Pădurariu Teofil
Grupa 345C1

Smart Home IoT

INTRODUCERE

Ce presupune?

În contextul creșterii interesului pentru automatizarea locuințelor și integrarea tehnologiilor inteligente în viața de zi cu zi, acest proiect se concentrează pe dezvoltarea unui sistem IoT (Internet of Things) complet funcțional. Scopul său este să ofere utilizatorilor o soluție accesibilă și eficientă pentru monitorizarea și controlul condițiilor de mediu și al dispozitivelor smart dintr-o locuință. Proiectul adresează atât nevoile de confort și personalizare, cât și cerințele de siguranță prin detecția în timp real a potențialelor riscuri precum scurgerile de gaze sau incendiile.

Sistemul este conceput să integreze un **hub central de senzori și actuatori** care colectează date din mediul înconjurător, le procesează și le transmite către un server prin protocolul MQTT (Message Queuing Telemetry Transport). Această soluție asigură o transmisie rapidă și eficientă a datelor, permițând integrarea unor multiple dispozitive și scalabilitatea sistemului. Datele colectate sunt apoi accesibile utilizatorilor printr-o **aplicație web interactivă**, care oferă funcții intuitive de monitorizare și control.

Obiectivele principale ale proiectului

1. Monitorizarea condițiilor de mediu

Sistemul colectează date în timp real despre:

- Temperatura și umiditatea aerului din interior
- Detectarea scurgerilor de gaze sau a fumului, pentru a crește siguranța utilizatorilor

2. Controlul automatizat și personalizat al luminii

- Sistemul utilizează LED-uri RGB pentru a simula iluminatul inteligent, permițând ajustarea intensității luminii în funcție de preferințe sau de condițiile ambientale. Aplicația web va permite controlul luminii.

3. **Controlul sistemului de climatizare**
    - Printr-un ventilator controlabil, sistemul simulează funcțiile de racire ale unui aer conditionat. Pornirea și oprirea acestuia se poate realiza în automat pe baza temperaturii sau manual, folosind aplicația web de control

4. **Control și monitorizare prin aplicația web**

    Utilizatorii vor avea acces la o aplicație web care oferă:

    - Afișarea datelor colectate de senzori (temperatură, umiditate, starea senzorilor de gaz).
    - Controlul dispozitivelor (ventilator, iluminat) și personalizarea setărilor acestora.
    - Notificări în timp real în cazul detecției unui pericol, cum ar fi scurgerile de gaz

Senzori

1. **DHT22 (Temperatură și umiditate)**
    - Acest senzor oferă măsurători precise ale temperaturii și umidității, informații esențiale pentru controlul sistemului de climatizare.

2. **MQ-2 (Detectarea gazelor)**
    - Senzorul MQ-2 detectează scurgerile de gaze inflamabile, cum ar fi metanul, propanul sau GPL, prevenind posibile incidente.
    - A fost simulat folosind un potențiometru

Actuatori

1. **LED-uri RGB**
    - Vor simula sursele de lumină smart dintr-o locuință

2. **Ventilator DC**
    - Va simula un aer condiționat care va regla temperatura pe baza datelor colectate despre aceasta sau în mod manual, folosind aplicația de control

ARHITECTURĂ

Sistemul folosește protocolul MQTT pentru a comunica eficient între senzori, actuatori și server, asigurându-se astfel o transmisie rapidă și fiabilă a datelor. Datele vor fi stocate într-o bază de date (Firebase), pe care o va folosi și aplicația web, care folosește HTTPS pentru comunicarea cu serverul.

Topologia rețelei

1. Senzori

- Acestea sunt dispozitivele care colectează date din mediu, precum temperatură, umiditate, gaze. Senzorii sunt conectați la un hub central care preia informațiile și le trimite mai departe.

2. Hubul central (ESP32)

- Acesta este componenta care coordonează comunicarea între senzorii locali și server. Hub-ul se conectează la rețea prin intermediul Wi-Fi Ethernet și trimite datele sensibile către server folosind MQTT. Hubul este un **ESP32**.

3. Actuatori

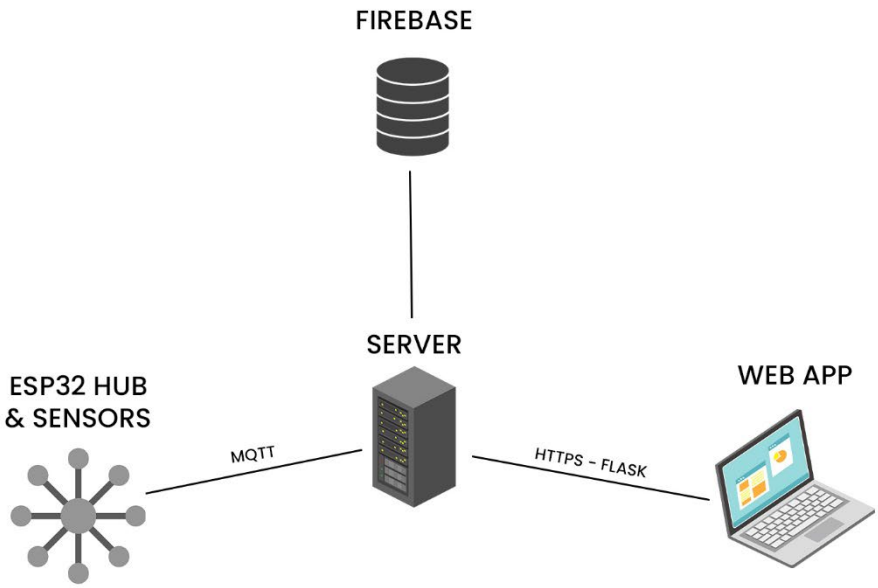
- LED-urile RGB și ventilatorul DC sunt componentele care răspund la comenzile primite de la server sau aplicația web. Ele reglează iluminatul și climatizarea în funcție de datele preluate de la senzori.

4. Serverul (Flask & MQTT)

- Serverul are două componente:
- gestionează procesarea și stocarea datelor primite de la hub, comunicând cu acesta prin MQTT
- se ocupă de gestionarea cererilor de control venite din partea aplicației web (cereri HTTP)

5. **Aplicația web**
    - Aceasta interacționează cu utilizatorul și permite monitorizarea și controlul tuturor dispozitivelor din rețea. Aplicația primește date în timp real de la server și trimite comenzi pentru controlul actuatorilor (lumini, ventilator).
6. **Firebase**
    - Datele vor fi stocate în timp real, într-o bază de date, folosind Firebase.

Diagrama topologiei rețelei



Protocoale de comunicație

1. **Protocolul MQTT**

    MQTT este un protocol de mesagerie ușor, proiectat pentru aplicațiile IoT și utilizat pentru comunicațiile între hubul de senzori și server. Este optimizat pentru dispozitive cu resurse limitate și pentru medii cu conexiuni de rețea instabile.

Rolul său în proiect:

- **Transmiterea datelor senzorilor:** Senzorii de temperatură, umiditate, gaz trimit date în topicuri MQTT către server.
- **Controlul actuatorilor:** Serverul publică comenzi (de exemplu, pentru LED-uri sau ventilator) către topicurile actuatorilor.
- **Comunicarea bidirecțională:** Hubul și serverul pot comunica eficient pentru a transmite și primi date sau comenzi.

2. Protocolul HTTPS

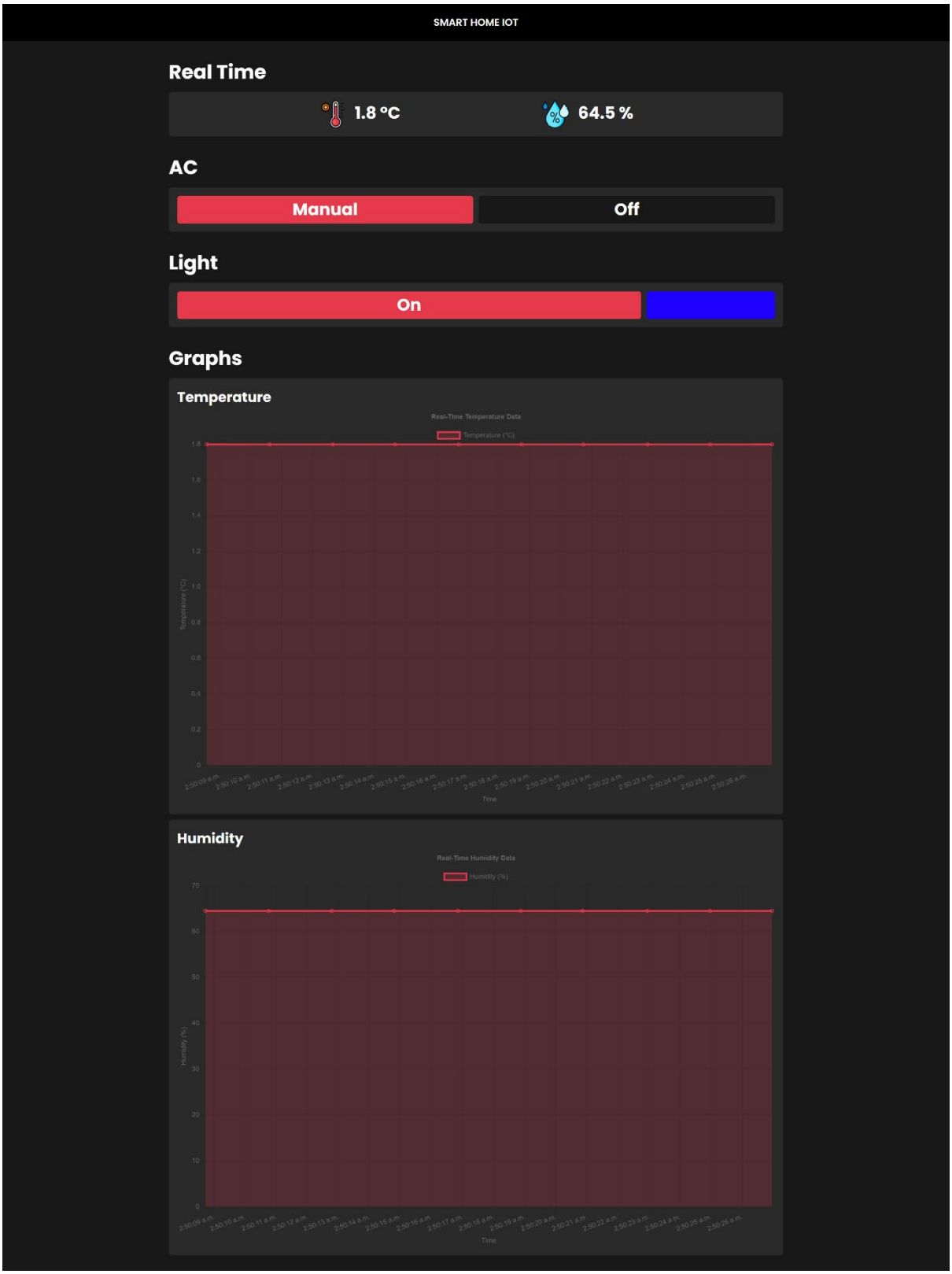
HTTPS este versiunea securizată a HTTP, utilizată pentru comunicația între aplicația web și server. Acesta garantează criptarea datelor transmise și asigură confidențialitatea și integritatea informațiilor.

Rolul său în proiect :

- **Comunicare între aplicația web și server:** Requesturile trimise de aplicația web (de exemplu, comenzi de control pentru dispozitivele IoT) sunt transmise în siguranță către server folosind HTTPS.
- **Transmiterea datelor către utilizator:** Serverul transmite datele colectate de la senzori către aplicația web, oferind utilizatorului informații în timp real despre starea locuinței.

Implementarea nu a permis folosirea unui self-signed certificate pentru realizarea comunicării prin HTTPS, însă cu un certificat autorizat lucrurile pot merge și în acest fel.

Prototip aplicație web



IMPLEMENTARE

Hardware

1. Cum?

- Partea de hardware am implementat-o folosind Wokwi și PlatformIO.
- **Wokwi** este o platformă online care permite simularea și testarea proiectelor electronice bazate pe microcontrolere, cum ar fi ESP32, direct în browser, fără a necesita hardware fizic, fiind ideală pentru prototipare și învățare. Am utilizat varianta integrată în Visual Studio Code.
- **PlatformIO** este un mediu de dezvoltare integrată (IDE) pentru microcontrolere, disponibil ca extensie pentru editoare precum Visual Studio Code, care simplifică programarea, compilarea și gestionarea proiectelor pentru o gamă variată de plăci și platforme.
- PlatformIO compilează codul, iar binarul rezultat este folosit de Wokwi pentru a rula simularea. Fișierul *wokwi.toml* a făcut legătura dintre binar și Wokwi.
- Pentru rulare, se utilizează butonul play din interfața grafică a aplicației.

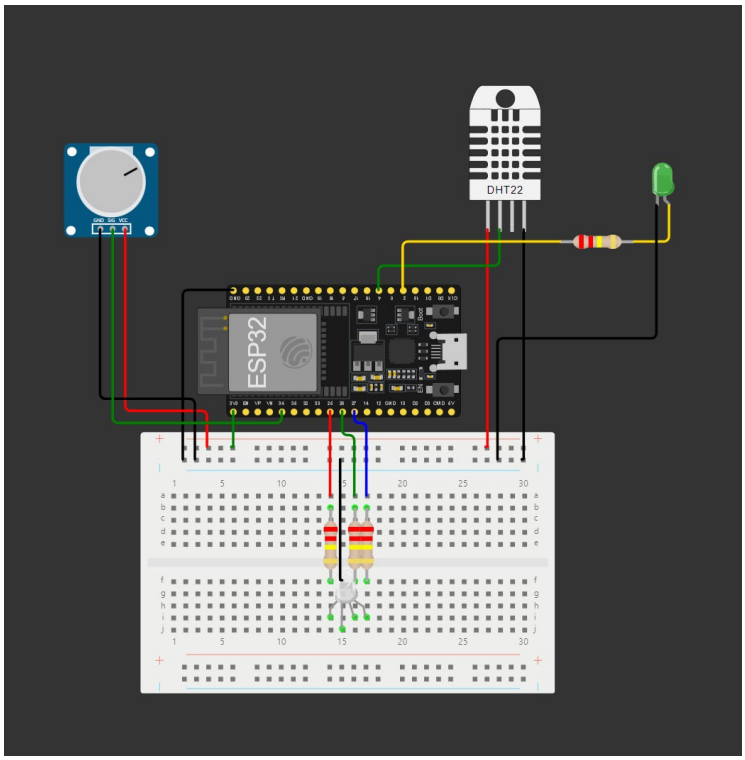
2. Componentele utilizate:

- **ESP32:** Microcontroller-ul (Hub-ul central) care controlează și colectează date de la toate celelalte componente.
- **DH22:** Senzorul de temperatură și umiditate. Temperatura și umiditatea pot fi modificate din interfața simulatorului, prin sliderele puse la dispoziție de acesta printr-un click pe senzor.
- **RGB LED:** Led-ul care simulează o sursă de lumină a locuinței
- **LED:** Simulează un posibil aer condiționat al unei locuințe
- **Potențiometru:** Simulează senzorul de gaz. Valorile sale analogice sunt echivalentul celor pe care le-ar scoate senzorul de gaz. Output-ul potențiometrului poate fi reglat din interfața simulatorului printr-un slider.
- **Rezistori de 220Ohmi:** Pentru LED-uri

3. Wiring

- **Potentiometru:**
- Pinul central al potentiometrului este conectat la pinul analogic 34 al ESP32.
- Celelalte două picioare sunt conectate la GND și VCC (3.3V).
- **DHT22:**
- Alimentare: VCC conectat la 3.3V și GND la masă.
- Date: Pinul de date al senzorului este conectat la pinul 4 al microcontroller-ului, care a fost setat din cod ca pin de intrare.
- **LED:**
- Un catod (picior negativ) al LED-ului este conectat la GND.
- Anodul (picior pozitiv) este conectat la pinul digital 2 al ESP32.
- **LED RGB:**
- Catodul (piciorul negativ) al LED-ului este conectat la GND.
- Picioarele corespunzătoare culorilor roșu, verde și albastru au fost conectate în serie cu o câte o rezistență de 220ohmi la pinii digitali 25, 26, respectiv 27 ai ESP32.

4. Schema finală



Software

1. Alcătuire

- Componenta software presupune 3 sub-componente:
  1. Codul care va rula pe ESP32
  2. Codul pentru serverul intermediar (backend-ul)
  3. Codul pentru client (frontend-ul)

2. Broker-ul online și topic-urile utilizate în comunicația ESP32 – Server Central

- ESP32 comunică prin protocolul MQTT cu serverul central cu ajutorul unui broker online: https://www.hivemq.com/
- Am creat un cont pe acest website și am creat gratuit un server reprezentat de un url public folosit în codul ESP32 și în cel al serverului pentru a putea fi realizată comunicația.
- Există 6 topic-uri:
  1. **smart-home/led**: mesaje specifice led-ului RGB
  2. **smart-home/dht-data**: pentru mesaje specifice senzorului de temperatură și de umiditate
  3. **smart-home/fan/mode**: pentru mesaje specifice modului de funcționare al ventilatorului: AUTO/MANUAL
  4. **smart-home/fan/state**: pentru mesaje specifice stării ledului care simulează AC-ul
  5. **smart-home/gas**: pentru mesaje specifice senzorului de gaz
  6. **smart-home/init**: pentru sincronizarea întregului sistem la pornirea ESP32

3. ESP32

- Așa cum am menționat anterior, ESP32 controlează sau preia date de la toate cele 4 componente conectate la el pe pinii enunțați în subcapitolul Wiring
- Comunicația cu serverul centeral o realizează prin protocolul MQTT, peste protocolul WiFi (ssid și parolă specifice simulatorului)

1. **Pornirea și inițializarea ESP32**

   După pornirea ESP32, programul începe cu funcția **setup()** care are rolul de a inițializa toate componentele hardware și conexiunile necesare pentru funcționarea dispozitivului. Aceasta include:

   - Inițializarea comunicării seriale cu Serial.begin(115200) pentru depanare și mesaje informative.
   - Configurarea pinilor GPIO ca ieșiri (pentru LED-uri RGB și ventilator) sau intrări (pentru senzorul de gaz) folosind pinMode.
   - Conectarea la rețeaua Wi-Fi, care încearcă să stabilească o conexiune cu rețeaua specificată prin ssid și password.
   - Configurarea clientului MQTT cu funcția mqtt_connect(), unde ESP32 se conectează la un broker MQTT definit și se abonează la toate cele 6 topicuri.
   - Inițializarea senzorului de temperatură și umiditate DHT22
   - Publicarea stării inițiale a sistemului prin subiectul MQTT init_topic: În firebase există un collection rtstate care conține un document actuators ce deține informațiile realtime ale sistemului (led on/off, ac running, etc). La pornire se inițializează aceste date cu stările inițiale ale senzorilor și actuatorilor atașați la ESP32.

2. **Bucla principală de rulare (loop())**

   După inițializare, ESP32 intră într-o buclă infinită în cadrul funcției loop(), care gestionează activitățile periodice și asigură conectivitatea:

   - Reconectează clientul MQTT dacă legătura este pierdută, apelând din nou funcția mqtt_connect().
   - Ascultă și procesează mesajele primite pe subiectele MQTT prin client.loop().
   - La intervale de 2 secunde, senzorii sunt citiți:
     - Senzorul DHT22 este citit cu read_dht_sensor() pentru a obține temperatura și umiditatea. Valorile sunt publicate pe subiectul dht_topic, iar ventilatorul este controlat automat dacă modul automat este activat (fan_mode == AUTO).
     - Senzorul de gaz este citit cu read_gas_sensor() și datele sunt publicate pe subiectul gas_sensor_topic.

3. **Gestionarea mesajelor primite**
   Callback-ul on_message() este apelat ori de câte ori ESP32 primește un mesaj MQTT. Aceasta procesează comenzile în funcție de subiect:

   - Subiectul led_topic: Controlează LED-ul RGB. Poate schimba culoarea sau poate activa/dezactiva LED-ul.
   - Subiectul fan_state_topic: Pornește sau oprește ventilatorul în funcție de mesajele "on" sau "off". Este de asemenea folosit pentru a trimite starea ventilatorului în modul auto, către server pentru a îi fi cunoscută starea de către baza de date
   - Subiectul fan_mode_topic: Schimbă modul ventilatorului între automat (AUTO) și manual (MANUAL).

4. Serverul intermediar

- Serverul intermediar este componenta centrală a întreg sistemului. El comunică cu ESP32 prin protocolul MQTT și este de asemenea și un server HTTP. Astfel prin intermediul său se pot trimite mesaje de la o aplicație web de control către ESP32.
- Serverul este de asemenea responsabil de a menține totul real time în baza de date.

1. **Inițializarea serverului**

La pornire, scriptul inițializează mai multe componente critice:

- **Broker-ul MQTT**
- **Firebase Firestore:** Se conectează la o bază de date Firestore utilizând un fișier de autentificare (serviceAccountKey.json), permițând stocarea datelor despre senzori și starea actuatoarelor.
- **Flask:** Server web care rulează API-urile REST pentru controlul manual al dispozitivelor.

2. **Conexiunea la MQTT**

În funcția on_connect, serverul se abonează la subiectele relevante (de exemplu, FAN_STATE_TOPIC, DHT22_TOPIC, GAS_TOPIC, etc.) pentru a primi date despre starea actuatoarelor și senzorilor.

Funcția on_message procesează mesajele primite și execută acțiuni în funcție de subiect:

- **Temperatura și umiditatea (DHT22_TOPIC):** Stochează valorile în Firestore și limitează numărul de citiri salvate (maxim 10 citiri recente).
- **Gaz (GAS_TOPIC):** Dacă valoarea detectată depășește un prag (2000), trimite o notificare prin Telegram.
- **Ventilator (FAN_STATE_TOPIC):** Actualizează starea ventilatorului în baza de date.
- **Inițializare (INIT_TOPIC):** Actualizează toate stările actuatoarelor (LED, ventilator, modul ventilator) pe baza informațiilor inițiale primite de la ESP32.

3. **API-uri REST**

Serverul expune mai multe endpoint-uri pentru controlul manual al sistemului:

- **/api/fan/state**: Controlează starea ventilatorului (on/off). Mesajele sunt publicate pe subiectul MQTT FAN_STATE_TOPIC.
- **/api/fan/mode**: Schimbă modul ventilatorului între automat (auto) și manual (manual), actualizând subiectul FAN_MODE_TOPIC.
- **/api/light/state**: Controlează starea LED-ului RGB (on/off) și publică comanda pe LED_TOPIC.
- **/api/light/color**: Schimbă culoarea LED-ului RGB prin trimiterea unei comenzi cu valorile R, G, B pe LED_TOPIC.

4. **Alte funcționalități**

- **Integrarea Telegram:** Permite trimiterea notificărilor de tip alertă pentru detectarea gazului printr-un bot Telegram.
- **Gestionarea stărilor actuatoarelor:** Funcții dedicate (updateActuators, updateColors) actualizează baza de date Firestore pentru a menține totul în real time în baza de date

5. Clientul

- Aplicație web prin intermediul căreia pot fi vizualizate datele în timp real și prin intermediul căreia actuatorii pot fi controlați de la distanță.

1. Interfața grafică: smart-home.html & smart-home.css

Fișierul HTML definește structura vizuală a aplicației.
Secțiuni principale:

- **Date în timp real:** Afișează temperatura și umiditatea curentă.
- **Controlul ventilatorului (AC):** Permite schimbarea modului între automat și manual, precum și pornirea/oprirea ventilatorului.
- **Controlul iluminatului:** Include un buton pentru activarea/dezactivarea LED-ului RGB și un selector de culoare.
- **Grafice:** Afișează date istorice despre temperatură și umiditate sub formă de grafice liniare generate cu Chart.js.
- **Resurse externe:**
  - Chart.js pentru grafice.
  - Google Fonts pentru stilizarea textului.

2. Scriptul principal: smart-home.js

Scriptul JavaScript implementează logica funcțională pentru interacțiunea cu serverul și actualizarea interfeței utilizatorului:

- **Inițializare Firebase:** Utilizează Firebase pentru a accesa baza de date în timp real. Stările dispozitivelor sunt gestionate prin documentul *actuators* din colecția *rtstate*.
- **Controlul dispozitivelor:**
  a. Ventilator: Endpoint-ul */api/fan/state* este utilizat pentru pornirea/oprirea ventilatorului. Endpoint-ul */api/fan/mode* permite schimbarea modului ventilatorului între manual și auto.
  b. Iluminat LED RGB: */api/light/state* controlează starea LED-ului (on/off). */api/light/color* permite schimbarea culorii prin selectarea unei valori RGB.
- **Actualizarea stărilor în UI:** Funcțiile setAC() și setLight() sincronizează stările ventilatorului și LED-ului cu valorile din Firebase.
- **Manipularea evenimentelor:** Click-urile pe butoane sau schimbările de culoare declanșează apeluri API către server.

3. Graficarea datelor: charts.js

Acest script gestionează generarea și actualizarea graficelor de temperatură și umiditate:

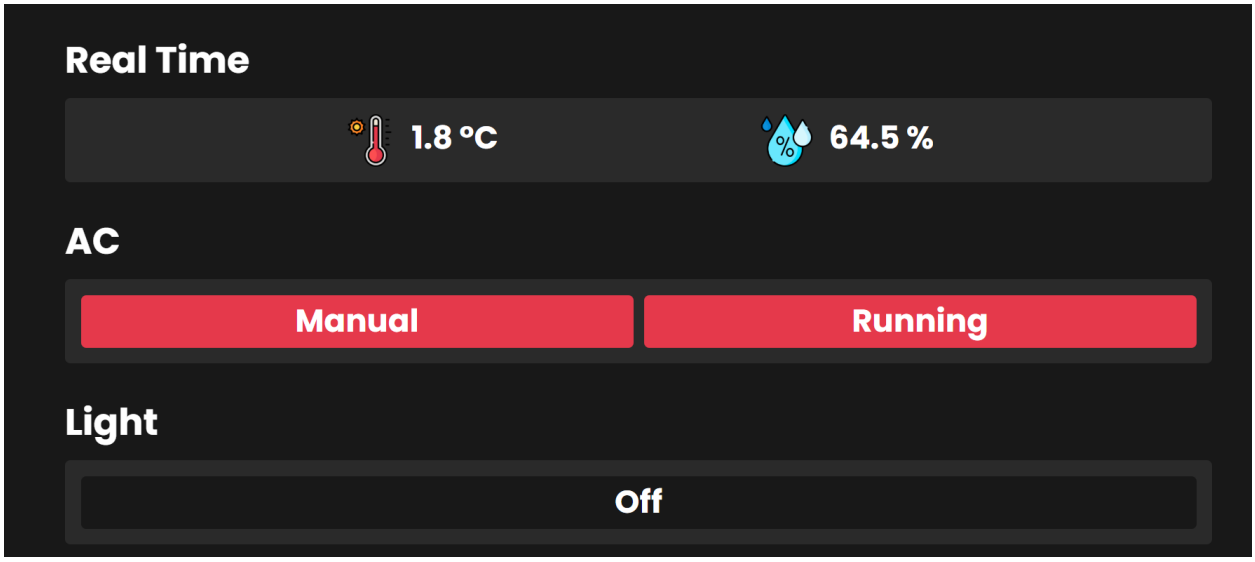
- **Configurare grafice**: Creează grafice liniare folosind Chart.js pentru afișarea datelor istorice. Datele sunt afișate pe axe temperatura(timestamp), respectiv umiditate(timestamp)
- **Obținerea datelor în timp real:** Utilizează Firebase pentru a monitoriza documentele *temperature* și *humidity* din colecția *sensor*. Datele noi sunt actualizate automat în grafice.
- **Valori curente:** Valorile cele mai recente de temperatură și umiditate sunt afișate în secțiunea „Real Time”.

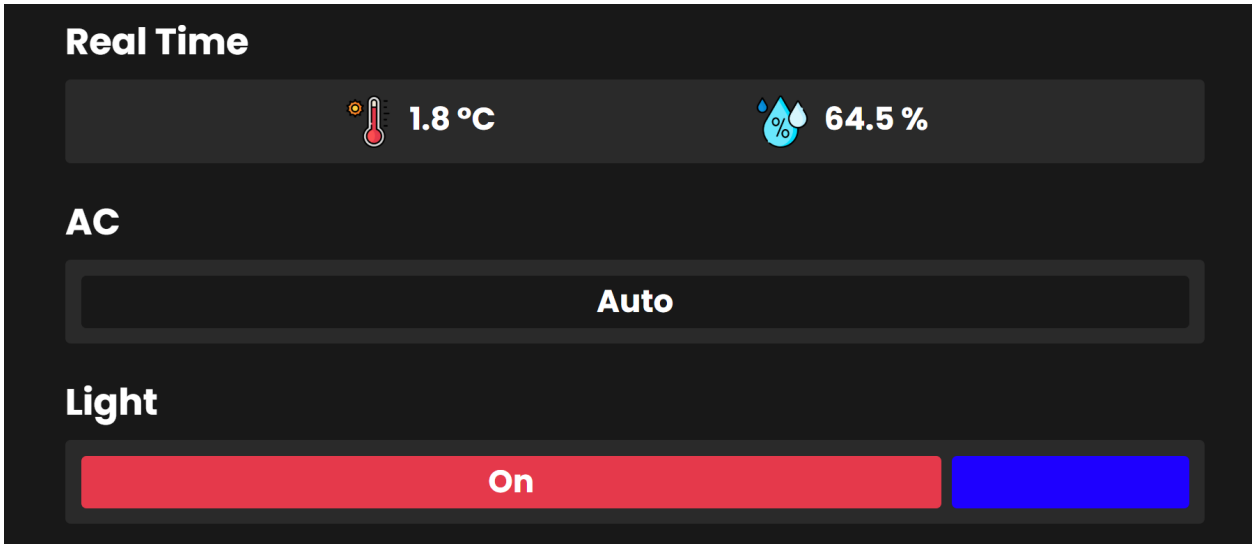
4. Fluxul de funcționare al aplicației

La încărcare, aplicația:

1. Inițializează conexiunea Firebase și obține stările dispozitivelor.
2. Configurează graficul pentru datele de temperatură și umiditate.
3. Interacțiunile utilizatorului (ex. apăsarea butonului pentru ventilator) sunt trimise către server prin endpoint-uri API REST.
4. Modificările din Firebase sunt sincronizate automat cu interfața utilizatorului, inclusiv actualizarea datelor în timp real și a graficelor.

5. UI snapshots:





6. Firestore & firebase

- Firebase este o platformă backend utilizată în acest sistem pentru stocarea și sincronizarea datelor în timp real, oferind suport esențial pentru monitorizarea și controlul dispozitivelor IoT.

1. Componente Firebase utilizate

Firestore (Baza de date NoSQL):

- Firestore este folosit pentru a stoca și sincroniza datele despre senzori și stările actuatoarelor.
- Structura colecțiilor și documentelor:
  - sensor/temperature: Conține date istorice despre temperatură, fiecare citire fiind stocată cu un timestamp.
  - sensor/humidity: Similar, stochează date despre umiditate.
  - rtstate/actuators: Conține stările actuale ale dispozitivelor, cum ar fi LED-ul (pornit/oprit), ventilatorul (mod manual/automat) sau culoarea LED-ului RGB.
- Reguli în timp real: Funcția onSnapshot este utilizată pentru a asculta modificările în baza de date și a sincroniza interfața utilizatorului (UI) cu datele actuale.

2. **Actualizarea și gestionarea datelor**
    - Datele senzorilor (temperatură și umiditate) sunt trimise de ESP32 către serverul backend, care le salvează în Firestore.
    - În documentele sensor/temperature și sensor/humidity, citirile recente sunt păstrate (maxim 10) pentru optimizarea utilizării spațiului și a performanței.
    - Serverul actualizează documentul rtstate/actuators atunci când starea unui dispozitiv este schimbată (ex. pornirea ventilatorului sau schimbarea culorii LED-ului RGB).
    - Aplicația client monitorizează acest document pentru a reflecta instantaneu modificările în UI.

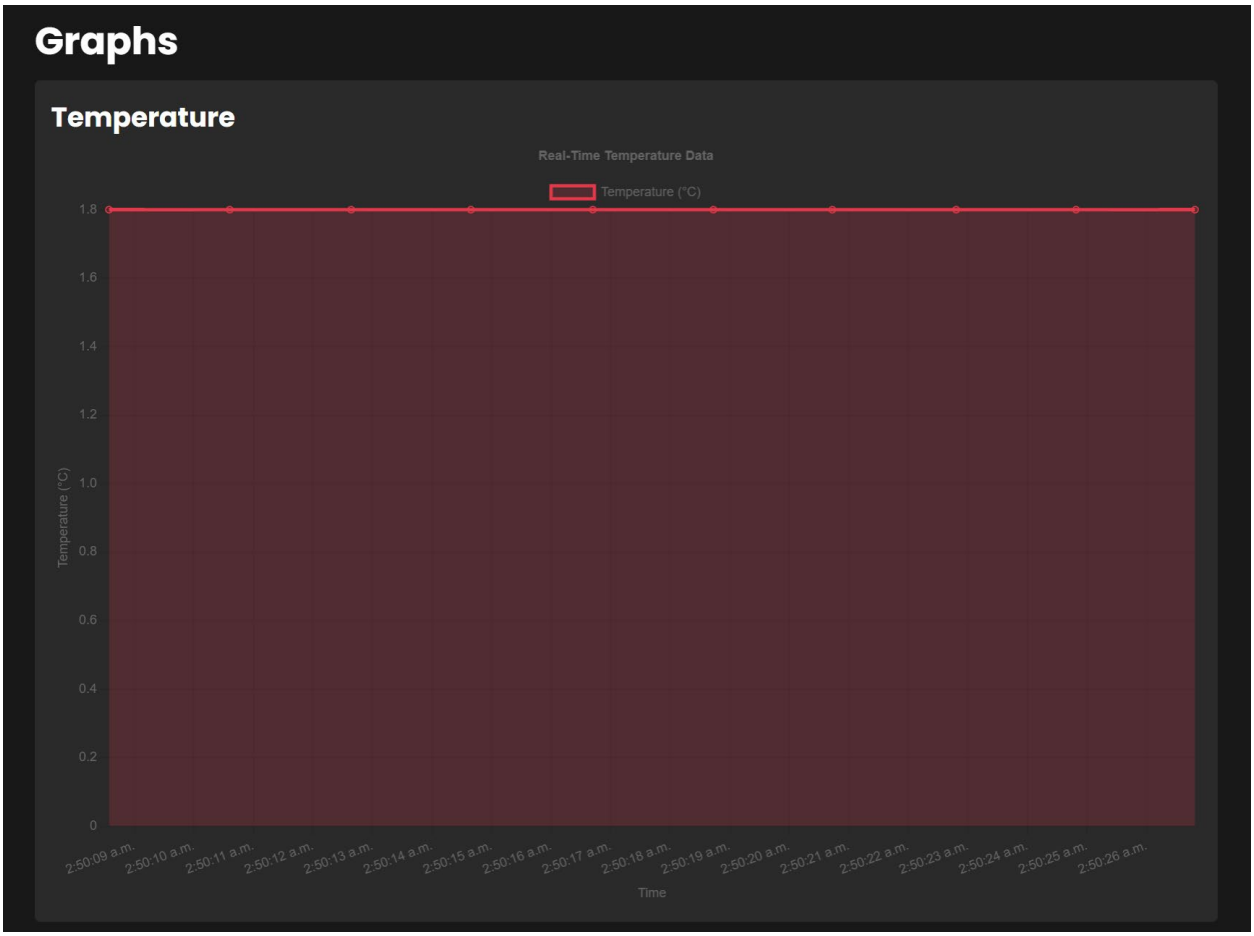
3. **Avantajele utilizării Firebase**
    - Orice modificare în baza de date este reflectată imediat în interfața utilizatorului, fără a fi necesară reîncărcarea paginii.
    - Accesul la baza de date este controlat prin reguli specifice definite în consola Firebase.

VIZUALIZARE ȘI PROCESARE DE DATE

Vizualizarea datelor

- Vizualizarea datelor este realizată printr-o interfață grafică intuitivă, care include:
- Temperatura și umiditatea curente sunt afișate într-o secțiune dedicată („Real Time”) din aplicația web.
- Temperaturile și valorile de umiditate istorice sunt prezentate sub formă de grafice liniare folosind Chart.js.
- Graficele sunt configurate să afișeze axe de timp (X), cu unități în secunde și minute, axe de valori (Y), adaptate intervalului specific al senzorului și date noi, care sunt adăugate automat fără a reîncărca pagina.
- Utilizatorii pot monitoriza stările dispozitivelor (LED-uri, ventilator) și datele senzorilor într-un mod vizual și ușor de interpretat.

Screenshot grafice



Procesare Date

- Datele sunt procesate de componentele descrise în amănunt în secțiunile anterioare. Firebase este cea mai importantă componentă în vizualizarea datelor fiind real time.

SECURITATE

Securitatea comunicării MQTT

Protocolul MQTT este utilizat pentru schimbul de mesaje între dispozitive și server. În cadrul proiectului, s-au implementat următoarele măsuri de securitate pentru a asigura integritatea și confidențialitatea datelor transmise:

- Comunicarea MQTT este securizată prin TLS (Transport Layer Security), folosind un certificat CA validat.
- Clientul MQTT utilizează un username (smart-home-iot) și o parolă pentru autentificare. Acest proces asigură că doar dispozitivele autorizate pot accesa brokerul.
- Serverul și dispozitivele IoT sunt configurate să subscrie doar la subiectele relevante (smart-home/dht-data, smart-home/fan/state, etc.), limitând astfel suprafața de atac.
- În cazul unei pierderi de conexiune, clientul MQTT încearcă reconectarea, reducând șansele ca atacatorii să exploateze întreruperile.

Securitatea datelor în Firebase

Securitatea Firebase este asigurată astfel:

- Accesul la baza de date este restricționat prin reguli specifice, definite în consola Firebase.
- Utilizatorii autentificați pot citi doar datele lor, dacă este cazul.
- Aplicația client utilizează un set unic de chei API pentru conectarea la Firebase, prevenind accesul neautorizat.