Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

Автоматизована система паркування автомобілів в межах розумного будинку

Текстова частина до дипломної роботи

за спеціальністю “Комп’ютерні науки”

Керівник дипломної роботи кандидат фізико-математичних наук, доцент

Нагірна А.М.

_____ (підпис)

“____” _____ 2025 р.

Виконав студент

КН-4 Челомбітько Д.С

“____” _____ 2025 р.

Київ 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Доктор технічних наук, доцент.

_____А. М. Глибовець

(підпис)

“____”_____2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломну роботу

студенту Челомбітько Д.С. факультету інформатики 4 курсу

Розробити: Автоматизована система паркування автомобілів в межах розумного будинку

Зміст ТЧ до бакалаврської роботи:

Зміст

Анотація

Вступ

Аналіз існуючих рішень

Проєктування серверної частини

Тестування і демонстрація роботи системи

Перспективи розвитку додатку

Висновки

Використана література

Додатки

Дата видачі “____” _____ 2025 р. Керівник _____ (підпис)

Завдання отримав _____ (підпис)

Зміст

Анотація.....4
ВСТУП.....5
Обґрунтування актуальності теми.....6
Мета та завдання дослідження.....7
РОЗДІЛ 1. Аналіз існуючих рішень.....9
1.1 Огляд популярних застосунків для паркування.....9
1.2 Недоліки та обмеження конкурентів.....11
1.3 Порівняння функціональності з розроблюваним застосунком.....11
РОЗДІЛ 2. Проєктування серверної частини.....13
2.1 Вибір мови програмування та фреймворку.....13
2.2 Архітектура RESTful API та структура серверного застосунку.....14
2.3 Структура бази даних та інструменти міграції.....14
2.4 Система автентифікації та підтвердження дій користувача.....16
2.5 Взаємодія із зовнішнім ANPR-сервісом через API.....17
РОЗДІЛ 3. Розробка мобільного застосунку.....19
3.1 Обґрунтування вибору мови Swift.....19
3.2 Архітектура застосунку: MVVM та SwiftUI.....20
3.3 Взаємодія з API (авторизація, робота з авто, гостями та воротами).....22
3.4 Реалізація інтерфейсу користувача відповідно до Apple HIG.....24
РОЗДІЛ 4. Тестування і демонстрація роботи системи.....31
4.1 Перевірка API-запитів та імітація ANPR.....31
РОЗДІЛ 5. Перспективи розвитку додатку.....36
5.1 Можливі розширення функціоналу.....36
Висновки.....37
Підсумок дослідження.....37
Основні результати та висновки з розробки додатку.....37
Список Використаної Літератури.....39
Додатки.....40
Додаток А.....41
Додаток Б.....42

АНОТАЦІЯ

Дипломна робота присвячена розробці автоматизованій системі паркування автомобілів в межах розумного будинку. Основною метою є створення зручного інструменту для керування доступом до паркінгу, зокрема реєстрації транспортних засобів, гостьових візитів і відкриття воріт.

У роботі описано архітектуру системи, реалізовано клієнтську частину на Swift із використанням SwiftUI, серверну частину на Python з FastAPI, а також REST API для взаємодії між ними. Проведено тестування функціональності розробленого застосунку та розглянуто можливості подальшого розвитку.

Ключові слова: розумний будинок, паркування, мобільний застосунок, SwiftUI, ANPR.

Перелік прийнятих скорочень

ANPR - Automatic Number Plate Recognition

IoT - Internet of Things

MVVM - Model-View-ViewModel

REST - Representational State Transfer

API - Application Programming Interface

HIG - Human Interface Guidelines

JWT - JSON Web Token

ВСТУП

Тема дипломної роботи полягає у дослідженні та розробці застосунку для системи паркування у межах розумного будинку. Основною метою роботи є створення зручного, функціонального та безпечного програмного засобу, який дозволяє мешканцям автоматизовано користуватися паркінгом, керувати власними автомобілями та дозволами доступу, а також додавати гостьові транспортні засоби.

Для реалізації цієї мети проаналізовано сучасні технології у сфері автоматизованого паркування та цифрової інфраструктури. Розглянуто використання ANPR сервісів, побудову серверної частини із застосуванням мови програмування Python і фреймворку FastAPI, а також створення бази даних на основі SQLite.

З боку користувача реалізовано нативний мобільний застосунок для операційної системи iOS, написаний мовою програмування Swift із використанням фреймворку SwiftUI. У роботі досліджено архітектурний патерн MVVM, особливості інтеграції з REST API та підходи до розробки інтерфейсу згідно з рекомендаціями Apple HIG.

Також розглянуто питання інтеграції мобільного застосунку із серверною частиною, взаємодії із сервісом ANPR, а також забезпечення надійної та зручної роботи користувачів із системою. Окрему увагу приділено аналізу існуючих рішень на ринку, визначенню їхніх переваг і недоліків, а також порівнянню з розробленим прототипом.

У результаті дослідження та розробки робочого прототипу мобільного застосунку для системи паркування у розумному будинку дипломна робота має на меті забезпечити користувачів зручним і ефективним інструментом для організації паркування, підвищити рівень автоматизації та безпеки, а також сприяти економії часу мешканців завдяки впровадженню сучасних інформаційних технологій.

Обґрунтування актуальності теми

У сучасних умовах стрімкої урбанізації та цифровізації житлового середовища все більшої популярності набуває концепція розумного будинку. Її впровадження забезпечує автоматизацію різноманітних процесів, таких як - контроль доступу, енергозбереження, безпеки, управління побутовими приладами. Однією з ключових складових функціональної інфраструктури розумного будинку є система паркування, яка безпосередньо впливає на щоденну зручність мешканців.

Традиційні системи паркування, що передбачають ручне відкриття воріт, використання фізичних ключів або обмежене коло доступу, вже не відповідають сучасним вимогам до автоматизації та безпеки. Водночас розвиток технологій, таких як ANPR, мобільні застосунки, системи аутентифікації та IoT, відкриває широкі можливості для створення інтелектуального, адаптивного та зручного середовища взаємодії з паркінгом.

Особливої актуальності набуває впровадження мобільних застосунків, які дозволяють мешканцям багатоквартирних будинків самостійно керувати власними автомобілями, відкривати ворота, додавати гостьові транспортні засоби, бронювати місця та отримувати миттєвий зворотний зв’язок від системи. Таким чином, система паркінгу переходить від традиційної ролі до інтерактивної складової сучасного розумного середовища.

Саме сегмент застосунків для паркування у житлових комплексах поки що не має глобальних гравців чи універсальних рішень, що дає можливість виходу з конкурентоспроможним продуктом. Згідно з аналітичним звітом The Business Research Company, світовий ринок систем розумного паркування у 2025 році був оцінений у 10,27 мільярдів доларів США, а до 2029 року очікується його зростання до понад 22,13 мільярдів доларів із середньорічним темпом зростання близько 20,8% [1]. Це свідчить про високу перспективність створення нішевого мобільного застосунку, орієнтованого саме на керування доступом до паркінгу у багатоквартирних будинках.

Отже, створення мобільного застосунку для управління системою паркування в межах розумного будинку є актуальним завданням, що відповідає сучасним технологічним трендам і соціальним потребам. Це дозволяє не лише підвищити комфорт мешканців, а й зменшити адміністративні витрати, підвищити рівень безпеки та адаптувати простір до реального стилю життя користувачів.

Мета та завдання дослідження

Метою дослідження є розробка та реалізація прототипу мобільного застосунку для iOS, який забезпечує інтелектуальне управління системою паркування в межах розумного будинку, з метою підвищення зручності, безпеки та ефективності використання паркувальних місць для мешканців.
Для досягнення поставленої мети необхідно вирішити такі завдання:
1) Аналіз існуючих мобільних застосунків для організації паркування, їхніх функціональних можливостей та недоліків.
2) Проєктування архітектури серверної частини системи із використанням мови програмування Python і фреймворку FastAPI, а також розробка структури бази даних.
3) Реалізація основного функціоналу застосунку, включаючи автоматизоване відкриття воріт, керування власними автомобілями, додавання гостьових номерів, перевірку статусу доступу та взаємодію із серверною частиною через REST API.
4) Розробка інтерфейсу мобільного застосунку з урахуванням принципів зручності, безпеки та відповідності рекомендаціям Apple HIG.
5) Тестування мобільного застосунку та серверної частини з метою виявлення та усунення можливих помилок і недоліків.
6) Розробка стратегії розвитку застосунку та визначення можливостей для розширення функціональності й підвищення зручності використання.

7) Оцінка ефективності розробленого рішення та його впливу на організацію паркування для мешканців розумного будинку.

Досягнення мети та виконання завдань дослідження передбачає вивчення актуальних джерел, проєктування архітектури системи, розробку програмного забезпечення, проведення його тестування та подальшого вдосконалення.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд популярних застосунків для паркування

На сучасному ринку мобільних застосунків для паркування існує велика кількість рішень, орієнтованих на міське середовище: пошук вільного місця, оплата паркування, інтеграція з муніципальними системами. Прикладами таких застосунків є «Київ Цифровий», Parkopedia, EasyPark, ParkMobile тощо. Вони забезпечують зручність для водіїв у щоденному пересуванні містом, проте не розраховані на використання в багатоквартирних житлових комплексів, де важливою є функціональність на кшталт обмеженого доступу, списку дозволених авто, гостьових перепусток тощо.

Метою аналізу конкурентів є вивчення функціональних можливостей існуючих застосунків для паркування, які орієнтовані на використання в багатоквартирних житлових будинках. Це дозволяє визначити найкращі практики, а також виявити недоліки, які можна врахувати під час розробки застосунку. Порівняння дає змогу краще зрозуміти очікування користувачів, наявні технічні рішення та знайти унікальні функції, які можуть забезпечити конкурентну перевагу майбутньому продукту.

На відміну від загальноміських рішень, існує низка спеціалізованих рішень для житлових комплексів.

Parklio — застосунок на IOS та Android для керування розумними бар'єрами, обмежувачами паркомісць та воротами. Користувачі можуть відкривати ворота або бар'єри зі смартфона через Bluetooth, а також створювати цифрові ключі для гостей із зазначенням терміну дії. Основна увага приділяється фізичному контролю доступу, проте відсутня централізована база транспортних засобів і облік історії використання. Додаток не підтримує ведення списку авто користувача чи контроль виїзду, що обмежує його функціональність у порівнянні з комплексними рішеннями [2].

ParqEx — додаток для IOS та Android для керування приватним паркуванням у житлових будинках. Застосунок дозволяє додавати власні автомобілі, створювати гостьові перепустки з конкретним періодом дії, переглядати історію паркування, а також дистанційно відкривати ворота. Підтримується інтеграція з ANPR-камерами для автоматичного в'їзду. Для адміністраторів доступні окремі панелі управління дозволами, паркомісцями та звітністю. Перевагою є гнучке управління доступом і розширений функціонал, проте система є складною у налаштуванні [3].

Smart Parking by Smart Spaces — система автоматизації доступу, що працює з ANPR-камерами, QR-кодами та цифровими запрошеннями. Користувачі можуть реєструвати свої авто, створювати запрошення для гостей, а система автоматично відкриває шлагбауми при розпізнаванні дозволених номерів. Доступ до функцій здійснюється через веб-інтерфейс, а не через класичний мобільний застосунок. Основна перевага — повна автоматизація процесу, проте відсутність мобільного додатка може бути незручною для користувачів [4].

Parkade — застосунок на мобільний телефон Apple або Android, орієнтований на простоту управління паркуванням у житлових комплексах. Користувачі можуть додавати авто, бронювати паркомісця для себе або гостей, отримувати сповіщення та відкривати ворота. Додаток також дозволяє мешканцям здавати свої паркомісця в оренду. Гостьовий доступ реалізовано через запрошення, а інтеграція з воротами забезпечує безконтактне відкриття. Перевагою є мінімалістичний інтерфейс і гнучкість [5].

Таким чином, розглянуті застосунки демонструють різні підходи до організації паркування у житлових комплексах: від фізичного контролю (Parklio) до комплексних платформ із автоматизацією та гнучким управлінням (ParqEx, Smart Parking by Smart Spaces, Parkade). Аналіз їхніх функціональних можливостей і обмежень дозволяє визначити напрями для вдосконалення власного рішення.

1.2 Недоліки та обмеження конкурентів

Аналіз популярних застосунків для організації паркування у житлових комплексах дозволяє виділити низку спільних недоліків та обмежень, які впливають на зручність і безпеку користувачів:

1) Обмежена персоналізація доступу. У більшості рішень відсутня можливість детально налаштовувати права для різних категорій користувачів (мешканці, гості, адміністратори), що ускладнює контроль за використанням паркомісць.

2) Недостатній контроль гостьового доступу. Зазвичай перепустки для гостей діють лише за часом, без можливості обмежити кількість в'їздів, що може призводити до зловживань.

3) Відсутність інтеграції з push-сповіщеннями для критичних подій наприклад, виїзд авто, коли за кермом не авторизована особа, що знижує рівень контролю та безпеки.

4) Фокус виключно на автоматизації, ігноруючи потребу мешканців у ручному управлінні або додаткових рівнях контролю.

Ці недоліки та обмеження створюють передумови для розробки нового рішення, яке поєднує автоматизацію з розширеним контролем, персоналізацією та підвищеною безпекою для мешканців житлових комплексів.

1.3 Порівняння функціональності з розроблюваним застосунком

Розроблюваний застосунок враховує недоліки конкурентів і пропонує низку унікальних функцій (таблиця 1.1), які забезпечують додаткову цінність для користувачів:

- Гнучка система керування доступом: можливість налаштовувати права для гостей, обмежувати кількість в'їздів для гостей, підтверджувати виїзд авто через push-сповіщення.

- Інтеграція з ANPR-сервісами для автоматичного розпізнавання номерних знаків, а також підтримка ручного управління для мешканців, які бажають мати додатковий контроль.
- Ведення персонального списку авто користувача, а також можливість створення гостьових перепусток із гнучкими параметрами.
- Реалізація push-сповіщень для інформування про важливі події, що підвищує рівень безпеки та дозволяє оперативно реагувати на зміни.
- Зручний і інтуїтивний інтерфейс, адаптований під потреби мешканців житлових комплексів, із дотриманням сучасних стандартів UX/UI.

Функціонал	Parklio	ParqEx	Smart Parking by Smart Spaces	Parkade	Розроблюваний застосунок
Додавання авто	+	+	+	+	+
Інтеграція з ANPR	+	+	+	–	+
Гостьові перепустки	–	+	+	+	+
Обмеження кількості в'їздів	–	–	–	–	+
Ручне управління доступом	+	+	–	+	+
Підтвердження виїзду (push)	–	–	–	–	+
Оренда паркомісця	–	–	–	+	–
Мобільний застосунок	+	+	– (web)	+	+

Таблиця 1.1 - Порівняння функціоналу застосунків для паркування

Таким чином, запропоноване рішення поєднує переваги автоматизації та персоналізованого контролю, що відрізняє його від існуючих аналогів і робить привабливим для цільової аудиторії.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ

2.1 Вибір мови програмування та фреймворку

Для розробки серверної частини було обрано мову програмування Python разом із веб-фреймворком FastAPI.

Вибір Python пояснюється його простим і зрозумілим синтаксисом, що дозволяє швидко писати та підтримувати код. До того ж, для Python існує багато готових бібліотек і велика спільнота, що допомагає легко інтегрувати зовнішні сервіси та реалізовувати різні функції, наприклад, роботу з API, базами даних та інтеграцію із зовнішніми сервісами, наприклад, для розпізнавання номерних знаків.

Фреймворк FastAPI було обрано, оскільки він дозволяє швидко і зручно створювати сучасні веб-сервіси типу REST API. FastAPI — один із найпродуктивніших фреймворків на Python: він працює дуже швидко завдяки використанню асинхронного сервера Uvicorn. Крім того, розробка на FastAPI відбувається швидше завдяки простому синтаксису та зручним інструментам. FastAPI підтримує типізацію Python і автоматично перевіряє, чи відповідають отримані дані потрібній структурі. Це зменшує кількість рутинного коду та допомагає уникати помилок. Ще одна важлива перевага — автоматичне створення документації для API, що дуже зручно для тестування та подальшої роботи з сервісом.

Асинхронна модель FastAPI дозволяє обробляти багато запитів одночасно, що особливо корисно для системи паркування: наприклад, сервер може одночасно реагувати на в’їзд авто та звертатися до зовнішнього сервісу. Отже, поєднання Python і FastAPI дало змогу швидко створити надійний і гнучкий серверний застосунок, який легко підтримувати та розвивати.

2.2 Архітектура RESTful API та структура серверного застосунку

Серверна частина системи побудована за принципами REST-архітектури, яка є сучасним стандартом для створення веб-сервісів. RESTful API дозволяє організувати взаємодію між клієнтом (мобільним застосунком) і сервером у вигляді чітко визначених HTTP-запитів до окремих ресурсів, таких як користувачі, автомобілі, гостьові перепустки, ворота тощо.

Використання RESTful API має декілька переваг для нашого проєкту:

- Універсальність: Мобільний застосунок, веб-інтерфейс або сторонні сервіси можуть легко взаємодіяти із сервером через стандартні HTTP-запити.
- Масштабованість: Кожен ресурс, такі як, авто чи гість має власний набір ендпоінтів, що спрощує розширення функціоналу без зміни всієї системи.
- Зрозуміла структура: REST-архітектура передбачає логічне групування запитів (GET, POST, PUT, DELETE), що робить API зрозумілим для розробників і полегшує підтримку.
- Безпека: REST дозволяє легко впроваджувати сучасні механізми автентифікації та авторизації.
- Документованість: Завдяки використанню FastAPI, документація для всіх ендпоінтів генерується автоматично, що спрощує тестування та інтеграцію.

Така архітектура ідеально підходить для системи паркування у житловому комплексі, оскільки дозволяє швидко додавати нові функції, забезпечує надійний обмін даними між клієнтом і сервером, а також гарантує зручність для розробників і користувачів.

2.3 Структура бази даних та інструменти міграції

Для зберігання даних у системі використовується реляційна база даних SQLite. Вона проста у налаштуванні, не потребує окремого серверного процесу і зберігає всю інформацію в одному файлі, що добре підходить для невеликих систем, таких як паркування у житловому комплексі. SQLite забезпечує

достатню продуктивність для локальних задач і підтримує основні властивості реляційних баз даних [6].

Структура бази даних описується за допомогою ORM-бібліотеки SQLAlchemy. Таблиці та зв’язки між ними визначаються у вигляді Python-класів, що спрощує розробку і дозволяє легко змінювати тип бази даних у майбутньому. Всі запити до БД виконуються через методи і вирази на Python, що під капотом генерують відповідні SQL-запити.
Для керування змінами у структурі бази даних використовується Alembic — інструмент для міграцій, який дозволяє поступово змінювати схему БД без втрати даних. Alembic зберігає історію змін і застосовує їх до реальної бази, що особливо зручно під час розробки та тестування [7].

На момент написання дипломної роботи в базі даних використовуються такі основні таблиці:
users — користувачі системи;
vehicles - автомобілі, включаючи власні та гостьові;
bookings — записи про бронювання паркомісць;
anpr_logs — події, пов’язані з розпізнаванням номерних знаків;
gate_commands — команди до воріт;
exit_confirmations — підтвердження факту виїзду автомобіля.

Ці таблиці формують основну структуру, необхідну для роботи застосунку. Такий підхід до роботи з базами даних забезпечує гнучкість і зручність у розробці: розробник працює з даними на рівні Python-об’єктів, а всі зміни у структурі БД можна впроваджувати поступово та без втрати інформації. У разі необхідності масштабування або переходу на іншу СУБД, структура моделей і логіка роботи з даними залишаться майже незмінними, що значно спрощує підтримку та розвиток системи.

2.4 Система автентифікації та підтвердження дій користувача

Для захисту доступу до системи паркування реалізовано схему автентифікації на основі JWT токенів [8]. Після успішного входу користувач отримує токен, який містить основну інформацію про нього (ідентифікатор, роль, час дії) та підписується секретним ключем сервера. Надалі цей токен використовується для доступу до захищених ресурсів: клієнт надсилає його у кожному запиті, а сервер перевіряє його дійсність і права доступу. Такий підхід забезпечує безсесійність автентифікації, спрощує масштабування та відповідає принципам REST-архітектури .

Паролі користувачів зберігаються у базі лише у вигляді хешів, отриманих за допомогою алгоритму bcrypt через бібліотеку Passlib [9]. Це гарантує, що навіть у разі витоку бази даних зловмисник не отримає справжніх паролів. Bcrypt автоматично додає сіль і дозволяє регулювати складність хешування, що підвищує стійкість до атак перебором .

Використання JWT-токенів дозволяє зручно ідентифікувати кожного користувача при виконанні дій у системі. Завдяки цьому можна персоналізувати функціонал: наприклад, додавати автомобілі саме до списку поточного користувача, керувати власними гостьовими доступами тощо. Таким чином, система чітко розмежовує дії між різними користувачами, забезпечуючи коректний розподіл прав і доступу до персональних даних.

Для підвищення безпеки та зручності підтвердження важливих дій таких як виїзд автомобіля або інших у системі використовується WebSocket[10]. Це дозволяє надсилати push-нотифікації власнику авто в реальному часі: коли система фіксує спробу виїзду, власник отримує повідомлення через WebSocket-з’єднання і може одразу підтвердити або відхилити запит. Такий підхід забезпечує оперативний зворотний зв’язок і підвищує рівень контролю над доступом .

Такий підхід забезпечує надійну, сучасну та зручну для користувачів автентифікацію і підтвердження дій, що є стандартом для подібних систем.

2.5 Взаємодія із зовнішнім ANPR-сервісом через API

Одна з найголовніших функцій системи є ANPR при в'їзді та виїзді з паркінгу. Для цього інтегровано зовнішній ANPR-сервіс, який надає REST API для розпізнавання номерів за зображенням. Такий підхід дозволяє отримати якісний результат розпізнавання, використовуючи сучасні напрацювання у сфері машинного зору, і зосередитися на розробці бізнес-логіки паркування.

Принцип роботи інтеграції.
Коли камера фіксує автомобіль, система формує HTTP POST-запит до ANPR API, передаючи зображення та токен автентифікації. У відповідь отримується JSON із розпізнаним номером, рівнем впевненості та, за потреби, додатковими даними. Далі серверна логіка перевіряє цей номер у базі даних: якщо авто має дозвіл — відкриває шлагбаум, якщо ні — фіксує спробу і записує номер та час невдалої спроби в'їзду.

Асинхронність і надійність.
Використання асинхронного HTTP-клієнта дозволяє не блокувати сервер під час очікування відповіді від ANPR-сервісу, що підвищує продуктивність системи. Усі помилки обробляються відповідними повідомленнями для користувача або оператора.

Гнучкість та модульність.
Інтеграція з ANPR-сервісом реалізована як окремий компонент із чітким інтерфейсом. Це дозволяє легко змінити постачальника ANPR або додати нові функції без впливу на основну логіку системи.

Документування та тестування.

Для зручності розробників і тестування API використовується автоматично згенерована документація у форматі Swagger UI [11]. Вона дозволяє переглядати всі доступні ендпоінти, їхні параметри та структуру відповідей, а також тестувати запити безпосередньо з браузера. На рисунку 2.1 показано основні ендпоінти для автентифікації, керування транспортними засобами, гостьовими перепустками та іншими функціями системи.

Рисунок 2.1 - Структура REST API для мобільного застосунку

РОЗДІЛ 3. РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Обґрунтування вибору мови Swift

Для розробки мобільного застосунку було обрано мову програмування Swift. Swift — це сучасна мова, створена Apple спеціально для розробки застосунків під iOS, macOS, watchOS та tvOS. Вона з'явилася у 2014 році як заміна Objective-C і швидко стала основним інструментом для створення програм у екосистемі Apple.

Swift активно розвивається не лише Apple, а й світовою спільнотою розробників. Це забезпечує швидке впровадження нових можливостей, регулярні оновлення та доступ до великої кількості сучасних бібліотек і готових рішень. Відкритість мови також гарантує прозорість, надійність і безпеку — будь-хто може переглянути її код, переконатися у якості реалізації та долучитися до розвитку.

Серед основних переваг Swift для розробки мобільних застосунків можна виділити:

- Популярність і підтримка: за даними Index, Swift входить до топ-3 найпопулярніших мов програмування для мобільних пристроїв у світі, а серед iOS-розробників є стандартом [12]. В App Store зараз понад 2 мільйона застосунків, більшість з яких створені саме на Swift [13].
- Висока продуктивність: код на Swift компілюється безпосередньо у машинний код пристрою, що забезпечує швидку та стабільну роботу застосунку без додаткових проміжних шарів.
- Сучасний синтаксис і простота: Swift легко читати й підтримувати, а лаконічний синтаксис дозволяє швидко реалізовувати навіть складні функції.

- Масштабованість і довгострокова підтримка: нативні застосунки на Swift простіше оновлювати разом із новими версіями iOS, а відкритий код і активна спільнота гарантують швидке впровадження сучасних рішень.

Завдяки цим перевагам Swift став оптимальним вибором для створення сучасного, зручного та надійного мобільного IOS застосунку для мешканців будинку.

3.2 Архітектура застосунку: MVVM та SwiftUI

Під час розробки застосунку було використано архітектуру MVVM, яка добре підходить для роботи зі SwiftUI. Цей підхід допомагає розділити програму на частини: Model зберігає дані та логіку, View відповідає за зовнішній вигляд, а ViewModel з’єднує ці дві частини — вона готує дані для відображення і реагує на дії користувача. Така структура дозволяє легше змінювати або тестувати окремі частини застосунку, не зачіпаючи весь код.

У дипломній роботі Model представлений такими структурами даних:

UserMe — структура для збереження профілю авторизованого користувача. Містить id, email, name, а також createdAt — дату створення облікового запису. Завантажується після успішного входу в систему.

VehicleOut — структура, що містить інформацію про автомобіль користувача. Включає такі поля, як id, licensePlate, name. Застосовується при відображенні списку авто, а також при створенні або редагуванні записів.

GuestOut — описує гостьовий доступ. Містить поля id, licensePlate, name, accessStart, accessEnd, passCount. Використовується для керування гостьовими перепустками та контролю часу/кількості доступів.

Усі структури відповідають протоколу Decodable — це дозволяє автоматично розбирати відповіді з API у Swift-об'єкти. Вони є частиною модельного шару Model та активно використовуються у ViewModel для передачі та відображення даних у застосунку.

ViewModel — це частина логіки, яка відповідає за зв’язок між даними Model і інтерфейсом користувача View. У застосунку реалізовано два основні ViewModel-класи: AuthViewModel та MainViewModel. Обидва успадковують ObservableObject, що дозволяє автоматично оновлювати інтерфейс при зміні даних.

AuthViewModel — керує логікою автентифікації: обробка входу, реєстрації, зберігання токена доступу (@AppStorage("accessToken")) та завантаження профілю користувача - UserMe. Має властивості isAuthenticated, errorMessage і user, що відображають поточний стан сесії. Використовує APIService для запитів до бекенду. Після успішного входу зберігає токен і автоматично оновлює статус авторизації.

MainViewModel — відповідає за основний функціонал після входу. Зберігає масиви cars та guests, що містять дані про особисті автомобілі та гостьові доступи. Має метод fetchData() для завантаження даних при відкритті головного екрану, який виконує паралельні запити через async let. Також реалізує методи для створення, редагування та видалення записів (CRUD), а також openGate() — для відправки команди на відкриття воріт.

Усі запити до сервера виконуються асинхронно. ViewModel ізольовує мережеву логіку від інтерфейсу, забезпечуючи чисту та керовану архітектуру. Зміни стану ViewModel автоматично відображаються у SwiftUI-представленнях, що забезпечує реактивність та зручність користування.

View — це частина інтерфейсу, яка відповідає за відображення даних та взаємодію користувача з застосунком. У роботі інтерфейс створено за допомогою SwiftUI, що дозволяє описувати UI декларативно, а також автоматично оновлювати його при зміні стану даних.

ContentView — кореневий вигляд застосунку. Містить таку логіку: якщо користувач авторизований, відображається MainView, а інакше — AuthView.

AuthView — інтерфейс авторизації та реєстрації. Включає поля для введення логіну і паролю, а також кнопку входу, що викликає метод login() у ViewModel. У разі помилки відображає повідомлення.

MainView — головний екран після входу. Містить вертикальний стек (VStack) з двома секціями:
carsSection — список зареєстрованих авто користувача.
guestsSection — список гостьових пропусків із деталями доступу.
Також містить кнопку «Відкрити ворота», яка викликає метод openGate() у MainViewModel.
У MainView також реалізовано onAppear, який запускає метод fetchData() — одночасно завантажуються списки авто та гостей. Дані автоматично оновлюються завдяки використанню @StateObject для ViewModel, а також реактивності SwiftUI.

Таким чином, інтерфейс програми повністю керується станом ViewModel. Зміни в даних автоматично оновлюють відображення, а логіка взаємодії із сервером — повністю ізольована у ViewModel, що робить код чистим і зручним для підтримки.

3.3 Взаємодія з API (авторизація, робота з авто, гостями та воротами)

Уся логіка взаємодії з сервером зосереджена у єдиному класі APIService, який реалізує шаблон Singleton. Цей клас виконує всі запити до бекенда: автентифікація користувачів, додавання або видалення транспорта, обробка гостьових пропусків і відкриття воріт.
Кожен публічний метод APIService асинхронний (позначений async) і використовує стандартні засоби Swift — URLRequest, URLSession, JSONDecoder — для обміну з REST API.

Авторизація

login() — надсилає запит на /auth/login з переданими обліковими даними. У відповіді очікується accessToken, який зберігається у полі token, а також повертається як результат методу.

register() — виконує POST-запит для створення нового користувача. У тілі передаються ім’я, email і пароль.

getMe() — отримує поточного користувача (/auth/me) після авторизації та повертає об’єкт UserMe.

Транспортні засоби

getVehicles() — завантажує список всіх авто користувача.

createVehicle() — додає новий автомобіль. Передаються номер, назва та тип (особистий або гостьовий).

updateVehicle() — оновлює дані автомобіля за id.

deleteVehicle() — видаляє автомобіль.

Ці методи працюють з моделлю VehicleOut, яка автоматично створюється з JSON-відповіді через протокол Decodable.

Гості

getGuests() — повертає список усіх активних/неактивних гостьових доступів.

createGuest() — створює гостьовий пропуск з заданими параметрами: номер, ім’я, час початку/завершення дії, ліміт входів.

updateGuest() — повністю оновлює гостьовий запис.

updateGuestName() — змінює лише ім’я гостя.

deactivateGuest() — деактивує пропуск.

activateGuest() — повторно активує гостьовий доступ із заданими параметрами.

deleteGuest() — видаляє гостьовий запис.

Усі методи повертають об’єкт GuestOut у разі успішного виконання, який потім передається у ViewModel.

Ворота та ANPR

openGate() — надсилає POST-запит на /gate/open для відкриття воріт.

sendANPR() — метод для обробки номерного знака, наприклад, з камери. Надсилає номер і час та отримує необроблений JSON у відповідь.

Клас має приватні методи:

makeRequest() — створює запит з підстановкою базового URL, заголовків, токена авторизації та тіла.

perform<T: Decodable> — виконує запит і автоматично декодує відповідь.

performVoid() — використовується для запитів без відповіді.

performAny() — повертає JSON як словник.

Крім того, допоміжний метод formURLEncoded використовується для кодування логіну (формат application/x-www-form-urlencoded).

Таким чином, APIService повністю інкапсулює всі взаємодії з сервером, забезпечуючи ViewModel-и простими методами для виклику необхідних дій. Це спрощує підтримку й дозволяє змінювати API в одному місці без потреби втручання в інші частини застосунку.

3.4 Реалізація інтерфейсу користувача відповідно до Apple HIG

Інтерфейс мобільного додатку було розроблено з дотриманням рекомендацій Apple щодо дизайну - Apple HIG, які охоплюють ключові аспекти сучасного дизайну: доступність, читабельність, інтуїтивну навігацію та візуальну ієрархію. Усі елементи ретельно розміщені з урахуванням зручності дотику, використовуються чіткі кольорові контрасти. Такий підхід забезпечує комфортну взаємодію користувача із застосунком і сприяє високій якості користувацького досвіду.

Нижче наведено приклади ключових екранів застосунку, які демонструють реалізацію інтерфейсу та дотримання принципів HIG:

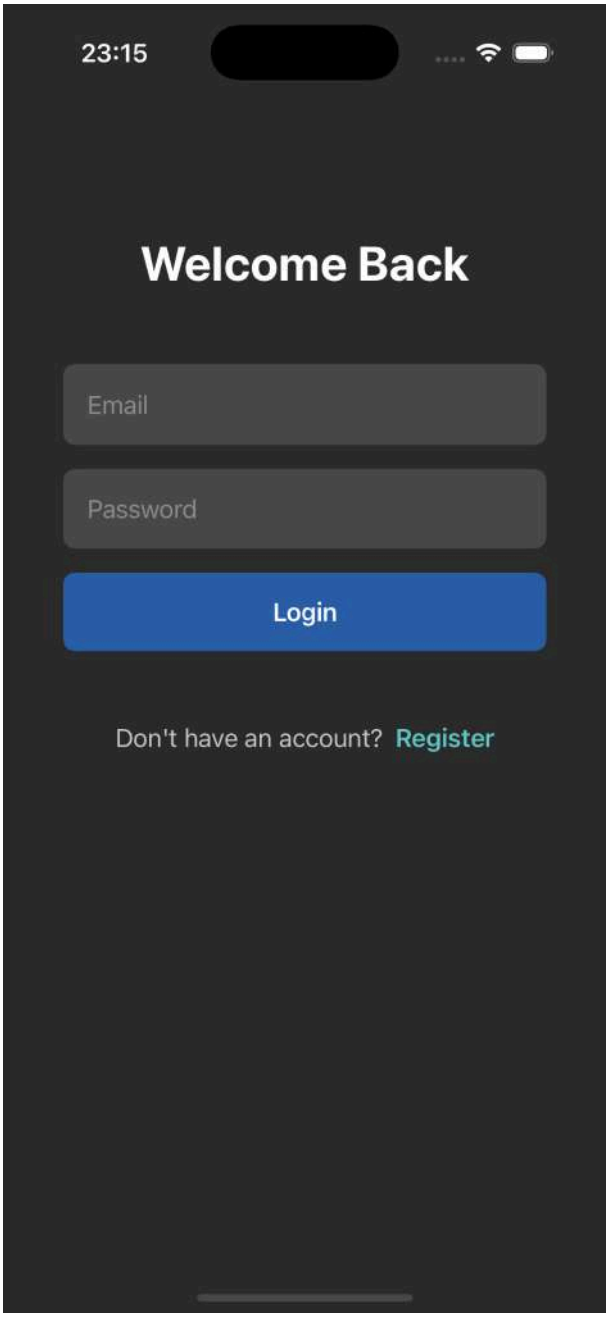


Рисунок 3.1 – Екран реєстрації

На рисунку 3.1 форма створення облікового запису з трьома полями (ім’я, email, пароль) та кнопкою реєстрації. Низом розташоване посилання на екран входу. Мінімалістичний стиль і великий розмір елементів забезпечують зручність користування.

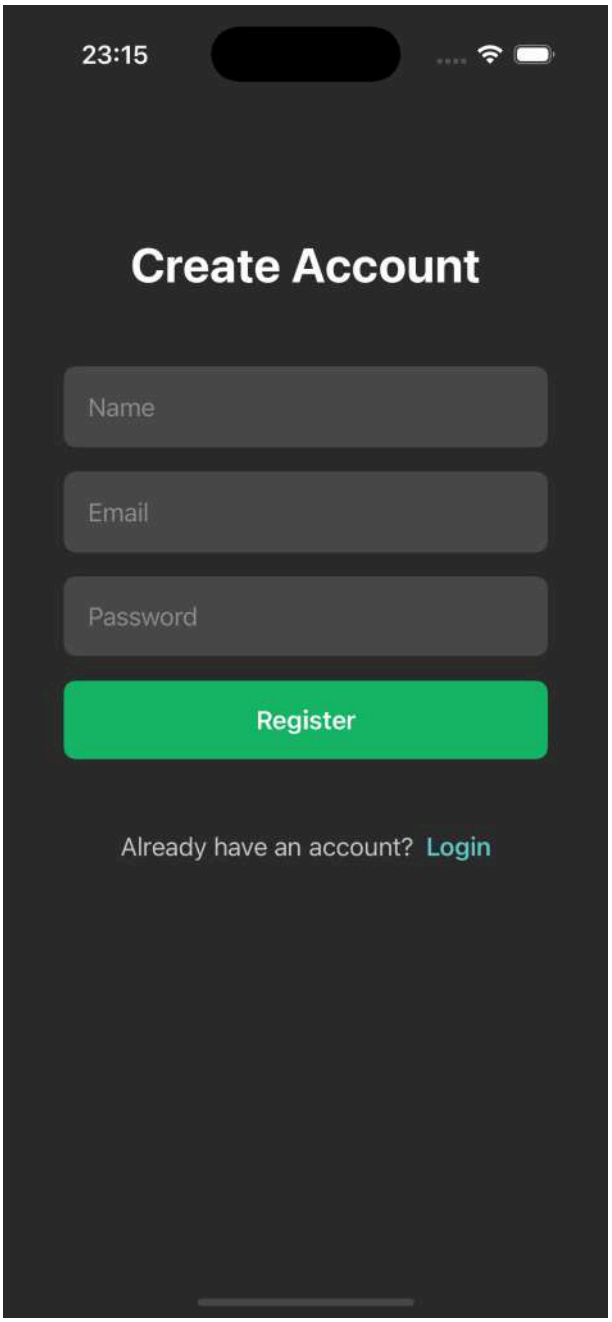


Рисунок 3.2 – Екран входу

На рисунку 3.2 простий інтерфейс для авторизації користувача з полями email і пароль. Синя кнопка Login чітко контрастує з фоном, а нижнє посилання веде до форми реєстрації.

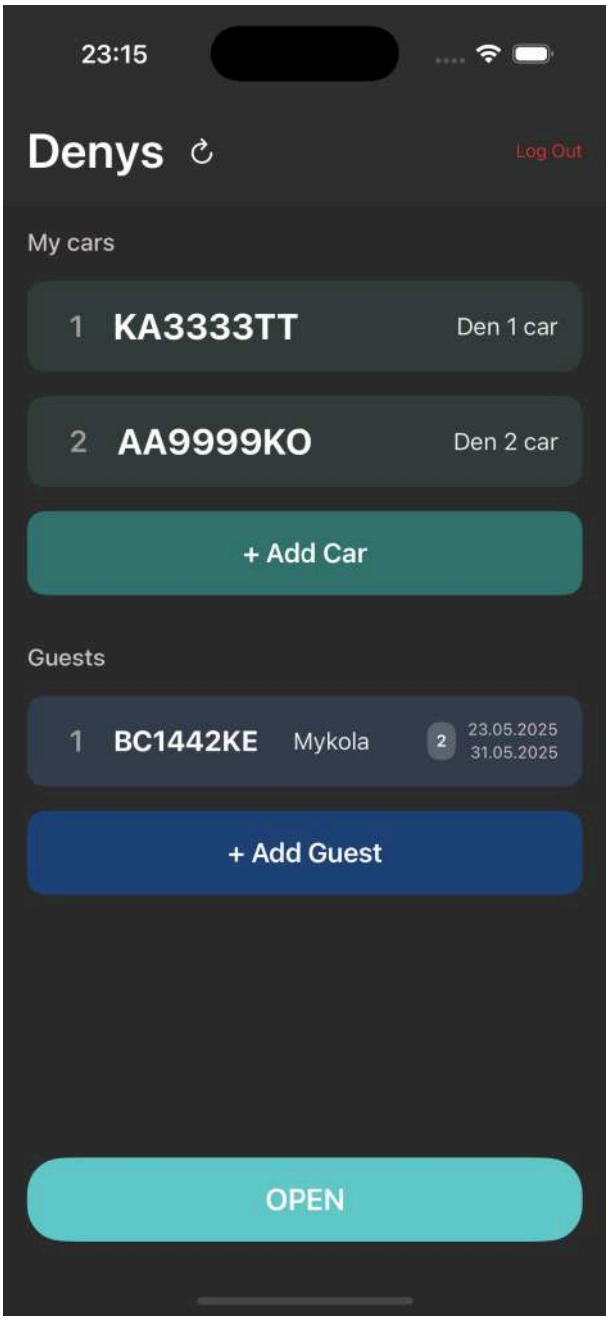


Рисунок 3.3 – Головний екран користувача

На рисунку 3.3 відображено список зареєстрованих автомобілів і гостей. Кнопки Add Car, Add Guest та OPEN чітко виділені. Екран організований у вертикальні секції з логічним поділом і кольоровими акцентами.

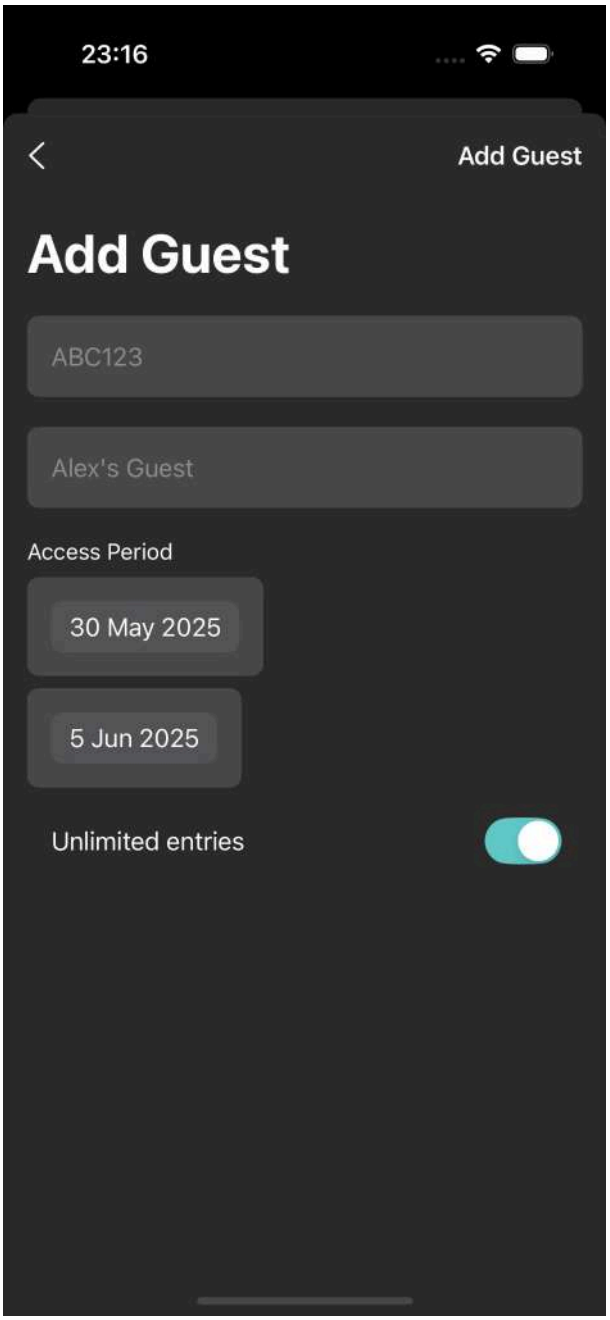


Рисунок 3.4 – Додавання гостя

На рисунку 3.4 форма створення гостьового доступу: вводиться номер авто, ім’я, дата початку та завершення доступу, а також кількість в’їздів. Перемикач дозволяє зробити перепустку необмеженою.

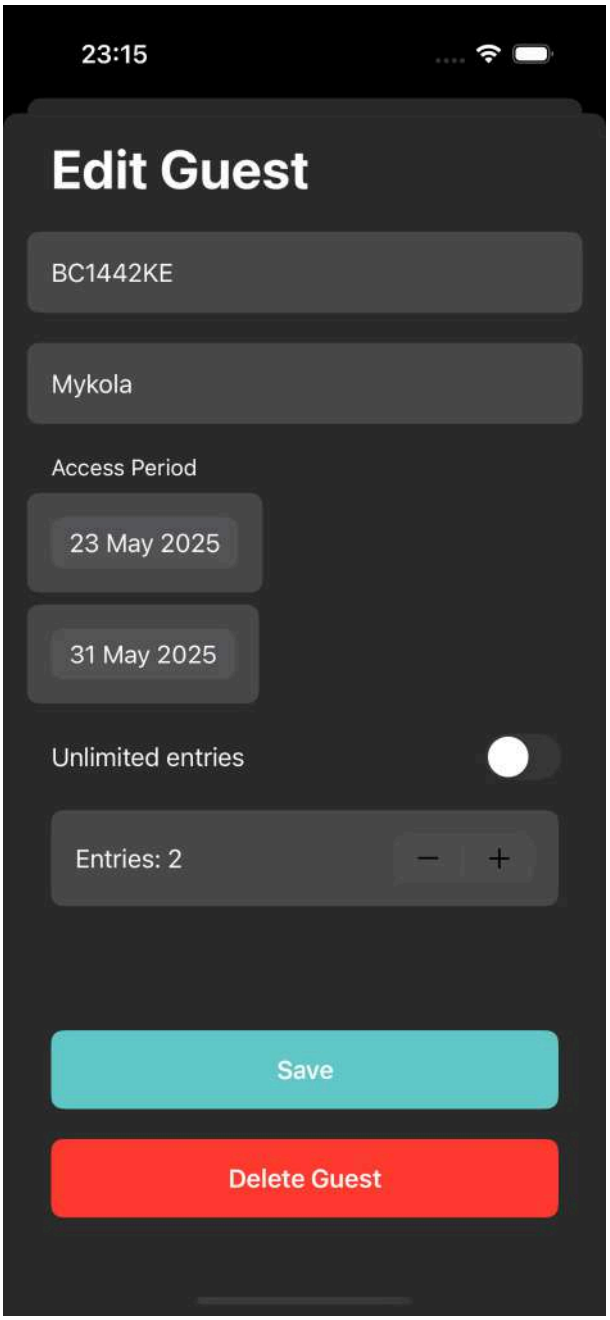


Рисунок 3.5 – Редагування гостьового доступу

На рисунку 3.5 екран для зміни даних гостьової перепустки. Окрім редагування полів, доступний перемикач на необмежений доступ та лічильник кількості дозволених в’їздів. Внизу — кнопки Save і Delete Guest.

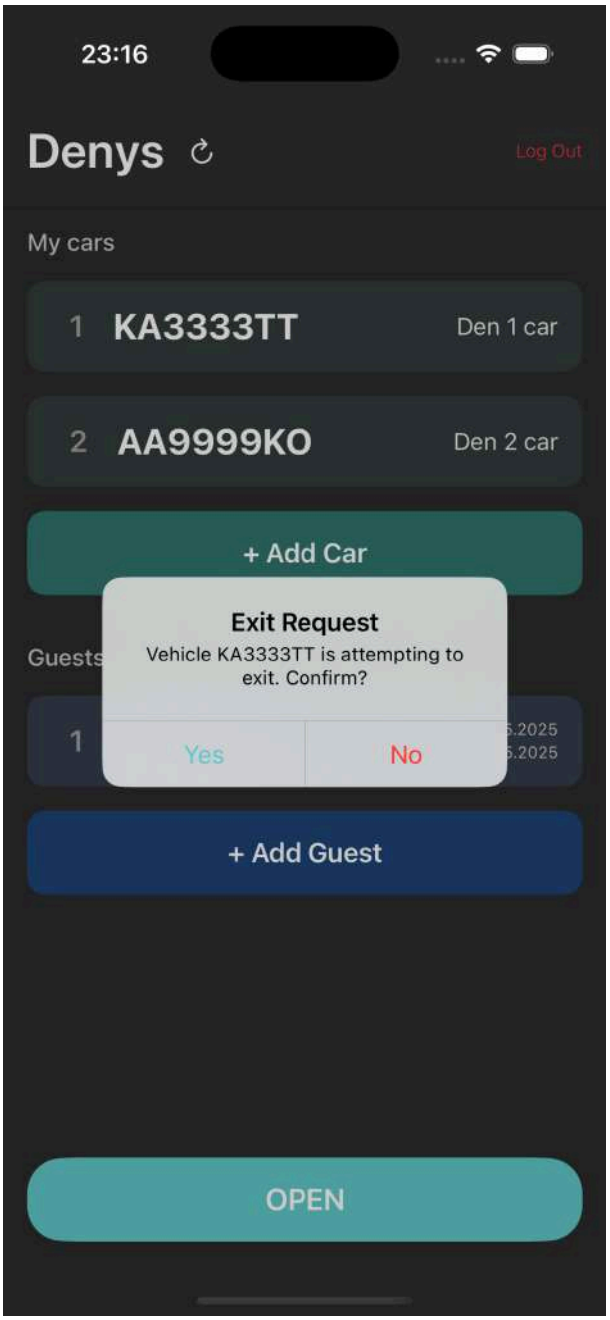


Рисунок 3.6 – Підтвердження виїзду авто

На рисунку 3.6 Випливаюче діалогове вікно (alert) з підтвердженням на виїзд авто. Реалізоване за допомогою стандартного компонента SwiftUI, що забезпечує швидку взаємодію з користувачем.

Таким чином, інтерфейс застосунку відповідає вимогам до зручності та доступності, забезпечуючи легку взаємодію для користувача.

РОЗДІЛ 4. ТЕСТУВАННЯ І ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ

4.1 Перевірка API-запитів та імітація ANPR

Тестування серверної частини проводилось шляхом надсилання HTTP-запитів до API за допомогою інструмента Postman. Це дозволило перевірити коректність обробки запитів, відповідей, а також поведінку системи у різних сценаріях.

Було протестовано ключовий ендпоінт /anpr, що імітує роботу камери, яка зчитує номер авто при в'їзді. До запиту передаються два параметри: номерний знак (license_plate) та час події (timestamp). У відповіді сервер повертає, чи дозволено в'їзд, а також причину дозволу або відмови.

На рисунку 4.1 демонстрація прикладу, коли автомобілю не дозволено в'їзд. Надіслано номер "AA9999KK", який відсутній у базі або не має активного дозволу. У відповіді ми бачимо access_granted: false та reason: not_allowed.

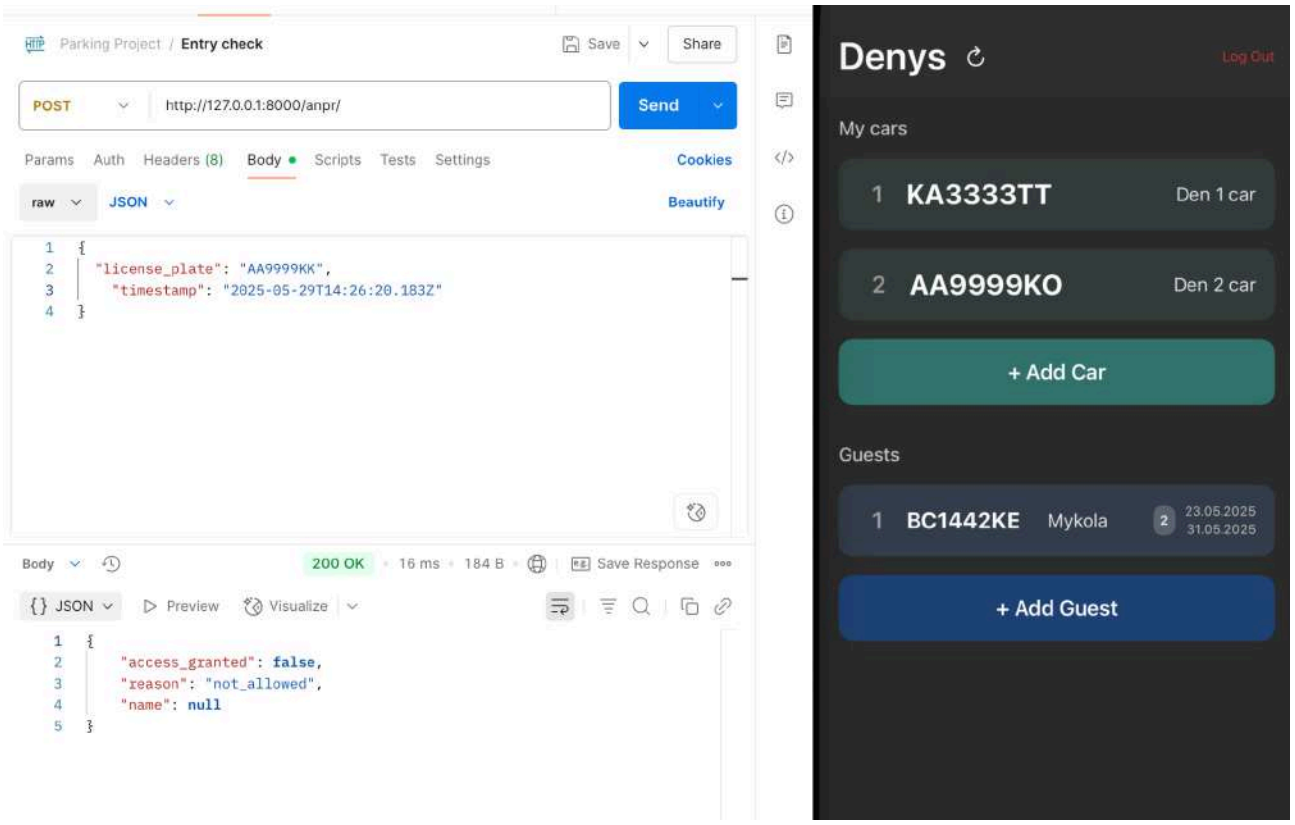


Рисунок 4.1 - Відхилення в'їзду на основі перевірки

На рисунку 4.2 показаний позитивний сценарій: номер “AA9999KO” є в базі як особистий транспорт користувача. У цьому випадку відповідь містить access_granted: true, reason: personal та name — назва авто в застосунку.

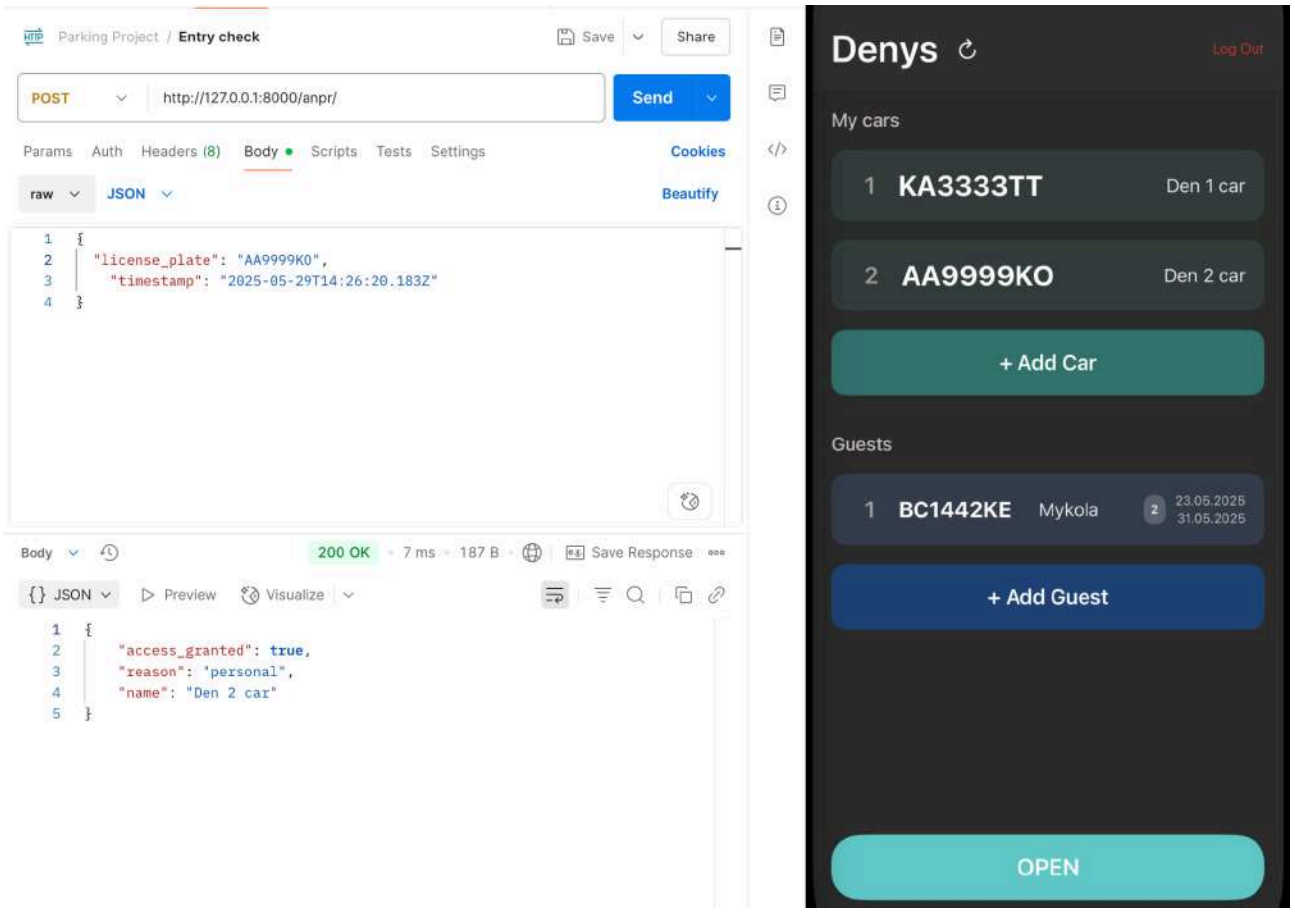


Рисунок 4.2 - Успішний в’їзд зареєстрованого особистого автомобіля

На рисунку 4.3 продемонстровано позитивний сценарій в’їзду гостьового авто. Номер “BC1442KE” занесений до системи як активний гостьовий доступ. У відповіді API вказано access_granted: true, reason: "guest", ім’я гостя: "Mykola", а також залишилась 1 спроба в’їзду ("passes_left": 1). Це свідчить про коректну роботу системи контролю доступу для тимчасових перепусток.

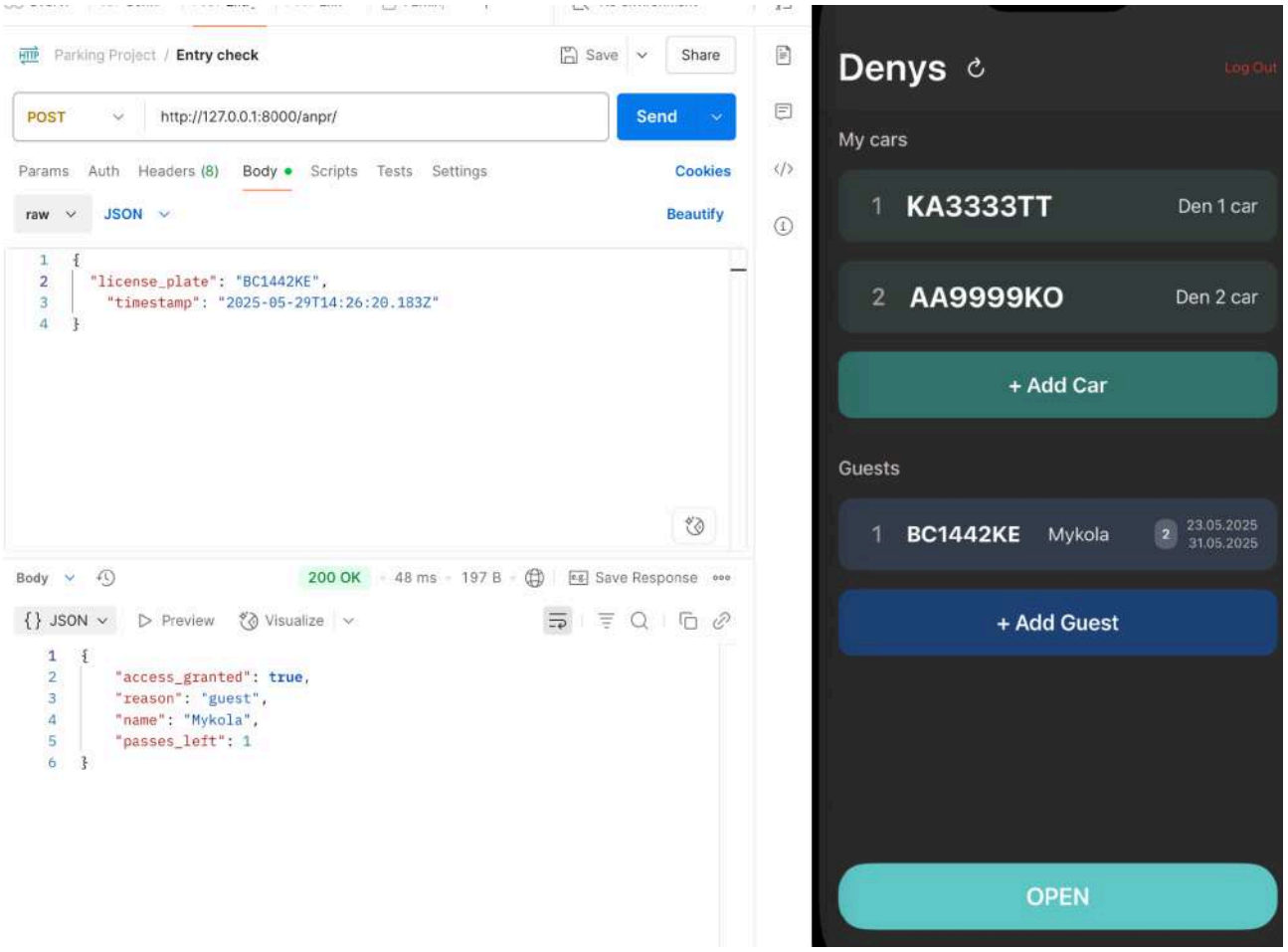


Рисунок 4.3 -Успішний в’їзд гостя із залишком перепусток

Також протестовано логіку виїзду з парковки, яка включає механізм підтвердження власником. На рисунку 4.4 продемонстровано запит на перевірку виїзду автомобіля “KA3333TT” через ендпоінт /exit/check. У відповіді: should_allow: false, а повідомлення message: "Owner confirmation required". На екрані застосунку з'являється спливаюче вікно з питанням підтвердити виїзд — це видно на знімку праворуч.

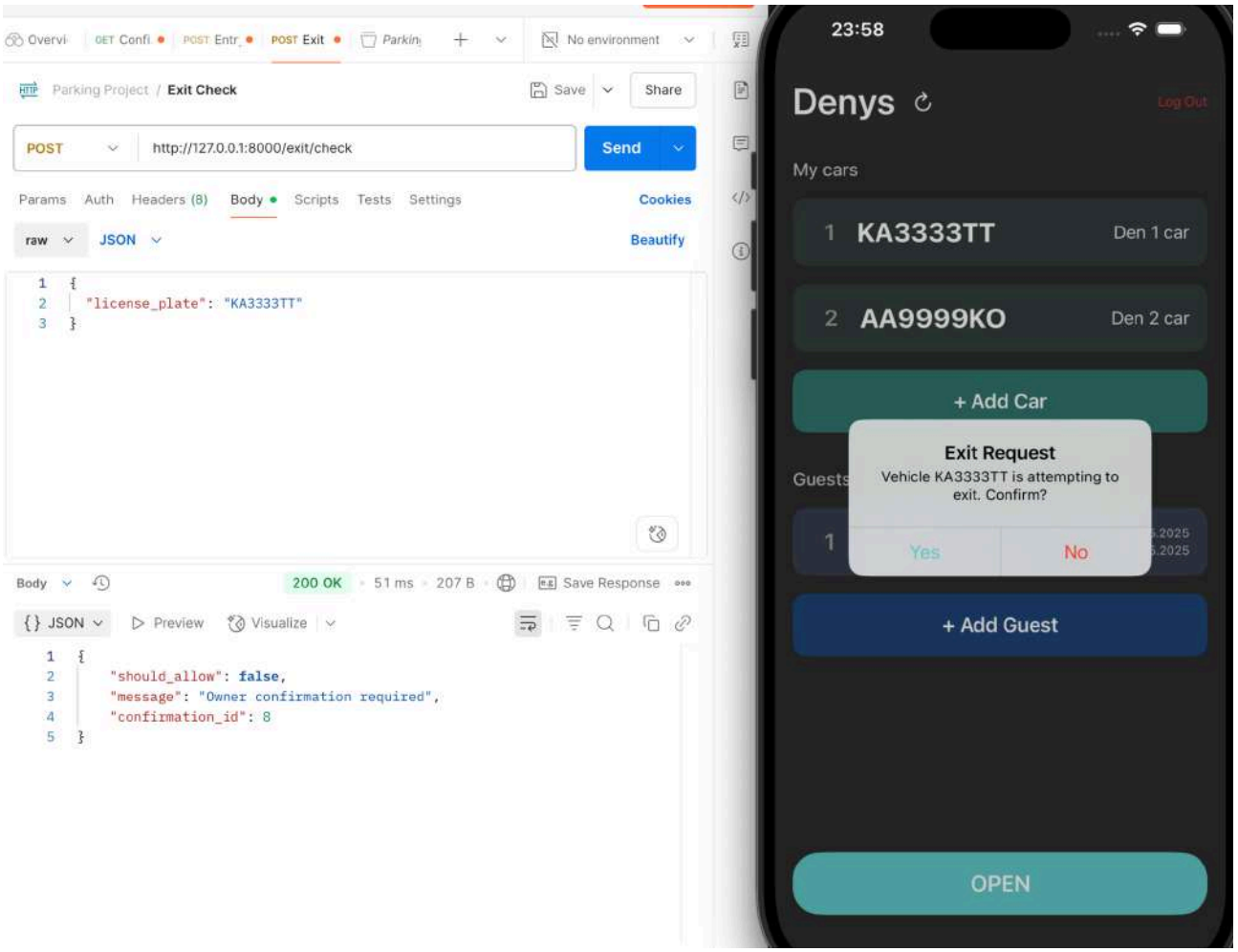


Рисунок 4.4 - Запит на перевірку виїзду

На рисунку 4.5 — запит до ендпоінта /exit/status/{id} після того, як користувач підтвердив виїзд в додатку. У відповіді сервер повертає should_allow: true і message: "Confirmed", що надає дозвіл для відкриття воріт.

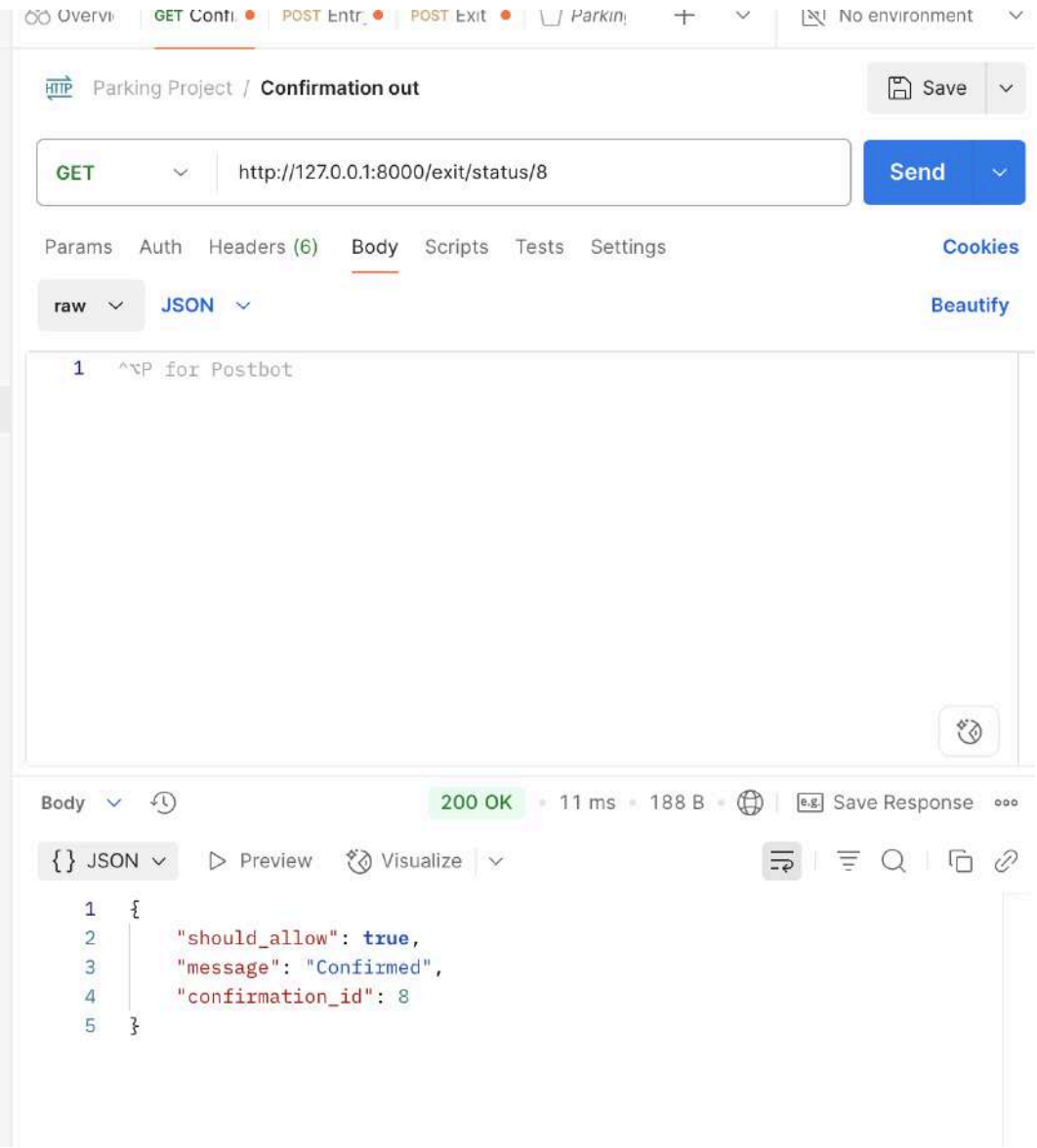


Рисунок 4.5 - Перевірка статусу підтвердження виїзду за ідентифікатором

Результати перевірки показали, що серверна частина стабільно обробляє запити на в’їзд і виїзд, коректно реагує на номерні знаки та дотримується логіки доступу. Усі сценарії — як з особистими авто, так і з гостьовими — успішно відпрацьовуються, а підтвердження виїзду відображається в інтерфейсі застосунку.

РОЗДІЛ 5. ПЕРСПЕКТИВИ РОЗВИТКУ ДОДАТКУ

5.1 Можливі розширення функціоналу

У майбутньому застосунок може бути розширений додатковими функціями, які підвищать його зручність, безпеку та функціональність до потреб корсистувачів. Буде впроваджено такі нові функції:

- моніторинг простору паркування. Дозволить мешканцям та адміністрації будинку в режимі реального часу бачити, які місця на паркінгу є вільними, а які вже зайняті.
- інтелектуальне освітлення. Система зможе автоматично керувати освітленням залежно від умов та подій, для підвищення енергоефективності та комфорту.
- інтеграція з камерами та ШІ для виявлення підозрілих дій. ШІ-моделі зможуть розпізнавати нетипову поведінку, наприклад, довге перебування біля авто без руху, спроби відкриття чужих транспортних засобів або проникнення пішоходів у зону паркінгу та повідомляти про це охороні.
- аналітичний модуль, який дозволить переглядати історію в’їздів/виїздів, тривалість перебування автомобілів, кількість використаних перепусток та інші статистичні дані. Це буде корисним не лише для користувача, а й для адміністрації будинку.

Такі функції дозволять додатку не лише залишатись актуальним, але й стати частиною повноцінної системи "розумного будинку".

ВИСНОВКИ

Підсумок дослідження

У межах цього дослідження було проаналізовано актуальність впровадження систем для керування доступом до паркінгу в межах розумних будинків. Враховуючи стрімкий розвиток технологій та зростання попиту на автоматизовані рішення для житлових будинків, застосування таких систем стає необхідним. Паркування є важливою складовою структури розумного будинку, яка впливає на комфорт, безпеку та користувацький досвід. Сучасні технології автоматизації, зокрема ANPR, мобільні застосунки та хмарні API, відкривають нові можливості для функціонального та зручного керування паркування у будинку.

Було обґрунтовано вибір мови Swift для створення iOS-застосунку завдяки її ефективності та зручності у поєднанні з архітектурою MVVM та SwiftUI. Також, мобільний додаток дозволяє легко виконувати всі необхідні дії — автентифікація, додавання авто, відкриття шлагбауму.
Також досліджено способи імітації роботи камер розпізнавання номерних знаків та перевірено повний цикл взаємодії системи на практиці. Реалізований застосунок підтвердив доцільність впровадження цифрових технологій у житлову інфраструктуру.

Основні результати та висновки з розробки додатку

У ході виконання роботи було розроблено застосунок, в якому реалізовано весь запланований функціонал: авторизація та реєстрація користувача, управління особистими автомобілями, створення та редагування гостьових

доступів, а також можливість дистанційного відкриття воріт безпосередньо зі смартфона.
Було досягнуто високого рівня зручності та швидкодії застосунку завдяки використанню SwiftUI та архітектури MVVM.
Особливу увагу приділено адаптації інтерфейсу до системних рекомендацій Apple HIG, що забезпечило інтуїтивну взаємодію користувача з системою.

Застосунок протестовано на симуляторі та вручну за допомогою API-запитів через Postman, що дозволило переконатися в стабільності та правильності роботи наявних функцій. Наявність відкритого API забезпечує широкі можливості інтеграції з іншими сервісами та платформами, що робить рішення гнучким у контексті майбутнього розвитку й сумісним із більшою екосистемою розумного дому.

Результатом роботи є готовий до використання додаток, який здатен покращити взаємодію мешканців із системою паркування в межах розумного будинку. Проєкт продемонстрував практичну реалізацію сучасних технологій мобільної розробки та має потенціал для подальшого розвитку з додаванням нових функцій.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

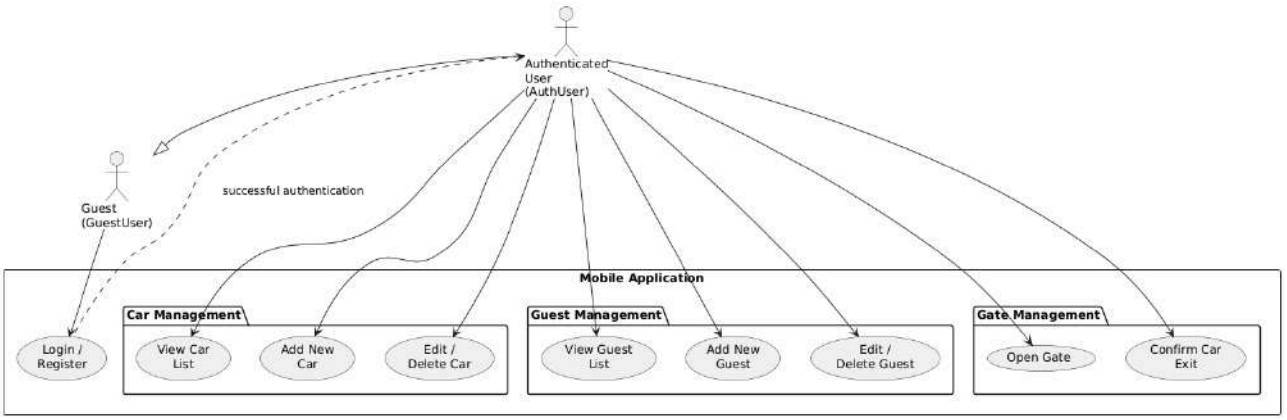
1) The Business Research Company. Smart Parking Systems Global Market Report. https://www.thebusinessresearchcompany.com/report/smart-parking-systems-global-market-report (дата звернення: 25.05.2025).
2) Parklio. Mobile App – Parklio Connect. https://parklio.com/ (дата звернення: 25.05.2025).
3) ParqEx. Smart Parking for Apartments. https://parqex.com/ (дата звернення: 25.05.2025).
4) Smart Spaces. Smart Parking System. https://www.smartspaces.app/smart-parking/ (дата звернення: 25.05.2025).
5) Parkade. App for Managing Parking in Apartments. https://parkade.com/ (дата звернення: 25.05.2025).
6) SQLAlchemy Documentation. https://docs.sqlalchemy.org/en/20/ (дата звернення: 25.05.2025).
7) Alembic Documentation. https://alembic.sqlalchemy.org/en/latest/ (дата звернення: 25.05.2025).
8) JWT.io. Introduction to JSON Web Tokens. https://jwt.io/introduction (дата звернення: 25.05.2025).
9) Passlib Documentation. https://passlib.readthedocs.io/en/stable/ (дата звернення: 25.05.2025).
10) FastAPI. WebSockets. https://fastapi.tiangolo.com/advanced/websockets/ (дата звернення: 25.05.2025).
11) Swagger.io. API Documentation Solutions. https://swagger.io/solutions/api-documentation/ (дата звернення: 25.05.2025).
12) Index.dev. Programming Languages for Mobile App Development. https://www.index.dev/blog/programming-languages-for-mobile-app-development (дата звернення: 25.05.2025).
13) Eltima Software. Mac App Store Stats. https://mac.eltima.com/app-store-stats/ (дата звернення: 25.05.2025).

ДОДАТКИ

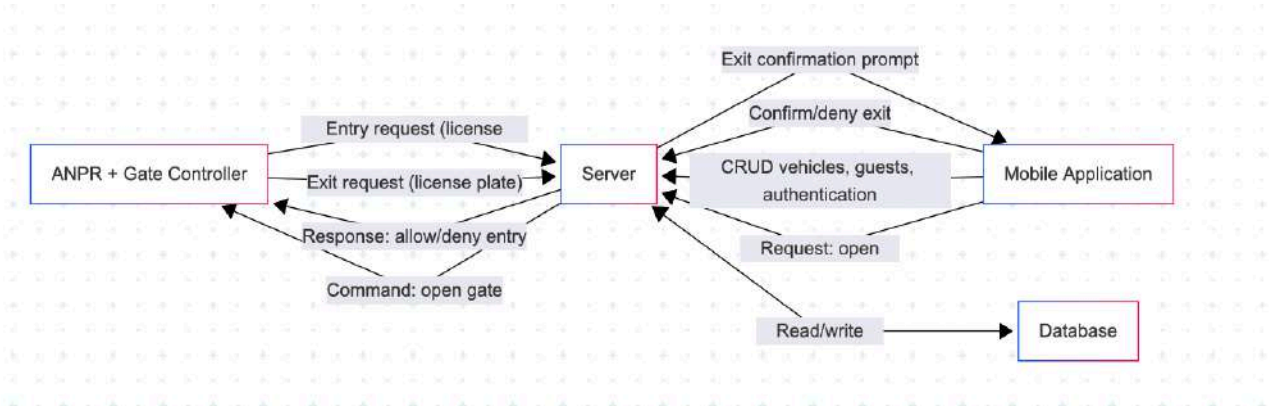
Додаток А

(довідниковий)

Основні сценарії роботи мобільного застосунку



Додаток Б

(довідниковий)

Загальну архітектура взаємодії компонентів системи паркування в застосунку