UNIVERSITATEA POLTEHNICĂ BUCUREȘTI
FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

BLĂNARU NICOLAE-BOGDAN
BLEOTU CĂTĂLIN-MIHAI
GRUPA 414A

Proiect – Informatică aplicată

LCD 16x2 control using Bluetooth Low Energy

I. Introducere

Programul va implementa controlul unui LCD 16x2 folosind acțiunile specifice solicitate de aplicația mobilă ProiectIA. Aceste acțiuni vor permite afișarea de texte, pictograme personalizate (caractere speciale) și derularea derularea textului în ambele direcții, ideal prin utilizarea de întreruperi.

II. Considerente teoretice

a) Modulul ESP32

ESP32 este o placă de dezvoltare (microcontroler) bazată pe un chip de la Espressif Systems, numit ESP32. Este o platformă populară pentru dezvoltarea de aplicații Internet of Things (IoT), datorită funcționalității sale avansate și a puterii de calcul. ESP32 oferă o conexiune Wi-Fi și Bluetooth integrate, multiple GPIO-uri (General Purpose Input/Output), interfețe pentru comunicare serială și posibilitatea de a rula cod personalizat prin programare în limbajul C/C++ sau utilizând platforma Arduino IDE. Este utilizat într-o gamă largă de aplicații IoT, precum automatizare casnică, monitorizare a mediului, dispozitive portabile și multe altele.

ESP32 dispune de o serie de componente și pini cheie, inclusiv:

1. CPU Dual-Core: ESP32 este echipat cu un CPU dual-core Xtensa LX6, care rulează la o frecvență de până la 240 MHz.
2. Flash Memory: ESP32 are o memorie flash internă de capacitate variabilă, care poate fi utilizată pentru stocarea codului programului și a datelor.
3. RAM: Dispune de memorie RAM internă pentru stocarea datelor și a stării temporare a programului.
4. GPIO (General Purpose Input/Output): ESP32 oferă mai mulți pini GPIO pentru conectarea dispozitivelor externe, cum ar fi butoane, LED-uri, senzori, module de comunicație etc.
5. UART (Universal Asynchronous Receiver/Transmitter): ESP32 are mai multe porturi UART care permit comunicarea serială asincronă cu alte dispozitive.
6. SPI (Serial Peripheral Interface): Dispune de porturi SPI pentru comunicarea sincronă cu dispozitive periferice, cum ar fi senzorii, ecranele TFT și modulele de memorie.
7. I2C (Inter-Integrated Circuit): ESP32 suportă interfața I2C pentru conectarea cu senzori, afișaje OLED, accelerometre și alte dispozitive periferice.
8. I2S (Inter-IC Sound): ESP32 are o interfață I2S pentru comunicarea cu dispozitivele audio, cum ar fi codec-urile audio sau modulele de înregistrare/reproducere audio.
9. ADC (Analog-to-Digital Converter): Dispune de convertor analogic-digitale pentru citirea valorilor de la senzori analogici, cum ar fi senzorii de temperatură sau de lumină.
10. WiFi și Bluetooth: ESP32 integrează module WiFi și Bluetooth, oferind conectivitate wireless pentru comunicația cu alte dispozitive și rețele.

Acestea sunt doar câteva dintre componentele și interfețele cheie ale ESP32. Placa mai are și alte componente precum modulul de alimentare, cristalul oscilator, regulatorul de tensiune etc., care sunt esențiale pentru funcționarea sa corectă.

b) Protocoalele și metodele de comunicație utilizate

În realizarea acestui proiect, am utilizat protocoalele si metodele de comunicație Bluetooth Low Energy si JSON.

- ### Bluetooth Low Energy

Bluetooth Low Energy (BLE) este o variantă a tehnologiei Bluetooth, concepută special pentru aplicații cu consum redus de energie. A fost introdusă pentru a satisface cerințele dispozitivelor IoT (Internet of Things) și a dispozitivelor mobile care necesită conectivitate fără fir cu consum redus de energie.

Principalele caracteristici ale Bluetooth Low Energy includ:

1. Consum redus de energie: BLE a fost proiectat pentru a fi extrem de eficient în ceea ce privește consumul de energie. Dispozitivele BLE pot funcționa cu baterii de dimensiuni mici sau chiar cu baterii buton timp îndelungat, în unele cazuri până la câțiva ani, în funcție de utilizare.
2. Comunicare la distanțe scurte: BLE este optimizat pentru comunicația la distanțe scurte, în general până la câțiva metri. Acest lucru este de obicei suficient pentru majoritatea aplicațiilor IoT și dispozitivelor portabile.
3. Viteze de transfer reduse: În comparație cu Bluetooth clasic, BLE oferă rate de transfer de date mai mici. Acest lucru este suficient pentru schimbul de date de dimensiuni mici, cum ar fi informațiile despre senzori sau comenzi de la dispozitive de control.
4. Profiluri și servicii: BLE utilizează conceptul de profiluri și servicii, care permit dispozitivelor să comunice într-un mod standardizat. Există o varietate de profiluri BLE disponibile, precum profilul de monitorizare a sănătății, profilul de telecomandă sau profilul de transfer de date.

Bluetooth Low Energy (BLE) funcționează pe principiul transmiterii și recepționării de pachete de date între dispozitive. Iată o descriere succintă a modului în care funcționează BLE:

1. Modele de dispozitive: Există două modele de bază în BLE: dispozitivul periferic și dispozitivul central. Dispozitivul periferic este de obicei un dispozitiv cu resurse reduse, cum ar fi senzori sau dispozitive portabile, care transmit date către un dispozitiv central. Dispozitivul central este în general un smartphone, tabletă sau computer care se conectează și interacționează cu dispozitivele periferice.
2. Anunțuri și scanare: Dispozitivele periferice pot trimite anunțuri periodice, cunoscute și sub numele de advertising packets. Dispozitivul central poate efectua scanarea pentru a detecta aceste anunțuri și pentru a identifica dispozitivele disponibile în apropiere.
3. Conexiune: După detectarea unui dispozitiv periferic de către dispozitivul central, acesta poate iniția o conexiune BLE. Acest proces implică un protocol de negociere și stabilire a parametrilor de comunicație între cele două dispozitive.
4. Servicii și caracteristici: Odată ce conexiunea BLE este stabilită, dispozitivul periferic expune servicii și caracteristici. Serviciile reprezintă funcționalități specifice oferite de dispozitivul periferic, iar caracteristicile reprezintă date specifice care pot fi citite sau scrise.
5. Cereri și răspunsuri: Dispozitivul central poate trimite cereri de citire sau scriere a datelor către dispozitivul periferic prin intermediul caracteristicilor expuse. Dispozitivul periferic va răspunde în consecință, fie prin trimiterea datelor solicitate, fie prin confirmarea efectuării unei acțiuni.
6. Deconectare: Odată ce comunicarea este încheiată sau când dispozitivele nu mai sunt în raza de acoperire, conexiunea BLE poate fi încheiată prin deconectare.

Acesta este un flux general al funcționării BLE, care poate varia în funcție de aplicație și de implementare. Protocoalele și specificațiile BLE sunt stabilite de Bluetooth Special Interest Group (SIG), care se asigură că dispozitivele compatibile cu BLE pot interacționa între ele în mod eficient și interoperabil.

Biblioteci specifice BLE

1. BLEDevice.h: Această bibliotecă oferă funcționalități esențiale pentru gestionarea dispozitivului BLE central sau periferic. Aceasta include inițializarea și configurarea dispozitivului BLE, gestionarea conexiunilor BLE și alte funcții asociate.

2. BLEServer.h: Această bibliotecă este utilizată pentru crearea unui server BLE pe dispozitivul tău. Poți crea servicii și caracteristici BLE personalizate care să fie expuse de dispozitivul tău periferic.

3. BLEUtils.h: Această bibliotecă conține funcții de utilitate pentru gestionarea și manipularea datelor BLE. Aceasta include funcții pentru construirea și parsarea pachetelor BLE, convertirea datelor în formate specifice și alte funcții utile.

4. BLE2902.h: Această bibliotecă oferă suport pentru descrierea și manipularea caracteristicilor de configurare a clientului BLE. Aceasta include, de exemplu, setarea de notificări sau indicații pentru o anumită caracteristică BLE.

Aceste biblioteci sunt frecvent utilizate în dezvoltarea aplicațiilor BLE cu platforma Arduino. Ele oferă funcționalități esențiale pentru gestionarea conexiunilor BLE, crearea și configurarea serviciilor și caracteristicilor BLE și manipularea datelor BLE.

- JSON

JSON (JavaScript Object Notation) este un format de date ușor de utilizat și ușor de citit de către oameni și mașini. A fost inițial dezvoltat pentru a fi utilizat în limbajul de programare JavaScript, dar a devenit un format de date popular și utilizat într-o varietate de limbaje de programare.

JSON este bazat pe două structuri de bază:

1. Obiecte: Un obiect JSON este o colecție neordonată de perechi cheie-valoare. Cheile sunt de obicei de tip șir de caractere și trebuie să fie unice în cadrul obiectului JSON. Valorile pot fi de diferite tipuri de date, cum ar fi șiruri de caractere, numere, valori booleene (true/false), alte obiecte JSON sau tablouri JSON.

Exemplu de obiect JSON:

```
{
    type: enum('16x2', '20x4', 'OLED 1306', 'Nokia 3310', 'Nokia 5510'),
    interface: enum('I2C', 'SPI', 'Parallel 4-bit'),
    resolution: string,
```

```
    id: int,
    teamId: string
}
```

2. Tablouri: Un tablou JSON este o colecție ordonată de valori. Aceste valori pot fi de orice tip de date acceptat în JSON, inclusiv șiruri de caractere, numere, valori booleene, obiecte JSON sau alte tablouri JSON.

Exemplu de tablou JSON:

```
icons: [
        {
            name: string,
            data: [byte array]
        }
        ...
    ]
```

Biblioteca <ArduinoJson.h>

Biblioteca <ArduinoJson.h> bibliotecă populară utilizată în programarea cu platforma Arduino pentru manipularea și procesarea datelor JSON. Această bibliotecă oferă funcționalități pentru crearea, citirea, scrierea și manipularea datelor JSON într-un mod simplu și eficient.

Cu ajutorul bibliotecii <ArduinoJson.h>, poți efectua următoarele operații:

1. Parsare JSON: Poți parsa un șir de caractere JSON și extrage valorile individuale pentru a fi utilizate în programul tău Arduino.
2. Serializare JSON: Poți crea un obiect JSON sau un tablou JSON și să-l transformi într-un șir de caractere JSON pentru a fi trimis sau stocat într-un alt dispozitiv sau serviciu.
3. Manipularea datelor JSON: Poți adăuga, șterge sau modifica chei și valori într-un obiect JSON sau un tablou JSON existent.
4. Validarea structurii JSON: Poți verifica dacă un șir de caractere JSON respectă structura JSON corectă sau dacă conține toate cheile și valorile necesare.

Biblioteca <ArduinoJson.h> oferă o interfață simplă și ușor de utilizat, care face manipularea datelor JSON accesibilă și convenabilă în mediul Arduino. Aceasta facilitează comunicarea cu servicii web, interacțiunea cu alte dispozitive sau stocarea datelor structurate în format JSON.

c) Componente utilizate

- LCD 16x2

Un LCD (Liquid Crystal Display) de tip 16x2 se referă la un tip specific de afișaj cu cristale lichide care are capacitatea de a afișa 16 caractere pe linie și 2 linii de text.

Noțiuni teoretice despre un LCD 16x2:

1. Structura fizică: Un LCD 16x2 este compus dintr-un ansamblu de cristale lichide dispuse între două straturi de substrat de sticlă. Fiecare caracter este format dintr-un set de segmente sau puncte care pot fi controlate individual pentru a crea litere, cifre și alte simboluri.

2. Controler: Un LCD 16x2 este echipat cu un controler care controlează modul în care caracterele și simbolurile sunt afișate pe ecran. Controlerele comune utilizate pentru LCD-urile de tip 16x2 sunt HD44780 sau compatibile cu acesta.

3. Interfață: LCD-urile 16x2 sunt de obicei conectate la un microcontroler sau alt dispozitiv controlat de la distanță prin intermediul unei interfețe, cum ar fi o interfață paralelă sau o interfață serială, cum ar fi I2C sau SPI.

Aceasta permite transmiterea datelor și a comenzilor de control către LCD și primirea informațiilor de stare sau a evenimentelor de la LCD.

4. Controlul afișării: Utilizând comenzile corecte, poți controla modul în care textul este afișat pe LCD 16x2. Acestea includ setarea poziției cursorului, afișarea și ascunderea cursorului, definirea caracterelor personalizate, setarea luminii de fundal și a contrastului etc.

5. Caractere personalizate: LCD-urile 16x2 pot fi configurate pentru a afișa caractere personalizate definite de utilizator. Acest lucru înseamnă că poți crea și afișa propriile simboluri și pictograme pe LCD.

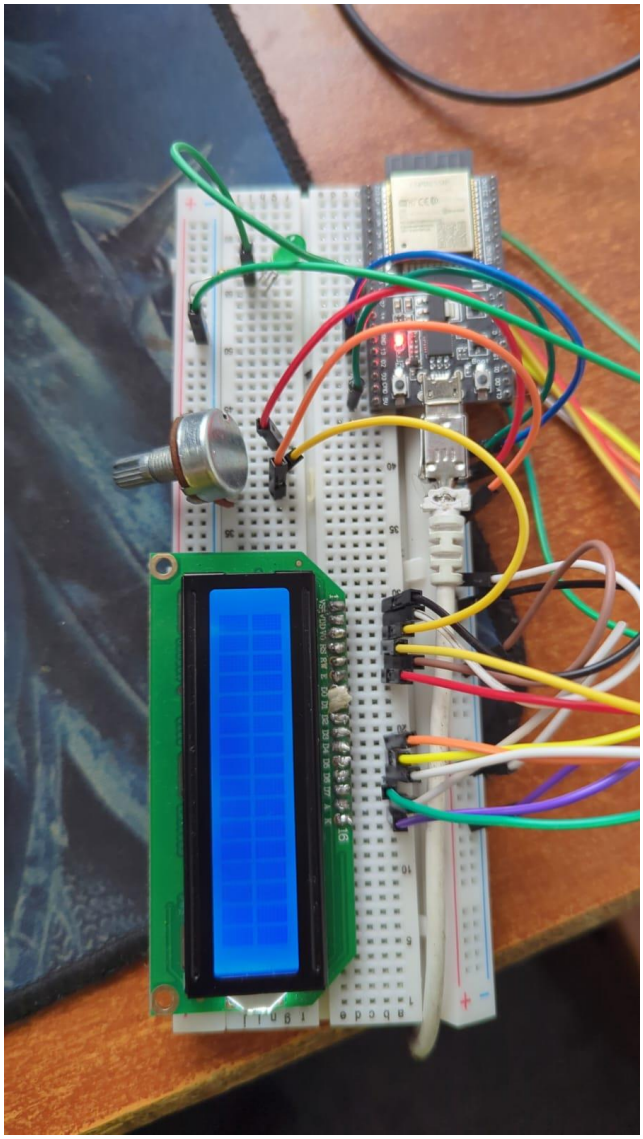
6. Scrolling: Un LCD 16x2 poate avea capacitatea de a derula textul pe ecran în ambele direcții (stânga-dreapta sau dreapta-stânga) pentru a afișa mai mult text decât capacitatea sa statică.

7. Alimentare: LCD-urile 16x2 necesită o sursă de alimentare adecvată pentru a funcționa corect. De obicei, au nevoie de o tensiune de alimentare de 5V sau 3.3V, împreună cu un consum redus de curent.

LCD-urile 16x2 sunt utilizate într-o varietate de aplicații, cum ar fi dispozitivele de control, afișaje de mesaje, contoare, termometre, dispozitive de monitorizare etc., datorită capacității lor de a afișa text și simboluri într-un mod clar și lizibil.

- Potențiometru

Potențiometrul constă în ajustarea nivelului de tensiune sau rezistență într-un circuit electronic. Acest dispozitiv cu trei terminale oferă posibilitatea de a varia valoarea rezistenței într-un interval specific, permițând controlul unei caracteristici specifice a circuitului.

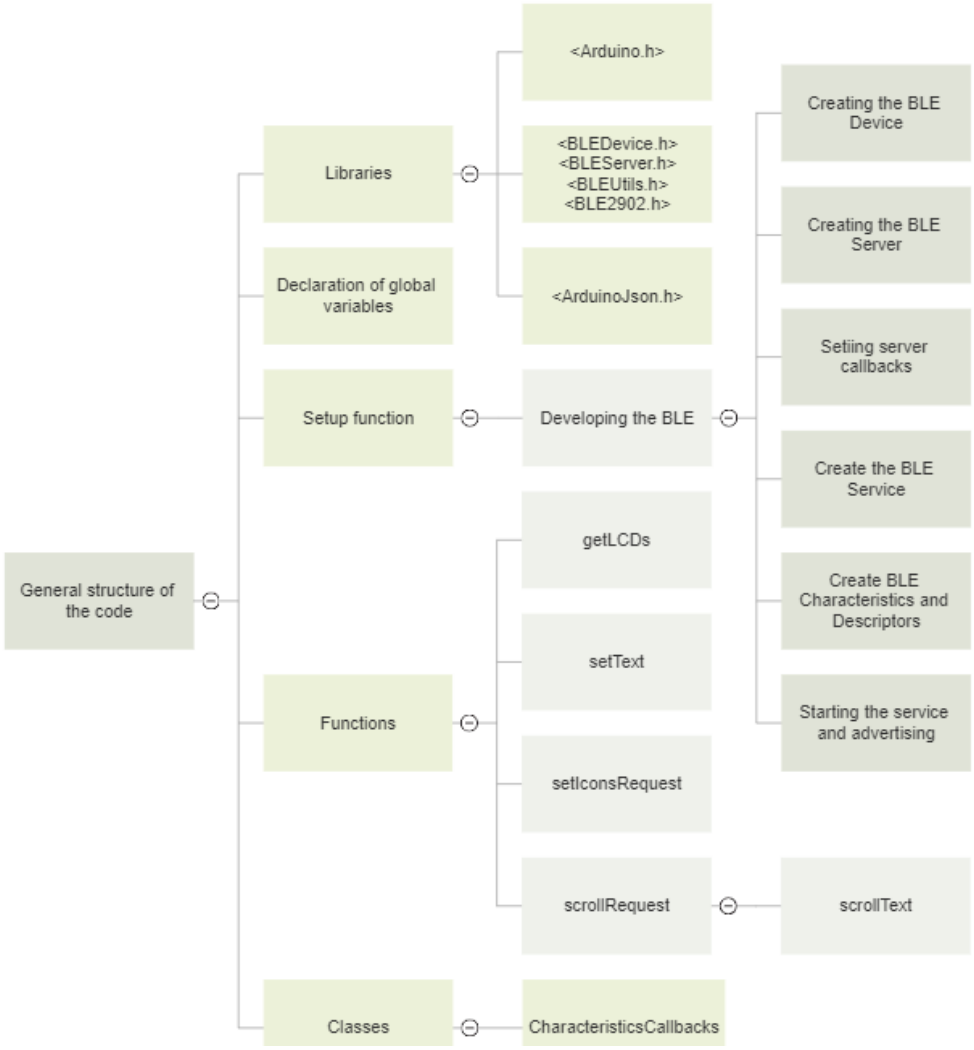


Biblioteca <LiquidCrystal.h>

Biblioteca <LiquidCrystal.h> este o bibliotecă standard pentru platforma Arduino utilizată pentru controlul LCD-urilor alfanumerice, cum ar fi LCD-urile 16x2. Această bibliotecă oferă funcționalități esențiale pentru inițializarea, configurarea și controlul afișajului LCD.

III. Implementare

Organigrama codului:



IV. Concluzii

Pe scurt, controlul unui LCD 16x2 printr-o aplicație mobilă folosind o conexiune Bluetooth Low Energy (BLE) oferă multe beneficii și oportunități:

1.Comenzi convenabile:
Cu Bluetooth Low Energy, LCD-ul 16x2 poate fi controlat printr-o aplicație mobilă, oferind o interfață convenabilă și ușor de utilizat. Acest lucru oferă utilizatorilor flexibilitatea de a opera ecrane LCD folosind dispozitivele lor mobile.

2. Caracteristici avansate:
O aplicație mobilă ne permite să implementăm diverse acțiuni și funcții pe LCD 16x2. Aceasta include afișarea textului, utilizarea simbolurilor personalizate (caractere speciale), derularea textului în ambele direcții etc. Comenzile și datele necesare pentru a controla aceste funcții pot fi trimise pe LCD printr-o conexiune Bluetooth Low Energy.

3. Integrare cu alte caracteristici:
Bluetooth Low Energy permite o mai bună integrare cu alte dispozitive și servicii, permițând sincronizarea cu alte aplicații și transferul de date către alte dispozitive conectate.

4. Flexibilitate și scalabilitate:
Bluetooth Low Energy permite flexibilitate și scalabilitate în implementarea sistemului. Putem folosi această soluție pentru a controla un singur LCD 16x2 sau puteți extinde sistemul pentru a controla mai multe LCD-uri sau alte periferice.

În general, integrarea unui LCD 16x2 cu Bluetooth Low Energy și controlul acestuia folosind o aplicație mobilă oferă multe avantaje în ceea ce privește ușurința în utilizare, scalabilitatea și integrarea cu alte caracteristici. Această soluție oferă un mod convenabil și modern de a opera afișajele LCD în sistemele electronice.

V. Bibliografie

http://proiectia.bogdanflorea.ro/esp32/examples/ble

https://curs.upb.ro/2022/pluginfile.php/487654/mod_resource/content/1/P1.%20Modulul%20ESP32%20%C8%99i%20resursele%20sale.pdf

https://curs.upb.ro/2022/pluginfile.php/487655/mod_resource/content/1/P2.%20Tipuri%20de%20semnale%2C%20protocoale%20de%20comunicatie%20si%20periferice.pdf

https://curs.upb.ro/2022/pluginfile.php/498930/mod_resource/content/2/P3.%20WiFi%2C%20HTTP%2C%20JSON%20si%20Bluetooth.pdf

https://cloud.smartdraw.com/?nsu=1

Mastering ArduinoJson 6 - Third Edition – PDF BOOK

Anexă (codul complet)

```
#include <Arduino.h>
#include <LiquidCrystal.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <ArduinoJson.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <iterator>

#define bleServerName "Bnb Bluetooth"
const int LED_BUILTIN=27;
const int LCD_COLUMNS = 16;
const int LCD_ROWS = 2;
LiquidCrystal lcd(19, 23, 18, 17, 16, 15); // Define LCD display pins RS, E, D4, D5, D6, D7
bool deviceConnected = false;
BLECharacteristic *pcharacteristic;

byte bluetooth[8] = {
  B00100, //4
  B10110, //22
  B01101, //13
  B00110, //6
  B01101, //13
  B10110, //22
  B00100, //4
  B00000  //0
```

```
};
byte favorite[8] = {
  B00000, //0
  B01010, //10
  B11111, //31
  B11111, //31
  B01110, //14
  B00100, //4
  B00000, //0
  B00000  //0
};
byte bolt[8] = {
  B00100, //4
  B01100, //12
  B11100, //28
  B11111, //31
  B00111, //7
  B00110, //6
  B00100, //4
  B00000  //0
};
byte adb[8] = {
  B10001, //17
  B01110, //14
  B10101, //21
  B11111, //31
  B00000, //0
  B11111, //31
  B11111, //31
  B01110  //14
```

```
};
byte notifications[8] = {
  B00100, //4
  B01110, //14
  B01110, //14
  B01110, //14
  B11111, //31
  B00000, //0
  B00100, //4
  B00000,  //0
};
byte hourglass_bottom[8] = {
  B11111, //31
  B10001, //17
  B01010, //10
  B00100, //4
  B01110, //14
  B11111, //31
  B11111, //31
  B00000  //0
};
byte dialpad[8] = {
  B10101, //21
  B00000, //0
  B10101, //21
  B00000, //0
  B10101, //21
  B00000, //0
  B00100, //4
  B00000, //0
```

```
};
byte lock[8] = {
  B01110, //14
  B10001, //17
  B10001, //17
  B11111, //31
  B11111, //31
  B11011, //27
  B11111, //31
  B11111 //31
};

#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
BLECharacteristic dataCharacteristic("beb5483e-36e1-4688-b7f5-ea07361b26a8");

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

// Setup callbacks onConnect and onDisconnect (no change necessary)
class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
    Serial.println("Device connected");
  };
  void onDisconnect(BLEServer* pServer) {
```

```
    deviceConnected = false;
    Serial.println("Device disconnected");
  }
};

std::string teamId="A71";

void getLCDs(BLECharacteristic* pcharacteristic){
  DynamicJsonDocument responseDoc(4096);
  JsonObject responseObj = responseDoc.to<JsonObject>();
  //LCD
  JsonObject lcd = responseObj.createNestedObject("lcd");
  responseObj["type"]="16x2";
  responseObj["interface"]="Parallel 4-bit";
  responseObj["resolution"]="16x2";
  responseObj["id"]=1;
  responseObj["teamId"]=teamId;
  //serialize the response to a JSON String
  String responseString;
  serializeJson(responseDoc, responseString);
  //set the response string to the ble characteristic
  pcharacteristic->setValue(responseString.c_str());
  pcharacteristic->notify();
}

void setText(BLECharacteristic* pcharacteristic, const JsonObject& request){
  int lcdId = request["id"].as<int>();
  JsonArray textArray = request["text"].as<JsonArray>();
  lcd.clear();
  for (int i = 0; i < textArray.size() && i < LCD_ROWS; i++) {
```

```
    String text = textArray[i].as<String>();
    lcd.setCursor(0, i);
    lcd.print(text);
  }
  // Prepare the response object
  DynamicJsonDocument responseDoc(4096);
  JsonObject responseObj = responseDoc.to<JsonObject>();
  responseObj["id"] = lcdId;
  responseObj["text"] = textArray;
  responseObj["teamId"] = teamId;

  // Serialize the response to a JSON string
  String responseString;
  serializeJson(responseDoc, responseString);

  // Set the response string to the BLE characteristic
  pcharacteristic->setValue(responseString.c_str());
  pcharacteristic->notify();
}

void setIconsRequest(BLECharacteristic* pcharacteristic, JsonObject& requestObj) {
  int lcdId = requestObj["id"].as<int>();
  JsonArray iconsArray = requestObj["icons"].as<JsonArray>();

  // Process the icons
  int numIcons = iconsArray.size();
  // Perform the necessary operations to set the icons on the LCD
  lcd.begin(16,2);
  lcd.clear();
```

```
for (const auto& icon : iconsArray) {
  // Extract the icon name and data
  String iconName = icon["name"].as<String>();
  JsonArray iconDataArray = icon["data"].as<JsonArray>();

  // Check the condition to determine which icon to display
  if (iconName == "bluetooth") {
    lcd.createChar(0, bluetooth);
    lcd.setCursor(0, 0);
    lcd.write((byte)0);
  }
  else
    if(iconName == "favorite"){
      lcd.createChar(1, favorite);
      lcd.setCursor(1, 1);
      lcd.write((byte)1);
    }
    else
      if(iconName == "bolt"){
        lcd.createChar(2, bolt);
        lcd.setCursor(2,0);
        lcd.write((byte)2);
      }
      else
        if(iconName == "adb"){
          lcd.createChar(3, adb);
          lcd.setCursor(3,1);
          lcd.write((byte)3);
        }
        else
```

```
        if(iconName == "notifications"){
          lcd.createChar(4, notifications);
          lcd.setCursor(4,0);
          lcd.write((byte)4);
        }
        else
          if(iconName == "hourglass-bottom"){
            lcd.createChar(5, hourglass_bottom);
            lcd.setCursor(5,1);
            lcd.write((byte)5);
          }
          else
            if(iconName == "dialpad"){
              lcd.createChar(6, dialpad);
              lcd.setCursor(6,0);
              lcd.write((byte)6);
            }
            else
              if(iconName == "lock"){
                lcd.createChar(7, lock);
                lcd.setCursor(7,1);
                lcd.write((byte)7);
              }
}
// Prepare the response
DynamicJsonDocument responseDoc(4096);
JsonObject responseObj = responseDoc.to<JsonObject>();
responseObj["id"] = lcdId;
responseObj["number_icons"] = numIcons;
responseObj["teamId"] = teamId;
```

```
  // Serialize the response to a JSON string
  String responseString;
  serializeJson(responseDoc, responseString);

  // Send the response to the app
  pcharacteristic->setValue(responseString.c_str());
  pcharacteristic->notify();
}

bool isScrolling = false;
bool scrollDirectionLeft = true;

void scrollText() { {
    if (scrollDirectionLeft) {
      lcd.scrollDisplayLeft();
    } else {
      lcd.scrollDisplayRight();
    }
    delay(100);
  }
}
void scrollRequest(BLECharacteristic *pcharacteristic, JsonObject &requestObj){
  int lcdId = requestObj["id"].as<int>();
  String scrollDirection = requestObj["direction"].as<String>();

  // Process the scroll direction
  if (scrollDirection == "Left") {
    // Start scrolling the text
    scrollDirectionLeft = true;
```

```
    isScrolling = true;
  } else if (scrollDirection == "Right") {
    // Start scrolling the text
    scrollDirectionLeft = false;
    isScrolling = true;
  } else if (scrollDirection == "Off") {
    // Stop scrolling the text
    isScrolling = false;
  }

  // Prepare the response
  DynamicJsonDocument responseDoc(4096);
  JsonObject responseObj = responseDoc.to<JsonObject>();
  responseObj["id"] = lcdId;
  responseObj["scrolling"] = scrollDirection;
  responseObj["teamId"] = teamId;

  // Serialize the response to a JSON string
  String responseString;
  serializeJson(responseObj, responseString);

  // Send the response to the app
  pcharacteristic->setValue(responseString.c_str());
  pcharacteristic->notify();
}

class CharacteristicsCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pcharacteristic) {
      std::string value = pcharacteristic->getValue();
```

```
    if (value.length() > 0) {
    // Parse the received JSON request
      DynamicJsonDocument requestDoc(4096);
      DeserializationError error = deserializeJson(requestDoc, value.c_str());

      if (error) {
        Serial.print("JSON parsing error: ");
        Serial.println(error.c_str());
        return;
      }
      JsonObject requestObj = requestDoc.as<JsonObject>();
      String action = requestObj["action"].as<String>();
      if (action == "getLCDs") {
        getLCDs(pcharacteristic);
      }
      else
        if (action == "setText") {
          setText(pcharacteristic, requestObj);
        }
        else
          if(action == "setIcons"){
            setIconsRequest(pcharacteristic, requestObj);
          }
          else
            if(action == "scroll"){
              scrollRequest(pcharacteristic, requestObj);
            }
    }
  }
};
```

```
void setup() {
  // Start serial communication
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);
  // BEGIN DON'T CHANGE
  // Create the BLE Device
  BLEDevice::init(bleServerName);

  // Create the BLE Server
  BLEServer *pServer = BLEDevice::createServer();
  // Set server callbacks
  pServer->setCallbacks(new MyServerCallbacks());

  // Create the BLE Service
  BLEService *bleService = pServer->createService(SERVICE_UUID);

  // Create BLE characteristics and descriptors
  bleService->addCharacteristic(&characteristic);
  characteristic.addDescriptor(characteristicDescriptor);

  // Set characteristic callbacks
  characteristic.setCallbacks(new CharacteristicsCallbacks());

  // Start the service
  bleService->start();
  // Start advertising
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pServer->getAdvertising()->start();
  Serial.println("Waiting a client connection to notify...");
```

```
  // END DON'T CHANGE
}

void loop() {
  if (isScrolling) {
    scrollText();
  }
}
```