Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Програмна система для керування платними парковками

(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-10

Марк РАДЬКО
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Наталя КРАВЕЦЬ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

**ЗАВДАННЯ**
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Радько Марку Максимовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для керування платними парковками
Затверджена наказом по університету від 19.05.2025р. №397Ст
2. Термін подання студентом роботи до екзаменаційної комісії 06.06.2025
3. Вихідні дані до роботи Розробити комплексну програмну систему для керування платними парковками, яка скадатиметься з 4 частин: серверна частина, клієнтська частина, мобільний застосунок та IoT пристрій.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

# КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної галузі | 20.05.2025 | *виконано* |
| 2 | Створення специфікації ПЗ | 20.05.2025 – 21.05.2025 | *виконано* |
| 3 | Проектування ПЗ | 21.05.2025 – 23.05.2025 | *виконано* |
| 4 | Розробка ПЗ | 23.05.2025 – 05.06.2025 | *виконано* |
| 5 | Тестування ПЗ | 05.06.2025 – 07.06.2025 | *виконано* |
| 6 | Оформлення пояснювальної записки | 07.06.2025 – 09.06.2025 | *виконано* |
| 7 | Підготовка презентації та доповіді | 10.06.2025 | *виконано* |
| 8 | Попередній захист | 13.06.2025 | *виконано* |
| 9 | Нормоконтроль, рецензування | 13.06.2025 | *виконано* |
| 10 | Здача роботи у електронний архів | 13.06.2025 | *виконано* |
| 11 | Допуск до захисту у зав. кафедри | 13.06.2025 | *виконано* |

Дата видачі завдання «19 » «  квітня  » 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. кафедри ПІ Наталя КРАВЕЦЬ
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

# РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 227 с., 22 рис., 28 табл., 11 додатків, 14 джерел.

БРОНЮВАННЯ, УПРАВЛІННЯ, ПАРКУВАЛЬНА СЕСІЯ, СИСТЕМА, КОРИСТУВАЧІ, ПАРКУВАЛЬНЕ МІСЦЕ, ПАРКІНГ, РОБОЧИЙ ЧАС.

Об’єкт розробки – автоматизована програмна система управління паркуванням.

У фокусі цієї роботи знаходиться вирішення проблеми ефективної організації та адміністрування інфраструктури паркінгів. Зі зростанням кількості автомобілів у містах значно ускладнюється процес управління вільними місцями для паркування, що призводить до заторів, втрати часу та незадоволення користувачів. У зв’язку з цим постає необхідність у створенні автоматизованої програмної системи, яка б забезпечувала раціональне використання паркувального простору та зручну взаємодію з водіями.

Метою проєкту є розробка сучасного цифрового рішення, здатного об'єднувати інформацію про наявні паркомісця, здійснювати облік транспортних засобів, координувати бронювання, контролювати оплату та надавати користувачам можливість у реальному часі переглядати статус паркінгу.

Для реалізації системи в якості серверної платформи використовується Node.js із фреймворком Express. Для збереження та обробки даних застосовується реляційна СУБД PostgreSQL. До системи інтегруються IoT-рішення на базі мікроконтролера ESP32. Веб-клієнт, реалізований на JavaScript з використанням бібліотеки React. Окрему частину становить мобільний додаток на базі Kotlin для Android.

У підсумку, програмний продукт, що буде створений у кваліфікаційної роботи, являтиме собою багатокомпонентну систему, яка поєднує серверну інфраструктуру, IoT-пристрої, веб-платформу та мобільний застосунок.

# ABSTRACT

RESERVATION, MANAGEMENT, PARKING SESSION, SYSTEM, USERS, PARKING SPACE, PARKING LOT, WORKING HOURS.

Object of development – an automated parking management software system.

This thesis focuses on solving the problem of efficient organization and administration of parking infrastructure. With the increasing number of vehicles in urban areas, the process of managing available parking spaces has become significantly more complex, leading to traffic congestion, time loss, and user dissatisfaction. Consequently, there arises a need to develop an automated software system that ensures the rational use of parking space and provides a convenient interaction experience for drivers.

The objective of the project is to develop a modern digital solution capable of integrating information about available parking spots, tracking vehicles, coordinating reservations, processing payments, and allowing users to monitor parking status in real-time.

The system is implemented using Node.js with the Express framework as the backend platform. A relational database management system, PostgreSQL, is used for data storage and processing. The system incorporates IoT solutions based on the ESP32 microcontroller. The web client is developed using JavaScript with the React library. A separate component is a mobile application built with Kotlin for Android.

As a result, the software product created as part of this qualification work will represent a multi-component system combining server infrastructure, IoT devices, a web platform, and a mobile application.

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

HTTP – Hypertext Trancfer Protocol

HTTPS – Hypertext Trancfer Protocol Secure

SSL – Secure Sockets Layer

API – Application Programming Interface

UML – Unified Modeling Language

SSR – Server Side Rendering

HTML – Hypertext Markup Language

CSS – Cascading Style Sheets

SQL – Structured Query Language

UI – User Interface

UX – User Experience

REST – Representational State Transfer

# ВСТУП

У сучасних містах питання організації паркувального простору набуває все більшої актуальності через стрімке зростання кількості транспортних засобів. Нестача вільних місць, неефективне використання наявних ресурсів, відсутність централізованого контролю та незручність для водіїв створюють суттєві труднощі як для керівників паркінгів, так і для звичайних користувачів. Ефективне управління мережею паркінгів потребує впровадження сучасних технологічних рішень, здатних автоматизувати процеси обліку, бронювання, контролю доступу та оплати послуг.

Метою даної роботи є розробка програмної системи для оптимального управління паркувальними майданчиками. Запропоноване рішення передбачає централізоване зберігання та обробку даних про паркінги, паркомісця, транспортні засоби, статус бронювання та інформацію про користувачів. Система забезпечує взаємодію всіх елементів інфраструктури в режимі реального часу, дозволяючи координувати роботу паркінгів і здійснювати моніторинг ключових операцій. До складу системи входять серверна частина з реалізованою бізнес-логікою, адміністративний веб-інтерфейс, IoT-пристрої для автоматизованого обліку вільних місць і контролю доступу, а також мобільний додаток для кінцевих користувачів.

Основними завданнями, які вирішуються в рамках кваліфікаційної роботи, є аналіз предметної області та формування вимог до системи, проєктування архітектури програмного забезпечення, розробка функціональних модулів для бронювання, обліку, оплати й моніторингу, створення користувацьких і адміністративних інтерфейсів, впровадження IoT-рішень для контролю стану паркомісць, а також реалізація засобів безпечного зберігання та передавання даних.

У процесі реалізації використано низку сучасних технологій: для управління базами даних застосовано PostgreSQL; для створення серверної частини — Node.js у поєднанні з фреймворком Express; апаратну частину реалізовано на базі мікроконтролерів ESP32 із сенсорами та модулями ідентифікації; веб-інтерфейс розроблено з використанням React, а мобільний додаток — мовою Kotlin для

платформи Android. Усі компоненти поєднуються в єдину систему, що дозволяє ефективно керувати паркінгом, забезпечуючи прозорість операцій і зручність для користувачів.

Результати цієї роботи можуть бути застосовані в муніципальних паркувальних системах, на приватних паркінгах торговельно-розважальних центрів, бізнес-центрів, аеропортів, у житлових комплексах із обмеженим доступом, а також у межах реалізації концепцій «розумного міста». Запропонована система має потенціал для масштабування й адаптації під різні потреби операторів паркінгів, що робить її актуальним і практично цінним програмним продуктом.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

## 1.1 Аналіз предметної галузі

### 1.1.1 Передумови

Сучасні міста стикаються з численними викликами в управлінні паркувальними просторами. Збільшення кількості транспортних засобів, обмежена кількість паркомісць, динамічні зміни вартості паркування та необхідність ефективного контролю за використанням паркувальних зон створюють значні труднощі для міської інфраструктури. Водії часто витрачають багато часу на пошук вільного місця для паркування, що призводить до заторів, підвищеного рівня забруднення повітря та зниження загальної мобільності.

Існуючі паркувальні системи не завжди враховують змінний попит на паркомісця, що може призводити до нерівномірного завантаження стоянок. Також проблемою є відсутність єдиного підходу до оплати паркування: деякі водії змушені користуватися різними додатками або фізичними паркоматами, що ускладнює процес оплати та створює незручності.

З розвитком цифрових технологій з’являються нові можливості для оптимізації управління паркінгами. Інтелектуальні паркувальні системи дозволяють автоматизувати бронювання місць, прогнозувати завантаженість стоянок, запроваджувати гнучку тарифікацію залежно від попиту та забезпечувати зручну оплату через мобільні додатки. Інтеграція таких рішень допоможе покращити ефективність використання паркувальних просторів, зменшити навантаження на дорожню мережу та підвищити комфорт для водіїв.

### 1.1.2 Аналіз ринку

Ринок паркувальних систем в Україні, у країнах СНД та Європи активно розвивається завдяки збільшенню кількості автомобілів, урбанізації та необхідності ефективного управління паркувальним простором. Основні проблеми, з якими стикаються міста та приватні власники паркінгів, включають нестачу місць, неефективну систему тарифікації, складність процесу оплати та

недостатню інтеграцію з іншими міськими сервісами. У відповідь на ці виклики з’явилося багато цифрових рішень, які автоматизують управління паркінгами.

Одним із найвідоміших сервісів у цій сфері є ParkMobile (https://parkmobile.io). Це американська компанія, що пропонує мобільний додаток для пошуку, бронювання та оплати паркування в містах США та Європи. Система використовує централізовану клієнт-серверну архітектуру, де всі транзакції проходять через хмарний сервер. Це дозволяє водіям переглядати доступні місця в реальному часі, оплачувати паркування через мобільний додаток або веб-інтерфейс і отримувати повідомлення про завершення оплаченої сесії.

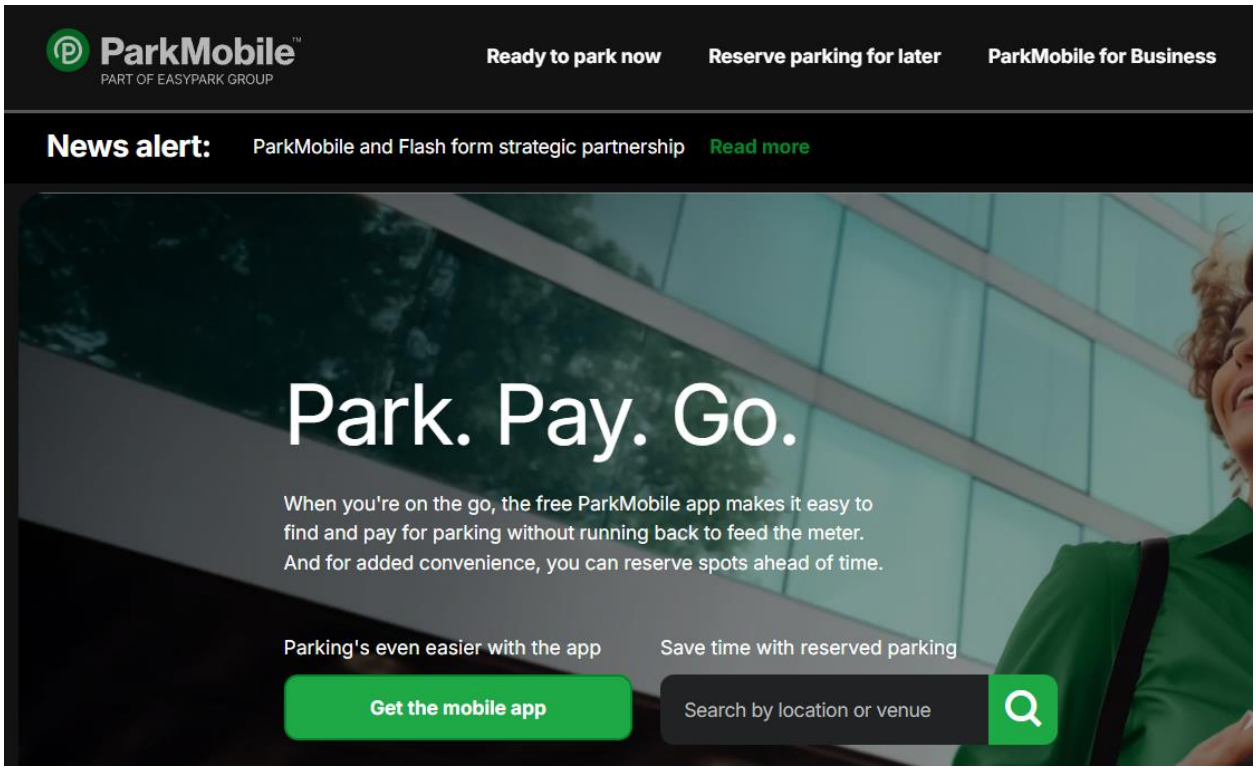


Рисунок 1.1 – Головна сторінка сайту компанії ParkMobile [1]

Інтерфейс ParkMobile максимально спрощений: користувач обирає місце на карті, встановлює час паркування та підтверджує оплату за допомогою збережених платіжних методів. Скріншот головної сторінки сайту наведений на рисунку 1.1. Основний спосіб монетизації – комісія з кожного платежу, що здійснюється через платформу. Додатково сервіс пропонує преміальні функції, такі як автоматичне продовження часу паркування або бронювання місця заздалегідь. ParkMobile став

популярним завдяки простоті використання та широкій інтеграції з муніципальними паркінгами, однак сервіс має недоліки: відсутність динамічного ціноутворення, що не дозволяє гнучко регулювати вартість залежно від попиту, а також високі комісії, які можуть робити паркування дорожчим для користувачів.

EasyPark – європейський сервіс (https://www.easyparkgroup.com), який працює в понад 20 країнах і дозволяє водіям оплачувати паркування через мобільний додаток. Архітектура EasyPark заснована на хмарному рішенні з інтеграцією в муніципальні паркінги, що дає змогу легко впроваджувати систему у великих містах. Користувач може знайти паркомісце, оплатити його через додаток і навіть отримати прогноз щодо завантаженості паркінгу. Основні переваги сервісу – простота інтерфейсу, висока швидкість обробки платежів і підтримка різних платіжних методів. Монетизація здійснюється через комісії з паркувальних платежів, а також через партнерські інтеграції з містами та операторами паркінгів. Недоліками системи є відсутність можливості бронювання паркомісця заздалегідь, а також відсутність алгоритмів динамічного ціноутворення, які дозволяли б регулювати тарифи залежно від часу доби та попиту. Зображення головної сторінки сайту наведено на рисунку 1.2.

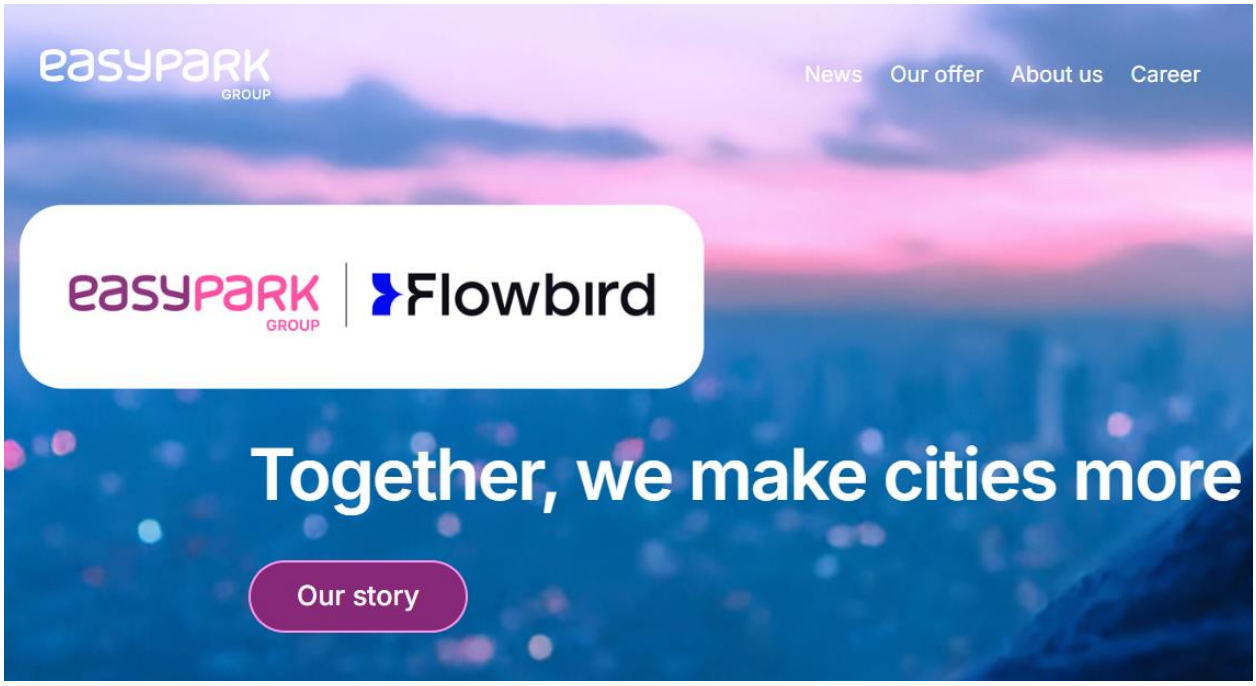


Рисунок 1.2 – Головна сторінка сайту компанії EasyPark [2]

Ще одним цікавим конкурентом є Parclick (https://parclick.com) – система, яка пропонує бронювання паркомісць у містах Європи, особливо в аеропортах, біля вокзалів і туристичних зон. Вона відрізняється від інших сервісів тим, що працює не тільки з муніципальними, але й з приватними паркінгами. Архітектурно Parclick використовує веб-платформу та мобільний додаток, які підключені до бази паркінгів, що підтримують онлайн-бронювання. Це дозволяє власникам стоянок здавати свої місця в оренду, а користувачам – резервувати їх за вигідною ціною. Монетизація Parclick включає комісійні з кожного бронювання, а також партнерські програми з операторами паркінгів. Основними перевагами є можливість бронювання місця заздалегідь, інтеграція з великими транспортними вузлами та конкурентні ціни завдяки системі знижок. Однак система має й недоліки: вона більше орієнтована на туристів і менш ефективна для щоденного міського паркування, а також вимагає від користувача попередньої реєстрації та внесення передоплати. Головна сторінка сайту наведена на рисунку 1.3.

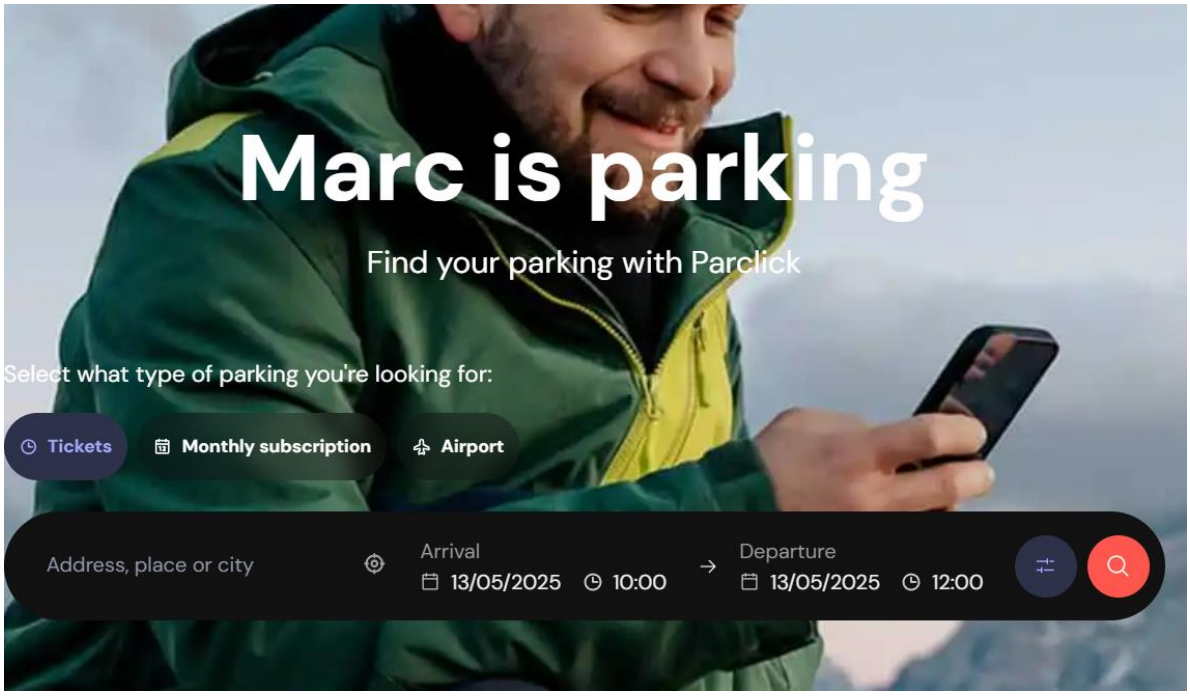


Рисунок 1.3 – Головна сторінка сайту компанії Parkclick [3]

JustPark – британська компанія (https://www.justpark.com), що спеціалізується на бронюванні паркомісць як у муніципальних, так і в приватних паркінгах.

Система працює у Великобританії та деяких інших європейських країнах, дозволяючи користувачам знаходити місця для паркування через мобільний додаток або веб-інтерфейс. Архітектура JustPark заснована на хмарному рішенні з інтеграцією в міську паркувальну інфраструктуру, що дозволяє автоматизувати управління місцями та спростити процес оплати. Одна з ключових особливостей сервісу – можливість оренди паркомісць не тільки у міських паркінгах, але й у приватних осіб, які можуть здавати у користування власні гаражі, під'їзні шляхи чи вільні ділянки землі. Це створює гнучку та доступну систему паркування, яка враховує нестачу місць у завантажених міських районах. Інтерфейс JustPark простий та інтуїтивний: користувач вводить адресу або назву місця, де хоче припаркуватися, отримує список доступних варіантів з цінами та може миттєво оформити бронювання. Додаток також підтримує функцію навігації до обраного місця та нагадування про завершення паркувального часу.

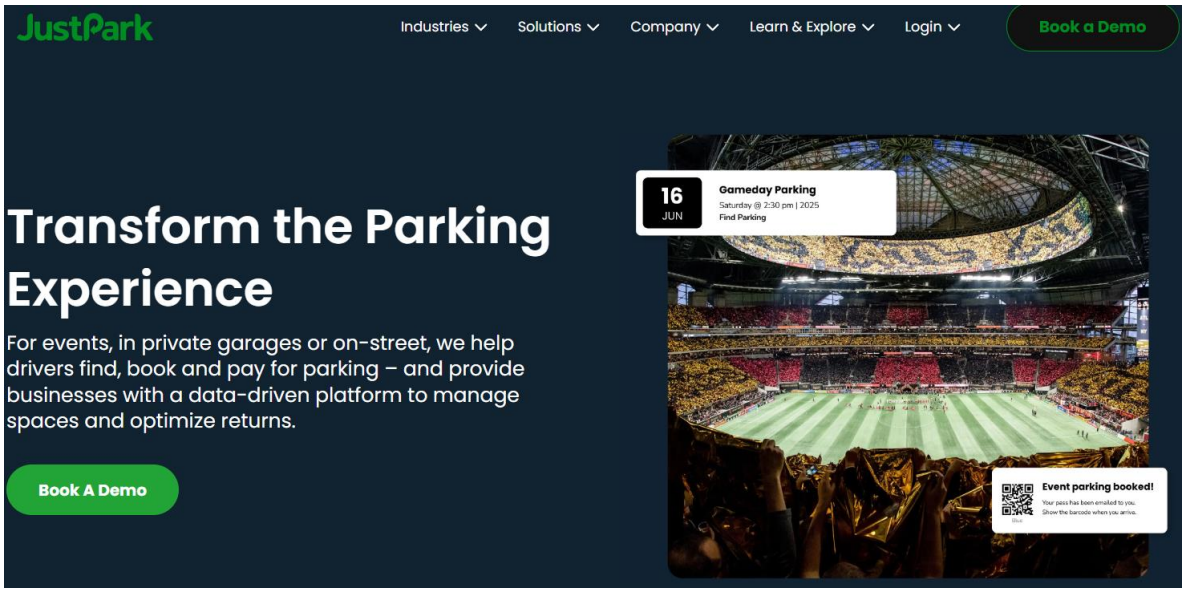


Рисунок 1.4 – Головна сторінка сайту компанії JustPark [4]

Стратегії монетизації JustPark включають кілька джерел доходу. По-перше, сервіс стягує комісію з водіїв за кожне бронювання. По-друге, власники паркінгів і приватні особи сплачують відсоток від прибутку за використання платформи. Крім того, компанія співпрацює з бізнесами, пропонуючи паркування для корпоративних клієнтів і фірмових автопарків. JustPark досяг великого успіху

завдяки своїй моделі, що поєднує традиційне міське паркування з принципами спільного використання ресурсів (sharing economy). Платформа виграла численні нагороди за інноваційність і зручність використання. На рисунку 1.4 наведений скріншот з веб - платформи. Основними перевагами є широка мережа партнерських паркінгів, можливість оренди приватних місць та проста система бронювання. Недоліки включають обмежене покриття за межами Великобританії, можливі проблеми з доступністю заброньованих місць (через людський фактор) і відсутність динамічного ціноутворення, яке могло б оптимізувати використання паркувальних зон залежно від попиту. JustPark є потужним конкурентом на ринку паркувальних сервісів, особливо в сегменті бронювання приватних місць. Однак його модель більше підходить для ринку Великобританії та менш ефективна в регіонах із жорстким державним регулюванням паркування.

### 1.1.3 Потреби клієнтів та ринку

Для успішного впровадження інноваційної паркувальної системи в межах сучасного міського середовища необхідно враховувати потреби як ринку, так і кінцевих користувачів — операторів паркінгів та водіїв. Їхні очікування формуються під впливом загальної цифровізації, потреби у зручності, безпеці та економічній ефективності.

#### 1.1.3.1 Потреби ринку

Сучасний ринок паркувальних послуг вимагає інноваційних підходів, орієнтованих на гнучкість та можливість масштабування. Малі приватні паркінги, муніципальні автостоянки та великі багаторівневі комплекси очікують єдиних рішень, здатних адаптуватися до різних моделей управління. Існує чітка потреба в ефективнішому використанні ресурсів, зниженні операційних витрат та підвищенні конкурентоспроможності на фоні активної урбанізації.

Також зростає зацікавленість в інтеграції паркувальної інфраструктури до концепції «розумного міста», де управління транспортними потоками,

інформаційна взаємодія з муніципальними системами та екологічні чинники мають першочергове значення.

Серед ключових ринкових запитів можна виокремити:

– зменшення витрат на обслуговування паркінгів;

– автоматизацію контролю за завантаженістю;

– впровадження динамічного ціноутворення;

– забезпечення цифрових каналів взаємодії з клієнтами;

– доступність аналітики для прийняття управлінських рішень;

– екологізацію міського середовища шляхом зниження зайвих переміщень у пошуках вільного місця.

1.1.3.2 Потреби кінцевих користувачів (водіїв)

Сучасні водії стикаються з низкою труднощів у щоденному користуванні паркінгами, серед яких — тривалий пошук вільного місця, непрозоре ціноутворення, незручні способи оплати, відсутність гарантії наявності місця тощо. У зв’язку з цим користувачі очікують:

– швидкий пошук доступного паркувального місця з можливістю прокладання оптимального маршруту;

– попереднє бронювання, яке зменшує стрес та час на пошук;

– зручні способи оплати, включно з безконтактними методами (Google Pay, Apple Pay тощо);

– своєчасні сповіщення про завершення часу паркування;

– доступ до актуальної інформації щодо тарифів та заповненості паркінгу в режимі реального часу;

– гарантовану наявність місця після бронювання;

– інформацію про безпеку паркінгу — наявність охорони, камер, освітлення;

– можливість дистанційного продовження паркування через мобільний застосунок.

1.1.3.3 Потреби кінцевих користувачів (власників)

Представники бізнес-сфери прагнуть мінімізувати ручне управління процесами та перейти до максимально автоматизованих моделей. Їхні очікування включають:

– гнучке управління тарифами залежно від часу доби та попиту;

– інтеграцію з системами розпізнавання номерів для автоматизації доступу;

– збір аналітики для моніторингу ефективності;

– налаштування паркомісць під різні категорії користувачів (жителі, служби доставки тощо);

– виявлення порушень правил паркування;

– сумісність з існуючими системами управління, що вже використовуються на об'єктах;

– зниження впливу людського фактора за рахунок автоматизації операцій.

1.1.4 Профілі зацікавлених сторін

Для ефективної реалізації та впровадження програмної системи необхідно визначити основних зацікавлених учасників процесу та врахувати їхні потреби. У таблиці 1 наведено профілі ключових сторін, їхнє ставлення до проекту, основні інтереси та можливі обмеження, які слід враховувати під час розробки.

Таблиця 1 – Профілі зацікавлених сторін (таблиця виконана самостійно)

| Зацікавлена сторона | Основна цінність | Ставлення | Основні інтереси | Обмеження |
|---|---|---|---|---|
| Власники паркінгів | Оптимізація роботи, збільшення прибутковості | Позитивне, прагматичне | Підвищення ефективності управління, мінімізація | Законодавчі обмеження, необхідність модернізації |

Продовження таблиці 1

| Зацікавлена сторона | Основна цінність | Ставлення | Основні інтереси | Обмеження |
|---|---|---|---|---|
| | | | витрат | |
| Адміністратори паркінгів | Ефективне керування паркінгом, зручний інтерфейс | Позитивне, практичне | Контроль за використанням місць, автоматичні звіти | Застаріле апаратне забезпечення |
| Водії | Зручність використання, економія часу | Позитивне, вимогливе, практичне | Доступні місця, швидка оплата, безпека | Проблеми з інтернет-з'єднанням, платіжні обмеження |
| Інвестори та партнери | Рентабельність інвестицій, стабільний розвиток | Позитивне, аналітичне | Прозорість фінансів, масштабування системи | Ринкові коливання, ризик кібератак |
| Інвестори та партнери | Рентабельність інвестицій, стабільний розвиток | Позитивне, аналітичне | Прозорість фінансів, масштабування системи | Ринкові коливання, ризик кібератак |
| Маркетологи та відділ продажу | Залучення нових користувачів | Позитивне, амбітне | Аналітика ринку, рекламні кампанії | Конкуренція, зміни у попиті |

Продовження таблиці 1

| Зацікавлена сторона | Основна цінність | Ставлення | Основні інтереси | Обмеження |
|---|---|---|---|---|
| Юридичний відділ | Дотримання законодавства, захист конфіденційної інформації | Позитивне, суворе | Відповідність нормативним вимогам | Законодавчі зміни, GDPR та інші регуляції |
| Постачальники IT-рішень та обладнання | Стабільна співпраця, зростання попиту | Позитивне, співробіт-ницьке | Інтеграція з іншими системами, чіткі технічні вимоги | Технічні обмеження, сумісність обладнання |

## 1.2 Виявлення та вирішення проблем

### 1.2.1 Бізнес – можливості

У сфері інтелектуального паркування існує потенціал для побудови прибуткової бізнес-моделі, що поєднує B2B- та B2C-підходи. Основними джерелами доходів можуть бути:

а) Партнерські програми з операторами паркінгів.

Оператори комерційних та муніципальних паркувальних майданчиків мають зацікавленість у використанні цифрових рішень для автоматизації управління, покращення аналітики та підвищення ефективності. Монетизація у цьому напрямку можлива через запровадження підписних тарифних планів, що включають базовий, розширений та індивідуальний рівні. Це дозволяє масштабувати модель для компаній різного розміру та типу. Перевагами для операторів є:

1) доступ до цифрових панелей управління;

2) інструменти прогнозування й аналітики;

3) автоматизація процесів зменшення витрат.

б) Комісійна модель у транзакціях з користувачами.

Зі зростанням популярності мобільних додатків для оплати послуг паркування ефективною є модель стягнення невеликої комісії з кожної транзакції. Такий підхід дозволяє отримувати змінний дохід, який напряму залежить від активності користувачів. При цьому розмір комісії має бути конкурентним щодо стандартних ставок фінансових посередників.

Поєднання двох моделей дозволяє створити збалансований потік доходів:

1) підписка забезпечує стабільність (навіть при низькій активності користувачів);

2) комісійна модель масштабована і зростає разом з популярністю рішення;

3) оператори отримують як додаткову клієнтську базу, так і ефективні засоби управління.

1.2.1.1 Ринкові передумови для реалізації

Ринки України, країн Європи та СНД демонструють високий потенціал для розвитку таких бізнес-моделей через сукупність економічних, соціальних та технологічних факторів:

а) Високий рівень урбанізації та автомобілізації.

У великих містах спостерігається значна концентрація транспортних засобів, що створює попит на оптимізацію паркувального простору.

б) Розвиток концепцій «розумного міста».

Цифровізація міської інфраструктури, підтримувана державними ініціативами, відкриває можливості для впровадження інтелектуальних систем управління транспортом, зокрема паркуванням.

в) Низька ефективність існуючої паркувальної інфраструктури.

Багато міст стикаються з проблемами хаотичного паркування, перевантаженням центрів, заторами та екологічним тиском, що створює попит на системні рішення.

г) Зростаюча популярність мобільних сервісів.

Користувачі все частіше віддають перевагу мобільним додаткам для пошуку та оплати послуг, що підвищує привабливість цифрових платформ.

Додаткові драйвери ринку:

– державне регулювання та стимулювання цифрових ініціатив;

– зростання екологічної свідомості;

– економічна доцільність впровадження інтелектуальних систем для міст та операторів.

Таким чином, загальна ринкова ситуація, потреба в автоматизації, інтерес до цифрових сервісів та зростаюча кількість транспортних засобів створюють сприятливе середовище для впровадження і розвитку бізнесу, пов’язаного з інтелектуальними системами управління паркуванням.

### 1.2.2 Бізнес – ризики

У процесі розробки, впровадження та комерціалізації цифрової платформи Park4Flow можуть виникати низка бізнес-ризиків, що потенційно впливають на її ефективність, фінансову стабільність та конкурентоспроможність. Для забезпечення стійкого розвитку продукту ці ризики класифікуються за наступними категоріями:

а) технічні ризики:

1) Невідповідність функціональним вимогам. Існує ймовірність, що кінцевий програмний продукт не забезпечить повну реалізацію заявлених функцій (зокрема, обробку платежів, інтеграцію з картографічними сервісами чи точність розрахунку тарифів), що може негативно позначитися на сприйнятті системи ринком.

2) Складнощі при інтеграції з зовнішніми системами. Інтеграція з існуючими рішеннями (контроль доступу, платіжні шлюзи, транспортні

платформи, системи розпізнавання номерів) може виявитися складною через технічну або нормативну несумісність.

3) Нестабільність роботи та технічні збої. Помилки в роботі платформи, збої під час бронювання або неточні нарахування можуть спричинити втрату довіри користувачів і, відповідно, — втрати доходів.

4) Недостатня продуктивність у пікові навантаження. У разі великого обсягу одночасних запитів (особливо в години пік), система повинна забезпечувати високу швидкість обробки даних без втрати стабільності.

5) Загрози інформаційній безпеці. Платформа оперує персональними та платіжними даними, тому піддається потенційним кіберзагрозам, витоку інформації та правовим ризикам.

б) фінансові ризики:

1) Перевищення бюджету проєкту. Фактичні витрати на розробку, тестування та запуск продукту можуть виявитися вищими за початкові розрахунки, що створює додаткове навантаження на бюджет компанії.

2) Низький рівень окупності інвестицій. Недостатній попит на продукт або відсутність готовності користувачів платити за послуги може призвести до тривалого періоду повернення інвестицій.

3) Відтермінована монетизація. Попри успішний запуск, отримання стабільного доходу може бути затримане через потребу у формуванні критичної маси користувачів.

4) Висока вартість експлуатаційної підтримки. Поточні витрати на технічне обслуговування, клієнтську підтримку, безпеку та правове забезпечення можуть бути значними та потребувати постійного фінансування.

в) ризики ринку та впровадження:

1) Висока конкуренція в галузі. На ринку вже функціонують подібні рішення, включно з муніципальними платформами. Конкурентна боротьба може ускладнити просування Park4Flow серед користувачів і партнерів.

2) Недовіра з боку операторів паркінгів. Інфраструктурні партнери можуть не підтримати впровадження нового рішення через побоювання щодо технічної складності або нестабільності роботи.

3) Обмежена готовність користувачів до цифрових рішень. Деякі водії можуть виявити небажання переходити на новий формат взаємодії з паркінгом, зокрема в частині електронних оплат або онлайн-бронювання.

4) Зміни у регуляторному середовищі. Регламентуючі норми у сфері транспорту, безпеки даних та міської інфраструктури можуть змінюватися, що вимагатиме адаптації бізнес-моделі та технічних рішень.

5) Проблеми з ліцензуванням технологій. Використання певних рішень (наприклад, відеоспостереження, автоматичне розпізнавання номерних знаків) може потребувати спеціальних дозволів або державної сертифікації.

6) Ускладнення під час виходу на міжнародні ринки. Розширення за межі внутрішнього ринку передбачає врахування локальних норм, платіжних систем, мовної адаптації та культурних особливостей.

г) Ризики, пов’язані з користувачами:

1) Необхідність навчання персоналу операторів паркінгів. Впровадження платформи може вимагати додаткових витрат часу і ресурсів на навчання персоналу та зміну бізнес-процесів.

2) Відмова користувачів від застосунку. Частина потенційних користувачів може відмовитися від встановлення додатка або реєстрації в системі через недовіру до цифрових сервісів.

3) Технічні обмеження у користувачів. Наявність застарілих мобільних пристроїв або нестабільне підключення до Інтернету можуть ускладнити використання функціоналу платформи.

4) Обмежена підтримка популярних платіжних методів. У разі відсутності підтримки локальних або альтернативних способів оплати система може бути менш привабливою для частини аудиторії.

## 1.3 Постановка задачі

У сучасних умовах стрімкого розвитку міської інфраструктури, зростання кількості транспортних засобів та обмеженості паркувального простору виникає нагальна потреба у впровадженні ефективних цифрових рішень для управління паркуванням. Існуючі системи часто не забезпечують належного рівня зручності, автоматизації та інтеграції з іншими міськими сервісами. Це призводить до перевантаження вулиць, нераціонального використання ресурсів і низької задоволеності водіїв.

Завдання полягає у створенні інтелектуальної багатофункціональної платформи Park4Flow, яка дозволяє автоматизувати процес бронювання, оплати, моніторингу та аналітики паркувального простору як для користувачів (водіїв), так і для операторів паркінгів. Система має забезпечити високу надійність, масштабованість, захист персональних даних та інтеграцію з зовнішніми сервісами (картографічними, платіжними, аналітичними).

### 1.3.1 Цілі та критерії успіху

Розробка інтелектуальної системи управління паркуванням Park4Flow має на меті створення ефективного цифрового інструменту, здатного оптимізувати використання міського паркувального простору, підвищити зручність для кінцевих користувачів (водіїв) та забезпечити операторів паркінгів засобами автоматизованого адміністрування. Враховуючи динаміку розвитку міської інфраструктури, стрімке зростання кількості транспортних засобів та обмеженість ресурсів, проєкт орієнтований на цифрову трансформацію процесів паркування, забезпечення прозорої взаємодії між усіма учасниками паркувальної екосистеми та досягнення стабільних економічних результатів.

З огляду на цю мету, сформульовано низку бізнес-цілей та критеріїв успіху, реалізація яких дозволить оцінити ефективність проєкту та визначити його практичну цінність для цільових груп.

а) Бізнес – цілі:

1) Оптимізація управління паркінгами та підвищення їх ефективності. Забезпечення операторів інструментами для управління паркомісцями, аналізу завантаженості та гнучкого ціноутворення сприятиме раціональному використанню ресурсів та зростанню доходів.

2) Покращення користувацького досвіду та задоволеності водіїв.Інтуїтивний інтерфейс, зручне бронювання, безконтактна оплата та навігація до паркомісця сприятимуть зменшенню часу пошуку паркування та покращенню вражень користувачів.

3) Забезпечення безпеки даних та надійності роботи системи. Імплементація сучасних механізмів шифрування, резервного копіювання та відповідність міжнародним стандартам кібербезпеки гарантують захист особистої та фінансової інформації користувачів.

4) Гнучкість і масштабованість програмної архітектури. Платформа має адаптуватися до потреб різних типів паркінгів — від невеликих приватних до великих муніципальних, забезпечуючи легке розширення функціоналу та вихід на нові ринки.

5) Впровадження аналітичних інструментів для операторів. Розширена звітність щодо заповненості паркінгу, середньої тривалості перебування, фінансових показників тощо дозволить операторам приймати обґрунтовані рішення щодо оптимізації діяльності.

6) Економічна ефективність та швидке повернення інвестицій. Автоматизація адміністративних процесів, цифровізація розрахунків та зниження витрат забезпечать фінансову доцільність проєкту як для приватних, так і для державних партнерів.

б) Критерії оцінки успіху проекту:

1) Досягнення окупності проєкту протягом першого року.Очікується, що завдяки гнучкій фінансовій моделі (підписка + комісія), платформа компенсує не менше 20% витрат на розробку в перший рік.

2) Підписання договорів із мінімум 50 операторами паркінгів.Вихід на ринок трьох найбільших міст України (Київ, Харків, Дніпро) протягом першого року забезпечить критичну масу для подальшого масштабування.

3) Залучення не менше 10 000 активних користувачів.Регулярне використання сервісу водіями для бронювання та оплати паркування буде ключовим показником затребуваності платформи.

4) Досягнення завантаженості паркінгів на рівні 80% у пікові години.Покращення завдяки аналітичним інструментам і динамічному ціноутворенню дозволить підвищити ефективність використання інфраструктури.

5) Підвищення користувацької задоволеності на ≥25% за перші 6 місяців. Оцінюватиметься за допомогою Net Promoter Score (NPS) та зворотного зв’язку з користувачами в мобільному застосунку.

6) Скорочення середнього часу пошуку паркомісця на ≥40%. Інтеграція картографічного сервісу та прогнозування доступності дозволять зменшити втрати часу та пального.

7) Зменшення операційних витрат операторів щонайменше на 30%. Автоматизація адміністрування, звітності та прийому платежів значно знизить витрати на персонал та інфраструктуру.

8) Отримання сертифікації ISO 27001 упродовж першого року. Це підвищить довіру до платформи серед партнерів та кінцевих користувачів.

9) Вихід на міжнародний ринок на третьому році реалізації. Першочерговою країною для масштабування визначено Польщу, як ринок із високим попитом на автоматизовані рішення для паркування.

### 1.3.2 Пріоритети проекту

Успішна розробка програмної системи вимагає чіткого визначення основних пріоритетів, які забезпечать ефективне впровадження та використання продукту. У таблиці 2 наведено основні показники виконання проекту, етапи їх реалізації, обмеження та допустимі відхилення.

Таблиця 2 – Пріоритети проекту (таблиця виконана самостійно)

| Показник виконання | Етапи | Обмеження (граничні значення) | Ступінь свободи (допустимий діапазон) |
|---|---|---|---|
| Термін впровадження | Початок розробки – лютий 2025 року | Впровадження не пізніше 30 червня 2025 року | Затримка не допускається |
| Функціональність системи | Впровадження основних функцій | Усі ключові функції, прописані в рамках первинного випуску, мають бути реалізовані | Можливе відкладення другорядних функцій |
| Безпека даних | Впровадження механізмів захисту | Відсутність витоків даних | Вразливості системи не допускаються |
| Кількість користувачів | Маркетингова кампанія, залучення клієнтів | Мінімум 8000 користувачів за перший рік | Допускається зменшення до 20% від запланованої кількості користувачів |

Продовження таблиці 2

| Показник виконання | Етапи | Обмеження (граничні значення) | Ступінь свободи (допустимий діапазон) |
|---|---|---|---|
| Якість продукту | тестування | Відсутність критичних помилок | Допускаються незначні баги, що не заважають роботі системи |
| Технічна підтримка | Організація служби підтримки | Відповідь на запити користувачів до 6 годин | Допускається збільшення до 24 годин у часи завантаженності |
| Бюджет проекту | Планування бюджету, розподіл ресурсів | Бюджет - $40 | Допустиме перевищення до 20% |

# 2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

## 2.1 Концепція рішення

### 2.1.1 Окреслення концепції

Метою проєкту Park4Flow є створення інноваційної програмної системи для управління паркінгами, що дозволить власникам паркінгів, операторам та водіям ефективно взаємодіяти у цифровому середовищі.

Система забезпечить гнучке управління паркомісцями, автоматизовану тарифікацію, підтримку онлайн-бронювання, а також динамічне ціноутворення на основі попиту та завантаженості. Для водіїв буде доступна зручна навігація до вільних місць, швидка оплата через мобільний додаток і можливість отримання сповіщень щодо стану їх паркування. Оператори паркінгів отримають доступ до аналітики, гнучкого налаштування тарифів, управління місцями та автоматизації процесів обліку та контролю.

### 2.1.2 Головна функціональність

Програмна система керування паркінгами Park4Flow розробляється з метою забезпечення ефективного управління паркувальними зонами, автоматизації процесів оплати, оптимізації використання паркомісць та підвищення зручності для водіїв і операторів паркінгів. Функціональність системи включає наступні основні модулі.

а) Управління користувачами:

1) реєстрація та авторизація користувачів із розмежуванням доступу за ролями (водій, оператор, адміністратор);

2) функціонал для відновлення доступу до облікового запису;

3) механізм блокування та розблокування акаунтів у разі порушень або технічних потреб;

4) забезпечення конфіденційності та цілісності персональних даних відповідно до чинного законодавства та політики безпеки.

б) Управління паркінгами та місцями:

1) реєстрація паркінгів у системі з перевіркою підписки та базовою модерацією;

2) можливість редагування профілю паркінгу (локація, кількість місць, тарифи, години роботи);

3) гнучке керування тарифами залежно від навантаження та попиту;

4) візуалізація реального стану паркомісць на інтерактивній карті;

5) автоматичне оновлення інформації за допомогою IoT-датчиків (в базовій реалізації).

в) Оплата та бронювання:

1) функціонал попереднього бронювання паркомісць через веб-інтерфейс та мобільний застосунок;

2) підтримка безготівкових способів оплати: банківські картки, Apple Pay, Google Pay, електронні гаманці;

3) гнучкі моделі оплати:

- оплата за фактичний час користування місцем;
- від'ємний баланс із можливістю погашення заборгованості;
- оплата бонусами;
- часткова оплата з переходом у кредитний ліміт;

4) генерація електронних квитанцій та автоматизоване списання коштів з рахунку користувача.

г) Аналітика та управління доходами:

1) автоматизоване формування фінансової звітності для операторів паркінгів;

2) моніторинг рівня заповненості паркінгів у режимі реального часу;

3) прогнозування попиту на паркомісця з використанням історичних даних;

4) побудова аналітичних візуалізацій для прийняття управлінських рішень.

г) Сповіщення та комунікація:

1) автоматичні нагадування для водіїв про завершення оплачуваного часу;

2) повідомлення про заборгованість та необхідність здійснити оплату;

3) сповіщення операторів про наближення до граничного рівня заповненості.

д) Інтеграції та API:

1) підключення IoT-датчиків та систем розпізнавання номерних знаків;

2) відкрите API для взаємодії з зовнішніми сервісами (таксі, подорожі, міські платформи);

3) інтеграція моделей прогнозування на основі нейромереж для окремих паркінгів;

4) застосування API платіжних систем LiqPay та PayPal для обробки транзакцій.

е) Підтримка та безпека:

1) реалізація двофакторної автентифікації для підвищення рівня захисту акаунтів;

2) автоматизоване резервне копіювання даних системи;

3) механізми контролю доступу до конфіденційної інформації відповідно до ролей;

4) впровадження сертифікованих протоколів захисту для обробки персональних та платіжних даних.

## 2.2 Рамки та обмеження

### 2.2.1 Рамки первинного випуску

Початковий реліз Park4Flow реалізує базову функціональність, необхідну для запуску повноцінного MVP (minimum viable product) рішення для керування паркувальними просторами. Включено основні компоненти серверної частини, клієнтського веб-інтерфейсу, мобільного застосунку та IoT-інтеграцій.

а) Серверна частина:

1) Реєстрація користувачів – створення акаунтів усіма типами користувачів. Бізнес-клієнти можуть створювати облікові записи адміністраторів після підтвердження підписки.

2) Авторизація – вхід до системи через email та пароль.

3) Оформлення підписки – адміністратори паркінгів активують паркінг після оплати.

4) Реєстрація паркінгів – додавання паркінгів з описом, тарифами, координатами та іншою інформацією.

5) Додавання паркомісць – налаштування типів місць та сумісних категорій транспортних засобів.

6) Налаштування тарифів – підтримка фіксованої та динамічної тарифікації.

7) Облік паркувальних сесій – збереження даних про тривалість, оплату, метод оплати.

8) Управління боргами – сповіщення про борги та терміни їх погашення.

9) Інтеграція з платіжними системами – підтримка LiqPay, PayPal.

10) Шифрування персональних даних – AES-256 для збереження, TLS 1.3 для передачі.

11) Резервне копіювання – щоденні резервні копії бази даних.

12) API для мобільного застосунку та IoT – REST API для інтеграцій.

13) Підтримка часових поясів – збереження часу в UTC, локалізація для клієнтів.

14) Блокування користувачів – автоматичне блокування за борги, ручне – за порушення.

15) Бонусна система – накопичення та витрата бонусів водіями.

16) Система бронювання – резервування паркомісць на певний час.

17) Реєстрація транспортного засобу – можливість додати транспортні засоби до профілю.

18) Локалізація мови – підтримка інтернаціоналізації.

19) Оплата частинами – дозвіл на дооплату при нестачі коштів.

20) Керування даними – інтерфейс адміністрування системних даних.

21) Баланс користувача – поповнення балансу для зручної оплати.

22) Система сповіщень – інформування про борги, завершення сесій, технічні події.

23) AI-прогнози завантаженості – аналітика попиту на паркінги на основі попередньої інформації з бази даних.

24) Відновлення доступу – функція зміни паролю.

б) Клієнтська частина:

1) Панель адміністратора паркінгу – статистика, тарифи, керування повідомленнями.

2) Локалізація інтерфейсу – підтримка декількох мов.

3) Онлайн-оплата паркування – через баланс чи платіжні сервіси.

4) Історія паркувань – квитанції, попередні сесії, борги.

5) Повідомлення для водіїв – завершення часу, заборгованість, оновлення тарифів.

6) Панель адміністратора системи – керування користувачами, даними, бекапами.

7) Форма реєстрації – створення облікових записів.

8) Форма авторизації – вхід за email і паролем.

9) Відновлення паролю – відновлення доступу до акаунта.

10) Оплата підписки – можливість оплати з клієнтського інтерфейсу.

11) Реєстрація паркінгу – вказання даних про паркінг та паркомісця.

12) Правила користування – доступна сторінка з правилами для користувачів.

в) Мобільний застосунок:

1) Реєстрація та авторизація – створення та вхід до акаунта з додатку.

2) Бронювання місць – пошук та бронювання паркомісць.

3) Онлайн-оплата – підтримка Apple Pay, Google Pay, карток.

4) Push-сповіщення – завершення бронювання, борги, оновлення тарифів.

5) Навігація до паркінгу – інтеграція з Google Maps.

6) Історія паркувань – перегляд квитанцій, бронювань.

7) Обране – додавання паркінгів до списку улюблених.

8) Локалізація – підтримка багатомовного інтерфейсу.

9) Темна тема – перемикання між темним та світлим інтерфейсом.

г) IoT-пристрої та обладнання:

1) Керування шлагбаумами – відкриття при підтвердженому бронюванні/оплаті.

2) QR-коди для бронювання – сканування коду для підтвердження на місці.

3) Сенсори зайнятості – визначення стану паркомісць у реальному часі.

### 2.2.2 Рамки наступних випусків

Для забезпечення подальшого розвитку та розширення функціональних можливостей програмної системи управління паркінгами Park4Flow, у наступних випусках планується реалізувати такі функції та покращення:

– Розширення підтримки IoT-пристроїв [5] — інтеграція з додатковими сенсорами зайнятості, камерами розпізнавання номерних знаків та автоматичними шлагбаумами для підвищення автоматизації.

– Інтеграція з міськими системами управління паркінгами — синхронізація з муніципальними, державними та приватними паркінгами для єдиного управління.

– Покращення алгоритмів динамічного ціноутворення — врахування погодних умов, трафіку, подій у місті та часу доби для розрахунку тарифів.

– Програма лояльності — впровадження системи бонусів, знижок та акцій для постійних клієнтів.

– Підключення додаткових платіжних шлюзів — підтримка Revolut, криптовалютних гаманців та нових банківських рішень.

– Відкрите API — надання можливості стороннім розробникам та міським сервісам інтегруватися з Park4Flow.

– Автоматизована аналітика — впровадження аналітичних модулів для прогнозування заповненості, побудови звітів та оптимізації тарифів за допомогою штучного інтелекту.

– Виставлення рахунків для корпоративних клієнтів — автоматизація підготовки інвойсів, контроль оплат і ведення звітності.

– Контроль часу паркування — система попередження про перевищення ліміту стоянки з можливістю подовження сесії через застосунок.

– Спеціалізоване бронювання для електромобілів — бронювання місць з зарядними станціями та інтеграція з сервісами зарядки.

– AI-помічник — розробка модуля штучного інтелекту для аналізу попиту, заповненості та автоматичного коригування тарифів.

– Інтеграція з державними системами боржників — автоматичне виявлення боржників і блокування доступу до сервісу в разі критичної заборгованості.

– Корпоративні акаунти — функціональність для компаній, які орендують декілька місць для співробітників, з можливістю керування користувачами та квотами.

– Система рейтингів та відгуків — користувачі зможуть залишати оцінки та коментарі до паркінгів, що сприятиме покращенню якості обслуговування.

– Альтернативні методи оплати — додавання можливості здійснювати оплату через SMS та USSD-коди для користувачів без доступу до інтернету або смартфонів.

### 2.2.3 Обмеження та виключення

Програмна система Park4Flow має низку технічних, організаційних, правових та функціональних обмежень, які необхідно враховувати при її впровадженні, експлуатації та масштабуванні. Також існують виключення з гарантій, що визначають сфери, у яких відповідальність постачальника програмного забезпечення обмежена.

а) Технічні обмеження:

1) Залежність від інтернет-з’єднання. Стабільне підключення до мережі Інтернет є критично необхідним для повноцінного функціонування системи. Без нього неможливі обробка платежів, синхронізація бронювань, відправка push-сповіщень та отримання аналітичних даних. У регіонах із нестабільним покриттям мобільного зв’язку може спостерігатися затримка або втрата даних.

2) Обмежена апаратна сумісність. Система підтримує роботу лише з рекомендованими моделями камер розпізнавання номерних знаків, датчиків заповненості та шлагбаумів. Використання нестандартизованого або несумісного обладнання потребує окремої технічної адаптації або повної його заміни.

б) Фінансові та операційні обмеження:

1) Обмежений перелік платіжних провайдерів. На момент поточної реалізації система інтегрована лише з обраними платіжними сервісами (зокрема, LiqPay та PayPal). Використання системи в юрисдикціях, де зазначені провайдери обмежені або заборонені, може бути ускладненим або неможливим.

2) Необхідність періодичного оновлення. Регулярні оновлення є обов'язковими для забезпечення стабільності, безпеки та відповідності чинним стандартам. Під час оновлення окремі сервіси можуть бути тимчасово недоступними.

3) Витрати на впровадження та обслуговування. Використання системи передбачає щомісячну або річну абонентську плату, а також початкові витрати на закупівлю та встановлення сумісного обладнання. Паркінги, які не забезпечили фінансування цих витрат, не зможуть користуватися повним функціоналом системи.

4) Потреба у навчанні персоналу. Ефективне використання програмного комплексу вимагає попереднього навчання персоналу. Відсутність базової підготовки операторів та адміністрації знижує якість сервісу та ускладнює впровадження.

в) Регуляторні та правові обмеження:

1) Вимоги до обробки персональних даних. Park4Flow повинен функціонувати у повній відповідності до законодавства з обробки персональних даних, включаючи GDPR (ЄС), CCPA (США) та локальні нормативи. У деяких країнах можуть застосовуватись додаткові вимоги до зберігання, обробки та передачі персональних відомостей.

2) Необхідність ліцензування в окремих регіонах. Для легального використання системи в деяких країнах або регіонах може знадобитися отримання додаткових ліцензій або погоджень від органів місцевого самоврядування.

г) Функціональні обмеження:

1) Обмеження динамічного ціноутворення. Функція автоматичного коригування тарифів на основі попиту, часу доби чи погодних умов можлива лише за наявності попередньо налаштованих алгоритмів. Паркінги з фіксованими цінами не зможуть скористатися цією можливістю.

2) Обмежена картографічна інтеграція. Інтеграція з картографічними сервісами реалізована на базі Google Maps. Підключення до інших сервісів (наприклад, локальних або державних) можливе лише за додаткової розробки.

г) Функціональні виключення:

1) Відсутність офлайн-режиму. Park4Flow не підтримує роботу без з'єднання з Інтернетом. У разі його відсутності оплата, бронювання та отримання сповіщень будуть неможливими.

2) Відповідальність за зовнішні сервіси. Система не несе відповідальності за непрацездатність платіжних або картографічних сервісів, API яких вона використовує.

3) Несертифіковане обладнання. Сумісність із обладнанням, яке не входить до офіційного списку підтримуваних пристроїв, не гарантується.

4) Обмеження платіжних методів. Park4Flow не підтримує альтернативні форми оплати, такі як криптовалюти, чеки, бартер або внутрішньокорпоративні токени.

5) Відсутність доступу без системних дозволів. Функціональність мобільного застосунку обмежена у разі відсутності дозволів на доступ до геолокації, камери, файлової системи чи push-сповіщень.

д) Виключення щодо безпеки:

1) Обмежена кібербезпека. Хоча система використовує сучасні протоколи шифрування (TLS 1.3, AES-256), вона не може повністю

захистити від усіх кіберзагроз, включаючи цільові DDoS-атаки або витоки даних через вразливості сторонніх пристроїв.

2) Помилки користувачів. Park4Flow не несе відповідальності за наслідки помилок користувачів, наприклад, у разі неправильного введення номерного знака при бронюванні.

е) Регуляторні виключення:

1) Невідповідність новим нормативним вимогам. У разі змін у законодавстві, адаптація системи потребуватиме часу та додаткових фінансових ресурсів. Постачальник не несе відповідальності за втрати, пов’язані з часовими затримками у відповідності.

є) Виключення щодо доступності

1) Планові технічні перерви. Під час планового обслуговування або оновлення можливі тимчасові перебої в роботі системи. Про такі перерви повідомляється заздалегідь.

2) Форс-мажорні обставини. Система не може гарантувати стабільну роботу в умовах надзвичайних подій, таких як природні катастрофи, масштабні кібератаки, відключення електропостачання або банкрутство паркінг-оператора.

### 2.2.4 Припущення та залежності

Для успішної розробки, впровадження та експлуатації програмної системи Park4Flow приймаються наступні ключові припущення:

– Доступ до стабільного інтернет-з’єднання. Передбачається, що всі паркінги, які інтегрують систему, мають постійне, стабільне та високошвидкісне підключення до Інтернету, що необхідне для безперебійної роботи функцій обробки оплат, синхронізації даних у реальному часі та взаємодії з хмарними сервісами.

– Забезпеченість відповідним технічним обладнанням. Оператори паркінгів повинні мати у своєму розпорядженні сучасні технічні засоби — комп’ютери, POS-термінали, планшети або мобільні пристрої — з відповідними характеристиками для повноцінної роботи з системою.

– Наявність базових навичок у користувачів. Очікується, що після проходження передбачених програмою навчань (у форматі відеоінструкцій, документації та інтерактивних тренінгів), персонал паркінгів і водії володітимуть необхідними навичками для ефективного використання системи.

– Активна участь адміністрацій паркінгів. Впровадження передбачає залучення адміністрацій об’єктів, які повинні забезпечити доступ до навчальних матеріалів, організацію навчання персоналу та надання зворотного зв’язку з метою вдосконалення роботи системи.

– Відповідність чинному законодавству. Система буде розроблена та адаптована з урахуванням усіх актуальних нормативно-правових вимог, зокрема у сфері обробки персональних даних, захисту інформації та електронних фінансових операцій.

– Наявність необхідних ліцензій і дозволів. До моменту запуску проекту передбачається отримання всіх необхідних ліцензій та дозволів на використання Park4Flow в юридичному полі конкретних країн або регіонів.

– Забезпечене фінансування проекту. Проект отримає повне фінансування на всіх етапах життєвого циклу системи — від розробки до технічної підтримки, включно з тестуванням, впровадженням та експлуатацією.

– Урахування супутніх витрат. До загального бюджету проекту буде включено витрати на навчання персоналу, технічну підтримку користувачів, оновлення програмного забезпечення та супутню документацію.

– Зацікавленість кінцевих користувачів. Очікується, що оператори паркінгів та водії будуть зацікавлені у використанні системи завдяки її інтуїтивному інтерфейсу, функціональності та можливості зменшити операційні витрати.

– Співпраця з командою розробників. Адміністрація паркінгів буде відкритою до співпраці з командою розробки задля оперативного вирішення технічних питань, внесення змін і покращень функціональності системи.

Ефективність функціонування системи Park4Flow залежить від ряду зовнішніх та внутрішніх факторів, які можуть безпосередньо впливати на її стабільність, масштабованість та адаптивність:

– Інтеграція із зовнішніми сервісами. Функціонування системи передбачає безперебійну інтеграцію з платіжними шлюзами, картографічними API (наприклад, Google Maps), службами сповіщення та аналітичними інструментами.

– Надійність зовнішніх технологічних рішень. Система критично залежить від стабільної роботи хмарних платформ, банківських процесингових центрів та API, що використовуються для виконання ключових операцій.

– Регулярне оновлення програмного забезпечення. Забезпечення актуальності, безпеки та відповідності функціоналу потребам ринку вимагає постійного вдосконалення системи через оновлення.

– Технічна підтримка користувачів. Наявність кваліфікованої підтримки, включаючи чат-боти, базу знань і підтримку в режимі реального часу, є важливим чинником стабільного користування системою.

– Відповідність законодавчим вимогам. Park4Flow повинен постійно відповідати чинним та майбутнім регуляторним актам у сфері обробки даних, кібербезпеки та електронної комерції.

– Адаптивність до змін. Система повинна мати гнучку архітектуру, яка дозволяє оперативно адаптуватися до змін у законодавстві, безпекових стандартах або потребах ринку.

– Використання зворотного зв’язку. Збір та аналіз відгуків користувачів дозволить цілеспрямовано вдосконалювати інтерфейс, логіку роботи та функціональність системи.

– Продуктивність і масштабованість. Система повинна витримувати збільшення кількості користувачів без погіршення продуктивності або якості обслуговування.

– Маркетингова підтримка. Ефективне просування системи на ринку передбачає реалізацію комплексних маркетингових заходів для залучення нових користувачів.

– Професійна команда розробки. Для забезпечення життєздатності продукту важлива наявність стабільної команди спеціалістів, здатної оперативно вирішувати технічні завдання, розвивати функціонал та підтримувати відповідність системи вимогам часу.

## 2.3 Робоче середовище

Програмна система Park4Flow повинна стабільно функціонувати в різноманітних умовах експлуатації, забезпечуючи надійну роботу для водіїв, операторів паркінгів та адміністративного персоналу. Нижче подано основні вимоги до середовища, у якому розгортатиметься та використовуватиметься система.

Операційні системи:

– Серверна частина має працювати на Windows 10 Pro. Клієнтська частина сумісна з Windows 10 і новішими версіями, macOS 11 і вище, а також сучасними дистрибутивами Linux. Мобільний додаток розроблено для пристроїв з Android 9 і новіше.

Підтримувані браузери:

– Система коректно працює у браузерах Google Chrome версії 95 і вище, Mozilla Firefox 90+, Microsoft Edge 95+ та Safari 14+.

Мобільні пристрої:

– Підтримуються пристрої з процесорами ARM і x86, мінімальним обсягом оперативної пам’яті 2 ГБ та роздільною здатністю екрана не менше 720x1280 пікселів (HD).

Інтернет-з’єднання:

– Для серверної інфраструктури рекомендоване підключення через Wi-Fi 5 ГГц або Ethernet. Мінімальна швидкість інтернету для користувачів — 50 Мбіт/с,

для серверів — 100 Мбіт/с. Мобільний застосунок має працювати через LTE або 4G-зв’язок.

Серверна інфраструктура:

– Вимоги до апаратного забезпечення включають 8-ядерний процесор AMD Ryzen 5600h, щонайменше 32 ГБ оперативної пам’яті та NVMe SSD накопичувач обсягом 1 ТБ. В якості СУБД використовується PostgreSQL версії 14 або новішої з підтримкою jsonb-типів.

Технології серверної частини:

– Back-end реалізований на JavaScript з використанням Node.js, Express і архітектури MVC. Для інтерактивної мапи застосовується WebSocket.

Технології клієнтської частини:

– Front-end розроблено на React з використанням Tailwind CSS і бібліотеки Axios для HTTP-запитів. Мобільний додаток створений на Kotlin з підтримкою WebSocket та Retrofit.

Інтеграція з IoT-пристроями:

– Система підтримує протоколи WebSockets та HTTP для обміну даними з пристроями, серед яких автоматичні шлагбауми, сенсори зайнятості паркомісць, контролери на базі ESP32. Прошивка таких пристроїв розробляється з використанням Arduino Sketch (C++).

Безпека та резервування:

– Передача даних здійснюється через захищені протоколи SSL та TLS 1.3. Збережені дані шифруються за стандартом AES-256. Паролі зберігаються у вигляді хешів. Система підтримує автоматичне щоденне резервне копіювання бази даних і конфігурацій.

Платіжні інтеграції:

– Передбачена підтримка онлайн-оплати через LiqPay і PayPal, а також безконтактних платежів через Apple Pay та Google Pay.

Система сповіщень:

– Для мобільних пристроїв реалізовано push-сповіщення на Android. Також система підтримує email-розсилки через SMTP або SendGrid.

Середовище розробки:

– Для розробки серверної та клієнтської частин використовується WebStorm Enterprise. Розробка мобільного додатку ведеться в Android Studio. Для IoT-пристроїв використовується емулятор Wokwi.

## 2.4 Конфіденційність даних та безпека

Програмна система Park4Flow обробляє конфіденційну інформацію користувачів, зокрема особисті дані, платіжну інформацію та історію паркувань. Захист цих даних є ключовим фактором довіри користувачів і вимогою для дотримання законодавчих норм. У цьому розділі описано категорії даних, що обробляються, заходи безпеки, політику зберігання, а також відповідність нормативним стандартам.

### 2.4.1 Категорії даних, що обробляються

Система Park4Flow зберігає та обробляє такі типи даних:

– Персональні дані користувачів, включаючи ім’я, прізвище, контактну інформацію (номер телефону, email), а також дані транспортного засобу (державний номер, марка, модель).

– Фінансова інформація, зокрема дані банківських карток (обробляються лише через сертифіковані платіжні шлюзи), баланс у системі та історія платежів.

– Дані паркувань і геолокації, включаючи координати автомобіля, час початку та завершення паркування, історію бронювань.

– Технічні дані, такі як IP-адреси користувачів, журнали активності та дані аутентифікації.

### 2.4.2 Політика зберігання та обробки даних

Для забезпечення безпеки та дотримання регуляторних вимог система застосовує такі практики:

– Зберігання персональних даних обмежується лише періодом, необхідним для надання послуг.

– Конфіденційна інформація шифрується з використанням алгоритму AES-256 (для даних у стані спокою) і TLS 1.3 (для даних у транзиті).

– У випадках, коли ідентифікація користувача не є критичною, дані підлягають анонімізації.

– Щоденне резервне копіювання здійснюється у захищеному середовищі з обмеженим доступом.

– Користувач має право на видалення своїх персональних даних відповідно до політики конфіденційності.

### 2.4.3 Відповідність нормативним стандартам

Park4Flow відповідає міжнародним і національним вимогам щодо захисту персональних даних, зокрема:

– Регламенту GDPR, що передбачає прозорість використання даних, право користувачів на редагування або видалення інформації, а також обов’язкове отримання згоди на обробку персональних даних.

– Міжнародному стандарту ISO/IEC 27001, який регулює управління інформаційною безпекою та контроль доступу до конфіденційних даних.

– Локальному законодавству, що регламентує зберігання та обробку персональної інформації відповідно до вимог національних органів контролю.

### 2.4.4 Заходи безпеки

Для запобігання витоку або несанкціонованому доступу до даних в системі реалізовано такі механізми:

– Двофакторна аутентифікація (2FA) для доступу до особистого кабінету користувача.

– Система ролей і прав доступу, що передбачає розмежування повноважень між звичайними користувачами, операторами паркінгу та адміністраторами.

– Безпечна обробка платежів через сертифіковані платіжні шлюзи без зберігання даних банківських карток на сервері.

– Механізми реагування на інциденти, включаючи автоматичне повідомлення про підозрілі спроби входу в акаунт.

### 2.4.5 Політика конфіденційності

Для забезпечення прозорого використання даних система надає:

– Детальну політику конфіденційності та договір користувача.

– Можливість самостійного керування персональними даними, зокрема їх редагування та видалення.

– Повідомлення про зміни у політиці конфіденційності або умовах користування.

# 3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 UML-проектування

### 3.1.1 Загальне проектування системи

На основі наданої UML-діаграми розгортання можна описати загальну структуру та архітектуру програмної системи для паркування Park4Flow. Діаграма наведена на рисунку 3.1.

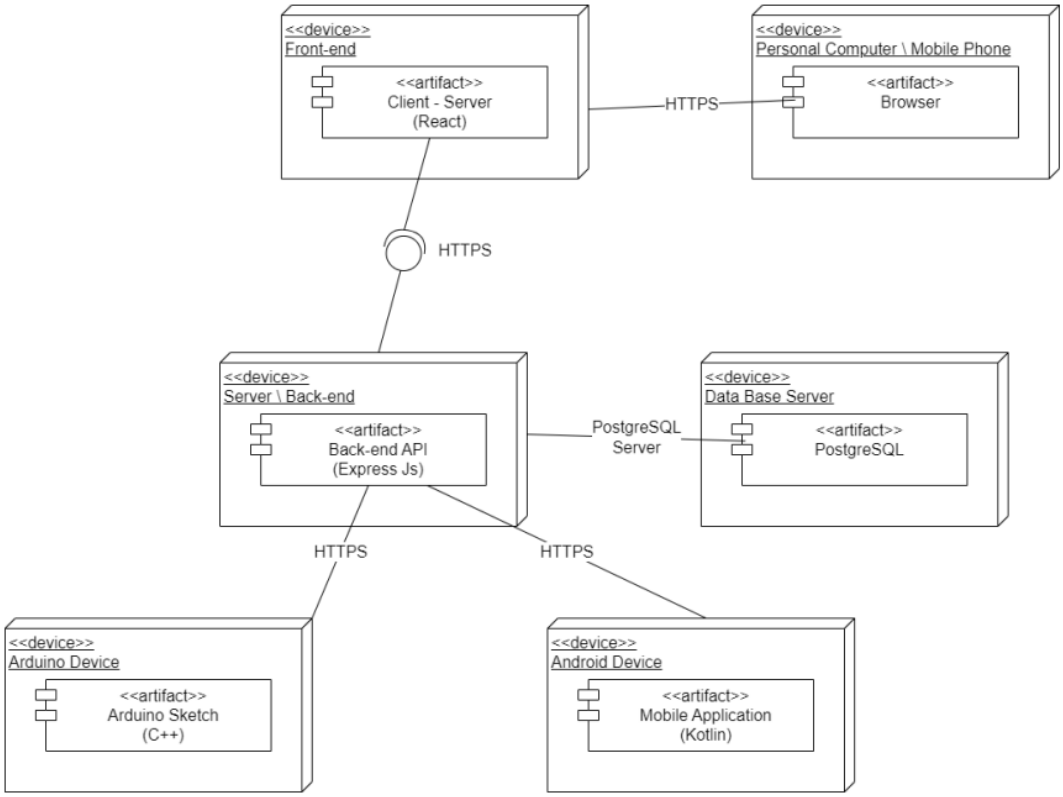


Рисунок 3.1– UML - діаграма розгортання (рисунок виконаний самостійно)

Дана система реалізована за клієнт-серверною моделлю та передбачає взаємодію між декількома апаратними пристроями (девайсами) та програмними артефактами (компонентами), які розгортаються на них.

У центрі архітектури знаходиться серверна частина (Back-end), представлена пристроєм Server \ Back-end, на якому працює Back-end API, розроблений з використанням Express.js. Цей сервер відповідає за обробку запитів від клієнтських додатків, взаємодію з базою даних, реалізацію логіки автентифікації, обробку бронювань, оплати та інші ключові функції.

З сервером взаємодіє клієнтська частина (Front-end), яка розгорнута на пристрої Front-end у вигляді клієнт-серверного додатку, створеного на React. Він забезпечує інтерфейс взаємодії з користувачем, надсилає запити до серверу через протокол HTTPS та отримує у відповідь оброблені дані.

Користувачі можуть працювати з системою через персональні комп'ютери або мобільні пристрої, які позначені як Personal Computer \ Mobile Phone. Вони взаємодіють з клієнтською частиною системи через веб-браузер.

Окрему роль відіграє мобільний додаток, встановлений на Android Device. Цей додаток, розроблений мовою Kotlin, також обмінюється даними з сервером через HTTPS. Його призначенням є забезпечення доступу користувачів до функціоналу системи у зручному мобільному форматі.

База даних розгорнута на Data Base Server і представлена PostgreSQL Server. Серверна частина взаємодіє з нею безпосередньо для зчитування та зберігання всієї інформації про користувачів, транспортні засоби, бронювання, транзакції тощо. Взаємодія між бекендом і базою даних також відбувається по захищеному протоколу.

Окремий пристрій Arduino Device представляє вбудовану систему, яка виконує скетчі (Sketches) на мові C++. Цей пристрій може використовуватись для моніторингу паркомісць, наприклад, за допомогою датчиків зайнятості. Він також підключений до серверної частини через HTTPS, забезпечуючи обмін технічною інформацією в реальному часі.

Таким чином, архітектура системи є розподіленою та масштабованою, де кожен компонент виконує свою роль і взаємодіє із серверною частиною через безпечні канали зв'язку (HTTPS або спеціалізовані протоколи для БД). Це забезпечує як надійність системи, так і її гнучкість для подальшого розвитку.

### 3.1.2 Проектування серверної частини

Серверна частина системи Park4Flow, реалізована за допомогою Node.js з використанням фреймворку Express, відповідає за обробку запитів, бізнес-логіку та взаємодію з базою даних. На основі наведеної UML-діаграми прецедентів для

серверної частини можна зробити детальний огляд ролей користувачів та функціоналу, який їм доступний. Сама діаграма наведена на рисунку 3.2.

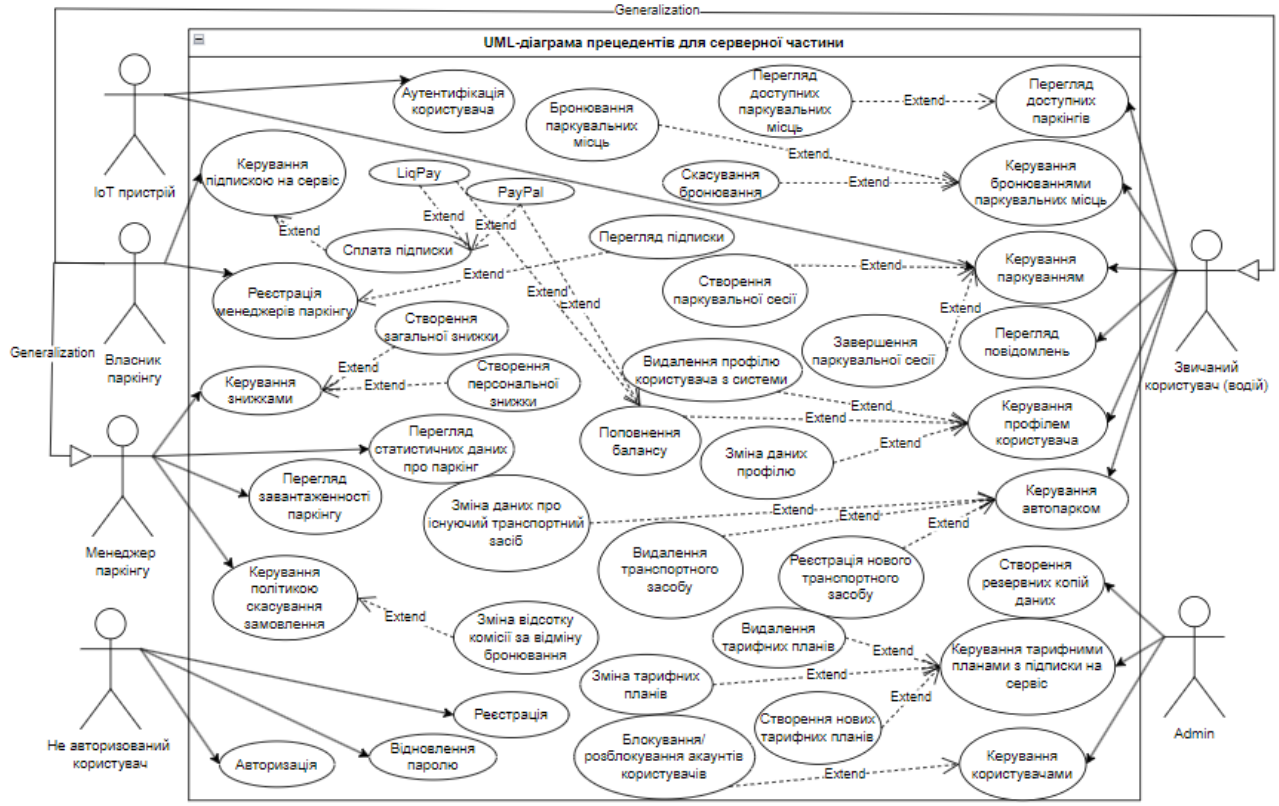


Рисунок 3.2 – UML - діаграма прецедентів для серверної частини (рисунок виконаний самостійно)

На основі цієї діаграма можна визначитись з функціональними можливостіми для користувачів:

а) Неавторизований користувач

Користувачі, які ще не зареєстровані або не увійшли до системи, мають базові можливості. Вони можуть здійснити авторизацію, реєстрацію нового облікового запису, а також відновлення паролю у разі його втрати. Цей функціонал є базовим для доступу до розширених можливостей системи та відкриває шлях до персоналізованої взаємодії з сервісом.

б) Звичайний користувач (водій)

Після авторизації звичайний користувач отримує доступ до повного спектру функцій, пов’язаних із користуванням паркувальними послугами. Зокрема,

користувач може здійснювати бронювання паркувальних місць, переглядати доступні паркувальні місця, а також за потреби скасовувати бронювання. Для зручності також реалізований перегляд історії бронювань.

Крім цього, користувач може створювати паркувальні сесії — тобто почати реальне використання забронюваного місця, а потім завершувати ці сесії. Після завершення паркування або оформлення бронювання користувач має змогу сплатити за послугу через платіжні сервіси PayPal та LiqPay. Крім того він має змогу переглядати повідомлення, що можуть включати важливі оновлення, нагадування чи інформацію від адміністраторів.

Звичайний користувач також керує своїм профілем: він може змінювати персональні дані, реєструвати новий транспортний засіб або видаляти існуючий, а також керувати автопарком — списком власних транспортних засобів. Для користувачів, які користуються розширеними функціями, є можливість керування підпискою на сервіс, що передбачає сплату підписки та перегляд поточної інформації про неї.

в) Власник паркінгу

Власник паркінгу має ширші повноваження, спрямовані на керування бізнес-процесами. Він може реєструвати менеджерів паркінгу, які згодом отримають обмежені права на управління паркінгом. Також доступна функція керування знижками — як загальними, так і персональними, що можуть бути надані окремим користувачам.

У межах взаємодії з системою власник також має змогу переглядати статистичні дані про паркінг — такі як завантаженість, прибутки, частота бронювань тощо, а також поповнювати баланс для оплати послуг системи.

г) Менеджер паркінгу

Менеджер паркінгу, як довірена особа власника, володіє інструментами для оперативного управління. Він може переглядати завантаженість паркінгу, змінювати дані про наявні транспортні засоби, а також керувати політикою скасування бронювань, що дозволяє йому визначати відсоток комісії при відмові

від замовлення. Також менеджер має доступ до загальної аналітики, яка дає змогу відслідковувати ефективність паркінгу.

г) Адміністратор (Admin)

Адміністратор є найвищим рівнем користувача в системі. Йому доступні всі функції, пов’язані з керуванням тарифами та користувачами. Зокрема, адміністратор може створювати, редагувати та видаляти тарифні плани, а також керувати тарифними планами з підписки на сервіс. Він має можливість створювати резервні копії бази даних, що критично важливо для збереження інформації в разі збоїв.

Також адміністратор відповідає за блокування та розблокування акаунтів, керування усіма користувачами, включно з водіями, власниками паркінгів і менеджерами. Крім того, йому доступна функція видалення профілю користувача з системи за потреби.

д) IoT-пристрій

Окрему роль у системі відіграє IoT-пристрій. Він взаємодіє з сервером для передачі технічних даних — наприклад, інформації про зайнятість паркомісць. Його взаємодія обмежена автоматизованими сценаріями (наприклад, створенням або завершенням паркувальної сесії), що сприяє безперебійному моніторингу та точному відображенню статусу місць у системі.

Таким чином, серверна частина системи реалізує чітко розмежовану рольову модель доступу, що забезпечує як гнучкість у використанні, так і безпеку. Усі запити передаються через REST API та обробляються у відповідних роутерах Express.js з урахуванням авторизації, ролей і прав доступу.

### 3.1.3 Проектування клієнтської частини

UML-діаграма прецедентів (див. Рисунок 3.3) для клієнтської частини програмної системи відображає основні сценарії взаємодії користувачів з веб-інтерфейсом сайту, реалізованого за допомогою React і Axios.

Ця діаграма ілюструє функціональні можливості, доступні користувачам різних ролей, включаючи неавторизованих відвідувачів, водіїв, власників

паркінгів, менеджерів паркінгу та адміністратора системи. Кожен актор на діаграмі відповідає за певну категорію дій, які відображаються у вигляді прецедентів – тобто дій або сценаріїв, які вони можуть ініціювати через вебінтерфейс. Це дозволяє чітко структуровано представити інтерфейсну логіку взаємодії користувача з системою через браузер.

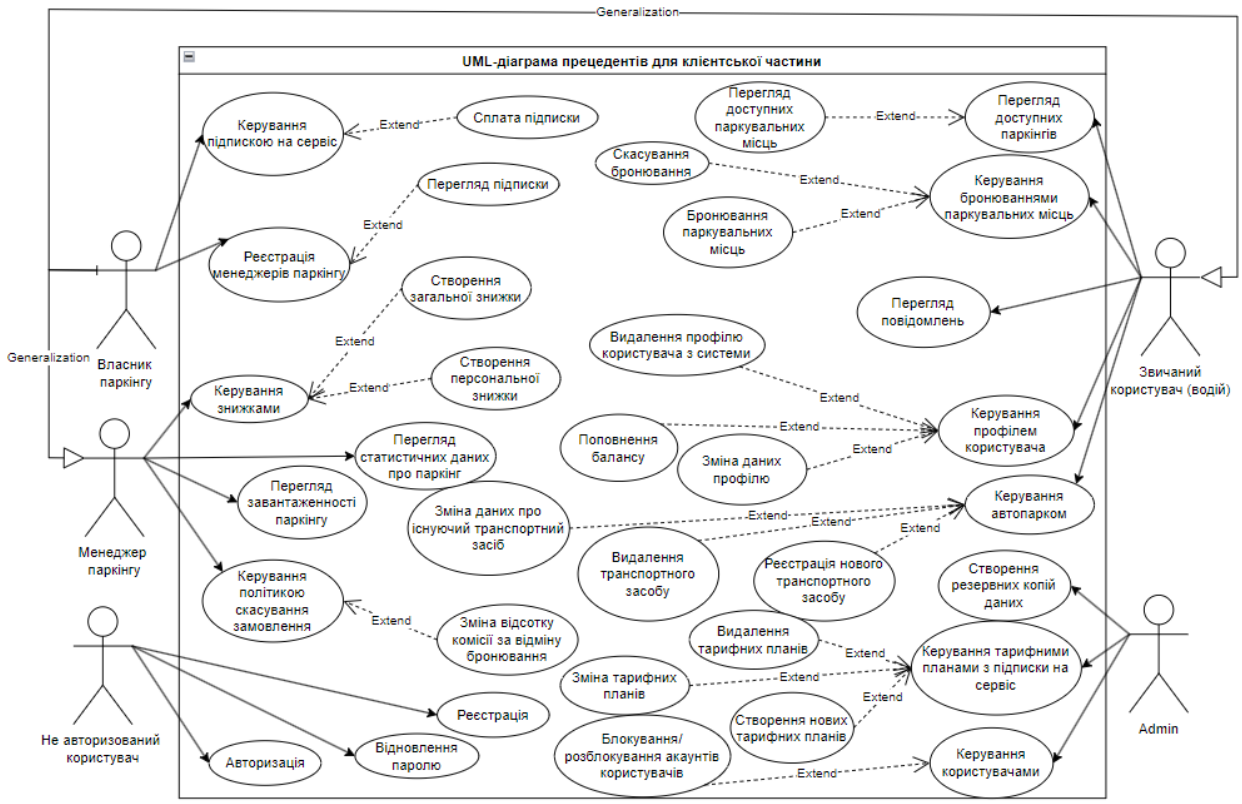


Рисунок 3.3 – UML - діаграма прецедентів для клієнтської частини (рисунок виконаний самостійно)

Веб клієнт має наступні функціональні можливості для користувачів:

а) Неавторизований користувач (гостьовий режим)

Через вебінтерфейс користувач, який ще не увійшов у свій обліковий запис, бачить базову навігацію. Він має можливість перейти на сторінку входу, створити новий акаунт або скористатися функцією відновлення пароля. Ці функції доступні одразу на сайті без потреби авторизації.

б) Звичайний користувач (водій)

Після входу в систему через вебклієнт водій отримує доступ до головної сторінки з картою паркувальних місць. Він може:

1) переглядати доступні місця на мапі;

2) бронювати паркомісця;

3) переглядати й скасовувати власні бронювання;

4) переглядати повідомлення (попередження, нагадування);

5) керувати своїм профілем: редагувати персональні дані;

6) додавати/видаляти власні транспортні засоби;

7) переглядати список свого автопарку;

8) переглядати/оплачувати підписку, якщо така активна.

в) Власник паркінгу

У своєму веб - кабінеті власник паркінгу має доступ до панелі керування, де може:

1) додавати нових менеджерів;

2) створювати/редагувати знижки (загальні й індивідуальні);

3) переглядати статистику роботи паркінгу: дохід, завантаженість, бронювання;

4) переглядати і поповнювати баланс для оплати сервісу.

г) Менеджер паркінгу

Через веб-панель менеджера доступні інструменти для щоденного управління:

1) перегляд поточної завантаженості паркінгу;

2) налаштування політики скасування бронювань;

3) перегляд загальної аналітики щодо ефективності роботи.

г) Адміністратор системи

У адміністратора в інтерфейсі доступна окрема адмін-панель, через яку він може:

1) створювати, змінювати та видаляти тарифні плани;

2) керувати підписками;

3) створювати резервні копії бази;

4) блокувати/розблоковувати акаунти;

Таким чином, функціонал клієнтської частини залежить від ролі користувача, яку клієнт завантажує одразу під час авторизації для визначення, який саме функціонал потрібний користувачу.

### 3.1.4 Проектування клієнтської частини для мобільних платформ

Клієнтська частина програмної системи для мобільних пристроїв реалізується у вигляді зручного додатка, що забезпечує доступ до основного функціоналу сервісу з будь-якого сучасного смартфона. Інтерфейс мобільного застосунку розроблено з урахуванням потреб кінцевих користувачів (водіїв), які користуються послугами паркування. Взаємодія користувача із системою моделюється за допомогою UML-діаграми прецедентів (див. Рисунок 3.4), що демонструє доступні сценарії використання для різних категорій користувачів, таких як неавторизований користувач та авторизований користувач.

Ця діаграма структуровано представляє типові сценарії взаємодії, що підтримуються мобільним додатком, зокрема — реєстрацію, перегляд паркувальних місць, бронювання, керування транспортними засобами, паркувальними сесіями, профілем та балансом. Вона відображає основні функціональні можливості, які відкриваються користувачу в залежності від його статусу (авторизований або неавторизований).

Дії, які доступні всім користувачам одразу після встановлення мобільного застосунку:

- Реєстрація нового облікового запису.
- Авторизація для входу в систему.
- Відновлення паролю у разі його втрати.

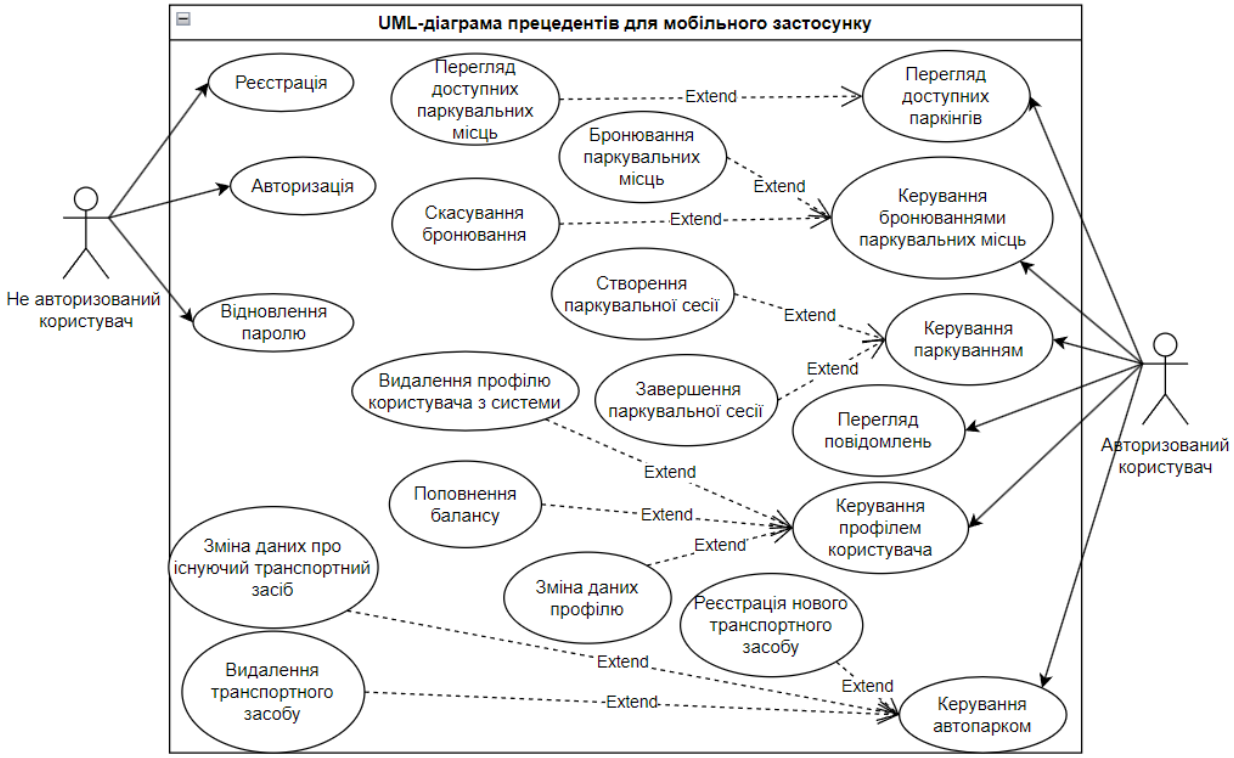


Рисунок 3.4 – UML - діграма прецедентів для мобільного застосунку (рисунок виконаний самостійно)

Після входу до системи відкривається доступ до повного спектру функцій, зокрема:

– Перегляд доступних паркінгів та паркувальних місць на мапі.

– Бронювання паркувальних місць та скасування бронювань.

– Створення та завершення паркувальної сесії, в залежності від статусу бронювання або факту прибуття на паркінг.

– Керування паркуванням: контроль поточних сесій, перевірка активних бронювань, завершення паркування.

– Перегляд повідомлень, які включають сповіщення про завершення часу паркування, нагадування про борги або інші події.

– Керування профілем користувача, зокрема редагування особистих даних та видалення облікового запису.

– Поповнення балансу для здійснення платежів за паркування або бронювання.

– Керування автопарком:
  – реєстрація нового транспортного засобу,
  – зміна даних існуючого ТЗ,
  – видалення транспортного засобу.

### 3.1.5 Проектування IoT терміналу

Для автоматизації контролю доступу до паркувального місця передбачено розгортання спеціалізованого IoT-терміналу, що встановлюється на в’їзді до кожного паркомісця. Такий термінал дозволяє реалізувати безконтактну перевірку права доступу користувача до забронюваного місця, а також забезпечує фізичне управління допуском за допомогою сервомеханізму.

Функціональність IoT пристрою моделюється за допомогою UML-діаграми прецедентів (див. рисунок 3.5) прецедентів, яка демонструє послідовність дій під час ідентифікації користувача та надання/відмови в доступі до паркінгу.

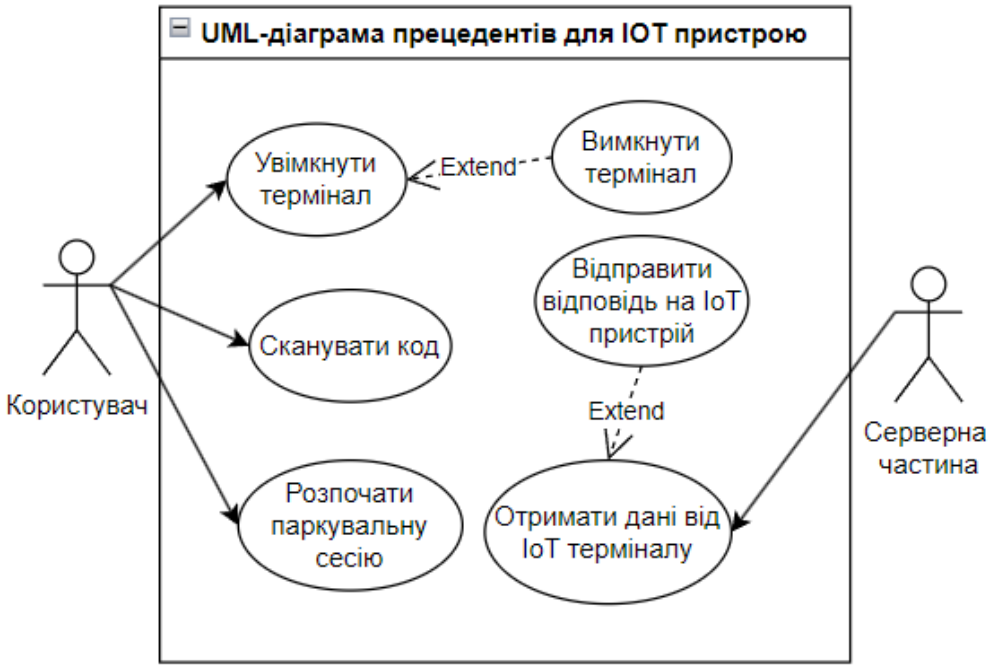


Рисунок 3.5 – UML - діаграма прецедентів для IoT терміналу (рисунок виконаний самостійно)

IoT термінал має наступну функціональність:

а) Сканування ідентифікатора користувача

1) Користувач наближається до IoT-терміналу та сканує QR-код з мобільного додатку або вводить код вручну (6 – значний код для ідентифікації користувача).

2) Термінал ініціює запит до серверної частини системи, передаючи отриманий ідентифікатор.

б) Обробка запиту на сервері

Сервер виконує перевірку:

– Чи існує активне бронювання за ідентифікатором.
– Чи відповідає час початку сесії поточному часу.
– Чи не минув термін бронювання.
– Чи не зайняте місце іншим користувачем.

в) Отримання відповіді від сервера

1) У разі позитивної перевірки сервер підтверджує початок паркувальної сесії, після чого термінал активує сервомеханізм, який відкриває фізичний доступ до місця (наприклад, підняття шлагбаума або розблокування в'їзду).

2) У випадку відмови (бронювання недійсне, сесія завершена або місце зайняте), термінал відображає повідомлення про помилку на вбудованому екрані або через інші інтерфейси (LED, аудіосигнал).

г) Фіксація події в системі

1) Всі події фіксуються у логах сервера із зазначенням часу, ID користувача та статусу спроби входу.

Взаємодія між користувачем, IoT-пристроєм, серверною частиною і базою даних системи моделюється за допомогою UML – діаграми взаємодії, яка наведена на рисунку 3.6.

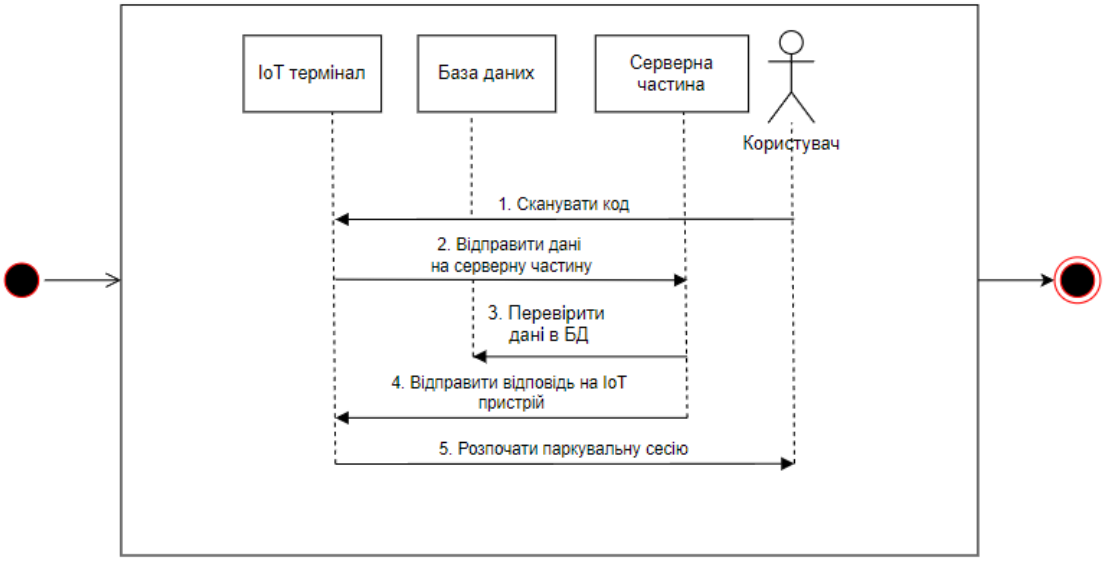


Рисунок 3.6 – UML - діграма взаємодії (рисунок виконаний самостійно)

Дана діаграма відображає процес взаємодії користувача з системою для запуску паркувальної сесії. Користувач сканує код (QR-код або штрих - код), після чого дані передаються на серверну частину. Сервер перевіряє їх у базі даних (підтверджує доступність місця та автентифікацію користувача) і надсилає відповідь на IoT-пристрій. На основі отриманих інструкцій IoT-пристрій ініціює паркувальну сесію — відкриває доступ до місця та фіксує час початку паркування.

Завдяки такій архітектурі, IoT-термінал виконує роль інтелектуального шлюзу між фізичним доступом до паркінгу та цифровими правами, визначеними сервером. Це дозволяє виключити людський фактор та зменшити час на перевірку доступу, що підвищує ефективність використання паркомісць.

## 3.2 Проектування структури зберігання даних

### 3.2.1 Вибір СУБД

Для забезпечення надійного, масштабованого та функціонального зберігання даних у програмній системі Park4Flow було обрано реляційну систему керування базами даних PostgreSQL версії 14 і вище. Ця СУБД є перевіреним рішенням з відкритим кодом, що широко застосовується в критично важливих комерційних і

державних проєктах, завдяки високій продуктивності, розвиненим механізмам безпеки, підтримці транзакцій та широким можливостям розширення.

PostgreSQL надає розширені засоби роботи з даними, зокрема підтримку як класичної реляційної, так і об'єктно-реляційної моделі. Однією з ключових особливостей є вбудована підтримка типу даних jsonb, що дозволяє ефективно працювати з напівструктурованими даними, зберігати конфігурації, лог-файли, історичні записи та інші динамічні структури. Доступні засоби індексації й пошуку по вкладених полях дозволяють використовувати PostgreSQL як гібрид між реляційною та NoSQL базами.

СУБД повністю відповідає вимогам ACID, що критично важливо для транзакційних операцій — обробки платежів, бронювань та реєстрації користувачів. Гарантії атомарності, узгодженості, ізольованості та довговічності даних знижують ризики дублювання або втрати записів у разі збоїв.

З точки зору безпеки PostgreSQL [6] надає гнучкі засоби контролю доступу — ролі, політики доступу на рівні бази, таблиць, представлень і навіть окремих рядків. Також реалізовано сучасні механізми аутентифікації (md5, scram-sha-256, GSSAPI), шифрування трафіку (SSL/TLS) і обмеження підключень. Це дозволяє ефективно розмежовувати доступ між адміністраторами, операторами паркінгу та звичайними користувачами.

PostgreSQL підтримує масштабовану архітектуру завдяки великій кількості розширень. Зокрема, використання модулю PostGIS дозволяє працювати з просторовими даними — координатами автомобілів і паркомісць. Розширення pg_cron забезпечує планування завдань, а pg_stat_statements — моніторинг і оптимізацію запитів. Це відкриває можливості для гнучкої адаптації системи до зростаючих навантажень без зміни базової інфраструктури.

Для забезпечення надійного зберігання та швидкого відновлення інформації PostgreSQL підтримує повне й інкрементальне резервне копіювання, журналювання транзакцій (WAL) та можливість відновлення до будь-якої контрольної точки в часі. Це відповідає політиці щоденного резервування даних у системі Park4Flow.

Крім того, PostgreSQL [6] ефективно працює з великими обсягами даних завдяки підтримці паралельного виконання запитів, різних типів індексації (B-tree, GIN, GiST, BRIN), партиціонування таблиць, матеріалізованих представлень і розвиненого планувальника запитів із повною статистикою.

СУБД має активну спільноту, що забезпечує регулярні оновлення, виправлення безпеки та стабільну довгострокову підтримку. Це дає можливість використовувати PostgreSQL як стабільну платформу в довготривалих проєктах без ризиків сумісності або припинення підтримки.

### 3.2.2 Проектування структури зберігання даних

У процесі розробки інформаційної системи для інтелектуального паркування «Park4Flow» важливим етапом є проєктування логічної структури бази даних. Для цього було побудовано ER - модель (модель «сутність–зв’язок»), яка відображає основні об’єкти предметної області, їхні атрибути та взаємозв’язки між ними. Дана модель забезпечує цілісність даних, спрощує реалізацію логіки взаємодії з базою даних та є фундаментом для фізичної реалізації в обраній СУБД — PostgreSQL. ER - модель даних наведена на рисунку 3.7.

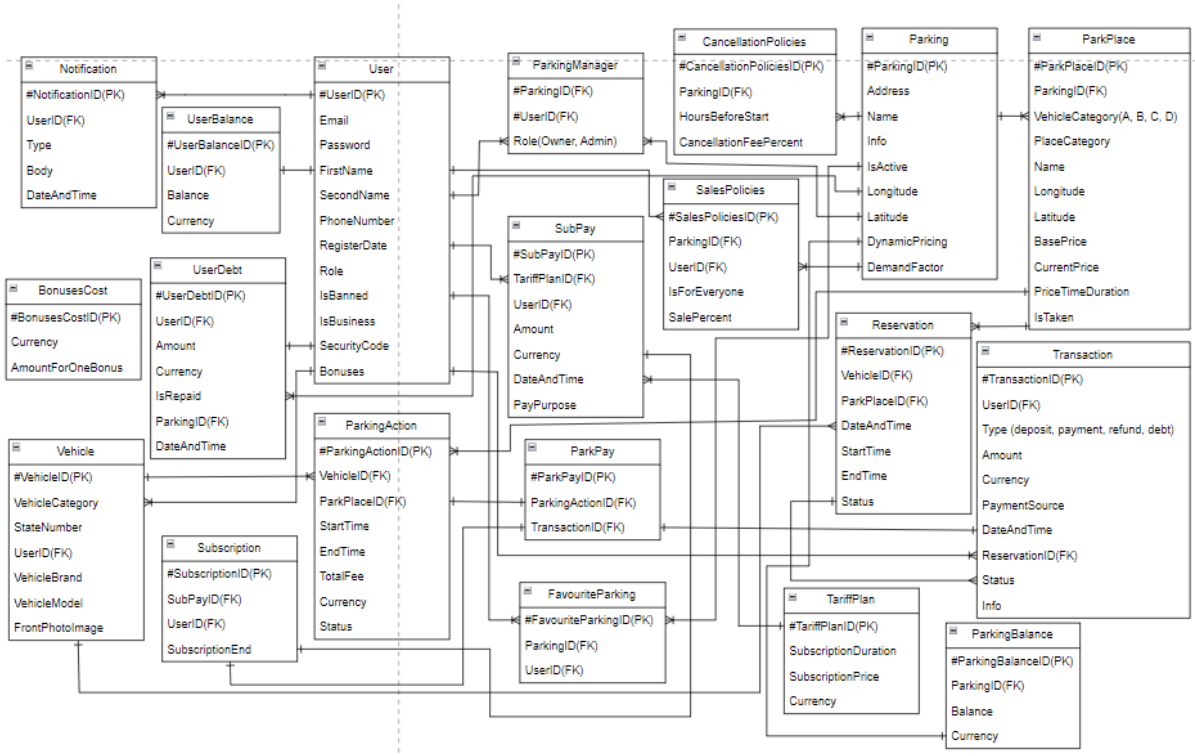


Рисунок 3.7 – ER - модель даних (рисунок виконаний самостійно)

Ця ER-модель дозволяє охопити всі ключові аспекти роботи системи Park4Flow: управління користувачами та їхніми авто, процеси паркування, бронювання, оплати, підписок, політик скасування та знижок. Це забезпечує повноцінну підтримку функціоналу смарт-системи паркування як для користувачів, так і для адміністраторів.

Нижче описано сутності, з яких складається база даних.

User є центральною у системі та представляє зареєстрованого користувача. Вона містить основну інформацію: електронну пошту, пароль, ім’я, прізвище, номер телефону, дату реєстрації, роль (користувач, власник, адміністратор), а також статуси (чи заблокований, чи є бізнес-користувач). Додатково зберігається поле для бонусів та коду безпеки.

UserBalance зберігає інформацію про фінансовий баланс користувача — доступну суму коштів та валюту. Ця таблиця дозволяє швидко здійснювати перевірку платоспроможності користувача при проведенні транзакцій.

UserDebt відображає заборгованості користувача за паркування. Вона містить суму боргу, валюту, інформацію про те, чи була заборгованість ліквідована, дату її виникнення та зв’язок із конкретним паркінгом.

Notification — таблиця, яка зберігає історію сповіщень, надісланих користувачеві. Кожне повідомлення має тип, текстове наповнення та дату й час надсилання.

Vehicle описує транспортні засоби, зареєстровані користувачами. Зберігається категорія, державний номер, марка, модель та фото автомобіля. Один користувач може мати кілька автомобілів.

ParkingManager відображає призначення користувачів на ролі адміністраторів або власників паркінгів. Це дозволяє розмежовувати права доступу до управління конкретними об’єктами паркування.

Parking описує паркінги в системі: їхню адресу, назву, статус активності, географічні координати, інформацію про динамічне ціноутворення та коефіцієнт попиту. Кожен паркінг може мати власні політики скасування та знижок.

ParkPlace представляє окремі місця паркування в межах конкретного паркінгу. Зберігається їхня категорія (A, B, C, D), координати, базова та поточна ціна, тривалість дії ціни, а також статус зайнятості.

Reservation фіксує факти бронювання паркомісць. Містить дані про транспортний засіб, обране місце, час початку та завершення бронювання, статус та дату створення.

Transaction містить інформацію про всі фінансові операції користувача, включаючи депозити, платежі, повернення коштів та борги. Зберігається тип операції, сума, джерело оплати, дата, статус, а також зв’язок із бронюванням.

FavouriteParking дозволяє користувачам зберігати улюблені паркінги для швидкого доступу. Це підвищує зручність використання системи.

ParkingAction описує реальний факт перебування транспортного засобу на паркомісці: час початку та завершення, загальну тривалість, валюту розрахунку та статус дії.

ParkPay зберігає інформацію про оплату кожної сесії паркування. Вказується сума, валюта, а також зв’язок із транзакцією, що підтверджує оплату, та дією паркування.

Subscription фіксує факт підписки користувача на тарифний план. Містить посилання на користувача та відповідний платіж.

SubPay зберігає інформацію про оплату тарифних планів: сума, валюта, дата оплати. Зв’язується із тарифним планом, до якого здійснено платіж.

TariffPlan описує умови підписок у системі. Вказується тривалість дії плану, його вартість та валюта.

ParkingBalance дозволяє вести фінансовий облік кожного паркінгу: поточний залишок та валюта. Це необхідно для розрахунків між адміністрацією паркінгу та власниками.

BonusesCost визначає політику нарахування бонусів: яку суму коштів потрібно внести для отримання одного бонусу, та в якій валюті ведеться розрахунок.

CancellationPolicies зберігає правила скасування бронювання: за скільки годин до початку потрібно відмовитись, щоб отримати певний відсоток повернення коштів.

SalesPolicies дозволяє реалізовувати знижки — як для всіх користувачів паркінгу, так і для конкретних осіб. Вказується відсоток знижки, її дія та належність до користувача або всіх.

## 3.3 Проектування архітектури ПЗ

### 3.3.1 Проектування архітектури серверної частини

Архітектура серверної частини програмної системи побудована за принципами модульної багатошарової архітектури [8], що забезпечує чітке розділення відповідальностей та високу масштабованість рішення.

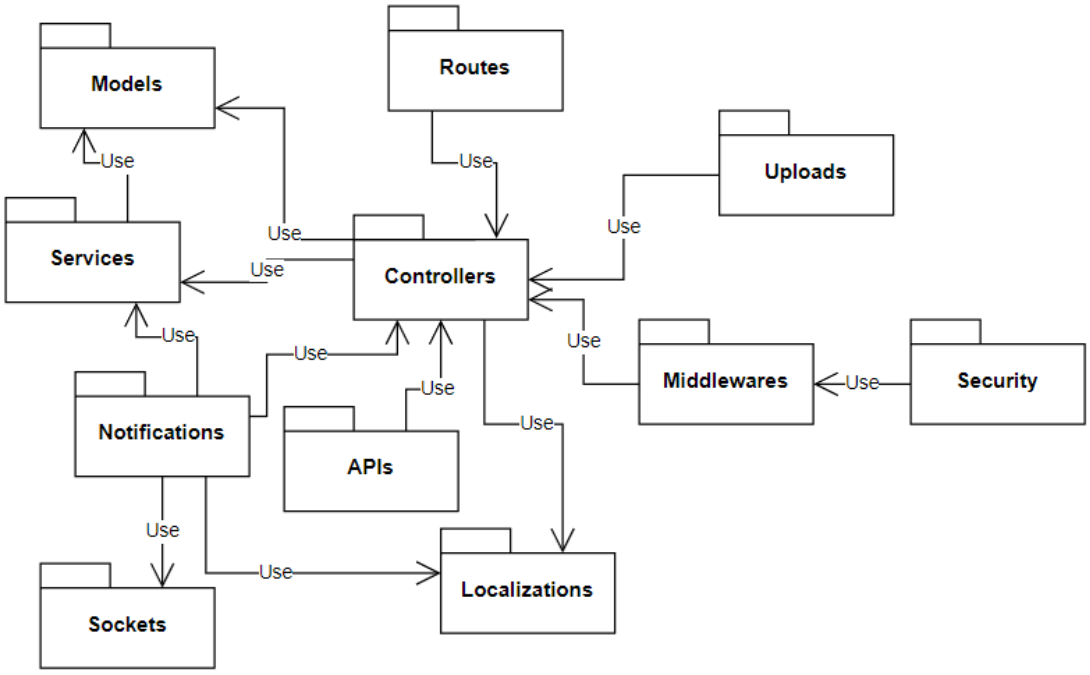


Рисунок 3.8 – UML - діаграма пакетів для серверної частини (рисунок виконаний самостійно)

Архітектура поділена на функціональні блоки (пакети), кожен з яких відповідає за окремий аспект логіки застосунку. Пакети наведені на UML – діграмі пакетів (див. Рисунок 3.8).

Центральним елементом архітектури виступає пакет Controllers, який координує взаємодію між користувачем і внутрішніми службами. Контролери приймають запити, валідують вхідні дані, викликають відповідні сервіси (Services) для обробки бізнес-логіки, після чого формують відповідь. Контролери тісно взаємодіють із маршрутами (Routes), які визначають, який саме контролер обробляє конкретний HTTP-запит. Таке розділення дозволяє централізовано керувати маршрутами, спрощує їх тестування та оновлення.

Пакет Services інкапсулює бізнес-логіку системи та використовує модельний рівень (Models) для доступу до даних. Цей підхід дозволяє уникати дублювання коду та полегшує зміну логіки, не торкаючись контролерів або моделей. Сервіси також використовуються іншими частинами системи — наприклад, модулями сповіщень (Notifications) і сокетами (Sockets), що свідчить про їхню повторну використовуваність та централізованість логіки.

Компонент APIs інтегрується з контролерами та зовнішніми чи внутрішніми API-інтерфейсами, виконуючи роль адаптера між системою та зовнішнім середовищем. Пакет Middlewares реалізує перехоплення та обробку запитів перед передачею їх у контролери, наприклад, для автентифікації, валідації або логування. Самі ж механізми безпеки винесено в окремий пакет Security, який використовується Middlewares, що забезпечує централізацію обробки автентифікації та авторизації.

Модуль Localizations взаємодіє як із API, так і з контролерами, надаючи інтерфейси для підтримки багатомовності, а Uploads відповідає за завантаження та обробку файлів. Всі ці модулі не мають жорсткої залежності один від одного, що свідчить про дотримання принципу низької зв’язаності.

Обрана архітектура забезпечує високу гнучкість у розширенні функціональності. Наприклад, додавання нового типу сповіщень або нового API-інтерфейсу не потребує змін у базових сервісах чи контролерах. Завдяки модульності легко впроваджувати нові сервіси або адаптувати існуючі під змінені вимоги без порушення цілісності системи. Розділення логіки, обробки запитів і доступу до даних полегшує тестування та налагодження, сприяє впровадженню

CI/CD-підходів і дотриманню принципів SOLID. Таким чином, дана архітектура є оптимальним вибором для розробки масштабованої, розширюваної та підтримуваної веб-системи.

### 3.3.2 Проектування архітектури мобільного застосунку

Структура додатку побудована з використанням сучасного підходу MVVM (Model-View-ViewModel) [7, 8], що забезпечує чітке розділення логіки, інтерфейсу та даних. Основні компоненти розподілені між пакетами, як показано на UML-діаграмі (див. Рисунок 3.9).

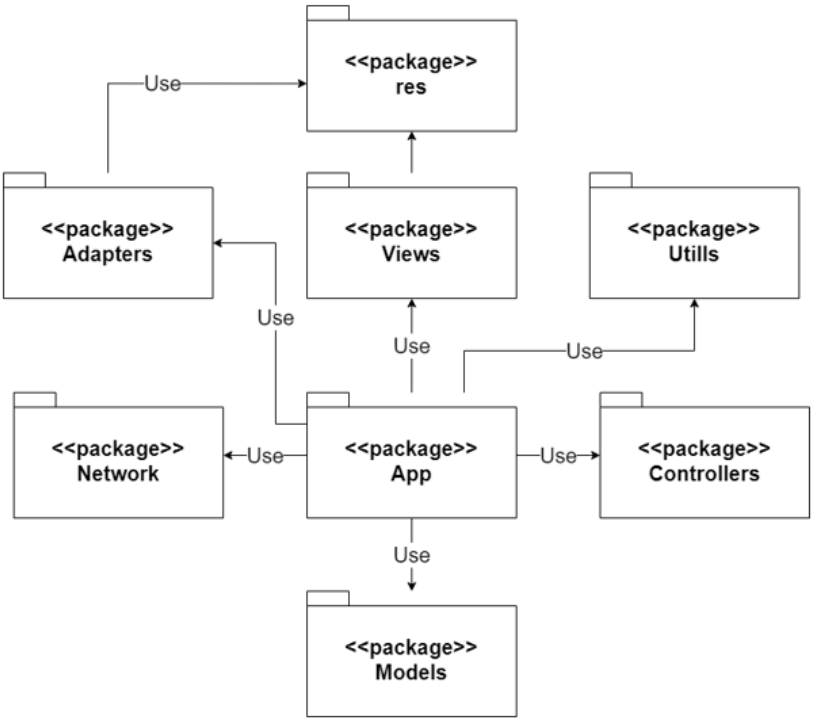


Рисунок 3.9 – UML - діграма пакетів для мобільного застосунку (рисунок виконаний самостійно)

Основу архітектури складають моделі (Models) [8], які відповідають за бізнес-логіку та взаємодію з даними. Вони ізольовані від інтерфейсу, що забезпечує незалежність логіки додатку від способу її відображення. Користувацький інтерфейс [9, 10] реалізований у пакеті Views, де відбувається візуалізація даних. Для зв’язку між інтерфейсом і логікою використовується ViewModel (розміщений

у пакеті Network). Вона виступає посередником: транслює дані з моделей у формат, зрозумілий для UI, та обробляє стан інтерфейсу, зберігаючи його навіть при зміні конфігурації (наприклад, під час повороту екрана).

Додаткові компоненти, такі як Adapters, забезпечують гнучке відображення даних у списках, а Utils містять допоміжні класи для оптимізації коду. Базова конфігурація додатку зосереджена у пакеті App, а ресурси (макети, іконки, тексти) винесені в окремий модуль res.

Переваги обраної архітектури включають чітке розмежування відповідальностей між компонентами, що значно знижує ризик конфліктів у коді та спрощує внесення змін. Взаємодія через ViewModel гарантує, що інтерфейс залишається «чистим» від бізнес-логіки, а дані зберігають актуальність незалежно від зовнішніх змін. Крім того, модульна структура дозволяє масштабувати додаток: нові функції інтегруються без порушення роботи існуючих систем, що є критичним для довгострокової підтримки проекту.

Такий підхід забезпечує гнучкість у адаптації до змін у вимогах, що є ключовим для сучасних мобільних додатків.

### 3.3.3 Проектування архітектури веб – клієнту

Структура веб-додатку базується на React [11] та Axios, що забезпечує модульність, гнучкість і легкість у підтримці. Архітектура організована за модульним принципом [8] з чітким розділенням відповідальностей між компонентами, що дозволяє ефективно масштабувати проект і інтегрувати нові функції без порушення існуючої логіки [7]. UML-діаграма пакетів, яка наведена на рисунку 3.10, візуалізує структуру додатку, відображаючи взаємозв’язки між основними модулями. Вона демонструє, як компоненти (API, Pages, Utils, Components, Styles, App, Locales) організовані у логічні групи та взаємодіють між собою.

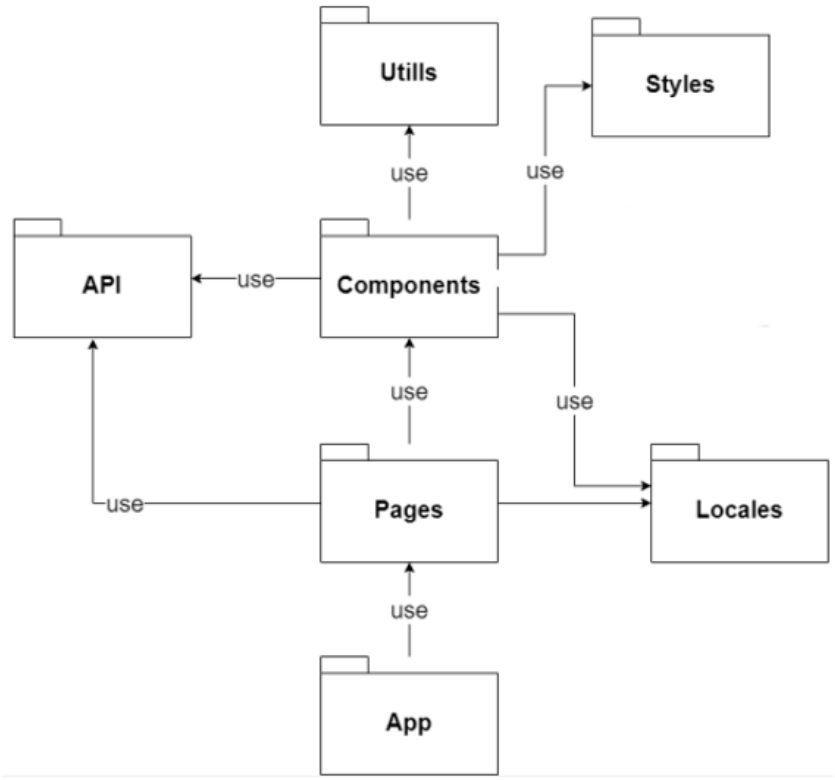


Рисунок 3.10 – UML - діаграма пакетів для веб – клієнту (рисунок виконаний самостійно)

Основу додатку складає модуль API, який інкапсулює всю взаємодію з бекендом. Тут розміщені моделі даних, що описують структуру інформації, яка передається між клієнтом і сервером, та apiService — сервіс на базі Axios. Цей сервіс виконує HTTP-запити (GET, POST, PUT тощо), обробляє помилки, додає заголовки та інтерсептори, забезпечуючи централізоване управління комунікацією. Використання моделей дозволяє стандартизувати формат даних, що спрощує їхню валідацію та трансформацію.

Інтерфейс додатку побудований за допомогою React-компонентів, які поділені на два типи: презентаційні (відповідають за візуалізацію) та контейнери (обробляють логіку, взаємодію з API). Таке розділення дозволяє повторно використовувати компоненти та ізолювати бізнес-логіку від UI. Наприклад, контейнери отримують дані через apiService і передають їх у презентаційні компоненти, які лише відображають інформацію.

Сторінки додатку згруповані в модулі Pages, де кожен екран формується шляхом композиції компонентів. Цей підхід спрощує навігацію (за допомогою React Router) та забезпечує логічну ієрархію. Для підтримки багатомовності використовується модуль Locales, який зберігає текстові ресурси для різних мов, що робить додаток доступним для міжнародної аудиторії.

Допоміжні функції, такі як форматування дат, валідація полів або утиліти для роботи з даними, зосереджені в Utils. Це усуває дублювання коду та забезпечує єдиний джерело істини для повторюваних операцій. Стилізація реалізована через модуль Styles, де містяться глобальні CSS-правила, теми та CSS-модулі для компонентів. Таке рішення забезпечує узгодженість дизайну та спрощує внесення змін у візуальну частину.

Базову конфігурацію додатку (ініціалізацію маршрутизації, підключення сервісів, налаштування провайдерів контексту або Redux) винесено в модуль App. Це дозволяє централізовано керувати основним функціоналом і швидко адаптувати додаток до нових вимог.

Ключовою перевагою такої архітектури є її модульність. Кожен компонент виконує чітко визначену роль, що спрощує тестування, відладку та розподіл задач між розробниками. Наприклад, зміни в API не впливають на компоненти, оскільки вся логіка комунікації інкапсульована в окремому сервісі. Використання Axios разом із моделями забезпечує безпеку типів даних і зменшує ймовірність помилок. Крім того, підтримка інтернаціоналізації та стандартизовані стилі роблять додаток гнучким і готовим до масштабування.

Така структура ідеально підходить для великих проектів, де важлива стабільність, чистота коду та можливість швидкої адаптації до змін.

## 3.4 Приклади найцікавіших алгоритмів та методів

### 3.4.1 Алгоритм роботи IoT частини

Алгоритм роботи IoT-термінала є ключовим компонентом системи керування паркуванням, оскільки забезпечує взаємодію між користувачем, сервером та фізичними пристроями. Він організований у вигляді послідовності дій,

які автоматизують процес ініціації паркувальної сесії, перевірки даних та обробки результатів. Цей алгоритм відображає логіку роботи терміналу, враховуючи можливі сценарії успіху та помилок, що робить систему надійною та зручною для користувачів. Роботу IoT терміналу добре відображає UML – діаграма діяльності (див. Рисунок 3.11).

Після увімкнення IoT-термінал [5] автоматично підключається до мережі, забезпечуючи стабільний зв’язок із серверною частиною. Користувач сканує QR-код або інший ідентифікатор паркувального місця, що ініціює зчитування даних (наприклад, унікального ID місця). Ця інформація передається на сервер через API-запит, де відбувається перевірка бронювання: система аналізує базу даних, зіставляючи ідентифікатор місця з ідентифікатором користувача та часом бронювання.

Якщо сервер підтверджує валідність даних, IoT-термінал отримує відповідь «Сесію створено» — на екрані з’являється повідомлення «Паркування розпочато», а фізичний пристрій (шлагбаум) активує механізм відкриття. Паралельно термінал фіксує час початку сесії та надсилає підтвердження на сервер для оновлення статусу місця.

У випадку помилки (наприклад, бронювання не знайдено, дані не збігаються або місце зайняте) сервер повертає статус «Незбіг» або «Не знайдено». Термінал відображає повідомлення «Це місце заброньовано», супроводжуючи його звуковим сигналом або підсвіткою для додаткової індикації. Користувач має можливість повторно відсканувати код або звернутися до служби підтримки для вирішення проблеми.

Весь процес будується на автоматизації, що мінімізує ручне втручання, та інтеграції з IoT-пристроями, які забезпечують фізичний контроль доступу. Алгоритм враховує різні сценарії, гарантуючи надійність системи та зручність для користувачів.

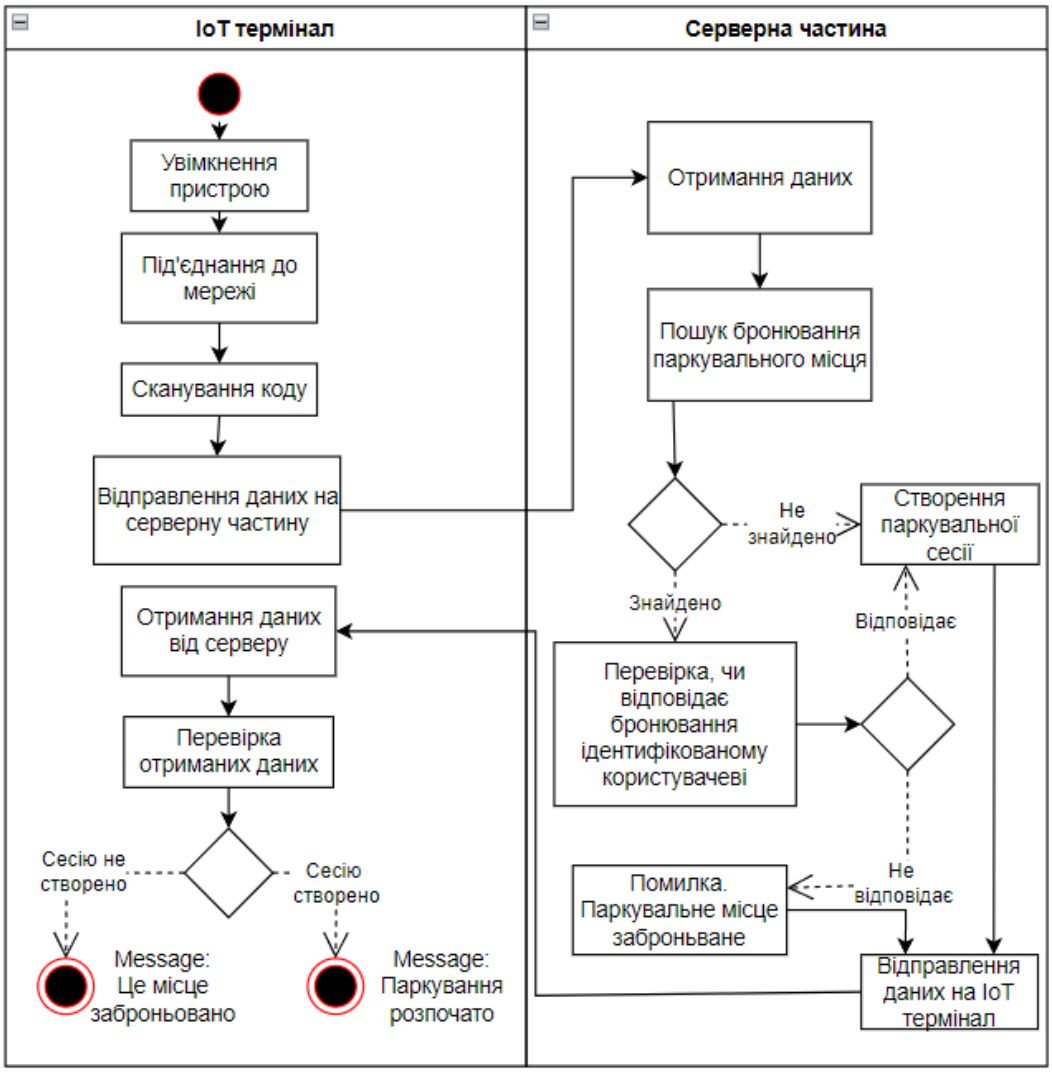


Рисунок 3.11 – UML - діаграми діяльності для IoT терміналу (рисунок виконаний самостійно)

UML - Діаграма станів IoT-терміналу візуалізує життєвий цикл пристрою, відображаючи його перехід між різними станами під час взаємодії з користувачем та серверною частиною. Вона описує, як термінал реагує на події (наприклад, успішне сканування коду, помилка мережі) та змінює свою поведінку. Ця діаграма є інструментом для аналізу логіки роботи терміналу, допомагаючи оптимізувати послідовність дій та мінімізувати ризики збоїв. Вона також підкреслює взаємозв’язок між програмним забезпеченням та фізичними компонентами системи. Сама діаграма наведена на рисунку 3.12.

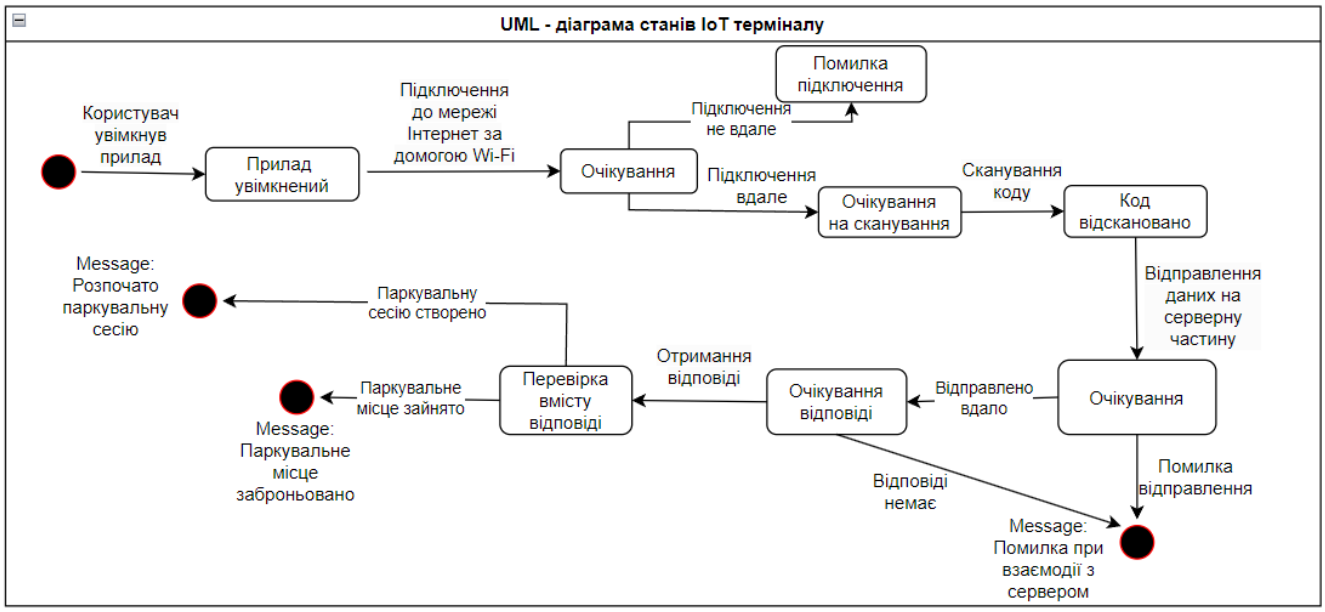


Рисунок 3.12 – UML - діаграма станів для IOT пристрою (рисунок виконаний самостійно)

Після увімкнення IoT-термінал переходить у початковий стан, де намагається підключитися до мережі через Wi-Fi. Якщо підключення вдале, пристрій переходить у режим очікування сканування коду. Користувач сканує QR-код паркувального місця, і термінал передає зчитані дані на сервер. У разі невдалого сканування система пропонує повторити спробу, залишаючись у стані готовності.

Якщо передача даних на сервер відбувається успішно, термінал очікує відповіді. При отриманні підтвердження бронювання («Паркувальну сесію створено») він активує фізичний пристрій (наприклад, відкриває шлагбаум) та відображає повідомлення «Розпочато паркувальну сесію». Якщо сервер повідомляє, що місце зайняте або не відповідає бронюванню, термінал інформує користувача через повідомлення «Паркувальне місце заброньовано».

У випадку технічних збоїв, таких як невдала передача даних або втрата зв’язку з сервером, система виводить повідомлення «Помилка при взаємодії з сервером» і повертається до стану очікування, даючи можливість повторити дії. На кожному етапі алгоритм враховує можливі помилки, автоматично перенаправляючи користувача або запропонуючи рішення, що робить процес інтуїтивним.

Діаграма станів наочно демонструє, як IoT-термінал керує складними сценаріями, поєднуючи автоматизацію, обробку даних і фізичну взаємодію. Вона підкреслює надійність системи, здатність адаптуватися до змін та зручність для користувача, забезпечуючи безперебійну роботу навіть у разі непередбачених ситуацій.

### 3.4.2 Алгоритми серверної частини

Бізнес-логіка серверної частини [12] системи бронювання паркомісць реалізує низку важливих функціональних сценаріїв, що забезпечують коректну взаємодію користувача з сервісом: від створення бронювання до його скасування, оплати через сторонні сервіси, обробки підписок, захисту персональних даних тощо. У цьому підрозділі розглянемо ключові алгоритми та їх реалізацію на прикладі програмного коду, а також звернемося до UML-діаграм діяльності (див. Рисунок 3.13), яка моделює життєвий цикл основних сутностей системи.

Однією з центральних функцій є створення бронювання. Метод createReservation (див. Додаток Г) виконує повний цикл перевірок і дій, необхідних для фіксації резервування паркомісця. Алгоритм починається з валідації вхідних параметрів (ідентифікатор паркінгу, транспортного засобу, час початку й завершення). Далі відбувається переведення часу у формат UTC, щоб уникнути помилок, пов’язаних із часовими зонами. Окрема увага приділяється категорії транспортного засобу: вона використовується для фільтрації сумісних паркомісць.

Після цього система аналізує вже створені бронювання, які перетинаються у часі з новим запитом. Застосовується логічна перевірка за допомогою конструкції Op.not(Op.or(...)), яка дозволяє виключити з результату зайняті місця. В результаті формується список доступних паркомісць, серед яких вибирається перше за списком.

Після вибору місця обчислюється загальна вартість резервування. Система підтримує гнучке ціноутворення: ціна залежить від тривалості бронювання та параметру PriceTimeDuration, який задає мінімальну одиницю часу для розрахунку вартості.

Наприклад:

```
      const [hours, minutes, seconds] =
selectedPlace.PriceTimeDuration.split(':').map(Number);
      const unitMs = ((hours * 60 + minutes) * 60 + seconds) * 1000;
      const units = Math.ceil(durationMs / unitMs);
      const totalPrice = currentPrice.mul(units);
```

Після успішного бронювання система створює відповідний запис у базі даних із зазначенням часу, статусу та зв’язків з транспортним засобом і паркомісцем. На UML-діаграмі станів для об'єкта Reservation цей етап позначається переходом у стан active після створення. Стан може змінитися на skipped, completed або expired залежно від подальших дій.

Скасування бронювання реалізується через метод skipReservation (див. Додаток Д), який включає логіку обробки політики скасування та повернення коштів. Спочатку перевіряється наявність бронювання та його статус. Далі система визначає, скільки часу залишилося до початку бронювання, і на основі цього обирає відповідну політику повернення коштів. Якщо політика відсутня, відбувається повне повернення суми користувачу через сервіс UserBalanceService. Інакше, розраховується відсоток утримання (CancellationFeePercent) і сума, яка підлягає поверненню. Код, що виконує ці обчислення, виглядає так:

```
      const cancelPercent = new Decimal(policy.CancellationFeePercent);
      const fee =
originalAmount.mul(cancelPercent.div(100)).toDecimalPlaces(2);
      const refund = originalAmount.minus(fee).toDecimalPlaces(2);
```

UML-діаграма діяльності (див. Рисунок 3.14) Reservation показує, як об'єкт переходить із active до skipped, при цьому паралельно створюється транзакція типу revenue для власника паркінгу, якщо було утримання коштів.

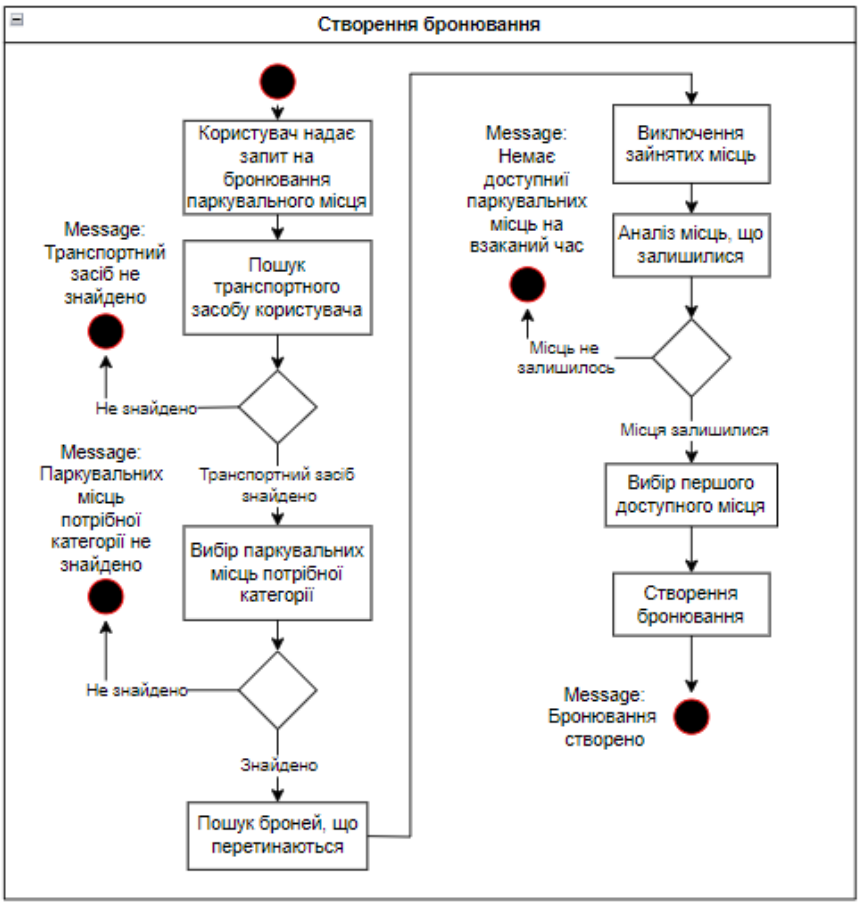


Рисунок 3.13 – UML - діаграма діяльності для створення бронювання (рисунок виконаний самостійно)

Важливу роль у бізнес-логіці відіграє система платежів. Реалізована інтеграція з двома провайдерами: LiqPay та PayPal. У випадку з LiqPay (див. Додаток Е) для створення платіжного замовлення формується об'єкт з параметрами платежу, який кодується у формат base64, а потім підписується за допомогою SHA1-хешування приватного ключа:

```
function generateSignature(data) {
    return crypto.createHash('sha1')
        .update(LIQPAY_PRIVATE_KEY + data + LIQPAY_PRIVATE_KEY)
        .digest('base64');
}
```

LiqPay надсилає підтвердження через webhook-запит, який перевіряється на достовірність шляхом перевірки цифрового підпису. У випадку успіху транзакція оновлюється до статусу success.

Аналогічно працює інтеграція з PayPal (див. Додаток Ж). Спочатку отримується access token, потім створюється замовлення і користувач перенаправляється на сторінку оплати. Після підтвердження платежу виконується запит на фінальне capture, після чого система фіксує результат.

Ще однією важливою частиною є захист даних користувачів. Для цього використовується симетричне шифрування на базі алгоритму AES-256-CBC. Секретний ключ і вектор ініціалізації зберігаються в середовищі виконання і повинні відповідати стандартам безпеки:

```
        const encryptData = (someData) => {
            if (!someData) return null;
            const cipher = crypto.createCipheriv(algorithm, key, iv);
            let encrypted = cipher.update(someData, 'utf8', 'hex');
            encrypted += cipher.final('hex');
            return encrypted;
};
        const decryptData = (someData) => {
            if (!someData) return null;
            const decipher = crypto.createDecipheriv(algorithm, key, iv);
            let decrypted = decipher.update(someData, 'hex', 'utf8');
            decrypted += decipher.final('utf8');
            return decrypted;
};
```

Функції encryptData та decryptData забезпечують безпечне зберігання конфіденційної інформації у базі даних.

Окрему роль у бізнес-логіці відіграє підписка на сервіс. Система дозволяє користувачам оформлювати підписки, які можуть зменшувати вартість паркування або надавати пріоритетний доступ до паркомісць. Підписки зберігаються як окремі об'єкти з терміном дії, і перед створенням кожного нового бронювання перевіряється активність підписки, щоб застосувати знижки.

Також реалізовано керування балансом користувача, що є окремою бізнес-сутністю. Баланс оновлюється при поповненні, поверненні коштів, списанні платежів, і кожна операція супроводжується транзакційним записом. Це забезпечує прозорість фінансових операцій і дозволяє проводити аудит у разі потреби.

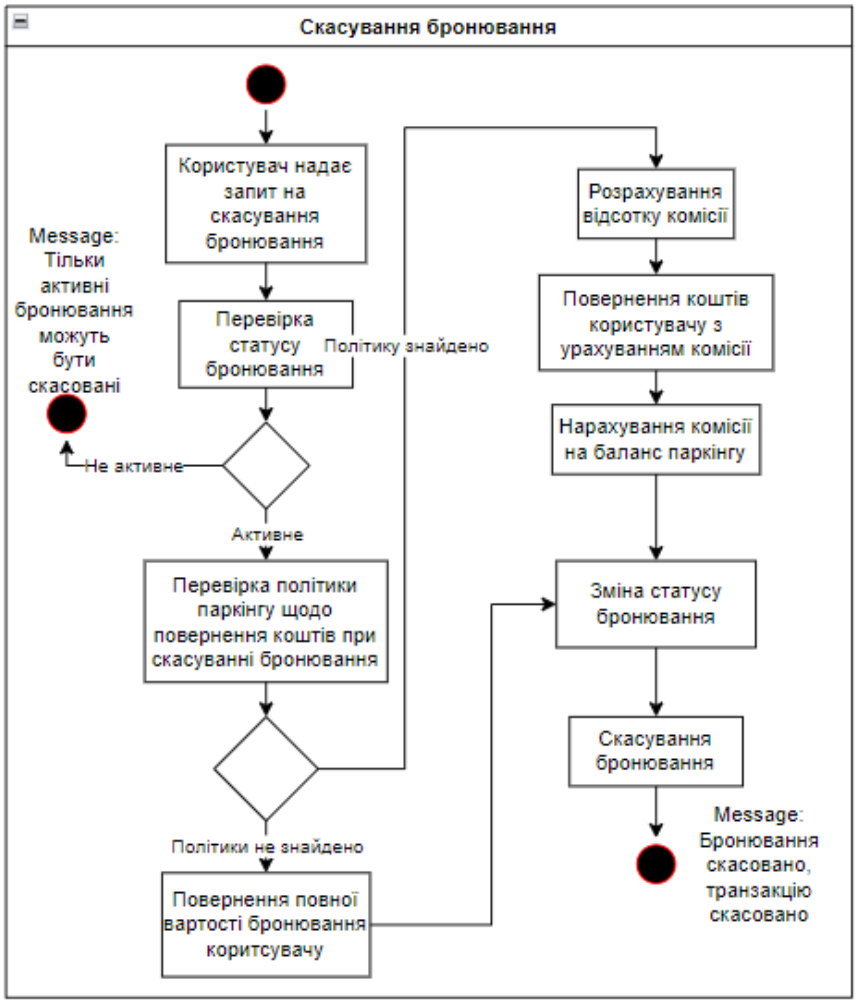


Рисунок 3.14 – UML - діаграма діяльності для скасування бронювання (рисунок виконаний самостійно)

Узагальнюючи, бізнес-логіка системи є багатокомпонентною та охоплює численні аспекти роботи з бронюванням, фінансами, безпекою та користувацьким досвідом. Вона підтримується чітко визначеними алгоритмами, що відповідають вимогам надійності та масштабованості, а також формалізована через UML-діаграми, які дозволяють візуально простежити всі можливі переходи між станами об'єктів.

# 4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

## 4.1 Використання сервісу ngrok

З метою забезпечення публічного доступу до серверного застосунку, що розгорнутий локально під час розробки, було прийнято рішення використати сервіс тунелювання ngrok. Даний сервіс надає можливість створити захищений публічний HTTPS-домен, який безпосередньо пересилає вхідні запити з мережі Інтернет до локального сервера, запущеного на робочому комп'ютері розробника. Таким чином, розробник отримує змогу проводити тестування, обробку зовнішніх HTTP-запитів, а також демонструвати функціональність додатку без необхідності розгортання його на віддаленому хостингу.

На рисунку 4.1 зображено приклад відкритого ngrok-тунелю, за допомогою якого публічно доступний локальний сервер, що прослуховує порт 5192, став публічно доступним.

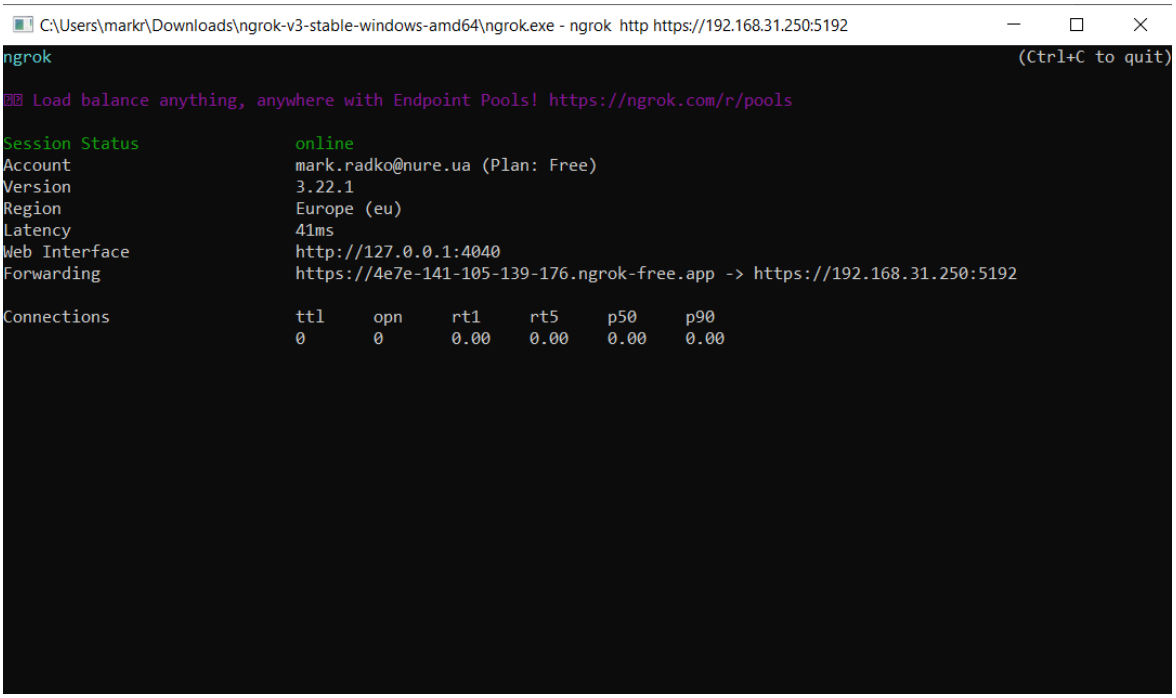


Рисунок 4.1 – Відкритий ngrok-тунель (рисунок виконаний самостійно)

Для відкриття тунелю використовується така команда:

```
ngrok http https://192.168.31.250:5192.
```

У даному випадку, ngrok надає домен на кшталт: https://4e7e-141-105-139-176.ngrok-free.app, який використовується як основа для формування повних URL-адрес до ресурсів сервера.

Окрім забезпечення публічного доступу, ngrok відіграє важливу роль у взаємодії з платіжною системою LiqPay. Після успішного проведення оплати, LiqPay надсилає POST-запит на зазначену callback-URL адресу:

```
server_url: `${process.env.NGROK_DOMAIN}/webhook`
```

Завдяки використанню ngrok, сервер отримує ці запити навіть у локальному середовищі, що дозволяє повноцінно тестувати бізнес-логіку обробки платежів.

Ще однією задачею, яку вдалося вирішити за допомогою додаткового маршруту на сервері, є обхід системного попередження ngrok при прямому зверненні до статичних ресурсів, зокрема зображень. При спробі завантажити зображення безпосередньо через посилання на ngrok-домен, ngrok може відобразити проміжну сторінку безпеки або заблокувати запит (див. Рисунок 4.2). Щоб цього уникнути, було реалізовано проксі-маршрут, який отримує запит від клієнта на певну адресу, а сервер самостійно читає відповідний файл із диску і відправляє його у відповідь. Це дозволяє обійти обмеження ngrok і забезпечити стабільне відображення зображень у клієнтському інтерфейсі.

Фрагмент відповідного коду реалізації маршруту виглядає наступним чином:

```
app.get('/proxy-image/:imageName', (req, res) => {
    const { imageName } = req.params;
    const imagePath = path.join(__dirname, 'uploads', imageName);
    if (fs.existsSync(imagePath)) {
        console.log(imagePath)
        return res.sendFile(imagePath);
    }
```

```
        return res.status(404).send('Image not found');
    });
```

У наведеному коді маршрут “/proxy-image/:imageName” приймає параметр imageName з URL-адреси, формує абсолютний шлях до файлу у локальній директорії uploads, перевіряє його наявність і, у разі успіху, надсилає зображення як відповідь. Якщо файл відсутній, клієнту повертається код 404 та відповідне повідомлення.

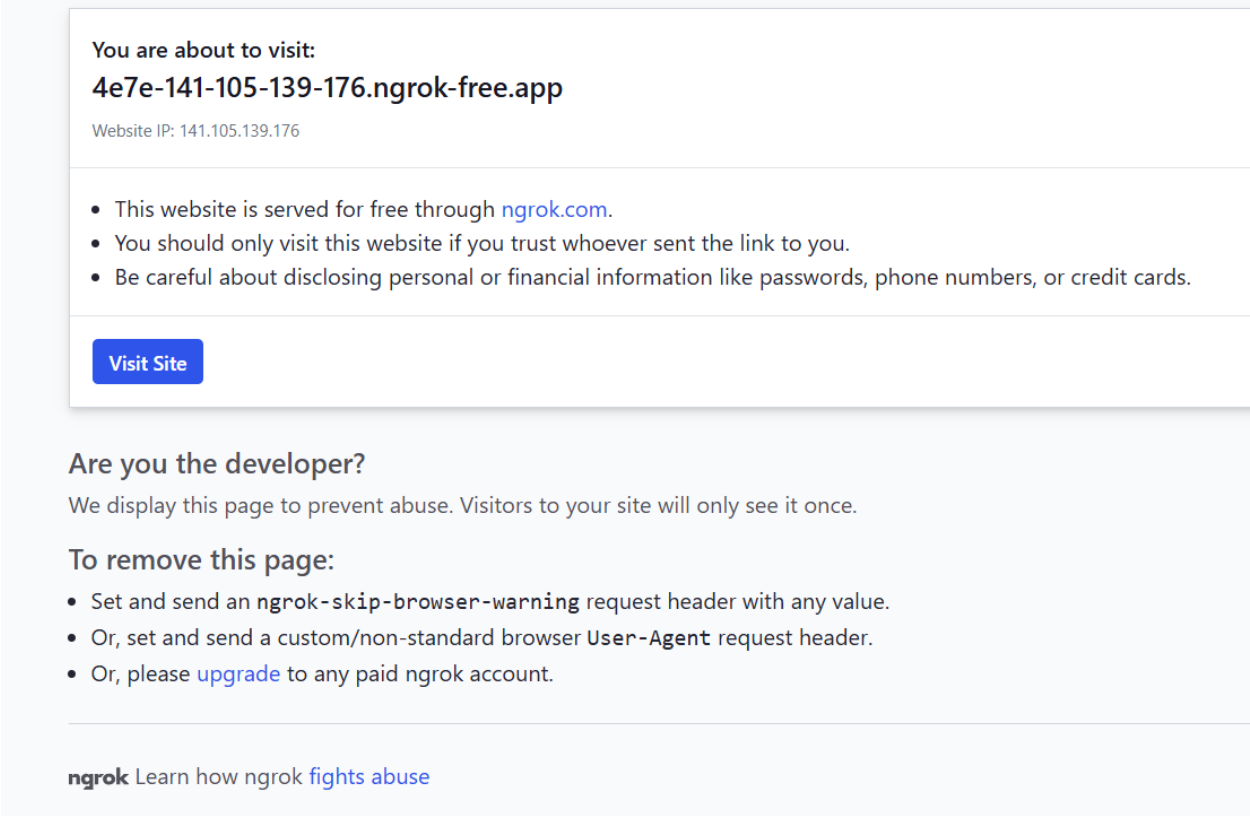


Рисунок 4.2 – Проміжна сторінка безпеки ngrok (рисунок виконаний самостійно)

Для формування повного посилання на зображення у клієнтській частині використовується змінна середовища NGROK_DOMAIN, до якої додається шлях до зображення за проксі-маршрутом. Формування URL картинки в контроллерах серверної частини виглядає так:

```
PhotoImage: `${process.env.NGROK_DOMAIN}/proxy-image/
${path.basename(parking.PhotoImage)}`
```

Таким чином, зображення передаються через посередника – сервер, а не напряму, що забезпечує стабільну інтеграцію з ngrok-доменом. Подібний підхід знайшов широке застосування у процесі розробки та тестування даного проєкту.

## 4.2 Налаштування HTTPS-взаємодії між клієнтом і сервером

Для забезпечення безпечної взаємодії між клієнтськими та серверним застосунками було реалізовано підтримку шифрування даних за допомогою протоколу HTTPS. Передавання інформації у відкритому вигляді через протокол HTTP створює серйозні ризики перехоплення або модифікації даних третіми особами, особливо у випадках, коли йдеться про обробку конфіденційної інформації, наприклад, платіжних реквізитів, облікових даних чи персональних зображень користувачів. У зв’язку з цим шифрування трафіку є критично важливим етапом при створенні сучасних вебсервісів і мобільних застосунків.

З метою реалізації HTTPS було прийнято рішення використати інструмент mkcert, який дозволяє швидко створювати самопідписані SSL-сертифікати, довірені локальною операційною системою або браузером. Mkcert створює власний кореневий сертифікат та встановлює його у список довірених, що дозволяє уникнути попереджень про недійсні сертифікати в браузерах і мобільних пристроях під час розробки.

Для серверного застосунку, що розгортається у локальній мережі за IP-адресою 192.168.31.250, було згенеровано пару ключів — сертифікат (192.168.31.250.pem) та приватний ключ (192.168.31.250-key.pem) командою:

```
mkcert 192.168.31.250
```

Отримані файли були підключені до Express-серверу при створенні HTTPS-сервера із використанням вбудованого модуля https:

```
const options = {
    key: fs.readFileSync('192.168.31.250-key.pem'),
    cert: fs.readFileSync('192.168.31.250.pem'),
    minVersion: 'TLSv1.3',
};
(async () => {
    try {
        const server = https.createServer(options, app);
        const LOCAL_IP = process.env.LOCAL_IP
        server.listen(PORT,  LOCAL_IP, () => {
            console.log(`Server started at
https://${LOCAL_IP}:${PORT}`);
        });
    } catch (error) {
        console.error('Unable to start the server:', error);
    }
})();
```

Таким чином, сервер надійно шифрує всі відповіді та приймає захищені запити клієнтів, які звертаються до нього за цією IP-адресою. Це дозволило безпечно виконувати запити, зокрема від Android-застосунку, у межах локальної мережі.

Крім того, для локального клієнта, який звертається до сервера через localhost, було згенеровано окремі сертифікати командою:

```
mkcert localhost
```

Це дозволило налаштувати безпечне з’єднання також у випадках, коли клієнтська частина запускається локально на тому ж комп’ютері, де працює сервер. Усі сертифікати були збережені у відповідних директоріях та підключені в налаштуваннях HTTPS як на серверному рівні, так і під час конфігурації клієнтської логіки, зокрема у запитах, що використовують модулі axios або fetch.

Завдяки впровадженню HTTPS-взаємодії вдалося забезпечити конфіденційність, цілісність і достовірність усіх даних, які передаються між клієнтом і сервером у рамках проєкту, що відповідає сучасним стандартам інформаційної безпеки.

## 4.3 Реалізація функцій контролю доступу

У рамках серверної частини програмного забезпечення було реалізовано низку middleware-функцій, які відповідають за попередню перевірку запитів, що надходять на захищені маршрути. Middleware виконує роль проміжного обробника запитів: перш ніж той потрапляє до основної бізнес-логіки, система перевіряє, чи відповідає користувач необхідним умовам доступу. Це дозволяє централізовано впровадити механізми авторизації, перевірки ролей, заборони доступу для заблокованих користувачів, а також перевірки активності підписки.

Однією з базових middleware-функцій є authMiddleware, яка відповідає за перевірку JWT-токена у заголовку запиту. Якщо токен відсутній, некоректний або прострочений, користувачу повертається повідомлення про помилку авторизації. Також у межах цієї функції виконується розшифрування токена, і вся інформація про користувача зберігається у полі req.user, що дозволяє подальшим middleware-функціям або обробникам маршрутів отримати доступ до даних користувача. Фрагмент коду виглядає наступним чином:

```
const authHeader = req.headers.authorization;
if (!authHeader) {
    return res.status(401).json("Not authorised");
}
const token = authHeader.split(' ')[1];
if (!token || token === 'Bearer') {
    return res.status(401).json("Not authorised");
}
req.user = jwt.verify(token, process.env.JWT_SECRET_KEY);
if (req.user.IsBanned) {
    return res.status(402).json("User account is banned");
```

```
}
```

Наступною middleware-функцією є checkAdminRoleMiddleware, яка виконується після authMiddleware та перевіряє, чи має користувач роль адміністратора. Якщо роль не відповідає значенню "admin", запит негайно відхиляється з відповідним повідомленням. Такий підхід дозволяє захистити адміністративні маршрути від доступу звичайних користувачів, навіть якщо вони є авторизованими.

Окремої уваги заслуговує складніша за логікою middleware-функція checkSubscriptionMiddleware, яка перевіряє наявність активної підписки в користувача, що є відповідальним менеджером паркування. Спочатку система знаходить обліковий запис менеджера у базі даних, використовуючи ідентифікатор користувача з JWT-токена. Якщо запис не знайдено або користувач не прив’язаний до жодного паркування, запит блокується. Далі система перевіряє, чи існує у цього паркування власник з роллю "owner" — саме через нього здійснюється підписка. Якщо активна підписка відсутня, система повертає повідомлення про заборону доступу. Також у тілі запиту (req) зберігаються додаткові службові дані: ідентифікатор паркування (req.ParkingID), ідентифікатор власника (req.OwnerID), а також ідентифікатор тарифного плану (req.TariffPlanID), що дозволяє подальшій логіці визначити функціональні обмеження. Нижче наведено фрагмент, що перевіряє підписку:

```
const subscription = await Subscription.findOne({
    where: { UserUserID: parkingManagerOwner.UserUserID, isActive: true }
});
if (!subscription) {
    return res.status(403).json({ message: "No active subscription found. Access denied." });
}
const subPay = await SubPay.findByPk(subscription.SubPaySubPayID)
const tariffPlan = await TariffPlan.findByPk(subPay.TariffPlanTariffPlanID)
```

```
req.TariffPlanID = tariffPlan.TariffPlanID
```

Завдяки застосуванню даних middleware-фільтрів було реалізовано гнучку та безпечну систему контролю доступу, яка враховує як технічну авторизацію, так і бізнес-логіку сервісу. Подібна архітектура дозволяє легко масштабувати систему у майбутньому, додаючи нові рівні перевірки без потреби змінювати код кожного окремого маршруту.

## 4.4 Система керування сессіями

У рамках проєкту реалізовано повноцінну систему керування сесіями на основі JWT (JSON Web Token), яка забезпечує безперервну авторизовану взаємодію користувача з сервером. На відміну від класичних сесій, JWT дозволяє зберігати дані авторизації у вигляді токена, що підписаний на стороні сервера і зберігається на клієнті — зазвичай у LocalStorage браузера або в пам’яті мобільного застосунку.

Усі захищені запити до сервера супроводжуються передачею токена в заголовку Authorization. Після певного часу дії (час життя токена задається на сервері під час генерації) токен втрачає чинність. У таких випадках, щоб не примушувати користувача авторизуватись повторно, реалізовано механізм автоматичного оновлення токена, який перевіряє актуальність поточного токена та отримує новий у разі потреби.

### 4.4.1 Керування токеном на стороні веб-клієнту

На клієнтському рівні реалізовано функцію refreshToken, яка виконується періодично або перед надсиланням важливих запитів. Її завдання — перевірити наявність збереженого токена та, у разі його дійсності, звернутися до сервера для отримання оновленої версії. У разі, якщо токен відсутній або визначений як некоректний (наприклад, має значення undefined, або сервер повертає помилку авторизації), виконується процедура автоматичного виходу користувача з облікового запису та перенаправлення його на головну сторінку.

Нижче наведено ключову логіку функції:

```
const storedToken = localStorage.getItem('jwtToken');
if (!storedToken || storedToken === "undefined") {
    return handleLogout();
}
```

Якщо токен існує, клієнтська частина надсилає запит checkAuth до спеціального маршруту на сервері. Якщо відповідь містить новий, валідний токен, він зберігається у localStorage замість старого:

```
const { token } = await apiService.checkAuth(storedToken);
if (!token || typeof token !== 'string') {
    return handleLogout();
}
localStorage.setItem('jwtToken', token);
```

Функція handleLogout, яка викликається у разі невдачі, очищає локальні збережені дані та перенаправляє користувача на публічну частину інтерфейсу:

```
function handleLogout() {
    localStorage.removeItem('jwtToken');
    localStorage.removeItem('Role');
    if (window.location.pathname !== '/home') {
        window.location.href = '/home';
    }
}
```

Таким чином, механізм оновлення JWT забезпечує безпечне та безперервне використання сервісу без необхідності ручного повторного входу в систему, що покращує користувацький досвід та відповідає сучасним стандартам безпеки.

### 4.4.2 Керування токеном на стороні мобільного застосунку

У мобільному клієнті, розробленому на платформі Kotlin, реалізовано власний механізм оновлення JWT-токена через клас JwtTokenController, який виконує роль координатора між локальним сховищем даних, ViewModel та відповіддю від серверного API. Цей клас забезпечує безпечне та своєчасне оновлення токена, необхідного для подальшої авторизованої взаємодії з сервером.

На початковому етапі метод refreshToken отримує токен, збережений у SharedPreferences, та перевіряє його на порожність. У разі відсутності токена виконується обробка помилки через виклик функції onFailure:

```
val storedToken = sharedPreferences.getString("jwtToken", "")
if (!storedToken.isNullOrEmpty()) {
    loginViewModel.refreshToken(storedToken)
} else {
    onFailure()
}
```

Якщо токен знайдено, контролер ініціює звернення до методу refreshToken у loginViewModel, який виконує відповідний API-запит на сервер. Після виконання запиту результат очікується через підписку на LiveData-поле loginResponse, яке містить новий токен у разі успішної авторизації. Новий токен одразу зберігається у SharedPreferences та передається у callback-функцію onSuccess, що дозволяє оновити інтерфейс користувача або надіслати запит з оновленим токеном.

```
    loginViewModel.loginResponse.observe(lifecycleOwner,    Observer    {
response ->
        response?.let {
            val newToken = it.token
            if (newToken != null) {
                sharedPreferences.edit().putString("jwtToken",
newToken).apply()
                onSuccess(newToken)
```

```
            } else {
                onFailure()
            }
        } ?: run {
            onFailure()
        }
    })
```

Також контролер відстежує помилки за допомогою LiveData error, яка оновлюється ViewModel у разі невдачі запиту на сервер. Це дозволяє забезпечити зворотний зв’язок із користувачем та ініціювати автоматичний вихід або повторну авторизацію.

Таким чином, мобільний застосунок забезпечує безпечне та контрольоване зберігання токенів, їхню своєчасну перевірку, оновлення, а також правильну реакцію у разі помилки.

## 4.5 Інтерфейс користувача

Інтерфейс користувача програмної системи Park4Flow розроблений відповідно до сучасних вимог до зручності, адаптивності та візуальної привабливості (див. Рисунок 4.3). Клієнтська частина реалізована на основі React з підтримкою анімацій, інтерактивних компонентів та приємної кольорової гами в зелених тонах. Цей інтерфейс забезпечує швидку взаємодію з сервером без перезавантаження сторінок, що критично важливо для зручної навігації користувача в умовах мобільного та настільного користування.

Кольорова схема оформлення обрана з урахуванням асоціацій з екологічністю та міським простором. Основним акцентом є зелений колір #2ecc71, який задає стилі для кнопок, заголовків та елементів навігації. Для фону використовуються світлі відтінки (#f7fcf9, #ffffff), а для тексту – темно-сірі (#1e293b, #64748b), що забезпечує комфортне читання та чіткий контраст.

Компоненти інтерфейсу організовані у вигляді адаптивної сітки, що автоматично підлаштовується під розмір екрана. Наприклад, у класі .profile-layout

використано grid-template-columns: 1fr 1.5fr, але при зменшенні ширини екрана (мобільна версія) блоки автоматично переходять у вертикальне положення:

```
@media (max-width: 1024px) {
    .profile-layout {
        grid-template-columns: 1fr;
    }
}
```

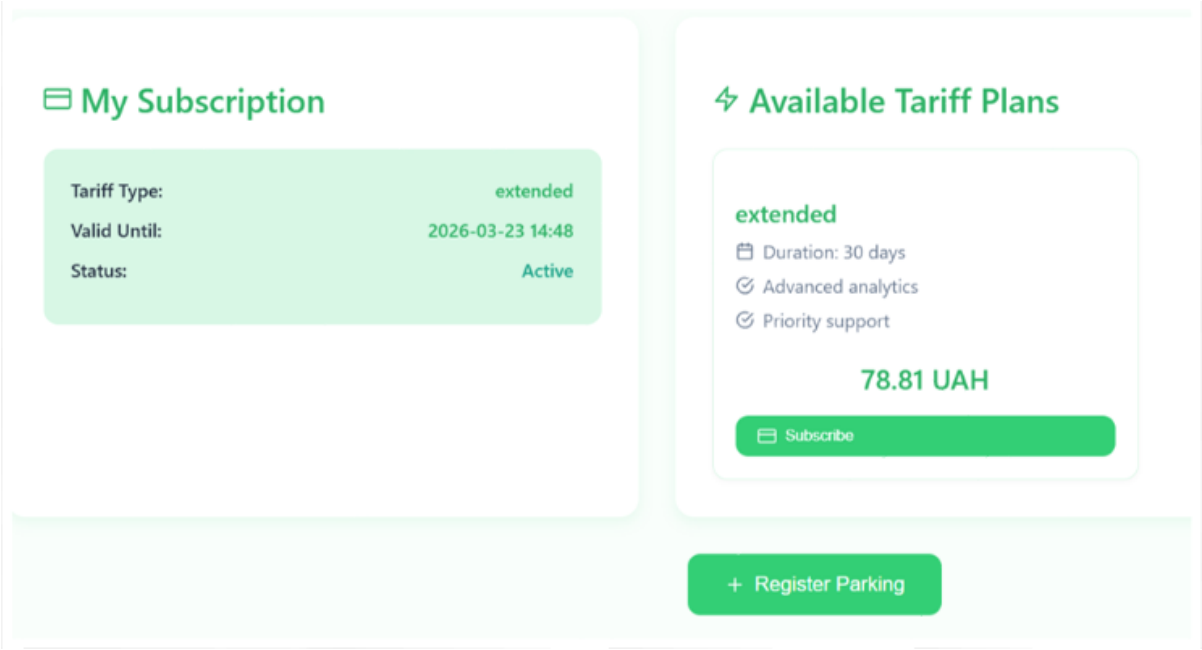


Рисунок 4.3 – Сторінка з керування підпискою на сервіс (рисунок виконаний самостійно)

Більшість інтерфейсних елементів мають закруглені кути, тіні та плавні переходи, що робить інтерфейс візуально м’яким та сучасним. Карточки, такі як блоки профілю, транспортних засобів або бронювання, мають однакову структуру та стилі, що забезпечує єдиний підхід до дизайну:

```
.profile-card {
    background-color: var(--white);
    border-radius: 1rem;
    box-shadow: 0 4px 12px rgba(46, 204, 113, 0.1);
```

```
    padding: 2rem;
}
```

Взаємодія з інтерфейсом супроводжується анімацією fade-in, яка додає плавність при завантаженні елементів на сторінку:

```
@keyframes fadeIn {
    from { opacity: 0; transform: translateY(10px); }
    to { opacity: 1; transform: translateY(0); }
}
.fade-in {
    animation: fadeIn 0.4s ease forwards;
}
```

Крім того, реалізована клієнтська валідація, яка захищає від помилок введення та забезпечує зворотній зв’язок із користувачем. Наприклад, при видаленні транспортного засобу користувачеві обов’язково пропонується підтвердити дію:

```
    if (!window.confirm("Are you sure you want to delete this vehicle?"))
return;
```

Також уся взаємодія з сервером супроводжується відповідними повідомленнями про помилки (alert(error.message)), що дозволяє користувачу розуміти стан операцій.

Отже, інтерфейс системи побудований на сучасних принципах UI/UX-дизайну з акцентом на простоту, зручність та приємну візуальну стилістику. Кожен елемент системи виконує чітко визначену функцію, легко сприймається та підтримується стилістичною єдністю по всьому додатку. Завдяки адаптивному макету, анімаціям, валідації та єдиній кольоровій палітрі користувач отримує приємний досвід взаємодії як з комп’ютера, так і з мобільного пристрою.

# 5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 5.1 Юніт-тестування бізнес-логіки серверної частини

Юніт-тестування є ключовим інструментом забезпечення надійності та стабільності серверної логіки програмного забезпечення. Метою даного виду тестування є перевірка роботи окремих функцій та сервісів у ізоляції від інших компонентів системи. Це дозволяє виявляти помилки ще на ранніх етапах розробки та локалізувати джерела проблем з високою точністю.

Під час тестування серверної частини було зосереджено увагу на сервісі UserDebtService, який відповідає за логіку обліку заборгованостей користувачів. Для написання юніт-тестів використовувався інструмент Jest, який дозволяє створювати моки для моделей, що звертаються до бази даних, та перевіряти виклики функцій.

У прикладі нижче перевіряється коректність створення заборгованості лише в тому випадку, якщо баланс користувача є від’ємним і не існує вже активного боргу:

```
    UserBalance.findOne.mockResolvedValue({ Balance: '-10' });
    UserDebt.findOne.mockResolvedValue(null);
    await UserDebtService.createUserDebt(1, 5, 'USD', 2);
    expect(UserDebt.create).toHaveBeenCalledWith(expect.objectContaining(
{
        UserUserID: 1,
        ParkingParkingID: 2,
        Amount: '5.00',
        Currency: 'USD',
        isRepaid: false
    }));
```

В іншому тесті перевіряється, що функція createUserDebt не створює борг, якщо поточний баланс є позитивним:

```
    UserBalance.findOne.mockResolvedValue({ Balance: '15' });
```

```
    await UserDebtService.createUserDebt(1, 5, 'USD', 2);
    expect(UserDebt.create).not.toHaveBeenCalled();
```

Окрему увагу приділено методам setIsRepaid, який оновлює статус боргу як погашеного у випадку нульового або позитивного балансу, та overdueDebt, що автоматично блокує користувачів у разі простроченої заборгованості більше ніж на 24 години. Наприклад, якщо борг був створений понад добу тому, а баланс користувача досі від'ємний, його обліковий запис буде заблоковано:

```
    UserDebt.findAll.mockResolvedValue([
        { UserUserID: 1, DateAndTime: new Date(Date.now() - 25 * 60 * 60 *
1000) }
    ]);
    UserBalance.findOne.mockResolvedValue({ Balance: '-1.00' });
    await UserDebtService.overdueDebt();
    expect(User.update).toHaveBeenCalledWith(
        { IsBanned: true },
        { where: { UserID: 1 } }
    );
```

Під час юніт-тестування були також виявлені окремі методи, які не пройшли перевірку з першого разу через недоопрацьовану логіку обробки граничних значень або помилок. Зокрема, метод CommissionService. calculateTotalWithCommission не пройшов тест на правильне округлення. При розрахунку суми 123.456 з комісією 7.5% очікуваним результатом було 132.71, однак фактично поверталося 132.72, що пов'язано з округленням за загальними математичними правилами замість фінансових. Цю помилку було виявлено, задокументовано та виправлено.

Крім того, метод CancellationPoliciesService.getWhereParkingAndTime повертав undefined замість об'єкта "{ CancellationFeePercent: 0 }" у разі відсутності відповідної політики або виникнення винятку з бази даних. В обох випадках помилки були виправлені: додано логування помилок та гарантовано повернення коректної структури з нульовим відсотком штрафу.

Детальний перелік результатів юніт-тестування наведено в додатку И, а узагальнена інформація представлена в таблиці И.1.

## 5.2 Тестування REST API через середовище Postman

Для перевірки коректності функціонування серверної частини розробленого програмного забезпечення було проведено тестування REST API за допомогою середовища Postman. Це тестування дозволило імітувати реальні клієнтські запити до серверу з метою перевірки відповідей на них, оцінки стабільності роботи обробників маршрутів, коректності валідації вхідних даних, а також дотримання логіки авторизації та контролю доступу.

Postman надає зручний інтерфейс для формування HTTP-запитів з різними типами даних, заголовками та параметрами, що дозволяє ефективно тестувати як публічні, так і захищені ендпоінти. Під час тестування були створені колекції запитів, що охоплювали всі основні функціональні модулі системи: реєстрацію та авторизацію користувачів, створення та оновлення даних про паркування, перевірку наявності активної підписки, здійснення платіжних операцій та інші.

Особливу увагу було приділено перевірці захищених маршрутів, доступ до яких мають лише користувачі з дійсним JWT-токеном, а також користувачі з роллю адміністратора або активною підпискою. У ході тестування перевірялась реакція сервера на спроби доступу без авторизації або з недійсним токеном, правильність обробки прав доступу та повернення відповідних кодів стану HTTP (401 Unauthorized, 403 Forbidden, 200 OK тощо).

Усі тести було успішно пройдено, сервер стабільно повертав очікувані відповіді, а бізнес-логіка контролерів працювала відповідно до технічних вимог. Узагальнені результати тестування були задокументовані у вигляді таблиці К.1, наведеної у додатку К.

## 5.3 Мануальне тестування функціональності веб-клієнта

Для забезпечення надійності та зручності взаємодії користувача з веб-інтерфейсом розробленого програмного забезпечення було проведено повне мануальне тестування функціональності веб-клієнта. Цей тип тестування дозволяє виявити помилки, які можуть не фіксуватись автоматизованими засобами,

особливо у сфері валідації введення, відображення повідомлень, динаміки інтерфейсу та UX-компонентів.

Мануальне тестування проводилось шляхом прямої взаємодії з веб-додатком у браузері, з фокусом на перевірку ключових сценаріїв використання: реєстрація та авторизація, введення персональних даних, взаємодія з формами, повідомлення про помилки, доступ до захищених сторінок, робота з інтерактивними елементами, адаптивність та загальна логічна послідовність дій користувача.

У результаті тестування було виявлено кілька функціональних недоліків. Зокрема, під час перевірки валідації електронної пошти при вході в систему тестовий сценарій із введенням невалідного email-адресу (test) та валідного пароля (12345678) не викликав жодного повідомлення про помилку, хоча вхід не був здійснений. Це свідчить про відсутність належної перевірки формату електронної адреси. Відповідно, тест не пройдено, і дана частина потребує виправлення.

Аналогічна ситуація була виявлена під час введення некоректного номеру автомобіля у форматі 12345678. Замість очікуваного повідомлення про помилку або попередження, поле дозволяло зберегти введене значення без жодної валідації. Цей тест був позначений як частково пройдений, оскільки логіка перевірки формату номерного знаку поки не реалізована.

Разом з тим, інші функціональні перевірки було успішно пройдено. Наприклад, тест із введенням правильного логіну (test@gmail.com) і неправильного пароля (wrongpass) коректно виводив повідомлення про неправильні облікові дані. Повідомлення відображалося у зрозумілому для користувача форматі, як зображено на рисунку 5.1. Це свідчить про належну обробку помилок авторизації та роботу відповідних компонентів інтерфейсу.

Мануальне тестування також охоплювало оцінку відповідності інтерфейсу загальним вимогам до UX-дизайну, перевірку зручності навігації, наочності повідомлень, коректності адаптації під різні розміри екранів, а також візуальну перевірку інтеграції логотипів, стилів і кольорової схеми. Завдяки цьому вдалося переконатися в стабільності поведінки клієнтської частини в умовах типової взаємодії користувачів.

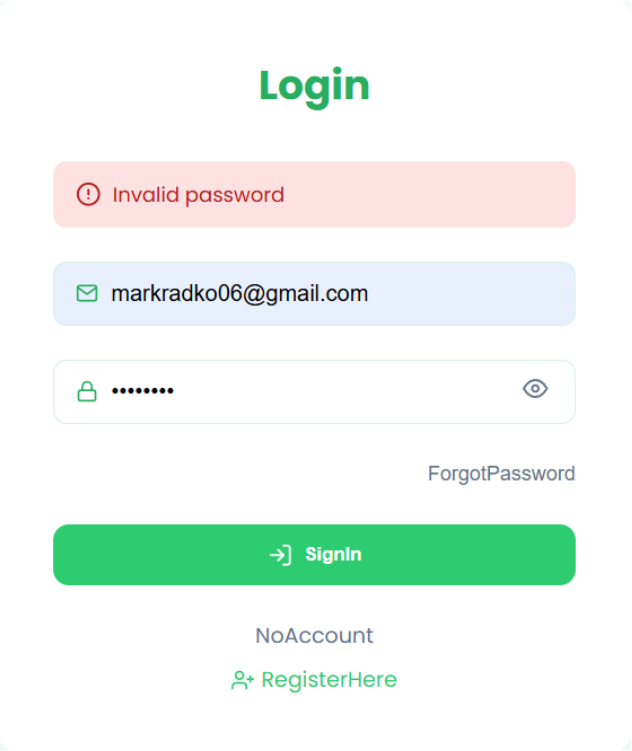


Рисунок 5.1 – Повідомлення про неправильні облікові дані (рисунок виконаний самостійно)

Узагальнені результати проведеного тестування відображено в додатку Л, де наведено таблиці з описом кожного перевіреного сценарію, введеними даними, очікуваними та фактичними результатами, а також висновками щодо успішності проходження тестів. Ця інформація дозволяє отримати об’єктивне уявлення про готовність веб-клієнта до використання в продуктивному середовищі та вказує на елементи, які потребують доопрацювання перед розгортанням системи.

## 5.4 Мануальне тестування мобільного застосунку

Мануальне тестування мобільного клієнта є невід’ємною частиною процесу перевірки якості програмного забезпечення, оскільки дозволяє безпосередньо оцінити роботу застосунку в реальних умовах використання на мобільних пристроях з різними характеристиками. У межах цього етапу було перевірено як функціональність основних сценаріїв, так і стабільність взаємодії інтерфейсу з користувачем у мобільному середовищі.

Застосунок, реалізований на Kotlin у середовищі Android Studio, було протестовано на відповідність вимогам щодо логіки авторизації, обробки помилок, взаємодії з сервером через JWT-токени, навігації між екранами та роботи з формами введення даних. Окрему увагу було приділено валідації введення користувачем інформації, реакції інтерфейсу на помилкові дії, а також коректному виведенню системних повідомлень.

У процесі тестування було виявлено помилку в частині відображення валідаційних повідомлень. Зокрема, при введенні некоректних даних (наприклад, неправильної адреси електронної пошти або порожнього поля пароля) обробка помилки відбувалася на рівні ViewModel, проте повідомлення не виводилось на екран користувача. Це призводило до ситуацій, коли застосунок не виконував дію (наприклад, не здійснював вхід), але й не пояснював користувачу причину. Така поведінка негативно впливає на користувацький досвід і була визнана критичною. Відповідна логіка виведення повідомлень була виправлена у подальших оновленнях.

Інші компоненти застосунку працювали стабільно. Було підтверджено коректність збереження токена доступу в SharedPreferences, виконання запитів через ViewModel, а також навігацію між екранами. Інтерфейс адаптується до різних розмірів екрану та підтримує зміну орієнтації. Реакція елементів керування (кнопок, форм, повідомлень) відповідає очікуваній, а продуктивність при взаємодії з сервером є стабільною.

Результати тестування зафіксовані в додатку М, де представлено таблиці з описом кожного сценарію, тестових даних, очікуваної та фактичної поведінки мобільного застосунку. Таблиці також містять висновки щодо проходження тестів і позначення функціоналу, який потребував доопрацювання. Надана звітність дозволяє оцінити готовність мобільного клієнта до практичного використання та забезпечує основу для подальшого розвитку застосунку з урахуванням отриманих результатів.

# ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було створено програмну систему, що забезпечує ефективне управління мережею паркінгів. До складу цієї системи входять серверна частина, веб-інтерфейс адміністратора, IoT-пристрої для контролю доступу та фіксації зайнятих місць, а також мобільний застосунок для зручної взаємодії користувачів із системою. Усі модулі інтегровані в єдину інформаційну структуру, що дозволяє забезпечити безперервну синхронізацію та обмін даними між ними.

Для уточнення вимог до майбутнього продукту та визначення очікувань зацікавлених сторін було створено документ Vision and Scope, на основі якого сформовано перелік функціональних можливостей системи. Подальше моделювання з використанням UML-діаграм (прецедентів, пакетів, активностей, станів і взаємодії) дало змогу детально описати поведінку системи та взаємозв’язки між її основними складовими. У результаті було запропоновано архітектуру, здатну підтримувати масштабування, тестування та подальше вдосконалення кожного з модулів.

Розробка програмного забезпечення відбувалася з використанням сучасних технологій, таких як Node.js, Express, PostgreSQL, React, Arduino з мікроконтролером ESP32, а також Kotlin для реалізації мобільного застосунку. Усі компоненти були протестовані в умовах моделювання реальної роботи системи.

Усі компоненти розробленої системи пройшли етап комплексного тестування. Було виконано юніт-тестування бізнес-логіки серверної частини, що дозволило перевірити правильність обробки фінансових операцій, управління заборгованістю та дотримання бізнес-вимог (див. Додаток Ж, таблиця Ж.1). Тестування REST API за допомогою Postman підтвердило стабільність роботи серверних ендпоінтів, правильність відповідей та обробку помилок (див. Додаток З, таблиця З.1). Мануальне тестування веб-клієнта виявило незначні помилки у валідації форм, які були виправлені на етапі завершення розробки (див. Додаток И). Окрему увагу було приділено мобільному застосунку, у якому усунуто проблеми з відображенням валідаційних повідомлень. Загалом проведені

тестування підтвердили відповідність реалізованої системи функціональним вимогам та її готовність до практичного використання в умовах реального паркінгу.

Побудована програмна система може бути використана у муніципальних чи приватних паркінгах для автоматизованого обліку транспортних засобів, бронювання місць, контролю часу перебування, управління доступом та здійснення оплати. Додатково реалізований IoT-модуль виконує функції автоматичного розпізнавання стану паркомісць, що сприяє зменшенню навантаження на персонал та підвищенню точності роботи системи.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ParkMobile URL: https://parkmobile.io (дата звернення: 06.05.2025).

2. EasyPark URL: https://www.easyparkgroup.com (дата звернення: 06.05.2025).

3. Find your parking with Parclick URL: https://parclick.com (дата звернення: 02.05.2025).

4. JustPark: Transform the Parking Experience. URL: https://www.justpark.com (дата звернення: 03.05.2025).

5. Nevliudov, Igor, Oleksandr Tsymbal, Artem Bronnikov, і Olexandr Mordyk. 2020. «ІНТЕРНЕТ РЕЧЕЙ ДЛЯ РОБОТОТЕХНІЧНИХ ПРОЄКТІВ». СУЧАСНИЙ СТАН НАУКОВИХ ДОСЛІДЖЕНЬ ТА ТЕХНОЛОГІЙ В ПРОМИСЛОВОСТІ, вип. 3 (13) (Вересень):58-64. https://doi.org/10.30837/ITSSI.2020.13.058.

6. Douglass J. Beginning PostgreSQL: From Novice to Professional. – Berkeley: Apress, 2021. – 500 p.

7. Мартін Р. Чистий код: створення і рефакторинг за допомогою AGILE. – ФАБУЛА, 2019. – 416 с.

8. Martin Fowler. Refactoring. Improving the Design of Existing Code– Addison-Wesley Professional, 1999. – 464 p.

9. Jemerov D., Isakova S. Kotlin Programming: The Big Nerd Ranch Guide. – Boston: Big Nerd Ranch Guides, 2018. – 480 p.

10. Patterson J. Android Programming: The Big Nerd Ranch Guide. – Boston: Big Nerd Ranch Guides, 2021. – 688 p.

11. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. – Sebastopol: O'Reilly Media, 2020. – 350 p.

12. Cantelon M., Harter M., Holowaychuk T., Rajlich N. Node.js in Action. – Shelter Island: Manning Publications, 2017. – 326 p.

13. Duckett J. HTML and CSS: Design and Build Websites. – Indianapolis: Wiley, 2011. – 512 p.

14. GitHub - NureRadkoMark/ 2025-B-PI-PZPI-21-10_Radko_M_M. *GitHub*. URL: https://github.com/NureRadkoMark/2025-B-PI-PZPI-21-10_Radko_M_M (дата звернення: 06.06.2025).