



Universitatea Politehnica Timișoara
Facultatea de Automatică și Calculatoare
Departamentul Calculatoare și
Tehnologia Informației



SMARTPARK

Sistem pentru monitorizarea locurilor de parcare publice

Lucrare de diplomă

Student:

Stefan CHIVU

Profesor Coordonator:

Prof. Mihai UDRESCU-MILOSAV

Timișoara
Iunie, 2023

Cuprins

1	Introducere	5
1.1	Context	5
1.1.1	Probleme din perspectiva cetățenilor	5
1.1.2	Probleme din perspectiva autorităților	5
1.2	Descrierea proiectului	5
1.3	Conținutul lucrării	6
2	Analiza domeniului	9
2.1	Aplicații mobile asemănătoare	9
3	Tehnologii folosite	11
3.1	Baza de date	11
3.1.1	MySQL	11
3.1.2	phpMyAdmin	11
3.2	Tehnologii folosite în cadrul aplicației mobile	11
3.2.1	Firebase Authentication	11
3.2.2	Flutter & Dart	12
3.2.3	Google Maps API	12
3.2.4	Mapbox Directions API	12
3.2.5	Riverpod	12
3.2.6	Isar	12
3.2.7	Geolocator	13
3.2.8	Android	13
3.2.9	Android Studio	13
3.3	Tehnologii folosite pentru realizarea infrastructurii server-side	13
3.3.1	Google Cloud Platform	13
3.3.2	Ubuntu Server	13
3.3.3	MySQL Server	13
3.3.4	Apache2	14
3.3.5	PHP	14
3.3.6	Python	14
3.3.7	Plate Recognizer API	14
3.4	Componente hardware folosite	15
3.4.1	ESP32-CAM	15
3.4.2	OV-2640 Camera module	16
3.4.3	HC-SR04 Sonar module	16
3.5	Tehnologii folosite pentru programarea senzorilor	17
3.5.1	MicroPython	17

3.5.2 Thonny	17
4 Proiectarea sistemului	19
4.1 Diagrame UML	19
4.1.1 Unified Modeling Language	19
4.1.2 Diagrama UML a cazurilor de utilizare	19
4.1.3 Sistemul	19
4.1.4 Actorii	20
4.1.5 Cazul de adăugare al unui loc de parcare	21
4.2 Proiectarea bazei de date	21
4.3 Configurația hardware:	22
5 Implementarea aplicației mobile	23
5.1 Modelul informațional	23
5.1.1 Modelul pentru utilizatori	23
5.1.2 Modelul de adresă	24
5.1.3 Modelul de mașină	24
5.1.4 Modelul unui senzor	25
5.1.5 Modelul unei plăți de parcare	26
5.1.6 Modelul pentru zone tarifare	27
5.2 Serviciile aplicației	27
5.2.1 Serviciul de autentificare	27
5.2.2 Serviciul de stocare locală	29
5.2.3 Serviciul de MySQL	31
5.2.4 Serviciul de localizare	42
5.2.5 Serviciul de căutare a adreselor	43
5.3 Provideri de date	43
5.3.1 Providerul de date despre senzori	43
5.3.2 Providerul de date pentru plățile de parcare	45
5.3.3 Providerul de date pentru pagina de adăugare de senzori	45
6 Configurarea plăcii de dezvoltare ESP32-CAM	47
6.1 Software-ul folosit pentru programarea plăcii de dezvoltare	47
6.2 Serviciul web folosit pentru realizarea comunicării dintre baza de date și placă de dezvoltare	48
6.2.1 Scriptul de PHP pentru prelucrarea cererilor HTTP	48
6.2.2 Scripturile pentru procesarea informațiilor primite de la senzori	49
7 Utilizarea aplicației	53
7.1 Autentificarea	53
7.2 Înregistrarea în aplicație	53
7.3 Formularul de bun-venit	55
7.4 Pagina principală	56
7.5 Pagina de navigare	57
7.6 Pagina de profil a utilizatorului	59
7.6.1 Adăugarea adreselor corespunzătoare domiciliului și locului de muncă	59
7.6.2 Adăugarea informațiilor despre mașinile deținute	59
7.7 Adăugarea unui nou loc de parcare	61
7.8 Deconectarea de la aplicație	62

7.9 Pagina de vizualizare în listă a locurilor de parcare disponibile	62
7.10 Pagina de vizualizare a stării plășilor de parcare	62
7.10.1 Pagina de plată a tarifului de parcare	63
8 Concluzii	65
8.1 Note generale	65
8.2 Discuție	65
8.3 Directii de dezvoltare	65

Capitolul 1

Introducere

1.1 Context

În cazul conducerilor auto, un fenomen neplăcut întâlnit adesea în momentul deplasării în zonele publice mai aglomerate constă în problema găsirii unui loc de parcare. Astfel scopul proiectului *SmartPark* este de a monitoriza situația locurilor de parcare publice în teren, pentru a ajuta în procesul de căutare a acestora.

1.1.1 Probleme din perspectiva cetățenilor

Nevoia de a efectua mai multe încercări în jurul unui anumit punct de interes în speranța găsirii unui loc de parcare este din păcate una întâlnită frecvent, ce provoacă nemulțumire și discomfort participanților la trafic.

Potrivit unui studiu^[1] realizat de University of Toronto, în timpul căutării unui loc de parcare, șoferii au tendința de a conduce mai încet și mai aproape de bordură, ceea ce poate duce la o creștere a valorilor de trafic. Totodată, aceștia tind să își axeze privirea în exteriorul carosabilului mai frecvent și pe perioade mai îndelungate de timp, ceea ce poate duce la apariția unor situații periculoase.

1.1.2 Probleme din perspectiva autorităților

În studiul menționat anterior^[1], s-a observat și faptul că în timpul căutării unui loc de parcare în zone aglomerate, șoferii tind să judece legalitatea locurilor de parcare pe baza prezenței altor mașini pe acestea, ignorând semnalizarea rutieră amplasată.

Astfel, dacă procesul de căutare a locurilor de parcare ar fi eficientizat, șoferii ar putea fi direcționați spre locuri de parcare amenajate corespunzător, reducându-se numărul de parcări ilegale ce pot pune în pericol alți participanți la trafic prin efecte precum reducerea vizibilității într-o intersecție.

1.2 Descrierea proiectului

Principala funcționalitate a sistemului „SmartPark” constă în obținerea indicațiilor de călătorie spre un loc de parcare liber în zonele aglomerate ale unui oraș. Alegerea destinației se bazează pe date furnizate în timp real de la o rețea de senzori amplasați pe fiecare loc de parcare,

și include proceduri de recalculare a traseului în cazul în care locul de parcare inițial devine indisponibil.

Aplicația mobilă are ca scop eficientizarea procesului de căutare a unui loc de parcare. Aceasta este concepută astfel încât utilizatorul să poată vizualiza situația locurilor de parcare din jurul destinației sale și să poată găsi cu ușurință un loc disponibil.

Din perspectiva autorităților, sistemul ajută la monitorizarea situației plății tarifelor de parcare prin fotografiarea numerelor de înmatriculare a autovehiculelor în momentul ocupării unui loc de parcare și procesarea imaginii respective folosind un model de inteligență artificială pentru extragerea acestuia din imagine.

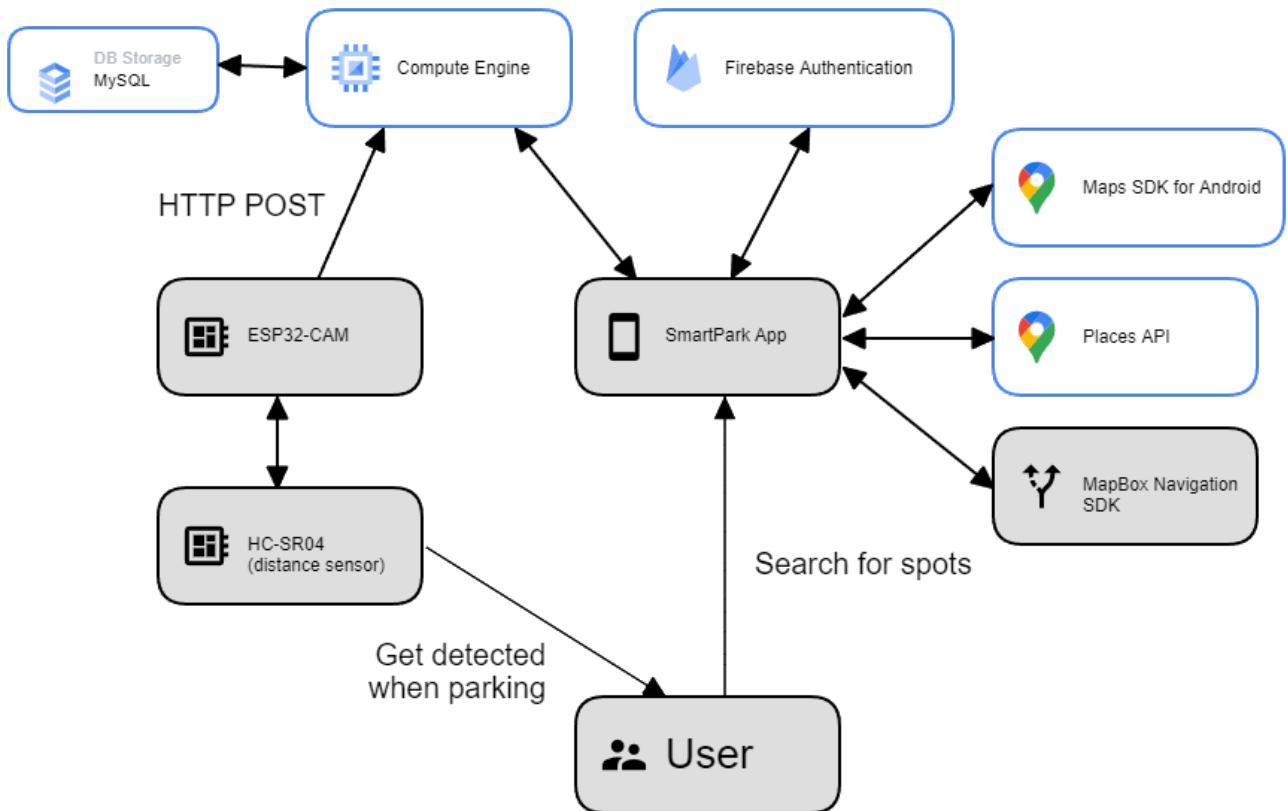


Figura 1.1: Diagrama generală a sistemului. Aceasta prezintă interacțiunile dintre anumite componente. Primul flux de informații constă în cel realizat de senzorul de distanță și placa de dezvoltare ESP32-CAM spre baza de date MySQL aflată pe o mașină virtuală. Acesta constă în transmiterea imaginii corespunzătoare ocupării/eliberării locului de parcare. Al doilea flux de informații este cel prin care utilizatorul, folosind aplicația mobilă, va putea vizualiza situația locurilor de parcare din teren, pentru a putea ulterior să navigheze spre acestea.

1.3 Conținutul lucrării

Lucrarea este împărțită în mai multe capituloare. În capitolul 2, se va efectua o analiză comparativă între sistemul propus, SmartPark, și alte două aplicații similare prezente pe piață. În capitolul 3, se vor prezenta sumar tehnologiile utilizate pentru implementarea sistemului. În capitolul 4, vor fi expuse anumite detalii de proiectare. În capitolul 5, se vor oferi informații despre implementarea aplicației mobile. În capitolul 6, se va prezenta software-ul folosit în

programarea plăcii de dezvoltare, precum și codul rulat pe server pentru comunicarea cu baza de date. În capitolul 7, vor fi prezentate ecranele aplicației mobile și modul de utilizare al acesteia. Nu în ultimul rând, în capitolul 8 se vor discuta concluziile obținute în urma implementării sistemului și prezenta posibilele direcții de dezvoltare viitoare.

Capitolul 2

Analiza domeniului

2.1 Aplicații mobile asemănătoare

Comparând aplicația SmartPark cu alte două aplicații pentru Android, respectiv **RingGo Parking**, cu peste 5.000.000 de descărcări, și **EasyPark - find & pay parking**, cu peste 10.000.000 de descărcări, se pot observa anumite similarități [2.1] în spectrul de funcționalități, precum căutarea unui loc de parcare în parcări private sau plata parcării din aplicație.

Tabelul 2.1: Analiză comparativă între unele dintre cele mai apreciate aplicații mobile pentru sistemul de operare Android (EasyPark - find & pay parking și RingGo Parking) și "SmartPark"; funcționalitățile analizate regăsite într-o aplicație sunt marcate cu **✓**, iar cele absente cu **✗**

Caracteristici și funcționalități	EasyPark - find & pay parking	RingGo Parking	SmartPark
Sistem de operare	Android	Android	Android/iOS
Magazin de aplicații	Google Play	Google Play	✗
Nota din Google Play	4,2/5	4,9/5	✗
Număr de instalări	10M+	5M+	✗
Număr de ratinguri	134 000	51 000	✗
Căutare locuri de parcare în parcări private	✓	✓	✗
Căutare locuri de parcare în spații publice	✗	✗	✓
Căutare stații de încărcare pentru mașini electrice	✓	✗	✓
Indicații către locuri de parcare publice	✗	✗	✓
Plata din aplicație	✓	✓	✓

Prin urmare, spre deosebire de celelalte două aplicații, SmartPark oferă posibilitatea găsirii de locuri de parcare publice, fiind posibilă extinderea sistemului pentru a fi suportată și locurile de parcare din parcări private. Acest lucru reprezintă un avantaj pentru utilizatori, deoarece au la dispoziție informații mai complete despre gradul de ocupare al locurilor de parcare din jurul destinației lor.

Capitolul 3

Tehnologii folosite

3.1 Baza de date

În această secțiune, se vor prezenta tehnologiile folosite pentru sistemul de management al bazelor de date utilizat de sistem.

3.1.1 MySQL

MySQL este un sistem open-source de gestiune a bazelor de date relationale oferit de Oracle Corporation. Aceasta utilizează limbajul de interogare SQL (Structured Query Language) pentru a interacționa cu datele stocate în baza de date. Folosind SQL, se pot crea, interoga, modifica și sterge date într-un mod eficient și sigur. Totodată, MySQL suportă multiple funcționalități precum relații între tabele, tranzacții și indecști.

Unul dintre avantajele folosirii MySQL este scalabilitatea pe care o oferă. Acest sistem poate gestiona volume mari de date și prezintă opțiuni de securitate robuste.

3.1.2 phpMyAdmin

Una dintre cele mai populare interfețe web de gestionare a bazelor de date MySQL este phpMyAdmin. Aceasta este o aplicație web open-source ce permite manipularea bazelor de date MySQL fără a fi necesară scrierea manuală a codului de SQL.

3.2 Tehnologii folosite în cadrul aplicației mobile

În această secțiune, vor fi prezentate tehnologiile utilizate în cadrul implementării aplicației mobile.

3.2.1 Firebase Authentication

Firebase Authentication este un serviciu de autentificare oferit de Google, prin intermediul căruia se pot integra cu ușurință funcționalități în cadrul aplicațiilor software.

Printre serviciile oferite se numără autentificarea cu e-mail și parolă, autentificarea cu ajutorul unui cont al unei rețele sociale (e.g. Facebook, Google) sau autentificarea în doi pași.

Unul din principalele avantaje al utilizării serviciului Firebase Authentication este beneficiul infrastructurii sigure și scalabile din spatele procesului de autentificare fără

problema menținerii unei astfel de infrastructuri. Datele utilizatorilor sunt criptate cu algoritmi moderni și siguri, iar infrastructura este protejată împotriva atacurilor cibernetice precum cele de forță brută.

3.2.2 Flutter & Dart

Flutter este un framework open-source pentru platforme multiple dezvoltat de Google. Acesta oferă posibilitatea dezvoltării de aplicații pentru Android, iOS, Web, Windows, macOS și Linux, iar limbajul folosit în cadrul acestui framework este Dart.

Dart este un limbaj modern și eficient cu tipare statică, oferind funcționalități precum gestionarea memoriei prin garbage collection, suport pentru programare asincronă și reactivă.

Unul dintre avantajele Flutter este experiența nativă pe fiecare dintre platformele suportate construită cu același cod sursă. Astfel, dezvoltatorii pot obține aplicații web, mobile și desktop cu aspect și performanțe similare pe diferite sisteme de operare. De asemenea, Flutter oferă și o documentație [2] extinsă și o bibliotecă stufoasă de widget-uri personalizabile.

3.2.3 Google Maps API

Google Maps API este un set de servicii și funcționalități oferit de platforma Google Cloud care permite dezvoltatorilor să integreze cu ușurință hărți interactive și diverse informații geografice în aplicațiile lor.

Printre funcționalitățile oferite de Google Maps API se numără afișarea hărților interactive și personalizabile, adăugarea de indicatori corespunzători unor locații geografice dorite de dezvoltator precum și furnizarea de sugestii pentru completarea automată în căutarea adreselor.

3.2.4 Mapbox Directions API

Acesti API este un serviciu oferit de Mapbox ce pune la dispoziție dezvoltatorilor indicații optime de călătorie bazate pe hărți. În momentul calculării rutei, se pot obține informații precum durată și distanța călătoriei și rute alternative.

3.2.5 Riverpod

Riverpod este o bibliotecă de gestionare a stării pentru framework-ul Flutter. Aceasta oferă un sistem robust și declarativ pentru gestionarea și partajarea stării aplicației într-o manieră reactivă și eficientă în aplicații. Dezvoltatorii pot astfel utiliza fluxuri de informații pentru a reacționa la schimbările de stare și a actualiza interfața utilizatorilor în mod reactiv și interactiv.

3.2.6 Isar

Isar este o bibliotecă pentru gestionarea bazei de date în aplicații Flutter. Prin intermediul acestuia, dezvoltatorii pot crea și manipula datele unei baze de date locale.

Principalul avantaj al bazei de date gestionată de Isar este viteza sa. Mecanismele de indexare și optimizare folosite asigură accesul rapid și eficient la date. De asemenea, sistemul oferă suport și pentru interogări complexe, filtrare și sortare a datelor.

3.2.7 Geolocator

Geolocator este o bibliotecă pentru Flutter care facilitează accesul la informațiile puse la dispoziție de modulul GPS al telefonului mobil. Prin intermediul acesteia, se pot obține coordonatele geografice corespunzătoare telefonului.

3.2.8 Android

Android este un sistem de operare open-source dezvoltat de Google, conceput inițial pentru dispozitive mobile. Acesta a devenit ulterior o platformă extinsă, putând fi folosit pe mai multe tipuri de dispozitive, precum telefoane inteligente, tablete, televizoare sau ceasuri inteligente.

Android se bazează pe nucleul Linux și oferă un set variat de API-uri și servicii care facilitează dezvoltarea aplicațiilor mobile, precum funcționalități de gestionare a interfeței grafice sau accesul la date și resurse.

3.2.9 Android Studio

Android Studio este mediul de dezvoltare integrat (IDE) oficial pentru platforma Android, oferit de Google. Acesta este bazat pe platforma IntelliJ IDEA și este optimizat special pentru dezvoltarea aplicațiilor Android.

Un aspect important al Android Studio este facilitarea unui emulator de dispozitive Android integrat, care permite dezvoltatorilor să testeze și să ruleze aplicațiile lor în diferite configurații de dispozitive virtuale. Aceste dispozitive emulare pot simula inclusiv poziția geografică, interacțiuni tactile, orientarea sau acceleratia telefonului.

3.3 Tehnologii folosite pentru realizarea infrastructurii server-side

În această secțiune, se vor prezenta tehnologiile utilizate pentru realizarea infrastructurii server-side.

3.3.1 Google Cloud Platform

Google Cloud Platform (GCP) este o platformă de servicii cloud oferite de Google. Printre numeroasele servicii puse la dispoziție, aceasta oferă posibilitatea creerii de mașini virtuale în cloud pe infrastructura gestionată de Google sau servicii de Kubernetes.

3.3.2 Ubuntu Server

Ubuntu Server este o distribuție de Linux dezvoltată de Canonical care este specializată pentru a rula pe servere și infrastructuri de rețea. Aceasta oferă o soluție fiabilă și scalabilă pentru gestionarea și implementarea serviciilor și aplicațiilor.

3.3.3 MySQL Server

MySQL Server este un sistem de gestionare a bazelor de date relaționale open-source popular deținut de Oracle Corporation. Aceasta oferă un mediu robust și scalabil pentru stocarea,

gestionarea și accesarea datelor într-o manieră sigură și eficientă. Unul dintre avantajele MySQL Server este performanța sa ridicată datorită tehniciilor avansate de optimizare a interogărilor și de indexare.

3.3.4 Apache2

Apache2 este un server web open-source care oferă o platformă pentru găzduirea de pagini web și aplicații. A fost inițial dezvoltat de Apache Software Foundation și este în prezent utilizat pe scară largă.

Arhitectura folosită de Apache2 este una client-server prin care serverul de Apache2 primește cereri HTTP de la clienți precum browsere web, urmând să le furnizeze conținutul solicitat, precum pagini web sau fișiere.

Serverul Apache2 este capabil să gestioneze multiple cereri simultan, distribuind eficient resursele disponibile pentru a asigura scalabilitatea și performanța.

3.3.5 PHP

PHP (Hypertext Preprocessor) este un limbaj de programare de uz general open-source, utilizat în special pentru dezvoltarea aplicațiilor web. PHP este interpretat de server și incorporat ulterior în codul HTML.

PHP oferă o gamă largă de funcții și caracteristici precum gestionarea formularelor, interacțiunea cu șiruri de caractere sau accesul facil la baze de date prin intermediul unor extensii specializare, cum ar fi MySQL.

3.3.6 Python

Python este un limbaj de programare interpretat foarte versatil, remarcat prin sintaxa simplă, clară și concisă. A fost creat de Guido van Rossum și a câștigat popularitate multumită modului intuitiv de folosire și abordarea de a rezolva probleme eficient.

Python oferă un set extins de biblioteci și module standard prin intermediul cărora sunt acoperite o gamă largă de funcționalități, precum crearea de cereri HTTP sau manipularea șirurilor de caractere.

3.3.7 Plate Recognizer API

Plate Recognizer API este un serviciu de recunoaștere a plăcuțelor de înmatriculare bazat pe machine learning și computer vision. Prin intermediul acestui API, dezvoltatorii pot trimite imagini sau fluxuri video către serviciu pentru a obține rezultate precise și rapide referitoare la plăcuțele de înmatriculare identificate în acestea. Răspunsul primit de la API în urma cererii HTTP este unul structurat în format JSON, care include diverse informații, precum țara, regiunea și numărul plăcuței de înmatriculare.

3.4 Componente hardware folosite

În această secțiune, se vor prezenta componentele hardware utilizate în montajul folosit pentru un senzor de parcare.

3.4.1 ESP32-CAM

Ai Thinker

Varianta de ESP32-CAM oferită de Ai Thinker reprezintă o placă de dezvoltare compactă, având suport pentru atașarea unui modul de cameră OV-2640 și 9 pini GPIO. Aceasta oferă și un modul bluetooth și Wi-Fi integrat.

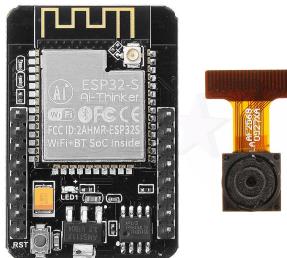


Figura 3.1: Imagine corespunzătoare plăcii de dezvoltare ESP32-CAM Ai Thinker. Sursa: <https://3dstar.ro/modul-esp32-cam>

Model	ESP32-CAM
Package	DIP-16
Size	27*40.5*4.5 (± 0.2) mm
RAM	Internal 520KB+ external 8MB PSRAM
Wi-Fi	802.11 b/g/n/e/i
Bluetooth	Bluetooth 4.2 BR/EDR and BLE standards
Working Temperature	-20 °C ~ 70 °C
Storage Environment	-40 °C ~ 125 °C, <90%RH
Support interface	UART、SPI、I2C、PWM
IO	9
Series Rate	Default 115200 bps
Security	WPA/WPA2/WPA2-Enterprise/WPS
SPI Flash	Default 32Mbit

Figura 3.2: Informații tehnice despre placă de dezvoltare ESP32-CAM oferită de AiThinker. Sursa: http://www.ai-thinker.com/pro_view-24.html.

Freenove WROVER Kit

Spre deosebire de varianta de ESP32-CAM oferită de Ai Thinker, cea oferită de Freenove oferă un număr mai mare de pini GPIO, respectiv 30. Și această placă oferă suport pentru comunicare Wi-Fi și Bluetooth prin intermediul modulelor integrate, însă spre deosebire de varianta celor de la AiThinker, oferă și un programator integrat pe placă ce se poate conecta la alte dispozitive prin intermediul unui cablu microUSB.



Figura 3.3: Imagine corespunzătoare plăcii de dezvoltare ESP32-CAM Freenove WROVER Kit. Sursa: https://github.com/Freenove/Freenove_ESP32_WROVER_Board

3.4.2 OV-2640 Camera module

Modulul de cameră OV2640 este un senzor de imagine CMOS ce poate captura imagini de până la 2MP. Acesta este utilizat frecvent în aplicații embedded și IoT datorită dimensiunilor sale compacte, consumului redus de energie și capacitaților sale de captare a imaginilor. Interfața de comunicare a senzorului este DVP (digital video port), o interfață de comunicare paralelă pe 8 biți.



Figura 3.4: Imagine corespunzătoare modulului de cameră OV2640. Sursa: <https://www.arducam.com/ov2640/>

3.4.3 HC-SR04 Sonar module

HC-SR04 este un senzor de distanță ultrasonic utilizat pentru a măsura distanța dintre senzor și un obiect. Acest modul este folosit frecvent în proiecte de robotică, IoT și alte aplicații care necesită detectarea și măsurarea distanțelor.

Senzorul HC-SR04 utilizează principiul ecouului ultrasonic pentru a măsura distanța. Acesta emite un semnal ultrasonic și măsoară timpul necesar pentru ca semnalul să se întoarcă, reflectat de obiectul din fața sa. Pe baza timpului de propagare a semnalului, senzorul poate calcula distanța față de obiect. Senzorul este compus dintr-un emițător și un receptor de ultrasunete. Emițătorul generează semnale ultrasonice cu frecvență de aproximativ 40 kHz, iar receptorul detectează semnalul reflectat. Senzorul are o gamă de măsurare de obicei între 2 cm și 4 m, iar precizia măsurătorilor depinde de calitatea și configurarea senzorului.



Figura 3.5: Imagine corespunzătoare senzorului ultrasonic de distanță HC-SR04. Sursa: <https://www.optimusdigital.ro/ro/senzori-senzori-ultrasonici/9-senzor-ultrasonic-hc-sr04-.html>

3.5 Tehnologii folosite pentru programarea senzorilor

În această secțiune se vor prezenta tehnologiile utilizate pentru programarea plăcii de dezvoltare ESP32-CAM.

3.5.1 MicroPython

MicroPython este o variantă a limbajului de programare Python optimizată pentru dispozitive cu resurse limitate, precum microcontrolere. Acesta a fost dezvoltat de către Damien George și este cunoscut pentru dimensiunea redusă și eficiența oferită.

MicroPython oferă un set de funcții și biblioteci specifice, adaptate microcontrolerelor, precum biblioteci de control al pinilor de intrare/ieșire sau biblioteci de gestiune a diferitelor module hardware (e.g. Wi-Fi, Bluetooth).

Unul dintre avantajele MicroPython este portabilitatea sa și suportul extins pentru o gamă largă de microcontrolere și plăci de dezvoltare, precum ESP8266, ESP32 sau Raspberry Pi Pico.

3.5.2 Thonny

Thonny este un mediu de dezvoltare integrat specializat pentru limbajul de programare Python, conceput pentru a fi ușor de utilizat. Acesta este utilizat pentru dezvoltarea de programe cu MicroPython, oferind suport direct pentru acesta și putând comunica cu dispozitivele care rulează MicroPython, precum microcontrolerle sau plăcile de dezvoltare compatibile.

Capitolul 4

Proiectarea sistemului

4.1 Diagrame UML

În această secțiune, se va prezenta o parte teoretică corespunzătoare diagramelor UML, precum și diagrama UML a cazurilor de utilizare concepută în etapa de proiectare a sistemului.

4.1.1 Unified Modeling Language

Unified Modeling Language [3], sau UML, este un limbaj standardizat de modelare ce constă într-un set de diagrame concepute pentru a ajuta dezvoltatorii de sisteme și programe prin vizualizarea și documentarea caracteristicilor unui sistem și a modurilor în care diverse componente ale acestuia interacționează cu celelalte.

4.1.2 Diagrama UML a cazurilor de utilizare

Diagrama UML cazurilor de utilizare modelează funcționalitatea sistemului din prisma utilizatorilor, denumiți actori. Un caz de utilizare este o unitate a funcționalității exprimată sub forma unei tranzacții dintre actori și sistem. Scopul diagramei de cazuri de utilizare este de a prezenta lista actorilor și cazurilor de utilizare și de a arăta ce actor participă în fiecare caz de utilizare.

4.1.3 Sistemul

Sistemul are la bază funcționalitatea de căutare și navigare către locurile de parcare disponibile. Acest lucru se realizează prin intermediul modulelor compuse din ESP32-CAM și HC-SR04 în următorul mod. Placa de dezvoltare ESP32-CAM amplasată pe locul de parcare va folosi senzorul ultrasonic de distanță pentru a detecta faptul că o mașină este parcată pe locul de parcare din fața sa. Ulterior, prin intermediul unei conexiuni Wi-Fi, va transmite o cerere HTTP către serverul centralizat care va conține identificatorul senzorului, starea de ocupare și o captură foto realizată la momentul ocupării locului de parcare.

Serverul va procesa cererea HTTP prin intermediul unui script de PHP pentru a stoca imaginea primită, ulterior apelând un script de Python pentru a procesa informațiile din imagine. Acest script va transmite din nou imaginea printr-o altă cerere HTTP către API-ul de recunoaștere de numere de înmatriculare, urmând să primească în corpul răspunsului diverse informații în format JSON despre ce mașină se află în imagine și care este numărul de înmatriculare al acesteia.

În continuare, se va interacționa cu baza de date pentru a se introduce o intrare în tabelul corespunzător stării locurilor de parcare prin care se va marca ocuparea locului.

Prin interacțiunea cu aceeași bază de date, aplicația mobilă folosită de utilizatori va prelua starea locurilor de parcare corespunzătoare senzorilor dintr-o anumită aria geografică dorită de utilizator pentru a găsi un loc de parcare disponibil.

4.1.4 Actorii

Când vine vorba de aplicația mobilă, există două categorii de actori, respectiv *Cetățeni* și *Administratori*. Administratorii vor avea acces la același set de funcționalități ca și cetătenii, însă, pe deasupra, vor putea manipula informațiile corespunzătoare senzorilor în baza de date.

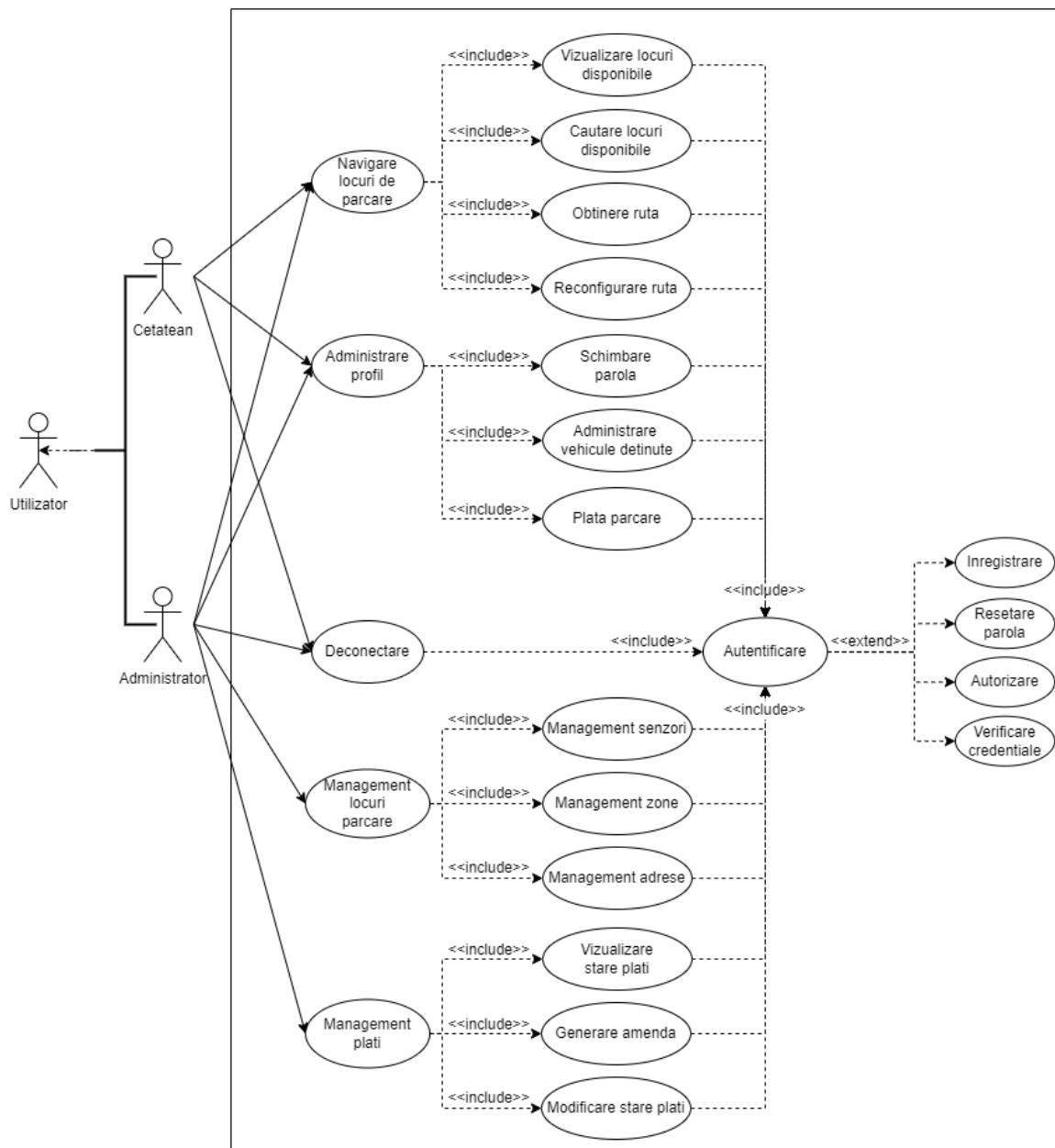


Figura 4.1: Diagrama UML a cazurilor de utilizare: actorii și principalele operații puse la dispoziția acestora.

4.1.5 Cazul de adăugare al unui loc de parcare

Datoria de adăugare a informațiilor despre un nou loc de parcare corespunzător unui senzor nou-amplasat în teren revine administratorilor. Senzorii vor transmite informația despre starea lor într-un tabel de ocupare, însă informațiile mai detaliate, precum latitudinea și longitudinea senzorului, sau zona tarifară în care se află senzorul vor trebui adăugate ulterior de administratori prin intermediul aplicației mobile.

4.2 Proiectarea bazei de date

Baza de date MySQL a fost proiectată conform unei diagrame entitate relație [4.2](#). Aceasta a fost supusă normalizăriilor aferente acestui tip de proiectare.

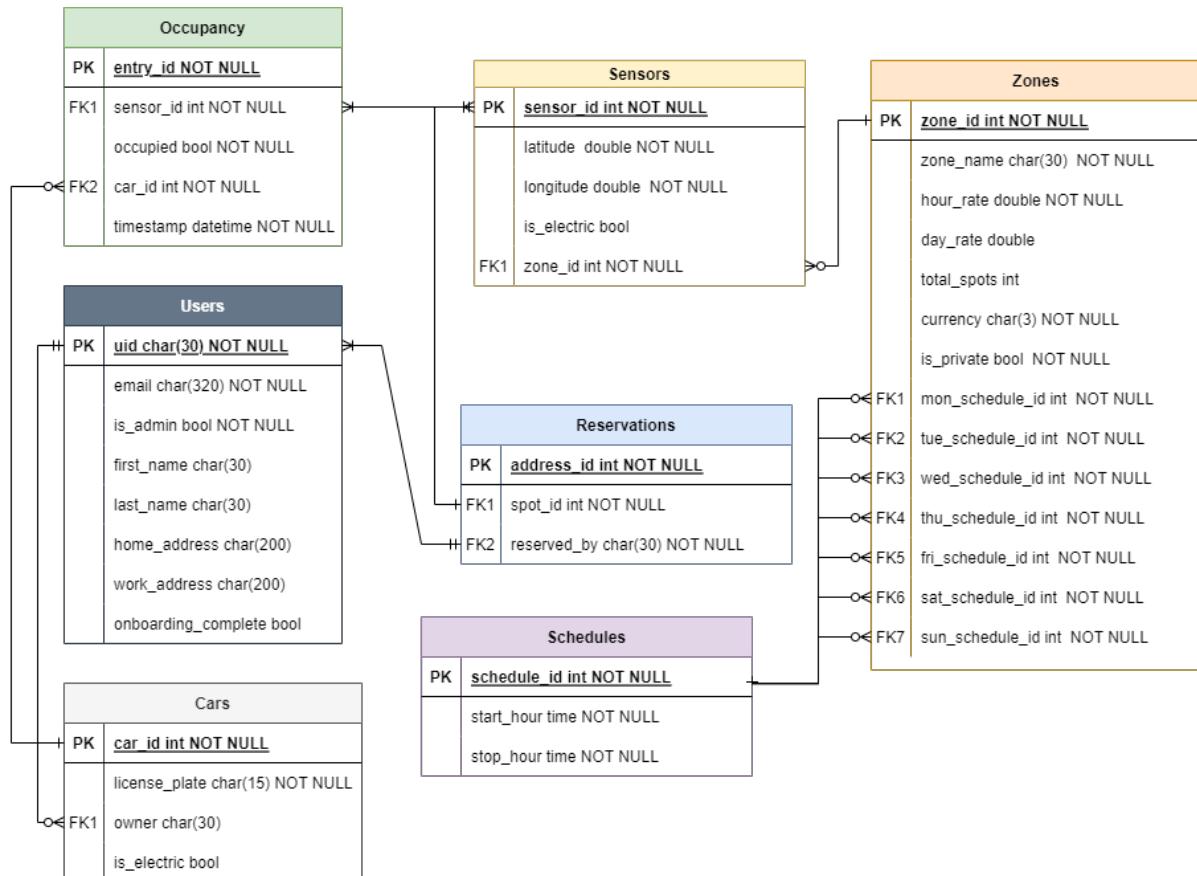


Figura 4.2: Diagrama Entitate-Relație a bazei de date. Tabelele și relațiile dintre acestea.

4.3 Configurația hardware:

Pentru montajul hardware, s-au folosit următoarele elemente:

- Senzorul ultrasonic HC-SR04
- Placa de dezvoltare ESP32-CAM
- Două LED-uri
- Două rezistențe de 220Ω

Acstea au fost interconectate conform figurii 4.3.

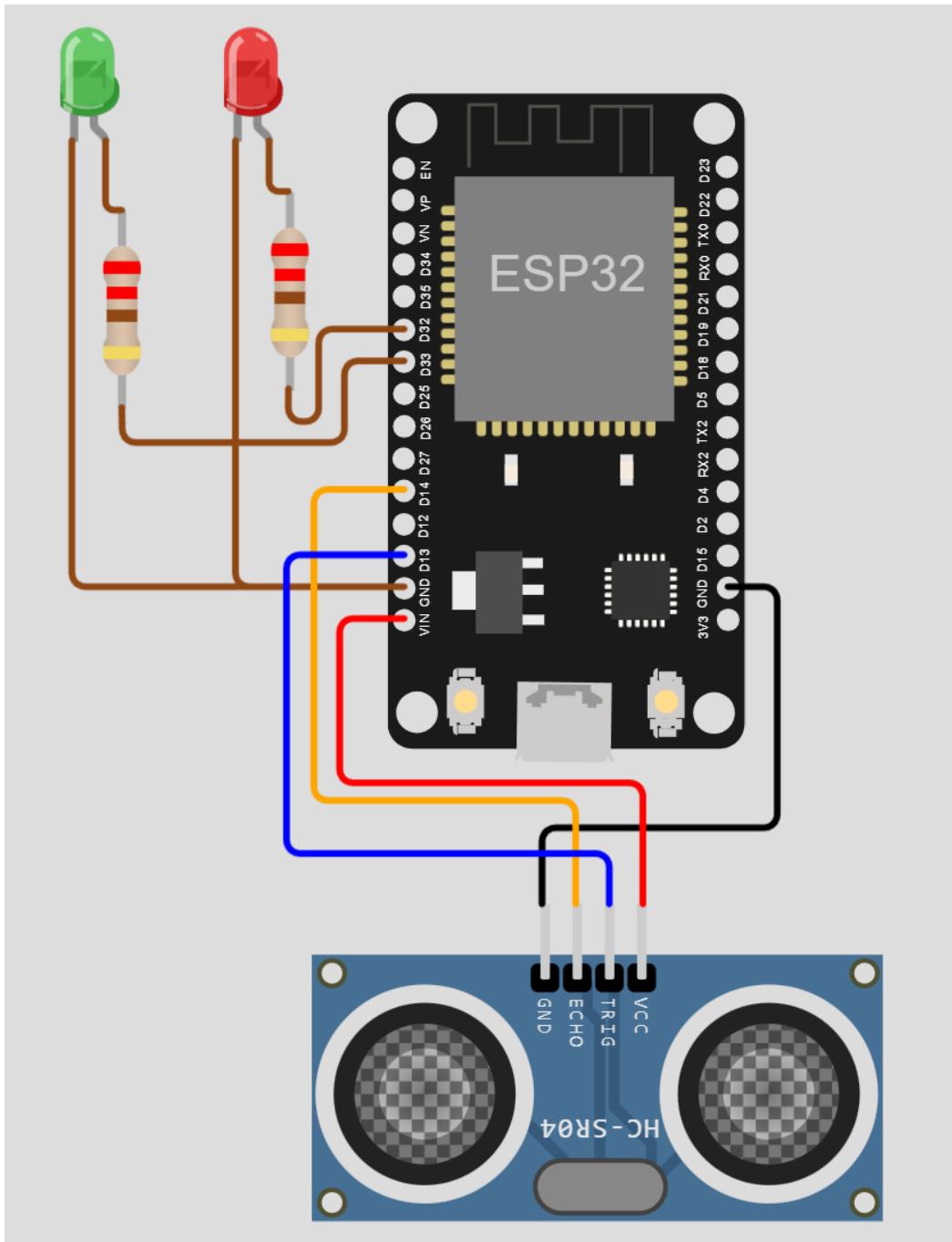


Figura 4.3: Schema circuitului.

Capitolul 5

Implementarea aplicației mobile

5.1 Modelul informațional

5.1.1 Modelul pentru utilizatori

Clasa IsarUser reprezintă modelul informațional folosit pentru stocarea informațiilor despre utilizatori în baza de date locală. Utilizatorii vor fi identificați în baza de date locală prin un identificator unic de tip Id, care va avea mereu valoarea 0, datorită faptului că un singur utilizator poate fi autentificat la un moment dat.

Celelalte câmpuri prezente în modelul pentru utilizatori sunt:

- o valoare de tip String, *uid* corespunzătoare identificatorului unic al utilizatorului oferit de Firebase
- o valoare de tip String pentru adresa de e-mail a utilizatorului
- două valori de tip String pentru nume și prenume
- două valori de tip String pentru adresa de acasă și de la lucru a utilizatorului; acestea vor fi stocate în format JSON
- o valoare booleană, *isAdmin* prin intermediul căreia se vor efectua diferite autorizări
- variabila booleană *onboardingComplete* prin care se va verifica starea completării formularului de bun-venit

```

class IsarUser {
  final Id id = 0;

  @Index(type: IndexType.value)
  String uid;

  String email;
  bool isAdmin = false;
  String firstName;
  String lastName;
  String homeAddress;
  String workAddress;
  bool onboardingComplete;

  IsarUser(
    {required this.uid,
    required this.email,
    required this.isAdmin,
    required this.firstName,
    required this.lastName,
    required this.homeAddress,
    required this.workAddress,
    required this.onboardingComplete});
}

```

Secvența de cod 5.1: "Modelul informațional folosit pentru utilizatori."

5.1.2 Modelul de adresă

Clasa Address va stoca informații despre adrese, conținând câmpuri pentru stradă, oraș, regiune și țară.

Această clasă poate fi instantiată și prin intermediul unui text în format JSON, respectiv transformată într-un text în format JSON.

```

class Address {
  late String street;
  late String city;
  late String region;
  late String country;

  Address(this.street, this.city, this.region, this.country);

  Address.fromJson(Map<String, dynamic> json)
    : street = json['street'] ?? '',
      city = json['city'] ?? '',
      region = json['region'] ?? '',
      country = json['country'] ?? '';

  Map<String, dynamic> toJson() =>
    {'street': street, 'city': city, 'region': region, 'country': country};

  @override
  String toString() {
    return "$street, $city";
  }
}

```

Secvența de cod 5.2: "Modelul informațional folosit pentru adrese."

5.1.3 Modelul de mașină

Clasa IsarCar este folosită pentru a reprezenta o mașină deținută de utilizatori.

Fiecare instanță va avea un identificator local unic de tip Id, care va fi incrementat automat. Câmpurile clasei constau în:

- un identificator de tip întreg, *carId*
- identificatorul unic al proprietarului, *ownerUid*, de tip String
- valoarea de tip String *licensePlate* pentru numărul de înmatriculare
- o valoare booleană, *isElectric* pentru a marca faptul că vehiculul este sau nu electric

```
class IsarCar {
    final Id id = Isar.autoIncrement;

    @Index(type: IndexType.value)
    int? carId;
    @Index(type: IndexType.value)
    String ownerUid;
    @Index(type: IndexType.value, unique: true)
    String licensePlate;
    bool isElectric;

    IsarCar(
        {this.carId,
        required this.ownerUid,
        required this.licensePlate,
        required this.isElectric});
}
```

Secvența de cod 5.3: "Modelul informațional folosit pentru mașinile utilizatorilor."

5.1.4 Modelul unui senzor

Clasa SpotInfo va modela informațiile corespunzătoare unui senzor aflat în teren.

Acesta va conține:

- identificatorul unic al locului de parcare, respectiv valoarea întreagă *sensorId*
- două valori reale corespunzătoare latitudinii și longitudinii la care se află senzorul
- o valoare booleană, *isElectric* ce va indica dacă la locul de parcare respectiv se află și o stație de încărcare pentru mașini electrice
- un obiect de tip Address, corespunzătoare adresei senzorului respectiv
- un obiect de tip Zone, corespunzătoare zonei tarifare în care este instalat senzorul
- un obiect de tip SpotState, care va corespunde stării de ocupare a senzorului

```
enum SpotState { free, occupied, reserved, freeingSoon, unknown }

class SpotInfo {
    int sensorId;
    double latitude;
    double longitude;
    bool isElectric;
    Address address;
    Zone zone;
    SpotState state;
    SpotInfo(this.sensorId, this.latitude, this.longitude, this.isElectric,
        this.address, this.zone, this.state);
}
```

Secvența de cod 5.4: "Modelul informațional folosit pentru senzori."

5.1.5 Modelul unei plăți de parcare

Clasa `ParkingPayment` va conține informații despre plățile tarifului de parcare.

Această clasă este gândită astfel încât să modeleze trei stări diferite ale parcării, respectiv:

- Mașina a părăsit locul de parcare și plata nu a fost încă efectuată
- Mașina a părăsit locul de parcare și plata a fost încă efectuată cu succes
- Mașina nu a părăsit locul de parcare și plata nu poate fi efectuată până la finalizarea parcării

Fiecare instanță a acestei clase va avea următoarele câmpuri:

- o valoare de tip întreg, `id`, care va fi non-null doar în cazul plăților finalizate
- un obiect de tip `spot`, corespunzător locului de parcare corespunzător plății
- un obiect de tip `IsarCar`, corespunzător mașinii care este sau a fost parcată pe locul respectiv
- un obiect de tip `DateTime?` `timestamp`, care va fi nul doar în cazul plăților finalizate
- un obiect de tip `DateTime` `parkingStart`, ce va marca momentul de început al duratei de parcare
- un obiect de tip `DateTime?` `parkingEnd`, ce va marca momentul de finalizare al duratei de parcare; acesta va fi null în cazul parcărilor nefinalizate
- un obiect de tip `Duration` corespunzător duratei de parcare pentru care se efectuează plata

```
enum PaymentState { ongoing, due, paid }

class ParkingPayment {
    int? id;
    SpotInfo spot;
    final IsarCar car;
    double totalSum;
    final DateTime? timestamp;
    final DateTime parkingStart;
    final DateTime? parkingEnd;
    final PaymentState state;
    final Duration parkingDuration;
    final int? startEntryId;
    final int? endEntryId;

    ParkingPayment(
        {this.id,
        required this.spot,
        required this.car,
        required this.totalSum,
        this.timestamp,
        required this.parkingStart,
        this.parkingEnd,
        required this.state,
        required this.parkingDuration,
        this.startEntryId,
        this.endEntryId});
}
```

Secvența de cod 5.5: "Modelul informațional folosit pentru plăți."

5.1.6 Modelul pentru zone tarifare

Clasa Zone modelează zonele tarifare. Fiecare zonă va avea un nume, un tarif pe oră, un tarif optional pe zi, numele monedei de tranzacționare precum și orarul de taxare a zonei tarifare respective. Variabila booleană isPrivate și valoarea întreagă totalSpots sunt specifice parcărilor private, în cazul extinderii aplicației spre acest gen de parcări în viitor.

```
class Zone {
    int id;
    String name;
    double hourRate;
    double? dayRate;
    String currency;
    bool isPrivate = false;
    int? totalSpots;
    Schedule schedule;

    Zone(this.id, this.name, this.hourRate, this.dayRate, this.currency,
        this.isPrivate, this.totalSpots, this.schedule);
}
```

Secvența de cod 5.6: "Modelul informațional folosit pentru zone tarifare."

5.2 Serviciile aplicației

5.2.1 Serviciul de autentificare

Metoda de autentificare

Autentificarea se realizează cu ajutorul metodei asincrone *Future<String> signInWithEmailAndPassword(String email, String password)*, ce primește ca parametru adresa de e-mail a utilizatorului și parola acestuia și returnează o valoare de tip String corespunzătoare stării de succes a autentificării.

Prin apelarea metodei de autentificare pe instanța *_auth* de tip *FirebaseAuth*, ce va verifica existența utilizatorului înregistrat cu credențialele primite ca parametru. În caz de succes, se va construi un obiect nou corespunzător modelului de utilizator folosit de serviciul de stocare locală (5.2.2), *IsarUser*, respectiv lista mașinilor deținute de utilizator, reprezentate prin obiecte de tip *IsarCar*. Acestea informații se vor obține prin intermediul serviciului de MySQL (5.2.3) și vor fi stocate local în baza de date Isar [4].

```

// sign in with email&password
Future<String> signInWithEmailAndPassword(
    String email, String password) async {
  try {
    UserCredential result = await _auth.signInWithEmailAndPassword(
        email: email, password: password);
    // user will be stored locally
    User? user = result.user;
    if (user != null && user.email != null) {
      IsarUser? isarUser = await SqlService.getUser(user.uid);
      if (isarUser == null) {
        throw FirebaseAuthException(code: 'sql-error');
      }
      List<IsarCar>? isarCars = await SqlService.getUserCars(user.uid);

      if (isarCars == null) {
        throw FirebaseAuthException(code: 'sql-error');
      }
      await IsarService.setUser(isarUser);
      await IsarService.setUserCars(isarCars);
    }
  } on FirebaseAuthException catch (e) {
    if (e.code == 'user-not-found') {
      return 'An user with this e-mail/password was not found.';
    } else if (e.code == 'wrong-password') {
      return 'An user with this e-mail/password was not found.';
    } else if (e.code == 'sql-error') {
      return 'We had some trouble signing you in, please try again.';
    } else {
      return 'There was a problem trying to sign you in. Please check your internet connection.';
    }
  }
  return "success";
}

```

Secvența de cod 5.7: "Corpul metodei de autentificare."

Metoda de înregistrare în aplicație

Înregistrarea în aplicație se realizează prin metoda asincronă `Future<String> registerWithEmailAndPassword(String email, String password)`. Aceasta va crea în Firebase un utilizator nou cu adresa de e-mail și parola primită ca parametru, ulterior adăugând o nouă intrare în tabela de utilizatori din MySQL corespunzătoare utilizatorului cu această adresă de e-mail și identificator unic.

În cazul înregistrării cu succes a utilizatorului, datele corespunzătoare acestuia vor fi stocate local prin intermediul bazei de date Isar.

```
// register in with email&password
Future<String> registerWithEmailAndPassword(
    String email, String password) async {
  try {
    UserCredential result = await _auth
        .createUserWithEmailAndPassword(email: email, password: password)
        .timeout(Constants.timeoutDuration);
    // user will be saved in provider
    User? user = result.user;
    if (user != null && user.email != null) {
      try {
        await SqlService.addUserToDatabase(
            user.uid, user.email!, false, '', '', '', '');
      } catch (e) {
        await user.delete();
        return 'Registration_failed';
      }
      await IsarService.createUserFromFirestoreUser(user, false);
      return "success";
    } else {
      return 'Registration_failed';
    }
  } on FirebaseAuthException catch (e) {
    if (e.code == 'weak-password') {
      return 'Weak_password';
    } else if (e.code == 'email-already-in-use') {
      return 'This e-mail is already in use!';
    }
  } catch (e) {
    return e.toString();
  }
  return "success";
}
```

Secvența de cod 5.8: "Corpu metodei de înregistrare."

Metoda de deconectare

Această metodă va șterge datele locale corespunzătoare utilizatorului autentificat și va semnala încheierea sesiunii de autentificare în Firebase, realizându-se astfel procesul de deconectare a utilizatorului.

```
// sign-out
Future signOut() async {
  try {
    await IsarService.deleteLocalUser();
    return await _auth.signOut();
  } catch (e) {
    return null;
  }
}
```

Secvența de cod 5.9: "Corpu metodei de deconectare."

5.2.2 Serviciul de stocare locală

Metoda de inițializare a utilizatorului curent

Această metodă are scopul de a prelua utilizatorul curent din baza de date locală. În cazul în care utilizatorul nu este autentificat, se va folosi un utilizator „gol”. Verificarea stării de autentificare se realizează prin intermediul valorii câmpului „uid” al utilizatorului local.

```

static Future<void> initUser() async {
  isarUser = await isar.isarUsers.get(0) ??
    IsarUser(
      uid: '',
      email: '',
      isAdmin: false,
      firstName: '',
      lastName: '',
      homeAddress: '',
      workAddress: '',
      onboardingComplete: false);
  await isar.writeTxn(() async {
    await isar.isarUsers.put(isarUser);
  });
}

```

Secvența de cod 5.10: "Metoda de inițializare a utilizatorului."

Metoda de creare a unui utilizator pe baza unui user oferit de Firestore

Metoda *static Future<void> createUserFromFirestoreUser(User user, bool isAdmin)* primește ca parametru un obiect de tip User, rezultat în urma autentificării cu serviciile Firebase, și construiește pe baza acestuia o instantă a clasei IsarUser pe care o va stoca local.

```

static Future<void> createUserFromFirestoreUser(
  User user, bool isAdmin) async {
  await isar.writeTxn(() async {
    await isar.isarUsers.delete(isarUser.id);
    IsarUser isarUserFromFirestoreUser = IsarUser(
      uid: user.uid,
      email: user.email ?? '',
      isAdmin: isAdmin,
      firstName: '',
      lastName: '',
      homeAddress: '',
      workAddress: '',
      onboardingComplete: false);

    isarUser = isarUserFromFirestoreUser;
    await isar.isarUsers.put(isarUser);
  });
}

```

Secvența de cod 5.11: "Metoda de creare a unui utilizator pe baza unui user oferit de Firestore."

Metode de interacționare cu profilul de utilizator

Următoarea serie de metode efectuează operații de actualizare, ștergere a informațiilor locale despre profilul de utilizator.

```

static Future<void> setUser(IsarUser user) async {
  isarUser = user;
  await isar.writeTxn(() async {
    await isar.isarUsers.put(user);
  });
}

```

Secvența de cod 5.12: "Metoda de setare locală a utilizatorului curent"

```

static Future<void> updateUser() async {
  await isar.writeTxn(() async {
    await isar.isarUsers.put(isarUser);
  });
}

```

Secvența de cod 5.13: "Metoda de actualizare a datelor locale ale utilizatorului curent"

```

static Future<void> deleteLocalUser() async {
    await isar.writeTxn(() async {
        await isar.isarCars.filter().ownerUidEqualTo(isarUser.uid).deleteAll();
        await isar.isarUsers.delete(isarUser.id);
    });
    isarCars = [];
    await initUser();
}

```

Secvența de cod 5.14: "Metoda de stergere a datelor locale despre utilizatorul curent"

Următoarele trei metode au rolul de a interacționa cu lista de mașini corespunzătoare utilizatorului autentificat:

```

static Future<void> setUserCars(List<IsarCar> cars) async {
    isarCars = cars;
    await isar.writeTxn(() async {
        await isar.isarCars.putAll(cars);
    });
}

```

Secvența de cod 5.15: "Metoda de setare locală a mașinilor utilizatorului curent"

```

static Future<void> addUserCar(IsarCar car) async {
    isarCars.add(car);
    await isar.writeTxn(() async {
        await isar.isarCars.put(car);
    });
}

```

Secvența de cod 5.16: "Metoda de adăugare a unei noi mașini în lista de mașini a utilizatorului curent"

```

static deleteUserCar(IsarCar car) async {
    isarCars.removeWhere((element) => element.licensePlate == car.licensePlate);
    await isar.writeTxn(() async {
        await isar.isarCars
            .filter()
            .licensePlateEqualTo(car.licensePlate)
            .deleteAll();
    });
}

```

Secvența de cod 5.17: "Metoda de stergere a unei mașini din lista de mașini ale utilizatorului curent"

Metoda de marcare a completării formularului de bun-venit

Metoda asincronă *static Future<void> markOnboardingCompleted()* va marca local faptul că formularul de bun-venit a fost completat cu succes de utilizator.

```

static Future<void> markOnboardingCompleted() async {
    isarUser.onboardingComplete = true;
    await updateUser();
}

```

Secvența de cod 5.18: "Metoda de marcare locală a completării formularului de bun-venit"

5.2.3 Serviciul de MySQL

Metoda de adăugare a unui utilizator în baza de date

Această metodă este apelată în timpul procesului de înregistrare. Prin intermediul acesteia, se va introduce o nouă intrare în tabela de utilizatori ce va corespunde unui nou cont de utilizator.

```

static Future<void> addUserToDatabase(
    String uid,
    String email,
    bool isAdmin,
    String firstName,
    String lastName,
    String licensePlate,
    String homeAddress,
    String workAddress) async {
    await pool.execute(
        "INSERT INTO `Users`(`uid`, `email`, `is_admin`, `first_name`, `last_name`, `license_plate`, `home_address`, `work_address`) VALUES (:uid, :email, :is_admin, :first_name, :last_name, :license_plate, :home_address, :work_address)",
        {
            "uid": uid,
            "email": email,
            "is_admin": isAdmin,
            "first_name": firstName,
            "last_name": lastName,
            "license_plate": licensePlate,
            "home_address": homeAddress,
            "work_address": workAddress,
        }).timeout(Constants.sqlTimeoutDuration,
        onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
}

```

Secvența de cod 5.19: "Metoda de adăugare a unui utilizator în baza de date."

Metoda de preluare a datelor unui utilizator

Această metodă este apelată în timpul procesului de autentificare. Prin intermediul acesteia, se vor prelua datele corespunzătoare utilizatorului cu identificatorul unic trimis ca parametru.

```

static Future<IsarUser?> getUser(String uid) async {
    try {
        var result = await pool.execute("SELECT * FROM Users WHERE uid = :uid", {
            "uid": uid,
        }).timeout(Constants.sqlTimeoutDuration,
            onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
        ResultSetRow data = result.rows.first;
        bool isAdmin = data.typedColByName<bool>("is_admin")!;
        bool onboardingComplete =
            data.typedColByName<bool>("onboarding_complete")!;
        String email = data.typedColByName<String>("email")!;
        String firstName = data.typedColByName<String>("first_name") ?? '';
        String lastName = data.typedColByName<String>("last_name") ?? '';
        String homeAddress = data.typedColByName<String>("home_address") ?? '';
        String workAddress = data.typedColByName<String>("work_address") ?? '';
        return IsarUser(
            uid: uid,
            email: email,
            isAdmin: isAdmin,
            firstName: firstName,
            lastName: lastName,
            homeAddress: homeAddress,
            workAddress: workAddress,
            onboardingComplete: onboardingComplete);
    } catch (e) {
        return null;
    }
}

```

Secvența de cod 5.20: "Metoda de preluare a datelor corespunzătoare unui profil de utilizator."

Metoda de actualizare a datelor unui utilizator

Această metodă va actualiza datele intrării corespunzătoare utilizatorului curent prin sincronizarea informațiilor din baza de date MySQL cu cele aflate în baza de date Isar (locală).

Aceasta este folosită în momentul finalizării formularului de bun-venit, precum și pentru a transmite orice posibile modificări realizate în pagina de vizualizare a profilului de utilizator.

```
static Future<void> pushLocalUserData() async {
  IsarUser isarUser = IsarService.isarUser;
  await pool.execute(
    "UPDATE `Users` SET `first_name`=:first_name WHERE `uid`=:uid LIMIT 1",
    {
      "uid": isarUser.uid,
      "first_name": isarUser.firstName
    }).timeout(Constants.sqlTimeoutDuration,
    onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
  await pool.execute(
    "UPDATE `Users` SET `last_name`=:last_name WHERE `uid`=:uid LIMIT 1",
    {
      "uid": isarUser.uid,
      "last_name": isarUser.lastName
    }).timeout(Constants.sqlTimeoutDuration,
    onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
  await pool.execute(
    "UPDATE `Users` SET `home_address`=:home_address WHERE `uid`=:uid LIMIT 1",
    {
      "uid": isarUser.uid,
      "home_address": isarUser.homeAddress
    }).timeout(Constants.sqlTimeoutDuration,
    onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
  await pool.execute(
    "UPDATE `Users` SET `work_address`=:work_address WHERE `uid`=:uid LIMIT 1",
    {
      "uid": isarUser.uid,
      "work_address": isarUser.workAddress
    }).timeout(Constants.sqlTimeoutDuration,
    onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
}
```

Secvența de cod 5.21: "Metoda de actualizare a datelor corespunzătoare unui profil de utilizator."

Metoda de preluare a mașinilor unui utilizator

Metoda asincronă `static Future<List<IsarCar>?> getUserCars(String uid)` este folosită pentru a obține lista mașinilor utilizatorului corespunzător identificatorului unic trimis prin parametrul `uid` de tip `String`.

```
static Future<List<IsarCar>?> getUserCars(String uid) async {
  List<IsarCar> cars = [];

  try {
    var result = await pool.execute(
      "SELECT `car_id`, `license_plate`, `is_electric` FROM `Cars` WHERE `owner`=:uid",
      {
        "uid": uid,
      }).timeout(Constants.sqlTimeoutDuration,
      onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
    for (ResultSetRow row in result.rows) {
      int? carId = row.typedColByName<int>("car_id");
      String licensePlate = row.typedColByName<String>("license_plate") ?? '';
      bool isElectric = row.typedColByName<bool>("is_electric")!;
      cars.add(IsarCar(
        carId: carId,
        ownerUid: uid,
        licensePlate: licensePlate,
        isElectric: isElectric));
    }
  } catch (e) {
    return null;
  }
  return cars;
}
```

Secvența de cod 5.22: "Metoda de preluare a mașinilor unui utilizator."

Metoda de adăugare a unei mașini noi

Această metodă va adăuga în baza de date MySQL o intrare nouă corespunzătoare unei noi mașini, conform informațiilor oferite prin intermediul obiectului de tip IsarCar primit ca parametru.

În cazul în care o mașină cu același număr de înmatriculare există deja în baza de date, adăugarea duplicată a acesteia nu va fi posibilă.

```
static Future<void> addUserCar(IsarCar car) async {
    var result = await pool.execute(
        "SELECT `owner` FROM `Cars` WHERE `license_plate` = :license_plate", {
            "license_plate": car.licensePlate,
        }).timeout(Constants.sqlTimeoutDuration,
            onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
    if (result.rows.isNotEmpty) {
        String ownerUid = result.rows.first.typedColByName<String>("owner") ?? '';
        if (car.ownerUid == ownerUid) {
            throw Exception('You have already added this car');
        } else {
            throw Exception('Someone else has already claimed this vehicle');
        }
    } else {
        await pool.execute(
            "INSERT INTO `Cars`(`license_plate`, `owner`, `is_electric`) VALUES (:license_plate, :owner, :is_electric)",
            {
                "license_plate": car.licensePlate.toUpperCase(),
                "owner": car.ownerUid,
                "is_electric": car.isElectric,
            },
        ).timeout(Constants.sqlTimeoutDuration,
            onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
    }
    await IsarService.addUserCar(car);
}
```

Secvența de cod 5.23: "Metoda de preluare a datelor corespunzătoare unui profil de utilizator din baza de date."

Metoda de ștergere a unei mașini

Această funcție va șterge intrarea corespunzătoare mașinii cu valorile numărului de înmatriculare și proprietarului ale obiectului de tip IsarCar primit ca parametru.

```
static Future<void> deleteUserCar(IsarCar car) async {
    await pool.execute(
        "DELETE FROM `Cars` WHERE `license_plate` = :license_plate AND `owner` = :owner",
        {
            "owner": car.ownerUid,
            "license_plate": car.licensePlate,
        }).timeout(Constants.sqlTimeoutDuration,
            onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
    await IsarService.deleteUserCar(car);
}
```

Secvența de cod 5.24: "Metoda de adăugare a unei mașini noi în MySQL."

Metoda de adăugare a informațiilor corespunzătoare unui senzor

Metoda *static Future<String> addSensor(String sensorId, LatLng latLng, bool isElectric, int zoneId)* va adăuga o nouă intrare în tabela senzorilor din baza de date. Informațiile despre senzorul respectiv vor fi transmise prin cei patru parametrii, respectiv identificatorul unic al senzorului, un obiect de tip LatLng ce va conține coordonatele poziției geografice a senzorului,

variabila `isElectric` prin care se va indica dacă la locul de parcare respectiv se află o stație de încărcare electrică și identificatorul corespunzător zonei tarifare.

```
static Future<String> addSensor(
    String sensorId, LatLng latLng, bool isElectric, int zoneId) async {
  try {
    var res = await pool.execute(
      "INSERT INTO `Sensors`(`sensor_id`, `latitude`, `longitude`, `is_electric`, `zone_id`)" +
      "VALUES(:sensor_id, :latitude, :longitude, :is_electric, :zone_id)",
      {
        "sensor_id": int.parse(sensorId),
        "latitude": latLng.latitude,
        "longitude": latLng.longitude,
        "is_electric": isElectric,
        "zone_id": zoneId,
      },
    ).timeout(Constants.sqlTimeoutDuration,
      onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
    print(res.affectedRows);

    res = await pool.execute(
      "INSERT INTO `Occupancy`(`sensor_id`)VALUES(:sensor_id)",
      {
        "sensor_id": int.parse(sensorId),
      },
    ).timeout(Constants.sqlTimeoutDuration,
      onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
    print(res.affectedRows);
  } catch (e) {
    return e.toString();
  }
  return "Sensor added successfully";
}
```

Secvența de cod 5.25: "Metoda de adăugare a unui nou senzor."

Metoda de preluare a datelor despre senzorii din jurul unei poziții

Metoda asincronă `static Future<List<SpotInfo>> getParkingSpotsAroundPosition(double latitude, double longitude, double rangeKm)` va prelua informația din tabelul de senzori corespunzătoare tuturor senzorilor aflați pe o rază de `rangeKm` kilometri (parametru) față de poziția geografică reprezentată de coordonatele primite ca parametru.

```

static Future<List<SpotInfo>> getParkingSpotsAroundPosition(
    double latitude, double longitude, double rangeKm) async {
  List<SpotInfo> parkingInfo = [];
  LatLongRangeLimits limits =
      LocationService.getPointRadiusKm(latitude, longitude, rangeKm);
  try {
    var result = await pool.execute(
        "SELECT * FROM Sensors WHERE latitude >= :minLat AND latitude <= :maxLat AND "
        "longitude >= :minLong AND longitude <= :maxLong",
        {
          "minLat": limits.minLat,
          "maxLat": limits.maxLat,
          "minLong": limits.minLong,
          "maxLong": limits.maxLong,
        }).timeout(Constants.sqlTimeoutDuration,
        onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));

    print(
        "Fetched Sensor SQL info within $rangeKm of ($latitude; $longitude)");
    for (var row in result.rows) {
      int sensorId = row.typedColByName<int>("sensor_id")!;
      double lat = row.typedColByName<double>("latitude")!;
      double long = row.typedColByName<double>("longitude")!;
      bool isElectric = row.typedColByName<bool>("is_electric")!;
      int zoneId = row.typedColByName<int>("zone_id")!;

      SpotState spotState = await getSensorStatus(sensorId);

      if (spotState == SpotState.occupied) {
        // Skip occupied spots
        continue;
      }

      LatLng position = LatLng(lat, long);
      Address address = await LocationService.addressFromLatLng(lat, long);
      Zone zone = await getZoneById(zoneId);

      parkingInfo.add((SpotInfo(sensorId, position.latitude,
          position.longitude, isElectric, address, zone, spotState)));
    }
  } catch (e) {
    print(e.toString());
  }
  return parkingInfo;
}

```

Secvența de cod 5.26: "Metoda de preluare a datelor despre toți senzorii din jurul unei poziții."

Metoda de obținere a celui mai apropiat senzor relativ la o poziție geografică

Această metodă va returna un obiect de tip `SpotInfo?` corespunzător unui loc de parcare. Locul de parcare respectiv se va afla la o distanță de cel mult `rangeKm` kilometri de poziția corespunzătoare coordonatelor primite ca și parametru. În cazul în care nici un loc de parcare nu se află în raza respectivă, va fi returnat null.

```

static Future<SpotInfo?> getNearestAvailableParkingSpotWithinRange(
    double latitude, double longitude, double rangeKm) async {
  LatLongRangeLimits limits =
    LocationService.getPointRadiusKm(latitude, longitude, rangeKm);
  try {
    var result = await pool.execute(
        "SELECT * FROM Sensors WHERE latitude >= :minLat AND latitude <= :maxLat AND "
        "longitude >= :minLong AND longitude <= :maxLong",
        {
          "minLat": limits.minLat,
          "maxLat": limits.maxLat,
          "minLong": limits.minLong,
          "maxLong": limits.maxLong,
        }).timeout(Constants.sqlTimeoutDuration,
        onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));

    print(
        "Fetched Sensor SQL info within $rangeKm of ($latitude; $longitude)");
    for (var row in result.rows) {
      int sensorId = row.typedColByName<int>("sensor_id")!;
      SpotState spotState = await getSensorStatus(sensorId);
      if (spotState != SpotState.free) {
        continue;
      }
      double lat = row.typedColByName<double>("latitude")!;
      double long = row.typedColByName<double>("longitude")!;
      bool isElectric = row.typedColByName<bool>("is_electric")!;
      int zoneId = row.typedColByName<int>("zone_id")!;

      LatLng position = LatLng(lat, long);

      Address address = await LocationService.addressFromLatLng(lat, long);
      Zone zone = await getZoneById(zoneId);

      return (SpotInfo(sensorId, position.latitude, position.longitude,
          isElectric, address, zone, spotState));
    }
  } catch (e) {
    print(e.toString());
    return null;
  }
  return null;
}

```

Secvența de cod 5.27: "Metoda de obținere a celui mai apropiat senzor relativ la o poziție geografică."

Metoda de obținere a stării unui senzor

Această metodă va returna starea locului de parcare cu identificatorul *sensorId* transmis ca parametru.

```

static Future<SpotState> getSensorStatus(int sensorId) async {
  print("Fetching sensor $sensorId status");
  try {
    var occupancyQuery = await pool.execute(
      "SELECT o.occupied, r.reservation_count FROM (SELECT occupied FROM Occupancy WHERE sensor_id = :sensorId ORDER BY timestamp DESC LIMIT 1) AS o, (SELECT COUNT(*) AS reservation_count FROM Reservations WHERE spot_id = :sensorId) AS r",
    );
    "sensorId": sensorId
    ), timeout(Constants.sqlTimeoutDuration,
    onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
    ResultSetRow data = occupancyQuery.rows.first;
    bool occupied = data.typedColByName<bool>("occupied")!;
    int reservationCount = data.typedColByName<int>("reservation_count")!;

    if (occupied) {
      return SpotState.occupied;
    }

    if (reservationCount > 0) {
      return SpotState.reserved;
    }
    return SpotState.free;
  } catch (e) {
    print(e.toString());
    return SpotState.unknown;
  }
}

```

Secvența de cod 5.28: "Metoda de obținere a stării de ocupare a unui loc de parcare."

Metode pentru interacțiunea cu sistemul de rezervări al locurilor de parcare

Următoarele două metode au rolul de a rezerva, respectiv de a elibera rezervarea unui loc de parcare identificat prin valoarea intreagă transmisă ca parametru.

```

static Future<void> reserveSpot(int spotId) async {
  await pool.execute(
    "INSERT INTO `Reservations` (`spot_id`, `reserved_by`) VALUES (:spot_id, :uid)",
    {
      "spot_id": spotId,
      "uid": IsarService.isarUser.uid,
    }, timeout(Constants.sqlTimeoutDuration,
    onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
}

```

Secvența de cod 5.29: "Metoda de rezervare a unui loc de parcare."

```

static Future<void> clearSpotReservation(int spotId) async {
  await pool.execute(
    "DELETE FROM `Reservations` WHERE `spot_id` = :spot_id",
    {
      "spot_id": spotId,
      "uid": IsarService.isarUser.uid,
    }, timeout(Constants.sqlTimeoutDuration,
    onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
}

```

Secvența de cod 5.30: "Metoda de anulare a rezervărilor unui loc de parcare."

Metoda de preluare a istoricului de parcare a unui utilizator

Metoda *static Future<List<ParkingPayment> getUserParkingHistory()* va verifica starea plății taxei de parcare pentru toate parcările realizate de mașinile utilizatorului curent.

În primul rând, parcările care au fost plătite vor fi preluate din tabela de plăți efectuate. Totodată, parcările care necesită să fie plătite sau care sunt în desfășurare vor fi preluate din tabela de ocupare, tabelă ce va avea informația primită efectiv de la senzorii din teren.

```

static Future<List<ParkingPayment>> getUserParkingHistory() async {
  List<ParkingPayment> parkingHistoryList = [];
  List<IsarCar> cars = await getUserCars(IsarService.isarUser.uid) ?? [];

  for (IsarCar car in cars) {
    if (car.carId == null) {
      car.carId = await getCarIdByLicensePlate(car.licensePlate);
      if (car.carId == null) {
        // This case means the car is not added in SQL
        // so an error might have occurred somewhere
        continue;
      }
    }

    var result = await pool.execute(
      "SELECT * FROM `Payments` WHERE car_id = :car_id",
      {"car_id": car.carId});

    for (ResultSetRow row in result.rows) {
      int paymentId = row.typedColByName<int>('payment_id')!;
      int sensorId = row.typedColByName<int>('sensor_id')!;
      double totalSum = row.typedColByName<double>('total_sum')!;
      String timestamp = row.typedColByName<String>('timestamp')!;
      String parkingStart = row.typedColByName<String>('parking_start')!;
      String parkingEnd = row.typedColByName<String>('parking_end')!;

      SpotInfo? spot = await getSpotById(sensorId, fetchState: false);

      Duration duration = DateTime.parse(parkingStart)
        .difference(DateTime.parse(parkingEnd))
        .abs();
      if (spot != null) {
        parkingHistoryList.add(ParkingPayment(
          id: paymentId,
          spot: spot,
          car: car,
          totalSum: totalSum,
          timestamp: DateTime.parse(timestamp),
          parkingStart: DateTime.parse(parkingStart),
          parkingEnd: DateTime.parse(parkingEnd),
          state: PaymentState.paid,
          parkingDuration: duration));
      }
    }

    result = await pool.execute(
      "SELECT * FROM `Occupancy` WHERE car_id = :car_id ORDER BY timestamp ASC",
      {"car_id": car.carId});
  }

  List<ResultSetRow> resultList = result.rows.toList();
  int length = resultList.length;

  if (length.isOdd) {
    length--;
  }
}

```

Secvența de cod 5.31: "Metoda de preluare a situației plășilor pentru parcare corespunzătoare utilizatorului curent. 1/2"

```

        for (int i = 0; i < length; i += 2) {
            int sensorId = resultList[i].typedColByName<int>('sensor_id')!;
            int startEntryId = resultList[i].typedColByName<int>('entry_id')!;
            int endEntryId = resultList[i + 1].typedColByName<int>('entry_id')!;
            bool startOccupied = resultList[i].typedColByName<bool>('occupied')!;
            bool endOccupied = resultList[i + 1].typedColByName<bool>('occupied')!;
            if (!startOccupied || endOccupied) {
                continue;
            }
            String startTimestamp =
                resultList[i].typedColByName<String>('timestamp')!;
            String stopTimestamp =
                resultList[i + 1].typedColByName<String>('timestamp')!;
            SpotInfo? spot = await getSpotById(sensorId, fetchState: false);
            if (spot != null) {
                Duration duration = DateTime.parse(startTimestamp)
                    .difference(DateTime.parse(stopTimestamp))
                    .abs();
                double totalSum = 0;
                int minutes = duration.inMinutes % 60;
                int hours = duration.inHours % 24;
                int days = duration.inDays;
                if (spot.zone.dayRate != null) {
                    totalSum += days * spot.zone.dayRate!;
                    if (hours > 4) {
                        totalSum += spot.zone.dayRate!;
                    } else {
                        totalSum += spot.zone.hourRate * (minutes / 60 + hours);
                    }
                } else {
                    totalSum = duration.inMinutes / 60 * spot.zone.hourRate;
                }
                parkingHistoryList.add(ParkingPayment(
                    spot: spot,
                    car: car,
                    totalSum: totalSum,
                    parkingStart: DateTime.parse(startTimestamp),
                    parkingEnd: DateTime.parse(stopTimestamp),
                    state: PaymentState.due,
                    parkingDuration: duration,
                    startEntryId: startEntryId,
                    endEntryId: endEntryId));
            }
        }
        if (resultList.length.isOdd) {
            int sensorId = resultList[length].typedColByName<int>('sensor_id')!;
            SpotInfo? spot = await getSpotById(sensorId, fetchState: false);

            if (spot != null) {
                String startTimestamp =
                    resultList[length].typedColByName<String>('timestamp')!;
                Duration duration =
                    DateTime.parse(startTimestamp).difference(DateTime.now()).abs();
                double totalSum = (duration.inMinutes / 60 * spot.zone.hourRate);
                parkingHistoryList.add(ParkingPayment(
                    spot: spot,
                    car: car,
                    totalSum: totalSum,
                    parkingStart: DateTime.parse(startTimestamp),
                    state: PaymentState.ongoing,
                    parkingDuration: duration));
            }
        }
    }
    parkingHistoryList.sortBy<num>((element) => element.state.index);
    return parkingHistoryList;
}

```

Secvența de cod 5.32: "Metoda de preluare a situației plășilor pentru parcare corespunzătoare utilizatorului curent. 2/2"

Metoda de procesare a unei plăti de parcare

Metoda *static Future<void> handlePayment(ParkingPayment parkingHistory)* va introduce o nouă intrare în tabelul de plăti finalizate corespunzătoare obiectului primit ca parametru, ștergând totodată intrările relative parcării plătite din tabela de ocupare. Această metodă este apelată după finalizarea cu succes a plății taxei de parcare.

```
static Future<void> handlePayment(ParkingPayment parkingHistory) async {
    if (parkingHistory.startEntryId == null ||
        parkingHistory.endEntryId == null) {
        throw Exception();
    }

    await pool.execute(
        "INSERT INTO `Payments` (`sensor_id`, `car_id`, `total_sum`, `parking_start`, `parking_end`)
        VALUES (:sensor_id, :car_id, :total_sum, :parking_start, :parking_end)",
        {
            "sensor_id": parkingHistory.spot.sensorId,
            "car_id": parkingHistory.car.carId,
            "total_sum": parkingHistory.totalSum,
            "parking_start": parkingHistory.parkingStart,
            "parking_end": parkingHistory.parkingEnd,
        },
    ).timeout(Constants.sqlTimeoutDuration,
        onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));

    await pool.execute(
        "DELETE FROM `Occupancy` WHERE `entry_id` IN (:start_entry_id, :end_entry_id)",
        {
            "start_entry_id": parkingHistory.startEntryId,
            "end_entry_id": parkingHistory.endEntryId
        }).timeout(Constants.sqlTimeoutDuration,
        onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
}
```

Secvența de cod 5.33: "Metoda de adăugare a unei noi plăți efectuate în MySQL."

Metoda de finalizare a procesului de inițializare a profilului de utilizator

După finalizarea procesului de înregistrare, noul profil de utilizator va avea o intrare corespunzătoare în baza de date ce va conține doar adresa de e-mail a acestuia. Pentru personalizarea acestui cont, la prima autentificare, utilizatorul va completa un formular prin care va oferi diferite date folositoare în cadrul aplicației, după placul său, precum adresa locului de muncă.

În momentul completării acestui formular de bun-venit, metoda *static Future<void> markOnboardingCompleted()* va marca în baza de date faptul că utilizatorul curent a completat procesul de inițializare a profilului.

```
static Future<void> markOnboardingCompleted() async {
    await IsarService.markOnboardingCompleted();
    IsarUser isarUser = IsarService.isarUser;

    await pool.execute(
        "UPDATE `Users` SET `onboarding_complete`=:onboarding_complete WHERE `uid`=:uid LIMIT 1",
        {
            "uid": isarUser.uid,
            "onboarding_complete": isarUser.onboardingComplete
        }).timeout(Constants.sqlTimeoutDuration,
        onTimeout: () => throw TimeoutException(Constants.sqlTimeoutMessage));
}
```

Secvența de cod 5.34: "Metoda prin intermediul se va marca completarea formularului de bun-venit de către utilizatorul curent."

5.2.4 Serviciul de localizare

Metoda de inițializare

Această metodă este apelată la momentul deschiderii aplicației pentru a asigura dreptul de acces al aplicației la modulul GPS al telefonului mobil.

```
static Future<void> init() async {
    bool serviceEnabled;
    LocationPermission permission;

    // Test if location services are enabled.
    serviceEnabled = await Geolocator.isLocationServiceEnabled();
    if (!serviceEnabled) {
        // Location services are not enabled don't continue
        // accessing the position and request users of the
        // App to enable the location services.
        return Future.error('Location services are disabled.');
    }

    permission = await Geolocator.checkPermission();
    if (permission == LocationPermission.denied) {
        permission = await Geolocator.requestPermission();
        if (permission == LocationPermission.denied) {
            // Permissions are denied, next time you could try
            // requesting permissions again (this is also where
            // Android's shouldShowRequestPermissionRationale
            // returned true. According to Android guidelines
            // your App should show an explanatory UI now.
            return Future.error('Location permissions are denied');
        }
    }

    if (permission == LocationPermission.deniedForever) {
        // Permissions are denied forever, handle appropriately.
        return Future.error(
            'Location permissions are permanently denied, we cannot request permissions.');
    }
}
```

Secvența de cod 5.35: "Metoda de inițializare a serviciului de localizare."

Metoda de obținere a unei adrese pe baza coordonatelor

Această metodă va returna adresa corespunzătoare coordonatelor primite ca parametru folosind pachetul geocoding [5].

```
static Future<Address> addressFromLatLng(
    double latitude, double longitude) async {
    try {
        List<geocodingpkg.Placemark> placemarks =
            await geocodingpkg.placemarkFromCoordinates(latitude, longitude);
        Address address = Address(
            placemarks.first.street ?? '',
            placemarks.first.locality ?? '',
            placemarks.first.administrativeArea ?? '',
            placemarks.first.country ?? '');
        return address;
    } catch (e) {
        return Future.error(e.toString());
    }
}
```

Secvența de cod 5.36: "Metoda de obținere a unei adrese pe baza coordonatelor."

5.2.5 Serviciul de căutare a adreselor

Acest serviciu folosește pachetul [dio \[6\]](#) pentru a realiza un HTTP GET request spre Google Maps Places API pentru a obține sugestile de adrese pe baza unei adrese parțiale transmise prin parametrul de tip String *input*.

```
static Future<List<Suggestion>> fetchSuggestions(
    String input, String lang, String sessionToken) async {
  List<Suggestion> suggestions = [];

  final request =
    'https://maps.googleapis.com/maps/api/place/autocomplete/json?input=$input&types=
      address&language=$lang&components=country:ro&key=$apiKey&sessiontoken=
      $sessionToken';
  final response = await client.get(request);

  if (IsarService.isarUser.homeAddress.isNotEmpty) {
    suggestions.add(Suggestion(
      'home',
      Address.fromJson(json.decode(IsarService.isarUser.homeAddress))
        .toString()));
  }

  if (IsarService.isarUser.workAddress.isNotEmpty) {
    suggestions.add(Suggestion(
      'work',
      Address.fromJson(json.decode(IsarService.isarUser.workAddress))
        .toString()));
  }

  if (response.statusCode == 200) {
    final result = response.data;
    if (result['status'] == 'OK') {
      suggestions.addAll(result['predictions']
        .map<Suggestion>((p) => Suggestion(p['place_id'], p['description'])))
        .toList();
    }
    return suggestions;
  } else {
    throw Exception('Failed to fetch suggestion');
  }
}
```

Secvența de cod 5.37: "Metoda de obținere de sugestii pentru adrese."

5.3 Provideri de date

5.3.1 Providerul de date despre senzori

Acest provider implementat cu ajutorul Riverpod [\[7\]](#) va furniza informații despre locurile de parcare în mod reactiv pentru a fi consumate în pagina principală a aplicației.

Prin intermediul serviciului de MySQL, acesta va obține informații despre toate locurile de parcare din jurul poziției transmise ca parametru, ulterior construind setul de markeri folosind funcția asincronă *Future<Set<Marker>> getMarkers(BuildContext context, List<SpotInfo>? spots, LatLng location)* [5.39](#)

```

final spotProvider =
  FutureProvider.family<SpotListData, SpotProviderInput>((ref, input) async {
  final locationData = await Geolocator.getCurrentPosition();
  input.position ??= LatLng(locationData.latitude, locationData.longitude);
  final parkingSpots = input.spots ??
    await SqlService.getParkingSpotsAroundPosition(
      input.position!.latitude, input.position!.longitude, 1);

  if (input.context != null && input.initialized != null) {
    int spotNo = parkingSpots
      .where((element) => element.state != SpotState.occupied)
      .length;
    ScaffoldMessenger.of(input.context!).showSnackBar(SnackBar(
      duration: const Duration(seconds: 5),
      content: ...
    });
  }

  Set<Marker> markers =
    await getMarkers(input.context!, parkingSpots, input.position!);

  return SpotListData(
    location: locationData,
    searchPosition: input.position!,
    spots: parkingSpots,
    markers: markers);
});

```

Secvența de cod 5.38: "Declararea providerului care furnizează informații despre locurile de parcare."

```

Future<Set<Marker>> getMarkers(
  BuildContext context, List<SpotInfo>? spots, LatLng location) async {
  if (spots == null) {
    return {};
  }

  Set<Marker> markers = {};
  for (SpotInfo spot in spots) {
    BitmapDescriptor icon = await getMarkerIcon(spot);
    markers.add(Marker(
      markerId: MarkerId(spot.sensorId.toString()),
      position: LatLng(spot.latitude, spot.longitude), // position of marker
      infoWindow: InfoWindow(
        //popup info
        title: 'Spot#${spot.sensorId}',
        snippet: 'Price: ${spot.zone.hourRate} ${spot.zone.currency}/h',
        onTap: () {
          showModalBottomSheet(
            context: context,
            builder: (context) {
              return Wrap(children: [
                SpotDetails(
                  spot: spot,
                  location: location,
                )
              ]);
            });
        },
        icon: icon));
  }
}

return markers;

```

Secvența de cod 5.39: "Functia de construire a setului de markeri bazat pe o listă de informații despre locuri de parcare."

5.3.2 Providerul de date pentru plătile de parcare

Rolul acestui provider este de a furniza în mod reactiv datele corespunzătoare istoricului de parcare a utilizatorului curent obținute prin intermediul metodei `getUserParkingHistory` [5.32] a serviciului de MySQL [5.2.3].

```
final paymentHistoryProvider =
    FutureProvider.autoDispose<List<ParkingPayment>>((ref) async {
    final List<ParkingPayment> parkingHistory =
        await SqlService.getUserParkingHistory();
    return parkingHistory;
});
```

Secvența de cod 5.40: "Providerul de istoric de plăți de parcare"

5.3.3 Providerul de date pentru pagina de adăugare de senzori

Acest provider va furniza informații despre poziția și adresa actuală a telefonului și a zonelor de parcare din baza de date cu scopul de a fi consumate în pagina de adăugare de noi senzori.

```
final addSensorProvider =
    FutureProvider.autoDispose<AddSensorInfo>((ref) async {
    final List<Zone> zones = await SqlService.getZones() ?? [];
    Position position = await Geolocator.getCurrentPosition();
    LatLng crtPosition = LatLng(position.latitude, position.longitude);
    Address crtAddress = await LocationService.addressFromLatLng(
        crtPosition.latitude, crtPosition.longitude);
    return AddSensorInfo(
        zones: zones, crtPosition: crtPosition, crtAddress: crtAddress);
});
```

```
class AddSensorInfo {
    List<Zone> zones;
    LatLng crtPosition;
    Address crtAddress;
    AddSensorInfo(
        {required this.zones,
        required this.crtPosition,
        required this.crtAddress});
```

Secvența de cod 5.41: "Providerul de date pentru pagina de adăugare senzori"

Capitolul 6

Configurarea plăcii de dezvoltare ESP32-CAM

6.1 Software-ul folosit pentru programarea plăcii de dezvoltare

Codul rulat pe placa de dezvoltare va executa citiri succesive ale valorii măsurate de senzorul de distanță ultrasonic HC-SR04 pentru a determina dacă o mașină este parcată în fața senzorului. În funcție de starea curentă, senzorul va transmite sau nu o cerere HTTP către server prin care va transmite o imagine ce va surprinde fie momentul în care mașina a ocupat locul de parcare, fie momentul în care aceasta la părăsit.

```
while True:
    distance = get_distance()
    if distance <= 0:
        # Possible sensor fault
        time.sleep(OCCUPIED_CHECK_INTERVAL)
        continue
    if occupied:
        # Spot is occupied
        if (distance > 100):
            # Spot got freed
            green_led.value(1)
            red_led.value(0)
            occupied = False
            send_capture(0)
            time.sleep(FREE_CHECK_INTERVAL)
        else:
            interval_count += 1
            # Spot still occupied
            if (interval_count > 2):
                send_capture(1)
                interval_count = 0
            time.sleep(OCCUPIED_CHECK_INTERVAL)
    else:
        # Spot is free
        if (distance <= 100):
            # Spot got occupied
            green_led.value(0)
            red_led.value(1)
            occupied = True
            send_capture(1)
            time.sleep(OCCUPIED_CHECK_INTERVAL)
        else:
            time.sleep(FREE_CHECK_INTERVAL)
```

Secvența de cod 6.1: "Codul de MicroPython folosit pentru programarea plăcii de dezvoltare"

6.2 Serviciul web folosit pentru realizarea comunicării dintre baza de date și placa de dezvoltare

În această secțiune se vor prezenta secvențele de cod utilizate în prelucrarea cererilor HTTP de către server.

6.2.1 Scriptul de PHP pentru prelucrarea cererilor HTTP

Această secțiune de cod reprezintă partea scriptului de PHP executat în momentul procesării cererii HTTP primite de la placa de dezvoltare ce se ocupă de stocarea imaginii în filesystem-ul serverului. Denumirea fișierului corespunzător imaginii va conține data și ora încărcării imaginii. În urma stocării imaginii, se va apela script-ul de python *update_db.py* prin intermediul căruia se vor extrage informații utile din aceasta.

```
<?php
$target_dir = "uploads/";
$datum = mktime(date('H')+0, date('i'), date('s'), date('m'), date('d'), date('Y'));
$target_file = $target_dir . date('Y.m.d_H:i:s_', $datum) . basename($_FILES["imageFile"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));

$sensorID = $_POST["sensorID"];
$occupied = $_POST["occupied"];

// Upload the received file to the path at $target_file
if (move_uploaded_file($_FILES["imageFile"]["tmp_name"], $target_file)) {
echo "The file ". basename( $_FILES["imageFile"]["name"]) . " has been uploaded.\n";
echo "\nStarting image processing for the uploaded file\n";

// Call the python script for image processing and save the response in $message
$message = exec("python3 /var/www/html/python/update_db.py $sensorID $occupied $target_file >&1");
echo "\nOUTPUT:\n";
$output = print_r($message, true);
echo "$output";
$outputLower = strtolower($output);
$error_msg = 'error';
$pos = strpos($outputLower, $error_msg);

if ($pos === false) {
echo "\nNo errors uploading the file";
} else {
echo "\nAn error occurred after uploading the file .";
}
?>
```

Secvența de cod 6.2: "Script-ul de PHP executat în momentul procesării cererii HTTP primite de la placa de dezvoltare"

6.2.2 Scripturile pentru procesarea informațiilor primite de la senzori

update_db.py

Acest script va primi ca parametru identificatorul senzorului, starea acestuia și calea către imaginea încărcată înainte de apelare ce va conține detalii despre mașina ce ocupă momentan locul de parcare.

```
if __name__ == "__main__":
    # parse the program arguments
    arg_parser = create_arg_parser()
    parsed_args = arg_parser.parse_args(sys.argv[1:])

    logging.basicConfig(filename='update_db.log', level=logging.INFO)

    logging.info("Attempting connection to database")

    # connect to the SQL database
    db = mysql.connector.connect(
        ...
    )

    db_cursor = db.cursor()
    logging.info("Successfully connected to database")

    # if the spot state is 1, then the spot is occupied
    if parsed_args.spotState == "1":
        # trigger the spot occupation function
        occupy_spot(parsed_args.sensorId, parsed_args.inputFile, db_cursor)
    if parsed_args.spotState == "0":
        # clear the spot state in the database
        clear_spot(parsed_args.sensorId, db_cursor)

    # commit the database changes to the SQL server
    db.commit()
    db_cursor.close()
    db.close()
    print("Success")
```

Secvența de cod 6.3: "Script-ul de Python pentru interacțiunea cu baza de date."

Interacțiunea efectivă cu baza de date se va realiza prin intermediul funcțiilor *occupy_spot* și *clear_spot*.

```
# Function to update the sensor's occupancy status to 1 (occupied) in the database
def occupy_spot(sensorID, image_path, db_cursor):
    # Boolean flag to check if the car has changed
    inserted = False

    logging.info("Sensor " + sensorID + " detected car entering the spot. Setting spot state to occupied")
    plate = "N/A"

    abs_path = os.path.abspath(image_path)
    if os.path.exists(abs_path):
        logging.info("File exists at path: " + abs_path)
        # Send the image at image_path to be processed by the PlateRecognizer API
        plate = process_image.process(abs_path)
        print("\nPlate: " + plate + "]")
    else:
        logging.error("Invalid file path: File does not exist")

    # Check if car has just changed by getting the current state of the sensor in the DB
    sql = "SELECT PLATE, OCCUPIED FROM parking_spots WHERE SPOT_ID='{}' ORDER BY TIME DESC LIMIT 1"
    logging.info(sql)
    db_cursor.execute(sql)

    existing_plate_list = db_cursor.fetchall()
```

Secvența de cod 6.4: "Funcția occupy_spot 1/2"

```

# if the sensor already has corresponding data in the database
# check the existing state and plate
if len(existing_plate_list) > 0:
    existing_plate = existing_plate_list[0][0]
    occupied = existing_plate_list[0][1]

    # if the sensor is already marked as occupied, check if car is the same
    if occupied == 1:
        print("Existing: " + existing_plate)
        print("New: " + plate)
        print("Occupied: " + str(occupied))

        # if the car has changed, insert the exit for the previous car and
        # occupy the spot for the new car
        if existing_plate != plate:
            print("Plates do not match... Car has changed")
            logging.info("Sensor " + sensorID + " detected the car changed. Updating spot state")
            sql = "INSERT INTO parking_spots(SPOT_ID, PLATE, OCCUPIED) VALUES(%s, %s, %s)"
            val = (sensorID, existing_plate, 0)
            db_cursor.execute(sql, val)
            time.sleep(3)
        else:
            # if the spot is still occupied by the same car, don't insert another 1 in the DB
            inserted = True

# insert 1 if the spot is empty or if the spot is occupied by a different car
if not inserted:
    sql = "INSERT INTO parking_spots(SPOT_ID, PLATE, OCCUPIED) VALUES(%s, %s, %s)"
    val = (sensorID, plate, 1)
    db_cursor.execute(sql, val)

```

Secvența de cod 6.5: "Functia occupy_spot 2/2"

```

# Function to update the sensor's occupancy status to 0 (empty) in the database
def clear_spot(sensorID, db_cursor):
    logging.info("Sensor " + sensorID + " detected car leaving the spot. Setting spot state to empty")

    # Fetch the current state of the spot from the database
    sql = "SELECT PLATE FROM parking_spots WHERE SPOT_ID=" + sensorID + " ORDER BY TIME DESC LIMIT 1"
    logging.info(sql)
    db_cursor.execute(sql)
    plate_list = db_cursor.fetchall()
    if len(plate_list) < 1:
        # if there are no entries corresponding to the sensor with ID: sensorID, exit the function
        # since there is nothing to clear
        return

    # get the current plate corresponding to the sensor in the DB
    plate = plate_list[0][0]

    # set the spot state to 0 (empty) and mark the exit of the current vehicle
    sql = "INSERT INTO parking_spots(SPOT_ID, PLATE, OCCUPIED) VALUES(%s, %s, %s)"
    val = (sensorID, plate, 0)
    db_cursor.execute(sql, val)
    logging.info("Cleared spot " + sensorID + " from car with plate [" + plate.upper() + "]")

```

Secvența de cod 6.6: "Functia clear_spot"

process_image.py

Partea din script-ul process_image.py prezintă mai jos reprezintă modul în care se va construi și trimite cererea HTTP către API-ul de recunoaștere de plăcuțe de înmatriculare. Răspunsul primit de la acest API va fi ulterior prelucrat în funcția *extract_data_from_response* prin care se vor extrage informațiile utile din JSON-ul primit.

```
def parse_image(imagePath):
    plate = ""

    # Authorization token for the Plate Recognizer API POST request
    headers = {
        'Authorization': '*****',
    }

    files = {
        'upload': (imagePath, open(imagePath, 'rb')),
        'regions': (None, 'us-ca'),
    }

    # Send image to API
    response = requests.post('https://api.platerecognizer.com/v1/plate-reader', headers=headers, files=files)
    logging.info("Received response from Plate Recognizer:")
    logging.info(response)

    # Extract the license plate from the JSON returned by the API
    plate = extract_data_from_response(response)

    return plate

# Function that checks if the image at path contains a license plate
def process(path):
    logging.info("Starting new image processing for image at path: " + path + " ")

    # send the image to the API
    plate = parse_image(path)

    if plate != "":
        logging.info("Found plate: " + plate + "\n")
        return plate

    # if no plate is found or if the plates found are invalid
    # return 'N/A' as the image processing response
    logging.info("No plate was found\n")
    return "N/A"
```

Secvența de cod 6.7: "Script-ul de Python pentru procesarea imaginii primite de la placa de dezvoltare."

Capitolul 7

Utilizarea aplicației

7.1 Autentificarea

Pentru a folosi aplicația, este necesar pasul de autentificare. În momentul deschiderii aplicației, dacă un utilizator nu este conectat prin intermediul contului său, va fi redirecționat pe pagina de autentificare [7.1](#).

În această pagină este prezent un formular de autentificare, ce conține două câmpuri corespunzătoare adresei de e-mail și a parolei contului de utilizator. După completarea formularului, utilizatorul se va putea autentifica prin intermediul butonului din partea de jos a formularului ce va declanșa procesul de autentificare și autorizare.

În cadrul procesului de autentificare, se va verifica existența unui profil de utilizator asociat adresei de e-mail introduse, precum și validitatea credențialelor de autentificare. Totodată, va fi obținut și nivelul de autorizare al utilizatorului pentru a-i se putea oferi acces la anumite funcționalități ce necesită un nivel elevat de permisiuni.

Dacă datele furnizate de utilizator sunt gresite, acesta va fi informat de eșecul procesului de autentificare printr-un mesaj situat în partea de jos a paginii.

În cazul în care utilizatorul nu are încă un cont creat, va fi nevoie să apese pe textul din partea de jos a paginii, "Don't have an account? Sign-up" pentru a își crea un cont nou.

7.2 Înregistrarea în aplicație

În cazul în care utilizatorul dorește să își creeze un nou cont apăsând pe textul din partea de jos a ecranului de autentificare [7.1](#), acesta va fi redirectionat pe pagina de înregistrare, unde va trebui să completeze un formular care să conțină adresa acestuia de e-mail și o parolă de minimum 8 caractere. Parola trebuie recompletată pentru verificare.

Dacă utilizatorul introduce o adresă de e-mail care este deja folosită, acesta va primi un mesaj de atenționare [7.3](#) în partea de jos a paginii în momentul în care va încerca să creeze contul.

Totodată, dacă adresa de e-mail introdusă de utilizator nu este o adresă de e-mail validă, acesta va primi un mesaj de atenționare [7.4](#).

Utilizatorul se poate întoarce la pagina de autentificare prin apăsarea textului „Already have an account?” din partea de jos a paginii.



Smart Park

E-mail

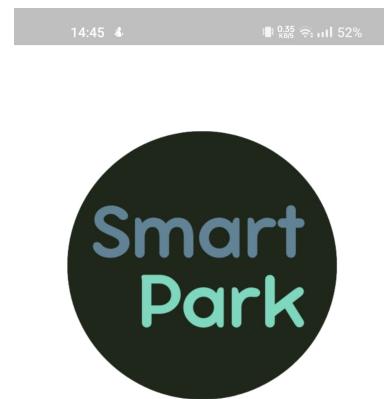
Password

Confirm password

Create account

Already have an account?

Figura 7.1: Formularul de autentificare în aplicație.



Smart Park

E-mail
test@email.com

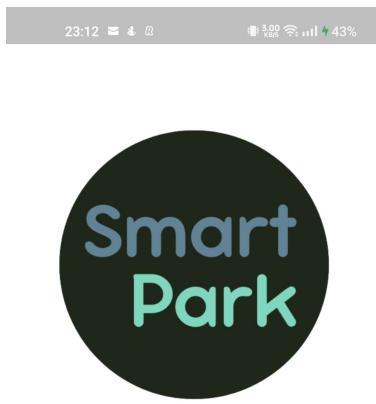
Password
.....

Confirm password
.....

Create account

Already have an account?

Figura 7.2: Formularul de înregistrare în aplicație.



Smart Park

E-mail
stefan.chivu1@yahoo.com

Password
.....

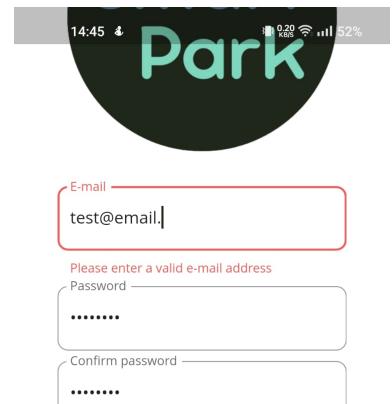
Confirm password
.....

Create account

Already have an account?

This e-mail is already in use!

Figura 7.3: Validarea formularului de înregistrare pentru unicitatea adreselor de e-mail.



Smart Park

E-mail
test@email.|

Please enter a valid e-mail address

Password
.....

Confirm password
.....

Create account

Keyboard view

Figura 7.4: Validarea formularului de înregistrare pentru validitatea adreselor de e-mail introduse.

7.3 Formularul de bun-venit

După înregistrare, utilizatorul va fi trimis spre un formular de bun-venit prin intermediului va putea să ofere informații suplimentare, respectiv numărul de înmatriculare **7.5** al uneia dintre mașinile sale, numele său **7.7** sau adresa de acasă și de la locul de muncă **7.6**. Acest formular va fi disponibil doar utilizatorilor care nu l-au completat după înregistrare.

14:46 4G 0:22 52%

Please provide the license plate of the car you are going to use to navigate:

License plate (required)

Electric vehicle

>

14:47 4G 4:00 52%

Work address (optional)

Home address (optional)

✓

Figura 7.5: Formularul pentru numărul de înmatriculare.

Figura 7.6: Alegerea adreselor de acasă și de la locul de muncă.

14:46 4G 0:29 52%

First Name (optional)

Last Name (optional)

>

Figura 7.7: Setarea numelui și prenumelui.

7.4 Pagina principală

În urma introducerii corecte a datelor de autentificare, utilizatorul este trimis spre pagina principală a aplicației **7.8**, unde poate vedea pe hartă locația sa, precum și locația locurilor de parcare din apropiere și starea acestora.

Dacă utilizatorul va interacționa cu harta prin mișcarea poziției afișate pe aceasta, se va afișa un buton de reîmpropătare **7.9** prin care se vor putea obține detalii despre senzorii de la noua poziție indicată de hartă. Apăsarea acestuia va reîmprospăta **7.10** starea markerilor de pe hartă.

Dacă utilizatorul apasă pe unul dintre markerii aflați pe hartă, va putea obține informații suplimentare **7.11** despre locul de parcare căruia îi corespunde acesta. Totodată, îi se va pune la dispoziție un buton de navigare, prin intermediul căruia poate naviga spre pagina de călătorie către acel loc.

Figura 7.8: Pagina principală a aplicației.

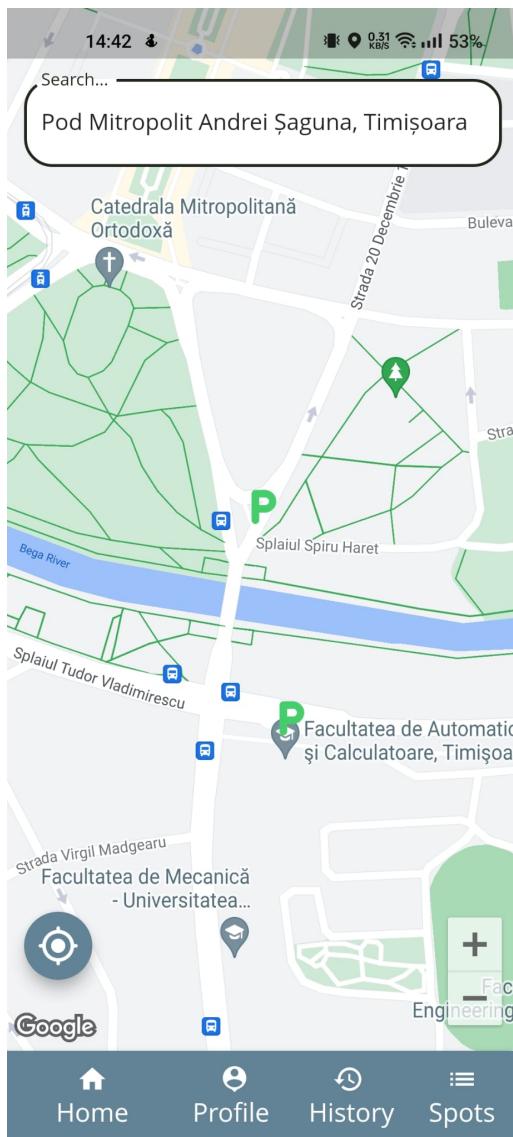
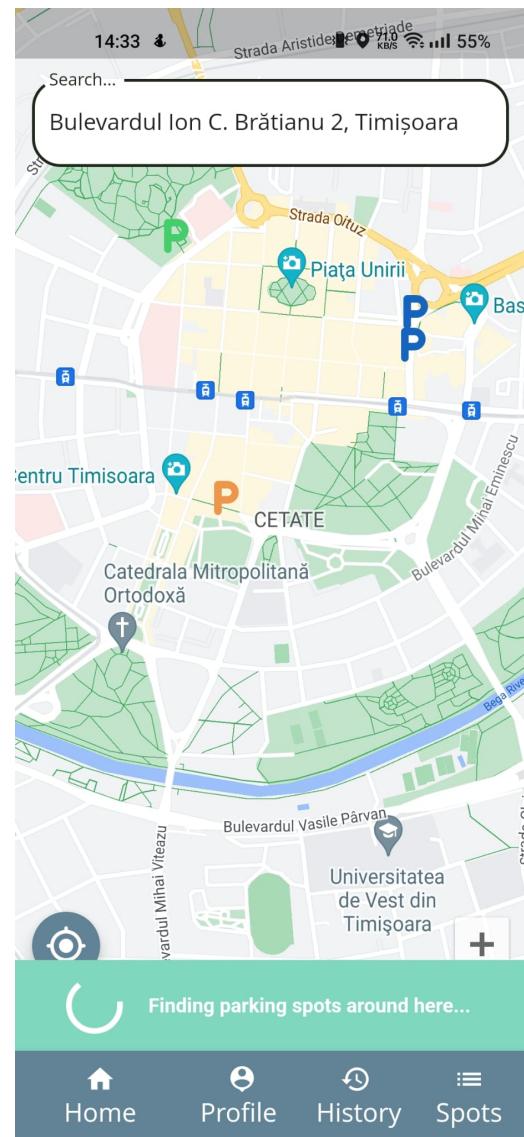


Figura 7.9: Indicatorul de reîmprospătare a indicatorilor pentru locuri de parcare.



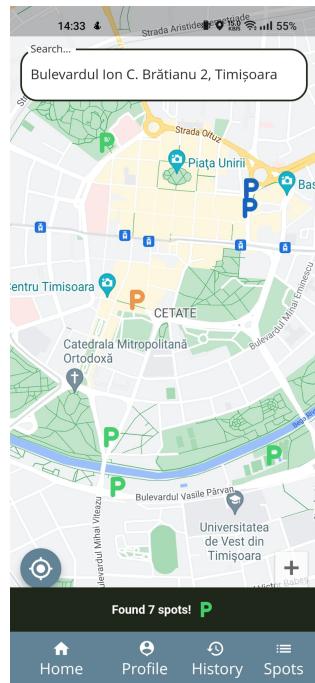


Figura 7.10: Actualizarea locurilor de parcare din zona cuprinsă de hartă.

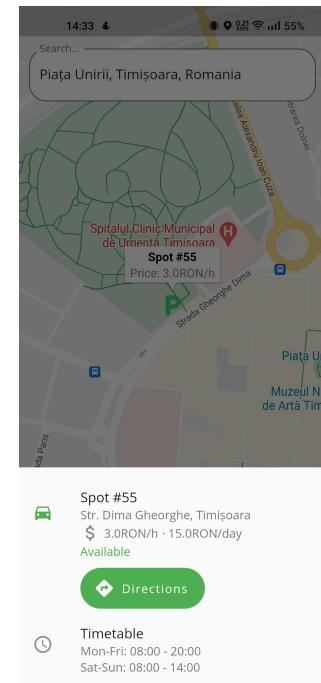


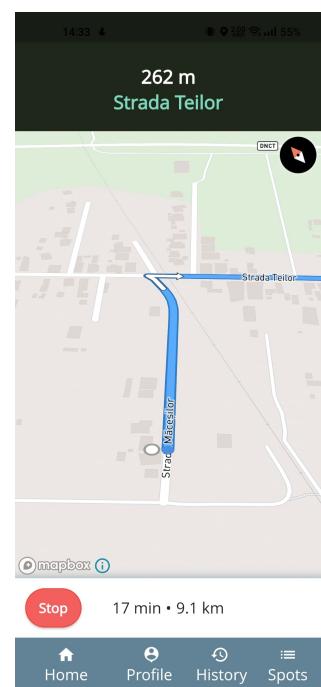
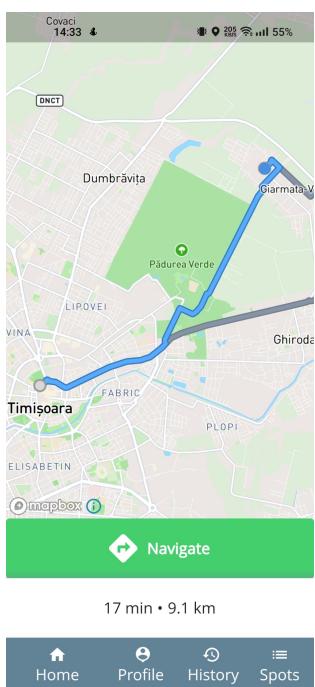
Figura 7.11: Informații despre locul de parcare.

7.5 Pagina de navigare

În momentul accesării acestei pagini, se va calcula ruta către locul de parcare ales și se va prezenta pe hartă [7.12](#). Aici vor fi disponibile și alte informații despre traseu, precum durata estimativă a acestuia și distanța totală de parcurs.

Figura 7.12: Ruta către locul de parcare ales.

Figura 7.13: Afisarea indicațiilor de călătorie detaliate către locul ales.



Pentru a declansa navigarea, utilizatorul va apăsa pe butonul de start din această pagină. Acest lucru va modifica aspectul paginii de navigare 7.13, iar indicații de călătorie mai detaliate vor fi disponibile în partea de sus a paginii.

În cazul în care locul de parcare selectat inițial este ocupat în timpul călătoriei spre acesta, se va încerca obținerea indicațiilor spre un alt loc de parcare din apropiere. Utilizatorul va fi întrebat prin intermediul unei alerte de acest fapt pentru a-și exprima acordul modificării destinației 7.14.

Dacă utilizatorul acceptă noua destinație, ruta se va modifica implicit, având ca destinație noul loc de parcare 7.15.

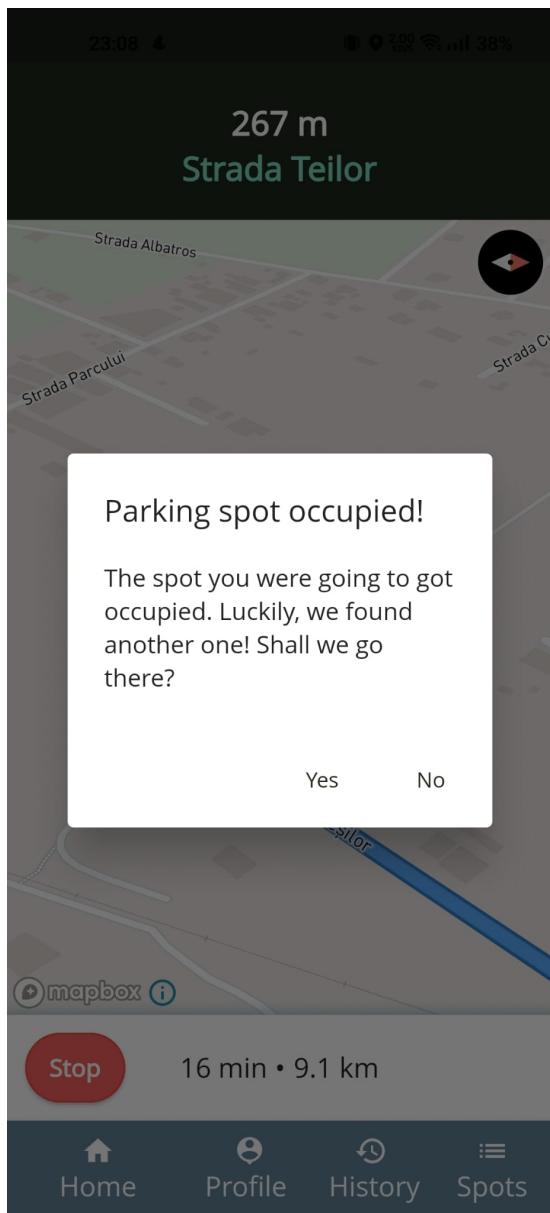


Figura 7.14: Avertizare asupra faptului că locul de parcare dorit a fost ocupat.

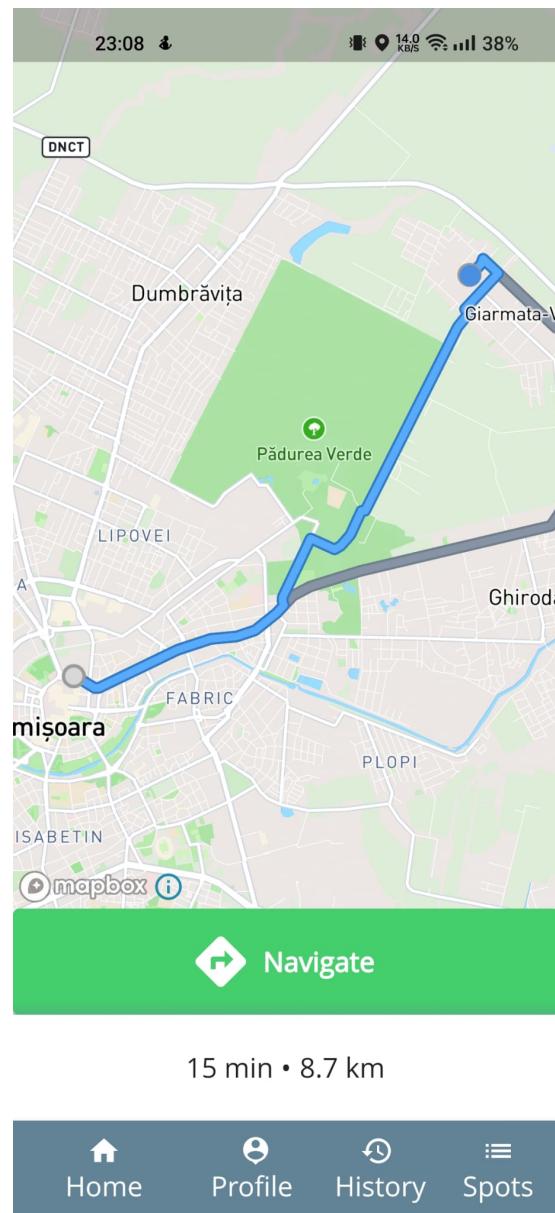


Figura 7.15: Ruta către un nou loc de parcare găsit în apropiere.

7.6 Pagina de profil a utilizatorului

Pentru a modifica datele corespunzătoare profilului său, utilizatorului îi este pusă la dispoziție pagina de profil **7.18**. În cadrul acesteia, se vor putea modifica informații precum numele și prenumele utilizatorului, adresa corespunzătoare domiciliului și locului de muncă, precum și adăugarea numerelor de înmatriculare aferente mașinilor deținute de acesta.

7.6.1 Adăugarea adreselor corespunzătoare domiciliului și locului de muncă

Utilizatorul va putea vizualiza și modifica prin intermediul paginii de profil adresele sale salvate. Prin setarea acestora, vor putea fi accesate în pagina principală în momentul căutării adreselor **7.21**.

7.6.2 Adăugarea informațiilor despre mașinile deținute

În cadrul paginii de profil, utilizatorul va putea adăuga numerele de înmatriculare ale mașinilor pe care le deține. Astfel, în pagina de istoric al plășilor de parcare, vor fi afișate detalii despre plășile corespunzătoare mașinilor adăugate în acest mod.

Figura 7.16: Formularul de adăugare a unei noi mașini de către utilizator.

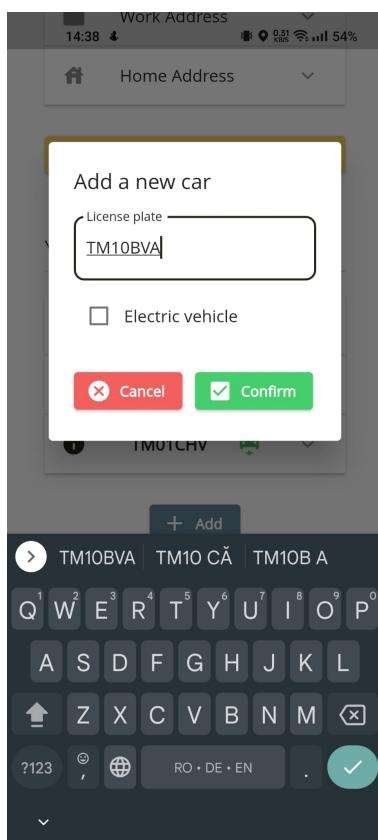


Figura 7.17: Lista de mașini deținute de utilizator.

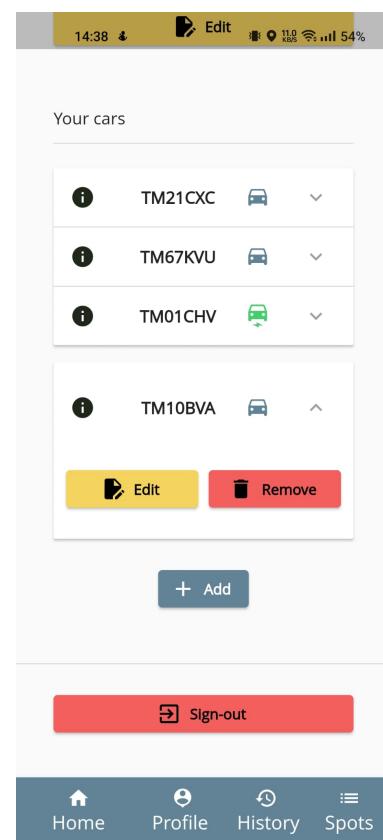


Figura 7.18: Pagina de profil a utilizatorului.

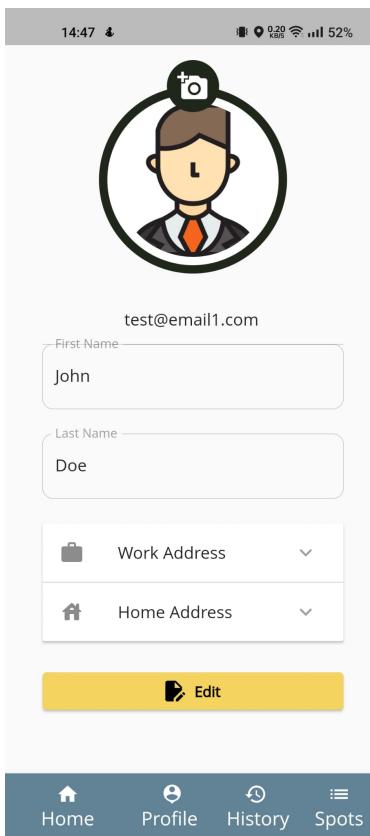


Figura 7.20: Adresele setate în pagina de profil, disponibile în formularul de căutare de adrese.

Figura 7.19: Detaliile despre adresa de domiciliu din pagina de profil.

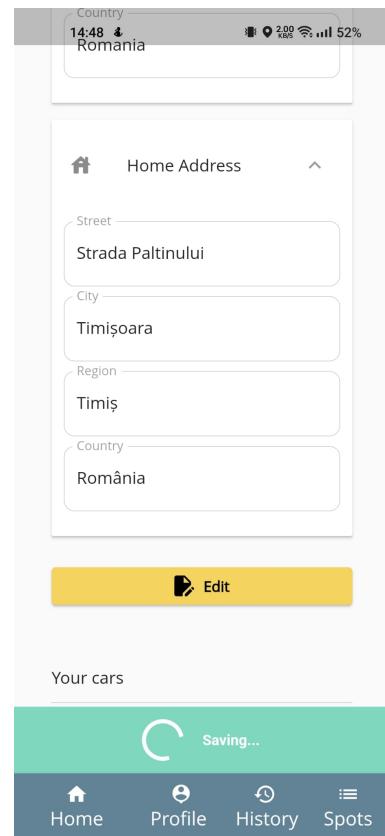
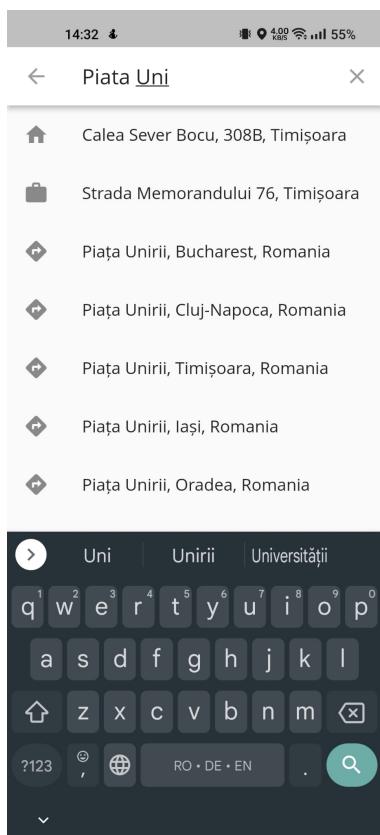
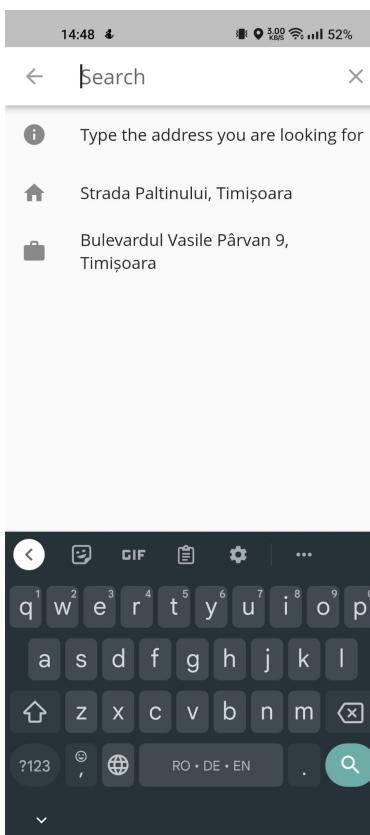


Figura 7.21: Formularul de căutare a adreselor.



7.7 Adăugarea unui nou loc de parcare

Administratorilor aplicației le este pus la dispoziție un buton de adăugare de noi senzori pe pagina de profil 7.25. Prin apăsarea acestuia, vor fi redirectionați către pagina de adăugare de senzori 7.22, unde este necesară alegerea poziției pe hartă a noului senzor, precum și atribuirea unei zone tarifare și identificator unic. Aceștia vor putea selecta prin intermediul unui checkbox dacă la locul de parcare respectiv există și o stație de încărcare electrică.

Prin apăsarea butonului de adăugare a senzorului, va fi redirectionat către pagina principală, unde se va putea regăsi markerul corespunzător noului senzor 7.23.

Figura 7.22: Pagina administrativă de adăugare a senzorilor.

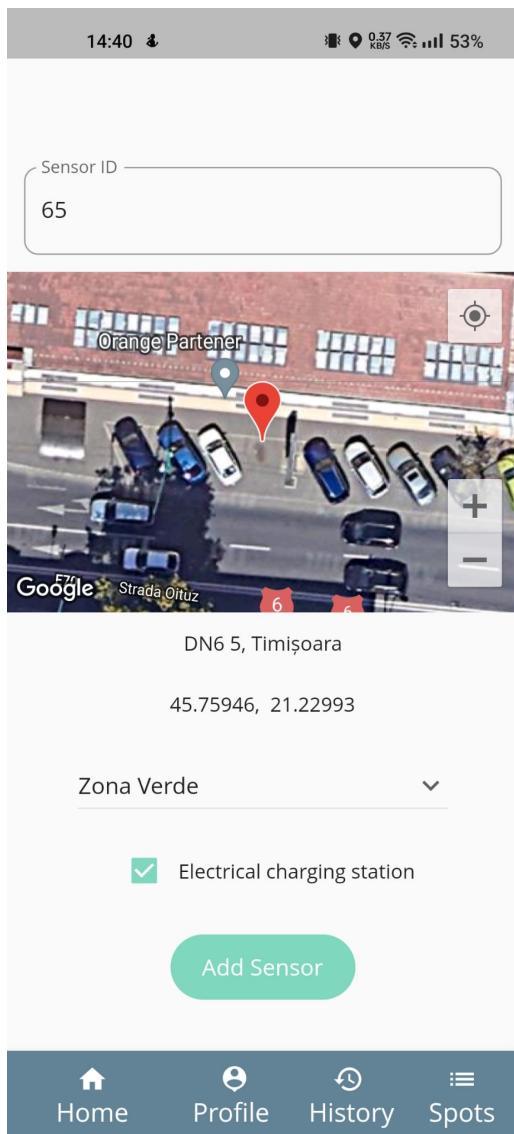
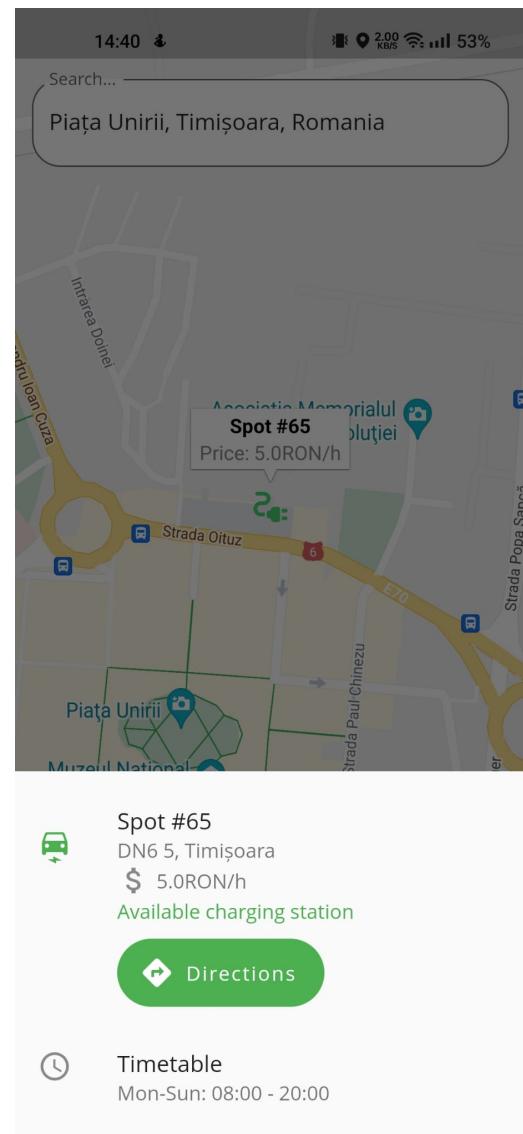


Figura 7.23: Locul de parcare adăugat.



7.8 Deconectarea de la aplicație

Prin apăsarea butonului de deconectare aflat în partea de jos a paginii de profil [7.24](#), utilizatorul va părăsi sesiunea de autentificare și va fi redirecționat către pagina de autentificare.

Figura 7.24: Butonul de deconectare din aplicație.

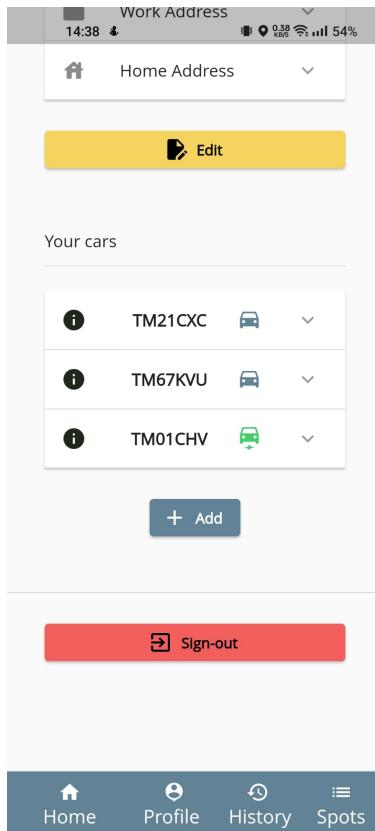
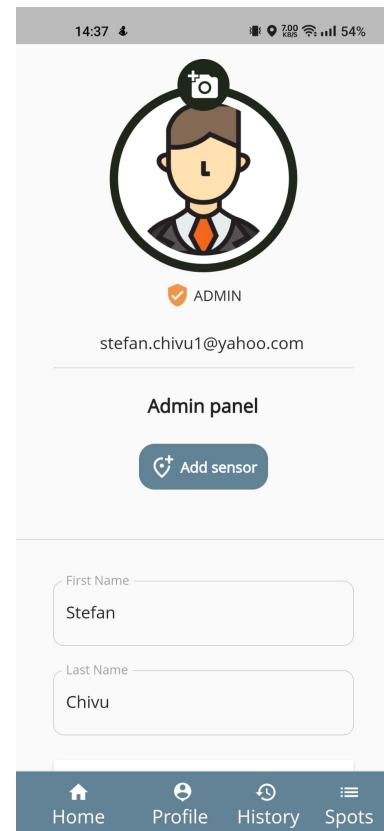


Figura 7.25: Secțiunea din pagina de profili disponibilă exclusiv administratorilor.



7.9 Pagina de vizualizare în listă a locurilor de parcare disponibile

Vizualizarea locurilor disponibile din jurul utilizatorului se poate realiza și prin intermediul paginii de vizualizare în listă a locurilor de parcare [7.26](#). Aceasta reprezintă o alternativă a vizualizării prin intermediul hărții.

7.10 Pagina de vizualizare a stării plăștilor de parcare

Utilizatorul va putea vizualiza starea plăștilor de parcare aferente mașinilor sale prin intermediul paginii de istoric al parcărilor [7.27](#).

În această pagină, lista va conține trei tipuri de parcare, respectiv:

- Parcări în curs de desfășurare - marcate cu portocaliu
- Parcări cu plata restantă - marcate cu roșu
- Parcări cu plata finalizată - marcate cu verde

7.10.1 Pagina de plată a tarifului de parcare

Prin apăsarea butonului de plată al tarifelor de plată restante de pe pagina de istoric al parcărilor, utilizatorul va fi trimis pe pagina de procesare a plăților **7.28**. Aici, va putea opta pentru plata cu Google Pay sau Apple Pay, respectiv cu un card bancar.

Figura 7.26: Pagina de vizualizare în listă a locurilor de parcare.

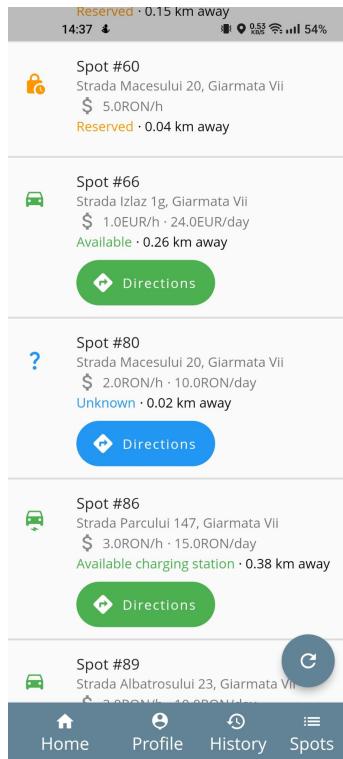


Figura 7.27: Pagina de vizualizare a istoricului plăților de parcare.

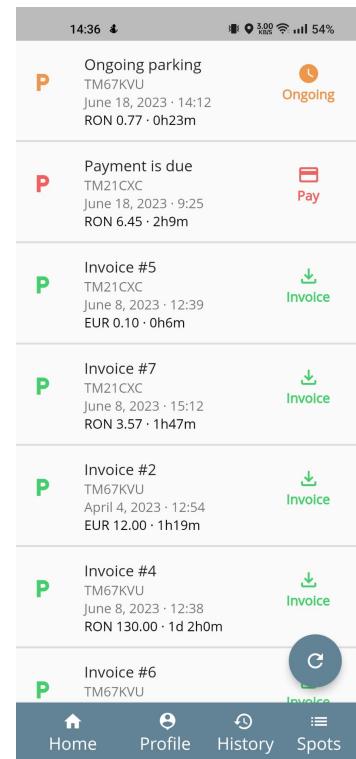
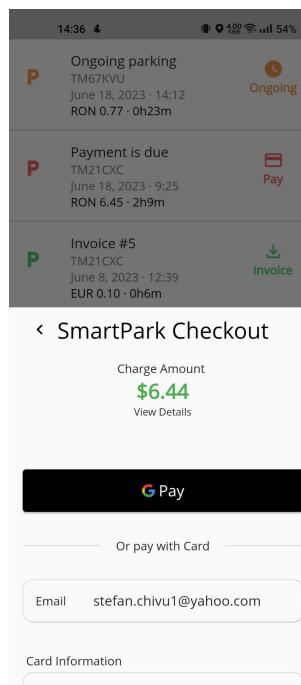


Figura 7.28: Pagina de plată a tarifului de parcare.



Capitolul 8

Concluzii

8.1 Note generale

Sistemul SmartPark își propune să fie o unealtă ce să ajute șoferii în căutarea unui loc de parcare pentru a reduce valorile de trafic și pentru a spori nivelul de siguranță al șoferilor. În urma implementării unui astfel de sistem pe scară largă, se poate obține o experiență mult mai plăcută de transport datorită eliminării parțiale a problemei găsirii unui loc de parcare. Acest sistem ar trebui însă integrat cu alte sisteme pentru orașe inteligente, precum un sistem de monitorizare a transportului public, pentru a oferi utilizatorilor cea mai bună variantă de transport spre zona dorită de aceștia.

8.2 Discuție

În urma implementării sistemului, s-au descoperit anumite dezavantaje ale implementării curente, respectiv costul ridicat de implementare a acesteia (un senzor per loc de parcare), precum și riscurile precum expunerea la vandalism sau la diverse factori dăunători din mediu. Acest mod de implementare al sistemului ar presupune un dezavantaj pentru locurile publice de parcare, datorită nevoii de pregătire a infrastructurii de alimentare la rețea a senzorilor, ce implică ocuparea unei porțiuni din trotuarele aflate în preajma locurilor de parcare vizate. Acest lucru s-ar putea rezolva prin alimentarea senzorilor utilizând surse alternative, precum energia solară, însă pot apărea probleme în cazul zonelor umbrite. Totodată, senzorii individuali amplasati în fața fiecărui loc de parcare sunt expuși factorilor de mediu, precum zăpada ce ar putea să îi acopere, ducând astfel la probleme în timpul utilizării. Nu în ultimul rând, senzorii sunt expuși acelor de vandalism, precum și accidentelor neintenționate, precum lovirea acestora cu botul mașinii în timpul parcării. Acești factori prezintă un dezavantaj pentru locurile de parcare publice, aflate pe stradă, însă pot fi atenuați în cazul montării echipamentului în cadrul unei parcări private, unde rețeaua de alimentare și de conectivitate poate fi introdusă încă de la început, fără a fi nevoie de costuri de implementare mult mai mari.

8.3 Direcții de dezvoltare

O principală direcție de dezvoltare viitoare constă în modificarea modului de preluare a informațiilor din teren. Solutia actuală, rețeaua de senzori amplasati pe fiecare loc de parcare, prezintă dificultăți de implementare datorită costului ridicat, numărului mare de componente hardware expuse daunelor și vandalismului, precum și dificultatea implementării

infrastructurii necesare alimentării și conectării unui număr atât de ridicat de senzori. Astfel, în viitor, informațiile ar putea fi colectate prin intermediul unor opritoare smart. Acestea ar putea fi amplasate la intrarea și ieșirea anumitor zone prestabile, care să contorizeze numărul de mașini care se află la un moment dat în zona respectivă. În acest mod, ar fi posibilă prelucrarea informațiilor despre volumul de trafic. Totodată, s-ar putea realiza estimări și în legătură cu gradul de ocupare al locurilor de parcare.

În cazul soluției actuale, pentru a oferi o experiență cât mai facilă în folosirea aplicației, un pas următor în dezvoltarea sistemului constă în prelucrarea informațiilor despre starea locurilor de parcare cu scopul de a extrage o serie de metri și indicatori precum rata timpului de ocupare a unui loc sau durata medie între eliberare și reocupare. Aceștia vor putea îmbunătăți modul în care aplicația recomandă șoferilor anumite locuri de parcare din diferite zone. Acest proces ar putea fi îmbunătățit prin crearea unui model de inteligență artificială supus unui proces de învățare continuă pe baza informațiilor preluate din teren.

Lista secvențelor de cod

5.1 "Modelul informațional folosit pentru utilizatori."	24
5.2 "Modelul informațional folosit pentru adrese."	24
5.3 "Modelul informațional folosit pentru mașinile utilizatorilor."	25
5.4 "Modelul informațional folosit pentru senzori."	25
5.5 "Modelul informațional folosit pentru plăți."	26
5.6 "Modelul informațional folosit pentru zone tarifare."	27
5.7 "Corpul metodei de autentificare."	28
5.8 "Corpul metodei de înregistrare."	29
5.9 "Corpul metodei de deconectare."	29
5.10 "Metoda de inițializare a utilizatorului."	30
5.11 "Metoda de creare a unui utilizator pe baza unui user oferit de Firestore."	30
5.12 "Metoda de setare locală a utilizatorului curent"	30
5.13 "Metoda de actualizare a datelor locale ale utilizatorului curent"	30
5.14 "Metoda de stergere a datelor locale despre utilizatorul curent"	31
5.15 "Metoda de setare locală a mașinilor utilizatorului curent"	31
5.16 "Metoda de adăugare a unei noi mașini în lista de mașini a utilizatorului curent"	31
5.17 "Metoda de stergere a unei mașini din lista de mașini ale utilizatorului curent"	31
5.18 "Metoda de marcare locală a completării formularului de bun-venit"	31
5.19 "Metoda de adăugare a unui utilizator în baza de date."	32
5.20 "Metoda de preluare a datelor corespunzătoare unui profil de utilizator."	32
5.21 "Metoda de actualizare a datelor corespunzătoare unui profil de utilizator."	33
5.22 "Metoda de preluare a mașinilor unui utilizator."	33
5.23 "Metoda de preluare a datelor corespunzătoare unui profil de utilizator din baza de date."	34
5.24 "Metoda de adăugare a unei mașini noi în MySQL."	34
5.25 "Metoda de adăugare a unui nou senzor."	35
5.26 "Metoda de preluare a datelor despre toți senzorii din jurul unei poziții."	36
5.27 "Metoda de obținere a celui mai apropiat senzor relativ la o poziție geografică."	37
5.28 "Metoda de obținere a stării de ocupare a unui loc de parcare."	38
5.29 "Metoda de rezervare a unui loc de parcare."	38
5.30 "Metoda de anulare a rezervărilor unui loc de parcare."	38
5.31 "Metoda de preluare a situației plășilor pentru parcare corespunzătoare utilizatorului curent. 1/2"	39
5.32 "Metoda de preluare a situației plășilor pentru parcare corespunzătoare utilizatorului curent. 2/2"	40
5.33 "Metoda de adăugare a unei noi plăș efectuate în MySQL."	41
5.34 "Metoda prin intermediul se va marca completarea formularului de bun-venit de către utilizatorul curent."	41
5.35 "Metoda de inițializare a serviciului de localizare."	42

5.36 "Metoda de obținere a unei adrese pe baza coordonatelor."	42
5.37 "Metoda de obținere de sugestii pentru adrese."	43
5.38 "Declararea providerului care furnizează informații despre locurile de parcare."	44
5.39 "Functia de construire a setului de markeri bazat pe o listă de informații despre locuri de parcare."	44
5.40 "Providerul de istoric de plăti de parcare"	45
5.41 "Providerul de date pentru pagina de adăugare senzori"	45
6.1 "Codul de MicroPython folosit pentru programarea plăcii de dezvoltare"	47
6.2 "Script-ul de PHP executat în momentul procesării cererii HTTP primite de la placa de dezvoltare"	48
6.3 "Script-ul de Python pentru interacțiunea cu baza de date."	49
6.4 "Functia occupy_spot 1/2"	49
6.5 "Functia occupy_spot 2/2"	50
6.6 "Functia clear_spot"	50
6.7 "Script-ul de Python pentru procesarea imaginii primite de la placa de dezvoltare."	51

Lista figurilor

Capitolul 1

1.1	Diagrama generală a sistemului. Aceasta prezintă interacțiunile dintre anumite componente. Primul flux de informații constă în cel realizat de senzorul de distanță și placa de dezvoltare ESP32-CAM spre baza de date MySQL aflată pe o mașină virtuală. Acesta constă în transmiterea imaginii corespunzătoare ocupării/eliberarii locului de parcare. Al doilea flux de informații este cel prin care utilizatorul, folosind aplicația mobilă, va putea vizualiza situația locurilor de parcare din teren, pentru a putea ulterior să navigheze spre acestea.	6
-----	--	---

Capitolul 3

3.1	Imagine corespunzătoare plăcii de dezvoltare ESP32-CAM Ai Thinker. Sursa: https://3dstar.ro/modul-esp32-cam	15
3.2	Informații tehnice despre placa de dezvoltare ESP32-CAM oferită de AiThinker. Sursa: http://www.ai-thinker.com/pro_view-24.html	15
3.3	Imagine corespunzătoare plăcii de dezvoltare ESP32-CAM Freenove WROVER Kit. Sursa: https://github.com/Freenove/Freenove_ESP32_WROVER_Board	16
3.4	Imagine corespunzătoare modulului de cameră OV2640. Sursa: https://www.arducam.com/ov2640/	16
3.5	Imagine corespunzătoare senzorului ultrasonic de distanță HC-SR04. Sursa: https://www.optimusdigital.ro/ro/senzori-senzori-ultrasonici/9-senzor-ultrasonic-hc-sr04-.html	17

Capitolul 4

4.1	Diagrama UML a cazurilor de utilizare: actorii și principalele operații puse la dispoziția acestora.	20
4.2	Diagrama Entitate-Relație a bazei de date. Tabelele și relațiile dintre acestea.	21
4.3	Schema circuitului.	22

Capitolul 7

7.1	Formularul de autentificare în aplicație.	54
7.2	Formularul de înregistrare în aplicație.	54
7.3	Validarea formularului de înregistrare pentru unicitatea adreselor de e-mail.	54
7.4	Validarea formularului de înregistrare pentru validitatea adreselor de e-mail introduse.	54
7.5	Formularul pentru numărul de înmatriculare.	55
7.6	Alegerea adreselor de acasă și de la locul de muncă.	55
7.7	Setarea numelui și prenumelui.	55
7.8	Pagina principală a aplicației.	56

7.9	Indicatorul de reîmprospătare a indicatorilor pentru locuri de parcare.	56
7.10	Actualizarea locurilor de parcare din zona cuprinsă de hartă.	57
7.11	Informații despre locul de parcare.	57
7.12	Ruta către locul de parcare ales.	57
7.13	Afișarea indicațiilor de călătorie detaliate către locul ales.	57
7.14	Avertizare asupra faptului că locul de parcare dorit a fost ocupat.	58
7.15	Ruta către un nou loc de parcare găsit în apropiere.	58
7.16	Formularul de adăugare a unei noi mașini de către utilizator.	59
7.17	Lista de mașini deținute de utilizator.	59
7.18	Pagina de profil a utilizatorului.	60
7.19	Detaliile despre adresa de domiciliu din pagina de profil.	60
7.20	Adresele setate în pagina de profil, disponibile în formularul de căutare de adrese.	60
7.21	Formularul de căutare a adreselor.	60
7.22	Pagina administrativă de adăugare a senzorilor.	61
7.23	Locul de parcare adăugat.	61
7.24	Butonul de deconectare din aplicație.	62
7.25	Secțiunea din pagina de profili disponibilă exclusiv administratorilor.	62
7.26	Pagina de vizualizare în listă a locurilor de parcare.	63
7.27	Pagina de vizualizare a istoricului plăților de parcare.	63
7.28	Pagina de plată a tarifului de parcare.	63

Lista tabelelor

2.1	Analiză comparativă între unele dintre cele mai apreciate aplicații mobile pentru sistemul de operare Android (EasyPark - find & pay parking și RingGo Parking) și "SmartPark"; funcționalitățile analizate regăsite într-o aplicație sunt marcate cu ✓, iar cele absente cu X	9
-----	--	---

Bibliografie

- [1] Teresa Ponnambalam and Birsen Donmez. *Searching for Street Parking: Effects on Driver Vehicle Control, Workload, Physiology, and Glances.* <https://doi.org/10.3389/fpsyg.2020.574262>, 2020. [Online; Accesat 18.12.2022].
- [2] *Documentație Flutter.* <https://flutter.dev/>, 2023. [Online; Accesat: 11.6.2023].
- [3] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley Longman Ltd., GBR, 1998.
- [4] *Documentație Isar.* <https://isar.dev/>, 2023. [Online; Accesat: 11.6.2023].
- [5] *Geocoding Flutter package.* <https://pub.dev/documentation/geocoding/>, 2023. [Online; Accesat: 9.1.2023].
- [6] *Dio Flutter package.* <https://pub.dev/packages/dio>, 2023. [Online; Accesat: 12.6.2023].
- [7] *Documentație Riverpod.* <https://riverpod.dev/>, 2023. [Online; Accesat: 9.1.2023].

**DECLARAȚIE DE AUTENTICITATE A
LUCRĂRII DE FINALIZARE A STUDIILOR***

Subsemnatul CHIVU STEFAN

legitimat cu C.İ. TZ nr. 578 366,

CNP 5010404 350011

autorul lucrării SMART PARK – SISTEM PENTRU MONITORIZAREA
LOCURILOR DE PARCARE PUBLICE

elaborată în vederea susținerii examenului de finalizare a studiilor de
LICENȚĂ din cadrul Universității
Politehnica Timișoara, sesiunea IUNIE 2023 a anului universitar
2022 - 2023, coordonator PROF. DR. HABIL. ING. MIHAI VARESCU-MILOSAV, luând în
considerare art. 34 din *Regulamentul privind organizarea și desfășurarea examenelor de
licență/diplomă și disertație*, aprobat prin HS nr. 109/14.05.2020 și cunoscând faptul că în
cazul constatării ulterioare a unor declarații false, voi suporta sancțiunea administrativă
prevăzută de art. 146 din Legea nr. 1/2011 – legea educației naționale și anume anularea
diplomei de studii, declar pe proprie răspundere, că:

- această lucrare este rezultatul propriei activități intelectuale;
- lucrarea nu conține texte, date sau elemente de grafică din alte lucrări sau din alte surse fără ca acestea să nu fie citate, inclusiv situația în care sursa o reprezintă o altă lucrare/alte lucrări ale subsemnatului;
- sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor;
- această lucrare nu a mai fost prezentată în fața unei alte comisii de examen/prezentată public/publicată de licență/diplomă/disertație;
- În elaborarea lucrării am utilizat instrumente specifice inteligenței artificiale (IA) și anume
(denumirea) (sursa), pe care le am citat în conținutul lucrării/nu am utilizat
instrumente specifice inteligenței artificiale (IA)¹.

Declar că sunt de acord ca lucrarea să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Timișoara,

Data

24. 06. 2023

Semnătura

CHS

*Declarația se completează de student, se semnează olograf de acesta și se inserează în lucrarea de finalizare a studiilor, la sfârșitul lucrării, ca parte integrantă.

¹ Se va păstra una dintre variante: 1 - s-a utilizat IA și se menționează sursa 2 – nu s-a utilizat IA