UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și tehnologia informației
SPECIALIZAREA: Tehnologia informației

Localizarea unui obiect prin atașarea unui dispozitiv portabil de acesta folosind comunicarea prin unde radio și o aplicație mobilă Key Finder

LUCRARE DE LICENȚĂ

Coordonator științific
Ș.l.dr. Nicolae Botezatu

Absolvent
Bogdan-Iulian Iurea

Iași, 2021

DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII
LUCRĂRII DE LICENȚĂ

Subsemnatul(a) _____,
legitimat(ă) cu _____ seria _____ nr. _____, CNP _____
autorul lucrării _____
_____
_____

elaborată în vederea susținerii examenului de finalizare a studiilor de licență organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi" din Iași, sesiunea _____ a anului universitar _____, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi" din Iași (Manualul Procedurilor, UTI.POM.02 – Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data Semnătura

_____

Cuprins

Localizarea unui obiect prin atașarea unui dispozitiv portabil de acesta folosind comunicarea prin unde radio și o aplicație Key Finder

Bogdan-Iulian Iurea

Rezumat

Prin prezenta lucrare, se expun cunoștințele teoretice și practice acumulate în timpul celor patru ani de studiu în cadrul facultății, cât și în timpul studiului individual extracurricular.

Au fost discutate, argumentate și implementate elemente teoretice și practice ce țin de disciplinele studiate, precum electrotehnică, dispozitive electronice și electronică analogică, electronică digitală, sisteme cu microprocesoare, rețele de calculatoare, programare orientată pe obiecte, programare Web și programarea dispozitivelor mobile.

Prin sistemul prezentat se face posibilă localizarea unui obiect de care este atașat un dispozitiv portabil. Pentru a putea găsi obiectul, prin aplicația mobilă se va putea emite un semnal ce activează rutina de localizare a dispozitivului anexat de acesta. La recepționarea semnalului, dispozitivul va emite semnale sonore și luminoase, astfel fiind posibilă crearea unui reper asupra obiectului ce a fost pierdut.

Au fost studiate în detaliu modalitățile de creare a programelor destinate dispozitivelor mobile, astfel a fost aleasă ca platformă de dezvoltare React Native. Cu ajutorul uneltelor furnizate de cadrul de lucru, se pot crea aplicații destinate fiecărei platforme mobile prin utilizarea aceluiași cod sursă.

Dispozitivul portabil ce poate fi atașat de obiecte este reprezentat de o placă de dezvoltare NodeMCU ESP-32S ce are la bază un microcontroler ESP-WROOM-32 și elementele de circuit conectate la pinii de intrare/ieșire ai acesteia. Dispozitivul a fost implementat doar la nivel de funcționalitate, în acest mod, nu a fost proiectat un prototip fizic care să încorporeze placa de dezvoltare și perifericele anexate. Pentru a programa microcontrolerul, a fost utilizat cadrul de dezvoltare Espressif IoT Development Framework(ESP-IDF).

Pentru ca dispozitivul mobil să poată interacționa cu dispozitivul portabil, a fost utilizat standardul tehnologic de comunicare prin unde radio Bluetooth Low Energy. La nivel de implementare, pentru aplicația mobilă a fost folosită o librărie ce pune la dispoziție o interfață cu ajutorul căreia se poate utiliza tehnologia Bluetooth asemeni metodei de comunicare cerere-răspuns. Pentru dispozitivul portabil, a fost necesară studierea arhitecturii standardului, deoarece librăriile furnizate nu oferă o interfață care să ascundă implementările de nivel jos ale acestuia.

Introducere

Lucrarea prezentată a fost aleasă, deoarece s-a dorit crearea unui sistem care să cuprindă elemente teoretice și practice studiate pe durata celor patru ani de facultate. Simultan, proiectul reprezintă și un obiectiv personal al autorului.

O aplicație mobilă este un program conceput pentru a rula pe un dispozitiv mobil, cum ar fi un telefon, o tabletă sau un ceas. La momentul actual cele mai utilizate sisteme de operare destinate dispozitivelor mobile sunt Android și iOS. O aplicație mobilă nu poate funcționa pe ambele sisteme de operare, deoarece fișierele binare rezultate în urma compilării unui proiect Android nu sunt compatibile cu sistemul iOS și nici invers.

Având în vedere faptul că se utilizează programarea microcontroler, putem spune că, tema cuprinde elemente din domeniul sistemelor încorporate. Un sistem încorporat poate fi considerat un sistem hardware ce poate executa programe software. Poate fi un sistem independent sau poate face parte dintr-un sistem mai mare. Un sistem încorporat este un microcontroler sau un sistem bazat pe microprocesor care este programat pentru a efectua o anumită sarcină.

Bluetooth-ul este un standard al tehnologiei fără fir cu rază de acțiune scurtă care elimină necesitatea de a folosi cabluri pentru a comunica cu mai multe dispozitive electronice. Funcționalitatea acestuia este foarte asemănătoare cu cele ale rețelei mobile sau ale tehnologiei Wi-Fi, dar diferă prin faptul că el este destinat comunicării pe rază scurtă având sarcini relativ simple, în timp ce tehnologiile anterior menționate sunt destinate conectării în masă a mai multor dispozitive într-o rețea largă sau chiar la internet.

Pentru proiectarea programului microcontroler-ului, a componentei hardware și a aplicației mobile, au fost folosite surse dobândite în urma studiului disciplinelor aflate în programa specializării, surse din mediul on-line, dar și surse dinafara curriculei. Astfel, după acomodarea cu dezvoltarea aplicațiilor mobile prin intermediul React Native, cu programarea microcontrolerului ESP32 folosind cadrul de lucru oferit de dezvoltatorii cipului, cât și cu utilizarea elementelor de circuit, au fost identificate necesitățile principale ale sistemului.

Proiectarea aplicației mobile a pornit de la ideea că, prin interfața cu utilizatorul se vor permite următoarele acțiuni: descoperirea serviciilor Bluetooth Low Energy de interes din proximitate, asocierea cu dispozitivele ce publică serviciile, contactarea dispozitivelor cu care s-a creat o asociere și ștergerea asocierilor existente.

După stabilirea funcționalității dispozitivului portabil, au trebuit identificate elementele necesare îndeplinirii acestora. Astfel, pentru ca dispozitivul să poată emite semnale sonore și luminoase, la placa de dezvoltare NodeMCU ESP-32S au fost atașate o diodă RGB și un buzer. Simultan, pentru a verifica dacă dispozitivul este pregătit de asociere, a fost anexat și un buton, care la apăsare are ca efect emiterea unei lumini de culoare albastră/verde ce semnifică dacă dispozitivul este conectat sau nu. Pentru a utiliza elementele de circuit conectate la pinii de intrare/ieșire ai plăcii de dezvoltare, programul pentru microcontroler include o unitate logică ce are în vedere configurarea modului de utilizare a pinilor și crearea unei rutine de întrerupere.

Pentru ca dispozitivul portabil să poată comunica prin unde radio, au trebuit inițializate stocarea non-volatilă, controlerul Bluetooth, stiva Bluetooth și procedura care să gestioneze evenimentele apărute asupra serviciului profilului generic de acces. Unitatea logică care cuprinde cele anterior enunțate a fost realizată prin aplicarea și adaptarea modelului de configurare al unui server Bluetooth Low Energy pus la dispoziție de către dezvoltatorii microcontrolerului.

Capitolul 1. Fundamentarea teoretică

1.1. Tema propusă

Scopul temei propuse este acela de a oferi posibilitatea găsirii obiectelor ce pot fi pierdute prin anexarea unor dispozitive portabile de acestea. Dispozitivele pot fi contactate prin intermediul undelor radio cu ajutorul unui alt dispozitiv ce are rol de emițător. Emițătorul poate trimite semnalul de activare al rutinei de localizare a dispozitivului receptor. La momentul recepționării semnalului, receptorul va emite semnale sonore și luminoase, acestea ajutând la crearea unui reper asupra obiectului de care dispozitivul a fost atașat.

Pentru ca semnalul de activare să poată fi trimis către dispozitivul ce trebuie localizat se va utiliza o aplicație mobilă ce permite vizualizarea dispozitivelor receptor din apropiere. Dintre dispozitive se poate alege unul pentru a realiza o asociere, asocierea fiind făcută prin validarea unei chei de securitate cunoscută de dispozitivul receptor și care trebuie introdusă de utilizator. Dacă există dispozitive cu care s-au realizat asocieri și care să fie și în proximitate, se poate trimite semnalul de activare către unul dintre acestea.

Având în vedere cele enunțate în paragraful anterior, se deduce faptul că dispozitivul emițător este reprezentat de un telefon inteligent. Dispozitivul receptor este reprezentat de o placă de dezvoltare NodeMCU ESP-32S care are la bază un microcontroler ESP-WROOM-32 ce integrează module Bluetooth/Wi-Fi. La pinii de intrare/ieșire ai plăcii de dezvoltare sunt conectate o diodă și un buzer prin care se vor emite semnalele de localizare. Concomitent, este anexat și un buton prin care se va afișa dacă dispozitivul este conectat Bluetooth sau nu.

Interacțiunea dintre dispozitive va fi realizată prin intermediul standardului tehnologic Bluetooth cu energie redusă.

1.1.1. Obiectivele propuse

Luând la cunoștință scopul temei propuse enunțat anterior, obiectivul principal este acela de a oferi posibilitatea găsirii unui obiect prin atașarea unui dispozitiv localizabil de acesta, cu precizarea că utilizatorul nu trebuie să fie deranjat de prezența atașamentului. Localizarea poate fi realizată dintr-o aplicație mobilă, indiferent de moment, iar dispozitivul trebuie să fie de mici dimensiuni și confortabil.

La primirea semnalului de activare, dispozitivul va alterna fiecare culoare a diodei și va emite un semnal sonor. Totodată, la apăsarea butonului din componență, acesta va indica dacă dispozitivul este pregătit de conectare prin emiterea unui semnal luminos de culoare verde sau va arăta dacă dispozitivul este deja conectat prin emiterea unui semnal luminos de culoare albastră.

Pentru aplicația mobilă, obiectivele propuse sunt următoarele:

- aceasta trebuie împărțită într-o parte ce conține strict vizualizarea dispozitivelor din proximitate și modalitatea de asociere cu acestea și într-o parte ce conține doar vizualizarea dispozitivelor deja asociate și operațiunile ce pot fi executate de acestea;
- căutarea dispozitivelor receptor să fie posibilă prin simpla apăsare a unui buton;
- utilizatorul să fie anunțat despre operațiunile ce necesită timp pentru a-și termina execuția(de exemplu: căutarea dispozitivelor din jur) prin diverse metode(cutii dialog cu mesaje de atenționare, loader-e etc.);
- la finalul fiecărei căutări să se poată vizualiza dispozitivele receptor din proximitate prin dispunerea informațiilor acestora într-o listă ce va fi afișată pe ecran;

- pentru a se realiza o asociere cu un dispozitiv portabil, trebuie introdusă cheia de securitate a dispozitivului și apoi trebuie așteptat rezultatul validării acesteia. Validarea cu succes va avea ca efect adăugarea dispozitivului în lista asocierilor realizate, iar în cazul în care validarea a eșuat, utilizatorul va fi anunțat în legătură cu acest lucru;
- dispozitivele asociate trebuie afișate indiferent dacă acestea sunt în proximitate sau nu, dar totodată, să existe o modalitate de a le diferenția;
- pentru un dispozitiv asociat să existe o cale de a elimina asocierea făcută, iar pentru cel aflat și în proximitate să se ofere posibilitatea de a trimite semnalul de activare.

1.1.2. Justificarea abordării

Tema a fost abordată deoarece pachetul de cunoștințe necesar dezvoltării unui astfel de proiect cuprinde o mare parte din programa domeniului studiat. În acest mod, în proiect sunt puse în practică elemente studiate la discipline precum: electrotehnică, dispozitive electronice și electronică analogică, electronică digitală, sisteme cu microprocesoare, rețele de calculatoare, programare orientată pe obiecte, programare Web și programarea dispozitivelor mobile.

Aceasta combină noțiuni ce țin de dezvoltarea aplicațiilor mobile și dezvoltarea sistemelor încorporate, existând posibilitatea ca produsele finale să poată comunica prin intermediul undelor radio. În acest mod, tema cuprinde caracteristici specifice domeniului IoT, un domeniu de actualitate, fapt ce aduce un plus în vederea luării deciziei de a aborda un astfel de proiect.

Totodată, tema a fost aleasă și pentru utilitatea unui astfel de sistem. Prin intermediul acestuia se poate economisi timp, astfel utilizatorul poate fi ușurat de nivelul de stres pe care l-ar dobândi în urma căutării unui obiect.

1.2. Domeniul și contextul abordării temei

Considerând obiectivele anterior enunțate și scopul temei propuse, se deduce că proiectul face parte din domeniul IoT(Internet of Things), deoarece se dorește ca un dispozitiv să fie controlat de la distanță prin intermediul unei aplicații mobile folosind comunicarea „prin aer”.

Internet of Things, tradus „Internetul lucrurilor” face referință la multitudinea de dispozitive ce pot comunica între ele, colectând și schimbând date. Prin aceste dispozitive se dorește ușurarea responsabilităților activităților de zi cu zi, scopul final fiind acela de a îmbunătăți calitatea vieții(eng.: QoL – Quality of Life). Din acest de domeniu fac parte aplicații/produse precum:

- accesorii inteligente(ceasuri, brățări etc.) ce pot informa purtătorul despre nivelul de stres acumulat, temperatura pielii, locația actuală ș.a.m.d. În urma analizei informațiilor primite se pot lua decizii ce pot întreține calitatea vieții purtătorului(de exemplu, se pot lua decizii în privința continuării antrenamentului sportiv, necesității de a reduce nivelul de stres etc.);
- dispozitive ce pot fi localizate prin GPS/Bluetooth(eng.: tracking device) cu ajutorul unui dispozitiv mobil sau a unei aplicații Web;
- aparatură casnică inteligentă(frigider/mașină de spălat/aspirator inteligent) ce poate fi controlată din telefon/calculator.

Pentru a crea aplicații ce țin de domeniul IoT sunt necesare cunoștințe în domeniile dezvoltării de aplicații mobile/Web, dezvoltării de sisteme încorporate și comunicării în rețea.

1.2.1. Sisteme încorporate

Un sistem este un aranjament în care toate unitățile sale lucrează împreună în conformitate cu un set de reguli. De asemenea, acesta poate fi definit ca un mod de lucru/organizare sau o modalitate de a face una sau mai multe sarcini în conformitate cu un plan fix. Într-un sistem, toate sub-componentele depind una de cealaltă.

Un sistem încorporat poate fi considerat un sistem hardware cu software încorporat în acesta. Poate fi un sistem independent sau poate face parte dintr-un sistem mai mare. Un sistem încorporat este un microcontroler sau un sistem bazat pe microprocesor care este proiectat pentru a efectua o anumită sarcină. Un sistem încorporat are trei componente: hardware, software și RTOS(eng.: Real Time Operating System). Un sistem de operare în timp real supraveghează aplicația ce este executată și oferă un mecanism de control al proceselor, definind modul în care funcționează sistemul.

Așadar, un sistem încorporat are la bază un microcontroler/microprocesor ce poate fi programat și care poate utiliza, dacă este cazul, un sistem de control în timp real.

Pentru a programa un microcontroler sunt necesare cunoștințe ale limbajelor de programare de nivel jos, deoarece memoria pusă la dispoziție este limitată, iar limbajele de nivel înalt nu sunt eficiente din punct de vedere al utilizării acesteia.

1.2.2. Tehnologia Bluetooth

Bluetooth-ul este un standard al tehnologiei fără fir cu rază de acțiune scurtă care elimină necesitatea de a folosi cabluri pentru a comunica cu mai multe dispozitive electronice. Funcționalitatea acestuia este foarte asemănătoare cu cele ale rețelei mobile sau ale tehnologiei Wi-Fi, dar diferă prin faptul că el este destinat comunicării pe rază scurtă având sarcini relativ simple, în timp ce tehnologiile anterior menționate sunt destinate conectării în masă a mai multor dispozitive într-o rețea largă sau chiar la internet.

Bluetooth-ul cu energie redusă(eng.: Bluetooth Low Energy) este un standard tehnologic de comunicare prin intermediul undelor radio, ce moștenește Bluetooth-ul clasic, dar care aduce îmbunătățiri asupra consumului de energie cu costul reducerii cantității de date trimise. Dispozitivele care adoptă standardul cu energie redusă pot intra într-un mod inactiv până la sosirea unui nou eveniment de conectare, astfel fiind redusă semnificativ cantitatea de energie utilizată.

Bluetooth-ul, din punct de vedere al implementării hardware este compus din două părți, una analogică radio și una digitală. Partea digitală este numită Host Controller(HC) și conține interfețele cu mediul gazdă, un procesor și modulul de procesare al semnalului digital(Link Controller). Pe nucleu sunt executate instrucțiuni care permit descoperirea și comunicarea cu alte dispozitive prin intermediul protocolului de gestiune al legăturilor(eng.: Link Manager Protocol - LMP). La nivelul software, pentru a asigura compatibilitatea între dispozitive cu implementări hardware diferite, se utilizează o interfață comună între dispozitivul gazdă și nucleul Bluetooth, astfel protocoalele de nivel superior sunt mascate de serviciile din banda de bază cu ajutorul protocolului de adaptare și control al legăturilor logice(eng.: Logic Link Control and Adaptation Protocol – L2CAP).

Standardul încorporează mai multe implementări ale protocoalelor de comunicare, astfel sunt definite protocoale ce stau la baza acestuia(LMP – stabilește și controlează legătura dintre dispozitivele Bluetooth, L2CAP – maschează serviciile din banda de bază, SDP – tabelează serviciile expuse ale altor dispozitive), protocoale de înlocuire a cablurilor(RFCOMM – asigură existența unei legături între două dispozitive folosind frecvența radio), protocoale de control(TCSBIN, HTTP, FTP) și protocoale adoptate(PPP, TCP/IP, UDP).

Tehnologia oferă posibilitatea de a conecta mai multe dispozitive electronice fără a exista un intermediar. Conține propriile implementări asupra menținerii relației de conectivitate dintre dispozitive, oferind posibilitatea alegerii protocolului de comunicare între emițător și receptor. Prin urmare, sunt definite seturi de profiluri Bluetooth, adesea numite servicii sau funcții care expun încapsulări ale funcționalității unui anumit dispozitiv.

Un profil Bluetooth este o specificație prin care se definesc aspecte asupra menținerii unei conexiuni fără fir între dispozitive. Specificația profilului trebuie să conțină un minim de informații asupra utilizării stivei de protocoale Bluetooth, cu scopul de a asigura interoperabilitatea între dispozitive. De-a lungul timpului au fost create și standardizate mai multe profiluri Bluetooth, dintre care, cele cu numărul cel mai mare de aplicații în momentul actual sunt:

- A2DP – Advanced Audio Distribution Profile(are ca scop de trimiterea în flux a fișierelor audio);
- FTP – File Transfer Profile(se ocupă de transferul de fișiere);
- BPP – Basic Printing Profile(are ca rol imprimarea documentelor);
- HFP – Hands-Free Profile;
- GAP – Generic Access Profile(controlul accesului);
- GATT – Generic Attribute Profile(transferul cantităților mici de date eficient din punct de vedere al consumului de energie).

1.2.3. Aplicații mobile

O aplicație mobilă este un program de calculator sau o aplicație software concepută pentru a rula pe un dispozitiv mobil, cum ar fi un telefon, o tabletă sau un ceas. Aplicațiile au fost inițial destinate pentru a spori productivitatea, dar cererea mare a cauzat extinderea rapidă în alte domenii(cum ar fi jocuri mobile, automatizări etc), așa încât la momentul actual există milioane de aplicații disponibile.

La momentul curent cele mai folosite sisteme de operare destinate dispozitivelor mobile sunt Android(dezvoltat de compania Google) și iOS(dezvoltat de compania Apple). O aplicație mobilă nu poate funcționa pe ambele sisteme de operare, deoarece fișierele binare rezultate în urma compilării unui proiect Android nu sunt compatibile cu sistemul iOS și nici invers. Programul rezultat este numit aplicație nativă fiindcă este destinat doar platformei pentru care a fost creat.

Pentru a crea o aplicație Android sunt necesare mediul de dezvoltare integrat(eng.: IDE – Integrated Development Environment) Android Studio, pachetul de dezvoltare Android pentru limbajele Java/Kotlin sau pachetul de dezvoltare nativ pentru limbajele C/C++. Simultan, pentru a crea o aplicație iOS sunt necesare mediul de dezvoltare integrat Xcode și pachetul de dezvoltare iOS. Aplicațiile native iOS pot fi scrise cu ajutorul limbajelor Swift sau Objective C.

Datorită incompatibilității aplicațiilor între platforme, au fost create noi medii de dezvoltare prin care să fie posibilă scrierea aplicațiilor mobile prin utilizarea unui singur cod sursă, programele rezultate fiind numite aplicații multi-platformă(eng.: Cross-Platform). Dintre aceste medii de dezvoltare, cele mai utilizate sunt:

- React Native – mediu de dezvoltare creat peste librăriile native ce face legătura între codul sursă scris și librăriile native ale fiecărei platforme. Oferă o experiență și o performanță apropiată de aplicațiile native;
- Flutter – mediu de dezvoltare ce are ca scop evitarea folosirii librăriilor native prin utilizarea propriei mașini virtuale. Acesta gestionează evenimentele sistemului(gesturi, animații etc.), cât și desenarea elementelor interfeței;
- Apache Cordova – mediu de dezvoltare ce permite dezvoltarea aplicațiilor Web

hibride pentru aplicații mobile folosind tehnologii Web. Majoritatea mediilor de dezvoltare destinate creării de aplicații mobile prin intermediul tehnologiilor Web au la bază această tehnologie(de exemplu: Ionic Framework, Quasar Framework etc.);

- Ionic Framework – oferă instrumente și servicii pentru dezvoltarea de aplicații mobile, aplicații desktop și aplicații Web progresive bazate pe tehnologii și practici moderne de dezvoltare Web. Ca rezultat, aplicațiile nu sunt native deoarece structura acestora este afișată prin vizualizări Web, dar nu sunt nici aplicații Web deoarece acestea pot fi împachetate și au acces și la librăriile native. Acest tip de aplicații sunt numite aplicații multi-platformă hibride.

1.3. Realizările actuale pe aceeași temă. Analiză comparativa

De-a lungul timpului au apărut în comerț diferite abordări în ceea ce prevede tema propusă, dezvoltatorii venind cu diferite soluții pentru a rezolva problema discutată. Astfel, dintre aplicațiile existente, enumerăm: Orbit, chipolo, tile, MYNT, Samsung Galaxy SmartTag, Apple AirTag, eSky Wireless Key Tracker și altele.

Aplicațiile anterior enumerate au același scop, de a localiza un obiect pierdut prin atașarea unui dispozitiv localizabil de acesta, fiecare putând avea funcționalități suplimentare. Astfel, aplicațiile pot diferi în ceea ce presupune cantitatea și calitatea funcționalității oferite, dar și prin modalitatea de a găsi un dispozitiv(folosirea unei telecomenzi, a unei aplicații mobile ce afișează distanța față de dispozitiv, locația acestuia etc.)

Având în vedere produsele existente din categoria temei, proiectul dezvoltat are funcționalități și caracteristici asemănătoare celor aflate pe piață, asemănările dintre acestea fiind dispozitivul portabil implementat, modalitatea de contactare a acestuia și funcția de localizare. Elementele diferite sunt reprezentate de opțiunile suplimentare pe care celelalte produse le oferă, precum localizare GPS, localizare GPS prin realitate augmentată, localizare bidirecțională etc.

Ceea ce este diferit în abordarea temei este modalitatea de a crea o asociere între dispozitivul receptor și dispozitivul mobil. Pentru ca dispozitivul să poată fi localizat este necesară autorizarea prin validarea unei chei de securitate. Cheia este cunoscută de dispozitivul receptor și de deținătorul dispozitivului, astfel accesul pentru cei ce nu cunosc cheia este interzis.

1.3.1. tile

Este o companie ce oferă dispozitive localizabile prin tehnologia Bluetooth Low Energy, permițând deținătorului localizarea unuia cu ajutorul unei aplicații mobile. Dimensiunile dispozitivului sunt relativ mici, acesta venind în diferite forme, precum breloc, abțibild sau card.

La nivel de funcționalitate, dispozitivul poate fi localizat prin semnale sonore, locație GPS, dar și prin analizarea distanței față de acesta. Accesul asupra funcționalităților este pus la dispoziție printr-o aplicație mobilă sau prin intermediul asistenților virtuali(Alexa, Google etc.).

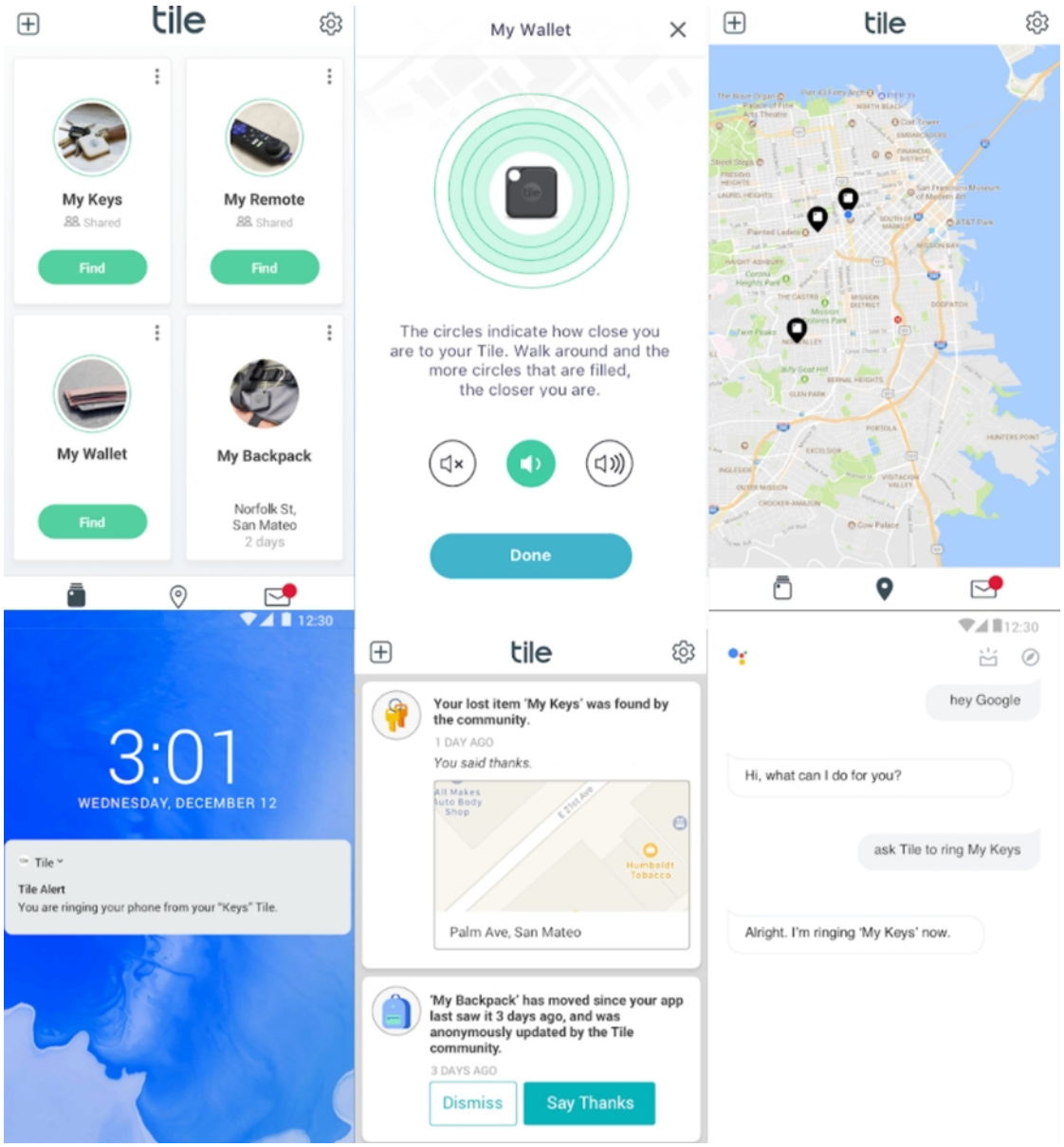

Figura 1.1. Aplicația „tile”

Figura 1.2. Dispozitivul localizabil „tile”

1.3.2. Samsung Galaxy SmartTag

Acesta oferă aceleași funcționalități precum cele ale dispozitivului descris anterior, ceea ce îl diferențiază fiind integrarea în ecosistemul SmartThings dezvoltat de Samsung. SmartThings este o aplicație ce permite controlul și gestiunea tuturor obiectelor „inteligente” ce sunt conectate la rețeaua locală, dar și a celor ce sunt conectate prin Bluetooth Low Energy.

Dezvoltatorii oferă posibilitatea de a verifica locația dispozitivului și prin vizualizarea poziției acestuia folosind realitatea augmentată. Simultan, o funcționalitate ce este greu de atins este aceea că dispozitivul poate fi localizat și prin comunicarea cu orice alte obiecte inteligente ce fac parte din ecosistem, fapt strict dependent de numărul de produse aflate în uz.

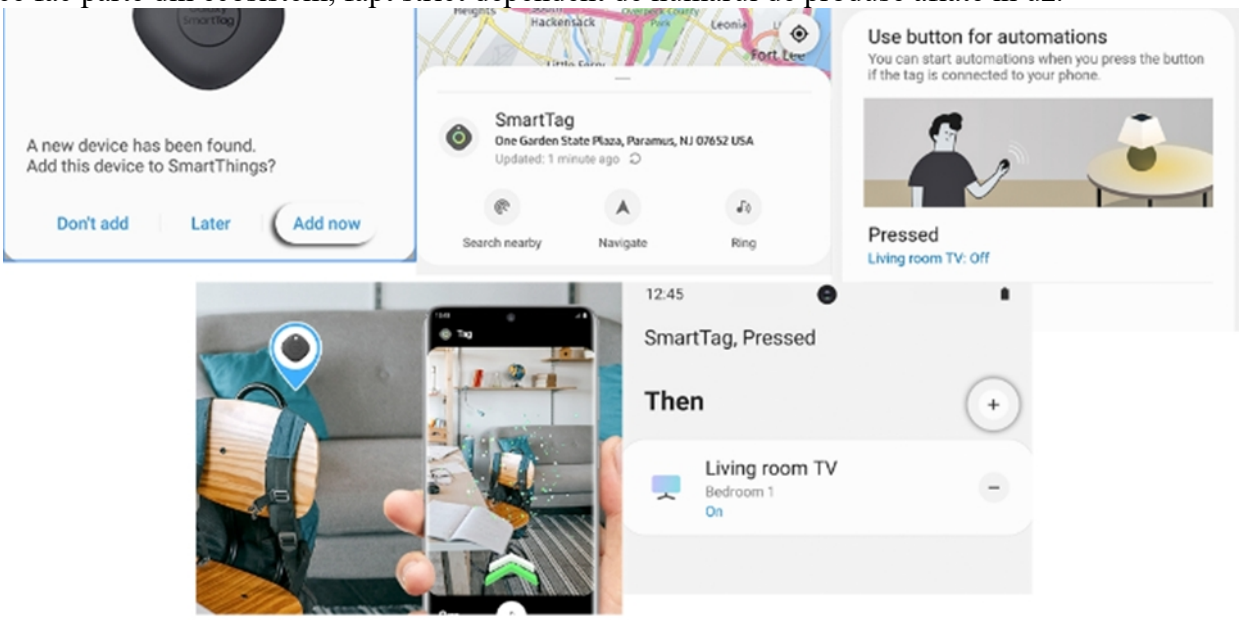

Figura 1.3. Aplicația SmartThings

În paralel cu scopul principal, se oferă și posibilitatea de a modifica funcționalitatea butonului dispozitivului. Printre acțiunile pe care le poate modifica un utilizator se numără: controlul altor dispozitive inteligente(cum ar fi pornirea/oprirea unor becuri inteligente, televizoare smart etc.), trimiterea de notificări sau mesaje prestabilite către o persoană, căutarea dispozitivului mobil asociat ș.a.m.d.

Figura 1.4. Dispozitivul localizator Samsung Galaxy SmartTag

1.4. Specificații privind caracteristicile așteptate de la aplicație

Considerând obiectivele propuse, semnalul sonor al dispozitivului trebuie să poată fi diferențiat foarte ușor de sunetele prezente în mediul înconjurător, astfel, se așteaptă ca acesta să poată fi recunoscut încă de la începutul rutinei de localizare. Simultan, semnalele luminoase trebuie să fie de intensitate maximă așa încât dispozitivul să poată fi localizat și în condiții de iluminare nefavorabile.

Perioada de scanare a dispozitivelor din jur nu trebuie să dureze o perioadă îndelungată de timp, deoarece aceasta ar putea crea o stare de confuzie pentru utilizator. Aceasta trebuie să fie realizată într-un interval de 3 sau 5 secunde.

Funcționalitățile trebuie oferite printr-o cale cât mai simplă, așa încât utilizatorul să nu se simtă deranjat de nivelul de accesibilitate al acestora. Elementele cu care utilizatorul poate interacționa trebuie să fie evidente(butoanele/câmpurile text să fie clare).

Dacă un utilizator este prea depărtat de un dispozitiv și dacă aplicația oferă posibilitatea de a-l contacta, comunicarea dintre acestea fiind imposibilă datorită distanței mult prea mari, utilizatorul va trebui avertizat de acest lucru. Distanța de la care un dispozitiv poate fi contactat trebuie să fie asemenea caracteristicilor standardului Bluetooth Low Energy, iar un dispozitiv trebuie să execute rutina de localizare timp de 4 secunde.

Aplicația mobilă va trebui să fie ușor de folosit și să nu creeze o stare de confuzie pentru utilizator. Funcționalitățile acesteia trebuie să fie ușor de remarcat și înțeles, așa încât utilizatorul să le poată folosi și fără citirea unui manual de instrucțiuni.

Capitolul 2. Proiectarea aplicației

2.1. Analiza platformelor hardware pe care vor fi executate aplicațiile

O platformă hardware este compusă din seturi de elemente hardware compatibile ce permit executarea programelor software. Fiecare astfel de platformă are definit propriul limbaj(limbajul mașină), iar programele ce rulează pe o astfel de platformă sunt special construite pentru fiecare categorie de procesor. Aceasta definește standardul în jurul căruia poate fi dezvoltat un sistem, este o bază de tehnologii pe care sunt construite alte tehnologii sau procese.

Pentru a îndeplini obiectivele propuse sunt necesare două dispozitive ce pot comunica, fiecare având un rol esențial în proces, unul va trimite instrucțiuni/comenzi, iar altul le va intercepta și interpreta.

Au fost analizate toate opțiunile în vederea selectării sistemelor ce vor putea comunica, iar ca rezultat, au fost alese ca inițiator/emițător mulțimea dispozitivelor mobile ce folosesc ca sistem de operare platforma Android, alegere motivată de excelența portabilității unui program pentru o astfel de platformă și de infrastructura de dezvoltare a aplicațiilor destinate acestuia. În același timp, componenta pe post de receptor este reprezentată de microcontrolerul ESP32, alegere motivată de independența cipului de periferice, având integrate module de Bluetooth și Wi-Fi.

ESP32 este o serie de sisteme „low-cost” cu consum redus de energie, având cip integrat Wi-Fi și Bluetooth cu mod dual. Seria ESP32 utilizează un microprocesor Tensilica Xtensa LX6 atât în variante cu două nuclee, cât și în variante cu un singur nucleu. Acesta include comutatoare de antenă încorporate, amplificator de putere, amplificator de recepție cu zgomot redus și module de gestionare a alimentării.

Dispozitivele mobile pe care rulează platforma Android pot avea la bază tipuri diferite de procesoare cu arhitectură ARM(Qualcomm Snapdragon, Samsung Exynos, HiSilicon Kirin etc) cu un număr de nuclee cuprins între unul și opt. Memoria RAM este în general cuprinsă între 0.5 și 8 gigaocteți. Totodată, acestea încorporează o gamă variată de module hardware, precum: modul de camere foto, modul Bluetooth, modul Wi-Fi etc.

Utilizarea tehnologiei Bluetooth în locul tehnologiilor cu funcționalități asemănătoare este motivată de faptul că, specificațiile acesteia sunt conforme cu necesitățile sistemului. Un alt motiv este acela că, tehnologia oferă și posibilitatea comunicării între dispozitive fără a utiliza intermediari care să poată manipula pachetele de date trimise, astfel fiind asigurată și securitatea la nivel de transfer al datelor.

2.2. Modulele proiectului

Un modul este o parte separabilă logic a unui program. În acest mod, proiectul este format din două programe software și o componentă hardware. Unul din programele software este destinat dispozitivelor mobile, iar cel de-al doilea este destinat microcontrolerului ESP32. Componenta hardware este reprezentată de placa de dezvoltare și ansamblul elementelor de circuit conectate la pinii de intrare/ieșire ai acestuia.

Prin aplicația mobilă se dorește accesarea componentei Bluetooth a dispozitivului, așa încât prin intermediul interfeței cu utilizatorul să se poată realiza descoperirea, asocierea și trimiterea comenzilor către dispozitivele-receptor din proximitate. Aceasta va împărțită în 3 module:

- components – încorporează elementele strict vizuale ale aplicației;
- logic – cuprinde implementările funcționalităților programului;

- screens – realizează cuplarea dintre module(Figura 2.1).

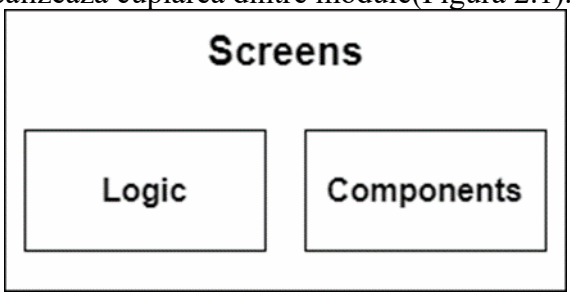

Figura 2.1. Cuplarea modulelor aplicației mobile

Programul pentru microcontroler va aștepta cererile de asociere, va interpreta mesajele primite de la dispozitivele asociate, va lua măsuri în funcție de natura mesajului primit și va gestiona evenimentele de întrerupere apărute în perioada de execuție. Așadar, acesta va cuprinde logica de inițializare și configurare a componentei Bluetooth, dar și logica de configurare a pinilor de intrare/ieșire ai acestuia.

În ceea ce privește componenta hardware, a fost luată în considerare siguranța implementării circuitului electric din punct de vedere electrotehnic. Prin aceasta se permite crearea evenimentelor de întrerupere ale programului microcontrolerului(prin apăsarea unui buton), cu scopul de a afișa starea curentă a controlerului Bluetooth(dacă există dispozitive conectate sau nu). Totodată, aceasta va emite și semnalele sonore și luminoase.

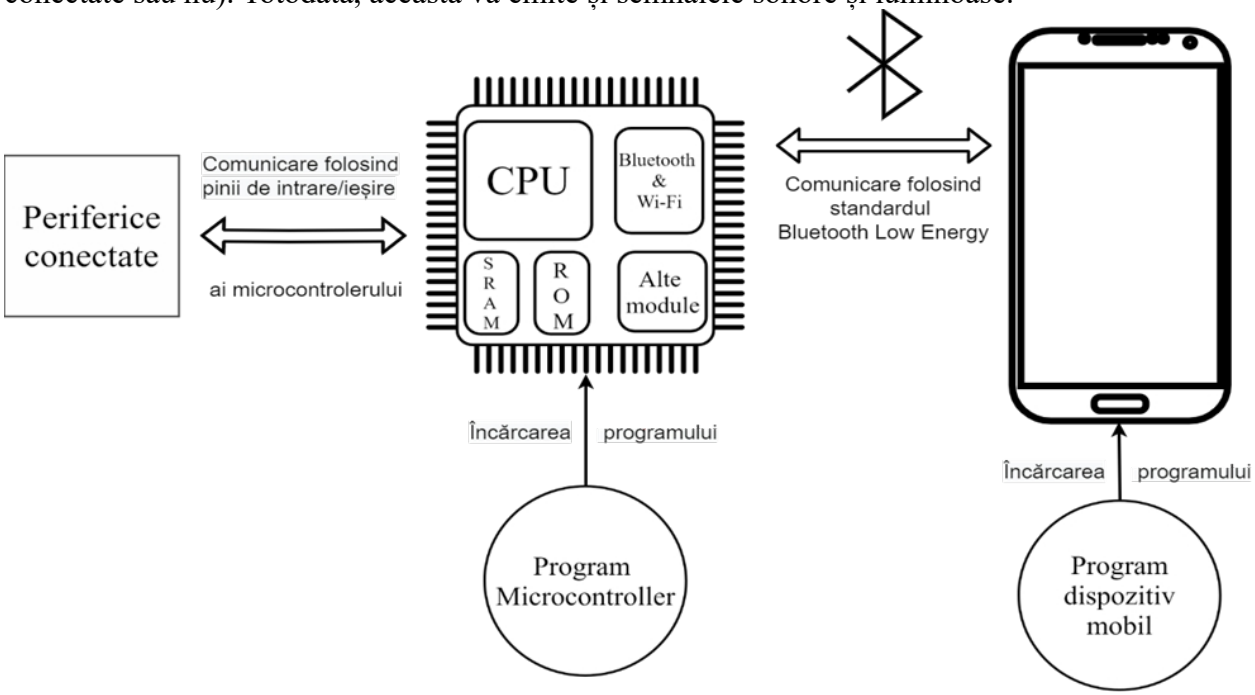

Figura 2.2. Interacțiunea dintre dispozitive

2.3. Avantaje și dezavantaje ale metodei alese

2.3.1. Avantaje și dezavantaje ale aplicațiilor Android

Principalul avantaj al dezvoltării unui program destinat sistemului de operare Android este acela că numărul de dispozitive mobile care folosesc această platformă software este majoritară în raport cu restul platformelor existente pe piață. De-a lungul perioadei de studiu a

platformelor dispozitivelor mobile luate în considerare, au fost evidențiate următoarele avantaje ale aplicațiilor Android:

- sistemul de operare are codul sursă accesibil(eng.: open source);
- pachetul nativ de dezvoltare al software-ului și mediul de dezvoltare sunt bine documentate;
- există multiple medii terțe ce au ca scop dezvoltarea aplicațiilor native prin utilizarea unui singur cod sursă sau a celor care au ca obiect dezvoltarea unui program ce poate rula pe orice platformă(eng.: cross-platform);
- infrastructura de dezvoltare nu este costisitoare;
- rapiditate la nivel de implementare;
- adaptare rapidă la mediul de dezvoltare folosit;
- publicarea aplicației este ușor și rapid de gestionat.

Simultan, cel mai conturat dezavantaj este acela că posibilitatea fragmentării aplicației este mult mai mare datorită numărului larg de dispozitive ce diferă în rezoluție și diagonală. Pentru a asigura compatibilitatea cu toate dispozitivele, perioada de testare a aplicației este lungită considerabil, astfel publicarea acesteia putând fi amânată.

2.3.2. Avantaje și dezavantaje ale microcontrolerului ESP32

Principalul avantaj al utilizării acestui microcontroler este acela că, acesta integrează un set de module ce permit dezvoltatorilor să creeze aplicații scalabile și adaptabile în timp. Simultan cu această idee, microcontrolerul dispune de mai multe avantaje, precum:

- are integrate module Wi-Fi și Bluetooth;
- unitățile de procesare sunt pe 32 de biți și funcționează la frecvențe între 160 și 240 megahertzi;
- pornirea este securizată și are ca scop menținerea integrității hardware și poate fi făcută prin întreruperi produse la nivelul pinilor de intrare/ieșire;
- cuprinde 34 de pini de intrare/ieșire programabili, generator de semnale PWM, și memorie flash criptată;
- consumul de energie redus.

Dezavantajul scos în evidență este acela că prețul de achiziție al cipului este aproape dublu comparativ cu predecesorul său, microcontrolerul ESP8266, fapt motivat de îmbunătățirile și noile funcționalități aduse.

2.3.3. Avantaje și dezavantaje ale tehnologiei Bluetooth Low Energy

Pe durata studiului a fost conturat evenimentul adoptării standardului tehnologic de cât mai multe companii de profil, lucru datorat avantajelor aduse, dintre care enumerăm:

- facilitează consumul redus de energie;
- nu apar interferențe cu alte dispozitive fără fir;
- conectarea mai multor dispozitive la un singur dispozitiv este posibilă;
- securitatea transferului de date este asigurată de faptul că nu există intermediari care să transmită datele de la un capăt la altul;
- compatibilitate asigurată între dispozitivele ce folosesc profilurile existente.

Simultan, au fost identificate și dezavantaje ale acestei tehnologii, printre care se numără:

- conexiunea dintre dispozitive se face doar pe distanțe scurte, ceea ce duce la pierderea acesteia dacă dispozitivele sunt prea depărtate;
- lățimea de bandă este mică;
- dispozitivele rămase pornite ce nu au un cod de securitate pentru asociere pot fi

foarte ușor penetrate de atacuri cibernetice, astfel datele stocate pot fi compromise.

2.4. Limite în care metoda aleasă va funcționa

Având în vedere analiza făcută și avantajele și dezavantajele anterior menționate, se înțelege că aplicația mobilă este destinată dispozitivelor Android, fapt ce creează limitări din punct de vedere al platformei pe care poate fi lansat în execuție programul.

Comunicarea dintre dispozitive este limitată de distanța dintre acestea, astfel se conturează faptul că dispozitivele portabile nu pot fi localizate dacă nu sunt în apropiere.

Este luată în considerare și capabilitatea tehnologiei Bluetooth de a putea oferi suportul ca un dispozitiv să poată avea mai multe conexiuni. Aceasta nu va li luată în calcul deoarece este neutră din punct de vedere al obiectivelor temei, fapt ce duce la o limitare de funcționare.

O altă limitare este aceea că, nu toate dispozitivele mobile au module ce încorporează și tehnologia Bluetooth Low Energy, astfel aplicația mobilă este destinată doar dispozitivelor ce satisfac acest criteriu.

2.5. Componentele proiectului

O componentă reprezintă o parte a unui întreg. Componentele software sunt părți ale unei aplicații sau ale unui sistem, fiecare având un scop unic, astfel complexitatea unei probleme este împărțită în fragmente ușor de gestionat. O componentă software poate fi implementată independent și este supusă compoziției de către terți. O componentă hardware este o unitate fizică autonomă ce poate fi încorporată într-un sistem complex(de exemplu un microcontroler și modulele încorporate de acesta).

2.5.1. Componentele software

2.5.1.1. Aplicația mobilă

Pentru schițarea aplicației mobile au fost luate în considerare obiectivele principale ale temei propuse, astfel dezvoltarea a pornit cu ideea că prin interfața cu utilizatorul se vor permite:

- descoperirea dispozitivelor din jur;
- asocierea cu unul sau mai multe din dispozitivele descoperite prin autorizare folosind o cheie de securitate;
- contactarea dispozitivelor asociate;
- ștergerea unei asocieri existente;

În acest mod a fost creată diagrama de flux(eng.: flow diagram) a ecranelor aplicației, diagramă ilustrată în Figura A.1 din Anexa 1.

2.5.1.1.1. Tehnologia aleasă pentru implementare

Platforma de lucru aleasă pentru program este React Native deoarece aceasta pune la dispoziție uneltele necesare dezvoltării de aplicații pentru orice platformă prin utilizarea unui singur cod sursă. La etapa de compilare trebuie specificată platforma pentru care se creează respectiva aplicație, mediul apoi făcând legăturile dintre librăriile terțe folosite/implementări și librăriile native ale fiecărei platforme.

Avantaje ale framework-ului React Native:

- dezvoltarea unei aplicații React Native este asemănătoare cu dezvoltarea aplicațiilor Web, deoarece acesta are ca bază framework-ul React;
- flexibilitate în ceea ce presupune managementul codului;

- rapiditatea conversiei proiectului multi-platformă către un proiect nativ;
- schimbările aduse codului sunt reflectate în pre-vizualizarea aplicației;
- sprijin deplin din partea dezvoltatorului și a comunității;
- eficient din punct de vedere al costului și timpului;
- folosește un limbaj multi-paradigmă.

Totodată, mediul permite împărțirea elementelor vizuale ale aplicației(eng.: views) în componente independente ce pot fi refolosite. O componentă este o clasă sau o funcție JavaScript ce are ca scop definirea comportamentului și a aspectului unui obiect de pe interfață. Pentru ca o componentă să fie definită corect, aceasta trebuie să conțină o modalitate prin care să se returneze un obiect JSX(JavaScript XML). JavaScript XML este o extensie adusă limbajului JavaScript prin care se poate descrie aspectul interfeței utilizator, în același timp oferind și posibilitatea de a lega variabilele de elementele vizuale(eng.: data binding).

2.5.1.1.2. Modulul pentru funcționalități

Clasa BtManager(Anexa 4.) are în vedere utilizarea componentei Bluetooth a dispozitivului mobil. Aceasta conține metode prin care se oferă posibilitățile de a scana dispozitivele din jur, de a crea o asociere cu acestea și de a trimite pachetul de date ce conține mesajul de activare al funcției de localizare a dispozitivului ce urmează să fie contactat.

În câmpul data al clasei BtManager se încarcă informațiile din fișierul „devices-data.json”, fișier ce conține id-urile serviciilor Bluetooth de interes și câmpuri ce descriu tipul operațiunilor executate de dispozitivele țintă. Câmpul manager reprezintă instanța clasei BleManger ce aparține pachetului react-native-ble-plx. Pachetul oferă o interfață pentru librăriile native ale componentei Bluetooth a oricărui tip de dispozitiv mobil.

Metoda searchForDevices() a clasei BtManager returnează un promise și are rolul de a descoperi dispozitivele Bluetooth Low Energy din apropiere al căror identificator de serviciu de advertising se găsește în lista de id-uri încărcată la momentul creării instanței BtManager. Scanarea se face timp de trei secunde urmând ca apoi să se returneze o listă ce conține detaliile dispozitivelor din proximitate, existând posibilitatea ca aceasta să poată fi și goală în cazul în care nu sunt dispozitive în jur;

Metoda addDevice() are ca scop trimiterea cheii de securitate către dispozitivul receptor. Aceasta primește doi parametri, cheia ce trebuie validată și id-ul dispozitivului pentru care se va face asocierea. Va realiza conexiunea între dispozitive și va descoperi serviciile și caracteristicile Bluetooth ale dispozitivului receptor. Apoi va trimite cheia de securitate către dispozitiv succedată de citirea și returnarea rezultatului obținut în urma validării cheii. În cazul în care cheia de securitate este validă, funcția va returna codul de acces al dispozitivului sau codul de eroare în cazul în care validarea a eșuat.

Metoda findDevice() are în vedere emiterea semnalului de activare al funcției de localizare a dispozitivului receptor, astfel se va trimite către dispozitiv codul de acces alături de mesajul specific operațiunii de găsire. Funcția primește doi parametri, id-ul dispozitivului ce trebuie localizat și codul lui de acces obținut la asociere.

Metodele findDevice() și addDevice() vor arunca erori în cazul în care operația cerută nu poate fi executată(din motive precum: Bluetooth-ul/serviciile de localizare nu sunt pornite, dispozitivul nu este în apropiere sau accesul la serviciile de localizare nu este permis). Pentru a verifica ce fel de eroare trebuie aruncată, a fost creată metoda throwErrorByType(), necesitatea acesteia fiind motivată de faptul că managerul Bluetooth din librăria react-native-ble-plx aruncă un singur tip de eroare(BtError), astfel identificarea acesteia trebuie făcută prin evaluarea mesajului erorii, ci nu prin tipul instanței.

Clasa StorageManager(Anexa 6.) are rolul de a accesa spațiul de stocare al dispozitivului

mobil, iar prin aceasta se dorește memorarea, preluarea și ștergerea informațiilor dispozitivelor asociate. Informațiile unui dispozitiv sunt stocate/preluate prin intermediul unui model de date(StoreDeviceDataModel - Anexa 7.) ce cuprinde detaliile acestuia. Prin model se dorește crearea unei structuri așa încât informațiile ce urmează a fi memorate/preluate să fie ușor de manipulat.

Pentru a scrie în spațiul de stocare al unui dispozitiv sunt folosite obiecte de tipul cheie-valoare(dicționare), existând o constrângere asupra tipului valorii. Valoarea memorată trebuie să fie un șir de caractere, astfel fiind necesare serializarea și deserializarea modelului.

Metodele clasei StorageManger:

- store() – are în vedere scrierea modelului de date în spațiul de stocare al unui dispozitiv, serializându-l și trimițându-l ca valoare. Modelul și cheia la care poate fi găsit sunt primite ca parametri;
- retrieve() – returnează informațiile din spațiul de stocare ce aparțin unui dispozitiv asociat prin accesarea valorii găsite la cheia primită ca parametru. Rezultatul este returnat fără a-l deserializa;
- remove() – elimină ceea ce se găsește în spațiul de stocare pentru cheia primită ca parametru;
- getAllDevices() – returnează informațiile tuturor dispozitivelor asociate sub forma modelului de date StoreDeviceDataModel. Acest lucru este făcut prin preluarea informațiilor stocate ale fiecărui dispozitiv, deserializarea și maparea către model. Fiecare model va fi adăugat într-o listă ce urmează să fie returnată la finalul procesării;
- cleanup() – elimină toate datele stocate de aplicație.

Atât clasa BtManager, cât și clasa StoreManager vor avea o singură instanță pe tot parcursul programului, acestea vor trebui implementate conform șablonului Singleton. Acest fapt este motivat de dorința de a exista instanțe unice în program care să poată accesa componenta Bluetooth a dispozitivului, respectiv spațiul de stocare al acestuia. Clasele nu vor conține un câmp cu o instanță proprie care să fie returnată atunci când este nevoie de aceasta, fapt neconform cu șablonul de proiectare Singleton, funcționalitatea asemănătoare fiind îndeplinită prin crearea unei instanțe și exportarea acesteia(funcționalitate specifică limbajului JavaScript).

2.5.1.1.3. Modulul elementelor strict vizuale

Scopul logicii stocate în acest sub-modul este de a oferi posibilitatea de a reutiliza și grupa componentele React ce nu au un comportament/stare.

Pentru ca o clasă să poată fi o componentă React validă, aceasta trebuie să extindă clasa Component și totodată să suprascrie metoda render()(va returna obligatoriu un obiect J.S.X.). Unei astfel de componente îi pot fi trimise proprietăți, caracteristică asemănătoare cu pasarea parametrilor unei funcții. Valorile proprietăților trimise pot fi accesate prin intermediul câmpului „props” al clasei (de exemplu: this.props.<nume_proprietate>).

Datorită asemănării în implementare a componentelor vizuale, se va descrie o singură componentă ce cuprinde elemente similare cu restul implementărilor, dar și elemente unice, astfel se va descrie clasa AppOverlay(Anexa 8.).

Rolurile proprietăților clasei AppOverlay:

- isVisible – determină dacă trebuie afișată componenta la momentul actualizării ecranului;
- onBackdropPress – callback declanșat la apăsarea pe marginea overlay-ului;
- deviceNameValue/securityCodeValue – au rolul de a afișa șirurile de caractere introduse în câmpurile casetelor text la momentul actualizării ecranului;

- onDeviceNameChange/onSecurityCodeChange – callback-uri ce au în vedere gestionarea evenimentelor de schimbare a șirurilor de caractere introduse în câmpurile casetelor text;
- deviceNameErrorMessage/securityCodeErrorMessage – reprezintă mesaje de eroare pentru introducerea invalidă în câmpurile casetelor text;
- deviceId – afișează id-ul dispozitivului selectat;
- onAddPress – callback ce se execută la momentul apăsării pe butonul de adăugare;
- isLoading – proprietate ce determină tipul de vizualizare a butonului(normal sau loader).

2.5.1.1.4. Modulul de cuplare

În funcție de tipul de platformă pe care este rulată o aplicație mobilă, Android sau iOS, ferestrele principale ale aplicației se numesc activități(eng.: activities), respectiv controlere vizuale(eng.: view controller). Pentru a avea o referință comună asupra ambelor denumiri, elementele vizuale principale ale interfeței vor fi denumite ecrane(eng.: screens).

Aplicația va fi compusă din două ecrane, unul prin care se permite descoperirea dispozitivelor din proximitate și asocierea cu acestea(clasa SearchScreen), iar altul prin care se oferă posibilitatea vizualizării dispozitivelor asociate din apropiere și totodată contactarea acestora(clasa DevicesScreen).

Trecerea de la un ecran la altul se va face prin intermediul unui container de navigare(eng.: navigation container) ce încorporează un navigator de tab-uri (eng.: tab navigator) care conține referințe către ecranele anterior menționate.

Spre deosebire de clasele pur vizuale, clasele ecran sunt proiectate să aibă comportament, vor conține variabile de stare(eng.: state variables). Prin intermediul variabilelor de stare este posibilă redesenarea interfeței în momentul în care una din acestea își modifică valoarea/referința.

Stările clasei SearchScreen(Anexa 10.) au următorul rol:

- overlayVisible – valoare booleană prin care se determină dacă se va afișa formularul de adăugare al unui dispozitiv;
- isLoading – permite afișarea unui loader în locul butonului de căutare cât timp se efectuează căutarea dispozitivelor;
- isCheckingSecurityCode – valoare booleană prin care se permite afișarea unui spinner în locul butonului de adăugare din cadrul formularului de asociere cu un dispozitiv cât timp se așteaptă terminarea execuției acțiunii butonului;
- devices – lista dispozitivelor găsite la momentul ultimei scanări;
- deviceId – id-ul dispozitivului pentru care se dorește asocierea;
- deviceName/securityCode – referințe către valorile introduse în câmpurile formularului de adăugare al unui dispozitiv;
- securityCodeErrorMessage/deviceNameErrorMessage – permit afișarea unor mesaje de eroare în cazul în care câmpurile formularului sunt completate necorespunzător.

În corpul metodei onSearchButtonPress() se apelează metoda searchForDevices() a clasei BtManager și se așteaptă căutarea dispozitivelor din proximitate. La finalul căutării, variabila de stare „devices” va fi actualizată cu noile dispozitive găsite în apropiere(lista poate fi goală) cauzând o redesenare a interfeței. Metoda este asociată evenimentului apăsării pe butonul de căutare situat în antetul ecranului.

După inițializarea clasei SearchScreen(aceasta urmează să fie afișată), se apelează

metoda onSearchButtonPress() pentru a afișa dispozitivele receptor din jur.

Metoda onSubmitAddDevice() are rolul de a gestiona evenimentele ce pot să apară în momentul în care butonul din formularul de asociere este apăsat. Sunt verificate câmpurile din formular și se actualizează variabilele de stare ce au rol de a afișa mesajele de eroare în privința completării incorecte. Apoi se încearcă adăugarea dispozitivului și memorarea cheii de acces a acestuia în cazul în care operația a fost realizată cu succes. Totodată, dacă va apărea o eroare, variabilele de stare „overlayVisible", „deviceName" și „securityCode" sunt resetate.

Spre deosebire de clasa SearchScreen, clasa DevicesScreen(Anexa 9.) are un număr mai mic de stări, deoarece aceasta nu cuprinde la fel de multe funcționalități. Variabila de stare „devices" este o referință către lista dispozitivelor asociate, iar variabila „isLoading" permite afișarea unui spinner în locul butonului de căutare cât timp se efectuează căutarea dispozitivelor asociate din proximitate.

După inițializarea clasei DevicesScreen sunt aduse din spațiul de stocare modelele dispozitivelor asociate și este apelată metoda refreshDevices(), în cazul în care există asocieri. Metoda apelează funcția de căutare a clasei BtManager urmând să filtreze lista obținută prin compararea id-urilor dispozitivelor asociate cu cele ale dispozitivelor din jur. Dacă au fost găsite dispozitive cu id-uri identice, obiectului din variabila de stare ce conține id-ul respectiv i se va seta câmpul „active" pe valoarea „true". Acest câmp are rolul de a diferenția dispozitivele asociate ce sunt în proximitate de cele care nu, simultan restricționând și accesul asupra funcției de localizare a fiecărui dispozitiv.

Metoda onFindPress() este asociată fiecărui obiect din lista-variabilă de stare „devices" și are rolul de a apela funcția de căutare a fiecărui dispozitiv asociat, iar metoda onDeletePress() are rolul de a șterge asocierea cu un dispozitiv.

2.5.1.2. Programul pentru microcontroler

2.5.1.2.1. Tehnologia aleasă pentru implementare

Platforma de lucru aleasă pentru program este Espressif IoT Development Framework(ESP-IDF), deoarece aceasta este interfața de programare oferită de dezvoltatorii microcontrolerului ESP32. Prin aceasta se pun la dispoziție un sistem de monitorizare al aplicației și librăriile de nivel jos necesare dezvoltării unui program(Figura 2.3). Această alegere este motivată și de faptul că restul opțiunilor luate în calcul nu au o manevrabilitate completă asupra microcontrolerului, dezvoltatorii acestora oferind interfețe ce încapsulează doar o parte dintre funcționalitățile librăriilor fiecărui cip.

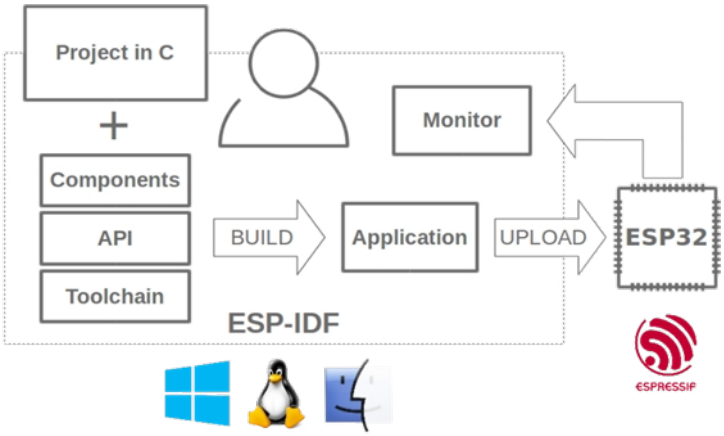

Figura 2.3: Uneltele ESP-IDF

Trebuie luat în vedere și faptul că, deși, platforma este foarte potentă din perspectiva

creării unui program, aceasta este și dificil de folosit datorită necesității de a avea cunoștințe asupra limbajului de programare de nivel jos pe care îl are la bază(limbajul C).

Mediul de dezvoltare integrat ales pentru a crea proiectul este PlatformIO(Figura 2.4), o extensie adusă editorului Microsoft Visual Studio Code ce accelerează și simplifică crearea și livrarea produselor încorporate.

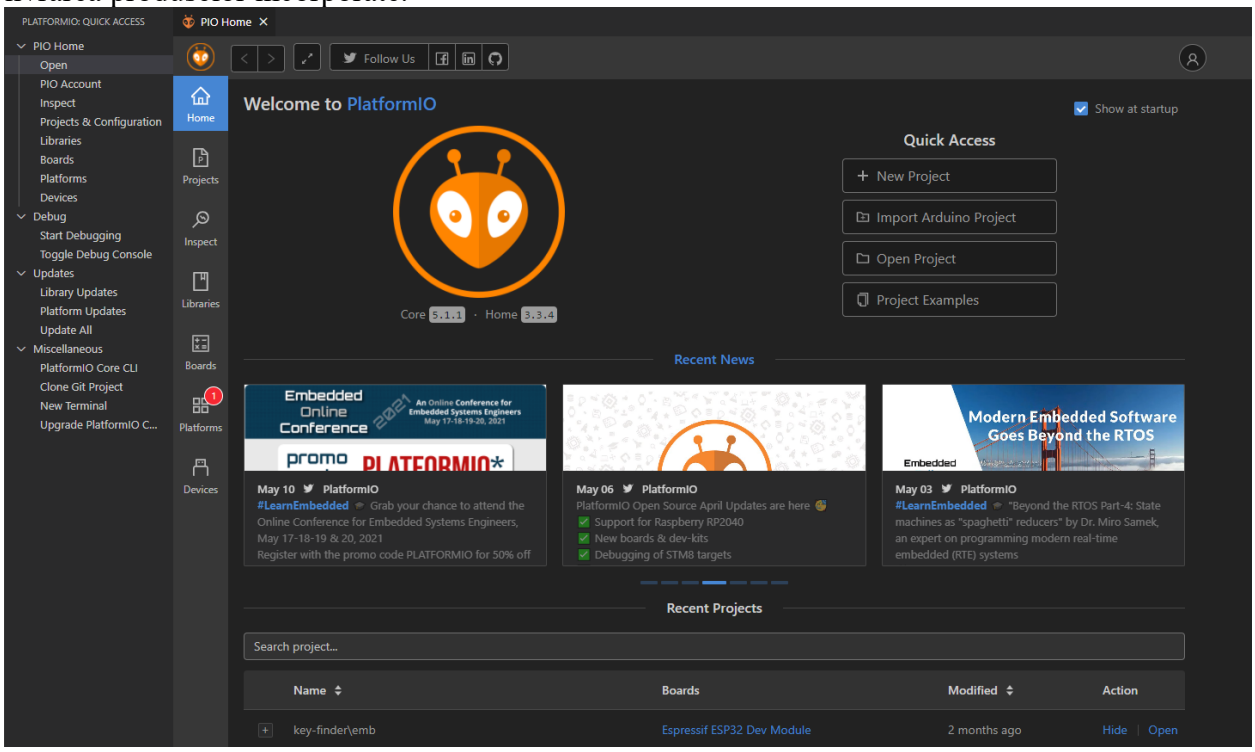

Figura 2.4: Mediul de dezvoltare integrat PlatformIO

Luând la cunoștință paragrafele anterior enunțate, primul pas în proiectarea programului a fost acela de a crea o diagramă prin care să fie redat fluxul aplicației(Figura 2.5).

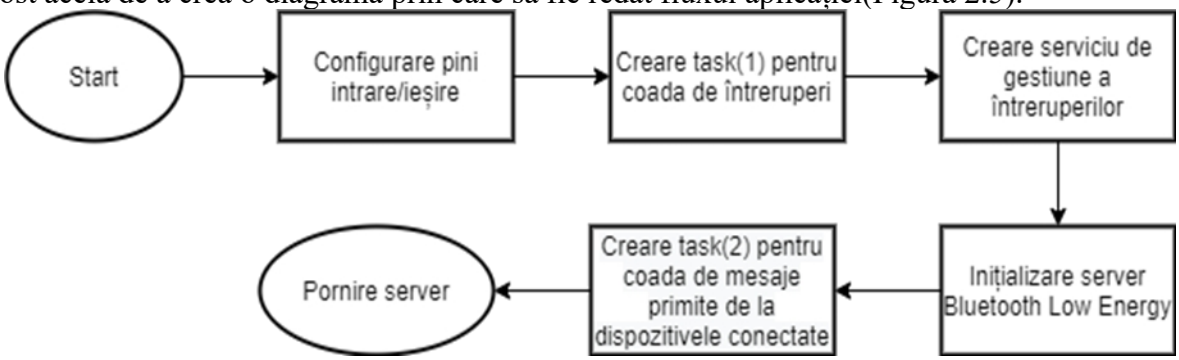

Figura 2.5: Diagrama de flux a aplicației microcontrolerului

2.5.1.2.2. Configurarea pinilor de intrare/ieșire

Utilizarea pinilor de intrare/ieșire ai microcontroler-ului necesită cunoașterea dispozitivelor legate de aceștia, fapt enunțat în subcapitolul 2.5.2. Astfel pentru a folosi pinii la care sunt conectate dioda, butonul și buzerul a fost creată funcția kf_config_gpio() – Anexa 12. Funcția conține logica de configurare a pinilor de ieșire așa încât prin aceștia să se poată trimite valori de 1 logic sau semnale PWM(eng.: pulse-width modulation) către periferice. Simultan, aceasta va conține și logica creării evenimentelor de întrerupere la momentul schimbării valorii

logice găsite la pinul setat ca intrare.

Modulația în lățime a impulsului este o tehnică pentru obținerea rezultatelor analogice prin mijloace digitale. Controlul digital este utilizat pentru a crea o undă pătrată(eng.: square wave), un semnal comutat între pornit și oprit.

Controlul intensității luminii emise de diodă nu este necesar, astfel pentru pinii asignați acesteia vor fi trimise doar valori de 1 logic, iar pentru pinul buzerului va fi necesară trimiterea semnalelor PWM fiindcă acesta nu poate funcționa altfel.

Pentru a localiza dispozitivul receptor a fost creată funcția kf_find() –Anexa 12. Aceasta cuprinde logica emiterii semnalelor luminoase și sonore într-un mod alternant. Aceasta primește un parametru ce are rolul de a seta perioada de emitere a semnalelor de localizare.

Modul de funcționare al rutinei de întreruperi este ilustrat prin intermediul unei scheme logice(Figura A.2). Trebuie avut în vedere faptul că, o întrerupere trebuie tratată cât de rapid posibil, astfel, funcția ce are rolul de a gestiona evenimentele apărute trebuie încărcată în memoria RAM pentru a optimiza accesul la aceasta.

2.5.1.2.3. Configurarea serverului Bluetooth Low Energy

Având în vedere API-ul ales pentru programarea microcontrolerului, nu există o modalitate care să facă ușoară utilizarea controlerului Bluetooth al dispozitivului, astfel pentru a configura serverul trebuie:

- inițializată stocarea non-volatilă;
- inițializat și activat controlerul Bluetooth;
- inițializată și activată stiva Bluetooth;
- create funcțiile de gestiune pentru profilul generic de atribut și pentru profilul generic de acces;
- setată limita de date maxim transmise;
- creat task-ul ce gestionează mesajele primite conform cu schema logică din Figura A.3.

Codul sursă al configurării serverului Bluetooth este găsit în Anexa 11.

2.5.2. Componenta hardware

Pentru a facilita procesul de dezvoltare al sistemului, a fost utilizată o placă de dezvoltare NodeMCU ESP-32S ce are la bază un microcontroler ESP-WROOM-32. Aceasta permite încărcarea și monitorizarea programului microcontrolerului printr-un cablu USB, eliminând nevoia de a folosi alte periferice. Totodată, la pinii de intrare/ieșire ai plăcii de dezvoltare sunt conectate un buzer pasiv, o diodă RGB și un buton(Figura 2.5).

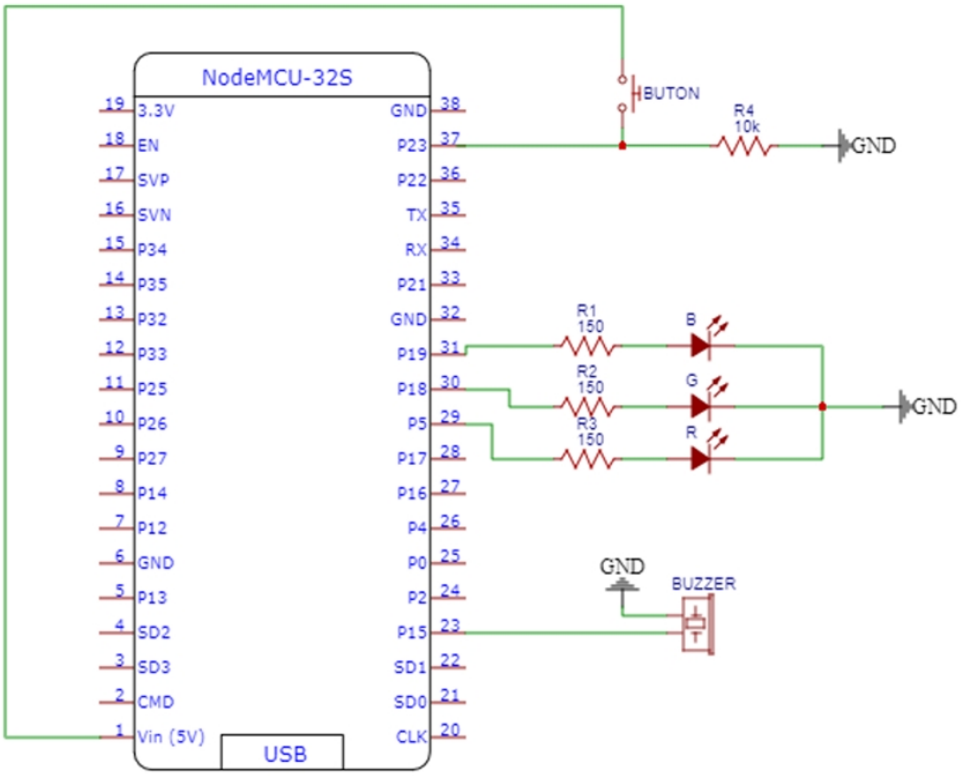

Figura 2.6: Schema componentei hardware

Capitolul 3. Implementarea aplicației

3.1. Descrierea generală a implementării

3.1.1. Aplicația mobilă

Având în vedere uneltele mediului de lucru folosit pentru a crea aplicația mobilă, au fost abordate paradigma orientată obiect, paradigma funcțională, principiul separării preocupărilor(eng.: Separation of Concerns – SoC), principiul responsabilității, principiul întreținerii și reutilizabilității și principiul coeziunii și al cuplării.

Fiecare funcționalitate a aplicației mobile prezentată la capitolul 2.5.1.1. a fost implementată treptat, urmând diagrama de activități prezentată în Figura A.1. Tema de culori a aplicației este încărcată dintr-un fișier JavaScript extern ce conține codurile hexazecimale ale culorilor principale, secundare și suplimentare ale interfeței cu utilizatorul.

Cunoscând faptul că JavaScript este un limbaj ce are un singur fir de execuție, dar care este totodată și asincron, au fost utilizate callback-uri, cu precizarea că au fost evitate implementările asemenea „callback hell” prin utilizarea promise-urilor și a funcțiilor asincrone.

3.1.2. Programul pentru microcontroler

Configurarea pinilor microcontrolerului a fost realizată prin utilizarea unei structuri de configurare(gpio_config_t) ce are în componență o mască de 64 de biți, fiecare poziție semnificând numărul pinului folosit ca intrare/ieșire. Valoarea(de 1 sau 0) găsită la fiecare poziție arată dacă pinul asociat poziției este folosit sau nu. Structura conține detalii despre activarea întreruperilor, modul de utilizare(intrare/ieșire) și tipul de rezistor folosit pentru fiecare pin menționat în mască. Totodată, în această funcție sunt create coada de întreruperi, serviciul ce adaugă elemente în coadă la momentul producerii unei întreruperi și task-ul ce gestionează elementele aflate în coadă. Configurarea pinului la care este conectat buzerul pasiv se face folosind librăria „ledc”, deoarece prin aceasta se oferă o interfață prin care se pot genera semnale PWM.

Pentru a crea serverul Bluetooth Low Energy a fost studiat și adaptat modelul din documentația oferită de dezvoltatorii ESP-IDF, astfel serverul are următoarea configurație:

- Există doar un profil ce expune un serviciu cu o caracteristică ce are un singur descriptor;
- Citirea caracteristicii serviciului publicat va returna codul de asociere al serviciului dacă cererea a fost precedată de o autorizare realizată cu succes și va reseta variabila ce verifică dacă cererea este autorizată. Dacă cererea nu este autorizată, atunci se va returna mesajul „NO_DATA”;
- Scrierea pe caracteristica serviciului va avea ca efect trimiterea mesajului scris către un task. Task-ul va interpreta mesajul și va lua măsuri în funcție de natura acestuia. Dacă mesajul este egal cu cheia de securitate, atunci se va pune pe „true” variabila de autorizare a serviciului, iar dacă mesajul este reprezentat de codul de acces concatenat cu mesajul specific operației de localizare, atunci se va inița rutina de localizare a dispozitivului;
- Evenimentul de deconectare va reseta variabila ce are ca scop memorarea răspunsului validării cheii de securitate;
- Unitatea maximă de transfer este setată la 23 octeți.

```
[0;32mI (523) cpu_start: Starting scheduler on PRO CPU.[0m
[0;32mI (0) cpu_start: Starting scheduler on APP CPU.[0m
[0;32mI (531) gpio: GPIO[5]| InputEn: 0| OutputEn: 1| OpenDrain: 0| Pullup: 0| Pulldown: 0| Intr:0 [0m
[0;32mI (541) gpio: GPIO[15]| InputEn: 0| OutputEn: 1| OpenDrain: 0| Pullup: 0| Pulldown: 0| Intr:0 [0m
[0;32mI (551) gpio: GPIO[18]| InputEn: 0| OutputEn: 1| OpenDrain: 0| Pullup: 0| Pulldown: 0| Intr:0 [0m
[0;32mI (561) gpio: GPIO[19]| InputEn: 0| OutputEn: 1| OpenDrain: 0| Pullup: 0| Pulldown: 0| Intr:0 [0m
[0;32mI (571) gpio: GPIO[23]| InputEn: 1| OutputEn: 0| OpenDrain: 0| Pullup: 0| Pulldown: 0| Intr:1 [0m
[0;32mI (591) BTDM_INIT: BT controller compile version [3723d5b][0m
[0;32mI (591) system_api: Base MAC address is not set[0m
[0;32mI (591) system_api: read default base MAC address from EFUSE[0m
[0;32mI (701) phy: phy_version: 4500, 0cd6843, Sep 17 2020, 15:37:07, 0, 0[0m
[0;32mI (1051) Key Finder: EVENIMENT DE ÎNREGISTRARE, Status={0}, ID={0}
[0m
[0;32mI (1061) Key Finder: CREARE SERVICIU DE ADVERTISING[0m
[0;32mI (1081) Key Finder: PORNINREA SERVICIULUI, Status={0}
[0m
[0;32mI (1081) Key Finder: ADAUGARE DESCRIPTOR CARACTERISTICA
[0m
[0;32mI (1081) Key Finder: ADAUGAREA DESCRIPTORULUI A REUSIT, Status={0}
```

Figura 3.1: Configurarea pinilor de intrare/ieșire și a serverului Bluetooth Low Energy

```
Key Finder: EVENIMET DE CONECTARE, ID={55:80:4b:0b:37:c7}[0m
Key Finder: EVENIMENT DE SCRIERE[0m
Key Finder: RGPZ6644[0m
Key Finder: CERERE DE AUTORIZARE[0m
Key Finder: EVENIMENT DE CITIRE - USER AUTORIZAT

Key Finder: EVENIMENT DE DECONECTARE, MOTIV={0x13}[0m
Key Finder: EVENIMET DE CONECTARE, ID={55:80:4b:0b:37:c7}[0m
Key Finder: EVENIMENT DE SCRIERE[0m
Key Finder: EFT324AAFIND[0m
Key Finder: CERERE GĂSIRE DISPOZITIV[0m
Key Finder: EVENIMENT DE DECONECTARE, MOTIV={0x13}[0m
```

Figura 3.2: Cererea de asociere cu și cererea de localizare

3.1.3. Implementarea componentei hardware

Pentru a implementa componenta hardware a fost urmată schema din Figura 2.6. Astfel cablajul a fost realizat în felul următor(Figura 3.3):

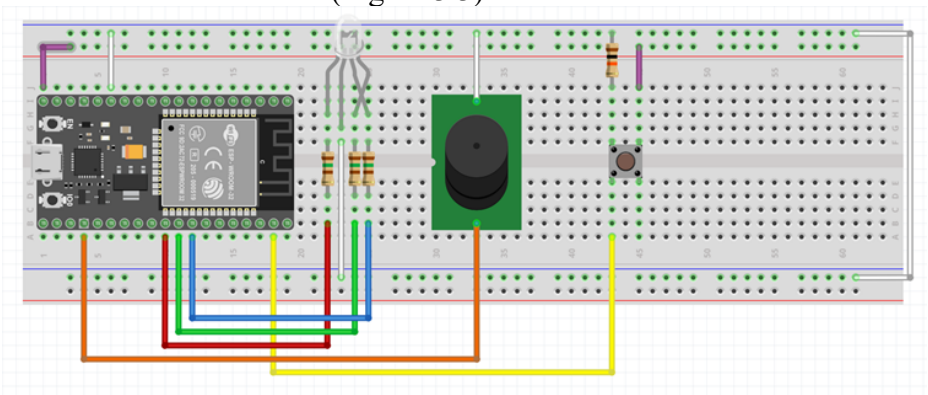

Figura 3.3: Implementarea componentei hardware

3.2. Probleme speciale/dificultăți întâmpinate și modalități de rezolvare

Inițial, aplicația mobilă a fost creată prin intermediul Expo, un lanț de unelte construite în jurul mediului React Native, deoarece instrumentele mediului sunt mult mai simplu și rapid de folosit în comparație cu cele ale React Native. Astfel, prima problemă întâmpinată a fost aceea că, deși uneltele sunt foarte potente, mediul nu oferă și posibilitatea de a folosi dependințe native. Pentru a rezolva această problemă, proiectul Expo a trebui migrat către un proiect React Native.

A doua problemă întâmpinată este aceea că, erorile React Native sunt în unele cazuri extrem de vagi. Spre exemplu, adăugarea unui pachet extern(prin intermediul managerului de pachete Node) fără a-i face legătura cu librăriile native poate cauza o eroare fără mesaj, stiva de apeluri fiind greu de citit/urmărit sau chiar neexistând în unele cazuri. Pentru a evita o astfel de situație, pachetele externe trebuie instalate și gestionate prin intermediul comenzilor specifice React Native. Astfel, pentru a adăuga o dependință externă se poate utiliza comanda „react-native install <nume-pachet-node>” sau comanda „npm i <nume-pachet-node>” urmată de comanda „react-native link <nume-pachet-node>”.

A treia problemă întâmpinată este aceea că, dezvoltarea aplicațiilor mobile necesită un spațiu de lucru generos, deoarece sunt necesare o multitudine de ferestre/unelte deschise pentru a putea avea un control asupra fluxului implementării. Pentru a putea rezolva această problemă este recomandată folosirea unui monitor suplimentar, dar și testarea funcționalităților prin intermediul unui dispozitiv mobil, ci nu printr-un emulator. Simultan, implementările pur JavaScript pot fi create/testate și înafara proiectului(în linia de comandă Node sau în cea a navigatorului Web), iar componentele React ce nu depind de librăriile native pot fi create/testate într-un proiect Expo separat sau prin intermediul Expo Snack.

A patra problemă întâmpinată a fost aceea că, la descărcarea proiectului din sistemul de versionare Git, au trebuit instalate toate dependințele mediului de lucru, apoi au trebuit compilate și legate(eng.: linking) dependințele și implementările făcute către platforma Android, proces ce poate fi foarte îndelungat. Totodată, acest fapt a fost întâmpinat și la actualizarea pachetelor Node.

```
BUILD SUCCESSFUL in 22m 11s
626 actionable tasks: 527 executed, 99 up-to-date
info Connecting to the development server...
debug Running command "adb -s 26e31714c2217ece reverse tcp:8081 tcp:8081"
info Starting the app on "26e31714c2217ece"...
debug Running command "adb -s 26e31714c2217ece shell am start -n com.oktan.keyfinder/com.oktan.keyfinder.Ma
inActivity"
Starting: Intent { cmp=com.oktan.keyfinder/.MainActivity }
```

Figura 3.4: Timpul de construire a aplicației

O altă problemă întâmpinată a fost aceea că, librăriile necesare folosirii modulului Bluetooth a microcontrolerului ESP32 nu puteau fi vizualizate de către mediul de dezvoltare integrat PlatformIO. Pentru a rezolva acest lucru, au trebuit referite căile către fiecare dependință a librăriei „est_bt.h” în configurarea proiectului PlatformIO.

3.3. Idei originale

Având în vedere subiectul temei propuse, originalitatea acesteia nu poate fi adusă în discuție datorită multitudinii de produse/aplicații existente pe piață ce servesc același scop. Ceea ce este implementat este similar ca funcționalitate cu variantele deja existente.

3.4. Comunicarea cu alte sisteme și stocarea informațiilor

Aplicația mobilă comunică cu microcontrolerul folosind undele radio prin intermediul standardului tehnologic Bluetooth Low Energy. Aplicația mobilă poate vizualiza dispozitivele-receptor din apropiere și poate trimite cereri către acestea. Pentru ca dispozitivele să poată comunica, aplicația mobilă va citi și scrie peste caracteristica profilului serviciului publicat de dispozitivului receptor.

Asocierea între dispozitive se face în felul următor:

1. Se realizează conexiunea între dispozitive;
2. Aplicația mobilă trimite un mesaj ce conține cheia de acces;
3. Programul pentru microcontroler verifică cheia primită și actualizează variabila de autorizare în funcție de validarea efectuată;
4. Aplicația mobilă face o cerere de citire a caracteristicii serviciului dispozitivului receptor;
5. Serverul, la momentul citirii caracteristicii returnează codul de acces sau mesajul „NO_DATA” în funcție de valoarea variabilei de autorizare;
6. Aplicația mobilă va salva codul de acces în spațiul de stocare al dispozitivului pentru a-l folosi pentru viitoare cereri fără a mai fi nevoie de autorizare;
7. Se realizează deconectarea dintre dispozitive.

Pentru a trimite semnalul de activare al funcției de localizare, aplicația mobilă va prelua din spațiul de stocare codul de acces al dispozitivului, va realiza conexiunea cu acesta și apoi va expedia codul alături de mesajul operațiunii de localizare(„FIND”). La final, conexiunea dintre dispozitive este întreruptă.

Pentru a contura comunicarea între dispozitivul mobil și microcontroler a fost creată următoarea diagramă de secvențe(Figura 3.5).

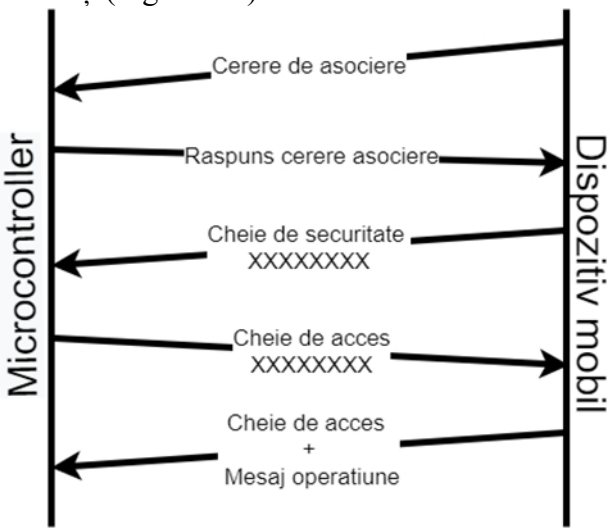

Figura 3.5: Diagrama de secvențe ce ilustrează comunicarea dintre dispozitive

3.5. Interfața cu utilizatorul

În Anexa 1. se regăsesc capturi de ecran ale vizualizărilor principale ale aplicației mobile:

- Ecranul de vizualizare a tuturor dispozitivelor din proximitate – Figura A.4;
- Formularul de asociere cu un dispozitiv – Figura A.5;
- Ecranul de vizualizare a dispozitivelor asociate – Figura A.6.

Se poate observa că în josul interfeței este situat un element ce conține două file (eng.: tabs). Acest element are rolul de a face posibilă navigarea de la un ecran la altul. Elementele din fila ce face referință către ecranul aflat în vizualizare vor fi colorate în culoarea principală a aplicației. Astfel, pentru ecranul de vizualizare a tuturor dispozitivelor din jur este colorată fila „Caută”, iar pentru celălalt ecran sunt colorate elementele filei „Asocieri”.

În partea de sus a interfeței este găsit un antet(eng.: header), conținutul acestuia fiind diferit în funcție de ecranul aflat în vizualizare.

Dacă serviciile de localizare sau componenta Bluetooth nu sunt pornite sau dacă nu se oferă acces la acestea, utilizatorul va fi anunțat și va trebui să ia măsuri pentru a putea folosi funcționalitățile ce le implică. Avertizarea utilizatorului se face prin casete dialog.

Butoanele ce au ca efect executarea unei acțiuni pe o durată îndelungată sunt înlocuite cu spinnere de-a lungul duratei de execuție.

Antetul ecranului de vizualizare a tuturor dispozitivelor din proximitate are în componență o etichetă cu numele aplicației și un buton-etichetă, „Caută”, care la apăsare afișează informațiile tuturor dispozitivelor receptor din proximitate. Informațiile despre fiecare dispozitiv aflat în proximitate sunt dispuse într-un element de tip dală(eng.: tile) ce este extinsă de-a lungul axei orizontale. Dalele sunt așezate în plan vertical într-o listă. O dală conține numele de advertising a unui dispozitiv, id-ul acestuia și un buton ce afișează formularul de asociere cu dispozitivul.

Formularul de asociere conține două casete text, una pentru a da un nume dispozitivului, iar cealaltă pentru a introduce cheia de securitate a acestuia. Pentru a realiza asocierea cu dispozitivul, ambele casete text trebuie completate. Cheia de securitate trebuie să aibă o lungime de strict 8 caractere, iar numele dispozitivului trebuie să conțină măcar un caracter. Totodată, formularul conține și un buton prin care este trimisă cererea de asociere către dispozitiv. Dacă asocierea a fost realizată cu succes, utilizatorul este redirecționat către ecranul de asocieri și trebuie să apese butonul-etichetă de reîmprospătare situat în antetul acestuia pentru a vedea noul dispozitiv adăugat.

Ecranul de vizualizare a dispozitivelor asociate are rolul de a oferi modalitatea de a contacta dispozitivele asociate. Informațiile despre fiecare dispozitiv asociat sunt dispuse într-o dală ce se poate extinde pe axa verticală. Dalele sunt situate într-o grilă ce permite așezarea a maxim două elemente pe axa orizontală. O dală conține numele dispozitivului introdus în formularul de asociere, id-ul acestuia, o pictogramă în formă de cheie, un buton ce șterge asocierea cu dispozitivul și un buton ce trimite semnalul de activare al rutinei de localizare a dispozitivului.

În ecranul de asocieri, indiferent dacă dispozitivele adăugate sunt în proximitate sau nu, sunt afișate mereu dalele cu informațiile acestora. În acest mod, se oferă posibilitatea de a gestiona un dispozitiv asociat chiar dacă nu este în jur. Pentru a diferenția dispozitivele asociate ce sunt în proximitate de cele care nu, dalele dispozitivelor ultim menționate sunt afișate inactiv.

Capitolul 4. Testarea aplicației și rezultate experimentale

Luând la cunoștință faptul că, aplicația mobilă este construită prin intermediul React Native, testarea acesteia poate fi făcută prin utilizarea uneltelor de testare software destinate limbajului JavaScript și prin testarea manuală. Principalele unelte software pentru testarea aplicațiilor React Native sunt Jest(pentru testarea unitară și pentru testarea integrării) și Appium/Detox(pentru testarea automată).

Testarea unităților are rolul de a verifica dacă modulele software ale unui program funcționează conform așteptărilor. Acestea sunt făcute de-a lungul perioadei de dezvoltare a programului, fiecare test fiind izolat de aplicație.

Testarea integrării are același rol și comportament cu testarea unităților, diferența dintre acestea fiind aceea că, testarea integrării are în vedere verificarea comportamentului modulelor ce grupează mai multe unități.

Testarea manuală verifică comportamentul unui program prin executarea unui set acțiuni de către un tester ce se pune în situația unui utilizator final. Acest procedeu este destul de costisitor în ceea ce prevede timpul acordat perioadei de testare, astfel au fost create unelte ce pot automatiza majoritatea interacțiunilor om-calculator. Modalitatea de automatizare a testării manuale este numită testare automată.

4.1. Punerea în funcțiune/lansarea aplicației

4.1.1. Lansarea aplicației mobile în modul de depănare

Având în vedere că, programul folosește controlerul Bluetooth al unui dispozitiv, utilizarea unui emulator nu este eficientă, așadar este necesară conectarea unui dispozitiv mobil cu sistem de operare Android la stația de lucru.

Trebuie pornit Metro Bundler, un program middleware ce are în vedere trimiterea pachetelor JavaScript către fiecare platformă pe care este lansat în execuție proiectul React Native. Pentru a porni middleware-ul se utilizează comanda „npx react-native start” într-o linie de comandă deschisă cu drepturi de administrator în rădăcina proiectului.

Pentru a lansa în execuție programul pe un dispozitiv Android, acesta trebuie conectat la stația de lucru prin intermediul USB, apoi trebuie rulată comanda „npx react-native run-android” într-o linie de comandă deschisă cu drepturi de administrator în rădăcina proiectului. Durata de compilare este suficient de îndelungată, mediul de lucru făcând cache asupra rezultatelor finale pentru a reduce timpii de compilare ai unei execuții viitoare.

4.1.2. Lansarea aplicației mobile în modul de lansare

Pentru a putea utiliza aplicația mobilă în mod independent de stația de lucru, este necesară compilarea acesteia în modul de lansare prin intermediul comenzii „npx react-native run-android --variant=release”. Această comandă are rolul de a face ca programul să nu mai depindă de middleware, acesta fiind folosit doar pentru depănare.

Luând în considerare paragraful anterior, se poate deduce că, modalitatea de instalare a aplicației este strict dependentă de stația de lucru. Pentru a evita acest lucru, aplicația poate fi împachetă într-un fișier de instalare „.apk” prin intermediul Android Studio. Simultan, pentru a putea instala programul prin intermediul rețelei Internet, acesta poate fi publicat și în aplicația Play Store.

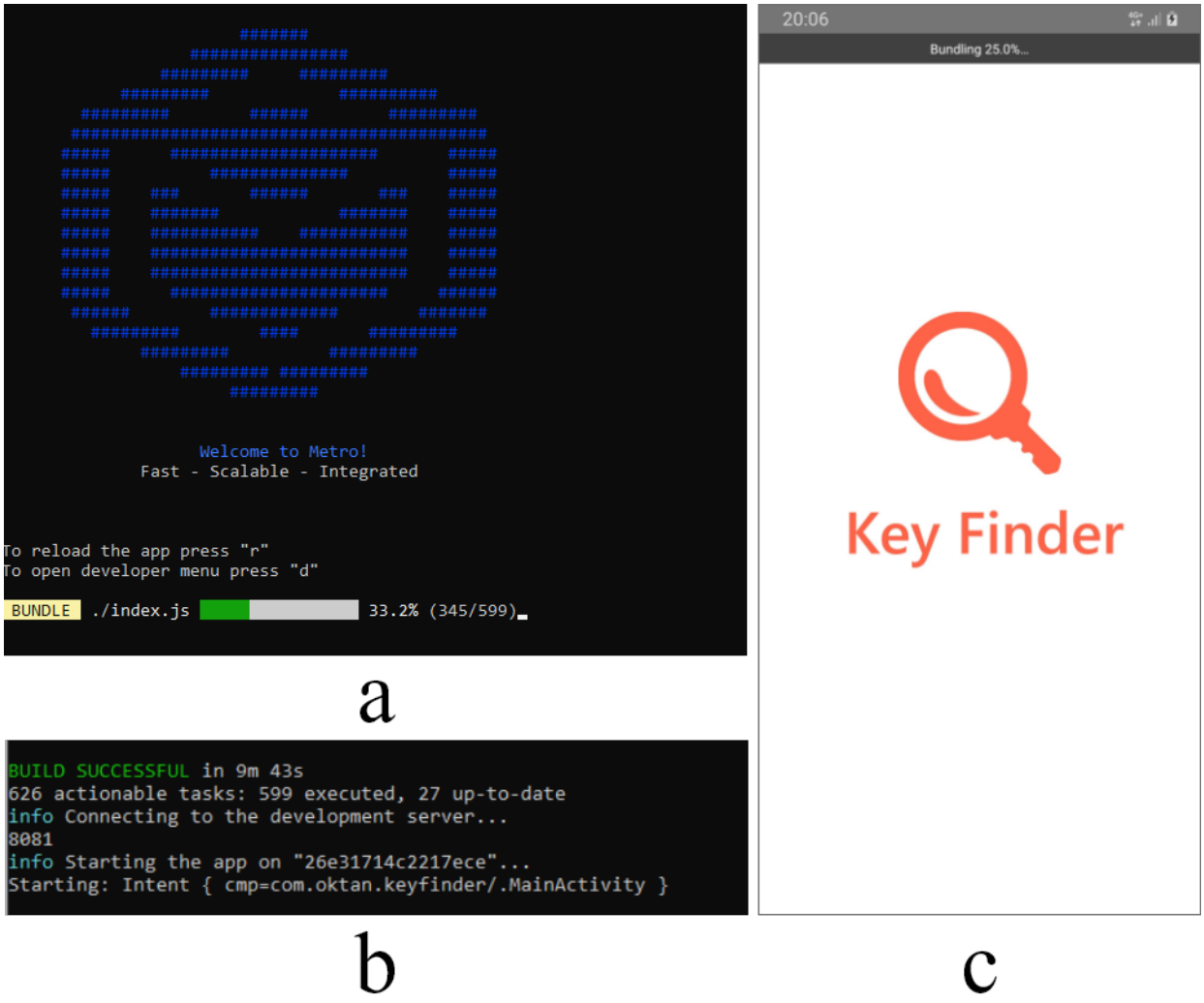

Figura 4.1: Lansarea în execuție a programului pentru dispozitivul mobil:
a – Trimiterea pachetelor JavaScript prin middleware-ul Metro Bundler;
b – Compilarea proiectului pentru platforma Android;
c – Aplicația mobilă – încărcarea pachetelor.

4.1.2.1. Lansarea programului pentru microcontroler

Pentru a putea lansa în execuție programul, trebuie conectate mai întâi perifericele la pinii de intrare/ieșire ai plăcii de dezvoltare. Apoi, aceasta trebuie conectată la stația de lucru, urmând ca proiectul să fie compilat și încărcat în memoria flash a cipului. Aceste operații se efectuează prin intermediul mediului de dezvoltare integrat PlatformIO. Odată ce programul este încărcat în microcontroler, acesta este pus în execuție în orice moment în care placa de dezvoltare este conectată la o sursă de curent.

4.2. Testarea sistemului

Ținând cont de tipurile de testare ce pot fi efectuate și de faptele enunțate în paragrafele cuprinse în capitolele 2 și 3, sistemul poate fi testat atât manual, cât și prin unelte software.

Pentru a testa dacă serverul Bluetooth al programului pentru microcontroler se comportă corespunzător, a fost utilizată aplicația mobilă „nRF Connect”. Astfel au fost executate următoarele teste manuale, prin care se verifică dacă:

- serverul Bluetooth Low Energy este public;
- se poate stabili o conexiune cu dispozitivul dacă acesta este/nu este deja conectat;
- pot fi trimise mesaje către dispozitiv;
- pot fi recepționate mesaje de la dispozitiv;
- dispozitivul autorizează cererea de asociere la trimiterea cheii de securitate;
- dispozitivul resetează variabila de autorizare la momentul trimiterii codului de acces și la momentul evenimentului de deconectare;
- dispozitivul inițiază rutina de localizare la momentul primirii codului de asociere concatenat cu mesajul specific acesteia.

Simultan, pentru a verifica dacă întreruperea produsă de evenimentul de apăsare al butonului conectat este tratată corespunzător, sunt necesare alte două teste manuale. Prin intermediul aplicației „nRF Connect" se stabilește o conexiune cu dispozitivul, iar apoi este apăsat butonul și se verifică dacă lumina emisă de diodă este de culoare albastră. Se realizează deconectarea de la dispozitiv și se verifică dacă la apăsarea butonului, lumina emisă de diodă este de culoare verde.

Comportamentul aplicației mobile poate fi verificat prin teste unitare, teste de integrare, teste automate și teste manuale. Testele unitare și de integrare pot fi aplicate pentru componentele din modulul „logic". Uneltele ce testează componente JavaScript sunt destinate în principal dezvoltării de aplicații mobile ce nu implică utilizarea modulelor hardware ale dispozitivului. Pentru a crea teste ce implică și utilizarea modulelor hardware, uneltele de testare trebuie foarte atent configurate. Procesul de creare al testelor este costisitor în ceea ce prevede timpul acordat și nivelul de cunoștințe necesar, deoarece fiecare unitate ce folosește un astfel de modul implică și simularea funcționalității modulului. În acest mod, aceste tipuri teste nu vor fi luate în considerare.

Pentru a testa automat aplicația mobilă, se poate folosi una din uneltele Appium/Detox. Ținând cont de faptul că aplicația mobilă nu este complexă în ceea ce prevede acțiunile pe care le poate face un utilizator, este mai rapidă și mai eficientă testarea manuală a acesteia.

Testarea funcționării corespunzătoare a componentelor ecranului de căutare a tuturor dispozitivelor din proximitate se face prin teste manuale. Testele verifică dacă:

1 La momentul apăsării pe butonul-etichetă de căutare situat în antet se întâmplă următoarele:
   - 1.a Butonul de căutare este înlocuit de un spinner;
   - 1.b Dalele afișate sunt șterse din vizualizare și se afișează un spinner în centrul ecranului;
   - 1.c La momentul finalizării acțiunii de căutare, dacă sunt găsite dispozitive în proximitate, sunt afișate dalele cu informațiile acestora, iar dacă nu, este afișat în centrul ecranului mesajul ce atenționează utilizatorul de acest lucru.
2 La apăsarea butonului de adăugare situat într-o dală, este afișat formularul de asociere;
3 Completarea câmpurilor formularului de asociere cu valori invalide produce mesaje de eroare;
4 Asocierea cu dispozitivul realizată cu succes redirecționează utilizatorul către ecranul dispozitivelor asociate și-l anunță de acest lucru;
5 Utilizatorul este avertizat în cazul în care asocierea a eșuat;

Testarea funcționării corespunzătoare a componentelor ecranului de vizualizare și căutare a dispozitivelor asociate se face prin teste manuale. Testele verifică dacă:

1 La momentul apăsării pe butonul-etichetă de reîmprospătare situat în antet se

întâmplă următoarele:

1.a Butonul de reîmprospătare este înlocuit de un spinner;

1.b Dalele sunt ascunse și se afișează un spinner în centrul ecranului;

1.c La momentul finalizării acțiunii de reîmprospătare, sunt reafișate dalele sub formă activă sau inactivă.

1.d Dacă sunt găsite dispozitive asociate în proximitate, dala asociată fiecăruia este marcată ca fiind activă, iar restul dalelor sunt marcate ca fiind inactive.

2 La apăsarea pe butonul de ștergere a unei asocieri este afișată o casetă dialog care cere utilizatorului confirmarea acțiunii comise;

3 Eliminarea unei asocieri are ca efect ștergerea permanentă a dalei din vizualizare;

4 Apăsarea pe butonul de căutare trimite semnalul de activare către dispozitivul ales.

4.3. Încărcarea procesorului și a memoriei

Ținând cont de faptul că aplicația mobilă a fost testată pe un dispozitiv cu AndroidX, dezvoltatorii sistemului de operare au eliminat posibilitatea de a vizualiza starea procesorului pentru această versiune. Astfel se va prezenta doar cantitatea de memorie utilizată de aplicația mobilă și de programul pentru microcontroler.

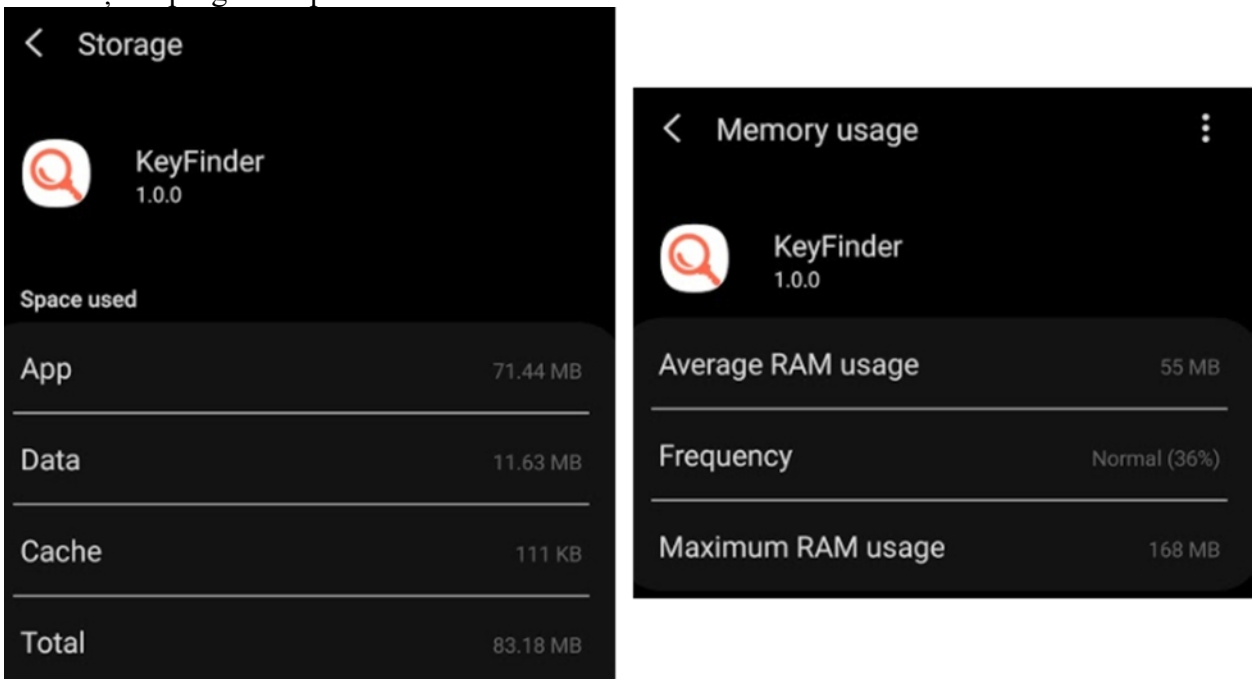

Figura 4.2: Memoria utilizată de aplicația mobilă

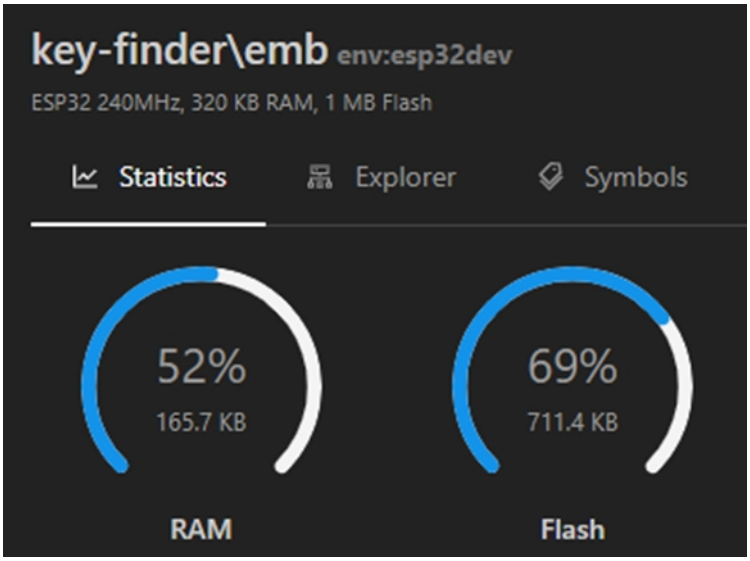

Figura 4.3: Memoria utilizată de programul microcontrolerului

4.4. Limitări în ceea ce privește transmisia datelor/comunicarea

Luând în considerare obiectivele temei propuse, dispozitivele comunică prin intermediul undelor radiu respectând standardul Bluetooth Low Energy. În acest mod, dispozitivul receptor a fost configurat așa încât dimensiunea maximă a unui pachet primit/transmis să nu depășească 23 octeți.

4.5. Utilizarea sistemului

Odată ce placa de dezvoltare este conectată la o sursă de curent, se poate verifica dacă aceasta este pregătită de conectare prin apăsarea butonului anexat. Prin aplicația mobilă se poate crea o asociere cu dispozitivul. Dacă asocierea a fost realizată cu succes, aplicația redirecționează utilizatorul către ecranul dispozitivelor asociate. Pentru a vizualiza dispozitivul nou adăugat, trebuie apăsat butonul de reîmprospătare. După reîmprospătare, se poate trimite semnalul de activare al rutinei de localizare a dispozitivului, dacă acesta este în apropiere.

Concluzii

Luând la cunoștință faptul că, aplicația mobilă a fost creată prin intermediul Expo și că este necesară utilizarea componentei Bluetooth a dispozitivului pe care este lansată în execuție aplicația, proiectul a fost migrat către React Native. Această decizie a fost luată deoarece, setul de unelte Expo nu include și o modalitate de a utiliza controlerul Bluetooth al unui dispozitiv mobil.

Aplicația mobilă a fost implementată conform cerințelor specificate, fiind respectat fluxul din diagrama de activități prezentată. Prin aceasta se pot căuta și contacta dispozitivele portabile implementate ce se află în proximitate cu ajutorul interfeței utilizator. Interfața cu utilizatorul răspunde corespunzător la acțiunile efectuate și afișează casete dialog în cazul în care sunt întâmpinate probleme la momentul contactării unui dispozitiv sau când nu se permite accesul asupra serviciilor de localizare/modulului Bluetooth. În casetele dialog, utilizatorul este rugat să ia măsuri în funcție de mesajul atenționării făcute.

Componenta hardware și programul pentru microcontroler au fost realizate conform obiectivelor prezentate. Altfel spus, s-a reușit crearea unui dispozitiv care poate gestiona cererile venite prin unde radio. Dispozitivul răspunde cererilor conform așteptărilor, astfel, la primirea cheii de securitate, trimite ca răspuns cheia de acces asupra rutinei de localizare, iar la momentul primirii mesajului de activare al rutinei, aceasta este inițiată.

Având în vedere contribuția personală asupra sistemului, autorul își asumă autenticitatea proiectării și implementării programelor, cu precizarea că, unica sursă de inspirație folosită a fost codul sursă al serverului Bluetooth pentru microcontroler. Cu toate că serverul Bluetooth cuprinde elemente din sursa oferită de dezvoltatori, nu există alte căi pentru a-l configura.

Sistemul poate fi asemănat ca funcționalitate cu restul proiectelor existente pe piață prin faptul că permite crearea unui reper asupra unui obiect de care este atașat un dispozitiv localizator. Totodată, modalitatea de contactare a dispozitivului este asemănătoare, aceasta fiind realizată prin intermediul standardului Bluetooth, operația de localizare putând fi activată cu ajutorul unei aplicații mobile.

Față de produsele existente pe piață, dispozitivul implementat nu are un sistem propriu de alimentare cu energie, acesta fiind dependent de conectarea prin USB la o sursă externă de alimentare. Concomitent, dispozitivul nu are proiectat/dezvoltat un prototip care să-i cuprindă elementele încorporate și care să-i ofere un aspect comparabil cu sistemele asemănătoare.

Trebuie luat în vedere că, pentru comparația făcută, sistemele similare cu implementarea discutată sunt dezvoltate de către companii de renume ce au echipe specializate pentru fiecare din domeniile cuprinse.

În viitor, se propune crearea unui prototip pentru dispozitiv, la care să se adauge un sistem de alimentare cu energie care să poate fi reîncărcat. Totodată, la nivel de funcționalitate se doresc a fi implementate în viitor: localizarea dispozitivului prin GPS, vizualizarea coordonatelor dispozitivului, localizarea dispozitivului mobil cu ajutorul dispozitivului portabil și extinderea modalității de comunicare folosind rețeaua locală fără fir prin intermediul tehnologiei Wi-Fi.

În final, se poate afirma, cu încredere, că lucrarea este una reușită, cu rezultate satisfăcătoare, luând în calcul experiența insuficientă a autorului și soluțiile propuse de acesta la problemele întâmpinate în timpul proiectării și implementării sistemului.

Bibliografie

[1] ESP-IDF Programming Guide [Online], Disponibil la adresa: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html, Accesat: 2021.

[2] ESP-IDF Programming Guide API Reference [Online], Disponibil la adresa: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/, Accesat 2021

[3] FreeRTOS [Online], Disponibil la adresa: https://www.freertos.org/about-RTOS.html, Accesat: 2021.

[4] The life and times of LED – a 100-year history [Online], Disponibil la adresa: https://web.archive.org/web/20120915034646/http://holly.orc.soton.ac.uk/fileadmin/downloads/100_years_of_optoelectronics__2_.pdf, Accesat: 2021.

[5] The History of Piezoelectricity [Online], Disponibil la adresa: https://piezo.com/pages/history-of-piezoelectricity, Accesat: 2021.

[6] Rachel Plotnick, Power Button: A history of Plkeasure, Panic and the Politics of Pushing, MIT Press, 2018

[7] NodeMCU-32S [Online], Disponibil la adresa: https://docs.platformio.org/en/latest/boards/espressif32/nodemcu-32s.html#frameworks, Accesat: 2021.

[8] Pull-up Resistors [Online], Disponibil la adresa: https://learn.sparkfun.com/tutorials/pull-up-resistors/all, Accesat: 2021.

[9] Understanding the Pull-up/Pull-down Resistors With Arduino [Online], Disponibil la adresa: https://www.instructables.com/Understanding-the-Pull-up-Resistor-With-Arduino/, Accesat: 2021.

[10] ESP-IDF Examples [Online], Disponibil la adresa: https://github.com/espressif/esp-idf/tree/master/examples, Accesat: 2021.

[11] Gatt Server Example Walkthrough [Online], Disponibil la adresa: https://github.com/espressif/esp-idf/blob/master/examples/bluetooth/bluedroid/ble/gatt_server/tutorial/Gatt_Server_Example_Walkthrough.md, Accesat: 2021.

[12] LEDC Example [Online], Disponibil la adresa: https://github.com/espressif/esp-idf/tree/cf457d4/examples/peripherals/ledc, Accesat: 2021.

[13] Example: GPIO [Online], Disponibil la adresa: https://github.com/espressif/esp-idf/tree/cf457d412a7748139b77c94e7debe72ee86199af/examples/peripherals/gpio/generic_gpio, Accesat: 2021.

[14] BLE UART demo [Online], Disponibil la adresa: https://github.com/pcbreflux/espressif/tree/master/esp32/app/ESP32_ble_UART, Accesat: 2021.

[15] Android Bluetooth Low Energy Communication – Simplified [Online], Disponibil la adresa: https://medium.com/android-news/android-bluetooth-low-energy-communication-simplified-d4fc67d3d26e, Accesat: 2021.

[16] Android Technology [Online], Disponibil la adresa: https://developer.android.com/guide, Accesat: 2021.

[17] Flutter [Online], Disponibil la adresa: https://flutter.dev, Accesat: 2021.

[18] Xamarin [Online], Disponibil la adresa: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin, Accesat: 2021.

[19] Ionic Framework [Online], Disponibil la adresa: https://ionicframework.com, Accesat: 2021.

[20] Apache Cordova [Online], Disponibil la adresa: https://cordova.apache.org, Accesat: 2021.

[21] React [Online], Disponibil la adresa: https://reactjs.org, Accesat: 2020.

[22] Expo [Online], Disponibil la adresa: https://docs.expo.io/, Accesat: 2020.

[23] React Native [Online], Disponibil la adresa: https://reactnative.dev, Accesat: 2021.

[24] JavaScript and HTML DOM Reference [Online], Disponibil la adresa: https://www.w3schools.com/jsref/default.asp, Accesat: 2019.

[25] Node.js [Online], Disponibil la adresa: https://nodejs.org/en/, Accesat: 2019.

[26] React Native BLE PLX [Online], Disponibil la adresa: https://github.com/dotintent/react-native-ble-plx, Accesat: 2021.

[27] Building Standalone Apps [Online], Disponibil la adresa: https://docs.expo.io/distribution/building-standalone-apps/, Accesat: 2021.

[28] Build a Bluetooth App with React Native and Expo[Online], Disponibil la adresa: https://blog.expo.io/so-you-want-to-build-a-bluetooth-app-with-react-native-and-expo-6ea6a31a151d, Accesat: 2021.

[29] Orbit [Online], Disponibil la adresa: https://findorbit.com, Accesat: 2020.

[30] chipolo [Online], Disponibil la adresa: chipolo.net/, Accesat: 2020.

[31] tile [Online], Disponibil la adresa: https://www.thetileapp.com, Accesat: 2020.

[32] Samsung Galaxy SmartTag [Online], Disponibil la adresa: https://www.samsung.com/ro/mobile-accessories/galaxy-smarttag-black-ei-t5300bbegeu/?cid=ro_pd_ppc_google_accessories-mobile-im_ongoing_smart-tag-corporate_text_1x1-none-search-product-20210310-traffic_ads&gclid=CjwKCAjwieuGBhAsEiwA1Ly_nUN1Us71qbkPmAH4937nI76R4Sj_Yg52cjzi6oDtETO908lZp9ykthoCKakQAvD_BwE, Accesat: 2020.

[33] Samsung Smart Things [Online], Disponibil la adresa: https://www.samsung.com/ro/apps/smartthings/, Accesat: 2020.

[34] Apple AirTag [Online], Disponibil la adresa: https://www.apple.com/airtag/, Accesat: 2020.

Anexe.

Anexa 1. Diagrame/scheme logice

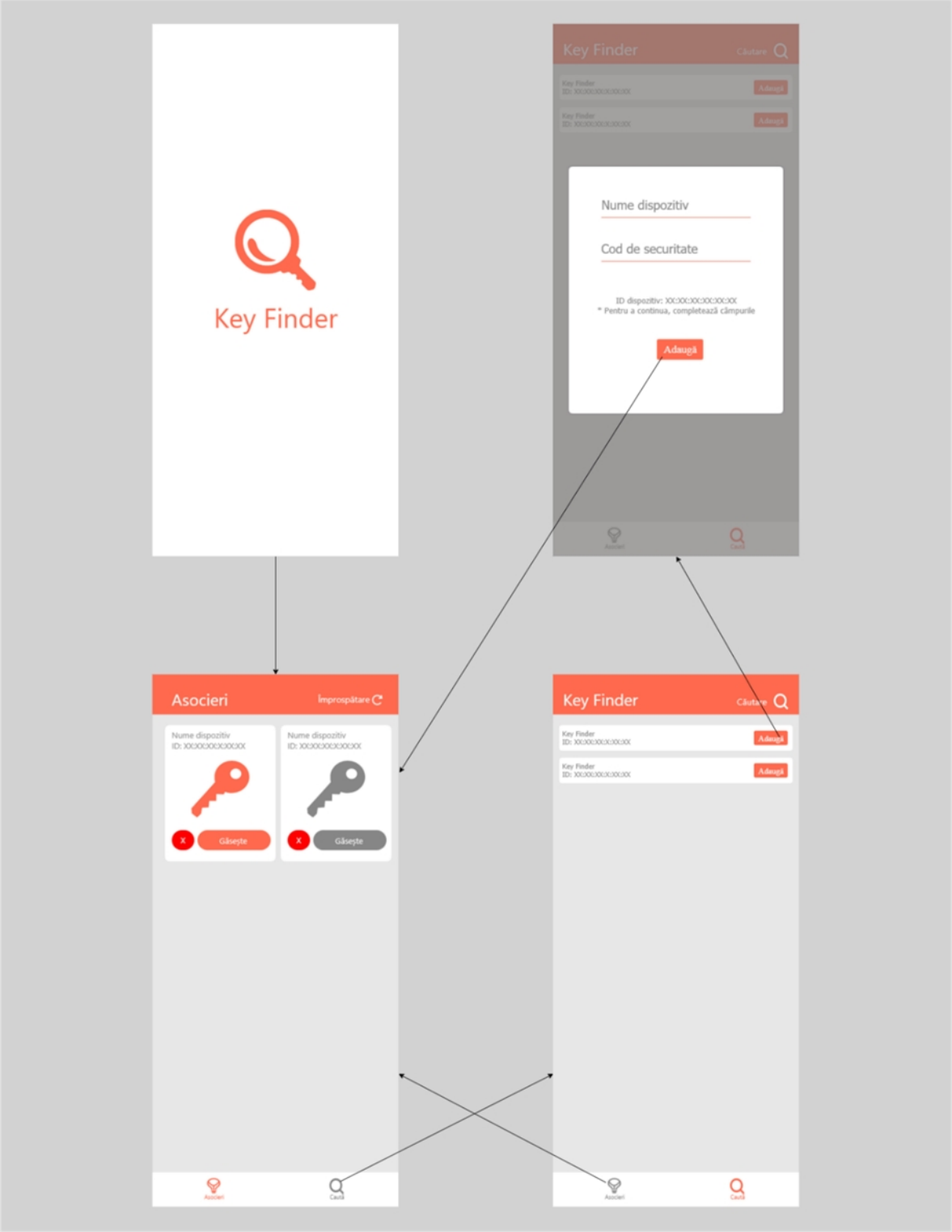

Figura A.1. Diagrama de activități a aplicației mobile

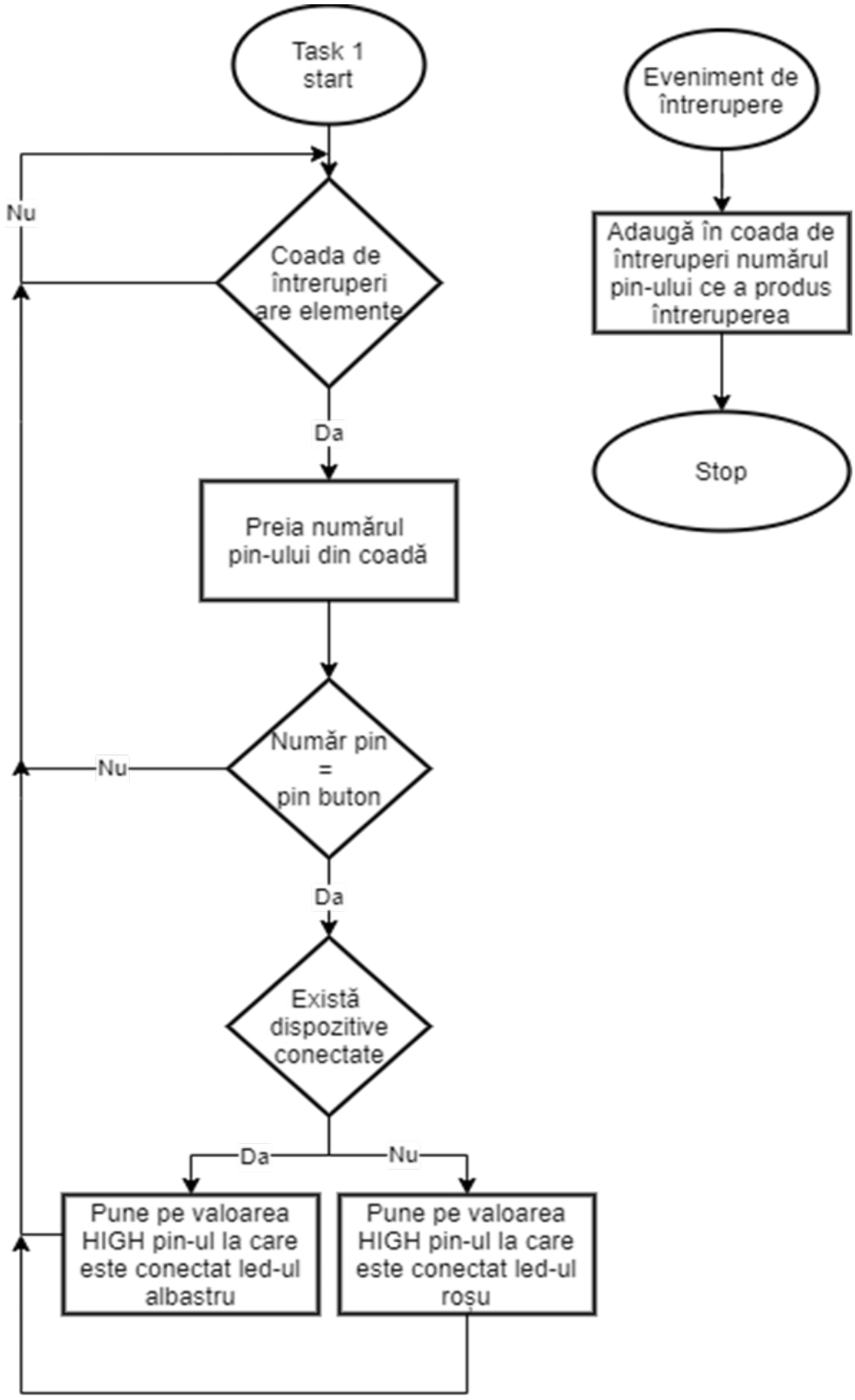

Figura A.2. Schema logică a rutinei de întreruperi

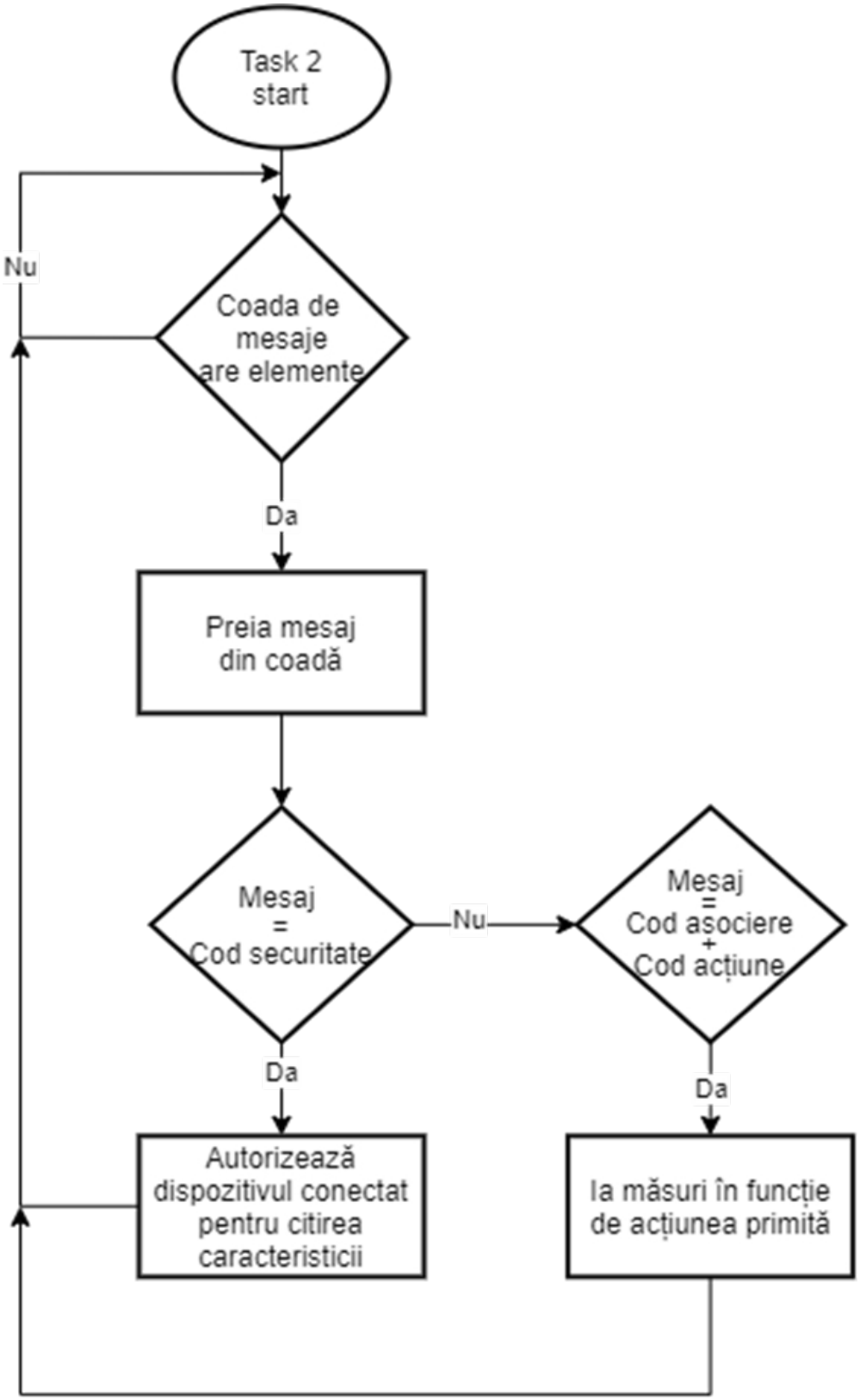

Figura A.3. Schema logică a task-ului ce interpretează mesajele aflate în coadă

Anexa 2. Interfața cu utilizatorul – vizualizări principale

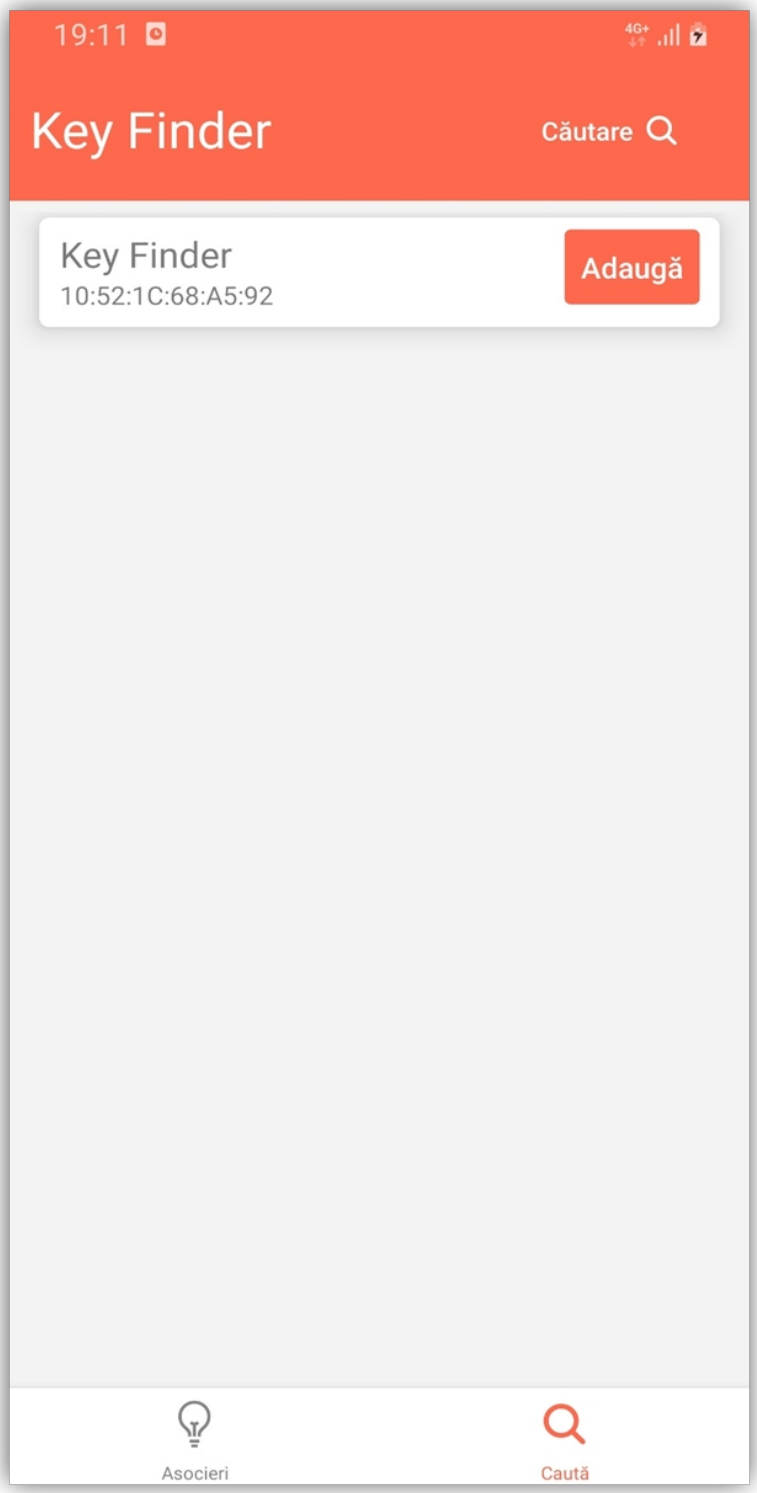

Figura A.4. Ecranul de vizualizare a tuturor dispozitivelor din proximitate

Figura A.5. Formularul de asociere cu un dispozitiv

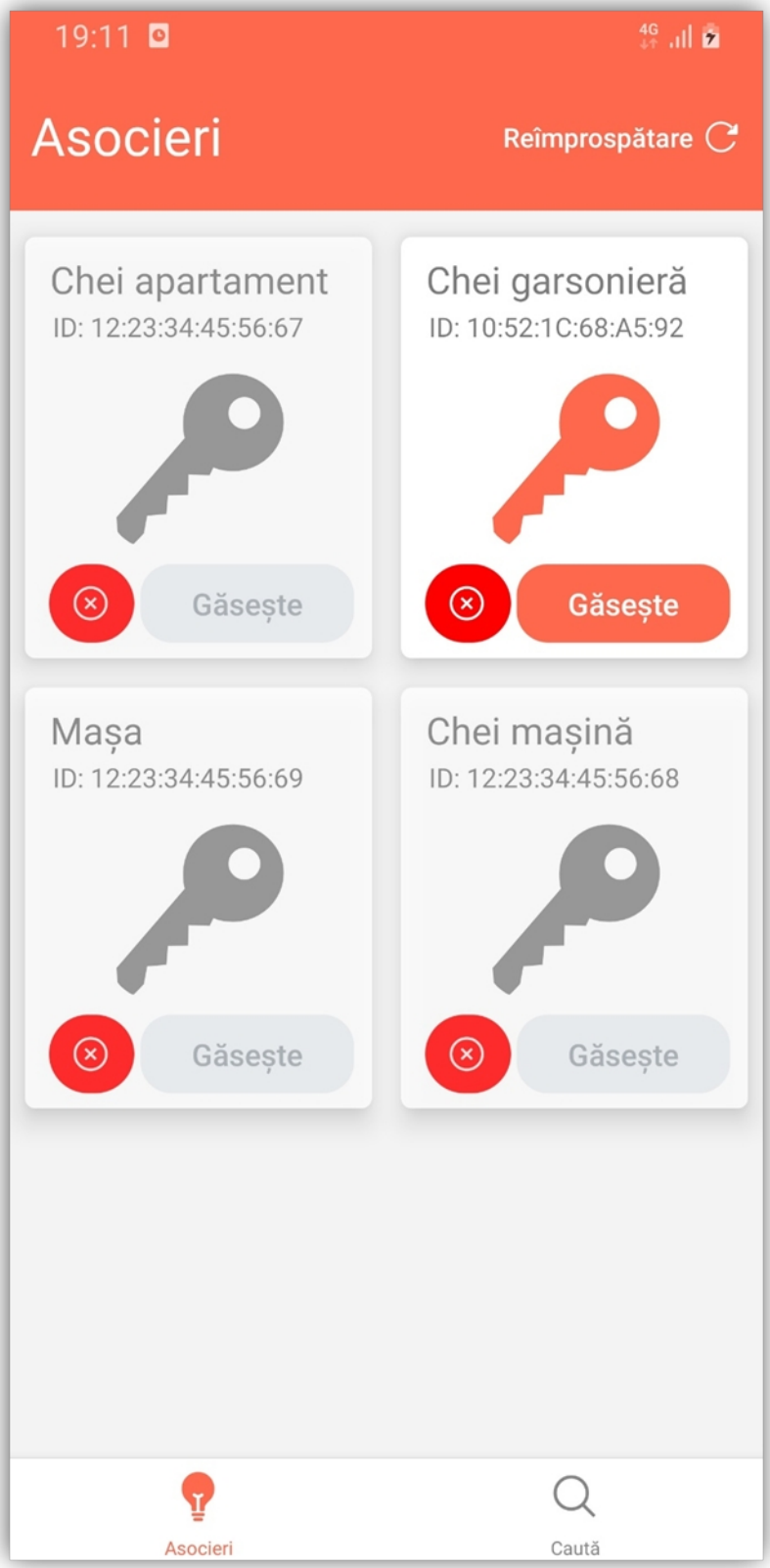

Figura A.6. Ecranul de vizualizare a dispozitivelor asociate

Anexa 3. Prototipul dispozitivului portabil

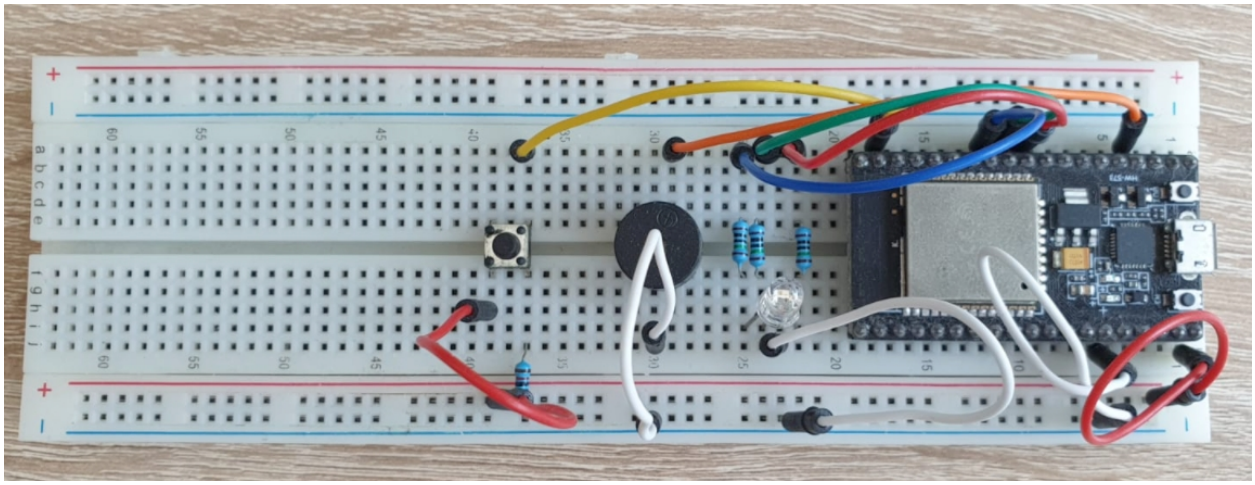

Figura A.7: Prototipul dispozitivului portabil

Anexa 4. Cod sursă – clasa BtManager

```
var base64 = require('base-64');
var utf8 = require('utf8');
class BtManager {
    constructor() {
        this.data = require("../../assets/data/devices-data.json");
        this.manager = new BleManager();
        this.blEnabled = true;
        this.manager.onStateChange(state => { this.blEnabled = state ==
'PoweredOn' ? true : false; });
    }

    searchForDevices() {
        return new Promise((resolve, reject) => {
            let devices = [];
            this.manager.startDeviceScan(this.data.adv_service_uuids,
null, (error, device) => {
                if (error) {
                    if (error.message == 'BluetoothLE is powered off')
                        reject(new BluetoothError('Bluetooth-ul nu este
pornit'));
                    else if (error.message == 'Device is not authorized
to use BluetoothLE')
                        reject(new LocationError('Nu s-a permis accesul
la locatie'));
                    else if (error.message == 'Location services are
disabled')
                        reject(new LocationServicesError('Serviciile de
localizare nu sunt pornite'));
                    reject(error);
```

```
                    }
                    if (!devices.find(d => d.id == device.id))
                        devices.push(device);
                });
                setTimeout(() => {
                    this.manager.stopDeviceScan();
                    resolve(devices);
                }, 3000)
            });
        }

        addDevice(deviceID, securityCode) {
            if (this.blEnabled) {
                return this.manager.isDeviceConnected(deviceID)
                    .then((isConnected) => {
                        if (!isConnected)
                            return this.manager.connectToDevice(deviceID)
                    })
                    .then(() => {
                        return
    this.manager.discoverAllServicesAndCharacteristicsForDevice(deviceID)
                    })
                    .then(() => {
                        return
    this.manager.writeCharacteristicWithoutResponseForDevice(
                            deviceID,
                            this.data.gatt_service[0].uuid,
                            this.data.gatt_service[0].characteristic_uuid,
                            base64.encode(utf8.encode(securityCode)));
                    })
                    .then(() => {
                        return this.manager.readCharacteristicForDevice(
                            deviceID,
                            this.data.gatt_service[0].uuid,
                            this.data.gatt_service[0].characteristic_uuid
                        );
                    })
                    .then(char => {
                        this.manager.cancelDeviceConnection(deviceID)
                            .catch(e => {
                                console.log('Eroare la deconectarea cu
    dispozitivul');
                                console.error(e);
                            });
```

```
                        return base64.decode(char.value);
                    })
                    .catch((e) => {
                        this.manager.isDeviceConnected(deviceID)
                            .then(isConnected => {
                                if (isConnected)
this.manager.cancelDeviceConnection(deviceID);
                            });
                        this.throwErrorByType(e);
                        throw new DeviceNotInRangeError('Dispozitivul nu
poate fi contactat.');
                    });
            }
            else throw new BluetoothError('Bluetooth-ul nu este pornit');
        }

        findDevice(deviceID, accessCode) {
            var message = accessCode.concat(this.data.write_message.FIND);
            if (this.blEnabled) {
                return this.manager.isDeviceConnected(deviceID)
                    .then(isConnected => {
                        if (!isConnected)
                            return this.manager.connectToDevice(deviceID)
                    })
                    .then(() => {
                        return
this.manager.discoverAllServicesAndCharacteristicsForDevice(deviceID);
                    })
                    .then(() => {
                        return
this.manager.writeCharacteristicWithoutResponseForDevice(
                            deviceID,
                            this.data.gatt_service[0].uuid,
                            this.data.gatt_service[0].characteristic_uuid,
                            base64.encode(utf8.encode(message)))
                    })
                    .then(char => {
                        this.manager.cancelDeviceConnection(deviceID)
                            .catch(e => {
                                console.log('Eroare la deconectarea cu
dispozitivul');
                                console.error(e);
                            });
                        return { success: true };
```

```
                    })
                    .catch((e) => {
                        this.manager.isDeviceConnected(deviceID)
                            .then(isConnected => {
                                if (isConnected)

this.manager.cancelDeviceConnection(deviceID);
                            });
                        this.throwErrorByType(e.message);
                        throw new DeviceNotInRangeError('Dispozitivul nu
poate fi contactat.');
                    });
            }
            else throw new BluetoothError('Bluetooth-ul nu este pornit');
        }

        throwErrorByType(error) {
            if (error.message == 'BluetoothLE is powered off')
                throw new BluetoothError('Bluetooth-ul nu este pornit');
            else if (error.message == 'Device is not authorized to use
BluetoothLE')
                throw new LocationError('Nu s-a permis accesul la locatie');
            else if (error.message == 'Location services are disabled')
                throw new LocationServicesError('Serviciile de localizare nu
sunt pornite');
        }
    }
```

Anexa 5. Cod sursă – clasa KfAppError

```
class KfAppError extends Error {
  constructor(message) {
    super(message);
    this.name = "KfAppError";
  }
}
class BluetoothError extends KfAppError {
  constructor(message) {
    super(message);
    this.name = "BluetoothError";
  }
}
class LocationError extends KfAppError {
  constructor(message) {
    super(message);
```

```
        this.name = "LocationError";
      }
    }
    class LocationServicesError extends KfAppError {
      constructor(message) {
        super(message);
        this.name = "LocationServicesError";
      }
    }
    class DeviceNotInRangeError extends KfAppError {
      constructor(message) {
        super(message);
        this.name = "DeviceNotInRangeError";
      }
    }
    export {
      KfAppError,
      BluetoothError,
      LocationError,
      LocationServicesError,
      DeviceNotInRangeError,
    };
```

Anexa 6. Cod sursă – clasa StorageManager

```
    class StorageManager {
        async store(key, storeDeviceDataModel) {
            try {
                await AsyncStorage.setItem(key,
    JSON.stringify(storeDeviceDataModel))
            }
            catch (e) {
                console.log(e);
            }
        }

        async retrieve(key) {
            try {
                return await AsyncStorage.getItem(key);
            }
            catch (e) {
                console.error(e);
            }
        }
```

```
        async remove(key) {
            try {
                return await AsyncStorage.removeItem(key);
            }
            catch (e) {
                console.error(e);
            }
        }

        async getAllDevices() {
            try {
                const keys = await AsyncStorage.getAllKeys();
                let devices = [];
                for (let i = 0; i < keys.length; i++) {
                    const d = await this.retrieve(keys[i]);

                    const obj = JSON.parse(d);
                    const device =
    Object.create(StoreDeviceDataModel.prototype,
    Object.getOwnPropertyDescriptors(obj));

                    devices.push(device);
                }
                return devices;
            }
            catch (e) {
                console.error(e);
            }
        }

        async cleanup() {
            try {
                await AsyncStorage.clear();
            }
            catch (e) {
                console.error(e);
            }
        }

    }

    const store = new StorageManager();
    export default store;
```

Anexa 7. Cod sursă – clasa StoreDeviceDataModel

```
export default class StoreDeviceDataModel {
    constructor(uuid, deviceName, accessCode) {
        this.uuid = uuid;
        this.deviceName = deviceName;
        this.accessCode = accessCode;
    }
}
```

Anexa 8. Cod sursă – clasa AppOverlay

```
export default class AppOverlay extends React.Component {
    constructor(props) {
        super(props);
    }

    render() {
        return (
            <Overlay
                backdropStyle={{ backgroundColor:
'rgba(125,125,125,0.6)' }}
                animationType='fade'
                transparent={true}
                statusBarTranslucent={true}
                isVisible={this.props.isVisible}
                onBackdropPress={this.props.onBackdropPress}>
                <View style={styles.container}>
                    <Input
                        value={this.props.deviceNameValue}
                        onChangeText={this.props.onDeviceNameChange}
                        errorMessage={this.props.deviceNameErrorMessage}

                        inputStyle={styles.inputStyle}
                        inputContainerStyle={styles.inputContainerStyle}
                        placeholder='Nume discpozitiv'
                    />
                    <Input
                        value={this.props.securityCodeValue}
                        onChangeText={this.props.onSecurityCodeChange}

errorMessage={this.props.securityCodeErrorMessage}

                        inputStyle={styles.inputStyle}
                        inputContainerStyle={styles.inputContainerStyle}
                        placeholder='Cod de securitate'
```

```
                        />
                        <View style={styles.formSubmit}>
                            <Text>ID dispozitiv:
{this.props.deviceId}</Text>
                            <Text style={{ color:
assets.color.additional.inactive }}> *Pentru a continua, completează
câmpurile.</Text>
                            <Button
                                onPress={this.props.onAddPress}
                                title='Adaugă'
                                loading={this.props.isLoading}
                                buttonStyle={styles.buttonStyle} />
                        </View>
                    </View>
                </Overlay>
            );
        }
    }

    const styles = StyleSheet.create({
        container: {
            borderRadius: 50,
            paddingTop: 10,
            minWidth: 250,
            minHeight: 250,
            width: 300,
            height: '40%',
        },
        inputStyle: {
            color: assets.color.additional.inactive,
        },
        inputContainerStyle: {
            borderBottomColor: assets.color.primary.basic,
        },
        buttonStyle: {
            backgroundColor: assets.color.primary.basic,
            width: 100,
            marginTop: 25,
        },
        formSubmit: {
            display: 'flex',
            alignItems: 'center',
        },
    });
```

Anexa 9. Cod sursă – clasa DevicesScreen

```
class DevicesScreen extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            isLoading: false,
            devices: [],
        }
    }

    componentDidMount() {
        store.getAllDevices()
            .then((devices) => {
                if (devices.length > 0)
                    this.refreshDevices()
            });
    }

    onDeletePress = (deviceId, deviceName) => {
        Alert.alert(
            "Atenție!!!",
            `Ești pe cale de a șterge asocierea cu dispozitivul '$
{deviceName}'. Ești sigur că vrei să continui această acțiune?`,
            [
                {
                    text: "Anulează",
                    onPress: () => { },
                    style: "cancel"
                },
                {
                    text: "Da",
                    onPress: async () => {
                        await store.remove(deviceId);
                        const filteredDevices =
this.state.devices.filter((d) => {
                            return d.uuid != deviceId;
                        });
                        this.setState({ devices: [...filteredDevices] },
() => { ToastAndroid.show(`${deviceName} a fost șters.`,
ToastAndroid.SHORT); });
                    }
                }
            ]);
    }
```

```
        onFindPress = (deviceId, accessCode) => {
            this.state.devices.forEach(d => {
                if (d.uuid == deviceId) {
                    d.loading = true;
                }
            })
            this.setState({ devices: [...this.state.devices] }, async () =>
    {
                try {
                    await BtManager.findDevice(deviceId, accessCode);
                    this.state.devices.forEach(d => {
                        if (d.uuid == deviceId) {
                            d.loading = false;
                        }
                    })
                    setTimeout(() => {
                        this.setState({ devices: [...this.state.devices] });
                    }, 4000)
                }
                catch (e) {
                    this.state.devices.forEach(d => {
                        if (d.uuid == deviceId) {
                            d.loading = false;
                        }
                    })
                    this.setState({ devices: [...this.state.devices] });
                    if (!ErrorAlert.handleAllErrors(e)) {
                        console.log('Eroare la localizarea
    dispozitivlului.');
                        console.error(e);
                    }
                }
            });

        }

        refreshDevices = () => {
            store.getAllDevices().then((devices) => {
                if (devices.length == 0) {
                    Alert.alert(
                        "Nu s-au găsit asocieri.",
                        `Apasă pe tab-ul 'Caută' pentru a descoperi noi
    dispozitive, apoi asociază telefonul cu unul sau mai multe din
    acestea.`,
```

```
                        [{ text: "Am înțeles", onPress: () => { } }]);
                    return;
                }
                devices.forEach(d => {
                    d['onFindPress'] = this.onFindPress.bind(this, d.uuid,
d.accessCode);
                    d['onDeletePress'] = this.onDeletePress.bind(this,
d.uuid, d.deviceName);
                    d['color'] = assets.color.additional.inactive;
                    d['active'] = false;
                    d['loading'] = false;
                });
                this.setState({ isLoading: true }, async () => {
                    let hasFound = false;
                    let hasErrors = false;
                    try {
                        const scannedDevices = await
BtManager.searchForDevices();
                        devices.forEach(d => {
                            const found = scannedDevices.filter(sd => sd.id
== d.uuid)
                            if (found.length > 0) {
                                d['active'] = true;
                                hasFound = true;
                            }
                            else d['active'] = false;
                        });
                    }
                    catch (e) {
                        hasErrors = true;
                        if (!ErrorAlert.handleAllErrors(e)) {
                            console.log('Eroare la căutarea
dispozitivelor');
                            console.error(e);
                        }
                    }
                    finally {
                        this.setState({ isLoading: false, devices:
[...devices] }, () => {
                            if (!hasErrors) {
                                if (hasFound)
                                    ToastAndroid.show("Au fost găsite
dispozitive în apropiere.", ToastAndroid.SHORT);
                                else
                                    ToastAndroid.show("Nu a fost găsit
```

```
        niciun dispozitiv asociat.", ToastAndroid.SHORT);
                                }
                            });
                        }
                    });
                });
            }

            render() {
                return (
                    <View>
                        <View>
                            <Header
                                title='Asocieri'
                                buttonAction={this.refreshDevices}
                                isLoading={this.state.isLoading}
                                buttonTitle="Reîmprospătare"
                            ></Header>
                            {
                                (this.state.devices.length > 0 && !
        this.state.isLoading) &&
                                <ScrollView style={{ marginBottom: 95,
        paddingTop: 6 }}>
                                    <View
        style={styles.associatedDevicesContainer}>
                                        {this.state.devices.length != 0 &&
        <TileList devices={this.state.devices} />}
                                    </View>
                                </ScrollView>
                            }
                            {
                                (this.state.devices.length > 0 &&
        this.state.isLoading) &&
                                <View style={styles.searchContainer}>
                                    <SearchView
                                        title="Se caută dispozitivele asociate"
                                        style={styles.blank} />
                                </View>
                            }
                            {
                                this.state.devices.length == 0 &&
                                < BlankView
                                    title="Nu există asocieri cu dispozitivele
        KeyFinder. Apasă pe tab-ul caută pentru a adăuga unul."
                                    style={styles.blank} />
```

```
                        }
                    </View>
                </View>
            );
        }
    }

    export default function (props) {
        const route = useRoute();
        return <DevicesScreen {...props} route={route} />
    }

    const styles = StyleSheet.create({
        associatedDevicesContainer: {
            width: '96%',
            marginHorizontal: '2%',
            marginBottom: 100,
        },
        blank: {
            marginVertical: 80,
        },
        searchContainer: {
            marginVertical: "50%",
        }
    });
```

Anexa 10. Cod sursă – clasa SearchScreen

```
    class SearchScreen extends React.Component {
        constructor(props) {
            super(props);
            this.state = {
                overlayVisible: false,
                isLoading: false,
                isCheckingSecurityCode: false,
                devices: [],
                deviceId: null,
                deviceName: '',
                securityCode: '',
                securityCodeErrorMessage: '',
                deviceNameErrorMessage: '',
            }
        }

        componentDidMount() {
```

```
            this.onSearchButtonPress();
        }

        onDeviceNameChange = (value) => { this.setState({ deviceName: value
    }) }

        onSecurityCodeChange = (value) => { this.setState({ securityCode:
    value }) }

        onToggleOverlay = () => {
            this.setState({
                deviceName: '',
                securityCode: '',
                securityCodeErrorMessage: '',
                deviceNameErrorMessage: '',
                overlayVisible: !this.state.overlayVisible
            })
        }

        onSearchButtonPress = () => {
            this.setState({ isLoading: true }, () => {
                BtManager.searchForDevices()
                    .then((d) => {
                        this.setState({ devices: [...d] }, () => {
                            if (this.state.devices.length > 0)
                                ToastAndroid.show("Au fost găsite
    dispozitive în apropiere.", ToastAndroid.SHORT);
                            else
                                ToastAndroid.show("Nu a fost găsit niciun
    dispozitiv.", ToastAndroid.SHORT);
                        });
                    })
                    .catch((e) => {
                        if (!ErrorAlert.handleAllErrors(e)) {
                            console.log('Eroare la căutarea
    dispozitivelor');
                            console.error(e);
                        }
                    })
                    .finally(() => {
                        this.setState({ isLoading: false });
                    })
            });
        }
```

```
        onAddDeviceButtonPress = (deviceId) => {
            this.setState({ deviceId: deviceId, overlayVisible: true });
        }

        onSubmitAddDevice = () => {
            let hasErr = false;
            if (this.state.deviceName == '') {
                hasErr = true;
                this.setState({ deviceNameErrorMessage: 'Numele
    dispozitivului este obligatoriu.' });
            }
            else this.setState({ deviceNameErrorMessage: '' });

            if (this.state.securityCode == '') {
                hasErr = true;
                this.setState({ securityCodeErrorMessage: 'Codul de
    securitate este obligatoriu.' });
            }
            else if (this.state.securityCode.length != 8) {
                hasErr = true;
                this.setState({ securityCodeErrorMessage: 'Lungimea codului
    de securitate trebuie să fie de 8 caractere.' });
            }
            else this.setState({ securityCodeErrorMessage: '' });

            if (!hasErr) {
                this.setState({ isCheckingSecurityCode: true }, async () =>
    {
                    try {
                        const accessCode = await
    BtManager.addDevice(this.state.deviceId, this.state.securityCode);
                        if (accessCode != "NO DATA") {
                            const storeModel = new
    StoreDeviceDataModel(this.state.deviceId, this.state.deviceName,
    accessCode);
                            await store.store(this.state.deviceId,
    storeModel);

                            this.setState({
                                overlayVisible: false,
                                deviceName: '',
                                securityCode: '',
                            }, () => {
                                ToastAndroid.show('Adăugat. Apasă pe
    împrospătare pentru a vedea noul dispozitiv.', ToastAndroid.LONG);
                                this.props.navigation.navigate('Asocieri');
```

```
                        });

                    }
                    else ToastAndroid.show('Codul de securitate este
incorect.', ToastAndroid.SHORT);
                }
                catch (e) {
                    this.setState({
                        overlayVisible: false,
                        deviceName: '',
                        securityCode: '',
                    }, () => {
                        if (!ErrorAlert.handleAllErrors(e)) {
                            console.log('Eroare la căutarea
dispozitivelor');
                            console.error(e);
                        }
                    });
                }
                finally {
                    this.setState({
                        deviceNameErrorMessage: '',
                        securityCodeErrorMessage: '',
                        isCheckingSecurityCode: false
                    });
                }
            });
        }
    }

    updateDevices = () => {
        let devices = [];
        this.state.devices.forEach(d => {
            devices.push({
                title: d.localName,
                deviceId: d.id,
                key: d.id,
                onPress: this.onAddDeviceButtonPress.bind(this, d.id)
            });
        });
        return devices
    }

    render() {
```

```
            return (
                <View>
                    <Header
                        title='Key Finder'
                        buttonAction={this.onSearchButtonPress}
                        isLoading={this.state.isLoading}
                        buttonTitle="Căutare" />
                    {
                        (this.state.devices.length > 0 && !
    this.state.isLoading) &&
                        <View>
                            <ScrollView style={{ marginBottom: 95,
    paddingTop: 6 }}>
                                <View style={styles.foundDevicesContainer}>
                                    <TileList devices={this.updateDevices()}
    />
                                </View>
                            </ScrollView>

                            <Overlay
                                isVisible={this.state.overlayVisible}
                                deviceId={this.state.deviceId}

                                deviceNameValue={this.state.deviceName}
                                onDeviceNameChange={(x) =>
    this.onDeviceNameChange(x)}

                                securityCodeValue={this.state.securityCode}

    onSecurityCodeChange={this.onSecurityCodeChange}


    deviceNameErrorMessage={this.state.deviceNameErrorMessage}

    securityCodeErrorMessage={this.state.securityCodeErrorMessage}


    isLoading={this.state.isCheckingSecurityCode}


    onAddPress={this.onSubmitAddDevice.bind(this)}
                                onBackdropPress={this.onToggleOverlay}
                            />
                        </View>
                    }
                    {
                        (this.state.devices.length > 0 &&
```

```
        this.state.isLoading) &&
                        <View style={styles.searchContainer}>
                            <SearchView
                                title="Se caută dispozitivele din
        proximitate"
                                style={styles.blank} />
                        </View>
                    }
                    {
                        this.state.devices.length == 0 &&
                        <BlankView
                            title="Nu s-au găsit dispozitive KeyFinder în
        apropiere. Apasă pe butonul de căutare situat mai sus pentru a verifica
        din nou."
                            style={styles.blank} />
                    }
                </View>
            );
        }
    }

    export default function (props) {
        const route = useRoute();
        return <SearchScreen {...props} route={route} />
    }

    const styles = StyleSheet.create({
        foundDevicesContainer: {
            width: '92%',
            marginHorizontal: '4%',
            marginBottom: 100,
        },
        blank: {
            marginVertical: 80
        },
        searchContainer: {
            marginVertical: "50%",
        },
    });
```

Anexa 11. Cod sursă – configurarea server-ului Bluetooth

```
    bool gv_isPaired = false;
    xQueueHandle ble_rcv_queue = NULL;
```

```
static uint8_t char1_str[] = {0x11, 0x22, 0x33};
static esp_gatt_char_prop_t property = 0;
static prepare_type_env_t prepare_write_env;
static uint8_t adv_config_done = 0;

static esp_attr_value_t gatts_char1_val = {
    .attr_max_len = GATTS_CHAR_VAL_LEN_MAX,
    .attr_len = sizeof(char1_str),
    .attr_value = char1_str,
};

static bool isAuthorized = false;

/* ADV ID */
static uint8_t adv_service_uuid128[ESP_UUID_LEN_128] = {
    /* LSB
<-----------------------------------------------------------------------
---------> MSB */

0xfb,0x34,0x9b,0x5f,0x80,0x00,0x00,0x80,0x00,0x10,0x00,0x00,0xEE,0x00,0x
00,0x00,
};

static esp_ble_adv_data_t adv_data = {
    .set_scan_rsp = false,
    .include_name = true,
    .include_txpower = false,
    .min_interval = 0x0006, //slave connection min interval, Time =
min_interval * 1.25 msec
    .max_interval = 0x0010, //slave connection max interval, Time =
max_interval * 1.25 msec
    .appearance = 0x00,
    .manufacturer_len = 0,
    .p_manufacturer_data = NULL,
    .service_data_len = 0,
    .p_service_data = NULL,
    .service_uuid_len = sizeof(adv_service_uuid128),
    .p_service_uuid = adv_service_uuid128,
    .flag = (ESP_BLE_ADV_FLAG_GEN_DISC |
ESP_BLE_ADV_FLAG_BREDR_NOT_SPT),
};

static esp_ble_adv_data_t scan_rsp_data = {
    .set_scan_rsp = true,
    .include_name = true,
    .include_txpower = true,
```

```
        .appearance = 0x00,
        .manufacturer_len = 0,
        .p_manufacturer_data = NULL,
        .service_data_len = 0,
        .p_service_data = NULL,
        .service_uuid_len = sizeof(adv_service_uuid128),
        .p_service_uuid = adv_service_uuid128,
        .flag = (ESP_BLE_ADV_FLAG_GEN_DISC |
    ESP_BLE_ADV_FLAG_BREDR_NOT_SPT),
    };

    static esp_ble_adv_params_t adv_params = {
        .adv_int_min = 0x20,
        .adv_int_max = 0x40,
        .adv_type = ADV_TYPE_IND,
        .own_addr_type = BLE_ADDR_TYPE_PUBLIC,
        .channel_map = ADV_CHNL_ALL,
        .adv_filter_policy = ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY,
    };

    static struct gatts_profile_inst gl_profile_tab[PROFILE_NUM] = {
        [PROFILE_APP_ID] = {
            .gatts_cb = kf_gatts_profile_event_handler,
            .gatts_if = ESP_GATT_IF_NONE,
        }};

    /**
     * @brief Received messages handler.
     * @return Does not return - infinite loop.
    */
    static void kf_handle_recv_message(void *arg)
    {
        uint8_t *msg;
        for (;;)
        {
            if (xQueueReceive(ble_rcv_queue, &msg, portMAX_DELAY))
            {
                printf("Handler: ");
                printf((char*)msg);
                printf("\n");
                if (strcmp((char*)msg, SECURITY_CODE) == 0)
                {
                    ESP_LOGI(GATTS_TAG, "CERERE DE AUTORIZARE");
                    isAuthorized = true;
```

```
                    kf_set_pin_on_high(LGREEN_PIN, 270);
                    vTaskDelay(95 / portTICK_PERIOD_MS);
                    kf_set_pin_on_high(LGREEN_PIN, 270);
                    vTaskDelay(95 / portTICK_PERIOD_MS);
                    kf_set_pin_on_high(LGREEN_PIN, 270);
                }
                else if (strcmp((char*)msg, FIND_ACCES_CODE) == 0)
                {
                    ESP_LOGI(GATTS_TAG, "CERERE GĂSIRE DISPOZITIV");
                    kf_find(4000);
                }
                else
                {
                    ESP_LOGI(GATTS_TAG, "CEREREA NU POATE FI INTERPRETATĂ");
                }
            }
        }
    }

    /**
     * @brief GAP event handler
     * @return void
     */
    static void gap_event_handler(esp_gap_ble_cb_event_t event,
    esp_ble_gap_cb_param_t *param)
    {
        switch (event)
        {
        case ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT:
            adv_config_done &= (~adv_config_flag);
            if (adv_config_done == 0)
            {
                esp_ble_gap_start_advertising(&adv_params);
            }
            break;
        case ESP_GAP_BLE_SCAN_RSP_DATA_SET_COMPLETE_EVT:
            adv_config_done &= (~scan_rsp_config_flag);
            if (adv_config_done == 0)
            {
                esp_ble_gap_start_advertising(&adv_params);
            }
            break;
        case ESP_GAP_BLE_ADV_START_COMPLETE_EVT:
            if (param->adv_start_cmpl.status != ESP_BT_STATUS_SUCCESS)
```

```
            {
                ESP_LOGE(GATTS_TAG, "Porinirea serviciului de advertising a
esuat\n");
            }
            break;
        case ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT:
            if (param->adv_stop_cmpl.status != ESP_BT_STATUS_SUCCESS)
            {
                ESP_LOGE(GATTS_TAG, "Oprirea serviciului de advertising a
esuat\n");
            }
            else
            {
                ESP_LOGI(GATTS_TAG, "Serviciul de advertising a fost oprit\
n");
            }
            break;
        case ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT:
            break;
        default:
            break;
        }
    }

    /**
     * @brief GATTS event handler
     * @return void
     */
    static void gatts_event_handler(esp_gatts_cb_event_t event,
    esp_gatt_if_t gatts_if, esp_ble_gatts_cb_param_t *param)
    {
        /* If event is register event, store the gatts_if for each profile
    */
        if (event == ESP_GATTS_REG_EVT)
        {
            if (param->reg.status == ESP_GATT_OK)
            {
                gl_profile_tab[param->reg.app_id].gatts_if = gatts_if;
            }
            else
            {
                ESP_LOGI(GATTS_TAG, "Inregistrarea aplicatiei a esuat,
    status %d\n", param->reg.status);
                return;
            }
```

```
        }
        int idx;
        for (idx = 0; idx < PROFILE_NUM; idx++)
        {
            if (gatts_if == ESP_GATT_IF_NONE ||
                gatts_if == gl_profile_tab[idx].gatts_if)
            {
                if (gl_profile_tab[idx].gatts_cb)
                {
                    gl_profile_tab[idx].gatts_cb(event, gatts_if, param);
                }
            }
        }
    }

    /**
     * @brief GATTS profile handler
     * @return void
     */
    void kf_gatts_profile_event_handler(esp_gatts_cb_event_t event,
    esp_gatt_if_t gatts_if, esp_ble_gatts_cb_param_t *param)
    {
        switch (event)
        {
        case ESP_GATTS_REG_EVT:
        {
            ESP_LOGI(GATTS_TAG, "EVENIMENT DE ÎNREGISTRARE, Status={%d},
    ID={%d}\n", param->reg.status, param->reg.app_id);
            gl_profile_tab[PROFILE_APP_ID].service_id.is_primary = true;
            gl_profile_tab[PROFILE_APP_ID].service_id.id.inst_id = 0x00;
            gl_profile_tab[PROFILE_APP_ID].service_id.id.uuid.len =
    ESP_UUID_LEN_16;
            gl_profile_tab[PROFILE_APP_ID].service_id.id.uuid.uuid.uuid16 =
    GATTS_SERVICE_UUID;

            esp_err_t set_dev_name_ret =
    esp_ble_gap_set_device_name(DEVICE_NAME);
            if (set_dev_name_ret)
            {
                ESP_LOGE(GATTS_TAG, "NUMELE DISPOZITIVULUI NU POATE FI
    SETAT, Cod eroare={%x}", set_dev_name_ret);
            }

            esp_err_t ret = esp_ble_gap_config_adv_data(&adv_data);
            if (ret)
```

```
        {
            ESP_LOGE(GATTS_TAG, "CONFIGURAREA DATELOR DE ADVERTISING A
ESUAT, Cod eroare={%x}", ret);
        }
        adv_config_done |= adv_config_flag;

        ret = esp_ble_gap_config_adv_data(&scan_rsp_data);
        if (ret)
        {
            ESP_LOGE(GATTS_TAG, "CONFIGURAREA RASPUNSULUI SCANARII
DATELOR DE ADVERTISING A ESUAT, Cod eroare={%x}", ret);
        }
        adv_config_done |= scan_rsp_config_flag;

        esp_ble_gatts_create_service(gatts_if,
&gl_profile_tab[PROFILE_APP_ID].service_id, GATTS_NUM_HANDLE);
        break;
    }
    case ESP_GATTS_READ_EVT:
    {
        if (!isAuthorized)
        {
            ESP_LOGI(GATTS_TAG, "EVENIMENT DE CITIRE - NO DATA\n");
            esp_gatt_rsp_t rsp;
            memset(&rsp, 0, sizeof(esp_gatt_rsp_t));

            rsp.attr_value.handle = param->read.handle;
            rsp.attr_value.len = 7;

            rsp.attr_value.value[0] = 'N';
            rsp.attr_value.value[1] = 'O';
            rsp.attr_value.value[2] = ' ';
            rsp.attr_value.value[3] = 'D';
            rsp.attr_value.value[4] = 'A';
            rsp.attr_value.value[5] = 'T';
            rsp.attr_value.value[6] = 'A';

            esp_ble_gatts_send_response(gatts_if, param->read.conn_id,
param->read.trans_id,
                                        ESP_GATT_OK, &rsp);
        }
        else
        {
            ESP_LOGI(GATTS_TAG, "EVENIMENT DE CITIRE - USER AUTORIZAT\
n");
```

```
                esp_gatt_rsp_t rsp;
                memset(&rsp, 0, sizeof(esp_gatt_rsp_t));

                rsp.attr_value.handle = param->read.handle;
                rsp.attr_value.len = 8;
                rsp.attr_value.value[0] = 'E';
                rsp.attr_value.value[1] = 'F';
                rsp.attr_value.value[2] = 'T';
                rsp.attr_value.value[3] = '3';
                rsp.attr_value.value[4] = '2';
                rsp.attr_value.value[5] = '4';
                rsp.attr_value.value[6] = 'A';
                rsp.attr_value.value[7] = 'A';

                esp_ble_gatts_send_response(gatts_if, param->read.conn_id,
    param->read.trans_id,
                                            ESP_GATT_OK, &rsp);
                isAuthorized = false;
            }
            break;
        }
        case ESP_GATTS_WRITE_EVT:
        {
            ESP_LOGI(GATTS_TAG, "EVENIMENT DE SCRIERE");
            if (!param->write.is_prep)
            {
                esp_log_buffer_char(GATTS_TAG, param->write.value, param-
    >write.len);
                xQueueSend(ble_rcv_queue, (void *)&param->write.value,
    (TickType_t)0);
            }

            esp_gatt_status_t status = ESP_GATT_OK;
            if (param->write.need_rsp)
            {
                if (param->write.is_prep)
                {
                    if (prepare_write_env.prepare_buf == NULL)
                    {
                        prepare_write_env.prepare_buf = (uint8_t
    *)malloc(PREPARE_BUF_MAX_SIZE * sizeof(uint8_t));
                        prepare_write_env.prepare_len = 0;
                        if (prepare_write_env.prepare_buf == NULL)
                        {
                            status = ESP_GATT_NO_RESOURCES;
```

```
                        }
                    }
                    else
                    {
                        if (param->write.offset > PREPARE_BUF_MAX_SIZE)
                        {
                            status = ESP_GATT_INVALID_OFFSET;
                        }
                        else if ((param->write.offset + param->write.len) >
PREPARE_BUF_MAX_SIZE)
                        {
                            status = ESP_GATT_INVALID_ATTR_LEN;
                        }
                    }

                    esp_gatt_rsp_t *gatt_rsp = (esp_gatt_rsp_t
*)malloc(sizeof(esp_gatt_rsp_t));
                    gatt_rsp->attr_value.len = param->write.len;
                    gatt_rsp->attr_value.handle = param->write.handle;
                    gatt_rsp->attr_value.offset = param->write.offset;
                    gatt_rsp->attr_value.auth_req = ESP_GATT_AUTH_REQ_NONE;
                    memcpy(gatt_rsp->attr_value.value, param->write.value,
param->write.len);
                    esp_err_t response_err =
esp_ble_gatts_send_response(gatts_if, param->write.conn_id, param-
>write.trans_id, status, gatt_rsp);
                    if (response_err != ESP_OK)
                    {
                        ESP_LOGE(GATTS_TAG, "EROARE LA TRIMITEREA
RASPUNSULUI\n");
                    }
                    free(gatt_rsp);
                    if (status != ESP_GATT_OK)
                    {
                        return;
                    }
                    memcpy(prepare_write_env.prepare_buf + param-
>write.offset,
                           param->write.value,
                           param->write.len);
                    prepare_write_env.prepare_len += param->write.len;
                }
                else
                {
                    esp_ble_gatts_send_response(gatts_if, param-
```

```
>write.conn_id, param->write.trans_id, status, NULL);
            }
        }
        break;
    }
    case ESP_GATTS_EXEC_WRITE_EVT:
        esp_ble_gatts_send_response(gatts_if, param->write.conn_id,
param->write.trans_id, ESP_GATT_OK, NULL);
        if (prepare_write_env.prepare_buf)
        {
            free(prepare_write_env.prepare_buf);
            prepare_write_env.prepare_buf = NULL;
        }
        prepare_write_env.prepare_len = 0;
        break;
    case ESP_GATTS_MTU_EVT:
        ESP_LOGI(GATTS_TAG, "ESP_GATTS_MTU_EVT, MTU %d", param-
>mtu.mtu);
        break;
    case ESP_GATTS_UNREG_EVT:
        break;
    case ESP_GATTS_CREATE_EVT:
    {
        ESP_LOGI(GATTS_TAG, "CREARE SERVICIU DE ADVERTISING");
        gl_profile_tab[PROFILE_APP_ID].service_handle = param-
>create.service_handle;
        gl_profile_tab[PROFILE_APP_ID].char_uuid.len = ESP_UUID_LEN_16;
        gl_profile_tab[PROFILE_APP_ID].char_uuid.uuid.uuid16 =
GATTS_CHAR_UUID;


esp_ble_gatts_start_service(gl_profile_tab[PROFILE_APP_ID].service_handl
e);
        property = ESP_GATT_CHAR_PROP_BIT_READ |
ESP_GATT_CHAR_PROP_BIT_WRITE;
        esp_err_t add_char_ret =
esp_ble_gatts_add_char(gl_profile_tab[PROFILE_APP_ID].service_handle,
&gl_profile_tab[PROFILE_APP_ID].char_uuid,

ESP_GATT_PERM_READ | ESP_GATT_PERM_WRITE,
                                                property,

&gatts_char1_val, NULL);
        if (add_char_ret)
        {
            ESP_LOGE(GATTS_TAG, "EROARE LA CREAREA SERVICIULUI DE
```

```
ADVERITISING, Cod eroare={%x}", add_char_ret);
        }
        break;
    }
    case ESP_GATTS_ADD_INCL_SRVC_EVT:
        break;
    case ESP_GATTS_ADD_CHAR_EVT:
    {
        uint16_t length = 0;
        const uint8_t *prf_char;

        ESP_LOGI(GATTS_TAG, "ADAUGARE DESCRIPTOR CARACTERISTICA\n");
        gl_profile_tab[PROFILE_APP_ID].char_handle = param-
>add_char.attr_handle;
        gl_profile_tab[PROFILE_APP_ID].descr_uuid.len = ESP_UUID_LEN_16;
        gl_profile_tab[PROFILE_APP_ID].descr_uuid.uuid.uuid16 =
ESP_GATT_UUID_CHAR_CLIENT_CONFIG;

        esp_err_t get_attr_ret = esp_ble_gatts_get_attr_value(param-
>add_char.attr_handle, &length, &prf_char);
        if (get_attr_ret == ESP_FAIL)
        {
            ESP_LOGE(GATTS_TAG, "HANDLE ILLEGAL");
        }
        esp_err_t add_descr_ret =
esp_ble_gatts_add_char_descr(gl_profile_tab[PROFILE_APP_ID].service_hand
le, &gl_profile_tab[PROFILE_APP_ID].descr_uuid,

ESP_GATT_PERM_READ | ESP_GATT_PERM_WRITE, NULL, NULL);
        if (add_descr_ret)
        {
            ESP_LOGE(GATTS_TAG, "ADAUGAREA DESCRIPTORULUI A ESUAT, Cod
eroare={%x}", add_descr_ret);
        }
        break;
    }
    case ESP_GATTS_ADD_CHAR_DESCR_EVT:
    {
        gl_profile_tab[PROFILE_APP_ID].descr_handle = param-
>add_char_descr.attr_handle;
        ESP_LOGI(GATTS_TAG, "ADAUGAREA DESCRIPTORULUI A REUSIT,
Status={%d}\n", param->add_char_descr.status);
        break;
    }
    case ESP_GATTS_DELETE_EVT:
```

```
            break;
        case ESP_GATTS_START_EVT:
        {
            ESP_LOGI(GATTS_TAG, "PORNINREA SERVICIULUI, Status={%d}\n",
    param->start.status);
            break;
        }
        case ESP_GATTS_STOP_EVT:
            break;
        case ESP_GATTS_CONNECT_EVT:
        {
            gv_isPaired = true;
            isAuthorized = false;
            esp_ble_conn_update_params_t conn_params = {0};
            memcpy(conn_params.bda, param->connect.remote_bda,
    sizeof(esp_bd_addr_t));
            conn_params.latency = 0;
            conn_params.max_int = 0x20; // max_int = 0x20*1.25ms = 40ms
            conn_params.min_int = 0x10; // min_int = 0x10*1.25ms = 20ms
            conn_params.timeout = 400;  // timeout = 400*10ms = 4000ms
            ESP_LOGI(GATTS_TAG, "EVENIMET DE CONECTARE, ID={%02x:%02x:%02x:
    %02x:%02x:%02x}",
                     param->connect.remote_bda[0], param-
    >connect.remote_bda[1], param->connect.remote_bda[2],
                     param->connect.remote_bda[3], param-
    >connect.remote_bda[4], param->connect.remote_bda[5]);
            gl_profile_tab[PROFILE_APP_ID].conn_id = param->connect.conn_id;

            //start sent the update connection parameters to the peer
    device.
            esp_ble_gap_update_conn_params(&conn_params);
            break;
        }
        case ESP_GATTS_DISCONNECT_EVT:
        {
            gv_isPaired = false;
            ESP_LOGI(GATTS_TAG, "EVENIMENT DE DECONECTARE, MOTIV={0x%x}",
    param->disconnect.reason);
            esp_ble_gap_start_advertising(&adv_params);
            break;
        }
        case ESP_GATTS_CONF_EVT:
            break;
        case ESP_GATTS_OPEN_EVT:
            break;
```

```
        case ESP_GATTS_CANCEL_OPEN_EVT:
            break;
        case ESP_GATTS_CLOSE_EVT:
            break;
        case ESP_GATTS_LISTEN_EVT:
            break;
        case ESP_GATTS_CONGEST_EVT:
            break;
        default:
            break;
        }
    }
    void kf_config_bt()
    {
        esp_err_t ret;

        ret = nvs_flash_init();
        if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
    ESP_ERR_NVS_NEW_VERSION_FOUND)
        {
            ESP_ERROR_CHECK(nvs_flash_erase());
            ret = nvs_flash_init();
        }
        ESP_ERROR_CHECK(ret);


    ESP_ERROR_CHECK(esp_bt_controller_mem_release(ESP_BT_MODE_CLASSIC_BT));

        esp_bt_controller_config_t bt_cfg =
    BT_CONTROLLER_INIT_CONFIG_DEFAULT();
        ret = esp_bt_controller_init(&bt_cfg);
        if (ret)
        {
            ESP_LOGE(GATTS_TAG, "%s Initializarea controller-ului BT a
    esuat: %s\n", __func__, esp_err_to_name(ret));
            return;
        }

        ret = esp_bt_controller_enable(ESP_BT_MODE_BLE);
        if (ret)
        {
            ESP_LOGE(GATTS_TAG, "%s Activarea controller-ului BT a esuat:
    %s\n", __func__, esp_err_to_name(ret));
            return;
        }
```

```
    ret = esp_bluedroid_init();
    if (ret)
    {
        ESP_LOGE(GATTS_TAG, "%s Initializarea BT a esuat: %s\n",
__func__, esp_err_to_name(ret));
        return;
    }

    ret = esp_bluedroid_enable();
    if (ret)
    {
        ESP_LOGE(GATTS_TAG, "%s Activarea BT a esuat: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    ret = esp_ble_gatts_register_callback(gatts_event_handler);
    if (ret)
    {
        ESP_LOGE(GATTS_TAG, "INREGISTRAREA PROFILULUI GENERIC DE ATRIBUT
A ESUAT, Cod eroare={%x}", ret);
        return;
    }

    ret = esp_ble_gap_register_callback(gap_event_handler);
    if (ret)
    {
        ESP_LOGE(GATTS_TAG, "INREGISTRAREA PROFILULUI GENERIC DE ACCES A
ESUAT, Cod eroare={%x}", ret);
        return;
    }

    ret = esp_ble_gatts_app_register(PROFILE_APP_ID);
    if (ret)
    {
        ESP_LOGE(GATTS_TAG, "INREGISTRAREA ID-ULUI APLICATIEI A ESUAT =
%x", ret);
        return;
    }

    esp_err_t local_mtu_ret = esp_ble_gatt_set_local_mtu(23);
    if (local_mtu_ret)
    {
        ESP_LOGE(GATTS_TAG, "MTU NU A PUTUT FI SETAT, Cod eroare={%x}",
local_mtu_ret);
```

```
    }

    gv_isPaired = false;
    ble_rcv_queue = xQueueCreate(10, MESSAGE_LEN * sizeof(char *));
    xTaskCreate(kf_handle_recv_message, "kf_handle_recv_message", 8192,
NULL, 10, NULL);
}
```

Anexa 12. Cod sursă – configurarea pinilor de intrare/ieșire

```
extern bool gv_isPaired;
xQueueHandle gpio_evt_queue = NULL;

static ledc_channel_config_t ledc_channel = {
    .channel = LEDC_CHANNEL_0,
    .duty = 0,
    .gpio_num = BUZZER_PIN,
    .speed_mode = LEDC_HS_MODE,
    .hpoint = 0,
    .timer_sel = LEDC_HS_TIMER
};

static ledc_timer_config_t ledc_timer = {
    .duty_resolution = LEDC_TIMER_13_BIT,
    .freq_hz = FREQUECY,
    .speed_mode = LEDC_HS_MODE,
    .timer_num = LEDC_HS_TIMER,
    .clk_cfg = LEDC_AUTO_CLK,
};

void kf_set_pin_on_high(int pin, int timeInMillis)
{
    gpio_set_level(pin, HIGH);
    vTaskDelay(timeInMillis / portTICK_PERIOD_MS);
    gpio_set_level(pin, LOW);
}

static void IRAM_ATTR gpio_isr_handler(void *arg)
{
    uint32_t gpio_num = (uint32_t)arg;
    xQueueSendFromISR(gpio_evt_queue, &gpio_num, NULL);
}

static void kf_button_action_task(void *arg)
{
```

```
        uint32_t io_num;
        for (;;)
        {
            if (xQueueReceive(gpio_evt_queue, &io_num, portMAX_DELAY))
            {
                if (gpio_get_level(io_num) == 1)
                {
                    if (gv_isPaired)
                    {
                        kf_set_pin_on_high(LBLUE_PIN, 1000);
                    }
                    else
                    {
                        kf_set_pin_on_high(LGREEN_PIN, 1000);
                    }
                }
            }
        }
    }

    void kf_config_gpio()
    {
        gpio_config_t io_conf;

        io_conf.intr_type = GPIO_INTR_DISABLE;
        io_conf.mode = GPIO_MODE_OUTPUT;
        io_conf.pin_bit_mask = OUTPUT_MASK;
        io_conf.pull_down_en = 0;
        io_conf.pull_up_en = 0;
        gpio_config(&io_conf);

        io_conf.intr_type = GPIO_INTR_POSEDGE;
        io_conf.mode = GPIO_MODE_INPUT;
        io_conf.pin_bit_mask = INPUT_MASK;
        gpio_config(&io_conf);

        ledc_timer_config(&ledc_timer);
        ledc_channel_config(&ledc_channel);
        ledc_fade_func_install(0);

        gpio_evt_queue = xQueueCreate(10, sizeof(uint32_t));

        xTaskCreate(kf_button_action_task, "kf_button_action_task", 2048,
    NULL, 10, NULL);
```

```
        gpio_install_isr_service(ESP_INTR_FLAG_LOWMED);
        gpio_isr_handler_add(BUTTON_PIN, gpio_isr_handler, (void
    *)BUTTON_PIN);
    }

    void kf_find(int timeInMillis)
    {
        int timer = 0;
        int duration = 400;
        while (timer < timeInMillis)
        {
            esp_err_t res = ledc_set_duty(ledc_channel.speed_mode,
    ledc_channel.channel, duration);
            if (res == ESP_OK)
            {
                ledc_update_duty(ledc_channel.speed_mode,
    ledc_channel.channel);
            }

            int delay = 50;
            while (duration >= 0)
            {
                kf_set_pin_on_high(LGREEN_PIN, delay);
                kf_set_pin_on_high(LBLUE_PIN, delay);
                kf_set_pin_on_high(LRED_PIN, delay);
                duration -= delay * 3;
            }

            res = ledc_set_duty(ledc_channel.speed_mode,
    ledc_channel.channel, 0);
            if (res == ESP_OK)
            {
                ledc_update_duty(ledc_channel.speed_mode,
    ledc_channel.channel);
            }

            duration = 400;
            timer += duration;
            vTaskDelay(100 / portTICK_PERIOD_MS);
        }
    }
```