



PROIECT FINAL IA

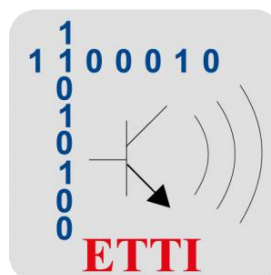
Intocmit de către:

Frunză Vladimir

Prefac Raluca

Facultatea de Electronică, Telecomunicații si Tehnologia Informației

Titular disciplina: Lect. Dr. Florea Bogdan Cristian



Sumar

1. Introducere

2. Considerente teoretice:

- Modulul ESP32: o scurta prezentare
- Protocoalele si metodele de comunicatie utilizate (Bluetooth Low Energy, WiFi, HTTP si JSON)

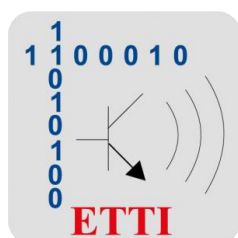
3. Implementare:

- Organigrama codului si explicatii
- Demonstratie vizuala a functionalitatii codului

4. Concluzii

5. Bibliografie

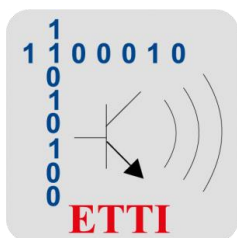
6. Anexa: codul complet



I. Introducere

Acest proiect final al materiei PIA (Proiect – Informatica Aplicata) presupune implementarea unui cod, folosind placa ESP32 proiectata de compania Espressif Systems pe care am primit-o la inceputul semestrului si scris in limbajul C++, care sa se foloseasca de capacitatea placutei de a se conecta la internet si la Bluetooth pentru a interoga un API (Application Programming Interface) si a comunica cu un dispozitiv mobil prin intermediul aplicatiei “ProiectIA” instalata prin intermediul site-ului <http://proiectia.bogdanflore.ro/>.

In primul rand, pentru conectarea placutei la internet se vor folosi bibliotecile WiFi si WiFiClient, iar pentru interogarea API-ului se vor folosi interogari http facilitate de biblioteca HTTPClient. De asemenea, conectarea la aplicatie se va face folosind Bluetooth Low Energy, iar datele transmise vor fi de tip JSON. Toate aceste concepte vor fi explicate in urmatorul subpunct al referatului.

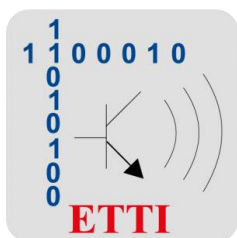


2. Considerente teoretice

2.1. O scurta prezentare a placutei 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:

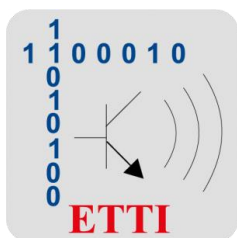
- 34 pini programabili GPIO (General Purpose Input/Output)
- 18 canale de conversie analog-digitală (ADC) cu rezoluție de 12 biți
- 2 canale de conversie digital-analogică (DAC) cu rezoluție de 8 biți
- 16 canale de ieșire PWM (Pulse Width Modulation)
- 10 senzori interni capacitivi
- 3 interfețe SPI (Serial Peripheral Interface)



- 3 interfețe UART (Universal Asynchronous Receiver-Transmitter)
- 2 interfețe I2C (Inter-Integrated Circuit)
- 2 interfețe I2 S (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. În Fig. 1 este prezentată diagrama unei plăci de dezvoltare cu 38 de pini, placă ce va fi folosită în cadrul acestui proiect.

Numerotarea pinilor de pe placa de dezvoltare este realizată în funcție de denumirea internă a pinilor modulului (GPIOxx). Placa dispune de un LED integrat care este conectat la pinul general de intrare/ieșire GPIO02, care poate fi accesat pe placă pe pinul G2.



 PWM
 PIN NUMBER
 NAME
 GROUND
 POWER
 CONTROL
 I/O
 ADC
 COMM. INTERFACE
 DAC
 I2C
 HS
 TOUCH

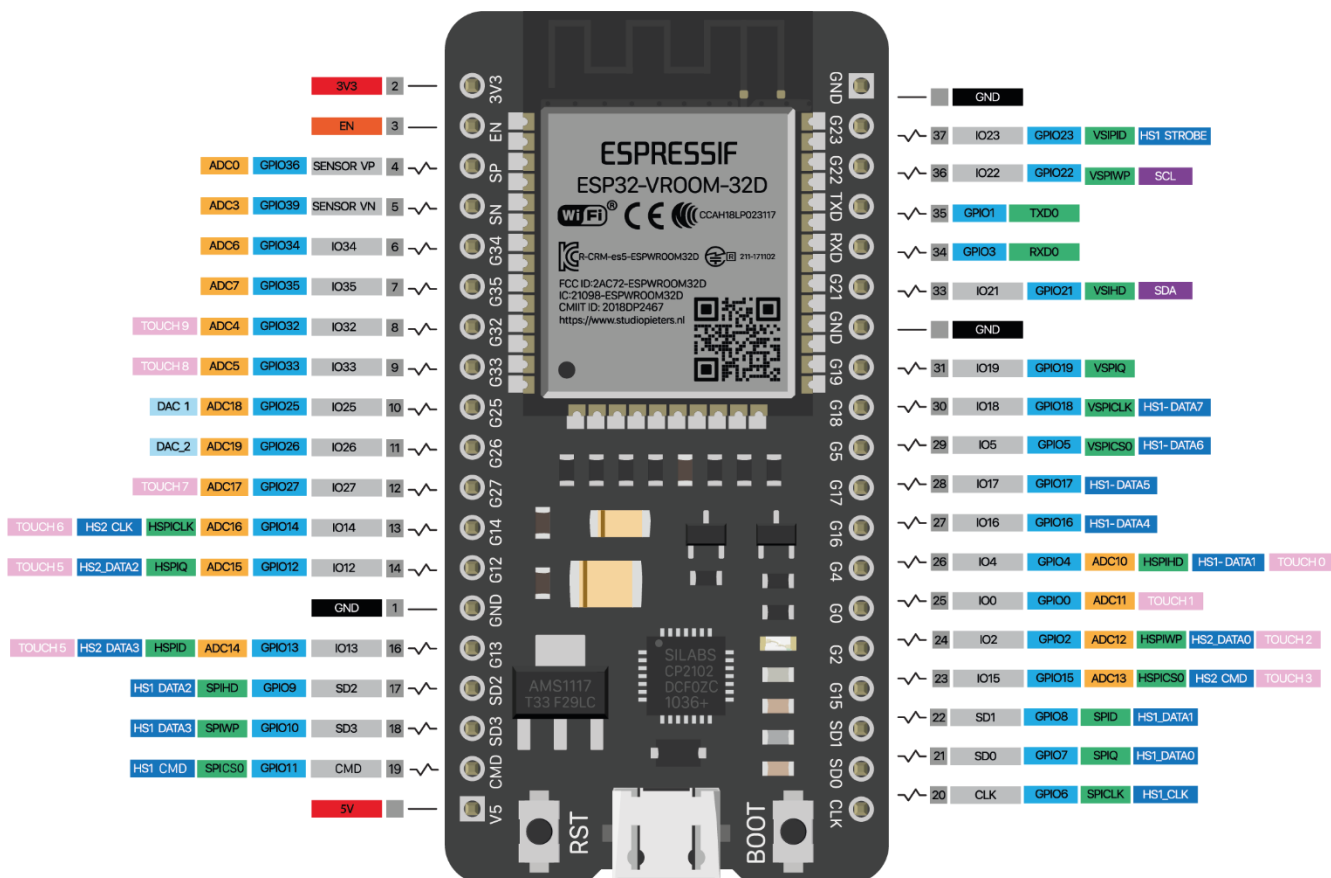


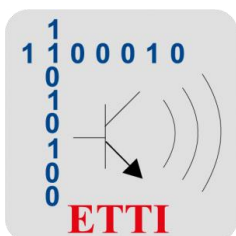
Fig. I: Diagrama unei placi de dezvoltare ESP32 cu 38 de pini

2.2. Explicarea termenilor teoretici

I. Protocolul Wi-Fi

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

Plăcuța ESP32 implementează standardele IEEE 802.11 b/g/n/e/i, oferindu-i posibilitatea să se conecteze la majoritatea dispozitivelor Wi-Fi moderne.



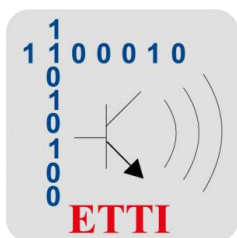
II. Biblioteca WiFi

Această bibliotecă permite conexiunea la rețea (locală și Internet) folosind scutul WiFi Arduino.

Cu această bibliotecă puteți instanția Servere, Clienți și puteți trimite/primi pachete UDP prin WiFi. Scutul se poate conecta fie la rețele deschise, fie criptate (WEP, WPA). Adresa IP poate fi atribuită static sau printr-un DHCP. Biblioteca poate gestiona și DNS.

Cu Arduino WiFi Shield, această bibliotecă permite unei plăci Arduino să se conecteze la internet. Poate servi fie ca server care acceptă conexiuni de intrare, fie ca client care face cele de ieșire. Biblioteca acceptă criptarea WEP și WPA2 Personal, dar nu și WPA2 Enterprise. De asemenea, rețineți că dacă SSID-ul nu este difuzat, scutul nu se poate conecta. În cod este folosită și bibliotecă WiFiClient, o bibliotecă care, în linii mari, are aceeași sarcină ca și bibliotecă WiFi. Pentru a utiliza aceste biblioteci, apelăm:

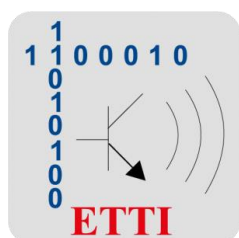
```
#include <WiFi.h>  
#include <WiFiClient.h>
```



III. Bluetooth si protocolul Bluetooth Low Energy

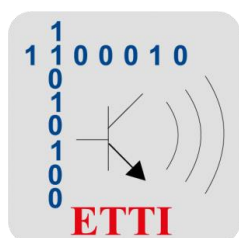
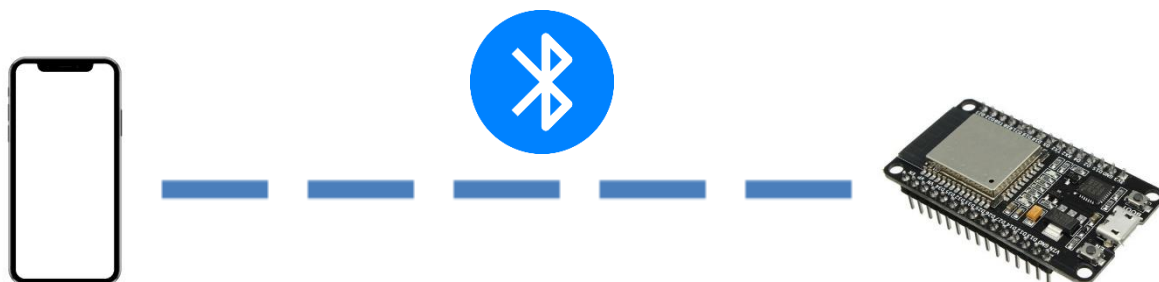
Bluetooth este un set de specificații (un standard) pentru o rețea personală (engleză: *personal area network*, PAN) fără fir (*wireless*), bazată pe unde radio. Bluetooth mai este cunoscut ca și standardul IEEE 802.15.1. Prin tehnologia Bluetooth se elimină firele și cablurile între dispozitive atât staționare cât și mobile, facilitează atât comunicațiile de date cât și pe cele vocale și oferă posibilitatea implementării unor rețele ad-hoc și a sincronizării între diverse dispozitive.

„Bluetooth” este o traducere în engleză a cuvântului scandinav Blåtand/Blåtann, cum era supranumit regele viking Harald I al Danemarcei din secolul X. Harald I era renumit ca fiind foarte comunicativ, unul dintre scopurile sale era să determine oamenii să comunice între ei și în timpul domniei sale Danemarca și Norvegia au fost unite. Logoul Bluetooth își are originile tot în istoria nordică. Pictograma este o combinație a inițialelor regelui Harald, Hagal și Bjarkan. Aceste simboluri provin dintr-un alfabet runic, folosit încă din secolul al IX-lea.



Bluetooth Low Energy (colocvial **BLE**, comercializat anterior ca **Bluetooth Smart**) este o tehnologie de rețea personală fără fir, concepută și comercializată de Bluetooth Special Interest Group (Bluetooth SIG), care vizează aplicații noi în domeniul sănătății, industriile de fitness, balize, securitate și divertisment la domiciliu. Este independent de Bluetooth Classic și nu are compatibilitate, dar Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) și LE pot coexista. Specificația originală a fost dezvoltată de Nokia în 2006 sub numele Wibree, care a fost integrat în Bluetooth 4.0 în decembrie 2009 ca Bluetooth Low Energy.

În comparație cu Bluetooth Classic, 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. Sistemele de operare mobile, inclusiv iOS, Android, Windows Phone și BlackBerry, precum și macOS, Linux, Windows 8, Windows 10 și Windows 11, acceptă în mod nativ Bluetooth Low Energy.



IV. Cele 4 biblioteci de implementare BLE

Pentru conecta placuta la Bluetooth Low Energy, vom folosi 4 biblioteci ce fac parte din acelasi „pachet” creat special pentru ESP32:

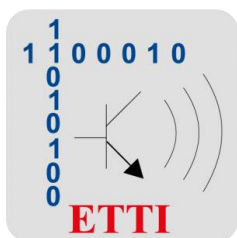
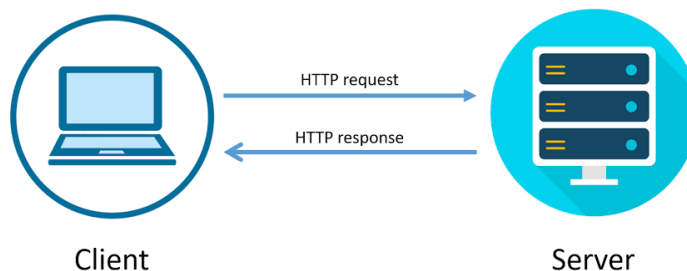
```
#include <BLEDevice.h> //biblioteca BLEDevice  
#include <BLEServer.h> //biblioteca BLEServer  
#include <BLEUtils.h> //biblioteca BLEUtils  
#include <BLE2902.h> //biblioteca BLE2902
```

Nu exista foarte multa documentatie care sa intre in detaliu si sa explice aceste biblioteci, insa ele fac, practic, aceleasi sarcini precum ArduinoBLE, biblioteca compatibila cu alte tipuri de placi precum Arduino MKR WiFi 1010, Arduino UNO WiFi Rev.2, Arduino Nano 33 IoT etc.

V. HTTP requests si biblioteca HTTPClient

HTTP definește un set de metode de solicitare pentru a indica acțiunea dorită care trebuie efectuată pentru o anumită resursă. Deși pot fi și substantive, aceste metode de solicitare sunt uneori denumite *verbe HTTP*. Fiecare dintre ele implementează o semantică diferită, dar unele caracteristici comune sunt împărtășite de un grup de ei: de exemplu, o metodă de solicitare poate fi sigură, idempotent, sau stocabilă în cache.

O solicitare HTTP este făcută de un client, către o gazdă numită, care se află pe un server. Scopul cererii este de a accesa o resursă de pe server. Pentru a face cererea, clientul folosește componente ale unui URL (Uniform Resource Locator), care include informațiile necesare pentru a accesa resursa. Componentele unui URL explică adresele URL.



O solicitare HTTP compusă corect conține următoarele elemente:

- O linie de solicitare.
- O serie de anteturi HTTP sau câmpuri de antet.
- Un corp de mesaj, dacă este necesar.

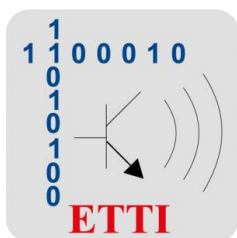
Fiecare antet HTTP este urmat de un carriage return line feed (CRLF). După ultimul antet HTTP, se folosește un CRLF suplimentar (pentru a da o linie goală) și apoi începe orice corp de mesaj.

HTTPClient este o bibliotecă cu ajutorul careia se pot face cu ușurință cereri HTTP GET, POST și PUT către un server web.

Funcționează cu orice clasă derivată de la Client - deci comutarea între Ethernet, WiFi și GSMClient necesită modificări minime de cod.

Pentru apelare, folosim:

```
#include <HTTPClient.h>
```

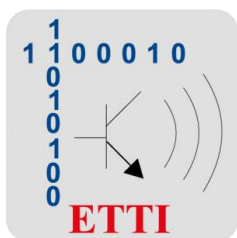


VI. Date de tip JSON si biblioteca ArduinoJson

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 (ECMA-262 3rd Edition), fiind utilizat alături de acest limbaj. Formatul JSON a fost creat de Douglas Crockford și standardizat prin RFC 4627. Tipul de media pe care trebuie să îl transmită un document JSON este *application/json*. Extensia fișierelor JSON este *.json*.

Pentru a prelucra și a transmite date de tip JSON folosim populara bibliotecă ArduinoJson. Aceasta acceptă: serializare, deserializare, MessagePack, alocare fixă, zero-copy, stream-uri, filtrare și multe altele. Pentru a o apela folosim:

```
#include <ArduinoJson.h>
```



3. Implementare

3.1. Organigrama codului

Definirea
tuturor
variabilelor

Definirea claselor
de verificare
a conexiunii BLE
si fetch data BLE

Definirea clasei
Constant ce are drept
atribute toate campurile
ce trebuie afisate si
crearea unui obiect

Definirea claselor
ce se ocupa cu
prelucrarea si
transmiterea
JSON-ului

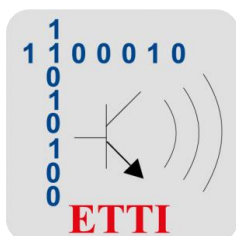
Conectarea
placutei
la Wi-Fi

Conectarea
placutei
la Bluetooth

Apelarea functiilor
definite in clasele
din antet pentru a
interoga API-ul, a
prelua JSON-ul si a
transmite datele
aplicatiei

Empty

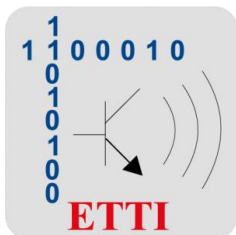
- antet - setup() - loop()



3.2. Explicatii

Incluziunea bibliotecilor necesare si declararea variabilelor globale:

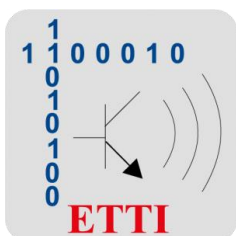
```
#include <Arduino.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <ArduinoJson.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <WiFiClient.h>
#define SERVICE_UUID      "4fafc201-1fb5-459e-8fcc-  
c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-  
ea07361b26a8"
const char* ssid      = "placeholder_ssid";
const char* password  = "placeholder_password";
bool deviceConnected = false;
```



Crearea a doua BLECharacteristics cu 3 proprietati (Read, Write si Notify). Acestea sunt niste „containere” ce stocheaza si prelucreaza o valoare. De asemenea, ele pot fi expuse de server si in ele se poate scrie sau pot fi citite de catre client:

```
BLECharacteristic indexCharacteristic(  
    "bc4196e1-05fd-4bcc-bfa7-7e2e00d03bba",  
    BLECharacteristic::PROPERTY_READ |  
    BLECharacteristic::PROPERTY_WRITE |  
    BLECharacteristic::PROPERTY_NOTIFY  
);
```

```
BLECharacteristic detailsCharacteristic(  
    "385e7304-06a1-4235-ba49-d084db02fca9",  
    BLECharacteristic::PROPERTY_READ |  
    BLECharacteristic::PROPERTY_WRITE |  
    BLECharacteristic::PROPERTY_NOTIFY  
);
```



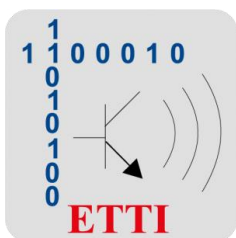
Crearea a 2 obiecte descriptor BLE. Acestia fac exact ce spune numele, descriind un periferic BLE. In ele se pot scrie date si din ele se pot citi date:

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

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

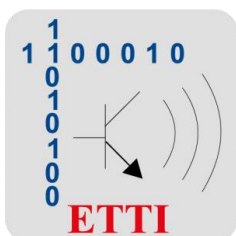
Definirea clasei ce verifica si afiseaza daca exista un dispozitiv Bluetooth conectat la placa:

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



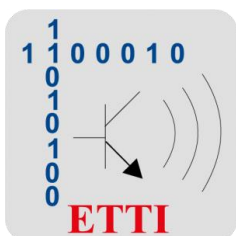
Crearea clasei Constant ce contine attributele corespunzatoare campurilor din JSON ce trebuie afisate, apoi crearea unui obiect:

```
class Constant{  
public:  
String bleServerName = "Marvel Movies";  
String apiList = "http://proiectia.bogdanflore.ro/api/marvel-cinematic-  
universe-movies/movies";  
String apiFetch = "http://proiectia.bogdanflore.ro/api/marvel-cinematic-  
universe-movies/movie/";  
String idProperty= "id";  
String titleProperty = "title";  
String coverProperty = "cover_url";  
String durationProperty= "duration";  
String releaseProperty = "release_date";  
String directorProperty = "directed_by";  
String sagaProperty = "saga";  
String chronologyProperty = "chronology"; };  
Constant *constant = new Constant();
```



Definirea clasei Utils si a primei functii din aceasta clasa, functia `fetchJson`, aceasta avand rolul de a se ocupa de cererea http si de fetch-ul datelor din API, de asemenea deserializand JSON-ul (impartind-ul pe campuri si facand-ul mai usor de citit si transmis):

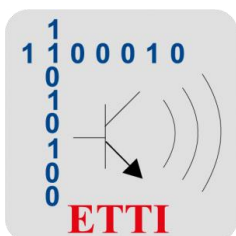
```
class Utils{  
    public:  
    static DynamicJsonDocument fetchJson(String url){  
        DynamicJsonDocument doc(8096);  
        HTTPClient http;  
        String response;  
        http.setTimeout(15000);  
        http.begin(url);  
        http.GET();  
        response = http.getString();  
        deserializeJson(doc, response);  
        return doc;  
    }  
}
```



Crearea unei liste prin intermediul unei functii de tip static ce va reprezenta primul lucru pe care utilizatorul il vede cand apasa pe butonul de „Get data” din aplicatie:

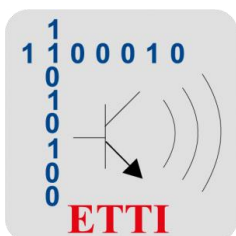
```
static DynamicJsonDocument createListDocument(JsonObject object){  
    DynamicJsonDocument listDocument(Json(8096));  
    listDocument["id"] = object[constant->idProperty];  
    listDocument["name"] = object[constant->titleProperty];  
    listDocument["image"] = object[constant->coverProperty];  
    return listDocument;  
}
```

Pe pagina urmatoare vom arata cealalta functie de tip static care este folosita pentru a crea lista care apare atunci cand utilizatorul apasa pe butonul „Details” ce va fi definit ulterior.



```
static DynamicJsonDocument
createDetailsDocumentJson(DynamicJsonDocument doc){
    DynamicJsonDocument detailsDocumentJson(8096);
    detailsDocumentJson["id"] = doc[constant->idProperty];
    detailsDocumentJson["name"] = doc[constant->titleProperty];
    detailsDocumentJson["image"] = doc[constant->coverProperty];
    detailsDocumentJson["description"] = String("Release date: ") + doc[constant-
>releaseProperty].as <String>() + String("\nDuration: ") + doc[constant-
>durationProperty].as <String>() + String("\nDirected by: ") +
    doc[constant->directorProperty].as<String>() + String("\nSaga: ") +
    doc[constant->sagaProperty].as <String>() +
    String("\nChronology: ") + doc[constant->chronologyProperty].as <String>();
    return detailsDocumentJson;
}
```

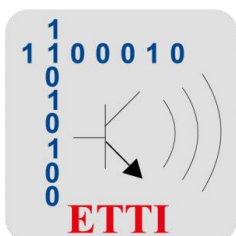
Aceste 2 functii prelucreaza, practic, datele primite de la placuta.



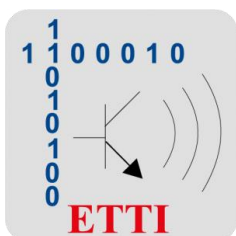
Crearea ultimei functii din clasa Utils, functia sendJson, functie ce se ocupa de transmiterea datelor din Json catre aplicatia de pe telefon.

```
static void sendJson(DynamicJsonDocument doc, BLECharacteristic
*characteristic){
    String returned;
    serializeJson(doc, returned);
    characteristic->setValue(returned.c_str());
    characteristic->notify();
}
```

Urmatoarea si ultima clasa din antet este clasa CharacteristicsCallbacks, clasa ce contine o singura functie, functia onWrite(), aceasta folosind, practic, functiile din clasa Utils si toate functiile Bluetooth pentru a lua fiecare entry din Json la rand si pentru a crea efectiv listele cu fiecare film Marvel ce se afla in Json-ul respectiv si cu toate attributele lor (explicatie mai detaliata in comentariile codului).

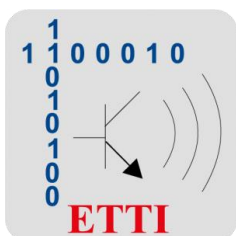


```
class CharacteristicsCallbacks: public BLECharacteristicCallbacks
{
void onWrite(BLECharacteristic *characteristic)
{
DynamicJsonDocument appRequest(8096);
//crearea obiectului
deserializeJson(appRequest, characteristic->getValue().c_str());
//deserializarea initiala a Json-ului pentru a nu crea erori
if(appRequest["action"] == "fetchData")
//daca actiunea data de user este „fetch data”
{DynamicJsonDocument webJSON = Utils::fetchJson(constant->apiList);
//fetch-ul Jsonului nostru
for(JsonObject object : webJSON.as<JsonArray>())
//trecerea prin fiecare element al Jsonului
{DynamicJsonDocument returnJSON =
Utils::createListDocumentJson(object);
Utils::sendJson(returnJSON, characteristic);
//crearea listei de filme  } }
```



```
else if(appRequest["action"] == "fetchDetails")
//altfel daca actiunea data de user este „fetch details”
    {DynamicJsonDocument webJSON = Utils::fetchJson(constant->apiFetch
+ appRequest["id"].as<String>());
//fetch-ul elementului din json cerut
    DynamicJsonDocument returnJSON =
Utils::createDetailsDocumentJson(webJSON);
    Utils::sendJson(returnJSON, characteristic);
//crearea listei de attribute
    } } };
```

Acestea au fost toate componentele antetului. Urmatoarea „oprire” este functia `setup()`, functie care va fi executata doar o data.

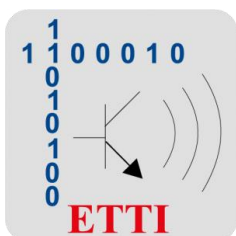


Definirea functiei `setup()`, conectarea la frecventa 115200 si aplicarea unei intarzieri de 5 secunde pentru lipsa confuziei la afisare:

```
void setup() {  
  Serial.begin(115200);  
  delay(5000);
```

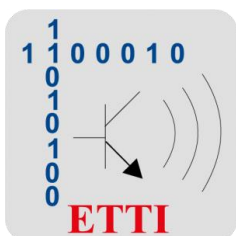
Conectarea placutei la Wi-Fi si afisarea mesajului „Connecting to WiFi.”, la care se va adauga cate un simbol „.” la fiecare 1.5 secunde pana cand placuta stabileste conexiunea Wi-Fi.

```
WiFi.mode(WIFI_STA);  
WiFi.begin(ssid, password);  
Serial.print("Connecting to WiFi.");  
while (WiFi.status() != WL_CONNECTED)  
{
```



```
delay(1500);  
  Serial.print("."); }  
Serial.println("");  
Serial.print("WiFi connected with IP: ");  
Serial.println(WiFi.localIP());
```

Urmeaza partea unde toate acele functii din antet vor fi folosite. Prima data, creem un server BLE si atribuim unor valori caracteristicilor `indexCharacteristic` si `detailsCharacteristic` (ce vor reprezenta practic cele 2 input-uri pe care un user le va putea da aplicatiei indata ce placuta este conectat la Bluetooth), iar apoi atribuim callbacks-urilor acestor caracteristici un obiect de tip `CharacteristicsCallbacks`, adica clasa ce se ocupa cu crearea efectiva a listelor de filme si de detalii despre aceste filme. Asta face ca atunci cand aceste caracteristici sunt expuse de server, ele vor apela toate functiile din antet si vor afisa in aplicatie fie lista de filme (daca cea de index este expusa), fie detaliile despre un film (daca cea de details este expusa):



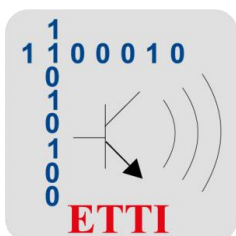
```
BLEDevice::init("Marvel Movies");  
BLEServer *pServer = BLEDevice::createServer();  
pServer->setCallbacks(new MyServerCallbacks());  
BLEService *bmeService = pServer->createService(SERVICE_UUID);  
bmeService->addCharacteristic(&indexCharacteristic);  
indexDescriptor->setValue("Get data list");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

Urmeaza ultima parte a codului. Aici, serverul de BLE este „pus pe piata” (adica va aparea in listele dispozitivelor disponibile din raza de actiune), iar placuta este, oficial, conectata la Bluetooth.

```
bmeService->start();  
  
BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();  
pAdvertising->addServiceUUID(SERVICE_UUID);  
pAdvertising->setScanResponse(true);  
pAdvertising->setMinPreferred(0x06);  
pAdvertising->setMinPreferred(0x12);  
pServer->getAdvertising()->start();  
Serial.println("Bluetooth Low Energy is ready to use!");  
}
```

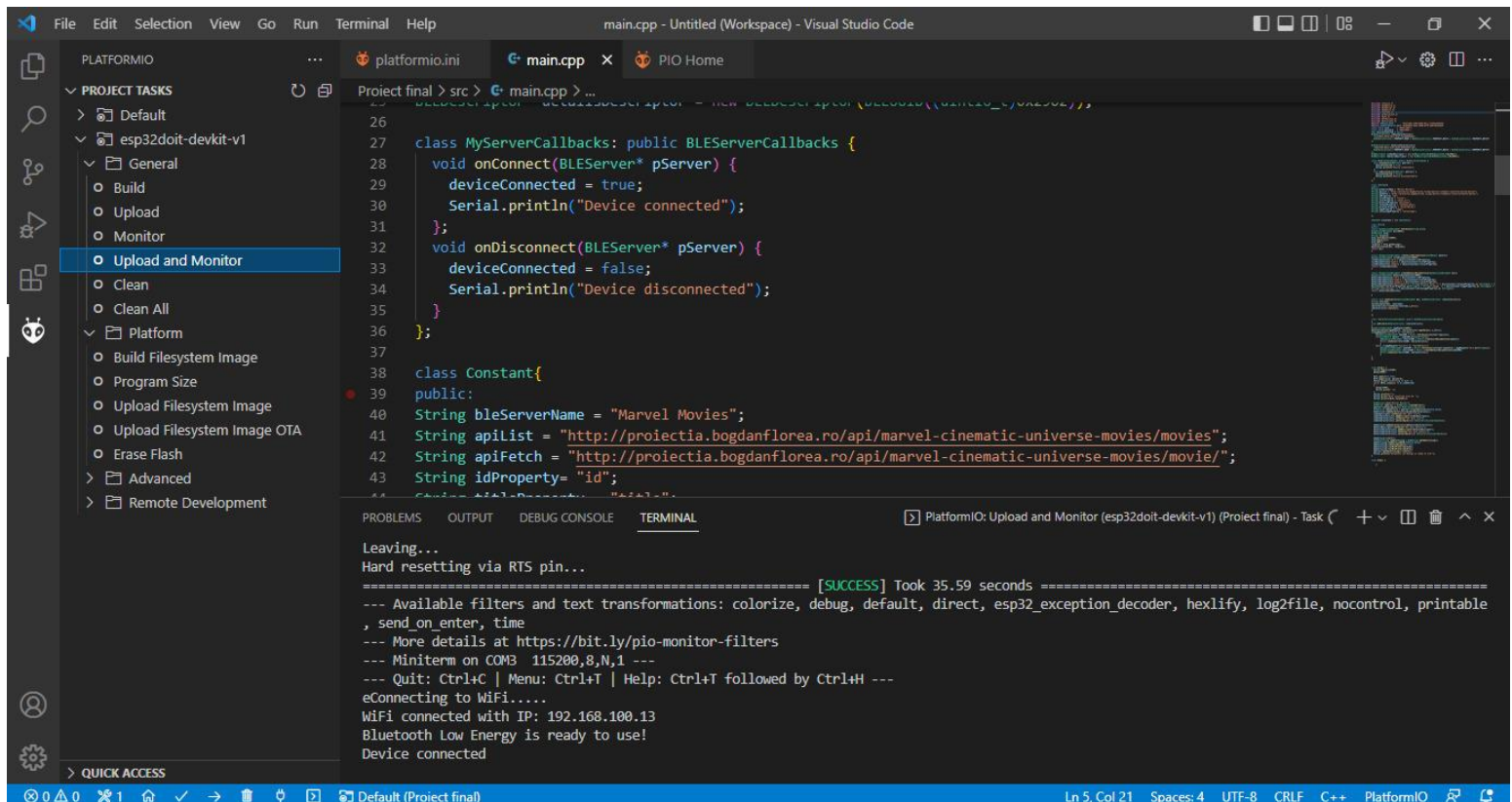
La final, apelam functia loop() obligatorie ca programul sa ruleze, insa o lasam goala deoarece totul se intampla in functia setup().

```
void loop() { }
```



3.3. Demonstratie vizuala a functionarii programului

Cand codul este uploadat prima data in placuta, dupa ce este lasat sa faca toate verificarile si sa se conecteze la Wi-Fi si BLE, in consola IDE-ului va arata asta:

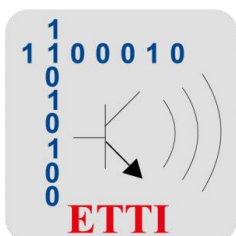


```

26
27 class MyServerCallbacks: public BLEServerCallbacks {
28     void onConnect(BLEServer* pServer) {
29         deviceConnected = true;
30         Serial.println("Device connected");
31     };
32     void onDisconnect(BLEServer* pServer) {
33         deviceConnected = false;
34         Serial.println("Device disconnected");
35     }
36 };
37
38 class Constant{
39 public:
40     String bleServerName = "Marvel Movies";
41     String apiUrl = "http://proiectia.bogdanflore.ro/api/marvel-cinematic-universe-movies/movies";
42     String apiFetch = "http://proiectia.bogdanflore.ro/api/marvel-cinematic-universe-movies/movie/";
43     String idProperty= "id";
44     String idValue= "1";
45 };
46
47 void setup() {
48     Serial.begin(115200);
49     while (!Serial) {
50         ; // wait for serial port to connect.
51     }
52     Serial.println("Setup complete");
53     BLEDevice::init("Marvel Movies");
54     BLEServer* pServer = BLEDevice::createServer();
55     pServer->setCallbacks(new MyServerCallbacks());
56     pServer->setName("Marvel Movies");
57     pServer->addService(new BLEService("Marvel Movies"));
58     pServer->addCharacteristic(new BLECharacteristic("Marvel Movies"));
59     pServer->start();
60     Serial.println("Server started");
61 }
62
63 void loop() {
64     // Do nothing here
65 }
66
67 #endif

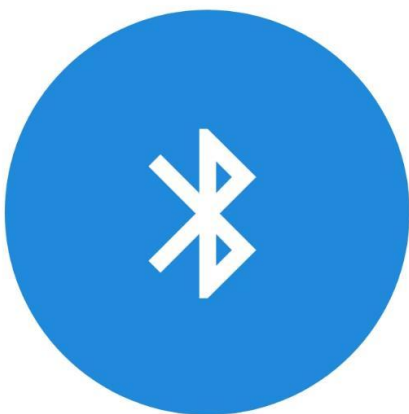
```

Leaving...
Hard resetting via RTS pin...
===== [SUCCESS] Took 35.59 seconds =====
--- Available filters and text transformations: colorize, debug, default, direct, esp32_exception_decoder, hexlify, log2file, nocontrol, printable, send_on_enter, time
--- More details at <https://bit.ly/pio-monitor-filters>
--- Miniterm on COM3 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Connecting to WiFi.....
WiFi connected with IP: 192.168.100.13
Bluetooth Low Energy is ready to use!
Device connected

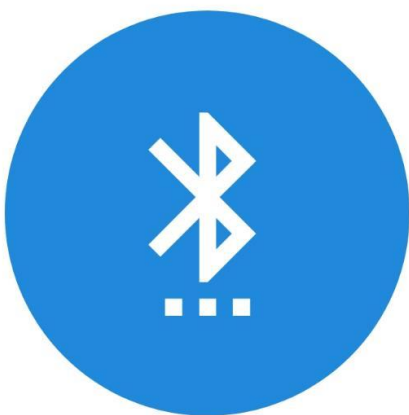


Trecem la aplicatia de pe telefon. La intrarea in ea, pe ecran va fi afisat urmatorul lucru:

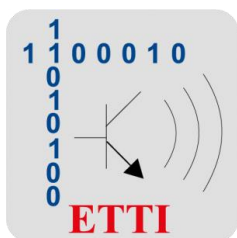
Home



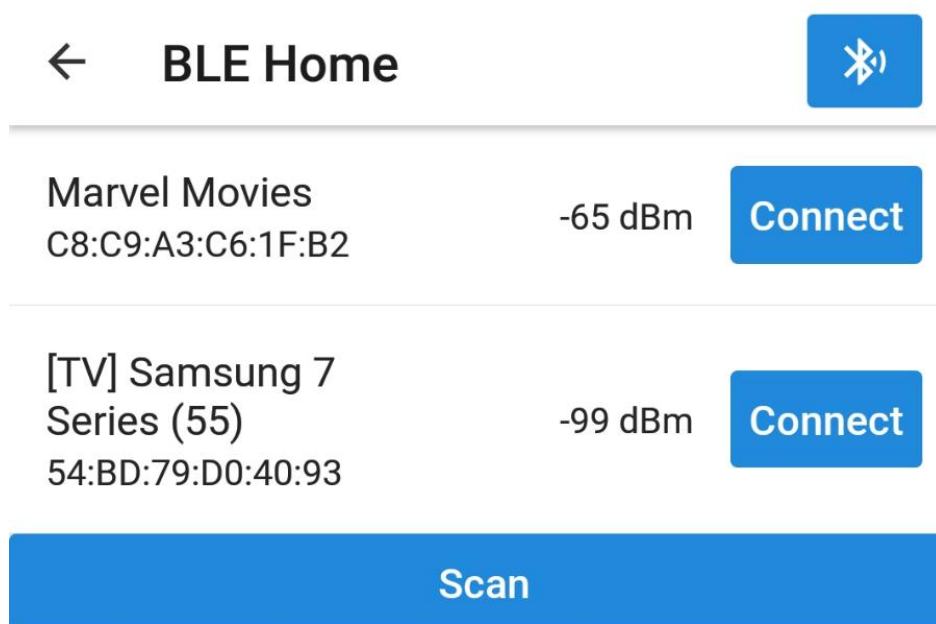
Bluetooth Classic



Bluetooth Low Energy



Alegand optiunea Bluetooth Low Energy, vom avea de a face cu urmatorul pas, unde trebuie sa apasam optiunea de „scan” (Bluetooth-ul telefonului trebuie sa fie pornit la acest pas, deoarece noi efectiv scanam imprejurimea pentru dispozitive BLE disponibile pentru conectare):



Dupa ce apasam „Connect” pe dispozitivul numit „Marvel Movies” (acesta este numele pus de noi), inseamna ca placuta este imperecheata prin Bluetooth cu telefonul. Putem observa frecventa pe care am setat-o la inceputul functiei setup() (data in dBm) si avem optiunea de Get data, optiune pe care o vom apasa.

← BLE Device

Marvel Movies

C8:C9:A3:C6:1F:B2










-65
dBm

Get data

687e2f19-d2cc-4def-a69d-e9711bcd2e1e

>

Dupa asta, o sa vedem lista de filme continuta in Json-ul pe care l-am preluat. Putem da scroll sa vedem lista completa si putem apasa butonul de „Details” pentru a vedea attributele fiecarui film.

← List		
	Iron Man 2 ID: 3	Details
	Thor ID: 4	Details
	Captain America: The First Avenger ID: 5	Details
	Iron Man 3 ID: 7	Details
	Guardians of the Galaxy ID: 10	Details
	Ant-Man ID: 12	Details
	Doctor Strange ID: 14	Details
	Guardians of the Galaxy Vol. 2 ID: 15	Details
	Spider-Man: Homecoming ID: 16	Details

Asa arata lista de detalii pentru fiecare film:

← Details

Guardians of the Galaxy



Description

Release date: 2014-08-01
Duration: 121
Directed by: James Gunn
Saga: Infinity Saga
Chronology: 11

← Details

Spider-Man: Far From Home

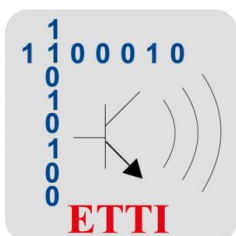


Description

Release date: 2019-07-02
Duration: 129
Directed by: Jon Watts
Saga: Infinity Saga
Chronology: 24

4. Concluzii

Lucrand de-a lungul semestrului cu placuta ESP32 si cu Arduino in general, am creat o apreciere enorma pentru aceasta tehnologie. Observandu-i potentialul si fiind nevoiti sa trecem peste mai multe neajunsuri academice pe care le-am avut in rezolvarea diverselor task-uri de la aceasta materie, putem spune ca am creat un atasament special fata de acest tip de programare si fata de aceasta latura a informaticii, o latura pe care cu siguranta o sa dorim sa o mai aprofundam si o latura in care o sa mai continuam sa ne dezvoltam si dupa terminarea anului universitar.



5. Bibliografie

Site-ul de unde am descarcat aplicatia si de unde am luat API-ul si documentatia acestuia:

<http://proiectia.bogdanflore.ro/>

De asemenea, pentru descrierea placutei ESP32 am preluat informatia din cursul de pe moodle al domnului profesor Florea.

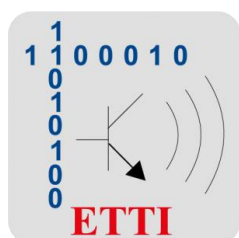
Site-ul oficial Arduino:

<https://www.arduino.cc/>

Wi-Fi:

<https://ro.wikipedia.org/wiki/Wi-Fi>

<https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/>



Bluetooth si Bluetooth Low Energy:

<https://ro.wikipedia.org/wiki/Bluetooth>

https://en.wikipedia.org/wiki/Bluetooth_Low_Energy

Format-ul Json:

<https://ro.wikipedia.org/wiki/JSON>

<https://arduinojson.org/>

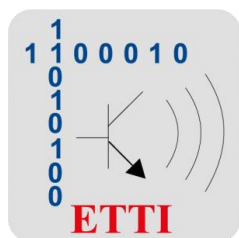
<https://ro.wikipedia.org/wiki/JSON>

<https://how2electronics.com/sim900-800-http-post-request-json-format-arduino/>

Metode si cereri HTTP:

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

<https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-requests>



6. Anexa – codul complet al programului

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

#define SERVICE_UUID      "4fafc201-1fb5-459e-8fcc-  
c5c9c331914b"

#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-  
b7f5-ea07361b26a8"

const char* ssid      = "placeholder_ssid";
const char* password  = "placeholder_password";
bool deviceConnected = false;

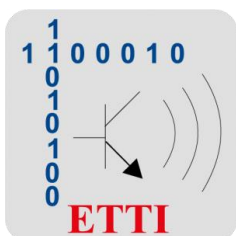
BLECharacteristic indexCharacteristic(
    "bc4196e1-05fd-4bcc-bfa7-7e2e00d03bba",
    BLECharacteristic::PROPERTY_READ |
    BLECharacteristic::PROPERTY_WRITE |
    BLECharacteristic::PROPERTY_NOTIFY
);
```

```
BLECharacteristic detailsCharacteristic(
    "385e7304-06a1-4235-ba49-d084db02fca9",
    BLECharacteristic::PROPERTY_READ |
    BLECharacteristic::PROPERTY_WRITE |
    BLECharacteristic::PROPERTY_NOTIFY
);

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

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 Constant{
public:

String bleServerName = "Marvel Movies";

String apiList = "http://proiectia.bogdanflore.ro/api/marvel-
cinematic-universe-movies/movies";

String apiFetch =
"http://proiectia.bogdanflore.ro/api/marvel-cinematic-
universe-movies/movie/";

String idProperty= "id";
String titleProperty = "title";
String coverProperty = "cover_url";
String durationProperty= "duration";
String releaseProperty = "release_date";
String directorProperty = "directed_by";
String sagaProperty = "saga";
String chronologyProperty = "chronology";
};
```

```
Constant *constant = new Constant();
```

```
class Utils{
public:

static DynamicJsonDocument fetchJson(String url){
DynamicJsonDocument doc(8096);

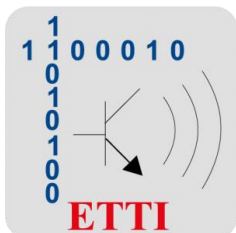
HttpClient http;

String response;

http.setTimeout(15000);

http.begin(url);

http.GET();
```



```
response = http.getString();

deserializeJson(doc, response);

return doc;
}

static DynamicJsonDocument
createListDocumentJson(JsonObject object){
DynamicJsonDocument listDocumentJson(8096);

listDocumentJson["id"] = object[constant->idProperty];

listDocumentJson["name"] = object[constant-
>titleProperty];

listDocumentJson["image"] = object[constant-
>coverProperty];

return listDocumentJson;
}

static DynamicJsonDocument
createDetailsDocumentJson(DynamicJsonDocument doc){
DynamicJsonDocument detailsDocumentJson(8096);

detailsDocumentJson["id"] = doc[constant->idProperty];

detailsDocumentJson["name"] = doc[constant-
>titleProperty];

detailsDocumentJson["image"] = doc[constant-
>coverProperty];

detailsDocumentJson["description"] = String("Release date:
") + doc[constant->releaseProperty].as <String>() +
String("\nDuration: ") + doc[constant->durationProperty].as
<String>() + String("\nDirected by: ") +

doc[constant->directorProperty].as<String>() +
String("\nSaga: ") + doc[constant->sagaProperty].as
<String>() +

String("\nChronology: ") + doc[constant-
>chronologyProperty].as <String>();

return detailsDocumentJson;
}

static void sendJson(DynamicJsonDocument doc,
BLECharacteristic *characteristic){

String returned;

serializeJson(doc, returned);
```



```
characteristic->setValue(returned.c_str());
characteristic->notify();
});

class CharacteristicsCallbacks: public
BLECharacteristicCallbacks
{
void onWrite(BLECharacteristic *characteristic)
{
DynamicJsonDocument appRequest(8096);
deserializeJson(appRequest, characteristic-
->getValue().c_str());
if(appRequest["action"] == "fetchData")
{
DynamicJsonDocument webJSON =
Utils::fetchJson(constant->apiList);

for(JsonObject object : webJSON.as<JsonArray>())
{
DynamicJsonDocument returnJSON =
Utils::createListDocumentJson(object);
Utils::sendJson(returnJSON, characteristic);
}
}
else if(appRequest["action"] == "fetchDetails")
{
DynamicJsonDocument webJSON =
Utils::fetchJson(constant->apiFetch +
appRequest["id"].as<String>());

DynamicJsonDocument returnJSON =
Utils::createDetailsDocumentJson(webJSON);
Utils::sendJson(returnJSON, characteristic);
}
}
};
```

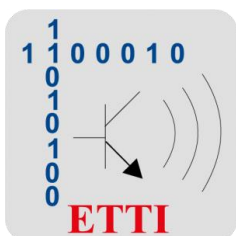
```
void setup() {
Serial.begin(115200);
delay(5000);
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
Serial.print("Connecting to WiFi.");
while (WiFi.status() != WL_CONNECTED)
{
delay(1500);
Serial.print(".");
}
Serial.println("");
Serial.print("WiFi connected with IP: ");
Serial.println(WiFi.localIP());

BLEDevice::init("Marvel Movies");
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);
pAdvertising->setScanResponse(true);
pAdvertising->setMinPreferred(0x06);
pAdvertising->setMinPreferred(0x12);
pServer->getAdvertising()->start();

Serial.println("Bluetooth Low Energy is ready to use!");
}

void loop() { }

```

