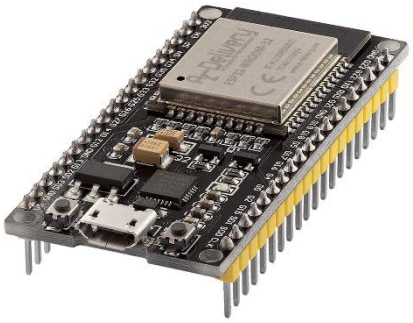
Proiect Informatică Aplicată

Student: Grigore George-Alexandru
Grupa: 414A
Nume proiect: Superhero API using Bluetooth Low Energy
Team ID: A45

Cuprins

Componente

-ESP32 : este un modul System on Chip, bazat pe microprocesorul Tensilica Xtensa LX6, cu unul sau două nuclee care au o frcvență de lucru între 160 și 240 MHz. Acest modul dispune și de posibilitatea de a transmite date prin Wi-Fi sau Bluetooth (Classic sau Low Energy).



Protocoale și metode de comunicație folosite

1. *Bluetooth Low Energy (BLE)*
   Este o tehnologie wireless pentru comunicații pe distanțe scurte, care consumă foarte puțină energie. Este folosită pentru a conecta dispozitive mici, cum ar fi senzori, ceasuri inteligente sau telefoane, și permite transmiterea rapidă și eficientă de date între ele. BLE organizează datele în servicii și caracteristici, fiecare identificată prin UUID-uri unice.
   Pentru a utiliza funcțiile pentru BLE am folosit librăriile: BLEDevice.h, BLEServer.h, BLEUtils.h și BLE2902.h

2. *Wi-Fi*
   Este o tehnologie care permite dispozitivelor să se conecteze la internet sau între ele fără fir, folosind unde radio. Este rapidă și folosită pentru acces la internet și transfer de date pe distanțe medii. Wi-Fi folosește rețele cu nume (SSID) și, de obicei, necesită o parolă pentru securitate. Modulul ESP32 poate fi configurat ca stație (mod STA) sau access point (mod AP). Am

folosit modul STA sau modul client care permite conectarea modulului ESP32 la un alt access point/router.
Pentru a utiliza funcțiile pentru WiFi am folosit librăria WiFi.h

3. *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.
   Pentru folosirea formatului JSON este necesară librăria ArduinoJson.h

4. *HTTP*
   Este protocolul de transfer al informației specific aplicațiilor web. El a fost creat pentru a facilita comunicarea între browsere web și servere web.

   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.
   Pentru a utiliza funcțiile pentru HTTP am folosit librăria HTTPClient.h

Descriere

Acest proiect folosește un ESP32 care acționează ca server BLE și comunică cu o aplicație mobilă numită ***ProiectIA***. Interacțiunea se face printr-o caracteristică BLE definită în UUID-ul **254bdafe-f9a9-11ed-be56-0242ac120002**.

Acțiuni:

1. **Scanare WiFi (action: "getNetworks"):**

La cererea aplicației, ESP32 scanează rețelele WiFi din jur și trimite, prin BLE, pentru fiecare rețea:

-SSID ("ssid")
-Putere semnal ("strength")
-Tip criptare ("encryption")
-ID-ul echipei ("teamId": "A45")

2. **Conectare WiFi (action: "connect"):**

Aplicația trimite numele rețelei și parola. ESP32 încearcă să se conecteze și răspunde cu:

-SSID-ul rețelei

-Status conexiune ("connected": true/false)

-ID echipă

3. **Obținere listă eroi (action: "getData"):**

După conectare ESP32 accesează API-ul:
`http://proiectia.bogdanflorea.ro/api/superhero-api/characters`

și trimite fiecare personaj în parte prin BLE, incluzând:

-ID

-Nume

-URL imagine

4. **Detalii despre erou (action: "getDetails"):**

Când utilizatorul selectează un erou, ESP32 face o cerere către:

`http://proiectia.bogdanflorea.ro/api/superhero-api/character?id=<id>`

<id>=ID-ul eroului selectat

și trimite înapoi date structurate precum:

-powerstats (intelligence, strength, speed, durability, power, combat)

-biography (real name, alter-egos, aliases, place-of-birth, first-appearance, publisher, alignment)

-appearance (gender, race, hair-color, eye-color, height, weight)

-work (occupation, base)

-connections (group-affiliation, relatives)

-image (URL imagine)

Cod

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <ArduinoJson.h>
#include "WiFi.h"
#include <HTTPClient.h>

#define bleServerName "A45"

bool deviceConnected = false;

#define SERVICE_UUID "1cce0140-f9a9-11ed-be56-0242ac120002"
#define CHARACTERISTIC_UUID "254bdafe-f9a9-11ed-be56-0242ac120002"


BLECharacteristic characteristic(
  CHARACTERISTIC_UUID,
  BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE |
BLECharacteristic::PROPERTY_NOTIFY
);
BLEDescriptor *characteristicDescriptor = new
BLEDescriptor(BLEUUID((uint16_t)0x2902));



class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
    Serial.println("Device connected");
  }
  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
    Serial.println("Device disconnected");
  }
};

class CharacteristicsCallbacks: public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *characteristic) {
    String data = characteristic->getValue();
    DynamicJsonDocument jsonDoc(1024); // Adjust buffer size based on JSON data
size
```

```
DeserializationError error = deserializeJson(jsonDoc, data.c_str());
if (!error) {
  JsonObject data = jsonDoc.as<JsonObject>();
  String action = data["action"].as<String>();

  if (action == "getNetworks") {
    String teamId = data["teamId"].as<String>();
    WiFi.mode(WIFI_STA);
    int n = WiFi.scanNetworks();
    for (int i = 0; i < n; ++i) {
         DynamicJsonDocument productDoc(15000);
         productDoc["ssid"]=WiFi.SSID(i);
         productDoc["strength"]=WiFi.RSSI(i);
         productDoc["encryption"]=WiFi.encryptionType(i);
         productDoc["teamId"]=teamId;
         String output;
         serializeJson(productDoc, output);
         characteristic->setValue(output);
         characteristic->notify();
         delay(200);
    }
  } else if (action == "connect") {
    String ssid = data["ssid"].as<String>();
    String password = data["password"].as<String>();
    Serial.println(ssid);
    Serial.println(password);
    WiFi.begin(ssid.c_str(),password.c_str());
      while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
        if (WiFi.status() == WL_CONNECTED) {
          break;
        }
      }
          DynamicJsonDocument productDoc(15000);
          productDoc["ssid"]=ssid;
          if(WiFi.status() != WL_CONNECTED){
            productDoc["connected"]=(WiFi.status() != WL_CONNECTED);
          }
          else{
            productDoc["connected"]=(WiFi.status() == WL_CONNECTED);
          }
          productDoc["teamId"]="A45";
          String output;
```

```
            serializeJson(productDoc, output);
            characteristic->setValue(output);
            characteristic->notify();
      } else if (action == "getData") {
          HTTPClient http;
          http.begin("http://proiectia.bogdanflorea.ro/api/superhero-
api/characters");
          int httpCode=http.GET();
          if(httpCode>0){
            String superhero = http.getString();
            DynamicJsonDocument JSONDocument(15000);
            DeserializationError error = deserializeJson(JSONDocument,
superhero.c_str());
            if (error) {
            Serial.println(error.c_str());
            } else {
              JsonArray list = JSONDocument.as<JsonArray>();
              for (JsonVariant value : list){
                JsonObject listItem = value.as<JsonObject>();
                DynamicJsonDocument outputDocument(15000);
                JsonObject object = outputDocument.to<JsonObject>();
                outputDocument["id"]=listItem["id"];
                outputDocument["name"]=listItem["name"];
                outputDocument["image"]=listItem["imageUrl"];
                outputDocument["teamId"]="A45";
                String output;
                serializeJson(outputDocument, output);
                characteristic->setValue(output);
                characteristic->notify();
              }
            }
          }
          http.end();
      } else if (action == "getDetails") {
        String id = data["id"].as<String>();
        HTTPClient http;
        http.begin("http://proiectia.bogdanflorea.ro/api/superhero-
api/character?id=" + id);
        int httpCODE = http.GET();
        if (httpCODE > 0) {
          String superhero= http.getString();
          DynamicJsonDocument JSONDocument(30000);
          DeserializationError error = deserializeJson(JSONDocument,
superhero.c_str());
          DynamicJsonDocument productDoc(30000);
```

```
          String name=JSONDocument["name"].as<String>();
          String
intelligence=JSONDocument["powerstats"]["intelligence"].as<String>();
          String strength=JSONDocument["powerstats"]["strength"].as<String>();
          String speed=JSONDocument["powerstats"]["speed"].as<String>();
          String
durability=JSONDocument["powerstats"]["durability"].as<String>();
          String power=JSONDocument["powerstats"]["power"].as<String>();
          String combat=JSONDocument["powerstats"]["combat"].as<String>();
          String powerstats=("intelligence: "+intelligence+"\n"+"strength:
"+strength+"\n"+"speed: "+speed+"\n"+"durability: "+durability+"\n"+"power:
"+power+"\n"+"combat: "+combat);
          String full_name=JSONDocument["biography"]["full-name"].as<String>();
          String alter_egos=JSONDocument["biography"]["alter-egos"].as<String>();
          JsonArray aliasesArray =
JSONDocument["biography"]["aliases"].as<JsonArray>();
          String aliases ="";
          for (int i = 0; i < aliasesArray.size(); i++) {
            if(i==aliasesArray.size()-1 && i!=0) aliases+="\n";
            else if (i > 0) aliases += ",\n";
            aliases += aliasesArray[i].as<String>();
          }
          String place_of_birth=JSONDocument["place-of-birth"].as<String>();
          String first_appearance=JSONDocument["biography"]["first-
appearance"].as<String>();
          String publisher=JSONDocument["biography"]["publisher"].as<String>();
          String alignment=JSONDocument["biography"]["alignment"].as<String>();
          String gender=JSONDocument["appearance"]["gender"].as<String>();
          String race=JSONDocument["appearance"]["race"].as<String>();
          String eye_color=JSONDocument["appearance"]["eye-color"].as<String>();
          String hair_color=JSONDocument["appearance"]["hair-
color"].as<String>();
          JsonArray heightArray =
JSONDocument["appearance"]["height"].as<JsonArray>();
          String height = heightArray[0].as<String>() + ",\n" +
heightArray[1].as<String>();
          JsonArray weightArray =
JSONDocument["appearance"]["weight"].as<JsonArray>();
          String weight = weightArray[0].as<String>() + ",\n" +
weightArray[1].as<String>();
          String occupation=JSONDocument["work"]["occupation"].as<String>();
          String base=JSONDocument["work"]["base"].as<String>();
          String work=("occupation: "+occupation+"\n"+"base: "+base);
          String group_affiliation=JSONDocument["connections"]["group-
affiliation"].as<String>();
```

```
          String relatives=JSONDocument["connections"]["relatives"].as<String>();
          String connections=("group-affiliation:
"+group_affiliation+"\n"+"relatives: "+relatives);
          productDoc["id"]=id;
          productDoc["name"]=name;
          productDoc["powerstats"]=powerstats;
          productDoc["biography"]["full-name"]=full_name;
          productDoc["biography"]["alter-egos"]=alter_egos;
          productDoc["biography"]["aliases"]=aliases;
          productDoc["biography"]["place-of-birth"]=place_of_birth;
          productDoc["biography"]["first-appearance"]=first_appearance;
          productDoc["biography"]["publisher"]=publisher;
          productDoc["biography"]["alignment"]=alignment;
          productDoc["appearance"]["gender"]=gender;
          productDoc["appearance"]["race"]=race;
          productDoc["appearance"]["height"]=height;
          productDoc["appearance"]["weight"]=weight;
          productDoc["appearance"]["eye-color"]=eye_color;
          productDoc["appearance"]["hair-color"]=hair_color;
          productDoc["work"]=work;
          productDoc["connections"]=connections;
          productDoc["image"]=JSONDocument["imageUrl"];
          String output;
          serializeJson(productDoc, output);
          characteristic->setValue(output);
          characteristic->notify();
          http.end();
        }
    }
  } else {
      Serial.println("JSON deserialization failed");
    }
  }
};

void setup() {
  Serial.begin(115200);

  BLEDevice::init(bleServerName);

  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  BLEService *bleService = pServer->createService(SERVICE_UUID);
  bleService->addCharacteristic(&characteristic);
```

```
  characteristic.addDescriptor(characteristicDescriptor);
  characteristic.setCallbacks(new CharacteristicsCallbacks());

  bleService->start();

  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->start();

  Serial.println("Waiting for a client connection to notify...");
}

void loop() {
  // Your loop code (if needed)
}
```

Concluzii

Dificultăți întâmpinate:

-pentru acțiunea ***getDetails***, ESP32 nu putea să trimită toate datele prin BLE deoarece depășeau memoria maximă, astfel apărea eroarea **Invalid Response** pe aplicație

În rest, desfășurarea proiectului a decurs ușor și repede.

Bibliografie

1. https://randomnerdtutorials.com/projects-esp32/
2. https://microdigisoft.com/esp32-with-arduino-json-using-arduino-ide/