

Project PR-IoT

Dumitru Bianca Ștefania

GitHub [link](#)

YouTube [link](#)

Introducere

Proiectul își propune verificarea și monitorizarea participantilor unei săli de sport. Acesta are două funcționalități principale: verificarea validității abonamentului personal și contorizarea numărului de prezențe la sală.

Identificarea userilor se face pe baza unui card personal de acces, iar sistemul comunică direct atât cu utilizatorul, prin diverse mijloace (audio, vizual) în funcție de validitatea abonamentului, cât și cu administratorul sălii de sport, prin intermediul site-ului web asociat.

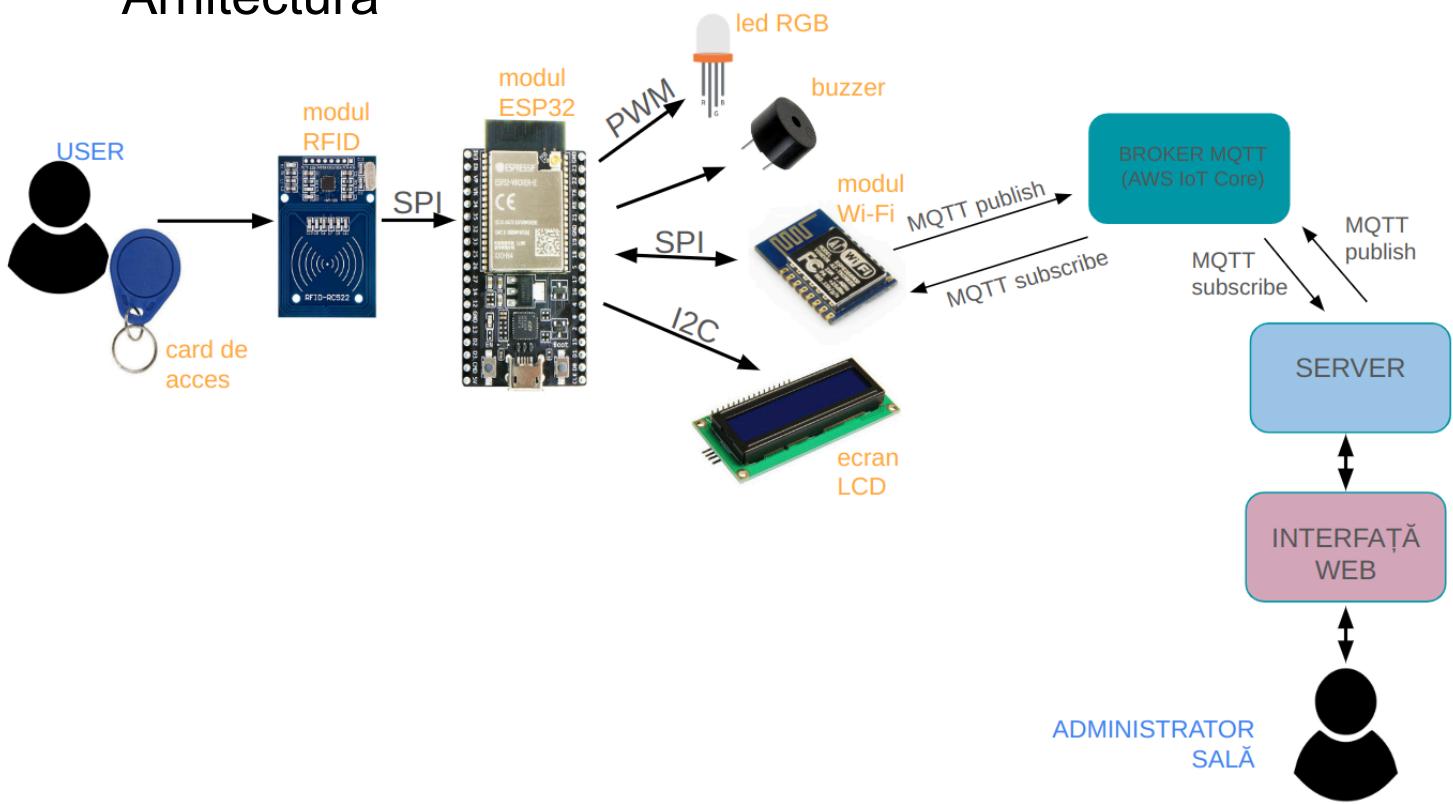
La fiecare scanare a cardului unui utilizator cu abonament valid, acestuia i se adaugă câte o prezentă. Numărul de prezențe al fiecărui user poate fi vizualizat în timp real în cadrul site-ului web. Există și posibilitatea înregistrării unui user nou, atunci când el își scanează pentru prima dată cardul. Diverse metrii și statistici sunt reprocesate odată cu adăugarea fiecărei prezențe: aglomerarea medie a sălii pe zile și ore, numărul de prezențe al fiecărui user, un heatmap de prezențe asociat fiecărui user, numărul mediu de prezențe per săptămână sau lună al unui user, etc.

Ideea proiectului este inspirată din sistemele de același tip folosite în prezent de sălile din oraș.

Proiectul este controlat de către un modul ESP32 ce suportă comunicație prin Wi-Fi. Userul interacționează cu sistemul prin intermediul cardului de acces, ce este citit de către un modul RFID. Datele cardului sunt transmise ESP-ului, care, folosindu-se de modulul Wi-Fi, comunică cu un server web personalizat pentru a determina validitatea abonamentului userului. În funcție de răspunsul primit, modulul ESP acționează un buzzer, un led RGB și un display LCD pentru a-i transmite outputul userului. Un output similar este trimis în același timp de la server către interfața site-ului web. Așadar, userul primește output direct de la componente fizice ale sistemului (LCD, buzzer, led), iar administratorul sălii de sport primește outputul de la interfața site-ului.

Interfața web va demonstra și controlul la distanță al proiectului IoT, de exemplu prin afișarea numelui userului care tocmai a fost înregistrat pe platformă sau prin controlul actuatorilor (LCD, led) odată cu închiderea și deschiderea sălii (controlată de pe site prin apăsarea unui buton). Astfel, se va demonstra și bidirectionalitatea comunicării între cele două componente.

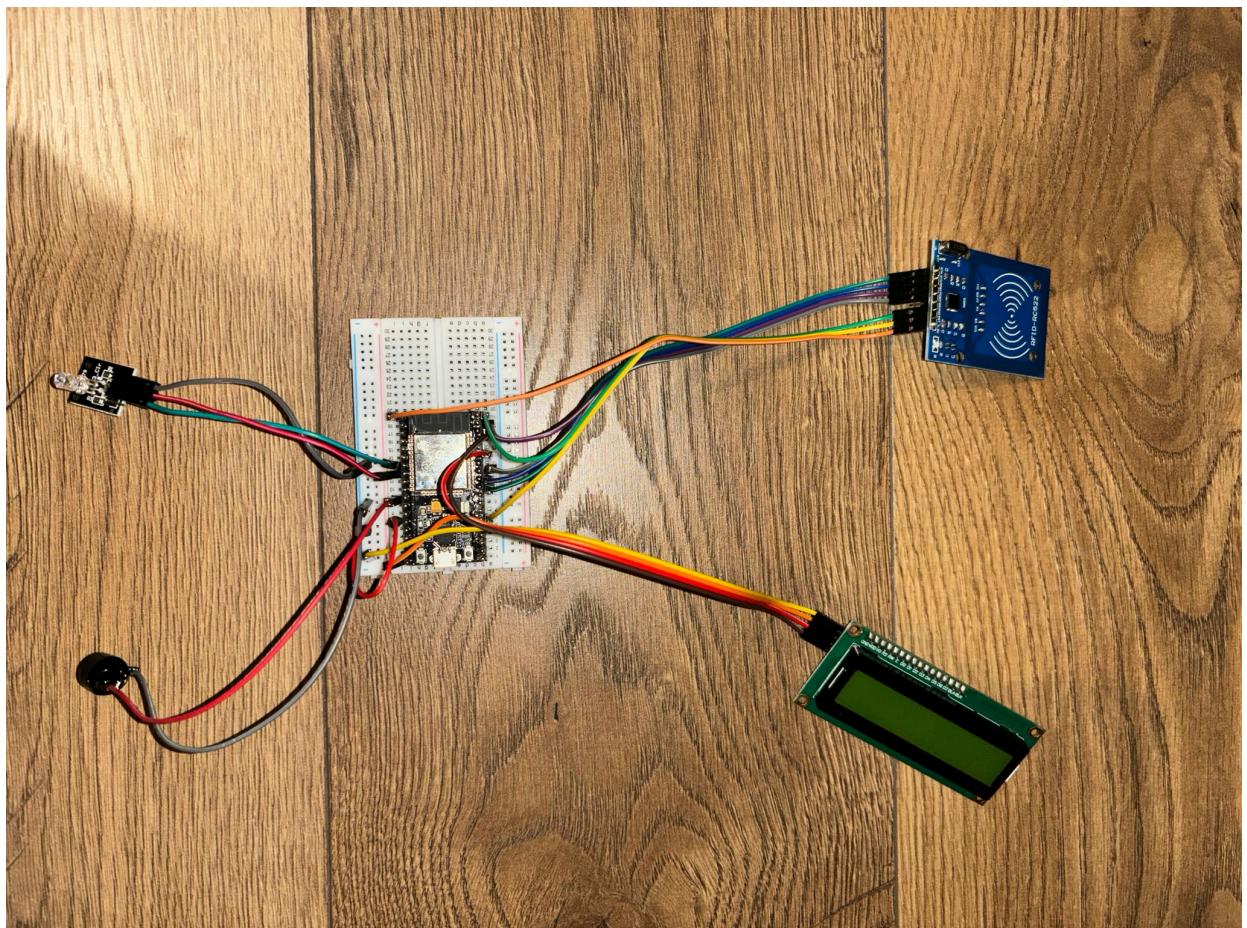
Arhitectură



Arhitectură hardware

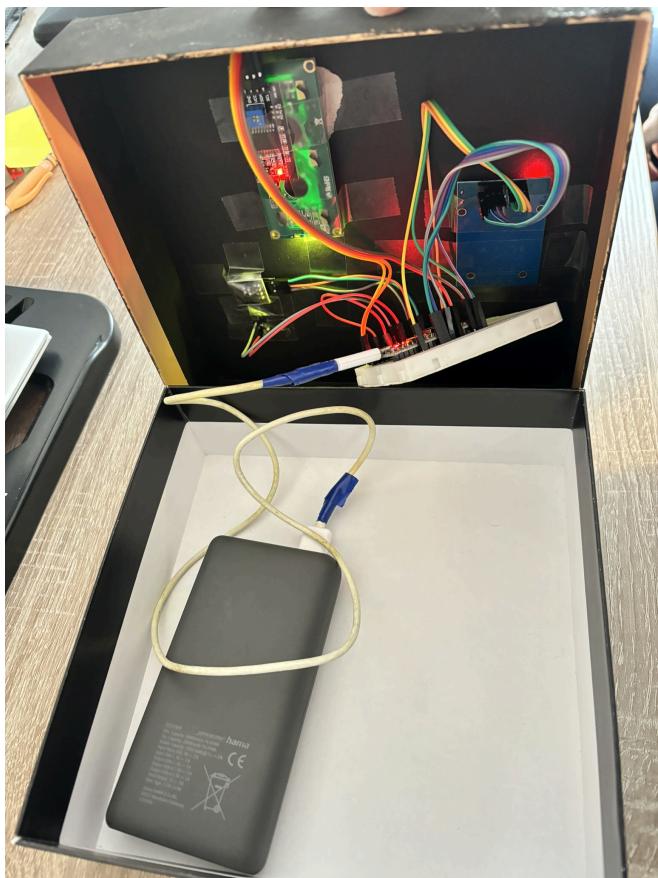
- **Microcontroller** ESP32 WROOM cu 38 de pini și modul Wi-Fi
- Modul RFID RC522 (**senzor**)
- Led RGB (**actuator**)
- LCD 16×2 cu modul I2C (**actuator**)
- Buzzer (**actuator**)
- Carduri de acces
- Breadboard
- Fita
- Sursă de alimentare - baterie externă sau direct de la laptop

Componentele asamblate:



După asamblarea inițială a componentelor, le-am testat pe rând folosind câte un exemplu generic găsit în lista de exemple din Arduino IDE, pentru a verifica corectitudinea asamblării. Apoi, am început să scriu codul care să asigure funcționalitatea dorită.

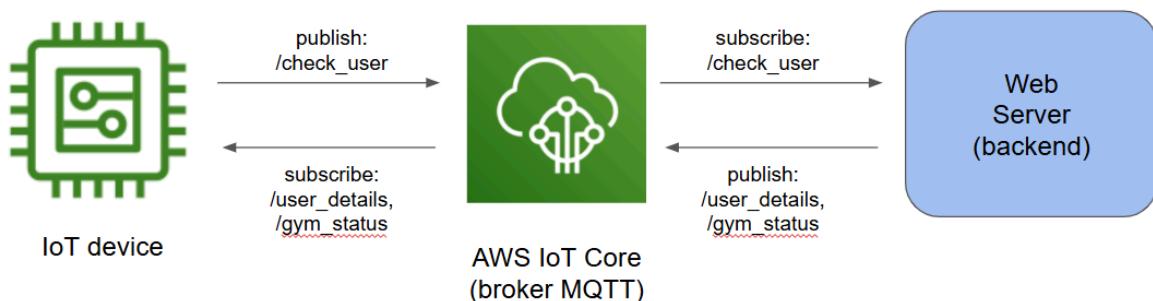
Pentru a semăna cât mai tare cu un produs real, care ar putea fi folosit într-o sală de fitness am realizat o “căcasă” pentru componente dintr-o cutie de carton, pe care am decupat-o. Am aranjat apoi componentele în căcasă și le-am lipit pentru a le fixa.



Arhitectură Software

Serverul (backend) a fost implementat în Flask, iar interfața web (frontend) în React. Comunicarea cu microcontrollerul se face prin protocolul de comunicație **MQTT**. Cele trei topicuri folosite sunt /check_user, /user_details și /gym_status.

Broker-ul de MQTT a fost creat în cadrul platformei AWS IoT Core. Atât dispozitivul IoT, cât și serverul web sunt conectate la broker pentru a facilita comunicarea.



Am decis să hostez serverul web pe Heroku, prin conectarea unui proiect cu repository-ul de pe Github. Serverul este disponibil la adresa <https://pr-project-f8c7fbee3ae5.herokuapp.com>.

Implementare

Configurarea brokerului MQTT

Pentru a putea crea un broker MQTT în cadrul platformei AWS IoT Core, am creat un nou “Thing”, cu un policy și certificate asociate, urmând pașii descriși [aici](#). Deși cred că as fi putut hosta propriul broker, am vrut să incerc să folosesc AWS IoT Core.

AWS IoT > Manage > Things > gym_attendance_system

gym_attendance_system Connectivity indexing is not enabled Info

Create secure tunnel | Edit | Delete

Thing details

Name gym_attendance_system	Type -
ARN arn:aws:iot:us-east-1:445567118675:thing/gym_attendance_system	Billing group -

Folosirea certificatelor generate (CA cert, client cert, private key) și a protocolului TLS în conectarea dispozitivului IoT și a serverului cu broker-ul AWS IoT Core asigura criptarea și autentificarea tuturor mesajelor.

Configurarea dispozitivului IoT

Pentru a dezvolta codul pentru dispozitiv, am folosit ca mediu de programare Arduino IDE.

Codul este organizat în mai multe fișiere, fiind împărțit între fișierele header cu funcții helper pentru inițializare și setare de stări (wifi.h, lcd.h, rgb.h, buzzer.h, rfid.h) și fișierul main, ce conține logica principală a programului.

Funcția de setup se ocupă cu inițializarea componentelor, realizarea conexiunii Wi-Fi cu TLS și conectarea la broker-ul MQTT:

```
void setup() {
    Serial.begin(9600);

    connectToWiFi();
    initializeRFID(rfid);
    initializeRGB(PIN_RED, PIN_GREEN);
    initializeBuzzer(PIN_BUZZER);
```

```

initializeLCD(lcd);

// Configure TLS
wifiClient.setCACert(ca_cert);
wifiClient.setCertificate(client_cert);
wifiClient.setPrivateKey(private_key);

mqttClient.setServer(mqtt_broker, mqtt_port);
mqttClient.setCallback(mqttCallback);

connectToMQTT();
}

```

Funcția `connectToMQTT` asigura conectarea corecta la broker și încercarea conexiunii în caz de fail:

```

void connectToMQTT() {
    while (!mqttClient.connected()) {
        Serial.println("Connecting to AWS IoT MQTT broker...");
        if (mqttClient.connect("ESP")) {
            Serial.println("Connected to AWS IoT!");
            mqttClient.subscribe(user_details_topic);
            mqttClient.subscribe(gym_status_topic);
        } else {
            Serial.print("Connection failed, rc=");
            Serial.println(mqttClient.state());
            delay(2000);
        }
    }
}

```

Funcția `mqttCallback` este apelată la fiecare mesaj primit pe unul din topicurile la care dispozitivul este subscribed. Răspunsul este parsat si salvat într-o variabilă globală pentru a putea fi folosit ulterior. În cazul în care mesajul este primit pe topicul `/gym_status`, acțiunile necesare sunt luate imediat (închiderea, respectiv deschiderea sălii). Stare actuatori în funcție de starea sălii:

- Închiderea sălii
 - LCD - Gym closed!
 - led - roșu

- redeschiderea sălii
 - LCD - Gym is open! (timp de 2 secunde, apoi Scan your card)
 - led - verde (timp de 2,5 secunde, apoi stins)

```

void mqttCallback(char *topic, byte *payload, unsigned int
length) {
    char message[length + 1];
    memcpy(message, payload, length);
    message[length] = '\0';

    mqttResponse = String(message);
    mqttResponseReceived = true;

    if(strcmp(topic, gym_status_topic) == 0) {
        // Parse response
        StaticJsonDocument<200> responseDoc;
        deserializeJson(responseDoc, mqttResponse);

        gymStatus = responseDoc["gym_status"];
        if(gymStatus == true) {
            closeGym();
        } else {
            openGym();
        }

        mqttResponseReceived = false;
    }
}

```

Funcția loop așteaptă detectarea unui card de către RFID, moment în care publică un mesaj de tip /check_user (cu ID-ul cardului scanat) pentru a interoga serverul referitor la cardul scanat. Pe baza răspunsului serverului, programul afișează diverse mesaje, revenind ulterior înapoi în starea de așteptare.

```

void loop() {
    // Maintain MQTT connection
    if (!mqttClient.connected()) {
        connectToMQTT();
    }
}

```

```

    }

    mqttClient.loop();

    if(gymStatus == true) return; // if the gym is closed don't
read cards

    // Reset the loop if no new card present on the
sensor/reader
    if (!rfid.PICC_IsNewCardPresent())
        return;

    // Verify if the ID has been read
    if (!rfid.PICC_ReadCardSerial())
        return;

    String cardID = "";
    bool validCard = getCardID(rfid, cardID);
    if (!validCard)
        return;

    // Publish card ID to MQTT broker
    StaticJsonDocument<200> doc;
    doc["card_id"] = cardID;
    String message;
    serializeJson(doc, message);
    mqttClient.publish(check_user_topic, message.c_str());

    // Wait for a response from the server
    unsigned long startTime = millis();
    mqttResponseReceived = false;
    while (!mqttResponseReceived && millis() - startTime <
10000) {
        mqttClient.loop();
    }

    if (mqttResponseReceived) {
        mqttResponseReceived = false;

        // Parse response and do necessary actions...
    }
}

```

```

    }

    // Go back to default state
    noColorRGB(PIN_RED, PIN_GREEN);
    displayScanCard(lcd);

    stopRFID(rfid);
}

```

Cele trei stări posibile în care se poate afla un card (user) sunt:

- valid - Userul este prezent în baza de date și are un abonament valid
- invalid - Userul este prezent în baza de date și are un abonament invalid
- not registered - Userul nu este încă înregistrat

În funcție de starea returnată pe topicul /user_details, programul afișează userului mesaje sugestive:

- valid
 - LCD - Welcome {nume}! {no_attendances} attendances
 - LED RGB - verde
 - Buzzer - sunet de confirmare
- invalid
 - LCD - Access denied
 - LED RGB - roșu
 - Buzzer - sunet de refuz
- not registered → Determină apariția unui prompt de înregistrare a userului în cadrul interfeței site-ului web. Device-ul așteaptă finalizarea înregistrării și datele nou-introduse despre user (ce se poate afla fie în starea validă, fie cea invalidă), pe care le va primi tot pe topicul /user_details. În timpul procesului de așteptare a primirii unui mesaj pe topic, se afișează următoarele:
 - LCD - Not registered! Registering...
 - LED RGB - portocaliu

Configurarea serverului web

Backend-ul este realizat în Flask. Acesta comunică atât cu broker-ul MQTT (și implicit cu dispozitivul IoT), cât și cu frontend-ul. Conexiunea la broker-ul MQTT se realizează, de asemenea, cu ajutorul protocolului TLS și a certificatelor, aşadar transmiterea mesajelor este una criptată și autentificată.

```
def init_mqtt_client():
    global mqtt_client
    mqtt_client = mqtt.Client()
    try:
        mqtt_client.tls_set(
            ca_certs=AWS_CA_CERT,
            certfile=AWS_CERT,
            keyfile=AWS_PRIVATE_KEY,
            tls_version=ssl.PROTOCOL_TLSv1_2,
        )

    except Exception as e:
        raise SystemExit("TLS setup failed. Exiting.")

    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message

    try:
        mqtt_client.connect(AWS_IOT_ENDPOINT, port=8883,
keepalive=60)
    except Exception as e:
        logger.error(f"Error connecting to MQTT broker:
{e}")
```

Serverul este hostat pe platforma Heroku, împreună cu frontend-ul. Frontend-ul a fost realizat în React. Baza de date este una simulată, datele despre useri și prezențele lor fiind stocate într-un dicționar local. Pentru a putea exemplifica funcționalitățile relevante ale proiectului, dicționarul este inițializat cu 15 useri, cu număr de prezențe variate.

Vizualizare și Procesare de Date

Interfața are câteva componente principale:

- buton de închidere/deschidere sală - controlează statusul sălii. Statusul este trimis de la frontend către backend, care publică un mesaj MQTT pe topicul /gym_status. Aceasta e primit de către dispozitiv, care acționează actuatorii corespunzător.

Gym Status: Closed

[Open Gym](#)

Gym Status: Open

[Close Gym](#)

- detalii useri - afișează pe pagina principală un tabel cu userii din baza de date, statusul abonamentului lor (valid/invalid) și numărul total de prezențe. Atunci când un card este citit de către RFID, dispozitivul trimite un mesaj MQTT pe topicul /check_user, cu ID-ul cardului scanat. Serverul procesează mesajul, iar dacă userul este înregistrat și are abonamentul valid, îi incrementează numărul de prezențe. Ulterior, notifică frontend-ul despre update cu ajutorul unui WebSocket, iar pagina primește automat un refresh cu noul attendance count.

Gym Members

Total Users: 15

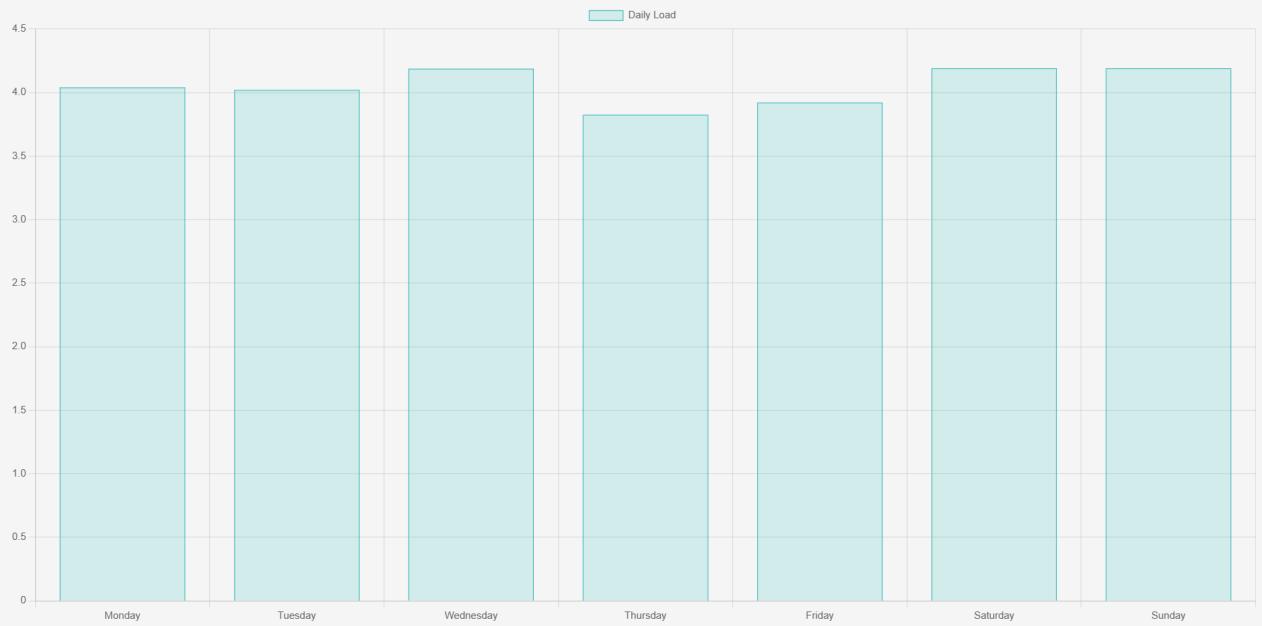
Total Users with Valid Membership: 12

Name	Membership	Attendances
Henry Ford	Invalid	0
Stefania Dumitru	Valid	218
Frank Sinatra	Valid	18
Karen Gillan	Valid	170
Donald Trump	Invalid	6
David Tennant	Invalid	0
Irene Adler	Valid	50
Bob Marley	Valid	51
Grace Hopper	Valid	71
Luna Lovegood	Valid	40
Eleanor Rigby	Valid	250
Jack Sparrow	Valid	168
Clara Oswald	Valid	90
Charlie Brown	Valid	101
Alice Wonderland	Valid	215

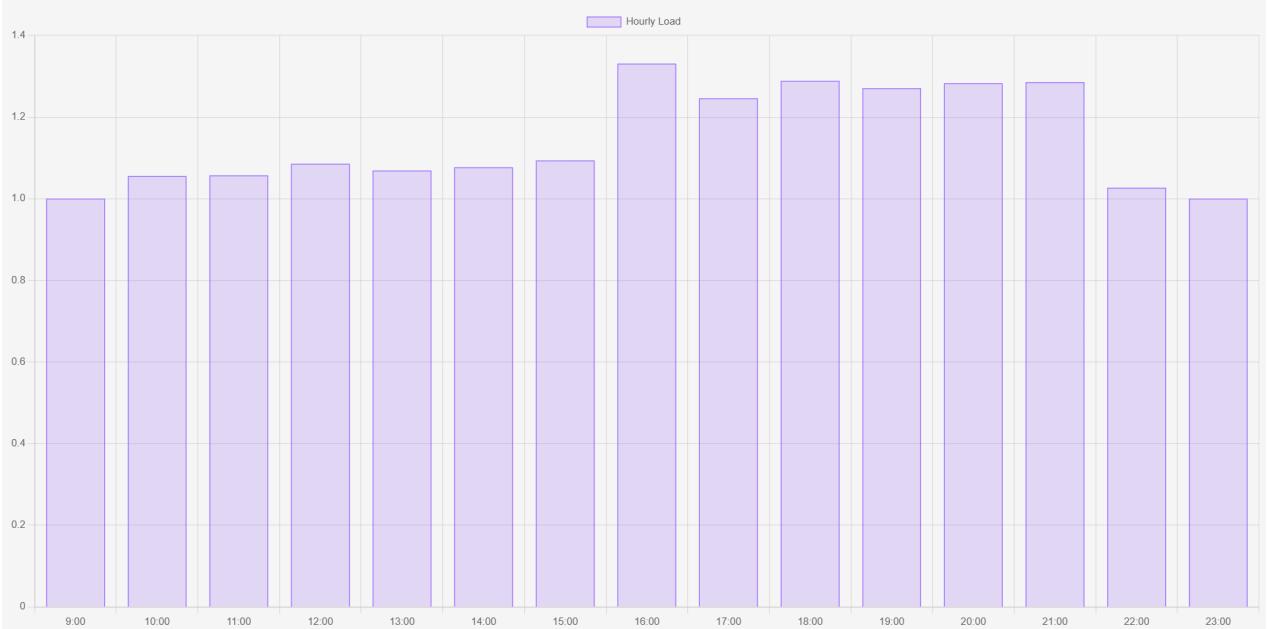
- statistici - afișează pe pagina principala 2 barplots, reprezentând daily si hourly load pentru sala de fitness. Acestea sunt calculate de către server pe baza timestamp-urilor intrărilor userilor. La fel ca la attendance count, și aceste statistici sunt updateate live, odată cu adăugarea fiecărei noi prezențe.

Gym Statistics

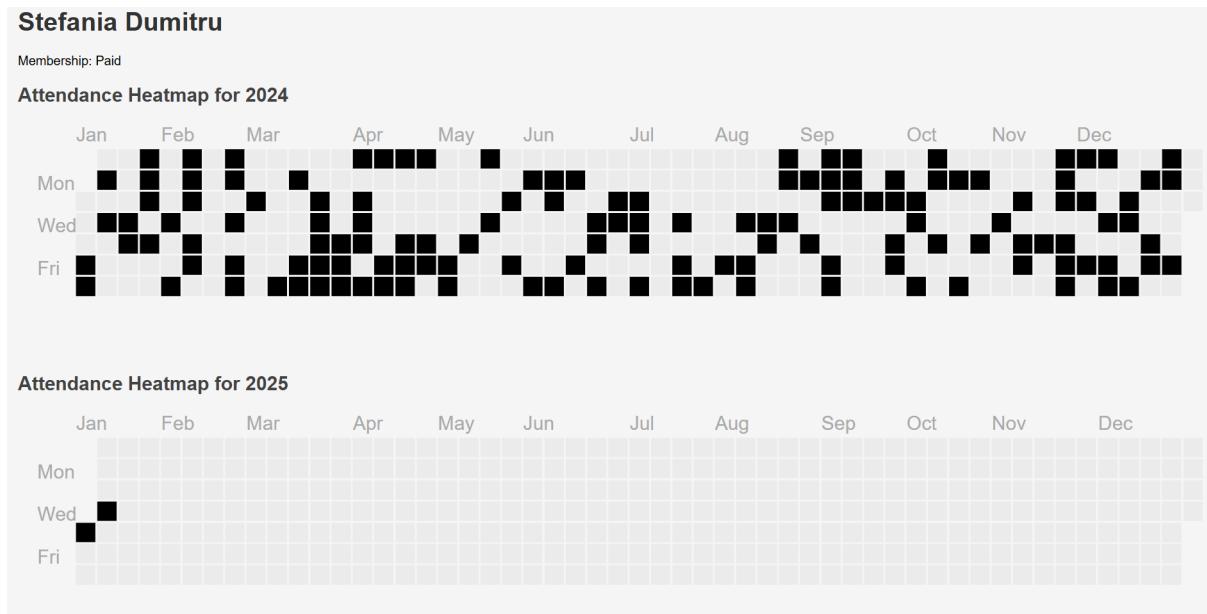
Daily Load (Monday to Sunday)



Hourly Load (9:00 - 23:00)



- pagină individuală user - Dând click pe fiecare user în parte, putem deschide o pagină mai detaliată, care ne prezintă un heatmap cu toate prezențele userului de-a lungul anilor (începând cu anul în care a avut prima prezență), câteva statistici legate de tendințele userului, dar și lista completă a timestamp-urilor când acesta a scanat cardul la intrarea în sală.



Statistics

Total Presence This Week: 0

Total Presence This Month: 4

Average Presence Per Week: 4.02

Average Presence Per Month: 18.08

Most Frequent Day of the Week: Friday

Total Attendances: 217

Attendance List

2024-01-01 - 2024-01-01 21:52:00

2024-01-01 - 2024-01-01 10:19:00

2024-01-05 - 2024-01-05 18:58:00

2024-01-05 - 2024-01-05 10:19:00

2024-01-06 - 2024-01-06 16:42:00

2024-01-08 - 2024-01-08 23:44:00

- register - Atunci când un card neinregistrat este citit de către modulul RFID, se afișează în interfață un prompt de înregistrare al noului user. După înregistrare, detaliile noului user sunt trimise către backend, care le publică pe topicul /user_details, pentru a putea fi primite și afișate de dispozitiv.

The diagram shows a wireframe of a user registration interface. At the top center is the title "Register New User". Below it are two input fields: "First Name" and "Last Name". Underneath these is a label "Paid Membership:" followed by an empty checkbox. A large green rectangular button below is labeled "Register". At the bottom left, there is a small, semi-transparent text area containing the word "Valid".

Securitate

Pentru asigurarea unui nivel ridicat de securitate, comunicarea dintre dispozitivul IoT, server și broker-ul MQTT se realizează folosind protocolul TLS (Transport Layer Security). Acest protocol criptografic garantează atât integritatea, cât și confidențialitatea datelor transmise. Pentru autentificarea entităților implicate, sunt utilizate certificate generate în cadrul AWS IoT Core, inclusiv un certificat CA (Certificate Authority), un certificat client și o cheie privată.

Astfel, toate mesajele trimise între dispozitiv, broker și server sunt criptate și autentificate. Acest lucru reduce riscul de interceptare sau manipulare a datelor, un aspect crucial pentru un sistem ce gestionează informații sensibile despre utilizatori. De asemenea, configurarea serverului și a dispozitivului pentru a folosi aceste certificate a fost o etapă esențială pentru a garanta că doar entitățile autorizate pot participa la comunicare.

Provocări și Soluții

- Integrarea componentelor hardware

Configurarea și sincronizarea componentelor hardware (RFID, LCD, LED RGB, buzzer) pentru a comunica corect între ele și cu broker-ul MQTT a fost o provocare. Unele componente au avut latențe neașteptate, iar debugging-ul

comunicației a fost dificil. Am testat individual fiecare componentă folosind exemple din Arduino IDE, iar apoi am realizat integrarea treptată a acestora. Am configurat timpi de așteptare și funcții de retry pentru a rezolva problemele de latență și comunicare.

- Conectarea dispozitivului și serverului la broker-ul MQTT

Procesul de configurare a broker-ului MQTT pe AWS IoT Core și de conectare a dispozitivului IoT și a serverului la acesta a fost una dintre cele mai dificile etape ale proiectului. Înțelegerea modului în care certificatele trebuie generate și utilizate a necesitat documentare extinsă. Am urmat pași clari din documentația AWS și am testat fiecare componentă individual. Am folosit exemple de configurare și debugging din comunitatea de dezvoltatori, iar verificarea conexiunii prin mesaje de test a fost esențială pentru identificarea problemelor.

- Crearea interfeței web

Realizarea interfeței web în React, astfel încât să fie intuitivă și estetic plăcută, a fost mai dificilă decât anticipam. Integrarea vizualizărilor live, precum statistici și heatmap-uri, și gestionarea datelor dinamic a necesitat multe iterări

- Deployment-ul pe Heroku

Procesul de deployment al serverului și al interfeței web pe Heroku a fost mai dificil decât mă așteptam. Am întâmpinat probleme legate de environment variables și compatibilitatea între pachetele folosite local și cele acceptate de Heroku. De asemenea, integrarea certificatelor pentru conectarea la broker-ul MQTT a generat erori neașteptate în timpul deployment-ului. Am rezolvat problemele prin configurarea corectă a fișierului Procfile, a unui script de deploy și a variabilelor de mediu pe Heroku. Am utilizat și logurile oferite de platformă pentru a identifica rapid cauzele erorilor.

Concluzie

Sunt foarte mulțumită de rezultatul final al proiectului, care reflectă chiar mai multe funcționalități decât ce mi le-am propus inițial. Dacă aş mai lucra pe anumite aspecte (precum optimizarea interfeței web și îmbunătățirea integrării hardware-software), proiectul ar putea fi utilizat într-o sală de fitness reală.