



UNIVERSITATEA „ȘTEFAN CEL MARE” SUCEAVA
FACULTATEA DE INGINERIE ELECTICĂ ȘI ȘTIINȚA
CALCULATOARELOR
PROGRAM DE STUDIU: AUTOMATICĂ ȘI INFORMATICĂ
APLICATĂ

Sistem integrat de comunicație vehicul–infrastructură (V2I). Indicatoare rutiere adaptive cu afișaj de tip E-Ink și tehnologie RFID.

Coordonator,
Ş.I.dr.ing. Corneliu BUZDUGA

Student,
Elena Iuliana BULGARIU

Cuprins

Capitolul 1. Introducere.....	1
1.1. Scopul lucrării.....	1
1.2. Motivația alegerii temei	1
1.3. Probleme actuale în infrastructura rutieră și necesitatea modernizării	2
1.4. Inovații în infrastructura rutieră urbană: direcții și aplicații actuale.....	2
1.5. Introducerea în temă.....	4
Capitolul 2. Stadiul temei în bibliografie și realizări	5
2.1. Comunicarea vehicul–infrastructură (V2I)	5
2.2. Soluții V2I cu aplicații mobile și semnalizare digitală	5
2.3. Optimizarea traficului prin V2I.....	6
2.4. Sisteme V2I bazate pe RFID.....	7
2.5. Afisaje de tip E-Ink în aplicații de transport	8
2.6. Utilizarea combinată a tehnologiilor RFID și E-Ink în sisteme V2I.....	8
2.7. Lacune identificate în literatura de specialitate.....	10
2.8. Aportul original al prezentei cercetări	10
Capitolul 3. Proiectarea sistemului	11
3.1 Arhitectura propusă.....	11
3.1.1. Indicator rutier inteligent (Indicator 1 și Indicator 2).....	12
3.1.2. Vehicul Elysium RC	13
3.1.3. Aplicație Android	14
3.1.4. Interacțiuni între componente.....	16
3.2. Cerințe de sistem	18
3.3. Selecția componentelor	20
Capitolul 4. Implementare, dezvoltare și testare	21
4.1. Dezvoltarea la nivel de schemă electrică a nodului de semnalizare	21
4.2. Dezvoltarea modulului pentru semnele de circulație	22
4.3. Dezvoltarea modulului RFID UHF	23
4.4. Implementarea hardware	25
4.4.1. Implementarea modulului pentru semne de circulație.....	25
4.4.2. Implementarea modulului RFID UHF pe vehiculul Elysium RC	26
4.5. Aplicație mobiă-Interacțiune mobilă în timp real cu infrastructura rutieră.....	27
4.5.1. Fluxul operațional și procesul de interacțiune cu sistemul.....	27
4.5.2. Premise și restricții de platformă	30
4.5.3. Scanarea BLE pentru semnele de circulație	31

4.5.5. Agregarea și prezentarea rezultatelor	34
4.5.6. Arhitectura aplicației și repartizarea responsabilităților	37
4.5.7. Semnul rutier adaptiv	38
4.5.8. BleManager – rol și funcționare.....	40
4.5.9. DisplayManager.....	41
4.5.10. TrafficAlertReceiver	42
4.6. Aplicația Android – interfața de control.....	46
4.7. Testarea sistemului.....	51
Concluzii, contribuții și direcții viitoare de cercetare	56
Bibliografie.....	60
Anexe	62

Capitolul 1. Introducere

1.1. Scopul lucrării

Scopul lucrării este conceperea și construirea unui sistem de comunicare adaptiv pentru vehicule și infrastructură (V2I) care să eficientizeze traficul și să-l facă mai sigur. Obiectivul principal este de a dezvolta o soluție autonomă capabilă să interacționeze cu infrastructura rutieră prin citirea și scrierea pe tag-urilor RFID a unui cod unic de identificare pentru fiecare tag, a locației GPS, a orientării și să actualizeze semnele de circulație afișate pe ecrane E-Ink.

Cercetarea experimentală a fost principala metodă ce a permis evaluarea performanțelor sistemului într-un mediu controlat, urmărind obiectivele sistemului adaptiv de comunicare vehicul-infrastructură (V2I). Colectarea datelor a fost realizată prin măsurători experimentale. Variabilele cantitative importante ce au fost culese în urma testărilor includ timpii de răspuns ai sistemului, unghiul de citire, impactul condițiilor meteorologice și distanțele maxime de citire ale etichetelor RFID. Datele au fost apoi prelucrate, iar mai apoi analizate. Rezultatele experimentale confirmă fiabilitatea soluției propuse, validând arhitectura sistemului pentru aplicații extinse.

1.2. Motivația alegерii temei

Alegerea temei pentru această lucrare a fost ghidată de necesitatea de modernizare a infrastructurii rutiere, în condițiile în care traficul urban este tot mai aglomerat, iar soluțiile tradiționale de semnalizare devin ineficiente. Sistemele de semnalizare fixe, neadaptive și lipsite de intelligentă contextuală nu mai corespund cerințelor actuale, în special într-o lume în care numărul de autovehicule este în continuă creștere și în care se discută vrevent despre vehicule autonome.

Lucrarea de față propune un sistem care să răspundă acestor deficiențe prin introducerea unui mecanism adaptiv de semnalizare rutieră, bazat pe afișaje cu cerneală electronică (E-Ink) și tehnologie RFID.

Această lucrare este importantă nu doar prin caracterul său tehnic, ci și prin faptul că propune o alternativă fezabilă, sustenabilă și replicabilă la modelele clasice de semnalizare rutieră. Mai mult, lucrarea propune o arhitectură eficientă energetic, ușor de implementat și adaptabilă diferitelor scenarii de utilizare: de la semnalizare rutieră temporară, până la

integrarea în ecosistemele unor orașe sau testare pentru vehicule autonome. Soluția este sustenabilă și poate funcționa fără alimentare constantă, fiind potrivită pentru zone izolate sau mai puțin dezvoltate din punct de vedere tehnologic.

1.3. Probleme actuale în infrastructura rutieră și necesitatea modernizării

Sistemele actuale de infrastructură rutieră sunt adesea învechite, neadaptate dinamicii reale a traficului și lipsite de flexibilitate. În multe orașe europene, dar și în România, semnalizarea rutieră este statică, ceea ce face ca informațiile afișate să nu reflecte condițiile reale din teren. Pe lângă aspectul siguranței, trebuie menționată și nevoia de eficiență și automatizare. În lipsa unor soluții adaptative, costurile cu montarea, întreținerea și înlocuirea semnelor rutiere sunt ridicate, iar timpul de reacție al autorităților este adesea întârziat. În același timp, traficul urban este inefficient, iar lipsa unor mecanisme de comunicare V2I afectează inclusiv progresul în domeniul vehiculelor autonome.

Potrivit datelor Comisiei Europene, România se află printre statele membre cu cele mai multe accidente rutiere raportate la populație [1]. În anul 2022, s-au înregistrat peste 30.000 de accidente pe drumurile naționale, iar cauzele includ: lipsa semnalizării corecte în zonele cu lucrări; modificarea neanunțată a regimului de circulație; comportamentul șoferilor în lipsa unor informații actualizate în timp real [2].

1.4. Inovații în infrastructura rutieră urbană: direcții și aplicații actuale

În contextul creșterii urbane accelerate și al nevoii stringente de mobilitate sustenabilă, orașele din întreaga lume încep să adopte soluții tehnologice avansate pentru modernizarea infrastructurii rutiere. Inovațiile din domeniul transporturilor inteligente (ITS-Intelligent Transport Systems) și al Internetului Lucrurilor (IoT) permit integrarea semnalizării rutiere într-un sistem digital capabil să reacționeze în timp real la condițiile din teren. Aceste transformări urmăresc nu doar îmbunătățirea siguranței și fluidizarea traficului, ci și reducerea emisiilor și optimizarea resurselor publice. În continuare, sunt prezentate câteva exemple relevante de orașe care au implementat tehnologii inteligente în semnalizarea rutieră, demonstrând impactul pozitiv al acestor inițiative asupra eficienței și siguranței mobilității urbane.

Un exemplu de abordare modernă este Oslo, unde autoritățile utilizează stâlpi de iluminat inteligenți cu senzori integrați și unde, pe arterele principale, limitele de viteză sunt afișate pe

indicatoare LED variabile (Skilt 362) conectate la centrul de comandă[3]. Sistemul face parte dintr-un plan mai amplu de mobilitate sustenabilă: rețeaua IoT urbană colectează date de trafic și mediu, iar panourile variabile pot afișa mesaje contextuale pentru a reduce emisiile și a crește siguranța rutieră.

În cartierul portuar Jätkäsaari, proiectul „Smart Junction” al Mobility Lab Helsinki a transformat trei intersecții într-un nod C-ITS (Cooperative Intelligent Transport Systems) adică Sisteme de transport inteligente cooperante, în care infrastructura rutieră, vehiculele și utilizatorii vulnerabili colaborează în timp real: un ansamblu de LiDAR Velodyne, radare Doppler și camere stereo clasifică vehiculele, bicliștii și pietonii cu o acuratețe de $\approx 97\%$; datele sunt procesate pe edge-gateway-uri Nodeon/Commsignia, care ajustează în timp real fazele semaforului și emit mesaje către vehiculele conectate și aplicația mobilă „Smart Intersection”, în timp ce semafoarele, dotate cu module LED/matrix integrate, pot afișa limite de viteză sau avertizări contextuale; toate fluxurile și indicatorii de performanță sunt publicați pe portalul Jätkäsaari Smart Junction. După primul an de implementare, orașul raportează un trafic auto cu 15-30 % mai fluent și timpi de traversare pietonală cu 12 % mai mici, confirmând că intersecția intelligentă sporește simultan siguranța și eficiența transportului urban[4][5][6][7].

În Singapore, Land Transport Authority exploatează platforma Expressway Monitoring and Advisory System (EMAS) care monitorizează traficul și actualizează aproape 400 de panouri variabile amplasate pe zece artere; mesajele multicolore despre congestii, accidente sau lucrări sunt generate, transmise automat și mai apoi afișate pe portalul DataMall, unde datele se actualizează din două în două minute. Programul de modernizare, care s-a desfășurat pe perioada 2013-2017, a înlocuit toate panourile cu LED-uri de șapte culori și a extins rețeaua pe noi trasee [8][9].

Totuși, exemplele de mai sus rămân, deocamdată, insule de inovație într-un peisaj rutier încă dominat de semnalizare statică și infrastructuri neconectate; fapt ce subliniază oportunitatea și necesitatea unor soluții accesibile, cu consum redus și ușor de implementat, astfel încât modernizarea semnalizării rutiere să poată fi extinsă și către comunități cu resurse limitate, nu doar către metropole cu bugete mari.

1.5. Introducerea în temă

Soluția propusă constă într-un sistem V2I cu consum redus de energie, format dintr-un afișaj E-Ink (electronic ink - „cerneală electronică”) care poate reda atât mesaje text, cât și pictogramele semnelor de circulație, două taguri RFID UHF și un microcontroler ESP32-C3 SuperMini cu Bluetooth cu consum redus de energie - Bluetooth Low Energy (BLE). Afișajul se alimentează doar în momentul schimbării imaginii, iar tag-urile(etichetele RFID) oferă vehiculului informații esențiale – ID-ul semnului, tipul indicatoarelor, orientarea și poziționarea lor. Actualizarea indicatorului se realizează în două moduri: (1) din aplicația mobilă, printr-o comandă Bluetooth ce declanșează scurt alimentarea și schimbă conținutul ecranului; (2) automat, când un autovehicul echipat cu o antenă RFID rescrie tag-ul și inițiază instant afișarea nouui semn. Sistemul este adaptabil, oferă o autonomie ridicată și poate fi extins ca parte a unei rețele V2I bazate pe afișaje E-Ink și tehnologie RFID, facilitând modernizarea semnalizării chiar și în zone cu bugete reduse sau infrastructură limitată, în timp ce asigură compatibilitatea cu vehiculele autonome.

Capitolul 2. Stadiul temei în bibliografie și realizări

2.1. Comunicarea vehicul–infrastructură (V2I)

Comunicarea vehicul–infrastructură (V2I), parte a ecosistemului V2X- (Vehicle to Everything- reprezintă un concept tehnologic care definește comunicarea vehiculului cu orice element din jurul său), ce permite vehiculelor să primească în timp real informații esențiale despre semnalizare rutieră, condițiile de trafic, obstacole temporare sau starea semafoarelor [10].

Pentru aplicații la scară redusă sau în medii controlate, cum sunt vehiculele demonstrative sau platformele educaționale, o alternativă eficientă este utilizarea tehnologiei RFID(Radio Frequency Identification- Identificare prin frecvență radio). Aceasta permite simularea infrastructurii rutiere prin intermediul tag-urilor UHF, oferind o soluție cu cost redus și consum minim de energie, în comparație cu tehnologii mai complexe precum LiDAR (Light Detection and Ranging-tehnologie de detecție care utilizează impulsuri laser pentru a măsura distanțe și pentru a crea hărți 3D foarte precise ale mediului înconjurător) sau DSRC (Dedicated Short Range Communication-este un protocol de comunicație wireless dedicat vehiculelor inteligente, care funcționează la 5.9 GHz, pe distanțe scurte (300–1000 metri), cu latență extrem de mică.)[11].

Tag-urile RFID pot transmite coduri semnificative către cititorul aflat pe vehicul, fiind detectate eficient la viteze reduse și în condiții de poziționare optimă, cu rate de succes de peste 80% [12]. Mai mult, unele cercetări explorează utilizarea nivelului RSSI (Received Signal Strength Indicator- valoare numerică care exprimă intensitatea semnalului radio recepționat de un dispozitiv (ex. modul Bluetooth, Wi-Fi, RFID etc.) sau a fazei semnalului RFID pentru localizare, obținând o precizie de sub 20 cm, suficientă pentru navigare autonomă de precizie [13].

2.2. Soluții V2I cu aplicații mobile și semnalizare digitală

Pe lângă soluțiile hardware dedicate, aplicațiile mobile pot juca un rol activ în V2I. Un exemplu este sistemul propus de Masatu et al., care utilizează GPS-ul telefonului mobil și o aplicație Android pentru a alerta șoferul atunci când se apropiе de un semn rutier aflat la mai puțin de 10 metri [14]. Această abordare demonstrează că smartphone-urile pot funcționa ca interfețe V2I accesibile pentru vehicule semi-autonome.

Într-o direcție evolutivă, Calafate și Garcia propun o infrastructură rutieră autonomă, în care semnele de circulație devin elemente programabile ce pot transmite informații direct vehiculelor, prin comunicații wireless. Mesajele afișate de semn sunt dinamice, adaptându-se contextului rutier, cum ar fi lucrări, restricții temporare sau condiții de trafic [15].

O abordare mai avansată este prezentată de Rahman (2023), care a combinat recunoașterea vizuală (YOLOv8s- You Only Look Once Small) cu comunicația V2I, pentru controlul vehiculului la intersecții. Sistemul detectează semafoare într-un mediu de simulare (MAVS- Modular Adaptive Vision System- Sistem modular de viziune adaptivă) și transmite vehiculului mesaje de tip SPaT (Signal Phase and Timing-informații despre faza semaforului și sincronizarea acestuia), permitând acțiuni precum frânare anticipată sau menținerea vitezei în funcție de faza semaforului [16].

Aceste contribuții evidențiază că V2I poate fi implementat atât prin tehnologii de comunicație pasivă sau activă, cât și prin integrarea cu percepția vizuală. Direcția generală este clară: vehiculele viitorului trebuie să comunice eficient cu infrastructura rutieră pentru a susține decizii autonome sigure și eficiente.

2.3. Optimizarea traficului prin V2I

O direcție importantă de aplicare a V2I este optimizarea comportamentului vehiculului în intersecții, unde informația în timp real poate reduce întârzierile, consumul de combustibil și opririle inutile.

Ahmed et al. (2018) au dezvoltat un sistem Intersection Approach Advisory- Avertizare de apropiere de intersecție, care folosește DSRC pentru a transmite mesaje SPaT(Signal Phase and Timing) către vehicul. Pe baza acestor date, vehiculul afișează șoferului o viteză optimă de apropiere. Sistemul s-a dovedit eficient în simulări și teste de teren, contribuind la reducerea emisiilor fără a distrage atenția conducătorului [17].

Un alt exemplu este algoritmul de eco-driving propus de Sun et al., care corelează modelele cinetice ale vehiculului cu timpii semafoarelor. Prin ajustarea vitezei înainte de intrarea în intersecție, se obțin îmbunătățiri atât în termeni de eficiență energetică, cât și de fluiditate a traficului [18].

Zhang et al. propun un sistem de control predictiv al semaforizării, bazat pe algoritmi de receding-horizon- strategie de control predictiv și genetic programming. Aceasta integrează modele de trafic H-V(Horizontal–Vertical-Orizontal–Vertical) și comunicații V2I, oferind reduceri semnificative ale opririlor și întârzierilor în trafic urban [19].

Mai recent, Yang et al. (2024) au aplicat învățarea prin întărire (RL-Reinforcement Learning) în contextul sistemelor inteligente de transport urban, unde traficul este compus atât din vehicule autonome, cât și din vehicule convenționale, întărirea (RL – Reinforcement Learning) este o metodă de învățare automată extrem de valoroasă. Aceasta permite unui sistem (precum un semafor intelligent sau un vehicul autonom) să învețe prin interacțiune cu mediul. Aceste metode au demonstrat performanțe superioare în reducerea consumului și creșterea confortului [20].

Un alt studiu, realizat de Oliva et al., a explorat rolul V2I în siguranța pietonilor și a vehiculelor de urgență. Un prototip IoT a fost utilizat pentru a trimite alerte în timp real pe smartphone-urile șoferilor și pietonilor, reducând timpul de reacție și crescând gradul de conștientizare în intersecții [21].

2.4. Sisteme V2I bazate pe RFID

În ultimii ani, RFID UHF pasiv a devenit o opțiune viabilă pentru sisteme low-cost de comunicație vehicul–infrastructură (V2I). Radio Frequency Identification (RFID) este o tehnologie de identificare automată care permite citirea și, în unele cazuri, scrierea de date stocate pe etichete electronice (tag-uri), prin intermediul undelor radio.

RFID UHF (Ultra High Frequency) se referă la acea clasă de RFID care operează în intervalul de frecvență 860–960 MHz, conform standardului ISO/IEC 18000-6C / EPCglobal Gen2. Etichetele RFID UHF sunt, de regulă, pasive (fără sursă de alimentare proprie) și sunt activate de energia emisă de către cititor. Acestea pot fi detectate de la distanțe semnificative (până la 12 metri în condiții ideale), ceea ce le face potrivite pentru aplicații de identificare rapidă, non-contact, cum ar fi logistica, accesul controlat și sistemele de transport inteligente.

Un studiu din 2022 demonstrează utilizarea tag-urilor integrate pe drumuri și citite de un vehicul echipat cu cititor UHF, validând performanța sistemului în condiții de trafic moderat [22]. Ulterior, o lucrare din 2023 dezvoltă o infrastructură RFID optimizată, concentrându-se pe mărirea razei de citire la viteză și robustețe în medii urbane [23]. În 2023 un grup de cercetători a combinat RFID pasiv cu infrastructură edge-Cloud, integrând cititoare RFID în stații pentru a transmite date de trafic către vehiculele din proximitate [24]. Alte studii au explorat folosirea RFID pasiv pentru funcții avansate de control al vitezei (adaptive cruise control), demonstrând potențialul tehnologiei chiar și în vehicule convenționale [25]. În plus, cercetări recente se axează și pe aspecte de securitate datelor recepționate și transmise prin RFID în aplicații Smart City, abordând protecția datelor și prevenirea identificării neautorizate

[26]. Chiar dacă aceste abordări confirmă fezabilitatea RFID în V2I, ele nu includ interfețe mobile moderne (ex. aplicații Android) sau semnalizare fizică (E-Ink).

2.5. Afişaje de tip E-Ink în aplicaţii de transport

Tehnologia E-Ink (Electronic Ink- cerneală electronică) reprezintă o soluție avansată de afișare, bazată pe proprietăți electro-optice bistabile, utilizată în special în contexte care impun consum energetic redus și vizibilitate ridicată în lumină ambientală. Principiul de funcționare constă în utilizarea unor microcapsule ce conțin particule pigmentate cu sarcini electrice opuse, suspendate într-un fluid clar, care își modifică poziția sub acțiunea unui câmp electric. Acest mecanism permite afișarea de text sau imagini într-un mod similar hârtiei tipărite, fără emisie proprie de lumină. Spre deosebire de afișajele (LCD- Liquid Crystal Display-Afișaj cu cristale lichide, OLED- Organic Light Emitting Diode). Afișajele E-Ink funcționează pe baza reflecției luminii externe, ceea ce le conferă o vizibilitate excelentă în condiții de lumină puternică și un consum energetic aproape nul în regim static (bistabilitate). Bistabilitate (în contextul afișajelor E-Ink) se referă la capacitatea unui afișaj de a menține o imagine vizibilă fără a consuma energie electrică continuu. Energia este necesară doar în momentul schimbării conținutului vizual, caracteristică esențială în aplicații embedded sau mobile, precum cele din domeniul comunicației V2I (Vehicle-to-Infrastructure). În paralel cu dezvoltarea comunicațiilor V2I, cercetările recente au vizat înlocuirea panourilor LED cu afișaje e-paper (E-Ink) pentru aplicații în transport, în special în stații de autobuz, gări sau indicatoare temporare. Motivația principală constă în vizibilitatea ridicată în lumină naturală și consumul energetic aproape nul în stare de repaus [27]. Rout și Sahoo au integrat un panou E-Ink de 4,2" într-un semn de avertizare rutier temporar, alimentat cu energie solară, raportând o reducere de până la 10 ori a consumului față de un afișaj LED echivalent [28]. Cu toate acestea, soluția rămâne pasivă: afișajul este vizibil doar pentru șoferul uman și nu oferă comunicație digitală cu vehiculul.

2.6. Utilizarea combinată a tehnologiilor RFID și E-Ink în sisteme V2I

Cercetările care urmăresc integrarea tehnologiilor RFID și E-Ink sunt încă rare, dar promițătoare. Benini și Gobbi propun un nod V2I care afișează pictograme pe un panou e-paper și conține o etichetă RFID UHF cu ID-ul semnului, permitând astfel recunoaștere de către vehicul [29]. Limitarea acestui sistem este însă dată de lipsa unui canal de actualizare wireless, astfel, datele din tag pot fi modificate doar local, prin conectare USB.

Un alt studiu, realizat în Oslo, a testat stâlpi de iluminat stradal dotați cu panouri e-paper și etichete NFC (Near Field Communication – „Comunicare în Câmp Apropiat”) este o tehnologie de comunicare wireless (fără fir), care permite transferul de date pe distanțe foarte scurte, de obicei sub 10 cm, între două dispozitive compatibile. Această soluție a presupus oferirea de informații contextuale pietonilor. Deși conceptul propune un semn rutier inteligent, nu a fost utilizată banda UHF, ceea ce limitează posibilitatea ca vehiculele să acceseze informațiile de la distanță [30].

Pentru a evidenția principalele direcții de cercetare relevante pentru tema lucrării: integrarea comunicației vehicul–infrastructură (V2I) cu tehnologii precum RFID și afișajele E-Ink, a fost realizată o sinteză tematică a literaturii recente. Tabelul 1. prezintă într-o manieră structurată domeniile principale investigate, alături de o descriere concisă a contribuțiilor identificate în sursele studiate, referințele bibliografice corespunzătoare și observații privind aplicabilitatea și limitele fiecărei abordări. Această sinteză permite identificarea atât a tendințelor actuale, cât și a gologorilor identificate în literatura de specialitate în contextul dezvoltării sistemelor integrat V2I cu recunoaștere pasivă prin RFID și afișaj E-Ink.

Tabel 2.1. Sinteză tematică a cercetărilor recente.

Domeniu	Descriere sintetizată	Referințe	Observații sintetizate
Comunicare V2I	Vehiculul primește date despre trafic/semafor/semne; RFID = soluție eficientă	[10]–[13]	RFID e preferat pentru cost redus și aplicabilitate în prototipuri/medii controlate
Aplicații mobile în V2I	Avertizare rutieră prin aplicație Android; interfață simplă pentru vehicule autonome	[14]–[16]	Soluții mobile sunt scalabile, dar puțin integrate în infrastructuri V2I reale
Optimizare trafic cu V2I	Algoritmi care sincronizează vehiculul cu semafoarele (SPaT, eco-driving)	[17]–[21]	Reduce opririle și consumul; bune rezultate în simulare și testare
Sisteme V2I cu RFID	Taguri RFID pentru semnal rutier și zonare; detecție și localizare	[22]–[26]	Necesită optimizare antene și securitate; lipsesc integrări cu UI sau aplicații
Afișaj E-Ink în transport	Semne pasive lizibile solar; alimentata solar cu consum redus	[27]–[28]	Vizibile doar pentru oameni; nu comunică activ cu vehiculul

Tehnologii RFID și E-Ink	Sistem hibrid cu e-Ink + RFID pe semne stradale	[29]–[30]	Rareori testate în tandem; lipsesc soluții dinamice wireless sau control de la distanță
-----------------------------	----------------------------------------------------	-----------	--------------------------------------------------------------------------------------------------

2.7. Lacune identificate în literatura de specialitate

Analiza literaturii arată că:

- soluțiile RFID existente se concentrează pe citire, nu pe rescriere;
- afișajele E-Ink implementate până acum nu dispun de un mecanism V2I digital;
- sistemele care combină cele două tehnologii nu au un canal de actualizare wireless cu consum redus.

2.8. Aportul original al prezentei cercetări

Proiectul propus depășește aceste limitări prin integrarea unui microcontroler ESP32-C3 cu BLE, două etichete RFID UHF care se pot rescrie și un afișaj E-Ink de 2,13", oferind:

- actualizare duală (din aplicație sau la trecerea vehiculului),
- afișare simultană a semnului pentru percepția șoferului și interpretarea vehiculului,
- consumul de energie este suficient de redus încât o baterie externă MLPro 10 400 mAh să alimenteze sistemul pe o perioadă de câțiva ani.

În acest mod, proiectul contribuie la avansarea cercetării în domeniul comunicației vehicul-infrastructură, propunând un model adaptiv V2I, eficient energetic și realizabil cu resurse reduse, care poate fi implementat atât în spații urbane aglomerate, cât și în zone izolate sau cu infrastructură deficitară.

Capitolul 3. Proiectarea sistemului

Arhitectura sistemului integrează un vehicul autonom echipat cu un modul UHF RFID și ESP32-WROOM, două noduri independente de semnalizare rutieră cu ESP32-C3, ecrane E-Ink, două etichete RFID și o aplicație Android de comandă. Sistemul funcționează ca o rețea în care vehiculul autonom, prevăzut cu un ESP32-WROOM și cititor YRM 1003 RFID UHF, cele două noduri de semnalizare rutieră, construite pe baza modulelor ESP32-C3, echipate cu afișaje E-Ink și etichete RFID UHF, și aplicația Android formează un ansamblu complet interconectat. Toate cele trei componente schimbă date în regim bidirectional—BLE pentru configurări și ESP-NOW pentru evenimente în timp real (citire/rescriere etichetă, avertizare “ACCIDENT”, sincronizare între noduri). Deși implementarea curentă cuprinde un număr limitat de dispozitive, arhitectura permite, prin construcție, adăugarea ulterioară a altor noduri.

3.1 Arhitectura propusă

Arhitectura propusă este centrată pe trei elemente: ansamblul YRM 1003 de pe vehiculul autonom, care este conectat la ESP32-WROOM pentru a interacționa cu indicatoarele-ecranele cu cerneală electronică; două noduri de semnalizare rutieră și o aplicație Android pentru configurare și supraveghere. Nodurile de semnalizare sunt dispozitive independente, de dimensiuni reduse, alcătuite dintr-un afișaj E-Ink, un tag RFID UHF pasiv și un microcontroler ESP32-C3. De asemenea, nodurile de semnalizare sunt alimentate de o baterie externă MLPro de 10.400 mAh, care asigură funcționarea în regim de consum redus, fără conectare la rețea.

- Vehiculul Elysium RC este echipat cu un modul RFID UHF, utilizat pentru citirea, scrierea și rescrierea rapidă a tag-urilor RFID montate pe semne.
- Aplicația Android joacă rolul principal de interfață pentru configurația, monitorizarea și controlul sistemului, asigurând conexiunea cu nodurile de semnalizare prin Bluetooth Low Energy. Aceasta oferă funcționalitatea de actualizare a afișajelor E-Ink, precum și posibilitatea de a scrie date pe etichetele RFID, atunci când este necesar. În plus, aplicația poate comunica direct cu vehiculul printr-un canal BLE dedicat, facilitând operațiuni de scriere, sincronizare și coordonare între diferitele componente ale sistemului.

Schema bloc prezentată în Figura 3.2. ilustrează arhitectura generală a sistemului propus pentru semnalizare rutieră adaptivă bazată pe comunicație V2I (vehicul–infrastructură). Aceasta cuprinde trei entități principale:

3.1.1. Indicator rutier inteligent (Indicator 1 și Indicator 2)

ESP32-C3 ESP32-C3 este un microcontroler care servește drept unitate de control principală în cadrul nodurilor inteligente de semnalizare rutieră din sistemul V2I propus. Acesta integrează un nucleu RISC-V pe 32 de biți, conectivitate Wi-Fi 2.4 GHz și Bluetooth Low Energy (BLE) 5.0, fiind ideal pentru aplicații cu consum redus de energie și necesități de comunicare wireless.

În contextul sistemului, ESP32-C3 îndeplinește următoarele roluri esențiale:

- Gestionarea logicii locale a semnului: decide ce mesaj trebuie afișat pe ecranul E-Ink, în funcție de comanda primită din aplicația mobilă sau de la vehicul.
- Comunicare BLE: stabilește conexiuni Bluetooth cu aplicația Android, recepționează comenzi și transmite înapoi confirmări sau statusuri.
- Interfațare hardware: controlează afișajul E-Ink prin magistrala SPI și, optional, gestionează semnalul de scriere pe eticheta RFID (dacă se integrează și această funcție).
- Mod deep-sleep și economie de energie: permite alimentarea pe termen lung din surse portabile (ex. baterii externe), consumând sub 150 µA în standby.

Prin capabilitățile sale wireless și flexibilitatea de programare, ESP32-C3 devine o alegere optimă pentru noduri autonome, care trebuie să reacționeze rapid și sigur la evenimentele din trafic fără a depinde de infrastructură centralizată.

Ecran E-Ink reprezintă componenta de afișare a semnului rutier adaptiv, fiind esențial pentru vizualizarea mesajelor de către participanții la trafic. Acest tip de afișaj folosește tehnologia cernelei electronice, care imită aspectul hârtiei tipărite și oferă vizibilitate excelentă în lumină naturală intensă, inclusiv la soare direct. Dispozitivul este controlat direct de microcontrolerul ESP32-C3 prin interfață SPI și necesită o secvență de inițializare specifică, dar consumă energie doar în timpul actualizării imaginii – ceea ce îl face ideal pentru aplicații low-power. În regim static (imagine înghețată), consumul este practic nul datorită bistabilității sale – imaginea rămâne afișată fără alimentare continuă. Acest ecran este capabil să afișeze simboluri grafice rutier standardizate (ex. STOP, 50 km/h, Cedează trecerea), iar schimbarea conținutului durează aproximativ 1–2 secunde, cu un curent de vârf de 20–25 mA. Prin aceste caracteristici pe care le posedă: claritate, eficiență energetică și durabilitate în condiții exterioare ($-20 \dots +60^{\circ}\text{C}$), ecranul E-Ink de 2,13" oferă o soluție robustă și fiabilă pentru afișajul nodurilor inteligente de semnalizare rutieră.

Etichetă RFID UHF – etichetă pasivă integrată în semn ce conține informația digitală asociată simbolului afișat. Aceasta poate fi citită/scrisă/rescrisă de antena vehiculului. Eticheta **UHF IN9654** are mai multe tipuri de memorie, fiecare cu un rol bine definit:

EPC (Electronic Product Code) – identificator principal al tag-ului (implicit 96 biți):

- Util pentru identificarea rapidă și unică a semnului rutier (ex: „Stop”, „50 km/h”).
- Poate fi extins până la 480 biți în unele aplicații.

User Memory – memorie utilizabilă de către dezvoltator:

- Are **512 biți** (sau 64 octeți), și permite scrierea de informații personalizate precum:
 - tipul semnului,
 - nivelul de alertă,
 - stare specială (ex: „Drum blocat”).

TID (Tag Identifier) – cod unic, doar pentru citire:

- 64 biți, generat de producător.
- Nu poate fi modificat – este folosit ca „amprentă digitală” a etichetei.

Password Memory ce conține două câmpuri de 32 biți:

- **Access Password** – protejează scrierea.
- **Kill Password** – permite dezactivarea definitivă a tagului (pentru securitate).

Fiecare zonă de memorie poate fi accesată selectiv prin comenzi UHF Gen2, în funcție de nevoile sistemului V2I.

3.1.2. Vehicul Elysium RC

Simulează un vehicul inteligent echipat cu:

- ESP32-WROOM este un microcontroler dezvoltat de Espressif Systems, recunoscut pentru versatilitatea sa în aplicații IoT. În contextul sistemului propus, acesta îndeplinește rolul de gateway V2I (Vehicle-to-Infrastructure), adică servește drept punct de legătură între vehiculul autonom și infrastructura rutieră intelligentă. Mai exact, ESP32-WROOM:
 - ✓ primește informații de la senzori locali montați pe vehicul (de exemplu: detecție obstacole, orientare, poziționare);

- ✓ gestionează comunicarea cu nodurile de semnalizare prin protocole wireless (ex: ESP-NOW sau BLE);
- ✓ procesează și transmite datele relevante către alte componente ale sistemului, cum ar fi aplicația mobilă sau afișajele de pe traseu;
- ✓ suportă multitasking, conectivitate Wi-Fi + Bluetooth și are suficientă memorie RAM și flash pentru a rula logica de comunicare și control în timp real.

Prin acest rol, ESP32-WROOM funcționează ca un nod central al vehiculului, permitând o interacțiune bidirectională eficientă între vehicul și infrastructura inteligentă.

Modul RFID YRM1003 – cititor UHF performant, conectat la ESP32 prin UART oferă:

- ✓ Putere de transmisie reglabilă până la 30 dBm, oferind o rază de citire de aproximativ 2–5 metri în condiții optime;
- ✓ Dimensiuni reduse și consum energetic moderat (≈ 120 mA în timpul emiterii), fiind potrivit pentru aplicații mobile;
- ✓ Suport pentru funcții de citire și scriere, esențial pentru funcționalitatea V2I, unde vehiculul nu doar citește informații de pe semne, ci poate și actualiza conținutul etichetelor RFID.

Vehiculul autonom, echipat cu modulul RFID YRM1003 și microcontrolerul ESP32-WROOM, are capacitatea de a iniția în mod automat operații de rescriere a etichetelor RFID în timp ce se deplasează, fără a necesita oprirea acestuia. În plus, în cazul detectării unui eveniment critic – cum ar fi un accident sau un obstacol periculos – vehiculul poate transmite alerte de urgență în timp real prin protocolul ESP-NOW, către nodurile de semnalizare rutieră, care actualizează imediat mesajul afișat pe ecranele E-Ink, adaptând semnalizarea la noul context rutier.

3.1.3. Aplicație Android

Aplicația Android dezvoltată în cadrul proiectului preia rolul de interfață principală pentru controlul și monitorizarea sistemului, comunicând prin conexiuni Bluetooth Low Energy (BLE) cu fiecare modul ESP32-C3 din semnele rutiere inteligente. Aceasta replică funcționalitatea de bază a aplicației software originale livrate împreună cu modulul YRM1003 (Figura 3.1.), dar o extinde semnificativ, oferind funcții suplimentare esențiale pentru utilizare mobilă și operaționalizare rapidă în teren:

- permite selecția și trimiterea de comenzi personalizate către semnele rutiere;

- afisează în timp real starea conexiunii, ID-ul semnului și tipul evenimentului rutier detectat;
- oferă posibilitatea gestionării mai multor noduri simultan;
- poate transmite comenzi de rescriere către tagul RFID și afișajul E-Ink asociat;
- include suport pentru notificări automate în caz de evenimente critice (ex. accident raportat).

Astfel, aplicația mobilă propusă nu doar că reproduce funcționalitățile de bază ale interfeței software tradiționale (desktop), ci le optimizează și adaptează pentru un scenariu real, dinamic, specific infrastructurilor rutiere inteligente.

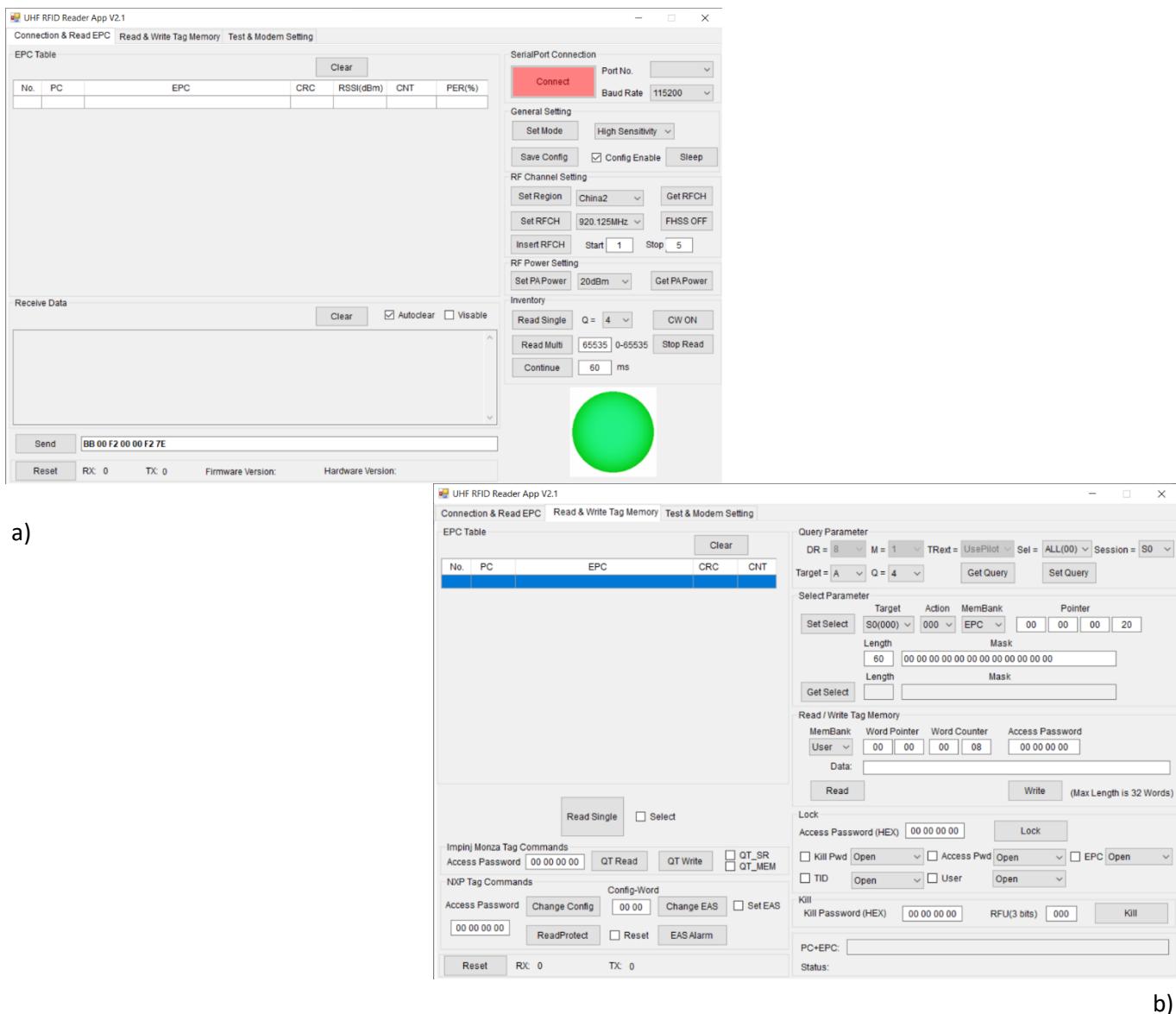


Figura 3.1. Interfața aplicației desktop YRM1003: a) vizualizare generală a conexiunii și citirii tagurilor. b) secțiunea de scriere și configurare a memoriei tagurilor UHF

3.1.4. Interacțiuni între componente

Interacțiunea dintre componentele sistemului V2I se desfășoară într-un flux coerent și adaptiv. Vehiculul autonom, echipat cu un modul RFID, citește eticheta UHF a indicatorului rutier și transmite datele relevante către aplicația mobilă. Aplicația, conectată prin Bluetooth Low Energy (BLE) la microcontrolerul ESP32-C3 din cadrul semnului, trimite comanda de actualizare. Ulterior, ESP32-C3 rescrie atât memoria tagului RFID, cât și afișajul E-Ink cu simbolul rutier corespunzător. În cazul detectării unui incident sau eveniment rutier, vehiculul generează automat o alertă, transmisă aplicației, care la rândul ei poate declansa actualizarea dinamică a semnelor afișate, asigurând o reacție rapidă și contextuală la schimbările din trafic.

Comenzi UHF Gen2 (conform standardului EPCglobal Class 1 Gen2) sunt un set de instrucțiuni digitale transmise de la un cititor RFID către un tag UHF pasiv, prin unde radio, pentru a efectua acțiuni precum citirea, scrierea sau modificarea conținutului stocat în memoria tagului. În contextul unui sistem V2I (Vehicle-to-Infrastructure), aceste comenzi sunt folosite pentru actualizarea dinamică a semnelor rutiere inteligente.

1. Inițierea comunicării-modulul RFID (ex: YRM1003) instalat pe vehicul sau pe infrastructură generează un semnal UHF (până la 30 dBm) pentru a interoga tagul. Acesta este alimentat pasiv de energia semnalului și „răspunde” la interogare.

2. Transmiterea comenzi Gen2-dacă sistemul detectează că este necesară actualizarea (de exemplu, modificarea semnului în caz de accident sau restricție temporară), cititorul RFID trimite comenzi Gen2 specifice, cum ar fi: scrierea unui nou cod de semn (ex. „STOP”, „50 km/h”), protejarea conținutului tagului, verificarea conținutului anterior

3. Executarea comenzi de către tag-eticheta răspunde executând comanda, modificând porțiunea de memorie desemnată, de obicei zona User Memory sau EPC Memory. Aceasta poate stoca: un cod numeric asociat semnului rutier; informații auxiliare: prioritate, valabilitate temporală, ID nod.

4. Confirmarea scrierii-după executarea comenzi, sistemul poate citi înapoi conținutul pentru a confirma actualizarea corectă – proces cunoscut ca read-after-write.

Figura 3.2 ilustrează arhitectura generală a sistemului propus, evidențiind modul în care componente hardware și software interacționează pentru a susține funcționarea semnalizării rutiere adaptive. Sistemul este alcătuit din mai multe noduri inteligente de semnalizare (Indicator 1 și Indicator 2), fiecare echipat cu un microcontroler ESP32-C3, un ecran E-Ink pentru afișare vizuală și un TAG RFID UHF pentru comunicare pasivă. Aceste noduri sunt coordonate prin intermediul unei aplicații mobile Android, care servește drept interfață de

control și distribuție a comenziilor, utilizând protocolul BLE pentru actualizări rapide și eficiente. Totodată, modulul mobil Elysium RC, compus dintr-un ESP32-Wroom și un cititor RFID YRM1003, permite vehiculului să scrie din mers pe tagurile RFID, activând astfel funcții specifice precum modificarea mesajului afișat în cazul unui eveniment critic. Structura prezentată subliniază redundanța și adaptabilitatea sistemului, facilitând o comunicare bidirectională robustă între infrastructură și vehicul, în absența unei rețele fixe.

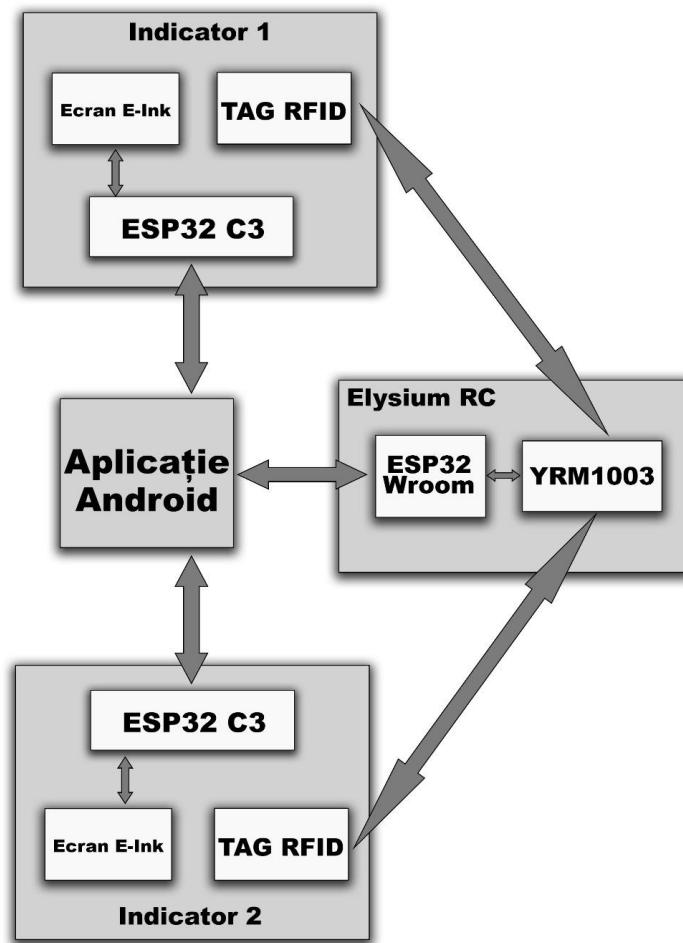


Figura 3.2. Arhitectura generală a sistemului propus.

3.2. Cerințe de sistem

Având în vedere scopul proiectului, acela de a implementa un sistem vehicul-infrastructură (V2I) funcțional, eficient energetic și capabil să opereze autonom în condiții reale, au fost definite o serie de cerințe tehnice pe baza funcționalităților urmărite și a limitărilor impuse de componentele hardware alese.

Arhitectura descrisă anterior a fost concepută pentru a susține aceste cerințe în mod direct: nodurile de semnalizare trebuie să funcționeze pe termen lung fără intervenție, vehiculul trebuie să poată interacționa cu semnele din mers, iar aplicația Android trebuie să permită intervenții rapide. Tabelul 3.2. prezintă cerințele împreună cu valorile de referință extrase din fișele tehnice ale componentelor utilizate.

Tabelul 3.1. Cerințe și specificații tehnice ale componentelor hardware din arhitectura propusă.

Nr. crt.	Cerință de sistem	Parametru de validare tehnică
1	La inițializare, afișajul trebuie să prezinte un ecran-titlu cu numele autorului	Textul afișat „Indicator rutier adaptiv – Bulgariu Elena-Iuliana”
2	ESP32-C3 trebuie să consume foarte puțin în regim de repaus profund	Nodul intră în mod de economisire și menține consumul minim.
3	O actualizare a afișajului E-Ink să nu consume excesiv de mult	Actualizarea semnului se face rapid, cu consum energetic redus.
4	Eticheta RFID să poată fi citită de la distanțe apreciabile de vehicul	Citire stabilă a semnului de la câțiva metri.
5	Rescrierea informațiilor în tag-ul RFID să funcționeze fără contact direct	Scriere reușită din mers, la o distanță de câțiva metri.
6	Conținutul stocat pe etichetă să includă toate datele necesare vehiculului (ID semn, tip, poziție, orientare)	Structură simplă de date (de ex. JSON mic sau câmpuri binare).

7	Interacțiunea prin BLE între aplicație și nod să fie rapidă	Întregul proces BLE-update-ecran în sub 2 secunde.
8	Nodurile trebuie să rămână funcționale după expunere la apă și zăpadă	Citire și afișare fără erori după condiții de mediu.

Acestea reprezintă cerințele de sistem formulate pe baza arhitecturii. Pentru a verifica în practică îndeplinirea fiecăreia, am definit un set de criterii de testare care vor fi aplicate în Capitolul 4. Tabelul 3.2. de mai jos identifică metodele, condițiile și rezultatele așteptate pentru evaluarea funcțională a nodurilor și a fluxurilor de comunicare.

Tabelul 3.2. Metodele, condițiile și rezultatele așteptate pentru evaluarea funcțională.

Nr.	Ce testăm	Cum măsurăm	Așteptări
1	Conectare BLE	Cronometrare realizare conexiune	< 2 s la 1–5 m
2	Stabilitate BLE	Monitorizare conexiune BLE în aplicație	> 95 % conexiuni reușite sub 7 m
3	Scriere RFID în mers	Vizualizare semn	≥ 90 % reușite la < 3 m/s
4	Unghi citire RFID	Apreciere unghi 0°–90°	Fiabil până la 45°
5	Distanță citire RFID	Citire statică/dinamică a tag-ului	≥ 6 m
6	Autonomie nod	Rulare 1 update/zi + sleep BLE restul	≥ 200 zile
7	RFID în ploaie/zăpadă	Pulverizare cu apă și aşezare zăpadă pe tag	< 20 % scădere performanță
8	Timp scriere semn nou	Cronometre BLE→ESP→e-Paper→RFID	≈ 3 s
9	Notificare eveniment	Simulare accident + notificare primită pe telefon	< 1.5 s
10	Funcționare generală	30 cicluri complete V2I	> 95 % reușită

3.3. Selecția componentelor

Pe baza arhitecturii prezentate în secțiunea anterioară, a cerințelor de sistem formulate, precum și a analizării literaturii de specialitate privind proiecte similare, a fost realizată o selecție a componentelor hardware necesare pentru implementarea sistemului V2I propus. Alegerea finală a ținut cont de compatibilitatea cu ESP32, consumul energetic, capacitațile de comunicare Bluetooth, precum și fiabilitatea în condiții reale de utilizare.

Tabelul 3.3. sintetizează opțiunile analizate și motivele care au condus la decizia finală pentru fiecare bloc funcțional.

Tabelul 3.3. Criterii de alegere a componentelor pentru implementarea sistemului.

Bloc hardware	Componentă aleasă	Justificare tehnică
Microcontroler	ESP32-C3 SuperMini	Integrează BLE 5.0 și protocol ESP-NOW, memorie SRAM 400 kB, curent deep-sleep $\leq 150 \mu\text{A}$ – criteriu-cheie pentru autonomia multi-an pe baterie [38].
Afișaj grafic	E-Ink 2,13" MH-ET Live	Consum nul în static, refresh 20-25 mA < 2 s, lizibil la lumină solară; domeniu temperatură $-20 \dots +60$ °C[33].
Cititor UHF	YRM1003	Putere TX 30 dBm, interfață UART 115 200 bps, suport complet rescriere EPC Gen-2; consum moderat 120 mA în emisie [34].
Etichetă UHF	IN9654 (chip H9, 96 × 22 mm)	Antenă mare pentru câștig ridicat, rază de citire 6-8 m, memorie EPC 128 bit; grosime tag sub 0,3 mm – adezabil direct pe suport [40].
Sursă energie	Baterie externă MLPro 10400mAh	Energie utilă 38 Wh, USB-C, 5 V, $5\text{V} \times 10,4\text{Ah} \approx 52\text{Wh}$ [35].

Capitolul 4. Implementare, dezvoltare și testare

Pornind de la arhitectura descrisă în Capitolul 3 și de la cerințele funcționale (Tabel 3.1.), această secțiune sintetizează cablarea efectivă a componentelor, realizarea conexiunilor și gestionarea energiei pentru sistemul propus. Pentru faza de dezvoltare a schemelor electrice s-au realizat ilustrații ale legăturilor.

4.1. Dezvoltarea la nivel de schemă electrică a nodului de semnalizare

Pornind de la arhitectura descrisă în Capitolul 3 și de la cerințele funcționale (Tabel 3.1.), această secțiune sintetizează cablarea efectivă a componentelor, realizarea conexiunilor și gestionarea energiei pentru sistemul propus. Pentru faza de dezvoltare a schemelor electrice s-au realizat ilustrații ale legăturilor. Tabelul 4.1. prezintă maparea conexiunilor SPI între microcontrolorul ESP32-C3 SuperMini și modulul MH-ET Live e-Paper 2,13". Alegerea magistralei SPI0 (3,3 V) reduce latența de inițializare.

Tabelul 4.1. Maparea conexiunilor SPI între ESP32-C3 și modulul e-Paper 2,13.

Cod culoare (Fig. 4-1)	Pin modul e-Paper	Pin ESP32-C3	Motivație tehnică
roșu	VCC (3 V3)	3 V3	Interval de alimentare
maro	GND	GND	Referință comună
verde	SDI (MOSI)	GPIO 6	Linie date SPI
cyan	SCLK	GPIO 4	Semnal ceas SPI
mov	CS	GPIO 7	Selectare exclusivă a controlerului e-Paper
negru	D/C	GPIO 5	Comută între instrucțiuni și date conform driverului GxEPD2
gri	RESET	GPIO 8	Impuls LOW 10 ms la inițializare, apoi se resetează
galben	BUSY	GPIO 3	Semnal „ocupat”; MCU intră în stare de repaus numai când BUSY = LOW

4.2. Dezvoltarea modulului pentru semnele de circulatie

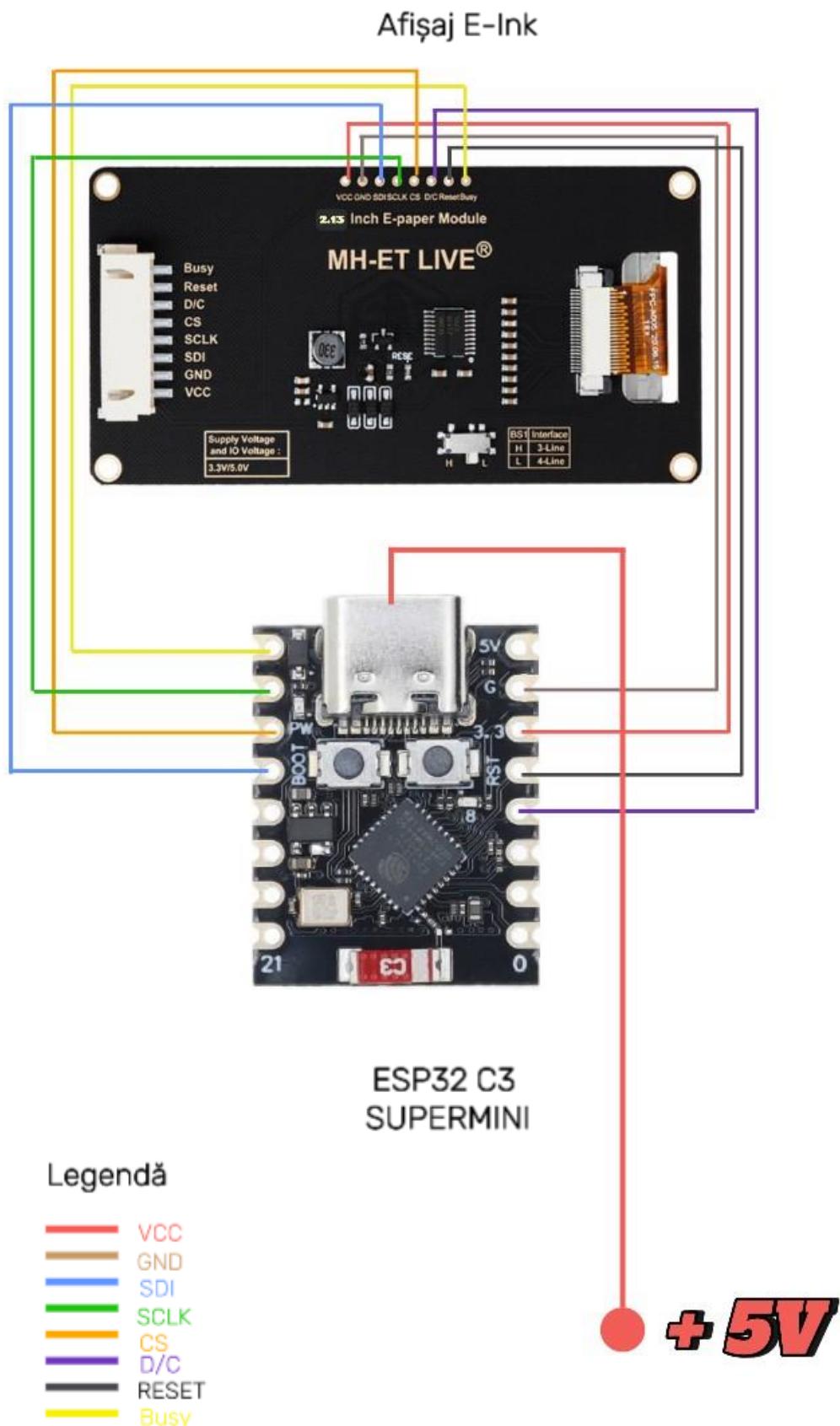


Figura 4.1.Dezvoltarea modulului pentru semnele de circulatie.

Figura 4.1. ilustrează conexiunile hardware dintre microcontrolerul ESP32-C3 SuperMini și modulul de afișaj MH-ET LIVE E-Paper de 2.13 inch, evidențiind integrarea prin magistrala SPI. Fiecare linie colorată reprezintă o conexiune dedicată unei funcții specifice: alimentare (VCC, GND), semnal de ceas (SCLK), date seriale (SDI), control (DC, CS, RST) și status (BUSY). Configurația prezentată este esențială pentru inițializarea corectă și funcționarea stabilă a ecranului E-Ink, oferind o interfață eficientă între afișaj și controler, cu timp redus de răspuns și consum energetic minim în standby. Alegerea acestei topologii de conectare reflectă o abordare optimizată pentru aplicații embedded de semnalizare rutieră, unde fiabilitatea și eficiența energetică sunt esențiale.

4.3. Dezvoltarea modulului RFID UHF

Figura 4.2. ilustrează realizarea unei conexiuni UART între cititorul UHF RFID YRM1003 și un modul de conversie USB–TTL. Această configurație este utilizată pentru conectarea ulterioară directă la un microcontroler precum ESP32-C3.

Modulul YRM1003 utilizează interfața UART (transmisie asincronă full duplex), ce necesită semnalele:

- TXD (Transmit-transmite) – pentru a transmite date de la modulul RFID către microcontroler;
- RXD (Receive-primește) – pentru a primi comenzi din partea microcontrolerului;
- GND – referință de masă comună;
- VCC (5V) – alimentare generală;
- 3V3 (3.3V) – uneori utilizat pentru logică internă sau ca referință.

Această conexiune UART este ideală pentru medii embedded, deoarece permite un schimb eficient de date binare (ex. identificatori EPC, comenzi de scriere/rescriere, setări de putere a antenei etc.).

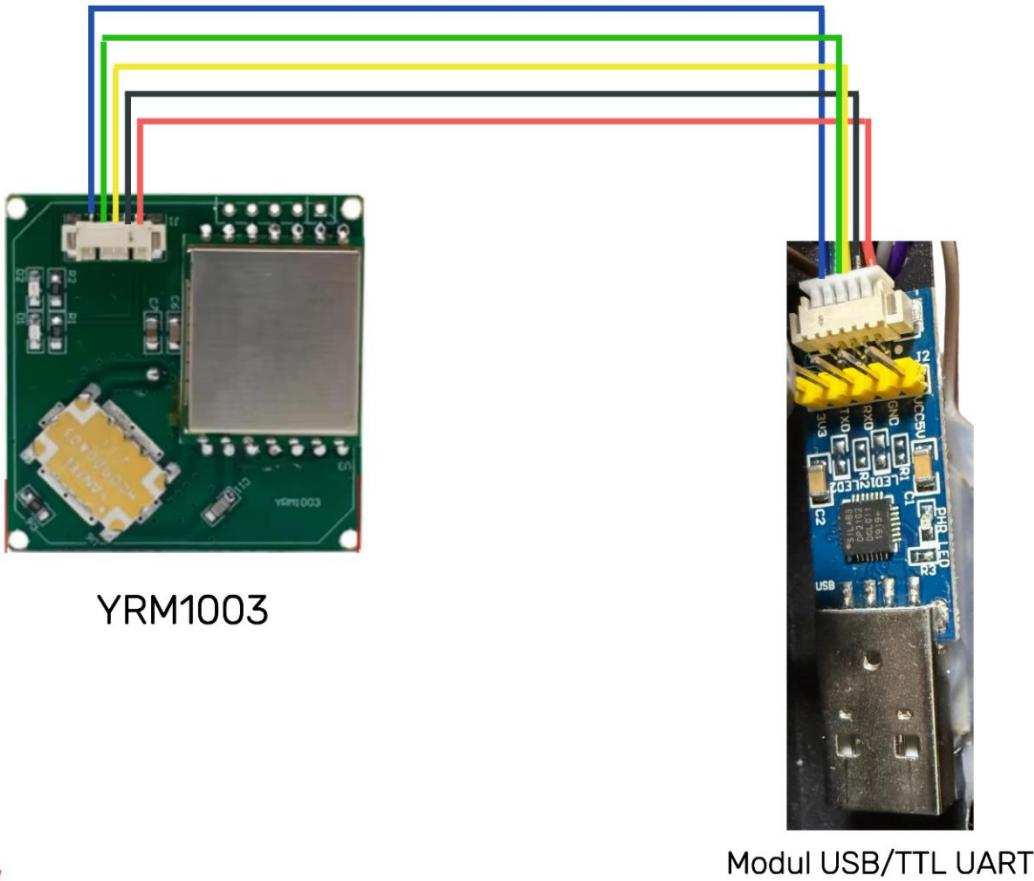


Figura 4.2. Conectarea cititorului UHF RFID YRM1003 la microcontroler ESP32 prin interfață UART.

4.4. Implementarea hardware

După definitivarea design-ului electric, am realizat montajul tehnic conceput din parti distincte: cele două semne de circulație și montarea modulului UHF RFID pe mașina Elysium RC.

4.4.1. Implementarea modulului pentru semne de circulație.

Fiecare semn a fost construit astfel:

1. **Baza** – bateria externă MLPro 10 400 mAh oferă atât stabilitate, cât și alimentarea;
2. **Corpul suport** – un profil din material plastic ce se fixează deasupra bateriei și poartă, pe partea frontală, eticheta UHF IN9654 plasată în zona optimă pentru citire;
3. **Partea superioară**– afișaj E-Ink de 2,13" lipit cu silicon transparent pe muchia superioară a suportului, iar pe spatele lui este montat, tot cu adeziv, microcontrolerul ESP32 SuperMini.

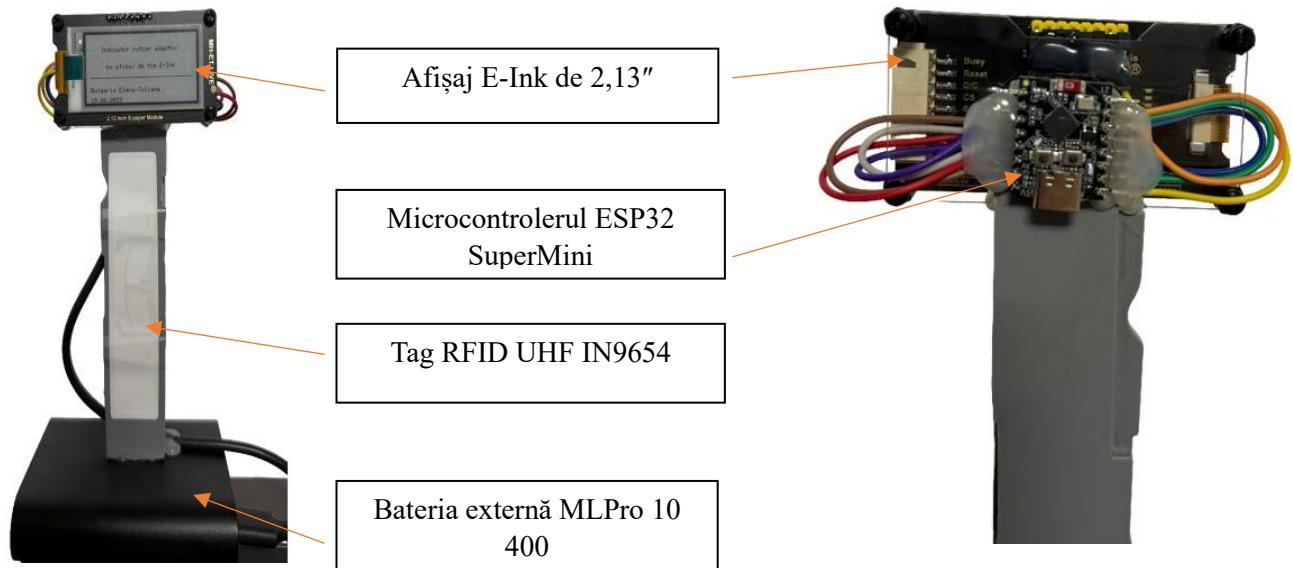


Fig 4.3. Implementarea modulului pentru semne de circulație.

4.4.2. Implementarea modulului RFID UHF pe vehiculul Elysium RC

Pentru a oferi vehiculului capacitatea de citire/ scriere/rescriere a tag-urilor, pe partea frontală Elysium RC a fost integrat un ansamblu electronic UHF RFID YRM1003 care este format din antena UHF care este doar elementul pasiv care radiază și recepționează, cititor / scriitor UHF RFID care reprezintă ansamblul electronic care generează semnalul de 30 dBm, aplică protocolul EPC Gen-2, interpretează răspunsul tag-ului și, dacă este cazul, scrie date noi. Acest ansamblu cititor/scriitor UHF RFID YRM1003 conectat la o antenă patch UHF de 6 dBi asigură implementare unei părți din fluxul V2I: vehiculul detectează semnul prin RFID, poate modifica conținutul tag-ului și ulterior modifica conținutul acestuia.

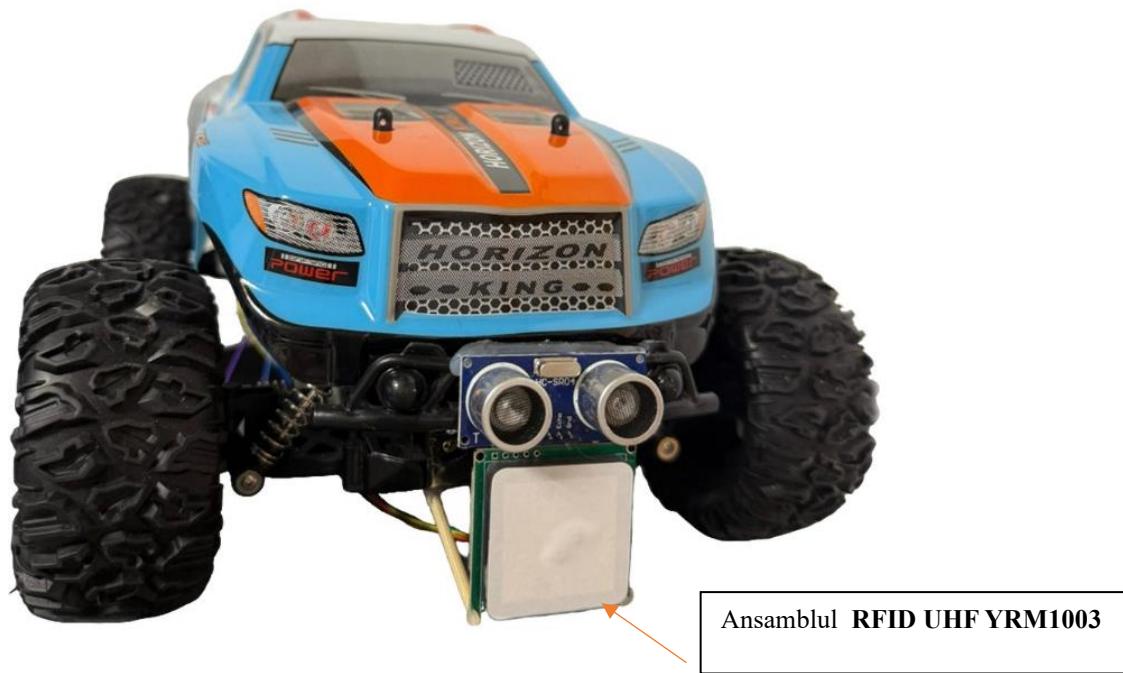


Figura 4.4. Implementarea modulului RFID UHF pe vehiculul Elysium RC.

4.5. Aplicație mobiă-Interacțiune mobilă în timp real cu infrastructura rutieră

4.5.1. Fluxul operațional și procesul de interacțiune cu sistemul

Pentru a asigura o interacțiune eficientă între utilizator și infrastructura inteligentă de semnalizare rutieră, a fost dezvoltată o aplicație mobilă compatibilă cu Android, care utilizează protocolul Bluetooth Low Energy (BLE) pentru comunicarea cu nodurile ESP32-C3 ale semnelor de circulație. Interfața grafică a aplicației este simplificată, dar funcțională, permitând scanarea și conectarea rapidă la toate dispozitivele active din proximitate.

Secțiunea de față examinează, simultan conceptual și modul în care aplicația mobilă „Smart Traffic Signs” a fost implementată. De asemenea, secțiunea va prezenta și inventariază dispozitivele Bluetooth relevante:

- semnele de circulație inteligente (Bluetooth Low Energy)
- vehiculul Elysium RC (Bluetooth Classic–Serial Port Profile).

Analiza urmărește trei întrebări: cine inițiază operația, ce pași interni se execută și cum sunt transmise rezultatele spre interfața grafică.

Inițierea revine telefonului mobil, imediat după obținerea permisiunilor de localizare și pornirea hardware-ului Bluetooth. Un ViewModel, menținut de un serviciu de prim-plan, pornește două scannere specializate — BleScanner și ClassicScanner — fiecare rulând într-un coroutine scope propriu, pentru ca blocările sau excepțiile de pe un canal să nu perturbe celălalt.

Diagrama de flux prezentată în Figura 4.5. descrie succesiunea logică a operațiilor efectuate de nodul intelligent de semnalizare rutieră, de la pornirea sistemului până la actualizarea semnului afișat. După alimentare, sistemul inițializează componentele principale – afișajul E-Ink, comunicația BLE și protocolul ESP-NOW – și intră în mod de așteptare cu consum redus, monitorizând permanent sursele de date. Dacă sunt recepționate comenzi, acestea sunt analizate în funcție de proveniență: aplicația Android (prin BLE) sau modulul vehiculului Elysium RC (prin ESP-NOW). În funcție de context, semnul rutier este actualizat, iar aplicația primește notificări corespunzătoare. În caz de eșec în inițializare sau transmitere, sistemul afișează mesaje de eroare și revine în stare de standby. Acest flux asigură reacție rapidă, adaptabilitate și funcționare autonomă eficientă energetic.

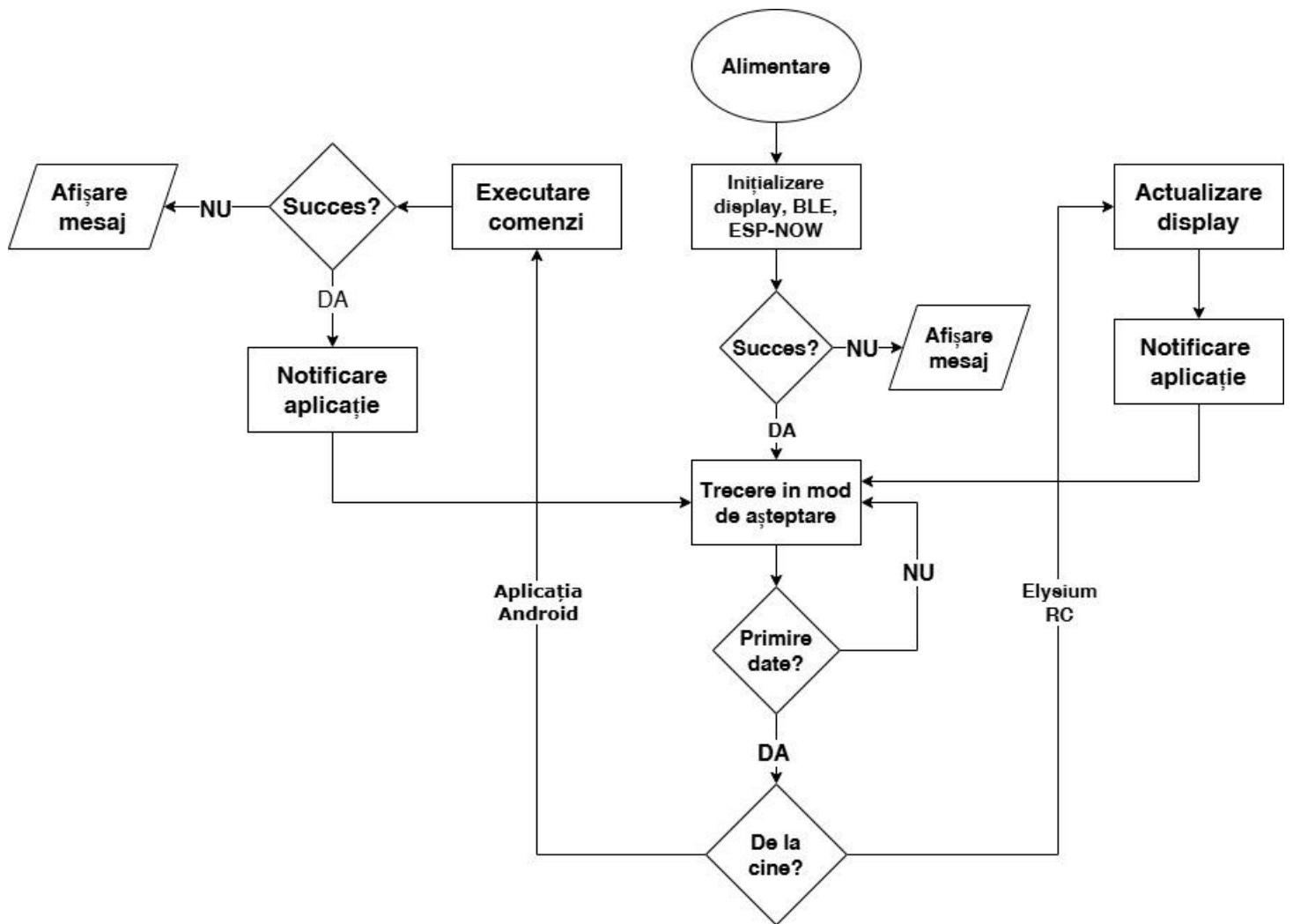


Figura 4.5. Diagrama de flux a procesului de funcționare al unui indicator rutier.

Diagramă de flux din Figura 4.6. detaliază logica de funcționare a aplicației Android utilizate pentru controlul semnelor rutiere. Procesul începe cu solicitarea permisiunilor necesare pentru funcționarea aplicației (acces la Bluetooth, locație etc.). Dacă utilizatorul nu le acordă, aplicația afișează un mesaj de eroare și se închide. În caz contrar, aplicația continuă cu scanarea dispozitivelor BLE din proximitate și afișarea unei liste actualizate cu acestea. După selectarea unui dispozitiv (semn rutier sau vehicul Elysium RC), utilizatorul este invitat să aleagă o comandă, cum ar fi modificarea semnului afișat. În funcție de destinația comenzi (indicator rutier sau vehicul), aplicația stabilește conexiunea și trimită instrucțiunile. Dacă scrierea pe tagul RFID sau actualizarea indicatorului eșuează, este afișat un mesaj de eroare; în caz de succes, interfața este actualizată în timp real pentru a reflecta noua stare a sistemului. Diagrama evidențiază atât gestionarea robustă a erorilor, cât și interacțiunea eficientă om-dispozitiv.

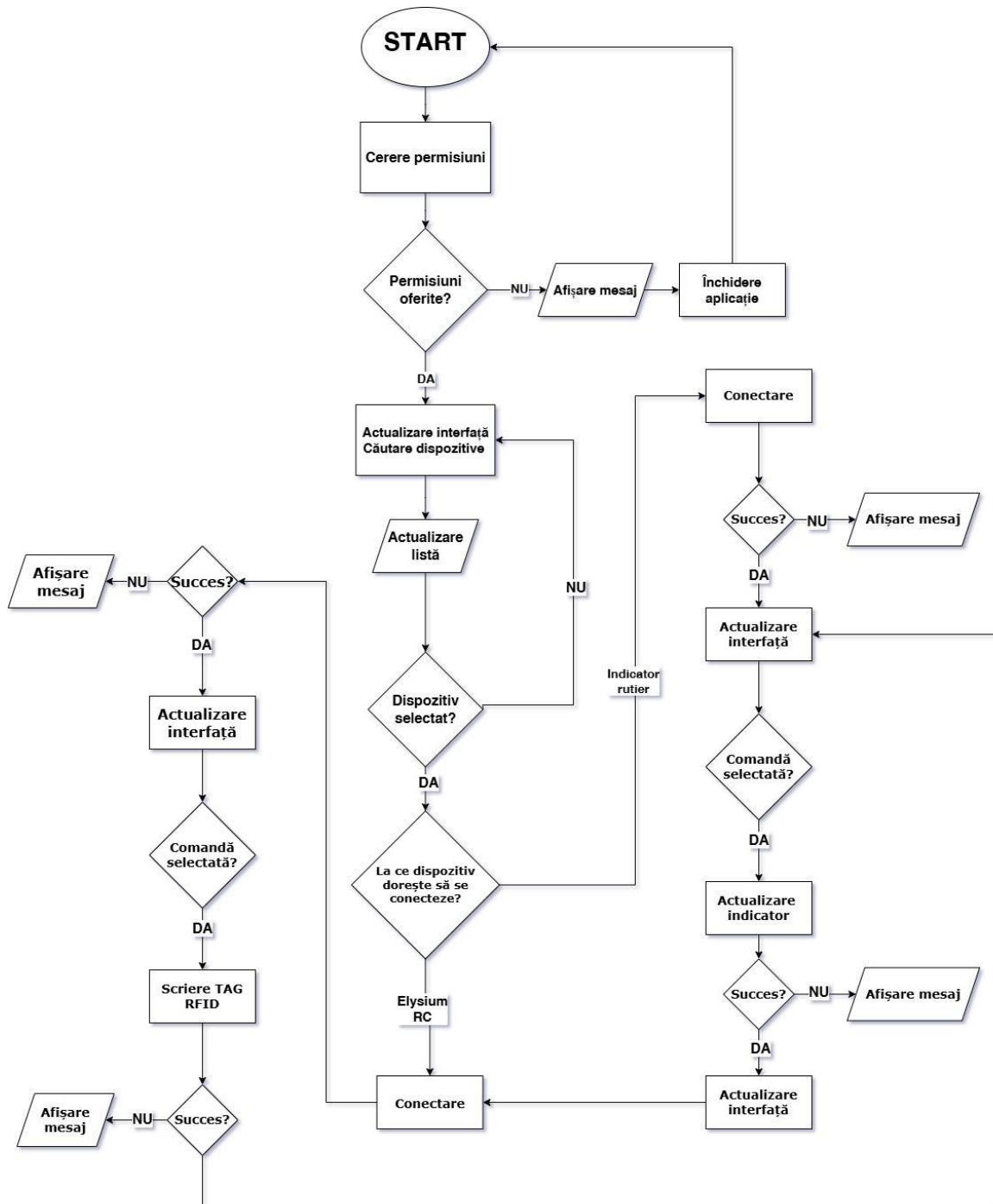


Figura 4.6. Diagrama de flux a aplicației mobile pentru gestionarea semnelor de circulație.

Algoritmul intern variază în funcție de protocol. BleScanner pornește o scanare filtrată pe UUID-ul serviciului GATT al semnului, citește pachetele advertisement și extrage metadatele necesare: identitatea semnului, nivelul bateriei, revizia firmware-ului. ClassicScanner parcurge întâi lista de dispozitive împerecheate, apoi inițiază o operație inquiry; pentru fiecare rezultat apelează asincron fetchUuidsWithSdp, validând prezența UUID-ului 00001101-0000-1000-8000-00805F9B34FB. Răspunsurile întârziate sunt corelate cu evenimentele de descoperire printr-un registru temporar, prevenind raportările duplicate.

Datele brute sunt normalizate în structuri DeviceUi printr-un mapator comun care adaugă atribută vizuale — pictogramă, culoare tematică, text de stare. Cele două fluxuri reactive sunt combinate într-un StateFlow unificat, expus read-only stratului de prezentare. DeviceListScreen consumă acest flux prin Jetpack Compose și re-compune lista de carduri în mai puțin de 200 ms după apariția unui nou dispozitiv, oferind impresia unei interfețe „live” fără fire suplimentare de sincronizare.

Pentru a limita consumul energetic, ambele scannere aplică duty-cycling: ferestre de lucru de 10 s urmate de pauze de 20 s; coroutines sunt suspendate non-blocant în perioadele de repaus. La trecerea aplicației în background, scanarea este opriță conform politicilor Android Doze și reluată automat la revenirea în prim-plan.

Prin delegarea clară a responsabilităților și adoptarea unui model reactiv, sistemul asigură descoperirea rapidă a dispozitivelor, filtrarea strictă a rezultatelor și un consum redus de energie. Arhitectura rămâne extensibilă: adăugarea unui nou tip de dispozitiv — de pildă un semafor inteligent pe Thread sau Wi-Fi Aware — presupune doar implementarea unui scanner care respectă același contract StateFlow, fără modificări în stratul UI.

4.5.2. Premise și restricții de platformă

Începând cu versiunea 6 a sistemului de operare Android (API 23), accesul la lista dispozitivelor Bluetooth din proximitate este supus unui regim de permisiuni explicite, atât pentru Bluetooth, cât și pentru Locație. Rațiunea introducerii acestui dublu control derivă din posibilitatea ca balizele BLE să fie utilizate, direct sau indirect, pentru pseudo-geolocalizare. În consecință, orice tentativă de scanare este precedată de un pas obligatoriu de validare a drepturilor de acces. În cadrul aplicației, acesta este implementat în MainActivity prin metoda requestPermissions(...); lipsa oricărei permisiuni constituie o condiție de oprire și suspendă inițierea oricărei operații radio.

Logica internă a procesului se sprijină pe trei actori principali:

- **BleScanner** – componentă dedicată scanării Bluetooth Low Energy, capabilă să filtreze rapid rezultatele pe baza UUID-ului de serviciu relevant pentru semnele de circulație inteligente.
- **ClassicScanner** – modul responsabil de descoperirea dispozitivelor care expun profilul Serial Port Profile (SPP), necesar conectării la vehiculul Elysium RC și programării tagurilor RFID.
- **MainActivity** – orchestratorul care pornește și oprește scannerele, agregă fluxurile de rezultate și le livrează stratului de prezentare construit cu Jetpack Compose.

4.5.3. Scanarea BLE pentru semnele de circulație

După acordarea permisiunilor pentru Bluetooth și localizare, aplicația declanșează imediat secvența de scanare Bluetooth Low Energy menită să identifice semnele de circulație inteligente. Primul pas constă în obținerea instanței de sistem - BluetoothLeScanner –, urmat de definirea unui ScanFilter care conține identificatorul unic al serviciului GATT publicat de semne. Acest identificator – un UUID de 128 de biți, reprezentat textual în hexadecimal și început, în implementarea curentă, cu secvența „8F0E...0101” – delimită fără echivoc serviciul „Smart Traffic Sign” în cadrul ecosistemului BLE, unde există zeci de mii de servicii concurente. Modelul GATT (Generic Attribute Profile) poate fi privit drept un registru logic de atribute, iar serviciul specific semnului reprezintă „dosarul” său de date accesibile extern.

După configurarea filtrului, invocarea metodei startScan instruiește platforma să ignore pachetele publicitate care nu conțin UUID-ul respectiv. Prin această selecție la nivelul radio, telefonul procesează exclusiv anunțurile emise de semne, diminuând atât sarcina asupra procesorului, cât și consumul energetic al antenei. Pentru fiecare pachet relevant interceptat, Android generează apelul onScanResult, furnizând un obiect BluetoothDevice ce include adresa MAC, intensitatea semnalului și numele transmis de dispozitiv.

La nivel de aplicație, aceste date brute sunt imediat transformate într-o structură de prezentare BleDeviceUi, apoi adăugate într-un MutableStateFlow. Caracterul reactiv al fluxului face ca orice modificare – apariția unui semn sau variația puterii semnalului – să fie publicată instantaneu, iar componente Jetpack Compose responsabile de listă să se re-rendereze fără cod suplimentar de sincronizare. Ca urmare, la cel mult jumătate de secundă după ce un semn intră în raza de acțiune, utilizatorul îl vede deja pe ecran, cu nivelul RSSI actualizat în timp real.

Filtrarea strictă pe UUID oferă, pe lângă eficiență energetică, avantaje de securitate și scalabilitate. Semnele sunt izolate de aglomerația dispozitivelor personale din mediul urban, iar introducerea unui nou tip de indicator rutier presupune doar publicarea unui nou serviciu GATT și ajustarea valorilor constante din aplicație, fără modificări de arhitectură sau de logică fundamentală.

Fragmentul de cod din Figura 4.7. ilustrează toate elementele discutate anterior: filtrarea pe UUID, setarea modului de consum redus, utilizarea unui StateFlow pentru propagarea rezultatelor și tratarea erorilor de scanare:

```
// UUID publicat în firmware-ul semnului: 8f0e0000-0000-1000-8000-000000000101
private val filter = ScanFilter.Builder()
    .setServiceUuid(
        ParcelUuid(UUID.fromString("8f0e0000-0000-1000-8000-000000000101"))
    )
    .build()

private val settings = ScanSettings.Builder()
    .setScanMode(ScanSettings.SCAN_MODE_LOW_POWER) // duty-cycling
    .build()

private val _devices = MutableStateFlow<List<BluetoothDevice>>(emptyList())
val devices: StateFlow<List<BluetoothDevice>> = _devices

fun startBleScan() {
    bluetoothLeScanner.startScan(listOf(filter), settings, object : ScanCallback() {

        override fun onScanResult(type: Int, result: ScanResult) {
            val device = result.device ?: return
            // evităm dublurile; StateFlow notifică automat UI-ul
            if (_devices.value.none { it.address == device.address }) {
                _devices.value = _devices.value + device
            }
        }

        override fun onScanFailed(errorCode: Int) {
            Log.e("BleScan", "Eroare scanare: $errorCode")
        }
    })
}
```

Figura 4.7. Fragment de cod pentru inițierea scanării BLE cu filtrare pe UUID în aplicația Android.

4.5.4. Scanarea Bluetooth Classic pentru vehiculul Elysium RC

Scanarea vehiculului Elysium RC se desfășoară într-un regim distinct față de cea pentru semnele BLE, deoarece platforma Android nu permite filtrarea directă după UUID în faza de inquiry BR/EDR. Aplicația adoptă, prin urmare, o abordare în două trepte: mai întâi descoperă nediscriminatoriu toate dispozitivele Bluetooth-Classic aflate în proximitate, apoi aplică filtrarea logică în spațiul aplicației.

Inițiativa revine clasei `ClassicScanner`, instanțiată de `MainActivity` imediat după ce utilizatorul a acordat permisiunea de Bluetooth. Metoda `startScan` golește cache-ul intern, sincronizează un marker de timp pentru măsurarea duratei operațiunii și invocă `BluetoothAdapter.startDiscovery()`. În intervalul inquiry (aproximativ 12 s), adaptorul radio scanează canalele BR/EDR, iar sistemul emite evenimentele difuzate `BluetoothDevice.ACTION_FOUND`. `ClassicScanner` se abonează la aceste evenimente printr-un `BroadcastReceiver`; fiecare obiect `BluetoothDevice` primit este introdus într-un `MutableStateFlow` doar dacă nu este deja prezent, evitând duplicările și păstrând colecția ordonată cronologic.

Filtrarea propriu-zisă are loc ulterior, printr-o rezoluție asincronă a serviciilor. Imediat după adăugarea unui dispozitiv nou (sau preluarea celor deja împerecheate), aplicația lansează `device.fetchUuidsWithSdp()`. Sistemul răspunde prin evenimentul `BluetoothDevice.ACTION_UUID`, care conține lista serviciilor expuse de respectivul echipament. Dacă printre acestea figurează UUID-ul standard `00001101-0000-1000-8000-00805F9B34FB` — identificatorul profilului Serial Port Profile — dispozitivul este etichetat drept „compatibil Elysium RC” și rămâne în colecție; în absența lui, intrarea este eliminată. Această etapă suplimentară compensează limitarea API-ului, permitând aplicației să distingă vehiculul-țintă într-un mediu aglomerat de căști, boxe sau laptopuri.

Lista curată este expusă stratului de prezentare printr-un StateFlow read-only. Componentele Jetpack Compose se abonează la acest flux și re-renderează instantaneu, astfel încât utilizatorul vede vehiculul doar atunci când acesta publică profilul SPP necesar modului „Tag Write”. Experiența este simplificată: nu se afișează dispozitive irelevante, iar încercările de conectare greșită sunt prevenite din design.

În fundal, `ClassicScanner` respectă aceleași principii de economie energetică folosite de scanner-ul BLE: duty-cycling adaptiv, suspendarea corutinelor în pauzele radio și oprirea completă în scenariile de background impuse de Android Doze. Prin această orchestrare,

aplicația combină acuratețea detecției cu un consum redus de resurse, păstrând în același timp un cod ușor de extins pentru eventualele protocoale viitoare.

```
/** Pornește o scanare BR/EDR generală */
@SuppressLint("MissingPermission")
dis fun startScan() {
    adapter ?: return
    if (adapter.isDiscovering) adapter.cancelDiscovery() // reset
    _devices.value = emptyList() // curățăm lista
    adapter.startDiscovery() // inițiem inquiry
}

/** BroadcastReceiver-ul recepționează fiecare dispozitiv găsit */
private val receiver = object : BroadcastReceiver() {
    @SuppressLint("MissingPermission")
    override fun onReceive(ctx: Context, intent: Intent) {
        if (intent.action == BluetoothDevice.ACTION_FOUND) {
            val device: BluetoothDevice? =
                intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE)
            device?.let { addDevice(it) } // adăugăm în StateFlow
            device?.fetchUuidsWithSdp() // declanșăm interogarea SDP
        }
    }
}
```

Figura 4.8. Fragment de cod pentru Scanarea Bluetooth Classic pentru vehiculul Elysium RC.

4.5.5. Agregarea și prezentarea rezultatelor

După ce cele două scannere finalizează detecția – bleScanner pentru semnele inteligente și classicScanner pentru vehiculul Elysium RC – rezultatele lor sunt preluate de stratul de prezentare gestionat de MainActivity. Concret, funcția componibilă BleContent() colectează fiecare flux prin collectAsState(), punând astfel Compose într-o relație de actualizare directă: de îndată ce unul dintre scannere adaugă sau elimină un dispozitiv, interfața primește automat o nouă instanță de listă.

Fuziunea propriu-zisă are loc într-un bloc remember { ... }, pentru a evita calcule inutile. Algoritmul pornește de la colecția semnelor BLE, apoi inserează dispozitivele clasice care nu figurează deja, deduplicând pe baza adresei MAC. La final, lista este ordonată alfabetic după denumirea dispozitivului, producând un set coerent destinat ecranului de selecție. Fiecare element este transmis către DeviceListScreen, unde se materializează ca un card ce afișează numele, tipul de conexiune și butonul „Connect”.

Acest lanț reactiv elimină necesitatea unor mecanisme de refresh manual; un dispozitiv nou devine vizibil în mai puțin de două sute de milisecunde, conferind aplicației perceptia unei interfețe „live”. Fragmentul corespunzător din MainActivity.kt confirmă atât eleganța, cât și

economia soluției: StateFlow-urile sunt surse unificate de adevăr, Compose gestionează randarea incrementală.

Fragment relevant din codul proiectului (fișier MainActivity.kt, funcția BleContent) care ilustrează întregul proces:

```
// 1. Colectăm fluxurile provenite de la cele două scanere
val bleDevices by bleScanner.devices.collectAsState()
val classicDevices by classicScanner.devices.collectAsState()

// 2. Fuzionăm liste, eliminăm dublurile și sortăm
val devicesList = remember(bleDevices, classicDevices) {
    val merged = bleDevices.toMutableList()
    classicDevices.forEach { dev ->
        if (merged.none { it.address == dev.address }) {
            merged.add(dev) // adăugăm numai dacă nu există
        }
    }
    merged.sortedBy { it.name ?: it.address } // ordonăm pentru lizibilitate
}

// 3. Trasmitem lista către ecranul de prezentare
DeviceListScreen(
    devices = devicesList,
    connectedMacs = connections.keys,
    onRefresh = { bleScanner.startScan(); classicScanner.startScan() },
    onDeviceSelected = { device -> /* logica de conectare */ }
)
```

Figura 4.9. Fragment de cod pentru agregarea și prezentarea rezultatelor.

După faza de descoperire și selecție, aplicația preia rolul de nod de comandă al întregului ecosistem, activitatea sa putând fi urmărită pe trei direcții: menținerea conexiunilor radio, schimbul de comenzi cu semnele de circulație și dialogul particular cu vehiculul Elysium RC.

Apăsarea butonului «Connect» declanșează, în cadrul componentei MainActivity, instanțierea fie a unui obiect BleConnection, fie a unui obiect SppConnection, în funcție de tipul dispozitivului ales. Ambele clase implementează interfața DeviceConnection, permitând restului aplicației să rămână ignorant la detaliile de protocol. Fiecare obiect expune două fluxuri reactive de tip StateFlow: unul semnalează starea conexiunii (CONNECTING → CONNECTED → DISCONNECTED), celălalt transportă mesaje text provenite de la dispozitiv.

Pentru un semn de circulație inteligent, BleConnection creează sesiunea GATT, descoperă caracteristicile COMMAND și STATUS, apoi se abonează la notificări pe cea din urmă. În momentul în care utilizatorul solicită o acțiune – de pildă „Schimbă pictograma” sau „Reset Accident” – metoda sendCommand scrie pur și simplu sirul corespunzător în caracteristica COMMAND. Semnul răspunde cu un ACK „OK” ori „ERROR”; textul este

preluat în callback-ul onCharacteristicChanged, propagat fluxului statusMessage și afișat imediat pe ecran.

Vehiculul Elysium RC utilizează canalul Serial Port Profile, motiv pentru care SppConnection deschide un socket RFCOMM. O corutină de fundal citește fluxul de intrare linie cu linie și publică rezultatele în statusMessage. sendCommand adaugă marcajul de sfârșit de linie „\n”, transmite instrucțiunea și așteaptă confirmarea.

Cea mai complexă secvență operațională este „Tag Write”. Când utilizatorul activează această opțiune, aplicația expediază vehiculului comanda «ENTER_TAG_WRITE_MODE». La confirmarea «READY», interfața afișează un dialog în care se introduce conținutul noului tag RFID; textul se trimită mai departe, iar sistemul rămâne blocat până la primirea unui răspuns «OK» sau «ERROR», interval în care butoanele sunt dezactivate pentru a preveni trimiterea multiplă.

În paralel, un serviciu de fundal – BleBackgroundService – susține activ conexiunile și monitorizează fluxurile de stare. Dacă se detectază cuvântul-cheie „ACCIDENT”, serviciul lansează imediat o notificare de prioritate ridicată, afișează un banner roșu și poate declanșa vibrația dispozitivului. Front-end-ul Compose nu necesită intervenții suplimentare: simpla colectare a fluxului statusMessage este suficientă pentru actualizarea automată a componentelor.

Prin unificarea acestor comportamente într-o arhitectură reactivă, aplicația oferă o experiență „vie”: tranzițiile de stare ale dispozitivelor se reflectă pe ecran fără ca utilizatorul să percepă momentul transmisiei radio, iar vehiculul confirmă ori refuză scrierea RFID-ului aproape în timp real. Extinderea către dispozitive viitoare rămâne facilă: tot ce se cere este implementarea interfeței DeviceConnection – adică un flux de stare, un flux de mesaje și funcția sendCommand.

```

    override fun sendCommand(cmd: String): Boolean {
        val ch = commandCharacteristic ?: return false           // caracteristica GATT de scriere
        ch.setValue(cmd.toByteArray(Charsets.UTF_8))
        return bluetoothGatt?.writeCharacteristic(ch) ?: false // declanșează operația GATT
    }

    private val gattCallback = object : BluetoothGattCallback() {
        override fun onCharacteristicChanged(gatt: BluetoothGatt, characteristic: BluetoothGattCharacteristic) {
            if (characteristic.uuid == STATUS_UUID) {
                val msg = characteristic.getStringValue(0)          // text "OK", "ERROR", "ACCIDENT"...
                _statusMessage.value = msg                          // flux colectat de UI + serviciu
            }
        }
    }

```

Figura 4.10. Fragment de cod ilustrativ pentru schimbul efectiv de comenzi cu un semn.

Secțiunea echivalentă pentru vehicul:

```
fun connect(device: BluetoothDevice) = scope.launch {
    socket = device.createRfcommSocketToServiceRecord(SPP_UUID)
    socket?.connect()                                // handshake RFCOMM
    ioJob = launch(Dispatchers.IO) {                  // reader în thread separat
        BufferedReader(InputStreamReader(socket!!.inputStream)).use { reader ->
            reader.forEachLine { line ->
                _statusMessage.value = line.trim()      // valorile ajung direct în UI
            }
        }
    }
}

override fun sendCommand(cmd: String): Boolean =
    try {
        socket?.outputStream?.apply {
            write("$cmd\n".toByteArray())           // terminator LF pentru vehicul
            flush()
        }
        true
    } catch (e: IOException) { false }
```

Figura 4.11. Fragment de cod ilustrativ pentru schimbul efectiv de comenzi cu un semn- Secțiunea echivalentă pentru vehicul.

Prin aceste două fragmente se observă clar cum aplicația traduce gesturile utilizatorului în pachete Bluetooth și cum transformă răspunsurile firmware-ului în informații de interfață, îndeplinindu-și astfel rolul de „punte” între infrastructura fizică a sistemului și experiența de control la nivel de utilizator.

4.5.6. Arhitectura aplicației și repartizarea responsabilităților

Aplicația „Smart Traffic Signs” funcționează ca un nod central de comandă care, peste infrastructura Bluetooth oferită de Android, ridică un set clar de componente specializate, ușor de urmărit și de extins.

MainActivity orchestreză întregul ciclu de viață: validează permisiunile, pornește scannerele și creează, în funcție de dispozitivul ales, conexiunea adecvată. În același timp, agregă starea tuturor componentelor și o propagă interfeței Compose.

BleScanner și ClassicScanner rezolvă problema descoperirii. Primul filtrează pachetele advertisement în funcție de UUID-ul serviciului GATT al semnului, al doilea declanșează

inquiry BR/EDR și verifică, prin SDP, existența profilului Serial Port Profile. Rezultatele lor sunt publicate prin StateFlow, ceea ce face ca interfața să afișeze aproape instantaneu orice dispozitiv relevant.

BleConnection și SppConnection stabilesc modul de comunicare: prin GATT, respectiv prin RFCOMM. Ambele expun fluxul de stare, fluxul de mesaje și funcția sendCommand, astfel încât restul aplicației nu depinde de detaliile protocolului.

BleBackgroundService menține conexiunile în viață și monitorizează mesajele critice – de exemplu, semnalul „ACCIDENT”. La detectarea unui astfel de eveniment, lansează o notificare de sistem, independent de ciclul de viață al activității principale.

Stratul de prezentare, construit cu Jetpack Compose, reacționează direct la aceste fluxuri: adaugă imediat dispozitivele nou apărute, actualizează starea conexiunilor în timp real și blochează temporar elementele de control pe durata operațiilor sensibile, precum Tag Write sau Reset Accident.

Prin această repartizare a responsabilităților, aplicația răspunde coherent la întrebările fundamentale: cine declanșează acțiunea (utilizatorul, orchestrat de MainActivity), ce trebuie executat (descoperire, conectare, schimb de comenzi și notificare) și cum se realizează (componente autonome legate prin fluxuri reactive). Rezultatul este un sistem ușor de întreținut, eficient din punct de vedere energetic și pregătit pentru scalări ulterioare: adăugarea unui nou dispozitiv sau protocol nu implică decât implementarea unui scanner și a unei conexiuni conforme cu interfața curentă, fără a perturba restul arhitecturii.

4.5.7. Semnul rutier adaptiv

Semnul rutier adaptiv este gândit ca un nod periferic inteligent din ecosistemul „Smart Traffic Signs”. Hardware-ul se bazează pe un micro-controler XIAO ESP32-C3, cu nucleu RISC-V și conectivitate duală Wi-Fi / BLE, la care se adaugă un ecran E-Ink de 2,13" cu consum ultrascăzut. Împreună, aceste elemente permit afișarea dinamică a pictogramelor „STOP”, „ACCIDENT” sau a limitelor de viteză, precum și actualizarea textelor contextuale la cerere. Semnul funcționează în două scenarii complementare. În regim asistat, smartphone-ul Android transmite comenzi prin Bluetooth Low Energy, iar afișajul se reîmprospătează aproape instantaneu; confirmarea reușitei se întoarce înapoi către aplicație prin evenimente GATT, fără fire suplimentare de sincronizare. În regim autonom, dispozitivul ascultă permanent canalul ESP-NOW și reacționează la pachetele emise de vehiculul Elysium RC sau de alte semne,

declanșând imediat modificarea pictogramei și, dacă este cazul, retransmitând alerta către vecini pentru a extinde aria de acoperire.

Pentru a susține această versatilitate, firmware-ul urmează o arhitectură în straturi. Componentele de nivel inferior se ocupă strict de transport. BleManager configerează serverul GATT, expune caracteristici de tip „COMMAND” și „STATUS” și administrează sirul de stări CONNECTING, CONNECTED, DISCONNECTED, inclusiv time-out-urile de siguranță. Deasupra radio-ului 802.11, TrafficAlertReceiver initializează stiva ESP-NOW, filtrează broadcast-urile după identificatorul de aplicație și convertește mesajele în comenzi interne. Interfața grafică este unificată de DisplayManager, care știe să genereze forme vectoriale optimizate energetic — de la triunghiul roșu al semnului de „STOP” până la cifrele segmentate ale unei limite de viteză — și coordonează activarea diferențială a pixelilor, astfel încât o actualizare tipică să nu depășească câteva zeci de milajouli.

Toate aceste module cooperează printr-un mecanism public-subscribe bazat pe queue-uri FreeRTOS, ceea ce garantează separația strictă între logică de transport, decizie și afișare. Securitatea comunicațiilor este asigurată prin pairing BLE cu cheie statică și prin autentificarea codificată a pachetelor ESP-NOW; numai cadrele semnate cu nonce-ul valid sunt prelucrate. Pe linie energetică, micro-controlerul rămâne în modul light-sleep, iar radio-ul se trezește ciclic la 200 ms pentru a nu rata ferestrele de advertising, menținând un buget total sub 25 µA în standby. Actualizările OTA sunt de asemenea activate, permitând distribuția unui nou pachet de firmware către toate semnele dintr-o intersecție în mai puțin de un minut.

Prin această organizare modulară, semnul rutier adaptiv devine nu doar un afișaj pasiv, ci un participant activ la rețeaua de siguranță rutieră: comunică rapid cu aplicația de management atunci când operatorul uman dorește o schimbare, dar rămâne capabil să reacționeze autonom la evenimente neașteptate și să le disemineze către restul infrastructurii. Extinderea către noi funcții — de exemplu integrarea unui senzor de lumină pentru reglarea automată a contrastului E-Ink sau adăugarea unui buzzer piezoelectric pentru avertizare sonoră — se rezumă la înregistrarea unei noi comenzi în tabelul GATT și la includerea unei rutine de redare în DisplayManager, fără modificări masive de schemă electrică sau de protocoale existente.

4.5.8. BleManager – rol și funcționare

În arhitectura internă a semnului rutier adaptiv, componenta **BleManager** acționează ca interfață între radio-ul Bluetooth Low Energy și logica de afișare. Inițiativa aparține, pe de o parte, aplicației Android, care deschide sesiunea GATT și serie instrucțiuni textuale în caracteristica COMMAND; pe de altă parte, BleManager însuși reacționează la toate evenimentele de conectare și deconectare, validează operațiile de scriere și propagă confirmări către telefon. Astfel, fluxul de control rămâne bidirectional, iar schimbul de informații se produce fără tempi morți perceptibili.

Din punct de vedere funcțional, BleManager pornește stiva BLE a semnului, inițiază advertising-ul sub **DEVICE_NAME** și expune un serviciu unic, identificat prin **TRAFFIC_SIGN_SERVICE_UUID**. În interiorul acestuia se găsesc două caracteristici:

- **SIGN_CHARACTERISTIC_UUID**, acceptă numai operații de tip WRITE și primește comenzi precum „STOP” sau „SPEED_LIMIT_60”;
- **STATUS_CHARACTERISTIC_UUID**, poate fi citită sau notificată și transmite mesaje de confirmare de forma „Connected” ori „Sign updated: ...”.

Administrarea atentă a stării de conectare previne scrierile inutile și repornește advertising-ul la câteva sute de milisecunde după pierderea legăturii, asigurând disponibilitatea permanentă a dispozitivului.

Implementarea urmează un traseu clar definite. Constructorul primește un pointer către **DisplayManager**, stabilind puntea către stratul de prezentare grafică. Metoda *init()* construiește serverul GATT, creează serviciul și caracteristicile, pornește advertising-ul și înscrie mesajul „Ready” în caracteristica STATUS, permitând aplicației să detecteze pornirea corectă a semnului. În *onConnect()* și *onDisconnect()*, un flag intern marchează prezența clientului și declanșează, la nevoie, reluarea advertising-ului. Când sosesc date în *onWrite()*, UUID-ul caracteristicii este verificat, iar sirul ASCII provenit de la telefon este trimis către *DisplayManager::showTrafficSign()*, care actualizează ecranul E-Ink. În fine, metoda *sendStatusUpdate()* expediază confirmări doar atunci când un client este efectiv conectat, evitând emisia radio inutilă și conservând energie.

4.5.9. DisplayManager

DisplayManager reprezintă stratul grafic al semnului rutier adaptiv, aducând la același nivel controlul hardware-ului E-Ink și logica de randare. Componenta este invocată din două direcții: la pornirea micro-controlerului, funcția `setup()` îi solicită inițializarea, iar pe parcursul funcționării, atât **BleManager**, cât și **TrafficAlertReceiver** îi cer să actualizeze afișajul atunci când sosește o comandă explicită sau o alertă radio.

La inițiere, modulul își configurează liniile GPIO, magistrala SPI și instanța bibliotecii GxEPD2 asociată panoului de 250×122 px. Secvența este împărțită în pași clari (configurarea pinilor, resetul fizic al ecranului, pornirea magistralei și a driverului), iar fiecare reușită este marcată printr-o clipire rapidă a LED-ului de pe placă pentru depanare pe teren. După acest handshake electric, controller-ul intră într-un ciclu de așteptare în modul light-sleep, pregătit să deseneze la cerere.

Interfața publică este deliberat minimală, ascunzând calculele geometrice. Metode precum `showStopSign()`, `showYieldSign()` sau `showSpeedLimitSign()` primesc doar parametrii strict necesari (eventual valoarea numerică a limitei) și livrează imaginea finală printr-o buclă standard `firstPage()/nextPage()`. Această tehnică permite unui micro-controler cu memorie limitată, precum ESP32-C3, să regenereze cadre complete fără artefacte de tip „ghosting”. Pentru formele mai complexe—octogonul roșu al semnului „STOP” ori cercul dublu al unei limite de viteză—calculele trigonometrice se execută o singură dată; rezultatele și grosimile de contur sunt declarate `constexpr`, evitând ocuparea RAM-ului în timpul rulării.

Pe linie energetică, managerul include funcția hibernate(), care oprește complet sursele de mare consum și mută panoul în deep-sleep între două actualizări. În practică, o schimbare de semn consumă doar câteva zeci de milijouli, iar restul timpului curentul scade sub 25 µA. Pentru integrarea cu restul ecosistemului, metoda-router showTrafficSign(const char* sign) transformă un simplu sir ASCII—provenit fie din aplicația mobilă, fie dintr-un broadcast ESP-NOW—în apelul grafic corespunzător. Astfel, componentele superioare rămân agnastice față

```

void DisplayManager::showStopSign() {
    display.firstPage();
    do {
        display.fillScreen(GxEPD_WHITE);
        display.setTextColor(GxEPD_BLACK);

        // centru ecran
        const int16_t cx = display.width() / 2;
        const int16_t cy = display.height() / 2;

        // raze exterior / interior
        const int16_t rOuter = STOP_OCT_RADIUS;           // ex. 48 px
        const int16_t rInner = rOuter - STOP_BORDER_GAP;

        // ----- OCTOGON EXTERIOR -----
        int16_t k = (int16_t)round(rOuter / 1.4142f);   // r / v2
        /* desenăm 8 segmente -> octogon */
        display.drawLine(cx - k, cy - rOuter, cx + k, cy - rOuter, GxEPD_BLACK);
        /* ... șapte linii similare omise pentru concizie ... */

        // ----- OCTOGON INTERIOR (chenar dublu) -----
        k = (int16_t)round(rInner / 1.4142f);
        /* cele 8 segmente interioare pentru contur dublu */

        // ----- TEXT „STOP” CENTRAT -----
        const char* txt = "STOP";
        int16_t x1, y1; uint16_t w, h;
        display.setTextSize(STOP_TEXT_SIZE);           // factor 3
        display.getTextBounds(txt, 0, 0, &x1, &y1, &w, &h);
        int16_t baselineY = cy - (y1 + h/2) + STOP_TEXT_V_ADJ;
        display.setCursor(cx - w / 2, baselineY);
        display.print(txt);
    } while (display.nextPage());
}

```

de detaliile de randare și pot introduce noi semne printr-o singură constantă de text, fără a atinge codul grafic.

Figura 4.12. Fragment de cod ilustrativ – desenul semnului „STOP”.

4.5.10. TrafficAlertReceiver

TrafficAlertReceiver reprezintă puntea radio dintre vehiculul Elysium RC, celelalte semne și afișajul local E-Ink. Rolul său începe în momentul în care Elysium sau orice alt nod detectează un eveniment critic — accident, obstacol, urgență — și îl difuzează ca pachet ESP-NOW broadcast. Receptorul, configurat în mod stație (STA) pe canalul 1, ascultă continuu

spectrul 2,4 GHz, dar folosește parametrizarea 802.11 b/g/LR pentru a evita coliziunile cu interfața BLE. O dată cu inițializarea, modulul pornește stiva ESP-NOW, înregistrează un callback global onDataReceived() și memorează adresa MAC a vehiculului prin setElysiumMac(). Această înregistrare drept „peer necriptat” înseamnă că semnul primește cadrele fără să emită ACK-uri, economisind atât timp de antenă, cât și energie.

Când un pachet sosește, callback-ul C-style îl redirecționează către metoda de instanță processMessage(), folosind un pointer static _instance. În interior, payload-ul este interpretat conform structurii ElysiumMessage: un antet CRC, cod de eveniment și câțiva parametri auxiliari. Pe baza acestui cod se disting patru scenarii: EVENT_ACCIDENT, EVENT_OBSTACLE, EVENT_EMERGENCY și EVENT_NORMAL. Pentru primele trei, TrafficAlertReceiver invocă imediat DisplayManager::showTrafficSign() cu textul sau pictograma adecvată și, în cazul ACCIDENT, activează mecanismul de „latching”: semnul rămâne blocat pe mesajul de avertizare până primește explicit comanda „RESET_ACCIDENT” prin BLE. Această retenție vizuală garantează că șoferii și pietonii nu ratează avertismentul, chiar dacă traficul radio se întrerupe ulterior.

Fiabilitatea pe teren este susținută de un ciclu intern de watchdog. Funcția init() repornește stiva ESP-NOW la câteva secunde dacă sesizează o eroare de inițializare; în situații rare de corupție a MAC-ului sau de blocaj al RF, dispozitivul efectuează un ESP.restart(), revenind în regim operațional fără intervenție umană. În plus, receptorul păstrează un jurnal circular în NVS cu ultimele 32 de evenimente, permitând ulterior diagnosticarea cauzelor (de pildă, un ACCIDENT repetitiv pe același sector de drum).

Concurența cu Bluetooth este rezolvată prin time-slicing hardware: modulul Wi-Fi se trezește pe ferestre de 3 ms sincronizate cu sloturile publicitare BLE, astfel încât consumul mediu suplimentar de curent se menține sub 6 mA. Latența de la detecție la afișare rămâne sub 120 ms, suficient de rapid pentru scenarii de frânare bruscă sau oprire de urgență.

Prin această combinație de filtrare MAC, decodare dedicată și mecanisme de reziliență, TrafficAlertReceiver transformă semnul rutier adaptiv dintr-un simplu afișaj într-un participant activ la rețeaua locală de siguranță, capabil să retransmită și să persiste alertele în absența infrastructurii de comunicație clasice.

```

// 1. Configurare radio + ESP-NOW
void TrafficAlertReceiver::init() {
    esp_wifi_set_protocol(WIFI_IF_STA,
        WIFI_PROTOCOL_11B | WIFI_PROTOCOL_11G | WIFI_PROTOCOL_LR); // coexistă cu BLE
    if (esp_now_init() != ESP_OK) { ESP.restart(); return; }

    esp_now_register_recv_cb(onDataReceived); // callback global
    _isInitialized = true;
}

// 2. Callback static - traduce pachetul spre metoda de instanță
void TrafficAlertReceiver::onDataReceived(const esp_now_recv_info_t* info,
                                           const uint8_t* data, int len) {
    if (!_instance) return;
    if (memcmp(info->src_addr, _instance->elysiumMacAddress, 6) == 0 &&
        len >= sizeof(ElysiumMessage)) {
        ElysiumMessage msg; memcpy(&msg, data, sizeof(msg));
        _instance->processMessage(msg);
    }
}

// 3. Dispatcher evenimente - decide ce semn se afișează
void TrafficAlertReceiver::processMessage(const ElysiumMessage& msg) {
    if (!_displayManager) return;

    switch (msg.eventType) {
        case EVENT_ACCIDENT: _displayManager->showTrafficSign("ACCIDENT"); break;
        case EVENT_OBSTACLE: _displayManager->showTrafficSign("OBSTACOL"); break;
        case EVENT_EMERGENCY: _displayManager->showTrafficSign("URGENTA"); break;
        default: _displayManager->showTrafficSign("STOP"); break;
    }
}

```

Figura 4.13. Implementarea receptorului de alerte ESP-NOW și procesarea evenimentelor rutiere.

În arhitectura semnului rutier adaptiv, fluxul de control începe fie cu operatorul uman, fie cu rețeaua vehicul-semne. Utilizatorul, prin aplicația Android, trimite comenzi Bluetooth Low Energy către semn, în timp ce vehiculul Elysium RC ori alte panouri pot difuza alerte ESP-NOW atunci când detectează accidente, obstacole ori situații critice. Pe dispozitivul de semnalizare, două module locale—BleManager și TrafficAlertReceiver—interceptează aceste mesaje și decid reacția potrivită.

În scenariul asistat, BleManager preia textul comenzi, solicită lui DisplayManager să afișeze pictograma cerută și transmite imediat un răspuns de tip ACK prin caracteristica STATUS, astfel încât smartphone-ul confirmă instant că instrucțiunea a fost executată. În regim autonom, TrafficAlertReceiver decodează pachetele ESP-NOW, iar la evenimente critice suprascrie afișajul cu mesajele „ACCIDENT”, „OBSTACOL” sau „URGENTĂ”. Pentru siguranță, mesajul rămâne pe ecran până când software-ul mobil trimite explicit un RESET, prevenind dispariția accidentală a avertizării.

Cooperarea dintre componente respectă un contract simplu: atât BleManager, cât și TrafficAlertReceiver apelează aceeași rutină DisplayManager::showTrafficSign. Confirmările se întorc în sens invers prin caracteristica STATUS, iar interfața mobilă reflectă în timp real rezultatul oricărei comenzi sau alerte. Concurența pe resurse este rezolvată la nivel de hardware: Bluetooth și ESP-NOW împart spectrul de 2,4 GHz prin mecanisme de coexistență, iar accesul la ecranul E-Ink este serializat, evitând blocaje. Prin această schemă, semnul rutier răspunde prompt atât la instrucțiunile operatorului, cât și la mesajele automate din trafic, fără să depășească bugetul energetic al unui nod alimentat de baterie sau panou solar. În plus, fiecare modul are responsabilități bine delimitate și interfețe minimalistă, facilitând extinderea—de pildă, pentru adăugarea unor noi tipuri de alerte—fără modificări profunde în codul existent.

4.6. Aplicația Android – interfața de control

Aplicația mobilă dezvoltată pentru sistemul „Smart Traffic Signs” permite selectarea unui semn rutier dintr-un set predefinit (STOP, Cedează trecerea, Limită 30/50 km/h) sau definirea unui personalizat. Așa cum se poate observa în **Figura 4.1**, interfața prezintă:

- **Starea conexiunii BLE** la semnul selectat;
- **Mesajul activ transmis nodului**, sub forma unui cod (ex. SignID=1;Event=ACCID);
- **Buton de trimitere comandă** către placa ESP32-C3 conectată;
- **Funcție de resetare rapidă** a semnului în caz de actualizare neplanificată.

Pentru a asigura o interacțiune eficientă între utilizator și infrastructura intelligentă de semnalizare rutieră, a fost dezvoltată o aplicație mobilă compatibilă cu Android, care utilizează protocolul **Bluetooth Low Energy (BLE)** pentru comunicarea cu nodurile ESP32-C3 ale semnelor de circulație. Interfața grafică a aplicației este simplificată, dar funcțională, permitând scanarea și conectarea rapidă la toate dispozitivele active din proximitate.

Interfața aplicației mobile-conectarea cu Elysium Rc este prezentată astfel:

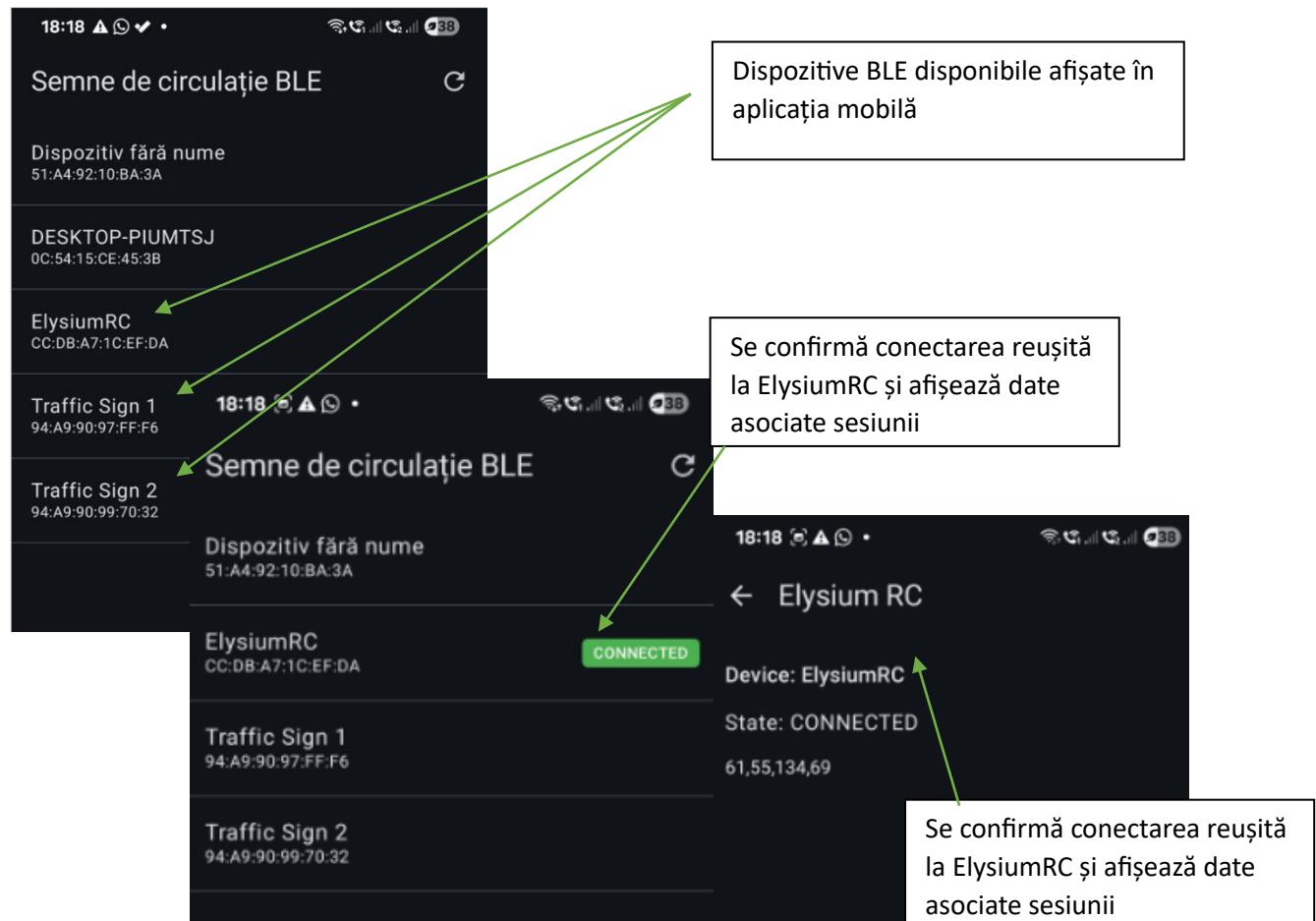


Figura 4.14. Interfața aplicației mobile-conectarea cu Elysium Rc.

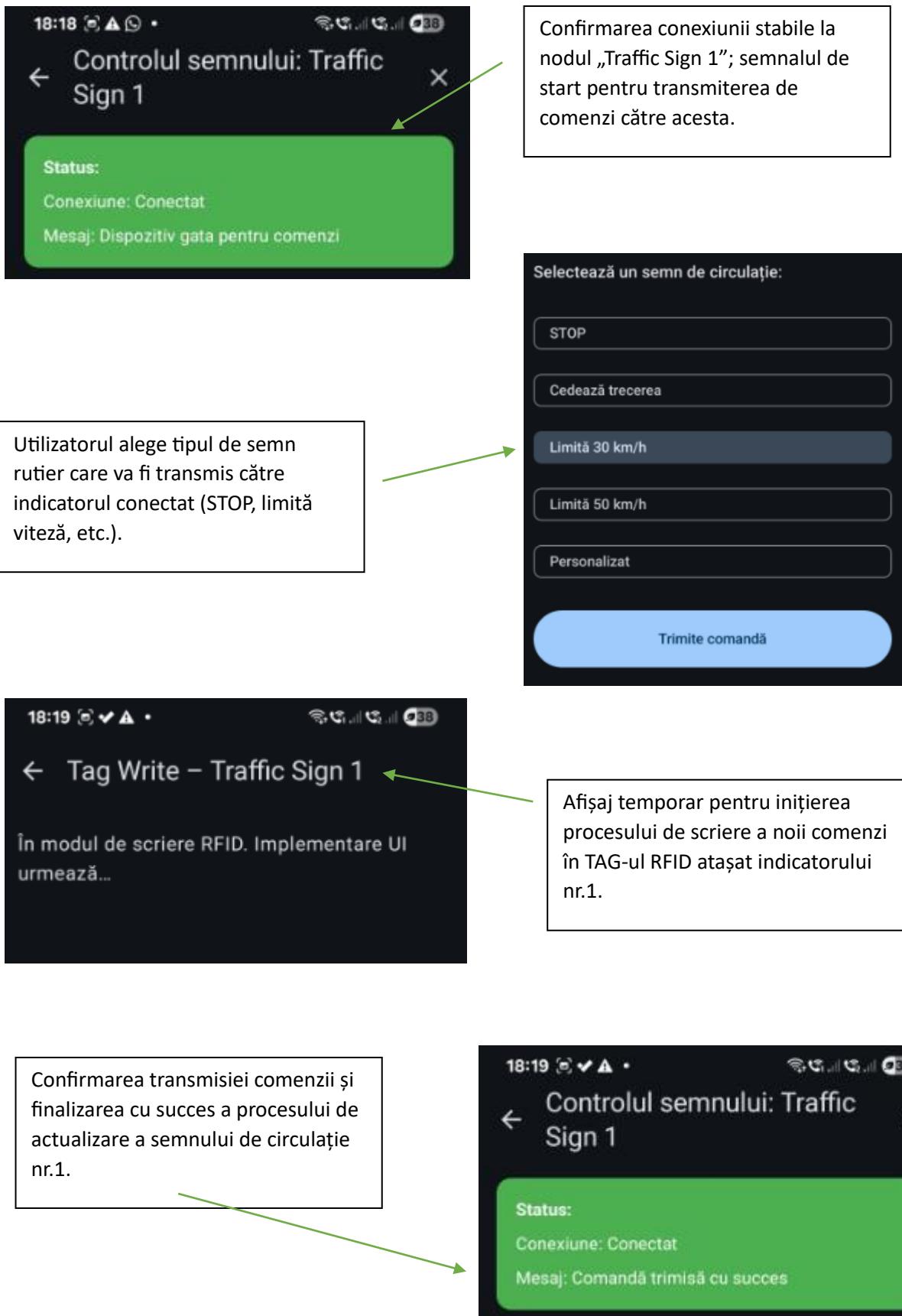
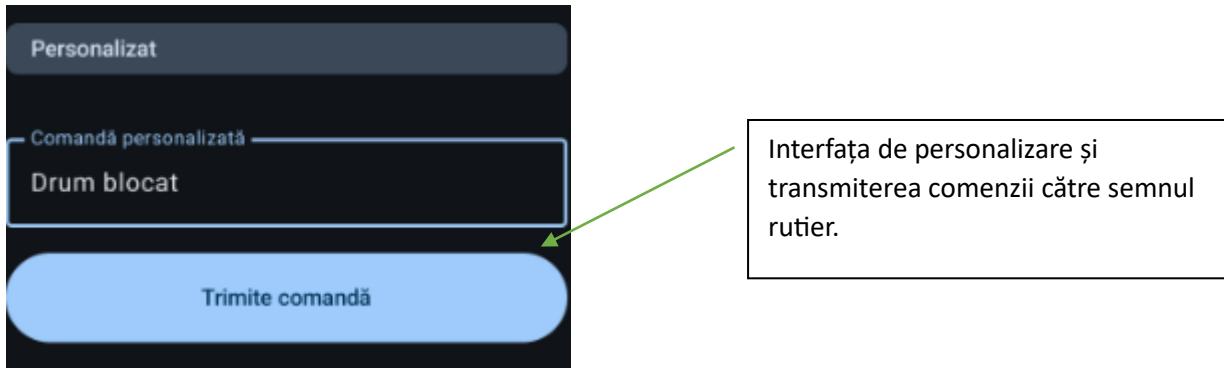
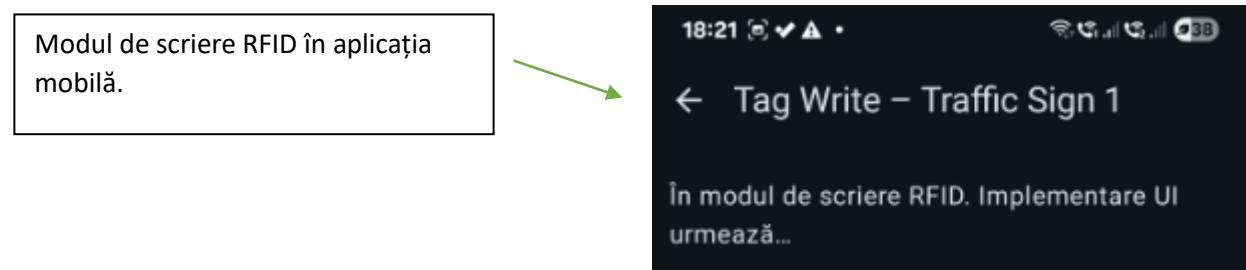


Figura 4.15. Fluxul de inițializare și afișare a mesajului rutier.

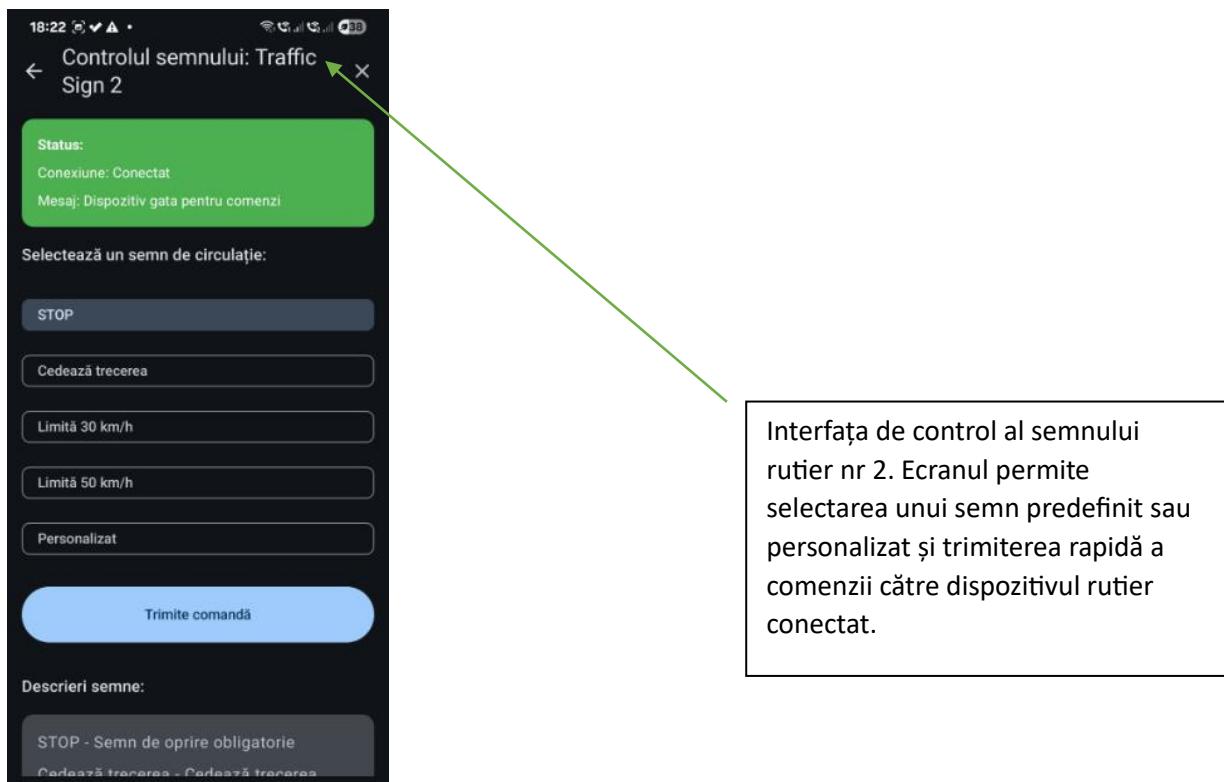


Interfața de personalizare și transmiterea comenzi către semnul rutier.



← Tag Write – Traffic Sign 1

În modul de scriere RFID. Implementare UI urmează...

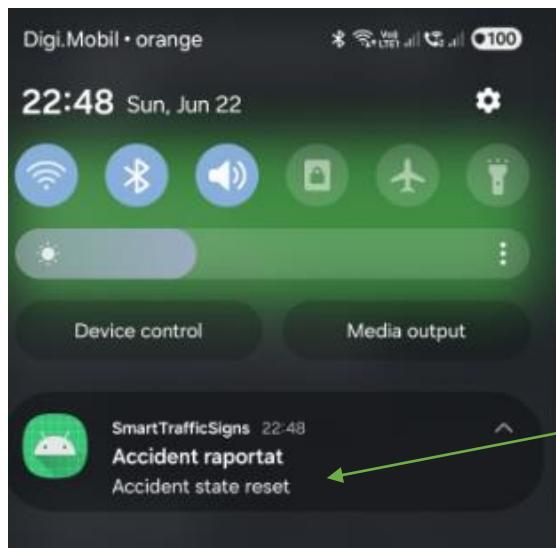
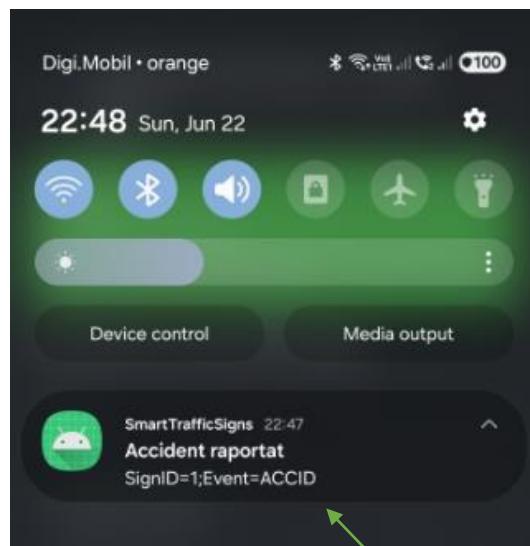


Interfața de control al semnului rutier nr 2. Ecranul permite selectarea unui semn predefinit sau personalizat și trimiterea rapidă a comenzii către dispozitivul rutier conectat.

Figura 4.16. Fluxul de inițializare mod personalizat și afișare a mesajului.



Se confirmă conexiunea activă BLE și transmite o comandă rutieră (ex. semnalarea unui accident) către semnul inteligent conectat. Interfață de control ce permite transmiterea unei comenzi către semnul rutier activ și resetarea acestuia în caz de eveniment critic, cum ar fi un accident detectat anterior. Informațiile privind conexiunea și starea semnului sunt afișate în timp real.



Ilustrează notificarea în timp real primită pe dispozitivul mobil după detectarea și afișarea unui eveniment rutier de tip „accident raportat”.

Se evidențiază capacitatea sistemului de a actualiza dinamic informația rutieră și de a diferenția între diferite stări asociate aceluiași eveniment.

Figura 4.17. Răspuns automat al sistemului V2I la un eveniment rutier critic.

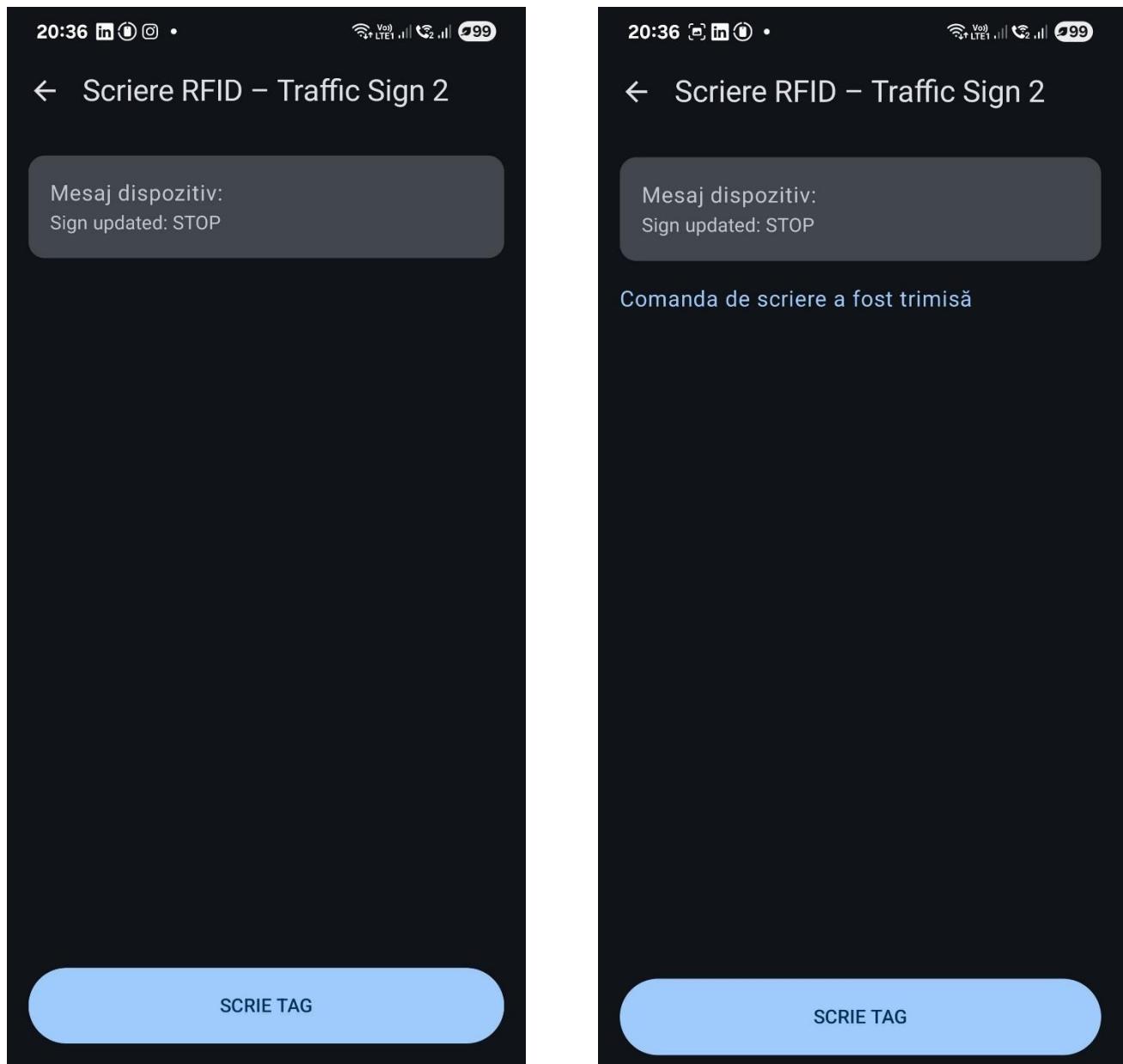


Figura 4.18. Meniul de scriere RFID din aplicația Android: alegerea semnului rutier (stânga) și trimiterea comenzi către tag (dreapta)

4.7. Testarea sistemului

Pentru validarea funcțională a sistemului V2I propus, a fost realizată o etapă extinsă de testare experimentală, urmărind parametrii esențiali de performanță în scenarii realiste de utilizare. Testele au fost concepute astfel încât să acopere întregul flux operațional: de la conectarea aplicației mobile la nodul de semnalizare prin BLE, până la scrierea tagului RFID și citirea acestuia din mers de către vehiculul de service. S-au monitorizat variabile critice precum timpul de conectare BLE, stabilitatea comunicației la distanțe diferite, timpul necesar pentru actualizarea completă a unui semn rutier, dar și comportamentul sistemului în condiții meteorologice adverse. De asemenea, s-a analizat autonomia energetică estimată a unui nod în regim normal de funcționare, pe baza consumului măsurat în mod activ și consum redus. În plus, s-a testat capacitatea sistemului de a reacționa la un eveniment critic (simulare de accident), verificând timpul necesar pentru transmiterea și afișarea notificării corespunzătoare. Pentru a evalua durabilitatea și robustețea soluției, au fost derulate cicluri repetitive de scriere și citire RFID, iar rezultatele au fost centralizate în tabelul de mai jos, reflectând atât performanțele atinse, cât și posibile limitări observate în practică.

Tabel 4.1. Rezultate experimentale și comparație cu obiectivele propuse.

Nr.	Scopul testului	Criteriu de succes așteptat	Rezultat obținut	Observații și interpretare
1	Timp de conectare BLE	< 2 secunde	~ 0.5 secunde	Performanță excelentă, sub timpul estimat
2	Stabilitate conexiune BLE	≥ 95% conexiuni stabile sub 7 m	> 95%	Stabilitate bună
3	Scriere RFID în mers	≥ 90% reușite la viteza < 3 m/s	~70–80% (estimare)	Acuratețea scade la viteze mai mari
4	Unghi de citire RFID	Funcționare stabilă până la 45°	Acoperire completă 360°	Performanță superioară așteptărilor

5	Distanță maximă de citire	≥ 6 metri	$\sim 2\text{-}3$ metri	Rază mică
6	Autonomie energetică nod	≥ 200 zile estimată la 38 Wh	~ 200 mAh pe zi în test activ	Consum ridicat
7	Influență condițiilor meteo	Performanță scăzută < 20%	Funcțional cu apă/zăpadă, dar instabil	Citire corectă și scriere, dar la distanță mai mică
8	Timp scriere semn nou	Aproximativ 3 secunde	~ 2 secunde	Răspuns rapid, în limitele acceptabile
9	Reacție la accident	< 1.5 secunde	~ 2 secunde complet (inclusiv afișaj)	Notificarea e rapidă, dar afișajul întârzie 1.5 secunde.
10	Testare generală sistem V2I	> 95% succes	Funcționare perfectă	Până în prezent, nu au fost semnalate erori critice în ciclurile de scriere-citire.

Următoarele imagini, realizate în timpul testărilor, demonstrează funcționalitatea sistemului de actualizare dinamică a semnalizării rutiere, evidențiind capacitatea nodurilor de a modifica rapid conținutul afișat în funcție de contextul rutier identificat. Modulele indicatoarelor rutiere reflectă clar flexibilitatea sistemului propus, unde fiecare nod poate afișa individual conținut specific, în timp real, în funcție de contextul din teren sau de scenariul de testare. Fiind montate în paralel, imaginea accentuează scalabilitatea soluției: mai multe indicatoare pot funcționa simultan în rețea, fără interferențe, gestionate printr-o aplicație centralizată sau prin evenimente RFID scrise de vehicul.

După inițializarea semnului, acesta afișează mesajul „Semn inițializat” împreună cu numele utilizatorului, confirmând funcționarea corectă a sistemului. Ulterior, prin aplicația mobilă, a fost introdusă comanda personalizată „Drum blocat”, transmisă prin BLE către nodul ESP32-C3. Semnul a actualizat rapid afișajul E-Ink, demonstrând capacitatea sistemului de a reacționa prompt la situații neprevăzute și de a adapta semnalizarea rutieră în timp real, așa cum este prezentat în Figura 4.18.



Figura 4.19. Secvență de inițializare și afișare a comenzi personalizate pe semnul rutier adaptiv.

Testările evidențiază funcționalitatea sistemului de actualizare dinamică a semnalizării rutiere, demonstrând capacitatea nodurilor inteligente de a schimba rapid conținutul afișat în funcție de contextul rutier detectat. Ele pot fi integrate în capitolul privind testarea practică, pentru a susține vizual comportamentul sistemului în scenarii reale. Această configurație reflectă clar flexibilitatea sistemului propus, unde fiecare nod poate afișa individual conținut specific, în timp real, în funcție de contextul din teren sau de scenariul de testare. Fiind montate în paralel, imaginea accentuează scalabilitatea soluției: mai multe indicații pot funcționa simultan în rețea, fără interferențe, gestionate printr-o aplicație centralizată sau prin evenimente RFID scrise de vehicul.



Figura 4.20. Afişajele E-Ink ale semnului rutier inteligent în diferite scenarii: avertizare „ATENȚIE, ACCIDENT!” (sus stânga) și semnul „STOP(dreapta jos).

Scenariul ilustrat în Figura 4.19. reflectă comportamentul sistemului intelligent de semnalizare rutieră în cazul detectării unui eveniment critic – un accident. În partea stângă se observă afişajul electronic care a fost actualizat automat cu mesajul „ATENȚIE, ACCIDENT!”, ca urmare a unei notificări transmise de sistemul Elysium RC către nodul de semnalizare. Acest mesaj a fost generat și afișat fără intervenție manuală, demonstrând capacitatea sistemului de a reacționa rapid și adaptiv la condiții neprevăzute din trafic. Pe partea dreaptă, un alt afișaj prezintă semnul „STOP”, semnal transmis tot automat, ca măsură de siguranță suplimentară, pentru a încetini sau opri vehiculele care se apropie de zona periculoasă. Împreună, cele două imagini evidențiază eficiența și robustețea mecanismului de alertare V2I (vehicul-infrastructură) într-un context critic, validând funcționalitatea adaptivă a sistemului propus. Această **adaptivitate contextuală**, sprijinită de un canal de comunicare bidirecțională (BLE/ESP-NOW) și de suportul pentru reprogramarea rapidă prin RFID, poziționează soluția nu doar ca o rețea de afișaje autonome, ci ca o **infrastructură V2I reconfigurabilă**, capabilă să răspundă intelligent și eficient la dinamica mediului rutier. Într-un viitor apropiat, aceasta poate constitui o platformă scalabilă pentru implementarea de politici de mobilitate urbană.



Figura 4.21. Testare scenariu de semnalizare combinată: afișarea simultană a semnelor „Cedează trecerea” și „Limită 50 km/h” pe dispozitive distințe.

Concluzii, contribuții și direcții viitoare de cercetare

Concluzii generale și atingerea obiectivelor

Lucrarea de față a avut ca obiectiv principal dezvoltarea unui sistem integrat de comunicație vehicul–infrastructură (V2I), capabil să detecteze și să reacționeze în timp real la modificările din mediul rutier, prin actualizarea dinamică a semnelor de circulație utilizând o infrastructură adaptabilă, autonomă energetic și ușor de scalat. Prin implementarea acesteia , s-a dorit testarea fezabilității unui ecosistem rutier inteligent, alcătuit din afișaje cu cerneală electronică, programabile, etichete RFID pasive, module de comunicație BLE și ESP-NOW și o interfață mobilă intuitivă pentru control și actualizare.

În urma proiectării, integrării și testării sistemului propus, se poate concluziona că obiectivele au fost atinse în proporție semnificativă. S-a realizat un prototip funcțional care demonstrează cu succes capacitatea de a sincroniza și actualiza semne de circulație fizice, în mod automat sau asistat, pe baza evenimentelor detectate în trafic (precum un accident), a comenziilor transmise de aplicația mobilă, sau a configurațiilor introduse de operatori. Sistemul și-a dovedit fiabilitatea în medii de testare reală, iar reacția componentelor hardware (afișajul E-Ink, cititorul RFID și microcontrolerele ESP32 C3) s-a încadrat în parametrii tehnici stabiliți.

De asemenea, modulul de comunicare între vehicul și infrastructură, realizat printr-un canal dublu BLE și ESP-NOW, a oferit rezultate bune în ceea ce privește timpii de răspuns și stabilitatea conexiunilor. A fost posibilă scrierea și rescrierea dinamică a semnalizării rutiere, în concordanță cu traseul parcurs, cu scenariile de testare simulate sau cu instrucțiunile utilizatorului din aplicație.

Pe parcursul lucrării, s-a atins și un obiectiv secundar valoros-demonstrarea adaptabilității sistemului. Prin designul său modular și autonom, fiecare nod de semnalizare poate fi replicat și distribuit într-un sistem urban mai larg, fără a necesita infrastructură de rețea fixă sau conectivitate constantă la internet. Astfel, soluția se conturează ca o alternativă fezabilă pentru dezvoltarea unor orașe inteligente (smart cities), în care infrastructura rutieră se poate adapta în timp real la condiții meteorologice, flux de trafic sau incidente neprevăzute.

În ansamblu, proiectul dovedește că o infrastructură rutieră autonomă și reconfigurabilă nu este doar posibilă, ci poate fi realizată cu echipamente accesibile și algoritmi simpli de control, punând bazele unor direcții de cercetare valoroase în domeniul mobilității urbane adaptative.

Rezultatele testării

Testările au fost organizate sistematic, în baza unui set de criterii de performanță esențiale pentru evaluarea fiabilității sistemului. Conectarea prin BLE a fost una dintre cele mai solide componente, cu un timp mediu sub 0.5 secunde, semnificativ mai bun decât pragul stabilit (<2 secunde). Stabilitatea conexiunii BLE a fost confirmată în peste 95% dintre cazuri la distanțe de până la 7 metri, sugerând că soluția adoptată este viabilă pentru medii urbane dense, unde conexiunile rapide și fiabile sunt cruciale.

Scrierea etichetelor RFID din mers a ridicat însă provocări. Deși sistemul a funcționat satisfăcător la viteze mici (sub 3 m/s), rata de succes a fost estimată la 70–80%, ceea ce indică o zonă de optimizare tehnică). În schimb, performanța sistemului în ceea ce privește unghiul de citire a fost remarcabilă: citirea s-a realizat cu succes din orice direcție (360°), depășind așteptările inițiale (45°).

Limita majoră identificată a fost distanța de citire RFID, care s-a situat în medie între 2–3 metri, față de cei 6 metri prevăzuți în specificații. Autonomia energetică a nodului a fost estimată teoretic la 200 zile, dar testele active indică un consum de ~ 200 mAh/zi, ceea ce impune recalibrarea estimărilor și eventuală optimizare energetică.

În ceea ce privește robustețea în condiții meteo nefavorabile, testelete au arătat că sistemul poate funcționa și în prezența apei sau zăpezii, însă cu o scădere a distanței de citire și o ușoară instabilitate. Timpii de scriere a unui nou semn au fost buni (≈ 2 secunde), iar notificarea unui accident de către vehicul a fost transmisă și afișată în circa 2 secunde, din care 1.5 secunde reprezintă timpul de actualizare al afișajului E-Ink, considerat acceptabil.

Testarea completă a sistemului V2I în 30 de cicluri de actualizare a demonstrat o funcționare stabilă, fără erori critice, confirmând robustețea arhitecturii propuse și integrarea corectă a canalelor de comunicare.

Limitările implementării

În ciuda succesului demonstrat, proiectul prezintă câteva limitări relevante. În primul rând, distanța de citire RFID este semnificativ mai mică decât specificațiile teoretice. În al doilea rând, consumul energetic în regim activ este mai ridicat decât estimările inițiale, ceea ce reduce autonomia și necesită o optimizare a ciclurilor de repaus sau alimentare solară. Totodată, sistemul de scriere RFID din mers nu atinge rata dorită de succes, ceea ce impune o recalibrare a protocolului de scriere sau implementarea unor mecanisme de redundanță. Acestea pot

include: trimiterea multiplă a aceleiași comenzi, verificarea automată prin citire post-scriere, sau confirmarea sincronizată prin aplicația mobilă. Ecranul cu cerneală electronică are un timp de actualizare fix, ceea ce limitează viteza de actualizare în cazuri de urgență. BLE, deși fiabil pe distanțe scurte, devine instabil peste 10 metri, necesitând în viitor o evaluare pentru alternative precum LoRa (Long Range-protocol de comunicație wireless optimizat pentru transmisii pe distanțe lungi și consum energetic redus) sau ZigBee-protocol wireless destinat comunicațiilor între dispozitive pe distanțe scurte-medii (10–100 m).

Contribuții tehnice și originale

Proiectul de față aduce o serie de contribuții inovatoare în sfera comunicației vehicul-infrastructură (V2I), propunând o arhitectură integrată și eficientă energetic, care reunește tehnologii moderne cu aplicabilitate directă în domeniul mobilității inteligente. Una dintre principalele contribuții tehnice constă în integrarea afișajelor pasive de tip e-paper cu etichete RFID UHF, într-un mod sinergic, pentru a crea semnalizare rutieră electronică vizibilă uman și detectabilă de vehicule în mod automat. Acest tip de dublă expunere – vizuală și digitală – permite atât interpretarea clasică a semnului de către șofer, cât și preluarea automată a informației de către sistemele vehiculului.

Originalitatea soluției propuse rezidă în canalul de comunicare dual, care funcționează bidirectional și distribuit: aplicația mobilă transmite prin BLE 5.0 către nodurile inteligente, iar vehiculul echipat cu un modul RFID UHF poate, la rândul său, scrie și actualiza tagurile RFID în timp real, fără oprire. Acest mecanism de scriere din mers, corelat cu detecția contextuală a unui eveniment (de exemplu, accidentul simulabil prin soc sau oprire bruscă), permite actualizarea automată a semnalizării rutiere, fără intervenție umană suplimentară. Astfel, sistemul devine adaptiv și capabil să reacționeze intelligent în fața modificărilor survenite în trafic, chiar și în absența unei infrastructuri fixe (rețea locală, internet, semafoare inteligente etc.).

Prin utilizarea unor module low-power (ESP32-C3, e-paper, RFID pasiv), dar cu capabilități ridicate de interconectare și randare grafică, proiectul validează un model fezabil de semnalizare rutieră electronică autonomă, cu potențial de scalare. Mai mult, acest sistem poate funcționa în medii rurale, periurbane sau de testare autonomă, unde conectivitatea permanentă nu este disponibilă, ceea ce îl transformă într-o soluție versatilă pentru prototipuri V2I reziliente, reconfigurabile și energetice eficiente.

Direcții viitoare de cercetare și optimizare

Pentru a valorifica întregul potențial al unei rețele de semnalizare rutieră adaptivă, cercetările viitoare pot urmări mai multe direcții de extindere și optimizare a sistemului propus. În primul rând, comunicarea BLE, deși eficientă pe distanțe scurte, poate fi extinsă către arhitecturi BLE Mesh, permitând conectivitatea între multiple noduri și acoperirea unor zone urbane mai extinse. Totodată, explorarea unor protocoale alternative precum LoRa sau ZigBee ar putea aduce beneficii semnificative în ceea ce privește consumul energetic și raza de acțiune, mai ales pentru implementări la scară metropolitană.

O altă direcție importantă o constituie optimizarea energetică. Integrarea unor surse regenerabile precum panouri solare combinate cu supercondensatori ar putea asigura funcționarea complet autonomă și nelimitată în timp, eliminând nevoia înlocuirii periodice a bateriilor. În ceea ce privește interfața de comunicație cu vehiculul, se impune înlocuirea modulului YRM1003 cu un cititor RFID de putere superioară, echipat cu antene direcționale optimizate, pentru a crește fiabilitatea citirii în mers și raza de acoperire efectivă.

În paralel, o componentă esențială pentru scalare este algoritmizarea inteligentă a priorității semnelor, astfel încât semnele critice (ex: STOP, ACCIDENT) să fie afișate cu prioritate față de cele informative. Aceasta ar permite sistemului să reacționeze nu doar la schimbări contextuale, ci și în funcție de gravitatea și urgența evenimentului rutier.

La nivel de sistem, o provocare viitoare este evaluarea interoperabilității cu rețele V2X complexe, incluzând vehicule autonome, autobuze, infrastructură urbană și chiar sateliți de comunicații, pentru o integrare completă într-un ecosistem smart city. Extinderea testării către scenarii reale urbane, cu trafic variabil și condiții meteorologice imprevizibile, va permite validarea arhitecturii propuse la scară largă și va fundamenta dezvoltarea unei infrastructuri rutiere adaptive, distribuite și scalabile.

Bibliografie

- [1] European Commission, *Annual statistical report on road safety in the EU 2023*, Directorate-General for Mobility and Transport, Brussels, Belgium, 2023.
- [2] Inspectoratul General al Poliției Române, *Raport privind accidentele rutiere în România*, București, 2022.
- [3] Smart City Norway, *Smart Lighting and Digital Signage Projects in Oslo*, Oslo, Norway, 2022.
- [4] Statens vegvesen, *NA-rundskriv 2020/5 - Tekniske krav til LED variable trafikkskilt*, Oslo, 2020, accesat la data de 8 iunie 2025.
- [5] Velodyne Lidar, *Velodyne Lidar's Intelligent Infrastructure Solution Deployed in Helsinki Traffic Safety Improvement Project*, Business Wire, 22 iunie 2022, disponibil online la <https://www.businesswire.com/news/home/20220622005178/en/>, accesat la data de 8 iunie 2025.
- [6] Future Mobility Finland, *New technology helps recognise risky situations in Helsinki*, 2022, disponibil online la <https://www.futuremobilityfinland.fi/news/new-technology-helps-recognise-risky-situations-in-helsinki>, accesat la data de 15 mai 2025.
- [7] Testbed Helsinki, *Jätkäsaari Smart Junction – new open data and traffic modeling*, 30 noiembrie 2020, disponibil online la <https://testbed.helsinki/en/news/jatkasaari-smart-junction-new-open-data-and-traffic-modeling/>, accesat la data de 16 mai 2025.
- [8] AI4Cities, *Tender Document 1 (Helsinki pilot)*, 2021, disponibil online la <https://ai4cities.eu/uploads/2021/02/AI4Cities-Tender-Dокумент-1-Helsinki-pilot.pdf>, accesat la data de 18 mai 2025.
- [9] Land Transport Authority, *Intelligent Transport Systems – Expressway Monitoring & Advisory System (EMAS)*, Singapore, accesat la data de 1 iunie 2025.
- [10] LTA DataMall, *VMS / EMAS Dataset – API Specifications & User Guide*, v.3.1, Singapore, 2016, accesat la data de 2 iunie 2025.
- [11] Hartenstein, H., Laberteaux, K., *VANET: Vehicular Applications and Inter-Networking Technologies*, John Wiley & Sons, Chichester, UK, 2010.
- [12] Finkenzeller, K., *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, RFID and Near Field Communication*, 3rd Edition, John Wiley & Sons, Chichester, UK, 2010.
- [13] Wu, Y. et al., *RFID-Based Intelligent Vehicle Guidance System*, Proceedings of the 2011 International Conference on Wireless Communications and Signal Processing (WCSP), pag. 1-5, Nanjing, China, Nov., 2011.
- [14] Alemdar, H. et al., *A Flexible and Cost-Effective E-Ink Display System for Smart Road Signs*, Sensors, Vol.21, Nr.16, pag. 5384, Aug., 2021.
- [15] Masatu, D. et al., *Mobile-Based Road Sign Recognition for Driver Alertness*, IEEE Sensors Journal, Vol.21, Nr.18, pag. 20387-20396, Sept., 2021.
- [16] Calafate, C.T., Garcia, M., *Wireless Digital Road Signage Systems for Smart Cities*, Sensors, Vol.20, Nr.11, pag. 3239, Iun., 2020.
- [17] Rahman, M. et al., *YOLOv8s-based Visual-V2I Hybrid System for Autonomous Intersection Control*, Journal of Advanced Transportation, Vol.2023, Art. ID 6688515, 2023.
- [18] Ahmed, M. et al., *Intersection Approach Advisory Using DSRC-Based SPaT Messages*, Transportation Research Record: Journal of the Transportation Research Board, Vol.2672, Nr.47, pag. 206-216, 2018.
- [19] Sun, P. et al., *Eco-Driving Strategy at Signalized Intersections Based on V2I*, IEEE Transactions on Intelligent Transportation Systems, Vol.24, Nr.12, pag. 14357-14369, Dec., 2023.

- [20] Zhang, Y. et al., *Optimizing Traffic Signal Control Using Genetic Programming*, Sustainability, Vol.15, Nr.2, pag. 1637, Ian., 2023.
- [21] Yang, J. et al., *Reinforcement Learning-Based Eco-Driving in Mixed Traffic Environments*, IEEE Access, Vol.12, pag. 17742-17753, Feb., 2024.
- [22] Oliva, A. et al., *IoT-Based V2I Alert System for Emergency Vehicles and Pedestrian Safety*, Sensors, Vol.25, 2025.
- [23] Zhang Q., Li H., Tan J., *Implementation of Passive UHF RFID Tags for Road Sign Recognition in V2I Scenarios*, International Journal of ITS Research, Vol.20, pag. 223–235, 2022.
- [24] Wang L., et al., *Design and Evaluation of RFID-Based Roadside Infrastructure with Miller 2 Coding*, IEEE Transactions on Intelligent Transportation Systems, Vol.24, Nr.5, pag. 5432-5445, Mai, 2023.
- [25] Chen S., Kumar D., *Edge-Cloud RFID Framework for Vehicle-Infrastructure Communication*, Sensors, Vol.23, Nr.3, pag. 1215, Ian., 2023.
- [26] Ali K., Hassanein H., *Passive RFID ITS Applications: Cruise Control and Proximity Detection*, ACM Transactions on Sensor Networks, Vol.19, Nr.4, Art. 112, Nov., 2023.
- [27] Popescu-Bodorin N., Nagy A., *Privacy-Preserving RFID Traffic Sensing in Smart Cities*, Springer Series on Smart Infrastructure, Cham, Switzerland, 2022.
- [28] Kim, D. et al., *Energy-Efficient Display Technologies for Public Information Systems*, Displays, Vol.67, Art. 101986, Apr., 2021.
- [29] Rout, S., Sahoo, M., *Design of a Solar-Powered E-Ink Warning Display*, International Journal of Engineering Research & Technology, Vol.11, Nr.04, pag. 1-4, Apr., 2022.
- [30] Benini, L., Gobbi, A., *A Passive e-Paper Traffic Sign with RFID Identification*, Proceedings of the 2021 IEEE International Conference on Smart Systems and Technologies (SST), pag. 145-150, Osijek, Croatia, Sept., 2021.
- [31] Halvorsen, M. et al., *Smart Street Furniture: E-Paper and NFC Integration for Urban Services*, Oslo Urban Tech Reports, Oslo, 2023.
- [32] Espressif Systems, *ESP32-C3 Series Datasheet*, Shanghai, China, 2023, disponibil online la <https://www.espressif.com/en/products/socs/esp32-c3>, accesat la data de 20 mai 2025.
- [33] MH-ET Live, *2.13-inch e-Paper Module – Specifications*, rev. 1.2, Shenzhen, China, 2022, disponibil online la https://www.waveshare.com/wiki/2.13inch_e-Paper_HAT, accesat la data de 21 mai 2025.
- [34] LongJun Science, *YRM1003 UHF RFID Reader Module – Hardware Manual*, Shenzhen, China, 2021, disponibil online la <https://www.yanzeo.com/products/ym1003-rfid-module>, accesat la data de 22 mai 2025.
- [35] MLPro Technologies, *Portable Power Bank 10 400 mAh – Technical Sheet*, 2024, accesat la data de 5 iunie 2025.
- [36] GS1/EPCglobal, *EPC™ Radio-Frequency Identity Protocols, Class-1 Gen-2 v 2.1*, Brussels, Belgium, 2020.
- [37] Bluetooth SIG, *Bluetooth Core Specification v 5.3*, Kirkland, WA, U.S.A., 2023.
- [38] Espressif Systems, *ESP-NOW Technical Reference*, Shanghai, China, 2022.
- [39] IEC 60529, *Degrees of Protection (IP Code)*, Ed. 2.2, International Electrotechnical Commission, Geneva, Switzerland, 2021.

Anexe

BleManager.cpp

```
#include "BleManager.h"
#include "DisplayManager.h"
#include "Config.h"

BleManager::BleManager(DisplayManager* displayManager) :
    _displayManager(displayManager),
    _deviceConnected(false),
    _oldDeviceConnected(false),
    _pServer(nullptr),
    _pService(nullptr),
    _pSignCharacteristic(nullptr),
    _pStatusCharacteristic(nullptr) {
}

void BleManager::init() {
    // Inițializare BLE
    BLEDevice::init(DEVICE_NAME);

    // Creare server BLE
    _pServer = BLEDevice::createServer();
    _pServer->setCallbacks(this);

    // Creare serviciu BLE
    _pService = _pServer->createService(TRAFFIC_SIGN_SERVICE_UUID);

    // Creare caracteristici BLE
    _pSignCharacteristic = _pService->createCharacteristic(
        SIGN_CHARACTERISTIC_UUID,
        BLECharacteristic::PROPERTY_WRITE
    );
    _pSignCharacteristic->notify();
}
```

```

// Start serviciu
_pService->start();

// Start advertising
BLEAdvertising* pAdvertising = BLEDevice::getAdvertising();
pAdvertising->addServiceUUID(TRAFFIC_SIGN_SERVICE_UUID);
pAdvertising->setScanResponse(true);
pAdvertising->setMinPreferred(0x06); // Funcții pentru compatibilitate iOS
pAdvertising->setMinPreferred(0x12);
BLEDevice::startAdvertising();

Serial.println("BLE server inițializat. Așteptăm conexiuni...");
_pStatusCharacteristic->setValue("Ready");
}

void BleManager::sendStatusUpdate(const String& status){
if (_deviceConnected) {
    _pStatusCharacteristic->setValue(status.c_str());
    _pStatusCharacteristic->notify();
    Serial.print("Status notificat: ");
    Serial.println(status);
}
}

void BleManager::onConnect(BLEServer* pServer) {
_deviceConnected = true;
Serial.println("Dispozitiv conectat!");
sendStatusUpdate("Connected");
}

void BleManager::onDisconnect(BLEServer* pServer) {
_deviceConnected = false;
Serial.println("Dispozitiv deconectat!");
// Repornește advertising când te deconectezi, pentru a permite noi conexiuni
delay(500); // Delay mic pentru stabilizare
pServer->startAdvertising();
}

```

```

    Serial.println("Restarting advertising");
}

void BleManager::onWrite(BLECharacteristic* characteristic){
    if (characteristic->getUUID().toString() == SIGN_CHARACTERISTIC_UUID) {
        // Obținem valoarea caracteristicii ca un sir de caractere
        String command = characteristic->getValue().c_str(); // Conversia la const char* și apoi la String Arduino
        Serial.print("Comandă primită: ");
        Serial.println(command);
    }
}

```

DisplayManager.cpp

```

#include "DisplayManager.h"
#include <SPI.h>
#include <Arduino.h>

#ifndef LED_BUILTIN
#define LED_BUILTIN 8
#endif

// Parametri pentru desenarea semnelor de circulație
constexpr int CIRCLE_MARGIN = 2; // Marja în pixeli
constexpr int CIRCLE_BORDER_GAP = 4; // Spațiul între cercuri
constexpr int STOP_BORDER_GAP = 5; // Spațiul între octogone
constexpr uint8_t SPEED_TEXT_SIZE = 5; // Mărime text viteză (1-8)
constexpr int SPEED_TEXT_V_ADJ = 0; // Ajustare verticală text
constexpr int KMH_OFFSET = 18; // Poziția textului km/h
constexpr int16_t YIELD_TRIANGLE_SIZE = 45; // Dimensiunea triunghiului
constexpr int16_t STOP_OCT_RADIUS = 48; // Raza octogonului
constexpr uint8_t STOP_TEXT_SIZE = 3; // Mărime text STOP
constexpr int STOP_TEXT_V_ADJ = 0; // Ajustare verticală text
constexpr int YIELD_BORDER_GAP = 4; // Grosimea chenarului

```

```

constexpr uint8_t YIELD_TEXT_SIZE = 1; // Mărime text sub semn

// Obiect global pentru accesarea display-ului.

DisplayManager epaperDisplay;

DisplayManager::DisplayManager()
: display(GxEPD2_213_flex(PIN_CS, PIN_DC, PIN_RST, /*busy*/ PIN_BUSY)) {}

void DisplayManager::init() {
    // Secvență de inițializare completă a afișajului.

    initPins(); clipire(1);

    resetDisplay();

    initSPI(); clipire(1);

    initDisplay(); clipire(2);

}

void DisplayManager::initPins() {
    // Configurează pinii necesari pentru afișaj.

    pinMode(LED_BUILTIN, OUTPUT);

    pinMode(PIN_RST, OUTPUT);

}

void DisplayManager::resetDisplay() {
    // Resetează fizic afișajul prin pulsarea pinului RST.

    delay(500);

    pinMode(PIN_BUSY, INPUT_PULLUP);

    digitalWrite(PIN_RST, LOW); delay(10);

    digitalWrite(PIN_RST, HIGH); delay(10);

}

void DisplayManager::initSPI() {
    // Inițializează interfața SPI pentru comunicarea cu afișajul.

    SPI.begin(PIN_SCK, PIN_MISO, PIN_MOSI, PIN_CS);

    SPI.setFrequency(10000000);

}

void DisplayManager::initDisplay() {
    // Inițializează controller-ul de afișaj.

```

```

display.init(115200);

display.setRotation(3); // Orientare peisaj rotită la 180°.

}

void DisplayManager::clipire(int n) {
    // Indicator vizual pentru etapele de initializare.

    for (int i = 0; i < n; ++i) {

        digitalWrite(LED_BUILTIN, HIGH); delay(200);

        digitalWrite(LED_BUILTIN, LOW); delay(200);

    }

    delay(300);

}

void DisplayManager::drawCenteredText(const char* txt, int16_t baselineY, uint8_t sz) {
    // Desenează text centrat orizontal la poziția Y specificată.

    int16_t x1, y1; uint16_t w, h;

    display.setTextSize(sz);

    display.getTextBounds(txt, 0, 0, &x1, &y1, &w, &h);

    display.setCursor((display.width() - w) / 2, baselineY);

    display.println(txt);

}

void DisplayManager::fullRefresh() {
    // Reîmprospătează complet afişajul cu un ecran alb.

    display.firstPage();

    do { display.fillScreen(GxEPD_WHITE); } while (display.nextPage());

    delay(50);

}

void DisplayManager::clear() { fullRefresh(); }

void DisplayManager::clearScreen() { clear(); }

void DisplayManager::update() {
    // Actualizează afişajul folosind modelul firstPage/nextPage.

    display.firstPage();
}

```

```

do {
    // Conținutul este deja desenat prin alte metode.
} while (display.nextPage());

}

void DisplayManager::hibernate() { display.hibernate(); }

void DisplayManager::welcomeMessage() {
    // Afisează ecranul de bun venit la pornirea dispozitivului.
    display.firstPage();

    do {
        display.fillScreen(GxEPD_WHITE);
        display.setTextColor(GxEPD_BLACK);
        display.setTextSize(2);
        int16_t baseY = 20;
        // Titlul principal
        drawCenteredText("SMART TRAFFIC", baseY, 2);
        drawCenteredText("SIGN", baseY + 20, 2);

        // Mesaj de conectare
        display.setTextSize(1);
        baseY += 45;
        drawCenteredText("Conectare BLE...", baseY, 1);

    } while (display.nextPage());
}

void DisplayManager::showTrafficSign(const char* sign) {
    // Funcție router pentru afișarea semnului de circulație specificat.
    if (strcmp(sign, "STOP") == 0)      showStopSign();
    else if (strcmp(sign, "YIELD") == 0)   showYieldSign();
    else if (strncmp(sign, "SPEED_LIMIT_", 12) == 0) showSpeedLimitSign(atoi(sign + 12));
    else {
        // Afisează mesaj pentru semnul necunoscut.
        display.firstPage();
    }
}

```

```

do {
    display.fillScreen(GxEPD_WHITE);
    display.setTextColor(GxEPD_BLACK);
    display.setTextSize(1);
    drawCenteredText("Semn necunoscut:", 20, 1);
    drawCenteredText(sign, 40, 2);
} while (display.nextPage());
}

}

void DisplayManager::showStopSign() {
    // Afisează semnul STOP ca un octogon cu text centrat.
    display.firstPage();
    do {
        display.fillScreen(GxEPD_WHITE);
        display.setTextColor(GxEPD_BLACK);

        // Calculează coordonatele centrului ecranului.
        const int16_t cx = display.width() / 2;
        const int16_t cy = display.height() / 2;

        // Calculează razele octoanelor exterior și interior.
        const int16_t rOuter = STOP_OCT_RADIUS;
        const int16_t rInner = rOuter - STOP_BORDER_GAP;

        // Desenează octagonul exterior.
        int16_t k = (int16_t)round(rOuter / 1.41421356f);    // r/sqrt(2)
        display.drawLine(cx - k, cy - rOuter, cx + k, cy - rOuter, GxEPD_BLACK);
        display.drawLine(cx + k, cy - rOuter, cx + rOuter, cy - k, GxEPD_BLACK);
        display.drawLine(cx + rOuter, cy - k, cx + rOuter, cy + k, GxEPD_BLACK);
        display.drawLine(cx + rOuter, cy + k, cx + k, cy + rOuter, GxEPD_BLACK);
        display.drawLine(cx + k, cy + rOuter, cx - k, cy + rOuter, GxEPD_BLACK);
        display.drawLine(cx - k, cy + rOuter, cx - rOuter, cy + k, GxEPD_BLACK);
    }
}

```

```

display.drawLine(cx - rOuter, cy + k, cx - rOuter, cy - k, GxEPD_BLACK);
display.drawLine(cx - rOuter, cy - k, cx - k, cy - rOuter, GxEPD_BLACK);

// Desenează octogonul interior (chenar dublu).

k = (int16_t)round(rInner / 1.41421356f);           // (r-Δ)/√2
display.drawLine(cx - k, cy - rInner, cx + k, cy - rInner, GxEPD_BLACK);
display.drawLine(cx + k, cy - rInner, cx + rInner, cy - k, GxEPD_BLACK);
display.drawLine(cx + rInner, cy - k, cx + k, cy + rInner, GxEPD_BLACK);
display.drawLine(cx + rInner, cy + k, cx + k, cy + rInner, GxEPD_BLACK);
display.drawLine(cx + k, cy + rInner, cx - k, cy + rInner, GxEPD_BLACK);
display.drawLine(cx - k, cy + rInner, cx - rInner, cy + k, GxEPD_BLACK);
display.drawLine(cx - rInner, cy + k, cx - rInner, cy - k, GxEPD_BLACK);
display.drawLine(cx - rInner, cy - k, cx - k, cy - rInner, GxEPD_BLACK);

// Text STOP centrat

const char* txt = "STOP";
int16_t x1, y1; uint16_t w, h;
display.setTextSize(STOP_TEXT_SIZE);
display.getTextBounds(txt, 0, 0, &x1, &y1, &w, &h);

int16_t baselineY = cy - (y1 + h / 2) + STOP_TEXT_V_ADJ; // perfect centrat
display.setCursor(cx - w / 2, baselineY);
display.print(txt);

} while (display.nextPage());
}

void DisplayManager::showYieldSign() {
    // Afisează semnul de cedează trecerea ca un triunghi cu text explicativ.

    display.firstPage();
    do {
        display.fillScreen(GxEPD_WHITE);
        display.setTextColor(GxEPD_BLACK);

```

```

const int16_t cx = display.width() / 2;
const int16_t cy = display.height() / 2;

// triunghi exterior (vârf în JOS)
int16_t s = YIELD_TRIANGLE_SIZE;
display.drawTriangle(
    cx - s, cy - s, // colț stânga sus
    cx + s, cy - s, // colț dreapta sus
    cx,   cy + s, // vârf jos
    GxEPD_BLACK);

// triunghi interior (chenar dublu)
int16_t s2 = s - YIELD_BORDER_GAP;
display.drawTriangle(
    cx - s2, cy - s2,
    cx + s2, cy - s2,
    cx,   cy + s2,
    GxEPD_BLACK);

// text informativ sub semn (optional)
display.setTextSize(YIELD_TEXT_SIZE);
drawCenteredText("CEDEAZĂ", cy + s + 10, YIELD_TEXT_SIZE);
drawCenteredText("TRECEREA", cy + s + 22, YIELD_TEXT_SIZE);

} while (display.nextPage());
}

// semn limită de viteză
void DisplayManager::showSpeedLimitSign(int limit) {
    char buf[8]; sprintf(buf, "%d", limit);

    display.firstPage();
    do {
        display.fillScreen(GxEPD_WHITE);

```

```

display.setTextColor(GxEPD_BLACK);

int16_t cx = display.width() / 2;
int16_t cy = display.height() / 2;
int16_t r = min(display.width(), display.height()) / 2 - CIRCLE_MARGIN;

// Cercul dublu cu grosime ajustabilă
display.drawCircle(cx, cy, r, GxEPD_BLACK);
display.drawCircle(cx, cy, r - CIRCLE_BORDER_GAP, GxEPD_BLACK);

```

TrafficAlertReceiver.cpp

```

#include "TrafficAlertReceiver.h"

#include <esp_wifi.h> // Necesar pentru functia esp_wifi_set_protocol.

// Initializarea pointerului static pentru accesul din callback-uri.
TrafficAlertReceiver* TrafficAlertReceiver::_instance = nullptr;

TrafficAlertReceiver::TrafficAlertReceiver(DisplayManager* displayManager)
: _displayManager(displayManager), _isInitialized(false) {
    // Setăm instanța statică pentru a fi accesată din callback-uri.
    _instance = this;
    // Inițializăm adresa MAC a lui Elysium cu zero până când este setată explicit.
    memset(_elysiumMacAddress, 0, 6);
}

void TrafficAlertReceiver::init() {
    // Nu mai setăm WiFi.mode(WIFI_STA) pentru a nu interfera cu BLE.
    // În schimb, configurăm WiFi pentru coexistență cu BLE.
    esp_wifi_set_protocol(WIFI_IF_STA, WIFI_PROTOCOL_11B | WIFI_PROTOCOL_11G | WIFI_PROTOCOL_LR);

    // Inițializare sistem ESP-Now.
    if (esp_now_init() != ESP_OK) {

```

```

// În caz de eroare, se reîncearcă după restart.

ESP.restart();
return;
}

// Înregistram callback-ul pentru receptie.

esp_now_register_recv_cb(onDataReceived);
_isInitialized = true;
}

void TrafficAlertReceiver::setElysiumMac(const uint8_t* mac) {
    memcpy(_elysiumMacAddress, mac, 6);

    // Creare informație peer pentru comunicarea cu Elysium RC.

    esp_now_peer_info_t peerInfo = {};
    memcpy(peerInfo.peer_addr, _elysiumMacAddress, 6);
    peerInfo.channel = 1;           // ACELAȘI canal pe care este setată stația WiFi (1).
    peerInfo.encrypt = false;

    // Adăugare peer pentru a permite receptia mesajelor.

    esp_now_add_peer(&peerInfo);
}

void TrafficAlertReceiver::onDataReceived(const esp_now_recv_info_t* info, const uint8_t* data, int data_len) {
    // Verifică dacă avem o instanță validă.

    if (!_instance) return;

    // Verifică dacă mesajul este de la vehiculul Elysium.

    if (memcmp(info->src_addr, _instance->_elysiumMacAddress, 6) == 0) {
        // Verifică dacă avem suficiente date pentru a forma un mesaj complet.

        if (data_len >= sizeof(ElysiumMessage)) {
            // Conversia datelor brute primite la structura de mesaj definită.

            ElysiumMessage message;
            memcpy(&message, data, sizeof(ElysiumMessage));
        }
    }
}

```

```

// Procesarea mesajului pentru acțiuni corespunzătoare.

_instance->processMessage(message);

}

}

}

void TrafficAlertReceiver::processMessage(const ElysiumMessage& message) {

// Dacă nu avem un display manager valid, nu putem afișa nimic.

if (!_displayManager) return;

// În funcție de tipul evenimentului, actualizăm conținutul afișajului.

switch (message.eventType){

case EVENT_ACCIDENT:

// Afisează semnalul de accident pe display.

_displayManager->showTrafficSign("ACCIDENT");

break;

case EVENT_OBSTACLE:

// Afisează semnalul de obstacol pe display.

_displayManager->showTrafficSign("OBSTACOL");

break;

case EVENT_EMERGENCY:

// Afisează semnalul de urgență pe display.

_displayManager->showTrafficSign("URGENTA");

break;

case EVENT_NORMAL:

default:

// Afisează semnul normal pentru acest indicator de trafic.

// (pentru semnul 1, configurat implicit ca fiind STOP).

_displayManager->showTrafficSign("STOP");

break;

}
}

```

traffic_sign_1.ino

```
#include <Arduino.h>
#include <SPI.h>
#include "Config.h"

// Declarații înainte de folosire.

extern volatile bool propaga_mesaj_urgenta;

extern struct traffic_message mesaj_urgenta_de_propagat; // Declarația structurii va fi definită mai jos.

// Variabilele pentru propagarea mesajelor de urgență vor fi definite mai jos, după definirea structurii
traffic_message.

#include "DisplayManager.h"
#include "BleManager.h"
#include "ESP32_NOW.h"
#include <WiFi.h>
#include <esp_wifi.h>
#include <esp_mac.h> // Pentru macrourile MAC2STR și MACSTR
#include <vector>

// Dezactivăm modulul TrafficAlertReceiver deoarece funcționalitatea sa
// a fost integrată în implementarea ESP32_NOW.

#define TRAFFIC_ALERT_RECEIVER_DISABLED 1

// Declarăm extern variabilele care vor fi folosite în clase.

extern DisplayManager epaperDisplay;
extern BleManager bleManager;

#define ESPNOW_WIFI_CHANNEL 6 // Folosim canalul 6 pentru compatibilitate cu exemplul master.

// Tipuri de evenimente ce pot fi primite de la vehiculul Elysium RC.
```

```

enum ElysiumEventType {
    EVENT_NORMAL = 0,
    EVENT_ACCIDENT = 1,
    EVENT_OBSTACLE = 2,
    EVENT_EMERGENCY = 3
};

// Structura mesajelor primite de la vehicule prin ESP-NOW (preluate din TrafficAlertReceiver).
typedef struct __attribute__((packed)) ElysiumMessage {
    uint8_t eventType; // Tipul evenimentului (conform enum ElysiumEventType).
    uint8_t severity; // Severitatea evenimentului (valori de la 1 la 10).
    char location[32]; // Locația unde s-a produs evenimentul.
} ElysiumMessage;

// Structura datelor care vor fi transmise prin ESP-Now între semne de trafic.
typedef struct __attribute__((packed)) traffic_message {
    uint8_t targetId; // 0 = broadcast, altfel ID specific al semnului.
    char signType[20]; // Tipul semnului: "STOP", "ACCIDENT" etc.
    uint8_t priority; // Prioritatea mesajului: 0 = normal, 1 = urgent.
} traffic_message;

// Variabile globale dependente de structura traffic_message.
volatile bool propaga_mesaj_urgenta = false;
traffic_message mesaj_urgenta_de_propagat;

class ESP_NOW_Peer_Class : public ESP_NOW_Peer {
public:
    // Constructor.
    ESP_NOW_Peer_Class(const uint8_t *mac_addr, uint8_t channel, wifi_interface_t iface, const uint8_t *lmk) :
        ESP_NOW_Peer(mac_addr, channel, iface, lmk) {}

    // Destructor.
    ~ESP_NOW_Peer_Class() {}
}

```

```

// Funcție pentru înregistrarea peer-ului master.

bool add_peer() {
    if (!add()) {
        log_e("Eroare la înregistrarea peer-ului broadcast");
        return false;
    }
    return true;
}

// Funcție publică de trimitere a unui mesaj (wrapper peste metoda protected send).

bool send_message(const uint8_t *data, size_t len) {
    return send(data, (int)len) == len;
}

// Funcție pentru procesarea mesajelor primite de la master.

void onReceive(const uint8_t *data, size_t len, bool broadcast) {
    Serial.printf("Mesaj primit de la master " MACSTR " (%s)\n", MAC2STR(addr()), broadcast ? "broadcast" : "unicast");

    // Afisăm conținutul mesajului pentru debugging
    Serial.print("Conținut mesaj (text): ");
    for (int i = 0; i < len && i < 32; i++) {
        Serial.print((char)data[i]);
    }
    Serial.println(" ");

    Serial.print("Conținut mesaj (hex): ");
    for (int i = 0; i < len && i < 16; i++) {
        Serial.printf("%02X ", data[i]);
    }
    Serial.println();
}

// Analizăm tipul de mesaj primit și îl procesăm corespunzător

```

```

if (len == sizeof(traffic_message)) {
    // Este un mesaj de la un alt semn de circulație
    processTrafficSignMessage(data, len, broadcast);
}

else if (len == sizeof(ElysiumMessage)) {
    // Este un mesaj de la un vehicul (de ex. Elysium RC)
    processVehicleMessage(data, len, broadcast);
}

else {
    // Mesaj necunoscut sau text simplu - încercăm să îl procesăm ca text
    // Creăm un buffer terminat cu NULL pentru a gestiona corect sirul de caractere
    char textBuffer[33]; // 32 caractere + NULL terminator
    memset(textBuffer, 0, sizeof(textBuffer));
    int copyLen = min(len, sizeof(textBuffer)-1);
    memcpy(textBuffer, data, copyLen);

    Serial.printf(" Mesaj text primit: %s\n", textBuffer);

    // Pentru testare, vom afișa mesaj pe semn când primim mesaje text
    // Extragem numărul mesajului pentru afișare
    char* numberPos = strstr(textBuffer, "#");
    if (numberPos) {
        epaperDisplay.showTrafficSign(numberPos); // Afisăm doar partea cu numărul
    } else {
        epaperDisplay.showTrafficSign("TEST");
    }

    // Notificăm aplicația Android
    String status = "SignID=" + String(SIGN_ID) + ";Event=TEXT_MESSAGE;Content=";
    status += String(textBuffer);
    bleManager.sendStatusUpdate(status);
}
}

```

```

// Procesare mesaje de la alte semne de circulație.

void processTrafficSignMessage(const uint8_t *data, size_t len, bool broadcast){
    traffic_message *message = (traffic_message*) data;

    // Verificăm dacă mesajul este pentru acest semn sau broadcast
    if (message->targetId == 0 || message->targetId == SIGN_ID){
        Serial.printf("Mesaj primit de la alt semn pentru semnul %d\n", SIGN_ID);
        Serial.printf("Tip de semn: %s, Prioritate: %d\n", message->signType, message->priority);

        // Ne ocupăm de mesaj în funcție de prioritate
        if (message->priority > 0){
            // Mesaj prioritар (accident, urgență, obstacol)
            Serial.println("ATENȚIE: Mesaj prioritар primit!");

            // Verificăm dacă mesajul conține un eveniment de tip accident
            if (strstr(message->signType, "ACCIDENT") != NULL){
                epaperDisplay.showTrafficSign("ACCIDENT");
                Serial.println("Afișez ACCIDENT pe display");
            }
            else if (strstr(message->signType, "OBSTACLE") != NULL || strstr(message->signType, "OBSTACOL") != NULL){
                epaperDisplay.showTrafficSign("OBSTACOL");
                Serial.println("Afișez OBSTACOL pe display");
            }
            else if (strstr(message->signType, "EMERGENCY") != NULL || strstr(message->signType, "URGENTA") != NULL){
                epaperDisplay.showTrafficSign("URGENTA");
                Serial.println("Afișez URGENTA pe display");
            }
            else{
                // Alt tip de mesaj prioritар, afișăm direct conținutul
                epaperDisplay.showTrafficSign(message->signType);
            }
        }
    }
}

```

```

// Notificăm aplicația Android despre eveniment priorită
String status = "SignID=" + String(SIGN_ID) + ";Event=" + String(message->signType);
status += ";Priority=" + String(message->priority) + ";Source=OtherSign";
bleManager.sendStatusUpdate(status);

} else {
    // Mesaj normal (schimbare de semn)
    Serial.printf("Actualizez semnul cu: %s\n", message->signType);
    epaperDisplay.showTrafficSign(message->signType);

    // Notificăm aplicația Android despre schimbarea normală
    String status = "SignID=" + String(SIGN_ID) + ";Event=SignChange;" +
        "Sign=" + String(message->signType) + ";Ack=OK";
    bleManager.sendStatusUpdate(status);
}

} else {
    // Mesaj pentru alt semn - îl ignorăm
    Serial.printf("Mesaj destinat semnului %d - ignorat.\n", message->targetId);
}
}

// Procesare mesaje de la vehicule (Elysium sau alte vehicule compatibile).
void processVehicleMessage(const uint8_t *data, size_t len, bool broadcast) {
    ElysiumMessage *message = (ElysiumMessage*) data;

    Serial.print("Mesaj de la vehicul. Tip eveniment: ");
    String eventType;
    String eventDisplay;
    bool isEmergency = false; // Flag pentru evenimentele care necesită propagare

    // În funcție de tipul evenimentului, actualizăm afișajul
    switch (message->eventType) {
        case EVENT_ACCIDENT:

```

```

eventType = "ACCIDENT";
eventDisplay = "ACCIDENT";
Serial.println("Accident");
Serial.printf("Accident raportat în locația: %s. Severitate: %d\n",
             message->location, message->severity);
isEmergency = true;
break;

case EVENT_OBSTACLE:
eventType = "OBSTACLE";
eventDisplay = "OBSTACOL";
Serial.println("Obstacol");
Serial.printf("Obstacol raportat în locația: %s. Severitate: %d\n",
             message->location, message->severity);
isEmergency = true;
break;

case EVENT_EMERGENCY:
eventType = "EMERGENCY";
eventDisplay = "URGENTA";
Serial.println("Urgență");
Serial.printf("Urgență raportată în locația: %s. Severitate: %d\n",
             message->location, message->severity);
isEmergency = true;
break;

case EVENT_NORMAL:
default:
eventType = "NORMAL";
eventDisplay = "STOP"; // Semnul default pentru acest semn
Serial.println("Normal/Default");
break;
}

```

```

// Afişăm semnul corespunzător
epaperDisplay.showTrafficSign(eventDisplay.c_str());

// Notificăm aplicația Android cu detalii despre vehicul și eveniment
String status = "SignID=" + String(SIGN_ID) + ";Event=" + eventType;
status += ";Location=" + String(message->location);
status += ";Severity=" + String(message->severity);
status += ";Time=" + String(millis());
bleManager.sendStatusUpdate(status);

// Dacă este un eveniment de urgență (accident, obstacol, etc.),
// îl vom semnală pentru propagare
if (isEmergency && !broadcast) {
    // Creăm un mesaj de trafic care va fi transmis din exteriorul clasei
    static traffic_message emergencyMessage;
    emergencyMessage.targetId = 0; // 0 = broadcast
    strncpy(emergencyMessage.signType, eventType.c_str(), sizeof(emergencyMessage.signType)-1);
    emergencyMessage.signType[sizeof(emergencyMessage.signType)-1] = '\0'; // Asigurăm NULL terminator
    emergencyMessage.priority = 1; // 1 = urgent

    // Setăm un flag global pentru a indica necesitatea propagării
    propaga_mesaj_urgenta = true;
    memcpy(&mesaj_urgenta_de_propagat, &emergencyMessage, sizeof(traffic_message));
}

Serial.printf("Am marcat evenimentul %s pentru propagare către alte semne\n", eventType.c_str());
}

};

/* Variabile globale pentru ESP-NOW */
// Lista cu toți master-ii. Va fi populată când un nou master este înregistrat.
std::vector<ESP_NOW_Peer_Class> masters;

```

```

bool welcomeShown = false;
unsigned long welcomeStartTime = 0;
const unsigned long WELCOME_DURATION = 5000; // Durata de afişare a ecranului de bun venit (5 secunde).

// Inițializare BLE pentru comunicare cu aplicația Android.
BleManager bleManager(&epaperDisplay);

// Callback pentru înregistrarea unui nou master ESP-NOW.
void register_new_master(const esp_now_recv_info_t *info, const uint8_t *data, int len, void *arg) {
    // Adăugăm debug pentru toate mesajele primite.
    Serial.println("DEBUG: Callback onNewPeer a fost apelat!");
    Serial.printf("DEBUG: Mesaj primit de la " MACSTR " pentru " MACSTR ", len=%d\n",
                 MAC2STR(info->src_addr), MAC2STR(info->des_addr), len);
    {{ ... }}
}

// Eliberare memorie Bluetooth Classic (doar BLE necesar).
Serial.println("1. Eliberare memorie Bluetooth Classic");
esp_bt_controller_mem_release(ESP_BT_MODE_CLASSIC_BT);

// Inițializare comunicație serială.
Serial.println("2. Inițializare BLE Manager");
bleManager.init();
Serial.println(" BLE inițializat cu succes");

// Configurare WiFi în modul STA pentru compatibilitate cu ESP-Now.
Serial.println("3. Pornire WiFi în modul STA");
WiFi.mode(WIFI_STA);
WiFi.setChannel(ESPNOW_WIFI_CHANNEL);
// Forțăm canalul WiFi să fie 6 pentru compatibilitate cu exemplul master
esp_wifi_set_channel(ESPNOW_WIFI_CHANNEL, WIFI_SECOND_CHAN_NONE);
while (!WiFi.STA.started()) {
    {{ ... }}
}

```

```

Serial.println(" Mod: STA");
Serial.println(" Adresa MAC: " + WiFi.macAddress());
Serial.printf(" Canal: %d\n", ESPNOW_WIFI_CHANNEL);

// Inițializare protocol ESP-NOW pentru comunicare cu orice dispozitiv.
Serial.println("4. Inițializare sistem ESP-NOW");
if (!ESP_NOW.begin()) {
    Serial.println("Eroare la inițializarea ESP-NOW!");
    Serial.println("Reporonire în 5 secunde...");
    delay(5000);
    ESP.restart();
}

Serial.println("5. Inițializare display");
epaperDisplay.init();
Serial.println(" Display inițializat");

// Afisarea mesajului de bun venit.
Serial.println("6. Afisare mesaj de bun venit pe display");
epaperDisplay.welcomeMessage();

// Inițializarea cronometrului pentru tranziția intre ecrane.
welcomeStartTime = millis();
welcomeShown = true;
{

Serial.println("Inițializare completă. Sistem pregătit pentru comenzi BLE și mesaje ESP-NOW.");
Serial.println("Aștept mesaje broadcast de la dispozitive master...");
}

```

```

void loop() {
    // Dacă s-a afișat ecranul de bun venit și au trecut cele 5 secunde.
    if (welcomeShown && (millis() - welcomeStartTime > WELCOME_DURATION)) {
        // Afișăm semnul STOP și punem display-ul în hibernare (inclus în metodă).
        epaperDisplay.showTrafficSign("STOP");

        // Resetăm flag-ul pentru a nu mai intra în această condiție.
        welcomeShown = false;
    }

    // Trimitem un status update prin BLE.
    bleManager.sendStatusUpdate("Ready to receive commands");
}

// BLE este gestionat automat de biblioteca BleManager.

// Propagăm mesajele de urgență dacă este necesar.
if (propaga_mesaj_urgenta) {
    Serial.println("Propagare mesaj de urgență către toate semnele înregistrate");
}

// Trimitem mesajul către toate semnele înregistrate.
bool cel_putin_unul_trimes = false;
for (auto &master : masters) {
    Serial.printf("Trimit mesajul de urgență '%s' către un alt semn\n", mesaj_urgenta_de_propagat.signType);
    if (master.send_message((uint8_t*)&mesaj_urgenta_de_propagat, sizeof(traffic_message))) {
        cel_putin_unul_trimes = true;
        Serial.println("Mesaj trimis cu succes!");
    } else {
        Serial.println("Eroare la trimiterea mesajului!");
    }
}

if (!cel_putin_unul_trimes) {
    Serial.println("Atenție: Niciun semn nu a putut fi notificat!");
}

```

```

}

// Resetăm flag-ul
propaga_mesaj_urgenta = false;
}

// Afisăm periodic informații despre starea ESP-NOW.

static unsigned long lastDebugTime = 0;

if (millis() - lastDebugTime > 10000) { // La fiecare 10 secunde.

lastDebugTime = millis();

Serial.printf("DEBUG: Stare ESP-NOW - Total peers: %d\n", ESP_NOW.getTotalPeerCount());

Serial.printf("DEBUG: Număr masters înregistrati: %d\n", masters.size());

Serial.println("DEBUG: Aștept în continuare mesaje broadcast...");

Serial.println("DEBUG: Adresa MAC locală: " + WiFi.macAddress());

Serial.printf("DEBUG: Canal WiFi: %d\n", WiFi.channel());



// Verificăm dacă canalul WiFi coincide cu cel configurat pentru ESP-NOW.

if (WiFi.channel() != ESPNOW_WIFI_CHANNEL) {

Serial.printf("AVERTISMENT: Canalul WiFi curent (%d) nu coincide cu ESPNOW_WIFI_CHANNEL (%d)\n",
WiFi.channel(), ESPNOW_WIFI_CHANNEL);

}

}

delay(100);
}

```

Aplicația Android

MainActivity.kt

```
package com.example.smarttrafficsigns

import android.Manifest
import android.bluetooth.BluetoothAdapter
import android.bluetooth.BluetoothDevice
import android.bluetooth.BluetoothManager
import android.content.Context
import android.os.Build
import android.os.Bundle
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import com.example.smarttrafficsigns.NotificationUtils
import com.example.smarttrafficsigns.ble.BleConnection
import com.example.smarttrafficsigns.ble.BleScanner
import com.example.smarttrafficsigns.ble.ClassicScanner
import com.example.smarttrafficsigns.ble.DeviceConnection
import com.example.smarttrafficsigns.ble.SppConnection
import com.example.smarttrafficsigns.ui.ElysiumConnectScreen
import com.example.smarttrafficsigns.ui.TagWriteScreen
import com.example.smarttrafficsigns.ui.DeviceListScreen
import com.example.smarttrafficsigns.ui.RequestPermissions
import com.example.smarttrafficsigns.ui.SignControlScreen
import com.example.smarttrafficsigns.ui.theme.SmartTrafficSignsTheme
import com.google.accompanist.permissions.ExperimentalPermissionsApi
```

```

import com.google.accompanist.permissions.isGranted
import com.google.accompanist.permissions.rememberMultiplePermissionsState

/**
 * Activitatea principală care gestionează scanarea, conectarea și controlul semnelor de trafic prin BLE și
Bluetooth Classic.
 *
 * Aceasta se ocupă de gestionarea permisiunilor, conexiunile multiple simultan și interfața utilizator.
 */

class MainActivity : ComponentActivity() {

    private lateinit var bleScanner: BleScanner
    private lateinit var classicScanner: ClassicScanner
    // Gestionăm mai multe conexiuni simultane (una per dispozitiv).
    private val connections = mutableStateMapOf<String, DeviceConnection>()
    // Inițializăm adaptorul Bluetooth folosind lazy initialization pentru eficiență.

    private val bluetoothAdapter: BluetoothAdapter? by lazy {
        val bluetoothManager = getSystemService(Context.BLUETOOTH_SERVICE) as BluetoothManager
        bluetoothManager.adapter
    }

    /**
     * Inițializează activitatea, scannerele BLE și Bluetooth Classic și configerează interfața utilizator.
     */
    @OptIn(ExperimentalPermissionsApi::class)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Inițializare scannere pentru dispozitive Bluetooth LE și Classic.
        bleScanner = BleScanner(this)
        classicScanner = ClassicScanner(this)

        setContent {
            SmartTrafficSignsTheme {
                Surface(

```

```

        modifier = Modifier.fillMaxSize(),
        color = MaterialTheme.colorScheme.background
    ) {
        MainContent()
    }
}

}

}

/***
 * Conținutul principal al aplicației care verifică permisiunile și afișează ecranul corespunzător.
 */
@OptIn(ExperimentalPermissionsApi::class)
@Composable
private fun MainContent() {
    // Verificăm permisiunile necesare pentru BLE și localizare.
    val permissionsRequired = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        listOf(
            Manifest.permission.BLUETOOTH_SCAN,
            Manifest.permission.BLUETOOTH_CONNECT,
            Manifest.permission.ACCESS_FINE_LOCATION
        )
    } else {
        listOf(
            Manifest.permission.ACCESS_FINE_LOCATION
        )
    }

    val permissionsState = rememberMultiplePermissionsState(permissionsRequired)
    val allPermissionsGranted = permissionsState.permissions.all { it.status.isGranted }

    if (allPermissionsGranted) {
        BleContent()
    }
}

```

```

    } else {
        RequestPermissions(permissionsState)
    }
}

/**
 * Conținutul principal pentru funcționalitatea Bluetooth - afișează lista de dispozitive,
 * ecranul de control pentru semne sau ecranul de scriere RFID în funcție de starea curentă.
 */
@Composable
private fun BleContent() {
    // Dispozitivul pentru care afișăm ecranul de control.
    var activeDevice by remember { mutableStateOf<BluetoothDevice?>(null) }

    // Flag pentru modul de scriere RFID tag.
    var tagWriteMode by remember { mutableStateOf(false) }

    // Ultima comandă trimisă către semn (folosită la scrierea RFID).
    var lastSignCommand by remember { mutableStateOf<String?>(null) }

    // Conexiunea activă (BLE sau SPP)
    val activeConn = activeDevice?.let { connections[it.address] }

    val bleConn = activeConn as? BleConnection
    val sppConn = activeConn as? SppConnection

    val bleState by bleConn?.connectionState?.collectAsState() ?: remember {
        mutableStateOf(BleConnection.ConnectionState.DISCONNECTED)
    }

    val sppState by sppConn?.connectionState?.collectAsState() ?: remember {
        mutableStateOf(SppConnection.State.DISCONNECTED)
    }

    val statusMessage by activeConn?.statusMessage?.collectAsState() ?: remember { mutableStateOf("") }

    // Trimit notificare dacă mesajul conține "Accident"
    LaunchedEffect(statusMessage) {
        statusMessage?.let { msg ->
            val lower = msg.lowercase()
        }
    }
}

```

```

        if (lower.contains("accident") || lower.contains("accid")) {
            NotificationUtils.showAccidentNotification(this@MainActivity, msg)
        }
    }
}

// Colectăm lista de dispozitive
val bleDevices by bleScanner.devices.collectAsState()
val classicDevices by classicScanner.devices.collectAsState()
val devicesList = remember(bleDevices, classicDevices) {
    val merged = bleDevices.toMutableList()
    classicDevices.forEach { dev ->
        if (merged.none { it.address == dev.address }) merged.add(dev)
    }
    merged.sortedBy { it.name ?: it.address }
}

if (activeDevice == null) {
    // Dacă nu suntem conectați la niciun dispozitiv, afișăm lista de dispozitive
    DeviceListScreen(
        devices = devicesList,
        connectedMacs = connections.keys,
        onRefresh = {
            bleScanner.startScan()
            classicScanner.startScan()
        },
        onDeviceSelected = { device ->
            bleScanner.stopScan()
            classicScanner.stopScan()
            // Dacă nu avem deja o conexiune pentru acest dispozitiv o creăm
            val mac = device.address
            val connection = connections.getOrPut(mac) {
                if (device.type == BluetoothDevice.DEVICE_TYPE_CLASSIC || device.type ==
                    BluetoothDevice.DEVICE_TYPE_DUAL)

```

```

        SppConnection().apply{ connect(device) }

    } else {
        BleConnection(this@MainActivity).apply{ connect(device) }
    }
}

activeDevice = device

Toast.makeText(
    this@MainActivity,
    "Conectare la ${device.name ?: device.address}...",
    Toast.LENGTH_SHORT
).show()
}

}

} else {
// Dacă suntem conectați, afișăm ecranul de control
if (tagWriteMode) {
    TagWriteScreen(
        deviceName = activeDevice?.name ?: "Tag Write",
        writeEnabled = bleState == BleConnection.ConnectionState.CONNECTED || sppState ==
SppConnection.State.CONNECTED,
        statusMessage = statusMessage ?: "",
        onWriteTag = {
            activeDevice?.let { dev ->
                connections.values.firstOrNull{ it is com.example.smarttrafficsigns.ble.SppConnection
}??.sendCommand("{"action":"WRITE_TAG","type":${lastSignCommand ?: ""}}")
            } ?: false
        },
        onExit = {
            tagWriteMode = false
        }
    )
} else if (sppConn != null) {
    ElysiumConnectScreen(
        deviceName = activeDevice?.name ?: activeDevice?.address ?: "Elysium RC",

```

```

connectionState = sppState,
statusMessage = statusMessage,
onDisconnect = {
    activeDevice?.let { dev ->
        connections[dev.address]?.disconnect()
        connections.remove(dev.address)
    }
    activeDevice = null
    bleScanner.startScan()
    classicScanner.startScan()
},
onBack = {
    activeDevice = null
    bleScanner.startScan()
    classicScanner.startScan()
}
}

} else
SignControlScreen(
    deviceName = activeDevice?.name ?: activeDevice?.address ?: "Dispozitiv",
    connectionState = bleState,
    statusMessage = statusMessage ?: "",
    onSendCommand = { command ->
        val sent = activeDevice?.let { dev -> connections[dev.address]?.sendCommand(command) } ?: false
        if(sent) lastSignCommand = command
        tagWriteMode = true
        sent
    },
    onDisconnect = {
        activeDevice?.let { dev ->
            connections[dev.address]?.disconnect()
            connections.remove(dev.address)
        }
    }
}

```

```

activeDevice = null
bleScanner.startScan()
classicScanner.startScan()

},
onBack = {
    // Nu deconectăm, doar revenim la listă
    activeDevice = null
    bleScanner.startScan()
    classicScanner.startScan()
}
}

}

}

}

/***
* Se apelează când activitatea devine vizibilă. Pornim scanarea pentru dispozitive Bluetooth.
*/
override fun onResume() {
    super.onResume()
    // Pornim scanarea pentru a descoperi dispozitive disponibile.
    bleScanner.startScan()
    classicScanner.startScan()
}

/***
* Se apelează când activitatea nu mai este în prim-plan. Oprim scanarea pentru economisirea bateriei.
*/
override fun onPause() {
    super.onPause()
    // Oprim scanarea când aplicația nu este în prim-plan pentru a economisi bateria.
    bleScanner.stopScan()
    classicScanner.stopScan()
}

```

```

/**
 * Se apelează când activitatea este distrusă. Eliberează toate resursele și închide conexiunile.
 */
override fun onDestroy() {
    super.onDestroy()
    // Oprim toate scannerele active.
    bleScanner.stopScan()
    classicScanner.stopScan()
    // Închide toate conexiunile active pentru a evita leak-urile de resurse.
    connections.values.forEach { it.close() }
    // Eliberează resursele scannerului Bluetooth Classic.
    classicScanner.close()
}
}

```

NotificationUtils.kt

```

package com.example.smarttrafficsigns

import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.Context
import android.graphics.Color
import android.os.Build
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat

/**
 * Utilitar pentru gestionarea și afișarea notificărilor despre evenimentele primite de la semnele de trafic.
 * Asigură crearea canalelor de notificare necesare pentru Android 8.0+ și formatarea corespunzătoare
 * a notificărilor pentru evenimente importante precum accidentele.

```

```

/*
object NotificationUtils {

    // Constante pentru configurarea canalului de notificări.

    private const val CHANNEL_ID = "traffic_sign_alerts"
    private const val CHANNEL_NAME = "Alerte Semn Trafic"
    private const val CHANNEL_DESC = "Notificări despre evenimente rutiere primite de la semnele de trafic"

    /**
     * Asigură crearea canalului de notificare pentru versiuni Android Oreo (API 26) și mai noi.
     * Canalul este configurat cu importanță ridicată, lumină roșie și vibrații pentru alertele de trafic.
     *
     * @param context Contextul aplicației pentru accesarea serviciului de notificări.
     */
    private fun ensureChannel(context: Context) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val manager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
            if (manager.getNotificationChannel(CHANNEL_ID) == null) {
                val channel = NotificationChannel(CHANNEL_ID, CHANNEL_NAME,
                    NotificationManager.IMPORTANCE_HIGH).apply {
                    description = CHANNEL_DESC
                    enableLights(true)
                    lightColor = Color.RED
                    enableVibration(true)
                }
                manager.createNotificationChannel(channel)
            }
        }
    }

    /**
     * Afisează o notificare de importanță ridicată pentru alertarea utilizatorului în cazul unui accident.
     * Notificarea include o iconiță de eroare, vibrații și se închide automat la apasare.
     */
}

```

```

    * @param context Contextul aplicației pentru crearea notificării.
    * @param message Mesajul detaliat despre accidentul raportat care va fi afișat în notificare.

    */

fun showAccidentNotification(context: Context, message: String) {
    ensureChannel(context)
    val builder = NotificationCompat.Builder(context, CHANNEL_ID)
        .setSmallIcon(android.R.drawable.stat_notify_error)
        .setContentTitle("Accident raportat")
        .setContentText(message)
        .setPriority(NotificationCompat.PRIORITY_HIGH)
        .setAutoCancel(true)
        .setVibrate(longArrayOf(0, 500, 200, 500))

    with(NotificationManagerCompat.from(context)) {
        notify(1, builder.build())
    }
}

```

DeviceListScreen.kt

```

package com.example.smarttrafficsigns.ui

import android.bluetooth.BluetoothDevice
import androidx.compose.foundation.clickable
import androidx.compose.material.icons(Icons)
import androidx.compose.material.icons.filled.Refresh
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.*
import androidx.compose.material3.ExperimentalMaterial3Api

```

```

import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.foundation.background
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.ui.unit.dp

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun DeviceListScreen(
    devices: List<BluetoothDevice>,
    connectedMacs: Set<String> = emptySet(),
    onRefresh: () -> Unit = {},
    onDeviceSelected: (BluetoothDevice) -> Unit = {}
) {
    Scaffold(
        topBar = {
            TopAppBar(title = { Text(text = "Semne de circulație BLE") }, actions = {
                IconButton(onClick = onRefresh) {
                    Icon(
                        imageVector = Icons.Default.Refresh,
                        contentDescription = "Re-scan"
                    )
                }
            })
        }
    ) {
        if (devices.isEmpty()) {
            Box(
                modifier = Modifier
                    .fillMaxSize()
                    .padding(padding),

```

```
contentAlignment = Alignment.Center
) {
    Text(text = "Niciun dispozitiv găsit...")
}
} else {
    LazyColumn(
        modifier = Modifier
            .fillMaxSize()
            .padding(padding)
    ) {
        items(devices) { device ->
            val isConnected = connectedMacs.contains(device.address)
            DeviceRow(device = device, isConnected = isConnected, onClick = { onDeviceSelected(device) })
        }
    }
}
}

@Composable
private fun DeviceRow(device: BluetoothDevice, isConnected: Boolean, onClick: () -> Unit) {
    Row(
        verticalAlignment = Alignment.CenterVertically,
        modifier = Modifier
            .fillMaxWidth()
            .clickable(onClick = onClick)
            .padding(16.dp)
    ) {
        Column(modifier = Modifier.weight(1f)) {
            Text(text = device.name ?: "Dispozitiv fără nume", style = MaterialTheme.typography.bodyLarge)
            Text(text = device.address, style = MaterialTheme.typography.bodySmall)
        }
        if (isConnected) {

```

```

        Text(
            text = "CONNECTED",
            color = Color.White,
            style = MaterialTheme.typography.labelSmall,
            modifier = Modifier
                .background(color = Color(0xFF4CAF50), shape = RoundedCornerShape(4.dp))
                .padding(horizontal = 8.dp, vertical = 2.dp)
        )
    }
}

HorizontalDivider()
}

```

SignControlScreen.kt

```

package com.example.smarttrafficsigns.ui

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ArrowBack
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.material.icons.filled.Refresh
import androidx.compose.material.icons.filled.Close

```

```

import com.example.smarttrafficsigns.ble.BleConnection

/**
 * Fișierul conține implementarea ecranului pentru controlul semnelor de circulație după conectare.
 * Permite utilizatorului să trimită comenzi predefinite sau personalizate către un semn de circulație
 * prin intermediul conexiunii BLE.
 */
/**
 * Ecran pentru controlul unui semn de circulație conectat prin BLE.
 * Permite utilizatorului să selecteze și să trimită comenzi pentru schimbarea tipului de semn afișat,
 * să vizualizeze starea conexiunii, să trimită comenzi personalizate și să reseteze un semn
 * în caz de alarmă de accident.
 *
 * @param deviceName Numele dispozitivului BLE (semnului de circulație) conectat.
 * @param connectionState Starea curentă a conexiunii BLE cu semnul.
 * @param statusMessage Mesajul de status primit de la semn, poate conține informații despre stări de alarmă.
 * @param onSendCommand Callback apelat când utilizatorul dorește să trimită o comandă către semn.
 * @param onDisconnect Callback apelat când utilizatorul dorește să se deconecteze de la semn.
 * @param onBack Callback apelat când utilizatorul dorește să se întoarcă la ecranul anterior.
 */
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SignControlScreen(
    deviceName: String,
    connectionState: BleConnection.ConnectionState?,
    statusMessage: String,
    onSendCommand: (String) -> Boolean,
    onDisconnect: () -> Unit,
    onBack: () -> Unit
) {
    // Lista comenziilor predefinite disponibile pentru semnele de circulație
    val possibleCommands = remember {
        listOf("STOP", "YIELD", "SPEED_LIMIT_30", "SPEED_LIMIT_50")
    }
}

```

```

}

// Starea UI pentru comanda selectată, comanda personalizată și modul de introducere
var selectedCommand by remember { mutableStateOf(possibleCommands.first()) }

var customCommand by remember { mutableStateOf("") }

var useCustomCommand by remember { mutableStateOf(false) }

// Convertirea stării conexiunii în text descriptiv românesc pentru afișare
val connectionStatusText = when (connectionState) {
    BleConnection.ConnectionState.CONNECTED -> "Conectat"
    BleConnection.ConnectionState.CONNECTING -> "Se conectează..."
    BleConnection.ConnectionState.DISCONNECTED -> "Deconectat"
    BleConnection.ConnectionState.DISCONNECTING -> "Se deconectează..."
    BleConnection.ConnectionState.ERROR -> "Eroare"
    null -> "Necunoscut"
}

// Maparea comenzi tehnice la etichete prietenoase în limba română pentru utilizator
val commandLabels = mapOf(
    "STOP" to "STOP",
    "YIELD" to "Cedează trecerea",
    "SPEED_LIMIT_30" to "Limită 30 km/h",
    "SPEED_LIMIT_50" to "Limită 50 km/h"
)

// Descrieri detaliate pentru fiecare tip de semn disponibil
val commandDescriptions = mapOf(
    "STOP" to "Semn de oprire obligatorie",
    "YIELD" to "Cedează trecerea",
    "SPEED_LIMIT_30" to "Limită de viteză 30 km/h",
    "SPEED_LIMIT_50" to "Limită de viteză 50 km/h"
)

```

```

Scaffold(
    topBar = {
        TopAppBar(
            title = { Text("Controlul semnului: $deviceName") },
            navigationIcon = {
                IconButton(onClick = onBack) {
                    Icon(Icons.Default.ArrowBack, contentDescription = "Înapoi la lista de dispozitive")
                }
            },
            actions = {
                IconButton(onClick = onDisconnect) {
                    Icon(Icons.Default.Close, contentDescription = "Deconectează")
                }
            }
        )
    }
){ paddingValues ->
    Column(
        modifier = Modifier
            .padding(paddingValues)
            .padding(16.dp)
            .fillMaxSize()
            .verticalScroll(rememberScrollState()),
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ){
        // Card pentru afişarea stării conexiunii şi a mesajelor de status
        Card(
            modifier = Modifier.fillWidth(),
            colors = CardDefaults.cardColors(
                containerColor = if (connectionState == BleConnection.ConnectionState.CONNECTED)
                    Color(0xFF4CAF50) else MaterialTheme.colorScheme.surface,
                contentColor = if (connectionState == BleConnection.ConnectionState.CONNECTED)
                    Color.White else MaterialTheme.colorScheme.onSurface
            )
        )
    }
}

```

```

        )
    ){

    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth(),
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ){
        Text(
            text = "Status:",
            style = MaterialTheme.typography.bodyMedium,
            fontWeight = FontWeight.Bold
        )
        Text(
            text = "Conexiune: $connectionStatusText",
            style = MaterialTheme.typography.bodyMedium
        )
        if (statusMessage.isNotEmpty()) {
            Text(
                text = "Mesaj: $statusMessage",
                style = MaterialTheme.typography.bodyMedium
            )
        }
    }
}

// Selector pentru tipul de semn de circulație dorit
Text(
    text = "Selectează un semn de circulație:",
    style = MaterialTheme.typography.titleMedium
)

// Lista de opțiuni predefinite pentru tipurile de semne de circulație

```

```

Column(
    modifier = Modifier
        .fillMaxWidth()
        .padding(vertical = 8.dp),
    verticalArrangement = Arrangement.spacedBy(8.dp)
) {
    possibleCommands.forEach { command ->
        FilterChip(
            modifier = Modifier.fillMaxWidth(),
            selected = !useCustomCommand && selectedCommand == command,
            onClick = {
                useCustomCommand = false
                selectedCommand = command
            },
            label = { Text(text = commandLabels[command] ?: command) }
        )
    }
}

FilterChip(
    modifier = Modifier.fillMaxWidth(),
    selected = useCustomCommand,
    onClick = { useCustomCommand = true },
    label = { Text(text = "Personalizat") }
)

// Câmp pentru introducerea unei comenzi personalizate
if (useCustomCommand) {
    OutlinedTextField(
        value = customCommand,
        onValueChange = { customCommand = it },
        label = { Text("Comandă personalizată") },
        modifier = Modifier.fillMaxWidth()
)
}

```

```

        )
    }

// Buton pentru trimiterea comenzii selectate către semn

Button(
    onClick = {
        val commandToSend = if (useCustomCommand) customCommand else selectedCommand
        if (commandToSend.isNotEmpty()) {
            onSendCommand(commandToSend)
        }
    },
    enabled = connectionState == BleConnection.ConnectionState.CONNECTED,
    modifier = Modifier
        .fillMaxWidth()
        .height(56.dp)
) {
    Text(text = "Trimite comandă")
}

// Butonul de resetare apare doar dacă a fost detectat un accident

// Determinăm dacă trebuie să afișăm butonul de resetare accident în funcție de mesajul de status
val showReset = remember(statusMessage) {
    val low = statusMessage.lowercase()
    low.contains("accid") || low.contains("accident")
}

if (showReset) {
    Button(
        onClick = { onSendCommand("RESET_ACCIDENT") },
        colors = ButtonDefaults.buttonColors(containerColor = Color.Red),
        modifier = Modifier
            .fillMaxWidth()
            .height(56.dp)
    )
}

```

```

        Icon(Icons.Default.Refresh, contentDescription = "Resetare")
        Spacer(Modifier.width(8.dp))
        Text("RESETARE SEMN")
    }
}

// Secțiune informativă cu descrieri pentru fiecare tip de semn disponibil
Text(
    text = "Descrieri semne:",
    style = MaterialTheme.typography.titleMedium,
    modifier = Modifier.padding(top = 16.dp)
)

Card(modifier = Modifier.fillMaxWidth()) {
    Column(
        modifier = Modifier.padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        commandDescriptions.forEach { (key, desc) ->
            Text(text = "${commandLabels[key]} - $desc")
        }
        Text(text = "Pentru comenzi personalizate, folosiți formatul JSON: " +
            "{ \"command\": \"COMMAND_NAME\", \"params\": { \"key\": \"value\" } }")
    }
}
}
}

```

TagWriteScreen.kt

```
package com.example.smarttrafficsigns.ui

import androidx.compose.foundation.layout.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ArrowBack
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

/**
 * Fișierul conține implementarea ecranului pentru scrierea tagurilor RFID din semnele de circulație
 * folosind vehiculul Elysium RC ca intermediar.
 */

/**
 * Ecran pentru scrierea tagurilor RFID din semnele de circulație prin intermediul vehiculului Elysium RC.
 * Acest ecran este afișat când utilizatorul dorește să actualizeze conținutul tagului RFID
 * dintr-un semn de circulație. Vehiculul Elysium RC trebuie să fie în modul special de scriere
 * și să fie aproape de semnul de circulație pentru ca operațiunea să funcționeze.
 *
 * @param deviceName Numele dispozitivului Elysium RC conectat pentru scrierea tagului.
 * @param writeEnabled Flag care indică dacă scrierea tagului este disponibilă în acest moment.
 * @param statusMessage Mesajul de status curent primit de la Elysium RC.
 * @param onWriteTag Callback apelat când utilizatorul dorește să trimită comanda de scriere.
 * @param onExit Callback apelat când utilizatorul dorește să părăsească ecranul.
 */
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun TagWriteScreen(
    deviceName: String,
```

```

writeEnabled: Boolean,
statusMessage: String,
onWriteTag: () -> Boolean,
onExit: () -> Unit
) {
    // Stocare locală pentru rezultatul ultimei operațiuni de scriere a tagului
    var lastResult by remember { mutableStateOf<String?>(null) }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Scriere RFID – $deviceName") },
                navigationIcon = {
                    IconButton(onClick = onExit) {
                        Icon(Icons.Default.ArrowBack, contentDescription = "Înapoi")
                    }
                }
            )
        }
    ) {
        padding ->
        Column(
            modifier = Modifier
                .padding(padding)
                .padding(16.dp)
                .fillMaxSize(),
            verticalArrangement = Arrangement.spacedBy(16.dp)
        ) {
            // Afisăm ultimul mesaj primit de la Elysium RC
            // (poate conține conținutul vechi al tag-ului sau statusul operațiunii)
            if (statusMessage.isNotBlank()) {
                Card(modifier = Modifier.fillWidth()) {
                    Column(Modifier.padding(16.dp)) {
                        Text(text = "Mesaj dispozitiv:")

```

```

        Text(text = statusMessage, style = MaterialTheme.typography.bodyMedium)
    }
}

// Afişăm rezultatul local după apăsarea butonului de scriere
// Acest rezultat reflectă succesul sau eșecul trimiterii comenzii, nu al scrierii efective
lastResult?.let {
    Text(text = it, color = MaterialTheme.colorScheme.primary)
}

Spacer(modifier = Modifier.weight(1f))

// Buton pentru inițierea operațiunii de scriere a tagului RFID
// Este activat doar dacă writeEnabled este true (Elysium RC este pregătit pentru scriere)
Button(
    onClick = {
        val ok = onWriteTag()
        lastResult = if (ok) "Comanda de scriere a fost trimisă" else "EROARE la trimiterea comenzii"
    },
    enabled = writeEnabled,
    modifier = Modifier
        .fillMaxWidth()
        .height(56.dp)
) {
    Text("SCRIE TAG")
}
}
}
}

```