Sem. I 2024-2025

Proiect PATR
Sistem de parcare cu barieră

Echipa:
Florea Iustin, Marinescu Andrei-Teodor, Ionescu Costin-Marius, Negoiță Gabriel

Data: 17 ianuarie 2025

Membrii / Grupa332AC	A	C	D	PR	CB	e-mail
Florea Iustin	40%	40%	40%	40%	100%	iustin.florea@stud.acs.upb.com
Marinescu Andrei	14%	33%	27%	14%	100%	andrei.marinescu@stud.acs.upb.com
Ionescu Costin	33%	37%	22%	19%	100%	costin.ionescu@stud.acs.upb.com
Negoiță Gabriel	39%	11%	32%	11%	100%	gabriel.negoita@stud.acs.upb.com

A - analiză problemă și concepere soluție, C - implementare cod, D - editare documentație, PR - "proofreading", CB - contribuție individuală totală (%)

Membrii echipei declară că lucrarea respectă toate regulile privind onestitatea academică. În caz de nerespectare a acestora, tema va fi notată cu ***0 (zero) puncte****.*

Cuprins

Capitolul 1

Introducere

Definirea problemei Problema propusă pentru implementare se referă la gestionarea unei parcări de vehicule. Parcarea este dotată cu un număr de locuri de parcare și un sistem de acces pentru vehicule. În această temă, vom considera că există 4 locuri de parcare și 4 vehicule care încearcă să acceseze parcarea.

Descrierea procesului de funcționare al parcării Parcarea are următoarele caracteristici:

- **Locuri de parcare**: Parcarea are un număr fix de locuri de parcare. Locurile pot fi ocupate doar de vehicule care intră în parcarea disponibilă.
- **Accesul vehiculelor**: Vehiculele pot accesa parcarea doar dacă există locuri disponibile. Când un vehicul sosește și există un loc liber, acesta parchează. Dacă toate locurile sunt ocupate, vehiculul trebuie să aștepte până când un loc devine disponibil.
- **Coada de așteptare**: Dacă parcarea este plină, vehiculele vor forma o coadă de așteptare. Coadă poate conține un număr limitat de vehicule. După ce un vehicul eliberează un loc de parcare, vehiculul din fața cozii intră pe locul disponibil.
- **Durata parcării**: Fiecare vehicul poate rămâne într-un loc de parcare pentru o perioadă limitată de timp, iar dacă depășește această perioadă, se aplică o penalizare.
- **Închiderea parcării**: La sfârșitul fiecărei zile, parcarea se închide. Toate vehiculele rămase trebuie să părăsească parcarea înainte de închiderea sistemului. Dacă există vehicule care sunt încă în parcarea activă la finalul zilei, ele își vor termina parcarea înainte ca închiderea să fie finalizată.
- **Ieșirea vehiculelor**: Vehiculele pot ieși din parcare în momentul în care își eliberează locul de parcare. Procesul de ieșire va fi coordonat astfel încât să nu existe conflicte sau blocaje.

Capitolul 2

Analiza problemei

Implementarea are ca scop simularea unui sistem de parcare cu 4 locuri și 2 bariere de acces. Atunci când o mașină ajunge în fața unui senzor, bariera corespunzătoare se ridică și mașina intră în parcare. După ce mașina intră, bariera se închide, locul de parcare devine ocupat, iar ledul corespunzător locului se face roșu. De asemenea, pe display locul ocupat este înlocuit cu simbolul "-". Acest proces continuă până când parcarea este complet ocupată, iar mesajul „Parcare plină!" va apărea pe display.

Atunci când o mașină părăsește parcarea, locul de parcare devine disponibil din nou. LED-ul de la locul respectiv se schimbă în verde, iar pe display locul respectiv va afișa numărul locului de parcare (de la 1 la 4). Bariera de ieșire se ridică, mașina părăsește parcarea și bariera se închide din nou.

Secvențele corecte de execuție pentru a confirma rularea corespunzătoare a aplicației sunt următoarele:

- Secvența 1:
  - O mașină ajunge la senzorul barierei 1
  - Bariera 1 se ridică și mașina intră în parcare
  - Bariera 1 se închide
  - Locul de parcare 1 devine ocupat și LED-ul se face roșu
  - Display-ul arată: Locuri libere: -, 2, 3, 4
- Secvența 2:
  - O a doua mașină ajunge la senzorul barierei 1
  - Bariera 1 se ridică și mașina intră în parcare
  - Bariera 1 se închide
  - Locul de parcare 2 devine ocupat și LED-ul se face roșu
  - Display-ul arată: -, -, 3, 4
- Secvența 3:
  - O a treia mașină ajunge la senzorul barierei 1
  - Bariera 1 se ridică și mașina intră în parcare
  - Bariera 1 se închide
  - Locul de parcare 3 devine ocupat și LED-ul se face roșu
  - Display-ul arată: -, -, -, 4
- Secvența 4:

  - O a patra mașină ajunge la senzorul barierei 1
  - Bariera 1 se ridică și mașina intră în parcare
  - Bariera 1 se închide
  - Locul de parcare 4 devine ocupat și LED-ul se face roșu
  - Mesajul: „Parcare plină!” apare pe display, iar bariera 1 nu se mai ridica

- Secvența 5:
  - O mașină părăsește parcarea de la locul 1
  - LED-ul locului 1 devine verde
  - Display-ul arată: 1, -, -, -
  - Bariera 2 de ieșire se ridică
  - Mașina părăsește parcarea și bariera 2 de ieșire se închide
  - Locul 1 devine disponibil și este afisat pe display

  Secvențele greșite de execuție, care trebuie evitate, sunt următoarele:

  - Secvența 1 - nu se respectă condiția de parcare plină:
    * Se încearcă parcarea unei a cincea mașini când toate locurile de parcare sunt ocupate.
    * Mesajul de parcare plină nu este afișat corect pe display.
  - Secvența 2 - nu se respectă ordinea de ridicare a barierei pentru mașini:
    * O mașină ajunge la senzorul barierei 1, iar bariera 1 se ridică corect
    * Însă o altă mașină ajunge la senzorul barierei 2, dar bariera 2 nu se ridică la timp
    * Locul 2 rămâne disponibil, dar ar trebui să fie ocupat
  - Secvența 3 - nu se respectă ordinea de eliberare a locurilor de parcare:
    * O mașină părăsește parcarea, dar LED-ul nu se schimbă din roșu în verde
    * Locul nu devine disponibil pe display după eliberarea acestuia
    * Bariera de ieșire nu se ridică
  - Secvența 4 - nu se respectă intervalul de timp necesar pentru ridicarea barierei de ieșire:
    * O mașină ajunge la barierea de ieșire, dar bariera se ridică prea repede
    * Mașina părăsește parcarea, iar bariera se închide prea devreme
  - Secvența 5 - nu se actualizează corect statusul locurilor după plecarea mașinii:
    * După ce o mașină pleacă și locul devine liber, locul rămâne marcat ca ocupat pe display

Capitolul 3

Aplicația. Structura și soluția de implementare propusă

3.1 Definirea structurii aplicației

Aplicația va fi împărțită pe mai multe task-uri corespunzătoare diferitelor entități care vor interacționa / executa anumite operații: Bariere, Senzori, Display, Leduri. În această implementare, task-urile corespund unor fire de execuție.

Task pentru adaptarea la sloturi

Acest task gestionează locurile de parcare disponibile. El verifică continuu ocuparea sloturilor, actualizează starea acestora și semnalizează componentelor relevante (de exemplu, barierei de intrare sau ieșire) modificările survenite. De asemenea, se ocupă cu rezervarea și eliberarea locurilor de parcare conform cerințelor clienților.

Task pentru Bariera 1 (Intrare)

Bariera de intrare verifică periodic dacă există clienți care solicită intrarea în parcare. Când detectează un client, verifică disponibilitatea locurilor prin intermediul taskului de adaptare la sloturi. Dacă există locuri libere, se inițiază procedura de intrare, rezervând sloturile necesare și notificând clientul despre succesul operației.

În cazul în care parcarea este plină, task-ul notifică clientul printr-un mesaj corespunzător afișat pe LED-uri. Bariera rămâne activă până la oprirea aplicației, moment în care finalizează procedurile curente înainte de a intra în starea de pauză.

Task pentru Bariera 2 (Ieșire)

Bariera de ieșire gestionează fluxul vehiculelor care pleacă din parcare. Când detectează un vehicul care dorește să iasă, verifică ocuparea slotului asociat prin intermediul taskului de adaptare la sloturi, eliberând locul în sistem.

Acest task notifică LED-urile și alte componente despre modificarea statusului parcării. După finalizarea operației de ieșire, task-ul continuă să monitorizeze fluxul de vehicule până la oprirea aplicației.

Task pentru actualizarea LED-urilor

Task-ul pentru actualizarea LED-urilor gestionează afișarea informațiilor despre parcarea curentă. Acesta afișează numărul de locuri disponibile și mesajul "Parcare plină!" atunci când toate locurile sunt ocupate.

LED-urile sunt actualizate de fiecare dată când există o modificare în starea parcării, fie că este vorba despre intrarea unui nou vehicul, ieșirea unuia sau eliberarea unor locuri.

3.2 Definirea soluției în vederea implementării

Soluția a fost implementată pe o plăcuță Arduino Mega 2560, folosind `FreeRTOS.h` pentru gestionarea task-urilor și sincronizarea acestora.

Pentru a putea satisface condițiile de funcționare corectă a aplicației, am ales să utilizăm următoarele mecanisme în plus față de soluția bazată pe un sistem de operare în timp real (RTOS):

- 4 timere pentru gestionarea diverselor stări ale barierei și a fluxului de clienți, inclusiv gestionarea temporizării pentru intrarea și ieșirea vehiculelor din parcare.
- 1 timer dedicat pentru monitorizarea și actualizarea informațiilor de la Display, care oferă feedback în timp real despre starea parcării.
- Utilizarea senzorilor de proximitate pentru detectarea vehiculelor în apropierea locurilor de parcare.

Pentru ca modul de funcționare a aplicației să poată fi urmărit mai ușor, am adăugat o serie de LED-uri care să acționeze drept indicatoare vizuale pentru stadiul de execuție al task-urilor corespunzătoare. Fiecare loc de parcare este dotat cu un LED RGB.

Pentru locuri de parcare:

- verde: locul este liber
- roșu: locul este ocupat

3.2.1 Mecanismele de sincronizare

Pentru a implementa sincronizarea între taskuri, am utilizat alte mecanisme precum variabile de condiție și fluxuri de execuție bine coordonate. Barierele sunt sincronizate cu senzorii și clienții printr-o logică de control ce asigură accesul exclusiv al unui client la un loc de parcare într-un moment dat. Actualizarea informațiilor despre starea parcării în Display și gestionarea comenzilor clienților sunt coordonate printr-o secvențiere strictă a operațiilor, evitând conflictele prin proiectarea logicii aplicației.

3.2.2 Plăcuța Arduino

Aplicația este construită pe plăcuța Arduino Mega 2560, care permite integrarea unui număr mare de componente hardware. În proiect sunt folosiți senzorii IR pentru a detecta vehiculele care se apropie de locurile de parcare, iar barierele sunt controlate de servomotoare pentru a ridica sau coborî barierele în funcție de semnalele primite de la clienți și senzori. Plăcuța Arduino permite și conectarea la Display pentru a oferi feedback vizual utilizatorilor.

3.2.3 Schema circuitului

Circuitul folosește mai multe componente:

- Arduino Mega 2560 ca unitate centrală de control
- 6xSenzori IR pentru detectarea vehiculelor
- 2xServomotoare pentru controlul barierei
- 4xLED-uri RGB pentru indicarea stării fiecărui client și barieră
- Display pentru a afișa statusul parcării

3.3 Implementarea soluției

În această secțiune se va prezenta codul aplicației. Fie se vor insera comentarii în cod, fie se vor discuta fragmente mai mari pentru a explica modul de lucru.

Exemplu

```
#include <LiquidCrystal_I2C.h>  // Biblioteca pentru ecranul LCD cu I2C
#include <Servo.h>              // Biblioteca pentru controlul servomotoarelor
#include <Arduino_FreeRTOS.h>   // Biblioteca pentru utilizarea FreeRTOS (multithreading)
#include <semphr.h>             // Biblioteca pentru semafoare (folosit pentru sincronizare)


LiquidCrystal_I2C lcd(0x27, 16, 2);  // Ini ializarea obiectului LCD cu adresa 0x27, 16 coloane  i  2 linii
Servo inComingServo;                 // Obiect pentru controlul servomotorului barier  de intrare
Servo outGoingServo;                 // Obiect pentru controlul servomotorului barier  de ie ire

const int servo1 = 51;               // Pinul pentru servomotorul de intrare
const int servo2 = 53;               // Pinul pentru servomotorul de ie ire

// Variabilele pentru starea barierelor
bool inComingBarrierClose = false;   // Stare barier  intrare (  nchis sau deschis )
bool outGoingBarrierClose = false;   // Stare barier  ie ire (  nchis sau deschis )
```

```
17 String slotsAvailability[4] = {"1", "2", "3", "4"};  // Sloturile de
       parcare disponibile (1 - disponibil, - - ocupat)
18
19 // Senzorii pentru sloturi  i  bariere
20 const int inComingSensor = 49;       // Senzorul pentru detectarea
       vehiculului la intrare
21 const int outGoingSensor = 47;       // Senzorul pentru detectarea
       vehiculului la ie ire
22
23 // Definirea pinilor RGB pentru fiecare slot de parcare
24 // Slot 1
25 const int r1 = 22;
26 const int g1 = 24;
27 const int b1 = 26;
28 const int sensor1 = 42;
29
30 // Slot 2
31 const int r2 = 28;
32 const int g2 = 30;
33 const int b2 = 32;
34 const int sensor2 = 44;
35
36 // Slot 3
37 const int r3 = 39;
38 const int g3 = 36;
39 const int b3 = 38;
40 const int sensor3 = 46;
41
42 // Slot 4
43 const int r4 = 40;
44 const int g4 = 35;
45 const int b4 = 37;
46 const int sensor4 = 48;
47
48 // Semafore pentru sincronizarea accesului la resurse partajate
49 SemaphoreHandle_t servoSemaphore;
50 SemaphoreHandle_t ledSemaphore;
51 SemaphoreHandle_t displaySemaphore;
52
53 void setup() {
54   lcd.init();           // Ini ializeaz  LCD-ul
55   lcd.backlight();      // Activeaz  iluminarea de fundal a LCD-ului
56   Serial.begin(9600);   // Deschide comunicarea serial  la 9600 bps
57
58   // Ata eaz  servomotoarele la pini
59   inComingServo.attach(servo1);
60   outGoingServo.attach(servo2);
61
62  // Seteaz  pinurile pentru senzori  i  LED-uri ca input/output
63   pinMode(sensor1, INPUT);
64   pinMode(sensor2, INPUT);
65   pinMode(sensor3, INPUT);
66   pinMode(sensor4, INPUT);
67   pinMode(inComingSensor, INPUT);
68   pinMode(outGoingSensor, INPUT);
69
70   pinMode(r1, OUTPUT);
71   pinMode(g1, OUTPUT);
72   pinMode(b1, OUTPUT);
73
74   pinMode(r2, OUTPUT);
```

```
75    pinMode(g2, OUTPUT);
76    pinMode(b2, OUTPUT);
77
78    pinMode(r3, OUTPUT);
79    pinMode(g3, OUTPUT);
80    pinMode(b3, OUTPUT);
81
82    pinMode(r4, OUTPUT);
83    pinMode(g4, OUTPUT);
84    pinMode(b4, OUTPUT);
85
86    // Crearea semafoarelor
87    servoSemaphore = xSemaphoreCreateBinary();
88    ledSemaphore = xSemaphoreCreateBinary();
89    displaySemaphore = xSemaphoreCreateBinary();
90
91    // Ini ializare semafoare
92    xSemaphoreGive(servoSemaphore);
93    xSemaphoreGive(ledSemaphore);
94    xSemaphoreGive(displaySemaphore);
95  }
96
97  void loop() {
98    // Sarcini concurrente pentru actualizarea sloturilor, barierei de
         intrare/ie ire  i  LED-urilor
99    taskUpdateSlots();
100   taskManageIncomingBarrier();
101   taskManageOutgoingBarrier();
102   taskUpdateLEDs();
103 }
104
105 // Actualizeaz  starea sloturilor de parcare  i  afi eaz  pe LCD
106 void taskUpdateSlots() {
107   if (xSemaphoreTake(displaySemaphore, portMAX_DELAY)) {
108     static unsigned long lastDisplayUpdate = 0;
109     const unsigned long displayInterval = 500;
110
111     // Verific  intervalul de actualizare al LCD-ului
112     if (millis() - lastDisplayUpdate >= displayInterval) {
113       lastDisplayUpdate = millis();
114
115       // Cite te statusul senzorilor pentru fiecare slot
116       int slotSensorStatus_1 = digitalRead(sensor1);
117       int slotSensorStatus_2 = digitalRead(sensor2);
118       int slotSensorStatus_3 = digitalRead(sensor3);
119       int slotSensorStatus_4 = digitalRead(sensor4);
120
121       // Actualizeaz  disponibilitatea sloturilor
122       slotsAvailability[0] = (slotSensorStatus_1 == 1) ? "1" : "-";
123       slotsAvailability[1] = (slotSensorStatus_2 == 1) ? "2" : "-";
124       slotsAvailability[2] = (slotSensorStatus_3 == 1) ? "3" : "-";
125       slotsAvailability[3] = (slotSensorStatus_4 == 1) ? "4" : "-";
126
127       lcd.clear();// Cur      LCD-ul
128       if (slotsAvailability[0] != "-" || slotsAvailability[1] != "-" ||
             slotsAvailability[2] != "-" || slotsAvailability[3] != "-") {
129         lcd.setCursor(2, 0); // Seteaz  cursorul la prima linie, a doua
               coloan
130         lcd.print("Locuri libere");
131         lcd.setCursor(0, 1); // Seteaz  cursorul pe a doua linie
132         lcd.print(slotsAvailability[0] + ", " + slotsAvailability[1] + ", "
```

```
            + slotsAvailability[2] + ", " + slotsAvailability[3]);
      } else {
        lcd.setCursor(2, 0);
        lcd.print("Parcare plina!");
      }
    }
    xSemaphoreGive(displaySemaphore); // Elibereaz  semaforul pentru alte
       taskuri
  }
}

// Gestioneaz  bariera de intrare
void taskManageIncomingBarrier() {
  if (xSemaphoreTake(servoSemaphore, portMAX_DELAY)) {
    static unsigned long lastServoMove = 0;
    const unsigned long servoInterval = 20;

    int inComingSensorStatus = digitalRead(inComingSensor);  // Cite te
       statusul senzorului de intrare

    // Verific  dac  parcarea este plin
    bool parkingFull = true;
    for (int i = 0; i < 4; i++) {
      if (slotsAvailability[i] != "-") {
        parkingFull = false; // Dac  exist  cel pu in un loc liber
        break;
      }
    }

    if (parkingFull) {
      // Dac  parcarea este plin , afi eaz  mesajul  i  ine  bariera
           nchis
      lcd.clear();
      lcd.setCursor(2, 0);
      lcd.print("Parcare plina! ");
      xSemaphoreGive(servoSemaphore);
      return; // Ie ire din func ie f  r  a ac iona bariera
    }

    // Dac  parcarea nu este plin , ac ion m bariera
    if (inComingSensorStatus == 1) { // Senzorul detecteaz  un vehicul
      if (millis() - lastServoMove >= servoInterval) {
        lastServoMove = millis();
        inComingServo.write(max(inComingServo.read() - 1, 0)); // Coboar
           bariera
        if (inComingServo.read() == 0) {
          inComingBarrierClose = false; // Bariera este complet cobor t
        }
      }
    } else { // Senzorul nu mai detecteaz  vehicul
      if (millis() - lastServoMove >= servoInterval) {
        lastServoMove = millis();
        inComingServo.write(min(inComingServo.read() + 1, 90)); // Ridic
           bariera
        if (inComingServo.read() == 90) {
          inComingBarrierClose = true; // Bariera este complet ridicat
        }
      }
    }

    xSemaphoreGive(servoSemaphore); // Elibereaz  semaforul
```

```
188   }
189 }
190
191 // Gestioneaz  bariera de ie ire
192 void taskManageOutgoingBarrier() {
193   if (xSemaphoreTake(servoSemaphore, portMAX_DELAY)) {
194     static unsigned long lastServoMove = 0;
195     const unsigned long servoInterval = 20;
196
197     int outGoingSensorStatus = digitalRead(outGoingSensor); // Cite te
          statusul senzorului de ie ire
198
199     if (outGoingSensorStatus == 1) { // Senzorul detecteaz  un vehicul
200       if (millis() - lastServoMove >= servoInterval) {
201         lastServoMove = millis();
202         outGoingServo.write(max(outGoingServo.read() - 1, 0)); // Coboar
              bariera
203         if (outGoingServo.read() == 0) {
204           outGoingBarrierClose = false; // Bariera este complet cobor t
205         }
206       }
207     } else { // Senzorul nu mai detecteaz  vehicul
208       if (millis() - lastServoMove >= servoInterval) {
209         lastServoMove = millis();
210         outGoingServo.write(min(outGoingServo.read() + 1, 90)); // Ridic
              bariera
211         if (outGoingServo.read() == 90) {
212           outGoingBarrierClose = true; // Bariera este complet ridicat
213         }
214       }
215     }
216
217     xSemaphoreGive(servoSemaphore); // Elibereaz  semaforul
218   }
219 }
220
221
222 // Actualizeaz  LED-urile pentru sloturi pe baza disponibilit  ii
223 void taskUpdateLEDs() {
224   if (xSemaphoreTake(ledSemaphore, portMAX_DELAY)) {
225     static unsigned long lastLEDUpdate = 0;
226     const unsigned long ledInterval = 100;
227
228     if (millis() - lastLEDUpdate >= ledInterval) {
229       lastLEDUpdate = millis();
230
231       // Verific  disponibilitatea fiec rui slot  i  seteaz  culoarea
            corespunz toare a LED-urilor
232       if (slotsAvailability[0] == "-") {
233         color(255, 0, 0, 1);  // Slotul 1 ocupat (ro u)
234       } else {
235         color(0, 255, 0, 1);  // Slotul 1 liber (verde)
236       }
237
238       if (slotsAvailability[1] == "-") {
239         color(255, 0, 0, 2);  // Slotul 2 ocupat (ro u)
240       } else {
241         color(0, 255, 0, 2);  // Slotul 2 liber (verde)
242       }
243
244       if (slotsAvailability[2] == "-") {
```

```
245         color(255, 0, 0, 3);  // Slotul 3 ocupat (ro u)
246       } else {
247         color(0, 255, 0, 3);  // Slotul 3 liber (verde)
248       }
249
250       if (slotsAvailability[3] == "-") {
251         color(255, 0, 0, 4);  // Slotul 4 ocupat (ro u)
252       } else {
253         color(0, 255, 0, 4);  // Slotul 4 liber (verde)
254       }
255     }
256     xSemaphoreGive(ledSemaphore);  // Elibereaz  semaforul
257   }
258 }
259
260 // Seteaz  culoarea LED-urilor RGB pentru fiecare slot
261 void color(unsigned char red, unsigned char green, unsigned char blue, int
       light) {
262   if (light == 1) {
263     analogWrite(r1, red);
264     analogWrite(b1, blue);
265     analogWrite(g1, green);
266   } else if (light == 2) {
267     analogWrite(r2, red);
268     analogWrite(b2, blue);
269     analogWrite(g2, green);
270   } else if (light == 3) {
271     analogWrite(r3, red);
272     analogWrite(b3, blue);
273     analogWrite(g3, green);
274   } else if (light == 4) {
275     analogWrite(r4, red);
276     analogWrite(b4, blue);
277     analogWrite(g4, green);
278   }
279 }
```

Capitolul 4

Testarea aplicației și validarea soluției propuse

Aplicația a fost testată pe parcursul întregului proces de dezvoltare, începând cu testele unitare pentru fiecare funcționalitate de bază și până la testarea întregului sistem integrat. Astfel, fiecare task a fost testat independent pentru a verifica corectitudinea funcționării lor în scenarii izolate. După finalizarea testării fiecărui task în parte, aplicația a fost testată în ansamblu pentru a verifica interacțiunile între componente.

În cadrul testării s-au utilizat mai multe secvențe de intrare, astfel încât să se simuleze diferite scenarii de parcare, inclusiv clienți care ajung într-un moment de vârf, unii care așteaptă să găsească un loc de parcare liber.

Validarea soluției Validarea aplicației a fost realizată prin rularea unor secvențe de intrare care reflectă diverse situații ce pot apărea într-o parcare reală. Printre aceste secvențe se numără:

Clienți care ajung simultan și caută locuri de parcare disponibile, iar unii dintre ei sunt plasați în așteptare dacă nu există locuri libere.

Rezultatele testării În urma testării, s-a constatat că aplicația respectă funcționalitățile dorite și că mecanismele de sincronizare sunt eficiente, gestionând corect clienții care așteaptă să se elibereze locuri de parcare și alocarea locurilor disponibile. De asemenea, interacțiunea dintre task-uri nu prezintă blocaje sau erori de sincronizare.

Observații și concluzii În urma observațiilor făcute pe baza testelor, aplicația a fost ajustată pentru a îmbunătăți gestionarea locurilor de parcare și pentru a asigura că toți clienții sunt serviți corespunzător. În stadiul actual, aplicația răspunde eficient și corect la toate tipurile de intrare, iar sincronizarea între componentele aplicației a fost optimizată pentru a minimiza orice tip de blocaj.

Capitolul 5

Bibliografie

- **Arduino**, *Arduino Documentation*. https://www.arduino.cc/en/Guide/Introduction.
- **FreeRTOS**, *FreeRTOS - Real-Time Operating System*. https://www.freertos.org/.
- **SparkFun Electronics**, *Guide to Servo Motors*.https://learn.sparkfun.com/tutorials/servo-motors/all.
- **Adafruit**, *Adafruit RGB LEDs Guide*. https://learn.adafruit.com/adafruit-neopixel-uberguide.
- **Instructables**, *Smart Parking System using Arduino*.https://www.instructables.com/Smart-Parking-S
- **Random Nerd Tutorials**, *Guide for Ultrasonic Sensor HC-SR04 with Arduino*. https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/.
- **Electronics Hub**, *Automatic Car Parking System using Arduino*. https://www.electronicshub.org/automatic-car-parking-system/.
- **Robocraze**, *How to Interface LCD with Arduino*. https://robocraze.com/blogs/post/how-to-interfac