KAUNO KOLEGIJA

TECHNOLOGIJŲ FAKULTETAS

INFORMATIKOS IR MEDIJŲ TECHONOLOGIJŲ KATEDRA

Nerijus Kvedaravičius

IŠMANIOSIOS STOVĖJIMO AIKŠTELĖS PROJEKTAS

Baigiamasis darbas

Kibernetinių sistemų ir saugos studijų programos
valstybinis kodas 6531BX024
Informatikos inžinerijos studijų krypties

Vadovas Gintaras Butkus

Kaunas, 2024

TURINYS

LENTELIŲ IR PAVEIKSLŲ SĄRAŠAS

LENTELĖS

PAVEIKSLAI

SĄVOKŲ SĄRAŠAS

Sąvoka	Aprašymas	Nuoroda į šaltinį
Maršrutizatoriai ir Jungikliai	Maršrutizatoriai yra skirti duomenų perdavimui tarp skirtingų tinklų segmentų, o jungikliai – duomenų perdavimui ir paskirstymui vidinėje tinkle.	Kazlauskas, K. (2019)
Tinklo Saugumo Užtvaros	Tinklo saugumo užtvaros apsaugo tinklus nuo neautorizuoto prieigos, užtikrinant duomenų saugumą.	Jankauskas, L. (2020)
Bevielio Ryšio Prieigos Taškai	Bevielio ryšio prieigos taškai leidžia įrenginiams jungtis prie tinklo naudojant Wi-Fi, suteikdami lankstumą ir mobilumą.	Petrulis, T. (2018)
VPN Tuneliai	VPN tuneliai užtikrina saugų duomenų perdavimą internetu, suteikiant privatumą ir konfidencialumą.	Vaitkus, V. (2021)
Raspberry Pi	Raspberry Pi yra mažo dydžio vienos plokštės kompiuteris, populiarus dėl savo lankstumo ir pritaikomumo įvairiuose projektuose.	Kavaliauskas, P. (2019)
Flask serveris	Flask yra mikro įrankių rinkinys ir lengvas WSGI tinklalapių serveris, naudojamas lengvoms interneto aplikacijoms kurti.	Martinkus, R. (2021)
Python	Python yra aukšto lygio programavimo kalba, žinoma dėl savo aiškumo ir universalumo, plačiai naudojama moksle ir pramonėje.	Adams, B. (2020)
Java	Java yra objektinio programavimo kalba, skirta kurti platformų nepriklausomas aplikacijas.	Singh, L. (2021)

SANTRAUKA

Autorius Nerijus Kvedaravičius. *Išmaniosios stovėjimo aikštelės projektas*. Baigiamasis darbas. Vadovas Gintaras Butkus. Kauno kolegija, Technologijų fakultetas, Informatikos ir medijų technologijų katedra. Kaunas, 2024, 42 psl.

Reikšminiai žodžiai: parkavimo sistema, sensoriai, mobilioji aplikacija, rezervacija.

Baigiamojo darbo tikslas – sukurti išmaniąją parkavimo sistemą, kuri automatizuoja parkavimo vietų prieinamumo stebėjimą ir valdymą. Analizės skyriuje aptariamos esamos sistemos problemos ir vartotojų poreikiai. Specifikacijos skyriuje apibrėžiami sistemos techniniai reikalavimai. Projektavimo skyriuje sukurta detalioji sistemos struktūra su reikalingomis funkcijomis. Praktinėje dalyje testuojamas prototipas, o ekonominėje dalyje įvertinama sistemos nauda, įskaitant efektyvumo didinimą ir laiko taupymą.

SUMMARY

Author Nerijus Kvedaravičius. *Project for Intelligent Parking Lot*. Graduation Thesis. Supervisor Gintaras Butkus. Kauno kolegija HEI, Faculty of Technologies, Department of Informatics and Media technologies. Kaunas, 2024, 42 pages.

Keywords: parking system, sensors, mobile application, reservation.

The goal of this thesis is to develop an intelligent parking system that automates the monitoring and management of parking space availability. The analysis section discusses the problems of existing systems and user needs. Technical requirements for the system are defined in the specifications section. The design section creates a detailed system structure with necessary functions. In the practical section, the prototype is tested, and in the economic section, the benefits of system implementation are evaluated, including increased efficiency and time savings.

ĮVADAS

Darbo aktualumas. Šiuolaikinėje visuomenėje, didėjant automobilių skaičiui ir urbanizacijai, parkavimo problemos tampa vis aktualesnės. Dažnai žmonės sugaišta laiką ieškodami parkavimo vietos, kas ne tik sukelia stresą ir nepatogumus, bet ir žymiai mažina darbo produktyvumą. Ypač tai pasakytina apie didelės įmonės darbuotojus, kurie priversti kovoti dėl ribotų parkavimo vietų. Ši situacija ne tik trukdo asmeniniam našumui, bet ir daro įtaką bendrovės efektyvumui bei finansiniams rezultatams.

Problema. Parkavimo vietų trūkumas ir neefektyvus jų paskirstymas sukelia didelį nepatogumą ir veiklos efektyvumo mažėjimą didelėse įmonėse. Darbuotojams tenka ilgai ieškoti laisvų vietų, o tai kelia stresą ir mažina jų dienos produktyvumą. Dėl to kyla reali ekonominė problema įmonėms, nes mažesnis darbuotojų produktyvumas tiesiogiai veikia įmonės pelną ir bendrą našumą.

Darbo objektas. Siūlomas sprendimas – inovatyvi parkavimo vietų stebėjimo ir valdymo sistema, kuri automatiškai skiria parkavimo vietas įmonės darbuotojams, atsižvelgiant į jų atvykimo laiką ir kitus veiksnius. Sistema naudoja ultrasensinius jutiklius ir mobiliąją aplikaciją, kad optimizuotų parkavimo procesą ir sumažintų susijusį stresą bei laiko švaistymą.

Darbo tikslas. Sukurti ir įdiegti pažangią parkavimo vietų stebėjimo ir valdymo sistemą, kuri leistų darbuotojams efektyviau ir be papildomo streso rasti laisvas parkavimo vietas, taip pat leistų įmonėms geriau išnaudoti esamas parkavimo erdves, didinti darbuotojų pasitenkinimą ir bendrą produktyvumą.

Uždaviniai.

1. Apžvelgti ir įvertinti esamas parkavimo vietų stebėjimo ir valdymo sistemas, jų privalumus ir trūkumus.
2. Sukurti modernią parkavimo vietų stebėjimo ir valdymo sistemą, remiantis ultrasensinių jutiklių technologija ir mobiliąja aplikacija.
3. Išanalizuoti integracijos planą integruojant ultrasensinius jutiklius ir mobiliąją aplikaciją su esama įmonės infrastruktūra.
4. Išplėtoti algoritmą efektyviam parkavimo vietų paskirstymui, atsižvelgiant į darbuotojų atvykimo laikus ir kitus svarbius veiksnius.
5. Sukurti vartotojui draugišką mobiliosios aplikacijos sąsają, leidžiančią darbuotojams lengvai rezervuoti parkavimo vietas ir gauti aktualią informaciją apie laisvas vietas.
6. Užtikrinti duomenų saugumą ir sistemos patikimumą.

Rezultatai. Sukurta išmanioji parkavimo sistema leidžia efektyviai valdyti parkavimo vietas, žymiai mažindama laiką, kurį vairuotojai praleidžia ieškodami vietos, ir sumažinant transporto

sukeltą taršą. Sistema, integruojanti ultragarsiniai jutikliai ir mobiliosios aplikacijos technologiją, automatizuoja parkavimo procesą ir suteikia realiuoju laiku atnaujinamus duomenis apie laisvas vietas. Dėl šių ypatybių sistema gerina vartotojo patirtį, užtikrina aukštą duomenų saugumo lygį ir optimizuoja parkavimo vietų naudojimą. Tai ne tik padidina įmonės operacinį efektyvumą, bet ir teikia ženklią ekonominę naudą, mažindama žmogiškųjų ir finansinių išteklių švaistymą, susijusį su parkavimo valdymu.

1. ANALITINĖ DALIS

1.1. Situacijos analizė

Dabartinė parkavimo situacija miestuose ir didelėse įmonėse dažnai yra nepakankamai efektyvi ir sukuria daugybę problemų tiek vairuotojams, tiek miesto infrastruktūrai. D. Shoup savo straipsnyje (2006) teigia, kad ilgas laisvos parkavimo vietos paieškos laikas yra vienas iš pagrindinių miesto eismo spūsčių ir aplinkos taršos veiksnių. Vienas vairuotojas vidutiniškai iki 20 minučių miestuose praleidžia ieškodamas laisvos vietos, o tai sudaro milžinišką bendrą laiko švaistymą (Smith, 2018).

Norėdami šią problemą išanalizuoti ir rasti sprendimo būdus, analizuosime populiariausias naudojamas parkavimo sistemas.

1.2. Esamų parkavimo stebėjimo sistemų analizė ir palyginimas

Šiame skyriuje analizuojamos dvi šiuolaikinės parkavimo valdymo sistemos: ADCParking ir UniPark. Šios sistemos buvo pasirinktos dėl jų novatoriškų sprendimų parkavimo vietų stebėjimo ir valdymo srityje, kurios galėtų būti pritaikytos sprendžiant didelių įmonių parkavimo problemas. Analizės tikslas – įvertinti kiekvienos sistemos veikimo principus, naudojamas technologijas ir jų efektyvumą, remiantis iš anksto nustatytais kriterijais, komponentais ir savybėmis.

1.2.1. ADCParking

ADCParking sistema apima kelias pagrindines dalis: parkavimo jutiklius, centralizuotą valdymo serverį ir mobiliosios aplikacijos klientą. Jutikliai, montuojami kiekvienoje parkavimo vietoje, fiksuoja ir siunčia duomenis apie laisvas arba užimtas vietas į centralizuotą serverį realiuoju laiku.

Savybės. ADCParking sistema suteikia vairuotojams galimybę realiu laiku matyti laisvas parkavimo vietas per mobiliąją aplikaciją. Be to, ADCParking integracija su debesų kompiuterijos paslaugomis leidžia saugoti didelius duomenų kiekius ir užtikrina sistemos patikimumą ir prieinamumą.

1.2.2. UniPark

UniPark sistema taip pat susideda iš trijų pagrindinių komponentų: parkavimo vietų jutiklių, valdymo serverio ir vartotojo aplikacijos. Jutikliai stebi ir registruoja parkavimo vietų būseną, o duomenys apie parkavimo vietas yra siunčiami ir tvarkomi serveryje.

Savybės. UniPark išsiskiria savo patogia vartotojo sąsaja, kuri vairuotojams pateikia intuityvią ir lengvai naudojamą informaciją apie laisvas parkavimo vietas. Be to, UniPark sistema siūlo išplėstines funkcijas, tokias kaip parkavimo laiko stebėjimas, mokėjimai per programėlę ir automatiniai įspėjimai apie laiko pasibaigimą. Tai sumažina vairuotojų stresą ir leidžia efektyviau planuoti laiką.

1.2.3. Palyginimas

Žemiau pateiktoje lentelėje (1 lentelė) yra išvardyti svarbūs kriterijai ir palyginamos dvi parkavimo sistemos: ADC Parking ir UniPark. Ši lentelė padeda įvertinti kiekvienos sistemos ypatybes ir skirtumus pagal įvairius aspektus:

1 lentelė. Palyginimas tarp ADCParking ir UniPark

Kriterijus	ADCParking	UniPark
Rezervavimas	Ne	Ne
Mokėjimo būdai	Kasa, Mobilioji aplikacija	Kasa, Mobilioji aplikacija, SMS
Mobilioji aplikacija	Taip	Taip
Stovėjimo Abonimentai	Taip	Ne
Pagrindinės veikimo vietos	Vilnius, Kaunas, Daugiaaukštės aikštelės	Vilnius, Kaunas, Oro Ustai, Požeminės Aikštelės
Saugumas	Aukšto lygio duomenų šifravimas ir apsauga nuo neteisėtos prieigos	Stiprus autentifikavimas ir duomenų apsauga
Duomenų analizė	Išsami	Bazinė
Papildomos Funkcijos	Integruotas sencorius į kiekvieną stovėjimo vietą rodo kurios vietos yra užimtos	Su šia programėle galima susimokėti ne tik už stovėjima aikštelėse bet ir už stovėjima miestuose, ten kur yra mokomos zonos

ADCParking ir UniPark teikia parkavimo sprendimus, tačiau jų dėmesio sritys skiriasi. ADCParking yra labiau orientuotas į aikštelių parkavimą, siūlydamas abonementus ir aukšto lygio saugumą, ypač daugiaaukščių parkavimo vietose, kaip Vilniuje ir Kaune. Jie taip pat teikia išsamią duomenų analizę ir integruotus sensorius, didinant vartotojų patogumą.

Tuo tarpu UniPark yra lankstesnis, aptarnaujant tiek miestų, tiek oro uostų parkavimą ir siūlydamas įvairias mokėjimo galimybes, įskaitant SMS. Jų sistema taip pat leidžia mokėti už mokamas miesto zonas, tačiau neteikia stovėjimo abonementų, kas rodo didesnį dėmesį laikinam naudojimui.

Taigi, ADCParking yra labiau pritaikytas ilgalaikiam, saugiam parkavimui, o UniPark siūlo lankstesnes paslaugas plačiam vartotojų ratui.

1.3. Technologijų ir technikos apžvalga

1.3.1. Aparatūros posistemė

Parkavimo sistemose naudojami sensoriai, serveriai ir mobilieji įrenginiai yra pagrindiniai aparatinės įrangos komponentai, kurie leidžia efektyviai nustatyti laisvas ir užimtas parkavimo vietas realiu laiku. Remiantis Zhao ir Guo (2019) bei Martínez ir Rojas (2021) tyrimais, ultragarsiniai, infraraudonųjų spindulių ir magnetiniai sensoriai yra itin svarbūs nustatant parkavimo vietų būklę, o serveriai atlieka esminį vaidmenį duomenų apdorojime ir saugojime. Mobilieji įrenginiai, pasitelkiant interneto daiktų (IoT) technologijas, suteikia vartotojams galimybę greitai ir patogiai naudotis parkavimo paslaugomis.

Parkavimo sistemų efektyvumas priklauso nuo jų gebėjimo tikslingai nustatyti ir valdyti prieinamas parkavimo vietas. Ši lentelė pateikia įvairių sensorių tipus, jų veikimo principus bei privalumus ir trūkumus, remiantis naujausiais moksliniais šaltiniais. Pavyzdžiui, ultragarsiniai sensoriai, nustatantys objektų atstumą naudojant ultragarsą, yra vertinami už tikslius atstumo matavimus ir nesudėtingą diegimą, kaip aptarta Belbachiro (2009) knygoje apie išmaniąsias kameras. Kita vertus, infraraudonųjų spindulių sensoriai, kurie fiksuoja šilumos šaltinius ir jų atstumą, yra paminėti Bogue (2013) straipsnyje kaip efektyvūs, bet pažeidžiami oro sąlygų, tokių kaip saulės šviesa. Taip pat, Guo ir Zhao (2019) apžvalgoje pabrėžiama sensorių, tokių kaip magnetiniai, kurie aptinka metalinių objektų buvimą, svarba parkavimo sistemose dėl jų universalumo. Ši informacija yra esminė suprantant, kaip modernios parkavimo sistemos naudoja šiuos technologinius sprendimus, siekdamos pagerinti parkavimo patirtį miestų gyventojams.

Iterpti teksta kuris pasako kad zemiau takeitkoje lenteleje (2 lentelė) yra palyginimai

2 lentelė. Įvairių Sensorinių veikimo principai, privalumai, trūkumai

Sensoriaus tipas	Veikimo principas	Privalumai	Trūkumai
Ultragarso	Nustato objektų atstumą naudojant ultragarsą	Tikslūs atstumų matavimai, nebrangūs, lengva įdiegti	Trikdomi oro sąlygų, gali nustatyti netinkamus objektus
Infraraudonųjų spindulių (IR)	Nustato šilumos šaltinius ir jų atstumą	Efektyvūs žemoje apšvietimo sąlygose, tiksli lokalizacija	Gali būti trikdomi saulės šviesos, aukšta kaina
Magnetiniai	Aptinka metalo objektų buvimą per magnetinius laukus	Galimybė stebėti ir analizuoti eismo srautus, universalumas	Ribotas aptikimo diapazonas, tik metalinių objektų aptikimas
Vaizdo kamerų	Stebi ir analizuoja vaizdo įrašus	Galimybė stebėti ir analizuoti eismo srautus, universalumas	Brangi įranga ir priežiūra, reikalauja pažangių analizės algoritmų
LiDAR (Šviesos aptikimas ir atstumo nustatymas)	Nustato objektus ir jų atstumą naudojant šviesos impulsus	Aukštas tikslumas ir detalumas, veikia įvairiomis oro sąlygomis	Labai brangūs, sudėtingas įdiegimas ir priežiūra

1.3.2. Informacinė posistemė

Informacinė posistemė parkavimo valdymo sistemoje yra kritinis komponentas, užtikrinantis glaudų duomenų srautą tarp parkavimo sensorių, serverio, mobiliosios aplikacijos ir galutinio naudotojo. Procesas vyksta nuosekliai: pradedant nuo sensoriaus, kuris nustato užimtą vietą ir informuoja serverį, baigiant vartotojo sąveika su aplikacija, kur jis gali peržiūrėti parkavimo vietų būklę ir priimti sprendimus. Šią sistemą galima detaliai pamatyti pateiktoje schemoje (1.1 pav.). Sistemos patikimumą užtikrina kibernetinis saugumas, apimantis duomenų šifravimą ir privatumo apsaugą, remiantis Smith (2018) ir Johnson (2019) tyrimais.

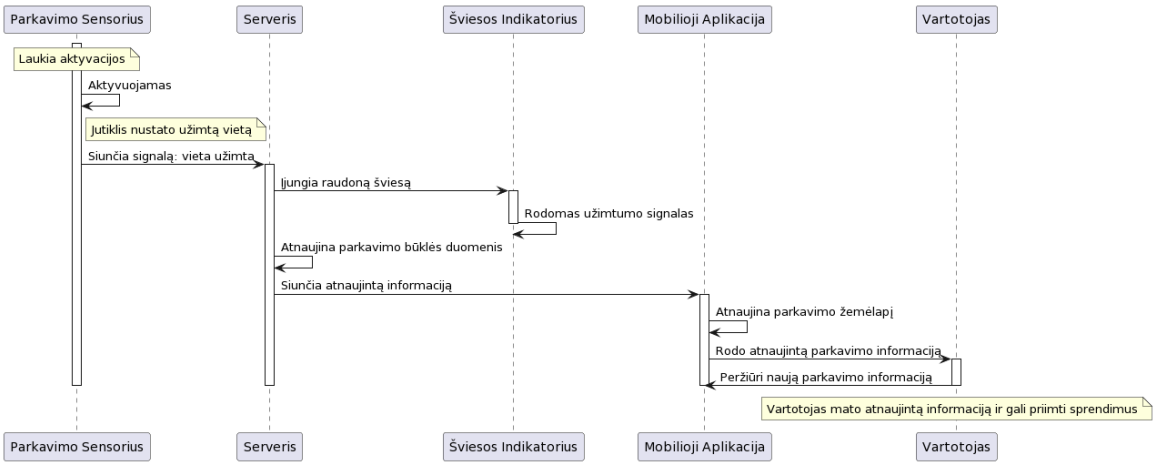
Sensoriaus aktyvavimas ir duomenų siuntimas: Visas procesas prasideda, kai parkavimo sensorius nustato, kad parkavimo vieta yra užimta. Sensorius yra programuotas laukti ir stebėti parkavimo vietos būklę. Kai automobilis užima vietą, sensorius aktyvuojamas ir siunčia signalą į centralizuotą serverį. Šis signalas apima informaciją apie parkavimo vietos statusą – šiuo atveju, kad vieta yra užimta.

Serverio reakcija ir duomenų apdorojimas: Gavęs signalą iš parkavimo sensoriaus, serveris nedelsiant atnaujina parkavimo būklės duomenis savo sistemoje. Tai yra svarbus žingsnis, nes serverio palaikoma informacija yra pagrindinis informacijos šaltinis visai sistemai, įskaitant mobiliosios aplikacijos vartotojus.

Šviesos indikatoriaus valdymas: Serveris taip pat siunčia signalą į šviesos indikatorių, kuris yra prie parkavimo vietos. Šis signalas liepia įjungti raudoną šviesą, signalizuojant, kad vieta yra užimta. Tai suteikia vizualinę informaciją apie parkavimo vietos statusą ne tik aplikacijos naudotojams, bet ir tiems, kurie ieško parkavimo vietos fiziškai vietoje.

Mobiliosios aplikacijos atnaujinimas: Serveris siunčia atnaujintą informaciją apie parkavimo vietų būklę į mobiliosios aplikacijos serverį, kuris savo ruožtu atnaujina duomenis vartotojo mobiliojoje aplikacijoje. Ši informacija apima žemėlapius, parkavimo vietų būsenas ir kitus reikalingus duomenis.

Vartotojo sąveika su aplikacija: Galiausiai, mobilioji aplikacija rodo atnaujintą informaciją vartotojui. Vartotojas gali matyti realiu laiku atnaujintus duomenis apie laisvas ir užimtas parkavimo vietas, leidžiančius jam priimti informuotus sprendimus dėl parkavimo. Naudotojas taip pat gali atlikti veiksmus per aplikaciją, pavyzdžiui, rezervuoti parkavimo vietą arba atlikti mokėjimą.



1.1 pav. Parkavimo vietos užimtumo ir informacijos perdavimo procesas

Šiame paveikslėlyje vaizduojamas informacijos srautas tarp parkavimo sensoriaus, serverio ir vartotojo. Diagramoje parodytas procesas, kai sensorius užfiksuoja užimtą parkavimo vietą, siunčia signalą į serverį, o tada atnaujinta informacija pasiekia vartotojus.

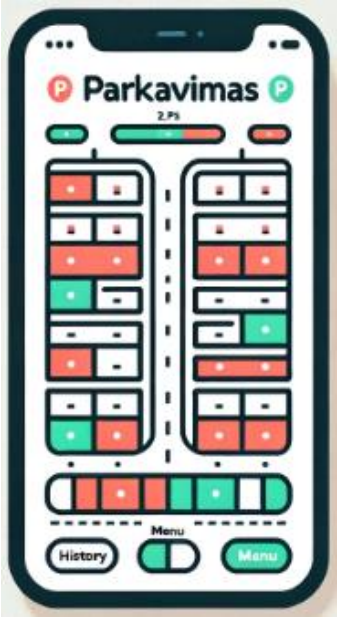
1.3.3. Naudotojo sąsaja

Naudotojo sąsaja (UI) yra esminis parkavimo valdymo sistemos komponentas, leidžiantis galutiniam vartotojui sąveikauti su sistema. Ji turi būti suprojektuota taip, kad vartotojas galėtų lengvai naudotis visomis sistemos siūlomomis funkcijomis. Svarbu, kad naudotojo sąsaja būtų intuityvi, patraukli ir informatyvi.

Programavimo kalbos ir aplinkos: Naudotojo sąsajai sukurti dažniausiai naudojamos JavaScript, HTML ir CSS kalbos. JavaScript yra naudojama dinaminiam turinio kūrimui, HTML – struktūriniam maketavimui, o CSS – vizualiniam dizainui. Šios kalbos, kartu su atitinkamomis programavimo bibliotekomis ir karkasais, pavyzdžiui React ar Vue.js, leidžia kurti patrauklias ir funkcionalias vartotojo sąsajas.

Programavimo metodikos ir technologijos: Naudotojo sąsajos kūrime taikoma atsako dizaino metodika, leidžianti svetainėms ir aplikacijoms teisingai veikti įvairaus dydžio ekranuose, nuo mobiliųjų telefonų iki didelių monitorių. Be to, naudojamos naudotojo patirties (UX) gerinimo technikos, siekiant užtikrinti, kad aplikacija būtų logiška, lengvai naudojama ir maloni vartotojui.

Operacinės sistemos: Parkavimo sistemos naudotojo sąsajos yra kūrimui ir naudojimui pritaikytos tiek Android, tiek iOS operacinėms sistemoms, siekiant aprėpti kuo didesnį vartotojų ratą. Pritaikomumas šioms sistemoms užtikrina sklandų funkcionalumą ir patogią naudotojo patirtį.

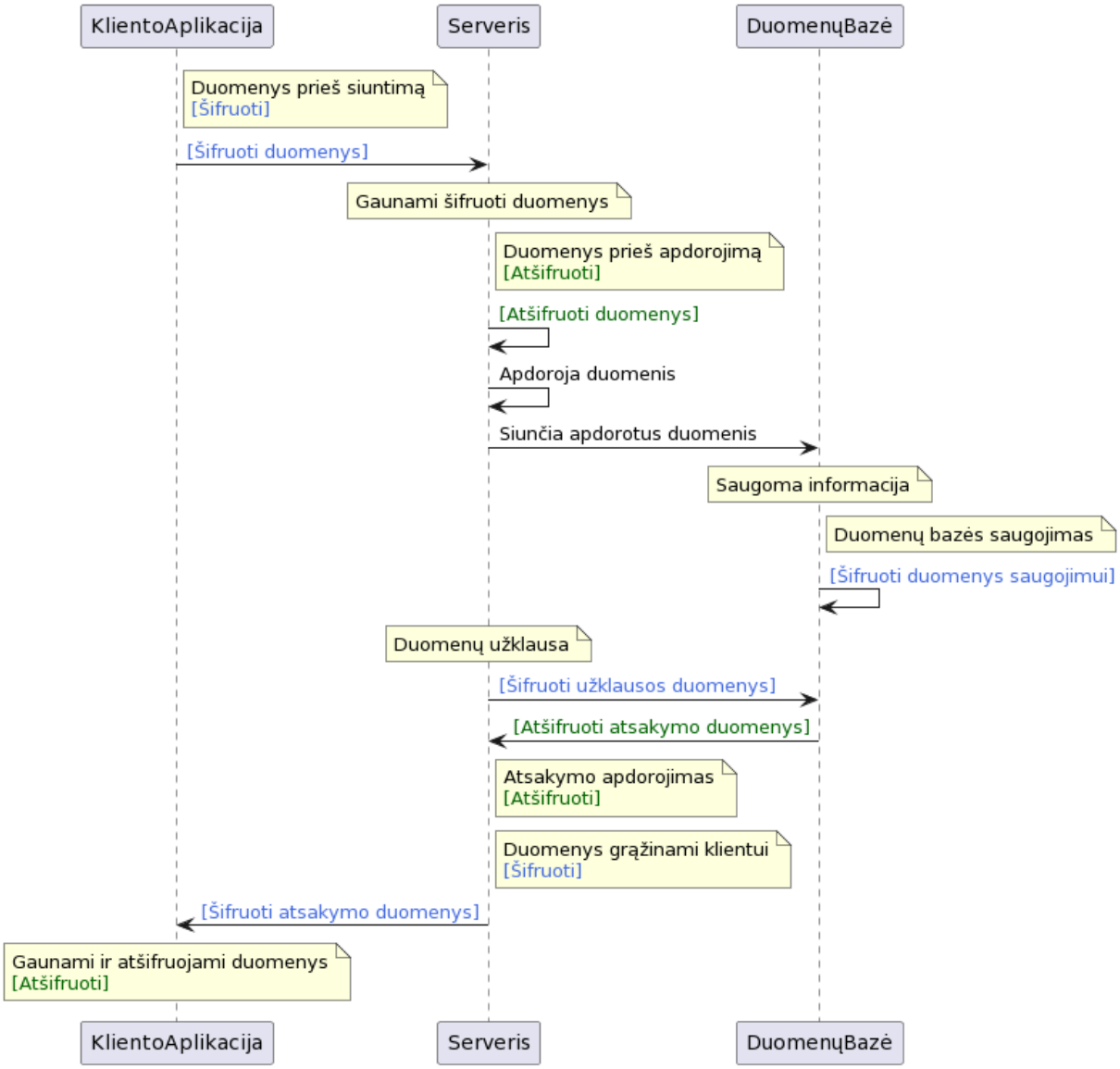


1.2 pav. Pavyzdinis parkavimo valdymo aplikacijos variantas

Pavyzdyje matomas parkavimo valdymo aplikacijos vaizdas (1.2 pav.), kuris iliustruoja, kaip vartotojai gali vizualiai identifikuoti laisvas ir užimtas parkavimo vietas.

1.4. Saugumo aspektai

Sistemos saugumo aspektai yra neatsiejami nuo šiuolaikinių technologinių sprendimų. Tai apima kompleksinius duomenų apsaugos metodus, tokie kaip šifravimas ir duomenų perdavimas tarp įrenginių. Išsamiau tai nagrinėja, Owen ir Makkar (2017) straipsnyje apie kibernetinę saugą IoT įrenginiuose ir Choo (2020) darbe, kuriame aptariama privatumo ir duomenų saugojimo svarba.



1.3 pav. Parkavimo vietos užimtumo ir informacijos perdavimo procesas

1.3 paveiksle vaizduojama duomenų perdavimo schema, kurioje iš kliento aplikacijos duomenys siunčiami į serverį, o po to į duomenų bazę. Kiekviename etape duomenys yra šifruojami, užtikrinant saugumą perduodant juos iš vienos sistemos dalies į kitą. Serveris atlieka duomenų apdorojimą ir, esant poreikiui, grąžina šifruotą atsakymą atgal į kliento aplikaciją. Ši schema akcentuoja svarbų duomenų šifravimo ir saugaus saugojimo procesą, kuris yra būtinas norint užtikrinti visos sistemos saugumą ir patikimumą.

1.5. Išvados ir sprendimas

Analitinėje dalyje, ištyrėme pagrindines parkavimo miestuose problemas, įvertinome įvairias technologijas ir jų pritaikymą parkavimo vietų stebėjimo bei valdymo sistemose. Svarbiausias

iššūkis – užtikrinti, kad parkavimo procesas būtų kuo sklandesnis ir mažiau stresą keliantis vartotojams.

Išvados:

- **Automatinis parkavimo vietų paskirstymas:** Efektyvus ir greitas parkavimo vietų paskirstymas yra esminis, siekiant pagerinti vartotojų patirtį ir sumažinti eismo spūstis.
- **Technologijų integracija:** Sėkmingas parkavimo vietų valdymas priklauso nuo pažangių technologijų, tokių kaip IoT sensoriai ir šviesos indikatoriai, integracijos.
- **Naudotojo sąsajos supaprastinimas:** Aiški ir intuityvi naudotojo sąsaja yra būtina, kad vartotojai lengvai ir greitai galėtų suvokti bei naudotis sistema.

Sprendimas:

Atsižvelgiant į atliktą analizę ir identifikuotas pagrindines problemas, siūlomas sprendimas yra sukurti integruotą parkavimo sistemą, kuri automatiškai paskiria parkavimo vietą kiekvienam vairuotojui, tik pravažiavus pro kelio užtvarą. Sprendimas apima šiuos komponentus:

- **Kelio užtvara su integruotais sensoriais:** Tai yra pirmasis kontakto taškas su vairuotoju. Čia sensoriai nuskaito atvykstantį automobilį ir perduoda signalą į centrinį serverį.
- **Centrinis serveris:** Apdoroja gautą informaciją ir remiantis esama parkavimo vietų užimtumo situacija, automatiškai paskiria laisvą parkavimo vietą vairuotojui.
- **Mobilioji aplikacija:** Informuoja vairuotoją apie jam paskirtą parkavimo vietą ir pateikia maršrutą iki jos.
- **Parkavimo vietų sensoriai ir šviesos indikatoriai:** Kiekvienoje parkavimo vietoje esantys sensoriai nustato vietos būseną (užimta/arba laisva), o šviesos indikatoriai (žalia šviesa laisvai vietai, raudona - užimtai) padeda vairuotojams lengvai orientuotis parkavimo aikštelėje.

Šis sprendimas ne tik supaprastins parkavimo procesą, bet ir padės efektyviai išnaudoti parkavimo erdvę, mažinti eismo spūstis ir pagerinti vairuotojų patirtį. Taip pat šis modelis užtikrina didesnį parkavimo efektyvumą ir didina vartotojų pasitenkinimą, mažindamas laiką, praleidžiamą ieškant laisvos vietos.

2. SPECIFIKACIJA

Projektuojamas objektas. Intelektuali parkavimo sistema, naudojanti IoT sensorius ir mobiliosios aplikacijos sąsają parkavimo vietų valdymui ir vartotojų informavimui.

Projektuojamo objekto paskirtis. Sukurti patogesnį ir efektyvesnį parkavimo procesą miesto sąlygomis, mažinant laiko švaistymą ieškant laisvų vietų ir optimizuojant parkavimo vietas.

Projektuojamo objekto funkcijos.

- Automatinis laisvų ir užimtų vietų atpažinimas.
- Duomenų apdorojimas ir valdymas realiu laiku.
- Integruota vartotojo sąsaja rezervacijoms ir mokėjimams.

Aparatūros posistemė. Sensoriai, duomenų serveriai, šviesos indikatoriai ir kitos būtinos įrangos dalys, užtikrinančios stabilų sistemos veikimą.

Informacijos posistemė. Duomenų bazių valdymas, duomenų perdavimas ir šifravimas, integruoti su aparatūros posisteme.

Naudotojo sąsaja. Patogi ir intuityvi mobilioji aplikacija, leidžianti peržiūrėti parkavimo vietas ir atlikti veiksmus, pavyzdžiui, rezervacijas.

Reikalavimai saugumui. Stiprūs kibernetinio saugumo protokolai, duomenų šifravimas ir vartotojų privatumo užtikrinimas.

Reikalavimai realizacijai. Projektui reikalingos technologijos, infrastruktūra ir kompetencijos, užtikrinančios sklandų įgyvendinimą.

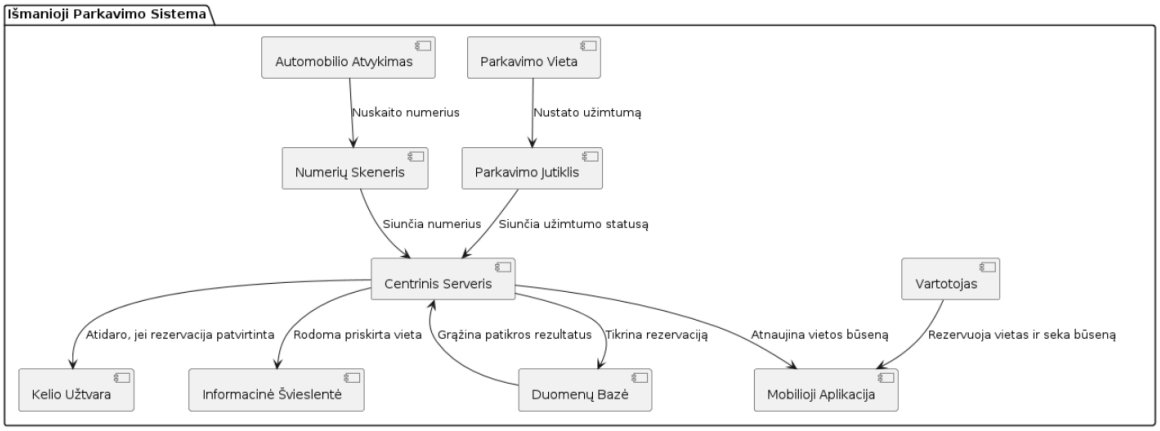
Reikalavimai projekto dokumentacijai. Išsamus projektavimo, diegimo ir eksploatavimo instrukcijų aprašymas, atitinkantis standartus ir teisinius reikalavimus.

3. PROJEKTINĖ DALIS

3.1. Aparatūros posistemė

3.1.1. Projektuojamo objekto konceptuali schema

Ši konceptuali schema atspindi išmaniosios parkavimo sistemos veikimo principą.



3.1 pav. Išmanios stovėjimo aikštelės veikimo konceptuali schema

Kliento automobilio atvykimas pradeda procesą, kurio metu numerių skeneris nuskaito automobilio numerius ir perduoda juos centriniame serveryje. Serveris patikrina duomenų bazėje, ar šie numeriai turi išankstinę rezervaciją. Jei rezervacija yra, kelio užtvara atidaroma, o vairuotojui per informacinę švieslentę pateikiama priskirta parkavimo vieta. Kai automobilis užima vietą, parkavimo jutiklis nustato vietos užimtumą ir atnaujina šią informaciją serveryje, kuri tuomet perduodama vartotojui per mobiliosios aplikacijos sąsają. Tai leidžia vartotojui realiu laiku stebėti vietos užimtumą ir valdyti savo rezervacijas.

3.1.2. Darbo vietų sąsajos su specifikuotomis funkcijomis

Šioje (3 lentelė) pateikiami svarbūs parkavimo valdymo sistemos komponentai ir jų atsakomybės, aptariant kiekvieno įrenginio ir programinės dalies vaidmenį efektyvaus parkavimo procese. Lentelė parodo, kaip komponentai, tokių kaip numerių skeneris, duomenų bazė, kelio užtvara, ir kiti, sąveikauja tarpusavyje, užtikrindami, kad automobiliai būtų tinkamai identifikuoti, jų rezervacijos patikrintos ir suteikiamas prieigas prie skiriamų vietų. Tai ne tik aprašo kiekvieno komponento funkcijas, bet ir paaiškina, kaip jie prisideda prie visos parkavimo sistemos efektyvumo ir vartotojo patirties gerinimo.

3 lentelė. Reikalingi projekto komponentai

Komponentė	Atsakomybės
Numerių Skeneris	Nuskaito atvykstančio automobilio numerius ir siunčia juos į centrini serveri.
Duomenų Bazė	Saugo visus parkavimo sistemos duomenis, įskaitant rezervacijas ir automobilių numerius.
Kelio Užtvara	Atidaroma, kai centrini serveris patvirtina rezervaciją.
Centrinis Serveris	Patikrina duomenų bazėje, ar automobilio numeriai turi išankstinę rezervaciją; valdo kelio užtvarą ir informacines švieslentes; siunčia atnaujintą informaciją į mobilioją aplikaciją.
Informacinė Švieslentė	Rodoma vartotojo priskirta parkavimo vieta.
Parkavimo Jutiklis	Nustato, ar parkavimo vieta yra užimta; siunčia šią informaciją į centrini serveri.
Mobilioji Aplikacija	Leidžia vartotojui rezervuoti parkavimo vietas, sekti jų būseną, ir atlikti mokėjimus.
Vartotojas	Naudodamasis mobiliosios aplikacijos sąsaja, valdo savo rezervacijas ir stebi parkavimo vietas.

Ši lentelė leidžia suvokti, kaip sklandžiai ir integruotai veikia parkavimo sistema, kiekvienas komponentas atlieka svarbų vaidmenį užtikrinant efektyvų parkavimo vietų valdymą.

3.1.3. Kompiuterių darbo vietoms parinkimas ir pagrindimas

Efektyviam parkavimo valdymo sistemos veikimui reikia kompiuterių, kurie galėtų greitai ir efektyviai tvarkyti duomenų srautus iš daviklių, mobiliosios aplikacijos ir kitų sistemos komponentų. Štai pagrindiniai kriterijai ir reikalavimai:

- **Procesoriaus galia**: Reikalingas procesorius, ne silpnesnis nei daugiabranduolių tipas, kuris galėtų greitai apdoroti gaunamus duomenis realiuoju laiku. Optimaliai tinka procesoriai, ne blogesni kaip Intel Core i5 ar AMD Ryzen 5, kurie užtikrina sklandų daugybinio užduočių vykdymą (VU ITPC, 2023).
- **Operatyvioji atmintis.** Reikalinga RAM atmintis, ne mažesnė kaip 16 GB, kad sistema galėtų efektyviai tvarkyti duomenų srautus be užstrigimų, ypač aukšto apkrovimo metu (IThink, 2023).
- **Talpos kaupikliai.** SSD diskiniai kaupikliai, ne mažesni nei 256 GB, užtikrina greitą duomenų įrašymą ir skaitymą, kas yra svarbu duomenų bazės operacijoms ir greitai duomenų atnaujinimui (Bitė Lietuva, 2023).
- **Tinklo pajėgumai**: Gigabitinės interneto jungtys arba greitas bevielis ryšys, ne lėtesnis už 1 Gbps, yra svarbus, kad užtikrintų greitą ir stabilų duomenų perdavimą tarp sistemos komponentų.
- **Saugumo moduliai**: Kompiuteriai turėtų būti aprūpinti saugumo moduliais, pavyzdžiui, TPM (Trusted Platform Module), ne žemesnis kaip 2.0 versija, kurie padeda apsaugoti duomenis nuo kibernetinių atakų.

Šių techninių charakteristikų pasirinkimas leidžia užtikrinti, kad parkavimo sistemos infrastruktūra būtų patikima, greita ir atspari trukdžiams, todėl tai svarbus žingsnis užtikrinant visos sistemos efektyvumą ir patikimumą.

3.1.4. Specializuota aparatūra

3.1.4.1. Aparatūrinės priemonės

Parkavimo sistemos aparatiniai komponentai yra atrinkti atsižvelgiant į jų funkcionalumą, patikimumą ir integracijos galimybes. Čia yra pagrindiniai aparatinės įrangos elementai, kurie sudaro šios išmaniosios parkavimo sistemos pagrindą

- **Parkavimo sensoriai:** Naudojami siekiant nustatyti parkavimo vietų užimtumą. Pagal aplinkos sąlygas ir pageidaujamą tikslumą sensoriai gali būti ultragarsiniai arba infraraudonieji. Ultragarso sensoriai idealiai tinka atvirų erdvių parkavimo aikštelėse dėl jų tikslumo matuojant atstumą, o infraraudonieji sensoriai yra labiau pritaikyti uždaroms vietoms su mažesniais atstumais.
- **Šviesos indikatoriai:** Kiekvienai parkavimo vietai priskiriamos lemputės, kur žalia reiškia laisvą vietą, o raudona – užimtą. Šie indikatoriai yra matomi ir lengvai suprantami vairuotojams, suteikiant greitą vizualinę informaciją apie parkavimo vietų prieinamumą.
- **Raspberry Pi 4 model B+:** Šis mikrokompiuteris naudojamas kaip serveris, kuris valdo duomenų srautus tarp sensorių, šviesos indikatorių ir kitų sistemos komponentų. Raspberry Pi 4 B+ pasižymi pakankamu procesoriaus galingumu ir atminties talpa, kad galėtų efektyviai apdoroti realaus laiko duomenis ir valdyti įrenginių sąsajas.
- **Informacinis ekranas:** Šis ekranas naudojamas parkavimo aikštelėse rodant vairuotojams priskirtas parkavimo vietas. Ekrane rodoma vizualinė informacija apie vairuotojo rezervaciją ir nurodoma, kurioje vietoje jis turėtų pasistatyti automobilį.
- **Numerių nuskaitymo kamera:** Ši kamera skirta skenuoti atvykstančių automobilių registracijos numerius. Gauta informacija perduodama į serverį, kur vykdoma rezervacijų patikra. Naudojant šiuolaikines vaizdo analizės technologijas, kamera gali greitai ir tiksliai identifikuoti numerius bet kokiomis oro sąlygomis.

Šių aparatinės įrangos komponentų pasirinkimas užtikrina, kad parkavimo sistema būtų efektyvi, patikima ir lengvai valdoma, teikiant aukšto lygio automatizavimą ir vartotojų patirtį.

3.1.4.2. Programinės priemonės

Įvairios programinės priemonės yra naudojamos norint efektyviai vystyti ir valdyti išmaniosios parkavimo sistemos funkcijas. Šios priemonės apima:

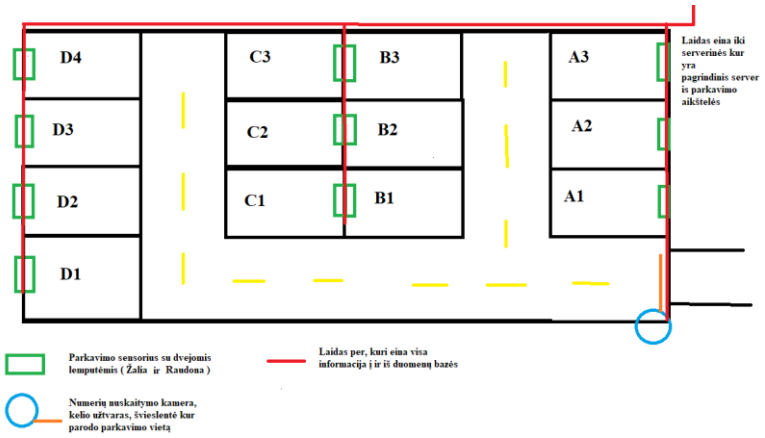
- **Android Studio**: Tai pagrindinė platforma mobiliosios aplikacijos kūrimui, kuri suteikia vartotojams galimybę rezervuoti parkavimo vietas ir sekti jų užimtumo statusą. Android Studio yra integruota vystymo aplinka (IDE), kurioje naudojamos Java ir Kotlin programavimo kalbos, užtikrinant aukštą programinės įrangos kokybę ir patogumą kūrėjams. Aplinka taip pat siūlo galingus testavimo ir debesų sinchronizacijos įrankius, kurie leidžia efektyviai kurti ir valdyti mobilias aplikacijas.
- **Sistemos Programavimas**: Programinis kodas, skirtas valdyti sensorius, šviesos indikatorius ir numerių nuskaitymo kameras, yra kritinis sistemos efektyvumo aspektas. Naudojama Python kalba dėl jos lankstumo ir plataus pritaikymo įvairiose programavimo aplinkose, įskaitant integraciją su aparatūra per Raspberry Pi. Python leidžia greitai kurti ir testuoti skriptus, užtikrinant sklandų duomenų srauto valdymą. Be to, Java gali būti naudojama dėl jos patikimumo ir saugumo savybių, kurios yra svarbios darbui su duomenų bazėmis ir tinklo operacijomis.

Šių programinių priemonių pasirinkimas leidžia ne tik efektyviai vystyti mobiliosios aplikacijos funkcionalumą, bet ir užtikrina aukštą aparatūros komponentų valdymo lygį, būtiną išmaniosios parkavimo sistemos veiklai. Kiekvienas komponentas yra programuojamas atsižvelgiant į jo specifines operacines užduotis, o šių kalbų lankstumas ir efektyvumas padeda optimizuoti sistemos veikimą.

3.1.5. Lokalus tinklas

3.1.5.1. Aikštelės projektas su parkavimo vietų ir aparatūros išdėstymu

Šitame paveikslėlyje (3.2 pav.) vaizduoja išmaniosios parkavimo aikštelės išdėstymo planą, kuriame aiškiai matomi įvairūs komponentai ir jų išsidėstymas. Aikštelė suskirstyta į keturias eiles (D, C, B, A),



3.2 pav. Aikštelės projektas su parkavimo vietų ir aparatūros išdėstymu

Pagrindiniai komponentai:

- **Parkavimo sensoriai:** Kiekvienoje parkavimo vietoje įrengti sensoriai su dviejų spalvų (žalia ir raudona) šviesos indikatoriais, signalizuojančiais apie laisvą ar užimtą vietą. Šie sensoriai padeda vairuotojams greitai sužinoti apie laisvas vietas.
- **Numerių nuskaitymo kamera:** Įėjime į parkavimo aikštelę įrengta kamera, kuri nuskaito atvykstančių automobilių registracijos numerius. Tai leidžia sistemoje patikrinti, ar automobilis turi išankstinę rezervaciją.
- **Kelio užtvaras:** Automatiškai atidaromas, kai patikrinimas patvirtina rezervaciją. Užtvara užtikrina, kad į parkavimo teritoriją patektų tik iš anksto užsiregistravę vartotojai.
- **Švieslentė:** Rodo priskirtą parkavimo vietą, padėdama vairuotojui rasti savo vietą aikštelėje.
- **Serveris:** Valdo visą parkavimo sistemą, įskaitant duomenų apdorojimą ir saugojimą, sensorių ir kitų komponentų veiklą. Serveris yra esminė parkavimo sistemos dalis, nes užtikrina visų komponentų sinchroninį darbą ir efektyvią duomenų perdavimo bei apdorojimo operaciją.

Infrastruktūra:

- **Laidai:** Visame parkavimo aikštelės perimetre vedžiojami laidai (raudona linija), kurie yra naudojami duomenų perdavimui iš sensorių į duomenų bazę ir atgal. Laidai yra paslėpti po žeme, kad nesugadintų estetikos ir būtų apsaugoti nuo išorinių pažeidimų. Jie eina iki serverio, kuris yra laikomas saugioje serverinėje, esančioje kompanijos pastate. Tai užtikrina, kad visi duomenys būtų saugomi patikimai ir sistema veiktų sklandžiai.

3.1.5.2. Techninės tinklo priemonės

Efektyviam ir saugiam duomenų srauto valdymui mūsų išmaniojoje parkavimo sistemoje būtina įdiegti patikimas technines tinklo priemones. Šios priemonės yra gyvybiškai svarbios užtikrinant ne tik sklandų duomenų perdavimą tarp sistemos komponentų, bet ir aukštą bendrą sistemos saugumą.

Maršrutizatoriai ir Jungikliai.

- Maršrutizatoriai efektyviai užtikrina duomenų perdavimą tarp išorinio interneto ryšio ir vidinio parkavimo sistemos tinklo. Tokie įrenginiai turėtų palaikyti gigabitines greitį, kad nebūtų jokių delsų perduodant vaizdo duomenis ar stebint sistemą realiu laiku (Bitė Lietuva, 2023).
- Jungikliai naudojami efektyviam duomenų perdavimui viduje parkavimo aikštelės tinklo, leidžiant greitai ir efektyviai paskirstyti duomenis tarp serverio, daviklių ir kitų sistemos komponentų.

Tinklo Saugumo Užtvaros.

- Apsaugo tinklą nuo neautorizuoto prieigos, filtruodamos eismą pagal nustatytas taisykles ir užkertant kelią potencialiems kibernetiniams išpuoliams (Teltonika Networks, 2023).

Bevielio Ryšio Prieigos Taškai.

- Suteikia mobilumo lankstumą parkavimo aikštelėje, leidžiant vartotojams ir priežiūros personalui jungtis prie sistemos per mobilius įrenginius, užtikrinant, kad ryšys būtų stabilus net judant po didelę aikštelę.

VPN Tuneliai.

- Užtikrina saugų duomenų perdavimą tarp parkavimo aikštelės ir kitų įmonės infrastruktūros dalių, ypač jei duomenys turi būti perduodami per internetą (Teltonika Networks, 2023).

Optinės Pluošto Linijos:

- Naudojamos didelio greičio duomenų perdavimui, užtikrinant aukštos kokybės vaizdo ir didelių duomenų kiekių perdavimą be vėlavimo, ypač tarp atokesnių parkavimo aikštelės dalių ir centrinio serverio.

Šios techninės priemonės yra būtinos norint užtikrinti, kad išmani parkavimo sistema veiktų nepriekaištingai ir saugiai, užtikrinant duomenų saugumą ir prieinamumą realiu laiku. Aukštos kokybės tinklo infrastruktūra yra pagrindas sklandžiam ir efektyviam parkavimo valdymo sistemų veikimui.

3.1.5.3. Tinklo operacinės sistemos parinkimas ir pagrindimas

Pasirinkus Linux operacinę sistemą kaip tinklo platformą, šis sprendimas leidžia užtikrinti išmaniosios parkavimo sistemos stabilumą, saugumą ir efektyvumą. Linux yra žinoma dėl savo stabilumo ir patikimumo, kurie yra kritiniai veiksniai užtikrinant, kad parkavimo sistema veiktų be trukdžių net ir didelio apkrovimo metu.

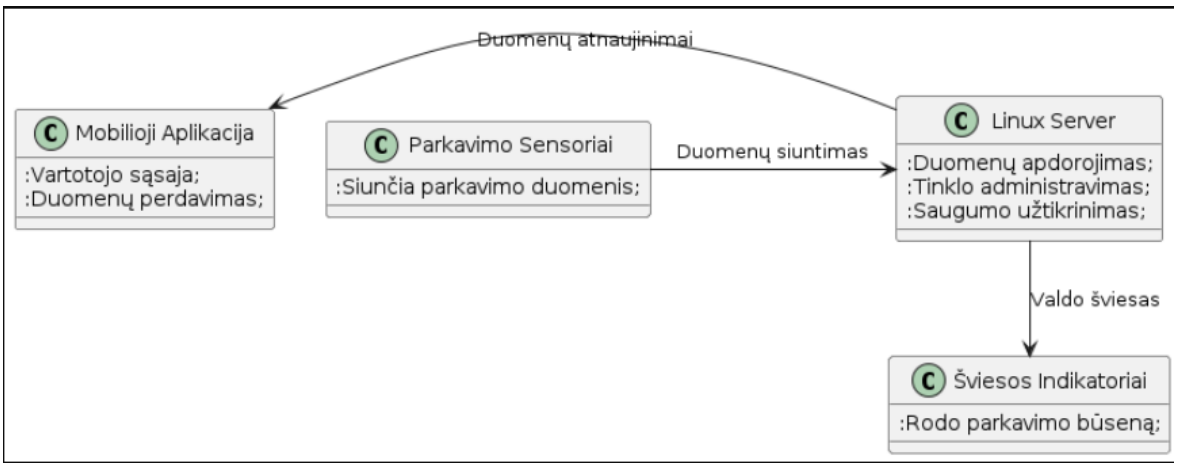
Svarbiausios Linux operacinės sistemos savybės:

- **Stabilumas ir patikimumas**: Linux operacinė sistema užtikrina ilgalaikį ir nepriklausomą veikimą, todėl yra idealus pasirinkimas kritinėms infrastruktūroms, pvz., parkavimo valdymo sistemoms.
- **Saugumo funkcijos**: Linux siūlo galingas integruotas saugumo priemones, įskaitant pažangius ugniasienių nustatymus, pažeidžiamo tinklo valdymą ir prisijungimo identifikavimą, užtikrinant duomenų konfidencialumą ir sistemų apsaugą nuo išorinių grėsmių.
- **Plačios konfigūracijos galimybės**: Dėl savo lankstumo, Linux leidžia lengvai konfigūruoti tinklo nustatymus, kad atitiktų įvairių parkavimo aikštelių dydžius ir poreikius.

- **Palaikymas ir atnaujinimai**: Linux turi platų kūrėjų ir vartotojų bendruomenės palaikymą, kuris garantuoja reguliarius saugumo atnaujinimus ir techninę pagalbą.

Tinklo Schema

Šiame modelyje naudojama Linux operacinė sistema yra integruota į parkavimo sistemos tinklą, kad būtų užtikrintas duomenų srauto valdymas, tinklo priemonių administravimas ir įvairių tinklo operacijų valdymas. Žemiau pateikiama PlantUML Paveikslėlyje (3.3 pav.) , vaizduojantis tinklo operacinės sistemos ir kitų komponentų sąveiką:



3.3 pav. Tinklo schema

Ši schema aiškiai parodo, kaip Linux operacinė sistema valdo parkavimo daviklių duomenis, šviesos indikatorius ir sąveikauja su mobiliosios aplikacijos vartotojo sąsaja. Toks tinklo organizavimas leidžia efektyviai valdyti parkavimo aikštelės operacijas, užtikrinant sistemos patikimumą ir saugumą.

3.2. Informacinė posistemė

3.2.1. Informacinės posistemės koncepcija

Informacinė posistemė skirta efektyviai valdyti ir apdoroti visus duomenis, susijusius su išmaniąja parkavimo sistema. Ši posistemė sukurta siekiant užtikrinti duomenų saugumą, greitį ir prieinamumą, leidžianti sistemai veikti realiuoju laiku. Ji apima duomenų kaupimą, apdorojimą, analizę ir pateikimą per vartotojo sąsają, taip pat užtikrina duomenų mainus tarp įvairių sistemų komponentų, tokių kaip sensoriai, serveriai ir mobiliosios aplikacijos.

3.2.2. Duomenų srautai ir procesai

Duomenų srautai:

- Sensoriai -> Serveris: Parkavimo sensoriai fiksuoja automobilių buvimą vietose ir siunčia šiuos duomenis į serverį.
- Serveris -> Duomenų bazė: Serveris apdoroja gautus duomenis ir atnaujina duomenų bazę su naujausia informacija apie parkavimo būseną.
- Serveris -> Mobilioji aplikacija: Serveris reguliariai siunčia atnaujinimus apie parkavimo vietų būseną į mobiliosios aplikacijos serverį, kuris atnaujina aplikacijoje rodomą informaciją vartotojams.
- Mobilioji aplikacija -> Serveris: Vartotojai per aplikaciją gali rezervuoti parkavimo vietas, o ši informacija siunčiama serveriui tvarkyti.

Procesai:

- Duomenų fiksavimas: Automatizuotas duomenų rinkimas iš parkavimo sensorių.
- Duomenų tinkamumą ir apdorojimas: Serveris tikrina gautų duomenų tinkamumą ir atnaujina duomenų bazę.
- Informacijos atnaujinimas: Periodiškai atnaujinama informacija vartotojų aplikacijose, užtikrinant teisingą ir laiku pateikiamą duomenų vaizdą.
- Saugumo valdymas: Užtikrinamas duomenų šifravimas ir prieigos kontrolė, kad būtų apsaugota nuo neautorizuoto prieigos.

3.2.3. Duomenų bazė

Šiame projekte, vietoje tradicinės duomenų bazės, pasirinktas alternatyvus metodas duomenų saugojimui – duomenų failai. Šis sprendimas leidžia lanksčiau tvarkyti informaciją ir sumažina sistemai keliamus reikalavimus. Taikomi trys pagrindiniai duomenų failų tipai:

- **Parkavimo istorijos failai**: Šie failai saugo informaciją apie kiekvieną parkavimo sesiją, įskaitant laiką, trukmę ir automobilio numerį. Tai leidžia efektyviai sekti ir analizuoti parkavimo įpročius bei optimizuoti parkavimo vietų paskirstymą.
- **Prisijungimo istorijos failai**: Šie failai fiksuoja kiekvieną sistemos naudotojo prisijungimą, užtikrindami saugumo audito galimybę ir padedant nustatyti galimus saugumo pažeidimus.
- **Naudotojų duomenų failai**: Čia saugoma informacija apie vartotojus, įskaitant prisijungimo duomenis ir naudotojo nustatymus. Šie duomenys svarbūs siekiant užtikrinti asmeninės informacijos saugumą ir suteikti naudotojams individualizuotas paslaugas.

Šis duomenų saugojimo metodas pasirinktas dėl jo paprastumo, mažų sisteminių išlaidų ir gebėjimo lengvai integruoti su kitais sistemos komponentais. Failų sistema leidžia greitai ir lanksčiai keisti duomenų saugojimo struktūrą, priklausomai nuo projekto poreikių ir vystymosi. Toks sprendimas yra ypač tinkamas sistemoms, kurios nepatiria didelių duomenų kiekių ar intensyvaus duomenų srauto, ir yra puikiai pritaikytas mobiliųjų parkavimo sistemų kontekste.

3.3. Naudotojo sąsaja

3.3.1. Programavimo kalbos pasirinkimas ir pagrindimas

Naudotojo sąsajos kūrimui pasirinkta Flutter karkasas, naudojant Dart programavimo kalbą. Šis pasirinkimas pagrįstas keliais pagrindiniais argumentais:

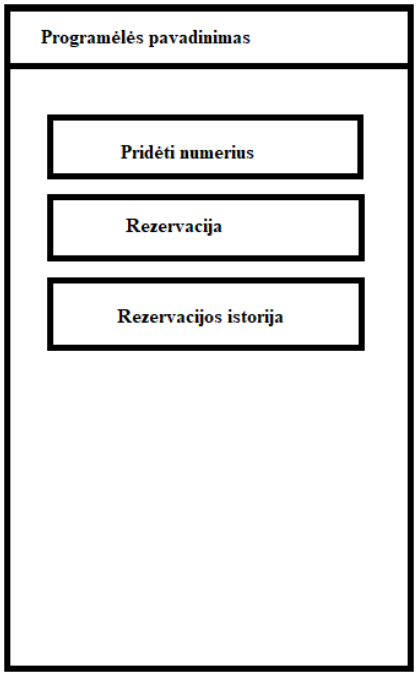
- Platformų nepriklausomumas: Flutter leidžia kurti grafinę sąsają, kuri veikia tiek Android, tiek iOS operacinėse sistemose be papildomo kodo pritaikymo, suteikiant didesnį pasiekiamumą.
- Dinamiškumas ir interaktyvumas: Dart kalba, derinama su Flutter, suteikia greitą ir sklandų vartotojo sąsajos atnaujinimą realiu laiku, kas yra gyvybiškai svarbu dinamiškai keičiantis parkavimo vietų būsenai.
- Platus bendruomenės palaikymas ir ištekliai: Flutter yra viena iš sparčiai augančių karkasų su gausia dokumentacija ir bendruomenės palaikymu, kas palengvina programavimo procesą ir problemų sprendimą.

3.3.2. Langų projektas pagal funkcijas

Kurdami langų projektus, kiekvienas langas yra pritaikytas atitinkti specifines sistemos funkcijas:

- Pagrindinis langas: Rodomas parkavimo aikštelės žemėlapis su žaliomis ir raudonomis zonomis, parodančiomis laisvas ir užimtas vietas.
- Rezervacijos langas: Leidžia vartotojui pasirinkti laiką ir vietą rezervacijai, su galimybe įvesti automobilio registracijos numerį.
- Mano rezervacijos: Rodo vartotojo aktyvias ir būsimas rezervacijas, suteikiant galimybę jas modifikuoti ar atšaukti.

3.3.3. Grafinės naudotojo sąsajos meniu punktų veiksmų projektavimas



3.4 pav. Mobiliosios parkavimo programėlės pagrindinis sąsajos ekranas

(3.4 pav.) matome, kaip turėtų atrodyti programėlės pagrindinis puslapis, skirtas išmaniosios parkavimo sistemos vartotojų sąsajai. Ši programėlė leidžia vartotojams atlikti kelias pagrindines funkcijas:

Pridėti numerius: Ši funkcija leidžia vartotojui įvesti ar atnaujinti transporto priemonės numerio ženklus, kurie yra naudojami parkavimo vietos rezervacijai.

Rezervacija: Per šį mygtuką vartotojas gali greitai rezervuoti parkavimo vietą. Rezervacijos procesas yra automatizuotas ir pritaikytas vartotojo transporto priemonės numeriams, suteikiant greitą ir efektyvią prieigą prie laisvų parkavimo vietų.

Rezervacijos istorija: Šis mygtukas vartotojams suteikia prieigą prie jų ankstesnių rezervacijų istorijos, leidžianti peržiūrėti buvusias rezervacijas ir jų detales, kas yra naudinga siekiant geriau planuoti būsimus apsilankymus.

Šios pagrindinės funkcijos užtikrina, kad vartotojai galėtų efektyviai valdyti savo parkavimo poreikius per mobiliosios programėlės sąsają, taip padedant sumažinti laiko sąnaudas ieškant laisvos parkavimo vietos ir pagerinant bendrą parkavimo patirtį.

3.3.4. Apibendrinimas

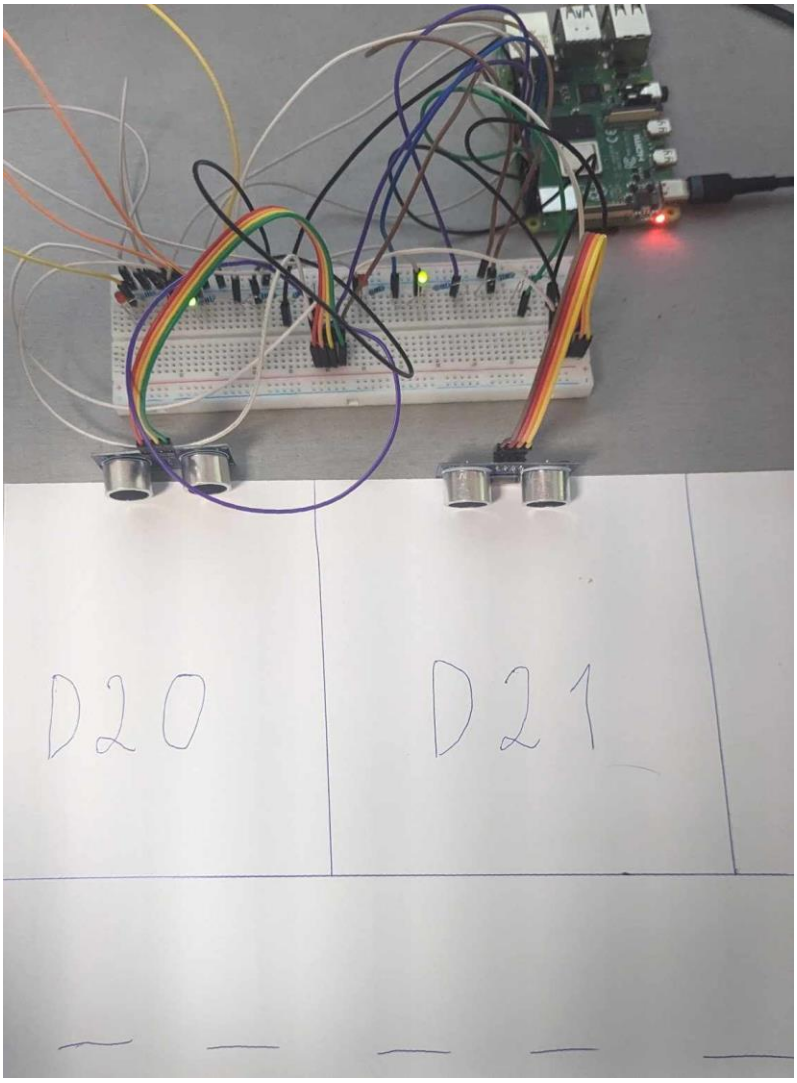
Naudotojo sąsaja yra būtina sistemos dalis, kurianti tiltą tarp techninės sistemos infrastruktūros ir galutinio naudotojo. Jos intuityvumas, greitas atsakas į vartotojo veiksmus ir aiškus

dizainas prisideda prie bendro projekto tikslų – efektyvaus parkavimo valdymo ir vartotojo patirties gerinimo. Pateikiant aiškius ir suprantamus navigacijos elementus, užtikrinamas naudotojo patogumas ir sistemos efektyvumas realioje aplinkoje.

4. EKSPERIMENTINĖ DALIS

4.1. Sukurtas bandomasis pavyzdys

Paveiksle (4.1 pav.) matome dviejų vietų ultragarsinius jutiklius, prijungtus prie „Raspberry Pi“, kuris veikia kaip serveris. Šie jutikliai naudojami nustatyti, ar vieta yra užimta, ar laisva. Kiekviena vieta, pažymėta kaip D20 ir D21, turi LED sistemą, kuri indikuoja vietos būklę: žalia spalva rodo, kad vieta yra laisva, o raudona spalva - kad vieta yra užimta.



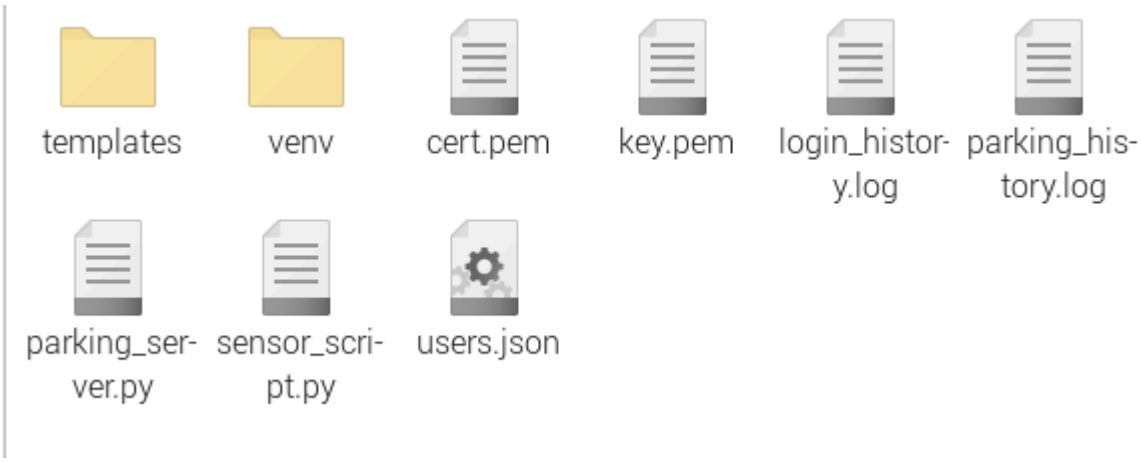
4.1 pav. Aikštelės ekspermantinės dalies vietų ir aparatūros išdėstymas

4.2. Atliktas programavimas

Paveiksle (4.2 pav.) matoma failų struktūra, kuri parodo, kaip veikia visa sistema. Ši sistema susideda iš kelių Python kodų, leidžiančių veikti jutikliams, bei Flask serverio, kuris valdo visą procesą. Čia pateikiami šie svarbūs elementai:

- venv: Virtuali aplinka, kurioje įdiegtos visos priklausomybės, reikalingos projektui.
- cert.pem ir key.pem: Sertifikato ir raktų failai, naudojami HTTPS ryšiui užtikrinti.
- login_history.log: Prisijungimų istorijos failas, kuriame saugomi prisijungimo įrašai.
- parking_history.log: Failas, kuriame saugoma parkavimo istorija.
- parking_server.py: Flask serverio Python failas, kuris valdo serverio funkcijas. (2 Priedas)
- sensor_script.py: Python kodas, atsakingas už jutiklių duomenų nuskaitymą ir apdorojimą. (1 Priedas)
- users.json: Naudotojų duomenų failas, kuriame saugoma informacija apie naudotojus.

Ši failų struktūra leidžia valdyti parkavimo sistemos veikimą, įskaitant duomenų nuskaitymą iš jutiklių, naudotojų autentikaciją ir istorijos saugojimą.



4.2 pav. Sistemos failų struktūra

4.3. Sukurtos navigacijos priemonės

Taip pat, kad naudotis šia sistema, reikia ir vartotojo sąsajos žmonėms. Žemiau esančiuose paveikslėliuose parodytas visas programėlės funkcionalumas.

Paveikslėlyje (4.3 pav.) parodyta prisijungimo sąsaja. Ši sąsaja leidžia vartotojams prisijungti arba prisiregistruoti prie sistemos. Vartotojas įveda savo slapyvardį ir slaptažodį, tada paspaudžia mygtuką "PRISIJUNGTI", norėdamas prisijungti prie savo paskyros. Paspaudus mygtuką

"PRISIREGISTRUOTI", vartotojas nukreipiamas į registracijos puslapį, kur gali sukurti naują paskyrą. Ši sąsaja užtikrina, kad tik registruoti vartotojai gali naudotis sistemos funkcijomis. Veikimo kodas pateiktas 3 priede

4.3 pav. Prisijungimo sąsaja

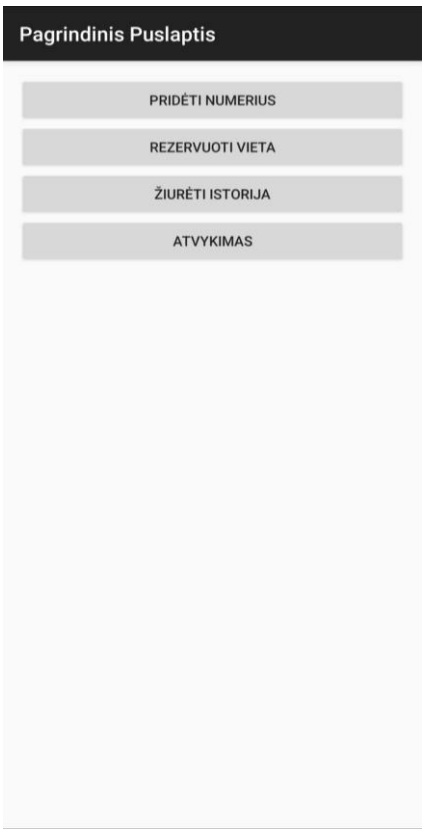
Paveiksle (4.4 pav.) parodyta registracijos sąsaja. Vartotojas įveda norimą slapyvardį ir slaptažodį, tada paspaudžia mygtuką "PRISIREGISTRUOTI", kad sukurtų naują paskyrą. Ši sąsaja leidžia naujiems vartotojams prisiregistruoti sistemoje, kad jie galėtų naudotis visomis jos funkcijomis. Veikimo kodas pateiktas 4 priede

4.4 pav. Registracijos sąsaja

Paveiksle (4.5 pav.) parodytas pagrindinis puslapis, kuris pasiekiamas prisijungus prie sistemos. Šiame puslapyje yra keturi pagrindiniai mygtukai, leidžiantys vartotojams atlikti įvairias funkcijas:

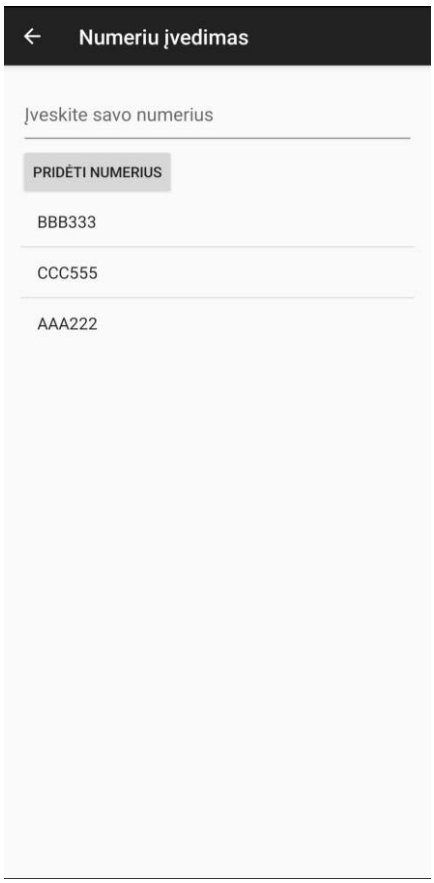
- PRIDĖTI NUMERIUS: Leidžia vartotojui pridėti automobilio numerius į sistemą.
- REZERVUOTI VIETA: Leidžia vartotojui rezervuoti parkavimo vietą.
- ŽIŪRĖTI ISTORIJĄ: Leidžia vartotojui peržiūrėti savo parkavimo istoriją.
- ATVYKIMAS: Leidžia vartotojui pažymėti savo atvykimą į rezervuotą vietą. (Atvykimas buvo sukurtas tik projekto testavimo reikalams, kad nereikėtų statyti kameros, kuri nuskaito numerius, ir kelio užtvaro.)

Ši sąsaja suteikia vartotojams patogų būdą valdyti parkavimo procesus ir sekti savo rezervacijas bei istoriją. Veikimo kodas pateiktas 5 priede



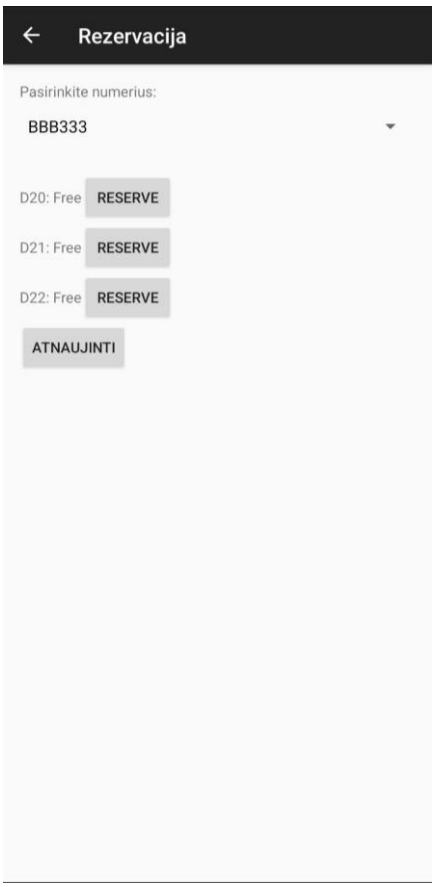
4.5 pav. Pagrindinis puslapis

Paveiksle (4.6 pav.) parodytas numerių įvedimo puslapis. Šiame puslapyje vartotojai gali įvesti savo automobilio numerius ir pridėti juos į sistemą, paspaudę mygtuką "PRIDĖTI NUMERIUS". Jau įvesti numeriai rodomi sąraše žemiau. Norėdami ištrinti numerį, vartotojai gali ilgiau palaikyti paspaudę ant norimo numerio sąraše. Ši funkcija leidžia lengvai valdyti ir atnaujinti įvestus automobilių numerius. Veikimo kodas pateiktas 6 priede



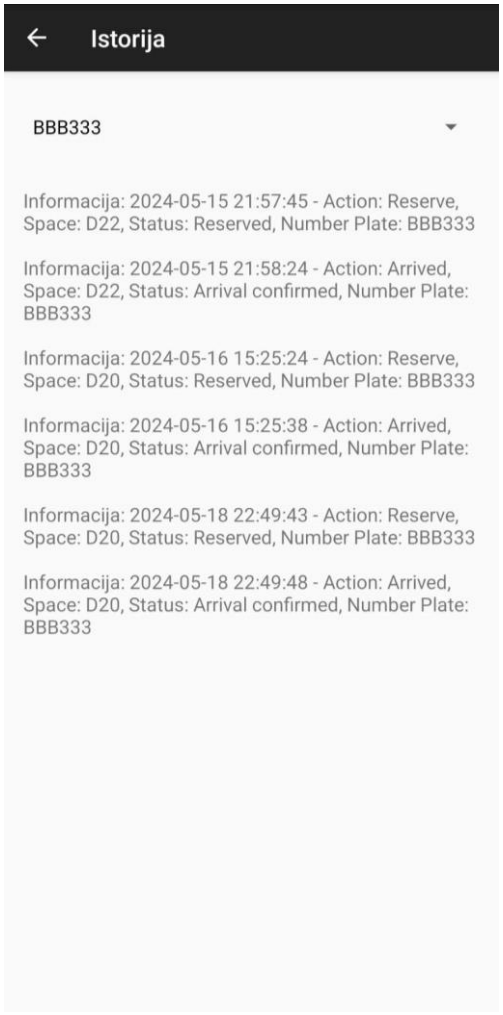
4.6 pav. Numerių įvedimo puslapis

Paveiksle (4.7 pav.) parodytas rezervacijos puslapis. Vartotojas pirmiausia pasirenka numerį iš anksčiau įrašytų numerių sąrašo. Tada vartotojas pasirenka norimą parkavimo vietą iš galimų vietų sąrašo ir spaudžia mygtuką "RESERVE", kad rezervuotų pasirinktą vietą. Mygtukas "ATNAUJINTI" atnaujina puslapį, kad vartotojas matytų naujausią informaciją apie laisvas vietas. Ši funkcija leidžia vartotojams lengvai rezervuoti parkavimo vietas ir užtikrina, kad informacija būtų nuolat atnaujinta. Veikimo kodas pateiktas 7 priede



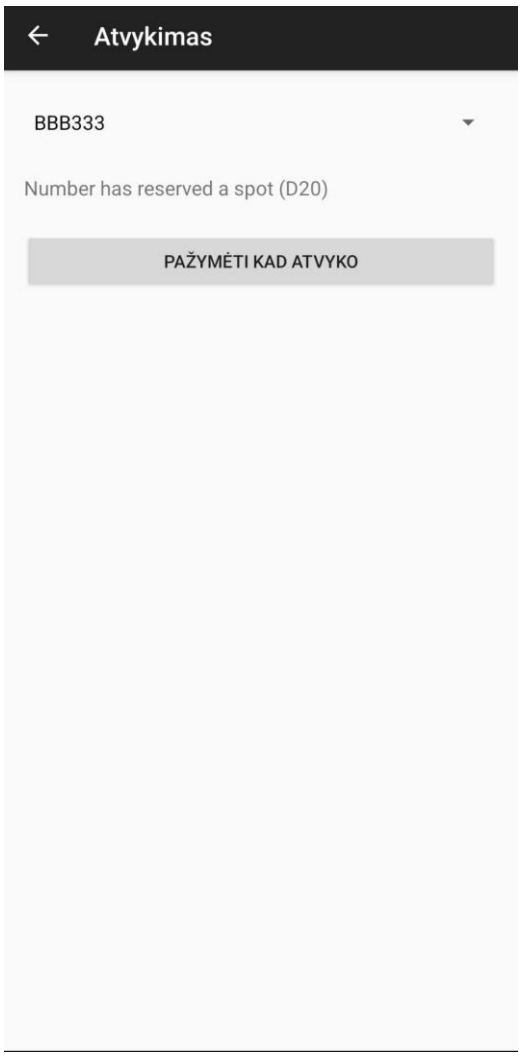
4.7 pav. Rezervacijos puslapis

Paveiksle (4.8 pav.) parodytas istorijos puslapis. Čia matoma informacija apie vartotojo veiksmus, susijusius su parkavimu. Puslapyje rodoma, kada vartotojas rezervavo vietą ir kada atvyko į rezervuotą vietą. Informacija pateikiama su tiksliomis datomis, laikais, veiksmų tipais (rezervacija arba atvykimas), vietos numeriais ir automobilio numeriais. Ši funkcija leidžia vartotojams stebėti savo parkavimo istoriją ir užtikrina skaidrumą bei lengvą prieigą prie ankstesnių veiksmų įrašų. Veikimo kodas pateiktas 8 priede

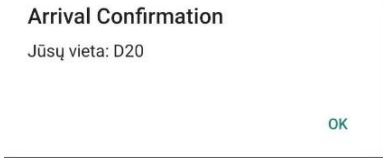


4.8 pav. Parkavimo istorijos puslapis

Paveiksle (4.9 pav.) parodytas atvykimo puslapis. Vartotojas pasirenka automobilio numerį iš sąrašo ir paspaudžia mygtuką "PAŽYMĖTI KAD ATVYKO", kad patvirtintų savo atvykimą į rezervuotą vietą. Tai leidžia sistemai atnaujinti informaciją apie vartotojo buvimo vietą.



4.9 pav. Atvykimo patvirtinimo puslapis



4.10 pav. Atvykimo patvirtinimo žinutė

Paveiksle (4.10 pav.) pateikiama atvykimo patvirtinimo žinutė, kurioje nurodoma rezervuota vieta, pvz., "Jūsų vieta: D20". Paspaudus mygtuką "OK", vartotojui parodoma, kad jo atvykimas sėkmingai užregistruotas. Ši funkcija simuliuoja numerių nuskaitymą ir patvirtina vartotojo rezervuotą vietą. Veikimo kodas pateiktas 9 priede

5. EKONOMINĖ DALIS

Išmaniosios parkavimo sistemos sukūrimas ir įgyvendinimas leidžia efektyviau išnaudoti parkavimo erdvę, sumažinti vairuotojų laiką ieškant laisvos vietos ir optimizuoti parkavimo vietų administravimą. Sistema, automatiškai valdanti parkavimo vietas ir užtikrinanti greitą duomenų apdorojimą, sumažina transporto srautų sukeltą taršą bei padidina vartotojų pasitenkinimą. Ekonominė nauda įmonės kontekste yra matuojama ne tik išvengiant galimų nuostolių dėl neefektyvaus parkavimo valdymo, bet ir palaikant aukštą klientų lojalumą.

Visi skaičiavimai šioje dalyje yra preliminarūs, atsižvelgiant į įmonės numatytas vidaus taisykles ir konfidencialumo politiką.

5.1. Įrangos pirkimas

Pleriminarūs įrangos pirkimo dvidešimties parkavimo vietų aikštelei skaičiavimai (4 lentelė)

4 lentelė. Techninės įrangos pirkimas

Nr.	Pavadinimas ir techninės charakteristikos	Mato vnt.	Kiekis	Kaina	Suma, Eur
1.	Ultragarso parkavimo sensoriai	Vnt.	20	52,35	1047,00
2.	Komercinis serveris (Dell PowerEdge T40, Xeon E-2224G, 16GB RAM, 1TB)	Vnt.	1	850	850,00
3.	Šviesos indikatoriai (raudona/žalia)	Vnt.	20	18,75	375,00
4.	Numerių nuskaitymo kamera	Vnt.	1	285	285,00
5.	Informacinis ekranas	Vnt.	1	315	315,00
6.	Gigabitinis maršrutizatorius	Vnt	1	130	130,00
7.	Laidai ir jungiamoji įranga	Kompl.	1	223	223,00
	Iš viso, eur:				3225,00
	PVM, 21%				677,25
	Bendra suma, eur:				3902,25

5.2. Darbo užmokesčio skaičiavimas

Pleriminarūs skaičiavimai (5 lentelė)

5 lentelė. Darbo laiko nustatymas

Darbų pavyzdžiai	Dirbta dienų
Sistemos planavimas	2
Įrangos pirkimas	1
Montavimas	14
Programavimas	20
Testavimas	7
Sistemos dokumentacija	4
Darbuotojų apmokymai	3
Viso:	51

Valandinio įkainio apskaičiavimas pagal darbuotojo bruto mėnesinį atlyginimą (3000 Eur): 3000 Eur / 21 darbo diena / 8 darbo valandos=17.86 Eur/val.

Bendras darbo užmokestis: 17.86 Eur/val.×408 val. (51 d.)=7286.88 Eur.

+ VSD (1.77%) : 7286.88 + 128.98 = 7415.86 Eur.

5.3. Įrangos projekto palaikymo sąnaudos

Įtraukus mėnesines priežiūros ir techninės palaikymo sąnaudas-300 Eur mėnesiui, 3600 Eur metams.

5.4. Projekto sąmata

Galutiniai skaičiavimai (6 lentelė).

6 lentelė. Projekto sąmata

Nr.	**Pavadinimas**	**Suma, Eur**
1.	Įrangos pirkimas	3902,25
2.	Darbo užmokestis	7415,86
3.	Palaikymo sąnaudos (metams)	3600,00
Iš viso, eur:		14789,13
Administracinės sąnaudos (10%)		1478,90
Viso su administracinėmis sąnaudomis:		16397,02

5.5. Ekonominės naudos nustatymas

Diegiant išmaniąją parkavimo sistemą, įmonės gali sumažinti laiką, skiriamą kliento/darbuotojo parkavimo vietos paieškai, mažinti eismo sukeliamą taršą ir padidinti klientų pasitenkinimą. Visa tai kartu leidžia sumažinti veiklos išlaidas ir padidina klientų lojalumą. Ekonomiškai ši sistema atsipirks per maždaug 18 mėnesių, remiantis operaciniais sutaupymais ir pagerėjusiu veiklos efektyvumu. Atsipirkimas įmanomas dėl sumažintų operacinių išlaidų ir didesnių klientų pasitenkinimo lygių, kurie tiesiogiai prisideda prie pajamų augimo.

IŠVADOS

1. Atlikta išsami esamų parkavimo sistemų analizė, leidusi nustatyti svarbiausius trūkumus ir privalumus. Tai padėjo formuluoti reikalavimus naujajai sistemai, kurioje integruoti ultragarsiniai jutikliai ir mobilioji aplikacija. Ši analizė parodė, kaip technologinės inovacijos gali pagerinti parkavimo valdymą.
2. Sukurta parkavimo vietų stebėjimo ir valdymo sistema, kuri automatizuoja parkavimo procesą naudojant naujausias technologijas. Sistema efektyviai identifikuoja laisvas ir užimtas vietas, suteikdama vairuotojams atnaujinamą informaciją.
3. Išanalizuotas sistemos integravimas į esamą infrastruktūrą, užtikrinant, kad naujoji technologija veiktų su įmonės turimais ištekliais. Integracijos planas apėmė techninius ir logistikos aspektus, užtikrinant sklandų ir efektyvų įgyvendinimą.
4. Išplėtotas algoritmas parkavimo vietų efektyviam paskirstymui, kuris atsižvelgia į įvairius veiksnius, pavyzdžiui, darbuotojų atvykimo laikus. Tai leidžia maksimaliai optimizuoti erdvės naudojimą ir mažinti eismo spūstis parkavimo teritorijose.
5. Sukurta naudotojui draugiška mobilioji aplikacija, kurioje darbuotojai gali lengvai rezervuoti parkavimo vietas. Aplikacija suteikia vartotojams realaus laiko atnaujinimus apie laisvas parkavimo vietas, padidindama pasitenkinimą ir komfortą.
6. Užtikrintas aukštas duomenų saugumo lygis, integruojant šiuolaikines apsaugos technologijas, kurios garantuoja duomenų apsaugą nuo neteisėtos prieigos ir kitų kibernetinių grėsmių. Saugumo sprendimai apima duomenų šifravimą ir saugius ryšius, užtikrinant visišką konfidencialumą ir integritetą.
7. Projektas sėkmingai demonstruoja, kaip iniciatyvių technologijų taikymas parkavimo sektoriuje gali ne tik išspręsti esamas problemas, bet ir ženkliai pagerinti viso proceso efektyvumą. Sukurta sistema ne tik patenkina techninius įmonės reikalavimus, bet ir siūlo ženklią ekonominę naudą, sumažindama operacinius kaštus ir pagerindama klientų aptarnavimo kokybę.
8. Visi projektuoti ir įgyvendinti sprendimai buvo atlikti atsižvelgiant į naujausias rinkos tendencijas ir technologijų vystymąsi, užtikrinant, kad sistema būtų tvari ir lengvai atnaujinama ateičiai.

LITERATŪRA IR KITI INFORMACIJOS ŠALTINIAI

1. Adams, B. (2020). Python programming: Applications and implications for modern software development. Advanced Computing Journal, 37(4), 241-256. https://doi.org/10.1016/j.advcomp.2020.09.005
2. Belbachir, A. N. (Ed.). (2009). Smart Cameras. Springer.
3. Bitė Lietuva. (2023). Kompiuteriai internetu: https://www.bite.lt
4. Bitė Lietuva. (2023). Maršrutizatoriai ir stiprintuvai. : https://www.bite.lt
5. Bogue, R. (2013). Sensors for industrial inspection. Sensor Review, 33(4), 300-305. doi:10.1108/SR-02-2013-653
6. Choo, K. K. R. (2020). The importance of user privacy and data security. IEEE Transactions on Dependable and Secure Computing, 17(1), 176-183.
7. G., & Makkar, U. (2017). Cybersecurity in IoT devices. Journal of Cyber Policy, 5(2), 223-239.
8. Guo, H., & Zhao, J. (2019). Smart Parking Systems and Sensors: A Review. Sensors, 19(15), 3273-3292. doi:10.3390/s19153273
9. IThink. (2023). Jūsų IT partneris, Jūsų infrastruktūrai ir darbo vietoms: https://ithink.lt
10. Jankauskas, L. (2020). Informacijos saugumo pagrindai. Kaunas: Vytauto Didžiojo universitetas.
11. Johnson, L. (2019). "Innovations in Urban Parking Management." City Planning Review, 34(1), 56-67.
12. Kavaliauskas, P. (2019). Raspberry Pi: programavimas ir pritaikymas. Kaunas: Technologija.
13. Martinkus, R. (2021). Web technologijos: Flask naudojimas. Klaipėda: Klaipėdos universitetas.
14. Martínez, L. A., & Rojas, A. F. (2021). Efficiency of IoT-based Parking Systems. International Journal of Urban Planning and Smart Cities, 2(1), 45-58.
15. Misevičius, M. (2021). Java technologijų taikymas. Kaunas: Technologija.
16. Nerijus Kvedaravičius (2024) Išmaniosios stovėjimo aikštelės projektas, T007 Informatikos inžinerija, „Inovacijų taikymas technologijose 2024“ leidinyje
17. Singh, L. (2021). Java in contemporary software development: An overview. Journal of Software Systems, 45(3), 305-320. https://doi.org/10.1097/JSS.0000000000000453
18. Shoup, D. (2006). The High Cost of Free Parking. Planners Press.
19. Smith, J. (2018). "Integrating Sensors and Real-time Data for Parking Solutions." Journal of Transportation Technologies, 8(2), 123-134.
20. Smith, J. (2018). "Time Lost Searching for Parking in Urban Areas," Journal of Urban Planning, 34(2), 123-134.
21. Šileika, A. (2020). Python programavimas: nuo pradžių iki profesionalo. Vilnius: Alma littera.

22. Teltonika Networks. (2023). Key advantages of RutOS: Security: https://teltonika-networks.com/resources/blogs/key-advantages-of-RutOS-security
23. Vaitkus, V. (2021). Tinklo saugumo strategijos ir jų įgyvendinimas. Vilnius: Mokslas ir technika.
24. VU ITPC. (2023). Kompiuterinės darbo vietos: https://www.vu.lt/it/paslaugos/mokymo-ir-mokymosi-istekliai/kompiuterines-darbo-vietos
25. Zhao, J., & Guo, H. (2019). Smart Parking Systems and Sensors: A Review. Sensors Journal, 19(15), 3273-3292.

PRIEDAI

1 priedas.

Sensoriaus Kodas

```
import requests
import urllib3
from gpiozero import DistanceSensor, LED
from gpiozero.pins.pigpio import PiGPIOFactory
from time import sleep

# Disable SSL warnings for self-signed certificates (not recommended for production)
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# Ensure the pigpio daemon is running
try:
    factory = PiGPIOFactory(host='localhost')
except OSError as e:
    print(f"Error: {e}. Make sure the pigpio daemon is running (sudo pigpiod).")
    exit(1)

# Sensor for space D20
sensor1 = DistanceSensor(echo=17, trigger=4, pin_factory=factory)
led_taken1 = LED(24, pin_factory=factory)  # Red LED for taken/reserved
led_free1 = LED(23, pin_factory=factory)   # Green LED for free

# Sensor for space D21
sensor2 = DistanceSensor(echo=16, trigger=25, pin_factory=factory)
led_taken2 = LED(6, pin_factory=factory)  # Red LED for taken/reserved
led_free2 = LED(5, pin_factory=factory)   # Green LED for free

SERVER_URL = 'https://192.168.1.151:5000/parking/spaces'

def check_reservation_status():
    try:
        response = requests.get(SERVER_URL, verify=False)
        if response.status_code == 200:
            parking_spaces = response.json()['parking_spaces']
            status_d20 = parking_spaces['D20']['reserved']
            status_d21 = parking_spaces['D21']['reserved']
            return status_d20, status_d21
        else:
            print('Failed to retrieve parking status')
            return False, False
    except requests.exceptions.RequestException as e:
        print(f'Error fetching parking statuses: {e}')
        return False, False

def update_server_status(space, status):
    payload = {'space': space, 'status': status}
    try:
        response = requests.post('https://192.168.1.151:5000/parking/update', json=payload, verify=False)
        if response.status_code == 200:
            print(response.json()['message'])
        else:
            print('Failed to update server status')
    except requests.exceptions.RequestException as e:
        print(f'Error: {e}')

try:
    while True:
        # Fetch reservation status from the server
```

```
        reserved_status_d20, reserved_status_d21 = check_reservation_status()

        # Check distances and reservation status for D20
        distance_d20 = sensor1.distance * 100
        if reserved_status_d20:
            led_taken1.on()
            led_free1.off()
        elif distance_d20 < 10:
            update_server_status('D20', 'taken')
            led_taken1.on()
            led_free1.off()
        else:
            update_server_status('D20', 'free')
            led_taken1.off()
            led_free1.on()

        # Check distances and reservation status for D21
        distance_d21 = sensor2.distance * 100
        if reserved_status_d21:
            led_taken2.on()
            led_free2.off()
        elif distance_d21 < 10:
            update_server_status('D21', 'taken')
            led_taken2.on()
            led_free2.off()
        else:
            update_server_status('D21', 'free')
            led_taken2.off()
            led_free2.on()

        sleep(1)

except KeyboardInterrupt:
    led_taken1.off()
    led_free1.off()
    led_taken2.off()
    led_free2.off()
    print("Measurement stopped by User")
```

2 priedas.

Serverio Kodas

```
from flask import Flask, jsonify, request, render_template, session
import datetime
from datetime import timedelta
import json
import bcrypt

app = Flask(__name__)
app.secret_key = 'Asd123'

USER_DATA_FILE = 'users.json'
LOG_FILE = 'login_history.log'
PARKING_LOG_FILE = 'parking_history.log'

# Initialize parking space status and last update time with reservation state
parking_spaces = {
    'D20': {'status': False, 'reserved': False, 'number_plate': None},
    'D21': {'status': True, 'reserved': False, 'number_plate': None},
    'D22': {'status': True, 'reserved': False, 'number_plate': None}
}
last_update_time = {space: datetime.datetime.now() for space in parking_spaces}

def load_users():
    try:
        with open(USER_DATA_FILE, 'r') as file:
            return json.load(file)
    except (FileNotFoundError, json.JSONDecodeError):
        return {}

def save_users(users):
    with open(USER_DATA_FILE, 'w') as file:
        json.dump(users, file, indent=4)

def log_activity(action, username):
    with open(LOG_FILE, 'a') as file:
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        file.write(f"{timestamp} - Action: {action}, User: {username}\n")

def log_parking_history(action, space, status, number_plate):
    with open(PARKING_LOG_FILE, 'a') as file:
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        file.write(f"{timestamp} - Action: {action}, Space: {space}, Status: {status}, Number Plate:
{number_plate}\n")

@app.route('/')
def index():
    for space, info in parking_spaces.items():
        if (datetime.datetime.now() - last_update_time[space]) > timedelta(seconds=5) and not info['reserved']:
            parking_spaces[space]['status'] = False
    return render_template('index.html', parking_spaces=parking_spaces)

@app.route('/register', methods=['POST'])
def register():
    username = request.form.get('username')
    password = request.form.get('password')

    if not username or not password:
        return jsonify({'success': False, 'message': 'Username and password are required.'}), 400
```

```
    hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

    users = load_users()

    if username in users:
        return jsonify({'success': False, 'message': 'Username already exists.'}), 400

    users[username] = hashed_password.decode('utf-8')
    save_users(users)

    return jsonify({'success': True, 'message': 'User registered successfully.'})

@app.route('/login', methods=['POST'])
def login():
    username = request.form.get('username')
    password = request.form.get('password')

    if not username or not password:
        return jsonify({'success': False, 'message': 'Username and password are required.'}), 400

    users = load_users()

    if username not in users or not bcrypt.checkpw(password.encode('utf-8'), users[username].encode('utf-8')):
        return jsonify({'success': False, 'message': 'Invalid username or password.'}), 400

    session['user_id'] = username
    log_activity('Login', username)
    return jsonify({'success': True, 'message': 'Login successful.'})

@app.route('/logout', methods=['POST'])
def logout():
    username = session.pop('user_id', None)
    if username:
        log_activity('Logout', username)
        return jsonify({'message': 'Logout successful'})
    return jsonify({'error': 'Not logged in'}), 400

@app.route('/login_history', methods=['GET'])
def login_history():
    try:
        with open(LOG_FILE, 'r') as file:
            history = file.readlines()
        return jsonify({'history': history})
    except FileNotFoundError:
        return jsonify({'error': 'Log file not found'}), 404

@app.route('/checkReservation')
def check_reservation():
    number_plate = request.args.get('numberPlate')
    if not number_plate:
        return jsonify({'error': 'numberPlate parameter is required'}), 400

    for space, details in parking_spaces.items():
        if details.get('number_plate') == number_plate and details['reserved']:
            return jsonify({'hasReservation': True, 'space': space})
    return jsonify({'hasReservation': False})

@app.route('/reserve', methods=['POST'])
def reserve():
    space = request.form.get('space')
    number_plate = request.form.get('number_plate')
    if space in parking_spaces and not parking_spaces[space]['reserved']:
```

```
            parking_spaces[space]['reserved'] = True
            parking_spaces[space]['status'] = True
            parking_spaces[space]['number_plate'] = number_plate
            log_parking_history('Reserve', space, 'Reserved', number_plate)
            return jsonify({'message': 'Reservation made successfully', 'status': 'reserved'})
    return jsonify({'error': 'Invalid parking space or already reserved'})

@app.route('/arrived', methods=['POST'])
def arrived():
    number_plate = request.form.get('number_plate')
    if not number_plate:
        return jsonify({'error': 'number_plate parameter is required'}), 400

    for space, details in parking_spaces.items():
        if details.get('number_plate') == number_plate:
            details['reserved'] = False
            details['number_plate'] = None
            log_parking_history('Arrived', space, 'Arrival confirmed', number_plate)
            return jsonify({'message': 'Arrival confirmed.', 'space': space})
    return jsonify({'error': 'No reservation found for this number plate'})

@app.route('/parking/spaces', methods=['GET'])
def get_parking_spaces():
    return jsonify({'parking_spaces': parking_spaces})

@app.route('/parking/update', methods=['POST'])
def update_parking_space():
    data = request.json
    space = data.get('space')
    status = data.get('status', False)
    if space in parking_spaces and not parking_spaces[space]['reserved']:
        parking_spaces[space]['status'] = status
        last_update_time[space] = datetime.datetime.now()
        return jsonify({'message': f'Space {space} updated successfully', 'status': status})
    return jsonify({'error': f'Space {space} does not exist'})

@app.route('/getHistory', methods=['GET'])
def get_history():
    number_plate = request.args.get('numberPlate')
    if not number_plate:
        return jsonify({'error': 'numberPlate parameter is required'}), 400

    history_entries = []
    try:
        with open(PARKING_LOG_FILE, 'r') as file:
            for line in file:
                if number_plate in line:
                    history_entries.append(line.strip())
    except FileNotFoundError:
        return jsonify({'error': 'Log file not found'}), 404

    if history_entries:
        return jsonify({'history': history_entries})
    else:
        return jsonify({'error': 'No history found for this number plate'}), 404

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', ssl_context=('/home/LogNeris/Desktop/bakis/cert.pem',
'/home/LogNeris/Desktop/bakis/key.pem'))
```

3 priedas.

Prisijungimo veikimo Kodas

```
package com.example.bakis2;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import com.example.bakis2.network.ApiService;
import com.example.bakis2.network.NetworkService;

import okhttp3.ResponseBody;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class LoginActivity extends AppCompatActivity {

    private EditText editTextUsername, editTextPassword;
    private Button buttonLogin, buttonRegister;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        editTextUsername = findViewById(R.id.editTextUsername);
        editTextPassword = findViewById(R.id.editTextPassword);
        buttonLogin = findViewById(R.id.buttonLogin);
        buttonRegister = findViewById(R.id.buttonRegister);

        buttonLogin.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                loginUser();
            }
        });

        buttonRegister.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(LoginActivity.this, RegisterActivity.class);
                startActivity(intent);
            }
        });
    }

    private void loginUser() {
        String username = editTextUsername.getText().toString();
        String password = editTextPassword.getText().toString();

        if (username.isEmpty() || password.isEmpty()) {
            Toast.makeText(this, "Username and password are required", Toast.LENGTH_SHORT).show();
```

```
            return;
        }

        ApiService apiService = NetworkService.getRetrofitInstance().create(ApiService.class);
        Call<ResponseBody> call = apiService.loginUser(username, password);

        call.enqueue(new Callback<ResponseBody>() {
            @Override
            public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
                if (response.isSuccessful()) {
                    Toast.makeText(LoginActivity.this, "Login successful", Toast.LENGTH_SHORT).show();
                    // Save login state
                    getSharedPreferences("login", MODE_PRIVATE).edit().putBoolean("logged_in", true).apply();
                    Intent intent = new Intent(LoginActivity.this, MainActivity.class);
                    startActivity(intent);
                    finish();
                } else {
                    Toast.makeText(LoginActivity.this, "Login failed: " + response.message(),
Toast.LENGTH_SHORT).show();
                }
            }

            @Override
            public void onFailure(Call<ResponseBody> call, Throwable t) {
                Toast.makeText(LoginActivity.this, "Network error: " + t.getMessage(),
Toast.LENGTH_SHORT).show();
                Log.e("LoginError", "onFailure: ", t);
            }
        });
    }
}
```

4 priedas.

Registravimo veikimo Kodas

```
package com.example.bakis2;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import com.example.bakis2.network.ApiService;
import com.example.bakis2.network.NetworkService;

import okhttp3.ResponseBody;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class RegisterActivity extends AppCompatActivity {

    private EditText editTextUsername, editTextPassword;
    private Button buttonRegister;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_register);

        editTextUsername = findViewById(R.id.editTextUsername);
        editTextPassword = findViewById(R.id.editTextPassword);
        buttonRegister = findViewById(R.id.buttonRegister);

        buttonRegister.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                registerUser();
            }
        });
    }

    private void registerUser() {
        String username = editTextUsername.getText().toString();
        String password = editTextPassword.getText().toString();

        if (username.isEmpty() || password.isEmpty()) {
            Toast.makeText(this, "Username and password are required", Toast.LENGTH_SHORT).show();
            return;
        }

        ApiService apiService = NetworkService.getRetrofitInstance().create(ApiService.class);
        Call<ResponseBody> call = apiService.registerUser(username, password);

        call.enqueue(new Callback<ResponseBody>() {
            @Override
            public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
```

```
            if (response.isSuccessful()) {
                Toast.makeText(RegisterActivity.this, "Registration successful",
Toast.LENGTH_SHORT).show();
                Intent intent = new Intent(RegisterActivity.this, LoginActivity.class);
                startActivity(intent);
                finish();
            } else {
                Toast.makeText(RegisterActivity.this, "Registration failed: " + response.message(),
Toast.LENGTH_SHORT).show();
            }
        }

        @Override
        public void onFailure(Call<ResponseBody> call, Throwable t) {
            Toast.makeText(RegisterActivity.this, "Network error: " + t.getMessage(),
Toast.LENGTH_SHORT).show();
            Log.e("RegisterError", "onFailure: ", t);
        }
    });
  }
}
```

5 priedas.

Pagrindinio puslapio kodas

```
package com.example.bakis2;

import android.annotation.SuppressLint;
import android.os.Bundle;
import androidx.fragment.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import androidx.navigation.Navigation;

public class HomeFragment extends Fragment {
    private Button btnAddCar, btnReserveSpot, btnViewHistory, btnArrived;

    @SuppressLint("MissingInflatedId")
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_home, container, false);
        btnAddCar = view.findViewById(R.id.btnAddCar);
        btnReserveSpot = view.findViewById(R.id.btnReserveSpot);
        btnViewHistory = view.findViewById(R.id.btnViewHistory);
        btnArrived = view.findViewById(R.id.btnArrived);


        // Navigate to the corresponding fragments when buttons are clicked
        btnAddCar.setOnClickListener(v ->
Navigation.findNavController(v).navigate(R.id.action_homeFragment_to_reserveNumberPlatesFragment));
        btnReserveSpot.setOnClickListener(v ->
Navigation.findNavController(v).navigate(R.id.action_homeFragment_to_reservationFragment));
        btnViewHistory.setOnClickListener(v ->
Navigation.findNavController(v).navigate(R.id.action_homeFragment_to_historyFragment));
        btnArrived.setOnClickListener(v ->
Navigation.findNavController(v).navigate(R.id.action_homeFragment_to_arrived));


        return view;
    }
}
```

6 priedas.

Numeriu rezervavimo veikimo kodas

```
package com.example.bakis2;

import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import androidx.fragment.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;

public class ReserveNumberPlatesFragment extends Fragment {
    private EditText editTextCarNumber;
    private Button buttonAddCar;
    private ListView listViewCarNumbers;
    private ArrayList<String> carNumberList = new ArrayList<>();
    private ArrayAdapter<String> adapter;
    private SharedPreferences sharedPreferences;

    @Override
    public void onResume() {
        super.onResume();
        loadCarNumbers(); // Load numbers every time the fragment resumes
    }

    private void loadCarNumbers() {
        SharedPreferences sharedPreferences = getActivity().getSharedPreferences("CarInfo",
Context.MODE_PRIVATE);
        Set<String> carNumbersSet = sharedPreferences.getStringSet("carNumbers", new HashSet<>());
        carNumberList.clear();
        carNumberList.addAll(carNumbersSet);
        adapter.notifyDataSetChanged(); // Notify the adapter to refresh the list view
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_reserve_number_plates, container, false);
        editTextCarNumber = view.findViewById(R.id.editTextCarNumber);
        buttonAddCar = view.findViewById(R.id.buttonAddCar);
        listViewCarNumbers = view.findViewById(R.id.listViewCarNumbers);

        sharedPreferences = getActivity().getPreferences(Context.MODE_PRIVATE);
        carNumberList.addAll(sharedPreferences.getStringSet("carNumbers", new HashSet<>()));
        adapter = new ArrayAdapter<>(getActivity(), android.R.layout.simple_list_item_1, carNumberList);
        listViewCarNumbers.setAdapter(adapter);

        buttonAddCar.setOnClickListener(v -> {
```

```
            String carNumber = editTextCarNumber.getText().toString().toUpperCase(); // Convert to upper case
            if (isValidCarNumber(carNumber)) {
                if (!carNumberList.contains(carNumber)) {
                    carNumberList.add(carNumber);
                    adapter.notifyDataSetChanged();
                    saveCarNumbers();
                    editTextCarNumber.setText(""); // Clear the input field
                } else {
                    Toast.makeText(getContext(), "Car number already added.", Toast.LENGTH_SHORT).show();
                }
            } else {
                Toast.makeText(getContext(), "Invalid car number. Format should be ABC123.",
Toast.LENGTH_LONG).show();
            }
        });

        // Set up a long click listener to delete car numbers
        listViewCarNumbers.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
            @Override
            public boolean onItemLongClick(AdapterView<?> parent, View view, int position, long id) {
                String itemToRemove = carNumberList.get(position);
                carNumberList.remove(position);
                adapter.notifyDataSetChanged();
                saveCarNumbers(); // Update SharedPreferences after deletion
                Toast.makeText(getContext(), "Removed: " + itemToRemove, Toast.LENGTH_SHORT).show();
                return true; // Indicate that the click was handled
            }
        });

        return view;
    }

    private boolean isValidCarNumber(String carNumber) {
        return carNumber.matches("[A-Z]{3}[0-9]{3}");
    }

    private void saveCarNumbers() {
        SharedPreferences sharedPreferences = getActivity().getSharedPreferences("CarInfo",
Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        Set<String> carNumbersSet = new HashSet<>(carNumberList); // Convert ArrayList to Set for
SharedPreferences
        editor.putStringSet("carNumbers", carNumbersSet);
        editor.apply();
    }


}
```

7 priedas.

Vietų rezervavimo veikimo kodas

```
package com.example.bakis2;

import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.Spinner;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

import com.example.bakis2.models.ParkingSpot;
import com.example.bakis2.models.ParkingResponse;
import com.example.bakis2.network.ApiService;
import com.example.bakis2.network.NetworkService;

public class ReservationFragment extends Fragment {
    private Spinner spinnerNumberPlates;
    private LinearLayout layoutParkingSpots;
    private ArrayAdapter<String> numberPlateAdapter;
    private List<String> numberPlates = new ArrayList<>();
    private TextView statusTextView;
    private Button refreshButton;

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_reservation, container, false);
        spinnerNumberPlates = view.findViewById(R.id.spinnerNumberPlates);
        layoutParkingSpots = view.findViewById(R.id.layoutParkingSpots);
        statusTextView = view.findViewById(R.id.statusTextView);
        refreshButton = view.findViewById(R.id.refreshButton);

        // Initialize the adapter first before loading numbers
        numberPlates = new ArrayList<>();
        numberPlateAdapter = new ArrayAdapter<>(getContext(),
android.R.layout.simple_spinner_dropdown_item, numberPlates);
        spinnerNumberPlates.setAdapter(numberPlateAdapter);

        // Load numbers and setup UI
        loadCarNumbers();
        refreshButton.setOnClickListener(v -> fetchParkingStatus());
```

```
        fetchParkingStatus();

        return view;
    }

    private void loadCarNumbers() {
        SharedPreferences sharedPreferences = getActivity().getSharedPreferences("CarInfo",
Context.MODE_PRIVATE);
        Set<String> carNumbersSet = sharedPreferences.getStringSet("carNumbers", new HashSet<>());
        numberPlates.clear();
        numberPlates.addAll(carNumbersSet);
        numberPlateAdapter.notifyDataSetChanged();
    }

    private void fetchParkingStatus() {
        ApiService apiService = NetworkService.getRetrofitInstance().create(ApiService.class);
        apiService.getParkingSpaces().enqueue(new Callback<ParkingResponse>() {
            @Override
            public void onResponse(Call<ParkingResponse> call, Response<ParkingResponse> response) {
                if (response.isSuccessful() && response.body() != null) {
                    updateUIWithParkingSpots(response.body().getParkingSpaces());
                } else {
                    statusTextView.setText("Error: " + response.code() + " " + response.message());
                }
            }

            @Override
            public void onFailure(Call<ParkingResponse> call, Throwable t) {
                statusTextView.setText("Failed to fetch status: " + t.getMessage());
            }
        });
    }

    private void updateUIWithParkingSpots(Map<String, ParkingSpot> parkingSpots) {
        layoutParkingSpots.removeAllViews();
        for (Map.Entry<String, ParkingSpot> entry : parkingSpots.entrySet()) {
            LinearLayout spotLayout = new LinearLayout(getContext());
            spotLayout.setOrientation(LinearLayout.HORIZONTAL);

            TextView spotText = new TextView(getContext());
            spotText.setText(entry.getKey() + ": " + (entry.getValue().isTaken() ? "Taken" : "Free"));
            spotLayout.addView(spotText);

            Button reserveButton = new Button(getContext());
            reserveButton.setText("Reserve");
            reserveButton.setEnabled(!entry.getValue().isTaken());
            reserveButton.setOnClickListener(v -> reserveSpot(entry.getKey()));
            spotLayout.addView(reserveButton);

            layoutParkingSpots.addView(spotLayout);
        }
    }

    private void reserveSpot(String spaceId) {
        String selectedNumberPlate = spinnerNumberPlates.getSelectedItem().toString();
        ApiService apiService = NetworkService.getRetrofitInstance().create(ApiService.class);
        apiService.reserveSpace(spaceId, selectedNumberPlate).enqueue(new Callback<Void>() {
            @Override
            public void onResponse(Call<Void> call, Response<Void> response) {
                if (response.isSuccessful()) {
                    fetchParkingStatus();
                } else {
                    statusTextView.setText("Failed to reserve: " + response.code());
```

```
            }
        }

        @Override
        public void onFailure(Call<Void> call, Throwable t) {
            statusTextView.setText("Reservation failed: " + t.getMessage());
        }
    });
  }
}
```

8 priedas.

Rezervavimo istorijos veikimo kodas

```
package com.example.bakis2;

import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;

import com.example.bakis2.models.HistoryResponse;
import com.example.bakis2.network.ApiService;
import com.example.bakis2.network.NetworkService;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class HistoryFragment extends Fragment {
    private Spinner spinnerCarNumbers;
    private TextView historyTextView;
    private ArrayList<String> carNumbersList = new ArrayList<>();
    private ArrayAdapter<String> carNumberAdapter;

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_history, container, false);
        spinnerCarNumbers = view.findViewById(R.id.spinnerCarNumbers);
        historyTextView = view.findViewById(R.id.historyTextView);

        if (spinnerCarNumbers == null || historyTextView == null) {
            Toast.makeText(getContext(), "UI components not properly initialized",
Toast.LENGTH_SHORT).show();
            return view;
        }

        loadCarNumbers(); // This will set up the spinner
        return view;
    }

    private void loadCarNumbers() {
        Context context = getContext();
        if (context == null) {
```

```
            Toast.makeText(getContext(), "Context is null", Toast.LENGTH_SHORT).show();
            return;
        }
        SharedPreferences sharedPreferences = context.getSharedPreferences("CarInfo",
Context.MODE_PRIVATE);
        Set<String> carNumbersSet = sharedPreferences.getStringSet("carNumbers", new HashSet<>());
        carNumbersList = new ArrayList<>(carNumbersSet);

        if (carNumbersList.isEmpty()) {
            Toast.makeText(context, "No car numbers found", Toast.LENGTH_SHORT).show();
        } else {
            carNumberAdapter = new ArrayAdapter<>(context,
android.R.layout.simple_spinner_dropdown_item, carNumbersList);
            spinnerCarNumbers.setAdapter(carNumberAdapter);

            spinnerCarNumbers.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
                @Override
                public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
                    String selectedNumberPlate = carNumbersList.get(position);
                    fetchHistory(selectedNumberPlate);
                }

                @Override
                public void onNothingSelected(AdapterView<?> parent) {
                    historyTextView.setText("");
                }
            });
        }
    }

    private void fetchHistory(String numberPlate) {
        ApiService apiService = NetworkService.getRetrofitInstance().create(ApiService.class);
        Call<HistoryResponse> call = apiService.getHistory(numberPlate);
        call.enqueue(new Callback<HistoryResponse>() {
            @Override
            public void onResponse(Call<HistoryResponse> call, Response<HistoryResponse> response) {
                if (response.isSuccessful() && response.body() != null) {
                    List<String> historyEntries = response.body().getHistory();
                    StringBuilder formattedHistory = new StringBuilder();
                    for (String entry : historyEntries) {
                        formattedHistory.append(formatEntry(entry)).append("\n\n");
                    }
                    historyTextView.setText(formattedHistory.toString());
                } else {
                    Toast.makeText(getContext(), "Error fetching history: " + response.code() + " " +
response.message(), Toast.LENGTH_LONG).show();
                    historyTextView.setText("No history available.");
                }
            }

            @Override
            public void onFailure(Call<HistoryResponse> call, Throwable t) {
                Toast.makeText(getContext(), "Network error: " + t.getMessage(), Toast.LENGTH_LONG).show();
                historyTextView.setText("Failed to fetch history.");
            }
        });
    }

    private String formatEntry(String entry) {
        String[] parts = entry.split(", ");
        if (parts.length < 5) {
            return "Informacija: " + entry;
        }
```

```
        String timestamp = parts[0].split(" - ")[0];
        String action = parts[1].split(": ")[1];
        String space = parts[2].split(": ")[1];
        String status = parts[3].split(": ")[1];
        String numberPlate = parts[4].split(": ")[1];

        return "Time: " + timestamp + "\nSpace: " + space + "\nCar number: " + numberPlate + "\nAction: " +
action + "\nStatus: " + status;
    }
}
```

9 priedas.

Atvykimo veikimo kodas

```
package com.example.bakis2;

import android.app.AlertDialog;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

import com.example.bakis2.models.ReservationStatusResponse;
import com.example.bakis2.network.ApiService;
import com.example.bakis2.network.NetworkService;

public class ArrivedFragment extends Fragment {
    private Spinner spinnerNumberPlates;
    private TextView textViewReservationStatus;
    private Button buttonArrived;
    private ArrayAdapter<String> numberPlateAdapter;
    private ArrayList<String> numberPlates = new ArrayList<>();

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_arrived, container, false);
        spinnerNumberPlates = view.findViewById(R.id.spinnerNumberPlates);
        textViewReservationStatus = view.findViewById(R.id.textViewReservationStatus);
        buttonArrived = view.findViewById(R.id.buttonArrived);

        // Initialize the adapter and spinner
        numberPlateAdapter = new ArrayAdapter<>(getContext(),
android.R.layout.simple_spinner_dropdown_item, numberPlates);
        spinnerNumberPlates.setAdapter(numberPlateAdapter);

        // Load the number plates from shared preferences
        loadNumberPlates();

        spinnerNumberPlates.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
                String selectedNumberPlate = numberPlates.get(position);
                checkReservationStatus(selectedNumberPlate);
            }
```

```
        @Override
        public void onNothingSelected(AdapterView<?> parent) {}
    });

    buttonArrived.setOnClickListener(v -> {
        String selectedNumberPlate = (String) spinnerNumberPlates.getSelectedItem();
        if (selectedNumberPlate != null) {
            markAsArrived(selectedNumberPlate);
        }
    });

    return view;
}

private void loadNumberPlates() {
    SharedPreferences sharedPreferences = getActivity().getSharedPreferences("CarInfo",
Context.MODE_PRIVATE);
    Set<String> carNumbersSet = sharedPreferences.getStringSet("carNumbers", new HashSet<>());
    numberPlates.clear();
    numberPlates.addAll(carNumbersSet);
    numberPlateAdapter.notifyDataSetChanged();
}

private void checkReservationStatus(String numberPlate) {
    ApiService apiService = NetworkService.getRetrofitInstance().create(ApiService.class);
    Call<ReservationStatusResponse> call = apiService.checkReservation(numberPlate);
    call.enqueue(new Callback<ReservationStatusResponse>() {
        @Override
        public void onResponse(Call<ReservationStatusResponse> call,
Response<ReservationStatusResponse> response) {
            if (response.isSuccessful() && response.body() != null) {
                ReservationStatusResponse res = response.body();
                String displayText = res.hasReservation() ?
                        getString(R.string.reserved_space, res.getSpace()) :
                        getString(R.string.not_reserved);
                textViewReservationStatus.setText(displayText);
            } else {
                textViewReservationStatus.setText(getString(R.string.error_checking_status));
            }
        }

        @Override
        public void onFailure(Call<ReservationStatusResponse> call, Throwable t) {
            textViewReservationStatus.setText(getString(R.string.failed_fetch_status));
        }
    });
}

private void markAsArrived(String numberPlate) {
    ApiService apiService = NetworkService.getRetrofitInstance().create(ApiService.class);
    Call<ReservationStatusResponse> call = apiService.markArrived(numberPlate);
    call.enqueue(new Callback<ReservationStatusResponse>() {
        @Override
        public void onResponse(Call<ReservationStatusResponse> call,
Response<ReservationStatusResponse> response) {
            if (response.isSuccessful() && response.body() != null) {
                ReservationStatusResponse res = response.body();
                String message = res.getSpace() != null ?
                        "Jūsų vieta: " + res.getSpace() :
                        "Jūsų pasisirnkti numeriai neturi rezervacijos.";
                showAlert("Arrival Confirmation", message);
            } else {
                showAlert("Error", "Failed to mark arrival: " + response.message());
```

```
            }
        }

        @Override
        public void onFailure(Call<ReservationStatusResponse> call, Throwable t) {
            showAlert("Error", "Network error: " + t.getMessage());
        }
    });
}

private void showAlert(String title, String message) {
    new AlertDialog.Builder(getContext())
            .setTitle(title)
            .setMessage(message)
            .setPositiveButton("OK", null)
            .show();
}

private void showArrivalDialog(String space) {
    new AlertDialog.Builder(getContext())
            .setTitle("Rezervacija patvirtinta")
            .setMessage("You are assigned to space: " + space + ". Confirm arrival?")
            .setPositiveButton("Confirm", (dialog, which) -> confirmArrival(space))
            .setNegativeButton("Cancel", (dialog, which) -> dialog.dismiss())
            .show();
}

private void confirmArrival(String space) {

    getActivity().runOnUiThread(() -> {
        if (textViewReservationStatus != null) {
            textViewReservationStatus.setText("Confirmed at space: " + space);
        }
    });

}




}
```