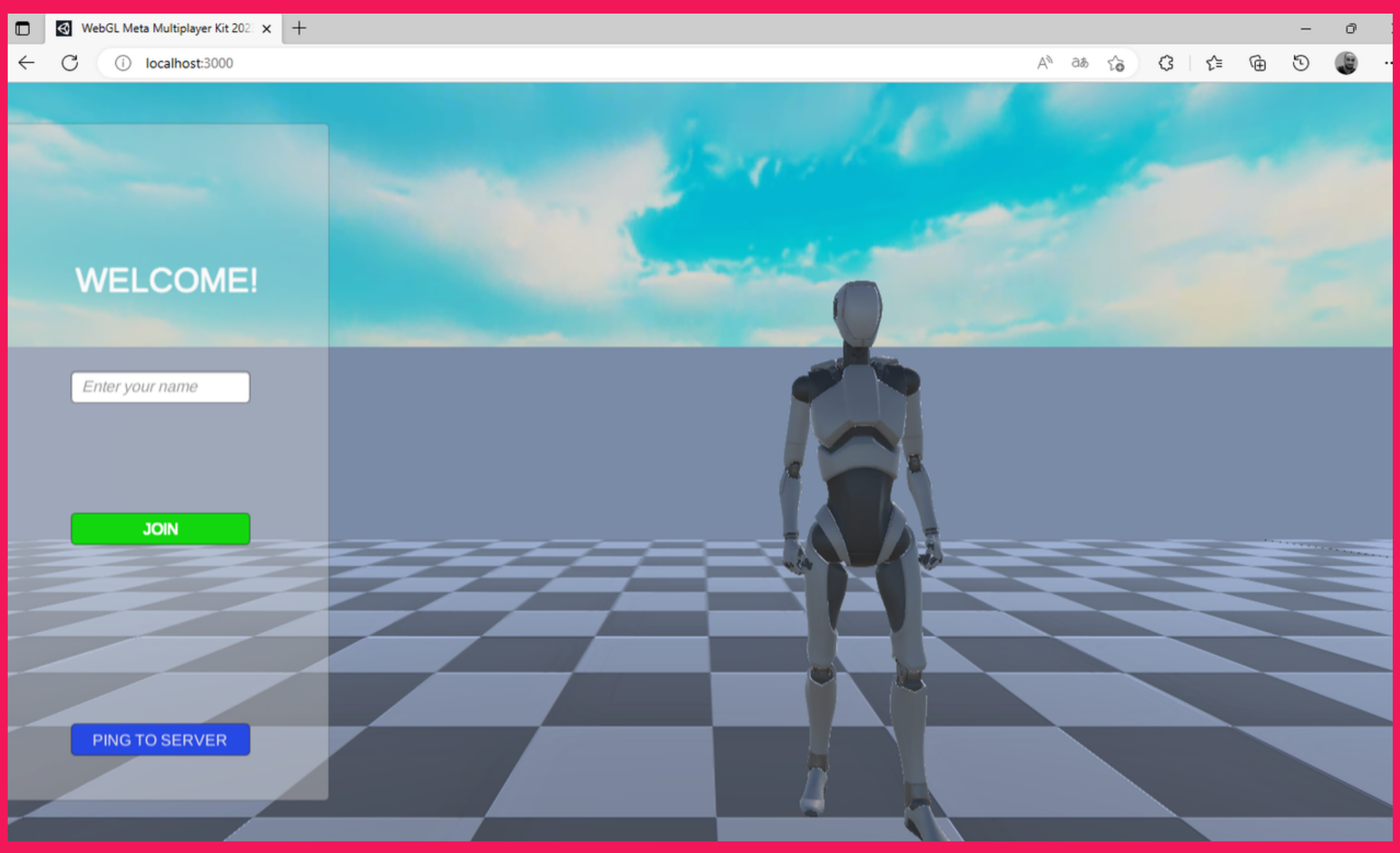
DOCUMENTATION

WEBGL META MULTIPLAYER KIT

RIO 3D STUDIOS



a guide to create your first online multiplayer game

learn step by step to develop your game!

SUMMARY

QUICK START

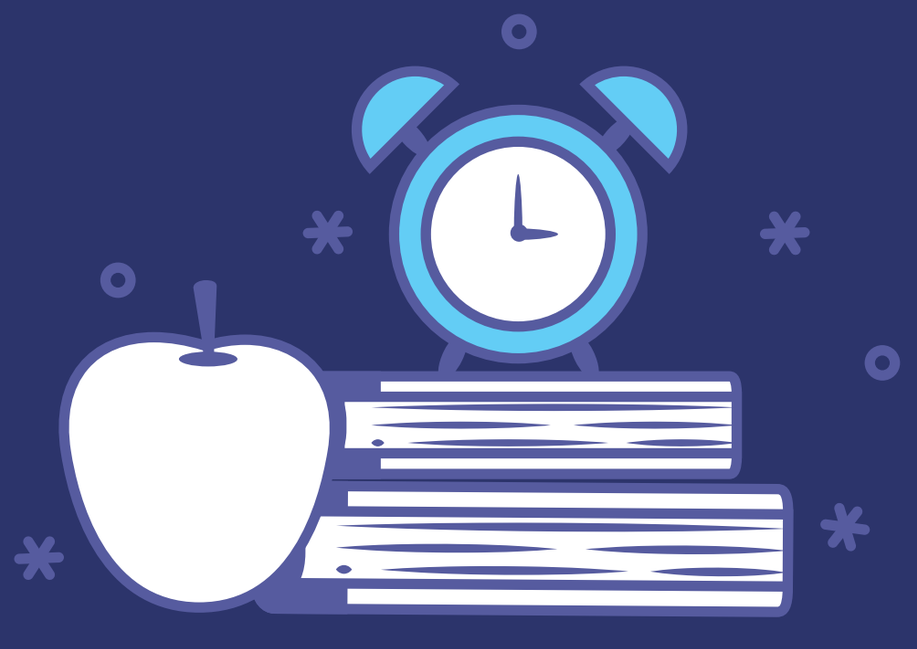
- setup webgl game
- setup node js server
- publish your game

HOW TO DEVELOP

- Game architecture
- Network Manager Class
- Node JS Server
- Voice Chat

USAGE SAMPLES

CONTACT & SUPPORT



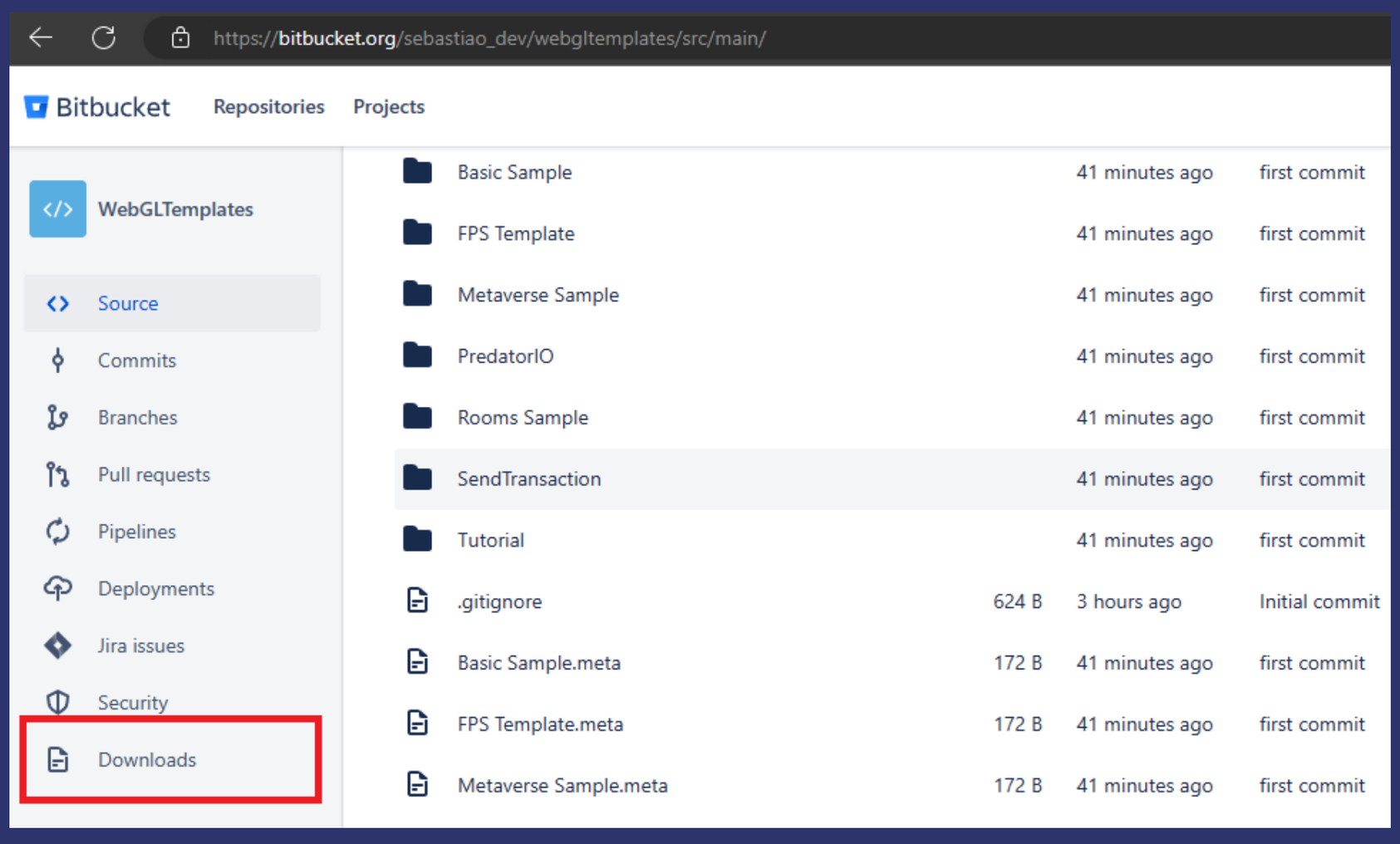
QUICK START

SETUP WEBGL GAME

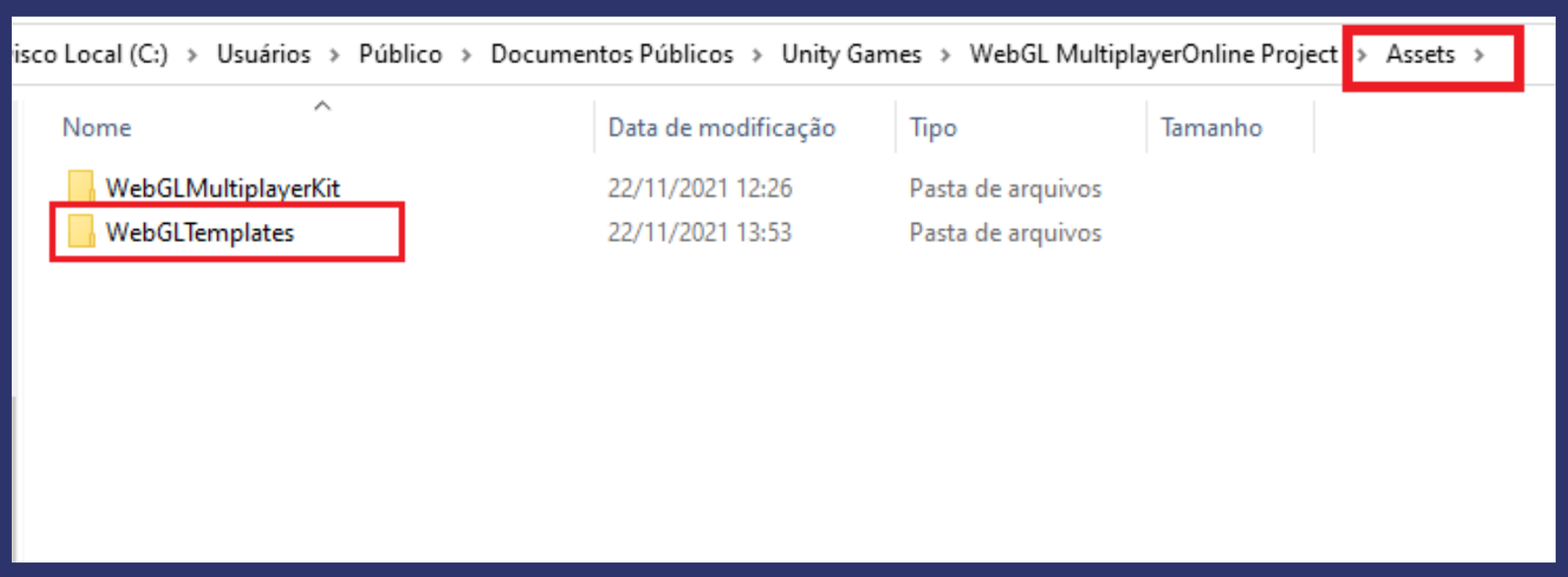
This basic project will allow you to build a WebGL HTML5 multiplayer online games for browsers like Google Chrome , firefox, etc. using the external calls (Application.ExternalCall()) provided by the unity.
This mechanism allows a unity WebGL html5 client to communicate externally with a java script server running in NodeJS.

Before starting it is recommended to download the WebGLTemplates from the our page on github. These templates have pre-configured the **index.html** file and the **client.js** file, which are essential for the proper functioning of each example present in the WebGL Multiplayer Kit.
To download the WebGLTemplates folder go to our page on bitbucket and download

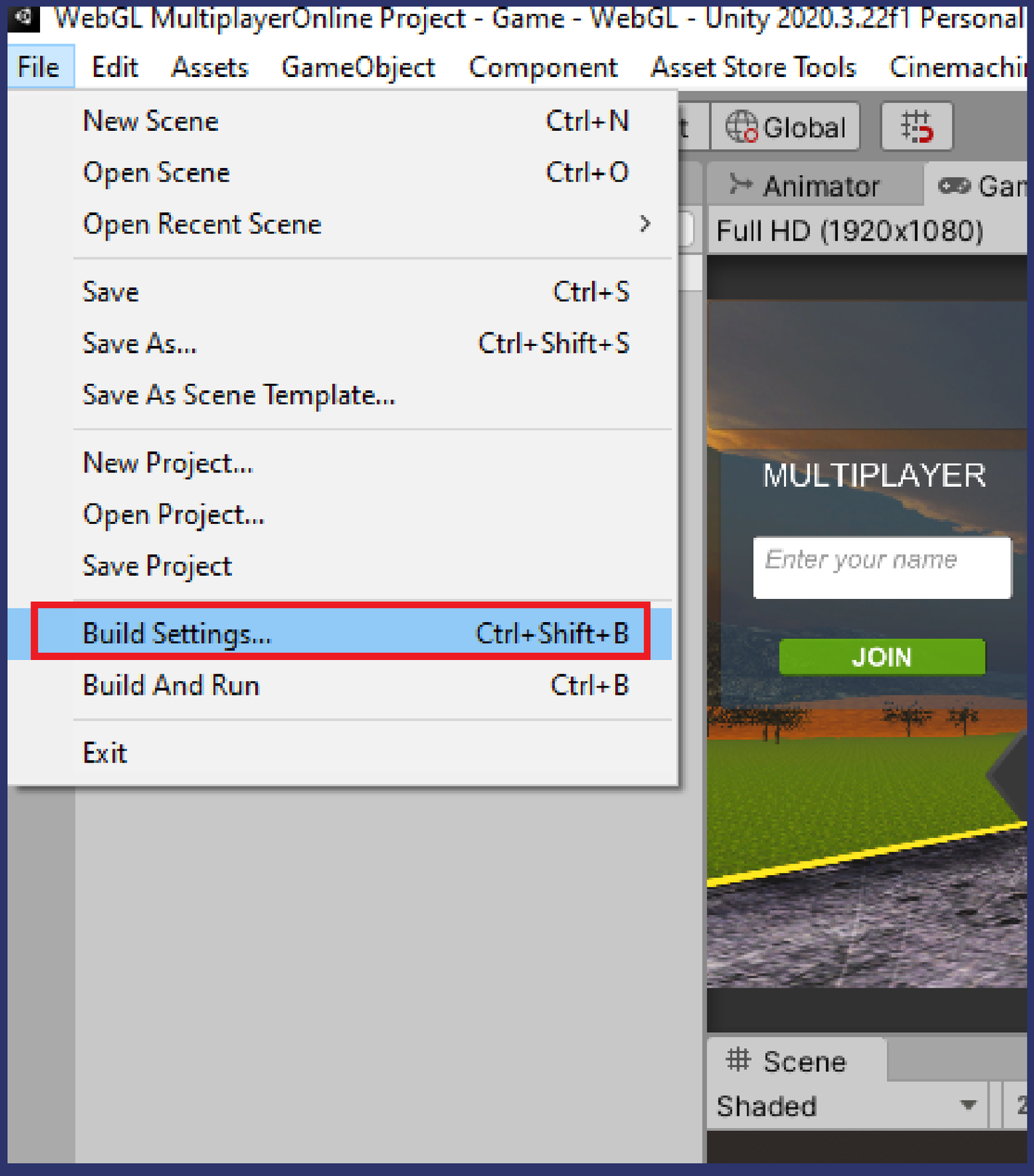
Caution: Rename the folder to "**WebGLTemplates**" otherwise unity will not recognize any templates!

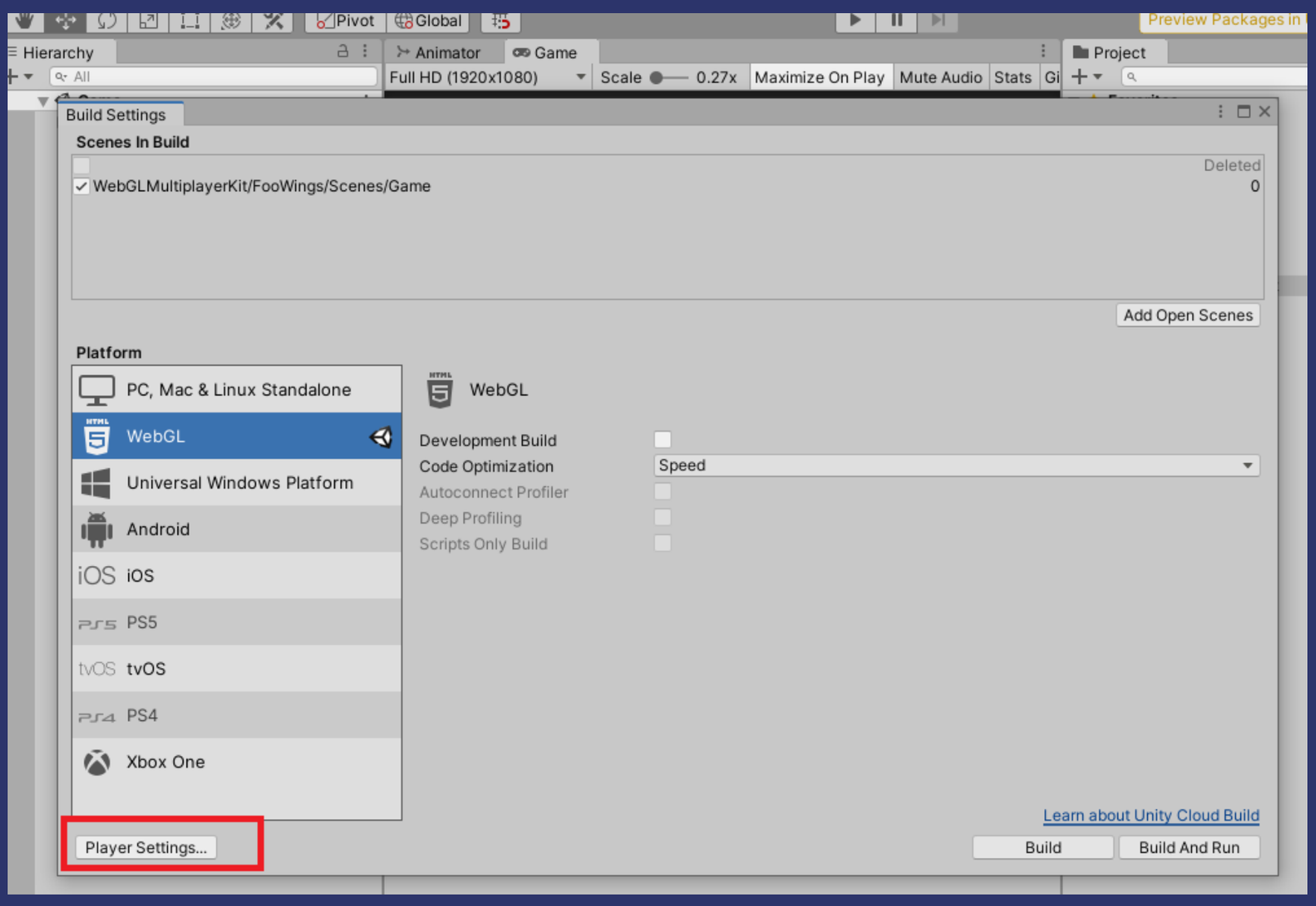


after downloading the WebGLTemplates folder, extract the folder and place it inside the **assets** folder of the WebGL Meta Multiplayer Kit project as shown in the figure below

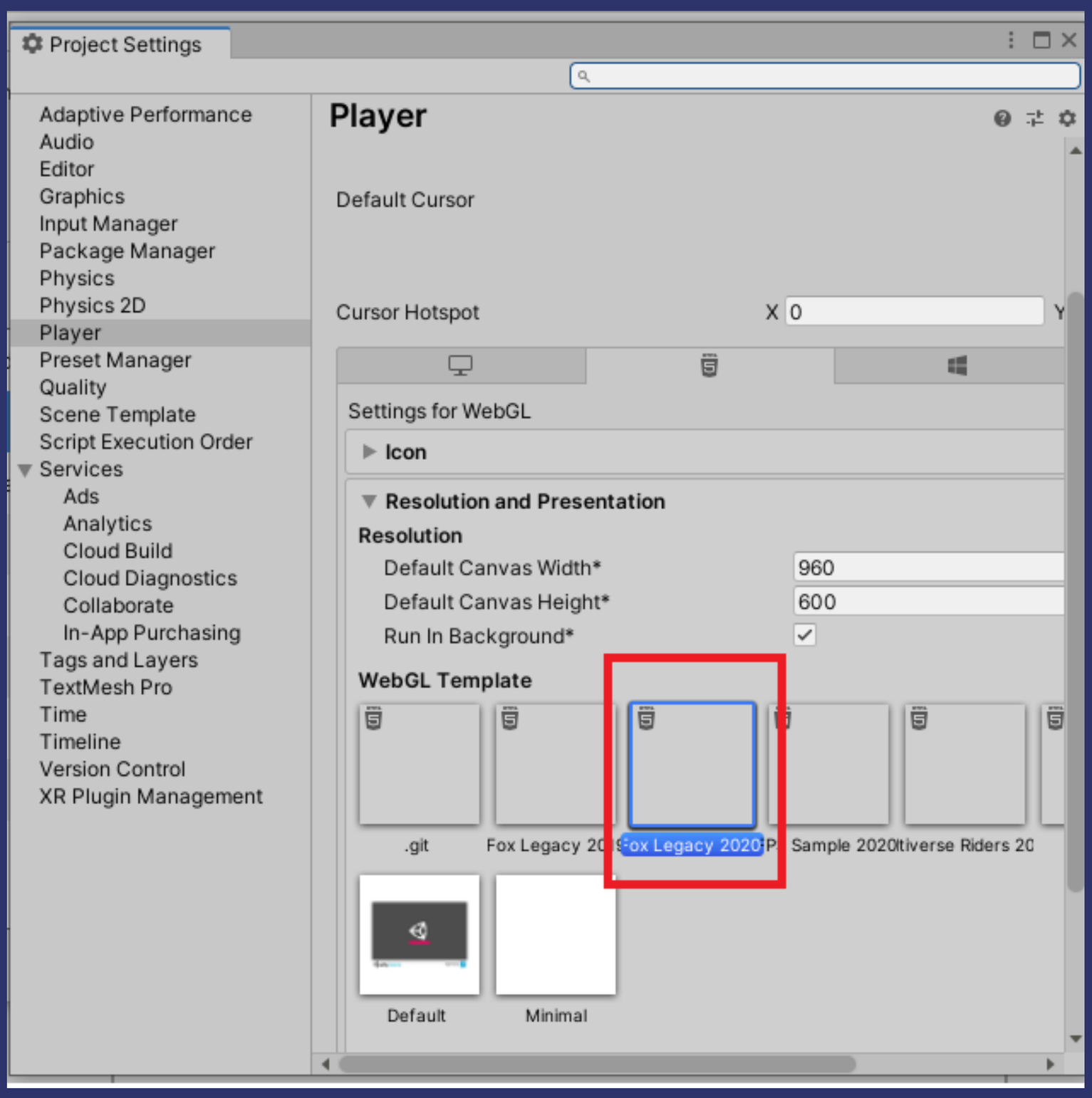


note that inside the WebGLTemplates folder there is a specific template for each example of the asset. The **index.html** file is the same for all examples, but the **client.js** file is specific for each example.
To apply a template to a given project, return to the Unity editor and open the **Player Settings** option:

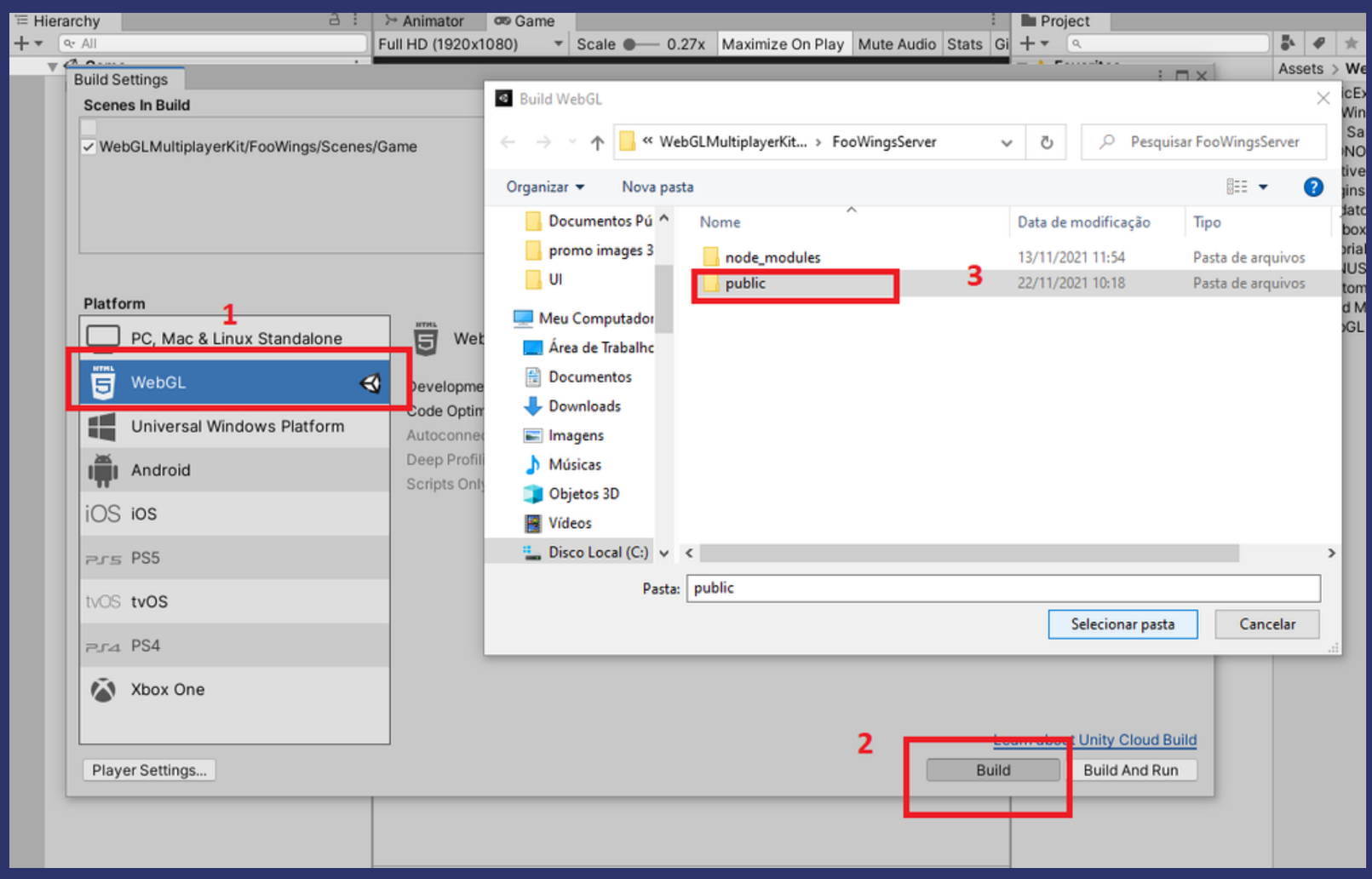




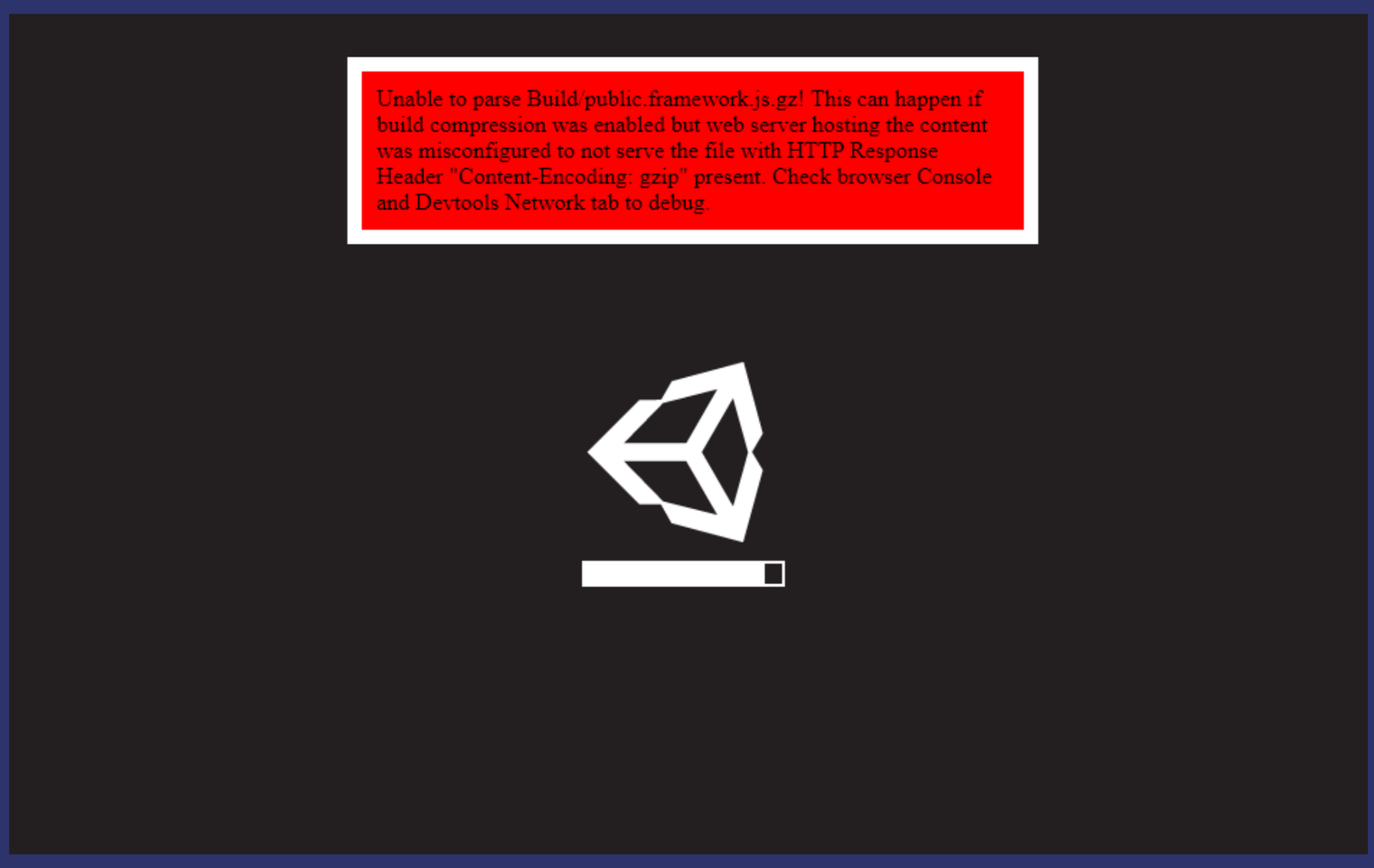
in **Resolution and Presentation** choose the desired template for your project



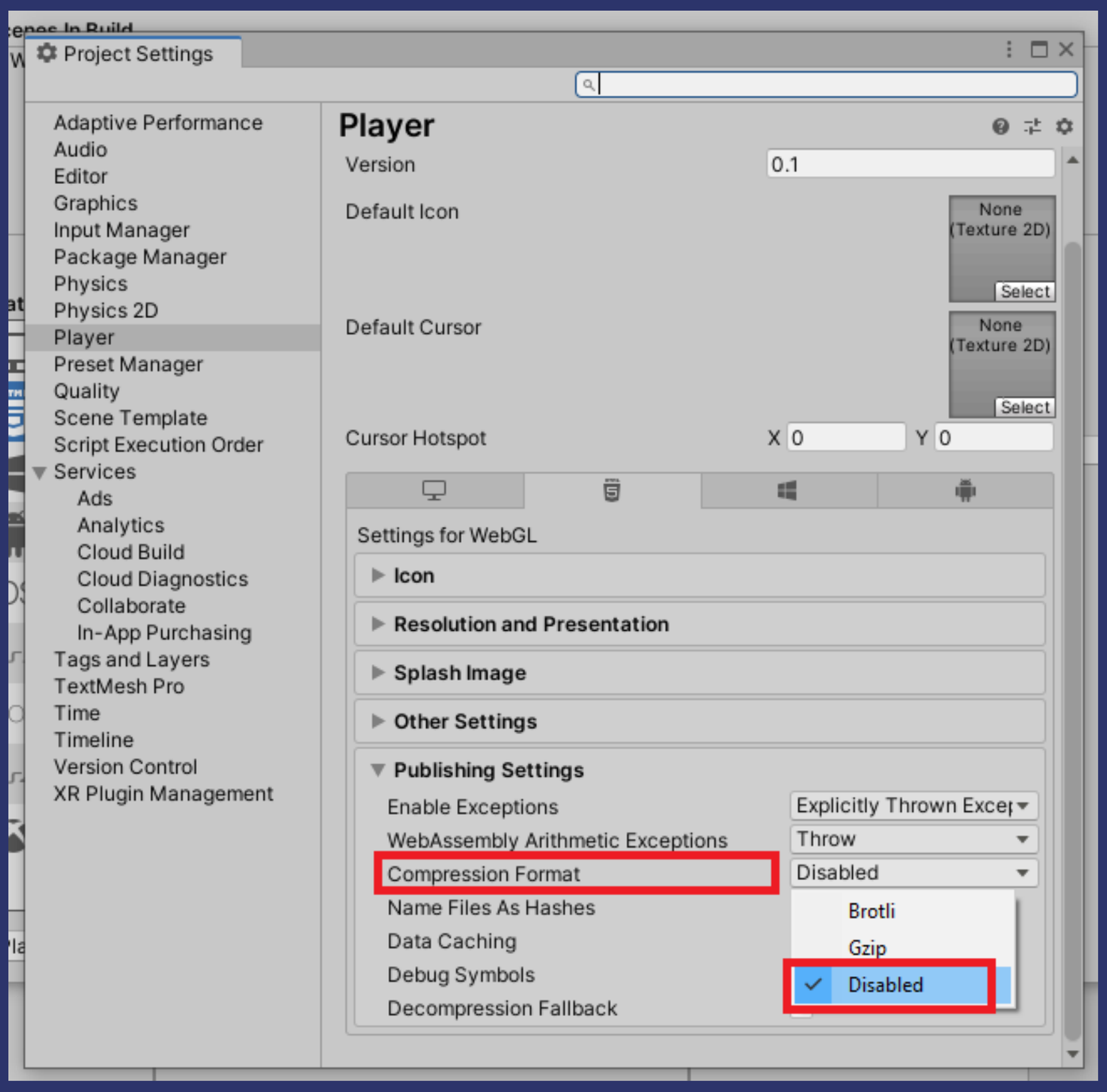
After choosing the template, export the project for the WebGL platform in a folder called "public" located somewhere in your computer (Desktop e.g.)



known ussues

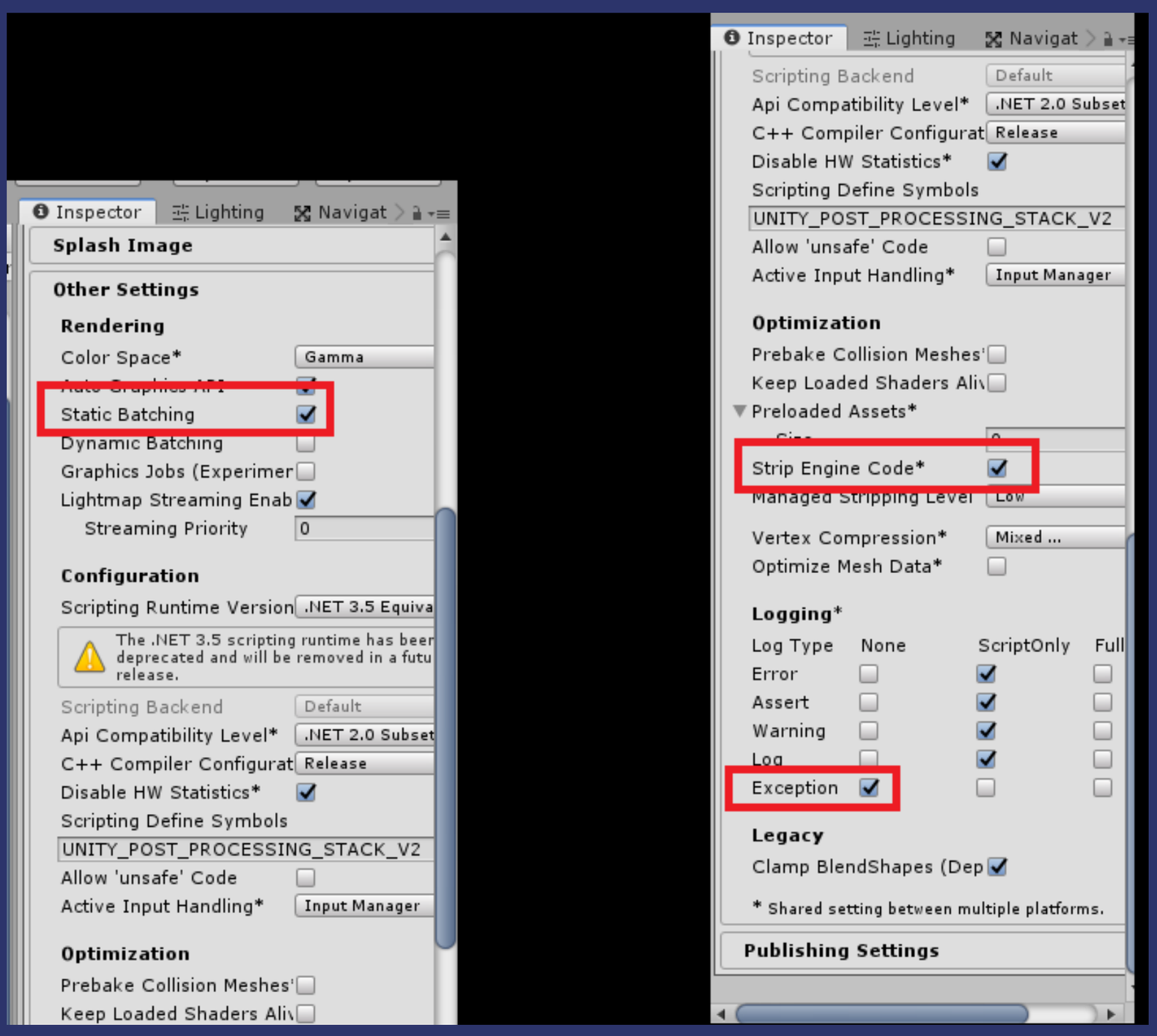


solution

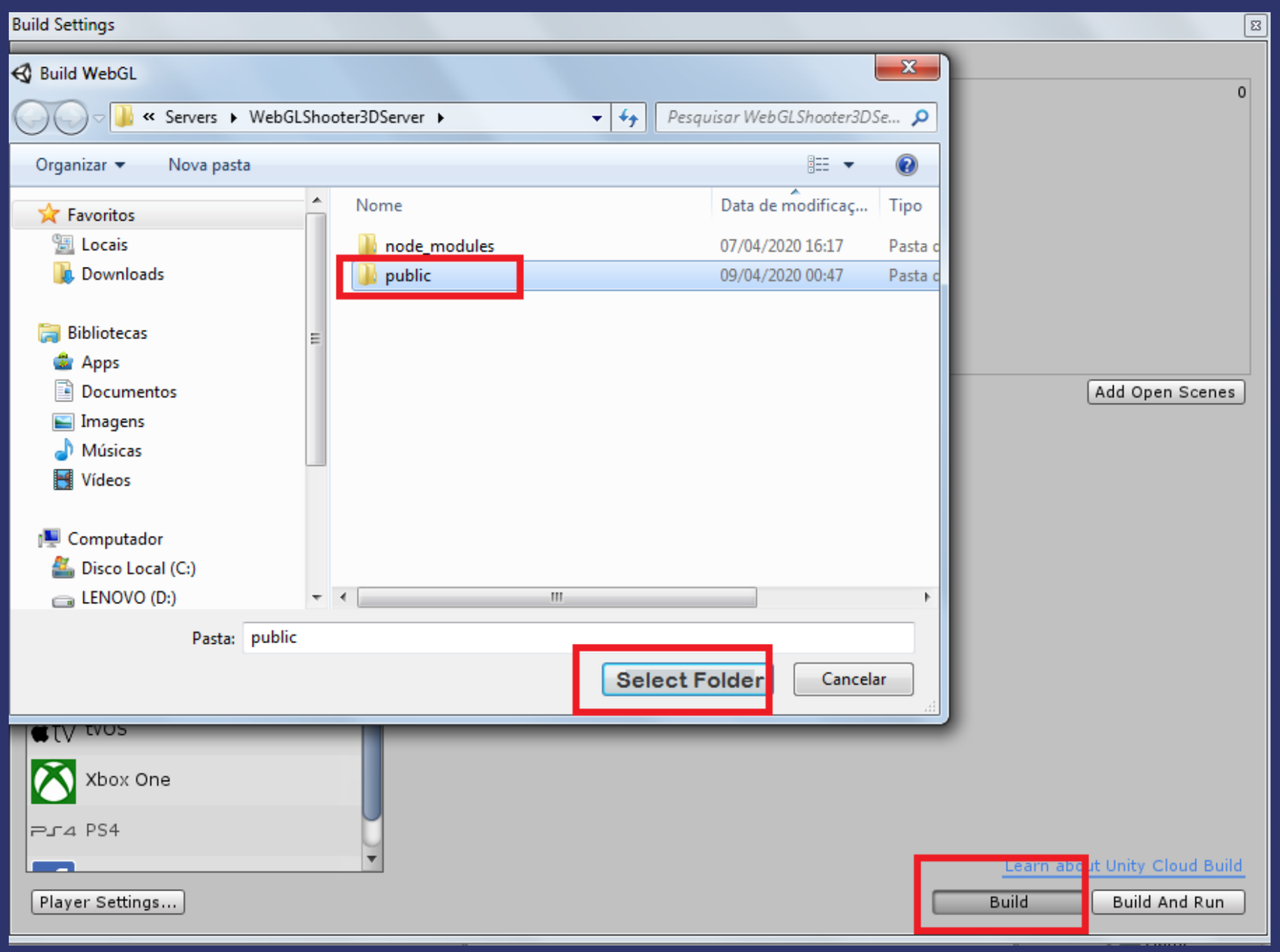


Optimization for WebGL

To make the game work faster in browsers, we recommend that you follow the steps below. In the Player Settings Inspector tab, go to the Other Settings option and check the options: static batching, Strip Engine Code and under Logging mark the Exception option as none.



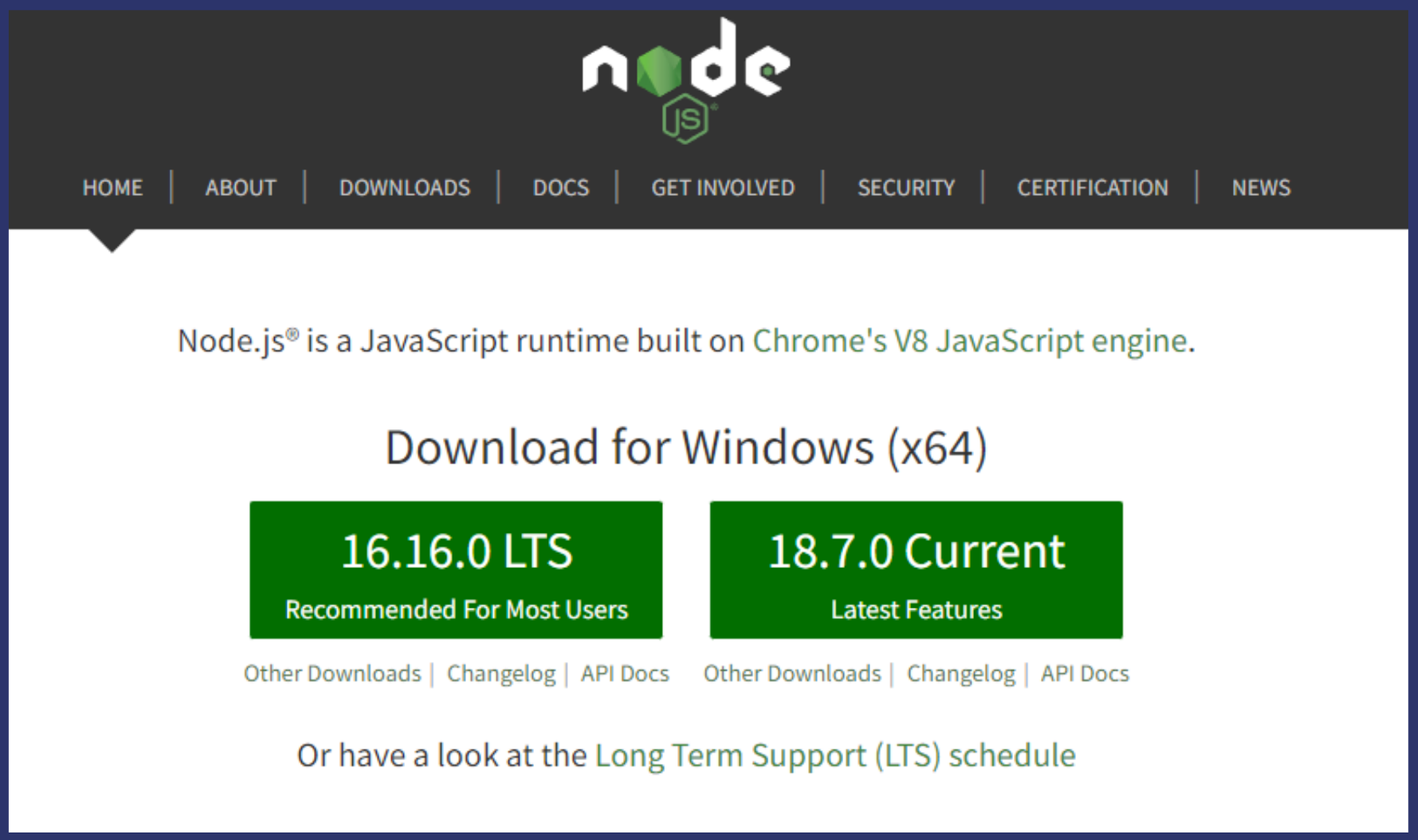
In build settings, go to build to compile a new version of the game:



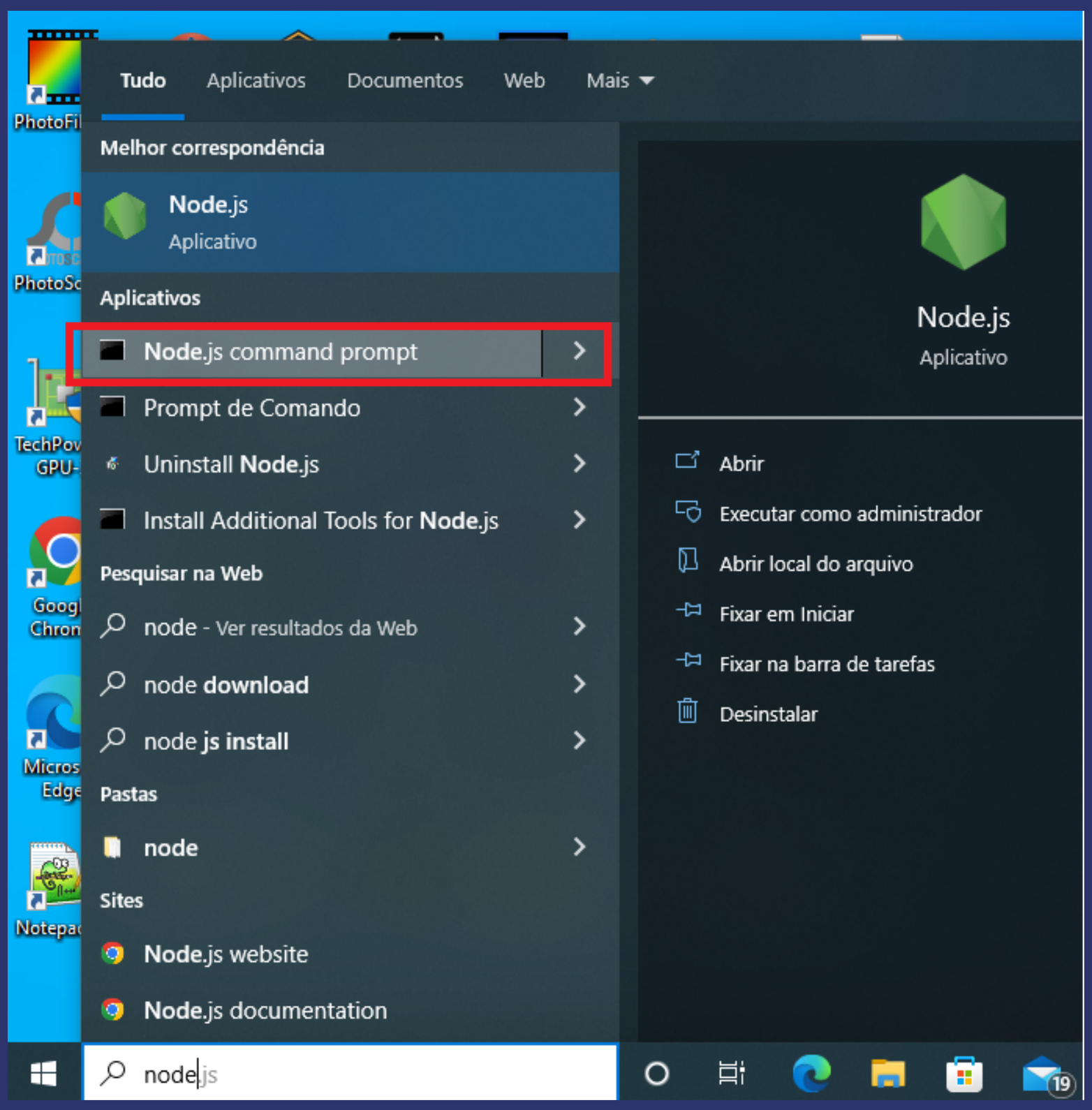
SETUP NODEJS SERVER

After configuring the content of the file (index.html) in the public folder in the previous steps, we should make the installation of NodeJS.
NodeJS is an interpreter java script runtime that acts of the server side.

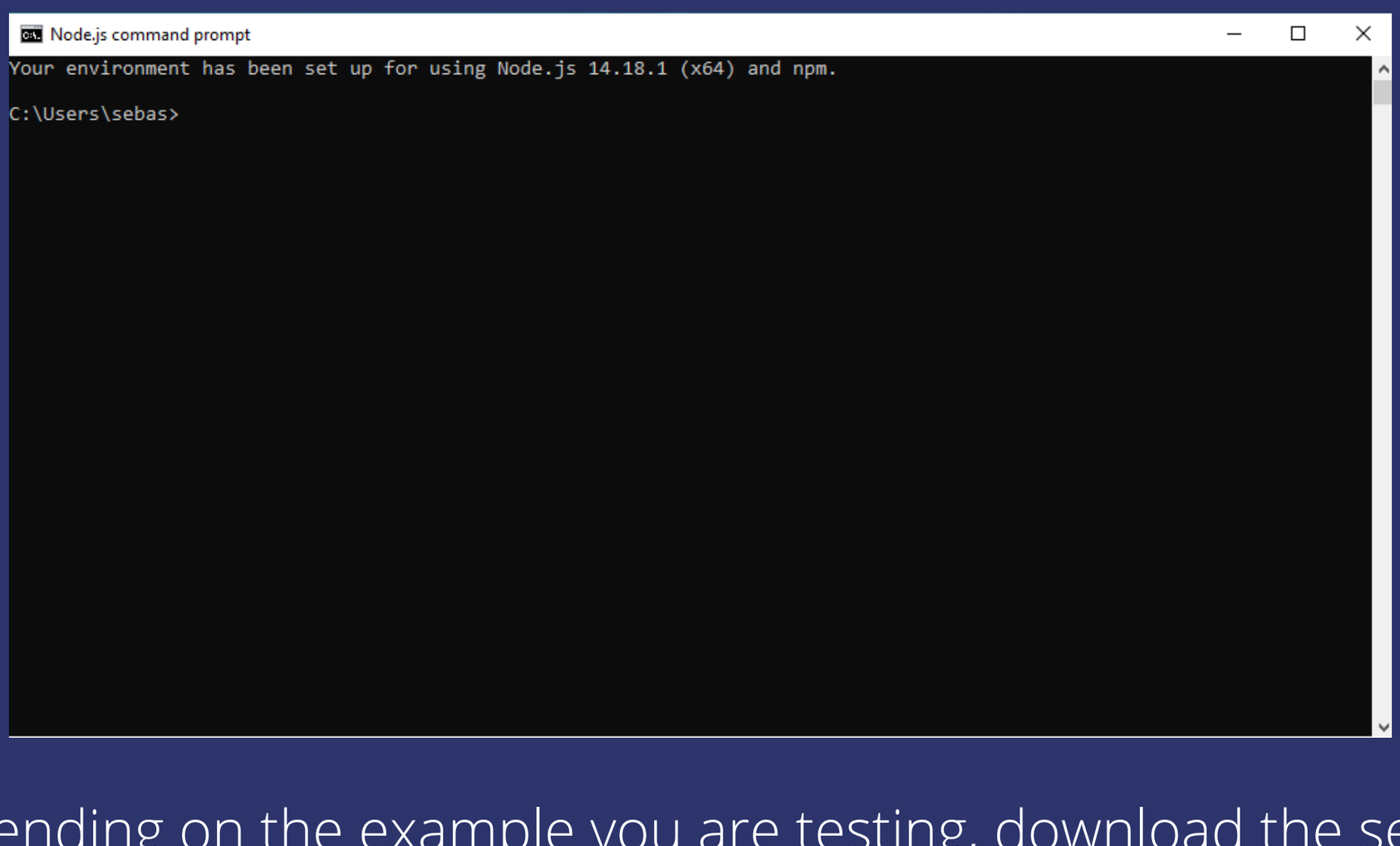
Download NodeJS



To execute Node.JS in windows , go to the windows toolbar and type in the search field : " node" , selects the Node.JS command prompt .



If the following screen be exhibited the NodeJS was installed correctly:



Depending on the example you are testing, download the server folder from the corresponding github project:

Tutorial Server

Send Transaction Server

Metaverse Server

Rooms Sample Server

Basic Sample Server

FPS Server

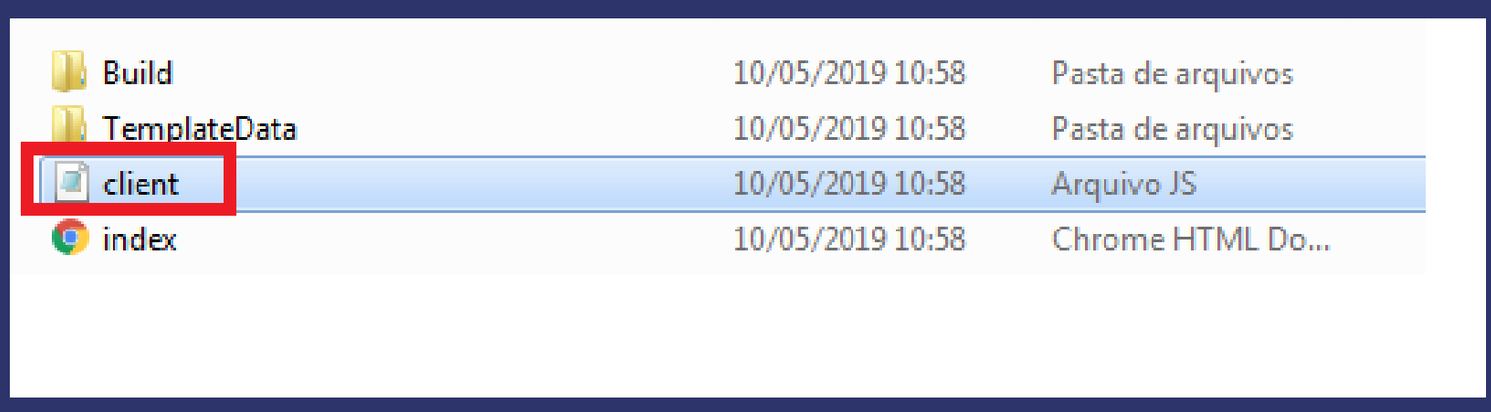
PredatorIO Server

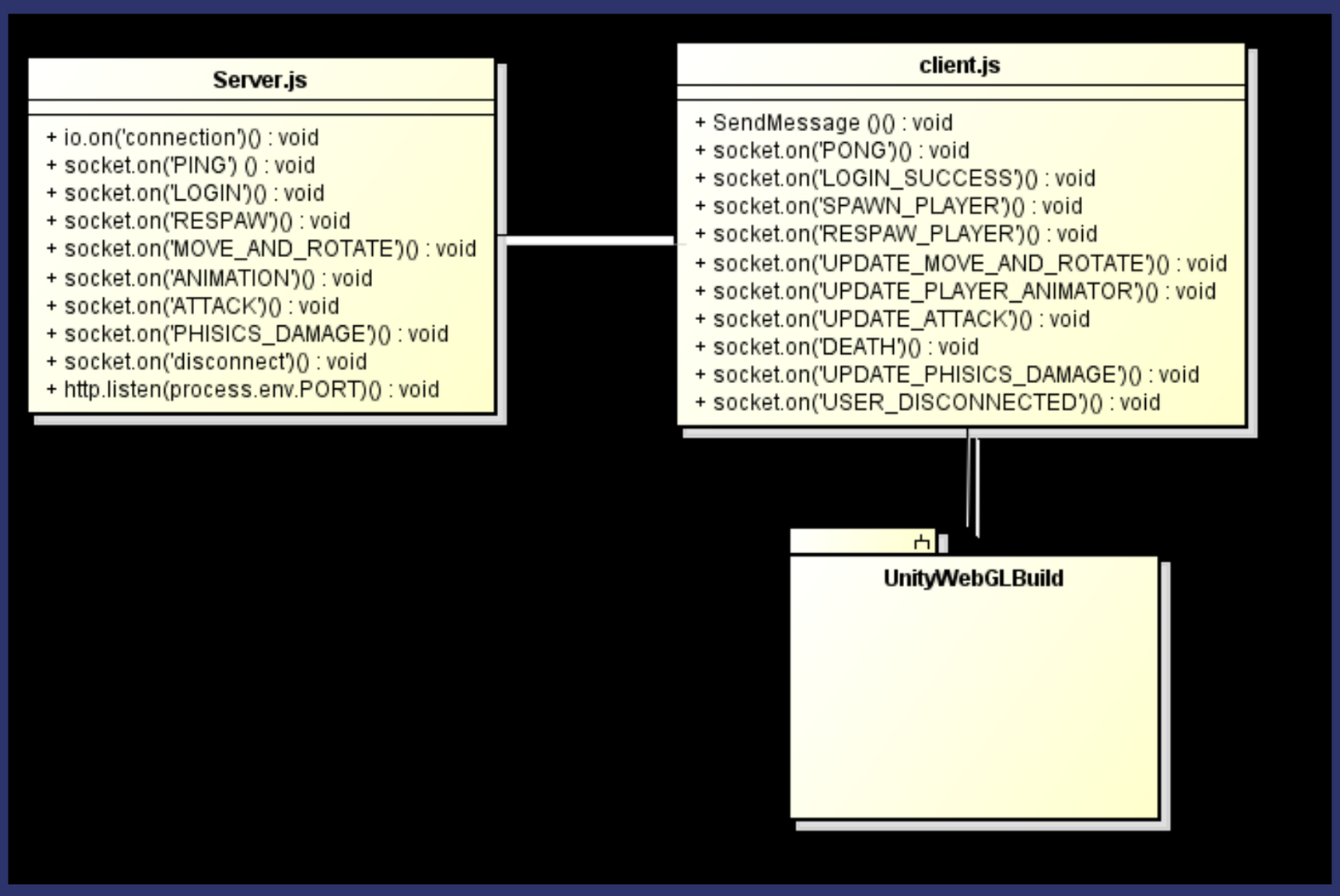
The client.js file

Note that along with the server.js script are included in the "public" folder, the project compiled in WebGL by Unity. This is the same project that you just compiled following this tutorial. The only difference is that a client.js file has been included which will explain its function below.

To communicate the server in nodeJS with the functions created in the WebGL client in unity, we need an intermediate client.js file.

The client.js script will be responsible for the communication among the java script server.js and the functions in the script NetworkManager.cs in Unity project.

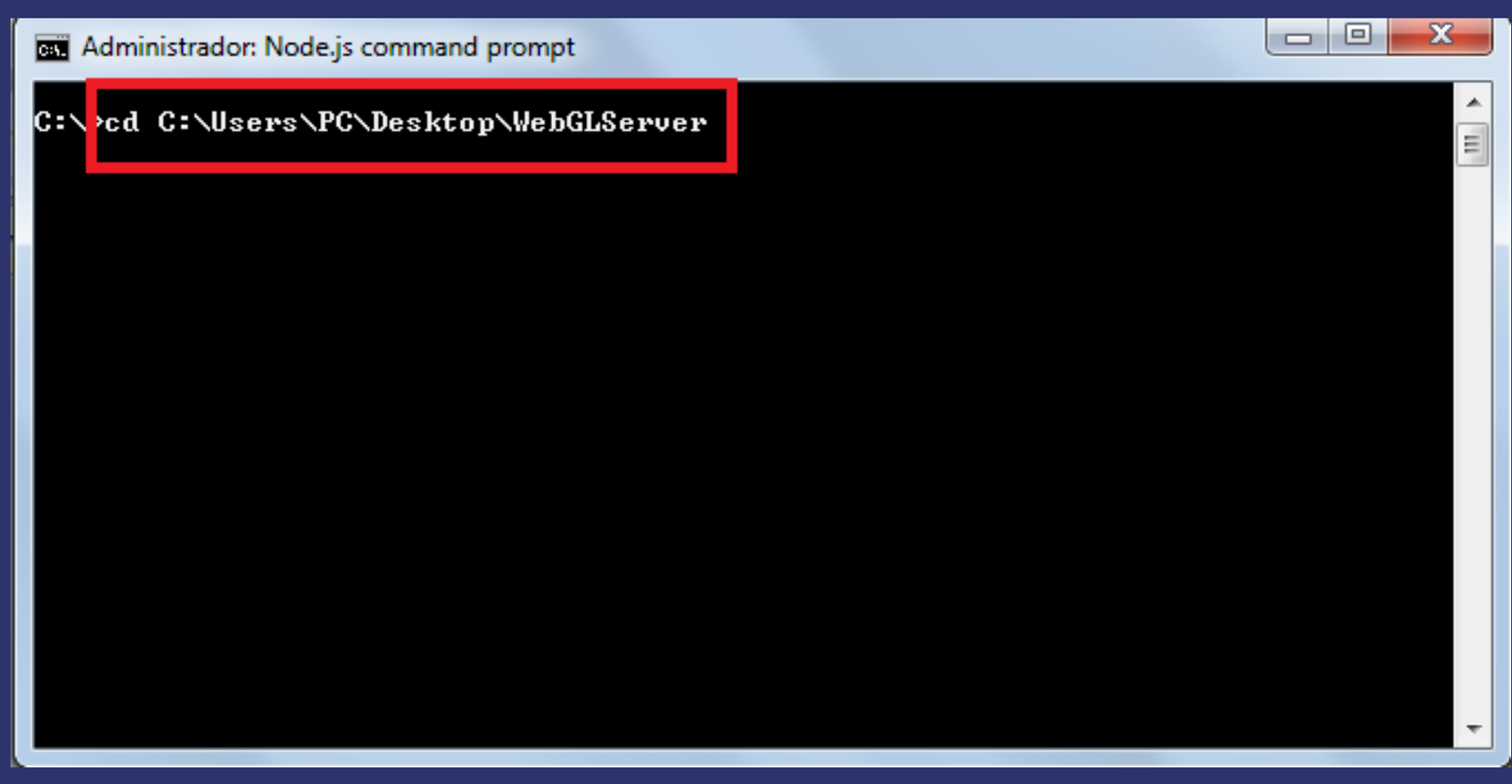




Running the server on a local machine

Select the Node.JS command prompt as in the image above and:

1. Navigate until the WebGLServer folder typing the command (cd + space_bar + url_folder)
eg: **cd** C:\Users\PC\Desktop\WebGLServer



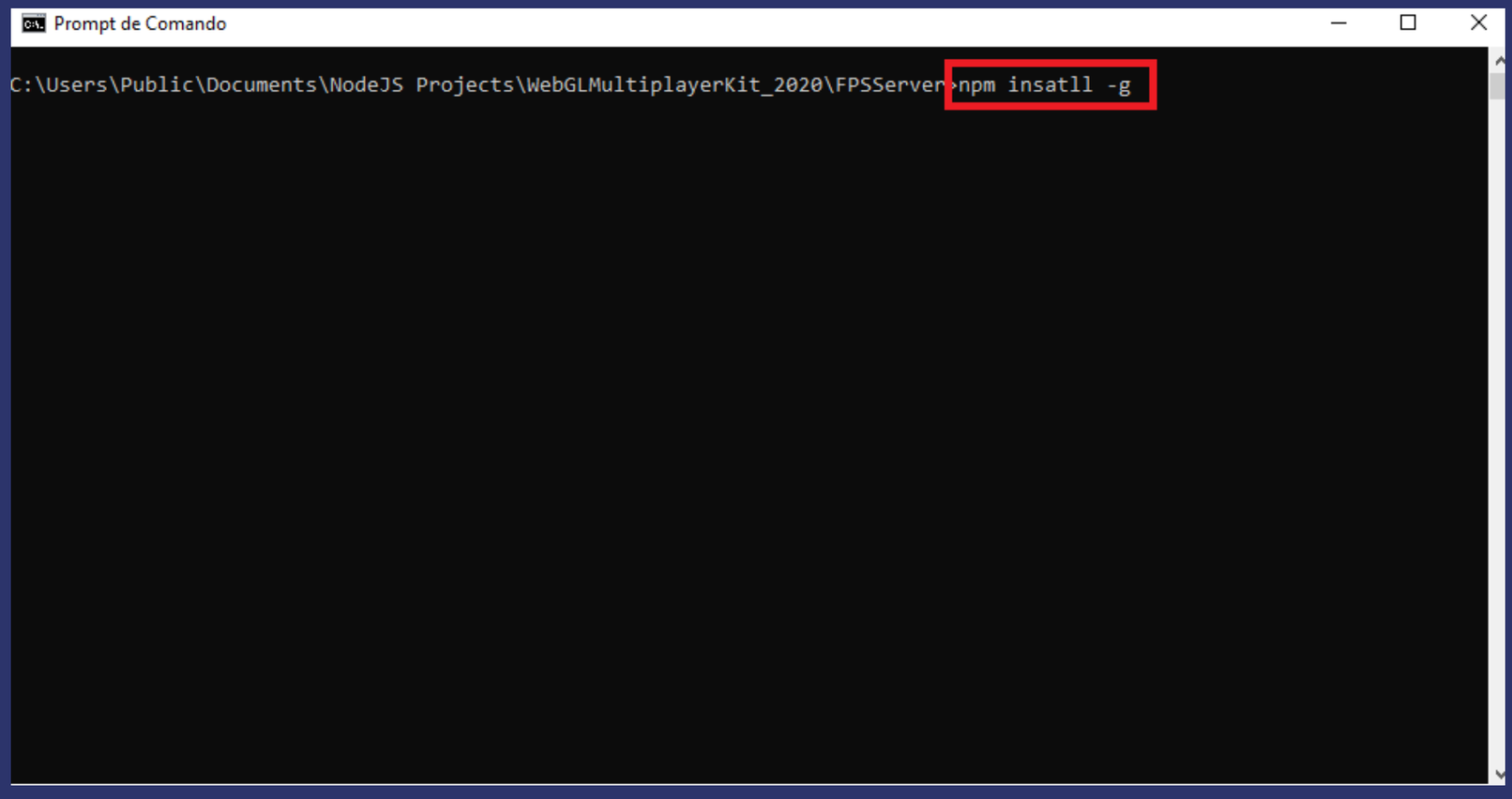
Soon afterwards we should install the necessary modules for the operation of the java script server.js through command npm install.

We will Install the modules: express(NodeJS Framework), socketio (sockets lib) and shortid (random numbers generator)

Select the Node.JS command prompt as in the image above and:

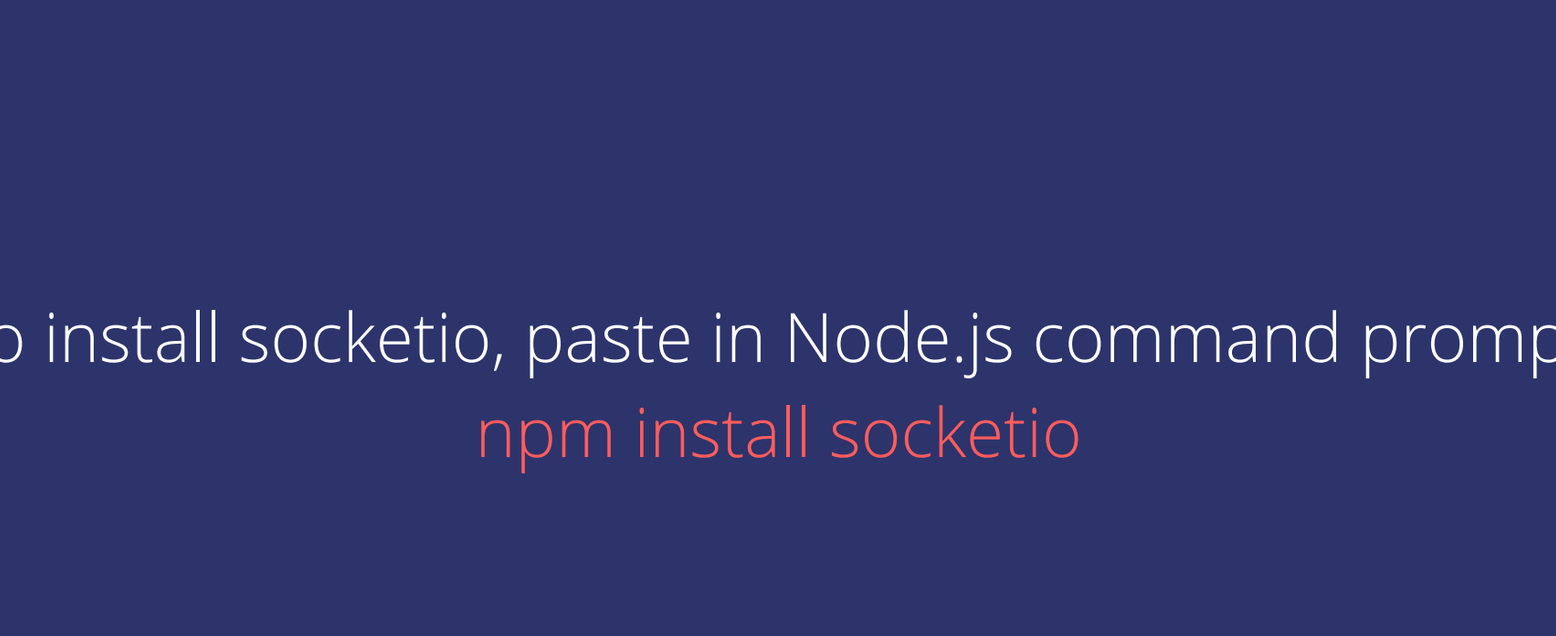
To install all the necessary nodejs libraries just type at the windows command prompt:

npm install -g

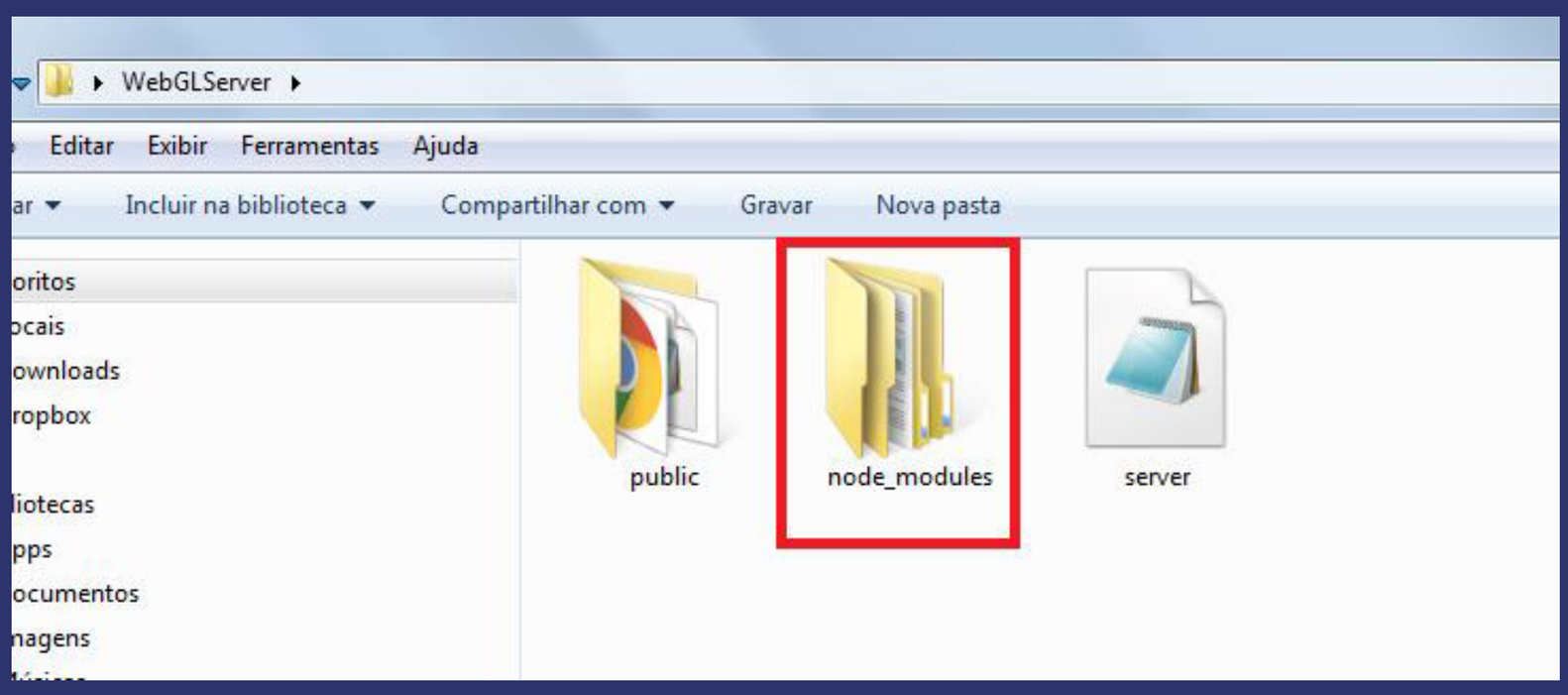


To install socketio, paste in Node.js command prompt:

npm install socketio

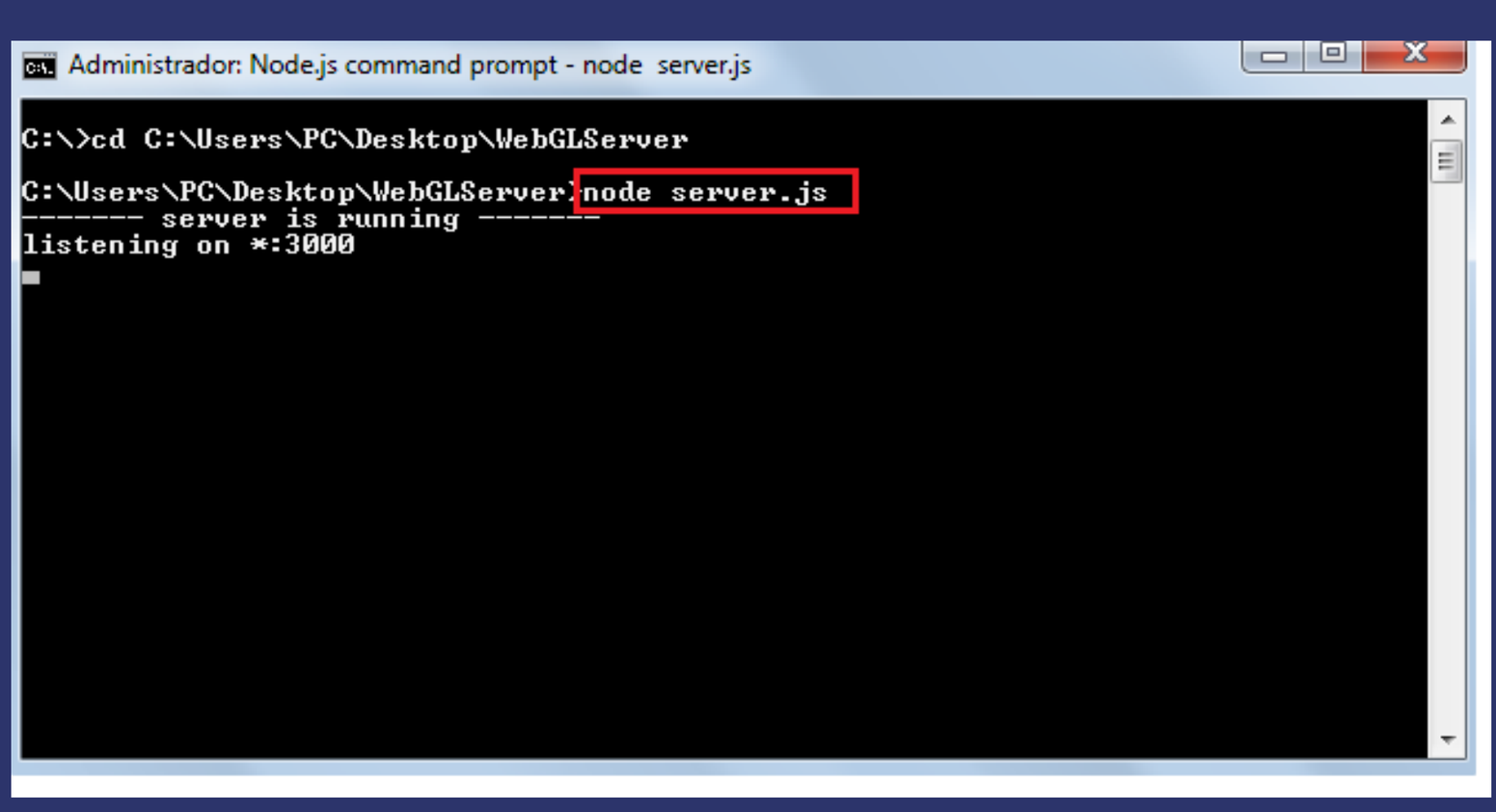


Notice that will go appears a new folder denominated node_modules inside of the WebGLServer folder, inside you will find the modules express, socketio and shortid.



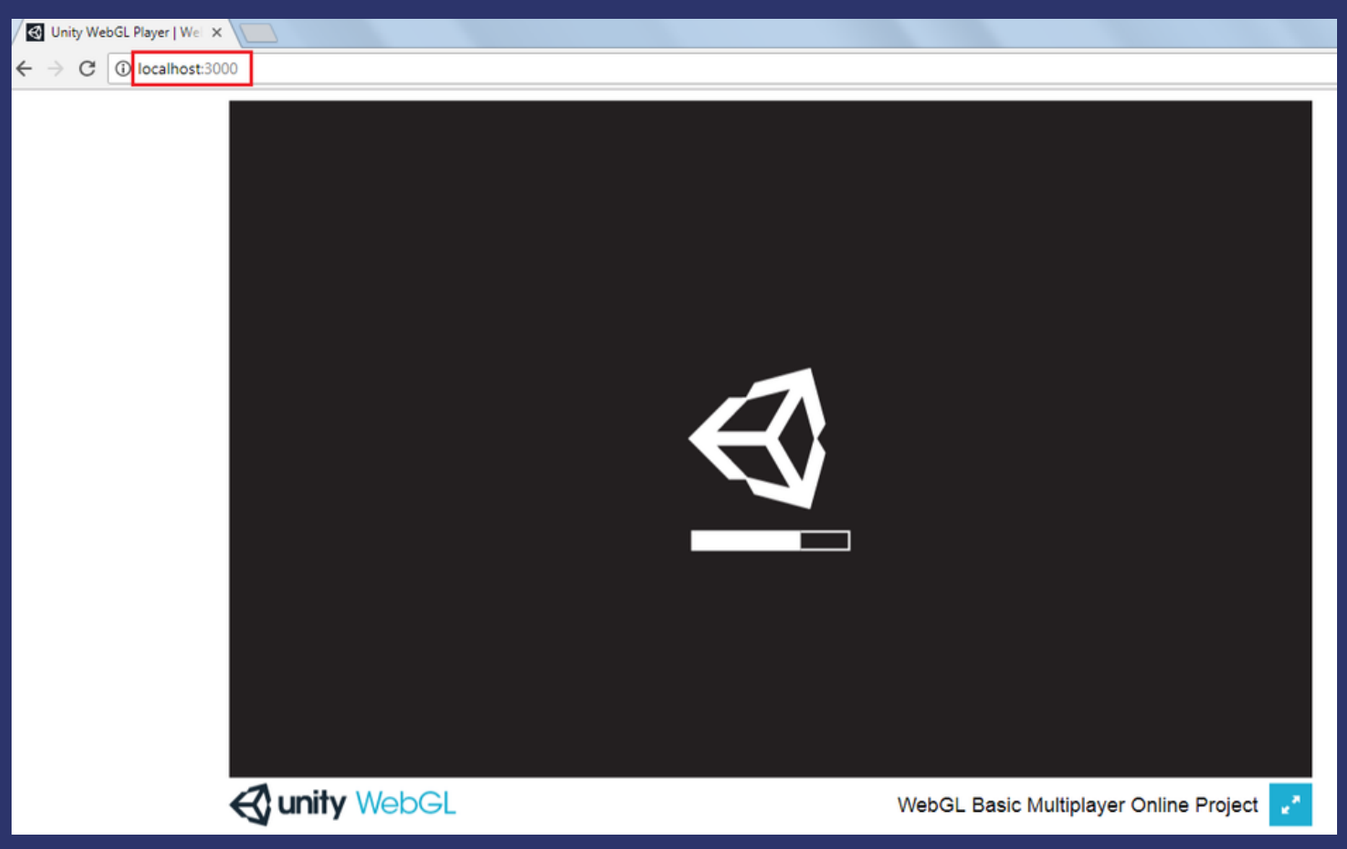
Now we can execute the server.js file through nodejs command line. Back to the NodeJs command prompt and type the follow command:

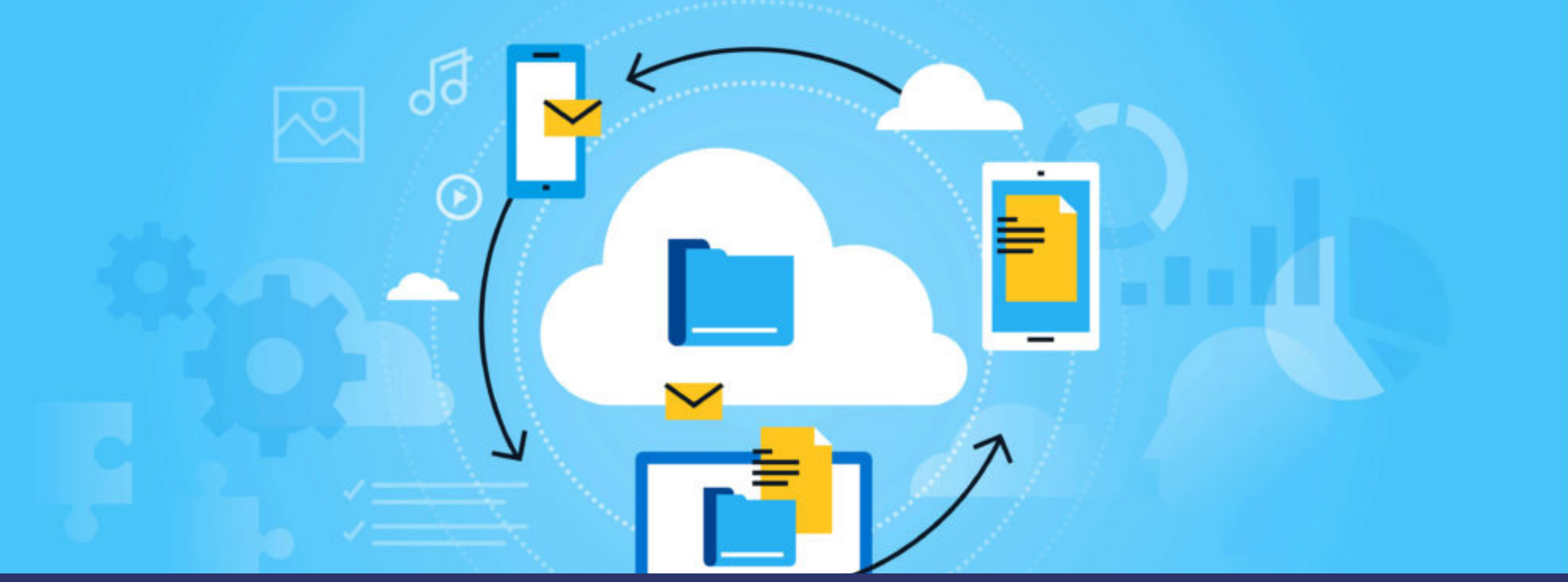
node server.js



To see the game running open the Google Chrome or Firefox and type the url:

localhost:3000





Publish your game

Server hosting, it's pretty simple on heroku, amazon aws, google cloud and azure, they all work very similary!

Basically we need to create a local repository using git (in this case in your nodejs server folder)

Then just use the "git push" command to send all files to the cloud service. In all cases you need to download a helper tool called CLI, each cloud service has its own. We will leave some tutorials and any question or difficulties, plyease contact us!

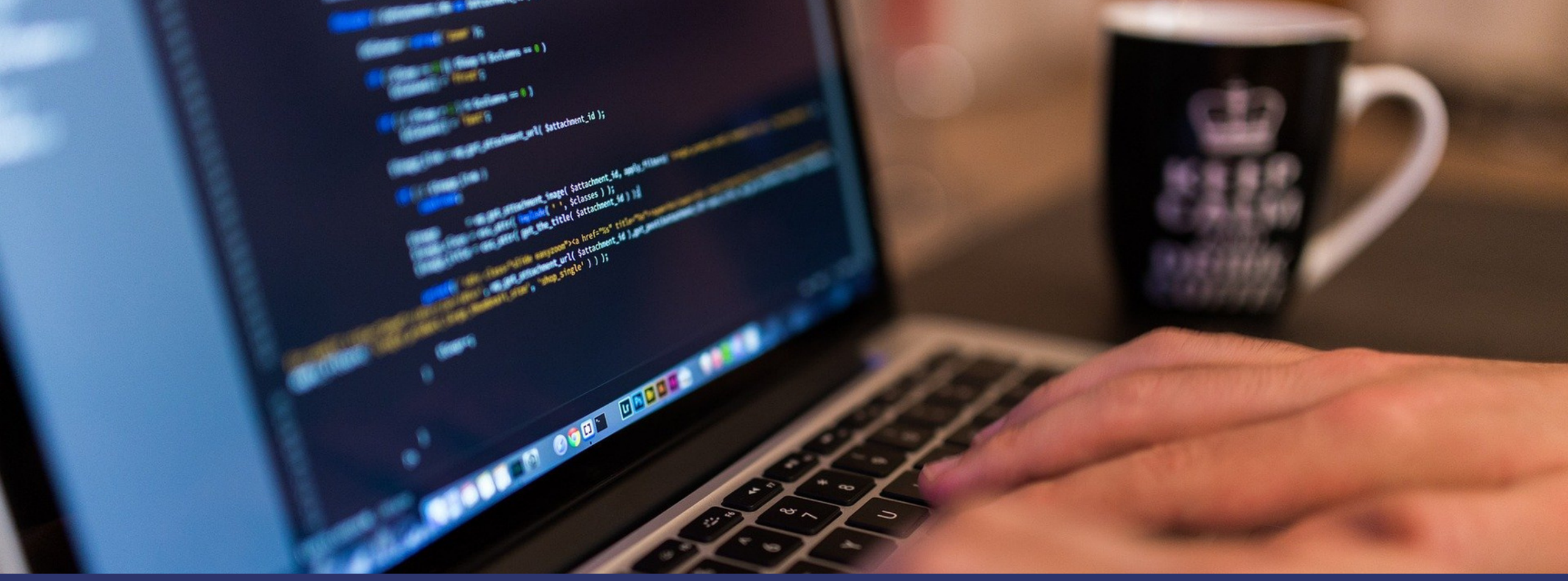
How deploy nodejs server to heroku tutorial 1

How deploy nodejs server to heroku tutorial 2

How deploy nodejs server to heroku tutorial 3

How deploy nodejs server to Amazon AWS

How deploy nodejs server to Google Cloud



How to develop

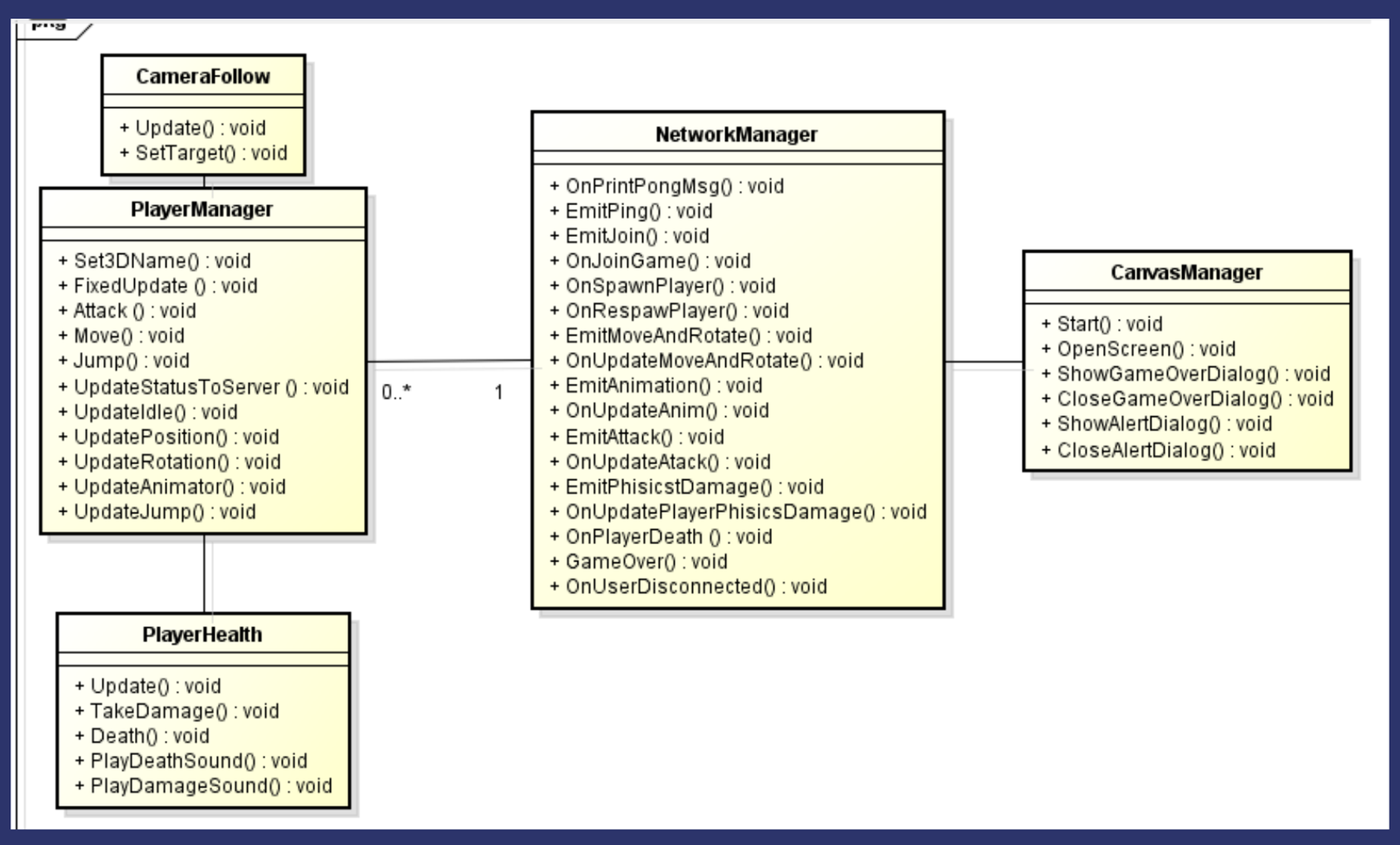
Before developing your multiplayer game in WebGL, keep in mind that the application will be divided into two parts:

- The Front-end: all visual, game mechanics and physics will be done in unity. Whatever game you have in mind, we will just use a main class called Network Manager to do all the interaction with the server.

- The Back-end: here comes the server in java script, which will make the connection between all game clients. Along with the server we will still have a script called client.js that will act as an intermediary between the back-end (server.js) and the front end (unity code compiled in WebGL).

The Network Manager

Network Manager class will control the state of this Multiplayer Online game, including game state management, spawn management and network players status.

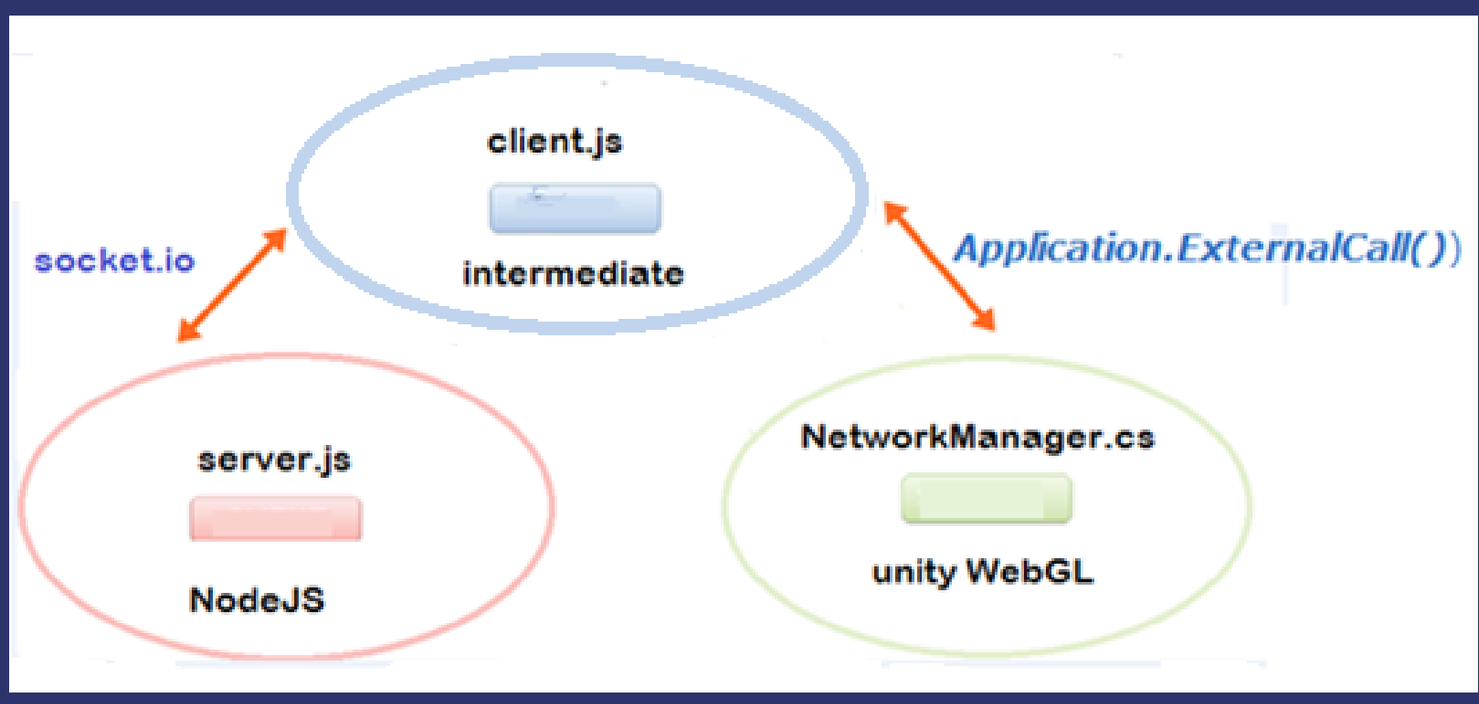


CONNECT TO SERVER

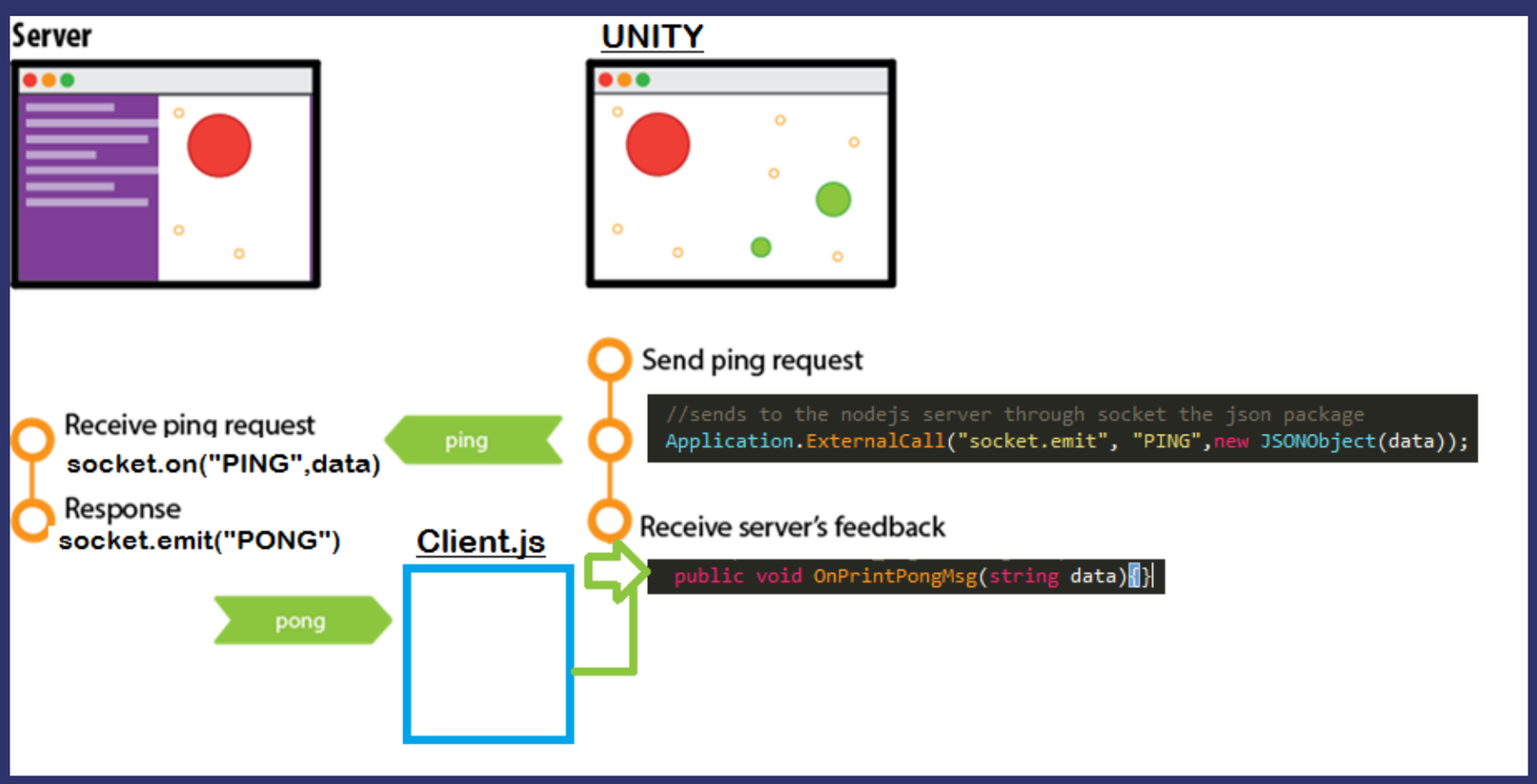
Connecting to a socket.io server is just one line:

NetworkManager.cs

```
void Awake()
{
    //connect to nodejs server
    Application.ExternalEval("socket.isReady = true;");
}
```



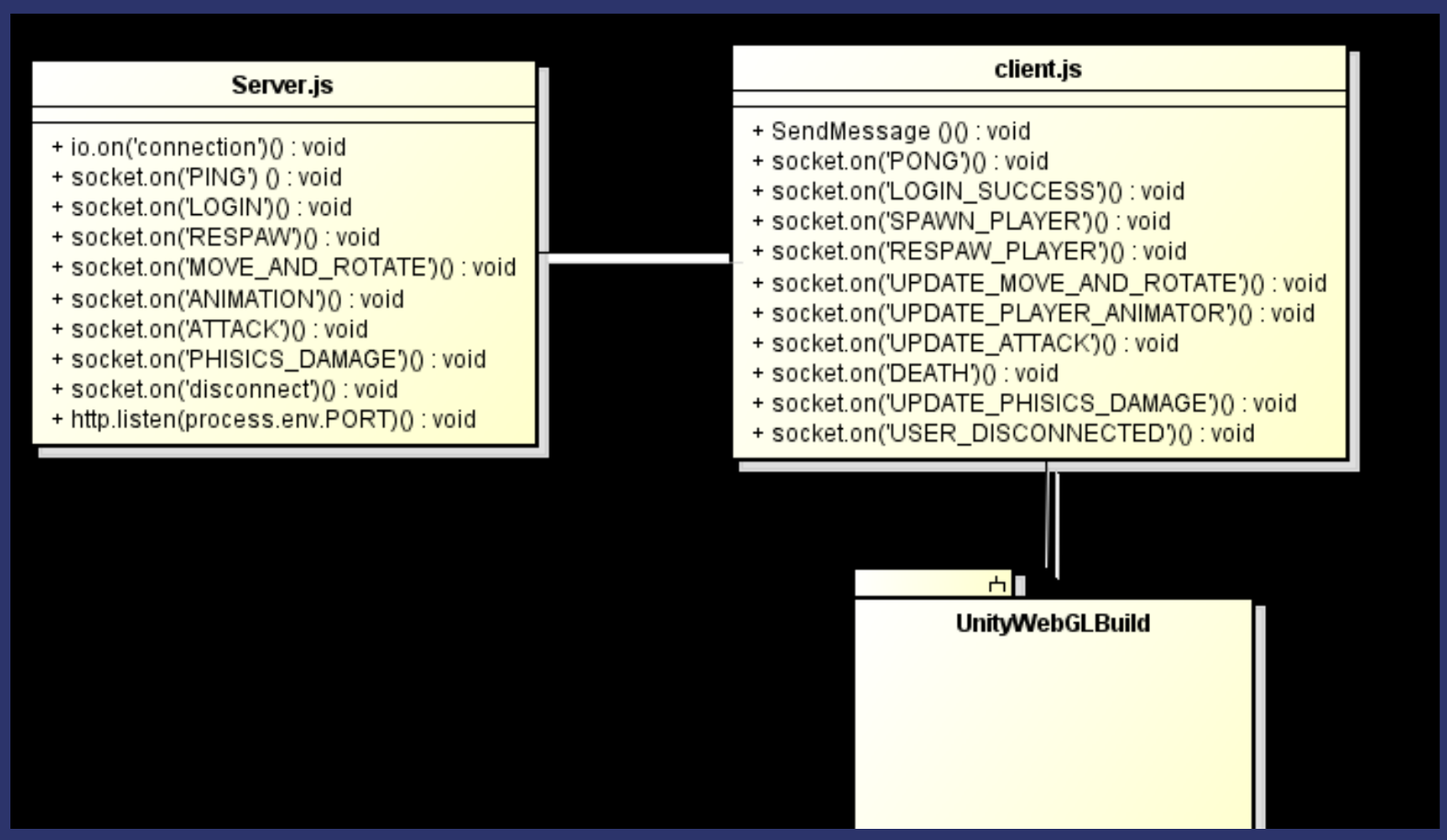
SENDING AND RECEIVING MESSAGES FROM THE SERVER.JS



You can use send information to socket.io server using the Application.ExternalCall method:

NetworkManager.cs

```
// <summary>
/// sends ping message to server.
/// </summary>
public void EmitPing() {

    //hash table <key, value>
    Dictionary<string, string> data = new Dictionary<string, string>();

    //store "ping!!!" message in msg field
    data["msg"] = "ping!!!!";

    //Send message in JSon formats to the NodeJS server
    Application.ExternalCall("socket.emit", "PING",new JSONObject(data));

}
```



the server.js get the unity network manager message using:

server.js

```
//create a callback fuction to listening EmitPing() method in NetworkMannager.cs unity script
socket.on('PING', function (_pack)
{
  var pack = JSON.parse(_pack);


});//END_SOCKET.ON
```

Receiving messages from server.js in unity:

- First: In server.js you can use send information to client.js using the Emit method:

```
//create a callback fuction to listening EmitPing() method in NetworkMann
socket.on('PING', function (_pack)
{
  var pack = JSON.parse(_pack);

    console.log('message from user# '+socket.id+": "+pack.msg);

     //emit back to NetworkManager in Unity per client.js script
    socket.emit('PONG', socket.id,pack.msg);

});//END_SOCKET_ON
```

- Second: In client.js setup the information and SendMessage method to send message to NetworkManager class in unity:

client.js

```
socket.on('PONG', function(socket_id,msg) {

  var currentUserAtr = socket_id+','+msg;

  // sends the package currentUserAtr to the method OnPrintPongMsg in the NetworkManager Game Object
  // with NetworkManager class on Unity
  gameInstance.SendMessage ('NetworkManager', 'OnPrintPongMsg', currentUserAtr);

});//END_SOCKET.ON
```

Game Object name with NetworkManager class

method of the Network Manager class responsible for receiving the message

The message

- Third: Go to NetworkManager.cs class and configure the OnPrintPongMsg method to receive the message:

```
/// <summary>
/// Prints the pong message which arrived from server.
/// </summary>
/// <param name="_msg">Message.</param>
public void OnPrintPongMsg(string _msg)
{

    var pack = _msg.Split (Delimiter);//separates the items contained in the package using the comma "," as sifter
    /*
     * pack[0]= socket_id
     * pack[1]= msg
    */
    CanvasManager.instance.ShowAlertDialog ("received message from server: "+pack[1]);
}
```

- The same steps can be applied for the other network functions in your game.

How to test the game without relying on the lengthy compilation for Web GL

unfortunately it is not possible to test multiplayer online in the unity editor, and we can only test it by compiling a build for WebGL. This is because the WebGL code communication is configured to work in conjunction with the NodeJS server,as if it were a web page served by a server.
So to test the game we need to build the game and insert it into the nodeJS server root folder.

The problem with this approach is the wait time for each WebGL build. so what's the possible solution?

Solution!

"Develop your project using socketIO for unity and then easily port to WebGL"

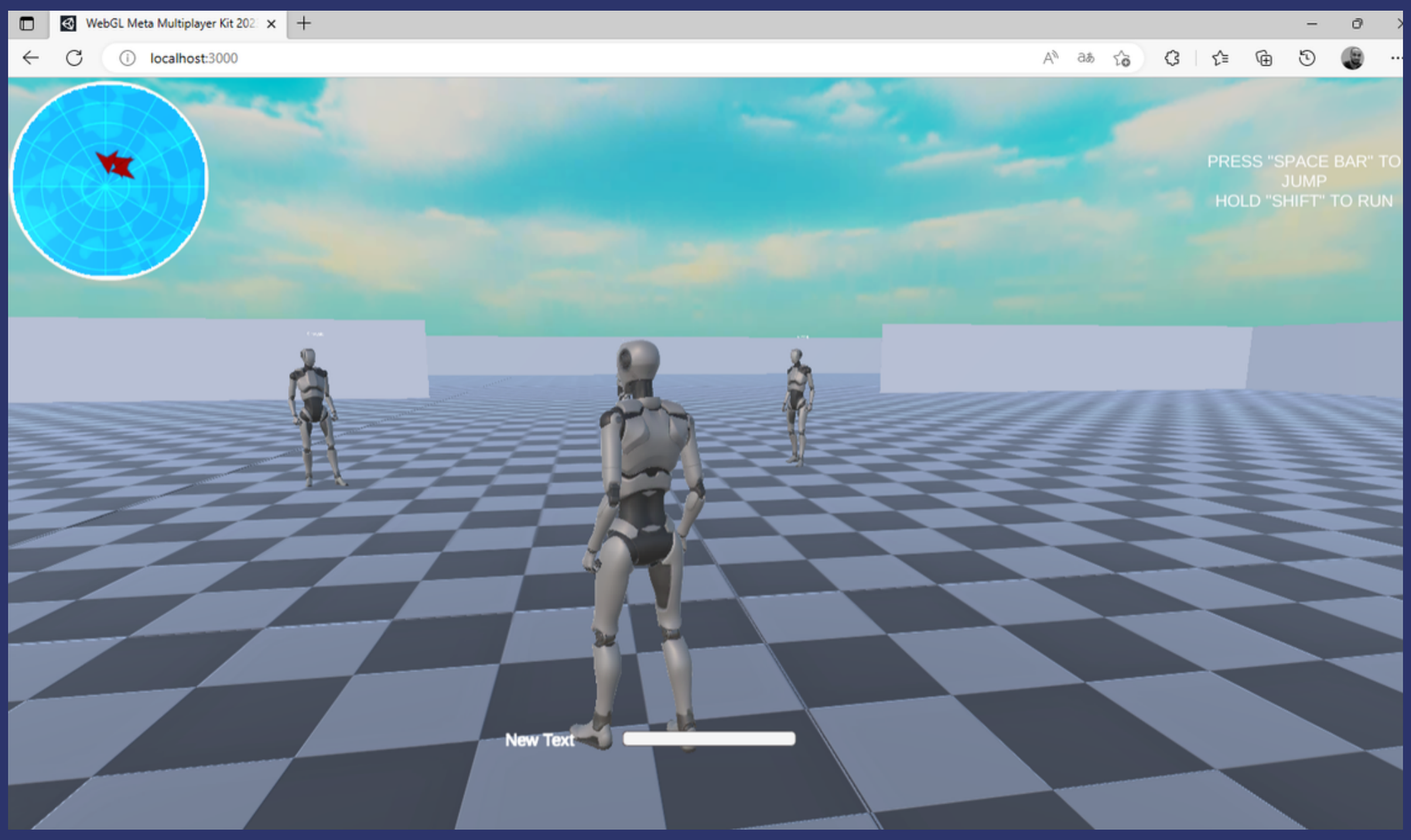
Currently there is an excellent free plugin called socketIO for Unity.

The way this plugin works with nodejs is very similar to the way we do it.Furthermore, converting a project made with socketIO to a WebGL project is extremely easy as we can see in the figure

To help you develop your game we will make the same examples available for free in the socketIO asset free of charge. You can download it through the link below:

Download PredatorIO and Helicpter Warzone projects in socketIO (download links inside asset folder)

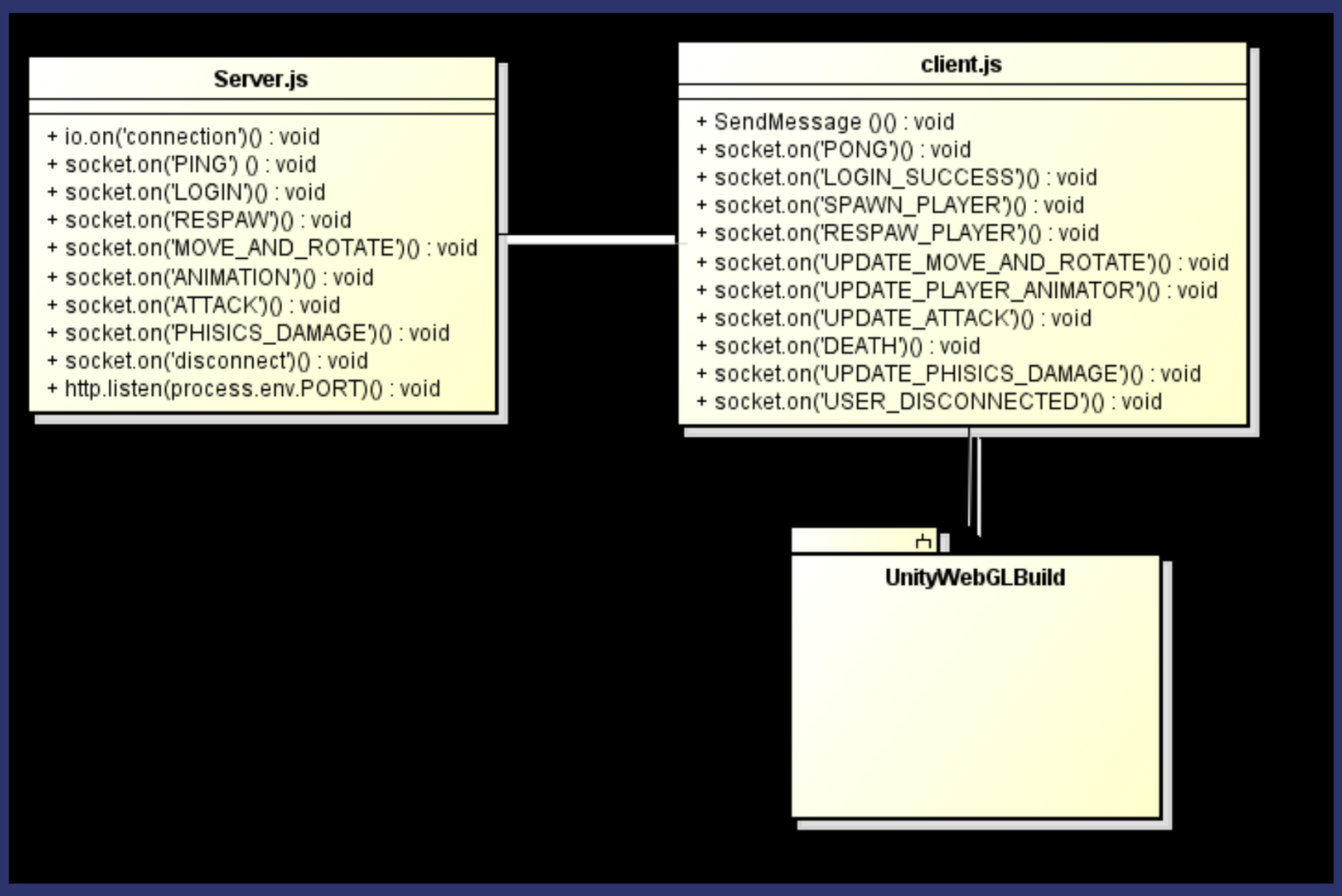
Basic Sample



Network Manager Class

```
13
14  public class NetworkManager : MonoBehaviour {
15
16    void Awake(){}
17
18    // Use this for initialization
19    void Start () {}
20
21    /// <summary>
22    /// Prints the pong message which arrived from server.
23    /// </summary>
24    /// <param name="_msg">Message.</param>
25    public void OnPrintPongMsg(string data){}
26
27    // <summary>
28    /// sends ping message to server.
29    /// </summary>
30    public void EmitPing() {}
31
32    //call be  OnClickJoinBtn() method from CanvasManager class
33    /// <summary>
34    /// Emits the player's name to server.
35    /// </summary>
36    /// <param name="_login">Login.</param>
37    public void EmitJoin(){}
38
39
40    /// <summary>
41    /// Joins the local player in game.
42    /// </summary>
43    /// <param name="_data">Data.</param>
44    public void OnJoinGame(string data){}
45
46    /// <summary>
47    /// Raises the spawn player event.
48    /// </summary>
49    /// <param name="_msg">Message.</param>
50    void OnSpawnPlayer(string data){}
51
52    //method to respawn  player called from client.js
53    void OnRespawPlayer(string data){}
54
55    //send position and rotation to  server
56    public void EmitMoveAndRotate( Dictionary<string, string> data){}
57
58    /// <summary>
59    /// Update the network player position and rotation to local player.
60    /// </summary>
61    /// <param name="_msg">Message.</param>
62    void OnUpdateMoveAndRotate(string data){}
63
```

Back-end side



Voice Chat

One of the most interesting features for user interaction in multiplayer applications is voice chat.

for this asset we chose to implement the chat voice directly in the back-end using socketIO and the native java script audio functions without a direct relationship with the source code in unity.

Voice chat was implemented in all examples, and the code referring to this feature is in the server.js and client.js files. next we will see the code related to the voice chat

Client.js

In client.js we will have the functions to capture the audio from the player's microphone and also the listenner to play the audio received via socketio.

in the **mainFunction()** , we perform the work of capturing the audio from the microphone, serialize the audio in base64 and send the audio packet to the server using the **socket.emit()** method.

```
function mainFunction(time) {


  navigator.mediaDevices.getUserMedia({ audio: true }).then((stream) => {
    var madiaRecorder = new MediaRecorder(stream);
    madiaRecorder.start();

    var audioChunks = [];

    madiaRecorder.addEventListener("dataavailable", function (event) {
      audioChunks.push(event.data);
    });

    madiaRecorder.addEventListener("stop", function () {
      var audioBlob = new Blob(audioChunks);

      audioChunks = [];

      var fileReader = new FileReader();
      fileReader.readAsDataURL(audioBlob);
      fileReader.onloadend = function () {


        var base64String = fileReader.result;
        socket.emit("VOICE", base64String);

      };

      madiaRecorder.start();


      setTimeout(function () {
        madiaRecorder.stop();
      }, time);
    });
```

record the microphone

serializes and sends the audio to the server

Server.js

on the server we just receive the audio and retransmit it to the other players depending on some conditions checked in a simple "if" conditional.

in the code below we receive the audio through the listenner **socket.on("VOICE")**, then we go through the client list, check if we are not sending the audio to the same player that is transmitting the audio and if the current client is not the audio is muted. if these conditions are met, we send the audio to the current client through the **sockets[u.id].emit** instruction

```
socket.on("VOICE", function (data) {


  if(currentUser)
  {

   var newData = data.split(",");//define a separator character
   newData[0] = "data:audio/ogg;"; //format the audio packet header
   newData = newData[0] + newData[1];//concatenate

   //go through the clients list and send audio to the current client "u" if the conditions are met
   clients.forEach(function(u) {

      if(sockets[u.id]&&u.id≠ currentUser.id&&!u.isMute)
      {
        sockets[u.id].emit('UPDATE_VOICE',newData);
      }
    });//END_FOREACH


  }//END_IF

});//END_SOCKETIO
```

additionally we also have the function to mute the audio in server.js. in Unity the user, if he prefers, can mute the audio by pressing a button.

```
  socket.on("AUDIO_MUTE", function (data) {

  if(currentUser)
  {
    currentUser.isMute = !currentUser.isMute;

  }

  });
```

Client.js

in client.js we have the listener socket.on("UPDATE_VOICE") that receives the audio and plays for the current client

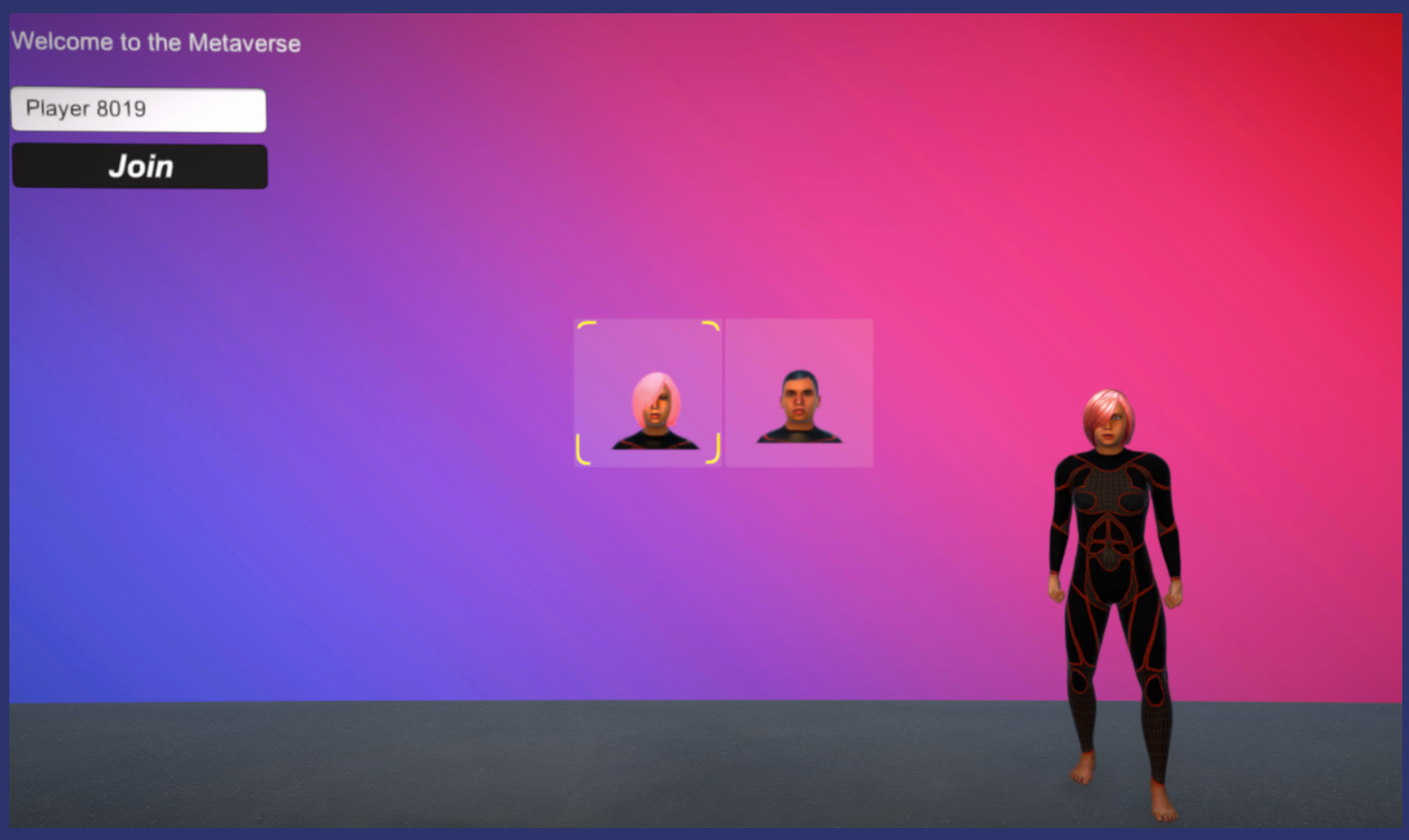
```
socket.on("UPDATE_VOICE", function (data) {
    var audio = new Audio(data);
    audio.play();
 });
```

METAVERSE

A metaverse is a network of 3D virtual worlds focused on social connection. In the metaverse sample you will find several features to implement your own metaverse. text chat, voice chat, pear to pear transaction system with Metamask and NFT purchase are some of the features you have available. In addition, it is possible to create apresentations using the video player.

Sign In and authentication system using Metamask

There are two ways for the user to enter the metaverse, by authentication via metamask or as a guest.



In the NetworkManager Class, we can see the **SignIn** method that switches the user depending on the type of Sign in chosen by the user.

```
// <summary>
/// manages and switches user login.
/// </summary>
public void SignIn(string _method)
{

    switch(_method)
    {
        case "metamask":
         MetamaskSignIn();
        break;
        case "guest":
        CanvasManager.instance.OpenScreen(2);
        break;
    }

}
```

If the user chooses to log in with his Metamask wallet, the **MetamaskSignIn** method of the external ***Web3bridge.jslib*** java script file is called

Next we see the java script function in Web3Bridge responsible for signing in the user through its Metamask wallet

```
MetamaskSignIn: function(){

 window.ethereum.request({method: 'eth_requestAccounts'}).then(accounts =>{

 account = accounts[0];
 console.log(account);


 //get the balance of the user and convert to MATIC
 window.ethereum.request({method: 'eth_getBalance' , params: [account, 'latest']}).then(result => {
          console.log(result);
          let wei = parseInt(result,16);
          let balance = wei / (10**18);
          console.log(balance + "MATIC");


          var currentUserAtr =  account+':'+ balance;

          if(window.unityInstance≠null)
          {

            //calls the OnMetamaskSignIn function of CanvasManager and sends the result to the unity application
            window.unityInstance.SendMessage ('CanvasManager', 'OnMetamaskSignIn', currentUserAtr);

          }//END_IF
        });

  });

},
```

In the CanvasManager Class, we can see the **OnMetamaskSignIn** method that allow user Sign

```
 /// <summary>
/// Joins the local player in game.
/// </summary>
/// <param name="_data">Data.</param>
public void OnMetamaskSignIn(string data)
{

    /*
     * pack[0] = my public address
     * pack[1]= balance (MATIC)

    */

    Debug.Log("Login successful");

    var pack = data.Split (Delimiter);

    // the local player now is logged
    NetworkManager.instance.onLoggedWithMetamask = true;

    if_myPublicAdrr.text = pack[0];

    myPublicAdrr = pack[0];

    balance = pack[1];

    if_balance.text = pack[1];

    defaultPublicAdrrTo = "0×2953399124F0cBB46d2CbACD8A89cF0599974963";

    OpenScreen(2);


}
```

finally the user sign in is completed sending more additional information to the nodeJS server through the **EmitJoin** method

```
/// <summary>
/// Emits the player's information to the server.
/// </summary>
/// <param name="_login">Login.</param>
public void EmitJoin()
{
    //hash table <key, value>
    Dictionary<string, string> data = new Dictionary<string, string>();


    //makes the draw of a point for the player to be spawn
    int index = Random.Range (0, spawnPoints.Length);

    //send the position point to server
    string msg = string.Empty;


    data["name"] = CanvasManager.instance.inputLogin.text;

    data["publicAddress"] = "none";

    if(onLoggedWithMetamask)
    {
        data["publicAddress"] = CanvasManager.instance.myPublicAdrr;
    }

    //store player's skin
    data["model"] = CharacterChoiceManager.instance.current_model.ToString();
    data["posX"] = spawnPoints[index].position.x.ToString();
    data["posY"] = spawnPoints[index].position.y.ToString();
    data["posZ"] = spawnPoints[index].position.z.ToString();

    //sends to the nodejs server through socket the json package
    Application.ExternalCall("socket.emit", "JOIN",new JSONObject(data));

    //obs: take a look in server script.
}
```

In the nodeJS server we have:

```
//create a callback fuction to listening EmitJoin() method in NetworkMannager.cs unity script
socket.on('JOIN', function (_data)
{

    console.log('[INFO] JOIN received !!! ');

    var data = JSON.parse(_data);

     // fills out with the information emitted by the player in the unity
    currentUser = {
                name:data.name,
                publicAddress: data.publicAddress,
                model:data.model,
                posX:data.posX,
                posY:data.posY,
                posZ:data.posZ,
                rotation:'0',
                id:socket.id,//alternatively we could use socket.id
                socketID:socket.id,//fills out with the id of the socket that was open
                muteUsers:[],
                isMute:true
                };//new user  in clients list
     You, agora • Uncommitted changes
    /**************************************************************************************/

    //send to the client.js script
    socket.emit("JOIN_SUCCESS",currentUser.id,currentUser.name,currentUser.posX,currentUser.posY,currentUser.posZ,data.model);

     //spawn all connected clients for currentUser client
```

Finally in the **socket.on('JOIN_SUCCESS')** callback in **client.js** java script file we send a response to the unity application to the **OnJoinGame** method of the **NetworkManager** class

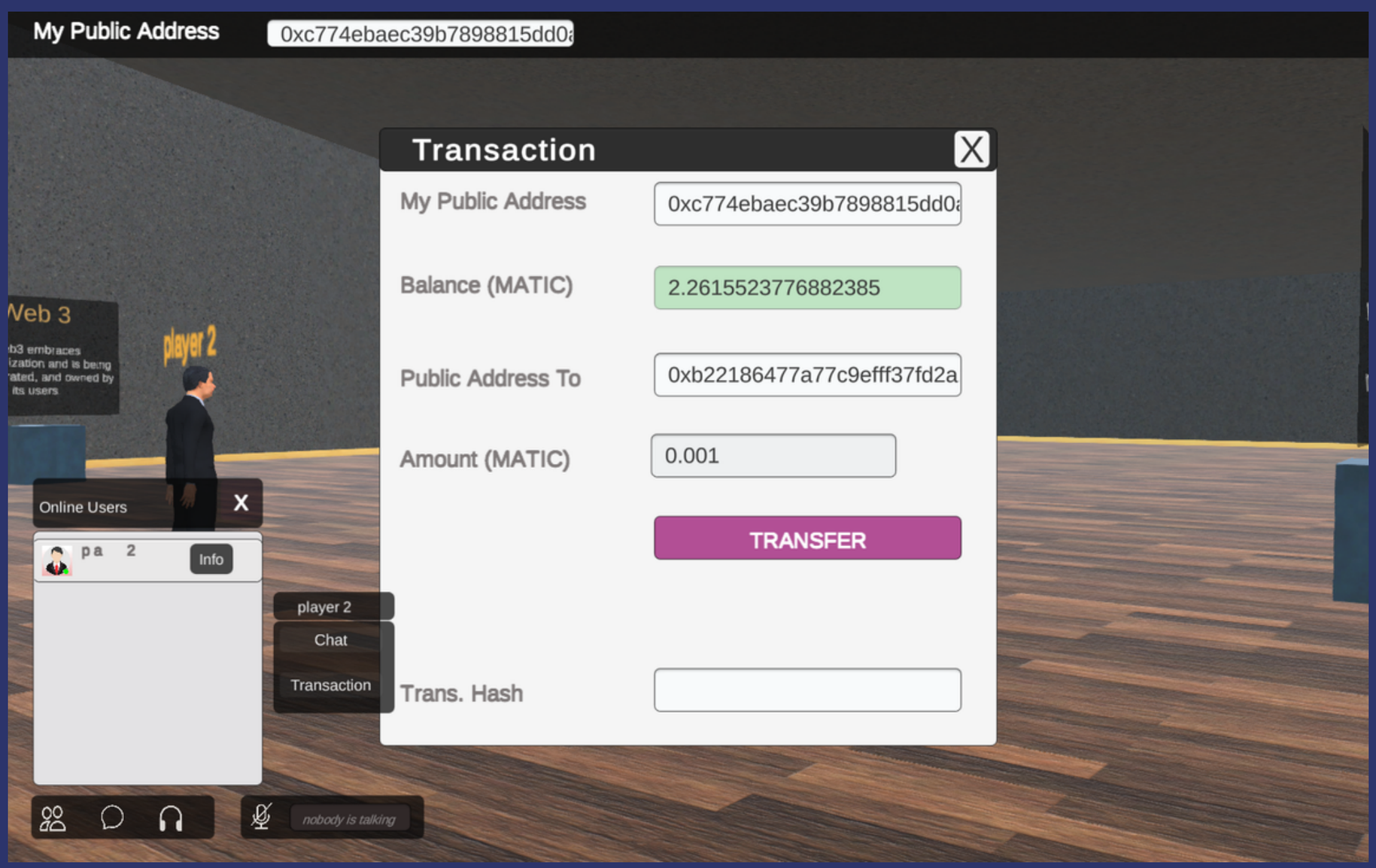
```
socket.on('JOIN_SUCCESS', function(id,name,posX,posY,posZ,model) {

  var currentUserAtr = id+':'+name+':'+posX+':'+posY+':'+posZ+':'+model;

   if(window.unityInstance≠null)
    {

      window.unityInstance.SendMessage ('NetworkManager', 'OnJoinGame', currentUserAtr);

    }

});//END_SOCKET.ON
```

In **OnJoinGame** we just instantiate the player with the information provided by the nodeJS server.

```
    /// <summary>
    /// Joins the local player in game.
    /// </summary>
    /// <param name="_data">Data.</param>
    public void OnJoinGame(string data)
    {


        /*
         * pack[0] = id (local player id)
         * pack[1]= name (local player name)
         * pack[2] = position.x (local player position x)
         * pack[3] = position.y (local player position  ... )
         * pack[4] = model

        */

        Debug.Log("Login successful, joining game");

        var pack = data.Split (Delimiter);

        // the local player now is logged
        onLogged = true;


        // take a look in NetworkPlayer.cs script
        PlayerManager newPlayer;

        // newPlayer = GameObject.Instantiate( local player avatar or model, spawn positi
        newPlayer = GameObject.Instantiate (playerPref,
```

Metamask transfer System

In this sample we implement a simple transaction system using the metamask wallet. With this system it is possible to make transfers to another user of the room. the hard work is done by the **web3Bridge.jslib** file.



When the user presses the "Transfer" button, the **TransferTo** method of the **CanvasManager** class is called. in the **TransferTo method** we make a call to the **MetamaskTransferTo function** of the **web3Bridge.jslib** java script file, sending the parameters Public wallet Adrress of the user, Public wallet Adrress of the recipient, and the value to be transferred

```
public void TransferTo()
{

    MetamaskTransferTo(if_myPublicAdrr.text,
            hudUserOptions.GetComponent<UserOptions>().if_publicAdrrTo.text,
                hudUserOptions.GetComponent<UserOptions>().if_amount.text);

}
```

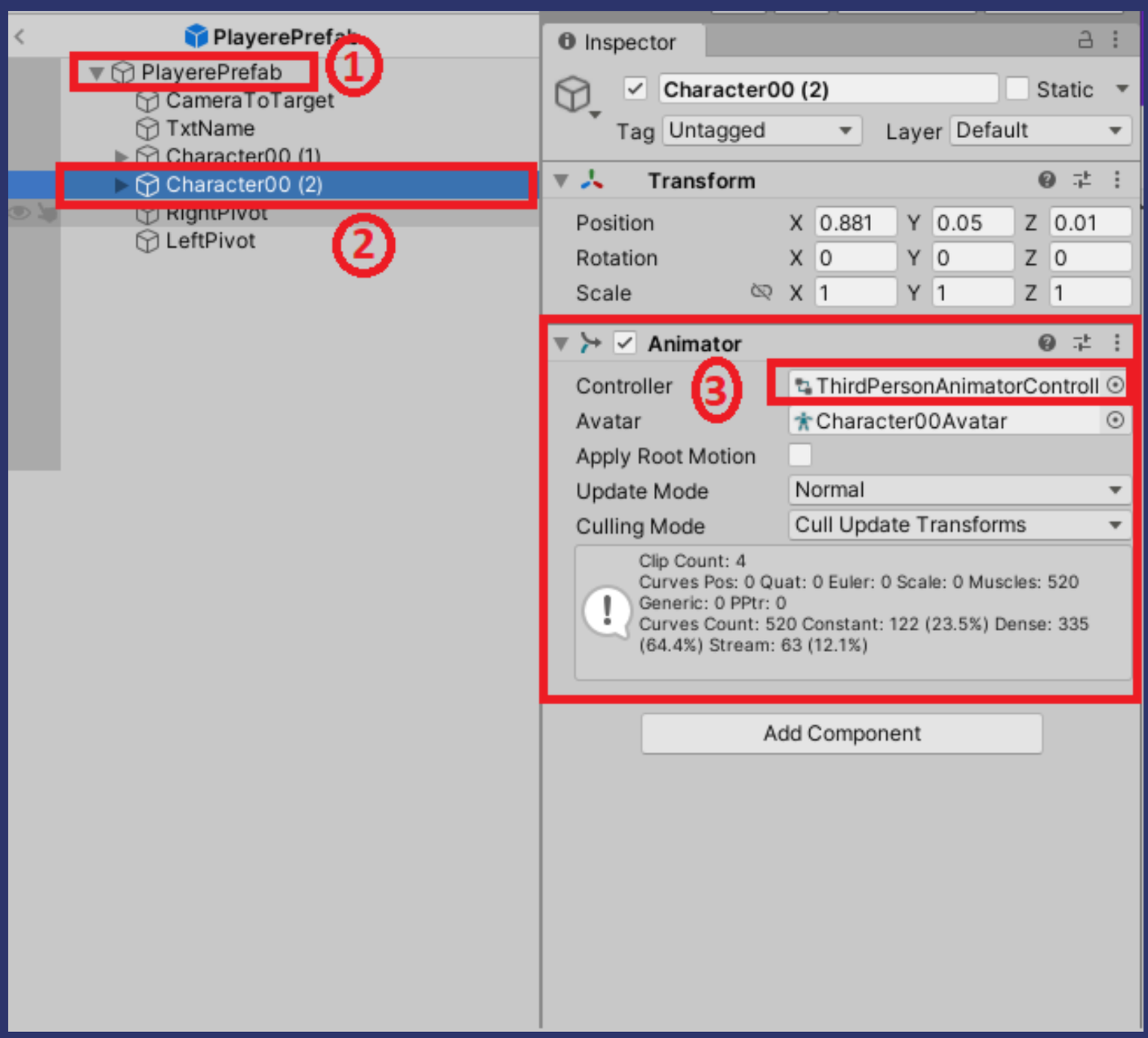
In **web3Bridge.jslib** we carry out all the communication work with the metamask wallet:

```
MetamaskTransferTo: function (_myPublicAdrr,_to_public_address, _amount) {

    var currentUserAtr =  '';
    var wei = parseFloat(UTF8ToString(_amount)) * Math.pow(10, 18);

     let transactionParam = {
           to: UTF8ToString(_to_public_address),
           from: UTF8ToString(_myPublicAdrr),
           value: wei.toString(16).toUpperCase() //convert to hex
         };



            window.ethereum.request({method: 'eth_sendTransaction', params:[transactionParam]}).then(txhash => {
              console.log(txhash);
              checkTransactionconfirmation(txhash).then(r =>
              {

              alert(r);
              currentUserAtr = txhash;

              window.ethereum.request({method: 'eth_getBalance' , params: [account, 'latest']}).then(result => {

                let wei = parseInt(result,16);
                let balance = wei / (10**18);
                console.log(balance + " MATIC");


                currentUserAtr =   currentUserAtr+':'+ balance;

                if(window.unityInstance≠null)
                {

                  window.unityInstance.SendMessage ('CanvasManager', 'OnEndTransaction', currentUserAtr);
```

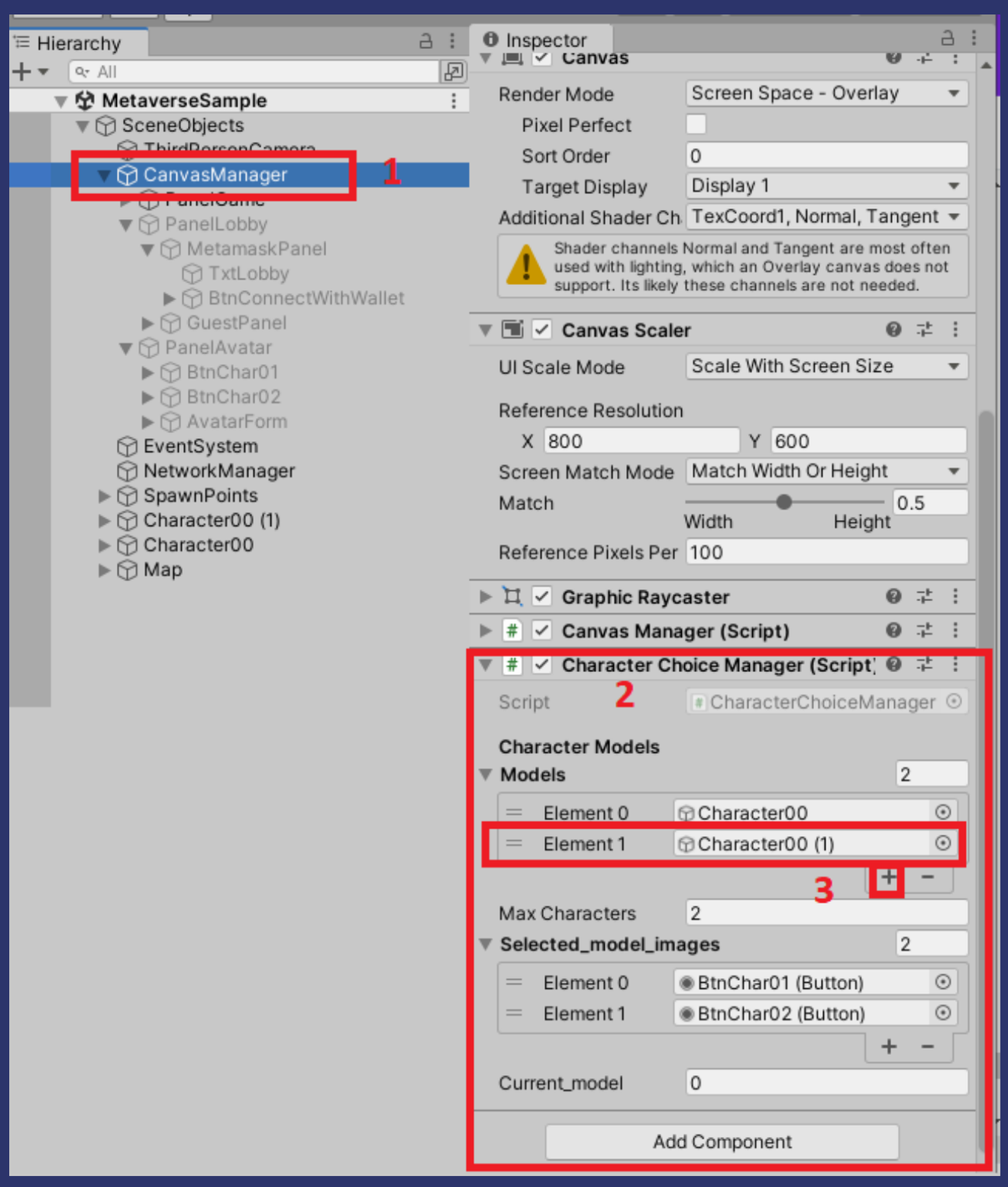
Finally, we send the result of the operation to the **OnEndTransaction** method of the networkManager class

How to customize the characters

To add your own character, open the Player Prefab , then add your character as a child of PlayerPrefab



CAUTION: When inserting a new character model, you must also go to the **CanvasManager** Component in unity editor, and make the necessary changes in **CharacterChoiceManager** , inserting the new character in the character choice interface.



Text Chat System

In this sample we have two types of text chat, private chat and general chat. in private text chat it is possible to send individual messages to each user independently like conventional chat apps. In general text chat, user messages are sent to all users in the metaverse

To handle text chat In the NetworkManager class we use the following methods:

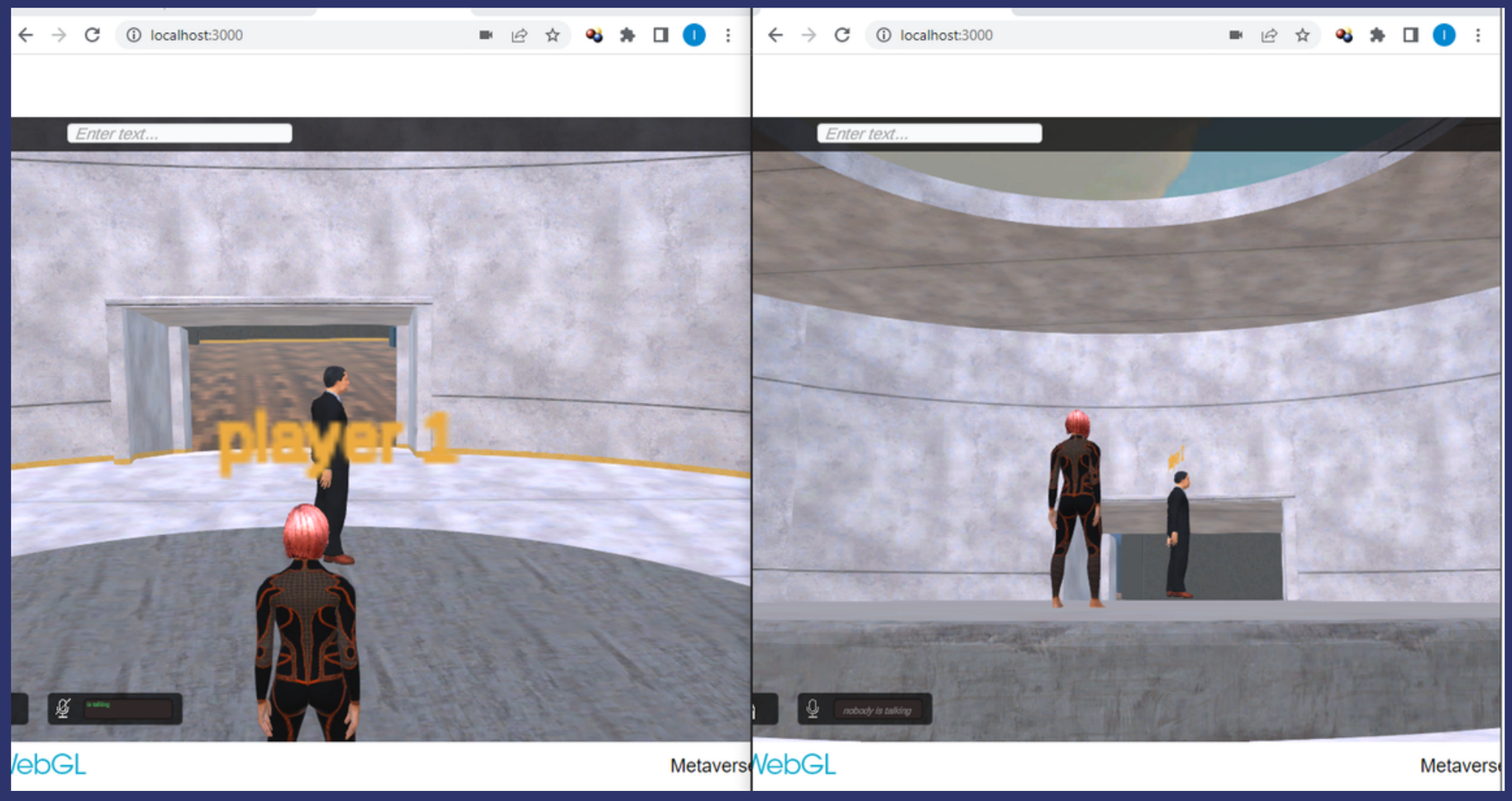
- **EmitOpenChatBox** : Send a private chat request to some user through the server
- **OnReceiveOpenChatBox**: Recive from server a private chat request
- **EmitPrivateMessage:** Send a private message to the another user
- **OnReceivePrivateMessage:** Receive a private message from another user through the server
- **EmitMessage :** Send a message to the chat room
- **OnReceiveMessage:** receives a message from the server to the chat room

To handle text chat In the NodeJS Server we use the following callbacks:

- **socket.on('SEND_OPEN_CHAT_BOX')**
- **socket.on('RECEIVE_OPEN_CHAT_BOX')**
- **socket.on('PRIVATE_MESSAGE')**
- **socket.on('MESSAGE')**

Voice Chat System

The voice chat system is done 100% by the server and client.js java script file via socketIO calbacks. in the unity application we only use some commands to silence the users' chat or allow the use of the microphone through buttons.



To handle chat voice In the client.js class and server.js, we use the following method:

Client.js

In client.js we will have the functions to capture the audio from the player's microphone and also the listenner to play the audio received via socketio.

in the **mainFunction()** , we perform the work of capturing the audio from the microphone, serialize the audio in base64 and send the audio packet to the server using the **socket.emit()** method.

```
function mainFunction(time) {


  navigator.mediaDevices.getUserMedia({ audio: true }).then((stream) => {
    var madiaRecorder = new MediaRecorder(stream);
    madiaRecorder.start();

    var audioChunks = [];

    madiaRecorder.addEventListener("dataavailable", function (event) {
      audioChunks.push(event.data);
    });

    madiaRecorder.addEventListener("stop", function () {
      var audioBlob = new Blob(audioChunks);

      audioChunks = [];

      var fileReader = new FileReader();
      fileReader.readAsDataURL(audioBlob);
      fileReader.onloadend = function () {


        var base64String = fileReader.result;
        socket.emit("VOICE", base64String);

      };

      madiaRecorder.start();


      setTimeout(function () {
        madiaRecorder.stop();
      }, time);
    });
```

record the microphone

serializes and sends the audio to the server

Server.js

on the server we just receive the audio and retransmit it to the other players depending on some conditions checked in a simple "if" conditional.

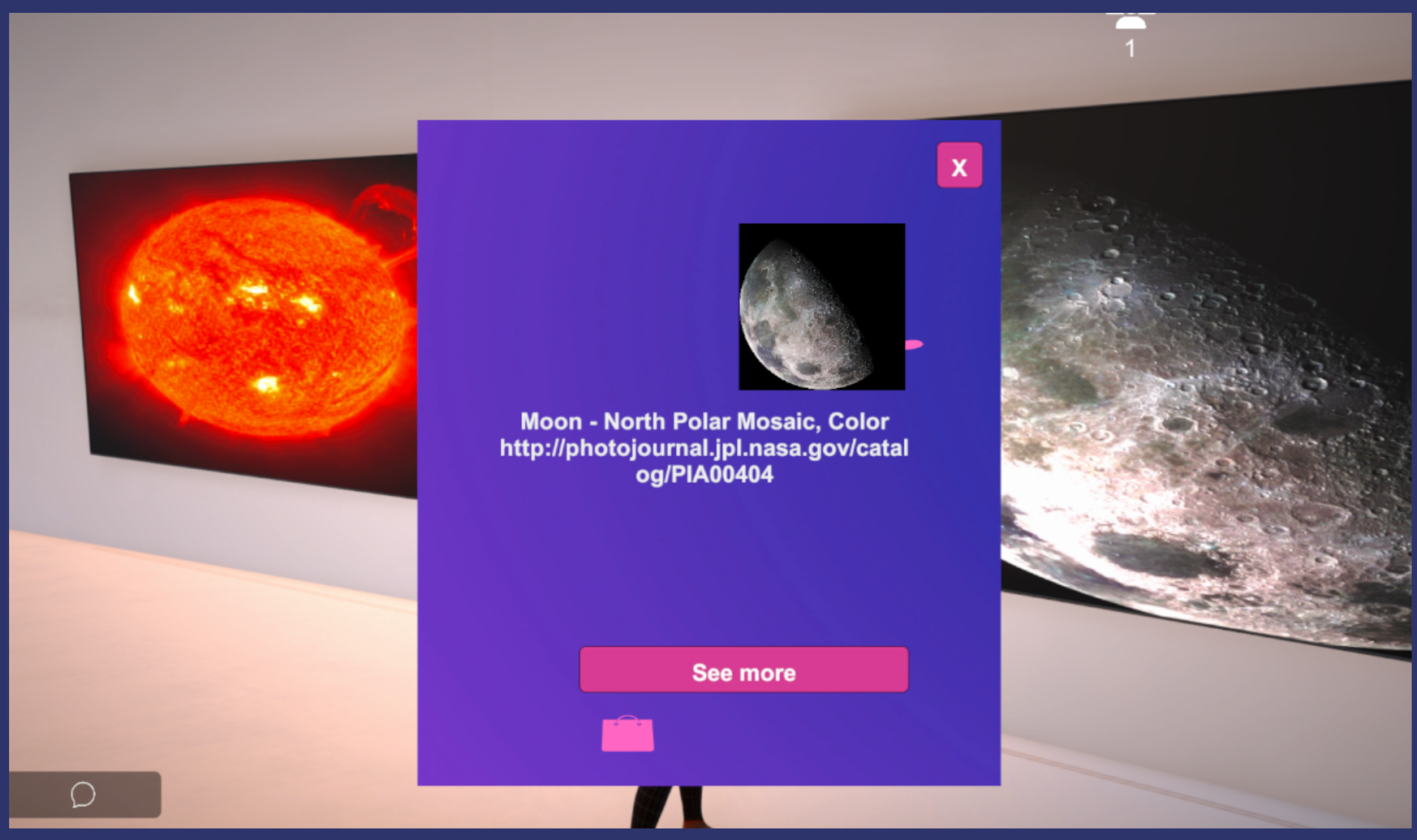
in the code below we receive the audio through the listenner **socket.on("VOICE")**, then we go through the client list, check if we are not sending the audio to the same player that is transmitting the audio and if the current client is not the audio is muted. if these conditions are met, we send the audio to the current client through the **sockets[u.id].emit** instruction

```
socket.on("VOICE", function (data) {


  if(currentUser)
  {

   var newData = data.split(",");//define a separator character
   newData[0] = "data:audio/ogg;"; //format the audio packet header
   newData = newData[0] + newData[1];//concatenate

   //go through the clients list and send audio to the current client "u" if the conditions are met
   clients.forEach(function(u) {

      if(sockets[u.id]&&u.id≠ currentUser.id&&!u.isMute)
      {
        sockets[u.id].emit('UPDATE_VOICE',newData);
      }
    });//END_FOREACH


  }//END_IF

});//END_SOCKETIO
```

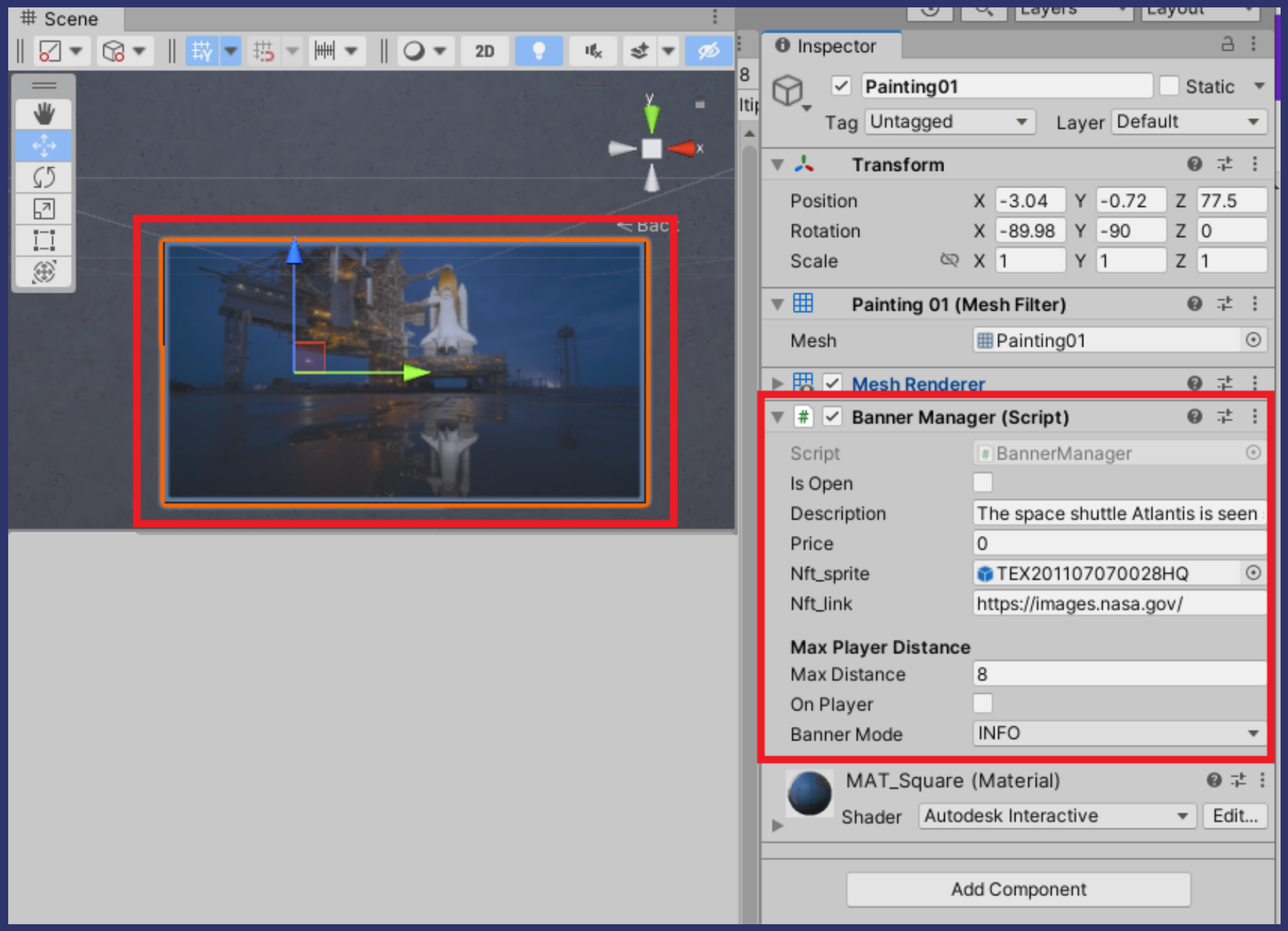
additionally we use a function to calculate the distance of each user creating a spatial audio

```
clients.forEach(function(u) {

    var distance = getDistance(parseFloat(currentUser.posX), parseFloat(currentUser.posY),parseFloat(u.posX), parseFloat(u.posY))

    var muteUser = false;

     for (var i = 0; i < currentUser.muteUsers.length; i++)
     {
        if (currentUser.muteUsers[i].id == u.id)
        {

            muteUser = true;


        };
    };
```

Adding your custom banner to the metaverse



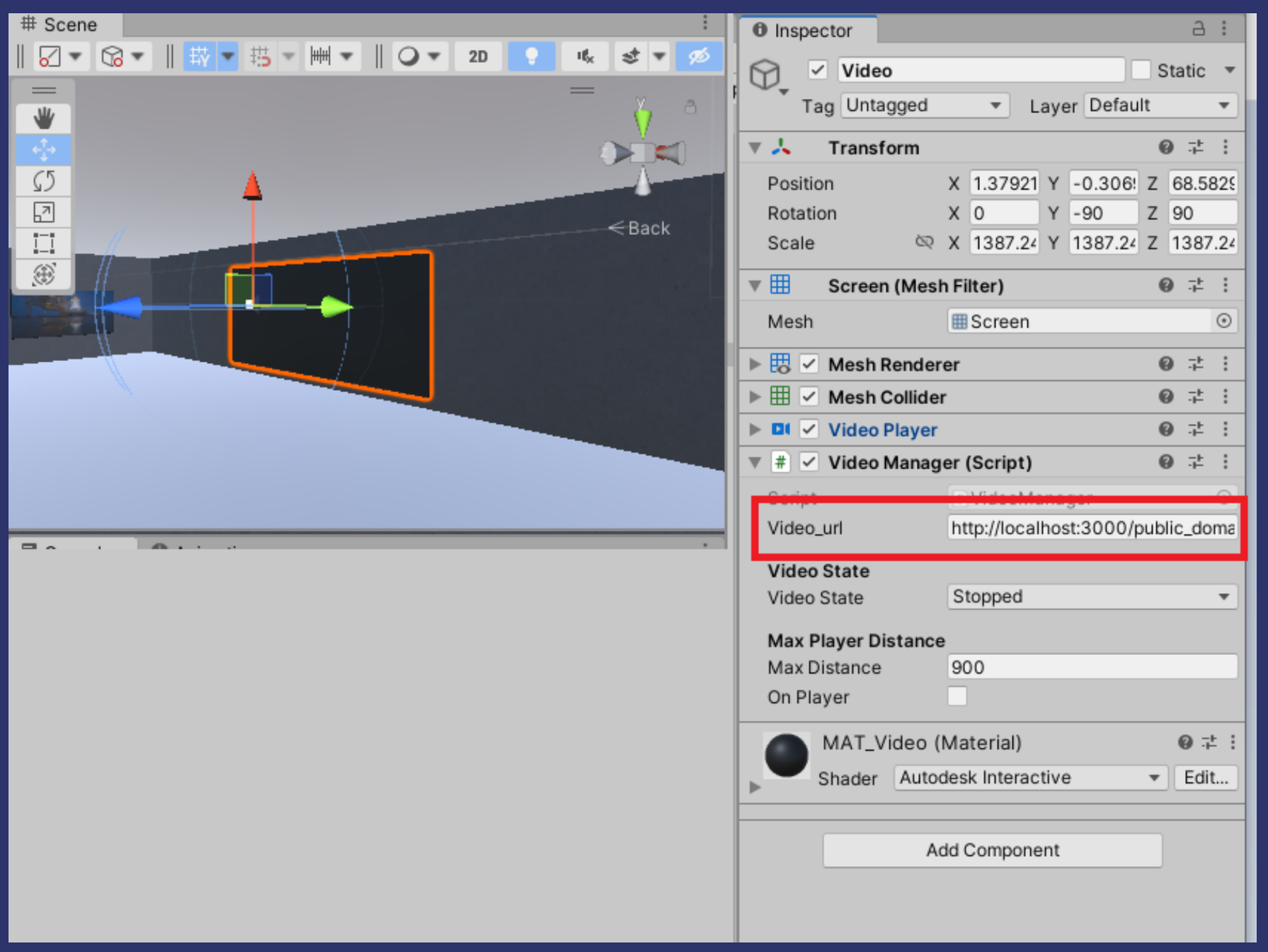
To add a custom banner to your metaverse you need to add a BannerManager script to your 3D banner



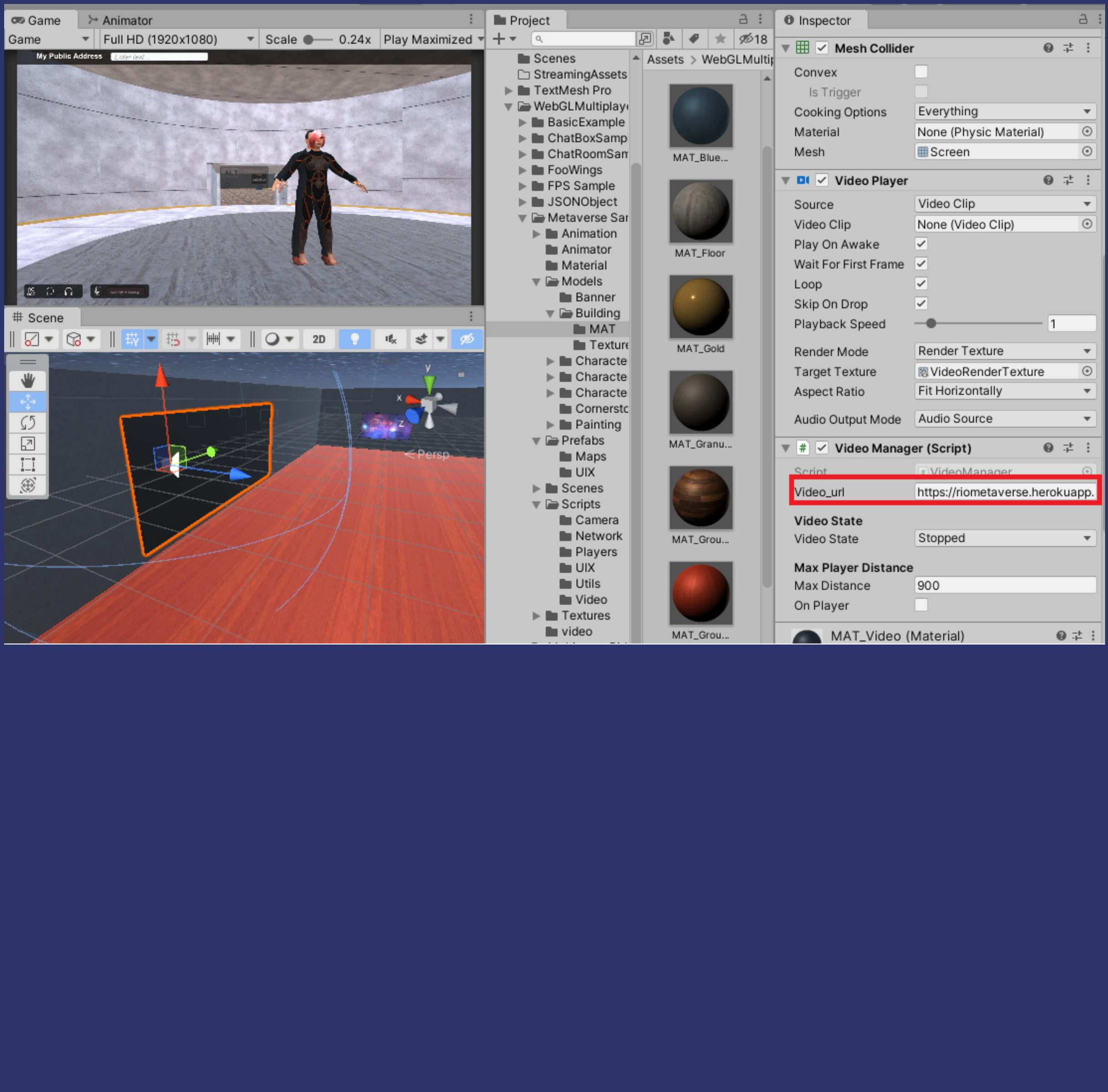
Adding your custom video to the project

To add a custom video to your metaverse you need to follow these steps:

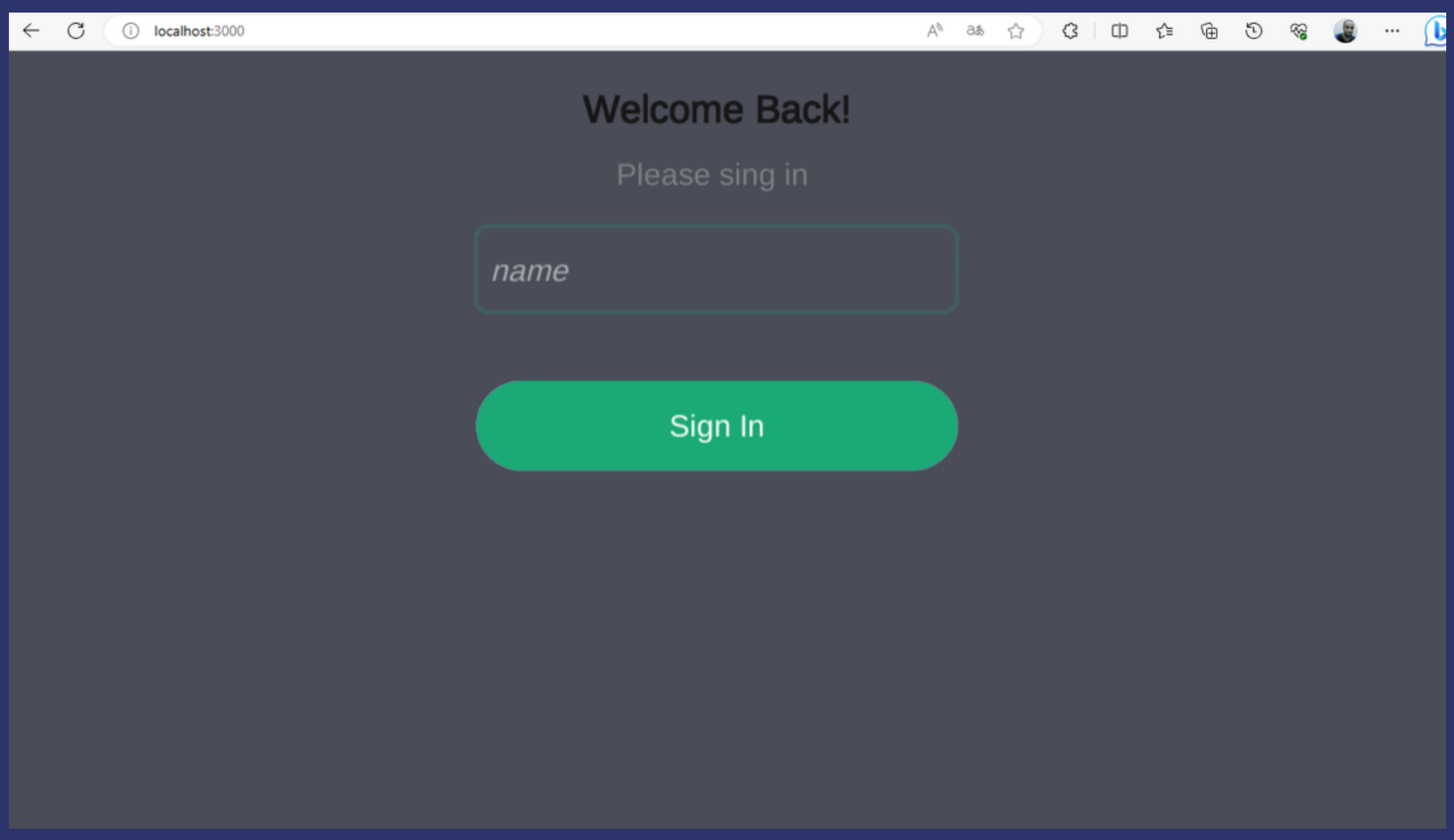
- Add the video to the Metaverse template folder you previously downloaded from github
- In the Hierarchy go to the game Object video and in VideoManager modify the field video url according to the **name of your video** file in the Metaverse Template folder.



- Don't forget when you send the application to the cloud (Amazon aws, heroku, google cloud, etc.), you need to change the **url** again according to the **url provided by the cloud server**. In the example below we modify the video url according to the url provided by Heroku!
- In our case the url was changed from **http://localhost:3000/video_name.mp4** to: **https://riometaverse.herokuapp.com/video_name.mp4**



ChatGPT & OpenAI API Integration

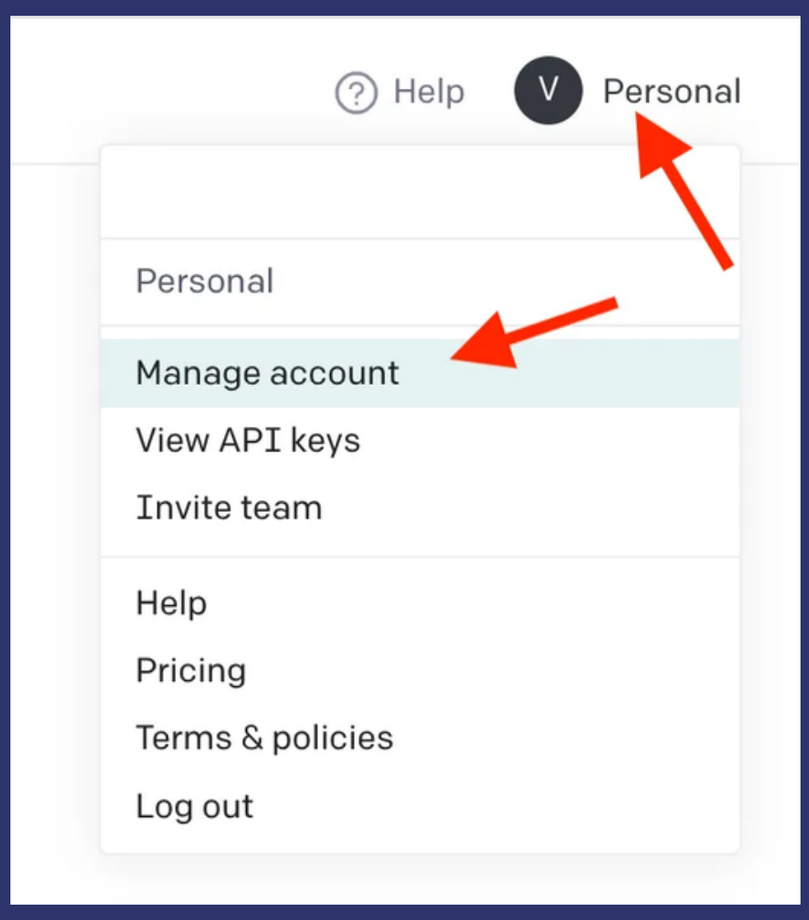


If you want to integrate ChatGPT to your Unity Project, you are at the right place.

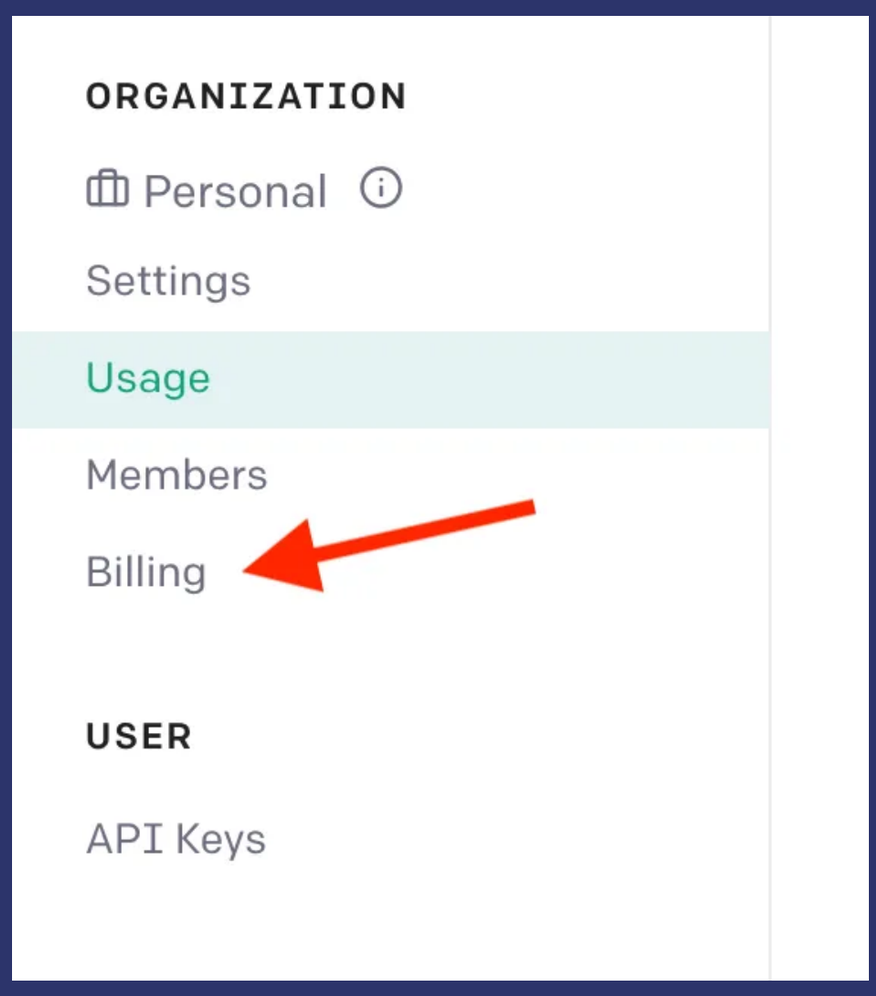
high-level architectural representation

Setting up a billing account with OpenAI

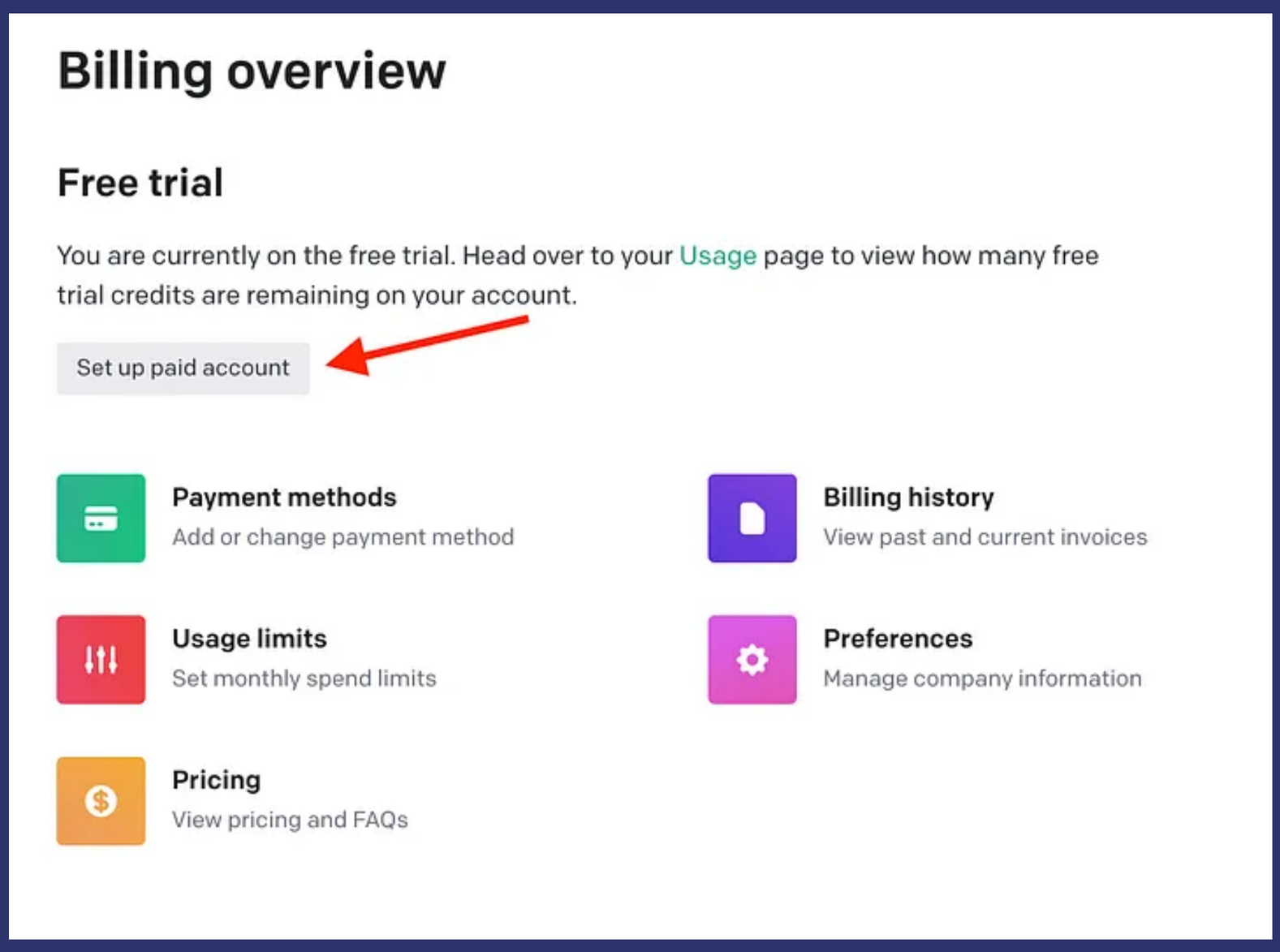
1. Go to platform.openai.com and log in.
2. Click on your profile name located in the top left corner and select “Manage account”.



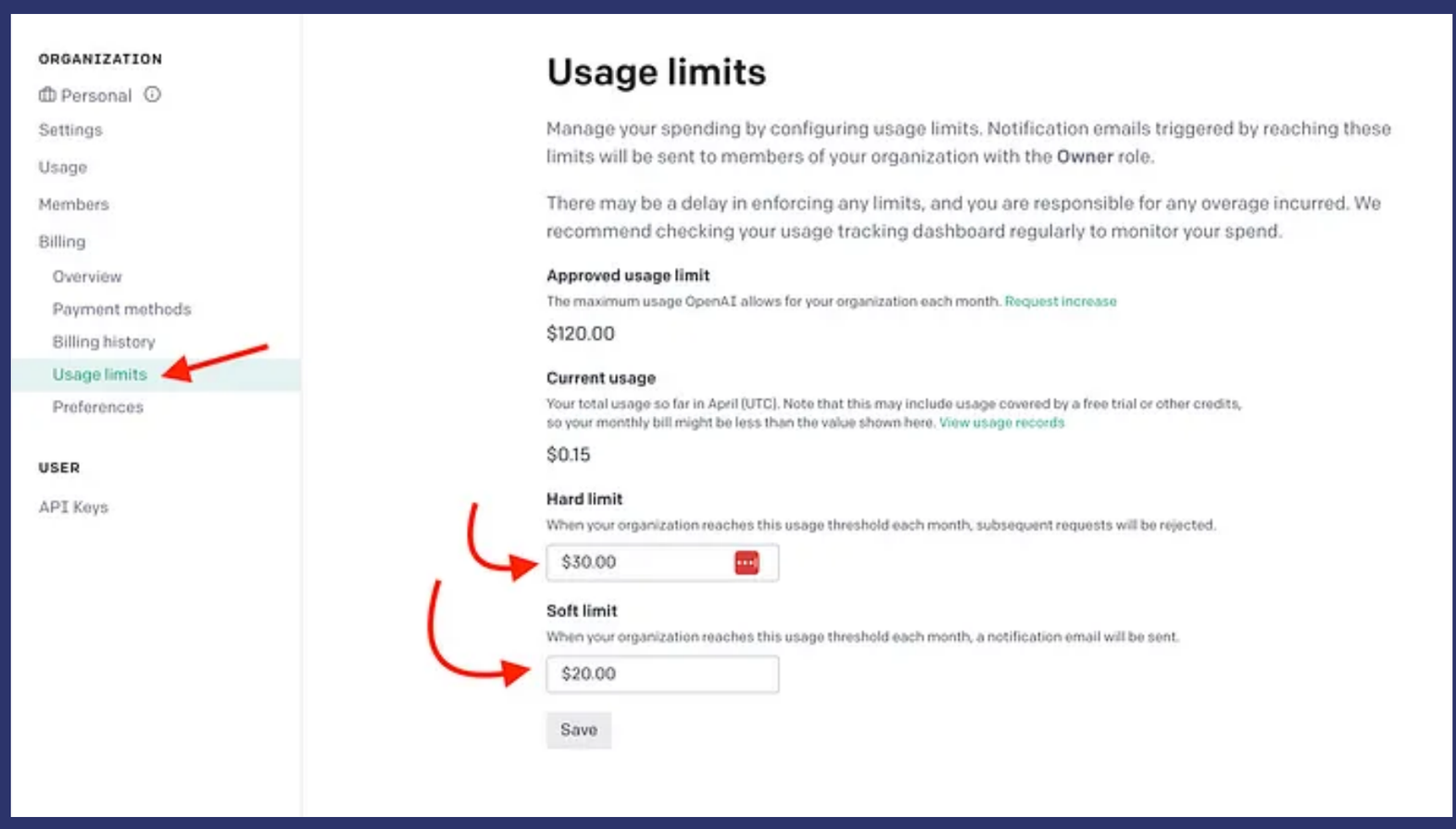
3. From the left side menu, select “Billing”.



4. Next, you need to click the button “Set up paid account”.



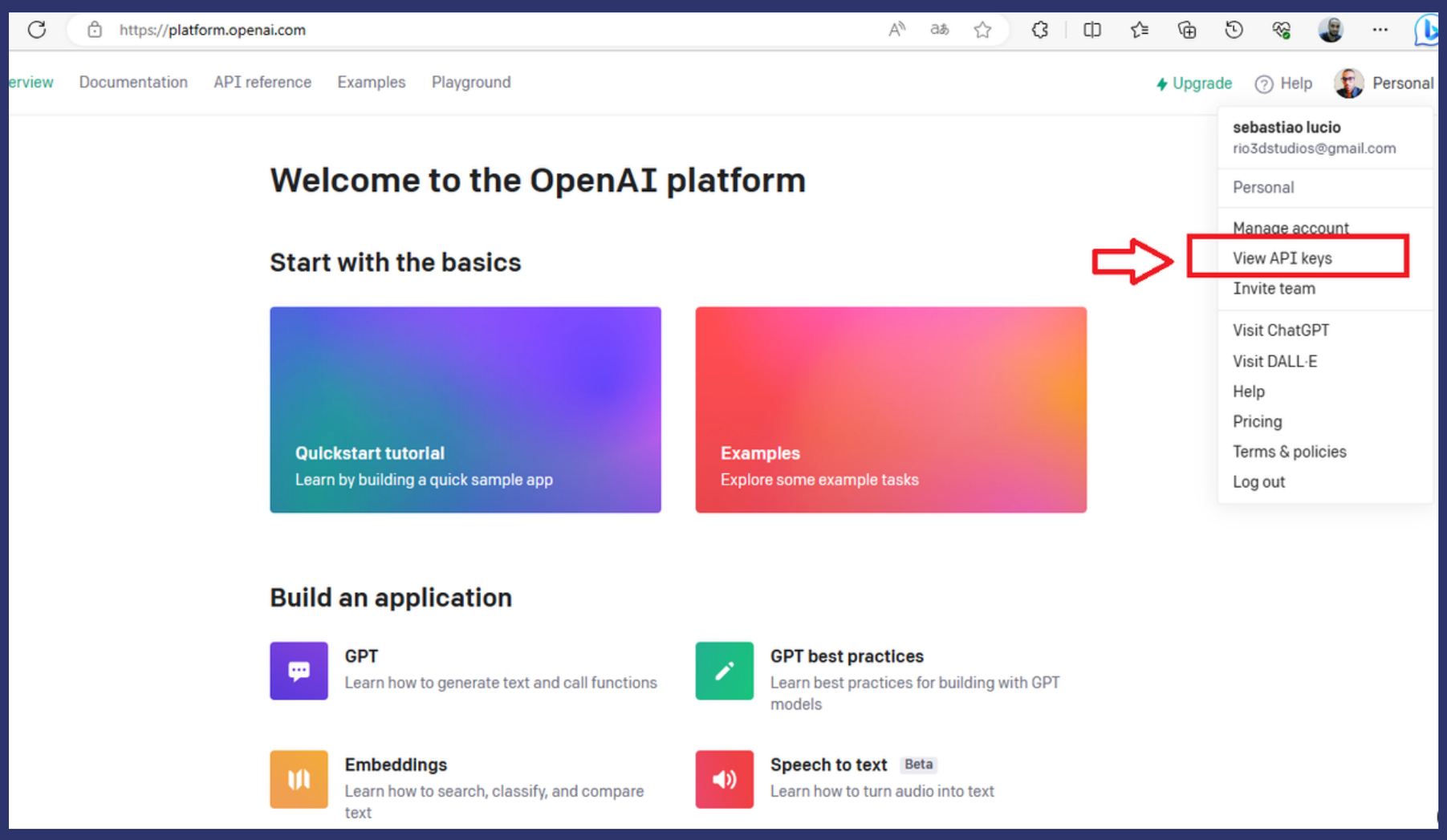
5. Your account will have a default usage limit. If you feel that is too high, we recommend setting a soft and a hard limit based on your needs.

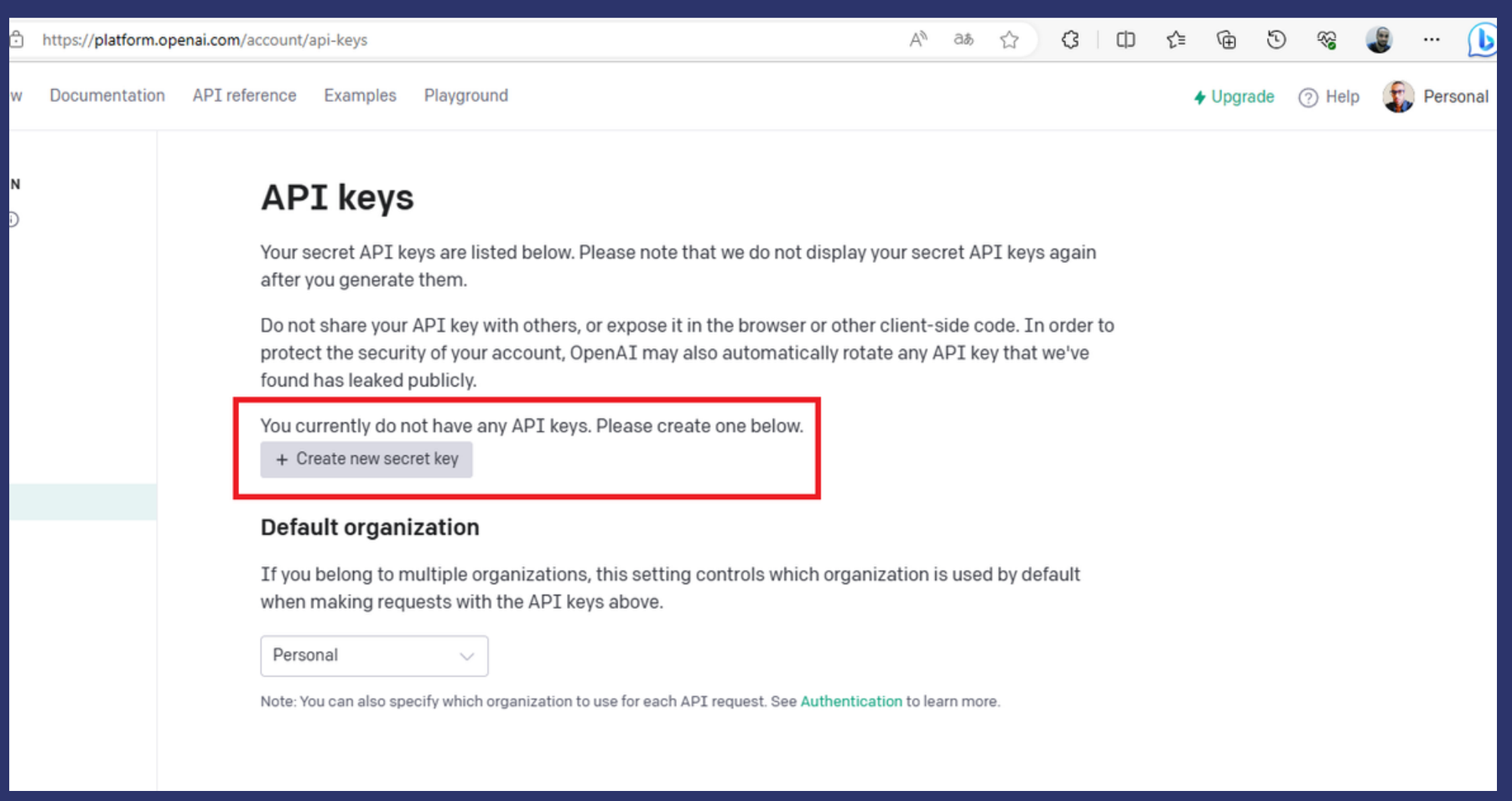


5. Your account will have a default usage limit. If you feel that is too high, we recommend setting a soft and a hard limit based on your needs.

Setting OPENAI_API_KEY on Node.js server

We will have to create an API key on the open AI platform





Copy the key -> in the folder of the nodeJS server, locate the config folder and inside this folder the api.json file, then seven the value of the variable "**OPENAI_API_KEY**" with your secret key

now we can make requests to the openIA API using the nodeJS server!

Node.js server

The required dependencies are imported and initialized

```
require('dotenv').config();
const express = require('express');
const app = express();
const http = require('http').Server(app);
const io = require('socket.io')(http);
const { Configuration, OpenAIApi } = require("openai");

const apiConfig = {
  OPENAI_API_KEY: process.env.OPENAI_API_KEY
};

const configuration = new Configuration({
  apiKey: apiConfig.OPENAI_API_KEY,
});

const messages = [];
const openai = new OpenAIApi(configuration);
```

Handling client messages:

```
socket.on('MESSAGE', async (_data) => {
  // Handling client message event
});
```

When the client sends a message, the 'MESSAGE' event is triggered. The server receives the message and adds it to the messages array. It then uses the OpenAI API to generate a response using the GPT-3.5 model and sends the response back to the client.

```
messages.push({ role: "user", content: data.message});

const completion = await openai.createChatCompletion({
  model: "gpt-3.5-turbo",
  messages:  messages,
});

const completion_text = completion.data.choices[0].message.content;
 console.log(completion_text);

 // send the completion text to the client
socket.emit('UPDATE_MESSAGE', completion_text);
```

Contact us

rio3dstudios@gmail.com

rio3dstudios.com