**PROIECT INFORMATICĂ APLICATĂ**

**Participanți:** Nica Mihai, Munteanu David-Andrei.

**Grupa**: 414A.

**Seria**: A.

**ID Echipă**: A48

**Introducere**

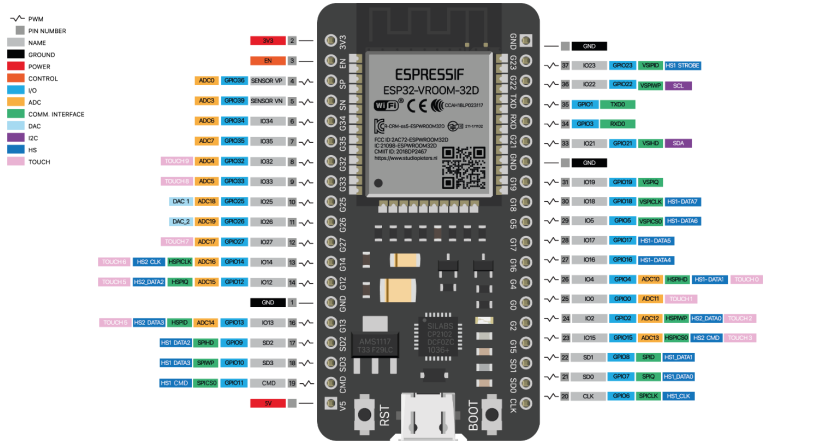
Pe parcursul realizării acestui proiect, am urmărit dezvoltarea unei aplicații interactive bazate pe ESP32, care să permită comunicarea prin Bluetooth și accesul la internet prin WiFi. Am implementat funcționalități precum scanarea rețelelor, conectarea la o rețea și preluarea de date de la un API extern. Prin acest proiect, ne-am familiarizat cu utilizarea ESP32 și am înțeles modul în care poate fi integrat în aplicații IoT.

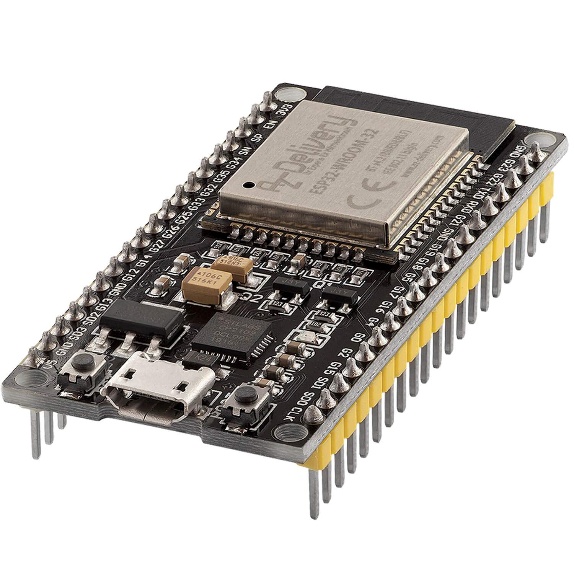
**Considerații teoretice**

**Modulul ESP32**

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.

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.

****



**Protocoale și metode de comunicație utilizate:**

**Bluetooth Classic**

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 fișieire, 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.

**Wi-Fi**

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 sau 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.).

În acest, funcțiile utilizate de către ESP32 sunt: setarea  modulului de funcționare(STA), scanarea rețelelor disponibile pentru conectare, conectarea la o rețea.

Pentru a utiliza funcțiile pentru WiFi este necesară librăria WiFi.h.

**JSON**

Formatul JSON reprezintă o modalitate de a transmite/primi mai ușor un set de date, astfel acest format reprezintă datele sub formă de text. El este folosit pentru reprezentarea unor valori simple, sau mai complexe, în vectori sau obiecte.

În acest cod, formatul JSON este folosit pentru 4 acțiuni:

* getNetworks- în acest caz plăcuța primește un obiect JSON, cu două date, action și teamId, în urma căreia plăcuța scanează toate rețelele de WiFi disponibile, și transmite către API un JSON obiect, în care sunt salvate datele pentru fiecare rețea: ssid, strength, encryption și teamId;
* connect- modulul primește un obiect JSON, în care sunt salvate ssid și parola pentru o anumită rețea, la care este posibilă conectarea la WiFi, se realizează conectarea, și transmite către API, ssid rețelei la care ne-am conectat, o variabilă connected care poate fi true sau false, care reprezintă starea de conectare sau nu a plăcuței la rețeaua dorită, și teamId;
* getData- plăcuța primește un JSON document, în care se păstrează acțiunea dorită, care face ca să se realizeze conectarea cu http-ul dorit, introdus deja în cod, de unde primește un JSON vector de obiecte, care sunt transmise către API;
* getDetails- modulul primește JSON obiectul cu id-ul produsului dorit prin intermediul unui http care este personal pentru fiecare produs în funcție de id, și transmite înapoi id-ul produsului, numele acestui caracterxx, imagine lui și descrierea produsului, unde descrierea este construită din datele primite nefolosite;

Pentru folosirea formatului JSON este necesară librăria ArduinoJson.h .

**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.

Protocolul HTTP folosește modelul client-server. În intermediul acestui protocol sunt definite un set de metode pentru cereri pentru a indica ce acțiune este dorită. Metoda folosită în cadrul acestui proiect este GET.

**Implementarea**

Conectarea la Bluetooth

Scanarea rețelelor Wi-Fi si transmiterea datelor rețelei dorite prin JSON

Conectarea la rețeaua Dorita prin intermediul parolei si id-ul rețelei

Primirea datelor din URL-ul listei de caractere prin intermediul protocolului HTTP si transmiterea aplicației unui vector de date JSON

Primirea detaliilor fiecarui caracter ales prin intermediul protocolului HTTP si trimiterea catre aplicatie caracterisiticile fiecarui personaj din lista

**Setup**

**ANEXE:**

#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 "A48 Serial"

BluetoothSerial SerialBT;

bool connected = false;

const long connection\_timeout = 10000;

long startConnection = 0;

String Id;

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;

}

}

void handleGetNetworks(StaticJsonDocument<1024>& pia) {

Id = pia["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 cap = JSON\_OBJECT\_SIZE(4) + JSON\_ARRAY\_SIZE(4);

DynamicJsonDocument pia\_retea(cap);

pia\_retea["ssid"] = WiFi.SSID(i);

pia\_retea["strength"] = WiFi.RSSI(i);

pia\_retea["encryption"] = open ? "Open" : "Protected";

pia\_retea["teamId"] = Id;

String output;

serializeJson(pia\_retea, output);

Serial.println(output);

SerialBT.println(output);

}

}

}

void handleConnect(StaticJsonDocument<1024>& pia) {

String nume = pia["ssid"].as<String>();

String parola = pia["password"].as<String>();

Serial.println(nume);

Serial.println(parola);

WiFi.mode(WIFI\_STA);

WiFi.disconnect();

delay(100);

WiFi.begin(nume.c\_str(), parola.c\_str());

Serial.println("Connecting");

startConnection = millis();

bool open = (WiFi.status() != WL\_CONNECTED);

const size\_t cap = JSON\_OBJECT\_SIZE(3) + JSON\_ARRAY\_SIZE(3);

DynamicJsonDocument pia\_conectare(cap);

pia\_conectare["ssid"] = nume;

pia\_conectare["connected"] = open;

pia\_conectare["teamId"] = Id;

String output;

serializeJson(pia\_conectare, output);

Serial.println(output);

SerialBT.println(output);

}

void handleGetData() {

HTTPClient http;

String site = "http://proiectia.bogdanflorea.ro/api/futurama/characters";

delay(3000);

http.begin(site);

delay(3000);

http.setConnectTimeout(10000);

http.setTimeout(10000);

int httpResponseCode = http.GET();

if (httpResponseCode > 0) {

Serial.print("HTTP Response code: ");

Serial.println(httpResponseCode);

String payload = http.getString();

http.end();

DynamicJsonDocument json\_mic(15000);

DeserializationError error = deserializeJson(json\_mic, payload);

if (error) {

Serial.println(error.c\_str());

} else {

JsonArray list = json\_mic.as<JsonArray>();

int index = 1;

for (JsonVariant value : list) {

JsonObject listItem = value.as<JsonObject>();

const size\_t cap = JSON\_OBJECT\_SIZE(4) + JSON\_ARRAY\_SIZE(20);

DynamicJsonDocument pia\_getData(cap);

JsonObject object = pia\_getData.to<JsonObject>();

object.set(listItem);

String output;

pia\_getData["id"] = listItem["Id"].as<String>();

pia\_getData["name"] = listItem["Name"].as<String>();

pia\_getData["image"] = listItem["PicUrl"].as<String>();

pia\_getData["teamId"] = Id;

serializeJson(pia\_getData, output);

SerialBT.println(output);

Serial.println(output);

index++;

}

}

}

}

void handleGetDetails(StaticJsonDocument<1024>& pia) {

String id\_char = pia["id"].as<String>();

String Site\_details = "http://proiectia.bogdanflorea.ro/api/futurama/character?Id=" + id\_char;

Serial.println(Site\_details);

HTTPClient http;

http.begin(Site\_details);

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();

Serial.println(payload);

StaticJsonDocument<1024> getDetails;

DeserializationError error = deserializeJson(getDetails, payload);

if (error) {

Serial.println(error.c\_str());

} else {

DynamicJsonDocument details(4096);

details["id"] = id\_char;

details["name"] = getDetails["Name"].as<String>();

details["image"] = getDetails["PicUrl"].as<String>();

String descriere = "Species: " + getDetails["Species"].as<String>() + "\n" +

"Age: " + getDetails["Age"].as<String>() + "\n" +

"Profesion: " + getDetails["Profession"].as<String>() + "\n" +

"Status: " + getDetails["Status"].as<String>() + "\n" +

"FirstAppearance: " + getDetails["FirstAppearance"].as<String>() + "\n" +

"Relatives: " + getDetails["Relatives"].as<String>() + "\n" +

"VoicedBy: " + getDetails["VoicedBy"].as<String>();

Serial.println(descriere);

details["description"] = descriere;

details["teamId"] = Id;

String output;

serializeJson(details, output);

Serial.println(output);

SerialBT.println(output);

}

}

}

void setup() {

Serial.begin(115200);

SerialBT.begin(btcServerName);

SerialBT.register\_callback(deviceConnected);

}

void loop() {

if (SerialBT.available()) {

Serial.println("---Nica Mihai---David Munteanu---");

String DATA = SerialBT.readString();

Serial.print("Comm:");

StaticJsonDocument<1024> pia;

DeserializationError error = deserializeJson(pia, DATA);

if (error) {

Serial.print(F("deserializeJson() failed: "));

Serial.println(error.f\_str());

} else {

String action = pia["action"].as<String>();

Serial.println(action);

Serial.println(Id);

if (action == "getNetworks") {

handleGetNetworks(pia);

} else if (action == "connect") {

handleConnect(pia);

} else if (action == "getData") {

handleGetData();

} else if (action == "getDetails") {

handleGetDetails(pia);

}

}

}

}