Then it creates delete request to user/delete/[userid] URL. It puts a user’s token in request. Then it catches the response of server, if message is success request, makeToast methods is called by giving success message of server. Then user that is deleted, is popped from client’s user list too. Otherwise, if server message is error response, makeToast methods is called by giving error message of server.

- **Class Users -Method onFiltered**

  It triggers pagination to update the number of buttons/pages due to filtering.

- **Class Users -Method info**

  It sets the modal title to username and last name for understand which user is wanted to be changed by admin. Then modal content is set to user that wanted to be updated. Then detail modal is shown by this method to user.

- **Class Users -Method resetInfoModal**

  It resets the modal information that shown for updating user.

- **Class Users -Method makeToast**

  This method creates a toaster according to arguments that are called type, variant and toastmessage. It renders a toaster component with error or success variant. And it gives a given message to toaster.

- **Class Users -Method updateUser**

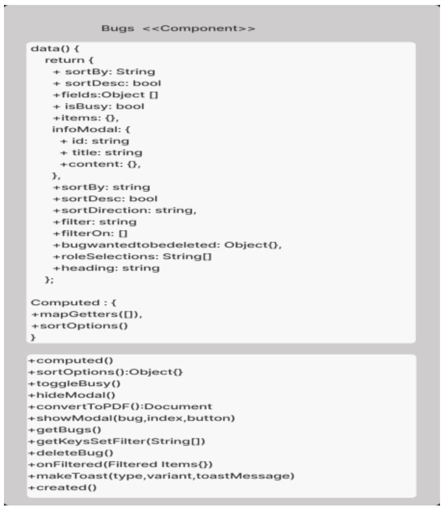
  It creates a put request to user/update/[userID] url. It puts a user object in string form in request which contains user firstname ,lastname, email,role and username. Then it catches the response of server, if message is success request, makeToast methods is called by giving success message of server. Then calls the hideUpdateModal for closing modal that shown for updating user. Otherwise, if server message is error response, makeToast methods is called by giving error message of server.

- **Class Users -Method created**

  This method is triggered on the created part in vuex system. It calls the getUsers methods for getting users and list them into web browser. When page is created this function will be triggered to collect users from server.

4.3.2.12 Bugs <<Component>> Class

bug.vue is created for handling bugs information that in system with javascript and rendering related html in that part. It manipulates the bug's information by reading and deleting them.The class object belonging to the UML design is shown in the **Figure 112**.



```
Bugs <<Component>>

data() {
   return {
     + sortBy: String
     + sortDesc: bool
     +fields:Object []
     + isBusy: bool
     +items: {},
     infoModal: {
       + id: string
       + title: string
       +content: {},
     },
     +sortBy: string
     +sortDesc: bool
     +sortDirection: string,
     +filter: string
     +filterOn: []
     +bugwantedtobedeleted: Object{},
     +roleSelections: String[]
     +heading: string
   };

Computed : {
+mapGetters([]),
+sortOptions()
}

+computed()
+sortOptions():Object{}
+toggleBusy()
+hideModal()
+convertToPDF():Document
+showModal(bug,index,button)
+getBugs()
+getKeysSetFilter(String[])
+deleteBug()
+onFiltered(Filtered Items{})
+makeToast(type,variant,toastMessage)
+created()
```

Figure 112: Bugs Component Class

- **Class Bugs -Method computed**

  This method is triggered on the computed part in vuex system. With the help of that, vuex store management system's states get the related getters about user states. And calls sortOptions methods while in computing state of vue.

- **Class Bugs -Method sortOptions**

  This method creates the sort labels and their key values for understanding any change occurs in fields. When data are changed which means that computing is occurred, this method is triggered.

- **Class Bugs -Method toggleBusy**

  This method is created for showing loading icon or not. It takes the reverse of Boolean of isBusy state when it is triggered.

- **Class Bugs -Method hideModal**

  This method is for clearing the object which is bugWantToDeleted and closing the modal called delete bug modal.

- **Class Bugs -Method convertToPDF**

  This method is for creating a pdf file of listed bugs that in browser and download it.

- **Class Bugs -Method showModal**

  This method takes two argument which is bug that wanted to be deleted and index of bug in buglist, then assigns the object called userWantedToDeleted to this bug that mentions before. Then it shows a delete bug modal for showing to delete warning modal to user.

- **Class Bugs -Method getBugs**

  It creates a get request to bug/all url. It puts a user's token in request. Then it catches the response of server, response of web server is a list of all bugs that saved in systems. Then it assigns the items to this bug list in json form. Then it extracts the keys of json form. Then it gives to this value to getKeysSetFilter methods by calling it. Then it toggles the busy state by calling toggleBusy method. Otherwise, if server message is error response, makeToast methods is called by giving error message of server.

- **Class Bugs -Method getKeysSetFilter**

  It takes the one argument called keys then it addes the key values to fields list in object form. This is for dynamic rendering of filtering options.

- **Class Bugs -Method deleteBug**

  This method takes the bugid and index of user that in bug list in client from bugwantedtodeleted object. Then it creates delete request to bug/delete/[bugid] URL. It puts a user's token in request. Then it catches the response of server, if message is success request, makeToast methods is called by giving success message of server. Then bug that is deleted, is popped from client's bug list too. Otherwise, if server message is error response, makeToast methods is called by giving error message of server.

- **Class Bugs -Method Onfiltered**

  It triggers pagination to update the number of buttons/pages due to filtering.

- **Class Bugs -Method makeToast**

  This method creates a toaster according to arguments that are called type, variant and toastmessage. It renders a toaster component with error or success variant. And it gives a given message to toaster.

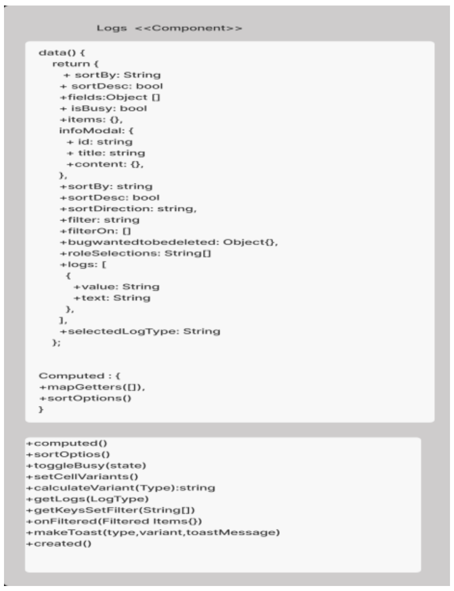
- **Class Users -Method created:**

  This method is triggered on the created part in vuex system. It calls the getBugs methods for getting bugs and list them into web browser. When page is created this function will be triggered to collect bugs from server.

4.3.2.13 Logs <<Component>> Class

logs.vue is created for handling logs information that in system with javascript and rendering related html in that part. It manipulates the logs information by reading them.

The class object belonging to the UML design is shown in the **Figure 113**.



```
Logs <<Component>>

data() {
   return {
      + sortBy: String
     + sortDesc: bool
     +fields:Object []
     + isBusy: bool
     +items: {},
     infoModal: {
       + id: string
       + title: string
       +content: {},
     },
     +sortBy: string
     +sortDesc: bool
     +sortDirection: string,
     +filter: string
     +filterOn: []
     +bugwantedtobedeleted: Object{},
     +roleSelections: String[]
     +logs: [
       {
         +value: String
         +text: String
       },
     ],
     +selectedLogType: String
   };


Computed : {
+mapGetters([]),
+sortOptions()
}

+computed()
+sortOptios()
+toggleBusy(state)
+setCellVariants()
+calculateVariant(Type):string
+getLogs(LogType)
+getKeysSetFilter(String[])
+onFiltered(Filtered Items{})
+makeToast(type,variant,toastMessage)
+created()
```

Figure 113: Logs Component Class

- **Class Logs -Method computed**

  This method is triggered on the computed part in vuex system. With the help of that, vuex store management system’s states get the related getters about user states. And calls sortOptions methods while in computing state of vue.

- **Class Logs -Method sortOptions**

  This method creates the sort labels and their key values for understanding any change occurs in fields. When data are changed which means that computing is occurred, this method is triggered.

- **Class Logs -Method toggleBusy**

  This method is created for showing loading icon or not. It takes the reverse of Boolean of isBusy state when it is triggered.

- **Class Logs -Method getLogs**

  It takes a logType for defining the log type such as user,record and bug, It creates a get request to [logType]/logs url. It puts a user's token in request. Then it catches the response of server, response of web server is a list of all log type(generic) that saved in systems. Then it assigns the items to this log type(generic) list in json form. Then it extracts the keys of json form. Then it gives to this value to getKeysSetFilter methods by calling it. Then it toggles the busy state by calling toggleBusy method. Otherwise, if server message is error response, makeToast methods is called by giving error message of server.

- **Class Logs -Method calculateVariant**

  This method is created for defining log type variant as primary,warning and danger. This for rendering the log type(update,create,delete) in yellow,blue and red colors.

- **Class Logs -Method setCellVariants**

  This method is created for updating log object to define a color of log activities such as delete, update and create. It calls calculateVariant methods for defining it. After that it adds log object to items list in object form.

- **Class Bugs -Method getKeysSetFilter**

  It takes the one argument called keys then it addes the key values to fields list in object form. This is for dynamic rendering of filtering options.

- **Class Bugs -Method Onfiltered**

  It triggers pagination to update the number of buttons/pages due to filtering.

- **Class Bugs -Method makeToast**

  This method creates a toaster according to arguments that are called type, variant and toastmessage. It renders a toaster component with error or success variant. And it gives a given message to toaster.

- **Class Users -Method created**

  This method is triggered on the created part in vuex system. It calls the getLogs methods for getting logs and list them into web browser. When page is created this function will be triggered to collect logs from server.

4.3.2.14 Error <<Component>> Class

error.vue is created for showing user is not directed the page that system does not have and rendering related html in that part.

The class object belonging to the UML design is shown in the **Figure 114**.

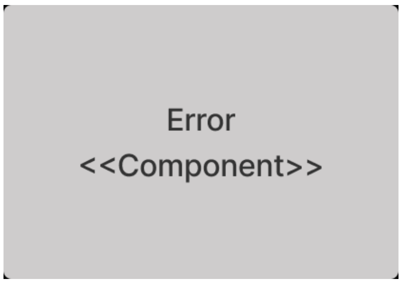
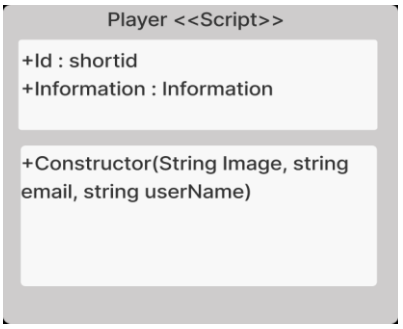


Figure 114: Error component Class

4.3.3 Online User Server Subsystem Classes

4.3.3.1 Player <<Script>> Class

Player.js is created for handling user's information that in system with It manipulates creting the user instance of user that connected to system. The class object belonging to the UML design is shown in the **Figure 115**.


Player <<Script>>
+Id : shortid
+Information : Information
+Constructor(String Image, string email, string userName)
Figure 115: Player Script Class

- **Class profile -Method Constructor**

This method is for creating a new Player class. Also, it exports the Player class to index.js

4.3.3.2 Information << Script >> Class

Information.js is created for storing user's information that in system with javascript. It is used by Player.js. The class object belonging to the UML design is shown in the **Figure 116**.

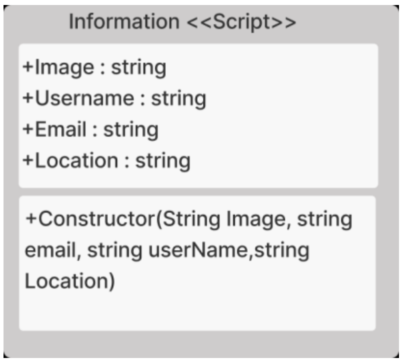


Figure 116: Information Script Class

- **Class Information -Method Constructor:**

  This method is for handling a new Player’s information status. Also, it exports the Information class to Player.js.

4.3.3.3 Index <<Component>> Class

Index.js is created for starting system with javascript and handling user’s connection. The class object belonging to the UML design is shown in the **Figure 117**.

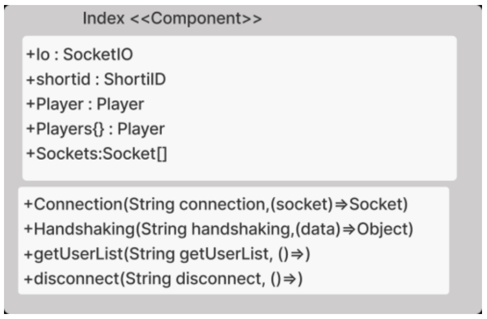

Index <<Component>>
+Io : SocketIO
+shortid : ShortiID
+Player : Player
+Players{} : Player
+Sockets:Socket[]
+Connection(String connection,(socket)⇒Socket)
+Handshaking(String handshaking,(data)⇒Object)
+getUserList(String getUserList, ()⇒)
+disconnect(String disconnect, ()⇒)

Figure 117: Index Component Class

- **Class profile -Method Connection**

  This method is for understand new connection is arrived at system. After that, it creates a new player instance. Then generated id is assigned to thisPlayerID for future operations.

- **Class profile -Method Handshaking**

  This method is for getting new user that is connected the system just now, information such as username, email, image and location information. After that, it adds this player to players and sockets list by using player id. Then server broadcast the connected player information to other clients also system sends the userlist to new connected player.

- **Class profile -Method getUserList**

  When this socket is triggered, players list is sent to user that triggers the socket.

- **Class profile -Method disconnect**

  When a client is disconnected from server, this method methods understand that then it deletes the user from players and socket list by using own id. After that, disconnected player’s information is sent to other clients by using broadcasting system.

5. Testing Design

- System will be tested to show that it can support multiple users at the same time.
- The accuracy of the landing, take-off, right and left turn movements of the specified drones will be tested.
- Login test checks the empty, false and tokenless operations for the authentication process. This test applied to both systems of Cyber Drone project which are unity and web part.
- Menu test checks the menu operations working properly such as when user clicks on exit button, software closes itself.
- Settings test checks the settings operations working properly such as when user changes the sound volume on changing value of music slider, then software sets the new music volume.

- For the web part of project, user operations are tested. This test contains creating new password, changing password, adding new user, deleting existing user, showing user's information. For example, changing new password test, all the cases are examined like new passwords are matching or these values are empty and nullable fields are exist. Other operations that mentioned above are examined like that.
- For the simulation part of project, the general drone movement, collision, camera movement, obstacles and platform tests are executed. For example, we checked each objects collider and tested each one of them on the game screen. For the platforms, platform movement script analyzed with different values and observed if they collide with other objects or platforms. Each pickup object tested by our drones for if their colliders were working properly. Camera movement and drone movement scripts were tested by different values (tilting values, speed, angles, the distance of camera, etc.) on the game screen.

References

[1] D. Team, "#DronoMatrix RSD Final," İzmir, 2021.

[2] #DronoMatrix, "#DronoMatrix UML class diagram," İzmir, 2021.

[3] T. Hietanen, "Eşler arası çok oyunculu oyun nasıl yapabilirim?," qastack, [Online]. Available: https://qastack.info.tr/gamedev/3887/how-can-i-make-a-peer-to-peer-multiplayer-game.

[4] V. Mallawaarachchi, "10 Common Software Architectural Patterns in a nutshell," towards data scince, 4 september 2017. [Online]. Available: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013.

[5] G. Fiedler, "What Every Programmer Needs to know about game networking," wayback machine, 24 february 2010. [Online]. Available: https://gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking/.

[6] O. Bulut, "Model View Controler," medium, 15 december 2020. [Online]. Available: https://medium.com/@omerbulut7878/mvc-model-view-controller-4d91ff1c1892.

CyberDroneUMLDiagrams.pdf CyberDroneRSDFinal.docx