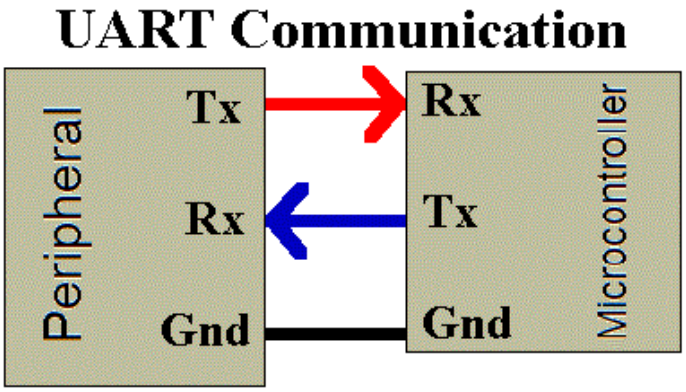
CHƯƠNG 2 CƠ SỞ LÝ THUYẾT

2.1. Giao thức UART

2.1.1 Khái niệm

- UART (Universal Asynchronous Receiver – Transmitter) là bộ truyền nhận nối tiếp không đồng bộ.



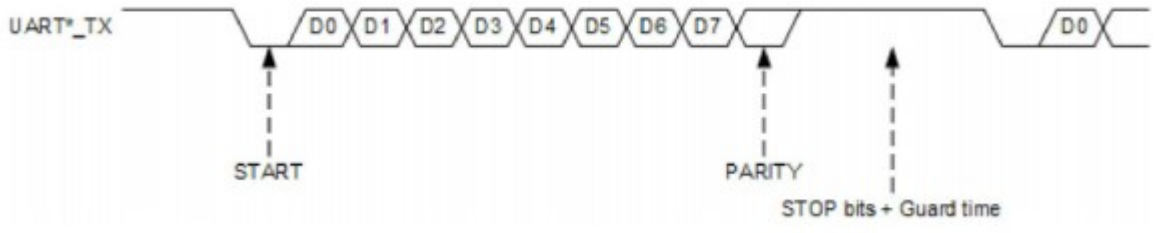
Hình 2.1 Sơ đồ truyền UART

2.1.2 Các thông số cơ bản

- Baud rate: thông số xác định xác định tốc độ truyền. Cần được cài cài đặt giống nhau ở cả thiết bị truyền và thiết bị nhận.
- Frame: Khung truyền quy định về số bit trong mỗi lần truyền.
- Start bit: Là bit đầu tiên được truyền trong 1 Frame. Báo cho thiết bị nhận có một gói dữ liệu sắp được truyền đến.
- Data: Dữ liệu cần truyền. Bit có trọng số nhỏ nhất LSB được truyền trước sau đó đến bit MSB.
- Parity bit: Bit kiểm tra dữ liệu.
- Stop bit: là bit báo cho thiết bị rằng các data đã được gửi xong. Thiết bị nhận sẽ tiến hành kiểm tra khung truyền nhằm đảm bảo tính đúng đắn của dữ liệu.

2.1.3 Hoạt động

– Để bắt đầu cho việc truyền dữ liệu bằng UART, một START bit được gửi đi, sau đó là các bit dữ liệu và kết thúc quá trình truyền là bit STOP bit.



Hình 2.2 Sơ đồ hoạt động của UART

– Khi ở trạng thái chờ mức điện thế ở mức 1 (high).

– Khi bắt đầu truyền, START bit sẽ chuyển từ 1 xuống 0 để báo hiệu cho bộ nhận là quá trình truyền dữ liệu sắp xảy ra.

– Sau START bit là đến các bit dữ liệu D0-D7 (các bit này có thể mức High hay Low tùy theo dữ liệu).

– Cuối cùng là STOP bit là bảo cho thiết bị rằng các bit đã được gửi xong. Thiết bị nhận sẽ tiến hành kiểm tra khung truyền nhằm đảm bảo tính đúng đắn của dữ liệu.

2.2. Linh kiện được sử dụng trong quá trình thực hiện đồ án.

2.2.1. Board Gamepad

Board được vẽ mạch và đặt làm mạch, ngoài ra, nhóm tự hàn các linh kiện điện tử lên board để đảm bảo cho board có thể hoạt động. Thành phần chính xử lý của board là chip NUC140, thuộc dòng vi điều khiển 32 bits của hãng Nuvoton, dựa trên lõi ARM Cortex-M0.

Hình 2.3 Board Gamepad

❖ Thông số kỹ thuật.

- Kiến trúc ARM Cortex M0 32 bits với tốc độ tối đa 50 MHZ.
- Chuẩn giao tiếp: I2C, SPI, UART/USART.
- Ngoại vi: ADC, DMA, PWM, WDT, RTC.
- Điện áp hoạt động: 2.5 V to 5.5 V.

2.2.2. Chip ESP8266

❖ ESP8266 sử dụng trong các sản phẩm IOT và ứng dụng truyền nhận dữ liệu qua wifi.

❖ Thông số kỹ thuật.

- Tốc độ xử lý: 80 MHZ.
- WiFi: 2.4 GHz hỗ trợ chuẩn 802.11 b/g/n
- Điện áp hoạt động: 3.3V.
- Số chân I/O: 11 (tất cả các chân I/O đều có Interrupt/PWM/I2C/One-wire, trừ chân D0).
- Số chân Analog Input: 1 (điện áp vào tối đa 3.3V).
- Bộ nhớ Flash: 4MB.
- Hỗ trợ bảo mật: WPA/WPA2.
- Tích hợp giao thức TCP/IP.
- Lập trình trên các ngôn ngữ: C/C++, Micropython, NodeMCU – Lua.

2.2.3. Động cơ rung.

Nhóm không hiện thực động cơ rung.

Các thiết bị ngoại vi này đều được gắn lên mạch Gamepad cho sự nhỏ gọn.

2.3. Socket IO

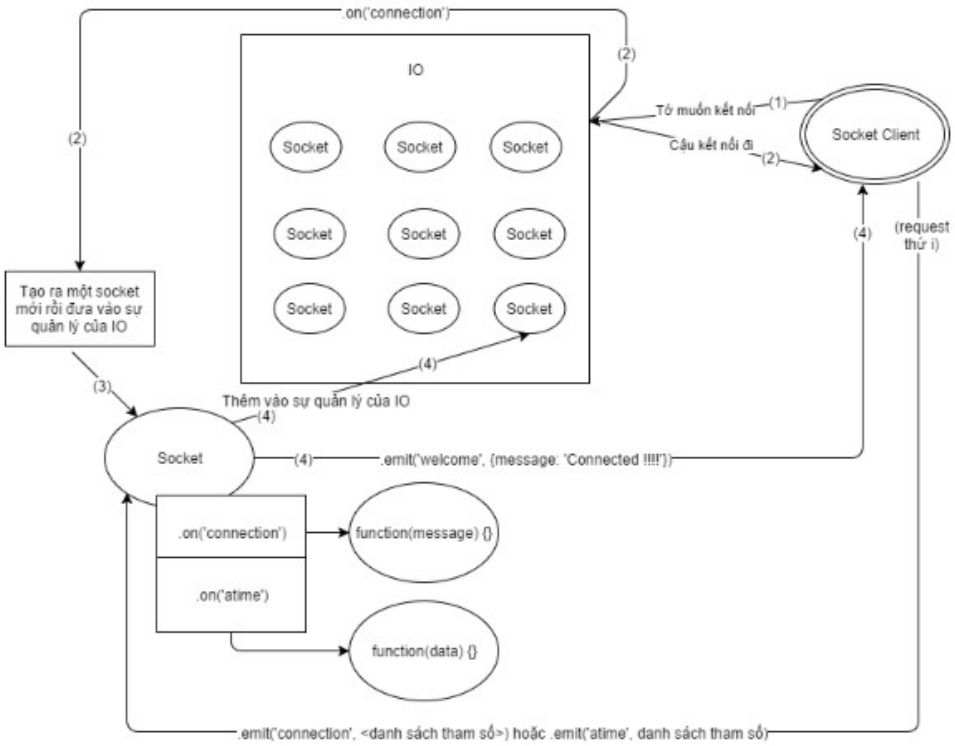
Socket IO là một bộ thư viện dành cho các ứng dụng web, mobile realtime. Với đặc trưng mạnh mẽ và dễ sử dụng. Thư viện gồm 2 phần:

- Phía client: Gồm bộ thư viện viết cho Web(JavaScript), IOS, Android.
- Phía server: Viết bằng JavaScript và sử dụng cho máy chủ Node.JS

Một số hàm cơ bản:

- Connect(): kết nối với server socket.
- On (event_name, listener): đăng kí lắng nghe sự kiện.
- Emit (event_name, data): gửi một sự kiện đi.
- Off(event_name): ngừng lắng nghe một sự kiện nào đó.

Cơ chế hoạt động.

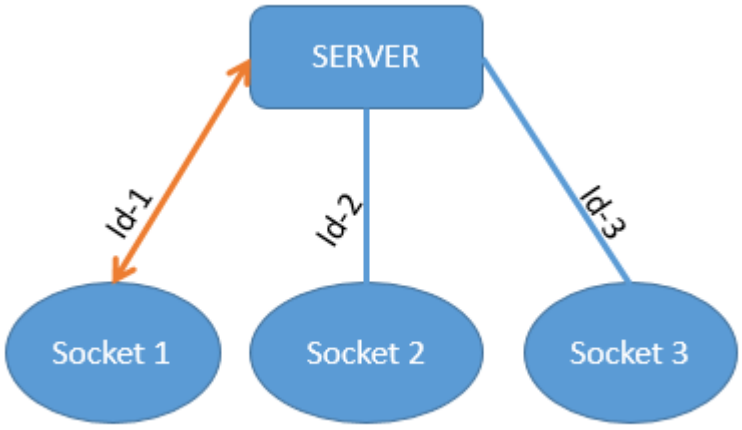


Hình 2.3 Mô hình hoạt động của socket IO

Khi client muốn kết nối tới server thì gửi socket.emit() với tham số là "Connection" tới server và server dùng hàm socket.on() dùng để lắng nghe sự kiện từ client. Nếu như server nhận được sự kiện thì nó sẽ dùng hàm socket.emit() để trả về cho client với tham số "connected". Socket sẽ được thêm vào sự quản lý của IO. Vì vậy mà ứng dụng dễ dàng quản lý và mở rộng khi cần thiết.

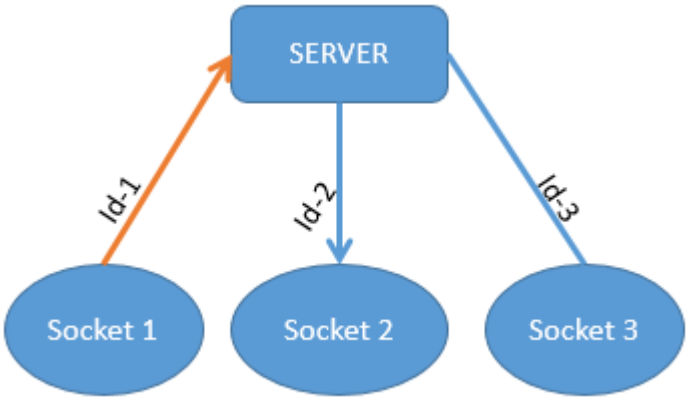
Socket.IO sử dụng kết nối socket nên client và server chỉ thực hiện bắt tay 3 bước lần đầu tiên, sau đó, khi kết nối đã được thiết lập, việc truyền dữ liệu qua lại giữa client-server và ngược lại chỉ đơn thuần dựa trên 2 sự kiện: on và emit. Điều này mang lại rất nhiều lợi ích cho ứng dụng cần tính realtime như chat app, multiplayer game, notification app, real-time app,... Cụ thể:

1. Tốc độ truyền rất nhanh vì bỏ qua được cơ chế bắt tay 3 bước. Ở HTTP request truyền thống, mỗi một request cần bắt tay 3 bước sau đó mới đến bước truyền dữ liệu. Với Socket.IO, cơ chế bắt tay 3 bước này chỉ xảy ra khi thực hiện một kết nối mới lần đầu tiên. Sau khi kết nối đã được thiết lập, các lần truyền nhận dữ liệu sau chỉ cần 1 bước, gửi hoặc nhận.
2. Cho phép server chủ động gửi dữ liệu xuống client thay vì server phải chờ request từ client mới gửi như HTTP request truyền thống. Điều này đảm bảo tính real-time trong ứng dụng, khi server cần thông báo ngay lập tức cho một hoặc các client khi có sự kiện nào đó xảy ra (ví dụ có người chơi mới connect,...)
3. Cho phép server gửi data cho client trong nhiều hoàn cảnh khác nhau:
    - Chỉ gửi lại cho socket đã emit lên: socket.emit("eventname", message)



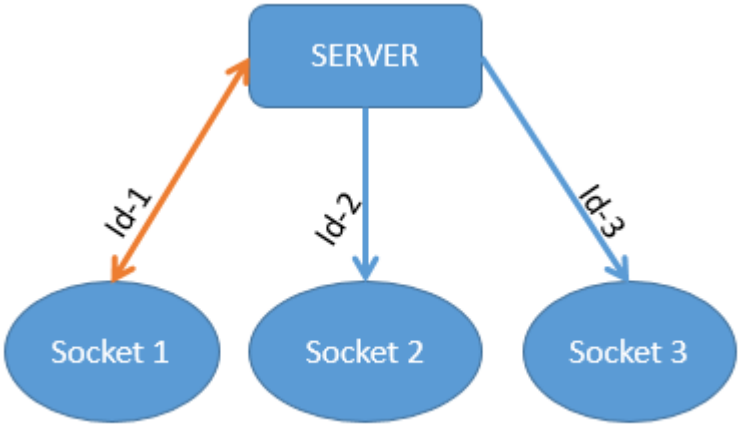
Hình 2.3.1. Chỉ emit cho socket đã emit lên

- Gửi cho socket với id được chỉ định:
  io.to(socketid).emit("eventname", message);



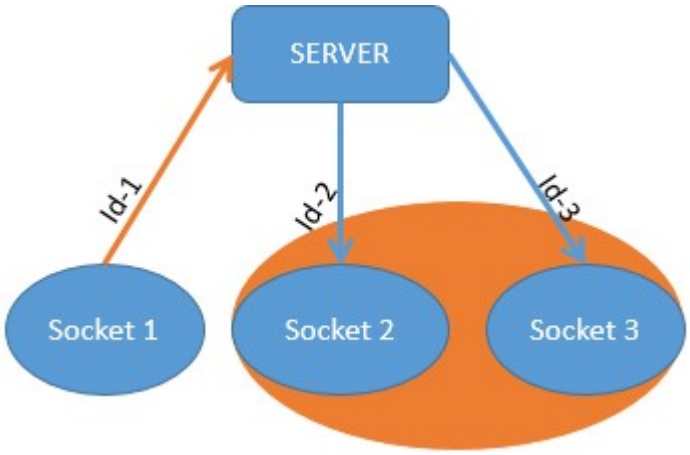
Hình 2.3.2. Emit cho Socket Id chỉ định

- Gửi broadcast cho tất cả socket trừ socket gửi lên:
  socket.broadcast.emit("eventname", message)



Hình 2.3.3. Emit broadcast

- Gửi cho room: io.to('some room').emit('some event'):



Hình 2.3.4: Emit vào room

2.4. Tổng quan về NodeJS

2.4.1. Khái niệm

Node.js là một nền tảng chạy trên môi trường V8 JavaScript runtime - một trình thông dịch JavaScript cực nhanh chạy trên trình duyệt Chrome.

2.4.2. Ưu điểm

- Phần core bên dưới của Nodejs được viết hầu hết bằng C++ nên cho tốc độ xử lý và hiệu năng khá cao.
- NodeJS có cú pháp Javascript nên dễ học, dễ viết
- Nodejs tạo ra được các ứng dụng có tốc độ xử lý nhanh, realtime thời gian thực.
- NodeJS áp dụng cho các sản phẩm có lượng truy cập lớn, cần mở rộng nhanh, cần đổi mới công nghệ.
- Những ứng dụng có thể viết bằng Nodejs: Websocket server, Fast file upload client, ad server, cloud services, Restful API.
- NodeJS sử dụng non-block single thread nên tương tự như ngắt trong các hệ thống Embedded, rất nhanh.

2.4.3. Nhược điểm.

- Khó kiểm soát luồng chạy của chương trình vì cơ chế bất đồng bộ
- Cũng do single thread nên nếu thread này bị trục trặc thì sẽ gây dừng hoạt động của cả hệ thống

2.5. Tổng quan về MySQL

2.5.1. Khái niệm

MySQL là hệ quản trị cơ sở dữ liệu tự do nguồn mở phổ biến nhất thế giới và được các nhà phát triển rất ưa chuộng trong quá trình phát triển ứng dụng. Vì MySQL là cơ sở dữ liệu tốc độ cao, ổn định và dễ sử dụng, có tính khả chuyển, hoạt động trên nhiều hệ điều hành cung cấp một hệ thống lớn các

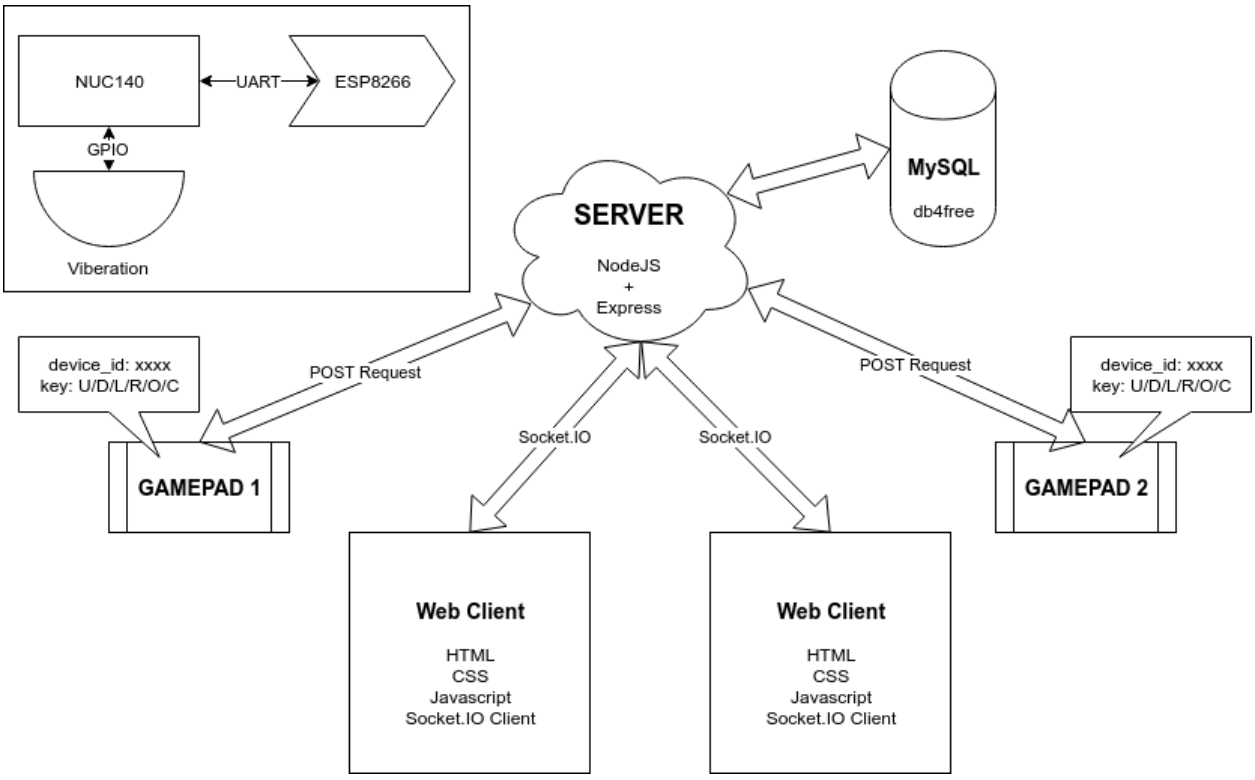
hàm tiện ích rất mạnh. Với tốc độ và tính bảo mật cao, MySQL rất thích hợp cho các ứng dụng có truy cập CSDL trên internet.

2.5.2. Ưu điểm

- Là hệ quản trị CSDL quan hệ (RDBMS)
- Hoạt động theo mô hình client/server, làm database server cho các client kết nối đến
- Hỗ trợ SQL tiêu chuẩn
- Có nhiều table format với đặc tính khác nhau, ví dụ MyISAM không hỗ trợ transaction, InnoDB hỗ trợ transaction…
- Có nhiều API, library hỗ trợ cho nhiều ngôn ngữ lập trình
- Hỗ trợ giao tiếp ODBC
- Có khả năng replication
- Có thể chạy trên nhiều platform: Mac OS X, Windows, Linux, Unix (HP-UX, Solaris,…)

CHƯƠNG 3 THIẾT KẾ VÀ HIỆN THỰC HỆ THỐNG

3.1. Mô hình thiết kế tổng thể của hệ thống



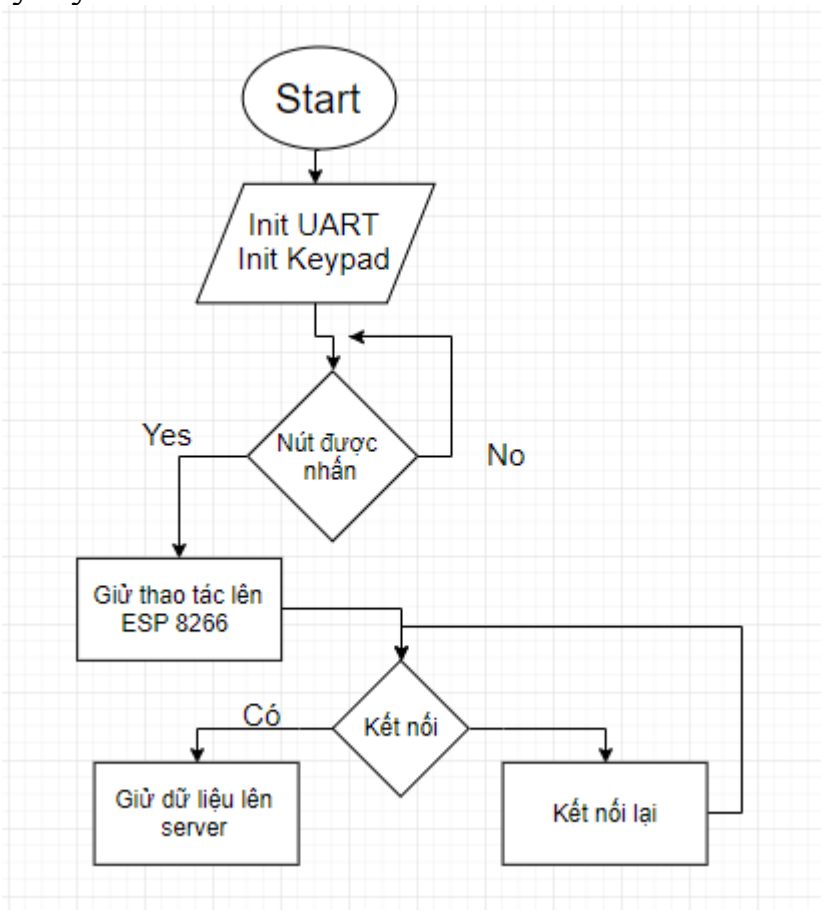
Hình 2.6 Mô hình tổng quát

❖ Hệ thống gồm 4 phần chính: Web client, Server, Database, Tay cầm.

- Web client sử dụng HTML, CSS, JavaScript và Socket.IO. Web client để nhận dữ liệu từ server gửi xuống, tạo room, join room, chọn tay cầm, bắn tàu, thời gian time out,….
- Server sử dụng NodeJS với framework Express, Ejs và Socket.IO. Dùng để nhận dữ liệu từ tay cầm gửi lên, xử lý các yêu cầu từ Webclient, giao tiếp dữ liệu với database.
- Database sử dụng MySQL kết nối đến trang bd4free cho mục đích testing
- Toàn bộ phần code server được deploy trên dịch vụ Heroku.
- Tay cầm là mạch tích hợp: NUC140+ESP8266+Các linh kiện khác.

Sử dụng UART để truyền nhận dữ liệu nút nhấn giữa NUC140 và ESP8266. ESP8266 sử dụng Lua script và module http để thực hiện HTTP Request lên server.

3.2. Quá trình xử lý tay cầm.



Hình 3.1: Sơ đồ khối hoạt động của tay cầm.

Khi nút được nhấn thì NUC140 sẽ gửi thao tác lên ESP8266 thông qua UART. ESP8266 sẽ gửi kèm mã phím được nhấn kèm theo device_id được hardcode với từng thiết bị thông qua HTTP POST request lên server.

3.3. Quá trình xử lý ở server.

a. Truyền dữ liệu từ client lên server:

Emit	Mô tả
socket.emit("room-username", logedInUser);	Gửi username lấy từ Cookie lên Server
socket.emit("room-logout");	Gửi lệnh Logout

socket.emit("room-setDevice", deviceName);	Gửi device name được player chọn
socket.emit("room-gotoRoom", newRoom);	Gửi lệnh chuyển màn hình vào chế độ chơi game khi người chơi tạo Room mới
socket.emit("room-gotoRoom", roomName);	Gửi lệnh chuyển màn hình vào chế độ chơi game khi người dùng Join Room

b. Truyền dữ liệu từ server xuống client:

Emit	Mô tả
io.sockets.emit("deviceList", deviceListToChoice);	Emit danh sách các thiết bị đã kết nối và đang chờ được chọn mỗi khi một người dùng login thành công vào giao diện set device và chọn room
io.sockets.emit("roomList", roomList);	Emit danh sách các phòng đang available (chỉ có 1 người) và đang chờ được chọn mỗi khi một người dùng login thành công vào giao diện set device và chọn room
socket.emit("logedOut");	Thông báo trên phía server đã xử lý logout thành công
socket.emit("goToNewRoomBro", room);	Thông báo trên server đã xử lý thành công yêu cầu vào phòng của client
socket.emit("yourMap", yourMap);	Gửi map đã được server random xuống cho client
socket.emit("joinRoomFail");	Thông báo nếu có bất cứ sự cố gì khi người chơi join vào room
socket.broadcast.emit("roomList", roomList);	Broadcast room list mới sau khi người chơi nào đó đã chọn gamepad của mình từ danh sách các gamepad available. Room list mới này tất nhiên sẽ trừ gamepad đã được chọn

	ra.
io.to(player.getRoom()).emit("gameOver");	Thông báo Game Over cho cả room
io.to(player.getSocketId()).emit("changeLocation", newP2Location);	Emit sự kiện người chơi vừa sử dụng gamepad để di chuyển trên map
io.to(player.getSocketId()).emit("hitOrMiss", hitOrMiss);	Trả về cho người chơi vừa bắn là trúng hay trượt
io.to(room.getPlayer2().getSocketId()).emit("opponentHitOrMissYou", notifyToPlayer2)	Trả về cho người chơi còn lại trong room là đối thủ của người chơi đó vừa bắn trúng hay trượt mình
io.to(player.getSocketId()).emit("yourTurn");	Thông báo cho người chơi đã đến lượt của mình bắn
io.to(player.getSocketId()).emit("opponentTurn");	Thông báo đến người chơi là lượt hiện tại là lượt của đối thủ

3.4. Luồng thao tác:

a. Người dùng đăng nhập hoặc đăng kí:

b. Người chơi kết nối tay cầm lên và chọn tay cầm của mình

c. Người chơi tạo room mới hoặc join room

d. Chơi game

e. Kết thúc

f. Logout

Code NUC140:

```
#include <stdio.h>
#include "Driver\DrvUART.h"
#include "Driver\DrvGPIO.h"
#include "Driver\DrvSYS.h"
#include "NUC1xx.h"

void uart_sendStr(uint8_t *str);

void GPIOAB_INT_CallBack(uint32_t GPA_IntStatus, uint32_t GPB_IntStatus)
{
if ((GPA_IntStatus>>12) & 0x01){
uart_sendStr("O!\n");
}
if ((GPA_IntStatus>>13) & 0x01){
uart_sendStr("C!\n");
}
}

void GPIOCDE_INT_CallBack(uint32_t GPC_IntStatus, uint32_t GPD_IntStatus, uint32_t GPE_IntStatus)
{
if ((GPC_IntStatus>>1) & 0x01){
uart_sendStr("D!\n");
}
if ((GPC_IntStatus>>2) & 0x01){
uart_sendStr("R!\n");
}
if ((GPC_IntStatus>>3) & 0x01){
uart_sendStr("L!\n");
}
```

```
if ((GPD_IntStatus>>7) & 0x01){
uart_sendStr("U!\n");
}
}

void initLed(void){
DrvGPIO_Open(E_GPA, 15, E_IO_OUTPUT); // GPC12 pin set to output mode
DrvGPIO_ClrBit(E_GPA, 15); // Goutput Hi to turn off LED
DrvGPIO_Open(E_GPA, 14, E_IO_OUTPUT); // GPC12 pin set to output mode
DrvGPIO_ClrBit(E_GPA, 14); // Goutput Hi to turn off LED
}

void interruptConfig(){
DrvGPIO_Open(E_GPA,12,E_IO_INPUT);
DrvGPIO_Open(E_GPA,13,E_IO_INPUT);
DrvGPIO_EnableInt(E_GPA, 12, E_IO_RISING, E_MODE_EDGE); //BTN OK
DrvGPIO_EnableInt(E_GPA, 13, E_IO_RISING, E_MODE_EDGE); //BTN CANCLE
DrvGPIO_EnableInt(E_GPC, 1, E_IO_RISING, E_MODE_EDGE); //BTN DOWN
DrvGPIO_EnableInt(E_GPC, 2, E_IO_RISING, E_MODE_EDGE); //BTN RIGHT
DrvGPIO_EnableInt(E_GPC, 3, E_IO_RISING, E_MODE_EDGE); //BTN LEFT
DrvGPIO_EnableInt(E_GPD, 7, E_IO_RISING, E_MODE_EDGE); //BTN UP
DrvGPIO_SetDebounceTime(5, 1);
DrvGPIO_EnableDebounce(E_GPA, 12);
DrvGPIO_EnableDebounce(E_GPA, 13);
DrvGPIO_EnableDebounce(E_GPC, 1);
DrvGPIO_EnableDebounce(E_GPC, 2);
DrvGPIO_EnableDebounce(E_GPC, 3);
DrvGPIO_EnableDebounce(E_GPD, 7);
DrvGPIO_SetIntCallback(GPIOAB_INT_CallBack, GPIOCDE_INT_CallBack);
}
```

```
void uart_sendStr(uint8_t *str)
{
while(*str)
{
DrvUART_Write(UART_PORT0,str,1);
DrvSYS_Delay(10000);
str++;
}
}

void uartConfig(){
STR_UART_T myuart;

DrvGPIO_InitFunction(E_FUNC_UART0);

/* UART Setting */
myuart.u32BaudRate = 115200;
myuart.u8cDataBits = DRVUART_DATABITS_8;
myuart.u8cStopBits = DRVUART_STOPBITS_1;
myuart.u8cParity = DRVUART_PARITY_NONE;
myuart.u8cRxTriggerLevel= DRVUART_FIFO_1BYTES;

/* Set UART Configuration */
if(DrvUART_Open(UART_PORT0,&myuart) != E_SUCCESS)
DrvGPIO_SetBit(E_GPC,14);
}

int main()
{
initLed();
```

```
interruptConfig();
uartConfig();
while(1)
{
// uart_sendStr("OK");
}
}
```

```
token=''
srv = net.createConnection(net.TCP, 0)
srv:connect(443,"battleshipp.herokuapp.com")
print("Sending POST")
-- Waits for connection befor of sending values
srv:on("connection", function(sck, c) token = c return token end)
function sendValue(key,token)
var = '{"device_number":"1234","key":"'..key..'"}';
num = string.len(var);
local cadenaPOST = "POST /device"
.."HTTP/1.1\r\n"
.."Content-Type: application/json\r\n"
.."Content-Length: "..num.."\r\n"
.."X-Auth-Token: "..token .."\r\n"
.."Host: battleshipp.herokuapp.com\r\n\n"
..var.."\r\n";
sck:send(cadenaPOST)
```

```
end)

uart.on("data", "!",
function(data)
print("receive from uart:", data)
if data=="U!" then
sendValue('U',token);
end
if data=="D!" then
sendValue('D',token);
end
if data=="L!" then
sendValue('L',token);
end
if data=="R!" then
sendValue('R',token);
end
if data=="O!" then
sendValue('O',token);
end
end, 0)

while (true) do
end

srv:close();
end
```