

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

ĐỖ VĂN KHẢI
NGUYỄN THỊ THANH THỦY
PHẠM NGUYỄN THIỆN MINH

ĐỒ ÁN MÔN HỌC
GAME BẮN TÀU

TP. HỒ CHÍ MINH, 2018

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

ĐỖ VĂN KHẢI – 15520331
NGUYỄN THỊ THANH THỦY - 15520865
PHẠM NGUYỄN THIỆN MINH - 15520491

ĐỒ ÁN MÔN HỌC
GAME BẮN TÀU

GIẢNG VIÊN HƯỚNG DẪN
TRẦN NGỌC ĐỨC

TP. HỒ CHÍ MINH, 2018

MỤC LỤC

Chương 1. TỔNG QUAN.....	7
1.1. Giới thiệu về game bắn tàu.....	7
1.2. Phân công công việc.....	8
1.3. Vi điều khiển NUC140	9
1.3.1. Sơ lược chung về NUC140	9
1.3.2. Đặc tính nổi bật NUC140	9
1.4. Module wifi ESP 8266 v12	10
1.4.1. Sơ lược chung về ESP8266 v12	10
1.4.2. Sơ lược về tập lệnh AT Command ESP8266 v12.....	12
Chương 2. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG	14
2.1. Phân tích mô tả và thiết kế phần cứng cho gamepad	14
2.2. Phân tích mô tả và thiết kế phần server AWS.....	20
2.2.1. Giao diện hệ thống đăng nhập và đăng kí.....	20
2.2.1.1. Giới thiệu về bootstrap.....	20
2.2.1.2. Thiết kế giao diện hệ thống đăng nhập đăng ký.....	20
2.2.2. Cơ sở dữ liệu để lưu trữ thông tin người dùng.....	22
2.2.2.1. Giới thiệu về MySQL	22
2.2.2.2. Cách tạo một database đơn giản với MySQL	23
2.2.3. Sơ đồ chức năng hệ thống đăng nhập và đăng kí.....	24
2.2.4. Giới thiệu nodejs, socket io, net	25
2.2.4.1. Nodejs	25
2.2.4.2. Socket io.....	26

2.2.4.3. Net.....	28
2.2.5. Thiết kế danh sách phòng chơi	30
2.2.6. Thiết kế phòng chờ	36
2.2.7. Nhận thông tin từ tay cầm và xử lý	46
Chương 3. KẾT QUẢ THỰC NGHIỆM	59
3.1. Tính năng đăng nhập vào hệ thống.....	59
3.1.1. Các trường hợp thử kiểm tra.	59
3.1.2. Video kiểm tra thực tế các trường hợp trên và link github	60

DANH MỤC HÌNH VẼ

Hình 1.1: Ví dụ đặt tàu hợp lệ.....	7
Hình 1.2: Sơ đồ chân NUC140	9
Hình 1.3: Module wifi ESP 8266 v12.....	10
Hình 1.4: Sơ đồ chân module wifi ESP 8266 v12	11
Hình 2.1: Sơ đồ khối chức năng của gamepad.....	14
Hình 2.2: Mạch nguyên lý của gamepad.....	15
Hình 2.3: Hình ảnh gamepad thực tế	16
Phân tích thiết kế phần mềm cho gamepad. Hình 2.3: Sơ đồ thuật toán tay cầm.....	17
Hình 2.3: Sơ đồ giải thuật hàm ngắt UART.....	18
Hình 2.3: Sơ đồ giải thuật hàm ngắt GPIO	19
Hình 2.3: Giao diện màn hình chính	20
Hình 2.3: Giao diện màn đăng kí	21
Hình 2.3: Giao diện màn hình đăng nhập	21
Hình 2.3: Giao diện màn tạo phòng chơi game	22
Hình 2.3: Sơ đồ chức năng hệ thống đăng nhập đăng kí	24

DANH MỤC BẢNG

Bảng 1.1: Một số lệnh AT Command cho ESP8266	12
--	----

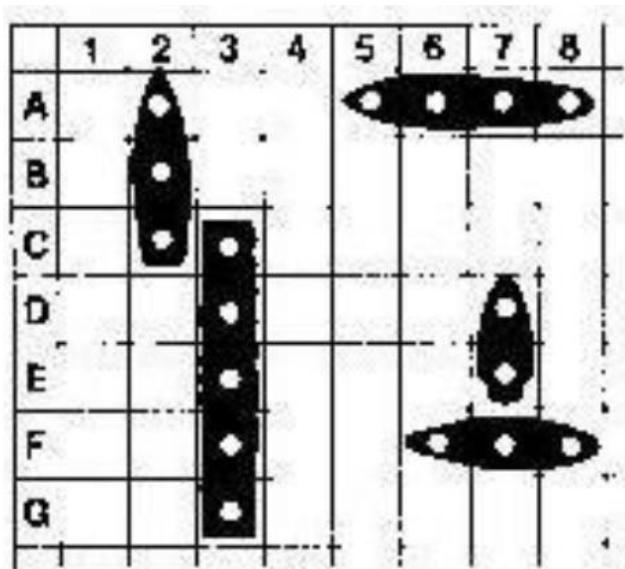
Chương 1. TỔNG QUAN

1.1. Giới thiệu về game bắn tàu

Hai người chơi truy cập vào trang web trò chơi tiếp hành đăng nhập tài khoản đã đăng kí để vào trò chơi. Sau đó tiến hành đặt 2 bản đồ sao cho không ai có thể nhìn thấy bản đồ của đối phương. Bí mật đặt 5 tàu chiến của bạn trong vùng biển của chính mình. Đối phương làm tương tự.

Luật đặt tàu chiến:

- Đặt tàu theo vị trí ngang hoặc dọc, không được chéo.
- Không được đặt các tàu chồng lên nhau để mà nó bị trùng số, chữ.
- Không được thay đổi vị trí của tàu một khi trò chơi bắt đầu.



Hình 1.1: Ví dụ đặt tàu hợp lệ

Cách chơi

Dùng tay cầm được thiết kế sẵn kết nối vào hệ thống để chọn phòng rồi chọn tay cầm để vào xếp tàu tiến hành chơi bắn tàu. Ai kết nối trước sẽ được bắn trước.

Bắn trúng!

Nếu bạn bắn trúng tàu đối phương thì thiết bị tay cầm bắn trúng rung mạnh, tay cầm người bị bắn trúng rung nhẹ hơn. Và tàu bị bắn trúng xuất hiện một viên đạn màu đỏ tại vị trí bắn trúng. Và tiếp tục lượt bắn đến khi bắn trật tàu của đối phương.

Bắn trật!

Tương tự bạn bắn trúng nhưng tay cầm sẽ không rung. Và có viên đạn màu đen tại vị trí bắn trật. Bắn trật sẽ mất lượt

Ngoài ra trò chơi còn chế độ sau 30s người tới lượt không bắn thì mất lượt.

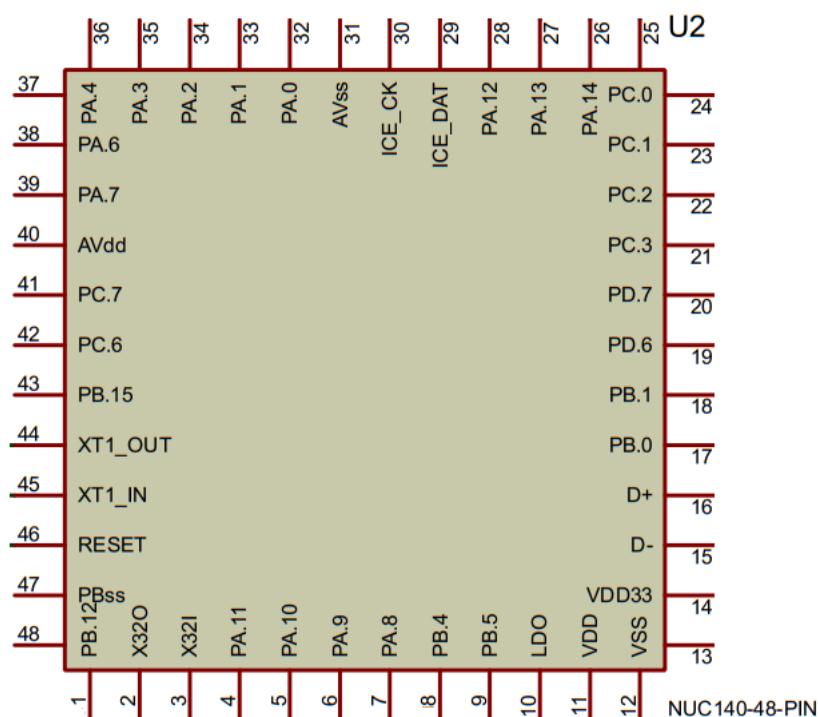
Chiến thắng!

Bạn chiến thắng khi đã đánh chìm tất cả 5 chiến hạm của đối phương.

1.2. Phân công công việc.

1.3. Vi điều khiển NUC140

1.3.1. Sơ lược chung về NUC140



Hình 1.2: Sơ đồ chân NUC140

NUC140 là sản phẩm của hãng Nuvoton là công ty con được tách ra từ Tập đoàn Điện tử Winbond – một hãng điện tử bán dẫn đứng hàng đầu Đài Loan. Hãng này có 3 dòng chip vi điều khiển (MCU) 4-bit, 8-bit và 32-bit (ARM Cortex). Dòng vi điều khiển ARM Cortex-M được thiết kế nhúng tối ưu hóa cho các ứng dụng vi xử lý MCU. Dòng ARM Cortex-M0 là dòng vi điều khiển lõi ARM có kích thước nhỏ nhất, tiêu thụ điện năng thấp nhất và có kiến trúc được sắp xếp hợp lý tương thích với việc sử dụng tools nạp của các hãng khác để phát triển các ứng dụng

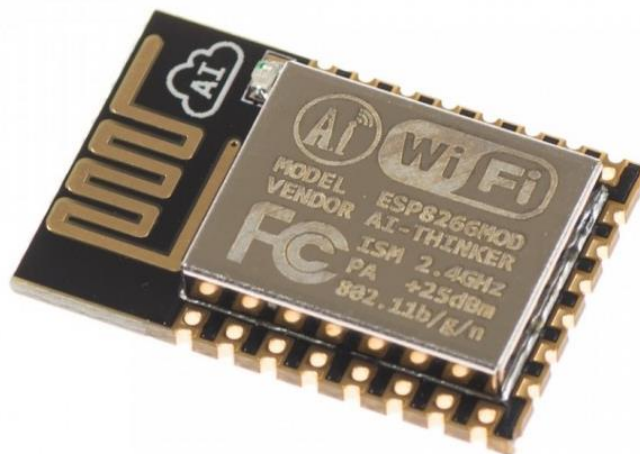
1.3.2. Đặc tính nổi bật NUC140

- NUC140 là vi điều khiển 32-bit lõi ARM Cortex-M0, trình đơn phần cứng 32 bit, chạy lên tới 50MHz.
- Có 4 mức ưu tiên ngắt đầu vào, có 128 KB flash ROM cho bộ nhớ chương trình. 16KB SRAM, 4KB bộ nhớ flash cho nạp chương trình trong hệ thống.

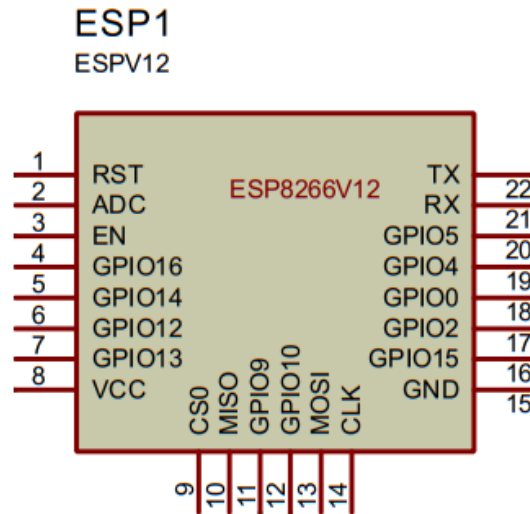
- Giao tiếp thiết bị ngoại vi: 8 kênh 12bit ADC, UART nối tiếp tốc độ cao, SPI lên đến 32MHz, I2C lên đến 1MHz; kết nối thiết bị ngoại vi USB 2.0, CAN, LIN...
- Thiết bị ngoại vi có tính năng phong phú: PWM, RTC, bộ ngắt nhận dạng Brownout, GPIO, PDMA và 4 bộ Timer 32 bit.
- Dải điện áp hoạt động rộng từ 2,5V~5,5V, chống nhiễu tiếng ồn tốt, tích hợp dữ liệu flash, dao động thạch anh nội chính xác $\pm 1\%$ với nhiệt độ phòng, có khả năng bảo mật trên chip, điện áp reset lại mạch thấp.

1.4. Module wifi ESP 8266 v12

1.4.1. Sơ lược chung về ESP8266 v12



Hình 1.3: Module wifi ESP 8266 v12



Hình 1.4: Sơ đồ chân module wifi ESP 8266 v12

- Tiêu chuẩn wifi : 802.11b/g/n, với tần số 2.4GHz, và hỗ trợ bảo mật WPA/WPA2
- Mạch nhỏ, gọn (24.75mm x 14.5mm)
- Tích hợp 10-bit ADC
- Tích hợp giao thức TCP/IP (hiện tại thời điểm này hỗ trợ ipv4)
- Tích hợp năng lượng thấp 32-bit MCU
- SDIO 2.0, SPI, UART, I2C
- STBC, 1x1 MIMO, 2x1 MIMO
- Điện áp làm việc 3.3v
- Có các chế độ: AP, STA, AT + STA
- Bộ nhớ Flash: 4MB
- Lệnh AT rất đơn giản, dễ dàng sử dụng
- Lập trình trên các ngôn ngữ: C/C++, Micropython, NodeMCU – Lua

1.4.2. Sơ lược về tập lệnh AT Command ESP8266 v12

ESP8266 sử dụng tập lệnh AT Command của riêng nó để có thể giao tiếp và lập trình.

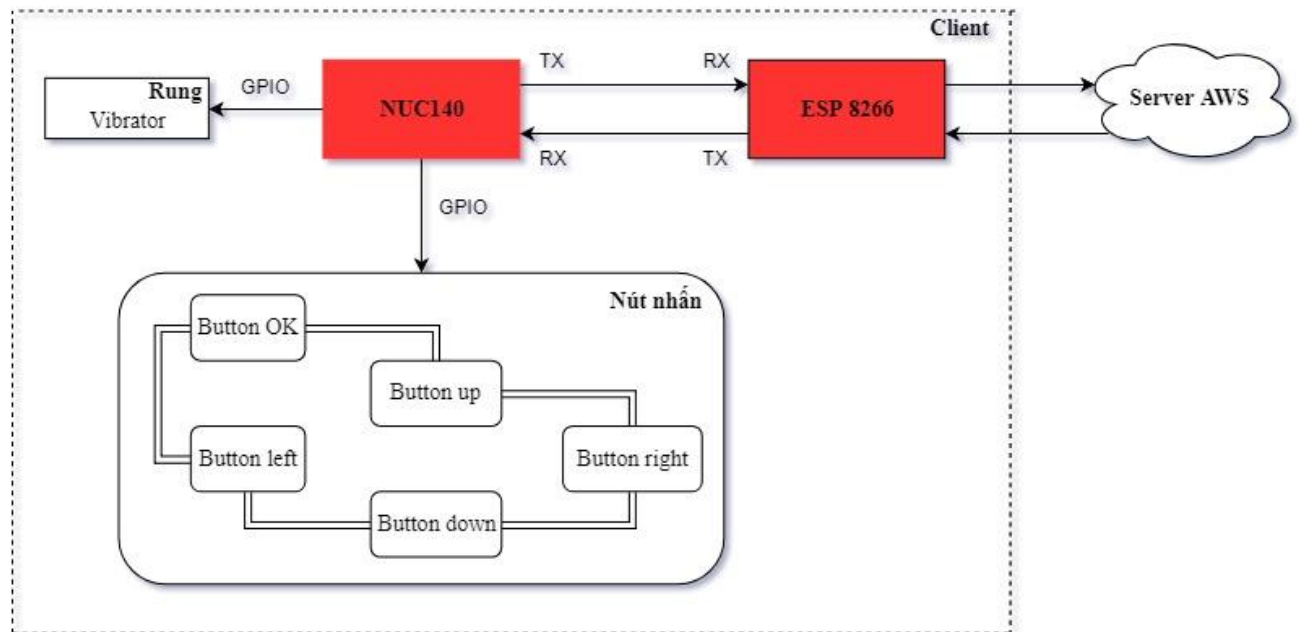
Lệnh AT Command	Mô tả chức năng	Ví dụ
AT+CWMODE = <mode>	Cài đặt chế độ 1 = Station 2 = Access Point 3 = Both	AT+CWMODE=1
AT+CIPMUX = <mode>	Cài đặt số lượng các kênh kết nối 0 = 1 kênh kết nối 1 = Nhiều kênh kết nối	AT+CIPMUX=1
AT+CWLAP	Truy vấn các mạng wifi có thể kết nối	AT+CWLAP
AT+CIPAPMAC?	Nhận địa chỉ mac của ESP8266 softAP.	AT+CIPAPMAC?
AT+CWJAP= <ssid>, <password>	Kết nối một mạng wifi với SSID và password	AT+CWJAP = "abc", "5678"
AT+CIPSTART= type, addr, port	Kết nối vào 1 TCP/UDP server của 1 server trên internet	AT+CIPSTART= "TCP", "34.205.32.160", 3333
AT+CIPSEND=length	Dùng để gửi dữ liệu có độ dài length	AT+CIPSEND=1

Bảng 1.1: Một số lệnh AT Command cho ESP8266

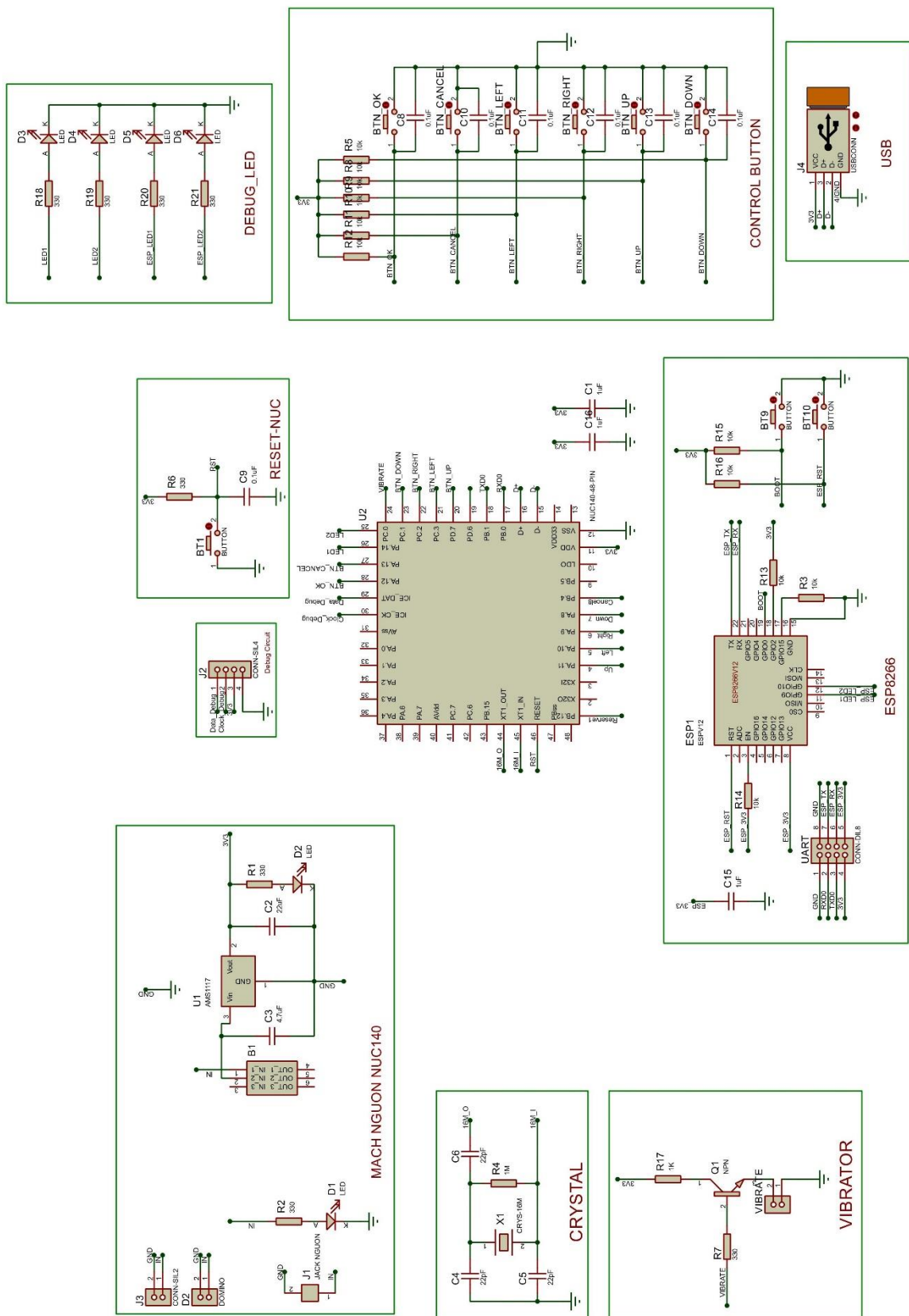
Chương 2. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

2.1. Phân tích mô tả và thiết kế phần cứng cho gamepad

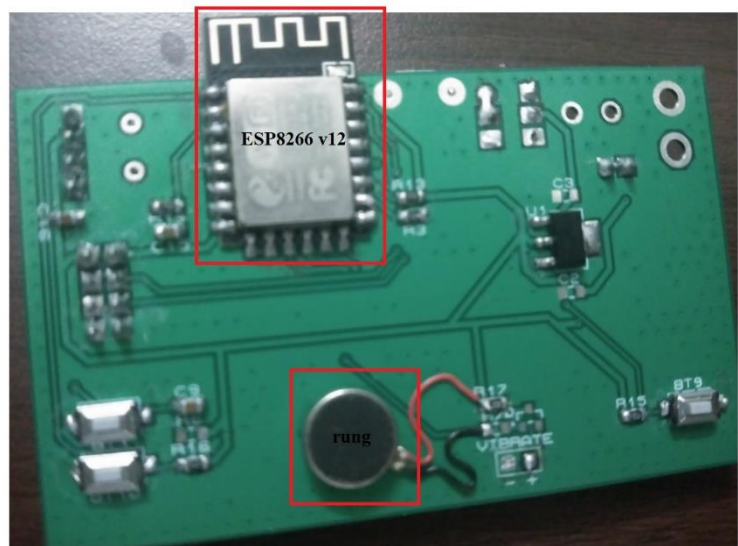
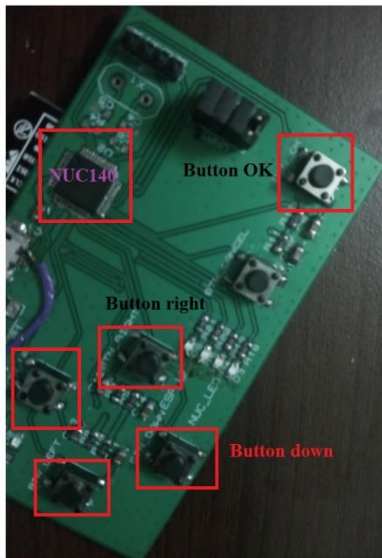
Tay cầm gọi chung là thiết bị phía client gồm có 4 nút nhấn để di chuyển lên (button up), xuống (button down), trái (button left) và phải (button right) như hình mô tả bên dưới. Cùng với một nút nhấn khác để thực hiện thao tác bắn tàu (button ok) của đối phương. Mạch bao gồm NUC140 kết nối với ESP8266 v12 thông qua UART. NUC140 là xử lý trung tâm điều khiển các thao tác di chuyển, nhận tín hiệu từ server truyền về mỗi khi bắn trúng để rung gamepad đồng thời NUC140 gửi tín hiệu để ESP8266 kết nối đến server và nhận tín hiệu từ server truyền về thông qua ESP8266.



Hình 2.1: Sơ đồ khối chức năng của gamepad

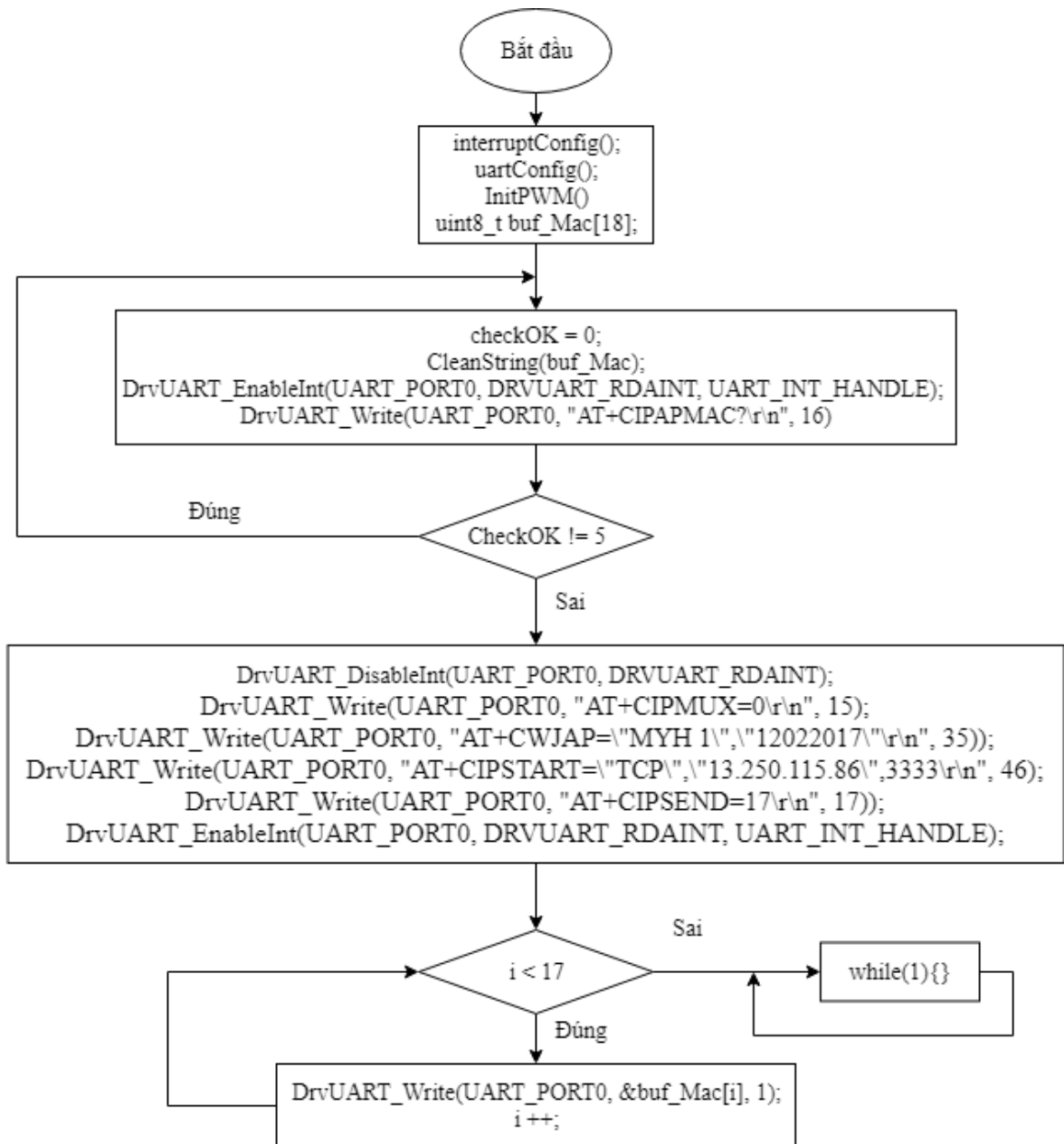


Hình 2.2: Mạch nguyên lý của gamepad

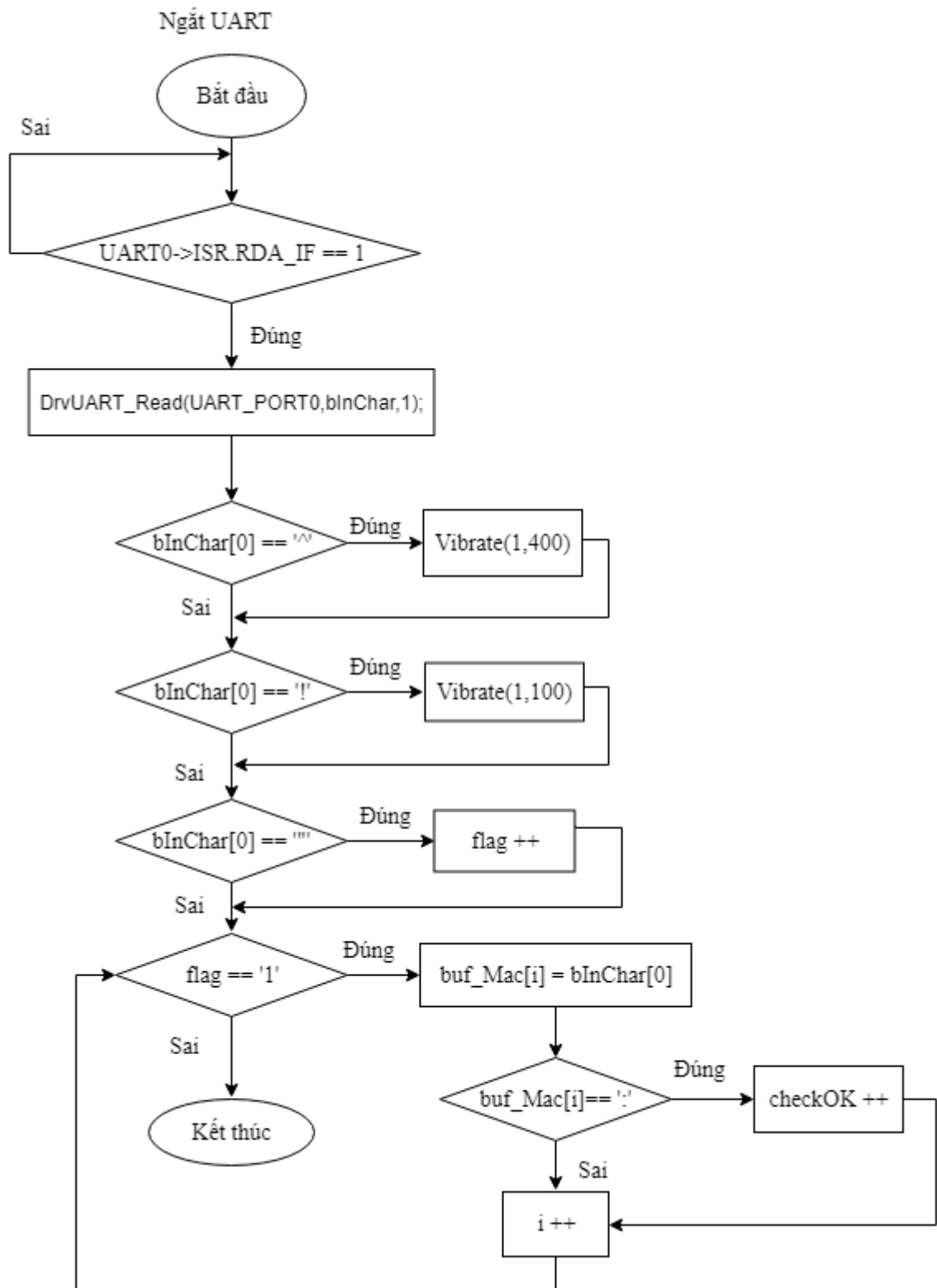


Hình 2.3: Hình ảnh gamead thực tế

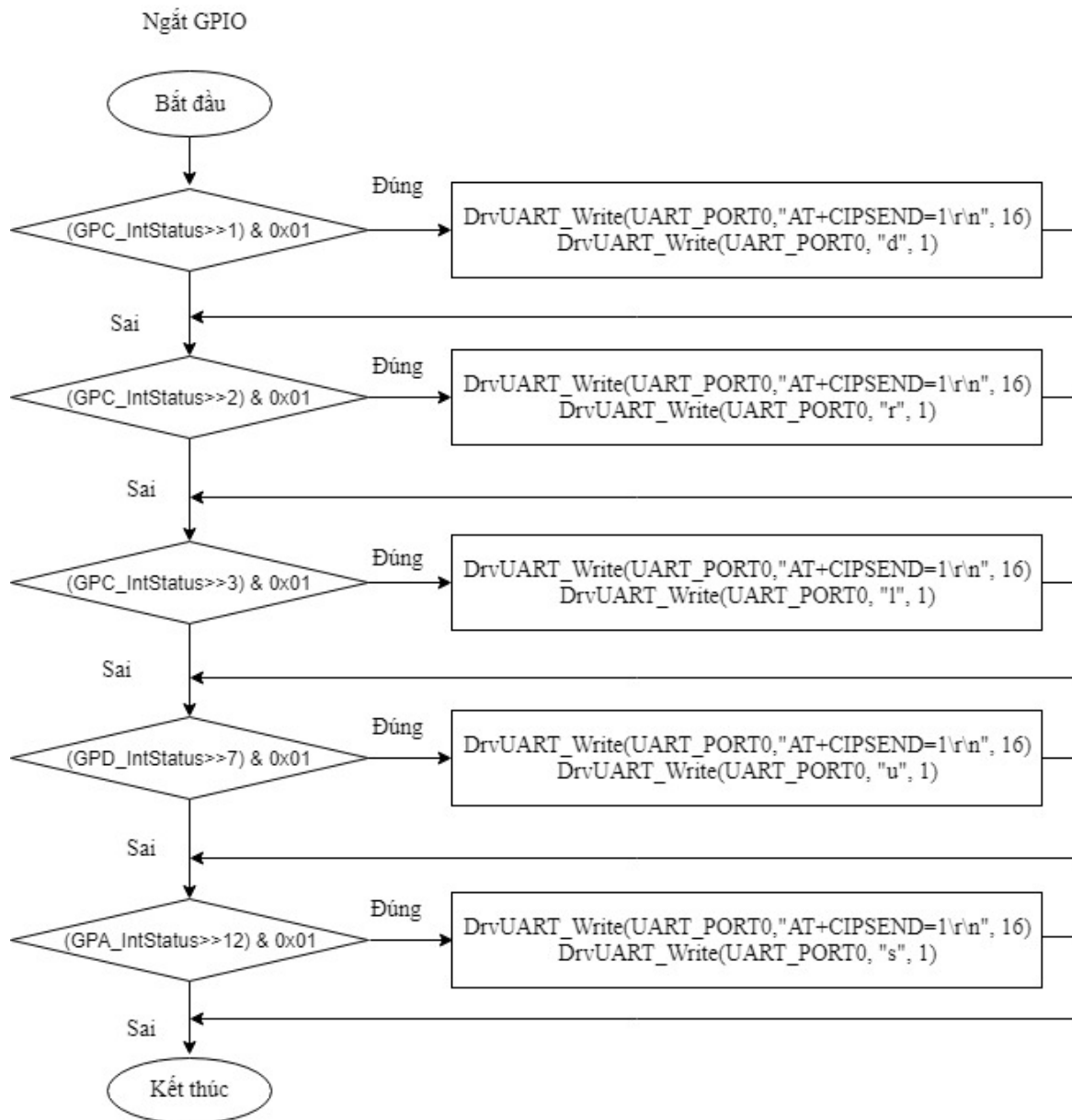
Phân tích thiết kế phần mềm cho gamepad.



Hình 2.4: Sơ đồ thuật toán tay cầm



Hình 2.5: Sơ đồ giải thuật hàm ngắt UART



Hình 2.6: Sơ đồ giải thuật hàm ngắt GPIO

2.2. Phân tích mô tả và thiết kế phần server AWS.

2.2.1. Giao diện hệ thống đăng nhập và đăng kí.

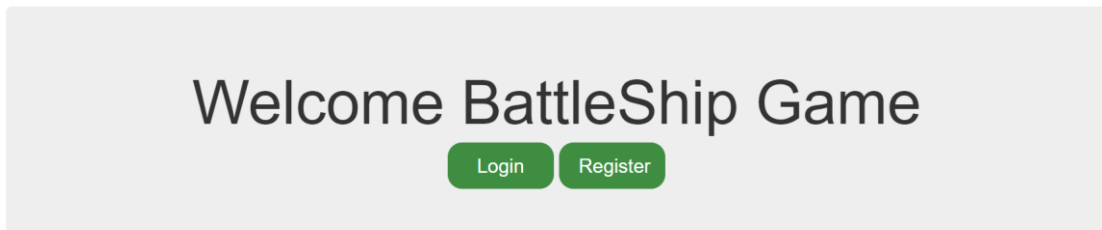
2.2.1.1. Giới thiệu về bootstrap

Bootstrap là một nền tảng (framework) miễn phí, mã nguồn mở, dựa trên HTML, CSS & Javascript, nó được tạo ra để xây dựng các giao diện Website tương thích với tất cả các thiết bị có kích thước màn hình khác nhau.

Bootstrap bao gồm những cái cơ bản có sẵn như: typography, forms, buttons, tables, navigation, modals, image carousels và nhiều thứ khác. Nó cũng có nhiều Component, Javascript hỗ trợ cho việc thiết kế Responsive của bạn dễ dàng, thuận tiện và nhanh chóng hơn.

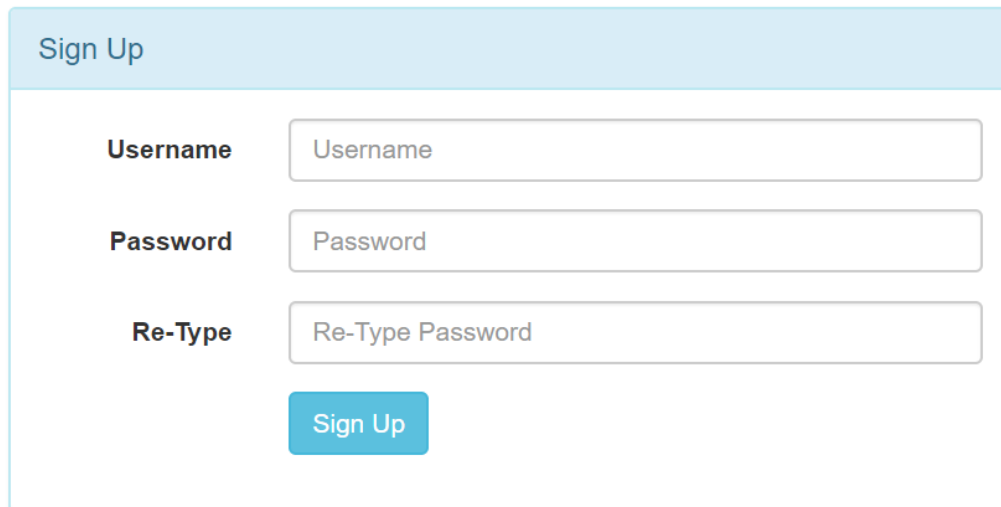
2.2.1.2. Thiết kế giao diện hệ thống đăng nhập đăng ký

- Màn hình chính gồm có hai nút nhấn để lựa chọn đăng nhập và đăng ký



Hình 2.7: Giao diện màn hình chính

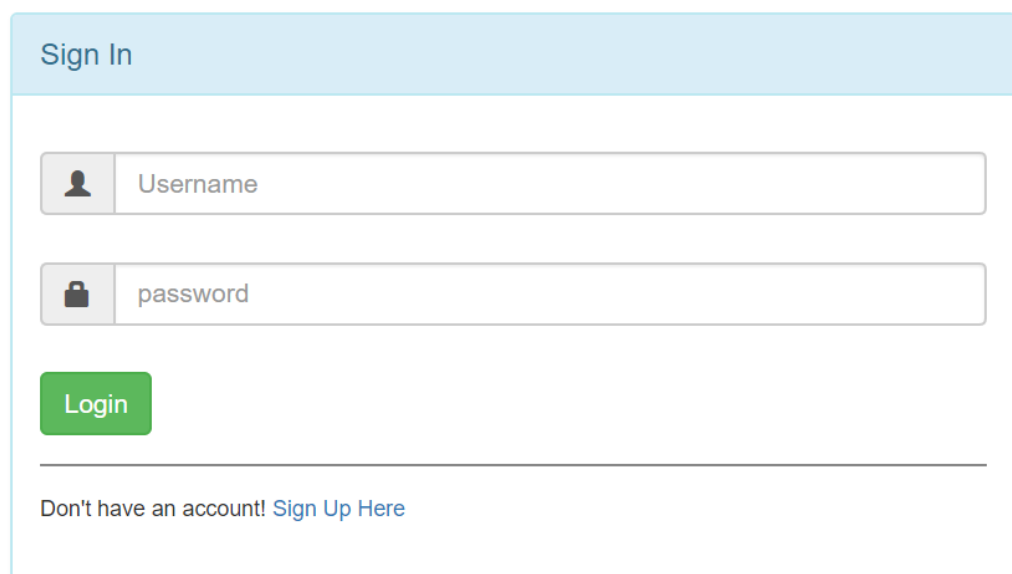
- Màn hình đăng kí gồm 3 trường yêu cầu nhập thông tin là username, password và repassword.



The image shows a 'Sign Up' form with a light blue header. Below the header, there are three input fields: 'Username', 'Password', and 'Re-Type', each with a corresponding label to its left. The 'Username' field contains the text 'Username', the 'Password' field contains 'Password', and the 'Re-Type' field contains 'Re-Type Password'. Below these fields is a blue 'Sign Up' button.

Hình 2.8: Giao diện màn đăng kí

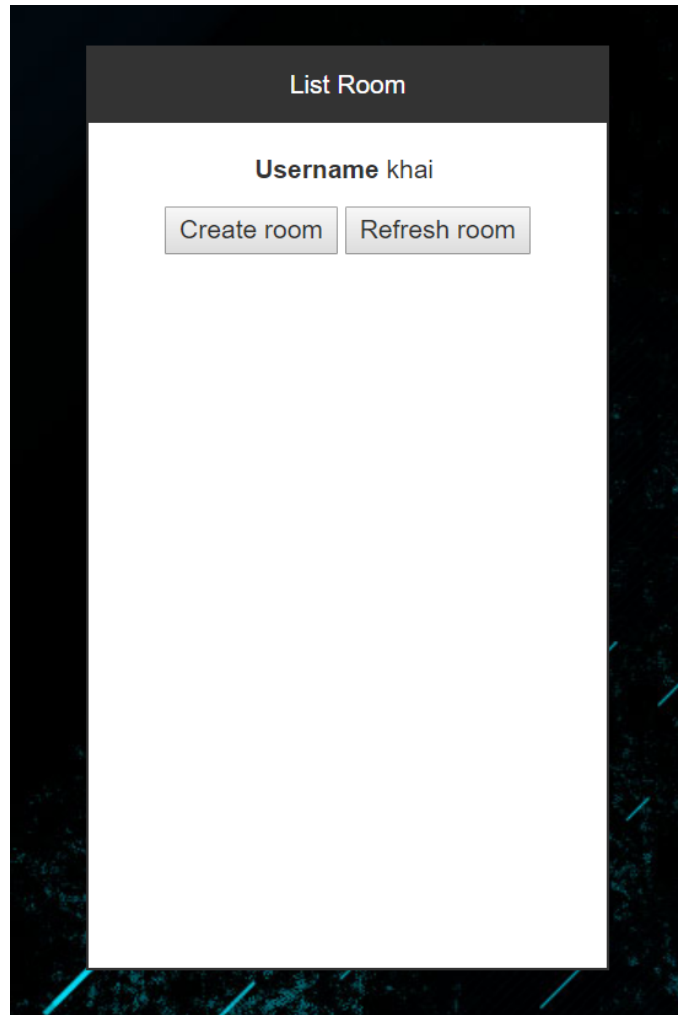
- Màn hình đăng nhập gồm 2 trường yêu cầu nhập thông tin là username, password.



The image shows a 'Sign In' form with a light blue header. Below the header, there are two input fields: 'Username' and 'password'. The 'Username' field has a user icon on the left, and the 'password' field has a lock icon on the left. Below these fields is a green 'Login' button. At the bottom of the form, there is a horizontal line and the text 'Don't have an account! [Sign Up Here](#)'.

Hình 2.9: Giao diện màn hình đăng nhập

- Sau khi đăng nhập thì sẽ chuyển tới giao diện tạo phòng bắt đầu chơi game



Hình 2.10: Giao diện màn tạo phòng chơi game

2.2.2. Cơ sở dữ liệu để lưu trữ thông tin người dùng.

2.2.2.1. Giới thiệu về MySQL

Được phát hành từ giữa thập niên 90s (sau đó bị thôn tóm bởi Oracle), MySQL ban đầu là một database mã nguồn mở và cũng vẫn mở cho tới tận bây giờ. Vì là mã nguồn mở, MySQL có rất nhiều phiên bản khác dựa trên nó. Sự khác biệt giữa các biến thể này là không lớn; cấu trúc và chức năng cơ bản tương đương nhau.

Một điều đã trở thành đặc tính riêng của MySQL là nó cực kỳ phổ biến trong cộng đồng startup. Vì nó là mã nguồn mở và miễn phí, lập trình viên có thể dễ dàng bắt đầu với MySQL, và chỉnh sửa code nếu họ cần làm vậy. MySQL thường được dùng

đồng thời với PHP và Apache Web Server, trên một bản Linux distribution, bộ tứ này đã trở thành một tên gọi nổi tiếng và quyền lực: LAMP (Linux, Apache, MySQL, PHP).

2.2.2.2. Cách tạo một database đơn giản với MySQL

- Tạo một database với tên là nodejs_game_v1

```
1. var mysql = require('mysql');
2.
3. var con = mysql.createConnection({
4.   host: "localhost",
5.   user: "root",
6.   password: ""
7. });
8.
9. con.connect(function(err) {
10.  if (err) throw err;
11.  console.log("Connected!");
12.  con.query("CREATE DATABASE Nodejs_Game_v1", function (err, result) {
13.    if (err) throw err;
14.    console.log("Database created");
15.  });
16. });
```

- Tạo một bản có tên user trong database nodejs_game_v1

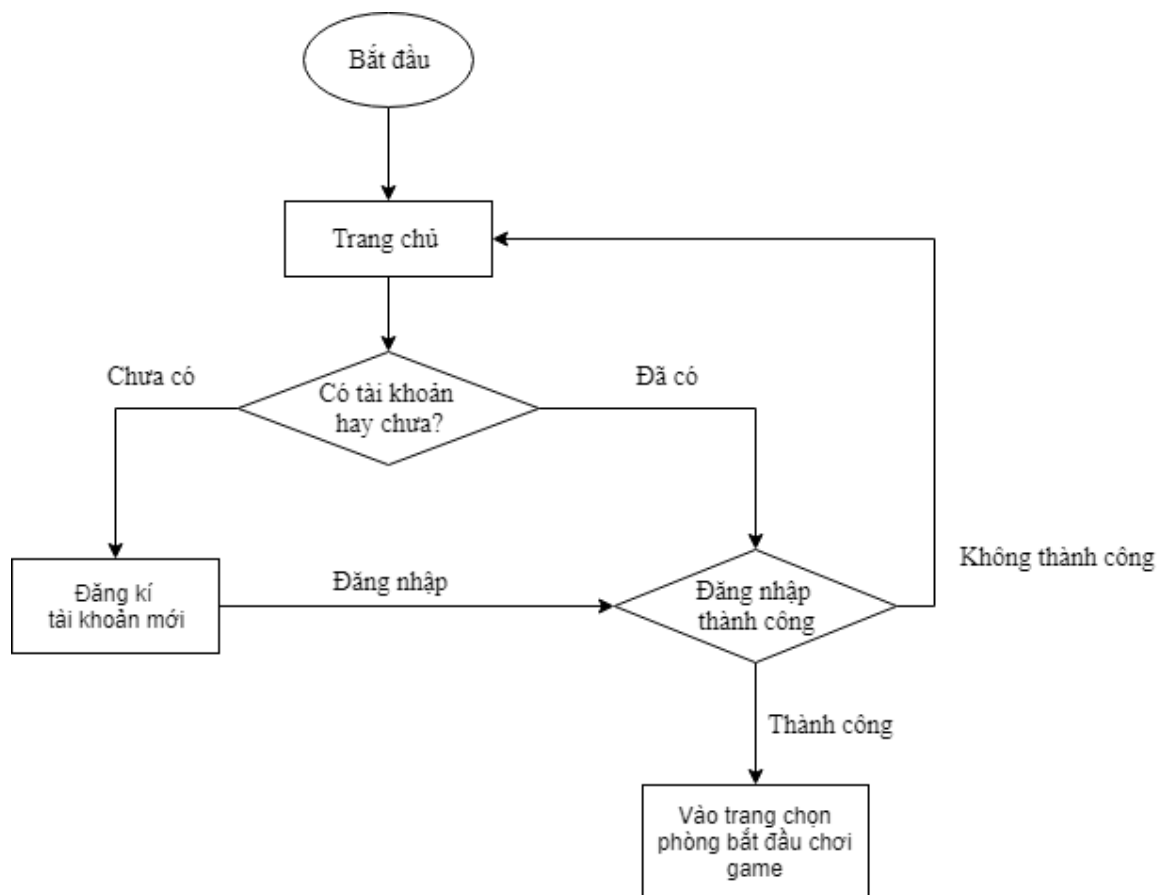
```
1. var mysql = require('mysql');
2.
3. var con = mysql.createConnection({
4.   host: "localhost",
5.   user: "root",
6.   password: "",
7.   database: "Nodejs_Game_v1"
8. });
9.
10. con.connect(function(err) {
11.  if (err) throw err;
12.  console.log("Connected!");
13.  var sql = "CREATE TABLE users (id INT primary key auto_increment, username VARCHAR(255), password VARCHAR(255), scores INT)";
```

```

14. con.query(sql, function (err, result) {
15.     if (err) throw err;
16.     console.log("table created");
17. });
18. });

```

2.2.3. Sơ đồ chức năng hệ thống đăng nhập và đăng kí



Hình 2.11: Sơ đồ chức năng hệ thống đăng nhập đăng kí

2.2.4. Giới thiệu nodejs, socket io, net

2.2.4.1. Nodejs

NodeJS là một mã nguồn mở được xây dựng dựa trên nền tảng Javascript V8 Engine, nó được sử dụng để xây dựng các ứng dụng web như các trang video clip, các forum và đặc biệt là trang mạng xã hội phạm vi hẹp.

NodeJS có thể chạy trên nhiều nền tảng hệ điều hành khác nhau từ WIndow cho tới Linux, OS X .NodeJS cung cấp các thư viện phong phú ở dạng Javascript Module khác nhau giúp đơn giản hóa việc lập trình và giảm thời gian ở mức thấp nhất.

Khi nói đến NodeJS thì phải nghĩ tới vấn đề Realtime. Realtime ở đây chính là xử lý giao tiếp từ client tới máy chủ theo thời gian thực.

+Các đặc tính của nodejs

- **Không đồng bộ:** Tất cả các API của NodeJS đều không đồng bộ (*non-blocking*), nó chủ yếu dựa trên nền của NodeJS Server và chờ đợi Server trả dữ liệu về. Việc di chuyển máy chủ đến các API tiếp theo sau khi gọi và cơ chế thông báo các sự kiện của Node.js giúp máy chủ để có được một phản ứng từ các cuộc gọi API trước (Realtime).
- **Chạy rất nhanh:** NodeJ được xây dựng dựa vào nền tảng V8 Javascript Engine nên việc thực thi chương trình rất nhanh.
- **Đơn luồng nhưng khả năng mở rộng cao:** Node.js sử dụng một mô hình luồng duy nhất với sự kiện lặp. cơ chế tổ chức sự kiện giúp các máy chủ để đáp ứng một cách không ngăn chặn và làm cho máy chủ cao khả năng mở rộng như trái ngược với các máy chủ truyền thống mà tạo ra hạn chế để xử lý yêu cầu. Node.js sử dụng một chương trình đơn luồng và các chương trình tương tự có thể cung cấp dịch vụ cho một số lượng lớn hơn nhiều so với yêu cầu máy chủ truyền thống như Apache HTTP Server.
- **Không đệm:** NodeJS không đệm bất kỳ một dữ liệu nào và các ứng dụng này chủ yếu là đầu ra dữ liệu.

- **Có giấy phép:** NodeJS đã được cấp giấy phép bởi MIT License

2.2.4.2. Socket io

Là một bộ thư viện dành cho các ứng dụng web, mobile realtime. Với đặc trưng mạnh mẽ và dễ sử dụng, [Socket.IO](#) đang dần trở nên quen thuộc với các nhà phát triển (Từ Microsoft Office, Yammer, Zendesk, Trello... tới những đội hackathon, những start up trẻ).

Thư viện này gồm 2 phần:

- **Phía client:** gồm bộ thư viện viết cho web(JavaScript), iOS, Android
- **Phía server:** viết bằng JavaScript và dùng cho các máy chủ node.JS

[Socket.IO](#) cung cấp cho các nhà phát triển một cách đơn giản và thuận tiện để xây dựng một ứng dụng realtime đa nền tảng (web và mobile). Với bộ thư viện này, làm việc với socket trở nên đơn giản hơn rất nhiều.

Thư viện [Socket.IO](#) trên Android cung cấp những **hàm cơ bản** sau:

- **connect():** kết nối với server socket
- **on(event_name, listener):** đăng kí lắng nghe sự kiện từ server trả về
- **emit(event_name, data):** gửi một sự kiện lên server
- **off(event_name):** ngừng lắng nghe một sự kiện nào đó

Các lệnh emit thường sử dụng:

```

io.on('connect', onConnect);

function onConnect(socket){

    // sending to the client
    socket.emit('hello', 'can you hear me?', 1, 2, 'abc');

    // sending to all clients except sender
    socket.broadcast.emit('broadcast', 'hello friends!');

    // sending to all clients in 'game' room except sender
    socket.to('game').emit('nice game', "let's play a game");

    // sending to all clients in 'game1' and/or in 'game2' room, except sender
    socket.to('game1').to('game2').emit('nice game', "let's play a game (too)");

    // sending to all clients in 'game' room, including sender
    io.in('game').emit('big-announcement', 'the game will start soon');

    // sending to all clients in namespace 'myNamespace', including sender
    io.of('myNamespace').emit('bigger-announcement', 'the tournament will start soon');

    // sending to a specific room in a specific namespace, including sender
    io.of('myNamespace').to('room').emit('event', 'message');

```

```

// sending to individual socketid (private message)
io.to(`${socketId}`).emit('hey', 'I just met you');

// WARNING: `socket.to(socket.id).emit()` will NOT work, as it will send to everyone in the room
// named `socket.id` but the sender. Please use the classic `socket.emit()` instead.

// sending with acknowledgement
socket.emit('question', 'do you think so?', function (answer) {});

// sending without compression
socket.compress(false).emit('uncompressed', "that's rough");

// sending a message that might be dropped if the client is not ready to receive messages
socket.volatile.emit('maybe', 'do you really need it?');

// specifying whether the data to send has binary data
socket.binary(false).emit('what', 'I have no binaries!');

// sending to all clients on this node (when using multiple nodes)
io.local.emit('hi', 'my lovely babies');

// sending to all connected clients
io.emit('an event sent to all connected clients');

};

```

2.2.4.3. Net

net Module trong Node.js được sử dụng để tạo Server và Client. Module này cung cấp một Network Wrapper không đồng bộ và có thể được import với cú pháp:

```
var net = require("net")
```

Một số phương thức của net:

Phương thức của net Module trong Node.js

Stt	Phương thức & Miêu tả
1	net.createServer([options][, connectionListener]) Tạo một TCP Server mới. Tham số connectionListener tự động được thiết lập để thành một Listener cho sự kiện 'connection'.
2	net.connect(options[, connectionListener]) Đây là một phương thức factory, trả về một 'net.Socket' mới và kết nối tới address và port đã cho.
3	net.createConnection(options[, connectionListener]) Đây là một phương thức factory, trả về một 'net.Socket' mới và kết nối tới address và port đã cho.
4	net.connect(port[, host][, connectionListener]) Tạo một kết nối TCP tới port trên host đã cho. Nếu host không được cung cấp, thì giá trị mặc định là 'localhost'. Tham số connectListener sẽ được thêm vào như là Listener cho sự kiện 'connect'.
5	net.createConnection(port[, host][, connectionListener]) Tạo một kết nối TCP tới port trên host đã cho. Nếu host không được cung cấp, thì giá trị mặc định là 'localhost'. Tham số connectListener sẽ được thêm vào như là Listener cho sự kiện 'connect'.
6	net.connect(path[, connectionListener]) Tạo một kết nối Unix Socket tới đường dẫn đã cho. Tham số connectListener sẽ được thêm vào như là Listener cho sự kiện 'connect'.
7	net.createConnection(path[, connectionListener]) Tạo một kết nối Unix Socket tới đường dẫn đã cho. Tham số connectListener sẽ được thêm vào như là Listener cho sự kiện 'connect'.
8	net.isIP(input) Kiểm tra xem input có phải là một địa chỉ IP address không. Trả về giá trị 0 cho một chuỗi không hợp lệ, 4 cho phiên bản địa chỉ IP v4, và trả về 6 cho địa chỉ IP v6.
9	net.isIPv4(input) Trả về true nếu input là địa chỉ IP v4, nếu không là false.
10	net.isIPv6(input) Trả về true nếu input là địa chỉ IP v6, nếu không là false.

Lớp net.Socket trong Nodejs:

Đối tượng này là lớp trừu tượng của TCP hoặc Local Socket. net.Socket kế thừa duplex Stream interface. Chúng có thể được tạo bởi người dùng hoặc bởi một Client (bởi phương thức connect()) hoặc có thể được tạo bởi Node.js và được truyền tới người dùng thông qua sự kiện 'connection' của một Server.

Sự kiện của net.Socket trong Node.js

net.Socket là một EventEmitter và nó sinh các sự kiện sau.

Stt	Sự kiện & Miêu tả
1	lookup Xảy ra sau khi đã Resolve một hostname nhưng trước khi kết nối
2	connect Xảy ra khi một kết nối Socket được thiết lập thành công
3	data Xảy ra khi dữ liệu đã được nhận. Tham số data sẽ là một Buffer hoặc String. Phần mã hóa của data được thiết lập bởi socket.setEncoding().
4	error Xảy ra khi xuất hiện bất kỳ lỗi nào. Sự kiện 'close' sẽ được gọi trực tiếp sau sự kiện này.
5	close Xảy ra khi Socket đã được đóng.

Thuộc tính của net.Socket trong Node.js

net.Socket có nhiều thuộc tính hữu ích giúp bạn điều khiển tốt hơn trong việc tương tác với Socket.

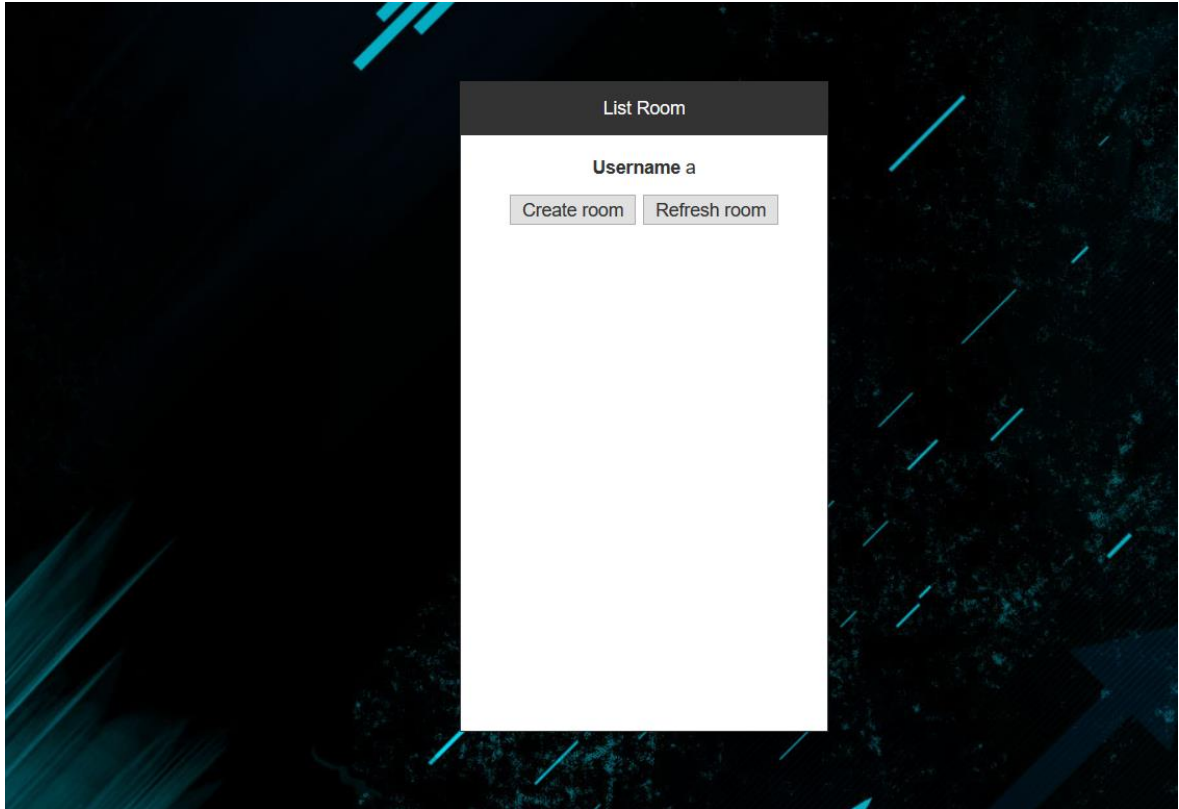
Stt	Thuộc tính & Miêu tả
1	socket.bufferSize Thuộc tính này chỉ số lượng ký tự đã được đệm
2	socket.remoteAddress Biểu diễn chuỗi của địa chỉ Remote IP.
3	socket.remoteFamily Biểu diễn chuỗi của Remote IP Family. Đó là 'IPv4' hoặc 'IPv6'.
4	socket.remotePort Biểu diễn dạng số của Remote Port. Ví dụ 80 hoặc 21.
5	socket.localAddress Biểu diễn chuỗi của địa chỉ Local IP mà một Remote Client kết nối tới. Ví dụ, nếu bạn đang lắng nghe trên '0.0.0.0' và Client kết nối trên '192.168.1.1', thì giá trị sẽ là '192.168.1.1'.
6	socket.localPort Biểu diễn dạng số của Local Port. Ví dụ 80 hoặc 21.
7	socket.bytesRead Số lượng byte đã nhận được.
8	socket.bytesWritten Số lượng byte đã được gửi.

Phương thức của net.Socket trong Node.js

Stt	Phương thức & Miêu tả
1	new net.Socket([options]) Xây dựng một đối tượng Socket mới.
2	socket.connect(port[, host], connectListener) Mở kết nối cho một Socket đã cho. Nếu bạn cung cấp hai tham số port và host, thì khi đó Socket sẽ được mở dưới dạng như là một TCP Socket. Nếu bạn không cung cấp host, thì giá trị mặc định là localhost. Nếu bạn cung cấp tham số path, thì Socket sẽ được mở dưới dạng như một Unix Socket tới đường dẫn path đó.
3	socket.connect(path[, connectListener]) Mở kết nối cho một Socket đã cho. Nếu bạn cung cấp hai tham số port và host, thì khi đó Socket sẽ được mở dưới dạng như là một TCP Socket. Nếu bạn không cung cấp host, thì giá trị mặc định là localhost. Nếu bạn cung cấp tham số path, thì Socket sẽ được mở dưới dạng như một Unix Socket tới đường dẫn path đó.
4	socket.setEncoding([encoding]) Thiết lập mã hóa encoding cho Socket dưới dạng như một Readable Stream.
5	socket.write(data[, encoding], callback) Gửi dữ liệu trên Socket. Tham số thứ hai xác định mã hóa trong trường hợp dữ liệu dạng chuỗi. Mã hóa mặc định là UTF8.
6	socket.destroy() Bảo đảm rằng không có bất kỳ hoạt động I/O xảy ra trên Socket này. Phương thức này chỉ cần thiết khi xuất hiện lỗi.
7	socket.pause() Tạm dừng việc đọc dữ liệu. Do đó sự kiện 'data' không được sinh
8	socket.resume() Tiếp tục việc đọc dữ liệu sau khi đã tạm dừng với phương thức pause().

2.2.5. Thiết kế danh sách phòng chơi

Sau khi đăng nhập thành công thì ta sẽ vào giao diện như hình:



Khi nhấn nút Create room thì chạy :

1. Từ phía **client**: (web đang đăng nhập sẽ có socket riêng biệt) sẽ dùng **socket.emit('create_room')** để thông báo về server có sự kiện tạo phòng
2. Từ phía **server**: **socket.on('create_room',function({}))** để lắng nghe sự kiện tạo phòng từ client ,hàm function sẽ được thực thi.

- Ta phải tạo trước 1 đối tượng:

```
var room = {  
  id: null,  
  players: null  
};
```

id	Mảng tên phòng
players	Mảng số lượng người trong phòng

- Hàm function sẽ thực hiện:

```
socket.on('create_room', function() {
    amount_room = amount_room + 1;
    players = 0;

    list_rooms.push(amount_room);
    players_inroom.push(players);

    room.id = list_rooms;
    room.players = players_inroom;

    socket.emit('join_r', amount_room-1);
    socket.emit('joinRoom', amount_room);
    io.sockets.emit('update_room', room);
});
```

-Ban đầu **amount_room** =0,khi tạo phòng thì tăng 1 và cũng là tên room rồi thêm vào mảng **list_rooms** .Người chơi trong phòng =0 rồi thêm vào mảng **player_inroom**. Đối tượng **room** lưu 2 mảng trên để cập nhật mới nhất

-Tiếp theo **socket.emit('join_r',amount_room-1)**

Để thông báo cho **phía client** sẽ vào phòng vừa tạo. Data của emit **join_r** là **chỉ số phòng trong mảng** trong đối tượng **room**. Do ban đầu amount_room luôn tăng 1 trước khi đưa vào mảng nên muốn có được chỉ số thì trừ -1 sẽ ra được

-**Phía client** lắng nghe sự kiện **join_r**:

```
socket.on('join_r',function(data){
    socket.emit('check_room',data);
});
```

Data là chỉ số phòng nhận được,ta sẽ thông báo lại về **phía server** sự kiện **check_room** đó

-Phía server lắng nghe sự kiện **check_room**:

```
socket.on('check_room', function(index) {  
    if (room.players[index] >= 2) {  
        socket.emit('not_joinRoom', index);  
    } else {  
        //chuyển giao diện  
        socket.emit('joinRoom', room.id[index]);  
  
        //xử lý  
        room.players[index]++; //tăng số lượng người trong phòng  
  
        socket.room = room.id[index]; //để biết socket hiện đang ở phòng nào  
  
        io.sockets.emit('update_room', room); //cập nhật phòng để hiển thị số người trong phòng  
  
        socket.join('room' + socket.room); //vào phòng và chờ đối thủ  
    }  
});
```

+Kiểm tra nếu người trong phòng lớn hơn 2 thì báo cho **client** sự kiện **not_joinRoom** không cho vào phòng

+Nếu ngược lại thì

+Báo cho **client** sự kiện **joinRoom** với **data** truyền đi là **id của phòng** đó,thực hiện việc chuyển sang **giao diện** phòng chờ đối thủ

+Tăng số lượng người trong phòng,thêm thuộc tính **socket.room** để biết **socket** hiện tại ở **room** nào

+Dùng **io.sockets.emit('update_room', room)** :thông báo cho **tất cả client** về sự kiện **update_room**,cập nhật hiển thị lại những thay đổi của danh sách **room**

+ **socket.join('room' + socket.room)**: vào phòng có tên là chuỗi **'room' + socket.room**

-Phía client:

+Lắng nghe sự kiện **not_joinRoom**,**data** nhận được là **chỉ số mảng trong room**

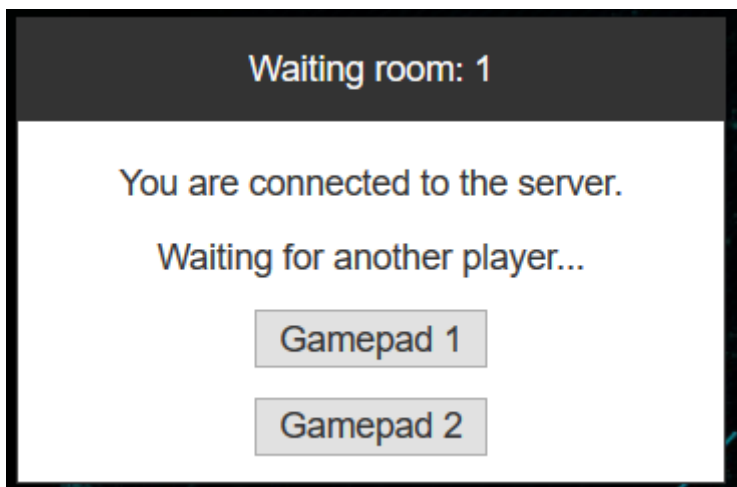
+Lắng nghe sự kiện **joinRoom**, data nhận được là id phòng

```
socket.on('joinRoom',function(data){  
    //giao diện  
    $('#rooms').hide();  
    $('#waiting-room').show();  
    document.getElementById("id_room").innerHTML = "Waiting room: " + data;  
    //xử lý  
    myFunction6();//show list gamepad  
});
```

+Giao diện thì ẩn rooms (danh sách phòng) đi, hiện waiting-room (phòng chờ đối thủ)

+Hiện thị tên phòng

+Hàm hiển thị danh sách tay cầm hiện có



+Lắng nghe sự kiện **update_room**, data nhận được là đối tượng room

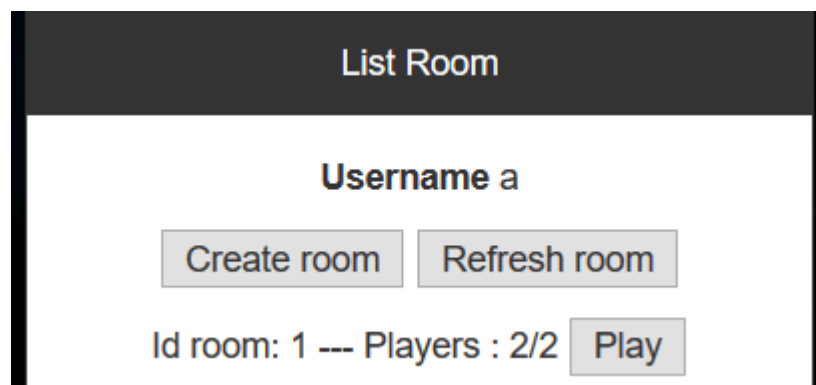
```

socket.on('update_room',function(data){
    $('#boxContent').html("");//tạo cho rỗng ruột để dễ cập nhập
    var c=0;
    for(;c<data.id.length;c++)
    {
        $('#boxContent').append("<div class='user'>" + "<p>Id room: "+data.id[c] + " --- Players : "+
            data.players[c] +"/2" + " <button type='button' id='"+c
            +"' onclick='myFunction5("+ c + ")' >Play</button>" + "</p></div>");
    }
});

```

+Có 1 div với id boxContext để hiện danh sách phòng.Mỗi lần cập nhập thì xóa hết nội dung cũ trong div đó

+Dùng for duyệt đối tượng room,mỗi lần duyệt sẽ thêm 1 **div**, gồm **tên phòng,số người** trong phòng, **nút play có id** với **data** truyền vào là chỉ số mảng khi duyệt để khi nhấn **nút play** thì nó chuyển qua giao diện vào phòng chờ đối thủ



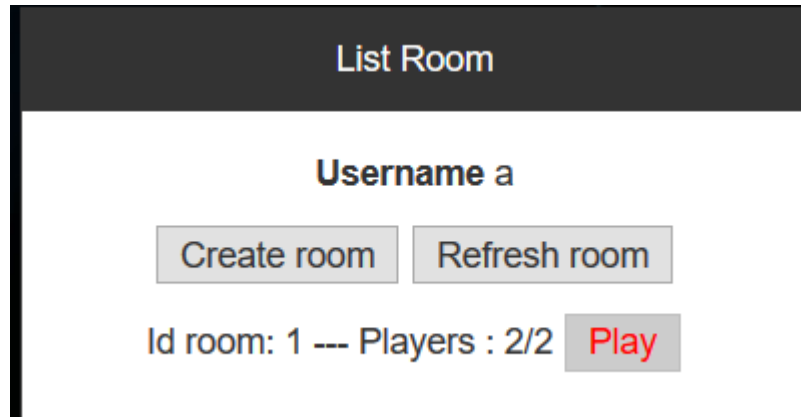
+Lắng nghe sự kiện **not_joinRoom**,data nhận được là chỉ số mảng trong **room**

```

socket.on('not_joinRoom',function(index){
    var x = document.getElementById(index);
    x.disabled=true;
    x.style.color="red";
});

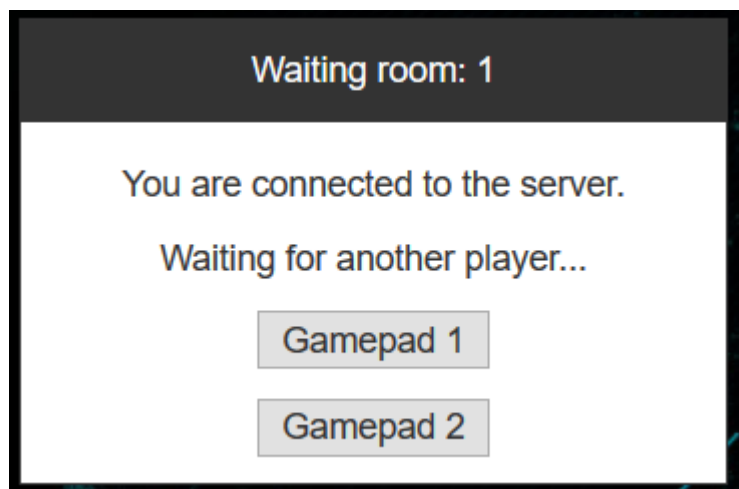
```

+do khi update room có tạo ra các div gồm nút play có id là chỉ số của mảng nên ta lấy id và disable nút đó,chuyển sang màu đỏ để cho biết phòng đã đầy



2.2.6. Thiết kế phòng chờ

Sau khi vào phòng chờ,sẽ hiển thị danh sách tay cầm để chọn



Tay cầm sử dụng tcp để kết nối,do vậy trên server thì nodejs có gói **net** hỗ trợ về tcp
Đầu tiên tạo 1 đối tượng gamepads như sau:

```
var gamepads = {
  id: 0,
  status: null,
  player: null,
  mac: null
};
```

id	Mảng chứa địa chỉ ip và port của tay cầm khi kết nối
status	Mảng Trạng thái tay cầm đã có người chọn hay chưa.0 là chưa chọn,1 là chọn rồi
player	Mảng Socket.id của người chơi
mac	Mảng Địa chỉ mac

Ta khai báo gói net :

```
// thư viện TCP
var net = require('net');
```

Rồi dùng hàm createServer trong gói net để tạo tcp server

```
// Start a TCP Server
net.createServer(function(socket) {
  // Identify this client
  socket.name = socket.remoteAddress + ":" + socket.remotePort
  console.log('CONNECTED: ' + socket.name);

  socket_tcp.push(socket);
  var x = null;
  flag_gamepad = 0;
  //socket.join('gamepad');
  clients.push(socket.name);
  check_gamepad.push(flag_gamepad);
  player_gamepad.push(x);

  gamepads.id = clients;
  gamepads.status = check_gamepad;
  gamepads.player = player_gamepad;

}).listen(3333, function() {
  console.log('TCP Server is listening on port 3333');
});
```

+Khi có tay cầm kết nối thì ta nhận được đối tượng socket

+ta thêm 1 thuộc tính name vào socket gồm địa chỉ ip và port rồi lưu nó vào mảng clients. Ta cũng khởi tạo các giá trị ban đầu của **status,player**.Cuối cùng đối tượng **gamepads** sẽ lưu các mảng vừa tạo ở trên.

+Tiếp theo từ **server tcp** sẽ thông báo cho tất cả **các client(web)** về sự kiện có tay cầm

```
io.sockets.emit('update_gamepad', gamepads);
```

+Phía **client** lắng nghe sự kiện **update_gamepad**:

```

//hiển thị tay cầm
//input:danh sách tay cầm gồm 3 mảng con.
//1 là id      : chứa địa chỉ ip và port tay cầm
//2 là status  : trạng thái tay cầm (= 0: chưa chọn, = 1: đã chọn)
//3 là player  : chứa socket id của người chơi
//xử lý: duyệt mảng, tạo button tay cầm và kiểm tra trạng thái button (khi button đã chọn thì sẽ disable button đó )
socket.on('update_gamepad',function(data){
    $('#box_gamepad').html("");
    $('#gamepad_selected').html("");
    var c=0,temp;
    for(;c<data.id.length;c++)
    {
        temp=c+1;
        $('#box_gamepad').append("<div class='user'>" + " <button type='button' id='bt"+c
        |"'" onclick='myFunction7("+ c + ")'> Gamepad "+temp+ "</button>" + "</p></div>");
        if(data.status[c]==1)
        {
            status_gamepad(c);
        }
    }
});

```

Ta duyệt mảng **gamepads** nhận được, từng phần tử sẽ tạo 1 nút nhấn có **id = chuỗi ("bt"+chỉ số phần tử)** và đồng thời kiểm tra tình trạng của nút đó.

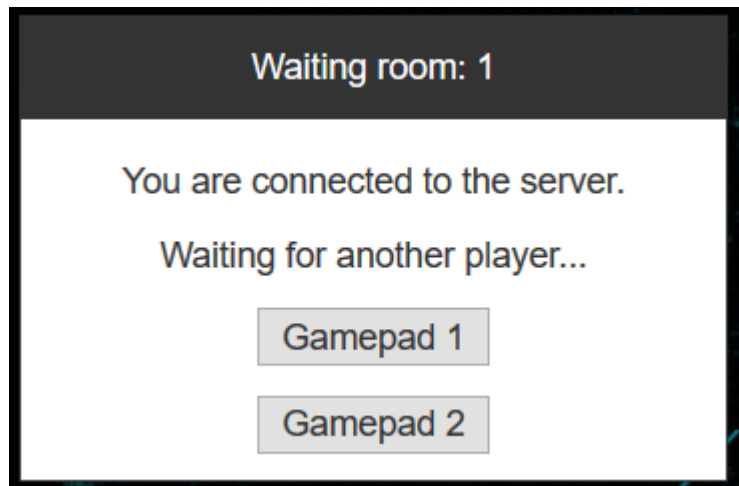
+nếu **status =1** thì tức tay cầm đã có người chọn. Ta gọi tới hàm **status_gamepad**(chỉ số phần tử) để **disable** nút đó không cho những người khác chọn và hiện màu đỏ làm dấu

```

function status_gamepad(index)
{
    var x = document.getElementById("bt"+index);
    x.disabled=true;
    x.style.color="red";
}

```

+Sau khi hiển thị, ta tới phần **chọn tay cầm**:



```
function myFunction7(index){  
    socket.emit('check_gamepad',index);  
}
```

Khi nhấn thì sự kiện nhấn nút sẽ chạy hàm myFunction7,data là chỉ số của phần tử trong mảng gamepads.Hàm này sẽ thông báo một sự kiện check_gamepad tới server để kiểm tra có được chọn hay không.

+Phía server sẽ lắng nghe sự kiện như sau:

```
socket.gamepad = null; //chứa địa chỉ mac của tay cầm
```



```

//kiểm tra tay cầm. Nếu tay cầm chưa được chọn và player chưa chọn tay cầm nào thì có thể chọn tay cầm
//input: chỉ số tay cầm trong danh sách
socket.on('check_gamepad', function(index) {

    if (gamepads.status[index] == 0 && socket.gamepad == null) {

        //giao diện
        socket.emit('select_gamepad', index);
        gamepads.status[index] = 1;
        gamepads.player[index] = socket.id;

        var c = 0;
        for (; c < player_socket.id.length; c++) {
            if (socket.id == player_socket.id[c]) {
                socket.gamepad = gamepads.mac[index];
                player_socket.mac_gamepad[c] = gamepads.mac[index];
            }
        }

        socket.broadcast.emit('update_gamepad', gamepads);
        //khi nhận tay cầm xong thì kiểm tra cả hai người trong phòng chờ đã chọn tay cầm chưa
        joinWaitingPlayers(socket.room);
    } else {
        socket.emit('not_select_gamepad');
    }
});

```

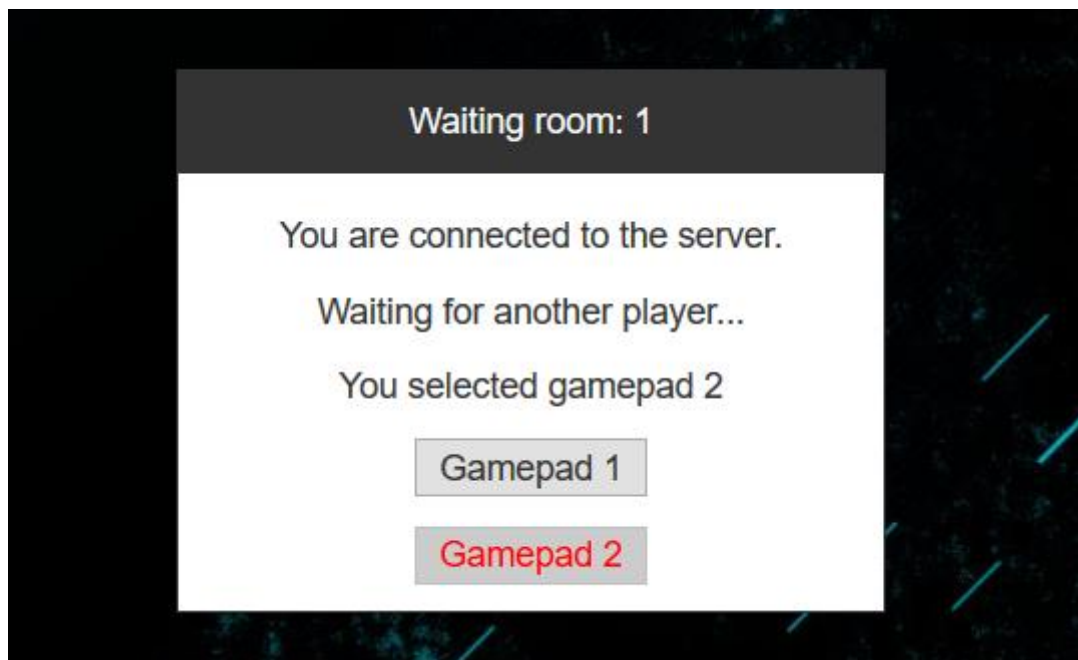
+kiểm tra nếu status của gamepad và socket.gamepad chưa chọn thì thực hiện:

+Thông báo cho client sự kiện **select_gamepad** chọn tay cầm đó, Phía client sẽ lắng nghe sự kiện và thay đổi giao diện

```

//giao diện chọn tay cầm.
//input :chỉ số vị trí tay cầm trong mảng tay cầm
//output:thông báo và đánh dấu nút đã chọn
socket.on('select_gamepad',function(index){
    status_gamepad(index);//đánh dấu nút
    var y=index+1;
    document.getElementById("gamepad_selected").innerHTML = "You selected gamepad " +y;
});

```

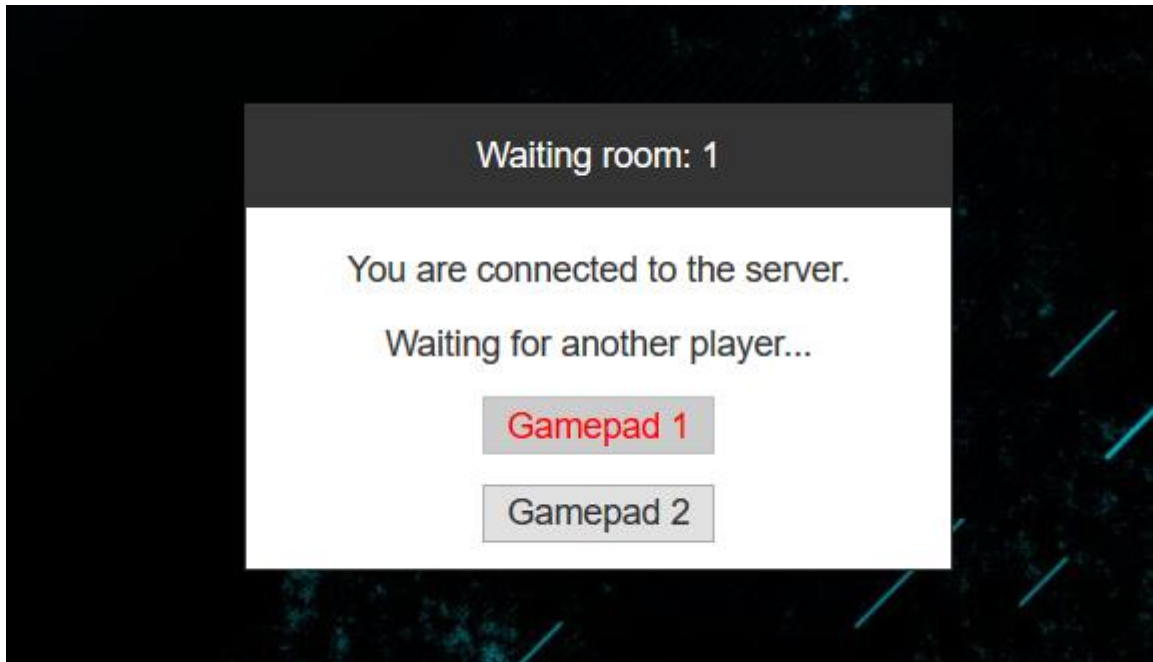


+Thiết lập trạng thái tay và lưu thông tin socket người chơi trong gamepads.

Gán địa chỉ mac của tay cầm thêm cho socket thuộc tính socket.gamepad

+Thông báo cho các client còn lại sự kiện `update_gamepad` để cho biết rằng có tay cầm đã chọn. Phía client sẽ lắng nghe sự kiện và thay đổi giao diện

```
//hiển thị tay cầm
//input:danh sách tay cầm gồm 3 mảng con.
//1 là id : chứa địa chỉ ip và port tay cầm
//2 là status : trạng thái tay cầm (= 0: chưa chọn, = 1: đã chọn)
//3 là player : chứa socket id của người chơi
//xử lý: duyệt mảng, tạo button tay cầm và kiểm tra trạng thái button (khi button đã chọn thì sẽ disable button đó)
socket.on('update_gamepad',function(data){
    $('#box_gamepad').html("");
    $('#gamepad_selected').html("");
    var c=0,temp;
    for(;c<data.id.length;c++)
    {
        temp=c+1;
        $('#box_gamepad').append("<div class='user'>" + " <button type='button' id='bt'+c+' onclick='myFunction7("+ c
        + ")> Gamepad "+temp+ "</button>" + "</p></div>");
        if(data.status[c]==1)
        {
            status_gamepad(c);
        }
    }
});
```



Khi đã có người chọn tay cầm đó thì sẽ disable nút đó, các người chơi còn lại sẽ không chọn tay cầm đó được nữa

+kiểm tra nếu status của gamepad và socket.gamepad đã chọn thì báo cho client đó sự kiện **not_select_gamepad**

```
//không chọn tay cầm
socket.on('not_select_gamepad',function(){
    //alert('selected gamepad');
});
```

Do ở trên đã xử lý disable nút đó rồi nên hàm này trống

+Việc cuối cùng là kiểm tra trong phòng đó có đủ 2 người và 2 người đó đã chọn tay cầm hay chưa, nếu rồi thì sẽ vào map game, còn không thì sẽ chờ cho đến khi điều kiện trên đúng

```
// Tạo trò chơi cho người chơi trong phòng chờ
function joinWaitingPlayers(index) {
    var players = getClientsInRoom('room' + index);

    if (players.length >= 2 && players[0].gamepad != null && players[1].gamepad != null) {
        console.log('id 1 : ' + players[0].id);
        console.log('id 2 : ' + players[1].id);
        console.log('tay cam 1 : ' + players[0].gamepad);
        console.log('tay cam 2 : ' + players[1].gamepad);
        // 2 player waiting. Create new game!
        selected_gamepad = 0;
        amount_gamepad = 0;
        //var game = new BattleshipGame(gameIdCounter++, players[0].id, players[1].id);
        var game = new BattleshipGame(players[0].room, players[0].id, players[1].id);
        console.log("da vào game : ");
        // create new room for this game
        players[0].leave('room' + index);
        players[1].leave('room' + index);
        players[0].join('game' + players[0].room);
        players[1].join('game' + players[0].room);

        users[players[0].id].player = 0;
        users[players[1].id].player = 1;
        users[players[0].id].inGame = game;
        users[players[1].id].inGame = game;
        // users[players[0].id].gamepad = clients[0];
        // users[players[1].id].gamepad = clients[1];

        io.to('game' + game.id).emit('join', game.id);

        // send initial ship placements
        io.to(players[0].id).emit('update', game.getGameState(0, 0));
        io.to(players[1].id).emit('update', game.getGameState(1, 1));

        console.log((new Date().toISOString()) + " " + players[0].id + " and " + players[1].id + " have joined game ID " + game.id);
    }
}
```

+Hàm này sẽ lấy các socket đã vào phòng rồi kiểm tra số lượng và thuộc tính gamepad của socket đã có ip mac hay chưa

+Nếu điều kiện đúng nó sẽ tạo game. Khởi tạo game cần 3 đầu vào theo thứ tự: phòng chơi, socket id người 1, socket id người 2

+Cả hai người sẽ thoát khỏi phòng chờ và nhảy vào phòng game.

```
players[0].leave('room' + index);
players[1].leave('room' + index);
players[0].join('game' + players[0].room);
players[1].join('game' + players[0].room);
```

+Thiết lập của người chơi:

```
users[players[0].id].player = 0;
users[players[1].id].player = 1;
users[players[0].id].inGame = game;
users[players[1].id].inGame = game;
// users[players[0].id].gamepad = cli
```

+Server sẽ thông báo cho các clietn trong phòng game đó sự kiện **join** thiết lập map game ban đầu,và thuyền của cả hai người chơi:

```
io.to('game' + game.id).emit('join', game.id);

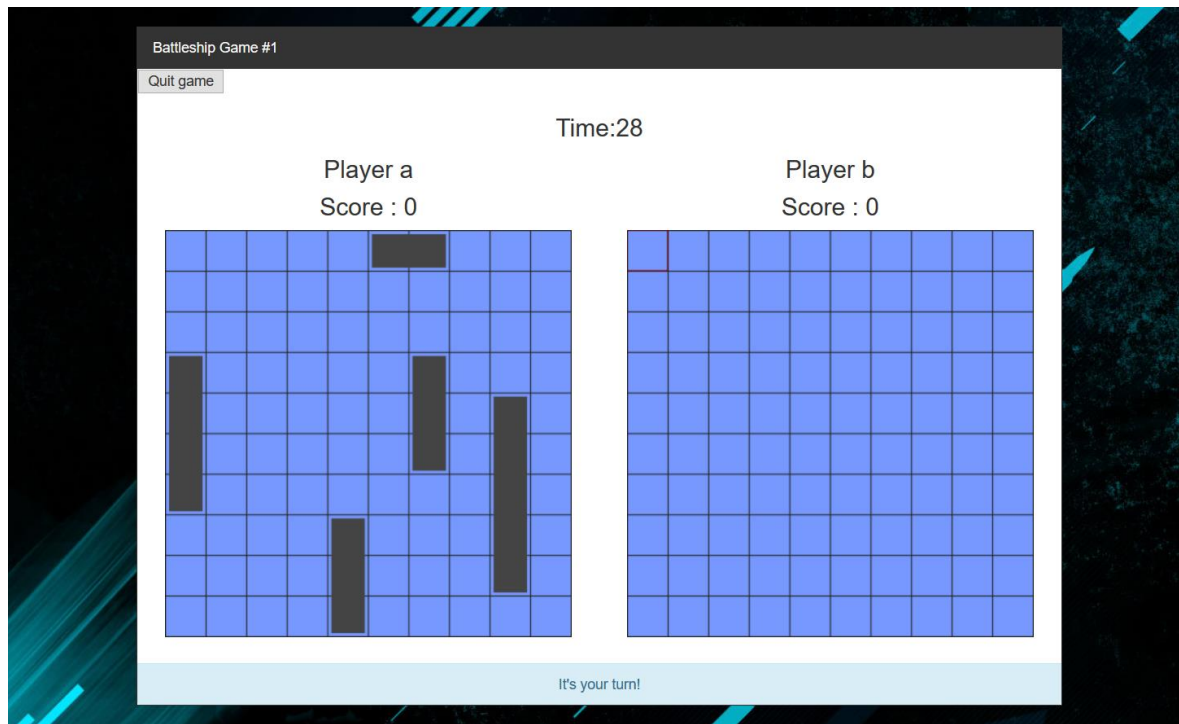
// send initial ship placements
io.to(players[0].id).emit('update', game.getGameState(0, 0));
io.to(players[1].id).emit('update', game.getGameState(1, 1));

console.log((new Date()).toISOString() + " " + players[0].id + " and " + players[1].id + " have joined game ID " + game.id);
```

```
socket.on('join', function(gameId) {
  Game.initGame();
  // $('#messages').empty();
  $('#disconnected').hide();
  $('#waiting-room').hide();
  $('#game').show();
  $('#game-number').html(gameId);

  var x = document.getElementById("user_you").textContent;
  socket.emit('user_opp',x);

})
```



2.2.7. Nhận thông tin từ tay cầm và xử lý

+Hàm nhận data trong server tcp sẽ có 2 trường hợp:

+**data là ip mac gồm 17 ký tự:** dùng để xác định ai đang chơi tay cầm nào, dù người chơi có rút tay cầm ra thì vẫn có thể kết nối lại nhờ ip mac

+**data là 1 ký tự:** dùng để di chuyển hoặc bắn

```

socket.on('data', function(data) {
    var line = data.toString(),
        c = 0,
        temp;
    if (line.length == 17) {
        console.log('id mac da nhan :' + line);
        mac_gamepad.push(line);
        var temp, index = gamepads.id.length - 1, tempmac, flag = 0;
        gamepads.mac = mac_gamepad;

        temp = gamepads.id[index];
        tempmac = gamepads.mac[index];
        for (c = 0; c < gamepads.id.length - 1; c++) {
            if (gamepads.mac[c] == tempmac)
            {
                gamepads.id[c] = temp;
                flag = 1;
            }
        }
        if (flag == 1)
        {
            gamepads.id.splice(index, 1);
            gamepads.status.splice(index, 1);
            gamepads.player.splice(index, 1);
            gamepads.mac.splice(index, 1);
            io.sockets.emit('update_gamepad', gamepads);
        }
        for (c = 0; c < gamepads.id.length; c++) {
            for (temp = 0; temp < player_socket.id.length; temp++) {
                if (gamepads.mac[c] == player_socket.mac_gamepad[temp]) {
                    gamepads.status[c] = 1;
                    gamepads.player[c] = player_socket.id[temp];
                    //xóa tay cầm tại vị trí c vừa tìm dc
                    io.sockets.emit('update_gamepad', gamepads);
                }
            }
        }
    }
}

```

+ code trên là nhận **ip mac**.

+ hình dưới là di chuyển hoặc bắn

```

for (c = 0; c < gamepads.id.length; c++) {
    //nhiều tay cam gửi lên --> tay chọn mới gửi
    if (socket.name == gamepads.id[c] && gamepads.status[c] == 1) {
        io.to(gamepads.player[c]).emit('thao_tac', line);
        setTimeout(function() {
            if (checkshoot == 2) {
                //trả rung về tay cầm hiện tại
                socket.write('^');
                //trả rung về tay cầm đối thủ
                var socket_opponent=null;

                //tìm socket id của đối thủ
                for(var x=0;x<gamepads.player.length;x++)
                {
                    if(player_opponent==gamepads.player[x])
                    {
                        socket_opponent=gamepads.id[x];
                    }
                }
                console.log('socket opp : ' + socket_opponent);
                //lấy socket của id đối thủ trả về kết quả
                for(var b1=0;b1<socket_tcp.length;b1++)
                {
                    if(socket_opponent==socket_tcp[b1].name)
                    {
                        socket_tcp[b1].write('!');
                    }
                }
                checkshoot = 0;
            }
            if (checkshoot == 1) {
                //socket.write('shoot missed');
                checkshoot = 0;
            }
        }, 500);
    }
}
}

```

+xử lý nhận dữ liệu 1 ký tự như sau:

+xét trong mảng id gamepads với socket.name (chứa ip : port) và trạng thái tay cầm của nó nếu đã chọn thì server io sẽ thông báo sự kiện thao_tac tới socket id đã lưu trong mảng gamepads.player

+Có một biến toàn cục tên checkshoot để kiểm tra nó đã bắn trúng hay trật ,khi ta chạy riêng các dòng lệnh của hàm trong setTimeout mà không có time out

thì biến checkshoot nó sẽ không cập nhập kết quả mới vì nguyên tắc thực thi chạy lệnh của nodejs.

Các lệnh trong hàm nó sẽ chạy các lệnh đồng bộ trước (tức là theo lệnh từ trên xuống dưới nhưng không phụ thuộc vào kết quả) rồi mới chạy các lệnh bất đồng bộ sau.

Do lệnh io .emit đã là bất đồng bộ nên ta cần đặt hàm checkshoot trong setTimeout để trở thành bất đồng bộ luôn. Thì kết quả là **io emit** sẽ **thực thi trước** xong rồi thì mới tới lệnh trong setTimeout sau **khoảng thời gian đủ** để io emit **thực thi xong**.

+Phía **client** nhận sự kiện **thao_tac**:

```
// Xu ly xu lieu nhan
socket.on('thao_tac', function(data) {
  //alert(data);
  Game.dichuyen_shoot(data);

  //add new,

});
```

```

function dichuyen_shoot(e) {
  if (turn) {
    if (e == 'r') //right
    {
      stepnew_old(toado.y, toado.x, 1, 0);

      toado.x = toado.x + 1;
      if (toado.x > (gridCols - 1)) {
        toado.x = 0;
      }
      stepnew_old(toado.y, toado.x, 1, 1);
    }
    if (e == 'l') //left
    {
      stepnew_old(toado.y, toado.x, 1, 0);

      toado.x = toado.x - 1;
      if (toado.x < 0) {
        toado.x = gridCols - 1;
      }
      stepnew_old(toado.y, toado.x, 1, 1);
    }
    if (e == 'u') //up
    {
      stepnew_old(toado.y, toado.x, 1, 0);

      toado.y = toado.y - 1;
      if (toado.y < 0) {
        toado.y = gridRows - 1;
      }
      stepnew_old(toado.y, toado.x, 1, 1);
    }
  }
}

```

```

        if (e == 'd') //down
        {
            stepnew_old(toado.y, toado.x, 1, 0);

            toado.y = toado.y + 1;
            if (toado.y > (gridCols - 1)) {
                toado.y = 0;
            }
            stepnew_old(toado.y, toado.x, 1, 1);
        }

        if (e == 's') //nut o ban
        {
            //if(grid[1].shots[toado.x * gridCols + toado.y] === 0)
            sendShot(toado);
            clearInterval(myVar);
        }
    }
};

```

Hàm này sẽ thực hiện di chuyển lên xuống qua lại và bắn.

+Khi di chuyển qua vị trí mới thì phải trả về trạng thái cũ của vị trí cũ. Hàm **stepnew_old** thực hiện việc này. Các tham số theo thứ tự là 2 tọa độ vị trí, vị trí map (có 0 là bên trái và 1 là bên phải), bước mới(=1) hoặc cũ (=0)

+Khi bắn thì gửi tọa độ về cho **server** xử lý:

```

function sendShot(square) {
    socket.emit('shot', square);
}

```

Phía server sẽ lắng nghe sự kiện **shot**:

```

// Xử lý client
socket.on('shot', function(position) {
    var game = users[socket.id].inGame,
        opponent;

    if (game !== null) {
        // Is it this users turn?
        if (game.currentPlayer === users[socket.id].player) {
            opponent = game.currentPlayer === 0 ? 1 : 0;

            if (game.shoot(position)) {
                if (game.check_shoot == 2) {
                    checkshoot = game.check_shoot;
                    //socket.emit('shoot succeeded');
                    //bắn trúng thì cập nhập điểm
                    io.to(socket.id).emit('score', game.get_score());
                    io.to(game.getPlayerId(opponent)).emit('score_opp', game.get_score());
                    player_opponent=game.getPlayerId(opponent);
                }
                if (game.check_shoot == 1) {
                    checkshoot = game.check_shoot;
                }
                // Valid shot
                checkGameOver(game);

                //console.log('score hien tai:'+game.get_score())

                // Update game state on both clients.
                io.to(socket.id).emit('update', game.getGameState(users[socket.id].player, opponent));
                io.to(game.getPlayerId(opponent)).emit('update', game.getGameState(opponent, opponent));
            }
        }
    }
});

```

+Đầu tiên lấy game của socket đang chơi thông qua socket id

+Kiểm tra đến lượt của mình chưa nếu đúng thì thực hiện việc bắn bằng hàm game.shoot(vị trí). Kết quả trả về là true nếu bắn được,false nếu không bắn được

```

BattleshipGame.prototype.shoot = function (position) {
    var opponent = this.currentPlayer === 0 ? 1 : 0,
        // vị trí bắn
        gridIndex = position.y * Settings.gridCols + position.x;

    //chưa bắn thì = 0
    if (this.players[opponent].shots[gridIndex] === 0 && this.gameStatus === GameState.inProgress) {
        // Square has not been shot at yet.
        // nhảy vào là if 1 do bắn
        if (!this.players[opponent].shoot(gridIndex)) {
            this.check_shoot=1;//shoot missed
            // chuyển lượt
            // Miss
            this.switchPlayer();
        }
        else
        {
            this.check_shoot=2;//shoot succeeded
            this.players[this.currentPlayer].score++;
        }

        // Check if game over
        if (this.players[opponent].getShipsLeft() <= 0) {
            this.gameStatus = GameState.gameOver;
            this.winningPlayer = opponent === 0 ? 1 : 0;
        }

        return true;
    }

    return false;
};

```

+Hàm này sẽ kiểm tra vị trí của đối thủ đã bắn hay chưa và trạng thái của game đang chơi hay kết thúc. Nếu đúng thì sẽ thực hiện bắn. Mỗi người chơi có 2 mảng :

Mảng shots	Giá trị ban đầu là 0,nếu bắn trúng =2,bắn trượt =1
Mảng shipGrid	Giá trị ban đầu là -1,nếu có tàu sẽ set >=0

+khi chạy tới lệnh thì player sẽ thực hiện hàm của nó:

```

this.players[opponent].shoot(gridIndex)

```

```

// Fire shot on grid
// Ban tau
// @param {type} gridIndex
// @returns {Boolean} True if hit
// tban trung hay ko bat trung true
Player.prototype.shoot = function (gridIndex) {
    //
    if (this.shipGrid[gridIndex] >= 0) {
        // Hit!
        this.ships[this.shipGrid[gridIndex]].hits++;
        this.shots[gridIndex] = 2;

        return true;
    } else {
        // Miss
        this.shots[gridIndex] = 1;
        return false;
    }
};

```

Kiểm tra mảng shipGrid có tàu hay không. Nếu có thì tăng biến bắn trúng tàu lên 1 và gán trong mảng shots tại vị trí đó =2 (tức bắn trúng). Ngược lại thì gán trong mảng shots tại vị trí đó =1 (tức bắn trượt)

+Sau khi thực hiện bắn xong thì sẽ check

```

if (!this.players[opponent].shoot(gridIndex)) {
    this.check_shoot=1;//shoot missed
    // chuyển lượt
    // Miss
    this.switchPlayer();
}
else
{
    this.check_shoot=2;//shoot succeeded
    this.players[this.currentPlayer].score++;
}

```

Nếu bắn trật sẽ set biến check_shoot =1 và đổi lượt,ngược lại nếu trúng thì check_shoot =2 và tăng số điểm hiện tại của người chơi

+sau khi bắn sẽ kiểm tra đã thua hay chưa:

```

// Check if game over
if (this.players[opponent].getShipsLeft() <= 0) {
    this.gameStatus = GameStatus.gameOver;
    this.winningPlayer = opponent === 0 ? 1 : 0;
}

```

+sau đó nó sẽ trả về true nếu bắn được và false nếu không bắn được.Quay về tiếp tục phía server sự kiện **shot**.

```

// xử lý client
socket.on('shot', function(position) {
    var game = users[socket.id].inGame,
        opponent;

    if (game !== null) {
        // Is it this users turn?
        if (game.currentPlayer === users[socket.id].player) {
            opponent = game.currentPlayer === 0 ? 1 : 0;

            if (game.shoot(position)) {
                if (game.check_shoot == 2) {
                    checkshoot = game.check_shoot;
                    //socket.emit('shoot succeeded');
                    //bắn trúng thì cập nhập điểm
                    io.to(socket.id).emit('score', game.get_score());
                    io.to(game.getPlayerId(opponent)).emit('score_opp', game.get_score());
                    player_opponent=game.getPlayerId(opponent);
                }
                if (game.check_shoot == 1) {
                    checkshoot = game.check_shoot;
                }
                // Valid shot
                checkGameOver(game);

                //console.log('score hiện tại:'+game.get_score())

                // Update game state on both clients.
                io.to(socket.id).emit('update', game.getGameState(users[socket.id].player, opponent));
                io.to(game.getPlayerId(opponent)).emit('update', game.getGameState(opponent, opponent));
            }
        }
    }
});

```

+sau khi thực thi lệnh **game.shoot(position)** ,ta kiểm tra nếu biến **check_shoot = 2** tức là trúng thì ta sẽ cập nhập biến **checkshoot**(dùng để trả về tín hiệu cho tay cầm bắn) và biến toàn cục **player_opponent= socket id** của đối thủ (để trả về tín hiệu tay cầm đối thủ)

+các câu lệnh tiếp là theo kiểm tra game kết thúc hay chưa và update lại mapgame của cả hai người chơi

```

}
// Valid shot
checkGameOver(game);

//console.log('score hiện tại:'+game.get_score())

// Update game state on both clients.
io.to(socket.id).emit('update', game.getGameState(users[socket.id].player, opponent));
io.to(game.getPlayerId(opponent)).emit('update', game.getGameState(opponent, opponent));

```

+Sau khi xử lý xong sự kiện **shot** thì hàm trong Settimeout sẽ thực thi


```

//nếu tay cầm gửi lên => tay chọn mới gửi
if (socket.name == gamepads.id[c] && gamepads.status[c] == 1) {
    io.to(gamepads.player[c]).emit('thao_tac', line);
    setTimeout(function() {
        if (checkshoot == 2) {
            //trả rung về tay cầm hiện tại
            socket.write('^');
            //trả rung về tay cầm đối thủ
            var socket_opponent=null;

            //tìm socket id của đối thủ
            for(var x=0;x<gamepads.player.length;x++)
            {
                if(player_opponent==gamepads.player[x])
                {
                    socket_opponent=gamepads.id[x];
                }
            }
            console.log('socket opp : ' + socket_opponent);
            //lấy socket của id đối thủ trả về kết quả
            for(var b1=0;b1<socket_tcp.length;b1++)
            {
                if(socket_opponent==socket_tcp[b1].name)
                {
                    socket_tcp[b1].write('!');
                }
            }
            checkshoot = 0;
        }
        if (checkshoot == 1) {
            //socket.write('shoot missed');
            checkshoot = 0;
        }
    }, 500);
}
}
}

```

Mức thời gian ví dụ như 500 trong hàm này tức là sau 500 ms thì hàm này sẽ được đưa vào hàng chờ để thực thi.

+Lúc này biến **checkshoot** và **player_opponent(socket id đối thủ)** đã được cập nhập xong.

+nếu **checkshoot =2** thì sẽ gửi về tay cầm hiện tại dấu ^ (**socket.write('^');**) . Để gửi về **tín hiệu !** cho phía **tay cầm đối thủ** thì ta dựa vào **player_opponent** để

duyệt mảng gamepads tìm được **socket.name** (địa chỉ **ip:port**) trong **gamepads.id[]**. Tiếp theo ta dựa vào **socket.name** để tìm trong mảng **socket_tcp** có thuộc tính **name** khớp với nó thì trả về !

```
socket_tcp[b1].write('!');
```

+ xử lý nhận data là ip mac gồm 17 ký tự:

+Mục đích:trong trường hợp rút tay cầm ra thì cấm lại vẫn kết nối với người chơi cũ

```
temp,
if (line.length == 17) {
    console.log('id mac da nhan :' + line);
    mac_gamepad.push(line);
    var temp,index=gamepads.id.length-1,tempmac,flag=0;
    gamepads.mac = mac_gamepad;

    temp=gamepads.id[index];
    tempmac=gamepads.mac[index];
    for (c=0; c < gamepads.id.length-1; c++) {
        if(gamepads.mac[c]==tempmac)
        {
            gamepads.id[c] = temp;
            flag=1;
        }
    }
    if(flag==1)
    {
        gamepads.id.splice(index, 1);
        gamepads.status.splice(index, 1);
        gamepads.player.splice(index, 1);
        gamepads.mac.splice(index, 1);
        io.sockets.emit('update_gamepad', gamepads);
    }
    for (c=0; c < gamepads.id.length; c++) {
        for (temp = 0; temp < player_socket.id.length; temp++) {
            if (gamepads.mac[c] == player_socket.mac_gamepad[temp]) {
                gamepads.status[c] = 1;
                gamepads.player[c] = player_socket.id[temp];
                //xóa tay cầm tại vị trí c vừa tìm dc
                io.sockets.emit('update_gamepad', gamepads);
                io.(gamepads.player[c]).emit('connect_gamepad',gamepads.mac[c]);
            }
        }
    }
}
```

+ý tưởng:khi vừa kết nối tay cầm sẽ gửi thêm địa chỉ **mac**,cũng lưu các thông tin rồi cập nhập vào gamepads như thường. Tiếp theo lấy ngay mac **vị trí**

cuối của mảng mac trong gamepad(tức cái vừa kết nối). Ta duyệt mảng mac từ đầu đến vị trí kế cuối, nếu đã có mac thì sẽ bật flag =1 và lưu id (ip :port) tại vị trí cuối vào **phần tử cùng mac** trong **mảng id** của **gamepads**(tức địa chỉ ip :port mới của tay cầm cũ). Xét nếu flag bằng 1 thì xóa phần tử vừa thêm vào đối tượng gamepads. Sau đó sẽ set trạng thái và socket id của người cũ.

Chương 3. KẾT QUẢ THỰC NGHIỆM

3.1. Tính năng đăng nhập vào hệ thống.

3.1.1. Các trường hợp thử kiểm tra.

- Kiểm tra trường hợp 1: Đăng nhập không có username password → Hệ thống báo lỗi.
- Kiểm tra trường hợp 2: Đăng nhập nhập username không nhập password → Hệ thống báo lỗi.
- Kiểm tra trường hợp 3: Đăng nhập không nhập username nhập password → Hệ thống báo lỗi.
- Kiểm tra trường hợp 4: Đăng nhập nhập username sai nhập password đúng → Hệ thống báo lỗi.
- Kiểm tra trường hợp 5: Đăng nhập nhập username đúng nhập password sai → Hệ thống báo lỗi.
- Kiểm tra trường hợp 6: Đăng nhập nhập username đúng nhập password đúng → **Hệ thống đăng nhập thành công.**
- Kiểm tra trường hợp 7: Đăng kí không nhập username password và repassword → Hệ thống báo lỗi.
- Kiểm tra trường hợp 8: Đăng kí nhập username không nhập password và repassword → Hệ thống báo lỗi.

- Kiểm tra trường hợp 9: Đăng kí nhập username nhập password và không nhập repassword → Hệ thống báo lỗi.
- Kiểm tra trường hợp 10: Đăng kí nhập username không nhập password và nhập repassword → Hệ thống báo lỗi.
- Kiểm tra trường hợp 11: Đăng kí nhập username nhập password và repassword nhưng không trùng khớp → Hệ thống báo lỗi.
- Kiểm tra trường hợp 12: Đăng kí nhập username nhập password và repassword khớp nhưng usenaem đã tồn tại → Hệ thống báo lỗi.
- Kiểm tra trường hợp 13: Đăng kí nhập username nhập password và repassword khớp và username chưa tồn tại → **Hệ thống đăng nhập thành công.**

3.1.2. Video kiểm tra thực tế các trường hợp trên

<https://bit.ly/2PU132h>

3.2. Tính năng kết nối gamepad chơi game.

3.2.1. Các trường hợp thử kiểm tra.

Đăng nhập, tạo phòng, chơi game, di chuyển bắn trúng rung va fbij bắn trúng cũng rung. Người chơi thắng hiện thông báo là chiến thắng và người thua thương tự.

3.2.2. Link github:

<https://github.com/dovankhai/Game-ban-tau>

Các video trong thư mục test

