Mihăeș Ionuț-George, Văceanu Bianca
Grupa 412B

PROIECT INFORMATICĂ APLICATĂ

Cuprins:

1. **Introducere** **2**
2. **Considerații teoretice** **2**
   Modulul ESP32 2
   Protocoale și metode de comunicație utilizate 3
3. **Implementare** 3
4. **Concluzii** 5
5. **Bibliografie** 6
6. **Anexe** 6

1. Introducere

Scopul proiectului pe care l-am realizat este de a implementa un program care poate accepta date prin metoda de comunicație Bluetooth Low Energy. În urma comenzilor primite de la aplicația mobilă se va executa interogarea prin cereri HTTP a API-ului http://proiectia.bogdanflorea.ro/api/better-call-saul/characters, folosind modulul ESP32, iar datele vor fi returnate prin Bluetooth Low Energy către aceasta.

Breaking Bad este considerat unul dintre cele mai mari seriale de televiziune din toate timpurile și, cu 16 premii Primetime Emmy, merită propriul API. API-ul Breaking Bad este o colecție de informații despre serialul lui Vince Gilligan, Breaking Bad, precum și despre Better Call Saul. Acest site este conceput pentru a facilita dezvoltatorului să vadă ce solicitări HTTP sunt posibile și ce informații sunt disponibile.

Prin acest API sunt furnizate date despre:

-personaje;
-episoade;
-citate;
-decese.

2. Considerații 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. 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.

Protocoalele si metodele de comunicație utilizate:

-Bluetooth Low Energy este menit să ofere un consum de energie și un cost redus considerabil, menținând în același timp o rază de comunicare similară. Tehnologia Bluetooth Low Energy operează în același interval de spectru ca tehnologia Bluetooth clasică, dar utilizează un set diferit de canale. În loc de clasicele canale Bluetooth, 79 de 1 MHz, Bluetooth Low Energy are 40 de canale de 2 MHz.

-WiFi este numele comercial pentru tehnologiile construite pe baza standardelor de comunicație din familia IEEE 802.11 utilizate pentru realizarea de rețele locale de comunicație fără fir (wireless, WLAN) la viteze echivalente cu cele ale rețelelor cu fir electric. Suportul pentru Wi-Fi este furnizat de diferite dispozitive hardware, și de aproape toate sistemele de operare moderne pentru calculatoarele personale, rutere, telefoane mobile, console de jocuri și cele mai avansate televizoare.

-Hypertext Transfer Protocol (HTTP) este metoda cea mai des utilizată pentru accesarea informațiilor în Internet care sunt păstrate pe servere World Wide Web (WWW). HTTP oferă o tehnică de comunicare prin care paginile web se pot transmite de la un computer aflat la distanță spre propriul computer. Dacă se apelează un link sau o adresă de web, atunci se cere calculatorului host să afișeze o pagină web. În prima fază, numele (adresa) este convertit de protocolul DNS într-o adresă IP.

-JSON (JavaScript Object Notation) este un format ușor de schimb de date, cu un limbaj complet independent, dar care utilizează convenții familiare programatorilor din familia de limbaje C, inclusiv C, C++, C#, Java, JavaScript, Perl, Python și multe altele. Aceste proprietăți fac din JSON un limbaj ideal pentru schimbul de date. JSON este construit pe două structuri:

- O colecție de perechi nume/valoare. În diferite limbaje, acest lucru este realizat ca obiect , înregistrare, structură, dicționar, tabel de dispersie, listă cheie sau matrice asociativă.
- O listă ordonată de valori. În majoritatea limbajelor, acest lucru este realizat ca o matrice , vector, listă sau secvență.

3. Implementare

În prima parte ne-am ocupat atât de definirea unui UUID unic generat, folosind link-ul https://www.uuidgenerator.net/, cât și de cea a caracteristicilor pentru datele ce vor fi trimise către placa ESP32. Am folosit „index” pentru datele din lista de personaje, iar „details” pentru detaliile despre fiecare personaj în parte.

```
bool deviceConnected = false;
#define SERVICE_UUID "91bad492-b950-4226-aa2b-4ede9fa42f59"


BLECharacteristic indexCharacteristic(
  "ca73b3ba-39f6-4ab3-91ae-186dc9577d99",
  BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE | BLECharacteristic::PROPERTY_NOTIFY
);

BLEDescriptor *indexDescriptor = new BLEDescriptor(BLEUUID((uint16_t)0x2901));

BLECharacteristic detailsCharacteristic(
  "183f3323-b11f-4065-ab6e-6a13fd7b104d",
  BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE | BLECharacteristic::PROPERTY_NOTIFY
);

BLEDescriptor *detailsDescriptor = new BLEDescriptor(BLEUUID((uint16_t)0x2902));
```

Apoi, am plasat în documentul ,,bine” (asemănător unui vector, map), datele din string-ul intitulat ,,data” pentru a le putea prelucra cu o ușurință mai mare. Ulterior, datele au fost trimise către placă.

```
class CharacteristicsCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *characteristic) {
      // Get characteristic value sent from the app, according to the specs
      std::string data = characteristic->getValue();
      DynamicJsonDocument bine(15000);
      deserializeJson(bine,data.c_str());
HTTPClient pl;
pl.begin(url);
```

Dacă trimiterea datelor s-a efectuat cu succes (if(pl.GET()>0), am folosit din nou funcția deserializeJson pentru a pune în documentul ,,doc”, datele string-ului trimis către placă.

Pentru lista cu personaje (if(bine["action"]=="fetchData")), am plasat în ,,temporar” datele ce vor apărea în aceasta, ele fiind ID-ul, numele și imaginea personajului. După prelucrare, am pus noile date în string, folosind funcția serializeJson, pentru ca ulterior acestea să poată fi trimise către placă.

```
if(pl.GET()>0)
{
  DynamicJsonDocument doc(15000);
      deserializeJson(doc,pl.getString());
       JsonArray list = doc.as<JsonArray>();

    if(bine["action"]=="fetchData")
    {for (JsonVariant value : list) {
      JsonObject listItem = value.as<JsonObject>();

      DynamicJsonDocument temporar(15000);
      temporar["id"]=listItem["char_id"].as<String>();
      temporar["name"]=listItem["name"].as<String>();
      temporar["image"]=listItem["img"].as<String>();

      String stringTemporar;
      serializeJson(temporar,stringTemporar);
      indexCharacteristic.setValue(stringTemporar.c_str());
      indexCharacteristic.notify();}}
```

Pentru detaliile despre personaj, am afișat din nou principalele date (ID, nume și imagine), urmând ca apoi să adăugam detaliile primite în string-ul ,,descriere”, fiecare detaliu accesându-se din URL-ul oferit. Pentru a oferi un rezultat mai placut, din punct de vedere al aspectului, am decis să eliminăm, folosind funcții ,,while”, parantezele și ghilimelele pentru a nu-și face apariția în aplicație. Am plasat, apoi, în ,,temporar” datele din ,,descriere”, urmând ca documentul ,,temporar” să fie adăugat în string pentru a fi trimis catre placă.

```
else if(bine["action"]=="fetchDetails"){
  for (JsonVariant value : list) {
JsonObject listItem = value.as<JsonObject>();
if(listItem["char_id"].as<String>()==bine["id"].as<String>()){
DynamicJsonDocument temporar(15000);
temporar["id"]=listItem["char_id"].as<String>();
temporar["name"]=listItem["name"].as<String>();
temporar["image"]=listItem["img"].as<String>();

String descriere="birthday:"+listItem["Birthday"].as<String>()+
"\n"+"Occupation:"+listItem["occupation"].as<String>()+
"\n"+"Status:"+listItem["status"].as<String>()+
"\n"+"Nickname:"+listItem["nickname"].as<String>()+
"\n"+"Portrayed:"+listItem["portrayed"].as<String>();
while(descriere.indexOf('[')!=-1)
descriere.remove(descriere.indexOf('['),1);
while(descriere.indexOf(']')!=-1)
descriere.remove(descriere.indexOf(']'),1);
 while(descriere.indexOf('"')!=-1)
descriere.remove(descriere.indexOf('"'),1);
temporar["description"]=descriere;
String stringTemporar;
serializeJson(temporar,stringTemporar);
detailsCharacteristic.setValue(stringTemporar.c_str());
detailsCharacteristic.notify();}
```

Am realizat ulterior conectarea plăcii la rețeaua de WiFi prin preluarea datelor definite la începutul codului (SSID și parolă).

```
void setup() {
  // Start serial communication
  Serial.begin(115200);
  Serial.printf("Connecting to %s ", ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
  }

  Serial.println("CONNECTED to WIFI");
```

În cele din urmă am creat atât serverul pentru Bluetooth Low Energy, făcând posibilă conectarea plăcii la telefon, cât și câteva funcții de apel invers (callbacks) ce permit folosirea unor funcții ca argument pentru alte funcții.

```
  BLEDevice::init(bleServerName);
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
  BLEService *bmeService = pServer->createService(SERVICE_UUID);
  bmeService->addCharacteristic(&indexCharacteristic);
  indexDescriptor->setValue("Get data list");
  indexCharacteristic.addDescriptor(indexDescriptor);
  indexCharacteristic.setValue("Get data List");
  indexCharacteristic.setCallbacks(new CharacteristicsCallbacks());
```

4. Concluzii

Cu ajutorul modulului ESP32 am reușit să implementăm un program care poate primi și transmite date (prin interogări HTTP) referitoare la personaje din serialul de success Breaking Bad, pe care le vom transmite în aplicația mobilă folosind Bluetooth Low Energy, oferind un consum de energie redus și menținând în același timp o rază de comunicare similară.

5. Bibliografie

https://breakingbadapi.com/
http://proiectia.bogdanflorea.ro/api
http://proiectia.bogdanflorea.ro/api/12
https://en.wikipedia.org/wiki/Bluetooth_Low_Energy
https://ro.wikipedia.org/wiki/Wi-Fi
https://ro.wikipedia.org/wiki/Hypertext_Transfer_Protocol
https://www.uuidgenerator.net/
https://www.json.org/json-en.html
http://proiectia.bogdanflorea.ro/app

6. Anexe

```
#include <Arduino.h>

#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <ArduinoJson.h>
#include <WiFi.h>
#include <HTTPClient.h>
#define ssid "TP-Link_13B0"
#define password "61011557"
#define url "http://proiectia.bogdanflorea.ro/api/better-call-saul/characters"
#define bleServerName "serverble"

bool deviceConnected = false;
#define SERVICE_UUID "91bad492-b950-4226-aa2b-4ede9fa42f59"


BLECharacteristic indexCharacteristic(
  "ca73b3ba-39f6-4ab3-91ae-186dc9577d99",
  BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE |
BLECharacteristic::PROPERTY_NOTIFY
);

BLEDescriptor *indexDescriptor = new BLEDescriptor(BLEUUID((uint16_t)0x2901));

BLECharacteristic detailsCharacteristic(
  "183f3323-b11f-4065-ab6e-6a13fd7b104d",
  BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE |
BLECharacteristic::PROPERTY_NOTIFY
```

```
);

BLEDescriptor *detailsDescriptor = 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) {
      std::string data = characteristic->getValue();
      DynamicJsonDocument bine(15000);
      deserializeJson(bine,data.c_str());
HTTPClient pl;
pl.begin(url);

if(pl.GET()>0)
{
  DynamicJsonDocument doc(15000);
      deserializeJson(doc,pl.getString());
       JsonArray list = doc.as<JsonArray>();

    if(bine["action"]=="fetchData")
    {for (JsonVariant value : list) {
      JsonObject listItem = value.as<JsonObject>();

      DynamicJsonDocument temporar(15000);
      temporar["id"]=listItem["char_id"].as<String>();
      temporar["name"]=listItem["name"].as<String>();
      temporar["image"]=listItem["img"].as<String>();

      String stringTemporar;
      serializeJson(temporar,stringTemporar);
      indexCharacteristic.setValue(stringTemporar.c_str());
      indexCharacteristic.notify();}}
```

```
      else if(bine["action"]=="fetchDetails"){
        for (JsonVariant value : list) {
      JsonObject listItem = value.as<JsonObject>();
      if(listItem["char_id"].as<String>()==bine["id"].as<String>()){
      DynamicJsonDocument temporar(15000);
      temporar["id"]=listItem["char_id"].as<String>();
      temporar["name"]=listItem["name"].as<String>();
      temporar["image"]=listItem["img"].as<String>();

      String descriere="birthday:"+listItem["Birthday"].as<String>()+
      "\n"+"Occupation:"+listItem["occupation"].as<String>()+
      "\n"+"Status:"+listItem["status"].as<String>()+
      "\n"+"Nickname:"+listItem["nickname"].as<String>()+
      "\n"+"Portrayed:"+listItem["portrayed"].as<String>();
      while(descriere.indexOf('[')!=-1)
      descriere.remove(descriere.indexOf('['),1);
      while(descriere.indexOf(']')!=-1)
      descriere.remove(descriere.indexOf(']'),1);
       while(descriere.indexOf("")!=-1)
      descriere.remove(descriere.indexOf(""),1);
      temporar["description"]=descriere;
      String stringTemporar;
      serializeJson(temporar,stringTemporar);
      detailsCharacteristic.setValue(stringTemporar.c_str());
      detailsCharacteristic.notify();}
      }
     }
     }

else
{
  Serial.println("eroare");
}
}
};

void setup() {
  Serial.begin(115200);
  Serial.printf("Connecting to %s ", ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
  }

  Serial.println("CONNECTED to WIFI");
```

```
  BLEDevice::init(bleServerName);
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
  BLEService *bmeService = pServer->createService(SERVICE_UUID);
  bmeService->addCharacteristic(&indexCharacteristic);
  indexDescriptor->setValue("Get data list");
  indexCharacteristic.addDescriptor(indexDescriptor);
  indexCharacteristic.setValue("Get data List");
  indexCharacteristic.setCallbacks(new CharacteristicsCallbacks());

  bmeService->addCharacteristic(&detailsCharacteristic);
  detailsDescriptor->setValue("Get data details");
  detailsCharacteristic.addDescriptor(detailsDescriptor);
  detailsCharacteristic.setValue("Get data details");
  detailsCharacteristic.setCallbacks(new CharacteristicsCallbacks());

  bmeService->start();

  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pServer->getAdvertising()->start();
  Serial.println("Waiting a client connection to notify...");
}

void loop() {
}
```