Proiect Informatică Aplicată

Grupa 413B
Stanov George-Cosmin
Steria Valentin-Mihai

1. Introducere

Proiectul se bazează pe implementarea unui program care să ofere printr-un API interogat prin cereri http datele unei liste. Prin folosirea metodei de comunicație Bluetooth Classic si a unui microcontroller ESP32 putem afișa datele listei pe aplicația support de pe telefon (ProiectIA). API este acronimul pentru Application Programming Interface, care este un intermediar software care permite a două aplicații să comunice între ele.

Tematica proiectului este jocul serialul Game of Thrones. Trebuie sa afișăm pe aplicația support caracteristicile personajelor, care sunt: ID-ul , prenumele, numele de familie, numele complet, titlul, familia, si o imagine cu acesta.

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. Programarea modulelor ESP32 se poate realiza folosind diverse medii de dezvoltare, cele mai populare fiind Arduino IDE si Visual Studio Code împreună cu extensia PlatformIO.

Un protocol de comunicație folosit este Bluetooth Classic. Aceasta este o tehnologie wireless care permite schimbul de date între diferite dispozitive. Diferența intre Bluetooth Classic si Bluetooth Low Energy este ca cel clasic poate gestiona o mulțime de date, dar consumă rapid durata de viață a bateriei și costă mult mai mult, pe când cel low energy este folosit pentru aplicații care nu au nevoie să facă schimb de cantități mari de date și, prin urmare, pot funcționa pe baterie ani de zile la un cost mai ieftin.

Wi-Fi este tehnologia wireless folosită pentru a conecta computere, tablete, smartphone-uri și alte dispozitive la internet. De asemenea, este semnalul radio trimis de la un router wireless către un dispozitiv din apropiere, care traduce semnalul în date pe care le puteți vedea și utiliza. Dispozitivul transmite un semnal radio înapoi către router, care se conectează la internet prin fir sau cablu.

Hypertext Transfer Protocol (HTTP) este fundamentul World Wide Web și este folosit pentru a încărca pagini web folosind link-uri hipertext. HTTP

este un protocol de nivel de aplicație conceput pentru a transfera informații între dispozitivele din rețea și rulează peste alte straturi ale stivei de protocoale de rețea. Un flux tipic prin HTTP implică o mașină client care face o solicitare către un server, care trimite apoi un mesaj de răspuns.

JSON este un acronim în limba engleză pentru *JavaScript Object Notation*, și este un format de reprezentare și interschimb de date între aplicații informatice. Este un format text, inteligibil pentru oameni, utilizat pentru reprezentarea obiectelor și a altor structuri de date și este folosit în special pentru a transmite date structurate prin rețea, procesul purtând numele de serializare. JSON este alternativa mai simplă, mai facilă decât limbajul XML. Eleganța formatului JSON provine din faptul că este un subset al limbajului JavaScript, fiind utilizat alături de acest limbaj.

3. Implementare

Am folosit următoarele biblioteci:

```
#include <Arduino.h>
#include <BluetoothSerial.h>
#include <ArduinoJson.h>
#include <WiFi.h>
#include <HTTPClient.h>
```

Am început prin a conecta microcontroller-ul la wi-fi, apoi folosind tehnologia Bluetooth clasic am conectat smartphone-ul.
Am creat funcția receivedData care ne returnează acțiunea pe care o îndeplinește microcontroller-ul în momentul respectiv, spre exemplu după apăsarea butonului „Get data" funcția va returna acțiunea „fetchData", pe aplicație va apărea lista cu id, nume si imagine pentru fiecare personaj, iar în consola serială vor apărea aceste detalii ale fiecărui personaj; ulterior funcția va returna „empty" si va aștepta următoarea comandă.
Am inițializat string-urile corespunzătoare caracteristicilor caracterelor cu datele din fișierul JSON.

Am creat funcția sendDataToApp pentru a lua toate detaliile personajelor (id, fullname, img) ca într-o lista, incrementând un index pe care îl atribuim fiecărui obiect si a le trimite către telefon pentru a fi afișate în aplicație.

4. Concluzii

Gama de aplicații a modulelor ESP32 este foarte largă. Acestea sunt deosebit de bine adaptate aparatelor pentru automatizare a casei, electronicelor uzate și Internetului Lucrurilor (IoT). Un procesor eficient și numeroase periferice vă permit să implementați proiecte foarte complexe și avansate.

Utilizarea ESP32 se dovedește a fi surprinzător de simplă grație numeroaselor soluții la gata. Toate software-urile necesare sunt disponibile gratuit și, în special, pe Internet, veți găsi numeroase biblioteci și numeroase proiecte finalizate și gata de folosit, care pot servi drept sursă de inspirație sau drept bază pentru propriile dumneavoastră studii.

5. Bibliografie

- Curs https://www.bluetooth.com/learn-about-bluetooth/tech-overview/
- https://www.argenox.com/library/bluetooth-classic/introduction-to-bluetooth-classic/
- https://ro.wikipedia.org/wiki/Bluetooth
- https://www.json.org/json-en.html
- https://arduinogetstarted.com/tutorials/arduino-http-request
- https://randomnerdtutorials.com/esp32-bluetooth-classic-arduino-ide/
- https://www.electronicshub.org/esp32-bluetooth-tutorial/
- https://arduinojson.org/v6/example/parser/

6. Anexe (codul complet)

```
#include <Arduino.h>
#include <BluetoothSerial.h>
#include <ArduinoJson.h>
#include <WiFi.h>
#include <HTTPClient.h>

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
  #error Bluetooth is not enabled! Please run `make menuconfig` to and enable
it
#endif

// TODO -- Define WiFi credentials
#define ssid "Xiaomi 11 Lite 5G NE"
#define password "123qwe456rty"

String baseURL = "http://proiectia.bogdanflorea.ro/api/game-of-thrones/";
String adder="";
String payload="";

#define btcServerName "Steria&Stanov"

byte BTData;
bool deviceConnected = false;

// TODO --  Generate a unique UUID for your Bluetooth service
// See the following for generating UUIDs:
// https://www.uuidgenerator.net/
#define SERVICE_UUID "2237db39-bad1-486d-bb0e-be326848c7cd"
String apiString;
String phRes;
String newID;
int typeReq;

// Define the BluetoothSerial
BluetoothSerial SerialBT;

// Received data
String data = "";

// The receivedData function is called when data is available on Bluetooth
(see loop function)
  String receivedData() {
  // Read the data using the appropriate SerialBT functions
```

```
  // according to the app specifications
  // The data is terminated by the new line character (\n)
  String dataFromApp = "";
  while (SerialBT.available()) {
    dataFromApp = SerialBT.readStringUntil('\n');
  }

   DynamicJsonDocument doc(4096);
   DeserializationError error = deserializeJson(doc, dataFromApp);

    // Test if parsing succeeds.
    if (error) {
      Serial.print(F("Do something: "));
      Serial.println(error.f_str());

     return "empty";
    }

    String action=doc["action"];
    if(action=="fetchData"){
        Serial.print("New action: ");
        Serial.println(action);
        return action;
    }
   else if(action=="fetchDetails"){
      String id=doc["id"];
      Serial.print("New action: ");
      Serial.print(action);
      Serial.print(" for id: ");
      Serial.println(id);
      newID=id;
      return action;
   }

}

String httpGETRequest(String adder) {

  HTTPClient http;
  String path=baseURL+adder;
  String payload ="";
  Serial.print("Attempting to connect to: ");
  Serial.println(path);  // <-- This is the message sent from the app,
according to the specs

  //Specify the URL
    http.setTimeout(12000);
    http.begin(path);
```

```
    //Make the request
    int httpCode = http.GET();

    if (httpCode == 200) { //Check for the returning code

        payload = http.getString();
        Serial.println("GOOD");
      }

    else {

      Serial.print("Error on HTTP request: ");
      Serial.println(httpCode);
    }

    http.end(); //Free the resources

  return payload;
}

  // Possible steps:
  // 1. Deserialize data using the ArduinoJson library
  // 2. Get the action from the JSON object
  // 3. Check action and perform the corresponding HTTP request (check the
method in the API specifications)
  // (Define the API url corresponding to the action and get the response)
  // IMPORTANT -- make sure to set the http request timeout to 10s or more
  // 4. Check the response code and deserialize the response data
  // IMPORTANT -- If using the ArduinoJson library, use a DynamicJsonDocument
with the size of 15000
  // 5. Define the response structure, according to the data returned by the
API
  // (Use JsonArray or JsonObject, depending on the response type)
  // IMPORTANT -- The cacapcity must not exceed 512B, especially for BLE
  // 6. Populate the response object according to the API and app specs
  // 7. Encode the response object as a JSON string
  // 8. Write value to the appropriate characteristic and notify the app

  // TODO -- Write your code
  void sendDataToApp_Details(String payload){
    StaticJsonDocument <768> JSONDocument;
    DeserializationError error = deserializeJson(JSONDocument,
payload.c_str());

  if (error) {
    Serial.print("Problem with fetchDetails: ");
    Serial.println(error.c_str());

  } else {
```

```
      String responseString;
      String id=JSONDocument["id"].as<String>();
      String firstName=JSONDocument["firstName"].as<String>();
      String lastName=JSONDocument["lastName"].as<String>();
      String fullName=JSONDocument["fullName"].as<String>();
      String title=JSONDocument["title"].as<String>();
      String family=JSONDocument["family"].as<String>();
      String imageUrl=JSONDocument["imageUrl"].as<String>();


      String description="ID: "+id+"\n"+"First Name: "+firstName+"\n"+"Last
Name: "+lastName+"\n"+"Full Name: "+fullName+"\n"+"Title:
"+title+"\n"+"Family:"+family+"\n"+"Image Url:"+imageUrl;
      StaticJsonDocument <512>
newJSONDocument;//https://arduinojson.org/v6/assistant

      JsonObject object = newJSONDocument.to<JsonObject>();

      object["id"] = id;
      object["name"] = fullName;
      object["image"] = imageUrl;
      object["description"] = description;

      serializeJson(newJSONDocument, responseString);

      SerialBT.println(responseString);

      Serial.println(responseString);

  }
}

void sendDataToApp(String payload){
  DynamicJsonDocument JSONDocument(8192);
  DeserializationError error = deserializeJson(JSONDocument, payload.c_str());

  if (error) {
    Serial.print("Problem with fetchData: ");
    Serial.println(error.c_str());

  } else {
    // Define a JsonArray from the JSONDocument, since the JSONString is an
array of objects
    JsonArray list = JSONDocument.as<JsonArray>();

    // Iterate the JsonArray array (see docs for the library)
    int index = 1;
    for (JsonVariant value : list) {
      JsonObject listItem = value.as<JsonObject>();
```

```
      // Get the current item in the iterated list as a JsonObject

      String id=listItem["id"].as<String>();
      String fullName=listItem["fullName"].as<String>();
      String imageUrl=listItem["imageUrl"].as<String>();
      String responseString;

      StaticJsonDocument <4096>
newJSONDocument;//https://arduinojson.org/v6/assistant

      JsonObject object = newJSONDocument.to<JsonObject>();

      object["id"] = id;
      object["name"] = fullName;
      object["image"] = imageUrl;
      serializeJson(newJSONDocument, responseString);
      SerialBT.println(responseString);

      Serial.println(responseString);

      delay(100);
      index++;
    }
  }
}

void setup() {

  // put your setup code here, to run once:
  Serial.begin(115200);
  // TODO -- Connect WiFi
  // TODO -- Write your code
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");

  // No need to change the lines below
  // Initialize BTC
 SerialBT.begin(btcServerName); //Bluetooth device name
 Serial.println("The device started, now you can pair it with bluetooth!");

}
```

```
void loop() {

   // put your main code here, to run repeatedly:
  // Check available Bluetooth data and perform read from the app
    if(WiFi.status()== WL_CONNECTED){
      if(SerialBT.connected()){

        phRes=receivedData();

       if(phRes=="fetchData"){
          String payload=httpGETRequest("characters");
          sendDataToApp(payload);
       }
       else if(phRes=="fetchDetails"){
          String payload=httpGETRequest("character/" + newID);
          sendDataToApp_Details(payload);
       }
       else if(phRes=="empty"){
         Serial.println("Waiting for a new task");
       }
      }
      else {
        Serial.print("Waiting for device to connect through bluetooth");
        while(!SerialBT.connected()){
          delay(500);
          Serial.print(".");
        }
        Serial.println();
      }
    }
   else {
      Serial.println("WiFi Disconnected");
   }
   delay(400);
}
```