UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

TASTATURĂ INTELIGENTĂ CU RECUNOAȘTERE COMPORTAMENTALĂ ASISTATĂ DE AI

Absolvent

Ivașcu Ioan-Andrei

Coordonator științific

Lect.dr. Bogdan Dumitru

București, iunie 2025

Rezumat

Această lucrare de licență urmărește dezvoltarea unei tastaturi inteligente de tip macro. Proiectul include atât partea hardware, adică proiectarea unui PCB și integrarea unor componente precum taste programabile, ecran LCD, potențiometre, microfon și senzori, cât și partea software, care presupune dezvoltarea firmware-ului pentru ESP32 și a unei aplicații desktop în Python. Un aspect important al proiectului constă în integrarea unui model de inteligență artificială antrenat să identifice utilizatorul pe baza modului în care tastează, contribuind astfel la securitatea utilizatorului. Rezultatele pe care am reușit să le obțin pun în valoare funcționalitățile tastaturii, iar implementarea permite adăugarea de funcționalități noi mai târziu. Proiectul este o interfață om-mașină (HCI), oferind o alternativă care concurează cu soluțiile comerciale existente.

Abstract

This bachelor's thesis focuses on the development of an intelligent macro keyboard. This project includes both the hardware part, meaning the design of a PCB and the integration of components such as programmable keys, LCD screen, potentiometers, microphone, and sensors, and the software part, which means developing the firmware for the ESP32 and the desktop app in Python. An important aspect of the project is the integration of an artificial intelligence model trained to identify the user based on their typing pattern, thus contributing to user security. The results I managed to obtain highlight the functionalities of the keyboard, and the implementation allows new features to be added later. This project is a human-computer interface (HCI), an alternative that competes with existing solutions.

Cuprins

Capitolul 1

Introducere

1.1 Nevoia unei tastaturi inteligente personalizabile

Tastaturile reprezintă una dintre cele mai utilizate interfețe în interacțiunea om-calculator. Într-o lume în care eficiența și productivitatea sunt cele mai importante lucruri când vine vorba de tehnologie, utilizatorii caută modalități rapide de a automatiza sarcinile repetitive. Tastaturile de tip macro oferă această funcționalitate, permițând declanșarea unor acțiuni complexe prin apăsarea unui singur buton.

Cu toate acestea, soluțiile comerciale existente sunt fie costisitoare, fie limitate în ceea ce privește personalizarea și integrarea cu alte componente (ecrane, senzori, control audio etc.). Astfel, a apărut nevoia unui dispozitiv accesibil, flexibil și extensibil, care să poată fi adaptat exact la nevoile utilizatorului, de aici și ideea proiectului meu de licență.

1.2 Scopul lucrării

Lucrarea urmărește realizarea unui dispozitiv care combină o tastatură macro complet programabilă cu un ecran LCD, potențiometre pentru controlul volumului, senzori, un microfon și un cititor de carduri SD pentru extinderea memoriei. Comunicarea cu calculatorul se face prin mesaje de tip JSON, prin interfață Serial USB sau Serial Bluetooth. Un aspect inovator al proiectului constă în integrarea unui model de inteligență artificială antrenat să poată detecta dacă utilizatorul este cel autentic, pe baza modului de tastare, al modului în care modifică volumul și al distanței palmei față de dispozitiv.

1.3 Obiective

Obiectivele urmărite în această lucrare sunt:

1. Proiectarea și realizarea unui PCB funcțional, optimizat pentru componenta hardware aleasă;

2. Proiectarea și realizarea unei carcase printate 3D pentru protejarea dispozitivului;
3. Dezvoltarea firmware-ului pentru microcontrolerul ESP32, care gestionează componentele tastaturii și comunicarea cu PC-ul;
4. Implementarea unei aplicații desktop în Python, care primește, procesează și afișează datele primite de la tastatură;
5. Antrenarea și integrarea unui model AI pentru identificarea utilizatorului în funcție de modul în care utilizează tastatura;
6. Testarea funcționalităților implementate și evaluarea extensibilității sistemului pe viitor.

1.4 Structura lucrării

Lucrarea se împarte în următoarele capitole:

- **Capitolul 1 – Introducere:** prezintă contextul, scopul, obiectivele, motivația și structura lucrării;
- **Capitolul 2 – Preliminarii:** descrie noțiuni și tehnologii relevante precum HCI, tastaturi macro, ESP32, componente folosite și Inteligență Artificială;
- **Capitolul 3 – Soluția propusă:** detaliază arhitectura hardware, cât și cea software, și modul în care funcționează dispozitivul;
- **Capitolul 4 – Implementare:** descrie procesul de construcție, dezvoltarea software și testarea funcționalităților;
- **Capitolul 5 – Rezultate si evaluare:** oferă o sinteză a rezultatelor obținute și limitari.
- **Capitolul 6 – Concluzii:** prezintă concluziile lucrării și propune idei pentru extinderea funcționalităților dispozitivului.

1.5 Motivația personală

Am ales această temă datorită interesului meu pentru electronică, cât și pentru proiectare, sisteme embedded, securitate și interacțiunea om-calculator. Mi-am dorit un proiect practic, care să aducă laolaltă hardware și software într-un mod util și să aducă ceva nou pe piață. Am considerat că o tastatură inteligentă, complet personalizabilă, este o provocare potrivită pentru a-mi pune în valoare cunoștințele acumulate în facultate și în afara facultății.

1.6 Istoricul tastaturilor macro și al personalizării HCI

Primele tastaturi macro erau dispozitive simple, folosite în special în domenii precum televiziune, radio, producție video sau difuzare în direct. Ulterior, au apărut variante comerciale precum Elgato Stream Deck sau Figma Creator Micro, care oferă taste programabile și ecrane. Totuși, aceste soluții adesea nu oferă acces la firmware-ul dispozitivului sau la aplicația desktop folosită, sunt scumpe și greu de extins pe viitor. În paralel, cercetarea în domeniul HCI și al autentificării pe baza comportamentului utilizatorului a dus la dezvoltarea unor metode de recunoaștere bazate pe stilul de tastare (keystroke dynamics), un domeniu din ce în ce mai relevant în contextul securității utilizatorului.

Capitolul 2

Preliminarii

2.1 Noțiuni tehnologice relevante

Acest proiect aduce împreună mai multe concepte și tehnologii din domeniul sistemelor embedded și al interfețelor om-mașină (HCI). Tastaturile macro sunt dispozitive fizice care permit programarea unor combinații de taste sau acțiuni complexe, utile în special în domenii precum streaming, design grafic, programare sau gaming.

ESP32 este un microcontroller performant, cu conectivitate Bluetooth și Wi-Fi, capabil să gestioneze sarcini multiple și să comunice cu aplicații externe prin protocoale seriale sau de rețea. În cadrul acestui proiect, ESP32 controlează interacțiunile cu toate componentele hardware: tastatură matriceală, ecran LCD, senzori, potențiometre și microfon.

Pe partea software, aplicația desktop este dezvoltată în Python și are rolul de a primi date în timp real de la tastatură, de a le vizualiza și de a oferi control bidirecțional asupra dispozitivului.

2.2 Stadiul actual al subdomeniului HCI și tastaturi macro

Pe piață există numeroase soluții comerciale de tastaturi macro, cum ar fi Elgato Stream Deck sau Figma Creator Micro. Brandul Elgato a înregistrat o creștere de peste 40% a veniturilor din periferice de streaming în 2021, datorită vânzărilor de dispozitive Stream Deck. Modelul de 15 taste a avut peste 200 de unități vândute într-o singură lună, conform datelor de retail, iar global vânzarea a aproximativ 3,7 milioane de unități până în 2025. Figma Creator Micro s-a vândut foarte rapid la prețul de 159 USD și toate exemplarele au fost complet epuizate imediat după lansare.

În paralel, cercetarea în domeniul HCI a evoluat în direcția personalizării experienței utilizatorului și a autentificării bazate pe comportament. Tehnica „keystroke dynamics”

permite identificarea utilizatorului în funcție de modul în care tastează, oferind o metodă pasivă de autentificare, greu de falsificat.

Prin combinarea acestor direcții, dispozitivul propune o platformă completă care îmbină funcționalitatea unei tastaturi macro cu algoritmi de recunoaștere a utilizatorului, adaptabilitate și modularitate.

2.3 Obiectivele lucrării în contextul actual

Într-un ecosistem tehnologic în continuă evoluție, în care securitatea personală și eficiența în utilizarea dispozitivelor sunt esențiale, această lucrare își propune:

1. să construiască un dispozitiv fizic complet, accesibil și ușor de dezvoltat;
2. să integreze funcționalități clasice și moderne (control audio, afișaj LCD, senzori de proximitate);
3. să implementeze un model AI pentru identificarea utilizatorului;
4. să demonstreze fezabilitatea unei soluții open-source, personalizabile, cu aplicație directă în HCI;
5. să compare soluția propusă cu alternativele comerciale și să evidențieze avantajele sale.

Capitolul 3

Soluția propusă

3.1 Concept și arhitectură generală

Acest proiect urmărește realizarea unei tastaturi macro inteligente, cu funcționalități extinse și un grad ridicat de personalizare. Ideea de bază pornește de la observația că majoritatea soluțiilor comerciale sunt fie închise, fie greu de extins, iar accesul la funcționalități avansate este limitat de cost sau de lipsa documentației.

Soluția propusă integrează componente hardware variate — butoane, potențiometre, ecran LCD, senzori de proximitate și microfon — într-un singur dispozitiv controlat de un microcontroller ESP32. Acesta comunică cu un calculator prin Bluetooth sau USB, trimițând și primind date în format JSON.

Partea software este împărțită în două mari componente:

- **Firmware-ul ESP32**, scris în C++, care gestionează evenimentele hardware (apăsări de taste, rotiri de potențiometre, date audio sau de proximitate) și le transmite către calculator.
- **Aplicația desktop**, scrisă în Python cu interfață grafică (PyQt6), care primește datele în timp real și oferă utilizatorului posibilitatea de a vizualiza și configura dispozitivul.
- **Componenta AI**, implementată pe PC, care analizează comportamentul utilizatorului și încearcă să identifice persoana care tastează folosind un model Random Forest antrenat local.

Proiectul este gândit modular, astfel încât fiecare funcționalitate (de exemplu, tastatura, controlul audio sau interfața grafică) să poată fi dezvoltată, testată și îmbunătățită independent.

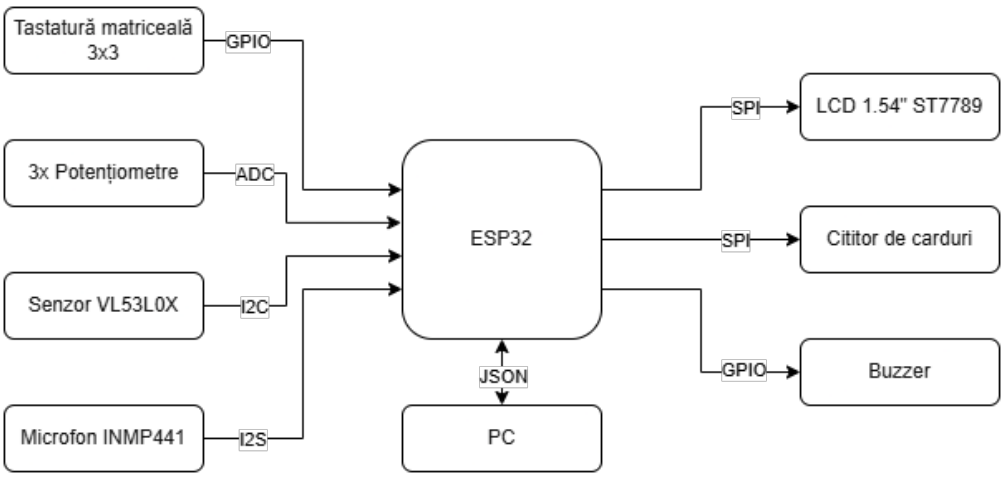

Figura 3.1: Arhitectura logică a sistemului.

3.2 Descompunerea problemei

Acest proiect propune integrarea unui număr mare de componente hardware și software, deschis și extensibil. Pentru a ușura dezvoltarea și testarea fiecărei funcționalități în parte, problema a fost descompusă în următoarele componente:

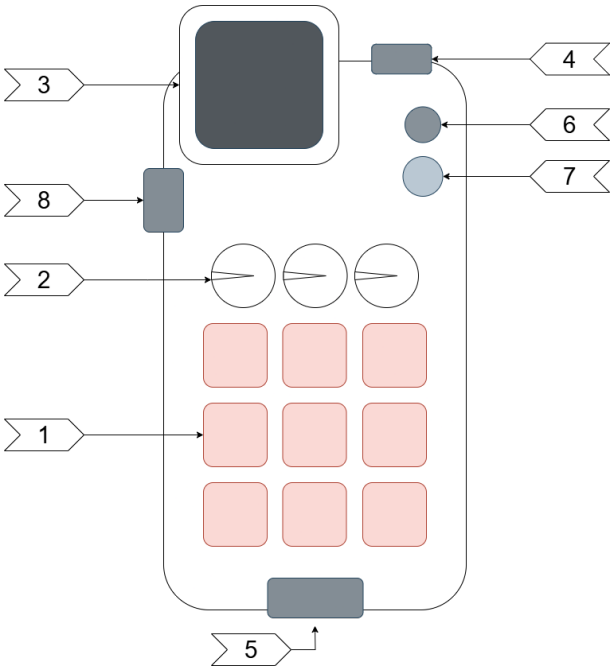

Figura 3.2: Descompunerea problemei

1. **Scanarea tastaturii matriceale** – citirea unei tastaturi 3x3 și detecția corectă a apăsărilor de taste; implementată în firmware-ul de pe ESP32, cu mesaj JSON trimis către PC la fiecare apăsare.

2. **Citirea potențiometrelor** – preluarea valorilor analogice de la cele 3 potențiometre de fiecare dată când acestea se modifică, trimiterea lor către aplicația desktop în format JSON și sincronizarea nivelurilor cu mixerul audio al sistemului de operare.

3. **Controlul afișajului LCD** – afișarea de imagini statice pe un ecran ST7789 (240x240px), controlat prin SPI. Imaginile sunt preluate de pe un card microSD, citit printr-un modul de card reader conectat la aceeași magistrală SPI.

4. **Cititorul de carduri microSD** – permite citirea fișierelor de imagine de pe card, care ulterior sunt procesate și afișate pe LCD.

5. **Prelucrarea datelor de la senzorul VL53L0X** – măsurarea distanței între dispozitiv și utilizator. Valorile sunt transmise către PC prin mesaje JSON și pot fi folosite ca input în modelul AI de identificare a utilizatorului.

6. **Buzzer-ul activ** – componentă audio de ieșire utilizată pentru feedback acustic la acțiuni precum apăsări de taste, erori sau confirmări. Poate fi activat la comanda aplicației desktop printr-un mesaj JSON către dispozitiv.

7. **Captarea sunetului prin microfon** – microfonul INMP441, conectat prin interfață I2S, permite captarea sunetului. Deși nu este utilizat încă, datele pot fi transmise către PC pentru aplicații de recunoaștere vocală sau detecție de zgomot.

8. **Rolul microcontrolerului ESP32** – placa ESP32 are rol de unitate principală de control, coordonând toate componentele hardware (taste, senzori, afișaj, volum etc.). Firmware-ul este scris în C++, asigurând o funcționare completă și eficientă a întregului sistem.

9. **Comunicarea între ESP32 și PC** – conexiune stabilă prin USB serial sau Bluetooth serial, bazată pe trimiterea și recepționarea de mesaje JSON structurate.

10. **Interfața grafică a aplicației desktop** – dezvoltarea unei interfețe cu PyQt6 care permite: monitorizarea în timp real a componentelor hardware, trimiterea de comenzi, configurarea keybind-urilor, a mixerului de volum și alegerea imaginilor de pe cardul SD pe care să le afișeze LCD-ul.

11. **Modelul AI pentru identificarea utilizatorului** – antrenarea și utilizarea unui model local de învățare automată, capabil să recunoască utilizatorul în funcție de stilul de tastare, variațiile de volum și distanța față de dispozitiv.

ESP32 (Firmware)	PC (Aplicație Desktop)
Scanare tastatură 3x3	Vizualizare taste apăsate în GUI
Citire potențiometre	Afișare și sincronizare volume
Control LCD	Trimitere fișiere imagine către LCD
Cititor card SD	Preprocesare și selecție fișiere imagine
Citire VL53L0X	Utilizare date pentru AI
Activare buzzer	Semnale acustice
Transmitere JSON prin Bluetooth/USB	Configurare, vizualizare, loguri

Tabela 3.1: Împărțirea responsabilităților între ESP32 și aplicația desktop

3.3 Avantaje față de alternative comerciale

Dispozitivele comerciale precum Elgato Stream Deck Mini sau Figma Creator Micro sunt populare în rândul creatorilor de conținut și programatorilor. Totuși, acestea vin cu o serie de limitări care pot fi evitate de acest dispozitiv. Acesta a fost creat pentru a răspunde acestor neajunsuri și pentru a oferi o soluție deschisă și personalizabilă.

Principalele avantaje ale soluției propuse sunt:

1. **Cost redus** – toate componentele hardware sunt ușor accesibile și pot fi înlocuite cu alternative mai scumpe sau mai ieftine, foarte ușor;
2. **Extensibilitate** – dispozitivul suportă adăugarea de noi funcționalități sau senzori prin SPI sau I2C;
3. **Personalizare completă** – atât firmware-ul, cât și aplicația desktop sunt open-source și ușor configurabile;
4. **Integrarea AI** – include un model local de identificare a utilizatorului, unic în această gamă de dispozitive;
5. **Stocare locală** – imaginile pot fi stocate pe un card microSD de până la 8GB;
6. **Control hardware dedicat pentru volum** – potențiometrele permit ajustarea volumelor din sistemul de operare, dar pot fi programate și pentru alte funcționalități, la alegerea utilizatorului.

Caracteristici	acest dispozitiv	Elgato Stream Deck	Figma Creator Micro
Open-source	Da	Nu	Nu
Număr de taste	9	4	12
Afișaj LCD integrat	Da	Da (pe fiecare tastă)	Nu
Potențiometre	Da (3 canale)	Nu	Da (1 canal)
Cititor microSD	Da	Nu	Nu
Recunoaștere cu AI	Da	Nu	Nu
Conectivitate	USB / Bluetooth	USB	USB
Preț estimativ	~35 EUR	80 EUR	159 EUR
Firmware modificabil	Da	Nu	Nu

Tabela 3.2: Comparație între *Acest dispozitiv* și alternative comerciale

3.4 Limitări și alegeri de design

În ciuda avantajelor semnificative oferite de acest dispozitiv, au existat și anumite compromisuri necesare pentru a menține proiectul accesibil, funcțional și ușor de realizat într-un cadru academic. Aceste alegeri au fost făcute conștient, cu scopul de a păstra un echilibru între complexitate, cost și timp de dezvoltare.

1. **Alimentare USB limitată** – ESP32 și componentele atașate sunt alimentate exclusiv prin USB, ceea ce limitează curentul disponibil pentru componente precum buzzer-ul sau ecranul LCD. Alimentarea externă a fost evitată pentru a menține designul compact și simplu.

2. **Număr redus de taste** – tastatura include doar 9 butoane fizice, suficiente pentru demonstrarea conceptului, dar sub necesitățile unor utilizatori avansați. Designul este însă extensibil.

3. **Lipsa unei interfețe de configurare avansate pe dispozitiv** – toate setările se realizează prin aplicația desktop, neexistând un meniu direct pe ecranul LCD. Această alegere a fost motivată de limitările de memorie și de complexitatea interfeței grafice embedded.

4. **Transmiterea audio neimplementată complet** – deși microfonul este funcțional și pregătit hardware, transmiterea fluxului audio prin Bluetooth către PC nu a fost finalizată, dar este prevăzută pentru o versiune viitoare.

5. **Lipsa stocării permanente a configurației** – setările se pierd la repornirea dispozitivului, neexistând momentan un mecanism de salvare EEPROM sau în sistemul de fișiere. A fost preferată o soluție temporară pentru simplitate.

6. **Calibrare simplificată pentru potențiometre și senzor** – nu a fost implementat un mecanism automat de calibrare dinamică, ci s-a folosit o scalare liniară empirică.

7. **Nu există suport multi-user nativ** – deși modelul AI poate identifica utilizatori diferiți, aplicația nu include încă un sistem complet de profiluri pentru setări personalizate per utilizator.

Aceste limitări nu afectează funcționarea de bază a dispozitivului, ci țin de rafinări și îmbunătățiri care pot fi abordate în iterații ulterioare. Proiectul rămâne complet funcțional și extensibil, fiind o bază solidă pentru dezvoltări viitoare.

Capitolul 4

Implementare

4.1 Implementare hardware

Partea hardware a proiectului a fost construită în jurul microcontrollerului ESP-WROOM-32, cu 38 de pini și cip CP2102, ales pentru performanțele sale superioare în comparație cu Arduino Leonardo folosit la iterații precedente.

Figura 4.1: ESP-WROOM-32, 38 pini cu cip CP2102

Tastatura matriceală 3x3 este formată din switch-uri mecanice MX cu diode pentru protecție, conectate într-o configurație 3x3 (3 rânduri, 3 coloane). Diodele sunt esențiale pentru a preveni scurgerea de curent într-o direcție nedorită (ghosting).

Figura 4.2: Switch-urile AKKO v3 pro fairy(silent)

Figura 4.3: Diodele 1N4148

Potențiometrele (x3) sunt conectate pe intrări analogice ale ESP32 și sunt folosite pentru controlul a trei volume la alegere din mixerul audio al sistemului de operare. Valorile sunt citite la fiecare schimbare semnificativă într-o marjă prestabilită și trimise către aplicația desktop.

Figura 4.4: Potentiometrele Taiwan ALPHA 09

LCD-ul de 1.54 inch cu cip ST7789 funcționează pe interfață SPI și este alimentat la 3.3V. Pinii de date și control (CS, DC, RST, SCL, SDA) au fost aleși pentru a utiliza magistrala VSPI.

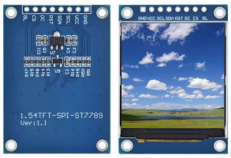

Figura 4.5: Ecran cu cip ST7789 de 1.54 inch

Cititorul de carduri microSD este conectat tot pe VSPI, partajând aceeași magistrală cu ecranul. Controlul accesului este gestionat prin pini de selecție (CS), unul pentru ecran și unul pentru cititor, pentru a alege dacă comunicarea se face cu ecranul sau cu cititorul.

Figura 4.6: Cititor de carduri SD

Senzorul VL53L0X pentru măsurarea distanței dintre dispozitiv și palma utilizatorului este conectat prin magistrala I2C (SDA, SCL) și alimentat la 3.3V. Datele de la senzor sunt preluate periodic și trimise spre PC.

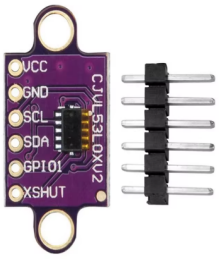

Figura 4.7: Senzorul care masoara distanta VL53L0X

Buzzer-ul activ este conectat pe un pin digital și acționat de software prin mesaje JSON către dispozitiv pentru a emite semnale sonore la diferite acțiuni alese de utilizator (apăsări de taste, feedback etc.).

Figura 4.8: 12085 42R buzzer pasiv

Microfonul INMP441 este un microfon digital I2S conectat la pinii WS, SCK, SD ai ESP32.

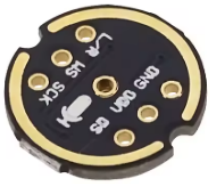

Figura 4.9: Microfon INMP441

Pentru o integrare cât mai ușoară și profesională, a fost proiectat un PCB personalizat în KiCad, care să includă toate traseele necesare pentru conectarea componentelor proiectului. Versiunile prototipului au fost testate pe breadboard, iar ulterior pe plăci de prototipare, înainte de realizarea designului final, acesta fiind a șasea versiune. (Vezi figura 7.1 si figura 7.2 din Anexe)

Pentru a oferi un aspect frumos și o fixare stabilă a PCB-ului, a fost proiectată și imprimată 3D o carcasă personalizată pentru dispozitiv. Designul a fost realizat astfel încât să fixeze în șuruburi PCB-ul și să aibă tăieturi pentru switch-urile mecanice, potențiometrele, ecranul LCD și celelalte module, respectând dimensiuni cât mai compacte și ergonomia unei tastaturi. Carcasa este formată din mai multe piese care se asamblează ușor folosind 4 șuruburi M4x20 și permit accesul la portul USB Type-C sau la cardul SD. (Vezi figura 7.3 din Anexe)

4.2 Implementare software

Arhitectura software a fost împărțită în trei componente: firmware-ul care rulează pe ESP32, aplicația desktop în Python și modelul de inteligență artificială care rulează pe PC.

4.2.1 Firmware ESP32

Firmware-ul rulat pe ESP32 este scris în C++ și încărcat pe dispozitiv cu ajutorul Arduino IDE și gestionează toate componentele hardware: tastatură, potențiometre, LCD, senzor de proximitate, cititor card SD și buzzer. Acesta comunică cu aplicația desktop prin Bluetooth Serial, trimițând și primind mesaje în format JSON.

1. **Inițializare componente**

   Toate modulele sunt inițializate în cadrul funcției `setup()`. Aceasta configurează pinii digitali, inițializează perifericele SPI și I2C, pornește conexiunea Bluetooth și verifică starea cardului microSD și a senzorului de proximitate.

   - **Pseudocod:**

     ```
     - Pornește Bluetooth Serial
     - Configurează pinii buzzer, tastatură, SPI, I2C
     - Inițializează LCD și afișează "LCD OK"
     - Verifică prezența cardului SD
     - Inițializează senzorul VL53L0X
     ```

     Vezi implementarea completă în Anexa 7.4.

2. **Scanarea tastaturii 3x3**

   Tastatura este scanată prin coborârea pe LOW a fiecărui rând și citirea fiecărei coloane. Se detectează apăsările noi și se trimite un eveniment JSON pentru fiecare tastă.

   - **Pseudocod:**

   ```
   Pentru fiecare rând:
     setează pinul de rând pe LOW
     Pentru fiecare coloană:
       dacă pinul de coloană e LOW și tasta nu era apăsată:
         trimite JSON {"type":"key", "key":"pressed", "value":<caracter>}
         trimite și distanța VL53L0X
         marchează tasta ca apăsată
       altfel: marchează ca neapăsată
     setează pinul de rând pe HIGH
   ```

   Vezi codul în Anexa 7.5.

3. **Citirea potențiometrelor**

   Potențiometrele sunt conectate pe intrări ADC. La fiecare ciclu din `loop()`, se citesc valorile și, dacă diferența față de valoarea anterioară depășește un prag (50), se trimite un mesaj JSON.

   - **Pseudocod:**

   ```
   Pentru fiecare potențiometru:
     citește valoarea curentă
     dacă diferența > prag:
       trimite JSON {"type":"volume", "key":<index>, "value":<valoare>}
       trimite și distanța VL53L0X
       actualizează valoarea precedentă
   ```

   Vezi codul în Anexa 7.6.

4. **Senzorul VL53L0X**

   Senzorul de proximitate este utilizat pentru a măsura distanța dintre dispozitiv și utilizator. Valorile sunt transmise la fiecare apăsare de tastă sau modificare de volum.

   - **Pseudocod:**

```
VL53L0X.rangingTest()
trimite JSON {"type":"proxy", "key":"distance", "value":<milimetri>}
```

Vezi codul relevant în Anexa 7.7.

5. **Controlul afișajului LCD**

   LCD-ul cu cip ST7789 este utilizat pentru afișarea de imagini în format `.raw`, transmise de aplicația desktop. Imaginea este desenată linie cu linie pe ecran cu `drawRGBBitmap()`.

   - **Pseudocod:**

   ```
   Când se primește comanda "display_raw":
     deschide fișierul de pe SD
     pentru fiecare linie de 240 pixeli:
       citește 480 bytes
       tft.drawRGBBitmap(0, linie, buffer, 240, 1)
     închide fișierul
     trimite {"status":"draw_raw_ok"}
   ```

   Vezi codul complet în Anexa 7.8.

6. **Comenzi JSON primite de la aplicație**

   Firmware-ul interpretează comenzile primite de la aplicația desktop. Se folosește biblioteca `ArduinoJson` pentru parsare. Comenzile implementate sunt:

   - **Pseudocod:**
     - `"buzz"` – pornește buzzer-ul cu o frecventa si o durată specificată;
     - `"list_sd"` – trimite lista de fișiere de pe cardul SD;
     - `"display_raw"` – afișează o imagine de pe cardul SD `.raw` pe LCD.

   ```
   Dacă JSON primit este valid:
     dacă cmd == "buzz":
       activează buzzer la frecvență și durată date
     dacă cmd == "list_sd":
       parcurge cardul și trimite lista fișierelor
     dacă cmd == "display_raw":
       citește fișierul și îl afișează pe LCD
   ```

   Vezi handler-ul JSON în Anexa 7.9.

4.2.2 Aplicația desktop

Aplicația desktop este implementată în Python, utilizând biblioteca PyQt6 pentru interfață grafică, 'pycaw' pentru controlul audio și 'pynput' pentru simularea de macro-uri. Aceasta comunică cu ESP32 prin Bluetooth Serial sau USB Serial, trimițând și primind mesaje JSON.

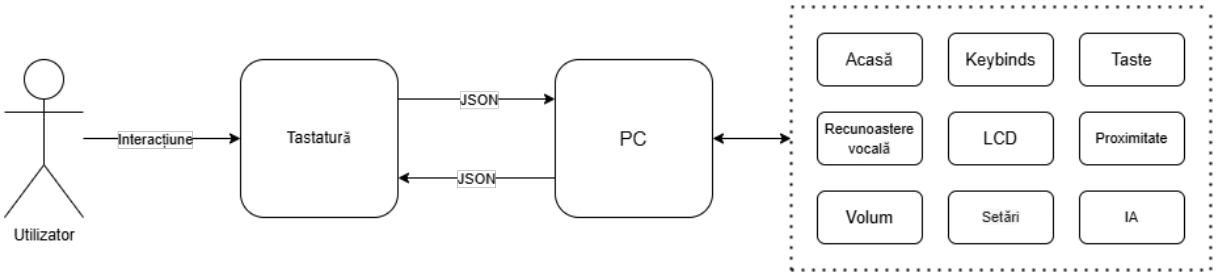

Figura 4.10: Interactiunea cu utilizatorul

Aplicația desktop include mai multe pagini, fiecare fiind afișată în interfață și putând fi selectată din meniul lateral. În Anexe sunt prezentate capturi de ecran pentru fiecare pagină:

- Pagina Taste – *vezi Anexa 7.20*
- Pagina Volum – *vezi Anexa 7.21*
- Pagina LCD – *vezi Anexa 7.22*
- Pagina Keybinds – *vezi Anexa 7.23*
- Pagina Proximitate – *vezi Anexa 7.24*
- Pagina Buzz – *vezi Anexa 7.25*
- Pagina Setări – *vezi Anexa 7.26*

1. **Inițializarea aplicației și jurnalul sesiunii**

   La fiecare pornire, aplicația curăță istoricul sesiunii trecute și scrie un istoric nou pentru sesiunea curentă.

   ```
   - Golește fișierul sesiune_curentă.txt
   - Scrie timpul de început în ambele fișiere
   - La închidere, adaugă timpul final
   ```

   Vezi codul în Anexa 7.10.

2. **Interfață grafică și navigare**

   Interfața este organizată într-un meniu vertical și un panou cu mai multe pagini. Fiecare pagină reprezintă o funcționalitate. Tema este întunecată, cu accente violet.

```
- QListWidget în stânga ca meniu
- QStackedWidget în dreapta pentru conținut
- Paginile UI sunt create dinamic
```

   Vezi codul în Anexa 7.11.

3. **Comunicare serială cu ESP32**

   Comunicarea serială este non-blocantă și rulează pe un thread dedicat. Se citește linie cu linie și mesajul este interpretat, iar acțiunea corespunzătoare este executată.

```
- Deschide portul serial selectat
- Pornește thread de ascultare
- La fiecare linie primită:
  - Dacă e JSON valid:
    - Trimite datele către handler-ul principal
```

   Vezi codul în Anexa 7.12.

4. **Taste și macro-uri**

   Tastele fizice sunt reprezentate în UI de o matrice 3x3. Când ESP32 trimite un eveniment de tip tastă la aplicație, butonul este evidențiat și, dacă are un macro asociat, se execută combinația de taste respectivă.

```
- La eveniment "key pressed":
  - Evidențiază butonul respectiv
  - Caută macro asociat
  - Execută macro cu pynput
```

   Vezi codul în Anexa 7.13 și 7.14.

5. **Control volum**

   Fiecare potențiometru este asociat unui slider vertical. Utilizatorul poate selecta aplicația căreia să îi fie ajustat volumul. Transformarea se face din 0–4095 (pin analogic) în 100–0

```
- La eveniment "volume":
  - Calculează procent
  - Actualizează sliderul
  - Setează volumul aplicației selectate
```

Vezi codul în Anexa 7.15.

6. **Control LCD, SD card si Buzzer**

   Aplicația permite afișarea de fișiere '.raw' pe ecranul LCD, precum și listarea fișierelor de pe cardul SD introdus în cititorul de carduri și acționarea buzzer-ului inclus la o frecvență și durată dorite de utilizator.

```
- Trimite {"cmd": "list_sd"} → ESP răspunde cu fișiere
- Trimite {"cmd": "display_raw", "filename": "..."}
-Trimite {"cmd": "buzz", "freq": ..., "duration": ...}
```

   Vezi codul în Anexa 7.16, Anexa 7.17 și Anexa 7.18

7. **Proximitate**

   Dispozitivul trimite distanța măsurată de senzorul VL53L0X. Aplicația le afișează ca un gradient de culoare în funcție de valoare.

```
- {"type": "proxy", "value": ...} → actualizează UI
```

   Vezi codul în Anexa 7.19.

4.2.3 Recunoașterea utilizatorului cu AI

Componenta AI are rolul de a recunoaște utilizatorul în funcție de comportamentul său în utilizarea dispozitivului: combinație de taste apăsate, pauze între acțiuni, distanță față de dispozitiv și controlul volumului. Algoritmul este antrenat local.

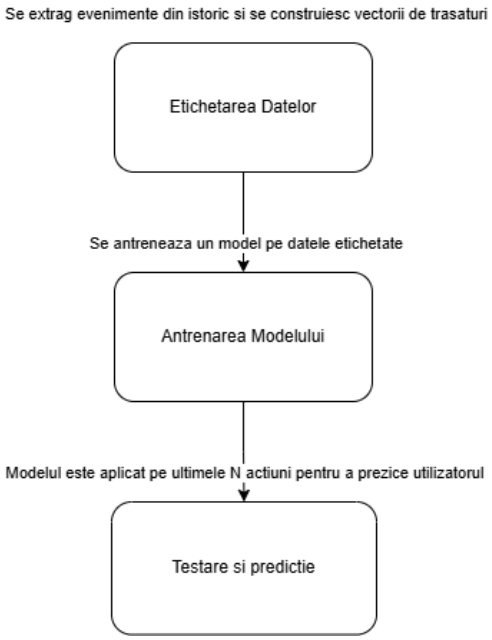

Figura 4.11: Fluxul procesului de IA

1. **Extracția de trăsături din loguri**

   Fișierul de istoric este parcurs linie cu linie, iar mesajele JSON sunt analizate pentru a extrage evenimentele dorite.

   ```
   - Se parcurg liniile din istoric
   - Se detectează evenimente de tip: key, proxy, volume
   - Se construiesc rânduri cu: tastă, delay, distanță, volum
   ```

   Vezi codul în Anexa 7.27.

2. **Etichetarea și salvarea datelor**

   Etapa de etichetare presupune adăugarea unui câmp `label` (ex: ”IVASCU”) pentru fiecare acțiune.

   ```
   - Pentru fiecare vector de trăsături, se adaugă un label: IVASCU, ALT, etc.
   - Se salvează în "labeled.csv"
   ```

   Vezi codul în Anexa 7.28.

3. **Antrenarea modelului**

   Modelul utilizat se antrenează folosind doar trăsături numerice relevante.

- Se folosește RandomForestClassifier(n=50)
- Se antrenează pe: key, delta, distance, vol_value
- Se salvează modelul în "ai_rf_model.pkl"

Vezi codul în Anexa 7.29.

4. **Predicția utilizatorului**

   Pe baza ultimelor N acțiuni extrase dintr-un fișier de istoric curent, modelul poate prezice utilizatorul.

   - Se încarcă modelul
   - Se extrag ultimele 20 de acțiuni dintr-un nou log
   - Se face inferență și se returnează utilizatorul cu cea mai frecventă apariție

   Vezi codul în Anexa 7.30.

Capitolul 5

Rezultate și evaluare

5.1 Funcționalitatea generală

Dispozitivul a fost testat pe parcursul mai multor sesiuni de utilizare, cu scopul de a valida stabilitatea firmware-ului, interfeței grafice și comunicarea serială. Toate funcțiile implementate au fost validate în mediu de lucru (Windows 11), incluzând:

- Afișare de imagini '.raw' pe LCD (testate peste 10 fișiere);
- Detecția tastelor;
- Actualizarea volumului pentru aplicații precum Discord, Spotify, OBS, Chrome și Master Volume;
- Timp mediu de inițializare: **4 secunde** până la afișaj complet operațional;
- Conectivitate Bluetooth stabilă, fără deconectări sau mesaje corupte în sesiuni de peste 2 ore.

5.2 Eficiența AI pentru recunoașterea utilizatorului

Recunoașterea utilizatorului a fost testată pe un set de date etichetat manual, colectat de la 5 persoane de vârste diferite și ocupații diferite (dintre care una este autorul). Pe baza unui model `RandomForestClassifier`, au fost realizate scenarii:

- Set de antrenament: **1000+ acțiuni** etichetate pentru utilizatorul principal;
- Set de test: **100 acțiuni** simulate în sesiuni reale;
- Acuratețe generală pe ultimele 20 de acțiuni: **70.0%**;

5.3 Limitări observate

- Modificarea volumului funcționează doar pe Windows 11, momentan;
- AI-ul poate fi influențat dacă utilizatorul își modifică brusc stilul de interacțiune cu dispozitivul;
- Dispozitivul nu are încă un mecanism de auto-calibrare.

Capitolul 6

Concluzii

6.1 Realizarea temei

Procesul de realizare a proiectului a fost unul complex și cu numeroase provocări, în special în zona de hardware și AI. Deși inițial părea un proiect ușor, construcția unei tastaturi macro cu funcții avansate a necesitat numeroase iterații și adaptări. În total, au fost proiectate și fabricate șase versiuni ale plăcii de circuit imprimat (PCB), până la o variantă complet funcțională.

Partea de comunicare între dispozitivul ESP32 și aplicația desktop a adus la rândul său dificultăți. De asemenea, dezvoltarea firmware-ului pentru ESP32, care trebuia să gestioneze simultan multiple componente (LCD, tastatură, senzori, potențiometre), a fost una dintre cele mai solicitante etape.

În partea de software, provocările nu au lipsit. Arhitectura aplicației desktop a fost construită modular, astfel încât să fie ușor de extins și personalizat. Interfața grafică, sistemul de istoric, integrarea comunicației și suportul pentru macro-uri au fost dezvoltate pas cu pas, cu testare atentă la fiecare pas.

Componentei AI i-a fost acordată o atenție sporită. Implementarea unui sistem de recunoaștere a utilizatorului în funcție de comportamentul său (modul de tastare, distanță, volum) a reprezentat o provocare nouă pentru mine. Datele de antrenament au fost colectate manual de la persoane de vârste diferite și am avut în considerare să fie și stângaci și dreptaci, prin etichetare, și apoi prelucrate pentru a construi vectori de trăsături. Antrenarea și testarea modelului Random Forest au fost realizate local, ceea ce a implicat ajustări manuale.

În ciuda acestor obstacole, proiectul a fost dus până la capăt, păstrând în mare parte planul cu care am plecat. Totuși, în funcție de problemele întâmpinate, unele aspecte au fost adaptate sau simplificate pentru a respecta constrângerile. Rezultatul final este un dispozitiv complet funcțional, bine documentat și deschis către extindere.

6.2 Dezvoltări viitoare

Deși proiectul este complet funcțional, există numeroase posibilități de îmbunătățire și extindere care pot fi aduse în viitor:

- **Implementarea completă a transmisiei audio**: deși microfonul I2S este funcțional și integrat în hardware, captarea și trimiterea fluxului audio în timp real către PC poate sa fie folosita pentru recunoaștere vocală sau control vocal.
- **Suport multi-utilizator avansat**: recunoașterea utilizatorului este deja implementată, însă ar putea fi extinsă cu profiluri personalizate, incluzând macro-uri.
- **Interfață grafică direct pe dispozitiv**: LCD-ul poate fi folosit și pentru o interfață locală, care să permită anumite actiuni direct de pe dispozitiv, fără aplicație PC.
- **Îmbunătățirea modelului AI**: prin colectarea unui set de date mai mare și testarea altor algoritmi (ex: XGBoost, rețele neuronale), precizia recunoașterii comportamentale ar putea crește semnificativ.
- **Suport pentru extensii hardware**: se pot adăuga componente suplimentare, precum encoder-e rotative in loc de potentiometre, touchscreen in loc de LCD sau LED-uri RGB pentru fiecare tasta, extinzând astfel interacțiunea cu utilizatorul.

Aceste direcții pot transforma acest proiect dintr-un proiect de licență într-o platformă extensibilă pentru control avansat, cu aplicații în streaming, gaming, productivitate și educație tehnologică.

6.3 Încheiere

Lucrarea de față a reprezentat un efort care a reunit hardware, software, interfață grafică și inteligență artificială într-un singur proiect. Dispozitivul nu este doar o soluție practică, ci și o bază solidă pentru explorări viitoare în domenii precum automatizare și interacțiune om-mașină.

Capitolul 7

Anexe

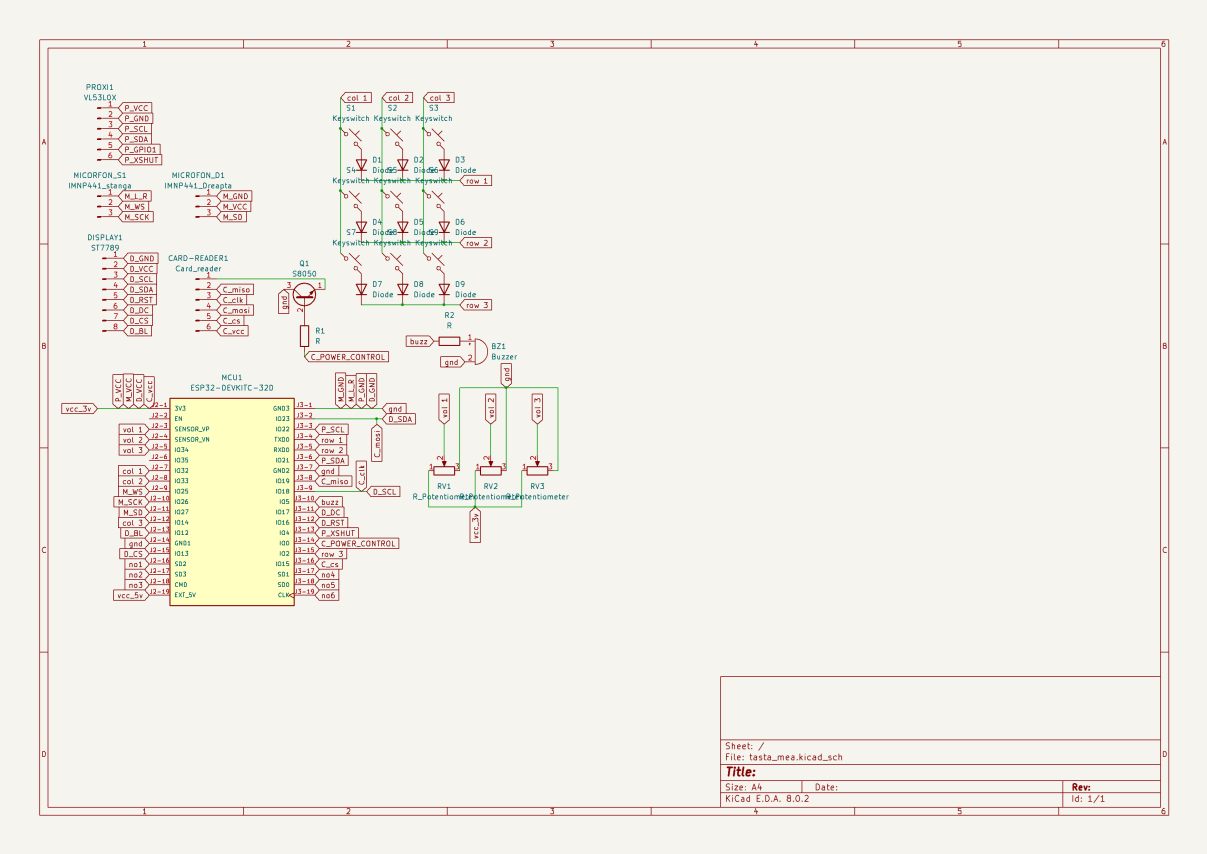

Figura 7.1: Schema logică a circuitului imprimat

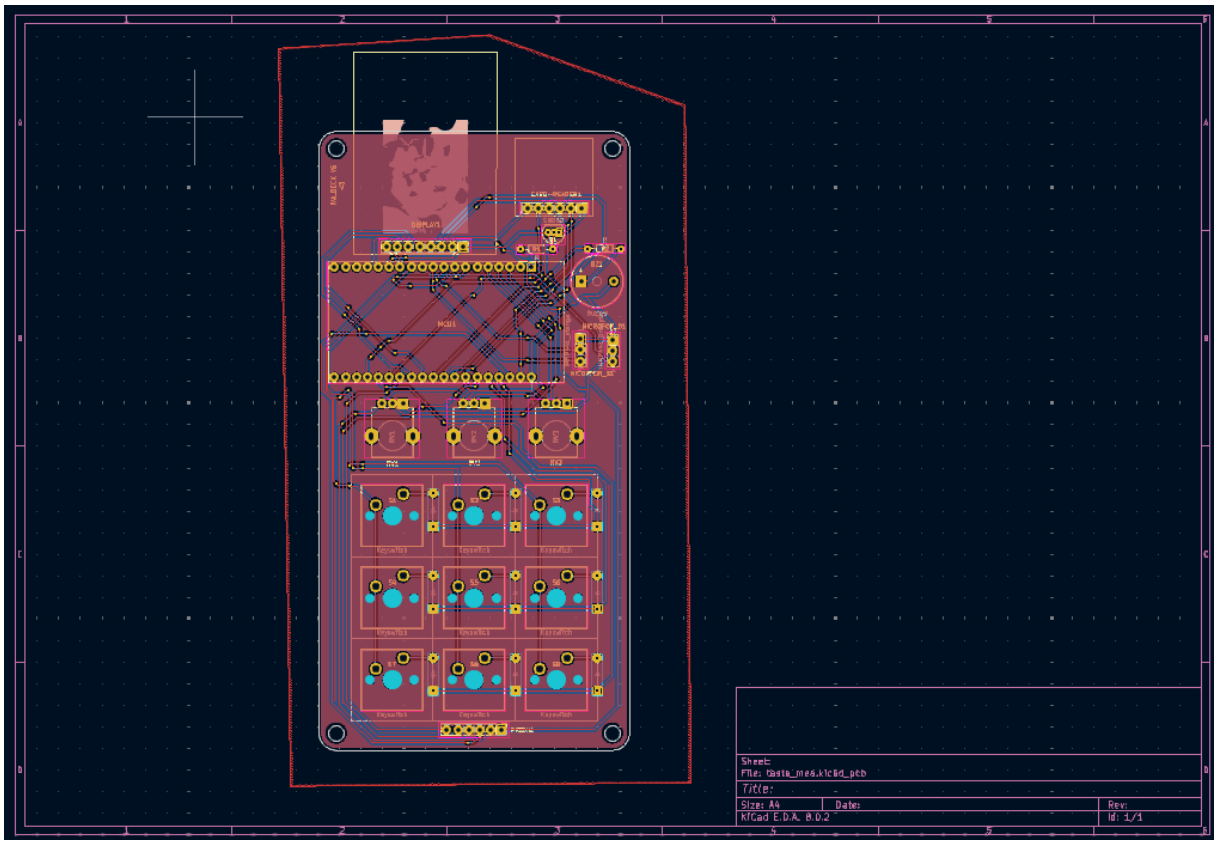

Figura 7.2: Traseele plăcii pe circuit imprimat (PCB)

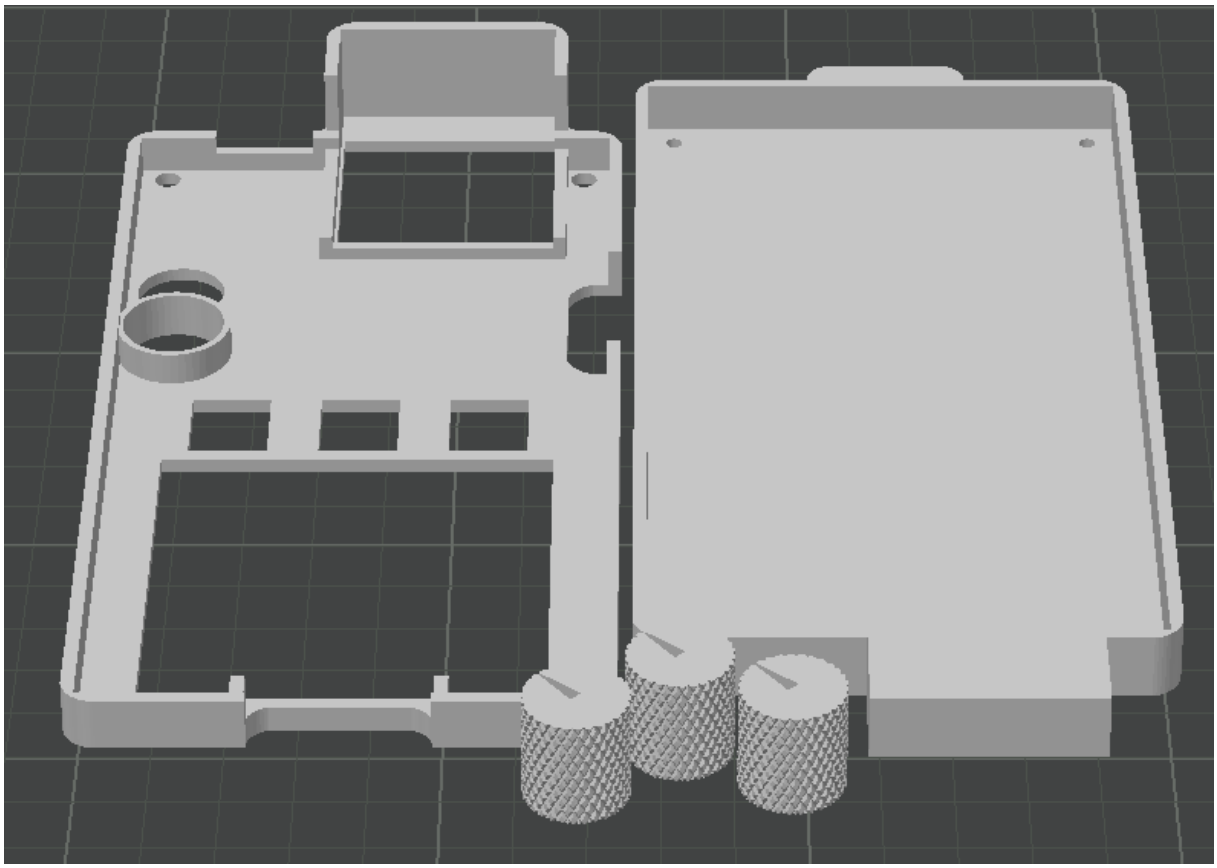

Figura 7.3: Carcasa personalizată a dispozitivului, imprimată 3D

```
void setup() {
  SerialBT.begin("IVA_Deck_FINAL");
  SerialBT.println("Bluetooth Serial pornit");
  delay(300);

  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(PROXY_XSHUT, OUTPUT);
  digitalWrite(PROXY_XSHUT, LOW);
  delay(10);

  spi.begin(SD_SCK, SD_MISO, SD_CLK_MOSI, SD_CS);

  Wire.begin(PROXY_SDA, PROXY_SCL);
  resetSDPower();
  bool sdOK = initSD();

  for (int col = 0; col < numCols; col++) {
    pinMode(colPins[col], INPUT_PULLUP);
  }
  for (int row = 0; row < numRows; row++) {
    pinMode(rowPins[row], OUTPUT);
    digitalWrite(rowPins[row], HIGH);
  }

  testBuzzer();
  initLCD();
  testVL53L0X();
}
```

Figura 7.4: Inițializarea componentelor în funcția `setup()`

```
void setup() {
  SerialBT.begin("IVA_Deck_FINAL");
  SerialBT.println("Bluetooth Serial pornit");
  delay(300);

  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(PROXY_XSHUT, OUTPUT);
  digitalWrite(PROXY_XSHUT, LOW);
  delay(10);

  spi.begin(SD_SCK, SD_MISO, SD_CLK_MOSI, SD_CS);

  Wire.begin(PROXY_SDA, PROXY_SCL);
  resetSDPower();
  bool sdOK = initSD();

  for (int col = 0; col < numCols; col++) {
    pinMode(colPins[col], INPUT_PULLUP);
  }
  for (int row = 0; row < numRows; row++) {
    pinMode(rowPins[row], OUTPUT);
    digitalWrite(rowPins[row], HIGH);
  }

  testBuzzer();
  initLCD();
  testVL53L0X();
}
```

Figura 7.5: Scanarea tastaturii matriceale – funcția `scanKeypadManual()`

```
int volume0Reading = analogRead(Volume0);
int volume1Reading = analogRead(Volume1);
int volume2Reading = analogRead(Volume2);

if (abs(previousVolume0 - volume0Reading) > 50) {
  sendEvent("volume", "0", volume0Reading);

  VL53L0X_RangingMeasurementData_t measure;
  lox.rangingTest(&measure, false);
  sendEvent("proxy", "distance", measure.RangeMilliMeter);

  previousVolume0 = volume0Reading;
}
if (abs(previousVolume1 - volume1Reading) > 50) {
  sendEvent("volume", "1", volume1Reading);

  VL53L0X_RangingMeasurementData_t measure;
  lox.rangingTest(&measure, false);
  sendEvent("proxy", "distance", measure.RangeMilliMeter);

  previousVolume1 = volume1Reading;
}
if (abs(previousVolume2 - volume2Reading) > 50) {
  sendEvent("volume", "2", volume2Reading);

  VL53L0X_RangingMeasurementData_t measure;
  lox.rangingTest(&measure, false);
  sendEvent("proxy", "distance", measure.RangeMilliMeter);

  previousVolume2 = volume2Reading;
}
```

Figura 7.6: Citirea potențiometrelor și trimiterea de mesaje JSON

```
  VL53L0X_RangingMeasurementData_t measure;
  lox.rangingTest(&measure, false);
  sendEvent("proxy", "distance", measure.RangeMilliMeter);
```

Figura 7.7: Citirea senzorului VL53L0X

```
for (int y = 0; y < h; y++) {
  int bytesRead = f.read((uint8_t*)lineBuffer, w * 2);
  if (bytesRead < w * 2) {
    SerialBT.print("{\"warn\":\"short_read\",\"line\":");
    SerialBT.print(y);
    SerialBT.println("}");
    break;
  }

  tft.drawRGBBitmap(0, y, lineBuffer, w, 1);

  if (y % 60 == 0) {
    SerialBT.print("{\"progress_line\":");
    SerialBT.print(y);
    SerialBT.println("}");
  }
}
```

Figura 7.8: Afișarea fișierelor `.raw` pe ecranul LCD

```
// Comenzi JSON
if (!receivingUpload && SerialBT.available()) {
  String msg = SerialBT.readStringUntil('\n');

  StaticJsonDocument<256> doc;
  DeserializationError error = deserializeJson(doc, msg);

  if (!error) {
    String cmd = doc["cmd"].as<String>();
    // Buzz
    if (cmd == "buzz") {
      int freq = doc["freq"] | 1000;
      int dur = doc["duration"] | 300;
      tone(BUZZER_PIN, freq, dur);
      SerialBT.println("{\"status\":\"buzz_ok\"}");
    }
    // List SD
    else if (cmd == "list_sd") {
      File root = SD.open("/");
      if (root) {
        SerialBT.println("{\"status\":\"listing_sd\"}");
        listSD(root);
        root.close();
      } else {
        SerialBT.println("{\"error\":\"sd_open_failed\"}");
      }
    }
    // Display raw
    else if (cmd == "display_raw") {
      String filename = doc["filename"].as<String>();
      filename.trim();
      if (!filename.startsWith("/")) filename = "/" + filename;
```

Figura 7.9: Interpretarea comenzilor JSON primite de la aplicația desktop

```python
LOG_FILE = os.path.join(os.path.dirname(__file__), '../../session_log.txt')
CURRENT_SESSION_FILE = os.path.join(os.path.dirname(__file__), '../../current_session_log.txt')

# La pornire, goli fisierul de sesiune curenta
with open(CURRENT_SESSION_FILE, 'w', encoding='utf-8') as f:
    f.write("")

session_start_time = datetime.datetime.now()
with open(LOG_FILE, 'a', encoding='utf-8') as f:
    f.write(f"\n=== SESSION START: {session_start_time.strftime('%Y-%m-%d %H:%M:%S')} ===\n")
with open(CURRENT_SESSION_FILE, 'a', encoding='utf-8') as f:
    f.write(f"\n=== SESSION START: {session_start_time.strftime('%Y-%m-%d %H:%M:%S')} ===\n")

def log_event(event):
    timestamp = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')[:-3]
    line = f"[{timestamp}] {event}\n"
    with open(LOG_FILE, 'a', encoding='utf-8') as f:
        f.write(line)
    with open(CURRENT_SESSION_FILE, 'a', encoding='utf-8') as f:
        f.write(line)

def end_session():
    session_end_time = datetime.datetime.now()
    end_line = f"=== SESSION END: {session_end_time.strftime('%Y-%m-%d %H:%M:%S')} ===\n"
    with open(LOG_FILE, 'a', encoding='utf-8') as f:
        f.write(end_line)
    with open(CURRENT_SESSION_FILE, 'a', encoding='utf-8') as f:
        f.write(end_line)

atexit.register(end_session)

def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())

if __name__ == "__main__":
    main()
```

Figura 7.10: Inițializarea aplicației și jurnalul sesiunii

```
class MainWindow(QMainWindow):
    highlight_taste_btn = pyqtSignal(int)

    def __init__(self):
        super().__init__()
        self.setWindowTitle("IVA Key App")
        self.setGeometry(100, 100, 1000, 600)

        self.set_dark_theme()
        self.setup_ui()
        self.serial_conn = None
        self.serial_running = False
        self.available_ports = {}
        self.update_serial_ports()
        self.serial_thread = None

        self.keyboard = Controller()
        self.keybinds = {}
        self.load_keybinds()

        self.highlight_taste_btn.connect(self._highlight_taste_btn)

    def setup_ui(self):
        main_widget = QWidget()
        main_layout = QHBoxLayout()
        main_widget.setLayout(main_layout)

        self.menu = QListWidget()
        self.menu.addItems([
            # "Home",  # comentat temporar
            "Taste", "Volum", "LCD", "Keybinds",
            "AI",
            # "Recunoaștere vocală",  # comentat temporar
            "Proximitate", "Buzz", "Setări"])
        self.menu.setStyleSheet("""
            QListWidget {
                background-color: #2a2a3d;
                color: #ffffff;
                font-size: 16px;
                padding: 10px;
            }
            QListWidget::item:selected {
                background-color: #d77aff;
                color: #000000;
            }
```

Figura 7.11: Interfața grafică: structură și navigare

```python
def serial_listen_loop(self):
    import json
    while self.serial_running and self.serial_conn and self.serial_conn.is_open:
        try:
            line = self.serial_conn.readline().decode("utf-8", errors="ignore").strip()
            if line:
                print(f"[UART] Primit: {line}")
                if line.startswith("{") and line.endswith("}"):
                    try:
                        data = json.loads(line)
                        self.handle_uart_message(data)
                    except Exception as e:
                        print(f"[UART] Eroare la parsare JSON: {e}")
                else:
                    pass
        except Exception as e:
            print(f"[UART] Eroare la citire: {e}")
```

Figura 7.12: Thread pentru citirea mesajelor seriale

```python
if data.get("type") == "key" and data.get("key") == "pressed":
    value = str(data.get("value"))
    idx = key_map.get(value)
    if idx and hasattr(self, "taste_buttons"):
        self.highlight_taste_btn.emit(idx-1)
        key = f"T{idx}"
        macro_name = self.taste_macro_map.get(key)
        if macro_name and macro_name != "--Niciun macro--":
            self.simuleaza_macro(macro_name)
```

Figura 7.13: Handler pentru apăsări de taste

```python
def simuleaza_keybind(self, command_str):
    try:
        import keyboard as kb
    except ImportError:
        kb = None
    from pynput.keyboard import Key, Controller
    keyboard = self.keyboard if hasattr(self, 'keyboard') else Controller()
    key_map = {
        "CTRL": Key.ctrl,
        "SHIFT": Key.shift,
        "ALT": Key.alt,
        "CMD": Key.cmd,
        "WIN": Key.cmd,
        "ENTER": Key.enter,
        "ESC": Key.esc,
        "SPACE": Key.space,
        "TAB": Key.tab,
        "BACKSPACE": Key.backspace,
        "UP": Key.up,
        "DOWN": Key.down,
        "LEFT": Key.left,
        "RIGHT": Key.right,
        "MEDIA_PLAY_PAUSE": "play/pause media",
        "MEDIA_NEXT": "next track",
        "MEDIA_PREV": "previous track",
        "MEDIA_PREVIOUS": "previous track",
        "MEDIA_STOP": "stop media"
    }
    keys = [k.strip().upper() for k in command_str.split("+")]
    if kb and len(keys) == 1 and keys[0] in ["MEDIA_PLAY_PAUSE", "MEDIA_NEXT", "MEDIA_PREV", "MEDIA_PREVIOUS", "MEDIA_STOP"]:
        try:
            kb.send(key_map[keys[0]])
            return
        except Exception as e:
            print(f"[Keybind] Eroare la simulare media: {e}")
    parsed_keys = [key_map.get(k, k.lower()) for k in keys]
    try:
        for k in parsed_keys:
            keyboard.press(k)
        for k in reversed(parsed_keys):
            keyboard.release(k)
    except Exception as e:
        print(f"[Keybind] Eroare la simulare: {e}")
```

Figura 7.14: Executarea macro-urilor asignate tastelor

```python
def get_audio_sessions_list(self):
    sessions = AudioUtilities.GetAllSessions()
    app_names = set()
    for session in sessions:
        if session.Process and session.Process.name():
            app_names.add(session.Process.name())
    return sorted(app_names)

def set_windows_volume(self, app_name, percent):
    if app_name == "Master Volume":
        devices = AudioUtilities.GetSpeakers()
        interface = devices.Activate(
            IAudioEndpointVolume._iid_, comtypes.CLSCTX_ALL, None)
        volume = comtypes.cast(interface, comtypes.POINTER(IAudioEndpointVolume))
        volume.SetMasterVolumeLevelScalar(percent/100, None)
    else:
        sessions = AudioUtilities.GetAllSessions()
        for session in sessions:
            if session.Process and session.Process.name() == app_name:
                volume = session._ctl.QueryInterface(ISimpleAudioVolume)
                volume.SetMasterVolume(percent/100, None)
```

Figura 7.15: Slidere pentru controlul volumului

```python
def afiseaza_sd_imagine(self):
    if not (self.serial_conn and self.serial_conn.is_open):
        self.lcd_status.setText("Serială neconectată!")
        return
    filename = self.sd_files_combo.currentText()
    if not filename:
        self.lcd_status.setText("Selectează un fișier!")
        return
    try:
        cmd = {"cmd": "display_raw", "filename": filename}
        self.serial_conn.write((json.dumps(cmd) + "\n").encode())
        self.lcd_status.setText(f"Comandă trimisă pentru {filename}")
    except Exception as e:
        self.lcd_status.setText(f"Eroare: {e}")
```

Figura 7.16: Trimitere de text sau imagini către LCD

```python
def listeaza_sd_card(self):
    import re
    if not (self.serial_conn and self.serial_conn.is_open):
        self.lcd_status.setText("Serială neconectată!")
        return
    try:
        self.sd_files_combo.clear()
        self.lcd_status.setText("Aștept răspuns de la ESP32...")
        self.serial_conn.reset_input_buffer()
        self.serial_conn.write(b'{"cmd":"list_sd"}\n')
        files = set()
        t0 = time.time()
        while time.time() - t0 < 2.5:
            line = self.serial_conn.readline().decode(errors="ignore").strip()
            if not line:
                continue
            if line.startswith('[ESP]'):
                line = line[5:].strip()
            name = line.split()[0] if line else ''
            if name.lower().endswith(('.raw', '.bmp', '.jpg', '.jpeg', '.png')):
                files.add(name)
        if files:
            self.sd_files_combo.addItems(sorted(files))
            self.lcd_status.setText(f"{len(files)} fișiere imagine găsite pe SD.")
        else:
            self.lcd_status.setText("Niciun fișier imagine găsit sau timeout.")
    except Exception as e:
        self.lcd_status.setText(f"Eroare: {e}")
```

Figura 7.17: Listarea fișierelor de pe cardul SD

```python
def trimite_buzz(self):
    try:
        freq = int(self.buzz_freq.text())
        dur = int(self.buzz_dur.text())
        cmd = {"cmd": "buzz", "freq": freq, "duration": dur}
        if self.serial_conn and self.serial_conn.is_open:
            self.serial_conn.write((json.dumps(cmd) + "\n").encode())
            self.buzz_status.setText(f"Comandă trimisă: {cmd}")
        else:
            self.buzz_status.setText("Serială neconectată!")
    except Exception as e:
        self.buzz_status.setText(f"Eroare: {e}")
```

Figura 7.18: Interfață pentru activarea buzzer-ului

```python
def handle_uart_message(self, data):
    if data.get("type") == "proxy" and data.get("key") == "distance":
        value = data.get("value", 0)
        if hasattr(self, "proxy_distance_label"):
            if value <= 30:
                color = QColor(0, 255, 0)
            elif value >= 80:
                color = QColor(255, 0, 0)
            else:
                ratio = (value-30)/50
                r = int(0 + ratio*(255-0))
                g = int(255 - ratio*(255-0))
                color = QColor(r, g, 0)
            self.proxy_distance_label.setText(f"Distanta: {value} mm")
            self.proxy_distance_label.setStyleSheet(f"color: {color.name()}; font-size: 32px; font-weight: bold;")
```

Figura 7.19: Vizualizarea distanței de proximitate în UI

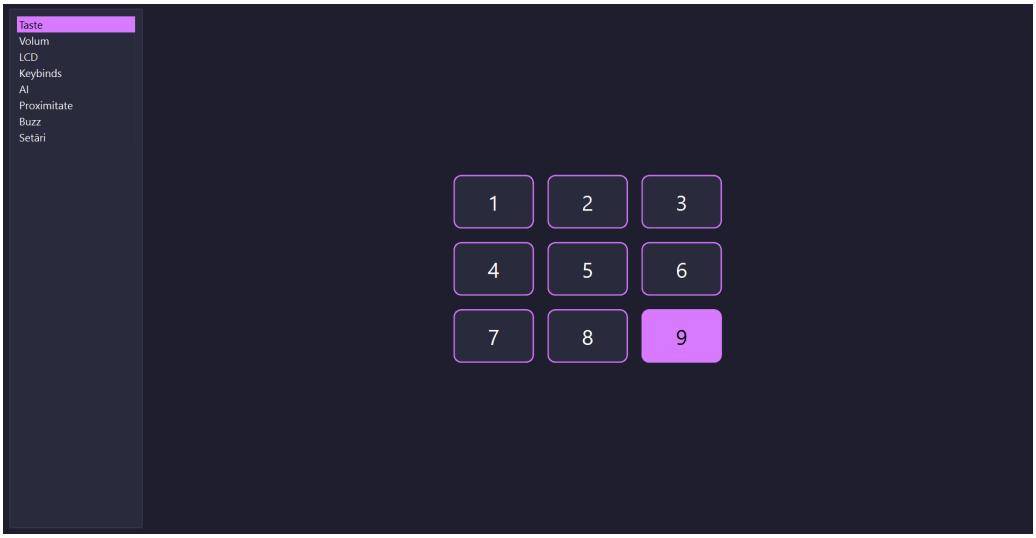

Figura 7.20: Pagina Taste

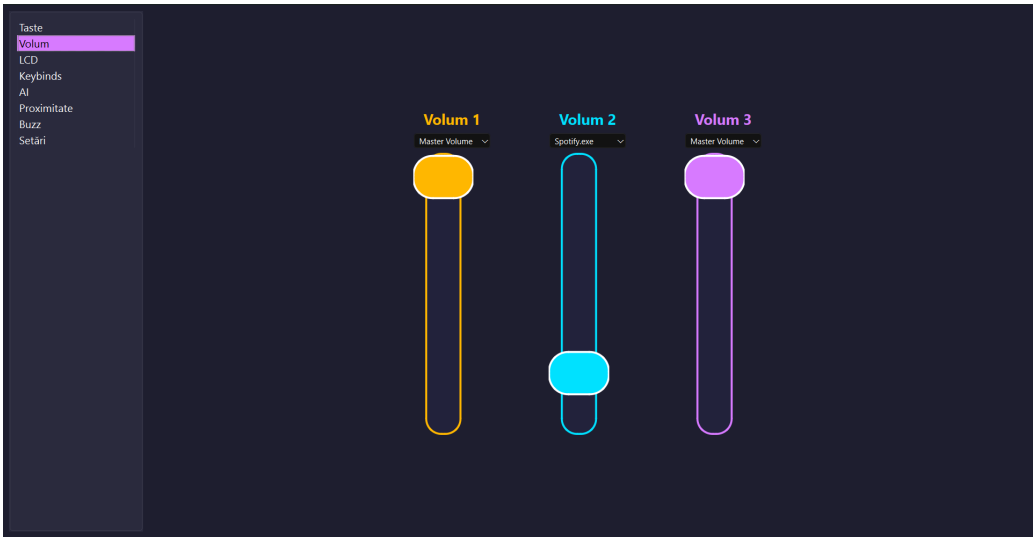

Figura 7.21: Pagina Volum

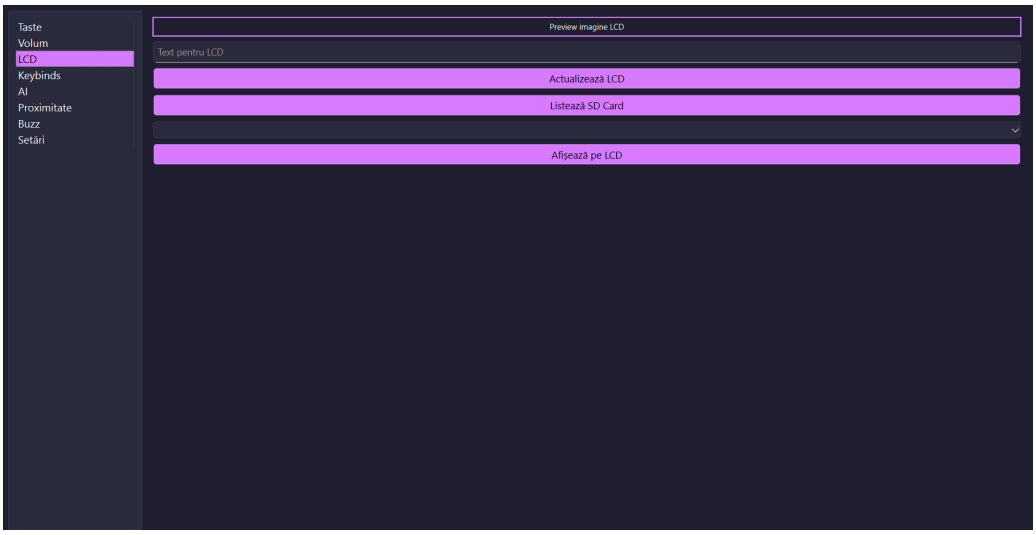

Figura 7.22: Pagina LCD

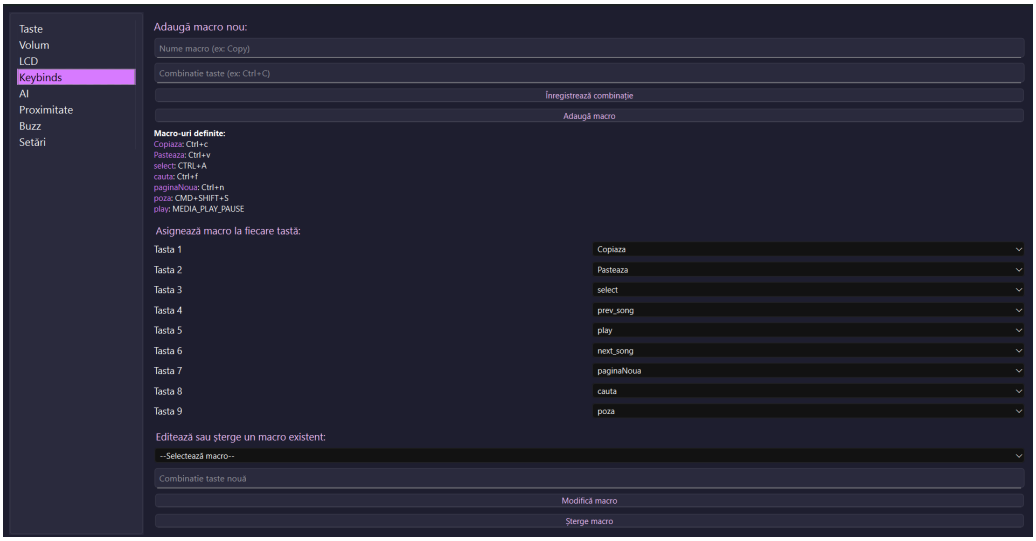

Figura 7.23: Pagina Keybinds

Figura 7.24: Pagina Proximitate

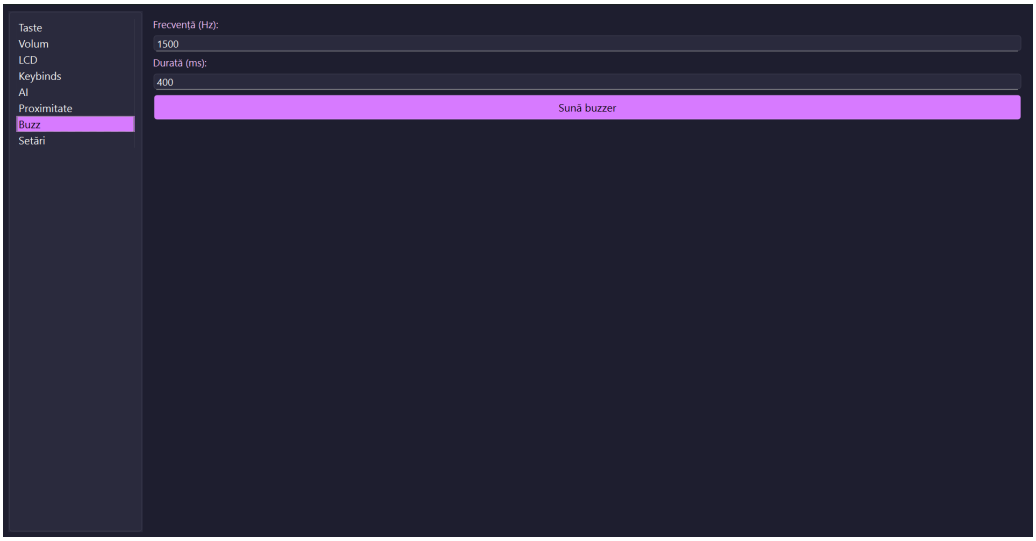

Figura 7.25: Pagina Buzz

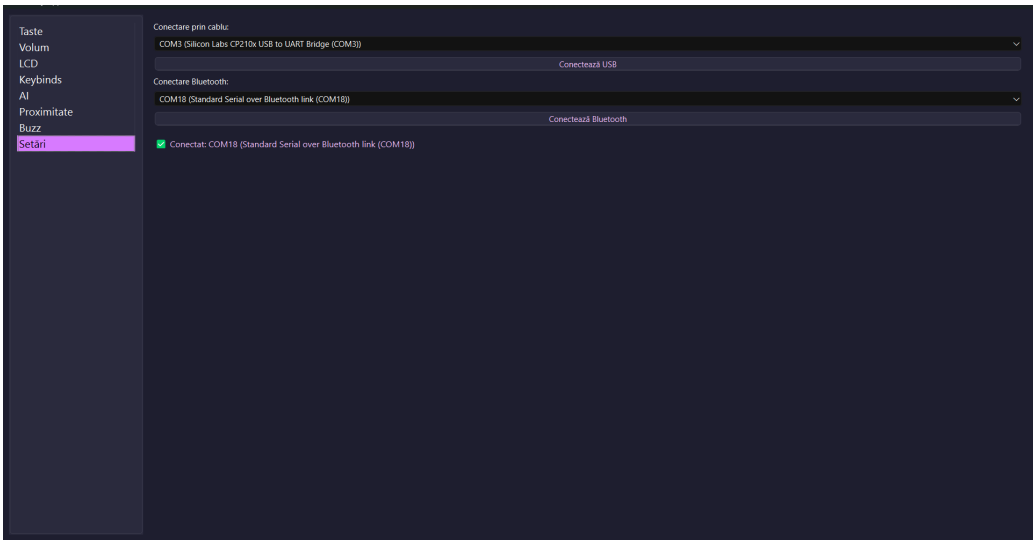

Figura 7.26: Pagina Setări

```python
def parse_log_file(filename):
    rows = []
    with open(filename, encoding="utf-8") as f:
        for line in f:
            m = re.match(r"\\[(.*?)\\] UART: ({.*})", line)
            if not m:
                continue
            timestamp, payload = m.groups()
            if "'type': 'key'" in payload:
                key = int(re.search(r"'value': (\\d+)", payload).group(1))
                rows.append({"timestamp": timestamp, "event": "key", "key": key})
            elif "'type': 'proxy'" in payload:
                dist = int(re.search(r"'value': (\\d+)", payload).group(1))
                rows.append({"timestamp": timestamp, "event": "proxy", "distance": dist})
            elif "'type': 'volume'" in payload:
                key_match = re.search(r"'key': '?(\\w+)'?", payload)
                value_match = re.search(r"'value': (\\d+)", payload)
                if key_match and value_match:
                    vol_key = key_match.group(1)
                    vol_val = int(value_match.group(1))
                    rows.append({"timestamp": timestamp, "event": "volume", "vol_key": vol_key, "vol_value": vol_val})
    return pd.DataFrame(rows)
```

Figura 7.27: Cod pentru parsarea istoricului și extragerea trăsăturilor

```python
def build_features(df):
    features = []
    last_key_time = None
    last_dist = 0
    last_vol_key = None
    last_vol_value = None
    for i, row in df.iterrows():
        if row["event"] == "proxy":
            last_dist = row["distance"]
        elif row["event"] == "volume":
            last_vol_key = row.get("vol_key", None)
            last_vol_value = row.get("vol_value", None)
        elif row["event"] == "key":
            t = pd.to_datetime(row["timestamp"])
            if last_key_time is not None:
                delta = (t - last_key_time).total_seconds()
            else:
                delta = 0
            last_key_time = t
            features.append({
                "key": row["key"],
                "delta": delta,
                "distance": last_dist,
                "vol_key": last_vol_key if last_vol_key is not None else "none",
                "vol_value": last_vol_value if last_vol_value is not None else 0
            })
    return pd.DataFrame(features)

def label_data(features_df, label):
    features_df["label"] = label
    return features_df
```

Figura 7.28: Cod pentru etichetarea datelor și salvarea în fișier CSV

```python
def train_model(labeled_df, model_path="ai_rf_model.pkl"):
    X = labeled_df[["key", "delta", "distance", "vol_value"]]
    y = labeled_df["label"]
    clf = RandomForestClassifier(n_estimators=50, random_state=42)
    clf.fit(X, y)
    joblib.dump(clf, model_path)
    print(f"[INFO] Model salvat în {model_path}")
```

Figura 7.29: Cod pentru antrenarea modelului de recunoaștere RandomForest

```python
def predict_on_log(logfile, model_path="ai_rf_model.pkl", n=20):
    df = parse_log_file(logfile)
    features = build_features(df)
    if len(features) < n:
        print("[WARN] Prea puține acțiuni pentru predicție!")
        return
    X = features[["key", "delta", "distance", "vol_value"]].tail(n)
    clf = joblib.load(model_path)
    preds = clf.predict(X)
    from collections import Counter
    c = Counter(preds)
    print(f"[PREDICT] Predicție pe ultimele {n} acțiuni: {c.most_common(1)[0][0]} ({c})")
```

Figura 7.30: Cod pentru aplicarea modelului AI pe datele din sesiunea curentă

Bibliografie

[1] T Bailey, T Deffenbaugh și Y Kim, „A Macro Keyboard for the dexterity impaired”, în *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, IEEE, 1992.

[2] Cameron Coward, *This macro keyboard incorporates machine learning to interpret gestures*, en, https://www.hackster.io/news/this-macro-keyboard-incorporates-machine-learning-to-interpret-gestures-a801dd92f2a4, Accessed: 2025-6-15, Iun. 2021.

[3] *ESP32*, en, https://www.espressif.com/en/products/socs/esp32, Accessed: 2025-6-15.

[4] Daniele Gunetti și Claudia Picardi, „Keystroke analysis of free text”, en, în *ACM Trans. Inf. Syst. Secur.* 8.3 (2005), pp. 312–347.

[5] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon și Daniel Ramage, „Federated learning for mobile keyboard prediction”, în (2018), eprint: 1811.03604.

[6] Alice Jovanée, *Elgato Stream Deck Plus review: not dialed-in enough*, en, https://www.theverge.com/23517751/elgato-stream-deck-plus-review, Accessed: 2025-6-15, Dec. 2022.

[7] Kevin S Killourhy și Roy A Maxion, „Comparing anomaly-detection algorithms for keystroke dynamics”, în *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, IEEE, 2009.

[8] Seyedmilad Komarizadehasl, Behnam Mobaraki, Haiying Ma, Jose-Antonio Lozano-Galant și Jose Turmo, „Low-cost sensors accuracy study and enhancement strategy”, en, în *Appl. Sci. (Basel)* 12.6 (2022), p. 3186.

[9] Ievgeniia Kuzminykh, Saransh Mathur și Bogdan Ghita, „Performance analysis of free text keystroke authentication using XGBoost”, en, în *Lecture Notes on Data Engineering and Communications Technologies*, Cham: Springer Nature Switzerland, 2023, pp. 429–439.

[10] Nikola Lakovic, Miodrag Brkic, Branislav Batinic, Jovan Bajic, Vladimir Rajs și Nenad Kulundzic, „Application of low-cost VL53L0X ToF sensor for robot environment detection”, în *2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH)*, IEEE, 2019.

[11] Fabian Monrose și Aviel D Rubin, „Keystroke dynamics as a biometric for authentication”, en, în *Future Gener. Comput. Syst.* 16.4 (2000), pp. 351–359.

[12] Panos Sakkos, Dimitrios Kotsakos, Ioannis Katakis și Dimitrios Gunopulos, „Anima: Adaptive Personalized Software Keyboard”, în (2015), eprint: 1501.05696.

[13] Rashik Shadman, Ahmed Anu Wahab, Michael Manno, Matthew Lukaszewski, Daqing Hou și Faraz Hussain, „Keystroke dynamics: Concepts, techniques, and applications”, en, în *ACM Comput. Surv.* 57.11 (2025), pp. 1–35.

[14] D Shanmugapriya și G Padmavathi, „A survey of biometric keystroke dynamics: Approaches, security and challenges”, în (2009), eprint: 0910.0817.

[15] Atharva Sharma, Martin Jureček și Mark Stamp, „Keystroke Dynamics for User Identification”, în (2023), eprint: 2307.05529.

[16] Pin Shen Teh, Andrew Beng Jin Teoh și Shigang Yue, „A survey of keystroke dynamics biometrics”, en, în *ScientificWorldJournal* 2013.1 (2013), p. 408280.

[17] *What is a programmable keyboard?*, en, https://protolab.in/what-is-a-programmable-keyboard/, Accessed: 2025-6-15, Sept. 2022.