Echipa: TĂTULESCU ANDREEA-ELENA și VĂDUVA MATIAS
Grupa: 413A

PROIECT IA

BOARD GAME ATLAS

1. INTRODUCERE

Un board game este definit ca un joc care include o tablă, zaruri sau cărți, pentru care este nevoie de 2 sau mai mulți jucători, în care câștigă jucătorul care îndeplinește primul sarcina descrisă în setul de reguli al respectivului joc.

Board Game Atlas este o comunitate de jocuri de societate. A fost fondată de CEO-ul Trent Ellingsen în 2018, iar în 2019 I s-a alăturat co-fondatorul Phil Ryuh, fiind lansat pe 9 ianuarie 2019. Cei doi au creat cel mai bun "loc" pentru ca oamenii doritori de distracție să se bucure, să cumpere și să discute între ei despre jocurile de societate în mediul online.

Board Game Atlas API este un API REST care vă permite să accesați și să utilizați compilarea tuturor datelor cunoscute despre jocurile de societate disponibile.

2. CONSIDERENTE TEORETICE

• API și REST API

Un API este un set de definiții și protocoale pentru construirea și integrarea aplicațiilor software. Este uneori denumit un contract între un furnizor de informații și un utilizator de informații - stabilirea conținutului solicitat de la consumator (apelul) și conținutul cerut de producător (răspunsul).

Cu alte cuvinte, dacă doriți să interacționați cu un computer sau un sistem pentru a prelua informații sau a îndeplini o funcție, un API vă ajută să comunicați ceea ce doriți acelui sistem, astfel încât să poată înțelege și îndeplini cererea.

Vă puteți gândi la un API ca un mediator între utilizatori sau clienți și resursele sau serviciile web pe care doresc să le obțină. Este, de asemenea, o modalitate pentru o organizație de a partaja resurse și informații, menținând în același timp securitatea, controlul și autentificarea - determinând cine are acces la ce.

Un alt avantaj al unui API este că nu trebuie să cunoașteți specificul memorării în cache - cum este preluată resursa sau de unde provine.

REST este un set de constrângeri arhitecturale, nu un protocol sau un standard. REST a fost creat de informaticianul Roy Fielding.

Un API REST (cunoscută și ca API RESTful) este o interfață de programare a aplicațiilor (API sau API web) care se conformează constrângerilor stilului arhitectural REST și permite interacțiunea cu serviciile web RESTful.

Când o solicitare a clientului este făcută printr-un API RESTful, acesta transferă o reprezentare a stării resursei către solicitant sau punct final. Aceste informații sau reprezentare sunt livrate în unul dintre mai multe formate prin HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP sau text simplu. JSON este cel mai popular format de fișier de utilizat, deoarece, în ciuda numelui său, este independent de limbă și poate fi citit atât de oameni, cât și de mașini.

- ## MODULUL ESP32

ESP32 este un mic **cip de tipul SoC (System on Chip)** inserat în carcasaa QFN48. Este produs de societatea **Espressif Systems** din Shangai. ESP32 reprezintă dezvoltarea, completarea și îmbunătățirea revoluționarului ESP8266. ESP8266 este o solutie economică care combină un microcontroller puternic și un modul Wi-Fi.

ESP32 se bazează pe un **microcontroller Xtensa 32 biți rapid si eficient** (80…160Mhz, dual core, de până la 600 DMIPS în funcție de versiune), cu 520 kB de memorie SRAM integrate și mecanisme integrate de Securitate a datelor.

Principalele caracteristici sunt prezența de vaste circuite de emisie/recepție la 2,4 GHz, care permimt comunicarea în norma Wi-Fi (cu o lățime de bandă de până la 150 Mbps), precum și în norma Bluetooth (BT), ambele în versiunea “clasică” si BLE cu economie de energie, cu opțiuni suplimentare, precum Piconet si Scatternet. Are 34 intrări/ieșiri numerice universal (GPIO), care pot, de asemenea, să aibă funcții opționale legate de blocurile de hardware incorporate, cum ar fi: convertor ADC de 12 biți cu 18 intrări, 2 DAC de 8 biți, 10 senzori tactili.

Modulele actuale, denumite ESP32, conțin o memorie Flash externă suplimentară cu o capacitate inclusă în plaja (4MB…16MB).

- ## PROTOCOALELE SI METODELE DE COMUNICATIE UTILIZATE (Bluetooth Classic sau Low Energy, WiFi, HTTP, JSON)

  - ### <u>Bluetooth Low Energy</u>

Bluetooth Low Energy (BLE, BTLE) în traducere Bluetooth cu consum redus sau Bluetooth cu energie redusă, este o tehnologie fără fir bazata pe Bluetooth care facilitează conexiunile pe distanțe mici (5…10m). A fost create de Nokia la începutul lui 2006 sub numele de Bluetooth Low End Extension, pentru ca în octombrie 2006 să fie numit Wibree.

BLE este folosit pentru rețelele personale (PAN) și se bazează pe conexiunile wireless pentru scurte distanțe, având un cost scăzut de implementare. În principiu, acest protocol,

împreună cu tradiționalul Bluetooth, au fost concepute pentru a scădea numărul de cabluri necesare conectării perifericelor la un calculator personal.

○ WiFi

WiFi este numele commercial 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 e furnizat de diferite dispositive hardware, și de aproape toate sistemele de operare modern pentru calculatoarele personale, rutere, telefoane mobile, console de jocuri, etc.

Standardul IEEE 802.11 descrie protocoale de comunicație aflate la nivelul gazdă-rețea al Modelului TCP/IP, respectiv la nivelurile fizic și legătură de date ale Modelului OSI. Aceasta înseamnă că implementările IEEE 802.11 trebuie să primească pachete de la protocoalele de la nivelul rețea (IP) și să se ocupe cu transmiterea lor, evitând eventualele coliziuni cu alte stații care doresc să transmită.

○ HTTP

Hypertext Transfer Protocol (**HTTP**) este metoda cea mai des utilizată pentru accesarea informațiilor în internet care sunt păstrate pe servere World Wide Web (WWW). Protocolul HTTP este un protocol de tip text, fiind protocolul "implicit" al WWW.

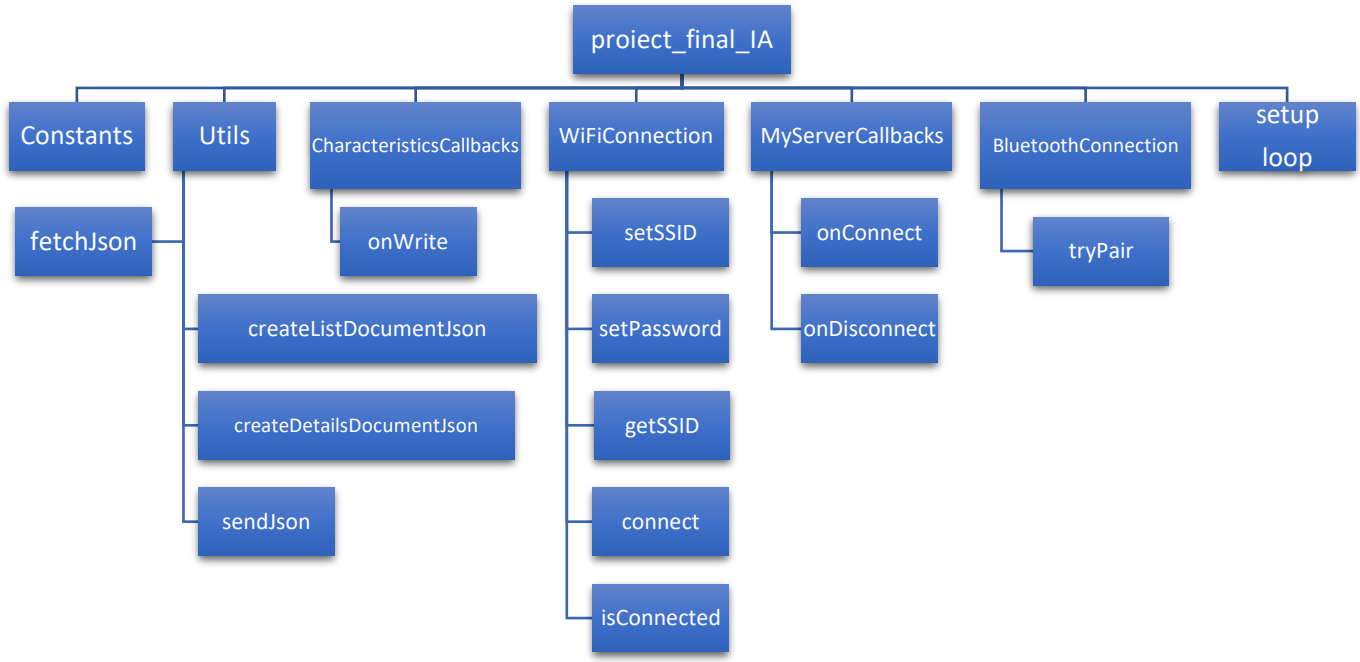
HTTP oferă o tehnică de comunicare prin care paginile web se pot transmite de la un computer aflat la distanță spre propriul computer. Dacă se apelează un link sau o adresă de web cum ar fi http://www.example.com , atunci se cere calculatorului host să afișeze o pagină web (index.html sau altele). În prima fază numele (adresa) www.example.com este convertit de protocolul DNS într-o adresă IP. Urmează transferul prin protocolul TCP pe portul standard 80 al serverului HTTP, ca răspuns la cererea HTTP-GET. În urma cererii HTTP-GET urmează din partea serverului răspunsul cu datele cerute, ca de ex.: pagini în (X)HTML, cu fișiere atașate ca imagini, fișiere de stil (CSS), scripturi (Javascript), dar pot fi și pagini generate dinamic (SSI, JSP, PHP și ASP.NET). Dacă dintr-un anumit motiv informațiile nu pot fi transmise, atunci serverul trimite înapoi un mesaj de eroare. Modul exact de desfășurare a acestei acțiuni (cerere și răspuns) este stabilit în specificațiile HTTP.

- **JSON**

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.

Eleganța formatului JSON provine din faptul că este un subset al limbajului JavaScript**.**

3.IMPLEMENTARE – program Arduino



Proiectul este format din 2 funcții globale (**setup si loop**) și celelalte componente principale ale programului (clasele sale). La rândul lor, aproape fiecare clasă are in structura ei o serie de funcții. Funcția **setup** menționează frecvența plăcuței și inițializează instanțele de clase menționate mai sus, iar funcția **loop** nu conține nicio instrucțiune.

O diferență o face clasa **Constants** care nu este alcătuită din functii. În această clasă păstrăm constantele necesare programului (șiruri de caractere).

Clasa **Utils** conține niște funcții statice:

- fetchJson se conectează la URL-ul specificat, preia informațiile de pe site și le transformă într-un document Json;
- cele 2 functii de creare, createListDocumentJson si createDetailsDocumentJson. Prima functie creează documentul pentru fiecare obiect din document, iar in cea de-a doua funcție sunt memorate și detaliile obiectelor;
- sendJson schimbă variabila characteristic, variabilă la care au acces în același timp atât telefonul, cât și plăcuța.

Clasa **WifiConnection (**instanță: WiFiInstance) conține cele 2 variabile(ssid, password) prin care se conectează la internet.

Clasa **BluetoothConnection** (instanță: BluetoothInstance) asigură conexiunea la rețeaua Bluetooth și permite accesul la aplicația mobilă.

Clasa **CharacteristicsCallbacks** cu functia onWrite preia informațiile din fișierul Json din characteristic, le interpretează (fetchData/fetchDetails) și în funcție de interpretare se conectează la internet, preia informațiile necesare și le trimite către aplicație pe telefon.

Clasa **MyServerCallbacks** verifică conexiunea la internet a aplicației.

Partea de afișare a fost realizată de altcineva.

4.CONCLUZII

În concluzie, plăcuța primește de la telefon o interogare prin Bluetooth (Low Energy) pe care o interpretează și, pe baza a ceea ce s-a cerut, răspunde în mod corespunzător.

Aplicația de pe telefon, “Proiect IA” trebuie să afișeze informații legate de jocuri de societate (**Board Game Atlas** – nr 14).

Ne asiguram ca avem pornit Bluetooth și din aplicație alegem modul Bluetooth Low Energy, scanăm și ne conectăm la plăcuță, selectăm “Get data” și va apărea lista cu jocurile de societate. Dacă dorim să vedem detaliile unui joc apăsăm pe “Details”.

5.BIBLIOGRAFIE

Informatii despre Board Game Atlas:

- Board Game Atlas Acquires Board Game Prices | Board Game Atlas
- About Board Game Atlas

Informatii despre API și REST API:

- https://www.redhat.com/en/topics/api/what-is-a-rest-api

Informatii despre Modulul ESP32:

- https://www.tme.eu/ro/news/library-articles/page/21733/O-gama-larga-de-module-ESP32/

Informatii despre Bluetooth Low Energy:

- https://ro.m.wikipedia.org/wiki/Bluetooth_Low_Energy

Informatii despre Wi-Fi:

- https://ro.m.wikipedia.org/wiki/Wi-Fi

Informatii despre HTTP:

- https://ro.m.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Informatii despre JSON:

- https://ro.m.wikipedia.org/wiki/JSON

6.ANEXE

Link pastebin către cod (parola: pjinx3SsJ6): https://pastebin.com/G6GavJ3Q (valabil doar un an).

Codul:

```
#include <Arduino.h>

#include <BLEDevice.h>

#include <BLEServer.h>

#include <BLEUtils.h>

#include <BLE2902.h>

#include <ArduinoJson.h>

#include <WiFi.h>

#include <HTTPClient.h>
```

```
//This code is released under the MIT license.
//Please check the license before redistributing.

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

//Constants
//Use https://www.uuidgenerator.net/ to get a random UUID.
class Constants{
public:
  String ssid = "Tătu";
  String password = "seminte22";
  String bleServerName = "Proiect_IA_Low_Energy";
  String service_uuid = "117178da-16d9-4c83-a248-1e0039b7f319";
  String apiListURL = "http://proiectia.bogdanflorea.ro/api/board-game-atlas/games";
  String apiFetchURL = "http://proiectia.bogdanflorea.ro/api/board-game-atlas/game?ids=";
  String idProperty = "id";
  String nameProperty = "name";
  String imageProperty = "image_url";
  String year = "year_published";
  String descriptionProperty = "price_text";
  bool deviceConnected = false;
};
Constants *constants = new Constants();
```

```
BLECharacteristic indexCharacteristic(
  constants->service_uuid.c_str(),
  BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE |
BLECharacteristic::PROPERTY_NOTIFY
);
BLECharacteristic detailsCharacteristic(
  constants->service_uuid.c_str(),
  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));

//Classes
class Utils{
public:
  static DynamicJsonDocument fetchJson(String url){
    DynamicJsonDocument document(8096);
    HTTPClient http;
    String payload;

    http.setTimeout(15000);
    http.begin(url);
    http.GET();
    payload = http.getString();
    deserializeJson(document, payload);

    return document;
```

```
}

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

    listDocumentJson["id"] = object[constants->idProperty];
    listDocumentJson["name"] = object[constants->nameProperty];
    listDocumentJson["image"] = object[constants->imageProperty];

    return listDocumentJson;
}

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

    detailsDocumentJson["id"] = document[constants->idProperty];
    detailsDocumentJson["name"] = document[constants->nameProperty];
    detailsDocumentJson["image"] = document[constants->imageProperty];
    detailsDocumentJson["description"] = document[constants-
>descriptionProperty].as<String>() + ", " + document[constants->year].as<String>();

    return detailsDocumentJson;
}

static void sendJson(DynamicJsonDocument document, BLECharacteristic *characteristic){
    String returned;
    serializeJson(document, returned);
```

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

class WiFiConnection{
protected:
  String ssid;
  String password;
public:
  WiFiConnection(String ssid, String password, bool begin = true) :
              ssid(ssid), password(password){
    if(begin){
      this->connect();
    }
  }

  void setSSID(String ssid){
    this->ssid = ssid;
  }
  void setPassword(String password){
    this->password = password;
  }

  String getSSID(){
    return this->ssid;
```

```
  }

  void connect(){
    WiFi.begin(this->ssid.c_str(), this->password.c_str());
  }
  bool isConnected(){
    if(WiFi.status() == WL_CONNECTED){
      return true;
    }

    return false;
  }
};
WiFiConnection* WiFiInstance;

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(constants->apiListURL);
        for(JsonObject object : webJSON.as<JsonArray>()){
          DynamicJsonDocument returnJSON = Utils::createListDocumentJson(object);
          Utils::sendJson(returnJSON, characteristic);
        }
      }else if(appRequest["action"] == "fetchDetails"){
```

```
      DynamicJsonDocument webJSON = Utils::fetchJson(constants->apiFetchURL +
appRequest["id"].as<String>());
      DynamicJsonDocument returnJSON = Utils::createDetailsDocumentJson(webJSON);
      Utils::sendJson(returnJSON, characteristic);
    }
  }
};

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

class BluetoothConnection{
protected:
  BLEServer *pServer;
  BLEService *bmeService;
  BLEAdvertising *pAdvertising;
  String name;
public:
  BluetoothConnection(String name, bool begin = true, bool alert = true) : name(name) {
```

```
  if(begin){
    this->tryPair(alert);
  }
}

void tryPair(bool alert = false){
  BLEDevice::init(this->name.c_str());

  this->pServer = BLEDevice::createServer();
  this->pServer->setCallbacks(new MyServerCallbacks());

  this->bmeService = this->pServer->createService(constants->service_uuid.c_str());
  this->bmeService->addCharacteristic(&indexCharacteristic);
  indexDescriptor->setValue("Get data list");
  indexCharacteristic.addDescriptor(indexDescriptor);
  indexCharacteristic.setValue("Get data List");

  indexCharacteristic.setCallbacks(new CharacteristicsCallbacks());

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

  detailsCharacteristic.setCallbacks(new CharacteristicsCallbacks());

  this->bmeService->start();
```

```
    this->pAdvertising = BLEDevice::getAdvertising();
    this->pAdvertising->addServiceUUID(constants->service_uuid.c_str());
    this->pServer->getAdvertising()->start();

    if(alert){
      Serial.println("Waiting a client connection to notify...");
    }
  }
};
BluetoothConnection* BluetoothInstance;

void setup() {
  Serial.begin(115200);

  WiFiInstance = new WiFiConnection(constants->ssid, constants->password);
  BluetoothInstance = new BluetoothConnection(constants->bleServerName);
}

void loop() {
  //Empty
}
```