The Wolf Among Us API using Bluetooth Classic

(Seria A, Grupa 414)

Profesor coordonator:

Bogdan Cristian Florea

Studenti:

Radulescu Gabriel

Sapunareanu Alexandra-Bianca

Cuprins

1. Introducere

Arduino este o companie open-source care produce atât plăcuțe de dezvoltare bazate pe microcontrolere, cât și partea de software destinată funcționării și programării acestora. Pe lângă acestea include și o comunitate uriașă care se ocupă cu creația și distribuirea de proiecte care au ca scop crearea de dispozitive care pot sesiza și controla diverse activități sau procese în lumea reală.

În acest proiect, ne vom folosi de placuta ESP32 pentru a dezvolta o interfață de programare a aplicației care sa interactioneze prin intermediul modulelor bluetooth si wifi cu o aplicatie (ProiectIA) de pe telefon.

2. Considerente teoretice

Modulul ESP32 este un modul SoC (System on Chip) fabricat de compania Espressif Systems, bazat pe microprocesorul Tensilica Xtensa LX6 cu unul sau două nuclee și o frecvență de lucru de între 160 și 240MHz precum și un coprocesor ULP (Ultra Low Power). Suplimentar, acesta dispune de comunicație WiFi și Bluetooth (clasic și low-energy) integrate, precum și de o gamă largă de interfețe periferice:

- 34 pini programabili GPIO (General Purpose Input/Output)
- 18 canale de conversie analog-digitală (ADC) cu rezoluție de 12 biți
- 2 canale de conversie digital-analogică (DAC) cu rezoluție de 8 biți
- 16 canale de ieșire PWM (Pulse Width Modulation)
- 10 senzori interni capacitivi
- 3 interfețe SPI (Serial Peripheral Interface)
- 3 interfețe UART (Universal Asynchronous Receiver-Transmitter)

• 2 interfețe I2C (Inter-Integrated Circuit)
• 2 interfețe I2S (Inter-IC Sound)
• 1 interfață CAN 2.0 (Controller Area Network)
• controler pentru conectarea dispozitivelor de stocare (carduri de memorie)

Modulul ESP32 poate fi integrat în plăci de dezvoltare ce pot expune toți pinii/interfețele modulului sau doar o parte din ele. Cele mai des întâlnite tipuri de plăci de dezvoltare bazate pe modulul ESP32 sunt cele cu 30 sau 38 de pini. În Fig. 1 este prezentată diagrama unei plăci de dezvoltare cu 38 de pini, placă ce va fi folosită în cadrul acestui proiect.

Numerotarea pinilor de pe placa de dezvoltare este realizată în funcție de denumirea internă a pinilor modulului (GPIOxx). Placa dispune de un LED integrat care este conectat la pinul general de intrare/ieșire GPIO02, care poate fi accesat pe placă pe pinul G2 (Fig. 1).

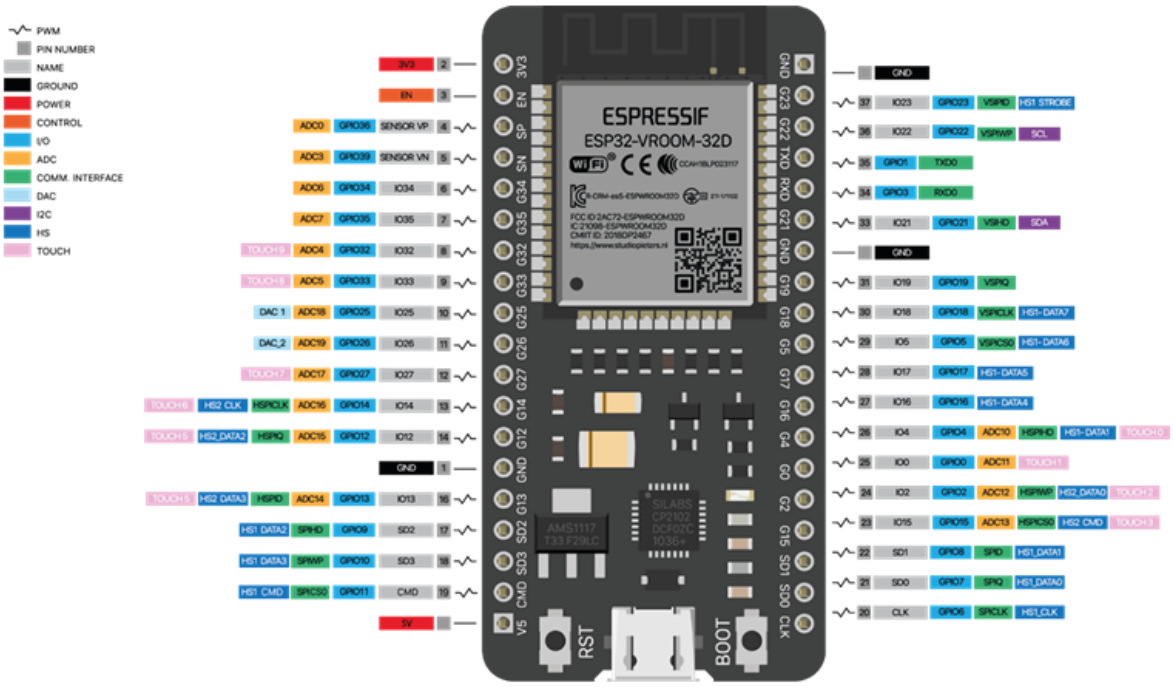


Fig. 1 - Diagrama pinilor plăcii de dezvoltare ESP32 – varianta cu 38 de pini [1]

Modulul ESP32 include elemente de conectivitate ce permit utilizarea acestuia în proiecte de tip IoT, automatizări de la distanță, etc. Acesta permite conectarea prin WiFi la o rețea locală sau internet, precum și conectivitatea Bluetooth (Classic sau Low Energy).

Bluetooth

Bluetooth este o tehnologie wireless cu rază scurtă de acțiune, folosită pentru transferul de date între diverse dispozitive. Raza de acțiune a transmisiei Bluetooth este în general între 10 - 400m, depinzând de variantă, puterea necesară, obstacole, etc. Tehnologia Bluetooth este folosită mai ales pentru conectarea dispozitivelor mobile (telefoane, laptopuri, tablete) cu diverse echipamente (căști audio, sisteme multimedia, televizoare, etc.) sau transfer de date.

Tehnologia Bluetooth poate fi împărțită în două categorii:

- Bluetooth Classic
- Bluetooth Low Energy

Ambele tehnologii folosesc protocolul Bluetooth dar sunt folosite cu scopuri diferite.

Bluetooth Classic (BTC) este implementarea originală a tehnologiei, introdusă în anul 1999. Această tehnologie a fost dezvoltată pentru aplicații de bandă largă, cum ar fi streaming audio/video sau transfer de fișiere, permițând dispozitive cu un consum mai mare de putere. Bluetooth Classic folosește banda de 2.4GHz și permite o rată maximă de transfer de 3Mbps.

Comunicația Bluetooth Classic este o comunicație serială, permițând transmisia bidirecțională a datelor.

Un dispozitiv Bluetooth poate avea rolul de client sau server. Serverul este dispozitivul care conține și transmite datele. Clientul este dispozitivul care se conectează la server pentru a accesa datele. Un dispozitiv Bluetooth poate fi client și server în același timp.

Pentru integrarea Bluetooth Classic, se va folosi librăria BluetoothSerial.h.

WiFi

WiFi reprezintă o familie de protocoale de rețea wireless bazate pe standardul IEEE 802.11, folosite pentru conectarea la rețele locale și internet. Modulul ESP32 suportă protocolul 820.11b/g/n și poate fi configurat ca stație (mod STA) sau access point (mod AP).

Modul STA su modul client permite conectarea modulului ESP32 la un alt access point/router. În modul AP, modulul ESP32 permite altor dispozitive să se conecteze cu acesta prin protocolul WiFi. De asemenea, permite conectarea la retele WiFi ce folosesc diverse metode de securitate (WPA2, WPA3, etc.). Metodele de securitate depind de configurarea rețelelor și nu fac obiectul acestei discipline.

Modulul ESP32 permite următoarele funcții specifice conexiunii WiFi:

- Setarea modului de funcționare (STA sau AP)
- Scanarea rețelelor disponibile
- Conectarea la o rețea WiFi
- Verificarea statusului conexiunii și a puterii semnalului
- Obținerea sau setarea adresei IP

• Deconectarea și reconectarea după pierderea conexiunii

• Definirea mai multor rețele și conectarea automată la rețeaua cu cel mai bun semnal

Funcțiile specifice funcționalităților WiFi sunt disponibile în librăria WiFi.h (WiFi API).

HTTP

Protocolul HTTP (HyperText Transfer Protocol) este protocolul de transfer al informației specific aplicațiilor web. El a fost creat pentru a facilita comunicarea între browsere web și servere web, însă poate fi folosit și pentru alte tipuri de aplicații.

Protocolul HTTP urmează modelul client-server, un client (browser, dispozitiv, etc.) inițiind o conexiune pentru a transmite o cerere (request) și asteptând până la primirea răspunsului (response). Protocolul HTTP este un protocol fără stare, ceea ce înseamnă că serverul web nu stochează în mod implicit nicio informație între două cereri distincte.

Protocolul HTTP definește un set de metode pentru cereri pentru a indica acțiunea ce va fi efectuată asupra unei anumite resurse. Aceste metode se mai numesc și verbe HTTP, fiecare metodă implementând semantică diferită. Principalele metode HTTP sunt:

• GET – Metoda GET reprezintă o cerere a unei reprezentări (imagini) a unei anumite resurse. Cererile transmise prin metoda GET ar trebui folosite pentru obținerea datelor de la un server (exemplu: o pagina web, o imagine, un fișier, etc.).

• POST – Metoda POST este in general folosită pentru a trimite o nouă înregistrare (entitate) către o resursă a serverului, de multe ori având ca efect modificare informației pe server sau alte efecte secundare, dacă aceste

acțiuni sunt implementate pe server (exemplu: datele unui nou utilizator, datele unui nou produs, etc.).

• PUT – metoda PUT a fost definită cu rolul de a înlocui reprezentarea curentă a unei resurse cu cea trimisă în cererea curentă (editare completă).

• PATCH – metoda PATCH a fost introdusă pentru a indica modificări parțiale ale unei resurse (editare parțială).

• DELETE – metoda DELETE re rolul de a șterge a anumită resursă.

Descrierea metodelor anterioare se referă la acțiunile sau intențiile definite de standard, însă aceste acțiuni trebuie implementate pe server pentru a reflecta rezultatul dorit. Pe lângă metodele principale definite mai sus, există și alte metode HTTP: HEAD, CONNECT, TRACE, OPTIONS.

JSON

Răspunsul returnat în urma accesării API-ului anterior este un text ce conține informații despre locație (latitudine și longitudine pentru București, precum și datele meteo curente). Aceste informații sunt reprezentate în formatul JSON (JavaScript Object Notation). Acest format reprezintă o modalitate facilă de transfer a datelor, utilizată în special atunci când datele trebuie să poată fi prelucrate indiferent de limbajul de programare folosit. Astfel, formatul JSON reprezintă datele sub forma de text și poate fi utilizat în majoritatea limbajelor de programare, datele reprezentate putând fi convertite în structuri specifice limbajului de programare utilizat.

Formatul JSON poate fi folosit pentru a reprezenta valori simple (de exemplu un număr sau un text), însă de cele mai multe ori este folosit pentru a reprezenta structuri de date mai complexe, precum vectori sau obiecte. Datele numerice sau booleene nu sunt delimitate, iar datele de tip text sunt delimitate de ghilimele (nu apostroafe). Spațiile albe nu sunt luate în

considerare, astfel încât textul JSON rezultat poate fi formatat pentru o vizualizare mai bună.

Vectori JSON

Vectorii în formatul JSON sunt delimitați de simbolurile [ ], iar elementele acestora sunt separate prin virgulă. Formatul JSON nu impune un tip de date al elementelor vectorilor, acesta rezultând din modul în care textul JSON a fost generat și limbajul de programare utilizat pentru creare (serializare) și conversie (deserializare).

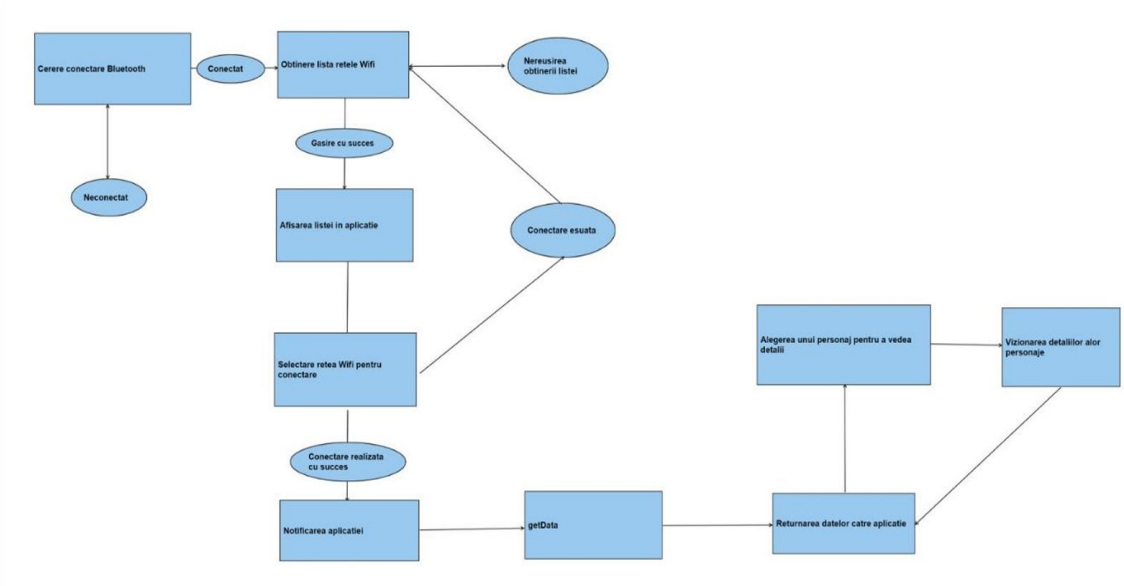
Obiecte JSON

Obiectele în formatul JSON sunt delimitate de simbolurile { }, iar elementele lor sunt perechi de tipul cheie/valoare. Cheile sunt în mod obligatoriu de tip text, delimitate cu ghilimele. Ca și în cazul vectorilor, tipul de date al valorilor nu este impus, fiecare valoare putând avea un alt tip de date (similar cu proprietățile claselor în limbajul C++).

Pentru a utiliza formatul JSON în programele ESP32, există mai multe librării disponibile, însă cea mai folosită librărie este ArduinoJSON. Documentația librăriei precum și numeroase exemple de serializare/deserializare și un utilitar de calcul al memoriei necesare sunt disponibile pe site-ul oficial al librăriei.

3. Implementare

- Organigrama codului si explicatii



```
// Define the BluetoothSerial
BluetoothSerial SerialBT;
// Define a variable for the connection status
bool connected = false;
// Callback function for connection events
void deviceConnected(
    esp_spp_cb_event_t event,
    esp_spp_cb_param_t *param)
{
    if (event == ESP_SPP_SRV_OPEN_EVT)
    {
        Serial.println("Device Connected");
        connected = true;
    }
    if (event == ESP_SPP_CLOSE_EVT)
    {
        Serial.println("Device disconnected");
        connected = false;
    }
}
#include <WiFi.h>
#include <HTTPClient.h>
const long connection_timeout = 15000; // 15s
long startConnection = 0;
String Id; // <-- id echipa
void setup()
{
    Serial.begin(115200);
    int time = 0;
    SerialBT.begin(btcServerName);
    SerialBT.register_callback(deviceConnected);
}
```

- Aceasta secventa de cod, asigura conectarea la bluetooth.

```
void loop()
{
    if (SerialBT.available())
    {
        Serial.println("<---------------------------------------->");
        String DATA = SerialBT.readString();
        String action;
        Serial.print("Comanda:");
        StaticJsonDocument<1024> doc;
        const char *json = DATA.c_str();
        DeserializationError error = deserializeJson(doc, json);
        if (error)
        {
            Serial.print(F("deserializeJson() failed: "));
            Serial.println(error.f_str());
        }
        else
        {
            action = doc["action"].as<String>();
            Serial.println(action);
            Serial.println(Id);
```

- Se citeste input ul primit prin intermediul bluetooth, se deserializeaza, variabila „action” reprezentand instructiunea care trebuie urmata.

```
if (action == "getNetworks")
{
    Id = doc["teamId"].as<String>();
    int networksFound = WiFi.scanNetworks();
    if (networksFound == 0)
    {
        Serial.println("No networks found");
    }
    else
    {
        Serial.print(networksFound);
        Serial.println(" networks found");
        for (int i = 0; i < networksFound; i++)
        {
            bool open = (WiFi.encryptionType(i) == WIFI_AUTH_OPEN);
            const size_t CAPACITY = JSON_OBJECT_SIZE(4) + JSON_ARRAY_SIZE(4);
            DynamicJsonDocument doc_retele(CAPACITY);
            doc_retele["ssid"] = WiFi.SSID(i);
            doc_retele["strength"] = WiFi.RSSI(i);
            if ((open) == 1)
                doc_retele["encryption"] = "Open";
            else
                doc_retele["encryption"] = "Protected";
            doc_retele["teamId"] = Id;
            String output;
            serializeJson(doc_retele, output);
            Serial.println(output);
            SerialBT.println(output);
        }
    }
}
```

- In aceasta situatie, „action” dicteaza programului cautarea retelelor de WiFi din apropiere si trimiterea lor inapoi aplicatiei prin intermediul incapsularii in format JSON.

```
if (action == "connect")
{
    String nume = doc["ssid"].as<String>();
    String parola = doc["password"].as<String>();
    Serial.println(nume);
    Serial.println(parola);
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);
    // conectare la wifi
    WiFi.begin(nume.c_str(), parola.c_str());
    Serial.println("Connecting");
    startConnection = millis();
    bool open = (WiFi.status() != WL_CONNECTED);
    const size_t CAPACITY = JSON_OBJECT_SIZE(3) + JSON_ARRAY_SIZE(3);
    DynamicJsonDocument doc_net_conectare(CAPACITY);
    doc_net_conectare["ssid"] = nume;
    doc_net_conectare["connected"] = open;
    doc_net_conectare["teamId"] = Id;
    String output;
    serializeJson(doc_net_conectare, output);
    Serial.println(output);
    SerialBT.println(output);
}
```

- In aceasta secventa de cod, variabila „action” impune programului sa se conecteze la una dintre retelele WiFi aleasa din aplicatie, fiind trimise din aplicatie atat SSID ul, cat si parola pentru conectare.

```
if (action == "getDetails")
{
    String id_detalii = doc["id"].as<String>();
    String details = "";
    String Site_detalii = "http://proiectia.bogdanflorea.ro/api/the-wolf-among-us/character?id=" + id_detalii;
    Serial.println(Site_detalii);
    HTTPClient http;
    http.begin(Site_detalii);
    http.setConnectTimeout(20000);
    http.setTimeout(20000);
    int httpResponseCode = http.GET();
```

- In aceasta secventa de cod, se acceseaza site ul „http://proiectia.bogdanflorea.ro/api/the-wolf-among-

us/character?id=" +id_detalii, in vederea extragerii prin metoda GET a informatiilor in format JSON, ca mai apoi sa fie deserializate si trimise, apoi, in aplicatie serializate individual in functie de ce se cere.

```
if (httpResponseCode > 0)
{   Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    // Get response data
    String payload = http.getString();
    Serial.println(payload);
    //http.end();
    StaticJsonDocument<1024> getDetails;
    const char *json = payload.c_str();
    DeserializationError error = deserializeJson(getDetails, payload);
    if (error)
    {
        Serial.println(error.c_str());
    }
    else
    {
        DynamicJsonDocument detalii(10000);
        detalii["teamId"] = Id;
        detalii["id"] = id_detalii;
        detalii["name"] = getDetails["name"].as<String>();
        detalii["image"] = getDetails["imagePath"].as<String>();
        details = details + "name: " + getDetails["name"].as<String>() + "\n";
        details +="species: " + getDetails["species"].as<String>() + "\n";
        details += "gender: " + getDetails["gender"].as<String>() + "\n";
        details += "occupation: " + getDetails["occupation"].as<String>() + "\n";
        details += "hairColour: " + getDetails["hairColour"].as<String>() + "\n";
        details += "eyeColour: " + getDetails["eyeColour"].as<String>() + "\n";
        details += "description: " + getDetails["description"].as<String>();
        detalii["description"] = details;
        String output;
        serializeJson(detalii, output);
        Serial.println(output);
        SerialBT.println(output);
```

- Aceasta ultima bucata din program, serializeaza informatiile despre caracterul ales in aplicatie ca date membre dintr-un obiect de tip JSON, ca mai apoi, sa fie trimise in aplicatie si vizualizate acolo.

4. Concluzie

Acest proiect a reusit sa combine implemetarea comunicarii eficiente intre o aplicatie mobila si un microprocesor, aducand in prim-plan noi si inovatoare concepte, cum ar fi preluarea si decodarea informatiilro folosind formatul JSON. Aceasta initiavita, ne-a provocat creativitatea prin cercetarea aprofundata in domeniu, si ne-a testat abilitatea de a lucra intr-o echipa unita, aducand laolalta competentele individuale si punandu-ne la incercare capacitatea de colaborare si sincronizare.

5. Bibliografie

P1. Modulul ESP32 și resursele sale.pdf

P2. Tipuri de semnale, protocoale de comunicatie si periferice.pdf

P3. WiFi, HTTP, JSON si Bluetooth.pdf

https://ro.wikipedia.org/wiki/Arduino

6. Codul complet

```
#include <Arduino.h>
#include <ArduinoJson.h>
#include <BluetoothSerial.h>
#include <string>
#include <WiFi.h>
#include <HTTPClient.h>
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled!
#endif
#define btcServerName "A50 BLE"
// Define the BluetoothSerial
BluetoothSerial SerialBT;
// Define a variable for the connection status
bool connected = false;
// Callback function for connection events
void deviceConnected(
    esp_spp_cb_event_t event,
    esp_spp_cb_param_t *param)
{
    if (event == ESP_SPP_SRV_OPEN_EVT)
    {
        Serial.println("Device Connected");
        connected = true;
    }
    if (event == ESP_SPP_CLOSE_EVT)
    {
        Serial.println("Device disconnected");
        connected = false;
    }
}
#include <WiFi.h>
#include <HTTPClient.h>
const long connection_timeout = 15000; // 15s
long startConnection = 0;
String Id; // <-- id echipa
```

```
void setup()
{
    Serial.begin(115200);
    int time = 0;
    SerialBT.begin(btcServerName);
    SerialBT.register_callback(deviceConnected);
}
void loop()
{
    if (SerialBT.available())
    {
        Serial.println("<-------------------------------------------->");
        String DATA = SerialBT.readString();
        String action;
        Serial.print("Comanda:");
        StaticJsonDocument<1024> doc;
        const char *json = DATA.c_str();
        DeserializationError error = deserializeJson(doc, json);
        if (error)
        {
            Serial.print(F("deserializeJson() failed: "));
            Serial.println(error.f_str());
        }
        else
        {
            action = doc["action"].as<String>();
            Serial.println(action);
            Serial.println(Id);
            if (action == "getNetworks")
            {
                Id = doc["teamId"].as<String>();
                int networksFound = WiFi.scanNetworks();
                if (networksFound == 0)
                {
                    Serial.println("No networks found");
                }
```

```
    else
    {
        Serial.print(networksFound);
        Serial.println(" networks found");
        for (int i = 0; i < networksFound; i++)
        {
            bool open = (WiFi.encryptionType(i) == WIFI_AUTH_OPEN);
            const size_t CAPACITY = JSON_OBJECT_SIZE(4) + JSON_ARRAY_SIZE(4);
            DynamicJsonDocument doc_retele(CAPACITY);
            doc_retele["ssid"] = WiFi.SSID(i);
            doc_retele["strength"] = WiFi.RSSI(i);
            if ((open) == 1)
                doc_retele["encryption"] = "Open";
            else
                doc_retele["encryption"] = "Protected";
            doc_retele["teamId"] = Id;
            String output;
            serializeJson(doc_retele, output);
            Serial.println(output);
            SerialBT.println(output);
        }
    }
}
```

```
if (action == "connect")
{
    String nume = doc["ssid"].as<String>();
    String parola = doc["password"].as<String>();
    Serial.println(nume);
    Serial.println(parola);
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);
    // conectare la wifi
    WiFi.begin(nume.c_str(), parola.c_str());
    Serial.println("Connecting");
    startConnection = millis();
    bool open = (WiFi.status() != WL_CONNECTED);
    const size_t CAPACITY = JSON_OBJECT_SIZE(3) + JSON_ARRAY_SIZE(3);
    DynamicJsonDocument doc_net_conectare(CAPACITY);
    doc_net_conectare["ssid"] = nume;
    doc_net_conectare["connected"] = open;
    doc_net_conectare["teamId"] = Id;
    String output;
    serializeJson(doc_net_conectare, output);
    Serial.println(output);
    SerialBT.println(output);
}
if (action == "getData")
{
    HTTPClient http;
    String site = "http://proiectia.bogdanflorea.ro/api/the-wolf-among-us/characters";
    delay(2000);
    http.begin(site);
    delay(2000);
    http.setConnectTimeout(20000);
    http.setTimeout(20000);
    int httpResponseCode = http.GET();
```

```
    if (httpResponseCode > 0)
    {
        Serial.print("HTTP Response code: ");
        Serial.println(httpResponseCode);
        String payload = http.getString();
        StaticJsonDocument<1024> getData;
        http.end();
        // deserializare in jsonuri mici
        DynamicJsonDocument minijson(15000);
        DeserializationError error = deserializeJson(minijson, payload);
        if (error)
        {
            Serial.println(error.c_str());
        }
        else
        {
            JsonArray list = minijson.as<JsonArray>();
            int index = 1;
            for (JsonVariant value : list)
            {
                JsonObject listItem = value.as<JsonObject>();
                const size_t CAPACITY = JSON_OBJECT_SIZE(4) + JSON_ARRAY_SIZE(12);
                DynamicJsonDocument doc_getData(CAPACITY);
                JsonObject object = doc_getData.to<JsonObject>();
                object.set(listItem);
                String output;
                doc_getData["id"] = listItem["id"].as<String>();
                doc_getData["name"] = listItem["name"].as<String>();
                doc_getData["image"] = listItem["imagePath"].as<String>();
                doc_getData["teamId"] = Id;
                serializeJson(doc_getData, output);
                SerialBT.println(output);
                Serial.println(output);
                index++;
            }
        }
    }
}
```

```
if (action == "getDetails")
{
    String id_detalii = doc["id"].as<String>();
    String details = "";
    String Site_detalii = "http://proiectia.bogdanflorea.ro/api/the-wolf-among-us/character?id=" + id_detalii;
    Serial.println(Site_detalii);
    HTTPClient http;
    http.begin(Site_detalii);
    http.setConnectTimeout(20000);
    http.setTimeout(20000);
    int httpResponseCode = http.GET();
    if (httpResponseCode > 0)
    {
        Serial.print("HTTP Response code: ");
        Serial.println(httpResponseCode);
        // Get response data
        String payload = http.getString();
        Serial.println(payload);
        //http.end();
        StaticJsonDocument<1024> getDetails;
        const char *json = payload.c_str();
        DeserializationError error = deserializeJson(getDetails, payload);
        if (error)
        {
            Serial.println(error.c_str());
        }
        else
        {
            DynamicJsonDocument detalii(10000);
            detalii["teamId"] = Id;
            detalii["id"] = id_detalii;
            detalii["name"] = getDetails["name"].as<String>();
            detalii["image"] = getDetails["imagePath"].as<String>();

            details = details + "name: " + getDetails["name"].as<String>() + "\n";
            details +="species: " + getDetails["species"].as<String>() + "\n";
            details += "gender: " + getDetails["gender"].as<String>() + "\n";
            details += "occupation: " + getDetails["occupation"].as<String>() + "\n";
            details += "hairColour: " + getDetails["hairColour"].as<String>() + "\n";
            details += "eyeColour: " + getDetails["eyeColour"].as<String>() + "\n";
            details += "description: " + getDetails["description"].as<String>();

            detalii["description"] = details;

            String output;
            serializeJson(detalii, output);
            Serial.println(output);
            SerialBT.println(output);
        }
    }
}
```