Smart House Layout - Ilie Viky-Andreea

Introducere

Proiectul “Smart House” implementează un sistem de control și monitorizare a casei inteligente bazat pe un microcontroler ESP32. Sistemul oferă o interfață web accesibilă dintr-un browser, permițând utilizatorilor autentificați să controleze diverse dispozitive (lumini, ușa de garaj) și să vizualizeze date de la senzori (temperatură și umiditate). Comunicația între client (browser) și server (ESP32) se realizează prin HTTP, cu datele de stare și control transmise adesea în format text simplu sau JSON.

Utilizatorul va putea interacționa cu macheta prin intermediul unei aplicații, având posibilitatea să:

- controleze luminile din fiecare cameră,

- deschidă sau închidă ușa de la garaj,

- contine un senzor de temperatura si umiditate.

Ideea proiectului a pornit de la dorința de a ilustra modul în care tehnologia poate fi integrată într-un mediu domestic pentru a spori confortul, securitatea și eficiența energetică.

Consider că un astfel de sistem este util atât pentru utilizatori, oferindu-le control de la distanță asupra casei lor, cât și pentru mine, ca dezvoltator, pentru aprofundarea cunoștințelor în domeniul automatizărilor și interfețelor inteligente.

Descriere generală

Locuința va fi formată din opt camere, dintre care una va fi destinată garajului. În fiecare cameră, cu excepția garajului, va fi instalat câte un LED care poate fi aprins sau stins direct din aplicația web, oferind un control inteligent al iluminatului.

Ușa garajului va putea fi controlată tot prin aplicație, fiind acționată de un servomotor, ceea ce permite deschiderea și închiderea acesteia de la distanță, în mod automatizat.

In aplicatia web, de asemenea, se va afla si o sectiune in care vei putea vedea umiditatea si temperatura din casa.

Hardware Design

Placa de dezvoltare ESP32 funcționează ca unitate de control principală, gestionând toate

componentele periferice ale sistemului: servomotorul utilizat pentru ridicarea și coborârea ușii garajului, buzzerul, LED-urile de stare și senzorul de detecție.

Alimentare: Dispozitivul este alimentat de un modul format din patru baterii AA (6V), care furnizează energia necesară pentru funcționarea servomotorului. Pentru a alimenta în siguranță placa ESP32, tensiunea este redusă la 3.3V cu ajutorul unui divizor de tensiune.

Pentru a controla eficient servomotorul, semnalul PWM generat de ESP32 (0–3.3V) este amplificat la 0–5V cu ajutorul unui amplificator operațional, asigurând astfel o acționare precisă a mecanismului de deschidere/închidere a garajului. Senzorul este alimentat direct din baterii și este conectat la pinul GPIO0 al plăcii ESP32. Buzzerul este conectat la pinul GPIO1, permițând generarea alertelor sonore în funcție de starea sistemului.

Această arhitectură asigură un control eficient și fiabil al sistemului de automatizare a ușii de garaj.

Piese:

Componentă	**Descriere / Observație**
ESP32	Placă Wireless cu Microcontroller
Servomotor SG92R 9g 2.5 kg.cm, 4.8 V	Motor utilizat pentru usa garajulu
Modul Senzor PIR HC-SR501	Senzor de Miscare
Tranzistor PNP 2n2907 TO-92	Pentru buzzer
Buzzer Activ	103450 – 3.7V, 2000mAh
LED-uri	lumina

Software Design

Mediu de Dezvoltare și Tehnologii

IDE: PlatformIO IDE (integrat cu Visual Studio Code) Framework Firmware: Arduino Core pentru ESP32 Limbaj Firmware: C++ Tehnologii Frontend: HTML5, CSS3, JavaScript Sistem de Fișiere pe ESP32: LittleFS (pentru stocarea fișierelor web statice)

Librării și Surse 3rd-Party

<Arduino.h>: Librăria de bază Arduino. <WiFi.h>: Pentru conectivitate Wi-Fi a ESP32. <WebServer.h>: Pentru crearea serverului HTTP pe ESP32. <LittleFS.h>: Pentru gestionarea sistemului de fișiere LittleFS. <ArduinoJson.h>: Pentru parsarea și generarea eficientă a obiectelor JSON (utilizată pentru răspunsurile la cererile privind ușa garajului și senzorul DHT). <DHT.h> & <DHT_U.h>: Pentru interfațarea cu senzorul de temperatură și umiditate DHT22. <ESP32Servo.h>: Pentru controlul servomotorului ușii de garaj.

Arhitectura Software

Sistemul este construit pe o arhitectură client-server: Server: Rulează pe ESP32. Gestionează conexiunea Wi-Fi. Rulează un server HTTP care ascultă cereri de la clienți. Implementează logica de control pentru dispozitivele fizice (LED-uri, servomotor). Citește date de la senzorul DHT22.

Gestionează autentificarea utilizatorilor. Servește fișierele statice ale interfeței web (HTML, CSS, JS) din LittleFS.

Client (Interfața Web): Rulează în browser-ul utilizatorului. Este compusă din fișiere HTML (index.html, login.html), CSS (style.css) și JavaScript (script.js). Prezintă interfața grafică pentru control și vizualizare. Trimite cereri HTTP (GET, POST) către serverul ESP32 pentru a efectua acțiuni sau a obține date. Actualizează dinamic interfața pe baza răspunsurilor de la server.

Comunicare: Protocol: HTTP/HTTPS (în acest caz, HTTP pe portul 80). Format de Date: Text simplu (pentru stările ON/OFF ale luminilor) și JSON (pentru datele senzorului DHT și starea complexă a garajului).

Componentele Software Detaliate

Fișierul de Configurare conține datele private: WIFI_SSID, WEB_USERNAME, WEB_PASSWORD.

Inițializare (setup()): Inițializează comunicarea serială pentru debugging. Configurează pinii GPIO pentru LED-uri ca ieșiri (OUTPUT) și setează starea inițială (OFF). Inițializează și atașează servomotorul la pinul SERVO_PIN, setând poziția inițială la închis (SERVO_CLOSED_POS). Inițializează senzorul DHT22. Montează sistemul de fișiere LittleFS. Afișează fișierele din rădăcină pentru diagnosticare. Se conectează la rețeaua Wi-Fi utilizând credențialele din credentials.hpp. Definește rutele HTTP și funcțiile handler asociate pentru: Autentificare (/login.html, /login, /logout). Servirea paginii principale (/) și a resurselor statice (/style.css, /script.js). Controlul individual al luminilor (ex: /toggleBedroom1, /stateBedroom1). Controlul ușii de garaj (/openGarageDoor, /closeGarageDoor, /garageDoorState). Citirea datelor de la senzorul DHT (/dhtdata). Gestionarea paginilor negăsite (handleNotFound). Pornește serverul web.

Bucla Principală (loop()): Apelează continuu server.handleClient() pentru a procesa cererile HTTP primite, doar dacă ESP32 este conectat la Wi-Fi.

Managementul Autentificării: Variabila isAuthenticated urmărește starea autentificării. handleLoginHTML(): Servește pagina login.html. handleLogin(): Procesează cererea POST de la login.html. Verifică username-ul și parola cu cele din credentials.hpp. Dacă sunt corecte, setează isAuthenticated = true și redirecționează către /. Altfel, returnează eroare 401 (Unauthorized) sau 400 (Bad Request). handleLogout(): Setează isAuthenticated = false și redirecționează către login.html. checkAuth(): Funcție helper apelată de majoritatea handler-elor pentru a verifica autentificarea înainte de a permite accesul la resurse.

Controlul Dispozitivelor: Lumini: Variabile de stare (ex: ledStateBedroom1) memorează starea fiecărei lumini. processToggleRequest(pin, stateVariable, ledName): Funcție centralizată pentru comutarea stării unui LED. Include logică de debounce (toggleDebounceDelay) pentru a preveni comutări multiple la cereri rapide. Actualizează pinul GPIO și variabila de stare. Trimite noua stare (“ON” sau “OFF”) ca răspuns. Handler-e specifice (ex: handleToggleBedroom1, handleStateBedroom1): Apelează processToggleRequest sau returnează starea curentă.

Ușa Garajului: garageServo: Obiect Servo pentru controlul motorului. isGarageDoorOpen: Variabilă booleană pentru starea ușii. handleOpenGarageDoor(), handleCloseGarageDoor(): Apelează garageServo.write() cu poziția corespunzătoare. Implementează debounce (garageDoorDebounceDelay). La deschidere, aprinde automat lumina din garaj dacă era stinsă. Returnează un obiect JSON cu doorStatus și garageLightStatus. handleGarageDoorState(): Returnează starea curentă a ușii și a luminii din garaj în format JSON.

Citirea Senzorilor (handleDHTData()): Citește temperatura și umiditatea de la senzorul DHT22. Returnează datele într-un obiect JSON. Include gestionarea erorilor (returnează null pentru valori dacă citirea eșuează).

Interfața Web (Frontend)

login.html: Un formular HTML simplu cu câmpuri pentru “username” și “password” și un buton de submit. La submit, trimite o cerere POST către /login pe serverul ESP32.

index.html (Pagina Principală): Structura: Definește layout-ul general al dashboard-ului. Buton de Logout. Legături: Include fișierul style.css pentru stilizare și script.js pentru funcționalitate.

Identificatori: Elementele interactive și de afișare au ID-uri unice (ex: toggleBedroom1, stateBedroom1, garageDoorStatusText, temperatureValue) pentru a fi manipulate de JavaScript.

style.css: Definește aspectul vizual al paginii index.html. Stilizează containerele principale, secțiunile, butoanele, textul. Implementează stilurile pentru switch-urile customizate (slider-ul ON/OFF pentru lumini). Utilizează Flexbox pentru aranjarea elementelor.

script.js (Logica Client-Side): Inițializare (DOMContentLoaded): Așteaptă ca întregul DOM să fie încărcat înainte de a executa codul. Referințe Elemente DOM: Obține referințe la elementele HTML relevante folosind document.getElementById().

Flow-ul aplicatiei

Autentificare: Utilizator accesează / → Server ESP32 verifică isAuthenticated. Dacă e false, redirect la /login.html. Utilizator introduce credențiale în login.html și submit → Browser trimite POST la /login cu datele. Server ESP32 (handleLogin) validează. Dacă OK, isAuthenticated = true, redirect la /. Browser încarcă /index.html și script.js.

Comutare Lumină: Utilizator apasă un switch în index.html. script.js trimite GET la /toggle<LightName>. Server ESP32 (handleToggle<LightName> → processToggleRequest) comută LED-ul, actualizează starea internă, aplică debounce. Server ESP32 răspunde cu noua stare (“ON” / “OFF”). script.js (updateSpecificLightUI) actualizează UI-ul.

Deschidere Ușă Garaj: Utilizator apasă “Open Door” în index.html. script.js trimite POST la /openGarageDoor. Server ESP32 (handleOpenGarageDoor) mișcă servomotorul, aprinde lumina garajului (dacă e cazul), actualizează stările interne, aplică debounce. Server ESP32 răspunde cu JSON { “doorStatus”: “OPEN”, “garageLightStatus”: “ON/OFF” }. script.js (handleGarageDoorActionResponse) actualizează UI-ul.

Afișare Date Senzor: script.js (la încărcare și setInterval) trimite GET la /dhtdata. Server ESP32 (handleDHTData) citește senzorul. Server ESP32 răspunde cu JSON { “temperature”: T, “humidity”: H }. script.js actualizează valorile în UI.

Concluzii și Puncte Cheie

Modularitate: Codul este împărțit logic în backend (ESP32) și frontend (HTML/CSS/JS). În cadrul backend-ului, funcțiile handler sunt dedicate pentru fiecare rută/acțiune.

Securitate: Implementează o autentificare simplă bazată pe username/parolă pentru a proteja accesul la funcționalitățile de control. Credențialele sunt stocate extern în credentials.hpp.

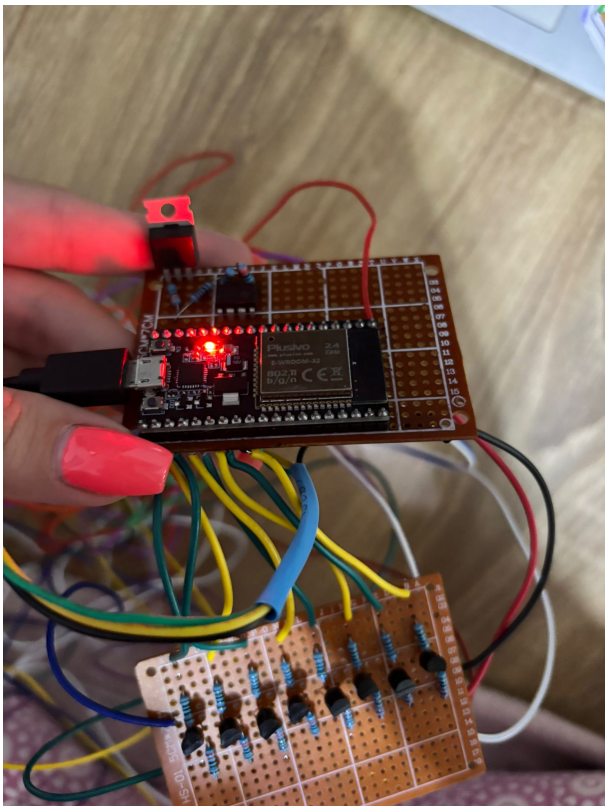
Interfață Utilizator: Oferă o interfață web intuitivă pentru control și monitorizare, cu actualizări dinamice fără reîncărcarea paginii.

Debounce: Mecanismele de debounce previn acționări multiple nedorite și protejează componentele hardware.

https://github.com/Vikyyz/ProiectPM/tree/main

Jurnal

Puteți avea și o secțiune de jurnal în care să poată urmări asistentul de proiect progresul proiectului.



Bibliografie/Resurse

https://hackaday.com/2017/01/20/cheating-at-5v-ws2812-control-to-use-a-3-3v-data-line/

https://www.amazon.com/Development-Bluetooth-Microcontroller-ESP-WROOM-32-Consumption/dp/B0BM5RNMZM

https://daumemo.com/lm317-voltage-regulator-calculator-voltage-source/