**Documentație –Proiect Informatică Aplicată**

1. **Prezentare generală**

Scopul proiectului este realizarea unui program ce faciliteaza conectarea telefonului, având instalată aplicația mobilă, prin Bluetooth Low Energy (BLE) la modulul ESP32, conectarea modulului la o retea WI-FI prin intermediul aplicației si afișarea unei liste de Board Games (jocuri de societate) cu detaliile aferente prin interacțiunea cu un server web, utilizând format JSON pentru schimb de date.

Obiective principale:  
- Configurarea și utilizarea modulului ESP32  
- Implementarea unui serviciu GATT BLE cu o caracteristică JSON  
- Scanarea și conectarea la rețele Wi-Fi  
- Realizarea de cereri HTTP GET către un API web  
- Serializare/ deserializare eficientă JSON cu ArduinoJson  
- Gestionarea corectă a comunicării asincrone pentru a evita blocările și deconectările

1. **Considerente 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. Pentru realizarea proiectului nostru nu a fost necesara programarea pinilor, ci s-a utilizat doar comunicatia WiFi si Bluetooth clasic.

**Wi-Fi**

ESP32 implementează TCP/IP, protocolul complet 802.11 b/g/n/e/i WLAN MAC și specificația Wi-Fi Direct. Aceasta înseamnă că ESP 32 poate comunica cu majoritatea routerelor WiFi de acolo atunci când este utilizat în modul client (in engleza: *station/client mode*). De asemenea, poate crea un punct de acces cu 802.11 b/g/n/e/i complet.

ESP32 acceptă și Wi-Fi Direct. Wifi-Direct este o opțiune bună pentru conexiunea *peer-to-peer* fără a fi nevoie de un punct de acces. Este folosit deoarece este mai ușor de configurat și vitezele de transfer de date sunt mult mai mari decât bluetooth. Acest lucru ar putea fi utilizat pentru a configura proiecte bazate pe ESP32 de pe un telefon/tabletă care acceptă WiFi direct.

**Bluetooth**

ESP32 accepta atat versiuni recente de bluetooth (BLE Bluetooth 4.2), cat si Bluetooth clasic; cu alte cuvinte, poate comunica cu dispozitive bluetooth vechi si noi deopotriva. Acest lucru prezinta un avantaj enorm daca dorim sa realizam un program care trebuie sa fie compatibil cu orice tip de telefon sau tableta, indiferent de vechimea sa. BLE este un protocol de joasă putere în banda 2.4 GHz, cu consum redus și MTU ~512 B. Folosește GAP pentru advertising și conectare, și GATT pentru servicii/caracteristici.

**JSON & ArduinoJson**

JSON este un format text pentru obiecte și array-uri. ArduinoJson oferă `deserializeJson()` și `serializeJson()` cu document static sau dinamic.

**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, însă poate fi folosit și pentru alte tipuri de aplicații.

Protocolul HTTP definește un set de metode pentru cereri pentru a indica acțiunea ce va fi efectuată asupra unei anumite resurse. Aceste metode se mai numesc și verbe HTTP, fiecare metodă implementând semantică diferită. Metoda folosita in acest proiect este:

GET – Metoda GET reprezintă o cerere a unei reprezentări (imagini) a unei anumite resurse. Cererile transmise prin metoda GET ar trebui folosite pentru obținerea datelor de la un server (exemplu: o pagina web, o imagine, un fișier, etc.).

1. **Arhitectura software**

**1.** BLE onWrite callback: primește JSON, deserializare în “pendingReq”, returnează rapid.  
**2.** Structură Request:  
   
 enum ActionType { NONE, GET\_NET, CONNECT, GET\_DATA, GET\_DETAILS };  
 struct Request { ActionType action; String param1, param2; } pendingReq;  
   
**3.** loop(): verifică “pendingReq.action”, apelează handler, apoi resetează.  
**4.** Handleri:  
- “doGetNetworks()”: scanare Wi-Fi → JSON per rețea → “notify()”.  
- “doConnect()”: Wi-Fi.begin + timeout → JSON de confirmare.  
- “doGetData()”: HTTP GET “/games” → JSON minim pentru fiecare joc.  
- “doGetDetails()”: HTTP GET “/game?id=” → construire “description” → JSON.

1. **Prezentarea codului cu comentarii detaliate**

Codul complet cu comentarii explicative:

#include <Arduino.h>            // Include biblioteca Arduino: oferă funcții de bază (setup, loop, Serial, etc.)

#include <WiFi.h>               // Include controlul modulului Wi-Fi pe ESP32

#include <HTTPClient.h>         // Include clientul HTTP pentru a face cereri GET/POST

#include <ArduinoJson.h>        // Include biblioteca ArduinoJson pentru serializare/deserializare JSON

#include <BLEDevice.h>          // Include inițializarea și operațiile de bază Bluetooth Low Energy

#include <BLEServer.h>          // Include crearea unui server BLE (GATT)

#include <BLEUtils.h>           // Include utilitare BLE, de ex. pentru lucrul cu UUID-uri

#include <BLE2902.h>            // Include descriptorul care permite notificări pe caracteristici BLE

// --- BLE & API setup ---

#define SERVICE\_UUID        "12345678-1234-1234-1234-123456789abc"

                                // UUID unic pentru serviciul BLE expus de ESP32

#define CHARACTERISTIC\_UUID "abcdefab-cdef-cdef-cdef-abcdefabcdef"

                                 // UUID unic pentru caracteristica BLE prin care comunicăm

const char\* API\_BASE = "http://proiectia.bogdanflorea.ro/api/board-game-atlas";

                                 // URL de bază al API-ului web care listează board games

// --- Pending request struct ---

enum ActionType { NONE, GET\_NET, CONNECT, GET\_DATA, GET\_DETAILS };

                                 // Reprezintă tipurile de cereri primite de la aplicația mobilă

struct Request {

  ActionType action = NONE;      // Tipul cererii (inițial NONE)

  String param1;                 // Primul parametru: teamId, ssid sau id

  String param2;                 // Al doilea parametru: parola Wi-Fi (doar pentru CONNECT)

} pendingReq;                    // Instanță globală care reține cererea în așteptare

// --- Globals ---

BLECharacteristic\* pCharacteristic;

                                 // Pointer la caracteristica BLE folosită pentru read/write/notify

String currentTeam;              // Reține teamId-ul curent pentru toate răspunsurile

// --- Helper: serialize & notify ---

void sendNotify(JsonDocument& doc) {

  String payload;                // Buffer pentru JSON-ul serializat

  serializeJson(doc, payload);   // Transformă `doc` într-un șir JSON și îl pune în `payload`

  pCharacteristic->setValue(payload.c\_str());

                                 // Încarcă textul JSON în caracteristica BLE

  pCharacteristic->notify();     // Trimite o notificare (push) către clientul mobil

  delay(20);                     // Mic delay pentru a asigura trimiterea completă

}

// --- Queue a request from BLE callback ---

void queueRequest(ActionType a, const String& p1 = "", const String& p2 = "") {

  pendingReq.action = a;         // Salvează tipul cererii

  pendingReq.param1 = p1;        // Salvează primul parametru

  pendingReq.param2 = p2;        // Salvează al doilea parametru

}

// --- Handlers executed in loop() ---

void doGetNetworks() {

  currentTeam = pendingReq.param1;  // Preia teamId din cererea pusă în coadă

  int n = WiFi.scanNetworks();      // Scanează rețele Wi-Fi disponibile

  for (int i = 0; i < n; ++i) {      // Pentru fiecare rețea găsită:

    StaticJsonDocument<256> doc;     // Creează un document JSON mic

    doc["ssid"]       = WiFi.SSID(i);              // Numele rețelei

    doc["strength"]   = WiFi.RSSI(i);              // Puterea semnalului

    doc["encryption"] = (int)WiFi.encryptionType(i);// Tipul de criptare

    doc["teamId"]     = currentTeam;               // Echipa care a cerut

    sendNotify(doc);                                // Trimite JSON-ul prin BLE

  }

}

void doConnect() {

  const char\* ssid = pendingReq.param1.c\_str();    // SSID-ul din param1

  const char\* pass = pendingReq.param2.c\_str();    // Parola din param2

  WiFi.begin(ssid, pass);                          // Începe conectarea la rețeaua Wi-Fi

  unsigned long start = millis();                  // Reține momentul de start

  bool ok = false;                                 // Flag pentru succes

  while (millis() - start < 10000) {               // Așteaptă până la 10 secunde

    if (WiFi.status() == WL\_CONNECTED) { ok = true; break; }

    delay(100);                                    // Pauză mică între verificări

  }

  StaticJsonDocument<256> doc;                     // Document JSON de răspuns

  doc["ssid"]      = String(ssid);                 // SSID

  doc["connected"] = ok;                           // True dacă s-a conectat

  doc["teamId"]    = currentTeam;                  // Echipa curentă

  sendNotify(doc);                                 // Trimite răspunsul prin BLE

}

void doGetData() {

  if (WiFi.status() != WL\_CONNECTED) return;      // Dacă nu suntem conectați, ieșim imediat

  HTTPClient http;                                // Creăm un client HTTP

  http.begin(String(API\_BASE) + "/games");        // Deschidem conexiunea la endpoint-ul /games

  if (http.GET() == 200) {                        // Dacă serverul răspunde cu 200 OK

    String pl = http.getString();                 // Citim tot răspunsul JSON ca text

    DynamicJsonDocument arr(20\*1024);             // Document JSON destul de mare pentru array

    if (!deserializeJson(arr, pl)) {              // Deserializeaza textul în structuri C++

      for (JsonObject item : arr.as<JsonArray>()) {  // Pentru fiecare obiect din array

        StaticJsonDocument<512> doc;              // Document JSON pentru trimitere

        doc["id"]     = item["id"].as<const char\*>();        // ID-ul jocului

        doc["name"]   = item["name"].as<const char\*>();      // Numele jocului

        doc["image"]  = item["image\_url"].as<const char\*>(); // URL-ul imaginii

        doc["teamId"] = currentTeam;                         // Echipa curentă

        sendNotify(doc);                                    // Trimite JSON-ul

      }

    }

  }

  http.end();                                     // Închide conexiunea HTTP

}

void doGetDetails() {

  if (WiFi.status() != WL\_CONNECTED) return;      // Ieșim dacă nu avem Wi-Fi

  String id = pendingReq.param1;                  // ID-ul jocului din coadă

  HTTPClient http;                                // Client HTTP nou

  http.begin(String(API\_BASE) + "/game?id=" + id);// Cerere GET cu parametru id

  if (http.GET() == 200) {                        // Dacă răspunsul e OK

    String pl = http.getString();                 // Citim textul JSON

    DynamicJsonDocument in(20\*1024);              // Document pentru deserializare

    if (!deserializeJson(in, pl)) {               // Deserializam JSON-ul

      JsonObject src = in.as<JsonObject>();       // Obiect JSON sursă

      StaticJsonDocument<1024> doc;               // Document JSON pentru răspuns

      doc["id"]    = src["id"].as<const char\*>();       // ID

      doc["name"]  = src["name"].as<const char\*>();     // Nume

      doc["image"] = src["image\_url"].as<const char\*>();// Imagine

      // Construim linie cu preț, jucători, timp și an

      String price    = String("$") + src["price"].as<const char\*>();

      String players  = String(src["players"].as<const char\*>());

      String playtime = String(src["playtime"].as<const char\*>());

      String year     = String(src["year\_published"].as<int>());

      String desc = "Price: "    + price   + "\n"

                  + "Players: "  + players + "\n"

                  + "Play time: "+ playtime+ "\n"

                  + "Year: "     + year;

      doc["description"] = desc;                 // Atribuim descrierea

      doc["teamId"]      = currentTeam;          // Echipa curentă

      sendNotify(doc);                           // Trimitem JSON-ul final

    }

  }

  http.end();                                    // Închide conexiunea HTTP

}

// --- BLE callback: queue the request, return immediately ---

class CB : public BLECharacteristicCallbacks {

  void onWrite(BLECharacteristic\* c) override {

    String s = c->getValue();                    // Textul JSON trimis de telefon

    StaticJsonDocument<1024> r;                  // Document pentru parsare

    if (!deserializeJson(r, s)) {                // Deserializam textul JSON

      String a = r["action"].as<String>();       // Extragem câmpul "action"

      // Alegem handler-ul potrivit în funcție de valoarea acțiunii

      if      (a == "getNetworks") queueRequest(GET\_NET,    r["teamId"].as<String>());

      else if (a == "connect")     queueRequest(CONNECT,    r["ssid"].as<String>(), r["password"].as<String>());

      else if (a == "getData")     queueRequest(GET\_DATA);

      else if (a == "getDetails")  queueRequest(GET\_DETAILS, r["id"].as<String>());

    }

  }

};

// --- Setup & loop ---

void setup() {

  Serial.begin(115200);                         // Începem comunicarea prin portul serial

  WiFi.mode(WIFI\_STA);                          // Setăm ESP32 ca stație Wi-Fi

  BLEDevice::init("ESP32-BoardGameAPI");        // Inițializăm BLE cu numele device-ului

  BLEServer\*  srv = BLEDevice::createServer();  // Creăm un server BLE

  BLEService\* svc = srv->createService(SERVICE\_UUID); // Adăugăm serviciul GATT cu UUID-ul definit

  // Cream caracteristica cu permisiuni de citire, scriere și notificare

  pCharacteristic = svc->createCharacteristic(

    CHARACTERISTIC\_UUID,

    BLECharacteristic::PROPERTY\_READ  |

    BLECharacteristic::PROPERTY\_WRITE |

    BLECharacteristic::PROPERTY\_NOTIFY

  );

  pCharacteristic->addDescriptor(new BLE2902()); // Adăugăm descriptorul necesar pentru notify

  pCharacteristic->setCallbacks(new CB());       // Setăm callback-ul pentru scriere

  svc->start();                                  // Pornim serviciul BLE

  BLEDevice::getAdvertising()->addServiceUUID(SERVICE\_UUID); // Facem advertising

  BLEDevice::getAdvertising()->start();          // Începem advertising-ul

  Serial.println("BLE ready");                   // Afișăm mesaj de stare

}

void loop() {

  // Verificăm ce cerere e în coadă și apelăm handler-ul respectiv

  switch(pendingReq.action) {

    case GET\_NET:     doGetNetworks();   break;

    case CONNECT:     doConnect();       break;

    case GET\_DATA:    doGetData();       break;

    case GET\_DETAILS: doGetDetails();    break;

    default: break;                       // Dacă nu e nicio cerere, nu facem nimic

  }

  pendingReq.action = NONE;                 // Resetăm coada după procesare

  delay(50);                                // Mic delay pentru stabilitate

}

1. **Concluzii**

Pe parcursul acestui proiect am dobândit competențe esențiale legate de utilizarea modulului ESP32, în special privind comunicarea asincronă prin Bluetooth Low Energy și gestionarea conexiunilor Wi-Fi și a cererilor HTTP. Am înțeles importanța unui design asincron, care a permis evitarea blocărilor și deconectărilor neprevăzute, oferind stabilitate și fluiditate comunicației dintre ESP32 și aplicația mobilă. De asemenea, am fost nevoit să adaptăm mesajele JSON la limitările hardware ale protocolului BLE, impunându-se optimizări și trunchieri pentru a respecta limita standard a MTU de aproximativ 512 octeti.  
  
Prin intermediul bibliotecii ArduinoJson, am explorat diverse tehnici eficiente de serializare și deserializare a datelor, simplificând semnificativ procesul de comunicare și manipulare a informațiilor structurate. În plus, debugging-ul aprofundat cu ajutorul monitorului serial și a logurilor detaliate mi-a oferit o perspectivă clară asupra modului în care datele circulă și mi-a permis să identificăm rapid și să remediez problemele întâlnite.

1. **Bibliografie**

1. P1: Modulul ESP32   
2. P2: Tipuri de semnale și protocoale de comunicație  
3. P3: Wi-Fi, HTTP, JSON și Bluetooth  
4. ArduinoJson Documentation