ESP Smart Home

Capitolul I. Utilitate practică

Problema

Una din cele mai raspandite probleme in domeniul dispozitivelor smart-home este faptul ca acestea sunt adeseori percepute ca fiind costisitoare si complicate pentru utilizatori. Exista conceptia ca implementarea acestora este grea si necesita cunostinte tehnice avansate, iar pretul ridicat al solutiilor existente poate descuraja noii utilizatori care isi doresc un plus de confort in cadrul locuinetei, fara a depune un efort financiar semnificativ.

ESP Smart Home vine in intampinarea acestei probleme, oferind o solutie simpla si accesibila oricui, indiferent de nivelul de cunostinte in domeniul informatic. Fiind bazat pe microcontroller-ul ESP32, acest sistem permite monitorizarea eficienta a mediului ambiant (temperatura, umiditate, luminozitate, dar si miscare) intr-o maniera simpla si intuitiva.

Descrierea Solutiei Propuse

ESP Smart Home propune un sistem simplu bazat pe un microcontroller ESP32, însoțit de următorii senzori:

- **DHT11** – pentru masurarea temperaturii și umiditatii
- **HC-SR501** – pentru detectarea miscarii
- **LDR**(Light Dependent Resistor) – pentru masurarea nivelului de luminozitate

Prin intermediului microcontroller-ului, dispozitivul, denumit nod, transmite datele catre un hub (**server de HTTP local**), folosind **WiFi-ul integrat**. Utilizatorii pot vizualiza aceste date:

- fie de pe **telefonul mobil** (client), prin intermediul unei aplicații dezvoltate
- conectandu-se la hub si vizualizand totul in **Dashboard**

Astfel, sistemul ofera o experiența simplificata și eficienta pentru monitorizarea acurata a conditiilor ambientale, fără dependenta de o platforma cloud sau costuri suplimentare.

Prin ce difera?

Problema costurilor ridicate și a complexitatii tehnice se rezolva într-un mod mai eficient prin utilizarea ESP Smart Home în comparație cu alte soluții comerciale existente din urmatoarele motive:

- **Cost redus**: Sistemul este construit cu componente accesibilie (ESP32, DHT11, HC-SR501, LDR), ceea ce îi reduce costul semnificativ fata de alte alternative comerciale
- **Ușurința în utilizare**: Interfata aplicației mobile dedicate este simpla și intuitiva, eliminand nevoia unor cunoștințe tehnice avansate
- **Independenta de platforme externe**: Sistemul funcționează local, pe reteaua utilizatorului, fără a depinde de servicii cloud sau conturi externe, reducand latenta, crescând intimitatea datelor utilizatorului și eliminand costurile abonamentelor lunare.
- **Accesibilitate tehnologica**: Prin designul sau simplu și documentatia riguroasa, proiectul poate fi replicat simplu de către entuziasti și elevi, fără a fi necesara o experiența avansata în electronica.
- **Posibilitate expansiunii** (pentru utilizatori experimentati): Deși sistemul este compact și protejat de o carcasa, exista opțiuni de extindere hardware limitate,

precum inlocuirea unor senzori ori adaugarea acestora. Acest proces nu este atât de ușor sau accesibil utilizatorilor fără experiența.

Capitolul II. Mecanica

Complexitatea

Numărul de motoare

Fiind un dispozitiv de timp smart-home, acest robot nu implica motoarele, sarcinile fiind indeplinite în mod static. Functionalitatea sa este bazata pe detectia miscarii și alte tipuri de senzori, fără a fi necesar un sistem de mișcare activa.

Gradele de libertate

Deoarece robotul nu include componente care sa necesite mișcare activa, nu exista grade de libertate asociate. Toate componentele sunt staționare și interactioneaza cu mediul prin mijloace pasive de detectie.

Eficienta în Construcție

Sursa de energie

Robotul utilizeaza un acumulator de tip 18650 Li-Po pentru alimentare. Acest tip de acumulator este o soluție sigura și eficienta pentru un dispozitiv de tip smart-home, datorită capacitatii sale de 2600mAh, care ii permite sa furnizeze energie pentru o perioada de timp indelungata de mai bine de doua săptămâni(16 zile, 20 de ore și 30 de minute). Acumulatorul este încărcat de un modul tip TP4056, un încărcător integrat eficient ce permite o încărcare controlata a bateriei.

Deoarece robotul se incadreaza în categoria dispozitivelor smart-home, acesta nu se poate baza pe sisteme precum panourile fotovoltaice sau turbinele eoliene. În schimb, acumulatorul asigura o sursa de curent constanta și pe termen lung. De asemenea, robotul este optimizat, citirile fiind făcute în cicluri. Un ciclu este alcătuit din 0.5 secunde, timp în care sunt citite valorile și transmise către Hub, urmate de 59.5 secunde de pauza. În a doua etapa, robotul intra într-un mod de economisire a energiei, în care microcontroller-ul consuma doar 0.15mA, în loc de 200mA cat a fost determinat pentru transmisia de date, reducand semnificativ consumul de energie.

Timpul de încărcare vs. timpul de funcționare

Din punct de vedere al echilibrului intre consumul de energie și timpul de funcționare, acumulatorul ofera o durabilitate suficienta pentru a susține activitatea continua a robotului pentru aproximativ 17 zile, luând în considerare ca robotul se afla în deep sleep în majoritatea timpului. Timpul de încărcare al acestuia este de aproximativ 2.6h la 1A (curent oferit de modulul TP4056), ceea ce permite un ciclu eficient de încărcare, având în vedere ca robotul va fi activ pentru doar câteva minute zilnic.

Capitolul III. Electronica

Complexitate

Robotul este pre-programat sa rezolve o problemă specifica de tip smart-home (detectia miscarii, masurarea conditiilor ambientale, transmiterea datelor către server) cu date de intrare variabile (prezenta sau absenta persoanei în fata senzorului, variatia temperaturii, gestionarea conexiunii în funcție de preferintele utilizatorului).

Arhitectura

Arhitectura este construita în jurul unui microcontroller tip ESP32, responsabil de procesarea datelor, controlarea fluxului de energie și comunicarea cu serverul prin reteaua Wi-Fi. Alegerea acestui chip a fost făcută în baza consumului sau redus de energie, dimensiunilor compacte și suportul integrat pentru protocoale de rețea precum 802.11b/g/n, TCP și HTTP.

Dispozitivul utilizeaza următorii senzori:

- **HC-SR501** (Senzor de mișcare PIR)
  - Tensiune de alimentare de 5V, dar poate fi modifcat pentru 3.3V
  - Componenta activa (are circuite de procesare)
  - Detecteaza: mișcarea într-o raza de 3 pana la 7 metri
  - Interval de detectie: unghi de 120°
  - Tip semnal de iesire: digital (HIGH atunci miscarea este detectata)
  - Utilizare: Declanseaza trezirea dispozitivului din modul deep sleep si trimiterea unei notificari catre server
  - **Justificare alegere**:
    - Consum redus de energie (50 µA in standby si 65mA in timpul detectarii miscarii)
    - Raza de detectie crescuta (specificat anterior)
    - Pret redus (5-10 RON)
    - Foarte raspandit si bine documentat (usor de implementat)
    - Output digital permite trezirea microcontroller-ului din deep sleep
- **LED**
  - Tensiune de alimentare 3.3-5V
  - Consum de curent: 20mA

- Raza de acoperire: 120°
- Preț tipic: 1-5 RON
- **Justificare alegere**:
  - Feedback vizual la fata locului (daca este activ și transmite date sau în standby)
  - Consum redus fata de alte soluții de iluminare
  - Usor de integrat

- **Buzzer**
  - Tensiune de alimentare 3.3-5V
  - Consum de curent: 30-100mA
  - Frecventa: 1 kHz la 5 kHz
  - Preț tipic: 3–5 RON
  - **Justificare alegere**:
    - Feedback acustic la fata locului (daca este activ și transmite date sau în standby ori în caz de eroare)
    - Consum redus fata de alte solutii de sonorizare
    - Usor de integrat în sisteme de microcontrollere

Capitolul IV. Software

Scrierea Codului

Programele sunt scrise urmărind cele mai bune practici în limbajul respectiv. Este urmărit standardul CamelCase pe parcursul scrierii și dezvoltarii aplicatiilor.

Arhitectura sistemului

Sistemul este alcătuit din 3 componente principale:

- Node: Dispozitivul principal responsabil cu colectarea datelor și trimiterea acestora către server.
- Hub: Acesta este un server remote de HTTP care colecteaza datele primite de la Node (ESP32) și trimite instrucțiuni și notificari către client. Hub-ul gestioneaza comunicatia cu nodurile și poate procesa datele în scopuri analitice. De asemenea, Dashboard-ul este accesibil pentru monitorizarea de către utilizator.
- Client: Aplicația care ruleaza pe smartphone-ul utilizatorului și se conecteaza la Hub pentru a obtine informații sau comenzi sau pentru a afisa notificari.

Node

Descriere generală

Proiectul **ESP Smart Home Node** este un sistem bazat pe un microcontroler ESP32 care colectează date de la mai mulți senzori, le procesează și le trimite către un server pentru a fi analizate. Sistemul include un senzor de mișcare PIR, un senzor de temperatură și umiditate DHT11, un senzor LDR pentru iluminat și un modul buzzer pentru alerte sonore. Acesta poate fi configurat printr-un server web intern, care permite setarea detaliilor de rețea și preferințele de funcționare.

Componente hardware

- **ESP32 (NodeMCU)**: Microcontroler pentru gestionarea procesării și comunicațiilor.
- **Senzor PIR** (Pin 16): Detectează mișcarea în zonă.
- **Senzor DHT11** (Pin 17): Măsoară temperatura și umiditatea.
- **Senzor LDR** (Pin 34): Măsoară intensitatea luminii ambientale.
- **LED** (Pin 23): Folosit pentru a indica starea sistemului.
- **Buzzer** (Pin 18): Alertează prin sunete.
- **WiFi**: Conexiune la rețeaua locală pentru a trimite date către un server.

Funcționalități

1. **Inițializare și Configurare WiFi**

- Funcţia `initWiFiSTA()` conectează nodul la rețeaua WiFi utilizând SSID-ul și parola configurate.
- Dacă nodul nu a fost configurat anterior, este activat modul **Access Point (AP)** pentru a permite configurarea inițială de către utilizator prin intermediul unui formular web.

2. Configurarea prin interfața web

- Nodul ESP32 poate fi configurat printr-o interfață web care permite introducerea detaliilor de rețea și preferințele de funcționare ale dispozitivului (SSID, parolă, adresă server, cameră, buzzer activat).
- Formularele HTML sunt furnizate de serverul web intern pe portul 80. După trimiterea datelor, configurațiile sunt stocate utilizând biblioteca `Preferences`, care le salvează în memoria internă non-volatilă.

3. Citirea datelor de la senzori

- **Senzorul PIR**: Detectează prezența mișcării și actualizează starea de mișcare (activ/inactiv).
- **Senzorul DHT11**: Citește temperatura și umiditatea din mediul înconjurător.
- **Senzorul LDR**: Măsoară intensitatea luminii ambientale și calculează procentajul de luminozitate.

4. Transmiterea datelor

- După ce datele au fost colectate de la senzori, acestea sunt trimise către un server configurat anterior folosind metoda HTTP POST. Datele trimise includ:
    - Starea camerei (mișcare: adevarat/fals)
    - Temperatura (°C)
    - Umiditatea (%)
    - Iluminatul (%)
- Serverul răspunde cu un pachet de confirmare.

5. Alerta prin buzzer

- Dacă este activat buzzer-ul (setat de utilizator prin interfața web), acesta emite un semnal sonor de 500 ms la fiecare 3 cicluri de blink ale LED-ului. Sunetul are rolul de a semnala funcționarea dispozitivului și poate contribui la descurajarea tentativelor de acces neautorizat.

6. Modul de economisire a energiei

- După fiecare procesare a datelor, ESP32 intră în **Deep Sleep Mode**, economisind energie pentru un interval de 60 de secunde.

Structura codului

Funcţii

`initWiFiSTA()`

- Conectează nodul la rețeaua WiFi.

`initWiFiAP()`

- Activează modul **Access Point** pentru configurarea inițială.

`initPreferences()`

- Încarcă setările salvate în memoria ESP32 folosind `Preferences`.

`handleRoot()`

- Serveşte formularul de configurare pentru utilizator.

`handleFormSubmit()`

- Procesează datele trimise prin formular şi salvează configuraţiile în memoria ESP32.

`pirHandler()`

- Gestionează citirea şi procesarea semnalului de la senzorul PIR pentru a detecta mişcarea.

`dhtHandler()`

- Gestionează citirea şi procesarea datelor de la senzorul DHT11 pentru temperatură şi umiditate.

`ldrHandler()`

- Gestionează citirea şi procesarea semnalului de la senzorul LDR pentru iluminat.

`update()`

- Trimite datele către serverul configurat prin HTTP POST.

`setup()`

- Inițializează pinurile, configurațiile şi serverul web.
- Configurează modul de economisire a energiei şi conectează la rețea.

`loop()`

- Verifică statusul senzorilor şi actualizează datele trimise la server.
- Activează starea de deep sleep pentru economisirea energiei.

Configurare inițială

1. **Conectare la AP**: Dacă nodul nu a fost configurat anterior, va funcționa în modul **Access Point**. Utilizatorul se va conecta la rețeaua WiFi creată de ESP32 şi va accesa interfața web la adresa `192.168.1.1` pentru a introduce setările de rețea şi preferințele.
2. **Configurare rețea**: Odată ce setările au fost completate, nodul se va conecta automat la rețeaua WiFi şi va salva configurațiile în memoria internă.

3. **Folosirea nodului**: După configurare, nodul va începe să colecteze date de la senzori și să le transmită către serverul configurat, care poate fi utilizat pentru a vizualiza sau procesa datele.

Hub

Descriere generală

Proiectul **ESP Smart Home Hub** este un server web construit cu ajutorul framework-ului **Flask**. Acesta colectează și afișează date provenite de la mai multe noduri de tip **ESP32** în cadrul unui sistem de smart home. Serverul permite actualizarea și vizualizarea datelor de la senzorii instalați în diferite camere, precum temperatura, umiditatea, luminozitatea și mișcarea. Aceste date sunt stocate și pot fi accesate în orice moment printr-o interfață web.

Tehnologii utilizate

- **Flask**: Framework Python pentru construirea serverului web.
- **HTML, CSS**: Pentru crearea interfeței vizuale.
- **JavaScript**: Pentru actualizarea dinamică a datelor în interfața web.

Configurare inițială

1. **Instalarea Flask**:
   Dacă nu aveți Flask instalat, o puteți face rulând comanda:

   ```
   pip install flask
   ```

2. **Pornirea serverului**:
   După instalarea Flask, serverul poate fi pornit folosind comanda:

   ```
   python hub.py
   ```

   Serverul va fi disponibil pe toate interfețele de rețea ale dispozitivului, la adresa: **http://0.0.0.0:8080/**.
   Aceasta înseamnă că aplicația poate fi accesată din orice rețea la care dispozitivul este conectat, folosind adresa IP corespunzătoare.

Funcționalități

1. Ruta `/update`

- **Metoda**: `POST`
- **Descriere**: Această rută este folosită pentru a actualiza datele de la noduri. Datele sunt trimise într-un format `form` (tip `application/x-www-form-urlencoded`), iar serverul va salva valorile trimise într-o structură de date internă.

- **Parametri**:
  - `room`: Numele camerei (ex: "Living Room").
  - `node`: Identificatorul nodului care trimite datele.
  - Parametrii adiționali: pot include `temperature`, `humidity`, `lightness`, `motion`, etc.
- **Exemplu de utilizare**:
  Un nod trimite date către server cu un formular care conține informațiile menționate mai sus, iar serverul va salva aceste date într-un dicționar intern pentru fiecare cameră.
- **Răspuns**: `OK` dacă actualizarea a fost realizată cu succes.

2. Ruta `/`

- **Metoda**: `GET`
- **Descriere**: Servește interfața de dashboard care permite utilizatorului să vizualizeze starea fiecărei camere, inclusiv temperatura, umiditatea, mișcarea și luminozitatea.
- **Răspuns**: Pagina HTML care conține interfața vizuală a dashboard-ului.

3. Ruta `/json`

- **Metoda**: `GET`
- **Descriere**: Returnează datele interne ale serverului într-un format JSON, incluzând informații despre camere și noduri.
- **Răspuns**: JSON cu datele stocate pentru fiecare cameră.

4. Ruta `/graph` `(în lucru)`

- **Metoda**: `GET`
- **Descriere**: Returneaza un set de date cu variatia citirilor în timp
- **Răspuns**: JSON cu datele stocate pentru fiecare cameră în timp.

Explicație structură date

Datele sunt stocate într-un dicționar global `data`, care conține informații despre camerele din sistem. Fiecare cameră are următoarele câmpuri:

- `temperature`: Temperatura din cameră.
- `humidity`: Umiditatea din cameră.
- `lightness`: Luminozitatea din cameră.
- `motion`: Starea mișcării (detecție de mișcare).

- `nodes`: Listă cu identificatorii nodurilor care au raportat date pentru această cameră.
- `last_update`: Timpul ultimei actualizări a datelor.

Exemplu de structură `data`:

```
{
  "rooms": {
    "Living Room": {
      "temperature": "22°C",
      "humidity": "45%",
      "lightness": "78%",
      "motion": "Not Detected",
      "nodes": ["Node1", "Node3"],
      "last_update": 1625071920
    },
    "Kitchen": {
      "temperature": "19°C",
      "humidity": "55%",
      "lightness": "40%",
      "motion": "Detected",
      "nodes": ["Node2"],
      "last_update": 1625071985
    }
  }
}
```

Dashboard HTML

Interfața web este realizată cu HTML și CSS, având o structură de tip grid pentru a organiza camerele într-un mod responsive. Fiecare cameră este reprezentată printr-o "card" care afișează informațiile actuale pentru:

- Temperatura
- Umiditatea
- Mișcarea (cu un text care indică dacă mișcarea a fost detectată sau nu)
- Luminozitatea
- Nodurile active (identificatorii nodurilor care au raportat date pentru acea cameră)

JavaScript-ul folosește `fetch` pentru a obține datele JSON de la `/json` și pentru a actualiza periodic interfața. Datele sunt actualizate automat la fiecare 2 secunde pentru a reflecta cele mai recente informații.

Exemplu de HTML generat:

```
<div class="card">
  <h3>Living Room</h3>
  <p>Temperature: 22°C</p>
  <p>Humidity: 45%</p>
  <p>Motion: <span class="status">Not Detected</span></p>
```

```
  <p>Lightness: 78%</p>
  <p>Active Nodes: Node1, Node3</p>
</div>
```

Rulare și Testare

1. **Pornirea serverului**: După configurarea și salvarea fișierului Python (`hub.py`), serverul poate fi pornit prin comanda `python hub.py`. Acesta va asculta pe portul 8080 și va fi accesibil prin orice browser la adresa `http://localhost:8080`.
2. **Testarea API-ului**: Se poate testa ruta `/update` trimițând date printr-o aplicație de tip `POST` sau printr-un client HTTP precum **Postman**.
3. **Interfața Web**: Accesați `http://localhost:8080` în browser pentru a vizualiza datele în timp real din sistem.

Client

Prezentare Aplicație

Aplicația ESP Smart Home permite utilizatorilor să interacționeze cu dispozitivele IoT din casa lor, oferind date în timp real precum temperatură, umiditate, luminozitate și detecție mișcare. Utilizatorii pot configura setări precum adresa serverului și intervalul de actualizare a datelor.

Funcționalități Cheie:

- **Obținerea Datelor în Timp Real:** Afișează datele senzorilor pentru fiecare cameră (temperatură, umiditate, luminozitate, mișcare).
- **Alerta de Detecție Mișcare:** Trimite notificări atunci când se detectează mișcare într-o cameră.
- **Configurare Adresă Server:** Permite utilizatorului să configureze și să schimbe adresa serverului.
- **Setare Interval Actualizare:** Permite utilizatorilor să definească intervalul de actualizare a datelor.
- **Vizualizare Evoluție Date:** Permite utilizatorilor sa vizualizeze evolutia datelor în timp

Componente Cheie:

1. **MainActivity:**
    - Afișează datele senzorilor (temperatură, umiditate, luminozitate și stare mișcare).
    - Trimite notificări atunci când se detectează mișcare.
    - Actualizează datele pe baza intervalului de timp configurat de utilizator.

2. **ApiClient:**
    - Obține datele senzorilor de pe serverul ESP prin cereri HTTP.
    - Permite actualizarea adresei serverului dinamic.
3. **NotificationWrapper:**
    - Creează și trimite notificări pentru evenimentele de mișcare.
    - Gestionează canalele de notificări.
4. **SettingsActivity:**
    - Oferă elemente UI pentru configurarea adresei serverului și a intervalului de actualizare.
    - Salvează și încarcă setările folosind SharedPreferences.

Metode & Funcţionalitate:

- **MainActivity:**
    - `refresh()`: Obține și afișează datele actualizate ale camerelor.
    - `startRefreshingData()`: Începe actualizarea periodică bazată pe intervalul configurat de utilizator.
    - `onCreateOptionsMenu()`: Afișează opțiunea de setări în meniu.
    - `onOptionsItemSelected()`: Deschide ecranul de setări.
- **ApiClient:**
    - `setUrl(url: String)`: Setează adresa serverului pentru obținerea datelor.
    - `update()`: Obține cele mai recente date.
    - `getJson()`: Returnează cele mai recente date ale camerelor în format JSON.
    - `setMotionCallback(callback: MotionCallback)`: Înregistrează un callback pentru evenimentele de mișcare.
- **NotificationWrapper:**
    - `createNotificationChannel()`: Inițializează canalul de notificări.
    - `sendNotification()`: Trimite notificări pentru evenimentele de mișcare.
- **SettingsActivity:**
    - `saveToPreferences(key: String, value: String)`: Salvează setările precum adresa și intervalul.

- `getFromPreferences(key: String, defaultValue: String)`: Încărcă setările salvate.

Fișiere de Configurare:

1. **SharedPreferences:**
    - Salvează setările adresei serverului și intervalului de actualizare.
    - **Chei:**
        - `address`: Adresa serverului.
        - `delay`: Intervalul în secunde între actualizările datelor.
2. **Layouts:**
    - **Layout Principal:** Conține interfața principală pentru vizualizarea datelor camerelor și accesarea setărilor.
    - **Layout Setări:** Oferă câmpuri de input pentru setările serverului și intervalul de actualizare.
    - **Layout Grafice:** Ofera capacitatea de a vizualiza evolutia datelor pe parcursul timpului (în lucru)

Permisiuni:

- **POST_NOTIFICATIONS:** Necesare pentru trimiterea notificărilor.
- **Acces la Internet:** Necesare pentru comunicarea cu dispozitivele ESP.