Practică

Sistem de monitorizare și iluminare inteligent utilizând o interfață utilizator

Cioponea Florentina-Teodora

Cuprins

Monitorizarea și iluminarea inteligentă utilizând o interfață utilizator ..... **Error! Bookmark not defined.**
Resurse Hardware..... **Error! Bookmark not defined.**
Resurse software 21
Implementare hardware..... 23
Principiul de funcționare al întregului ansamblu..... 28
Implementare software 30
Codul implementat pe modulul ESP32 31
Concluzii 48

Descrierea temei

Monitorizarea și iluminarea inteligentă utilizând o interfață utilizator

Aplicația utilizează un microcontroler de tip ESP32, care controlează luminozitatea și culoarea luminii într-o încăpere. Controlul poate fi realizat atât manual, prin intermediul unei interfețe accesate de utilizator, cât și automat, în funcție de ora din zi și de prezența în cameră. Sistemul menține lumina activă atâta timp cât senzorii detectează mișcare în incintă.

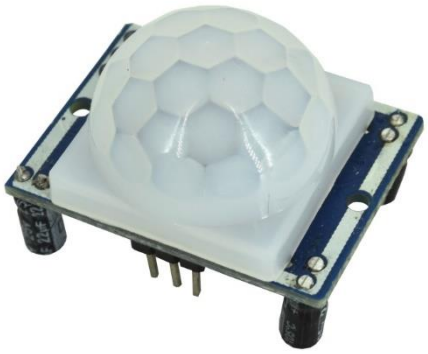
Pentru a afișa timpul pe interfața utilizator și pentru a gestiona funcționalitățile aplicației, am utilizat un modul RTC cu baterie dedicată, prevenind astfel desincronizarea cu fusul orar al României.

Aplicația monitorizează prezența în cameră cu ajutorul senzorilor de mișcare, iar datele privind accesul în incintă (inclusiv ora la care s-a detectat mișcare) sunt stocate local pe un card MicroSD.

Resurse Hardware

Senzori de mișcare

PIR HC-SR501



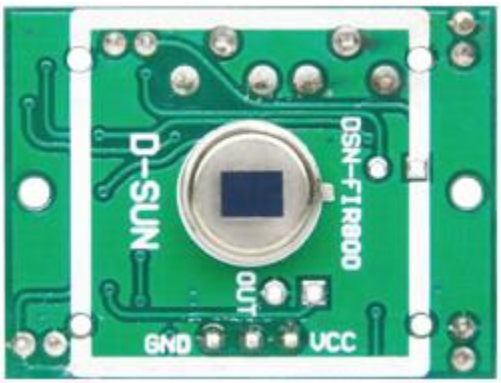
Acest senzor **conține un infraroșu pasiv**, are un material subțire de film piroelectric care răspunde la radiațiile infraroșii de unde rezultă electricitatea. Este un senzor economic ce are o durată mare de viață.

Toate obiectele, inclusiv corpul uman, aflate la temperaturi peste zero absolut (0 Kelvin / -273,15 °C) emit energie termică sub formă de radiații infraroșii. Cu cât un obiect este mai fierbinte, cu atât emite mai multe radiații. Această radiație nu este vizibilă pentru ochiul uman deoarece este emisă la lungimi de undă infraroșii. Acest senzor este special conceput pentru a detecta astfel de niveluri de radiații infraroșii.

Un **senzor PIR** constă din două părți principale:

Un **senzor piroelectric**, pe care îl puteți vedea în imaginea de mai jos ca un metal rotund cu un cristal dreptunghiular în centru**.**

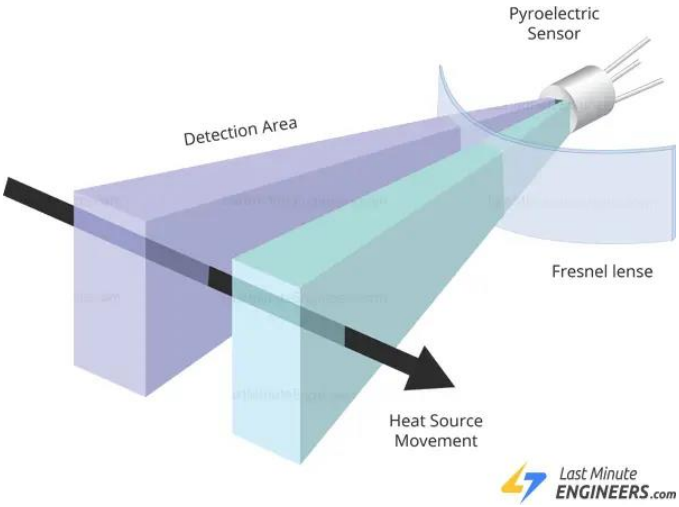
O lentilă specială numită lentilă Fresnel care focalizează semnalele infraroșii pe senzorul piroelectric. Această lentilă este cea care mărește raza și câmpul de vizualizare ale senzorului PIR. Construcția sa subțire și ușoară dar și capacitatea excelentă de colectare a luminii o fac extrem de utilă pentru a face PIR-uri de dimensiuni mici, dar puternice.



Un senzor piroelectric constă dintr-o fereastră cu două fante dreptunghiulare și este fabricat dintr-un material (de obicei silicon acoperit) care permite trecerea radiațiilor infraroșii. În spatele ferestrei se află doi electrozi separați ai senzorului infraroșu, unul responsabil de producerea semnalului pozitiv și celălalt de producerea semnalului negativ.

Cei doi electrozi sunt cablați astfel încât să se anuleze reciproc. Acest lucru se datorează faptului că noi căutăm modificări ale nivelului de infraroșu și nu ale

nivelului de infraroșu ambiental. Acesta este motivul pentru care atunci când o jumătate vede mai multă sau mai puțină radiație IR decât cealaltă, obținem ieșirea.

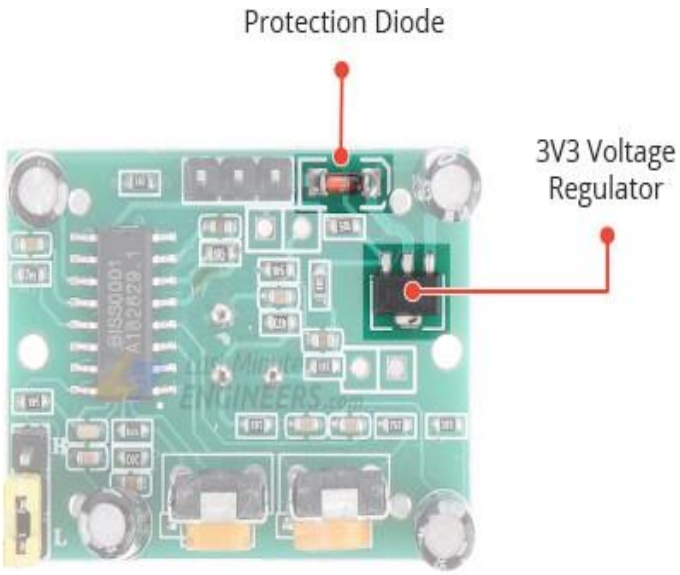


Atunci când nu există nicio mișcare în jurul senzorului, ambele fante detectează aceeași cantitate de radiații infraroșii, rezultând un semnal de ieșire zero.

Dar atunci când trece un corp cald, cum ar fi un om sau un animal, acesta interceptează mai întâi jumătate din senzor. Acest lucru determină o schimbare diferențială pozitivă între cele două jumătăți. Atunci când corpul cald interceptează cealaltă jumătate a senzorului (părăsește regiunea de detectare), se întâmplă invers, iar senzorul produce o variație diferențială negativă. Prin citirea acestei schimbări de tensiune, se detectează mișcarea.

Modulul este prevăzut cu un regulator de tensiune de precizie de 3,3 V, astfel încât poate fi alimentat de orice tensiune continuă de la 4,5 la 12 volți, deși 5V este utilizat în mod obișnuit.

Modulul este prevăzut cu o diodă de protecție (cunoscută și sub denumirea de diodă de siguranță) pentru a proteja modulul de tensiunea și curentul invers.

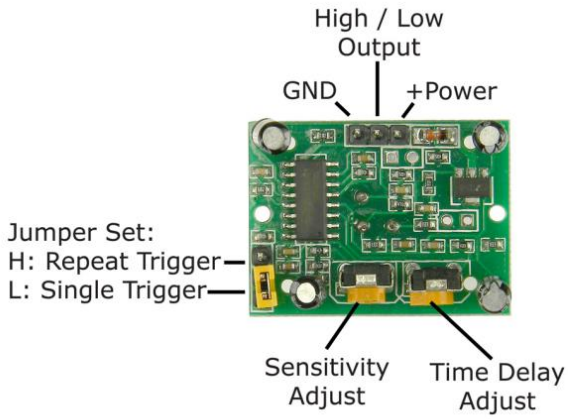


Senzorul PIR are un potențiometru pe spate pentru a regla sensibilitatea dar si timpul de delay. Există două moduri de declanșare care determină modul în care senzorul va reacționa atunci când este detectată o mișcare.

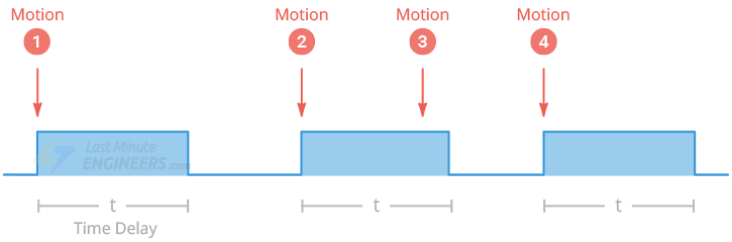
Modul de declanșare unică: Mișcarea constantă va provoca o singură declanșare.

Mod de declanșare multiplă: Mișcarea constantă va provoca o serie de declanșări.

Placa este prevăzută cu un **jumper berg** (unele module au un jumper cu punte de lipire) care vă permite să alegeți unul dintre cele două moduri:

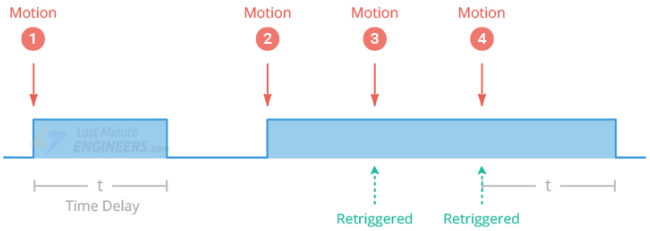


L - Selectarea acestei opțiuni va seta modul de declanșare unică. În acest mod, ieșirea devine HIGH imediat ce este detectată mișcarea și rămâne HIGH pentru o perioadă determinată de potențiometrul Time-Delay. Detectarea ulterioară este blocată până când ieșirea revine la LOW la sfârșitul timpului de întârziere. Dacă există încă mișcare, ieșirea va trece din nou la nivel HIGH. După cum puteți vedea în imaginea de mai jos, mișcarea nr. 3 este complet ignorată.

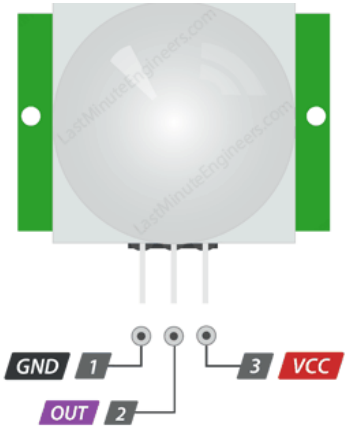


H - Selectarea acestei opțiuni va seta modul de declanșare multiplă. În acest mod, ieșirea devine HIGH imediat ce este detectată mișcarea și rămâne HIGH pentru o perioadă determinată de potențiometrul Time-Delay. Spre deosebire de modul cu

declanșare unică, detectarea ulterioară nu este blocată, astfel încât temporizarea este resetată de fiecare dată când este detectată o mișcare. Odată ce mișcarea se oprește, ieșirea revine la LOW numai după o întârziere de timp. De aici și denumirea de mod de declanșare multiplă.



Pini de conectare

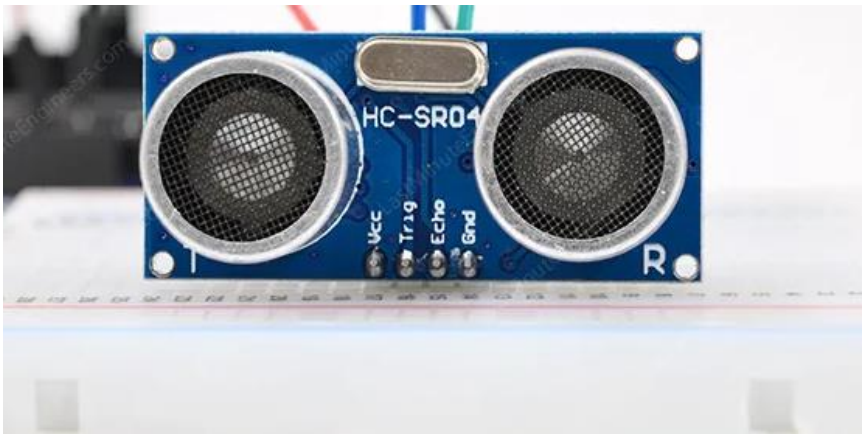


VCC este sursa de alimentare pentru senzor. Puteți conecta o tensiune de intrare oriunde între 5 și 12V la acest pin, deși 5V este utilizat în mod obișnuit.

Pinul de ieșire este ieșirea logică TTL de 3,3 V. Acesta trece pe HIGH atunci când este detectată o mișcare și trece pe LOW atunci când este inactiv (nu este detectată nicio mișcare).

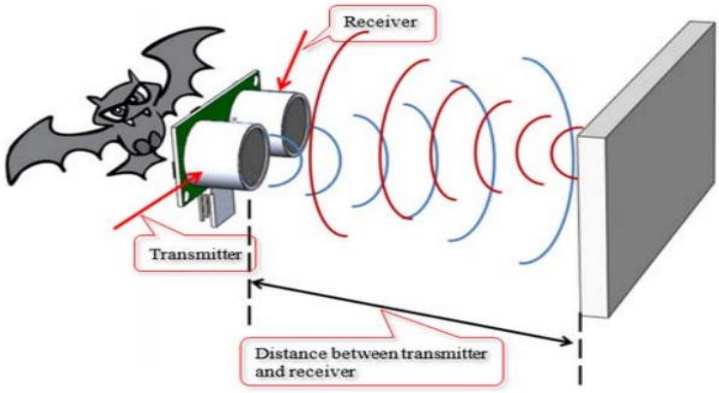
GND este pinul de masă

Ultrasonic HC-SR04



Ultrasunetele reprezintă o metodă eficientă pentru a detecta obiecte din apropierea senzorului.

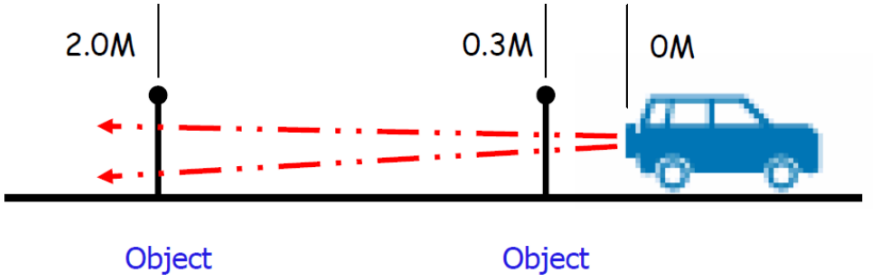
Principiul de funcționare este simplu: se emite un sunet, se așteaptă ecoul și, dacă sincronizarea este corectă, se poate determina prezența și distanța unui obiect. Acest proces, cunoscut sub numele de ecolocație, este similar cu modul în care liliecii și delfinii localizează obiecte în întuneric sau sub apă, deși aceștia folosesc frecvențe mai joase. Figura de mai jos ilustrează principiul funcționării telemetriei cu ultrasunete.



Scopul principal este detectarea distanței, care este un senzor de parcare utilizat pe scară largă pentru mașini. Senzorul este

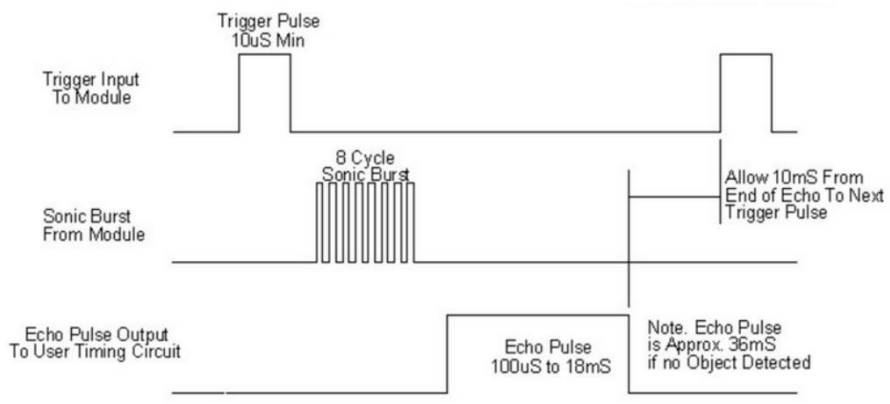
utilizat pentru a calcula distanța sau direcția unui obiect din timpul necesar pentru ca o undă sonoră să ajungă la

obiectului și a ecoului înapoi. Intervalul de detecție efectiv este de 0,3 m ~ 3,0 m.



Atunci când pinul de declanșare este activat pe HIGH timp de 10µs. În acest timp, senzorul emite o rafală de opt impulsuri ultrasonice la 40 kHz, concepută pentru a fi diferențiată de zgomotul ambiental. Aceste impulsuri se propagă prin aer, în timp ce pinul de ecou devine HIGH pentru a semnala inițierea răspunsului.

Dacă impulsurile nu se reflectă înapoi, semnalul de ecou se oprește după 38ms, indicând că nu există obstacole. Dacă impulsurile sunt reflectate, pinul de ecou trece la LOW imediat la recepție, generând un impuls a cărui lățime variază între 150µs și 25ms, în funcție de timpul de revenire a semnalului.



Lățimea impulsului recepționat este folosită pentru a calcula distanța până la obiectul reflectat, utilizând ecuația distanță = viteză x timp. De exemplu, pentru un impuls de 500µs, știm că viteza sunetului este de 0,034 cm/µs. Calculăm distanța astfel:

Distanța = 0,034 cm/µs x 500 µs = 17 cm

Pentru că impulsul include timpul dus-întors al semnalului, împărțim rezultatul la 2:

Distanța = 17 cm / 2 = 8,5 cm astfel, obiectul este la 8,5 cm de senzor.

Pini de conectare

VCC alimentează senzorul ultrasonic HC-SR04. Îl puteți conecta la ieșirea de 5V de la Arduino.

Pinul Trig (declanșator) este utilizat pentru a declanșa impulsuri sonore ultrasonice. Prin setarea acestui pin la HIGH timp de 10µs, senzorul inițiază o explozie ultrasonică.

Pinul Echo devine ridicat atunci când este transmisă explozia ultrasonică și rămâne ridicat până când senzorul primește un ecou, după care devine scăzut. Prin măsurarea timpului în care pinul Echo rămâne ridicat, se poate calcula distanța.

GND este pinul de masă.

RCWL-0516

Pentru majoritatea proiectelor care necesită să știm dacă cineva a ieșit sau a intrat în zonă, senzorul PIR este o alegere excelentă. Cu toate acestea, deoarece detectează doar mișcarea de la ființe vii, acestea vor genera mai puține alarme false.

Senzorul cu microunde precum RCWL-0516 este util. Acesta detectează orice mișcare de la orice obiect și nu se bazează pe semnăturile termice, ceea ce îl face mai fiabil în medii fierbinți, unde un senzor PIR poate să nu fie la fel de eficient.

Modulul RCWL-0516 utilizează „Doppler Radar” - un radar specializat care utilizează efectul Doppler (cunoscut și sub numele de deplasarea Doppler) pentru a detecta mișcarea și a declanșa alerte de proximitate.

Efectul Doppler

Efectul Doppler, numit după fizicianul austriac Christian Doppler care l-a propus în 1842, descrie schimbarea frecvenței observată de un observator staționar atunci

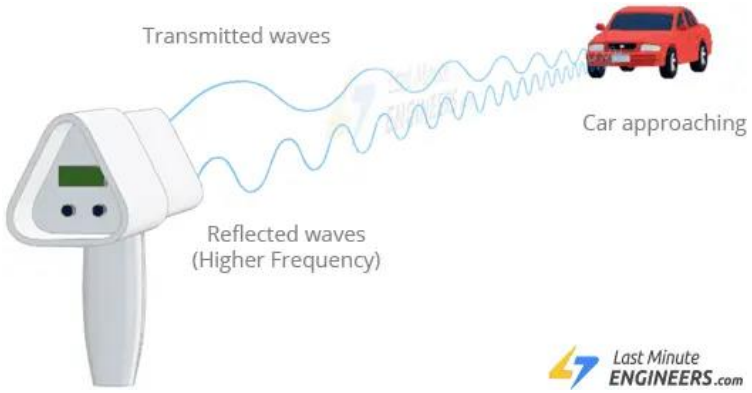
când sursa frecvenței se mișcă. Acest lucru este valabil pentru toate tipurile de unde, cum ar fi apa, lumina, radioul și sunetul.

Acest efect este des întâlnit, spre exemplu sirena care scade în înălțime atunci când trece o ambulanță. Pe măsură ce ambulanța se apropie de oameni, undele sonore ale sirenei sunt înghesuite pe o distanță mai mică, crescând frecvența lor, pe care o auzim ca o înălțime mai mare. Opusul se întâmplă atunci când ambulanța se îndepărtează de oameni, ceea ce face ca undele sonore să aibă o frecvență mai mică și o înălțime mai mică. Ca urmare, auzim o scădere vizibilă a tonului sirenei pe măsură ce trece.

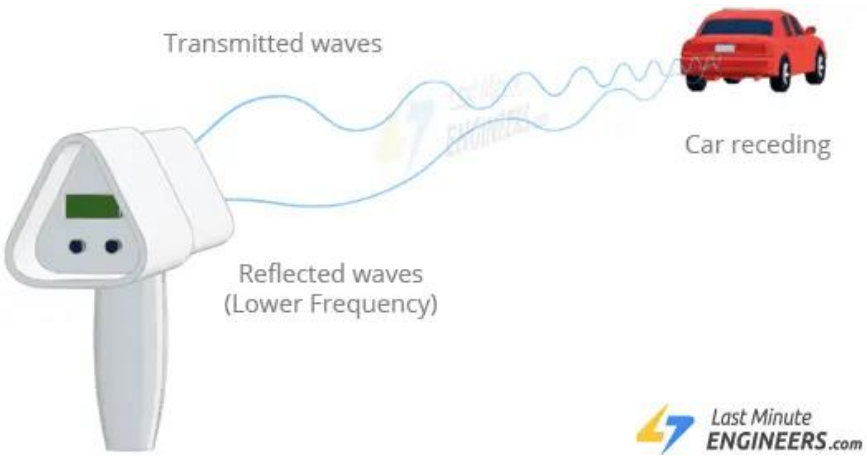
Un radar Doppler funcționează prin trimiterea unui semnal cu microunde către o țintă dorită și citirea frecvenței semnalului returnat. Analizând modul în care mișcarea țintei a modificat frecvența semnalului transmis, se poate măsura viteza țintei.

Spre exemplu ofițerii de poliție folosesc pistoale radar pentru a prinde persoane care conduc cu viteză prea mare. Aceste radare, ca și alte tipuri de radare, constau dintr-un emițător și un receptor cu microunde. Acestea trimit un semnal cu microunde și apoi îl recepționează după ce acesta ricoșează de pe țintă.

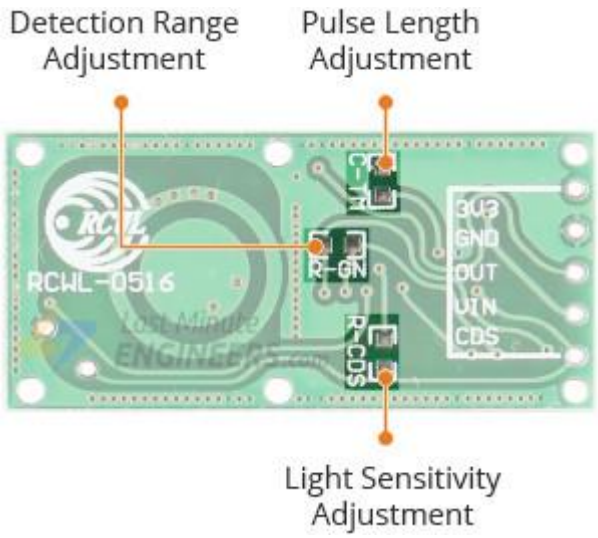
Datorită efectului Doppler, dacă obiectul se apropie sau se îndepărtează de pistol, frecvența semnalului cu microunde reflectat este diferită de cea a semnalului transmis.



Atunci când o mașină se apropie de radar, frecvența semnalului returnat este mai mare decât frecvența semnalului transmis; atunci când mașina se îndepărtează, frecvența este mai mică.



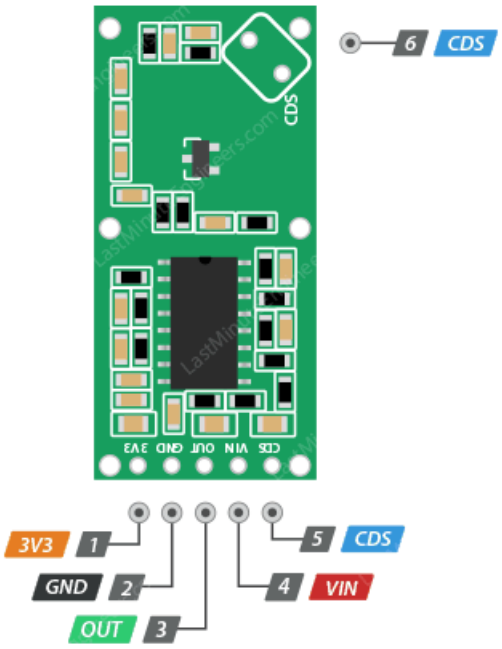
Pe baza acestei diferențe, pistolul radar calculează viteza mașinii de la care a ricoșat semnalul.



Prin adăugarea de rezistențe și condensatoare la jumperii specifici, se pot ajusta setările implicite ale senzorului:

- **C-TM (Lungimea impulsului):** Adăugarea unui condensator SMD poate prelungi impulsul de ieșire de la 2 secunde implicit la până la 250 de secunde (ex. 0,2µF pentru 50 secunde, 1µF pentru 250 secunde).
- **R-GN (Interval de detecție):** Inserarea unui rezistor reduce raza de detecție de la 7m implicit la valori mai mici (ex. 1MΩ pentru 5m, 270KΩ pentru 1,5m).
- **R-CDS (Sensibilitate la lumină):** Un rezistor de 47K-100KΩ reglează sensibilitatea la lumină; valori mai mici necesită o lumină mai puternică pentru dezactivarea senzorului.

Pini de conectare



3V3 este ieșirea de la regulatorul integrat de 3,3 V, nu intrarea sursei de alimentare. Dacă aveți nevoie de o ieșire curată de 3,3 V pentru a alimenta circuitele logice externe, o puteți utiliza. Aceasta poate furniza până la 100 mA de curent.

GND este pinul de masă.

Pinul OUT este ieșirea logică TTL de 3,3 V. Acesta trece pe HIGH timp de două secunde atunci când este detectată o mișcare și trece pe LOW atunci când este inactiv (nu este detectată nicio mișcare).

VIN este sursa de alimentare pentru senzor. Puteți conecta o tensiune de intrare oriunde între 4 și 28 V la acest pin, deși 5V este utilizat în mod obișnuit.

Pinii CDS sunt locul unde puteți atașa un rezistor dependent de lumină (LDR). Adăugarea acestei componente permite RCWL-0516 să funcționeze numai în întuneric.

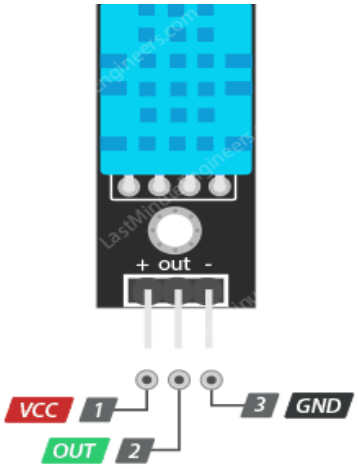
Senzor de temperatură DHT11

DHT11 poate măsura temperatura de la 0°C la 50°C cu o precizie de ±2,0°C și umiditatea de la 20 la 80% cu o precizie de 5%. Componenta de detectare a umidității are doi electrozi între care se află un substrat care reține umiditatea (de obicei o sare sau un polimer plastic conductiv).

Pe măsură ce umiditatea crește, substratul absoarbe vapori de apă, ceea ce duce la eliberarea de ioni și la o scădere a rezistenței dintre cei doi electrozi.

Această modificare a rezistenței este proporțională cu umiditatea, care poate fi măsurată pentru a estima umiditatea relativă.

Pini de conectare



Pinul + (VCC) asigură alimentarea senzorului. În ciuda faptului că tensiunea de alimentare a modulului variază de la 3,3 V la 5,5 V, se recomandă o alimentare de 5 V. Cu o sursă de alimentare de 5V, senzorul poate fi amplasat la o distanță de până la 20 de metri. Cu o tensiune de alimentare de 3,3 V, senzorul poate fi amplasat la doar 1 metru distanță; în caz contrar, căderea tensiunii de linie va cauza erori de măsurare.

Pinul de ieșire este utilizat pentru comunicarea dintre senzor și microcontroler.

GND este pinul de masă.

MicroSD Card Module

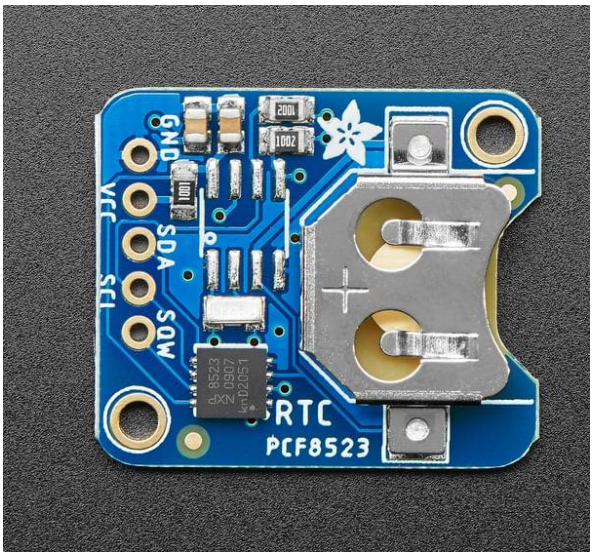
Permite citirea și scrierea de date pe un card microSD utilizând Arduino. Este ideal pentru stocarea datelor de senzor, jurnalizarea evenimentelor sau salvarea fișierelor. Folosește protocolul SPI (Serial Peripheral Interface) pentru comunicarea cu Arduino.

Modulul are patru pini de conectare:

- **MOSI (Master Out Slave In):** Transferă datele de la Arduino către cardul microSD.
- **MISO (Master In Slave Out):** Transferă datele de la cardul microSD către Arduino.
- **SCK (Serial Clock):** Semnalul de ceas pentru sincronizarea transferului de date.
- **CS (Chip Select):** Activează cardul microSD pentru comunicare.

Tensiune de Funcționare: De obicei, modulul funcționează la 3.3V, dar este compatibil cu 5V în anumite configurații.

RTC PCF8523



Acesta este un ceas de timp real (RTC) susținut de baterie care permite proiectului dvs. de microcontroler să țină evidența timpului chiar dacă este reprogramat sau dacă se pierde alimentarea. Perfect pentru înregistrarea datelor, construirea ceasului, marcarea timpului, cronometre și alarme etc. Echipat cu PCF8523 RTC - poate funcționa de la 3.3V sau 5V putere și logică

Resurse software

Am realizat un singur cod folosind mediul de dezvoltare Arduino, folosind librării speciale pentru anumiți senzori și module.

Librării folosite:

- WiFi.h : permite ESP32-ului să se conecteze la o rețea WiFi. Această bibliotecă oferă funcționalități pentru gestionarea conexiunilor WiFi, cum ar fi conectarea la o rețea, verificarea statusului conexiunii și gestionarea IP-ului.

- AsyncTCP.h : oferă suport pentru conexiuni TCP asincrone pe ESP32. Această bibliotecă este esențială pentru gestionarea comunicărilor TCP/IP fără a bloca execuția codului.

- ESPAsyncWebServer.h : permite crearea și gestionarea unui server web asincron pe ESP32. Această bibliotecă ajută la configurarea unui server web care poate răspunde la cereri HTTP fără a bloca execuția codului.

- DHT.h : oferă suport pentru senzorii de temperatură și umiditate DHT11 și DHT22. Această bibliotecă facilitează citirea valorilor de temperatură și umiditate de la acești senzori.

- Wire.h : oferă suport pentru comunicarea I2C. Este utilizată pentru a comunica cu dispozitivele care folosesc protocolul I2C (Inter-Integrated Circuit).

- RTClib.h : oferă suport pentru modulele RTC (Real Time Clock) care folosesc protocolul I2C. Permite gestionarea și citirea timpului real de la modulele RTC.

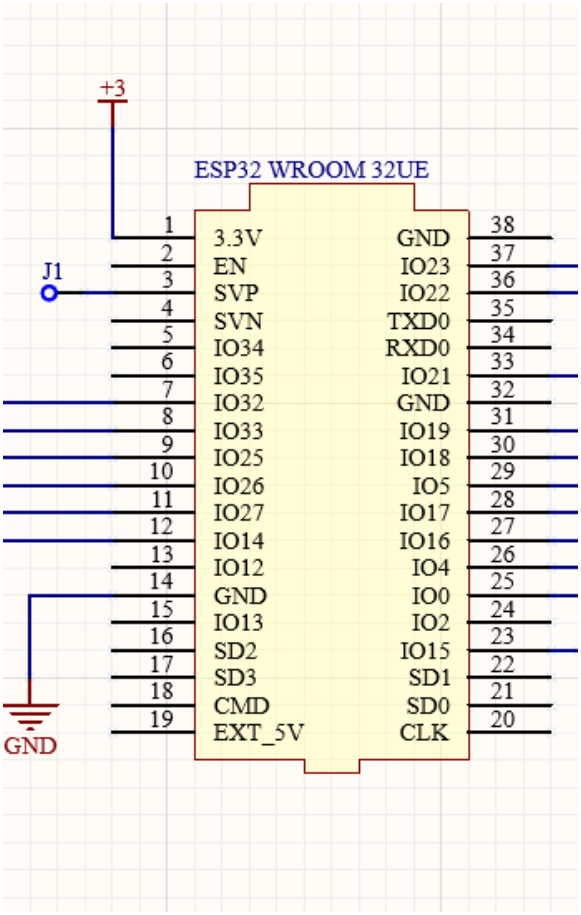
- SD.h : oferă suport pentru gestionarea cardurilor SD. Permite citirea și scrierea de date pe carduri microSD prin intermediul protocolului SPI.

Funcționalități:

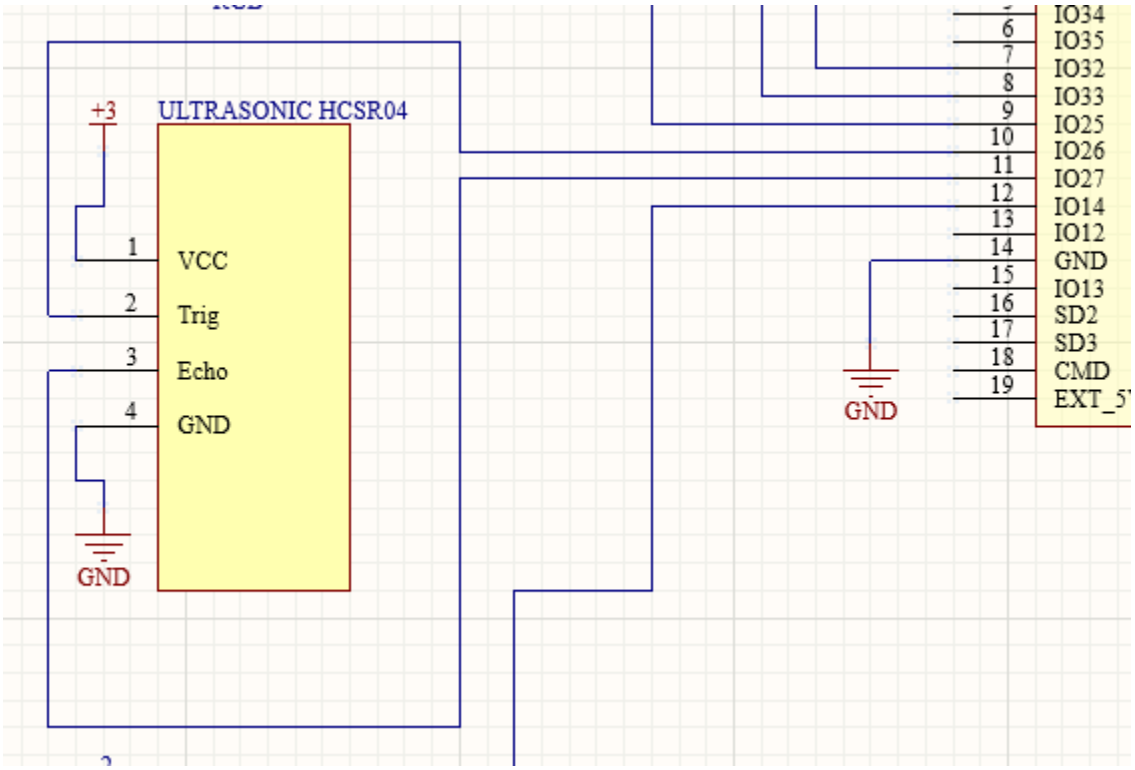
Codul ne ajută să controlăm LED-urile printr-o interfață web, folosind semnale PWM generate de pinii ESP32. De asemenea, modulul RTC ne furnizează ora curentă a României, iar prin modulul microSD stocăm datele privind alertele date de senzori atunci când detectează mișcare.

Implementare hardware

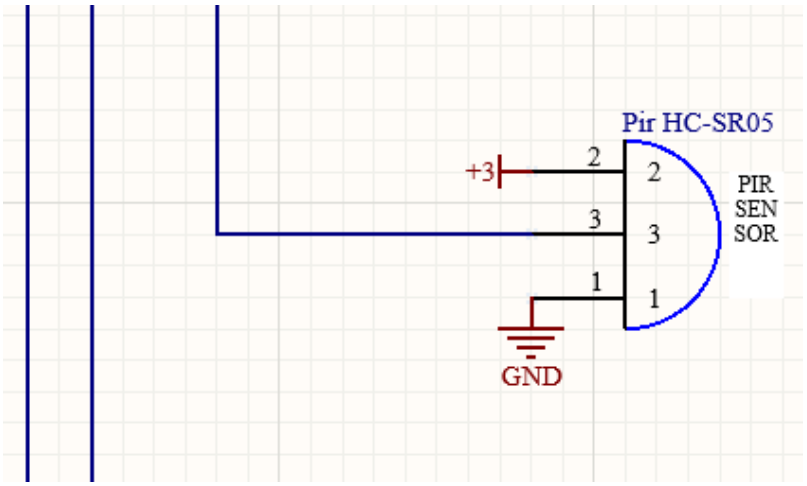
Schema electrică a microcontroller-ului ESP32 WROOM32UE



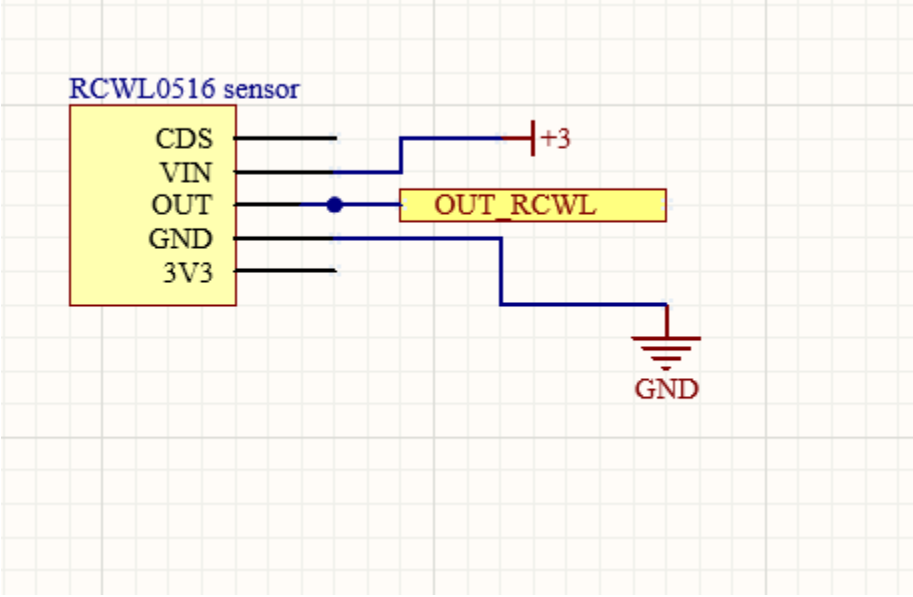
Schema electrică senzor ultrasonic



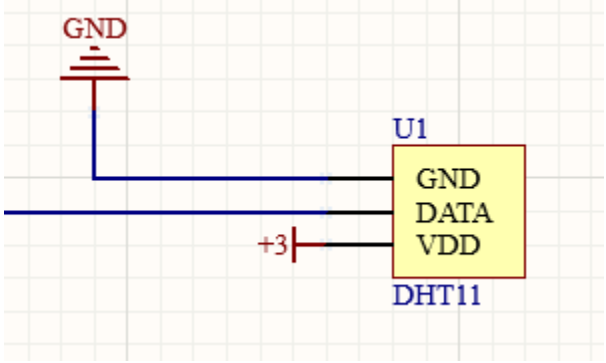
Schema electrică senzor PIR



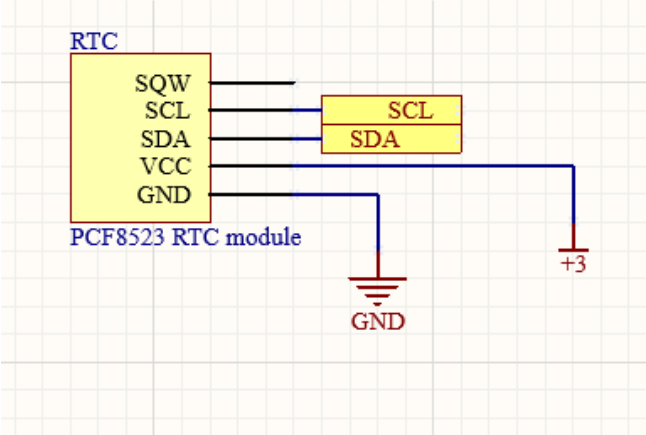
Schema electrică senzor de proximitate



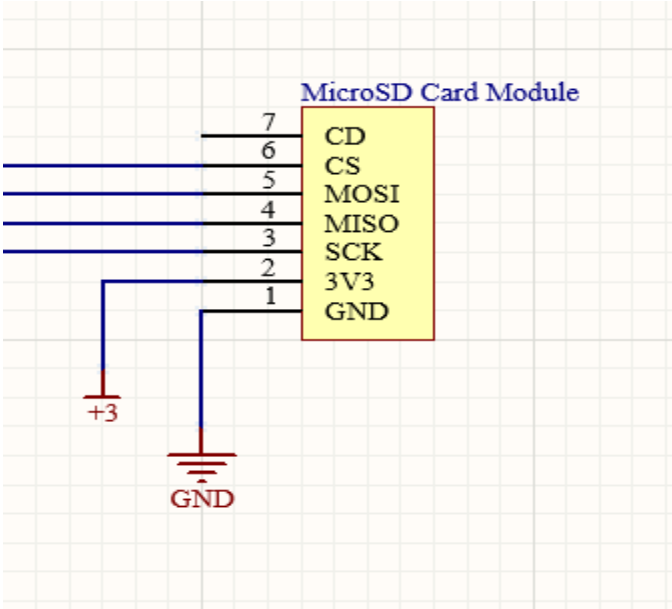
Schema electrică senzor de temperatură



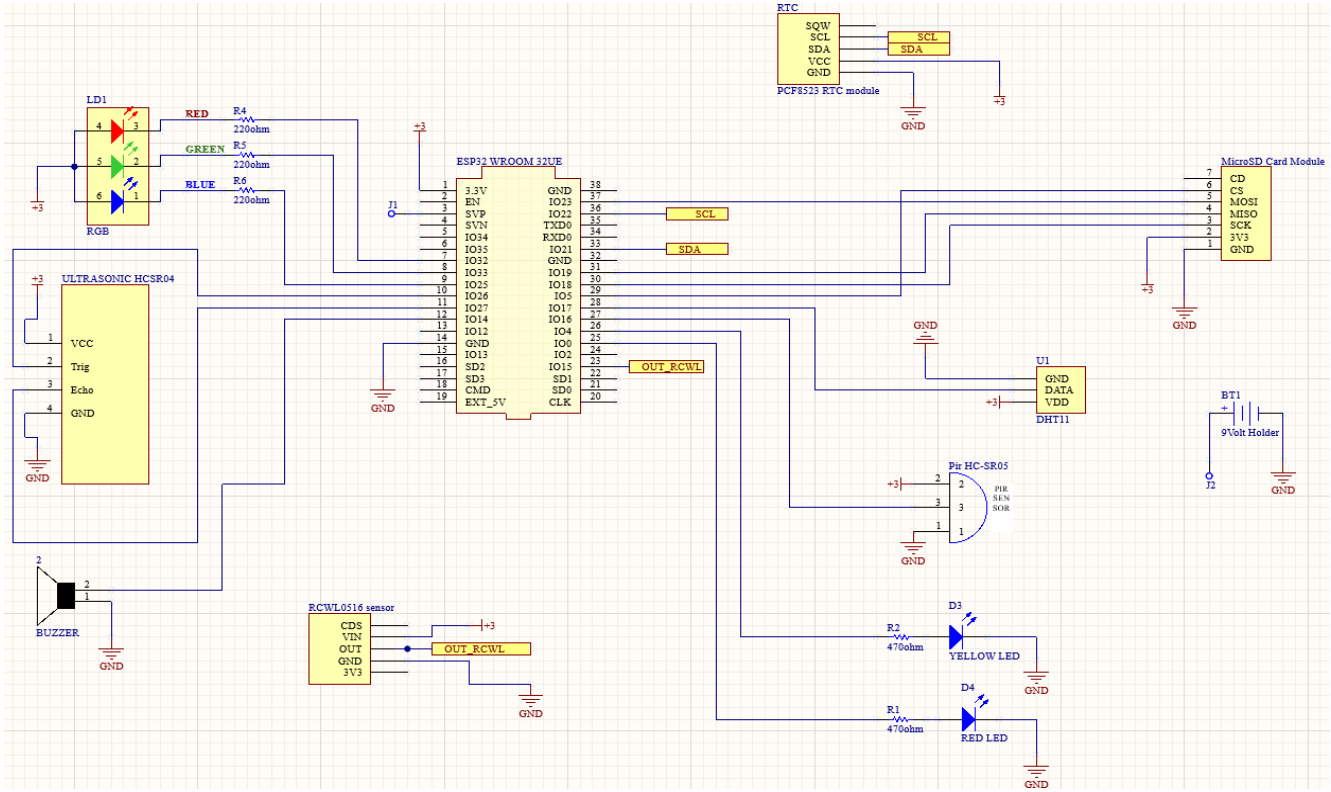
Schema electrică modul RTC



Schema electrică modul microSD



Schema electrica a întregului montaj



Principiul de funcționare al întregului ansamblu

Acest proiect utilizează microcontrolerul ESP32 ca unitatea centrală de control pentru un sistem automatizat de iluminat și securitate bazat pe senzori. ESP32 este conectat la mai mulți senzori (DHT11 pentru temperatură și umiditate, un senzor de proximitate, un senzor PIR, și un senzor ultrasonic) și controlează diverse LED-uri pentru a indica diferite stări.

ESP32 se conectează la o rețea WiFi și rulează un server web prin care utilizatorul poate monitoriza și controla sistemul. Pagina web afișează temperatura, umiditatea, și alerte de mișcare detectată în diferite camere.

Sistemul are două moduri de operare: automat și manual. În modul automat, ESP32 ajustează intensitatea LED-urilor în funcție de intervalul orar și detectarea mișcării în diverse zone ale casei.

În modul manual, utilizatorul poate ajusta intensitatea LED-urilor direct de pe pagina web folosind slider-ele.

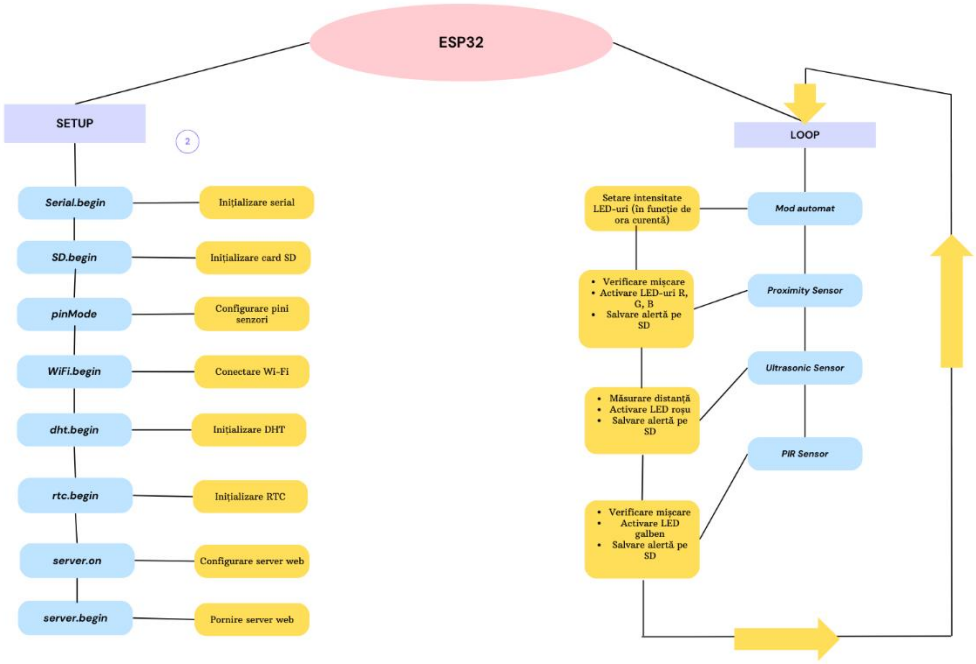
Senzorii de mișcare și proximitate detectează prezența în diverse camere (living, dormitor, baie). Când se detectează mișcare, se aprind LED-urile corespunzătoare și se salvează o alertă pe cardul SD pentru monitorizare ulterioară.

Senzorul ultrasonic măsoară distanța până la obiectele din fața sa. Dacă un obiect este detectat la o distanță mai mică de 5 cm, sistemul declanșează o alertă și aprinde un LED roșu pentru a semnala prezența.

Alertele sunt salvate pe un card SD, permițând analizarea ulterioară a evenimentelor înregistrate de sistem.

Acest sistem asigură automatizarea iluminatului în funcție de prezența în camere și oferă un nivel suplimentar de securitate prin monitorizarea mișcării și înregistrarea alertelor. De asemenea, permite intervenția utilizatorului prin control manual al luminilor prin intermediul unei interfețe web.

Implementare software



Codul implementat pe ESP32

```
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <DHT.h>
#include <Wire.h>
#include <RTClib.h>
#include <SD.h>

// Definire pini
#define DHTPIN 17
#define DHTTYPE DHT11
#define PROXIMITY_SENSOR_PIN 15
#define TRIG_PIN 26
#define ECHO_PIN 27
#define PIR_SENSOR_PIN 16
#define SD_CS_PIN 5
```

Am început prin utilizarea a mai multor librării esențiale pentru a controla și interacționa cu diverse componente hardware conectate la microcontrolerul ESP32.

Librăria **WiFi.h** permite dispozitivului să se conecteze la o rețea Wi-Fi, facilitând astfel crearea unui server web. Prin intermediul librăriilor **AsyncTCP.h și ESPAsyncWebServer.h,** este posibilă configurarea unui server web asincron, care oferă ESP32 capacitatea de a răspunde rapid și eficient la cererile HTTP, fără a bloca alte operațiuni.

Pentru monitorizarea temperaturii și umidității, am folosit librăria **DHT.h,** care facilitează interacțiunea cu senzorul DHT11. În plus, librăriile **Wire.h și RTClib.h sunt** utilizate pentru a comunica cu modulul RTC (Real-Time Clock), permițând obținerea timpului curent, necesar pentru diverse operațiuni temporizate. Librăria **SD.h** este inclusă pentru a permite interacțiunea cu un card SD, unde sunt stocate alertele generate de senzorii de mișcare.

Pinii dispozitivului sunt configurați astfel încât să corespundă fiecărei componente hardware conectate. Pinul **DHTPIN (GPIO 17)** este alocat senzorului **DHT11**, iar **DHTTYPE** definește tipul de senzor utilizat. Pinul **PROXIMITY_SENSOR_PIN (GPIO 15)** este conectat la un senzor de proximitate. Pinii **TRIG_PIN (GPIO 26) și ECHO_PIN (GPIO 27)** sunt folosiți pentru senzorul ultrasonic, care măsoară distanța. În ceea ce privește detectarea mișcării, **pinul PIR_SENSOR_PIN (GPIO 16)** este configurat pentru a comunica cu senzorul PIR. Pentru interfațarea cu cardul SD, se folosește pinul **SD_CS_PIN (GPIO 5),** care este responsabil pentru inițializarea și gestionarea cardului de memorie.

Această configurare hardware permite ESP32 să colecteze date de la senzorii conectați, să stocheze evenimentele detectate pe cardul SD și să gestioneze o interfață web interactivă pentru controlul LED-urilor și monitorizarea în timp real a stării senzorilor.

```
// Conectare WiFi
const char* ssid = "DIGI-g4WP";
const char* password = "4t2MC8wK";

// Pini LED-uri
const int output1 = 32; // R
const int output2 = 33; // G
const int output3 = 25; // B
const int output4 = 4;  // Yellow
const int output5 = 0;  // Red

// Variabile slider
String sliderValue1 = "0";
String sliderValue2 = "0";
String sliderValue3 = "0";
String sliderValue4 = "0";
String sliderValue5 = "0";

// Variabile control mode
String controlMode = "auto";

const char* PARAM_INPUT_1 = "value1";
```

```
const char* PARAM_INPUT_2 = "value2";
const char* PARAM_INPUT_3 = "value3";
const char* PARAM_INPUT_4 = "value4";
const char* PARAM_INPUT_5 = "value5";

// Creare obiecte
DHT dht(DHTPIN, DHTTYPE);
RTC_PCF8523 rtc;
AsyncWebServer server(80);
```

Conectez ESP32 la rețeaua Wi-Fi utilizând datele de autentificare definite, care includ SSID-ul și parola rețelei. Prin configurarea acestor informații, asigur conectarea dispozitivului la internet, permițând accesul la serverul web creat ulterior.

Am **alocat pinii GPIO pentru a controla cinci LED-uri de culori diferite**: pinul 32 pentru LED-ul roșu (R), pinul 33 pentru LED-ul verde (G), pinul 25 pentru LED-ul albastru (B), pinul 4 pentru LED-ul galben (Yellow), și pinul 0 pentru un al doilea LED roșu. Acești pini sunt specificați pentru a putea controla individual fiecare LED, în funcție de necesități.

Am **gestionat valorile asociate cu cinci slidere** , utilizând variabile de tip String. Fiecare variabilă, precum sliderValue1 sau sliderValue5, stochează valoarea curentă a unui slider, care va fi folosită pentru a ajusta intensitatea sau starea fiecărui LED.

Am **definit și o variabilă de tip String pentru modul de control al sistemului**, denumită **controlMode.** Aceasta este setată inițial la "auto", sugerând că LED-urile vor funcționa automat pe baza unor condiții predefinite, dar pot fi ajustate și manual prin intermediul interfeței web.

Pentru a face posibilă transmiterea valorilor sliderelor către server, **am definit parametrii de intrare, cum ar fi PARAM_INPUT_1 până la PARAM_INPUT_5.** Acești parametri vor prelua valorile sliderelor și vor trimite aceste informații către server pentru procesare.

Am **creat obiecte esențiale pentru funcționarea codului: un obiect DHT** pentru senzorul de temperatură și umiditate, **un obiect RTC_PCF8523** pentru ceasul de timp real, și un obiect **AsyncWebServer** pentru serverul web care va rula pe portul 80. **AsyncWebServer** este special conceput pentru a gestiona cereri asincron, ceea ce înseamnă că poate răspunde la multiple cereri simultan, fără a bloca execuția codului principal al ESP32.
Portul 80 este standard pentru protocolul HTTP, care este folosit pentru navigarea pe web.
Folosirea acestuia înseamnă că dispozitivul meu va putea fi accesat printr-un browser web fără a fi necesară specificarea unui port diferit. Majoritatea browserelor se conectează implicit la portul 80 când utilizează HTTP, ceea ce face accesul la server simplu și direct pentru utilizator.

```
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>ESP Web Server</title>
  <style>
    html {font-family: Arial; display: inline-block; text-align: center;}
    h2 {font-size: 2.3rem;}
    p {font-size: 1.9rem;}
    body {max-width: 400px; margin:0px auto; padding-bottom: 25px;}
    .slider { -webkit-appearance: none; margin: 14px; width: 360px;
height: 25px; background: #FFD65C;
      outline: none; -webkit-transition: .2s; transition: opacity .2s;}
    .slider::-webkit-slider-thumb {-webkit-appearance: none; appearance:
none; width: 35px; height: 35px; background: #003249; cursor: pointer;}
    .slider::-moz-range-thumb { width: 35px; height: 35px; background:
#003249; cursor: pointer; }
    .button { padding: 10px 20px; font-size: 1.2rem; cursor: pointer; }

    .alert {
    padding: 20px;
    background-color: #f44336;
    color: white;
    }

    .closebtn {
    margin-left: 15px;
    color: white;
    font-weight: bold;
    float: right;
    font-size: 22px;
    line-height: 20px;
    cursor: pointer;
```

```
    transition: 0.3s;
    }

    .closebtn:hover {
      color: black;
    }

  </style>
</head>
<body>
  <h2>ESP Web Server</h2>
  <p>Current Time: <span id="time">%TIME%</span></p>
  <p>Temperature: <span id="temperature">%TEMPERATURE%</span> &deg;C</p>
  <p>Humidity: <span id="humidity">%HUMIDITY%</span> &#37;</p>

  <div class="alert" id="alertLivingRoom">
    <span class="closebtn"
onclick="this.parentElement.style.display='none';" >&times;</span>
    <strong>Alert!</strong> Motion detected in the living room.
  </div>

  <div class="alert" id="alertBathroom">
    <span class="closebtn"
onclick="this.parentElement.style.display='none';" >&times;</span>
    <strong>Alert!</strong> Motion detected in the bathroom.
  </div>

  <div class="alert" id="alertBedroom">
    <span class="closebtn"
onclick="this.parentElement.style.display='none';" >&times;</span>
    <strong>Alert!</strong> Motion detected in the bedroom.
  </div>

  <button class="button" onclick="toggleMode()">Toggle Control Mode: <span
id="mode">%MODE%</span></button>

  <p>LED R Value: <span id="textSliderValue1">%SLIDERVALUE1%</span></p>
  <p><input type="range" onchange="updateSliderPWM1(this)" id="pwmSlider1"
min="0" max="255" value="%SLIDERVALUE1%" step="1" class="slider"></p>
  <p>LED G Value: <span id="textSliderValue2">%SLIDERVALUE2%</span></p>
  <p><input type="range" onchange="updateSliderPWM2(this)" id="pwmSlider2"
min="0" max="255" value="%SLIDERVALUE2%" step="1" class="slider"></p>
  <p>LED B Value: <span id="textSliderValue3">%SLIDERVALUE3%</span></p>
  <p><input type="range" onchange="updateSliderPWM3(this)" id="pwmSlider3"
min="0" max="255" value="%SLIDERVALUE3%" step="1" class="slider"></p>
  <p>LED Red Value: <span id="textSliderValue4">%SLIDERVALUE4%</span></p>
  <p><input type="range" onchange="updateSliderPWM4(this)" id="pwmSlider4"
min="0" max="255" value="%SLIDERVALUE4%" step="1" class="slider"></p>
  <p>LED Yellow Value: <span
id="textSliderValue5">%SLIDERVALUE5%</span></p>
  <p><input type="range" onchange="updateSliderPWM5(this)" id="pwmSlider5"
min="0" max="255" value="%SLIDERVALUE5%" step="1" class="slider"></p>
```

```
<script>
  function checkSensors() {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/getMode", true);
    xhr.onreadystatechange = function() {
      if (xhr.readyState == 4 && xhr.status == 200) {
        var controlMode = xhr.responseText;
        var motionLivingRoom = digitalRead(PROXIMITY_SENSOR_PIN); //
Simulează valoarea senzorului
        var motionBathroom = digitalRead(PIR_SENSOR_PIN);         //
Simulează valoarea senzorului
        var motionBedroom = measureDistance();                    //
Simulează valoarea senzorului

        if (controlMode == "auto") {
          document.getElementById('alertLivingRoom').style.display =
(motionLivingRoom == HIGH) ? 'block' : 'none';
          document.getElementById('alertBathroom').style.display =
(motionBathroom == HIGH) ? 'block' : 'none';
          document.getElementById('alertBedroom').style.display =
(motionBedroom <= 20) ? 'block' : 'none';
        } else {
          document.getElementById('alertLivingRoom').style.display =
'none';
          document.getElementById('alertBathroom').style.display = 'none';
          document.getElementById('alertBedroom').style.display = 'none';
        }
      }
    };
    xhr.send();
  }

  function updateSliderPWM1(element) {
    var sliderValue = document.getElementById("pwmSlider1").value;
    document.getElementById("textSliderValue1").innerHTML = sliderValue;
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/slider1?value1="+sliderValue, true);
    xhr.send();
  }

  function updateSliderPWM2(element) {
    var sliderValue = document.getElementById("pwmSlider2").value;
    document.getElementById("textSliderValue2").innerHTML = sliderValue;
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/slider2?value2="+sliderValue, true);
    xhr.send();
  }

  function updateSliderPWM3(element) {
    var sliderValue = document.getElementById("pwmSlider3").value;
    document.getElementById("textSliderValue3").innerHTML = sliderValue;
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/slider3?value3="+sliderValue, true);
```

```
    xhr.send();
  }

  function updateSliderPWM4(element) {
    var sliderValue = document.getElementById("pwmSlider4").value;
    document.getElementById("textSliderValue4").innerHTML = sliderValue;
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/slider4?value4="+sliderValue, true);
    xhr.send();
  }

  function updateSliderPWM5(element) {
    var sliderValue = document.getElementById("pwmSlider5").value;
    document.getElementById("textSliderValue5").innerHTML = sliderValue;
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/slider5?value5="+sliderValue, true);
    xhr.send();
  }

  function toggleMode() {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/toggleMode", true);
    xhr.onreadystatechange = function() {
      if (xhr.readyState == 4 && xhr.status == 200) {
        // Actualizează textul modului după toggle
        var modeElement = document.getElementById("mode");
        if (modeElement.innerHTML == "auto") {
          modeElement.innerHTML = "manual";
        } else {
          modeElement.innerHTML = "auto";
        }
      }
    };
    xhr.send();
  }

  // Refresh la fiecare 3 secunde fără a afecta starea butonului de toggle
  setInterval(function() {
    location.reload();
  }, 3000);
</script>
</body>
</html>
)rawliteral";
```

Secțiunea de cod HTML este esențială pentru interfața web a proiectului. Codul HTML este stocat în memoria flash a microcontrollerului (folosind directivele `PROGMEM`) și este trimis browserului unui utilizator atunci când acesta accesează adresa IP a ESP32 prin rețea. Pagina web afișată permite utilizatorului să interacționeze cu diferite funcții ale sistemului.

1. ***Structura HTML de bază:***

- Codul începe cu declarația `<!DOCTYPE HTML>` urmată de elementele standard HTML (`<html>`, `<head>`, `<body>`), care definesc structura paginii web.
- În `<head>`, sunt incluse meta-informații, titlul paginii ("ESP Web Server"), și stilurile CSS pentru aspectul paginii.

2. Stilizare CSS:

- Codul CSS încorporat în secțiunea `<style>` definește aspectul vizual al paginii. De exemplu:
  - `font-family: Arial;`: Setează fontul principal al paginii.
  - `.slider`: Stilizează sliderele folosite pentru a ajusta valorile LED-urilor.
  - `.alert`: Definește aspectul alertelor afișate atunci când este detectată mișcare în diferite camere.

3. Elemente interactive:

Slidere (`<input type="range">`):

Cinci slidere permit utilizatorului să ajusteze valorile de intensitate pentru diferite LED-uri. Fiecare slider este asociat unui LED specific (R, G, B, Red, Yellow).

Buton de schimbare a modului (`<button>`):

Un buton care permite utilizatorului să schimbe între modurile de control **"auto" și "manual".**

4. Informații dinamice:

- Pagina afișează date dinamice, precum:
  - **Ora curentă** (`%TIME%`).
  - **Temperatura** (`%TEMPERATURE%`) și umiditatea (`%HUMIDITY%`).
  - Valoarea curentă a fiecărui slider, afișată lângă fiecare slider.

5. Sisteme de alertă:

- Trei secțiuni de alertă **(`<div class="alert">`)** sunt dedicate pentru afișarea notificărilor atunci când este detectată mișcare în diverse camere (living room, bathroom, bedroom). Aceste alerte pot fi închise de utilizator prin apăsarea unui buton de închidere.

6. Scripturi JavaScript:

Funcționalitatea sliderelor:
Funcțiile `**updateSliderPWM1`, `updateSliderPWM2`,** etc., sunt utilizate pentru a trimite valorile selectate ale sliderelor către serverul ESP32. Aceste valori sunt apoi folosite pentru a ajusta intensitatea LED-urilor.

Controlul modului de funcționare:
Funcția `**toggleMode()**` trimite o cerere către server pentru a schimba modul de operare între "auto" și "manual". De asemenea, scriptul actualizează textul de pe buton pentru a reflecta noul mod.

Gestionarea alertelor:
Funcția `**checkSensors()**` trimite cereri către server pentru a verifica starea senzorilor și actualizează vizualizarea alertelor în funcție de rezultatele primite.

Actualizarea periodică:
Pagina este configurată să se reîncarce automat la fiecare 3 secunde pentru a reflecta orice schimbare recentă a datelor, fără a afecta starea butonului de toggle.

Această interfață web servește ca un panou de control pentru un sistem bazat pe ESP32, permițând utilizatorului să monitorizeze datele de la senzori (temperatură, umiditate, detectare de mișcare) și să controleze intensitatea LED-urilor. Totul se întâmplă într-un mod foarte intuitiv, accesibil din orice browser web conectat la aceeași rețea Wi-Fi ca și ESP32.

```
// Funcţie pentru înlocuirea variabilelor în HTML
String processor(const String& var){
  if (var == "SLIDERVALUE1"){
    return sliderValue1;
  }
  if (var == "SLIDERVALUE2"){
    return sliderValue2;
  }
```

```
  if (var == "SLIDERVALUE3"){
    return sliderValue3;
  }
  if (var == "SLIDERVALUE4"){
    return sliderValue4;
  }
  if (var == "SLIDERVALUE5"){
    return sliderValue5;
  }
  if (var == "TIME") {
    DateTime now = rtc.now();
    char timeStr[20];
    sprintf(timeStr, "%02d:%02d:%02d", now.hour(), now.minute(),
now.second());
    return String(timeStr);
  }
  if (var == "TEMPERATURE") {
    float t = dht.readTemperature();
    if (isnan(t)) {
      return "--";
    } else {
      return String(t);
    }
  }
  if (var == "HUMIDITY") {
    float h = dht.readHumidity();
    if (isnan(h)) {
      return "--";
    } else {
      return String(h);
    }
  }
  if (var == "MODE") {
    return controlMode;
  }
  return String();
}

// Funcție pentru măsurarea distanței cu senzorul ultrasonic
long measureDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH);
  long distance = (duration / 2) / 29.1; // Conversie la centimetri
  return distance;
}

// Funcție pentru salvarea alertelor în fișiere pe cardul SD
void saveAlert(const String& alertMessage) {
```

```
  File file = SD.open("/alerts.txt", FILE_APPEND);
  if (file) {
    file.println(alertMessage);
    file.close();
    Serial.println("Alert saved to SD card: " + alertMessage);
  } else {
    Serial.println("Error opening file on SD card.");
  }
}
```

processor(const String& var)

Această funcție are rolul de a înlocui variabilele specifice în HTML cu valorile lor corespunzătoare. Iată cum funcționează:

- **Parametru:** var - o variabilă de tip String care reprezintă variabila ce trebuie înlocuită.
- **Întoarce:** Valoarea corespunzătoare a variabilei, fie ca șir de caractere (String), fie ca șir gol (String()).

1. **Slider Values:**
    - SLIDERVALUE1 până la SLIDERVALUE5 - Acestea sunt înlocuite cu valorile variabilelor sliderValue1, sliderValue2, sliderValue3, sliderValue4, și sliderValue5, respectiv.
2. **Time:**
    - TIME - Înlocuiește cu timpul curent într-un format "HH:MM ". Se folosește un obiect DateTime pentru a obține ora, minutul și secunda curentă.
3. **Temperature:**
    - TEMPERATURE - Înlocuiește cu temperatura măsurată de senzorul DHT. Dacă citirea eșuează (valoare NaN), se returnează "--".
4. **Humidity:**
    - HUMIDITY - Înlocuiește cu umiditatea măsurată de senzorul DHT. Dacă citirea eșuează, se returnează "--".
5. **Mode:**
    - MODE - Înlocuiește cu valoarea variabilei controlMode.
6. **Fallback:**
    - Dacă niciuna din condițiile de mai sus nu este adevărată, funcția returnează un șir gol.

measureDistance()

Această funcție măsoară distanța folosind un senzor ultrasonic:

- **Funcționare:**
  - Se trimite un impuls de 10 microsecunde pe pinul TRIG (Trigger).
  - Senzorul ultrasonic trimite un semnal și măsoară timpul până când acest semnal revine pe pinul ECHO (Echo).
  - Durata măsurată este convertită în distanță în centimetri, folosind formula: (durata / 2) / 29.1. Acest calcul ia în considerare că timpul măsurat este pentru dus-întors.
- **Întoarce:** Distanța în centimetri.

saveAlert(const String& alertMessage)

Această funcție salvează mesajele de alertă pe un card SD:

- **Parametru:** alertMessage - mesajul de alertă ce trebuie salvat.
- **Funcționare:**
  - Deschide fișierul /alerts.txt în modul append (adăugare), pentru a adăuga mesajul la sfârșitul fișierului.
  - Dacă fișierul se deschide cu succes, scrie mesajul în fișier și îl închide.
  - Dacă fișierul nu se deschide (de exemplu, din cauza unei erori la cardul SD), afișează un mesaj de eroare pe monitorul serial.

- processor gestionează înlocuirea variabilelor în HTML cu valori specifice.
- measureDistance măsoară distanța folosind un senzor ultrasonic.
- saveAlert scrie mesaje de alertă într-un fișier pe cardul SD.

Aceste funcții sunt destinate să fie utilizate într-un context de tip Arduino, unde interacționezi cu senzori și salvezi date pe un card SD.

```
void setup(){
  // Initialize serial
  Serial.begin(115200);

  // Initialize SD card
  if (!SD.begin(SD_CS_PIN)) {
    Serial.println("SD card initialization failed!");
```

```
  return;
 }
 Serial.println("SD card initialized.");

 // Initialize senzori
 pinMode(PROXIMITY_SENSOR_PIN, INPUT);
 pinMode(TRIG_PIN, OUTPUT);
 pinMode(ECHO_PIN, INPUT);
 pinMode(PIR_SENSOR_PIN, INPUT);

 // Connect to Wi-Fi
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
   delay(1000);
   Serial.println("Connecting to WiFi..");
 }
 Serial.println(WiFi.localIP());

 // Initialize DHT sensor
 dht.begin();

 // Initialize RTC
 if (!rtc.begin()) {
   Serial.println("Couldn't find RTC");
   while (1);
 }

 // Route for root / web page
 server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
   request->send_P(200, "text/html", index_html, processor);
 });

 // Handle slider changes
 server.on("/slider1", HTTP_GET, [] (AsyncWebServerRequest *request) {
   if (request->hasParam(PARAM_INPUT_1)) {
     sliderValue1 = request->getParam(PARAM_INPUT_1)->value();
     if (controlMode == "manual") {
       analogWrite(output1, sliderValue1.toInt());
     }
   }
   request->send(200, "text/plain", "OK");
 });

 server.on("/slider2", HTTP_GET, [] (AsyncWebServerRequest *request) {
   if (request->hasParam(PARAM_INPUT_2)) {
     sliderValue2 = request->getParam(PARAM_INPUT_2)->value();
     if (controlMode == "manual") {
       analogWrite(output2, sliderValue2.toInt());
     }
   }

request->send(200, "text/plain", "OK");
 });
```

```
  server.on("/slider3", HTTP_GET, [] (AsyncWebServerRequest *request) {
    if (request->hasParam(PARAM_INPUT_3)) {
      sliderValue3 = request->getParam(PARAM_INPUT_3)->value();
      if (controlMode == "manual") {
        analogWrite(output3, sliderValue3.toInt());
      }
    }
    request->send(200, "text/plain", "OK");
  });

  server.on("/slider4", HTTP_GET, [] (AsyncWebServerRequest *request) {
    if (request->hasParam(PARAM_INPUT_4)) {
      sliderValue4 = request->getParam(PARAM_INPUT_4)->value();
      if (controlMode == "manual") {
        analogWrite(output4, sliderValue4.toInt());
      }
    }
    request->send(200, "text/plain", "OK");
  });

  server.on("/slider5", HTTP_GET, [] (AsyncWebServerRequest *request) {
    if (request->hasParam(PARAM_INPUT_5)) {
      sliderValue5 = request->getParam(PARAM_INPUT_5)->value();
      if (controlMode == "manual") {
        analogWrite(output5, sliderValue5.toInt());
      }
    }
    request->send(200, "text/plain", "OK");
  });

  // Handle mode toggle
  server.on("/toggleMode", HTTP_GET, [] (AsyncWebServerRequest *request) {
    controlMode = (controlMode == "auto") ? "manual" : "auto";
    request->send(200, "text/plain", "OK");
  });

  server.on("/getMode", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/plain", controlMode);
  });

  // Start server
  server.begin();
}
```

setup()

Această funcție configurează inițializarea componentelor sistemului:

1. **Inițializare Serială:**

- Serial.begin(115200); - Deschide comunicația serială la 115200 bps pentru monitorizare și debug.

2. **Inițializare Card SD:**
   - SD.begin(SD_CS_PIN) - Inițializează cardul SD. Dacă eșuează, afișează un mesaj de eroare.
3. **Inițializare Pini:**
   - Configurează pini pentru senzori și actuatori (intrare pentru senzori de proximitate și PIR, ieșire pentru trigger-ul senzorului ultrasonic).
4. **Conectare Wi-Fi:**
   - WiFi.begin(ssid, password); - Începe conexiunea Wi-Fi. Așteaptă până când se conectează și afișează adresa IP locală.
5. **Inițializare Senzor DHT:**
   - dht.begin(); - Inițializează senzorul de temperatură și umiditate DHT.
6. **Inițializare RTC:**
   - rtc.begin(); - Începe RTC (ceasul real). Afișează un mesaj de eroare dacă nu este găsit.
7. **Configurare Server Web:**
   - server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){...}); - Rotează cererile GET către rădăcina serverului pentru a servi pagina HTML (index_html), folosind funcția processor pentru înlocuirea variabilelor.
   - Configurarea endpoint-urilor pentru slider-e (/slider1 până la /slider5), actualizând valorile și ajustând ieșirile analogice în modul manual.
   - server.on("/toggleMode", HTTP_GET, [](AsyncWebServerRequest *request){...}); - Comută între modul „auto” și „manual”.
   - server.on("/getMode", HTTP_GET, [](AsyncWebServerRequest *request){...}); - Răspunde cu modul curent.
8. **Pornire Server Web:**
   - server.begin(); - Începe serverul web pentru a accepta cereri HTTP.

```
void loop() {
  if (controlMode == "auto") {
    DateTime now = rtc.now();
    int currentHour = now.hour();
    int intensity = 0;

    // Setare intensitate lumini în funcție de intervalul orar
    if (currentHour >= 7 && currentHour < 16) {
      intensity = 64; // Între orele 7:00 și 16:00
    } else if (currentHour >= 16 && currentHour < 21) {
      intensity = 128; // Între orele 16:00 și 21:00
    } else {
      intensity = 0; // Între orele 21:00 și 7:00
    }

    // Automatizare LED-uri bazată pe senzori și intervale de timp
    if (digitalRead(PROXIMITY_SENSOR_PIN) == HIGH) {
      analogWrite(output1, 0);   // Red LED off
      analogWrite(output2, 255); // Green LED off
      analogWrite(output3, 0);   // Blue LED on
      Serial.println("Miscare in dormitor");
      saveAlert("Motion detected in the bedroom");
    } else {
      analogWrite(output1, intensity);  // Red LED
      analogWrite(output2, intensity);  // Green LED
      analogWrite(output3, intensity);  // Blue LED
    }

    long distance = measureDistance();
    if (distance <= 5) {
      analogWrite(output5, 255);  // Red LED on
      Serial.println("Miscare in sufragerie");
      saveAlert("Motion detected in the living room");
    } else {
      analogWrite(output5, intensity);  // Control LED based on interval
    }

    if (digitalRead(PIR_SENSOR_PIN) == HIGH) {
      analogWrite(output4, 255);  // Yellow LED on
      Serial.println("Miscare in baie");
      saveAlert("Motion detected in the bathroom");
    } else {
      analogWrite(output4, intensity);  // Control LED based on interval
    }
  }
}
```

loop()

Această funcție gestionează automatizarea și controlul LED-urilor în funcție de modul curent și datele de la senzori:

1. **Mod Automat (auto):**
    - **Obține Ora Curentă:** Folosește RTC pentru a obține ora curentă.
    - **Setează Intensitatea LED-urilor:**

- Între orele 7:00 și 16:00: intensity = 64
- Între orele 16:00 și 21:00: intensity = 128
- Între orele 21:00 și 7:00: intensity = 0

Intervalul de valori de la 0 la 255 provine din faptul că PWM este reprezentat de un registru pe 8 biți în microcontroler. Un registru pe 8 biți poate stoca valori de la 0 la 255 (2^8 - 1). Aceasta înseamnă că avem 256 niveluri diferite de intensitate care pot fi setate pentru LED-uri.

2. **Automatizare LED-uri Bazată pe Senzori și Interval de Timp:**
    - **Senzor de Proximitate (PROXIMITY_SENSOR_PIN):**
        - Dacă detectează mișcare (HIGH), LED-urile sunt setate astfel:
            - LED roșu (output1): Oprit
            - LED verde (output2): Oprit
            - LED albastru (output3): Aprins
            - Afișează mesajul „Mișcare în dormitor” și salvează alerta „Motion detected in the bedroom”.
        - Dacă nu detectează mișcare (LOW), LED-urile sunt setate la intensity.
    - **Senzor Ultrasonic:**
        - Măsoară distanța.
        - Dacă distanța este mai mică sau egală cu 5 cm, LED-ul roșu (output5) este aprins, iar mesajul „Mișcare în sufragerie” este afișat și salvat.
        - Dacă distanța este mai mare de 5 cm, LED-ul roșu (output5) este setat la intensity.
    - **Senzor PIR (PIR_SENSOR_PIN):**
        - Dacă detectează mișcare (HIGH), LED-ul galben (output4) este aprins, iar mesajul „Mișcare în baie” este afișat și salvat.
        - Dacă nu detectează mișcare (LOW), LED-ul galben (output4) este setat la intensity.

Concluzii

Proiectul de control al LED-urilor bazat pe senzori și automatizare a demonstrat importanța unei abordări integrate și bine structurate în proiectarea sistemelor de control și monitorizare. Implementarea acestui sistem a implicat integrarea componentelor hardware, precum senzori de proximitate, ultrasonic și PIR, cu controlul software al LED-urilor prin tehnica PWM (Modulare a Lățimii Impulsului). Acest proiect a ilustrat cum combinația eficientă a acestor elemente poate oferi soluții funcționale pentru controlul dinamic al iluminatului în funcție de condițiile de mediu și de utilizator.

Un aspect crucial al proiectului a fost gestionarea intervalului de valori pentru PWM, care variază între 0 și 255. Această limitare a fost dictată de utilizarea unui registru pe 8 biți în microcontroler, care permite o granularitate adecvată a controlului intensității LED-urilor. Utilizarea valorilor mai mari de 255 nu ar fi fost benefică deoarece hardware-ul nu le recunoaște și, prin urmare, LED-urile nu ar fi funcționat diferit. Aceasta subliniază importanța cunoașterii limitărilor hardware în proiectele de automatizare și control.

Proiectul a abordat și probleme tehnice precum interferența senzorilor și gestionarea corectă a intensității LED-urilor. De exemplu, în modul automat, LED-urile sunt ajustate în funcție de ora din zi și de starea senzorilor, ceea ce a necesitat dezvoltarea unui algoritm robust pentru a asigura o funcționare coerentă și fiabilă. Interacțiunea dintre senzorii de proximitate, ultrasonic și PIR a fost critică pentru funcționarea corectă a sistemului, iar implementarea acestei funcționalități a necesitat o atenție sporită la detalii și o testare riguroasă.

În concluzie, finalizarea cu succes a proiectului a oferit oportunități valoroase pentru dezvoltarea competențelor în domenii precum programarea embedded, inginerie electronică și controlul sistemelor bazate pe senzori.