Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Навчально-науковий
інститут електроенергетики
(навчально-науковий інститут)
Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра

Здобувача вищої освіти Пасічної Анастасії Романівни
(ПІБ)
академічної групи 123м-23-1
(шифр)
спеціальності 123 Комп'ютерна інженерія
(код і назва спеціальності)
за освітньо-професійною програмою «Комп'ютерна інженерія»
(офіційна назва)

на тему «Обґрунтування структури інтелектуальної комп'ютерної системи паркінгу ЖК MARSHALL з урахуванням телеграм-бота»
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Цвіркун Л.І.			
розділів:				
синтез системи	доц. Ткаченко С.М.			
розроблення програмного забезпечення	доц. Бешта Л.В..			
Рецензент				
Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)
_____ В.В. Гнатушенко
(підпис) (ініціали, прізвище)

«_____» _____ 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня магістра
(бакалавра, магістра)

здобувача вищої освіти Пасічної А.Р. **академічної групи** 123м-23-1
(прізвище та ініціали) (шифр)

спеціальності 123 Комп'ютерна інженерія

за освітньою-професійною програмою «Комп'ютерна інженерія»
(офіційна назва)

на тему «Обґрунтування структури інтелектуальної комп'ютерної системи паркінгу ЖК MARSHALL з урахуванням телеграм-бота»,
затверджену наказом ректора НТУ «Дніпровська політехніка» від 17 жовтня 2024 р. №1388-с

Розділ	Зміст	Термін виконання
Стан питання та постановка завдання	На основі матеріалів практик, інших науково-технічних джерел сформулювати наукове завдання, конкретизувати предмет та мету досліджень	11.10.2024
Теоретичний	Обґрунтувати теоретичну базу автоматизованої системи управління паркінгом у ЖК MARSHALL з урахуванням інтеграції телеграм-бота.	25.10.2024
Синтез системи	Розробка комп'ютерної системи управління паркінгом у ЖК MARSHALL з інтеграцією телеграм-бота.	15.11.2024
Розроблення програмного забезпечення	Розробка програмного забезпечення телеграм-бота для управління паркінгом у системі ЖК MARSHALL.	29.11.2024
Експериментальний розділ	Проведення і обробка результатів експериментів з тестування телеграм-бота для управління паркінгом системи ЖК MARSHALL.	06.12.2024

Завдання видано _____ проф. Л. І. Цвіркун
(підпис керівника) (ініціали, прізвище)

Дата видачі 06 вересня 2024 р.

Дата подання до екзаменаційної комісії 10.12.2024 р.

Прийнято до виконання _____ Пасічна А.Р.
(підпис здобувача вищої освіти) (ініціали, прізвище)

РЕФЕРАТ

Пояснювальна записка 109 с., 35 рис., 1 дод., 4 табл., 20 джерел

TELEGRAM, СИСТЕМА ПАРКУВАННЯ, PYTHON, ТЕЛЕГРАМ-БОТ

Об’єкт розробки: інтелектуальна комп’ютерна система паркування в ЖК MARSHALL, з акцентом на автоматизацію її процесів.

Мета роботи: обґрунтування структури інтелектуальної комп’ютерної системи паркінгу ЖК MARSHALL з урахуванням впровадження телеграм-бота для автоматизації процесів контролю за паркомісцями, збору статистики та підняття шлагбауму.

У пояснювальній записці представлено аналіз систем паркування та можливості інтеграції мобільних додатків, зокрема використання телеграм-бота. На основі отриманих даних було визначено завдання для подальшого дослідження.

У теоретичному розділі проведено аналіз сучасних автоматизованих паркувальних рішень, виявлено їх ключові недоліки та аргументовано доцільність використання IoT-технологій у поєднанні з телеграм-ботом.

У розділі «Синтезу системи» сформульовані технічні вимоги до створення системи, враховано особливості функціонування паркінгу у житлових комплексах, такі як обмеженість простору, необхідність забезпечення високого рівня безпеки та зручності користувачів. Розроблено функціональну структуру системи, визначено апаратні та програмні компоненти, що забезпечують її ефективну роботу.

У розділі «Розробка програмного забезпечення» створено телеграм-бот на Python із використанням Telegram API. Бот забезпечує бронювання місць, контроль доступу та збір статистики, із детальним описом взаємодії компонентів і їх функцій.

В експериментальному розділі протестовано систему, підтверджено її ефективність, зручність і надійність, а також зменшення часу на пошук паркомісця.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

IoT – Internet of Things

BLE – Bluetooth Low Energy

LoRaWAN – Long Range Wide Area Network

API – Application Programming Interface

ЖК – Житловий Комплекс

ВСТУП

Сучасні житлові комплекси (ЖК) з кожним роком стають все більш інтегрованими з різноманітними технологічними рішеннями, що підвищують комфорт і безпеку їх мешканців. Інноваційні системи управління, автоматизація побутових процесів та впровадження цифрових технологій дозволяють створювати більш зручне та безпечне середовище для проживання. Одним з основних напрямків такого розвитку є впровадження рішень для організації та управління паркінгами, оскільки ця сфера безпосередньо впливає на повсякденне життя мешканців.

Однією з актуальних проблем, з якою стикаються мешканці багатоквартирних будинків, є організація паркувальних місць. Зростання кількості автомобілів та обмежений простір для паркування створюють значні труднощі. Невідповідність кількості місць потребам мешканців, незручний процес бронювання та відсутність ефективної системи управління паркінгом часто призводять до конфліктів і незадоволеності. Водії витрачають час на пошуки вільного місця, виникають суперечки через неправомірне використання місць, а відсутність централізованого контролю ускладнює адміністративне управління.

У цьому контексті розробка телеграм-боту для управління паркінгом у ЖК стає надзвичайно актуальною. Телеграм-бот може забезпечити зручний та інтуїтивно зрозумілий інтерфейс для бронювання паркувальних місць, автоматизувати процеси реєстрації та обліку автомобілів, а також надавати актуальну інформацію про наявність вільних місць у режимі реального часу. Такий підхід сприятиме підвищенню ефективності використання паркувальних ресурсів та покращенню взаємодії між мешканцями та адміністрацією ЖК.

Мета та завдання дослідження

Метою роботи є обґрунтування структури інтелектуальної комп’ютерної системи паркінгу ЖК MARSHALL з урахуванням

впровадження телеграм-бота для автоматизації процесів контролю за паркомісцями, збору статистики та підняття шлагбауму.

Основні завдання дослідження:

– охарактеризувати сучасний стан та можливості інтелектуальних систем паркування;

– проаналізувати вимоги до системи паркування;

– розробити концепцію телеграм бота, який забезпечить ефективне управління паркувальними місцями;

– оцінити можливість покращення управління інфраструктурою через автоматизацію процесів паркування та збір даних.

Об'єкт дослідження – інтелектуальна комп'ютерна система паркування в ЖК MARSHALL, з акцентом на автоматизацію її процесів.

Предмет дослідження – технологічні та організаційні аспекти на впровадження телеграм-бота в інтелектуальну систему паркування.

Методи дослідження:

– аналіз літературних джерел – для вивчення сучасних систем паркування;

– системний аналіз – для розробки архітектури системи та структури управління;

– експертні оцінки – для формування вимог до інтеграції телеграм-бота.

1 СТАН ПИТАННЯ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Стан питання

Паркувальна система є важливою складовою інфраструктури сучасних житлових комплексів. Вона не лише забезпечує мешканців можливістю зручного та безпечного паркування, але й сприяє ефективнішому використанню обмеженого простору в умовах зростання кількості автомобілів. У 21 столітті впровадження інтелектуальних технологій в сферу паркування стає не просто опцією, а необхідністю, що допомагає поліпшити якість життя у великих містах.

Для ЖК MARSHALL розробка паркувальної системи є важливим кроком у впровадженні інноваційних рішень для вирішення проблеми дефіциту паркувальних місць. Використання передових технологій, таких як сенсори для визначення зайнятості місць і мобільні додатки для управління паркуванням, сприятиме оптимізації простору та зменшенню часу на пошук вільного місця. Ці технології також підвищують рівень безпеки та зручності для мешканців.

Важливою частиною цієї інтелектуальної системи стане розробка спеціального телеграм-бота для взаємодії з користувачами. Він дозволить швидко отримувати інформацію про наявність вільних паркомісць, резервувати їх та отримувати сповіщення про статус паркування, що значно спростить користування системою. Інтеграція IoT-рішень разом із телеграм-ботом сприятиме створенню розумної інфраструктури, яка забезпечить високий рівень комфорту і ефективності в управлінні паркувальним простором.

1.2 Аналіз існуючих систем паркування

Аналіз існуючих систем паркування дозволяє виділити кілька основних типів рішень, що активно використовуються в сучасних міських умовах для забезпечення ефективного управління паркувальним простором. До найбільш поширених типів належать: традиційні механічні системи, автоматизовані паркінги та інтелектуальні (smart) паркувальні системи.

1.2.1 Традиційні механічні системи паркування

Традиційні механічні системи паркування функціонують на основі ручного управління паркувальними місцями, без використання сучасних цифрових технологій. Ці системи зазвичай використовуються в приватних будинках, офісах або невеликих комерційних об’єктах. Вони забезпечують базову організацію паркувального процесу з мінімальними витратами на інфраструктуру та експлуатацію. Однак основна проблема полягає у тому, що такі системи часто призводять до неефективного використання простору, що є критичним фактором в умовах щільної міської забудови та зростання кількості автомобілів [4].

У міських умовах традиційні паркувальні системи створюють додаткові незручності для користувачів, зокрема, через необхідність витрачати час на пошук вільного місця та ризик виникнення черг. Окрім цього, такі рішення не завжди є ефективними з точки зору використання землі, оскільки вимагають великих площ для розміщення парковок.

Хоча механічні системи мають низьку вартість і надійність, особливо у випадках відсутності електроживлення, їх недоліки стають більш очевидними у сучасних умовах, де є попит на ефективне використання обмеженого простору та зменшення навантаження на інфраструктуру.

1.2.2 Автоматизовані паркувальні системи

Автоматизовані паркінги – це інноваційне рішення, яке значно підвищує ефективність використання простору, особливо в густонаселених міських районах. Вони працюють за допомогою механізмів, які автоматично переміщують автомобілі на паркувальні місця, без необхідності водіїв маневрувати самостійно. Це дозволяє розміщувати машини у вертикальних або горизонтальних структурах, що суттєво зменшує площу, необхідну для паркування.

Основні переваги таких систем полягають у здатності значно збільшити кількість автомобілів, які можна розмістити на обмеженій площі, оскільки автомобілі паркуються щільніше і без потреби залишати простір для маневрів. Окрім цього, автоматизовані системи зменшують затори та час, який водії витрачають на пошук місця, що також покращує загальний користувацький досвід. Деякі рішення також інтегрують додаткові функції, як-от зарядні станції для електромобілів та можливості для зберігання речей [5].

Проте такі системи вимагають значних капіталовкладень на етапі впровадження, включаючи витрати на будівництво, монтаж та подальше обслуговування. Складність обслуговування може виникати через необхідність регулярного технічного догляду та заміну компонентів, таких як підйомні механізми чи автоматизовані ліфти[6].

1.2.3 Інтелектуальні системи паркування

Інтелектуальні паркувальні системи (smart parking) використовують різноманітні передові технології для оптимізації використання паркувальних місць та підвищення зручності для користувачів. Одним із основних елементів таких систем є сенсори та відеоспостереження, які автоматично визначають наявність вільних паркомісць. Дані з сенсорів передаються через мережі, такі як LoRaWAN або BLE (Bluetooth Low Energy), на центральні сервери, що дозволяє в реальному часі оновлювати інформацію про зайнятість місць.

Smart системи також інтегруються з мобільними додатками, які

дозволяють користувачам не лише швидко знаходити і бронювати місця, але й оплачувати паркування онлайн. Крім того, системи можуть використовувати технології розпізнавання номерних знаків для автоматизації в'їзду та виїзду з парковки. Це суттєво підвищує безпеку та комфорт для користувачів, скорочуючи час на пошук паркомісця та покращуючи загальне враження від процесу паркування.

Інтеграція smart parking з Інтернетом речей (IoT) також дозволяє вирішувати питання ефективного використання простору, зменшуючи потребу в додаткових паркомісцях. Такі системи збирають та аналізують дані про завантаженість паркувальних зон, що дозволяє прогнозувати потоки автомобілів і оптимізувати роботу паркувальних майданчиків. Ці рішення також можна поєднати з телеграм-ботами для ще більшого спрощення процесу для користувачів [7].

Розвиток smart parking рішень сприяє не тільки підвищенню ефективності використання простору, але й покращенню загальної міської інфраструктури, роблячи процес паркування простішим і швидшим для водіїв.

1.3 Проблеми сучасних паркувальних систем у житлових комплексах

Сучасні паркувальні системи в житлових комплексах стикаються з численними проблемами, що знижують їх ефективність та зручність для мешканців. Ключові виклики та проблеми, що впливають на функціонування таких систем це:

– обмеженість паркувального простору;

– нераціональне використання наявного простору;

– відсутність автоматизації процесів;

– проблеми з управлінням вільними місцями;

– питання безпеки та контролю доступу;

– високі витрати на модернізацію.

1.3.1 Обмеженість паркувального простору

Однією з найактуальніших проблем, з якою стикаються мешканці сучасних житлових комплексів, є обмеженість паркувальних місць. Зі збільшенням кількості приватних автомобілів, житлові комплекси, особливо в міських районах, не встигають адаптувати інфраструктуру під нові потреби. Проєкти багатьох житлових комплексів розробляються з урахуванням мінімальної кількості паркомісць, що базується на застарілих нормах або розрахунках, які не враховують тенденцію до збільшення кількості автомобілів на одну сім'ю. Така ситуація призводить до низки проблем, які негативно впливають як на зручність мешканців, так і на загальну інфраструктуру ЖК.

Основні причини обмеженості паркувального простору:

– неправильне планування інфраструктури. Багато ЖК проєктувалися з розрахунку на меншу кількість автомобілів, ніж їх фактично є зараз. Ця проблема особливо гостро відчувається в старих будинках та комплексах, де не передбачено достатньої кількості паркомісць;

– щільна забудова міських територій. У великих містах обмежений простір для забудови змушує девелоперів максимізувати кількість житлових площ, жертвуючи при цьому інфраструктурою для паркування. Будівництво підземних або багаторівневих паркінгів підвищує вартість проєкту, тому часто обираються дешевші варіанти;

– зростання кількості автомобілів. З роками автомобілізація населення зростає, і в багатьох родинах є не один, а два чи більше автомобілі. Це створює додатковий тиск на інфраструктуру житлових комплексів, яка не була розрахована на такий обсяг транспорту;

– недостатні норми проектування. Державні норми для проєктування житлових комплексів часто не враховують динаміку росту кількості автомобілів у міських районах. У результаті це призводить до того, що забудовники закладають лише мінімально необхідну кількість місць для паркування.

1.3.2 Нераціональне використання наявного простору

Навіть за умови достатньої кількості паркомісць у житловому комплексі, проблема їх нераціонального використання залишається серйозним викликом. Наявність паркувальних місць не завжди гарантує, що вони використовуються ефективно, що призводить до низки незручностей для мешканців та створює додаткове навантаження на інфраструктуру ЖК.

Основні причини нераціонального використання простору:

– відсутність системи бронювання місць. Багато житлових комплексів не мають систем, що дозволяли б мешканцям резервувати паркомісця заздалегідь. Через це мешканці змушені шукати вільні місця кожного разу при поверненні додому, що не лише забирає час, а й призводить до неефективного використання наявних ресурсів. Наприклад, місця можуть залишатися порожніми в пікові години, коли вони потрібні найбільше;

– неправильне планування розміщення місць. У багатьох ЖК розташування паркомісць не відповідає реальним потребам мешканців. Відсутність зонування для різних категорій користувачів, таких як мешканці, гості або обслуговуючий персонал, призводить до хаотичного використання простору. Це може включати залишення автомобілів на місцях, зарезервованих для інших цілей або певних категорій мешканців, що створює дискомфорт і конфлікти;

– нерегульований доступ до паркінгу. Якщо доступ до паркомісць не контролюється, мешканці та їхні гості можуть хаотично використовувати простір без урахування оптимальних варіантів паркування. Наприклад, автомобілі можуть бути припарковані занадто щільно або у неправильному порядку, що призводить до втрати можливих місць для паркування або заблокованості виїзду;

– відсутність системи контролю за зайнятістю. Без наявності систем, що відстежують статус зайнятості паркомісць у режимі реального часу, мешканці не мають змоги дізнатися, де є вільні місця. Це змушує їх об'їжджати територію в пошуках місця, що створює додаткове навантаження на трафік та

підвищує рівень стресу для водіїв. При цьому деякі місця можуть залишатися незайнятими через те, що вони не були вчасно помічені як вільні;

– індивідуальні місця для паркування. У деяких ЖК паркомісця закріплені за конкретними квартирами або мешканцями, що також може сприяти нераціональному використанню простору. Якщо мешканець не користується своїм місцем (наприклад, перебуває у відпустці), це місце залишається порожнім, хоча його могли б використовувати інші мешканці або гості. Така система не сприяє гнучкому та ефективному управлінню паркувальними ресурсами.

1.3.3 Відсутність автоматизації процесів

Однією з головних проблем сучасних паркувальних систем у житлових комплексах є відсутність автоматизації процесів, що значно ускладнює їх ефективне функціонування. Технологічна відсталість та використання застарілих механічних або ручних методів управління паркувальними місцями не дозволяють повною мірою забезпечити комфорт мешканців і призводять до втрати часу та ресурсів.

Основні проблеми, пов’язані з відсутністю автоматизації:

– труднощі в управлінні паркомісцями. У ЖК, де паркування контролюється вручну або за допомогою простих механічних систем, часто виникають ситуації, коли неможливо оперативно отримати інформацію про доступність вільних місць. Це ускладнює управління паркомісцями, особливо у великих комплексах із великою кількістю автомобілів. Мешканці змушені витрачати час на пошук місця для паркування, що знижує загальну зручність використання системи;

– відсутність інтегрованих технологій. Сучасні технології, такі як датчики зайнятості, системи розпізнавання номерних знаків або мобільні додатки для бронювання місць, не впроваджені у більшості ЖК. Це призводить до того, що інформація про зайнятість паркомісць не може бути оновлена в режимі реального часу, а мешканці не мають доступу до таких

зручностей, як онлайн-резервування чи дистанційний контроль за своїм паркувальним місцем;

– погіршення користувацького досвіду. Відсутність автоматизації процесів значно погіршує загальний досвід мешканців. Вони стикаються з низкою незручностей, таких як необхідність вручну шукати вільні місця, відсутність можливості швидко забронювати місце або незнання, чи є місце вільним. У випадках пікового навантаження, коли багато автомобілів намагаються припаркуватися одночасно, це призводить до заторів на території ЖК та підвищення рівня стресу серед мешканців;

– низька ефективність використання ресурсів. Без автоматизації важко ефективно керувати доступними ресурсами паркінгу. Паркомісця можуть залишатися порожніми через відсутність чіткої інформації або контроль за зайнятістю може бути надто обмеженим. У деяких випадках це призводить до нераціонального використання місць, що залишаються порожніми, тоді як мешканці витрачають час на їх пошук;

– недостатній рівень безпеки. Автоматизовані системи часто включають додаткові функції безпеки, такі як відеоспостереження, контроль доступу за допомогою розпізнавання номерних знаків або автоматизовані шлагбауми, які можуть суттєво підвищити рівень безпеки у паркувальних зонах. Відсутність таких технологій у ЖК робить систему паркування вразливою до неправомірного використання місць сторонніми особами, а також підвищує ризик крадіжок або пошкодження автомобілів.

Наслідки відсутності автоматизації:

– збільшення часу на паркування. Відсутність можливості автоматично знайти або забронювати місце збільшує час, який мешканці витрачають на пошук вільного місця. Це стає особливо проблемним в години пікового навантаження, коли багато водіїв намагаються одночасно припаркувати свої автомобілі;

– зниження загальної ефективності системи. Без автоматизованих рішень система паркування функціонує менш ефективно. Це стосується як

управління самими місцями, так і моніторингу їх використання. У результаті, паркомісця можуть залишатися незаповненими або використовуватися нераціонально;

– високі експлуатаційні витрати. Ручне управління паркувальною системою вимагає залучення додаткового персоналу для контролю за зайнятістю місць, що збільшує витрати на утримання такої системи. До того ж, технічне обслуговування механічних систем часто вимагає більше ресурсів порівняно з автоматизованими рішеннями;

– погіршення іміджу ЖК. Сучасні мешканці очікують від ЖК високого рівня комфорту, який включає сучасні технології. Відсутність автоматизації у паркувальних системах може негативно вплинути на репутацію житлового комплексу, оскільки потенційні покупці або орендарі нерухомості можуть вважати його застарілим або незручним для проживання.

1.4 Шляхи вирішення проблем сучасних паркувальних систем

Для вирішення проблем, пов'язаних із паркуванням у житлових комплексах (ЖК), необхідний комплексний підхід, що поєднує впровадження сучасних технологій та ефективні організаційні рішення. Важливо створити систему, яка забезпечить зручність для мешканців, безпеку та оптимальне використання обмеженого простору. У цьому розділі розглянемо ключові шляхи вирішення проблем паркувальних систем.

1.4.1 Впровадження інтелектуальних IoT-рішень

Інтернет речей (IoT) відкриває широкі можливості для автоматизації та оптимізації паркувальних процесів. Використання IoT дозволяє інтегрувати різні сенсори та системи моніторингу, що робить паркування більш ефективним та зручним для мешканців.

Встановлення сенсорів на кожному паркомісці дозволяє в режимі реального часу відстежувати, які місця вільні, а які зайняті. Це значно полегшує мешканцям процес пошуку місця, скорочуючи час і зменшуючи

транспортні затори в ЖК. Дані з датчиків можуть відображатися на екранах при в'їзді до паркінгу або в мобільних додатках.

Інтеграція з системами розпізнавання номерних знаків. Така система дозволяє автоматизувати доступ до паркінгу, що підвищує рівень безпеки та виключає можливість неправомірного використання паркомісць. Коли автомобіль мешканця під’їжджає до в’їзду, система розпізнає номер та автоматично відкриває шлагбаум.

1.4.2 Використання мобільних додатків для управління паркінгом

Мобільні додатки є одним з ключових інструментів для зручного управління паркувальними місцями в житлових комплексах. Вони надають мешканцям можливість бронювати місця, відстежувати їх зайнятість та здійснювати оплату за паркування у зручному форматі.

Завдяки мобільним додаткам мешканці можуть заздалегідь резервувати паркомісця, що гарантує наявність вільного місця до їх приїзду. Це усуває проблему з пошуком місць, особливо в пікові години.

Мобільний додаток може сповіщати мешканців про наявність вільних місць, а також інформувати про можливі зміни, наприклад, коли закінчується термін резервування або коли виникають проблеми з паркуванням.

Додатки можуть інтегруватися з платіжними системами, дозволяючи мешканцям швидко і зручно сплачувати за паркування без використання готівки або фізичних паркоматів. Це також зменшує ризик затримок при виїзді з паркінгу та скорочує час на виконання транзакцій.

1.4.3 Автоматизація управління та контроль за використанням паркомісць

Автоматизація управління паркувальними місцями дозволяє більш ефективно розподіляти доступні ресурси та мінімізувати ризик нераціонального використання простору.

Використання інтелектуальних систем дозволяє гнучко керувати розподілом паркомісць, автоматично визначаючи їх доступність залежно від часу дня або завантаженості. Наприклад, місця для гостей можуть бути доступними у вечірній час, коли менше мешканців користуються паркувальними місцями.

Автоматизовані системи контролю доступу можуть обмежувати доступ до паркінгу тільки для мешканців, що підвищує безпеку і виключає можливість сторонніх осіб використовувати паркомісця. Впровадження систем відеоспостереження та розпізнавання номерних знаків також сприяє підвищенню рівня безпеки [8][9].

1.5 Постановка завдання дослідження

Метою цього дипломного проєкту є обґрунтування структури інтелектуальної комп’ютерної системи паркінгу ЖК MARSHALL з урахуванням телеграм-бота.

Для реалізації цієї мети потрібно виконати такі завдання:

– проаналізувати наявну IoT-систему паркування та визначити, які функції можуть бути інтегровані з телеграм-ботом.

– дослідити кращі практики розробки ботів для управління паркувальними системами та взаємодії з користувачами.

– розробити функціональну структуру телеграм-бота, яка забезпечить зручний доступ до таких функцій, як бронювання місць, перевірка статусу зайнятості паркомісць та отримання сповіщень.

– створити програмне забезпечення для бота на базі мови Python із використанням Telegram API та інтеграції з IoT-системою паркування.

– провести тестування бота з реальними користувачами для перевірки ефективності його роботи та виявлення можливих проблем з використанням.

– оформити результати роботи у вигляді дипломного проєкту.

Автоматизація процесу паркування через телеграм-бот поліпшить зручність та швидкість взаємодії, а також зменшить навантаження на фізичний контроль за паркінгом.

2 ТЕОРЕТИЧНИЙ РОЗДІЛ

2.1 Загальна характеристика комп’ютерної системи інтелектуального паркінгу ЖК MARSHALL

Кіберфізична система паркування в житловому комплексі MARSHALL створена для моніторингу та оптимізації паркінгової інфраструктури, зокрема для управління транспортними потоками і забезпечення комфортного паркування для мешканців. Головна мета системи полягає в автоматизації управління паркувальними місцями, що дає змогу підвищити ефективність використання доступного простору, скоротити час на пошук вільних місць і поліпшити зручність для користувачів.

Ключовим аспектом системи є інтеграція різних технологій для забезпечення повноцінного функціонування інфраструктури. Камери відеоспостереження забезпечують контроль за зоною паркінгу, що сприяє підвищенню рівня безпеки. Датчики температури та вологості і датчики CO_2 стежать за мікрокліматом у паркінгу, що дозволяє підтримувати оптимальні умови для автомобілів та запобігати утворенню конденсату.

Крім того, кіберфізична система паркінгу управляє вентиляцією та контролем за рівнем забруднення повітря у паркінговій зоні, що є особливо важливим для підземних або закритих паркінгів, де існує ризик накопичення небезпечних газів. У разі підвищення рівня CO_2 система автоматично вмикає вентиляційні системи для зниження концентрації шкідливих речовин.

Загалом, комп'ютерна система інтелектуального паркінгу ЖК MARSHALL забезпечує синхронізацію всіх елементів, що гарантує безпеку, ефективність та зручність для користувачів, а також знижує експлуатаційні витрати на обслуговування паркінгу.

2.2 Структура об’єкту дослідження

Житловий комплекс MARSHALL розташований у місті Дніпро за адресою: вулиця Набережна Перемоги, 128, Дніпропетровська область (див. рисунок 2.1).

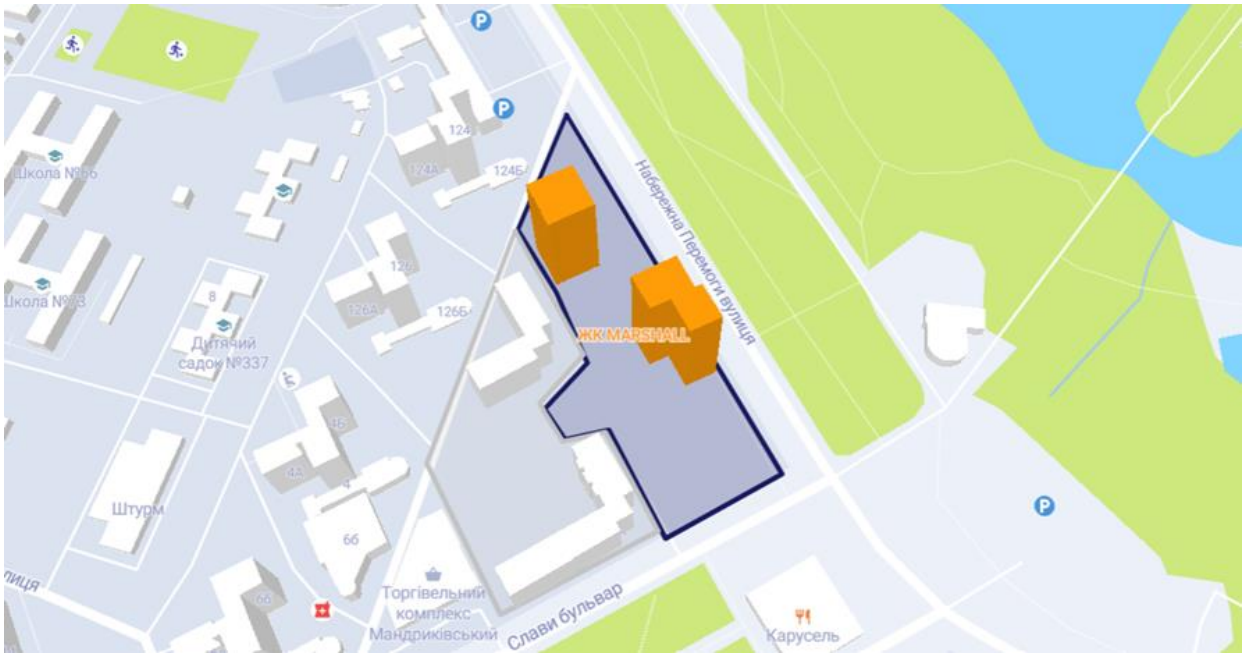

Рисунок 2.1 – Місцезнаходження ЖК MARSHALL на мапі наданою ріелтором

Головний офіс забудовника компанії Daytona Development Company знаходиться у Дніпрі за адресою: проспект Дмитра Яворницького, 22, офіс 705, що також розташований у Дніпропетровській області (див. рисунок 2.2). [10]

Рисунок 2.2 – Топологічна схема розташування головного офісу забудовника

Забудовник надає наступні послуги для обслуговування своїх проєктів:

– продаж: відділ продажу відповідає за реалізацію нерухомості;

– реклама: рекламний відділ займається просуванням нових проєктів;

– обслуговування комплексів: відділ обслуговування збирає інформацію про потреби клієнтів, організовує технічне обслуговування, ремонт, забезпечення чистоти та безпеки. Його мета – забезпечення комфорту для мешканців та бізнес-клієнтів.

Детальна структура управління Daytona Development Company, зокрема ЖК MARSHALL l, представлена на рисунку 2.3.

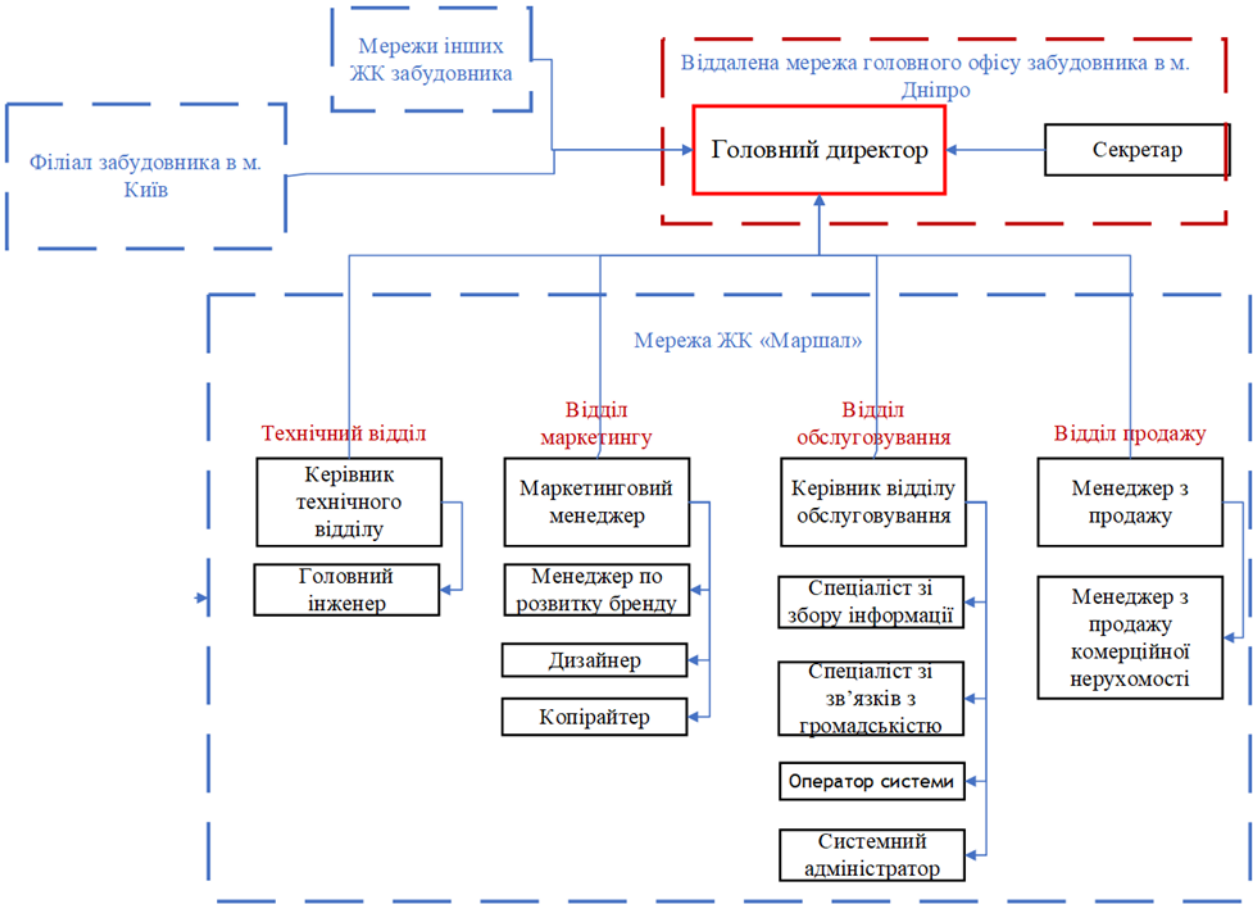

Рисунок 2.3 – Структура управління Daytona Development Company

Нижче показано план приміщень головного офісу забудовника та ЖК MARSHALL.

Рисунок 2.4 ілюструє офіс забудовника в ТЦ Atrium.

Рисунок 2.4 – Приміщення офісу на плані будівлі ТЦ Atrium

Площа головного офісу забудовника становить 60 м², у ньому розташовані робочі місця директора та його секретаря.

План першого та другого поверхів ЖК MARSHALL, де знаходяться технічний відділ, відділи продажу, маркетингу, обслуговування, а також паркінг і приміщення для оренди магазинів, взято з офіційного сайту ЖК (див. рисунок. 2.5-2.6).

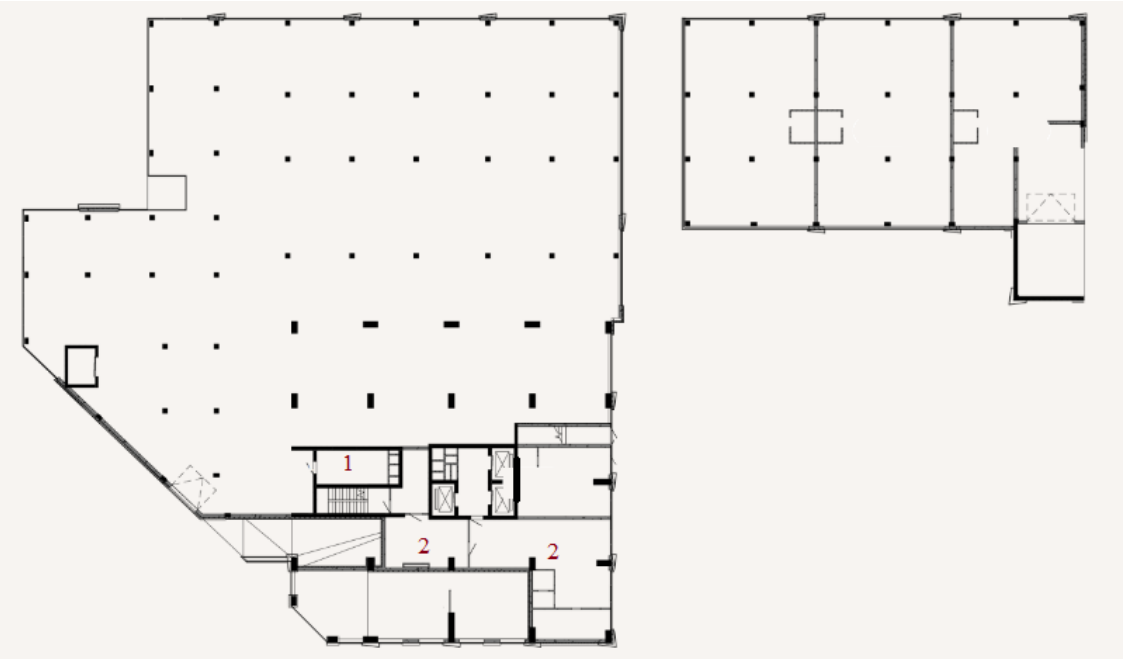

Рисунок 2.5 – План першого поверху: 1 серверна – 24,3 м²; 2 відділ продажу – 33,6 м² та 78,7 м²

Рисунок 2.6 – План другого поверху: відділ маркетингу – 153,79 м² (прим. 2-2); відділ обслуговування – 66,94 м² (прим. 2-7); технічний відділ – 122,19 м² (прим. 2-1)

2.3 Математична модель мережі ЖК MARSHALL як системи масового обслуговування замкнутого типу

У процесі дослідження структурної схеми комп'ютерної мережі ЖК MARSHALL, було здійснено детальний аналіз і розроблено математичну модель. Ця модель описує структуру комп'ютерної мережі як замкнуту систему масового обслуговування. Результати дослідження представлені на рисунку 2.7.

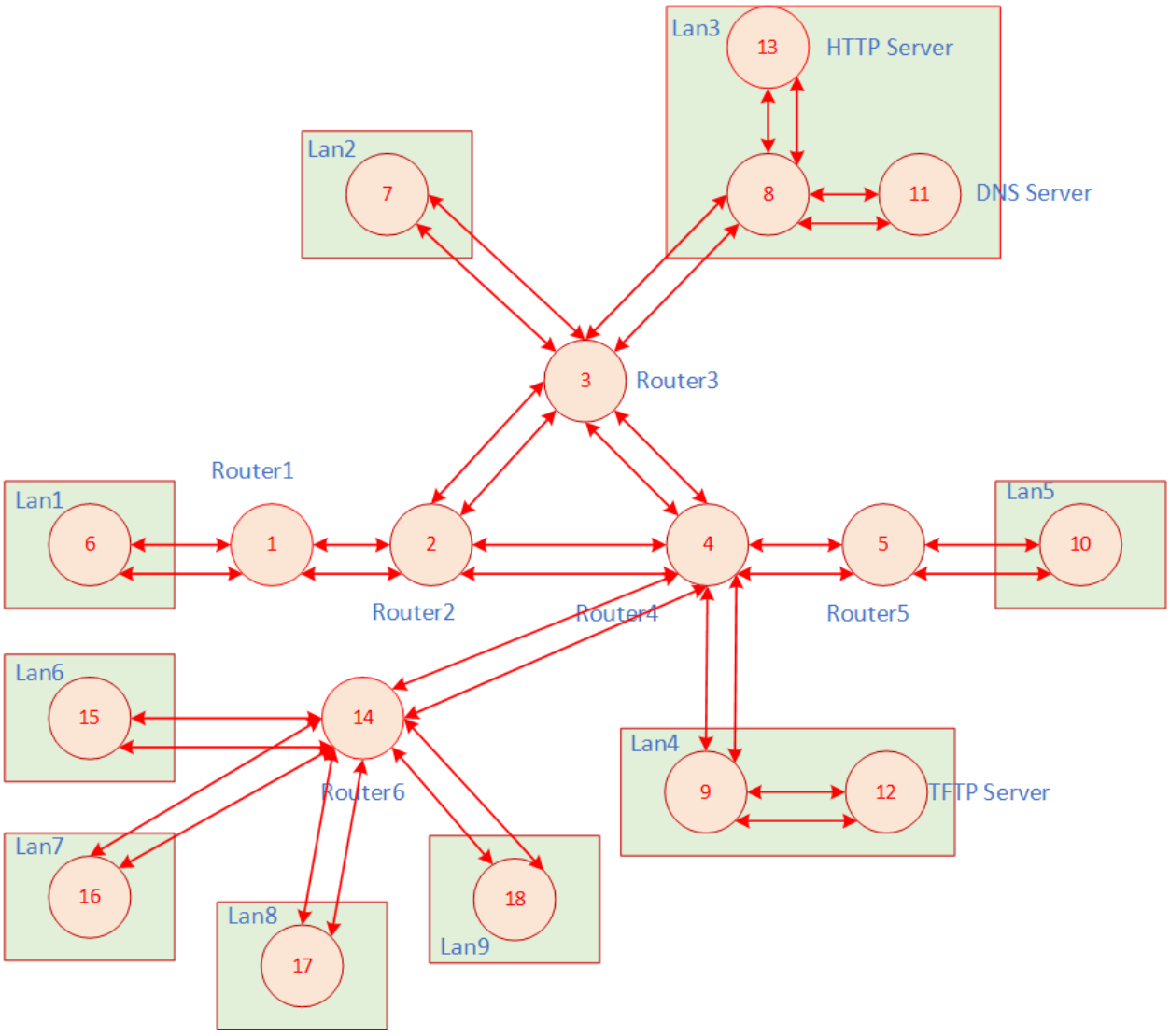

Рисунок 2.7 – Структура математичної моделі комп’ютерної мережі ЖК MARSHALL

Структура математичної моделі комп’ютерної мережі ЖК MARSHALL складається з різноманітних елементів, які забезпечують її функціональність і ефективність. Ключовими компонентами цієї мережі є комутатори, маршрутизатори та їх взаємодія, що визначається ймовірностями передачі даних між вузлами.

Комутатори (вузли 6, 7, 8, 9, 10, 15, 16, 17, 18). Ці вузли відповідають за обслуговування локальних мереж. Зокрема, вузли 6, 7, 8 та 9 виконують роль підмереж, які обслуговують паркувальні майданчики паркінгу ЖК MARSHALL. Вони забезпечують зв’язок між пристроями, розташованими у межах цих підмереж, та взаємодію з іншими елементами мережі.

Маршрутизатори (вузли 1, 2, 3, 4, 5, 14). Ці вузли відповідають за маршрутизацію та керування потоком даних у мережі. Вони приймають рішення про передачу пакетів даних між різними комутаторами та зовнішніми мережами, забезпечуючи надійний та оптимальний зв’язок.

Однією з важливих характеристик моделі є визначення ймовірностей передачі пакетів між вузлами. Кожен вузол розглядається як система масового обслуговування, де обробка даних залежить від взаємодії з іншими вузлами. Ці взаємодії описуються за допомогою таблиці ймовірностей (див. таблицю 2.1).

Таблиця 2.1 – Вірогідності зв’язку вузлів між собою

	R_1	R_2	R_3	R_4	R_5	R_6	LAN_1	LAN_2	LAN_3	LAN_4	LAN_5	LAN_6	LAN_7	LAN_8	LAN_9	DNS	TFTP	HTTP
R_1	0	0.5	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0
R_2	0.34	0	0.33	0.33	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R_3	0	0.25	0	0.25	0	0	0	0.25	0.25	0	0	0	0	0	0	0	0	0
R_4	0	0.2	0.2	0	0.2	0.2	0	0	0	0	0	0	0	0	0	0	0	0
R_5	0	0	0	0.5	0	0	0	0	0	0	0.5	0	0	0	0.5	0	0	0
R_6	0	0	0	0.2	0	0	0	0	0	0	0	0.2	0.2	0.2	0.2	0	0	0
LAN_1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LAN_2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LAN_3	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0	0.25
LAN_4	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0
LAN_5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
LAN_6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
LAN_7	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
LAN_8	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
LAN_9	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
DNS	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
TFTP	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
HTTP	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

2.4 Обґрунтування і вибір методів дослідження

Для успішної розробки телеграм-бота, інтегрованого в інтелектуальну IoT-систему паркування ЖК MARSHALL, важливо обрати правильні методи дослідження. Вони забезпечать належну оцінку існуючої кіберфізичної системи паркінгу та допоможуть визначити ефективність впровадження бота в її роботу.

2.4.1 Методи аналізу та синтезу

Методи аналізу та синтезу є фундаментальними в науково-дослідній роботі та забезпечують комплексний підхід до вивчення та розв'язання проблеми розробки телеграм-бота для інтелектуальної IoT-системи паркінгу.

Метод аналізу передбачає розділення складної системи або проблеми на окремі частини для їхнього детального вивчення. У контексті даного проєкту аналіз допоможе розглянути основні компоненти IoT-системи паркінгу ЖК MARSHALL, такі як:

– системи відеоспостереження;

– програмне забезпечення для управління та обробки даних.

Це дозволить краще зрозуміти, як працює кожна з цих підсистем окремо і як вони взаємодіють між собою. Важливо також провести аналіз можливих ризиків і недоліків існуючих рішень, зокрема, виявити потенційні загрози безпеці даних та стабільності роботи системи.

Метод синтезу є протилежним до методу аналізу і полягає в об'єднанні окремих елементів в єдину цілісну систему. Після того як всі компоненти IoT-системи паркінгу будуть проаналізовані окремо, синтез дозволить інтегрувати їх в єдину функціональну систему. У нашому випадку, це означає інтеграцію телеграм-бота з уже впровадженими рішеннями. Синтез допоможе створити взаємодію між сенсорами, системами відеоспостереження та телеграм -ботом для забезпечення ефективної роботи паркінгу.

Застосування методів аналізу та синтезу дозволяє комплексно підійти до розробки та вдосконалення програмного рішення, забезпечуючи як детальне вивчення окремих елементів, так і їх подальшу інтеграцію в єдину систему. Це підвищить ефективність функціонування IoT-системи паркінгу ЖК MARSHALL та забезпечить її зручність для користувачів.

2.4.2 Порівняльний аналіз існуючих систем інтелектуального паркінгу

У сучасних житлових комплексах інтелектуальні системи паркування (IoT-based Smart Parking Systems) стають важливим елементом для оптимізації управління паркувальними місцями, зниження заторів та покращення якості обслуговування мешканців. Проведення порівняльного аналізу існуючих рішень допомагає визначити найефективніші підходи для реалізації таких систем у ЖК.

Характеристики сучасних систем інтелектуального паркінгу:

– Parklio (ЄС). Система Parklio використовує IoT-технології для автоматизованого управління паркомісцями з функцією бронювання за допомогою мобільного додатку. Однією з ключових особливостей є

можливість інтеграції з системами контролю доступу та оплати паркування [11];

– Cisco Smart+Connected Parking (США). Рішення від Cisco базується на датчиках IoT, встановлених на паркомісцях, що дозволяють відслідковувати зайнятість місць в режимі реального часу. Система підтримує автоматизоване бронювання і використання мобільного додатка для оплати послуг паркування. Вона забезпечує високу масштабованість та інтеграцію з іншими системами розумного міста [12];

– Smart Parking Ltd (Австралія). Компанія Smart Parking надає рішення, які включають використання датчиків для моніторингу паркувальних місць, автоматизовану оплату через мобільні додатки та можливість бронювання місць наперед. Особливістю цієї системи є інтеграція з системами моніторингу трафіку та вбудовані інструменти аналітики [13];

– IoT-Base Parking System ЖК. Для житлових комплексів розумні системи паркінгу включають датчики для виявлення вільних місць, можливість бронювання через телеграм-бота та інтеграцію з системами контролю доступу. Система підтримує контроль за температурою і рівнем CO2 у закритих приміщеннях, що підвищує безпеку зберігання автомобілів та комфорт мешканців [14][15].

Порівняльні характеристики:

– Cisco та Smart Parking Ltd пропонують більш масштабовані рішення, орієнтовані на інтеграцію в загальні системи розумних міст. Водночас Parklio і системи для ЖК орієнтовані на більш локальні потреби з акцентом на зручність користування;

– усі аналізовані системи забезпечують інтеграцію з мобільними платформами, що дозволяє користувачам керувати паркуванням дистанційно;

– системи, такі як Cisco Smart+Connected Parking, мають інтеграцію з загальними системами розумного міста, включаючи управління трафіком та громадським транспортом.

Порівняльний аналіз показує, що системи інтелектуального паркінгу, які

орієнтовані на житлові комплекси, можуть відрізнятися своїми вимогами та масштабами впровадження від систем для великих міст. Вибір технології залежить від специфіки об'єкта: якщо мета полягає в оптимізації локального паркінгу, рішення на базі IoT з інтеграцією в мобільні додатки або телеграм-бот є найбільш ефективними.

2.4.3 Загальна модель системи

Система паркування в житловому комплексі MARSHALL побудована на основі кіберфізичних компонентів та технологій Інтернету речей. Її головне завдання – забезпечення автоматизованого контролю паркувальних місць, моніторинг умов зберігання транспортних засобів, а також управління процесом в’їзду та виїзду автомобілів.

Основні елементи системи:

– датчики руху, які встановлені в різних точках паркінгу та автоматично вмикають освітлення, коли фіксують наближення автомобіля або пішохода. Це дозволяє економити електроенергію, коли в зоні паркінгу немає активності;

– датчики клімат-контролю, які відстежують показники вологості, температури та рівня CO2 у приміщенні паркінгу, забезпечуючи комфортні умови для зберігання транспортних засобів і підтримуючи необхідні показники для запобігання утворенню конденсату, що може шкодити автомобілям;

– відеоспостереження, камери встановлені по периметру та всередині паркінгу для забезпечення безпеки транспортних засобів. Відеопотоки передаються на центральний сервер, де вони архівуються і можуть бути використані для аналізу інцидентів;

– IoT-сервер, всі дані з датчиків, камер та інших систем паркінгу надходять на сервер, де вони обробляються та зберігаються. Сервер здійснює збір статистики щодо заповнюваності паркінгу, аналізує частоту використання паркомісць та надає користувачам дані в реальному часі про доступні місця.

2.4.4 Інтеграція телеграм-бота в систему паркування ЖК MARSHALL

Для підвищення зручності використання паркінгу, планується впровадження суміжної системи з використанням телеграм-бота, який стане важливою частиною системи управління паркінгом. Бот буде взаємодіяти з сервером, використовуючи наступні функції:

– інформування про вільні місця, бот буде відображати в реальному часі кількість доступних паркомісць і їх точне розташування;

– бронювання паркомісць, користувачі зможуть через бот здійснювати бронювання паркомісця, після чого система автоматично відкриватиме шлагбаум;

– контроль та збір статистики, бот забезпечить збір і аналіз статистичних даних про використання паркінгу для оптимізації його роботи;

– оповіщення про ціни та доступні місця, через бот мешканці ЖК отримуватимуть повідомлення про зміни у вартості паркування та інформацію про спеціальні пропозиції чи знижки.

Ця модель забезпечить максимальну автоматизацію процесів, що сприятиме ефективному використанню паркомісць та покращенню обслуговування мешканців ЖК MARSHALL.

3 СИНТЕЗ СИСТЕМИ ПАРКУВАННЯ

3.1 Цілі впровадження системи

Основна мета вдосконалення системи паркування в ЖК MARSHALL полягає у впровадженні ефективного управління паркомісцями та забезпеченні зручності й простоти користування для мешканців завдяки використанню телеграм-бота. Інтеграція цього бота спрямована на спрощення комунікації з користувачами, забезпечення ефективного контролю доступності паркомісць і автоматизацію основних процесів у системі. Модель системи передбачає виконання таких функцій:

– збір та аналіз даних;

– інтеграція телеграм-бота для взаємодії з користувачами;

– автоматизація доступу до паркінгу.

Загалом, впровадження цієї системи сприятиме покращенню управління паркінгом, забезпечить інтеграцію сучасних технологій та підвищить ефективність використання інфраструктури ЖК MARSHALL.

3.2 Формулювання технічних вимог до системи паркування

3.2.1 Вимоги до реалізації системи паркування

Для успішного впровадження телеграм-бота у систему управління паркінгом житлового комплексу MARSHALL необхідно дотримуватися наступних вимог:

– вибір апаратного та програмного забезпечення має гарантувати швидку обробку запитів користувачів із часом відповіді в межах 200-500 мілісекунд, включаючи затримки на сервері та в мережі, для забезпечення зручності користувачів. Складніші запити можуть оброблятися до 900 мс, але для стандартних операцій, таких як перевірка статусу паркомісць або надсилання повідомлень через телеграм-бота, до 200-300 мс;

– необхідно забезпечити безперебійне функціонування сервісу в режимі реального часу для підтримання стабільності роботи та високого рівня

обслуговування;

– телеграм-бот повинен мати інтуїтивний інтерфейс.

3.2.2 Вимоги до функцій виконуваних системою паркування

Інтеграція телеграм-бота у систему паркування житлового комплексу MARSHALL передбачає виконання низки функцій для забезпечення ефективного управління паркомісцями, зручності для користувачів та автоматизації процесів. Система повинна виконувати такі основні функції:

– система повинна забезпечувати оперативне оновлення даних про доступність паркомісць, що надається через телеграм-бот.;

– телеграм-бот має надавати функцію бронювання паркомісць через інтерфейс мобільного додатка;

– система повинна забезпечувати автоматичне відкриття та закриття шлагбаума для гостьового паркінгу після підтвердження оплати;

– система повинна автоматично збирати статистичні дані щодо використання гостьових паркомісць, включаючи час перебування та вартість паркування, для аналізу та оптимізації роботи гостьового паркінгу;

– для паркомісць, закріплених за мешканцями, система повинна автоматично фіксувати, чи заброньовано місце, для забезпечення належного контролю та управління резидентським паркінгом;

– система повинна дозволяти мешканцям залишати відгуки або звернення через телеграм-бот.

3.2.3 Вимоги до видів забезпечення

Для інтеграції телеграм-бота необхідно налаштувати сервери, що забезпечують обробку запитів від користувачів та зв'язок з іншими елементами системи. Основні показники для забезпечення безперервної роботи та надійності:

– час відгуку сервера: не більше 200 мс для швидкої обробки запитів на інформацію про наявність місць, їх вартість і стан оплати ;

– пропускна здатність інтернет-каналу мінімум 100 Мбіт/с.

Телеграм-бот повинен мати інтуїтивно зрозумілий інтерфейс, що дозволяє переглядати інформацію про вільні місця, здійснювати бронювання, оплату, а також отримувати доступ до паркінгу. Він також інтегрований із системою управління шлагбаумом для автоматичного відкриття після підтвердження оплати. Для цього потрібні наступні показники:

– час відкриття шлагбаума до 2 секунд після підтвердження оплати;

– обсяг сховища для даних мінімум 4 ТБ.

Система повинна мати стабільне і захищене інтернет-підключення для безперебійного доступу користувачів до бота з будь-якої локації:

– шифрування SSL/TLS для захисту даних користувачів;

– резервний інтернет-канал з аналогічною пропускною здатністю 100 Мбіт/с.

3.2.4 Вимоги до захисту інформації

Для забезпечення безпеки даних у системі з телеграм-ботом для управління паркомісцями важливо дотримуватися вимог із захисту інформації:

– усі дані, які передаються через телеграм-бот, повинні бути захищені шифруванням AES з ключем не менше 256 біт;

– всі користувачі повинні проходити процес аутентифікації через телеграм-бот перед доступом до функцій системи;

– всі операції мають фіксуватися в журналі, який зберігає дані не менше

1 року;

– для захисту від DDoS-атак рекомендується використання фаєрволу на рівні мережі з потужністю обробки трафіку не менше 10 Гбіт/с.

3.2.5 Вимоги до ергономіки системи

При інтеграції телеграм-бота в систему паркування, необхідно врахувати вимоги до ергономіки для забезпечення зручності використання як для мешканців ЖК, так і для адміністрації. Основні вимоги до ергономіки включають:

– телеграм-бот повинен мати зрозумілий і мінімалістичний дизайн, який легко сприймається користувачами без необхідності додаткового навчання. Всі функції, такі як бронювання місць, оплата та відкриття шлагбауму, повинні бути доступні через кілька простих кроків;

– бот повинен реагувати на запити користувачів максимально швидко, а серверна частина системи – обробляти дані в режимі реального часу;

– телеграм-бот повинен бути однаково зручним для використання на різних платформах (смартфони, планшети, комп'ютери) і операційних системах;

– інформація, яку надає бот, повинна бути чіткою та конкретною. Для кожного кроку користувач має отримувати зручні підказки або пояснення, які допоможуть йому легко завершити операцію, наприклад, підтвердження успішної оплати або інструкції щодо відкриття шлагбауму;

– інтерфейс бота повинен мати можливість легко адаптуватися до змін вимог або додавання нових функцій.

3.2.6 Розробка схеми функціональної структури

Функціональна структура системи для автоматизованого управління паркуванням у ЖК MARSHALL передбачає інтеграцію з телеграм -ботом для збору статистики інформування про доступні місця, підняття шлагбаума та спрощення взаємодії з мешканцями. Нижче наведено ключові модулі системи.

1. Модуль "Інтерфейс користувача" (Телеграм-бот).

Цей модуль є основним засобом взаємодії користувачів із системою. Він надає мешканцям зручний інтерфейс для отримання інформації про доступні паркомісця, перегляду вартості та здійснення бронювання. Телеграм-бот слугує посередником між користувачем і сервером, передаючи запити на бронювання або оплату та повертаючи актуальні дані, такі як надання актуальної інформації про тарифи або статус паркомісця. Користувачі отримують миттєві сповіщення про результат своїх дій, наприклад, підтвердження успішного бронювання, що робить взаємодію простою та ефективною.

2. Модуль "Обробка даних" (Сервер).

Сервер є центральним вузлом системи, відповідальним за обробку та зберігання всіх даних. Він приймає запити від телеграм-бота, аналізує їх і відповідає відповідно до поточного стану системи. Сервер зберігає базу даних із інформацією про паркомісця, включно з їхнім статусом (доступне/заброньоване/зайняте) та історією використання. Крім того, він перевіряє доступність місця для бронювання. На основі успішної обробки даних сервер передає команди до шлагбаума, наприклад, на відкриття або закриття, забезпечуючи фізичний доступ до паркінгу. Водночас сервер повертає результати обробки запитів до телеграм-бота, інформуючи користувача про поточний стан його запиту.

3. Модуль "Контроль доступу" (Шлагбаум).

Шлагбаум виконує функції фізичного доступу до паркінгу, реагуючи на команди, що надходять від сервера. Після завершення бронювання сервер надсилає команду для відкриття шлагбаума, забезпечуючи безперешкодний

в’їзд автомобіля. Після виконання цієї дії шлагбаум реєструє свій стан (відкрито чи закрито) і повертає цю інформацію на сервер. Це дозволяє зберігати повний лог подій для подальшого аналізу. Крім того, шлагбаум автоматично закривається після в’їзду або виїзду.

На рисунку 3.1 наведена функціональна схема автоматизованого управління паркуванням у ЖК MARSHALL за допомогою телеграм-бота.

Рисунок 3.1 – Функціональна схема автоматизованого управління паркуванням

3.3 Вибір та обґрунтування застосування апаратних засобів

3.3.1 Вибір та характеристика мережевого обладнання

Для забезпечення належного функціонування системи управління паркінгом з інтеграцією телеграм-бота було відібране мережеве обладнання, яке дозволяє організувати стабільне мережеве з'єднання, забезпечує живлення підключеного обладнання та безперебійну передачу даних. Підключення до Wi-Fi дозволяє користувачам зручно взаємодіяти з ботом та отримувати інформацію про паркувальні місця.

Специфікація обладнання наведена у таблиці 3.1.

Таблиця 3.1 – Специфікація обладнання для впровадження телеграм - бота

Позиція	Найменування і технічна характеристика	Тип, марка, позначення документного листа	Одиниця виміру	Кількість	Умовне позначення	Примітка
1	2	3	4	5		6
1	Керований комутатор PoE, 24 порти	Cisco SG350X-24P	Од.	4	SW	Забезпечує мережеве з'єднання та живлення для підключеного обладнання
2	Точка доступу Wi-Fi з підтримкою 802.11ac	Ubiquiti UniFi AP AC Pro	Од.	4	AP	Створює стабільне покриття Wi-Fi для користувачів системи
3	Інтернет-шлюз для безпечного з'єднання	MikroTik hEX S	Од.	2	GW	Захищає мережу від зовнішніх загроз і забезпечує оптимальний трафік
5	Кабель живлення 220В		м	300	PWR	

Продовження Таблиці 3.1

Позиція	Найменування і технічна характеристика	Тип, марка, позначення документного листа	Одиниця виміру	Кількість	Умовне позначення	Примітка
5	ДБЖ для мережевого обладнання з підтримкою до 1000VA	APC Back-UPS 1000VA	Од.	4	UPS1	Забезпечує безперебійне живлення для мережевих компонентів у випадку перебоїв електрики
6	Шлагбаум, довжина стріли до 6 м, інтеграція через RS-485	Nice Signo 6	Од.	4	SB	Автоматизований доступ
7	Фотоелементи безпеки	Nice BF	Од.	4	PE	
8	Індуктивна петля,	Proloop 2	Од.	4	LP	Автоматичне закриття шлагбаума
9	Проміжний контролер, RS-485, Ethernet, підтримка релейних виходів	ZKTeco C3-400	Од.	4	CTR	Управління шлагбаумами
10	Адаптер RS-485 → Ethernet	Waveshare RS485 to Ethernet,	Од.	4	AD	
11	Платіжний термінал	Ingenico Move/5000	Од	2	PT	
12	Кабель Ethernet Cat 6		м	900	ETH	

Кінець Таблиці 3.1

Позиція	Найменування і технічна характеристика	Тип, марка, позначення документного листа	Одиниця виміру	Кількість	Умовне позначення	Примітка
1	2	3	4	5		6
13	Розподільчий щит із 12 модулями для захисту мережі	ABB Mistral41	Од.	4	DPS	Для розміщення автоматичних вимикачів та живлення
14	Автоматичний вимикач на 16 А, 3-полюсний	Schneider Electric Acti9	Од.	16		Захист від перевантажень у мережі
15	Захисний вимикач диференціального струму, 40 А, 30 мА	ABB F204AC	Од.	4		Захист від витоку струму
16	Блок живлення на DIN-рейку, 24 В, 5 А	Mean Well HDR-60-24	Од.	4		Живлення компонентів у щитових

3.3.2 Вибір та характеристика серверного обладнання

Для реалізації системи автоматизованого управління паркуванням у ЖК MARSHALL з інтеграцію телеграм-бота та підтримки всіх мережевих компонентів обрано серверне обладнання з відповідними характеристиками. В таблиці 3.2 наведено специфікацію обраного обладнання та детальний опис кожного компонента.

Таблиця 3.2 – Специфікація серверного обладнання

Позиція	Найменування і технічна характеристика	Тип, марка, позначення документного листа	Одиниця виміру	Кількість	Умовне позначення	Примітка
1	2	3	4	5		6
1	Сервер з процесором Intel Xeon E-2276, 32 ГБ оперативної пам'яті, 4x1TB HDD RAID 10, ОС Ubuntu Server 20.04 LTS	Dell PowerEdge R340	Од.	1	SRV	Dell PowerEdge R340: забезпечує обробку даних для роботи Telegram-бота та інфраструктури мережі
2	Система зберігання даних, підтримка до 48 ТБ HDD, RAID 5	Synology RackStation RS1219+	Од.	1	NAS	Synology RS1219+: сховище для зберігання та архівації даних, що надходять від серверів
3	Джерело безперебійного живлення потужністю 2000 VA	APC Smart-UPS 2000VA	Од.	2	UPS2	Забезпечує захист серверів від збоїв електроенергії

3.4 Синтез структурної схеми системи за заданими показниками

Відповідно до визначених технічних вимог і показників, була розроблена структурна схема системи автоматизованого управління паркуванням у ЖК MARSHALL. Ця схема відображає взаємодію всіх основних компонентів системи, забезпечуючи їх інтеграцію в єдине середовище для ефективного функціонування. Структурна схема представлена на рисунку 3.2 і демонструє логічний взаємозв’язок між компонентами

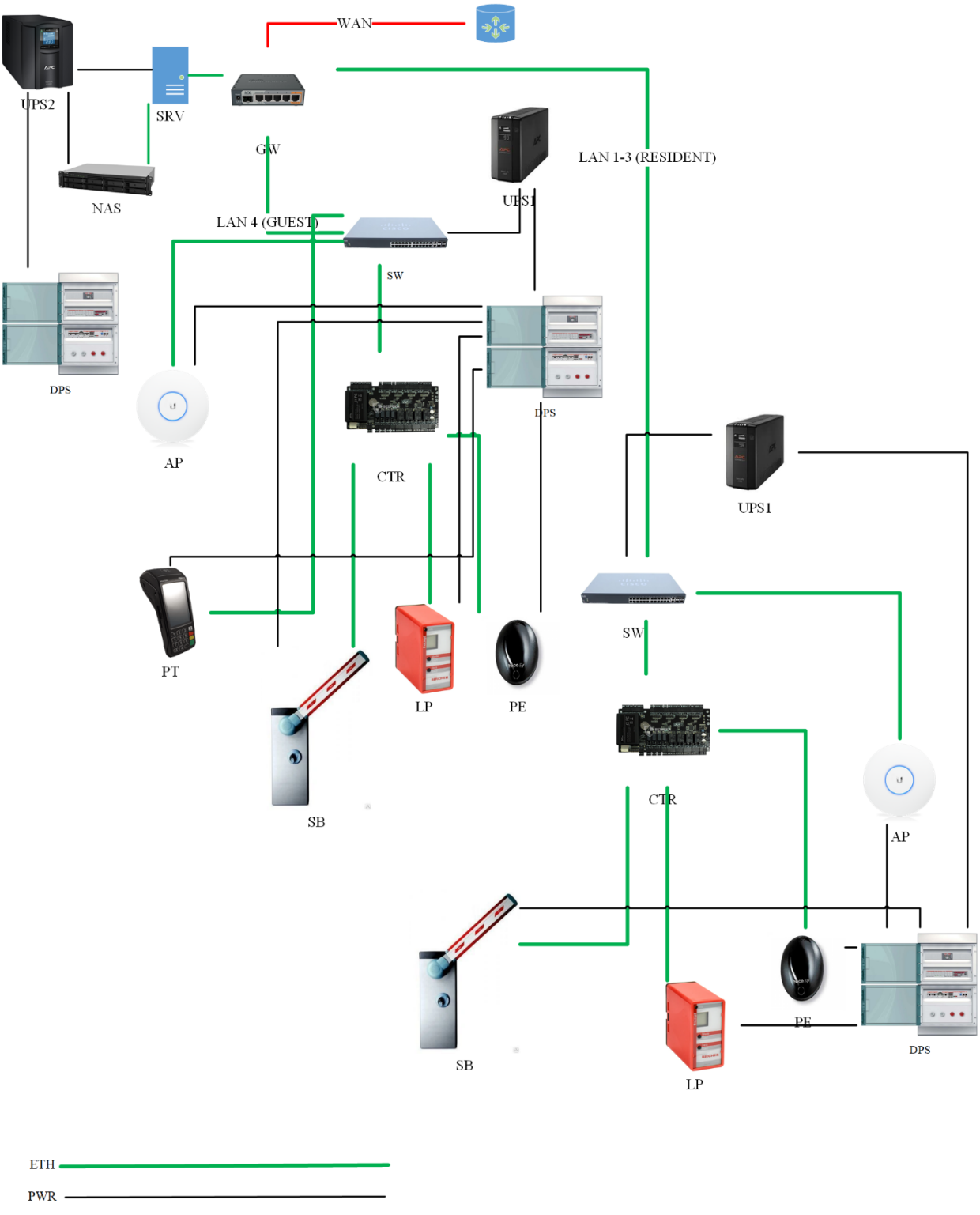

Рисунок 3.2 – Структурна схема системи

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ПАРКУВАННЯ

4.1 Призначення й область застосування програмного забезпечення

Розроблене програмне забезпечення (ПЗ) системи паркування, інтегроване з телеграм-ботом, призначене для автоматизації процесів керування паркуванням, бронювання місць та обробки фінансових операцій. Основна мета цього ПЗ – забезпечити мешканців житлового комплексу MARSHALL зручним інструментом для дистанційного управління паркувальними послугами та оптимізувати роботу паркінгу за допомогою інтелектуальних рішень.

Основні функціональні можливості ПЗ:

– надає користувачам легкий доступ до інформації про доступні паркомісця, можливість бронювання та автоматичного доступу через систему шлагбаумів;

– ПЗ забезпечує інтеграцію з системою контролю доступу для відкриття шлагбаумів;

– забезпечує адміністрацію ЖК засобами для керування паркувальними місцями, контролю доступу та отримання статистичних звітів про завантаженість паркінгу.

Система паркування застосовується на території житлового комплексу MARSHALL, що обслуговує мешканців, які потребують автоматизованої системи управління паркувальними місцями з можливістю бронювання, віддаленого доступу, а також захисту від несанкціонованого доступу.

4.2 Розробка принципових схем елементів і блоків

Процес взаємодії користувача з телеграм-ботом для управління паркуванням у ЖК MARSHALL побудований таким чином, щоб забезпечити максимально зручний і зрозумілий інтерфейс для виконання всіх необхідних дій: перегляд доступних місць, бронювання та автоматичне відкриття

шлагбаума. Кожна з функцій реалізується у вигляді послідовності етапів, які забезпечують інтеграцію між ботом, серверною інфраструктурою та апаратною частиною паркінгу.

4.2.1 Ініціація взаємодії з користувачем

Користувач починає спілкування з ботом, надсилаючи йому текстову команду або обираючи одну з інтерактивних кнопок. Наприклад:

– гість може відправити запит на перегляд вільних місць за допомогою команди /view_parking, щоб отримати інформацію про наявність паркомісць;

– мешканець може скористатися командою /open_gate, щоб відкрити шлагбаум для в'їзду.

Телеграм-бот обробляє ці команди та визначає, до якого функціонального блоку системи їх потрібно скерувати.

4.2.2 Передача команд на сервер

Бот передає сформований запит серверу через API-запит. Цей запит містить:

– тип дії (наприклад, перегляд паркомісць, бронювання);

– унікальний ідентифікатор користувача;

– додаткові параметри, номер автомобіля, час очікуваного перебування тощо.

Сервер отримує цей запит, виконує його валідацію та переходить до виконання завдання.

4.2.3 Обробка сервером

Сервер виконує кілька етапів обробки залежно від типу запиту:

– для перегляду вільних місць, сервер звертається до бази даних parking.db, перевіряє статус усіх паркомісць і формує відповідь із переліком вільних місць;

– для бронювання місця, сервер перевіряє, чи запитуване місце доступне.

У разі успіху записує статус "заброньовано" в базу даних, додає номер автомобіля користувача і час початку бронювання;

– для відкриття шлагбаума, сервер перевіряє права доступу користувача (гостьовий чи мешканець). У разі успішної перевірки надсилає команду пристрою управління шлагбаумом.

4.2.4 Повернення результату ботом

Після виконання завдання сервер передає результат телеграм-боту. Це може бути:

– список доступних місць із зазначенням їхньої зони розташування;

– підтвердження бронювання із зазначенням номера місця та тривалості;

– повідомлення про успішну оплату;

– інформація про успішне відкриття шлагбаума;

– бот форматує ці дані у зручному вигляді та надсилає користувачеві.

4.2.5 Логування операцій

Кожна взаємодія між користувачем і системою фіксується у журналі серверної частини. Лог включає:

– тип операції (перегляд, бронювання, оплата, відкриття);

– час виконання;

– ідентифікатор користувача та IP-адресу (для додаткового моніторингу).

Ці дані дозволяють адміністрації аналізувати ефективність роботи системи, відстежувати її завантаженість і вчасно реагувати на можливі технічні проблеми.

4.3 Опис розробленої програми

Телеграм-бот викликається за допомогою команди /start, після чого користувач потрапляє до головного меню (див. рисунок 4.1). Меню дозволяє вибрати потрібну функцію, таку як обрання ролі, мешканець чи гість та надає можливість перейти за посиланням на офіційний сайт обслуговуючої компанії та забудовника. Інтерфейс простий і інтуїтивно зрозумілий, що робить роботу з ботом зручною для всіх категорій користувачів.

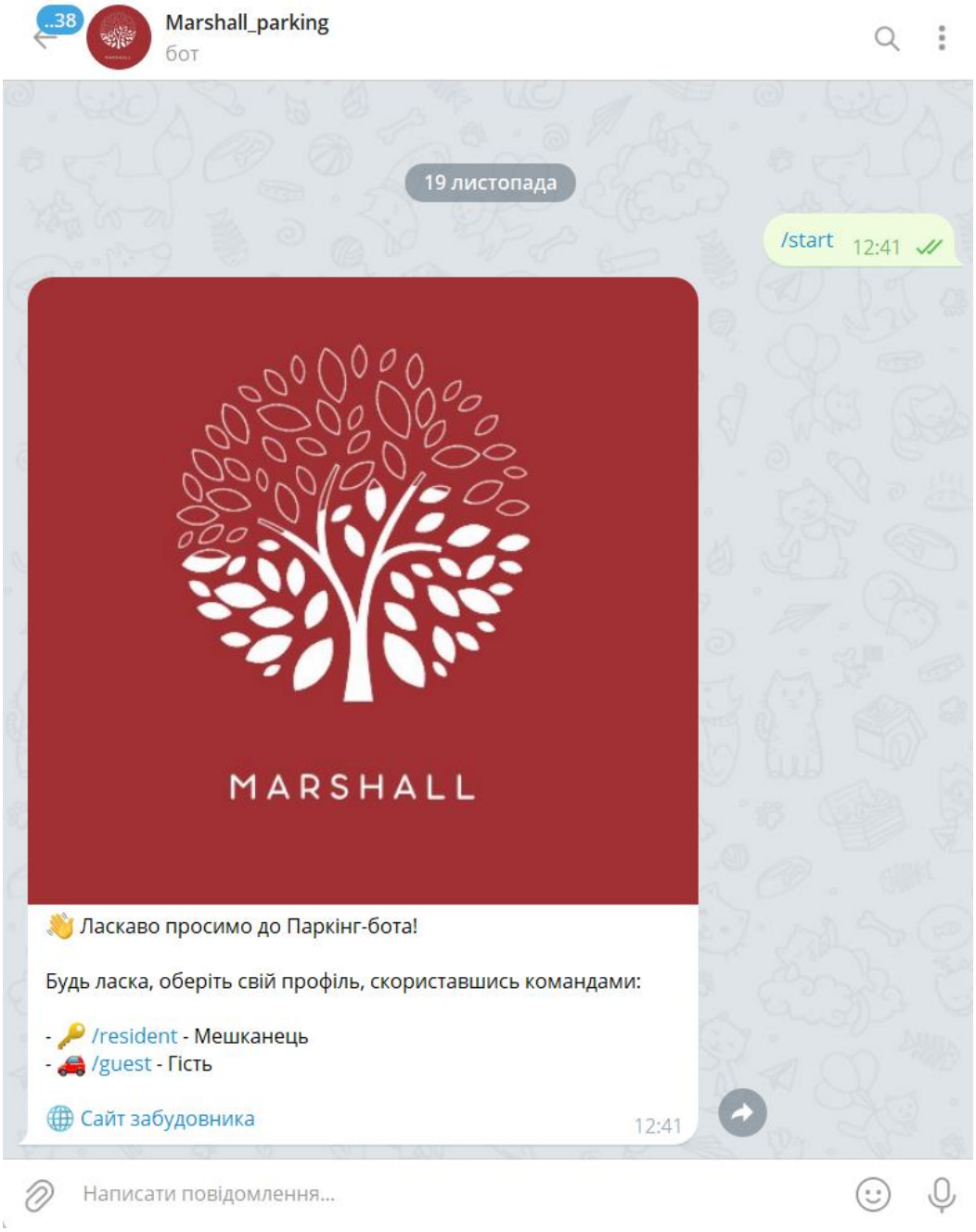

Рисунок 4.1 – Головне меню

При виборі посилання на сайт забудовника з'являється повідомлення для підтвердження дії користувача (див. рисунок 4.2).

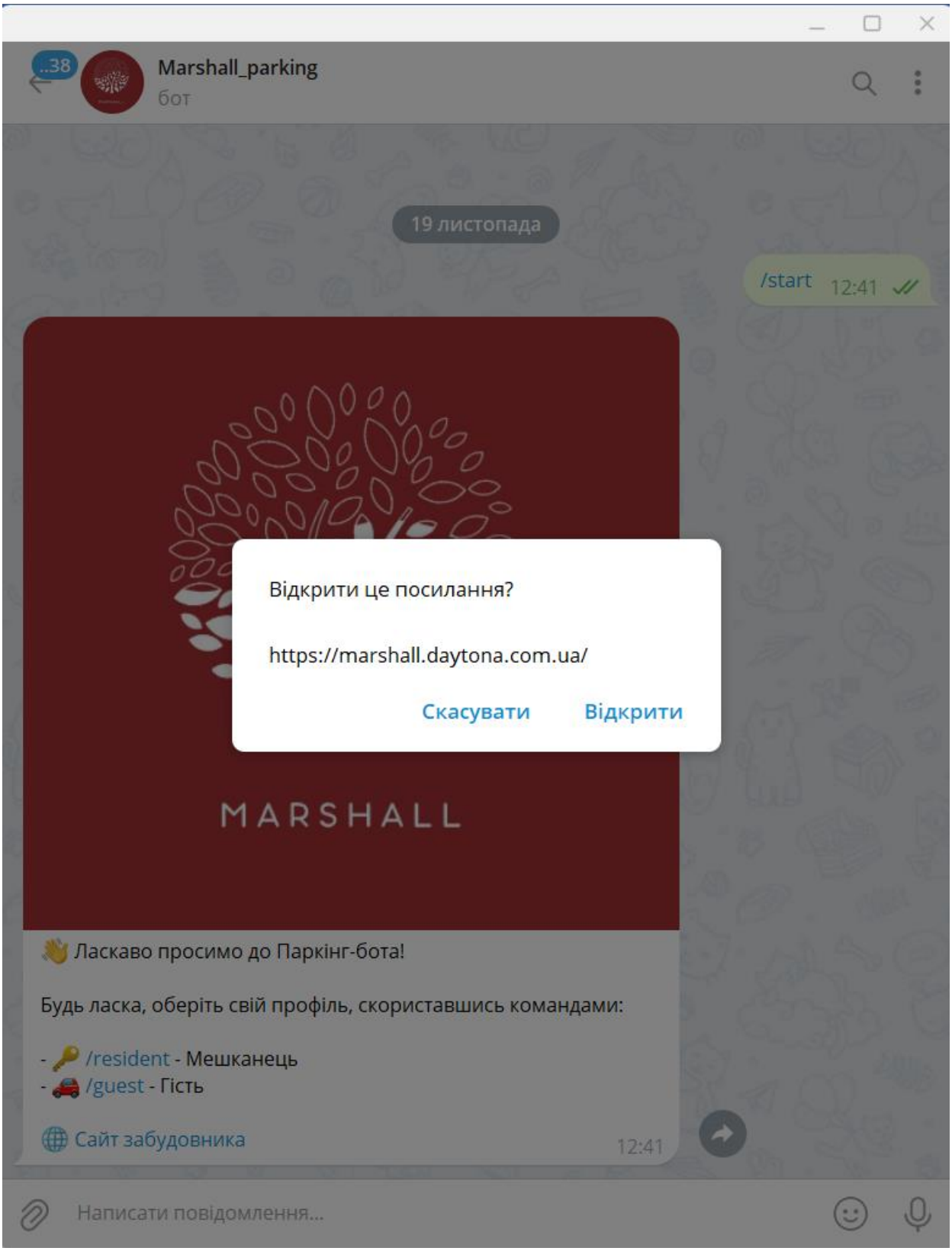

Рисунок 4.2 – Перехід на сайт забудовника за посиланням

Дії з гілкою «Мешканець», /resident.

При роботі з гілкою «Мешканець» програма передбачає додатковий рівень перевірки, щоб забезпечити доступ виключно для зареєстрованих мешканців ЖК. Перед тим як отримати доступ до команд, які призначені для мешканців, користувач зобов'язаний ввести номер свого автомобіля, зареєстрований у базі даних ЖК.

Цей номер використовується для ідентифікації користувача як мешканця, щоб гарантувати, що доступ до функцій цієї гілки отримують лише уповноважені особи.

Важливий аспект: користувач має право на дев’ять помилок введення. Це зроблено для того, щоб врахувати можливі технічні або людські помилки, наприклад, неправильний формат чи друкарську помилку (див. рисунок 4.3).

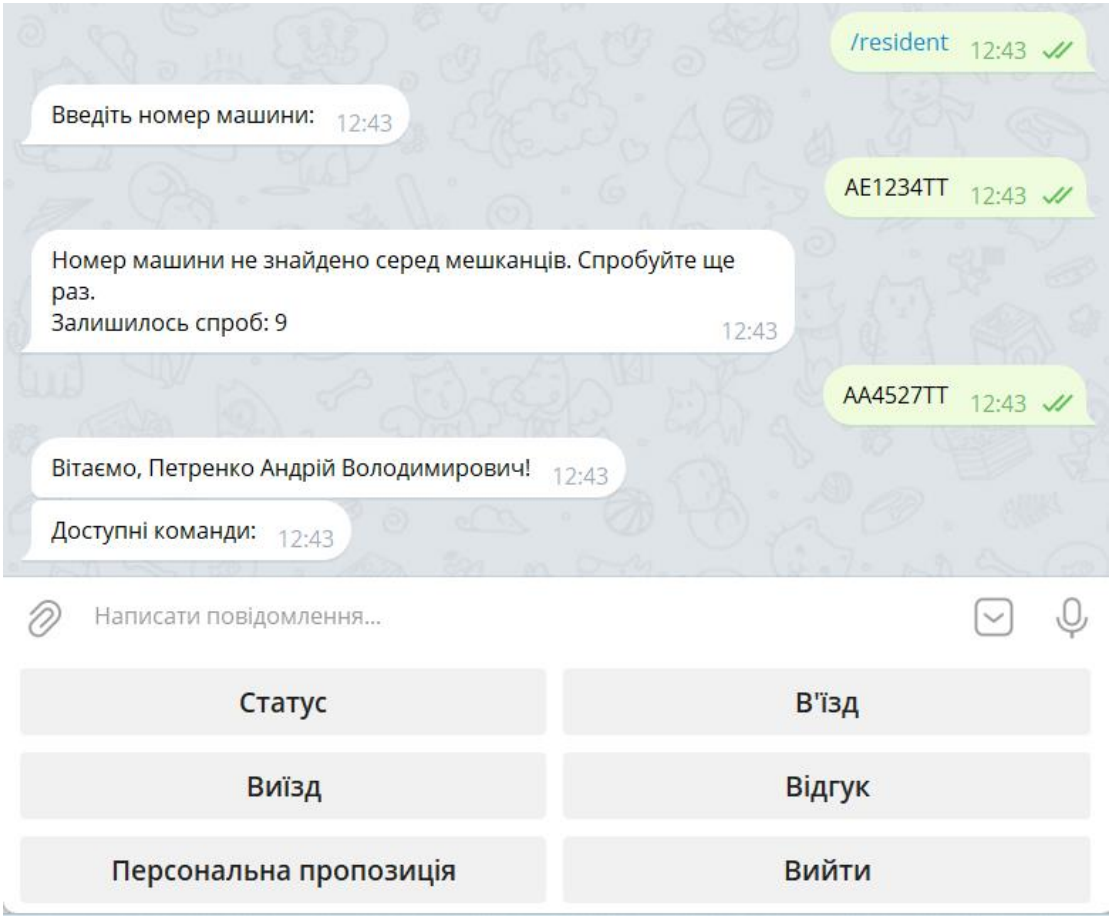

Рисунок 4.3 – Перехід та підтвердження зареєстрованих користувачів, мешканців ЖК

Після успішної авторизації та отримання доступу до профілю мешканця, програма надає широкий функціонал для зручного та інтерактивного управління можливостями, пов'язаними з паркінгом ЖК. Інтерфейс та доступні функції розроблені з урахуванням потреб мешканців, що дозволяє швидко й ефективно виконувати необхідні дії.

Після входу в профіль мешканця програма пропонує наступні можливості:

– перевірка статусу паркомісця, мешканець може дізнатися поточний статус свого паркомісця (наприклад, зайняте чи вільне);

– керування шлагбаумом, надається можливість дистанційного керування шлагбаумом через команди "Виїзд" та "В’їзд" (див. рисунок 4.4). Ця функція дозволяє мешканцям без зайвих затримок відкрити шлагбаум, забезпечуючи швидкий доступ до території ЖК або виїзд із неї;

– отримання спеціальної акційної пропозиції від керуючої компанії відповідно до профіля мешканця (див. рисунок 4.5);

– залишення відгуку (див. рисунок 4.6-4.7);

– кнопка виходу з меню мешканця (див. рисунок 4.8).

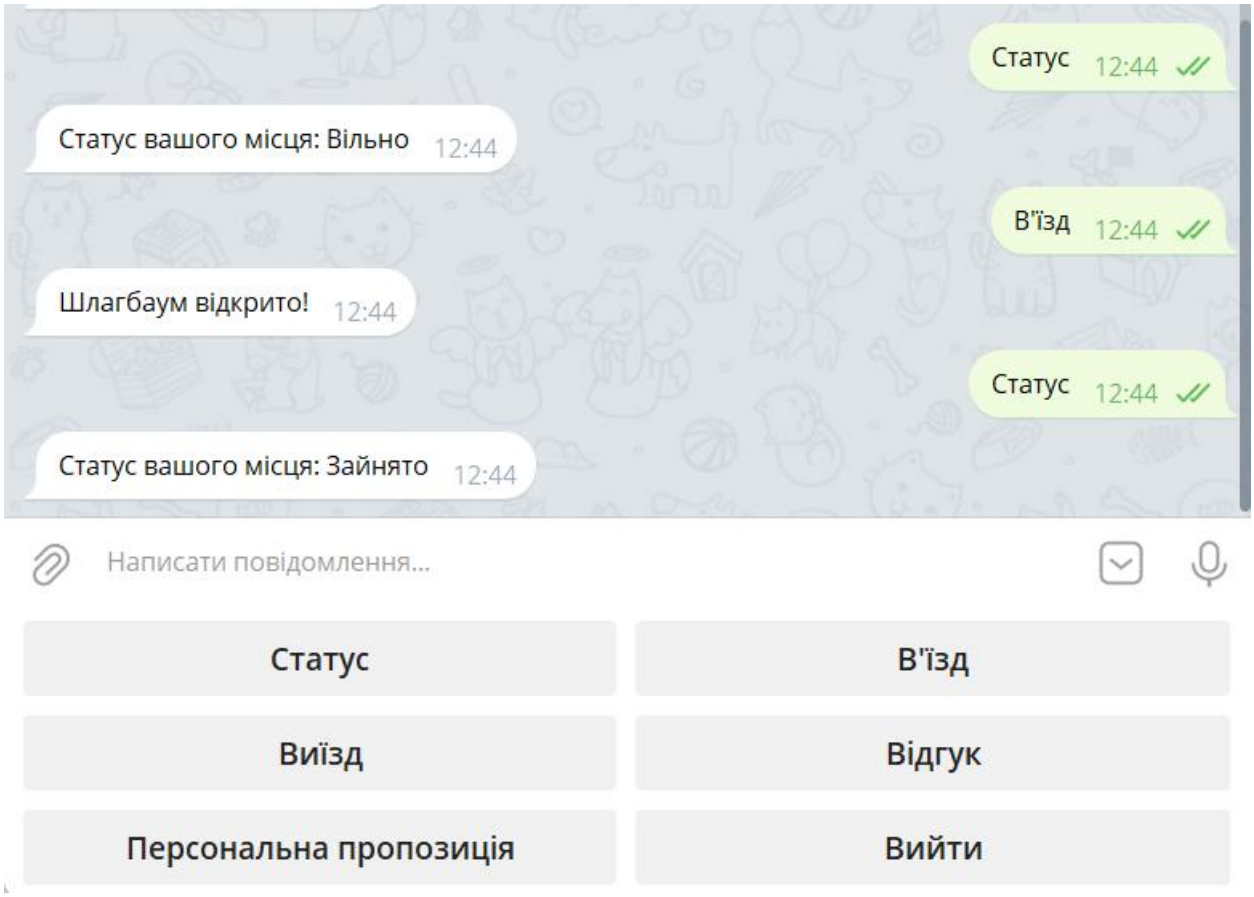

Рисунок 4.4 – Управління шлагбаумом через телеграм-бот

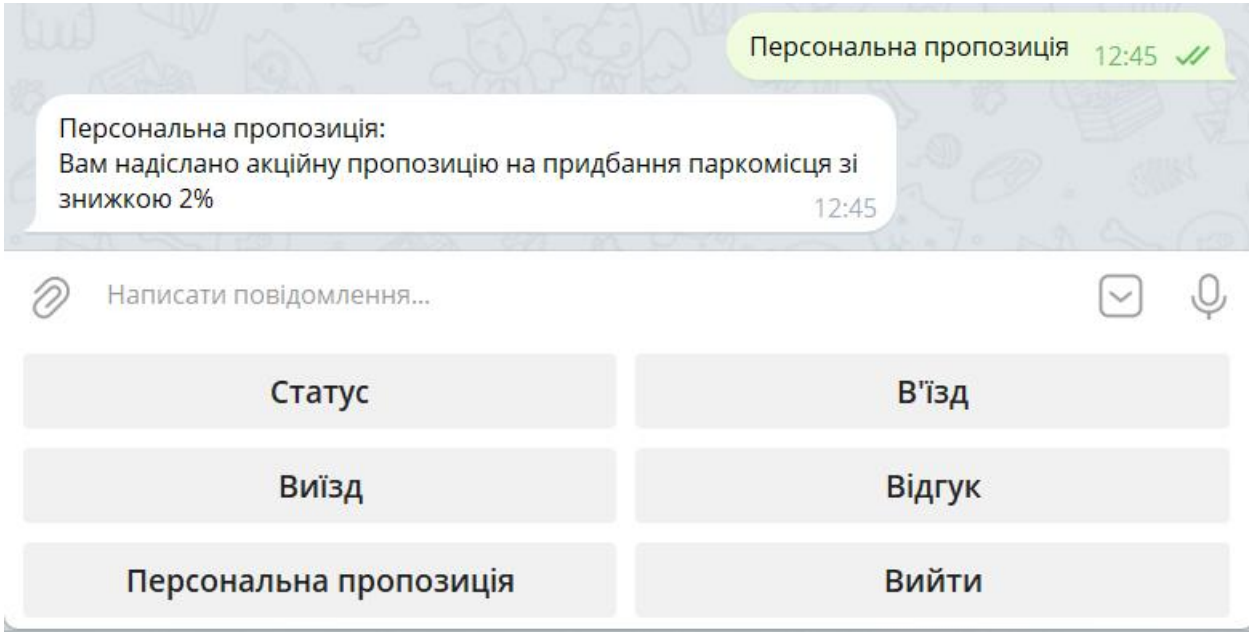

Рисунок 4.5 – Персональна акційна пропозиція для мешканця

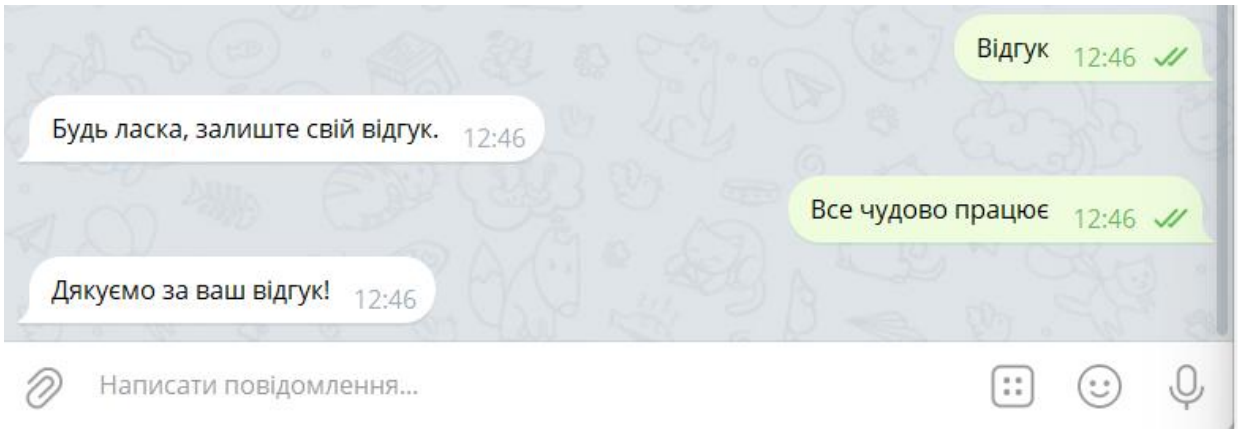

Рисунок 4.6 – Відгук про роботу телеграм-бота та паркінгу

```
2024-11-19 12:46:40,424 - httpx - INFO - HTTP Request: POST https://api.telegram
.org/bot7927229024:AAHntPx7-NnsypclfL7A5c1SPbo9JPjRAvU/getUpdates "HTTP/1.1 200
OK"
База даних для відгуків ініціалізована.
Відгук збережено для користувача з ID 346478985
2024-11-19 12:46:41,071 - httpx - INFO - HTTP Request: POST https://api.telegram
.org/bot7927229024:AAHntPx7-NnsypclfL7A5c1SPbo9JPjRAvU/sendMessage "HTTP/1.1 200
 OK"
2024-11-19 12:46:41,101 - handlers.resident_handler - INFO - Отримано відгук від
 мешканця AA4527TT: Все чудово працює
2024-11-19 12:46:50,515 - httpx - INFO - HTTP Request: POST https://api.telegram
.org/bot7927229024:AAHntPx7-NnsypclfL7A5c1SPbo9JPjRAvU/getUpdates "HTTP/1.1 200
OK"
```

Рисунок 4.7 – Занесення відгуку у БД

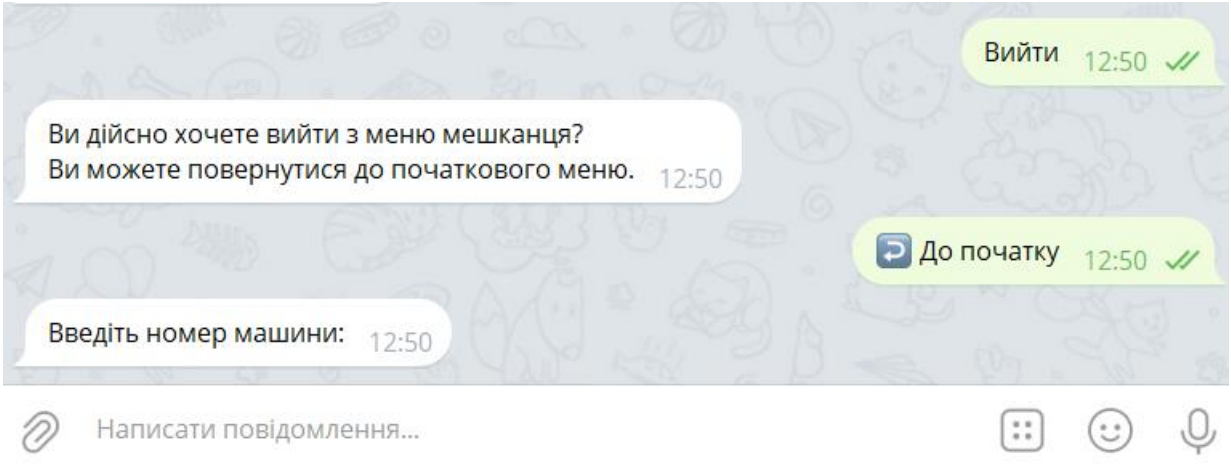

Рисунок 4.8 – Вихід з меню мешканець : перехід до початку авторизації мешканця

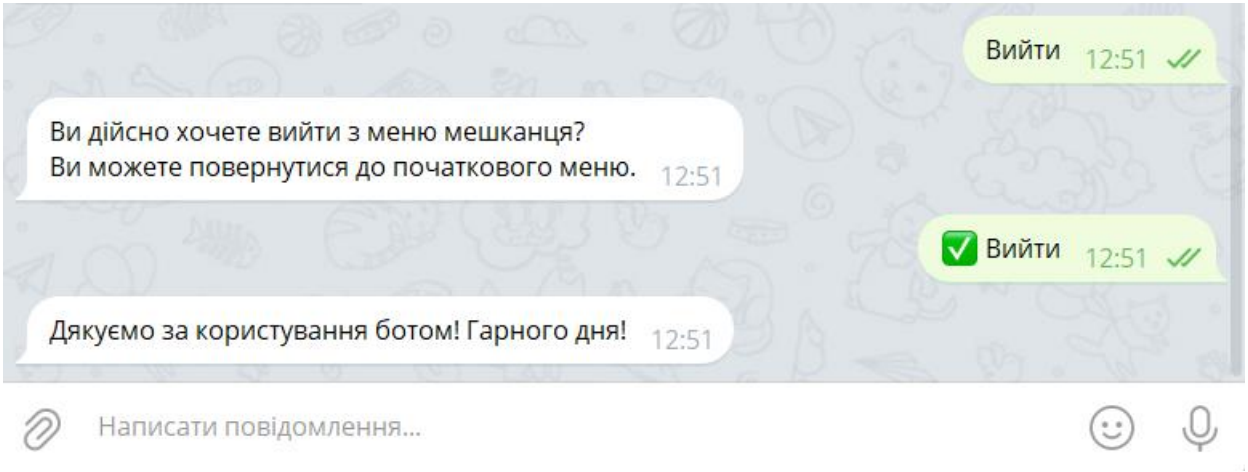

Рисунок 4.9 – Вихід з меню мешканець: завершення роботи

Гілка «Гість», /guest/.

При виборі гілки «Гість» програма надає набір команд, адаптованих для користувачів, які не є мешканцями ЖК, але мають намір скористатися послугами паркінгу. Інтерфейс для гостей розроблений таким чином, щоб забезпечити простоту і зручність виконання необхідних дій.

Після вибору цього режиму на екрані з’являється спливаюча клавіатура з доступними командами, які дозволяють гостю взаємодіяти із системою паркінгу (див. рисунок 4.10). До основних команд належать:

– забронювати місце, користувач має можливість заздалегідь забронювати паркомісце. Для цього пропонується вибрати сектор паркінгу і обрати конкретне місце в обраному секторі. Після вибору система запитує номер автомобіля для підтвердження бронювання. Номерний знак використовується для автоматичної ідентифікації користувача при в’їзді. Після успішного бронювання користувач отримує повідомлення про підтвердження, а також надається доступ для відкриття шлагбаума на в’їзді до паркінгу (див. рисунок 4.11-4.13);

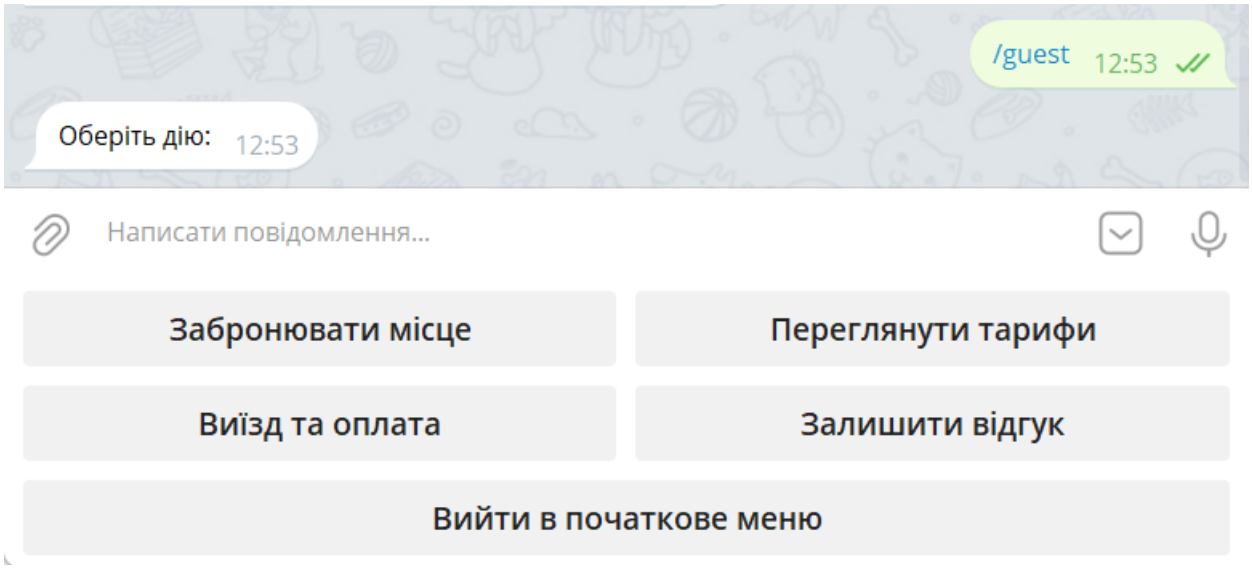

Рисунок 4.10 – Загальне меню користувача «Гість»

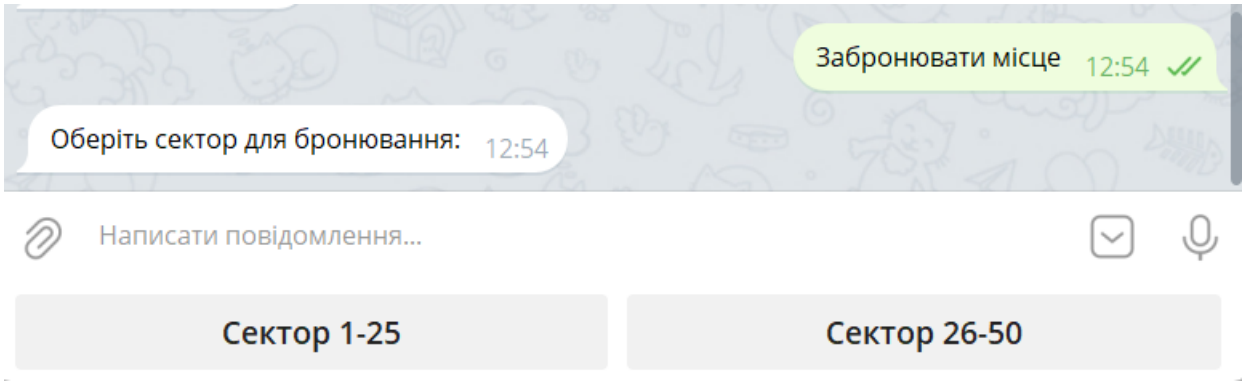

Рисунок 4.11 – Бронювання місця: обрання сектору

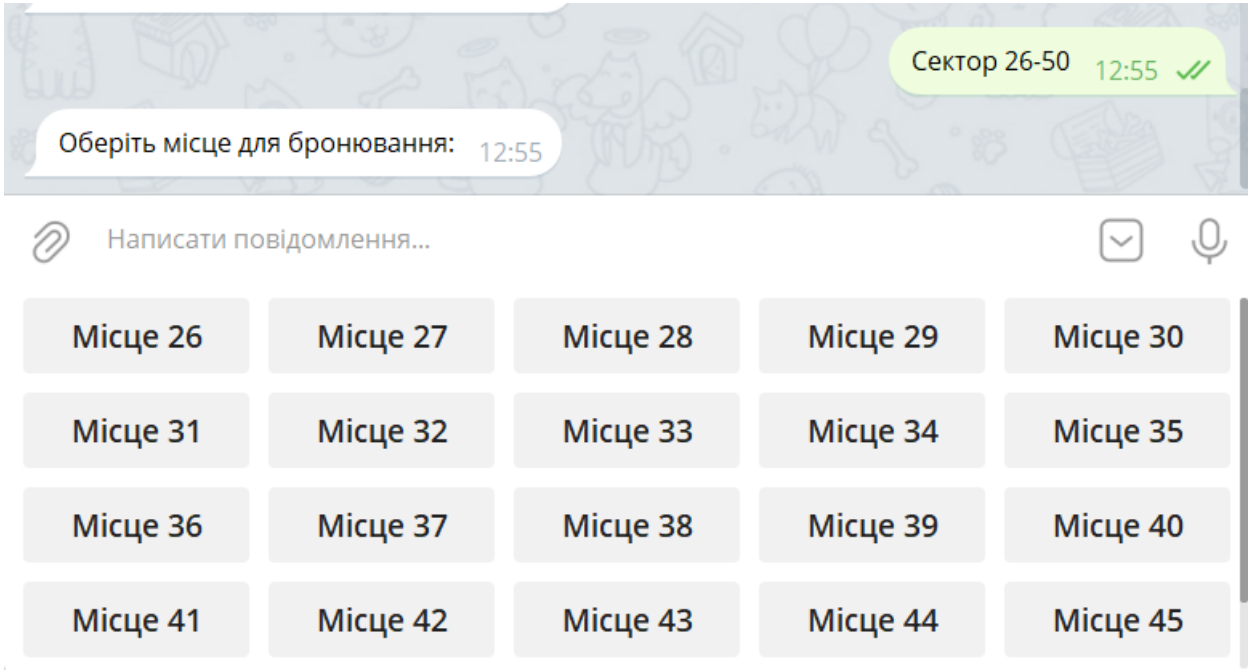

Рисунок 4.12 – Бронювання місця: вибір місця паркування

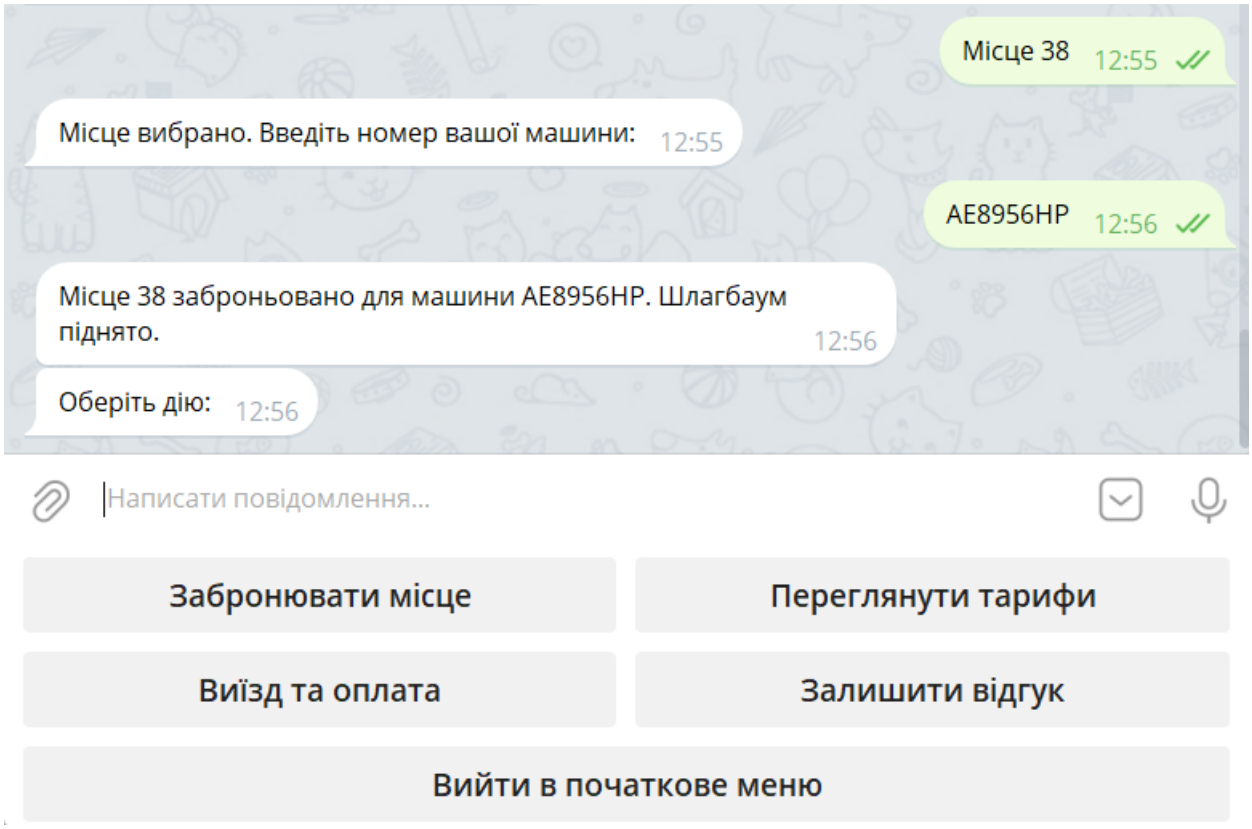

Рисунок 4.13 – Бронювання місця: ввід номерного знаку та в’їзду до паркінгу

– переглянути тарифи, ця команда дозволяє ознайомитися з актуальними тарифами на паркування (див. рисунок 4.14);

– виїзд та оплата, після завершення використання паркінгу гість може скористатися цією командою для виїзду. Програма автоматично підраховує вартість паркування, виходячи з тривалості перебування та обраного тарифу. Після підтвердження оплати шлагбаум відкривається, і користувач отримує дозвіл на виїзд (див. рисунок 4.15);

– залишити відгук, гостю надається можливість висловити свої враження від користування паркінгом або програмою. Відгуки можуть стосуватися як технічних аспектів роботи, так і загального обслуговування, що дозволяє адміністрації ЖК поліпшувати сервіс (див. рисунок 4.16);

– вихід з гілки "Гість", ця команда дозволяє завершити роботу в режимі "Гість" та повернутися до головного меню (див. рисунок 4.17).

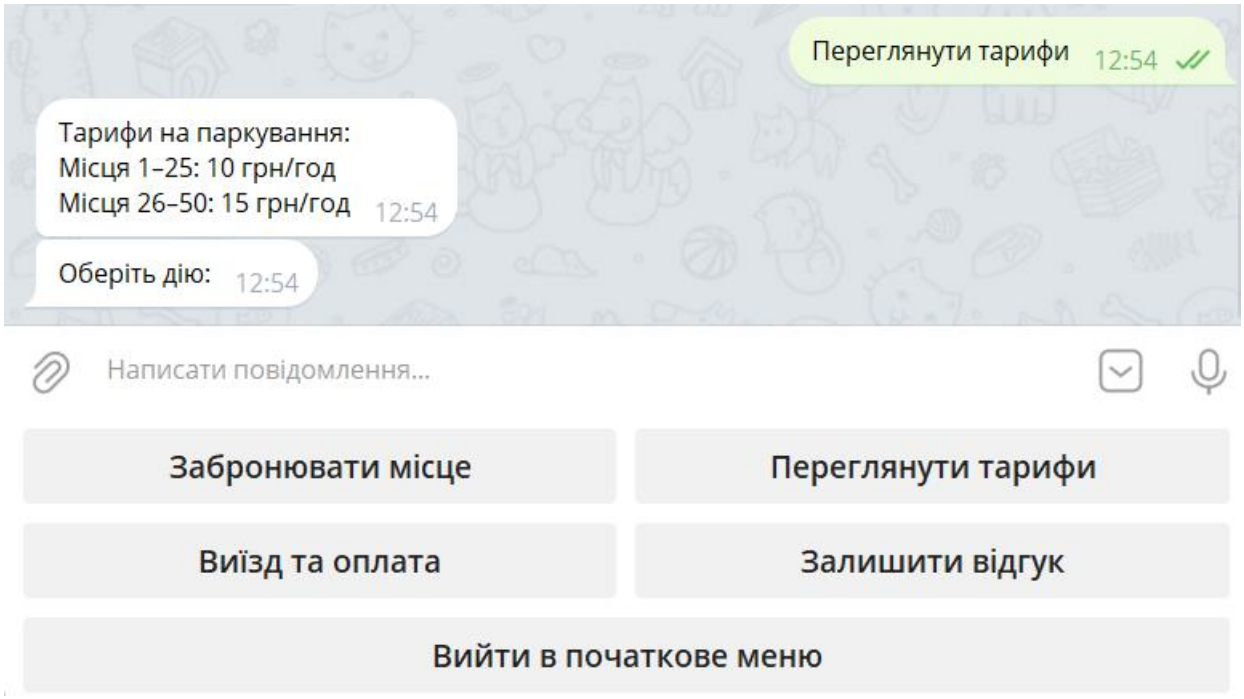

Рисунок 4.14 – Перегляд тарифів

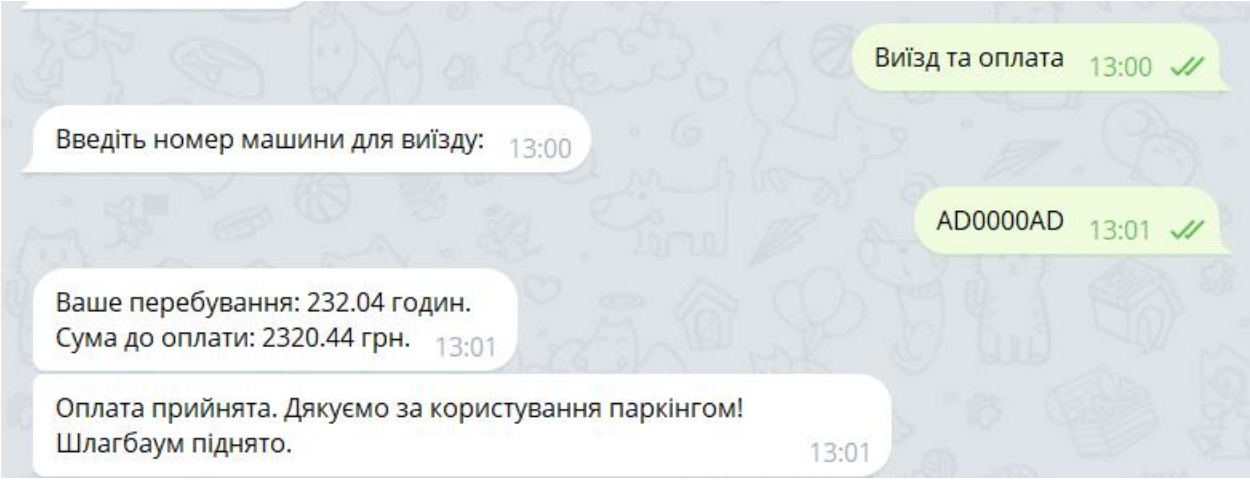

Рисунок 4.15 – Оплата та виїзд з паркінгу

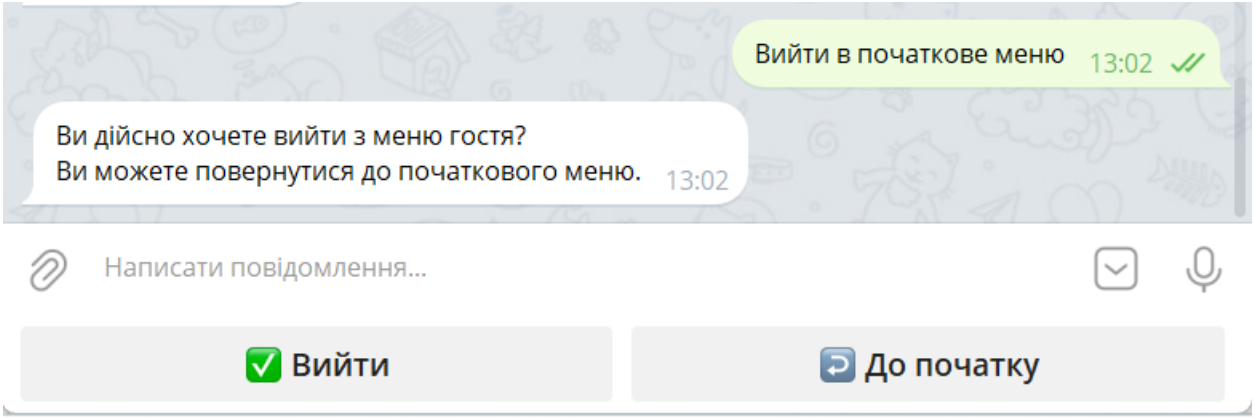

Рисунок 4.16 – Вихід з гілки «Гість»

Застосунок також має додаткову функцію розсилки новин, яка працює у фоновому режимі. Кожні 30 хвилин система автоматично надсилає актуальні новини та оновлення, пов’язані з паркінгом, усім зареєстрованим користувачам, включаючи мешканців і гостей. Ця можливість дає користувачам змогу бути в курсі останніх новин, змін або. важливих оголошень, що стосуються роботи паркінгу ЖК (див. рисунок 4.17).

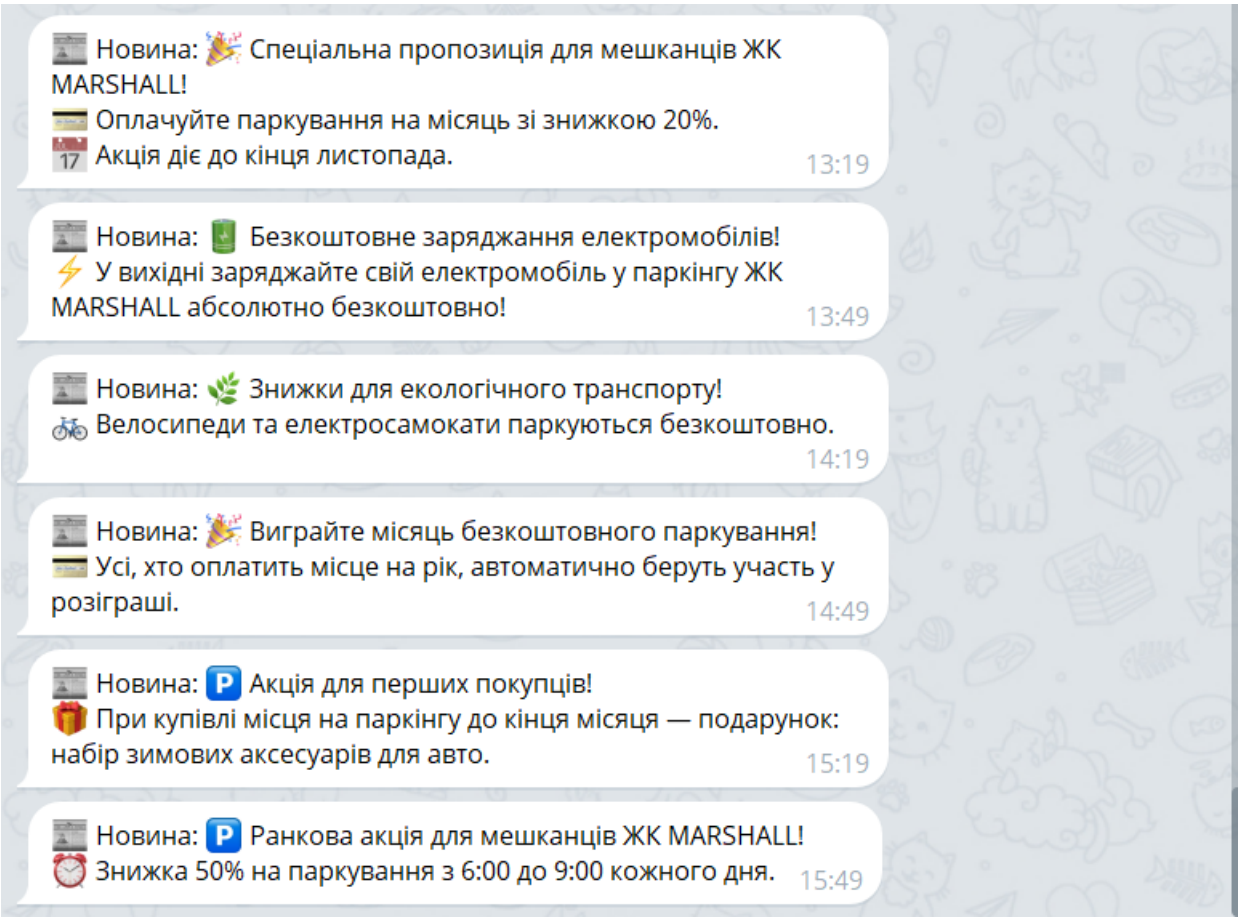

Рисунок 4.17 – Розсилка новин паркінгу

4.3.1 Загальні відомості

Програма реалізує телеграм-бота для керування паркуванням в ЖК MARSHALL. Бот дозволяє двом категоріям користувачів – гостям та мешканцям – взаємодіяти з паркінгом: бронювати паркомісця, переглядати тарифи, виїжджати із розрахунком вартості, залишати відгуки, а також отримувати персональні пропозиції (для мешканців).

Телеграм -бот створено з використанням мови програмування Python із застосуванням бібліотеки python-telegram-bot [16][17][18]. Обробка даних виконується через API, що зв’язує бот із сервером. Дані зберігаються в базі даних parking.db, яка працює на SQLite [19][20].

Програма сумісна з операційними системами Linux і має низькі вимоги до апаратних ресурсів, що робить її легкою для інтеграції та масштабування в інфраструктуру ЖК MARSHALL.

4.3.2 Функціональне призначення

Програма виконує роль інтеграційного інтерфейсу між користувачем і серверною частиною системи. Вона дозволяє автоматизувати основні операції, такі як перегляд доступності паркомісць, бронювання, здійснення оплати та управління шлагбаумом.

Основні функції

– перегляд доступних паркомісць з актуальним статусом;

– бронювання місця із зазначенням номера автомобіля;

– автоматичне відкриття шлагбаума для мешканців після підтвердження запиту.

4.3.3 Опис логічної структури програми

Логічна структура телеграм-бота для системи паркування ЖК MARSHALL складається з кількох взаємопов’язаних модулів, кожен з яких виконує свою роль у загальному функціонуванні програми. Це забезпечує гнучкість, масштабованість і простоту підтримки програмного забезпечення.

Програма складається з декількох основних компонентів (модулів), які забезпечують її функціонування.

Модуль взаємодії з користувачем:

– приймає текстові команди або вибір кнопок у Telegram;

– забезпечує доступ до головного меню та обробку основних функцій, таких як перегляд паркомісць, бронювання, оплата та відкриття шлагбаума;

– відповідає за генерацію відповідей і надсилання їх користувачу.

Модуль обробки запитів:

– формує запити до серверної частини системи залежно від введених команд;

– передає параметри, такі як номер автомобіля, для подальшої обробки;

– повертає результати запиту у вигляді даних, які передаються в модуль взаємодії з користувачем.

Модуль роботи з базою даних:

– взаємодіє з базою даних parking.db, яка містить інформацію про: статус паркомісць (вільне/зайняте), записи про бронювання, історію оплат;

– виконує операції зчитування, оновлення та запису даних.

Модуль управління шлагбаумом:

– інтегрується з апаратною частиною через API для надсилання команд на відкриття шлагбаума;

– забезпечує виконання запитів на основі перевірки статусу оплати або прав доступу мешканців.

Логічна структура програми:

1. main.py. Функціонал:

– ініціалізація бота;

– реєстрація обробників: start_handler.py, guest_handler.py, resident_handler.py;

– реєстрація основних команд: /start – запуск бота та вибір профілю, /guest — перехід до меню гостя, /resident – перехід до меню мешканця.

2. Обробники:

a) start_handler.py:

– відображення стартового меню з вибором ролі;

– перехід на сайт забудовника;

– команди: /start, виводить текст привітання та має опції – Мешканець (перехід до гілки мешканця), Гість (перехід до гілки гостя), Сайт забудовника (кнопка, що веде на зовнішній сайт);

b) guest_handler.py:

– реалізація функціоналу для гостей;

– команди: /book – Бронювання місця, /tariffs – Перегляд тарифів, /checkout – Виїзд та оплата, /feedback – Відгук, /exit_guest – Вихід із меню гостя;

c) resident_handler.py:

– реалізація функціоналу для мешканців;

– команди: /status – Перевірка статусу паркомісця, /enter – В'їзд,

/leave – Виїзд, /feedback – Відгук, /personal_offer – Персональна пропозиція; /exit_resident – Вихід із меню мешканця.

Нижче представлені фрагменти коду основних модулів програми разом із описом їх функціоналу. До складу цих модулів входять: main.py, start_handler.py, guest_handler.py та resident_handler.py (див. рисунок 4.18-4.23).

Детальний опис кожного модуля, а також повний текст програмного коду проекту, наведені у додатку А.

```
9
10  # Обробник для команди /start
11  async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
12      """
13      Відображає стартове привітання користувачам та пропонує вибір профілю.
14
15      - Зберігає chat_id активного користувача у боті.
16      - Надсилає повідомлення із логотипом та списком команд.
17      """
18      # Додати chat_id до active_chat_ids (для відстеження активних користувачів)
19      chat_id = update.effective_chat.id
20      active_chat_ids = context.bot_data.get("active_chat_ids", set())
21      active_chat_ids.add(chat_id)
22      context.bot_data["active_chat_ids"] = active_chat_ids
23      logger.info(f"Додано chat_id {chat_id} до active_chat_ids.")
24
25      # Шлях до зображення логотипу
26      logo_path = 'logo.png'
27
28      # Текст привітання та вибір профілю
29      welcome_text = (
30          "👋 Ласкаво просимо до Паркінг-бота!\n\n"
31          "Будь ласка, оберіть свій профіль, скориставшись командами:\n\n"
32          "- 🔑 /resident - Мешканець\n"
33          "- 🚗 /guest - Гість\n\n"
34          "🌐 [Сайт забудовника](https://marshall.daytona.com.ua)"
35      )
```

Рисунок 4.18 – Фрагмент коду start_handler.py

```python
# Підключення до бази даних і таблиць
def init_db():
    conn = sqlite3.connect('parking.db')
    cursor = conn.cursor()

    # Основна таблиця для паркомісць
    cursor.execute('''CREATE TABLE IF NOT EXISTS parking_spots (
                        id INTEGER PRIMARY KEY,
                        status TEXT,
                        car_number TEXT,
                        booking_time TEXT)''')

    # Таблиця для зберігання статистики
    cursor.execute('''CREATE TABLE IF NOT EXISTS booking_statistics (
                        spot_id INTEGER PRIMARY KEY,
                        booking_count INTEGER DEFAULT 0,
                        total_duration REAL DEFAULT 0.0
                    )''')

    # Ініціалізація таблиць, якщо ще не створено
    for i in range(1, 51):
        cursor.execute('INSERT OR IGNORE INTO parking_spots (id, status) VALUES (?, "вільне")', (i,))
        cursor.execute('INSERT OR IGNORE INTO booking_statistics (spot_id) VALUES (?)', (i,))

    conn.commit()
    conn.close()
```

Рисунок 4.19 – Фрагмент коду guest_handler.py чистина 1

```python
# Обробник розмови з гостем
guest_conversation_handler = ConversationHandler(
    entry_points=[CommandHandler("guest", guest)],
    states={
        SELECT_ACTION: [
            MessageHandler(filters.Regex("^Забронювати місце$"), book),
            MessageHandler(filters.Regex("^Переглянути тарифи$"), view_tariffs),
            MessageHandler(filters.Regex("^Виїзд та оплата$"), checkout_car_number),
            MessageHandler(filters.Regex("^Залишити відгук$"), feedback),
            MessageHandler(filters.Regex("^Вийти в початкове меню$"), exit_guest),
        ],
        SECTOR_SELECTION: [MessageHandler(filters.TEXT & ~filters.COMMAND, select_sector)],
        SPOT_SELECTION: [MessageHandler(filters.TEXT & ~filters.COMMAND, confirm_spot)],
        INPUT_CAR_NUMBER: [MessageHandler(filters.TEXT & ~filters.COMMAND, input_car_number)],
        CHECKOUT: [MessageHandler(filters.TEXT & ~filters.COMMAND, checkout)],
        FEEDBACK_TEXT: [MessageHandler(filters.TEXT & ~filters.COMMAND, handle_feedback)],
        EXIT_CONFIRMATION: [MessageHandler(filters.TEXT & ~filters.COMMAND, confirm_exit_guest)],
    },
    fallbacks=[CommandHandler("cancel", exit_guest)],
)
```

Рисунок 4.20 – Фрагмент коду guest_handler.py чистина 2

```python
async def handle_car_number(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    car_number = update.message.text.strip()  # Отримання номера машини
    logger.info(f"Отримано номер машини: {car_number}")

    # Перевірка номера машини
    if car_number not in residents:
        context.user_data["attempts"] += 1
        attempts_left = MAX_ATTEMPTS - context.user_data["attempts"]

        if context.user_data["attempts"] >= MAX_ATTEMPTS:
            await update.message.reply_text("На жаль, ви вичерпали всі спроби. Будь ласка, спробуйте пізніше.")
            logger.warning(f"Мешканець не зміг ввести правильний номер машини за {MAX_ATTEMPTS} спроб.")
            return ConversationHandler.END

        await update.message.reply_text(
            f"Номер машини не знайдено серед мешканців. Спробуйте ще раз.\n"
            f"Залишилось спроб: {attempts_left}"
        )
        logger.warning(f"Невірний номер машини: {car_number}. Залишилось спроб: {attempts_left}")
        return CAR_NUMBER

    resident_name = residents[car_number]
    await update.message.reply_text(f"Вітаємо, {resident_name}!")
    logger.info(f"Мешканець {resident_name} успішно пройшов перевірку за номером машини {car_number}.")

    # Виведення команд для мешканця у вигляді блоку кнопок
    resident_menu_keyboard = [
        ["Статус", "В'їзд"],
        ["Виїзд", "Відгук"],
        ["Персональна пропозиція", "Вийти"]
    ]
    reply_markup = ReplyKeyboardMarkup(resident_menu_keyboard, resize_keyboard=True)
```

Рисунок 4.21 – Фрагмент коду resident_handler.py чистина 1

```python
async def confirm_exit(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Підтвердження виходу до стартового меню через команду /start."""
    response = update.message.text
    if response == "✅ Вийти":
        await update.message.reply_text("Дякуємо за користування ботом! Гарного дня!", reply_markup=ReplyKeyboardRemove())
        logger.info("Користувач підтвердив вихід з гілки 'Мешканець'.")
        return ConversationHandler.END  # Завершуємо розмову без повернення до початкового меню
    elif response == "↩️ До початку":
        return await resident(update, context)  # Повернення в гілку мешканця
    else:
        await update.message.reply_text("Будь ласка, виберіть одну з опцій.")
        return EXIT_CONFIRMATION

# Обробник розмови з мешканцем
resident_conversation_handler = ConversationHandler(
    entry_points=[CommandHandler("resident", resident)],
    states={
        CAR_NUMBER: [
            MessageHandler(filters.Regex("^Статус$"), check_status),
            MessageHandler(filters.Regex("^В'їзд$"), open_gate_handler),  # В'їзд на паркомісце
            MessageHandler(filters.Regex("^Виїзд$"), leave_parking),      # Виїзд з паркомісця
            MessageHandler(filters.Regex("^Відгук$"), feedback),
            MessageHandler(filters.Regex("^Персональна пропозиція$"), show_personal_offer),
            MessageHandler(filters.Regex("^Вийти$"), exit_menu),
            MessageHandler(filters.TEXT & ~filters.COMMAND, handle_car_number)
        ],
        FEEDBACK_TEXT: [MessageHandler(filters.TEXT & ~filters.COMMAND, handle_feedback)],
        EXIT_CONFIRMATION: [MessageHandler(filters.Regex("^(✅ Вийти|↩️ До початку)$"), confirm_exit)],
    },
    fallbacks=[CommandHandler("cancel", exit_menu)],
)
```

Рисунок 4.22 – Фрагмент коду resident_handler.py чистина 2

```python
# Налаштування логування
logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO
)
logger = logging.getLogger(__name__)

async def send_random_news(context):
    """Функція для відправки випадкової новини."""
    news = get_random_news()
    active_chat_ids = context.bot_data.get("active_chat_ids", set())

    for chat_id in active_chat_ids:
        await context.bot.send_message(chat_id=chat_id, text=f" Новина: {news}")
        logger.info(f"Новина надіслана в чат {chat_id}.")

def main():
    app = ApplicationBuilder().token(config.BOT_TOKEN).build()

    # Реєстрація обробників команд
    app.add_handler(CommandHandler("start", start))                    # Обробник команди /start
    app.add_handler(resident_conversation_handler)                     # Обробник для мешканця
    app.add_handler(guest_conversation_handler)                        # Обробник для гостя
    app.add_handler(CommandHandler("open_gate", open_gate_handler))    # Обробник відкриття шлагбаума

    # Фонове завдання для новин
    job_queue = app.job_queue
    job_queue.run_repeating(send_random_news, interval=1800, first=10)

    logger.info("Bot is running in test mode...")
    app.run_polling()
```

Рисунок 4.22 – Фрагмент коду main.py

5 ЕКСПЕРИМЕНТАЛЬНИЙ РОЗДІЛ

5.1 Мета і завдання експерименту

Метою експерименту є перевірка функціональності розробленого телеграм-бота для управління системою паркування в житловому комплексі MARSHALL. Це включає оцінку роботи всіх ключових модулів, визначення відповідності програмного забезпечення висунутим технічним і функціональним вимогам, а також аналіз взаємодії бот-системи з користувачами і апаратними компонентами.

Завдання експерименту:

– перевірка коректності роботи основного функціоналу бота, обробка текстових команд користувача, таких як перегляд доступних паркомісць, бронювання, отримання інформації про тарифи та відкриття шлагбаума, відображення коректних відповідей на запити користувачів;

– аналіз роботи серверної частини, перевірка обробки запитів, що надходять від бота, верифікація правильності взаємодії з базою даних parking.db (перевірка, бронювання, оновлення статусу місць);

– тестування інтеграції з апаратними компонентами, оцінка часу відгуку системи на запит мешканця або гостя, оцінка продуктивності, визначення часу відповіді бота на стандартні запити, аналіз стійкості системи до одночасної обробки запитів від кількох користувачів;

– вивчення користувацького досвіду, збір відгуків тестових користувачів щодо зручності інтерфейсу та функціоналу телеграм-бота, оцінка інтуїтивності взаємодії та швидкості виконання операцій.

Очікувані результати експерименту:

– отримання даних про ефективність роботи бота;

– виявлення можливих недоліків у роботі програмного забезпечення;

– формулювання рекомендацій щодо подальшого вдосконалення системи паркування ЖК MARSHALL.

5.2 Методика експерименту

Методика проведення експерименту передбачає поетапне тестування функціональності телеграм-бота та серверної інфраструктури. Експеримент спрямований на перевірку роботи системи в умовах реального використання, оцінку її продуктивності та аналіз відповідності функціональних можливостей заданим вимогам.

Підготовчий етап:

а) налаштування тестового середовища:

– розгортання телеграм-бота на тестовому сервері;

– підключення бази даних parking.db із попередньо заданими тестовими записами про паркомісця;

б) підготовка тестових сценаріїв:

– формулювання сценаріїв взаємодії користувачів із ботом, що включають перегляд, бронювання та відкриття шлагбаума;

– визначення параметрів тестування, таких як кількість одночасних запитів і сценарії навантаження;

в) створення тестової групи, включення тестових користувачів (мешканців і гостей) із різними ролями та правами доступу.

Проведення експерименту

а) тестування функціональності телеграм-бота:

– надсилання базових команд (/resident, /book, /guest, /feedback) та оцінка їх обробки;

– перевірка правильності формування відповідей і повідомлень користувачам;

б) перевірка серверної частини:

– аналіз запитів, що надходять від бота на сервер, та їх відповідності параметрам;

– тестування взаємодії сервера з базою даних, включаючи операції читання, запису та оновлення статусу паркомісць;

в) тестування апаратної інтеграції:

– симуляція запитів на відкриття шлагбаума після підтвердження оплати або перевірки прав доступу;

– оцінка часу відгуку системи на команди для управління шлагбаумом;

г) оцінка продуктивності:

– визначення часу відповіді системи на стандартні запити;

– імітація одночасного використання бота кількома користувачами.

Збір і обробка результатів:

а) логування:

– аналіз журналів операцій для перевірки коректності виконання запитів;

– виявлення помилок або некоректної роботи на кожному етапі;

б) оцінка користувацького досвіду:

– проведення анкетування тестової групи для збору відгуків про зручність інтерфейсу та швидкість роботи бота;

– аналіз відповідності системи очікуванням користувачів;

в) формування висновків:

– узагальнення отриманих даних для оцінки продуктивності системи;

– розробка рекомендацій щодо вдосконалення телеграм-бота та серверної частини.

Методика експерименту дасть змогу виявити сильні та слабкі сторони системи, оцінити її надійність, продуктивність і зручність у використанні. На основі отриманих даних будуть зроблені висновки щодо доцільності впровадження телеграм-бота в реальне використання та запропоновані можливі шляхи вдосконалення.

5.3 Вимоги до експерименту

Для забезпечення точності, об'єктивності та практичної цінності результатів експерименту з тестування телеграм-бота для системи паркування ЖК MARSHALL необхідно дотримуватись ряду технічних, методологічних та організаційних вимог.

Технічні вимоги:

– сервер повинен бути налаштований на безперебійну роботу протягом усього експерименту. Використання реальної бази даних parking.db із наперед заданими тестовими записами;

– телеграм-бот має бути розгорнутий на тестовому сервері, ізольованому від реального середовища, щоб уникнути впливу на реальних користувачів;

– система повинна мати стабільне підключення до Інтернету зі швидкістю не менше 10 Мбіт/с для обробки запитів у реальному часі.

Методологічні вимоги:

– необхідно охопити всі основні сценарії використання телеграм-бота, включаючи перегляд доступних паркомісць, бронювання та відкриття шлагбаума. Додатково протестувати роботу в умовах підвищеного навантаження (одночасна робота 2+ користувачів);

– кожна операція повинна мати чітко визначені вхідні дані, очікуваний результат і спосіб перевірки правильності виконання. Усі результати повинні бути зафіксовані у вигляді логів або звітів;

– тестові сценарії повинні відображати реальні умови використання, включаючи запити від мешканців і гостей із різними ролями доступу.

Організаційні вимоги:

– експеримент має бути поділений на етапи з чітко визначеним часом виконання кожного завдання. Час відповіді на запити бота не повинен перевищувати 1 секунди для перегляду доступних місць і 3 секунд для бронювання;

– всі етапи експерименту повинні бути задокументовані, включаючи опис середовища, хід тестування та отримані результати. Для кожного

тестового випадку необхідно вказати, чи досягнуто мету, і записати будь-які виявлені відхилення;

– тестування має включати участь реальних користувачів (мешканців та гостей), які можуть надати зворотний зв’язок щодо зручності та функціональності телеграм-бота.

5.4 Результати експерименту

5.4.1 Сутність експерименту

Експеримент полягав у перевірці функціональності розробленого телеграм-бота для управління системою паркування ЖК MARSHALL. Основною метою було оцінити коректність виконання ключових функцій бота, його взаємодію з серверною частиною та апаратними компонентами, а також перевірити відповідність результатів очікуванням користувачів і технічним вимогам.

У рамках експерименту участь взяли 5 користувачів, яким запропонували виконати наступні дії за допомогою телеграм-бота:

– вибір ролі користувача, де б обрали, який рівень доступу мають до паркінгу, гість чи мешканець;

– пошук вільного паркомісця, в результаті якого користувачі повинні були отримати інформаці про доступність паркомісць в реальному часі;

– отримання доступу до паркінгу, після підтвердження бронюваннння місця для користувачів з рівнем доступу «Гість», або після запиту на в’їзд до паркінгу – від «Мешканець»;

– можливість залишення відгуку про досвід користування паркінгом;

– для користувачів «Мешканець» отримання персональних акційних пропозицій за запитом;

– отримання актуальної інформації по тарифам для користувачів «Гість»;

– при запиті на виїзд та оплату від користувача «Гість» отримання інформації стосовно терміну користування паркінгом та необхідної суми до

сплати.

Після завершення кожної дії користувачі оцінювали свій рівень задоволеності роботою системи за 10-бальною шкалою.

Можливі сценарії роботи з телеграм-ботом

1. Перевірка статусу паркомісця. Користувач надсилає запит до телеграм-бота. Телеграм-бот приймає цей запит через Telegram API та передає його на сервер. Після отримання запиту сервер звертається до бази даних, де перевіряє актуальний статус паркомісця. Результат перевірки (вільне/зайняте) сервер формує у вигляді відповіді та надсилає назад до телеграм-бота. Телеграм-бот отримує відповідь від сервера, інтерпретує її та надсилає користувачу.

Схема роботи за цим сценарієм детально представлена на рисунку 5.1, де показано всі ключові етапи взаємодії: від надсилання запиту користувачем до отримання відповіді про статус паркомісця.

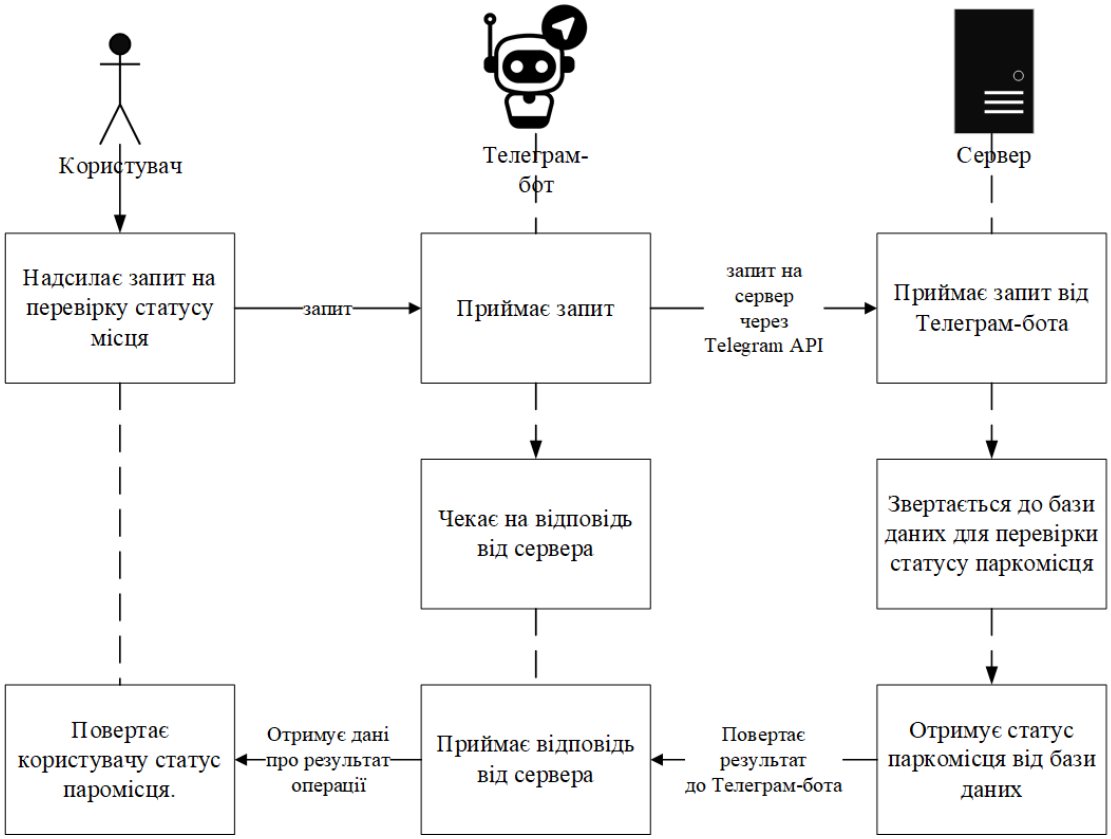

Рисунок 5.1 – Схема роботи за сценарієм «Перевірка статусу паркомісця»

2. Бронювання паркомісця. Користувач надсилає запит на бронювання паркомісця через телеграм-бот. Бот отримує цей запит і пересилає його на сервер. Сервер обробляє запит, звертається до бази даних і формує список доступних паркомісць. Отриманий список сервер повертає телеграм-боту, який відображає його користувачу у вигляді зручного повідомлення.

Користувач обирає номер бажаного паркомісця зі списку та надсилає цей вибір через бот. Телеграм-бот приймає номер місця і запитує у користувача додаткову інформацію для підтвердження бронювання, номер автомобіля. Користувач вводить номер авто, і бот пересилає ці дані на сервер.

Сервер отримує повний запит, що включає номер місця та номер авто, звертається до бази даних для внесення відповідного запису про бронювання. Після успішного виконання бронювання сервер формує відповідь для телеграм-бота з інформацією про результат. Одночасно сервер надсилає команду на пристрій шлагбаума (у даному випадку – його імітацію), щоб підготувати його до відкриття для обраного користувача.

Телеграм-бот отримує підтвердження бронювання і надсилає користувачу повідомлення про успішне завершення операції, включаючи деталі бронювання.

Схема роботи за цим сценарієм представлена на рисунку 5.2.

Рисунок 5.2 – Схема роботи за сценарієм «Бронювання паркомісця»

3. Виїзд та оплата. Користувач надсилає запит на сплату та виїзд через телеграм-бот. Бот приймає цей запит і запитує у користувача номер автомобіля, для якого необхідно розрахувати суму до сплати та надати дозвіл на виїзд. Користувач вводить номер автомобіля, і телеграм-бот передає отримані дані на сервер для обробки.

Сервер приймає запит, звертається до бази даних і визначає суму до сплати на основі часу користування паркомісцем. Ця інформація передається назад телеграм-боту, який відображає користувачу точну суму.

Користувач здійснює оплату через термінал (імітація процесу оплати), після чого сервер отримує підтвердження успішної транзакції. Сервер оновлює статус паркомісця в базі даних, змінюючи його на «вільне», і заносить відповідний запис до статистики використання.

Одночасно сервер надсилає підтвердження успішного виконання операції до телеграм-бота, який інформує користувача. Паралельно сервер імітує підняття шлагбаума, щоб забезпечити користувачу можливість виїзду.

Схема роботи наведена на рисунку 5.3.

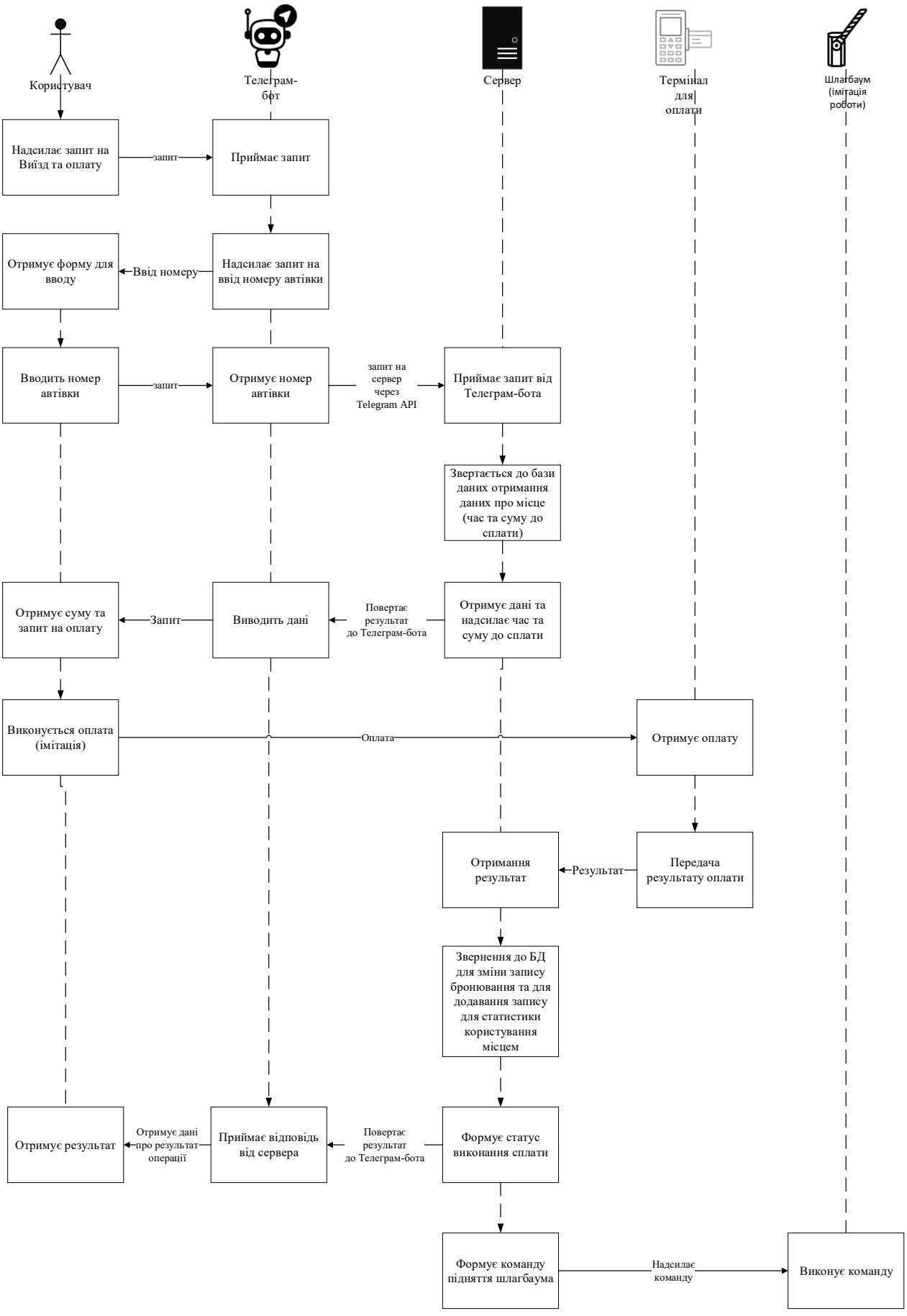

Рисунок 5.3 – Схема роботи за сценарієм «Виїзд та оплата»

5.4.2 Результати експерименту в цифрах і фактах

Результати експерименту дозволили оцінити зручність та ефективність розробленої системи. У процесі виконання завдань було проаналізовано такі показники: середній час виконання кожної дії, кількість помилок у роботі телеграм-бота, рівень задоволеності користувачів, а також відповідність функціональності системи технічним вимогам.

Для визначення часу виконання кожної операції було використано логування часових міток у коді системи. Логування дозволило автоматично фіксувати проміжки часу між ключовими етапами операцій, зокрема:

– момент надсилання запиту користувачем у телеграм-боті;

–час передачі запиту серверу;

–час обробки сервером запиту у базі даних;

–момент отримання результату та його відправки назад телеграм-боту.

Аналіз отриманих логів дозволив розрахувати час виконання кожної операції для 5 тестувальників. Це забезпечило точність отриманих даних і автоматизацію процесу вимірювань.

Середній час виконання дій склав:

– вибір ролі користувача 1,5 секунди;

– пошук вільного паркомісця 2 секунди;

– отримання доступу до паркінгу 1,8 секунди;

– залишення відгуку про досвід користування 1,2 секунди;

– запит персональних акційних пропозицій (для мешканців) 1,7 секунди;

– отримання актуальної інформації про тарифи (для гостей) 1,3 секунди;

– запит на виїзд та оплату (для гостей) 2,5 секунди.

Подробиці часу виконання кожної дії для окремого тестувальника наведено у таблиці 5.1.

Таблиця 5.1 – Результати для 5 тестувальників

Тестувальник	Вибір ролі (с)	Пошук паркомісця (с)	Доступ до паркінгу (с)	Відгук (с)	Персональні пропозиції (с)	Тарифи (с)	Виїзд та оплата (с)
1	1	2	2	1	2	1	3
2	2	2	2	1	2	1	3
3	1	2	2	1	2	1	2
4	2	2	2	1	1	1	2
5	1	2	1	1	2	2	3

Під час виконання дій помилки у роботі системи не зафіксовано.

Середній рівень задоволеності користувачів за результатами оцінки:

– вибір ролі користувача 9,5 балів;

– пошук вільного паркомісця 9,3 балів;

– отримання доступу до паркінгу 9,7 балів;

– залишення відгуку 9,2 балів;

– запит персональних акційних пропозицій 9,4 балів;

– отримання інформації про тарифи 9,6 балів;

– запит на виїзд та оплату 9,1 балів.

5.4.3 Аналіз відповідності досліджень

Проведений експеримент показав, що розроблена система управління паркінгом ЖК MARSHALL відповідає заявленим технічним вимогам і очікуванням користувачів. Всі ключові функції, включаючи пошук вільних місць, бронювання, автоматичний контроль доступу та оплату, працюють стабільно та без збоїв. Система демонструє високу швидкість обробки запитів: середній час реакції на дії користувача склав менше двох секунд, що відповідає стандартам для інтерактивних сервісів.

Одним із важливих аспектів дослідження стала оцінка функціональності телеграм-бота. Він забезпечує простий і зручний інтерфейс, який легко

освоїли всі учасники експерименту. Учасники відзначили інтуїтивно зрозумілу навігацію, що дозволило виконувати , такі дії як бронювання місць чи отримання детальної інформації про тарифи, без попереднього навчання. Це свідчить про правильний підхід до дизайну.

Позитивний результат підтверджується також рівнем задоволеності користувачів. Усі учасники високо оцінили систему, надавши їй середню оцінку понад 9 балів за 10-бальною шкалою. Відгуки вказують на те, що бот ефективно вирішує повсякденні завдання користувачів і значно полегшує управління паркінгом.

Таким чином, система повністю відповідає технічним вимогам, забезпечуючи зручність і швидкість виконання операцій,. Експеримент підтвердив її готовність до впровадження в реальних умовах експлуатації, що дозволить підвищити рівень комфорту для мешканців ЖК MARSHALL.

5.4.4 Характеристика новизни результатів та шляхи розвитку системи

Результати проведеного дослідження мають значну новизну, що проявляється у кількох ключових аспектах. Насамперед, однією з головних інновацій є використання телеграм-бота як центрального інструмента взаємодії користувачів із системою. Бот не лише надає інформацію про доступні місця, а й дозволяє виконувати бронювання, отримувати індивідуальні пропозиції та залишати відгуки. Такий рівень інтеграції значно спрощує процес управління паркінгом та мінімізує потребу у фізичному контакті користувачів із системою, що особливо актуально в умовах цифровізації послуг.

Ще одним новим аспектом є автоматизація доступу до паркінгу, яка базується на взаємодії телеграм-бота із шлагбаумами. Це рішення дозволяє мешканцям за лічені секунди отримувати доступ до паркінгу або виходити з нього, що істотно скорочує час на виконання таких дій. Особливістю цієї системи є також можливість оперативного оновлення даних про зайнятість

місць, що гарантує актуальність інформації для користувачів.

Додатковою новизною є персоналізований підхід до роботи з користувачами. Для мешканців передбачено доступ до акційних пропозицій, а для гостей – зручний розрахунок вартості послуг із детальною інформацією про тривалість користування паркінгом. Це забезпечує індивідуалізований досвід взаємодії з системою, що підвищує рівень задоволеності користувачів.

Разом із тим, система має великий потенціал для подальшого розвитку. Вдосконалення телеграм-бота може включати розширення його функціональності. Наприклад, впровадження голосових команд або автоматичних нагадувань дозволить зробити користування системою ще більш інтуїтивним. Крім того, інтеграція з мобільними додатками та іншими платформами, такими як популярні месенджери, відкриє нові можливості для користувачів, які надають перевагу різним каналам зв’язку.

Подальший розвиток також може бути зосереджений на більш глибокій персоналізації сервісу. Використання алгоритмів аналізу поведінки користувачів дозволить пропонувати їм індивідуальні рекомендації та оптимізувати процес бронювання. Наприклад, бот може автоматично запропонувати найближче доступне місце або нагадати мешканцю про можливі акційні пропозиції.

З точки зору безпеки, система має перспективи для впровадження додаткових заходів захисту, таких як багаторівнева аутентифікація чи шифрування даних із підвищеним рівнем надійності. Це гарантує, що користувачі можуть бути впевнені у безпеці своїх персональних даних під час використання бота.

Загалом, новизна розробленої системи полягає не лише у її функціональних можливостях, але й у підході до інтеграції технологій. Подальше вдосконалення та розвиток дозволять зробити її ще більш гнучкою, безпечною та персоналізованою, що відкриває нові перспективи для підвищення зручності користувачів та ефективності управління паркінгом.

ВИСНОВКИ

Кваліфікаційна робота є завершеною науковою роботою, в якій вирішено науково-практичне завдання автоматизації управління паркінгом у ЖК MARSHALL шляхом розробки та впровадження телеграм-бота, що інтегрується з IoT-системами, для бронювання, контролю доступу та збору статистики.

Основні висновки і результати кваліфікаційної роботи полягають у наступному:

1. Розроблено інтелектуальну систему паркінгу для ЖК MARSHALL, що включає телеграм-бот для автоматизації бронювання паркомісць, збору статистики та управління доступом, що підвищує комфорт мешканців.

2. Проведено аналіз особливостей функціонування паркінгу ЖК MARSHALL, який дозволив сформулювати вимоги до апаратної та програмної складових системи. Зокрема, це забезпечило оптимізацію процесів взаємодії між користувачами, телеграм-ботом та IoT-компонентами.

3. Розроблено функціональний телеграм-бот для мешканців ЖК MARSHALL, який забезпечує зручне і просте управління паркомісцями. Бот дозволяє мешканцям бронювати місця для, автоматично відкривати шлагбаум при під’їзді, а також отримувати актуальну інформацію про наявність вільних місць у режимі реального часу. Крім того, система збирає статистичні дані, що дає змогу проводити аналіз використання паркінгу для подальшого підвищення його ефективності.

4. Під час розробки системи обрано технічну базу, яка забезпечує стабільну і швидку роботу компонентів навіть при значному навантаженні. Це дозволяє мінімізувати затримки у виконанні запитів, що критично для ефективного функціонування паркінгу житлового комплексу.

5. Тестування системи підтвердило її відповідність вимогам: скоротився час пошуку вільного місця, а доступність інформації підвищила комфорт користування паркінгом.

6. Проект відзначається впровадженням інтелектуальної комп’ютерної системи паркінгу ЖК MARSHALL, яка поєднує IoT-технології та телеграм-бот. Запропонована система автоматизує процеси управління паркомісцями, забезпечує збір статистики та контроль доступу в реальному часі. Особливістю є використання телеграм-бота як зручного інтерфейсу для мешканців, що спрощує взаємодію із системою та підвищує ефективність використання паркувального простору. Це рішення не лише оптимізує інфраструктуру ЖК, а й створює перспективи для розвитку автоматизованих систем управління.

7. Подальший розвиток системи може передбачати інтеграцію платіжної платформи для автоматизації процесу оплати за паркування, а також створення алгоритмів прогнозування, які допоможуть оптимізувати завантаженість паркінгу під час пікових годин. Це дозволить підвищити продуктивність системи та покращити користувацький досвід.

ПЕРЕЛІК ПОСИЛАНЬ

1. Методичні рекомендації до виконання кваліфікаційної роботи магістра галузі знань 12 інформаційні технології спеціальності 123 комп'ютерна інженерія/ Л.І. Цвіркун, В.В. Гнатушенко, С.М. Ткаченко. – Д.М.: НТУ«Дніпровська політехніка», 2022.- 58 с.

2. Цвіркун Л.І. Використання мессенджерів як системи оповіщення користувачів локальних систем домашнього інтернету речей. / Л.І. Цвіркун, Л.В. Бешта, Ю.А. Миронов // Системні технології. Зб. наук. пр. НМЕтАУ. − 2021. − № 4. − с. 95−101.

3. Цвіркун Л.І. Проблеми масштабованості в розподіленій обробці даних для туманних IT-інфраструктур / Цвіркун Л.І. Соболевський І.О. // Міжнародна науково-технічна конференція «Інформаційні технології в металургії та машинобудуванні – ІТММ'2024», 10 − 11 квітня. – Дніпро, УДУНТ, 2024. – С. 398-404. DOI: 10.34185/1991-7848.itmm.2024.01.076.

4. Traditional vs Automated Parking System [електронний ресурс] – Режим доступу: URL https://blog.getmyparking.com/2019/07/09/traditional-vs-automated-parking-system/

5. Multilevel Car Parking System Cost: A Sustainable Solution for Urban Parking Challenges [електронний ресурс] – Режим доступу: URL https://theautomatedparkingcompany.com/multilevel-car-parking-system-cost-a-sustainable-solution-for-urban-parking-challenges/

6. Automated Parking Systems Can Solve Urban Architecture and Mobility Challenges [електронний ресурс] – Режим доступу: URL https://www.roboticparking.com/automated-parking-systems-can-solve-urban-architecture-and-mobility-challenges/

7. Parking IoT Technologies: About Smart Parking Systems [електронний ресурс] – Режим доступу: URL https://www.digiteum.com/iot-smart-parking-solutions/

8. 10 innovative and sustainable parking solutions in overcrowded cities

[електронний ресурс] – Режим доступу: URL https://www.re-thinkingthefuture.com/city-and-architecture/a3366-10-innovative-and-sustainable-parking-solutions-in-overcrowded-cities/

9. Mathias Gabriel Diaz Ogás, Ramon Fabregat and Silvana Aciar: Survey of Smart Parking Systems. River Publishers: Institute of Computer Science, Faculty of Exact, Physical and Natural Sciences, National University of San Juan, San Juan 5400, Argentina

10. Інтелектуальна комп'ютерна система паркінгу ЖК «Маршал» з детальним опрацюванням побудови та налаштування корпоративної мережі [електронний ресурс] – Режим доступу: URL http://ir.nmu.org.ua/handle/123456789/164670

11. Parklio Smart Parking Solution [електронний ресурс] – Режим доступу: URL https://parklio.com

12. Cisco Smart+Connected Parking Systems [електронний ресурс] – Режим доступу: URL https://www.cisco.com

13. Smart Parking Ltd Solutions [електронний ресурс] – Режим доступу: URL https://www.smartparking.com

14. IoT-Based Smart Parking System: How to Create & What Do You Need? [електронний ресурс] – Режим доступу: URL https://sirinsoftware.com/blog/iot-based-smart-parking-system-how-to-create-what-do-you-need

15. A Cloud-Based Smart-Parking System Based on Internet-of-Things Technologies [електронний ресурс] – Режим доступу: URL https://ieeeaccess.ieee.org/featured-articles/cloud-based-smart-parking-system-based-internet-things-technologies/

16. Telegram Bot API Documentation [електронний ресурс] – Режим доступу: URL https://core.telegram.org/bots/api

17. Python sqlite3 Documentation [електронний ресурс] – Режим доступу: URL https://docs.python.org/3/library/sqlite3.html

18. Telegram Bot Tutorials by Python-telegram-bot [електронний ресурс]

– Режим доступу: URL https://github.com/python-telegram-bot/python-telegram-bot/wiki

19. Advanced SQLite Features in Python [електронний ресурс] – Режим доступу: URL https://realpython.com/python-sql-libraries/#sqlite

20. Telegram Bot Examples on GitHub [електронний ресурс] – Режим доступу: URL https://github.com/topics/telegram-bot

Додаток А

Програмне забезпечення телеграм-бота системи паркування ЖК MARSHALL

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ТЕЛЕГРАМ-БОТА СИСТЕМИ ПАРКУВАННЯ ЖК MARSHALL

Текст програми

804.02070743.24015-01 12 01

Листів 21

2024

АНОТАЦІЯ

Програма реалізує телеграм-бот для системи паркування ЖК MARSHALL. Бот активується командою /start та надає доступ до таких функцій, як вибір і бронювання місця, перегляд тарифів, звільнення місця та залишення відгуків.

Для гостей доступні команди для вибору місця, перегляду тарифів, а для мешканців – управління закріпленим місцем та персональні пропозиції. Бот збирає статистику використання паркомісць, яка оновлюється в базі даних.

Програма написана на мові програмування Python із використанням python-telegram-bot для обробки запитів та sqlite3 для збереження даних.

ЗМІСТ

1. Текст програми файлу main.py

```
import logging
from telegram.ext import ApplicationBuilder, CommandHandler
from handlers.start_handler import start
from handlers.resident_handler import resident_conversation_handler,
open_gate_handler
from handlers.guest_handler import guest_conversation_handler  # Імпорт
обробника для гілки "Гість"
from utils.data_utils import get_random_news  # Імпорт функції для вибірки
новин
import config

# Налаштування логування
logging.basicConfig(
  format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
level=logging.INFO
)
logger = logging.getLogger(__name__)

async def send_random_news(context):
  """Функція для відправки випадкової новини."""
  news = get_random_news()
  active_chat_ids = context.bot_data.get("active_chat_ids", set())

  for chat_id in active_chat_ids:
    await context.bot.send_message(chat_id=chat_id, text=f"📰 Новина: {news}")
    logger.info(f"Новина надіслана в чат {chat_id}.")

def main():
  app = ApplicationBuilder().token(config.BOT_TOKEN).build()

  # Реєстрація обробників команд
  app.add_handler(CommandHandler("start", start))                # Обробник
команди /start
  app.add_handler(resident_conversation_handler)                 # Обробник для
мешканця
  app.add_handler(guest_conversation_handler)                   # Обробник для
гостя
  app.add_handler(CommandHandler("open_gate", open_gate_handler))     #
Обробник відкриття шлагбаума
```

```
    # Фонове завдання для новин
    job_queue = app.job_queue
    job_queue.run_repeating(send_random_news, interval=1800, first=10)

    logger.info("Bot is running in test mode...")
    app.run_polling()

if __name__ == "__main__":
    main()
```

2. Текст програми файлу handlers/guest_handler.py

```
import logging
import sqlite3
import pandas as pd
from telegram import Update, ReplyKeyboardMarkup, ReplyKeyboardRemove
from telegram.ext import (
    ContextTypes,
    ConversationHandler,
    CommandHandler,
    MessageHandler,
    filters
)
import datetime
import os
from database import save_feedback

logger = logging.getLogger(__name__)

# Визначення станів
SELECT_ACTION, SECTOR_SELECTION, SPOT_SELECTION, INPUT_CAR_NUMBER,
CHECKOUT_CAR_NUMBER, CHECKOUT, FEEDBACK_TEXT, EXIT_CONFIRMATION = range(8)

# Підключення до бази даних і таблиць
def init_db():
    conn = sqlite3.connect('parking.db')
    cursor = conn.cursor()

    # Основна таблиця для паркомісць
    cursor.execute('''CREATE TABLE IF NOT EXISTS parking_spots (
                id INTEGER PRIMARY KEY,
                status TEXT,
```

```
                    car_number TEXT,
                    booking_time TEXT)''')

        # Таблиця для зберігання статистики
        cursor.execute('''CREATE TABLE IF NOT EXISTS booking_statistics (
                    spot_id INTEGER PRIMARY KEY,
                    booking_count INTEGER DEFAULT 0,
                    total_duration REAL DEFAULT 0.0
                )''')

        # Ініціалізація таблиць, якщо ще не створено
        for i in range(1, 51):
            cursor.execute('INSERT OR IGNORE INTO parking_spots (id, status) VALUES (?, "вільне")', (i,))
            cursor.execute('INSERT OR IGNORE INTO booking_statistics (spot_id) VALUES (?)', (i,))

        conn.commit()
        conn.close()

    async def show_guest_menu(update: Update) -> int:
        # Меню для гостя з використанням клавіатури
        guest_menu_keyboard = [
            ["Забронювати місце", "Переглянути тарифи"],
            ["Виїзд та оплата", "Залишити відгук"],
            ["Вийти в початкове меню"]
        ]
        reply_markup = ReplyKeyboardMarkup(guest_menu_keyboard, resize_keyboard=True, one_time_keyboard=True)
        await update.message.reply_text(
            "Оберіть дію:",
            reply_markup=reply_markup
        )
        logger.info("Гість увійшов у початкове меню.")
        return SELECT_ACTION

    async def guest(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
        # Початкове меню для гостя
        return await show_guest_menu(update)

    async def book(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
        # Запит вибору сектора для бронювання.
```

```
            keyboard = [["Сектор 1-25", "Сектор 26-50"]]
            reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True,
one_time_keyboard=True)
            await update.message.reply_text("Оберіть сектор для бронювання:",
reply_markup=reply_markup)
            return SECTOR_SELECTION

        async def view_tariffs(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
int:
            # Відображає інформацію про тарифи паркування.
            await update.message.reply_text(
              "Тарифи на паркування:\n"
              "Місця 1–25: 10 грн/год\n"
              "Місця 26–50: 15 грн/год"
            )
            logger.info("Перегляд тарифів.")
            return await show_guest_menu(update)  # Повернення до меню гостя

        async def select_sector(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
int:
            # Обробляє вибір сектора для бронювання.
            sector_choice = update.message.text
            logger.info(f"Обраний сектор: {sector_choice}")

            if sector_choice == "Сектор 1-25":
              selected_spots = fetch_available_spots(1, 25)
            elif sector_choice == "Сектор 26-50":
              selected_spots = fetch_available_spots(26, 50)
            else:
              await update.message.reply_text("Невірний вибір сектора. Будь ласка,
оберіть сектор знову.")
              return SECTOR_SELECTION

            if not selected_spots:
              other_sector = "Сектор 26-50" if sector_choice == "Сектор 1-25" else "Сектор
1-25"
              await update.message.reply_text(
                f"Немає доступних місць у секторі {sector_choice}.\n"
                f"Чи хочете переглянути інший сектор ({other_sector})?",
                reply_markup=ReplyKeyboardMarkup([[other_sector, "Скасувати
бронювання"]], resize_keyboard=True, one_time_keyboard=True)
              )
```

```
            return SECTOR_SELECTION

        keyboard = [[f"Місце {spot}" for spot in selected_spots[i:i+5]] for i in range(0,
len(selected_spots), 5)]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True,
one_time_keyboard=True)
        await update.message.reply_text("Оберіть місце для бронювання:",
reply_markup=reply_markup)
        context.user_data["sector"] = sector_choice
        return SPOT_SELECTION

    def fetch_available_spots(start, end):
        # Отримує доступні місця в заданому діапазоні з бази даних.
        conn = sqlite3.connect('parking.db')
        cursor = conn.cursor()
        cursor.execute("SELECT id FROM parking_spots WHERE status = 'вільне' AND id
BETWEEN ? AND ?", (start, end))
        available_spots = [row[0] for row in cursor.fetchall()]
        conn.close()
        return available_spots

    async def confirm_spot(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
int:
        # Підтвердження вибраного місця для бронювання.
        spot_text = update.message.text.strip()
        spot_id = int(spot_text.split()[1])
        logger.info(f"Вибране місце: {spot_id}")

        if reserve_spot(spot_id):
            context.user_data["selected_spot"] = spot_id
            await update.message.reply_text("Місце вибрано. Введіть номер вашої
машини:", reply_markup=ReplyKeyboardRemove())
            return INPUT_CAR_NUMBER
        else:
            await update.message.reply_text("Це місце вже заброньоване. Оберіть
інше.")
            return SPOT_SELECTION

    def reserve_spot(spot_id):
        # Резервує місце в базі даних
        conn = sqlite3.connect('parking.db')
        cursor = conn.cursor()
```

```
            cursor.execute("SELECT status FROM parking_spots WHERE id = ?", (spot_id,))
            status = cursor.fetchone()

            if status and status[0] == "вільне":
                cursor.execute("UPDATE parking_spots SET status = 'заброньоване' WHERE id
= ?", (spot_id,))
                conn.commit()
                conn.close()
                return True
            conn.close()
            return False

        async def input_car_number(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
            car_number = update.message.text.strip()
            spot_id = context.user_data.get("selected_spot")

            conn = sqlite3.connect('parking.db')
            cursor = conn.cursor()
            booking_time = datetime.datetime.now()
            cursor.execute("UPDATE parking_spots SET status = 'заброньоване', car_number
= ?, booking_time = ? WHERE id = ?",
                           (car_number, booking_time.strftime("%Y-%m-%d %H:%M:%S"), spot_id))

            cursor.execute("UPDATE booking_statistics SET booking_count = booking_count +
1 WHERE spot_id = ?", (spot_id,))
            conn.commit()
            conn.close()

            context.user_data["car_number"] = car_number
            context.user_data["booking_time"] = booking_time
            await update.message.reply_text(f"Місце {spot_id} заброньовано для машини
{car_number}. Шлагбаум піднято.")
            return await show_guest_menu(update)

        async def checkout_car_number(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
            await update.message.reply_text("Введіть номер машини для виїзду:")
            return CHECKOUT

        async def checkout(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
            car_number = update.message.text.strip()
```

```
            conn = sqlite3.connect('parking.db')
            cursor = conn.cursor()
            cursor.execute("SELECT id, booking_time FROM parking_spots WHERE
car_number = ? AND status = 'заброньоване'", (car_number,))
            result = cursor.fetchone()

            if not result:
                await update.message.reply_text("Бронювання не знайдено для вказаного
номера машини.")
                conn.close()
                return await show_guest_menu(update)

            spot_id, booking_time = result
            booking_time = datetime.datetime.strptime(booking_time, "%Y-%m-%d
%H:%M:%S")
            duration_hours = (datetime.datetime.now() - booking_time).total_seconds() /
3600
            rate = 10 if spot_id <= 25 else 15
            amount_due = round(duration_hours * rate, 2)

            await update.message.reply_text(
                f"Ваше перебування: {duration_hours:.2f} годин.\n"
                f"Сума до оплати: {amount_due} грн."
            )
            await update.message.reply_text("Оплата прийнята. Дякуємо за користування
паркінгом! Шлагбаум піднято.")

            cursor.execute("UPDATE parking_spots SET status = 'вільне', car_number = NULL,
booking_time = NULL WHERE id = ?", (spot_id,))
            cursor.execute("UPDATE booking_statistics SET total_duration = total_duration +
? WHERE spot_id = ?", (duration_hours, spot_id))
            conn.commit()
            conn.close()
            return await show_guest_menu(update)

        async def feedback(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
            await update.message.reply_text("Будь ласка, залиште свій відгук.")
            logger.info("Очікуємо на ввід відгуку від гостя.")
            return FEEDBACK_TEXT

        async def handle_feedback(update: Update, context: ContextTypes.DEFAULT_TYPE)
-> int:
```

```
    # Обробляє текст відгуку і зберігає його у базу даних через `save_feedback`.
    feedback_text = update.message.text
    save_feedback(update.message.from_user.id, feedback_text)
    await update.message.reply_text("Дякуємо за ваш відгук!")
    logger.info(f"Відгук від гостя: {feedback_text}")
    return await show_guest_menu(update)


async def exit_guest(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    reply_keyboard = [["✅ Вийти", "🔙 До початку"]]
    await update.message.reply_text(
      "Ви дійсно хочете вийти з меню гостя?\n"
      "Ви можете повернутися до початкового меню.",
      reply_markup=ReplyKeyboardMarkup(reply_keyboard, resize_keyboard=True,
one_time_keyboard=True)
    )
    return EXIT_CONFIRMATION

async def confirm_exit_guest(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    response = update.message.text
    if response == "✅ Вийти":
      await update.message.reply_text("Дякуємо за користування ботом! Гарного
дня!", reply_markup=ReplyKeyboardRemove())
      return ConversationHandler.END
    elif response == "🔙 До початку":
      return await show_guest_menu(update)
    else:
      await update.message.reply_text("Будь ласка, виберіть одну з опцій.")
      return EXIT_CONFIRMATION

# Обробник розмови з гостем
guest_conversation_handler = ConversationHandler(
    entry_points=[CommandHandler("guest", guest)],
    states={
      SELECT_ACTION: [
        MessageHandler(filters.Regex("^Забронювати місце$"), book),
        MessageHandler(filters.Regex("^Переглянути тарифи$"), view_tariffs),
        MessageHandler(filters.Regex("^Виїзд та оплата$"), checkout_car_number),
        MessageHandler(filters.Regex("^Залишити відгук$"), feedback),
        MessageHandler(filters.Regex("^Вийти в початкове меню$"), exit_guest),
      ],
```

```
            SECTOR_SELECTION: [MessageHandler(filters.TEXT & ~filters.COMMAND,
select_sector)],
            SPOT_SELECTION: [MessageHandler(filters.TEXT & ~filters.COMMAND,
confirm_spot)],
            INPUT_CAR_NUMBER: [MessageHandler(filters.TEXT & ~filters.COMMAND,
input_car_number)],
            CHECKOUT: [MessageHandler(filters.TEXT & ~filters.COMMAND, checkout)],
            FEEDBACK_TEXT: [MessageHandler(filters.TEXT & ~filters.COMMAND,
handle_feedback)],
            EXIT_CONFIRMATION: [MessageHandler(filters.TEXT & ~filters.COMMAND,
confirm_exit_guest)],
        },
        fallbacks=[CommandHandler("cancel", exit_guest)],
    )

    init_db()
```

3. Текст програми файлу handlers/resident_handler.py

```
    import logging
    from telegram import Update, ReplyKeyboardMarkup, ReplyKeyboardRemove
    from telegram.ext import ContextTypes, ConversationHandler, CommandHandler,
MessageHandler, filters
    from utils.data_utils import (
        load_residents,
        get_random_news,
        load_personal_offer,
        open_gate,
        load_parking_spots,
        get_parking_status,
        update_parking_status
    )
    from database import save_feedback
    from handlers.start_handler import start  # Імпорт початкового обробника

    logger = logging.getLogger(__name__)

    # Стан для команди "Вийти"
    CAR_NUMBER, FEEDBACK_TEXT, EXIT_CONFIRMATION = range(3)

    # Завантаження даних
    residents = load_residents()        # Дані мешканців
    parking_spots = load_parking_spots() # Дані паркомісць
```

```
        MAX_ATTEMPTS = 10          # Максимальна кількість спроб введення номера машини

        async def resident(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
            context.user_data["attempts"] = 0  # Ініціалізація лічильника спроб
            await update.message.reply_text("Введіть номер машини:")
            logger.info("Очікуємо на ввід номера машини від мешканця.")
            return CAR_NUMBER

        async def handle_car_number(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
            car_number = update.message.text.strip()  # Отримання номера машини
            logger.info(f"Отримано номер машини: {car_number}")

            # Перевірка номера машини
            if car_number not in residents:
                context.user_data["attempts"] += 1
                attempts_left = MAX_ATTEMPTS - context.user_data["attempts"]

                if context.user_data["attempts"] >= MAX_ATTEMPTS:
                    await update.message.reply_text("На жаль, ви вичерпали всі спроби. Будь ласка, спробуйте пізніше.")
                    logger.warning(f"Мешканець не зміг ввести правильний номер машини за {MAX_ATTEMPTS} спроб.")
                    return ConversationHandler.END

                await update.message.reply_text(
                    f"Номер машини не знайдено серед мешканців. Спробуйте ще раз.\n"
                    f"Залишилось спроб: {attempts_left}"
                )
                logger.warning(f"Невірний номер машини: {car_number}. Залишилось спроб: {attempts_left}")
                return CAR_NUMBER

            resident_name = residents[car_number]
            await update.message.reply_text(f"Вітаємо, {resident_name}!")
            logger.info(f"Мешканець {resident_name} успішно пройшов перевірку за номером машини {car_number}.")

            # Виведення команд для мешканця у вигляді блоку кнопок
            resident_menu_keyboard = [
                ["Статус", "В'їзд"],
```

```
                ["Виїзд", "Відгук"],
                ["Персональна пропозиція", "Вийти"]
            ]
            reply_markup = ReplyKeyboardMarkup(resident_menu_keyboard,
resize_keyboard=True)

            await update.message.reply_text(
                "Доступні команди:",
                reply_markup=reply_markup
            )
            context.user_data['car_number'] = car_number
            return CAR_NUMBER

        async def check_status(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
None:
            car_number = context.user_data.get('car_number')
            status = get_parking_status(parking_spots, car_number)
            await update.message.reply_text(f"Статус вашого місця: {status}")
            logger.info(f"Перевірено статус місця для {car_number}.")

        async def open_gate_handler(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
            car_number = context.user_data.get('car_number')
            await update.message.reply_text(open_gate())
            update_parking_status(parking_spots, car_number, "Зайнято")  # Змінюємо
статус на "Зайнято"
            logger.info(f"Шлагбаум відкрито для мешканця {car_number} і статус
встановлено на 'Зайнято'.")

        async def leave_parking(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
None:
            car_number = context.user_data.get('car_number')
            if update_parking_status(parking_spots, car_number, "Вільно"):
                await update.message.reply_text("Ви виїхали з паркомісця. Статус змінено
на 'Вільно'.")
                logger.info(f"Встановлено статус 'Вільно' для {car_number}.")
            else:
                await update.message.reply_text("Не вдалося оновити статус паркомісця.")
                logger.warning(f"Не вдалося оновити статус для {car_number}.")

        async def feedback(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
            await update.message.reply_text("Будь ласка, залиште свій відгук.")
```

```
        logger.info("Очікуємо на ввід відгуку від мешканця.")
        return FEEDBACK_TEXT

    async def handle_feedback(update: Update, context: ContextTypes.DEFAULT_TYPE)
-> int:
        feedback_text = update.message.text
        car_number = context.user_data.get('car_number', 'невідомо')
        save_feedback(update.message.from_user.id, feedback_text)
        await update.message.reply_text("Дякуємо за ваш відгук!")
        logger.info(f"Отримано відгук від мешканця {car_number}: {feedback_text}")
        return CAR_NUMBER

    async def show_personal_offer(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
        car_number = context.user_data.get('car_number')
        personal_offer = load_personal_offer(car_number)
        await update.message.reply_text(f"Персональна
пропозиція:\n{personal_offer}")
        logger.info(f"Показано персональну пропозицію для {car_number}.")
        return CAR_NUMBER

    async def exit_menu(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
        """Запит підтвердження виходу з гілки 'Мешканець'."""
        reply_keyboard = [["✅ Вийти", "🔙 До початку"]]
        await update.message.reply_text(
            "Ви дійсно хочете вийти з меню мешканця?\n"
            "Ви можете повернутися до початкового меню.",
            reply_markup=ReplyKeyboardMarkup(reply_keyboard, resize_keyboard=True,
one_time_keyboard=True)
        )
        logger.info("Запит на підтвердження виходу.")
        return EXIT_CONFIRMATION

    async def confirm_exit(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
int:
        """Підтвердження виходу до стартового меню через команду /start."""
        response = update.message.text
        if response == "✅ Вийти":
            await update.message.reply_text("Дякуємо за користування ботом! Гарного
дня!", reply_markup=ReplyKeyboardRemove())
            logger.info("Користувач підтвердив вихід з гілки 'Мешканець'.")
            return ConversationHandler.END  # Завершуємо розмову без повернення до
```

```
        початкового меню
        elif response == "⮌ До початку":
            return await resident(update, context)  # Повернення в гілку мешканця
        else:
            await update.message.reply_text("Будь ласка, виберіть одну з опцій.")
            return EXIT_CONFIRMATION

    # Обробник розмови з мешканцем
    resident_conversation_handler = ConversationHandler(
        entry_points=[CommandHandler("resident", resident)],
        states={
            CAR_NUMBER: [
                MessageHandler(filters.Regex("^Статус$"), check_status),
                MessageHandler(filters.Regex("^В'їзд$"), open_gate_handler),  # В'їзд на
паркомісце
                MessageHandler(filters.Regex("^Виїзд$"), leave_parking),     # Виїзд з
паркомісця
                MessageHandler(filters.Regex("^Відгук$"), feedback),
                MessageHandler(filters.Regex("^Персональна пропозиція$"),
show_personal_offer),
                MessageHandler(filters.Regex("^Вийти$"), exit_menu),
                MessageHandler(filters.TEXT & ~filters.COMMAND, handle_car_number)
            ],
            FEEDBACK_TEXT: [MessageHandler(filters.TEXT & ~filters.COMMAND,
handle_feedback)],
            EXIT_CONFIRMATION: [MessageHandler(filters.Regex("^(✅ Вийти|⮌ До
початку)$"), confirm_exit)],
        },
        fallbacks=[CommandHandler("cancel", exit_menu)],
    )
```

4. Текст програми файлу handlers/start_handler.py

```
import logging
from telegram import Update
from telegram.ext import ContextTypes, CommandHandler

# Налаштування логування
logger = logging.getLogger(__name__)

# Обробник для команди /start
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
```

```
    """
    Відображає стартове привітання користувачам та пропонує вибір профілю.

    - Зберігає chat_id активного користувача у боті.
    - Надсилає повідомлення із логотипом та списком команд.
    """
    # Додати chat_id до active_chat_ids (для відстеження активних користувачів)
    chat_id = update.effective_chat.id
    active_chat_ids = context.bot_data.get("active_chat_ids", set())
    active_chat_ids.add(chat_id)
    context.bot_data["active_chat_ids"] = active_chat_ids
    logger.info(f"Додано chat_id {chat_id} до active_chat_ids.")

    # Шлях до зображення логотипу
    logo_path = 'logo.png'

    # Текст привітання та вибір профілю
    welcome_text = (
        "👋 Ласкаво просимо до Паркінг-бота!\n\n"
        "Будь ласка, оберіть свій профіль, скориставшись командами:\n\n"
        "- 🔑 /resident - Мешканець\n"
        "- 🚗 /guest - Гість\n\n"
        "🌐 [Сайт забудовника](https://marshall.daytona.com.ua)"
    )

    # Відправка зображення разом з текстом
    try:
        # Спроба надіслати логотип
        with open(logo_path, 'rb') as logo:
            await context.bot.send_photo(
                chat_id=chat_id,
                photo=logo,
                caption=welcome_text,
                parse_mode='Markdown'
            )
        logger.info("Привітання з логотипом надіслано.")
    except FileNotFoundError:
        # Логотип не знайдено, надсилаємо лише текст
        logger.error(f"Файл {logo_path} не знайдено.")
        await update.message.reply_text(welcome_text, parse_mode='Markdown')

# Створення обробника для команди /start
```

```
start_handler = CommandHandler("start", start)
```

5. Текст програми файлу utils/data_utils.py

```
import pandas as pd
import os
import logging
import random

logger = logging.getLogger(__name__)

RESIDENTS_FILE = 'residents.xlsx'
NEWS_FILE = 'news.xlsx'
OFFERS_FOLDER = 'offers'
PARKING_SPOTS_FILE = 'parking_spots.xlsx'

def load_residents():
  """Завантаження даних мешканців з файлу Excel."""
  if not os.path.exists(RESIDENTS_FILE):
    raise FileNotFoundError(f"Файл {RESIDENTS_FILE} не знайдено.")

  residents_data = pd.read_excel(RESIDENTS_FILE)
  logger.info(f"Завантажено файл {RESIDENTS_FILE} з колонками:
{residents_data.columns.tolist()}")

  # Перевірка, чи існують потрібні колонки
  if 'Номер машини' not in residents_data.columns or 'ПІБ' not in
residents_data.columns:
    raise KeyError("У файлі не знайдено колонок 'Номер машини' або 'ПІБ'.
Перевірте заголовки колонок.")

  residents = {
    row['Номер машини']: row['ПІБ']
    for _, row in residents_data.iterrows()
  }
  return residents

def get_random_news():
  """Отримання випадкової новини з файлу Excel."""
  if not os.path.exists(NEWS_FILE):
    logger.error(f"Файл {NEWS_FILE} не знайдено.")
    return "Новин поки немає."  # Повертаємо повідомлення, якщо файл
відсутній
```

```
            news_data = pd.read_excel(NEWS_FILE)
            news_list = news_data['Новини'].dropna().tolist()  # Отримуємо всі новини без порожніх рядків

            if not news_list:
                logger.warning("Новини відсутні у файлі.")
                return "Новин поки немає."

            random_news = random.choice(news_list)  # Вибираємо випадкову новину
            logger.info(f"Випадкова новина: {random_news}")
            return random_news

        def load_personal_offer(car_number):
            """Завантаження персональної пропозиції для мешканця за номером машини."""
            offer_file = os.path.join(OFFERS_FOLDER, f"{car_number}.txt")
            if not os.path.exists(offer_file):
                return "Персональних пропозицій немає."

            with open(offer_file, 'r', encoding='utf-8') as file:
                personal_offer = file.read().strip()
            logger.info(f"Персональна пропозиція для {car_number}: {personal_offer}")
            return personal_offer

        def open_gate():
            """Симуляція відкриття шлагбаума для тестового режиму."""
            logger.info("Шлагбаум відкрито для тестового режиму.")
            return "Шлагбаум відкрито!"

        def load_parking_spots():
            """Завантаження даних паркомісць з файлу Excel у словник."""
            if not os.path.exists(PARKING_SPOTS_FILE):
                raise FileNotFoundError(f"Файл {PARKING_SPOTS_FILE} не знайдено.")

            parking_data = pd.read_excel(PARKING_SPOTS_FILE)
            logger.info(f"Завантажено паркомісця з файлу {PARKING_SPOTS_FILE}")
            parking_spots = {
                row["Номер машини"]: {"place_id": row["Паркомісце"], "status": row["Статус"]}
                for _, row in parking_data.iterrows()
            }
```

```
    return parking_spots

def get_parking_status(parking_spots, car_number):
    """Отримує статус паркомісця для вказаного номера машини."""
    place_info = parking_spots.get(car_number)
    return place_info["status"] if place_info else "Невідоме місце"

def update_parking_status(parking_spots, car_number, new_status):
    """Оновлює статус паркомісця у файлі Excel для вказаного номера машини."""
    if car_number in parking_spots:
        parking_spots[car_number]["status"] = new_status
        save_parking_spots(parking_spots)
        return True
    return False

def save_parking_spots(parking_spots):
    """Зберігає оновлений статус паркомісць у файл Excel."""
    df = pd.DataFrame([
        {"Номер машини": car, "Паркомісце": info["place_id"], "Статус": info["status"]}
        for car, info in parking_spots.items()
    ])
    df.to_excel(PARKING_SPOTS_FILE, index=False)
```

6. Текст програми файлу database.py

```
import sqlite3
from datetime import datetime

# Функція для ініціалізації бази даних
def init_feedback_db():
    conn = sqlite3.connect('feedback.db')
    cursor = conn.cursor()

    # Створюємо таблицю для зберігання відгуків, якщо вона ще не існує
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS feedback (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER,
            feedback TEXT,
            timestamp TEXT
        )
```

```
    ''')

    conn.commit()
    conn.close()
    print("База даних для відгуків ініціалізована.")

# Функція для збереження відгуку в базу даних
def save_feedback(user_id, feedback):
    """
    Зберігає відгук користувача в базу даних.

    Параметри:
    - user_id: ID користувача
    - feedback: Текст відгуку
    """
    # Ініціалізуємо базу даних, якщо вона ще не була створена
    init_feedback_db()

    conn = sqlite3.connect('feedback.db')
    cursor = conn.cursor()

    # Збереження нового відгуку з поточною датою та часом
    cursor.execute(
        'INSERT INTO feedback (user_id, feedback, timestamp) VALUES (?, ?, ?)',
        (user_id, feedback, datetime.now().strftime('%Y-%m-%d %H:%M:%S'))
    )

    conn.commit()
    conn.close()
    print(f"Відгук збережено для користувача з ID {user_id}")
```

7. Текст програми файлу config.py

```
# Telegram Bot Token
BOT_TOKEN = "7927229024:AAHntPx7-NnsypclfL7A5c1SPbo9JPjRAvU"
```