DOCUMENTATION

WEBGL MULTIPLAYER ONLINE CLIENT

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

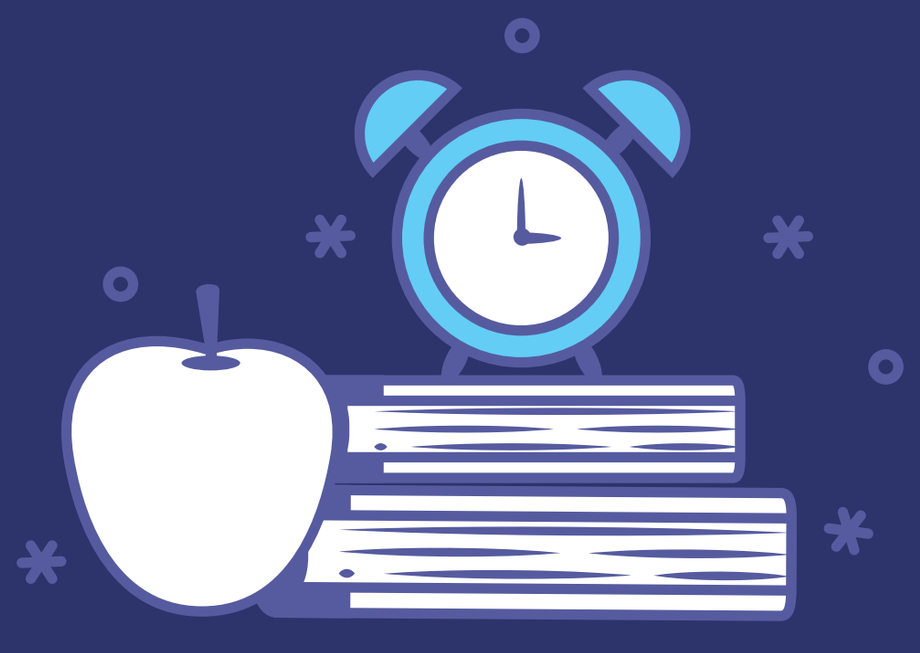
HOW TO DEVELOP

- Game architecture
- Network Manager Class
- Node JS Server

USAGE SAMPLES

- Basic Example
- 2D Shooter
- 3D Shooter

CONTACT & SUPPORT

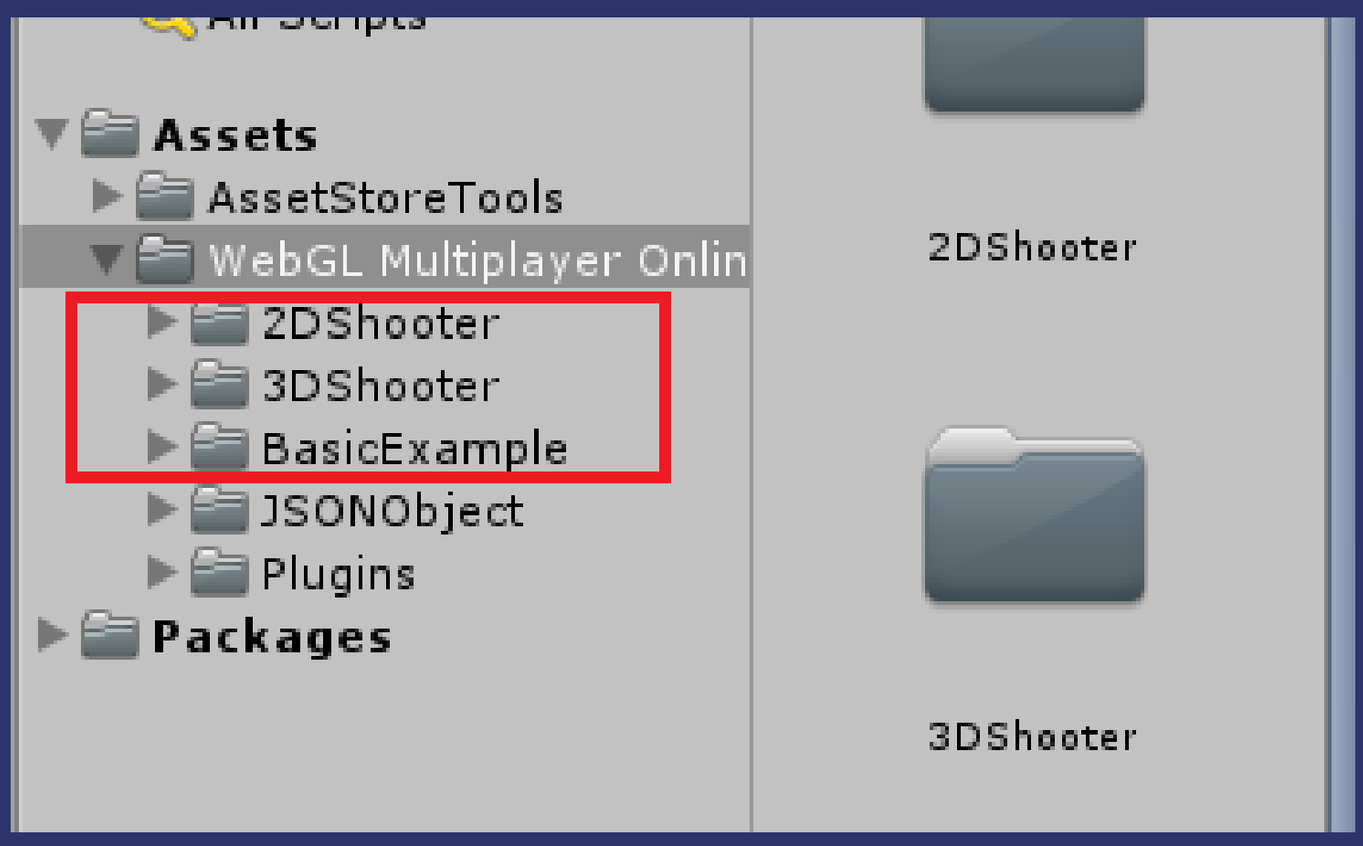


QUICK START

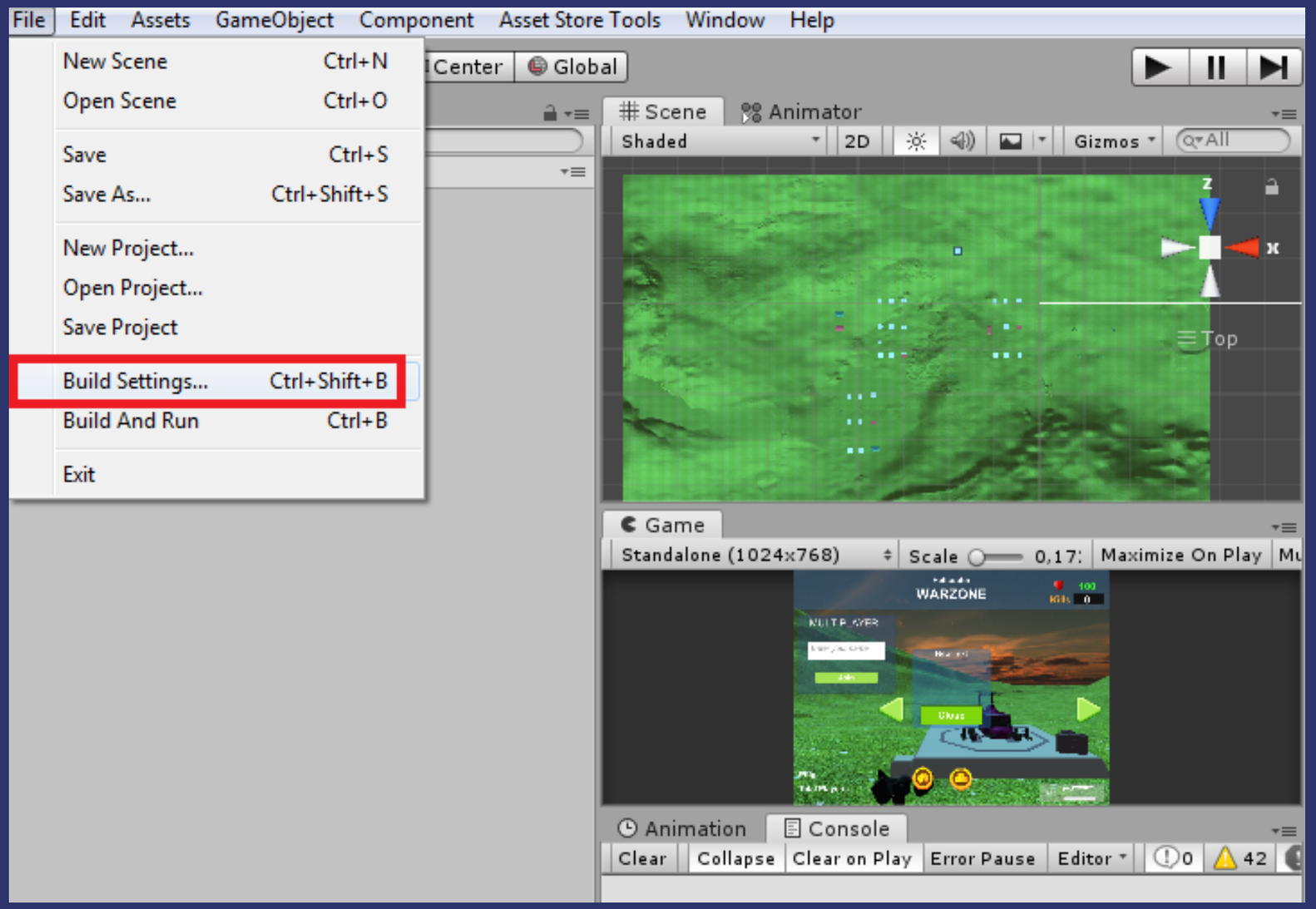
SETUP WEBGL GAME

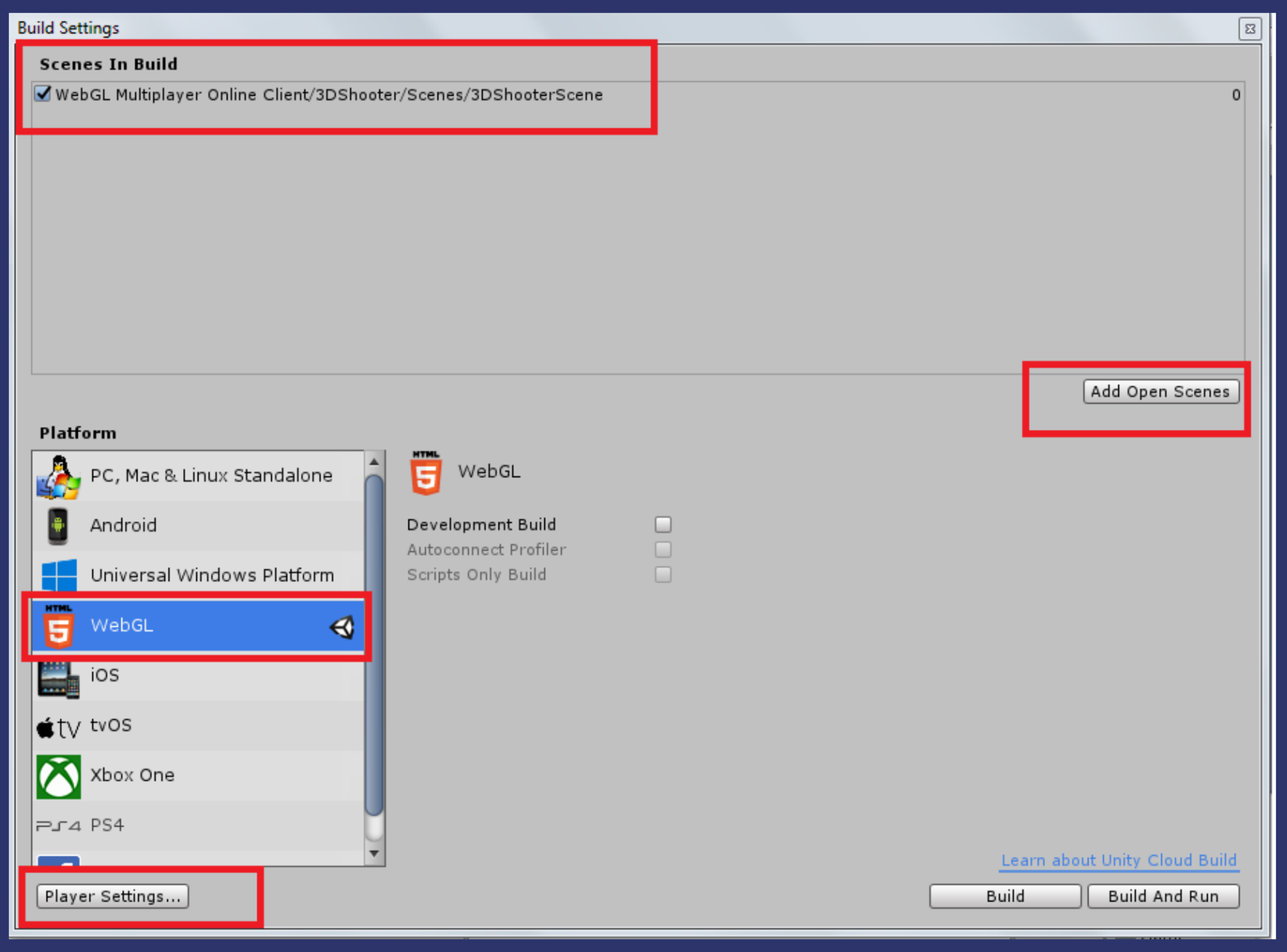
This basic project will allow you to build a WebGL HTML5 multiplayer online games for browsers like Google Chrome , firefox, etc. using the external calls (Application.ExternalCall()) provided by the unity.
This mechanism allows a unity WebGL html5 client to communicate externally with a java script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

In the project, choose an example of your preference and select the main scene of the example:



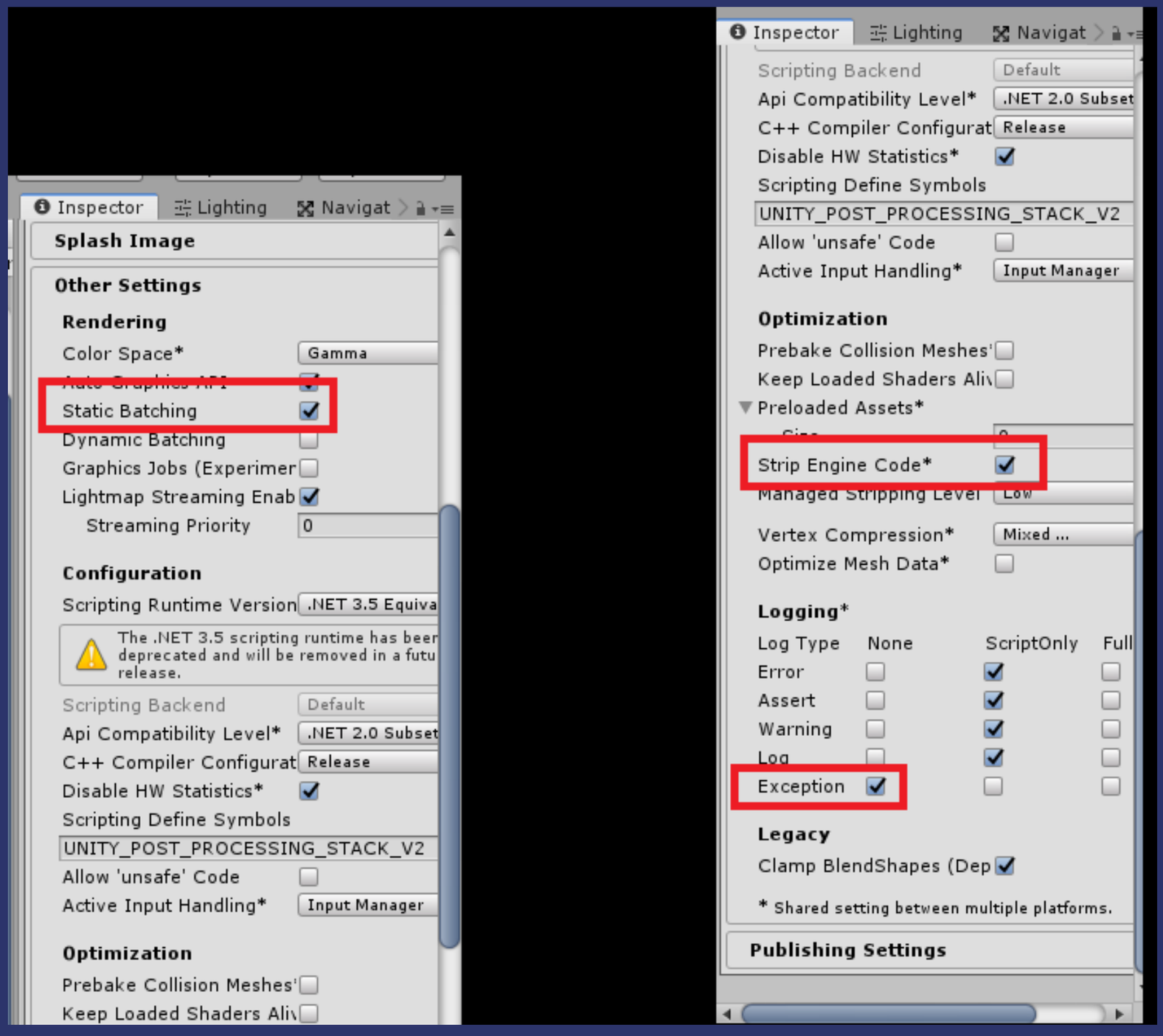
In Unity export the project for the WebGL platform in a folder called "public" located somewhere in your computer (Desktop e.g.)



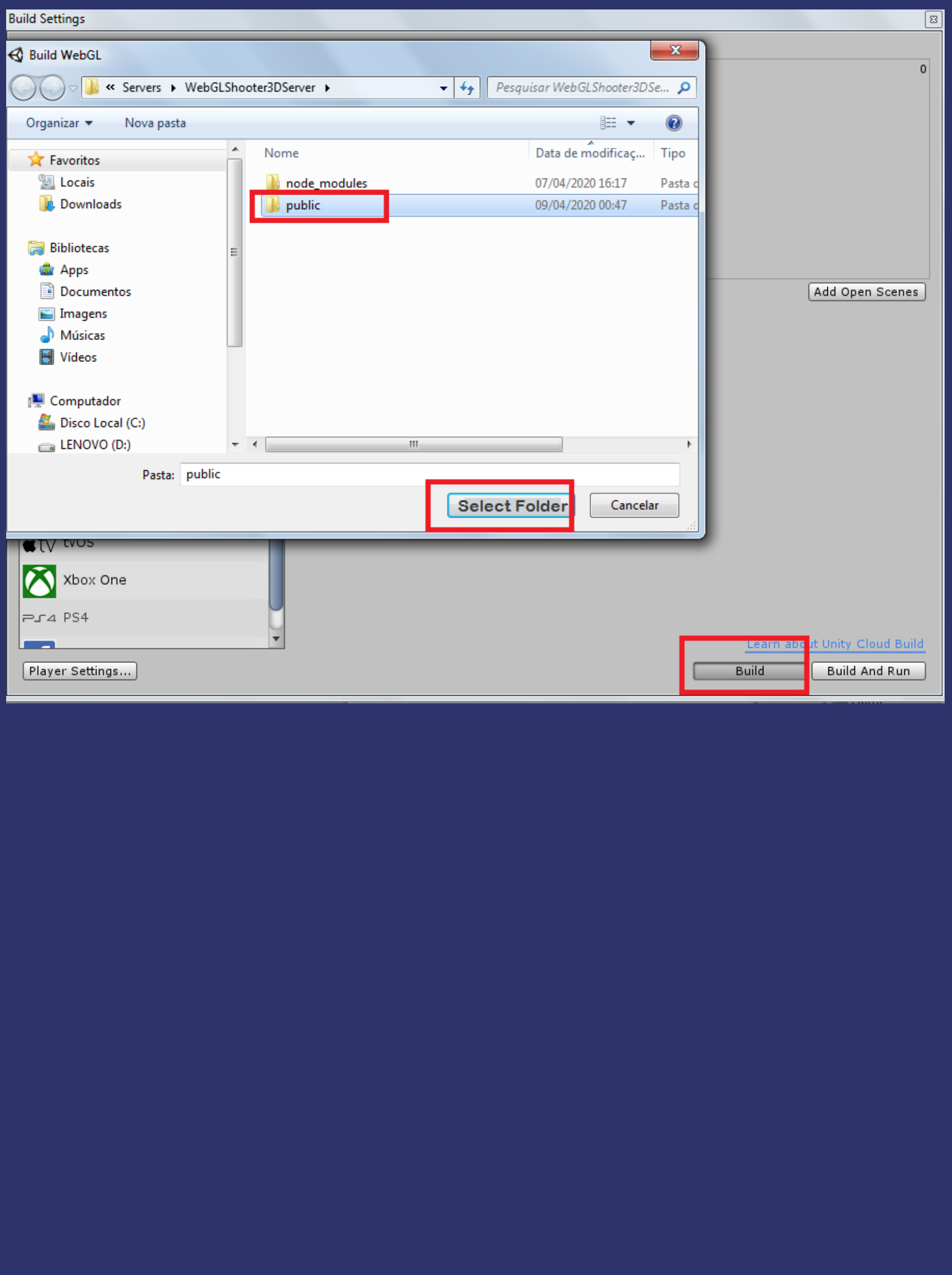


Optimization for WebGL

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.



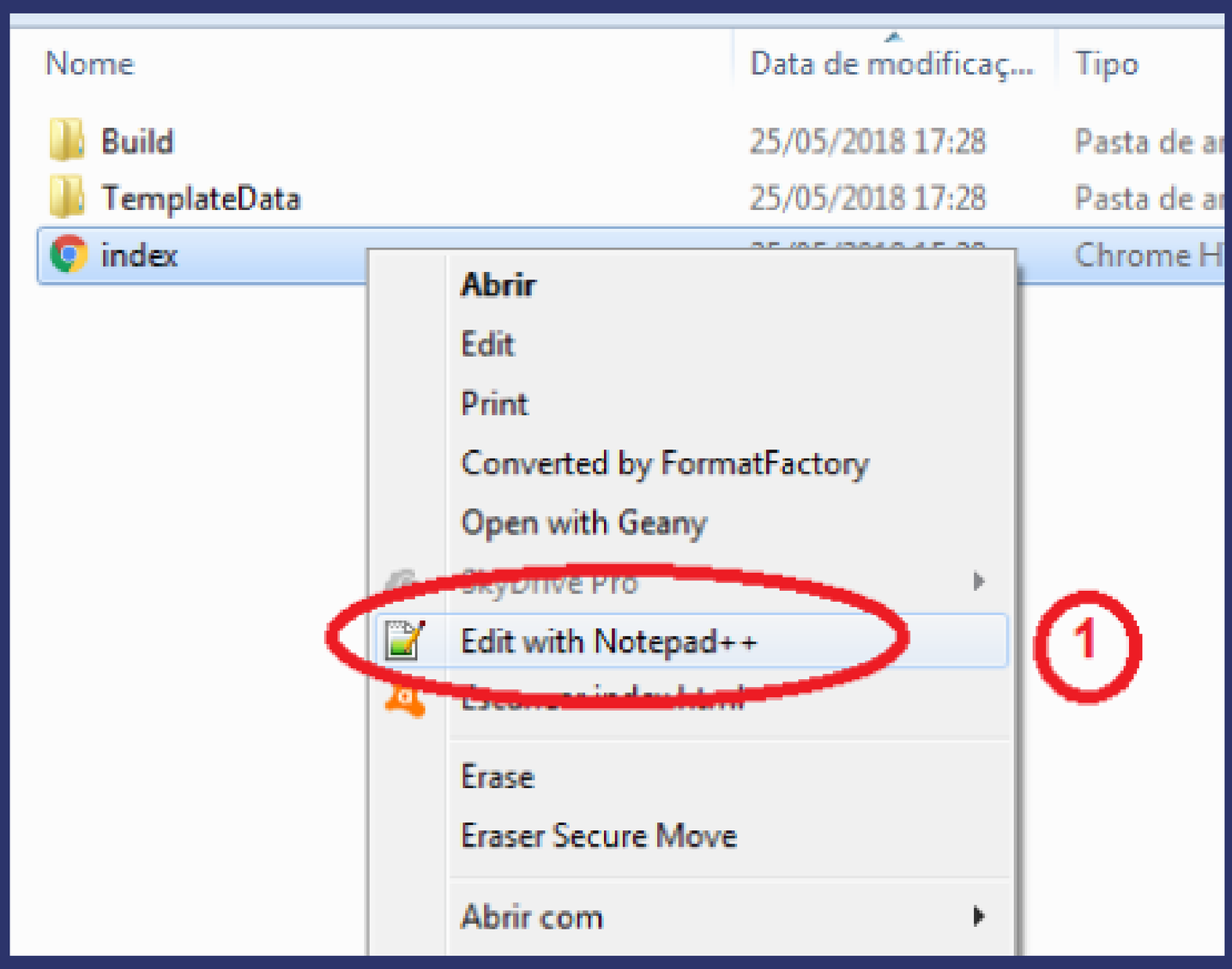
In build settings, go to build to compile a new version of the game



Configuring the index.html page

After export the project, open the public folder where the game was exported and locate the index.html file.
Open the index.html file with the editor of code of your preference (in my case notepad ++) add the two code lines in index.html header:

```
12      <script src="/socket.io/socket.io.js"></script>
13      <script src="client.js"></script>
14
```



setup the index.html page

index.html

```
<!DOCTYPE html>
<html lang="en-us">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Unity WebGL Player | Basic WebGL Multiplayer Online Project</title>
    <link rel="shortcut icon" href="TemplateData/favicon.ico">
    <link rel="stylesheet" href="TemplateData/style.css">
    <script src="TemplateData/UnityProgress.js"></script>
    <script src="Build/UnityLoader.js"></script>

    <script src="/socket.io/socket.io.js"></script>
    <script src="client.js"></script>

    <script>
      var gameInstance = UnityLoader.instantiate("gameContainer", "Build/public.
    </script>
  </head>
  <body>
    <div class="webgl-content">
      <div id="gameContainer" style="width: 960px; height: 600px"></div>
      <div class="footer">
        <div class="webgl-logo"></div>
        <div class="fullscreen" onclick="gameInstance.SetFullscreen(1)"></div>
        <div class="title">Basic WebGL Multiplayer Online Project</div>
      </div>
    </div>
  </body>
```

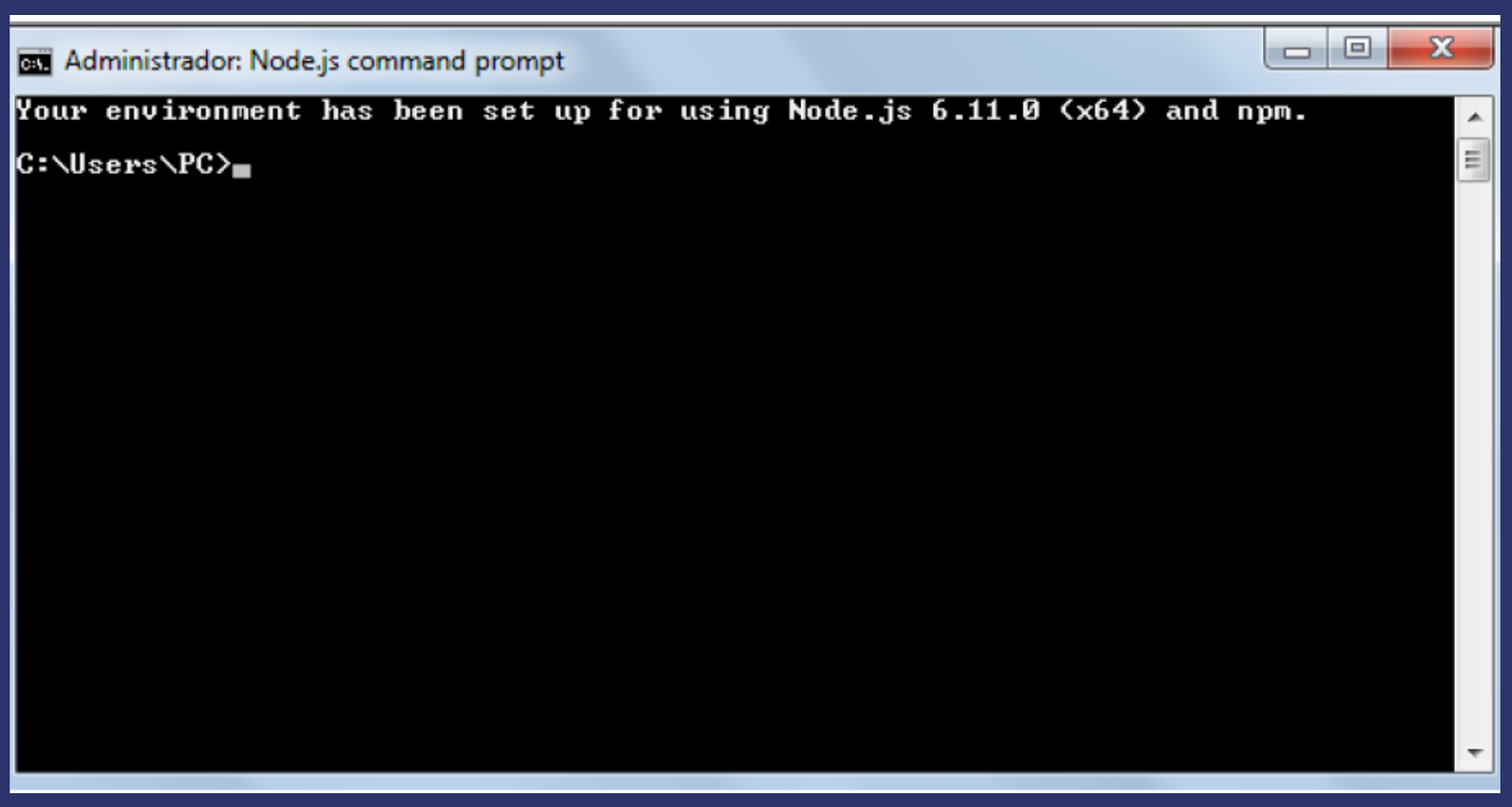
SETUP NODEJS SERVER

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

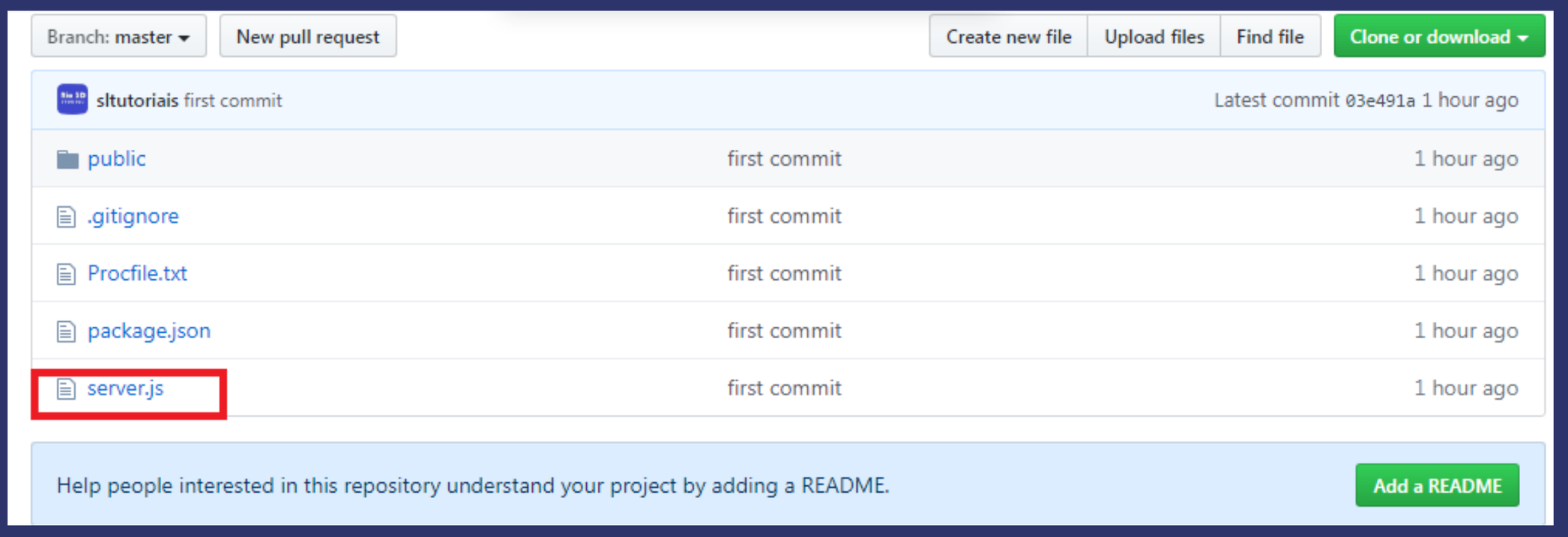
To execute Node.JS in windows , go to windows button and type in the search field : " node" , selects the Node.JS command prompt .

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



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

[Basic Example Server]
[2D Shooter Server]
[3D Shooter Server]

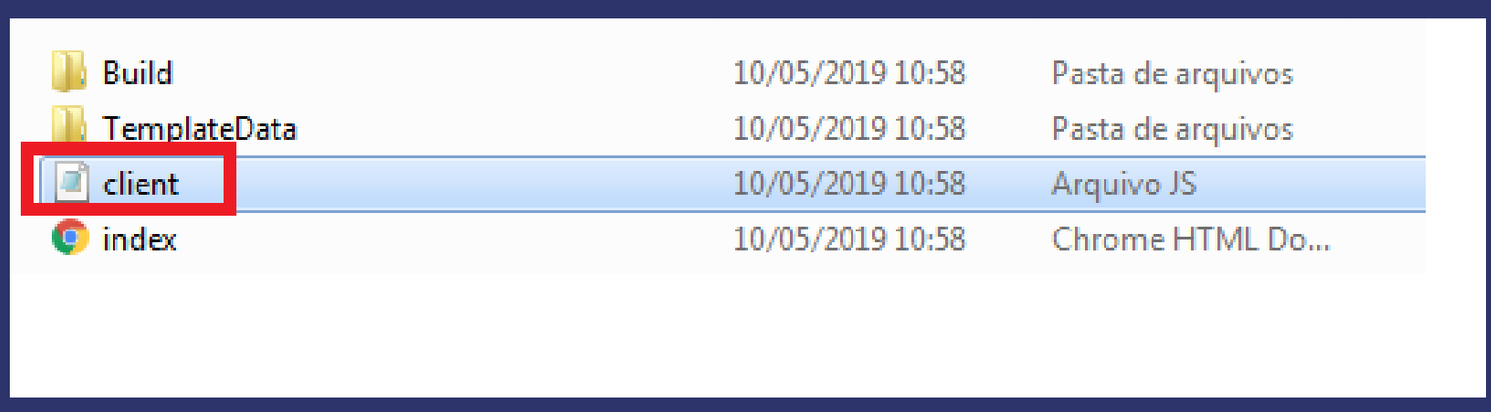


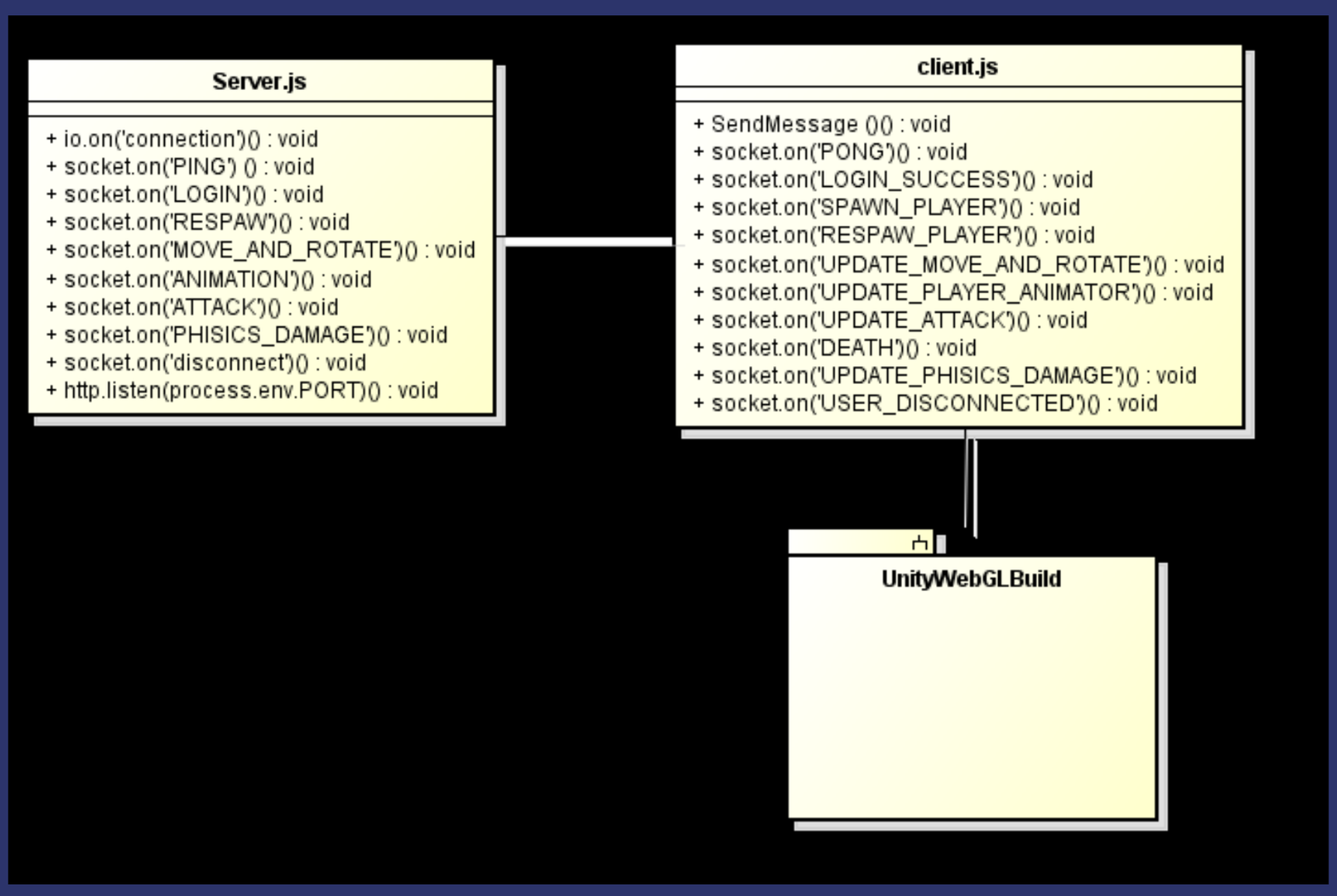
The client.js file

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.

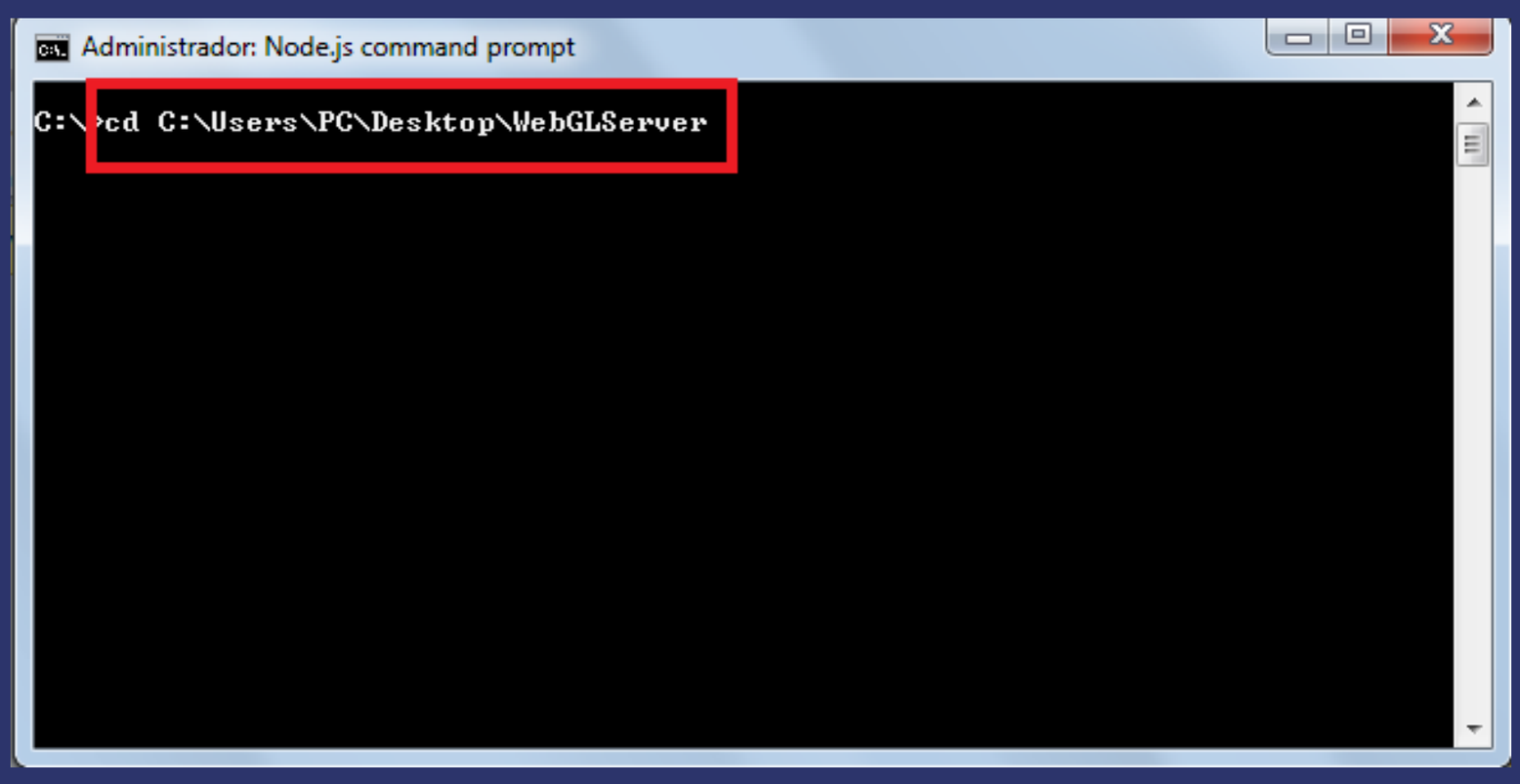




Running the server on a local machine

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

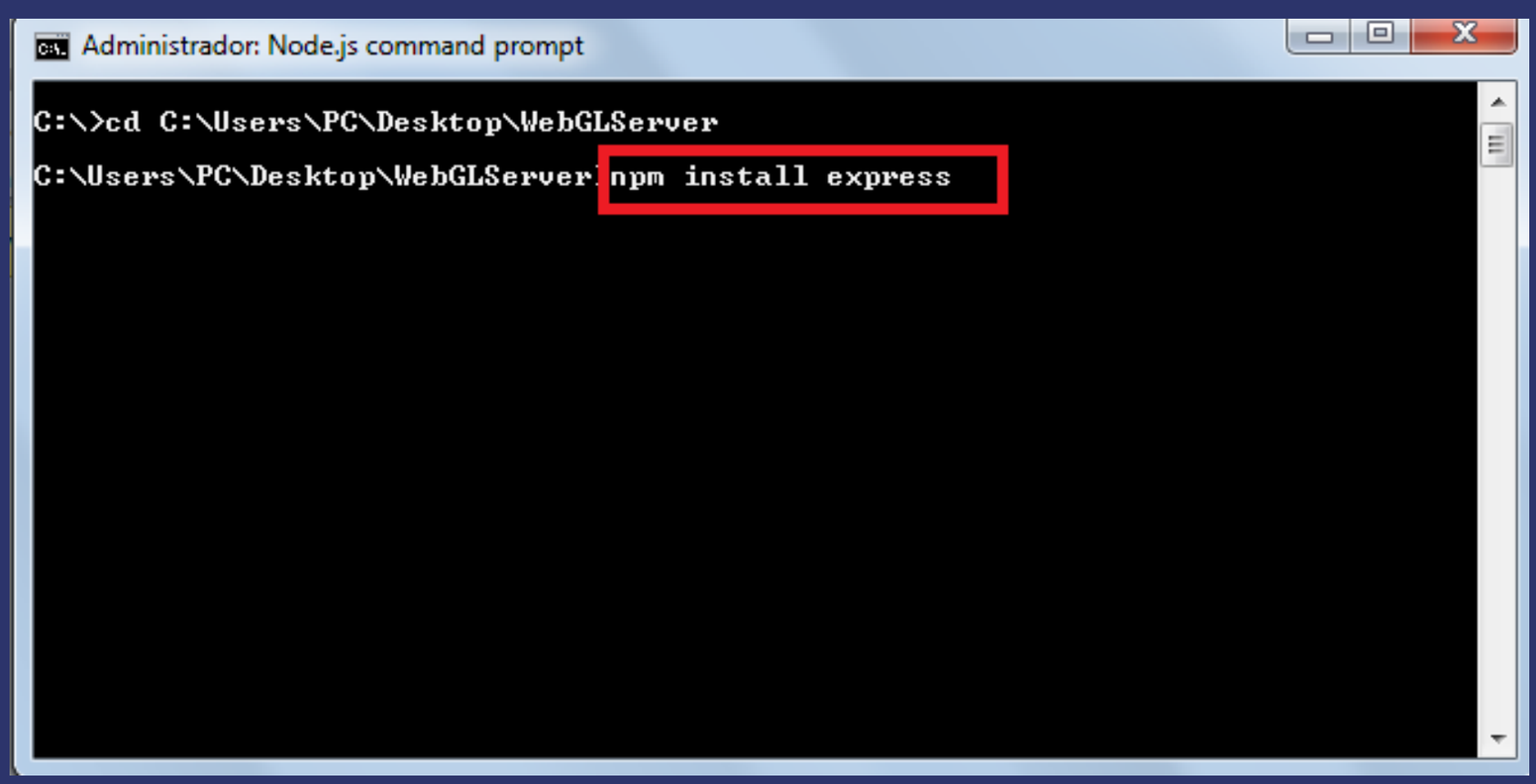
- Navigate until the WebGLServer folder typing the command (cd + space_bar + url_folder)
- eg: cd C:\Users\PC\Desktop\WebGLServer



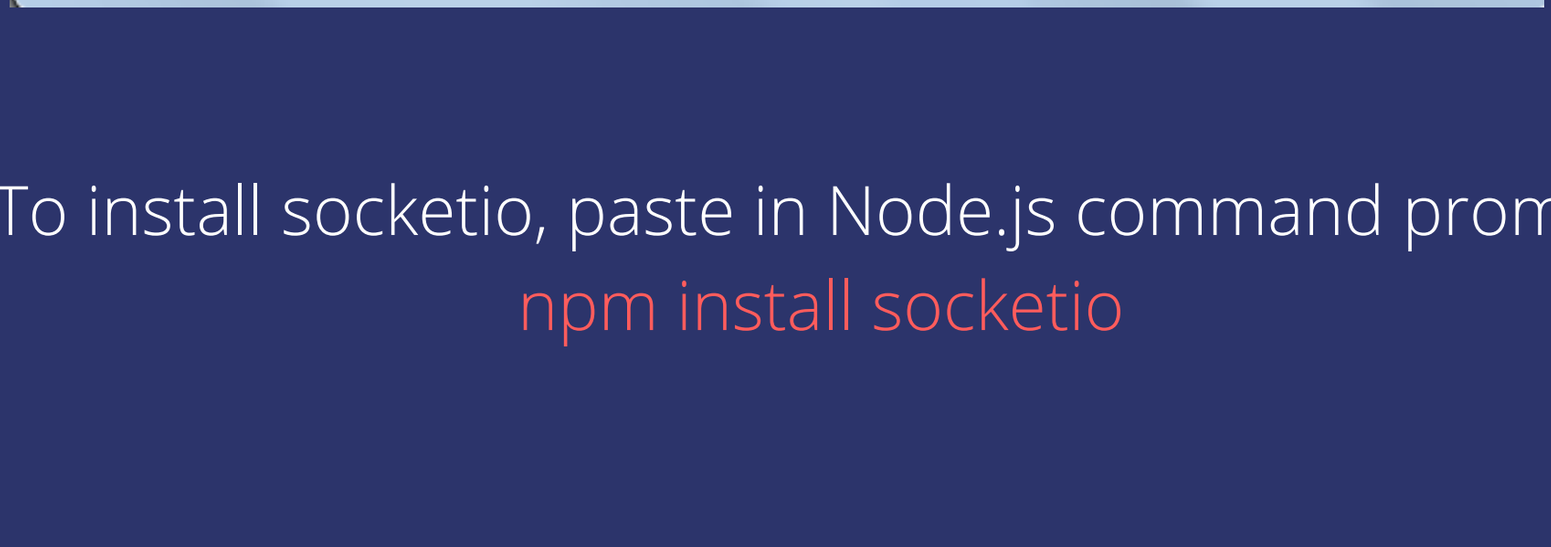
Soon afterwards we should install the necessary modules for the operation of the java script server.js through command npm install.
We will Install the modules: express(NodeJS Framework), socketio (sockets lib) and shortid (random numbers generator)

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

- To install express, paste in Node.js command prompt:
- npm install express

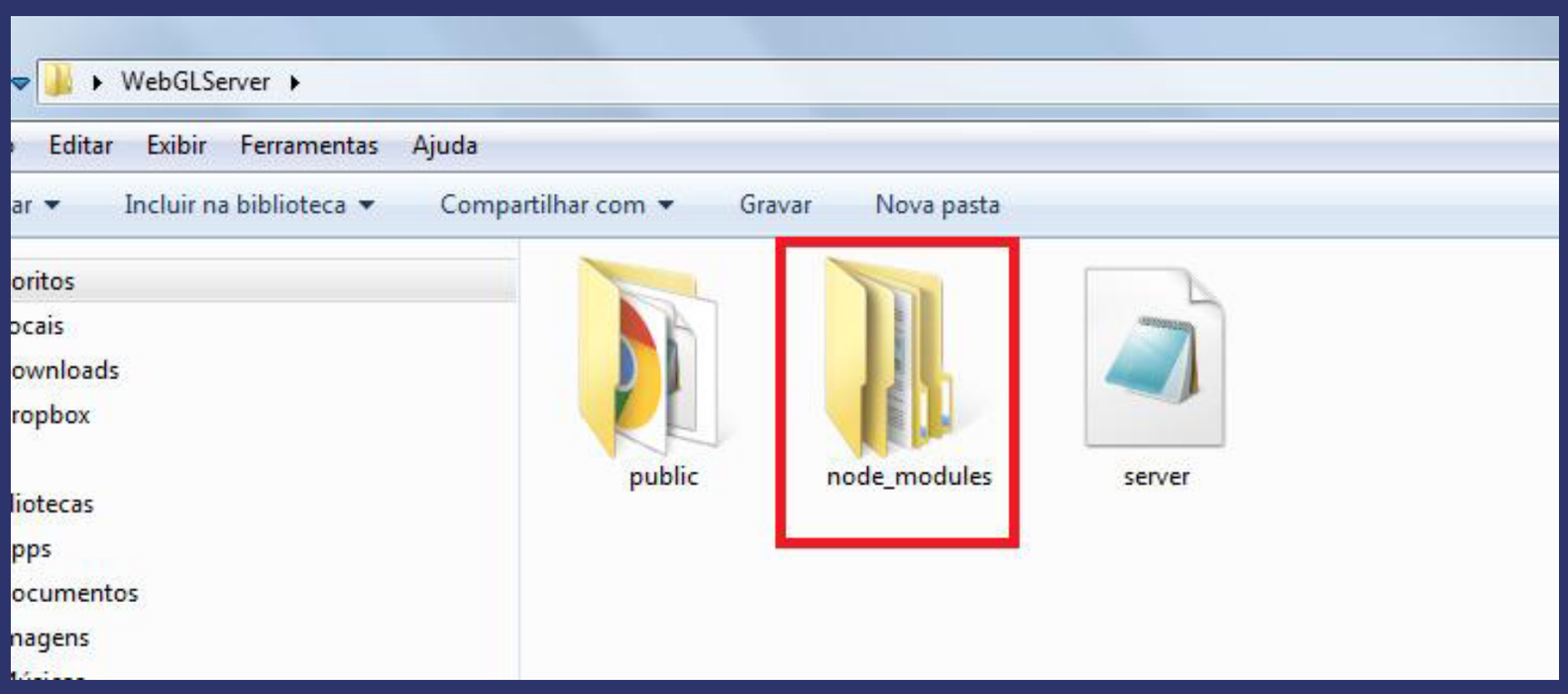


- To install socketio, paste in Node.js command prompt:
- npm install socketio



- To install shortid, paste in Node.js command prompt:
- npm install shortid

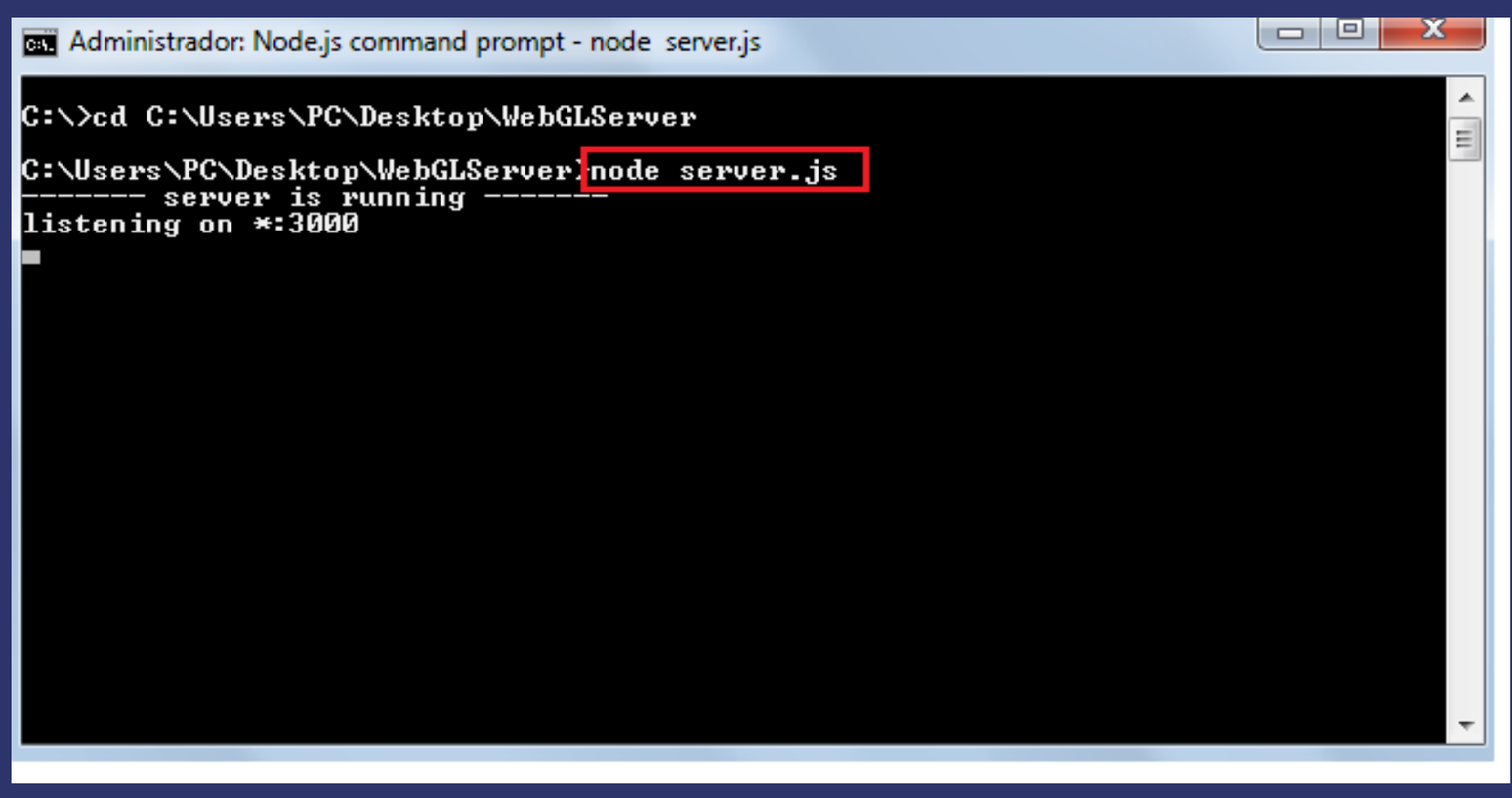
Notice that will go appears a new folder denominated node_modules inside of the WebGLServer folder, inside you will find the modules express, socketio and shortid.



Running the server on a local machine

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

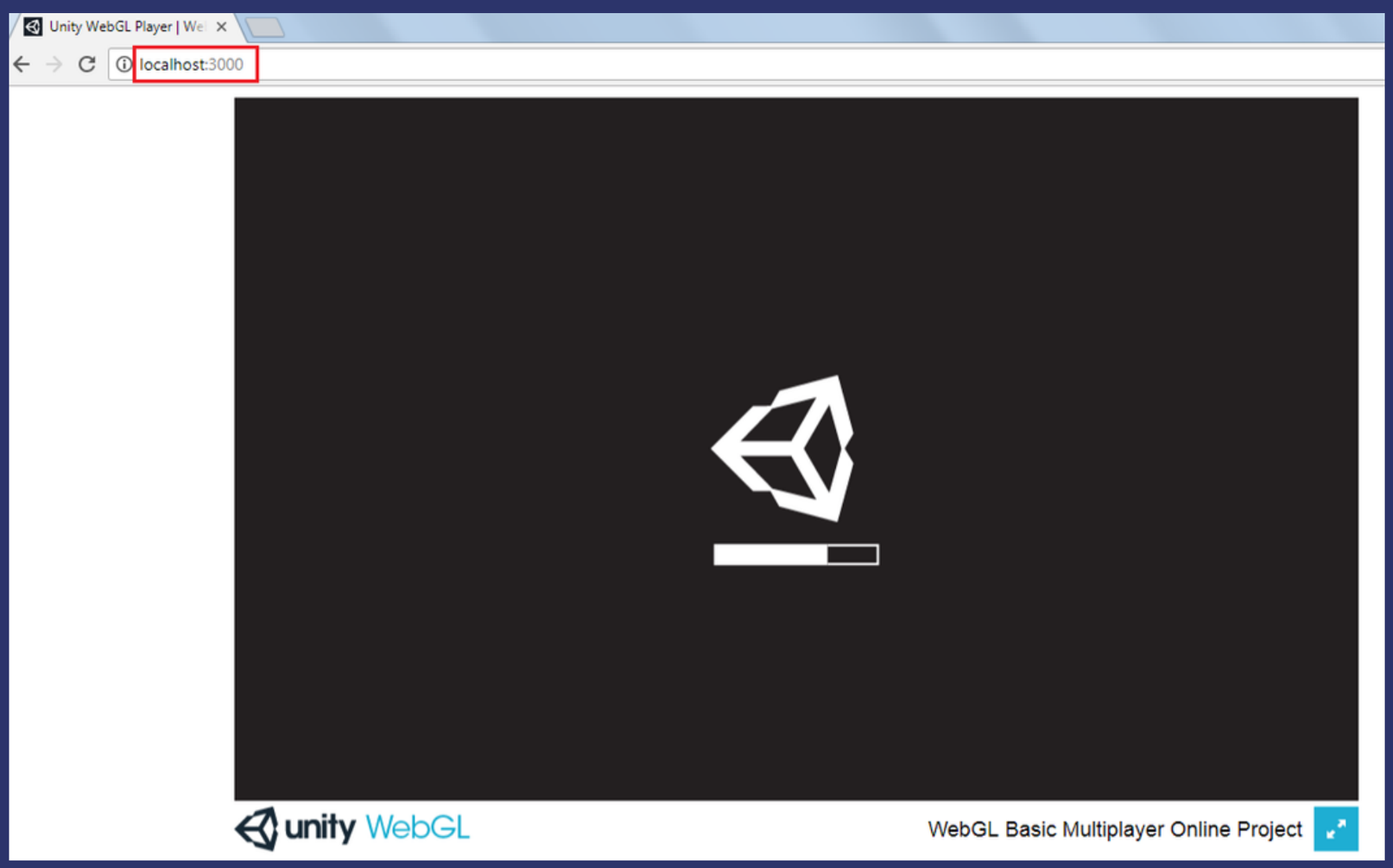
- node server.js

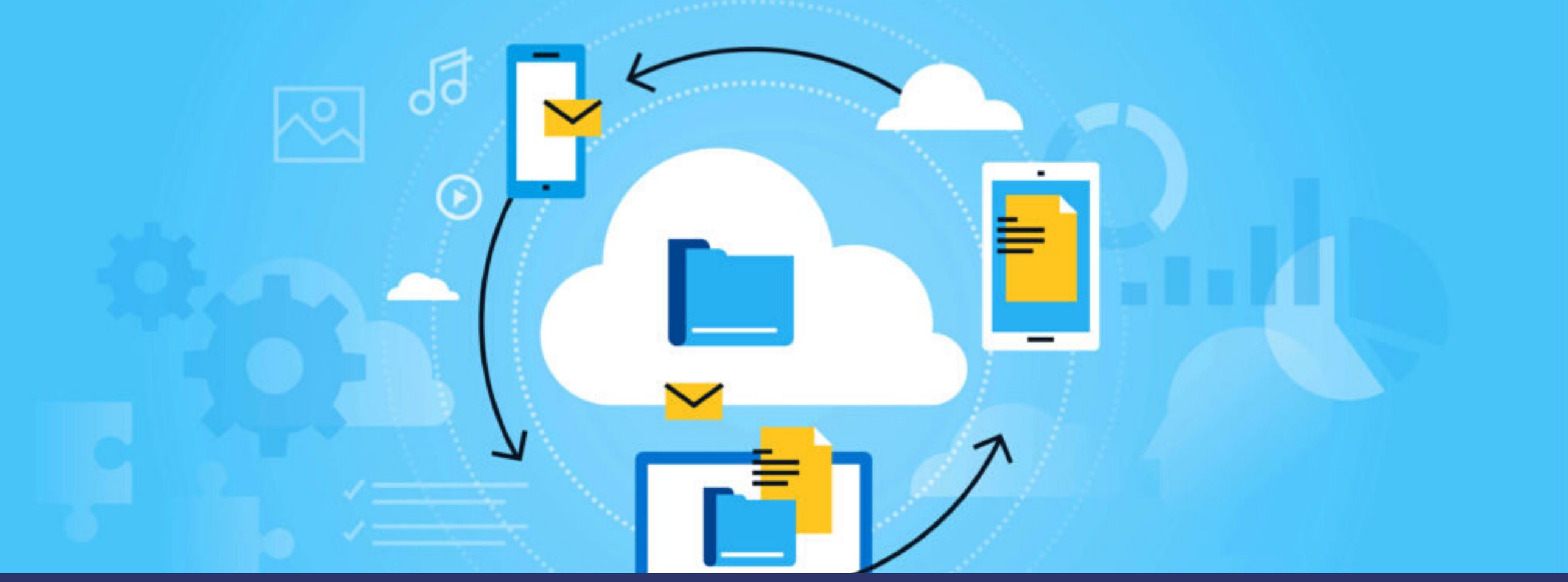


To see the game running open the Google Chrome and type the url:

localhost:3000

Running the server on a local machine





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

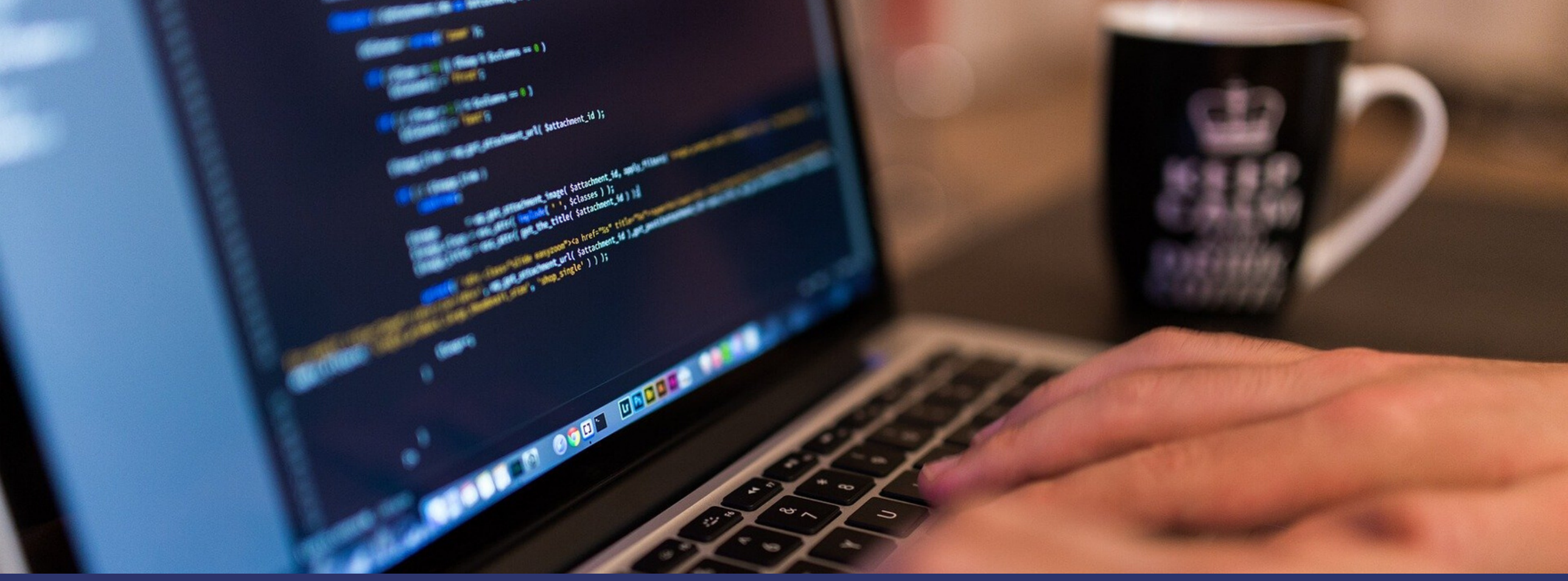
How deploy nodejs server to heroku tutorial 2

How deploy nodejs server to heroku tutorial 3

How deploy nodejs server to Amazon AWS

How deploy nodejs server to Google Cloud

how deploy nodejs server to Google Cloud tutorial 2



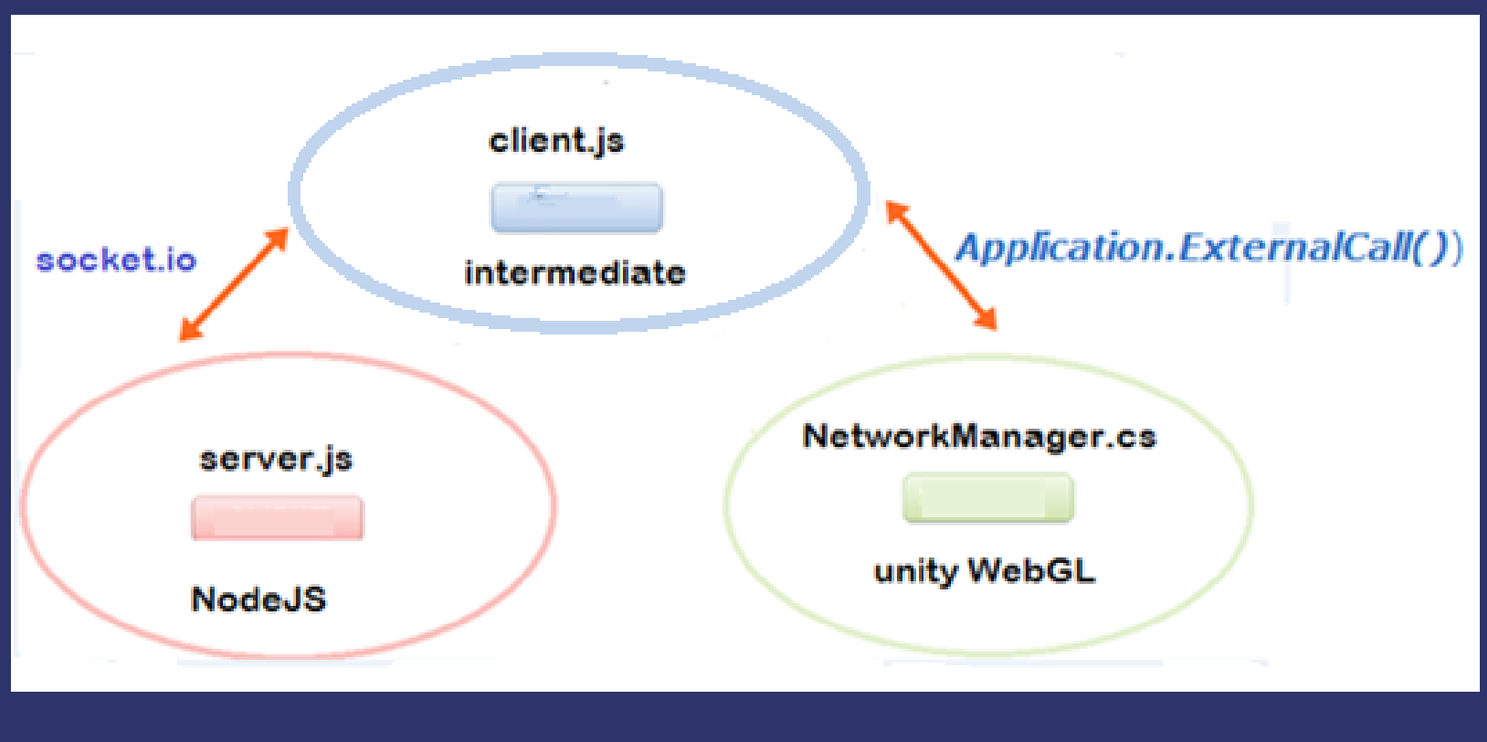
How to develop

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

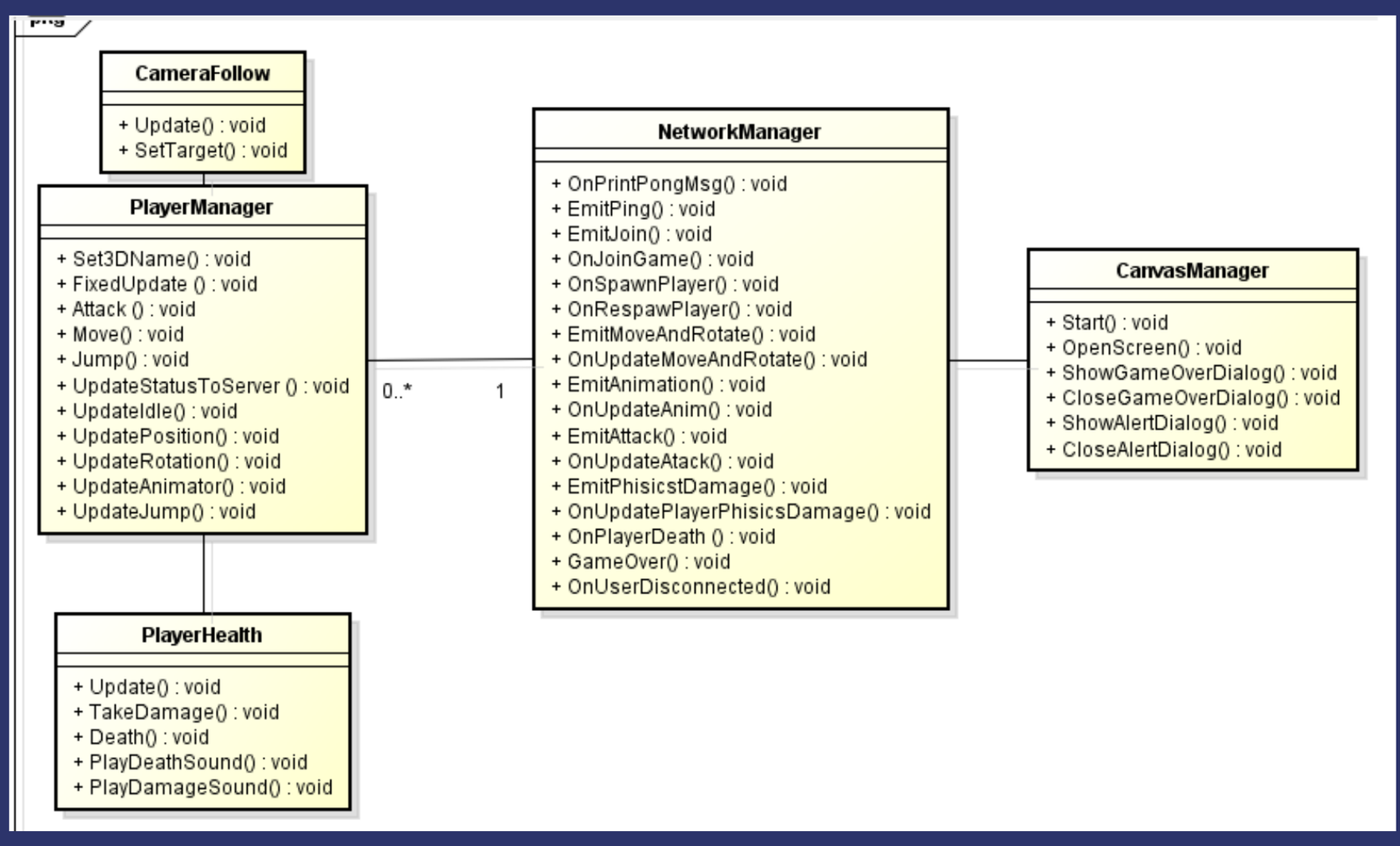
- The Front-end: all visual, game mechanics and physics will be done in unity. Whatever game you have in mind, we will just use a main class called Network Manager to do all the interaction with the server.

- 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.

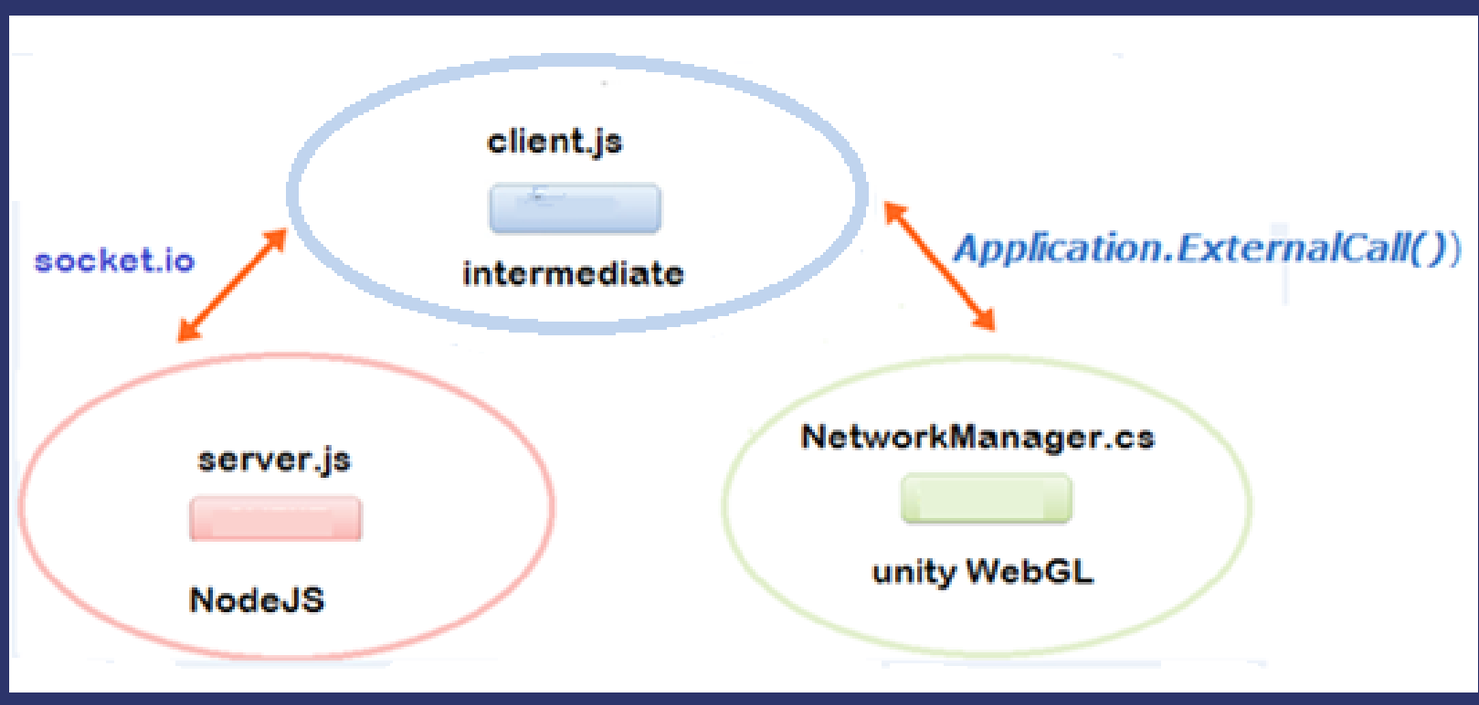


CONNECT TO SERVER

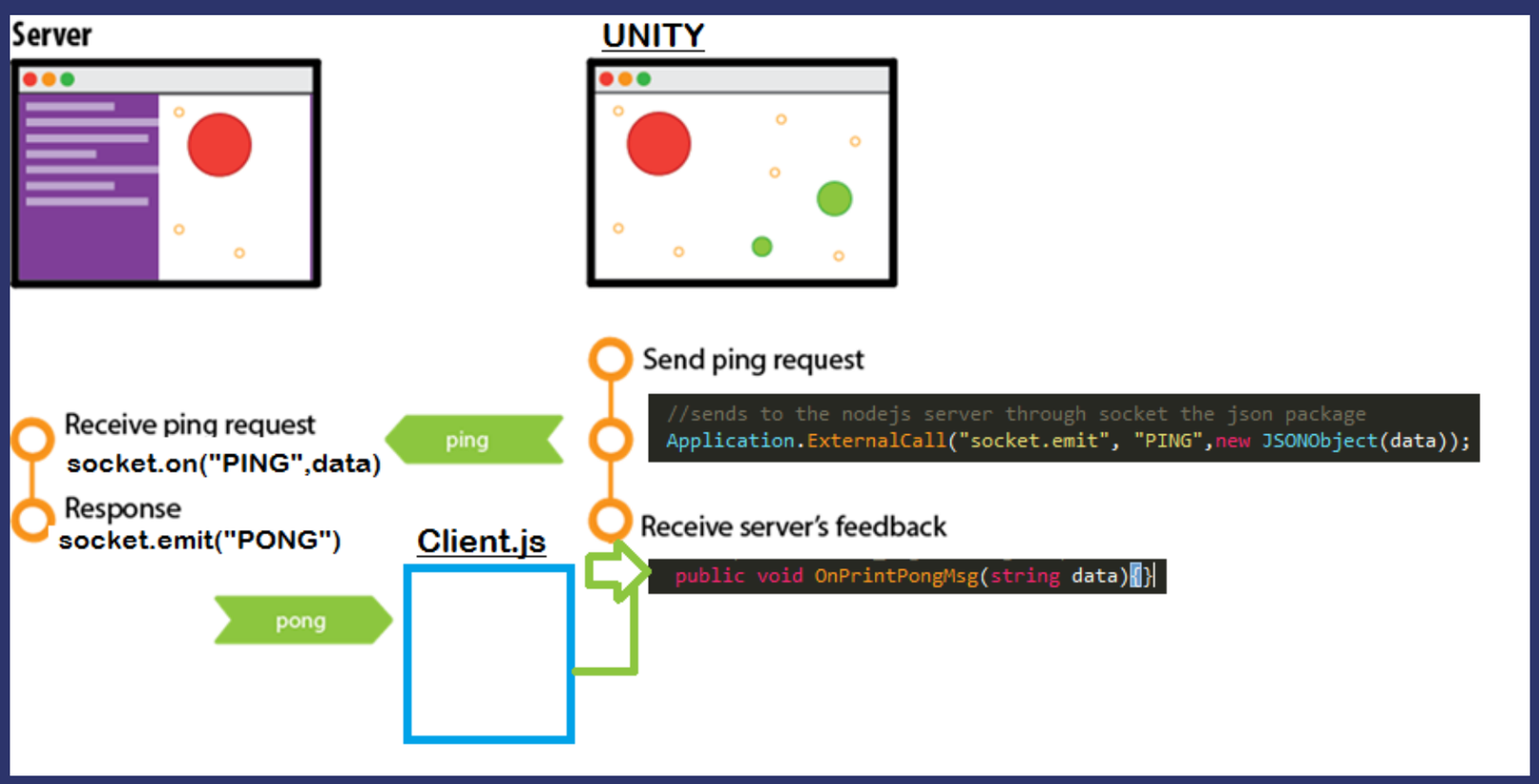
Connecting to a socket.io server is just one line:

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

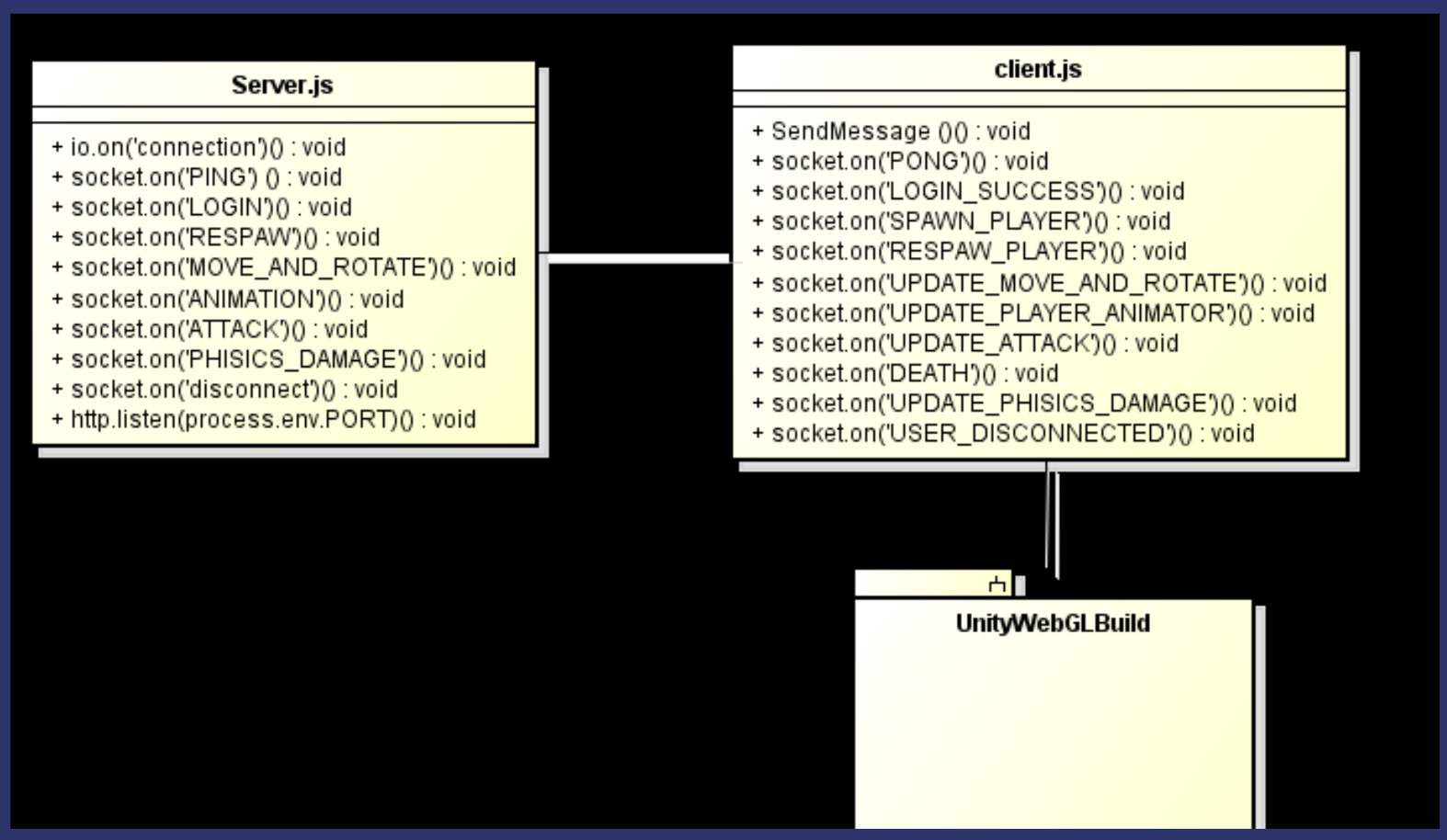
```
// <summary>
/// sends ping message to server.
/// </summary>
public void EmitPing() {

    //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 on the assetstore 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

Code snippet in WebGL

```
// <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!!!!";

  JSONObject jo = new JSONObject (data);

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

}
```

same piece of code in socketIO for Unity

```
/// take a look in PlayMakerSocketIOEvents script
/// </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!!!!";

  startTime = Time.time;

  JSONObject jo = new JSONObject (data);

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


}
```

Solution!

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

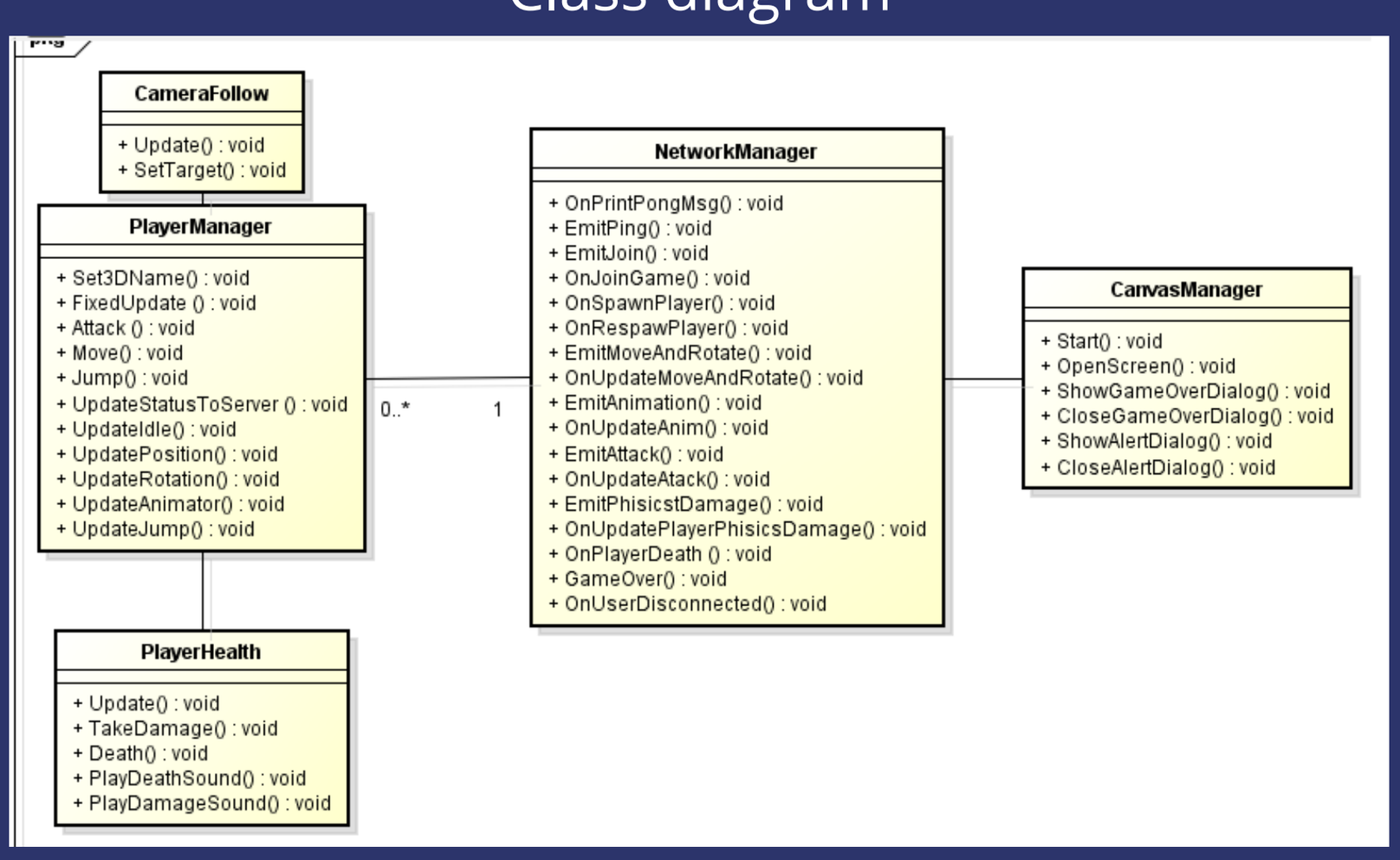
to help you develop your game we will make the same examples available for free in the socketIO asset free of charge. You can download it through the link below:

Download 2D and 3D shooter in socketIO:

- Unity package
- NodeJS Servers

BasicExample

Class diagram



Network Manager

```
public class NetworkManager : MonoBehaviour {


  private bool _isListenPortLogged = false;
  //Variable that defines comma character as separator
  static private readonly char[] Delimiter = new char[] {','};
  //useful for any gameObject to access this class without the need of instances her or you declare her
  public static NetworkManager instance;
  //flag which is determined the player is logged in the arena
  public bool onLogged = false;
  //store localPlayer
  public GameObject myPlayer;
  public string local_player_id;
  //store all players in game
  public Dictionary<string, PlayerManager> networkPlayers = new Dictionary<string, PlayerManager>();
  //store the local players' models
  public GameObject[] localPlayersPrefabs;
  //store the networkplayers' models
  public GameObject[] networkPlayerPrefabs;
  //stores the spawn points
  public Transform[] spawnPoints;
  //Standard Assets\Cameras\Prefabs\MultipurposeCameraRig.prefab
  public GameObject camRigPref;
  public GameObject camRig;
  public bool isGameOver;
```

```
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
```

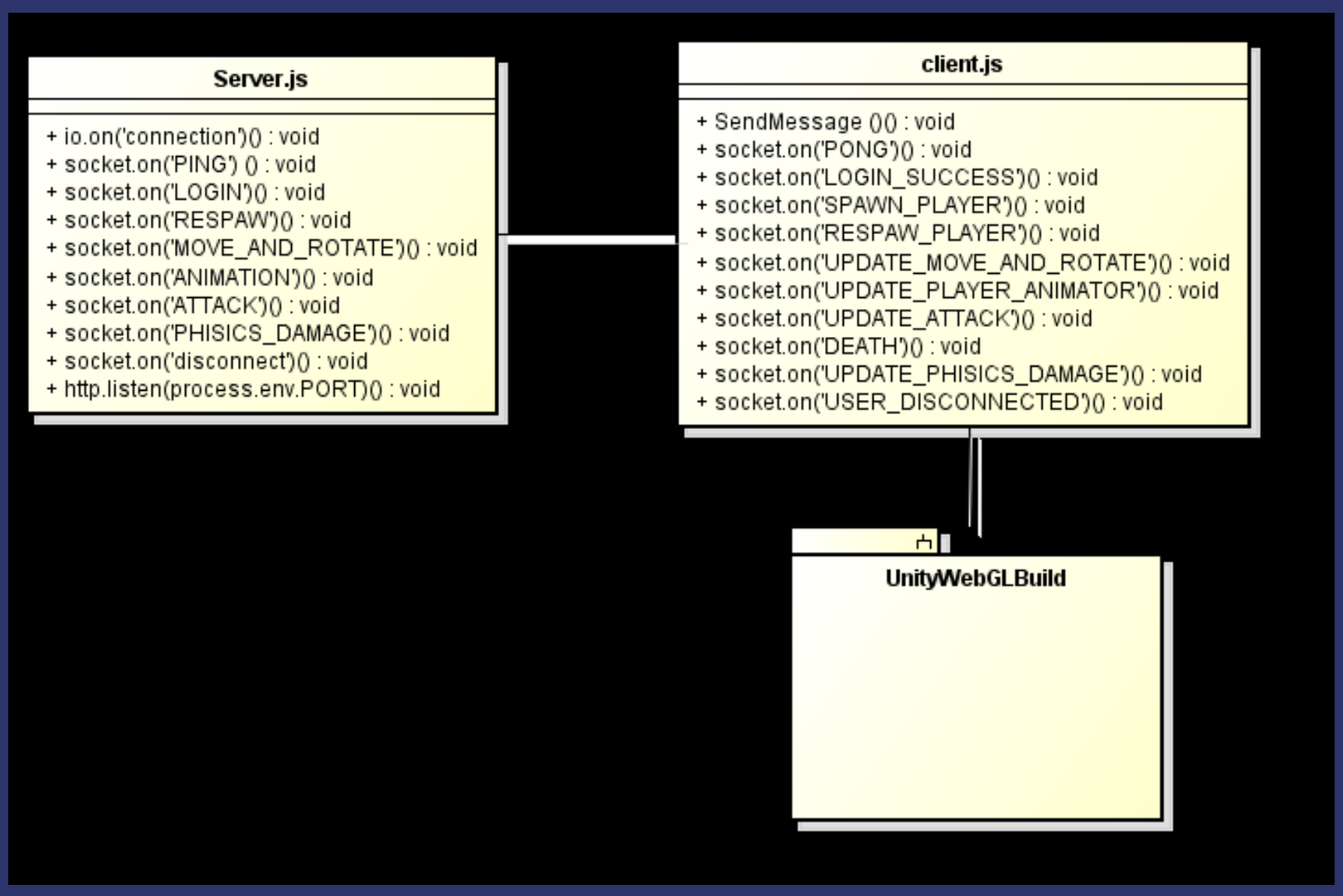
Network Manager

```
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
64
65      /// <summary>
66      /// Emits the local player animation to Server.js.
67      /// </summary>
68      /// <param name="_animation">Animation.</param>
69      public void EmitAnimation(string _animation){}
70
71      /// <summary>
72      ///  Update the network player animation to local player.
73      /// </summary>
74      /// <param name="_msg">Message.</param>
75      void OnUpdateAnim(string data){}
76
77      public void EmitAttack(string _id){}
78      /// <summary>
79      /// Update the network player rotation to local player.
80      /// </summary>
81      /// <param name="_msg">Message.</param>
82      void OnUpdateAtack(string data){}
83
84      //sends to server player damage
85      public void EmitPhisicstDamage(string _shooterId, string _targetId){}
86
87      //updates player damage
88      void OnUpdatePlayerPhisicsDamage (string data){}
```

Network Manager

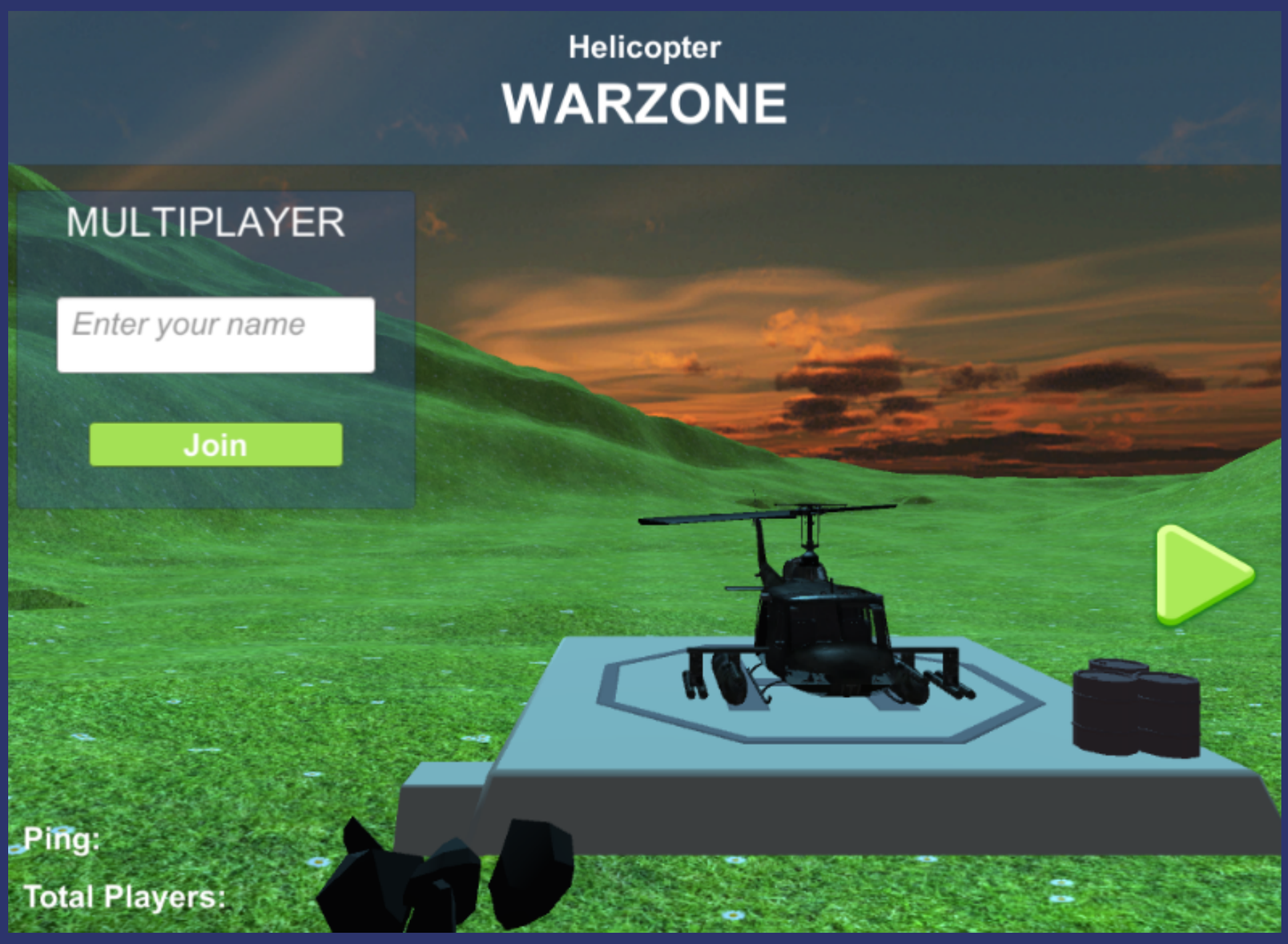
```
90      //Update player death
91      void OnPlayerDeath (string data){}
92
93      IEnumerator deathCutScene(PlayerManager PlayerTarget ){}
94
95      IEnumerator NetworkPlayerDeathCutScene(PlayerManager PlayerTarget ){}
96
97      void GameOver(){}
98
99      /// <summary>
100     /// inform the local player to destroy offline network player
101     /// </summary>
102     /// <param name="_msg">Message.</param>
103     //desconnect network player
104     void OnUserDisconnected(string data ){}
```

Server


Server.js
+ io.on('connection')() : void
+ socket.on('PING') () : void
+ socket.on('LOGIN')() : void
+ socket.on('RESPAW')() : void
+ socket.on('MOVE_AND_ROTATE')() : void
+ socket.on('ANIMATION')() : void
+ socket.on('ATTACK')() : void
+ socket.on('PHISICS_DAMAGE')() : void
+ socket.on('disconnect')() : void
+ http.listen(process.env.PORT)() : void
client.js
+ SendMessage ()() : void
+ socket.on('PONG')() : void
+ socket.on('LOGIN_SUCCESS')() : void
+ socket.on('SPAWN_PLAYER')() : void
+ socket.on('RESPAW_PLAYER')() : void
+ socket.on('UPDATE_MOVE_AND_ROTATE')() : void
+ socket.on('UPDATE_PLAYER_ANIMATOR')() : void
+ socket.on('UPDATE_ATTACK')() : void
+ socket.on('DEATH')() : void
+ socket.on('UPDATE_PHISICS_DAMAGE')() : void
+ socket.on('USER_DISCONNECTED')() : void
UnityWebGLBuild

2DShooter

3DShooter


Helicopter
WARZONE
MULTIPLAYER
Enter your name
Join
Ping:
Total Players:

Contact us

rio3dstudios@gmail.com

rio3dstudios.com