Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка Telegram-бота для системи управління служби таксі»

Виконав: студент ___4___ курсу
групи __1ПІ-20Б_____спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Юрченко А.О.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Романюк О.В.
(прізвище та ініціали)
Рецензент:. к.ф.-м.н., доц. каф. ЗІ Шелепало Г.В.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н, проф. Романюк О.Н.
« _10_ » ___червня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 12 » березня 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Юрченку Антону Олександровичу

1. Тема роботи – розробка Telegram-бота для системи управління служби таксі.
Керівник роботи: Романюк Оксана Володимирівна, к.т.н. доцент кафедри ПЗ затверджені наказом вищого навчального закладу від 11 березня 2024 р. №80.

2. Строк подання студентом роботи 3 червня 2024.

3. Вихідні дані до роботи: тип програми – Telegram-бот; модель розробки – ітеративна; засоби організації даних програми – фреймворк Spring та бібліотека TelegramAPI; вхідні дані: команда користувача, початкова та кінцева координата поїздки, повідомлення між користувачами; середовище розробки – Intellij IDEA; мова програмування – Java.

4. Зміст текстової частини: вступ; обґрунтування вибору методу розробки та постановка задач дослідження; розробка моделі, методу та алгоритмів роботи програмного продукту; розробка програмних модулів реалізації telegram-бота для системи управління служби таксі; тестування програми; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження; новизна і практична цінність отриманих результатів; порівняльний аналіз аналогів; діаграма класів; загальний алгоритм роботи Telegram-бота; метод і блок-схема алгоритму автоматичного призначення замовлень водіям; метод і блок-схема алгоритму оптимізації процесу видачі замовлень; меню авторизації застосунку; вибір

стартової кінцевої позиції маршруту; фінальний етап замовлення; тестування програми; стек інформаційних технологій для реалізації програмної частини системи; апробація та публікація матеріалів бакалаврської дипломної роботи; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Романюк О.В., к.т.н., доцент кафедри ПЗ	13.03.24	03.06.24

7. Дата видачі завдання 13 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Ч.ч.	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання та постановка задач дослідження	14.03.24 – 17.03.24	виконано
2	Розробка структури та моделі застосунку	18.03.24 – 28.03.24	виконано
3	Розробка загального алгоритму роботи програми	29.03.24 – 01.04.24	виконано
4	Розробка методу та алгоритму видачі замовлень водіям	02.04.24 – 04.04.24	виконано
5	Розробка методу та алгоритму автоматичного призначення замовлень водіям	05.04.24 – 07.04.24	виконано
6	Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення	08.04.24 – 12.04.24	виконано
7	Розробка інтерфейсу програмного застосунку	13.04.24 – 15.04.24	виконано
8	Програмна реалізація Telegram-боту	16.04.24 – 12.05.24	виконано
9	Тестування програмного застосунку	13.05.24 – 20.05.24	виконано
10	Оформлення матеріалів до захисту БДР	21.05.24 – 30.05.24	виконано

Студент _____ **А. О. Юрченко**
(підпис) (ініціали та прізвище)

Керівник бакалаврської дипломної роботи _____ **О. В. Романюк**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 53 сторінок формату А4, на яких є 28 рисунків, 6 таблиць, список використаних джерел містить 20 найменувань.

У бакалаврській дипломній роботі було визначено доцільність, практичну цінність та мету розробки, яка полягає в полегшення процесів замовлення та керування таксі через популярний месенджер Telegram. Вперше запропоновано метод автоматичного призначення замовлень водіям, у якому враховується їхня доступність та рейтинг, що дозволило підвищити ефективність розподілу замовлень та задоволення клієнтів. Удосконалено метод оптимізації процесу видачі замовлень водіям, який на відміну від відомих, дає водіям можливість переглянути список активних замовлень в невеликому радіусі від водія та пропонує брати замовлення, які довше знаходяться без відповіді, що дозволило підвищити ефективність та швидкість реагування на замовлення клієнтів.

Програмне забезпечення розроблено із використанням середовища розробки Intellij IDEA та мови програмування Java. Під час розробки було використано сторонні бібліотеки, зокрема Telegram API, Liquibase. У результаті виконання бакалаврської дипломної роботи розроблено програмний засіб, працездатність і правильність роботи якого перевірено, підготовлена інструкція користувача та визначено системні вимоги.

Отримані в бакалаврській дипломній роботі результати можна використати для простого замовлення таксі за допомогою Telegram-бота.

Ключові слова: телеграм бот, користувач, java, телеграм, замовлення.

ABSTRACT

The bachelor's thesis consists of 53 A4 pages, which have 28 figures, 6 tables, the list of sources used contains 20 items.

The bachelor's thesis identifies the relevance, practical value, and purpose of development aimed at streamlining taxi ordering and management processes through the popular messenger Telegram. For the first time, a method for automatic assignment of orders to drivers has been proposed, taking into account their availability and rating, which has improved the efficiency of order distribution and customer satisfaction. The method for optimizing the process of assigning orders to drivers has been improved. Unlike known methods, it allows drivers to view a list of active orders within a small radius from the driver and suggests taking orders that have been pending longer, which has increased the efficiency and speed of responding to customer orders.

The software was developed using the Intellij IDEA development environment and Java programming language. Third-party libraries were utilized during development, including Telegram API and Liquibase. As a result of the bachelor's thesis, a functional software tool was created, its performance and correctness verified, a user manual prepared, and system requirements defined.

The outcomes of the bachelor's thesis can be applied for simple taxi ordering through the Telegram bot.

Keywords: Telegram bot, user, java, telegram, order.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Таксі є одним із найпопулярніших та найзручніших засобів транспорту в містах. Популярність таксі зумовлена необхідністю швидкого та комфортного переміщення, особливо в умовах трафіку та зайнятості сучасних людей. Використання таксі є частиною повсякденного життя для багатьох мешканців міст.

В сучасному світі технології месенджерів стають все більш популярними серед користувачів мобільних пристроїв [1].

З розвитком технологій та зростанням популярності месенджерів, таких як Telegram, месенджери стають все більш популярними та важливими засобами комунікації. Вони дозволяють не лише обмінюватися повідомленнями, а й використовувати різноманітні сервіси та додатки, включаючи інтеграцію з різними сервісами.

Telegram-боти – це програмні рішення, які дозволяють взаємодіяти з користувачами через популярний месенджер Telegram [2]. Їх переваги включають простоту використання, доступність на різних платформах та можливість створення персоналізованих сервісів для кінцевих користувачів.

Поява Telegram-ботів для різноманітних потреб споживачів стала не лише тенденцією, а й необхідністю. Однак, попри широкий спектр ботів, які можуть допомогти у різних сферах, наразі відсутні рішення для системи управління служби таксі. Сучасні інструменти часто не враховують потреби цього сегмента, що створює значні труднощі у забезпеченні ефективного управління замовленнями, оптимізації маршрутів та зручної взаємодії з користувачами та водіями.

Розробка програмних модулів для Telegram-бота у сфері таксі є логічним кроком у напрямку інтеграції нових технологій у традиційні сфери обслуговування. Впровадження Telegram-бота для замовлення таксі стає актуальним завданням, оскільки це дозволить забезпечити зручність та доступність таксі через популярний месенджер, що використовується

мільйонами користувачів.

Таким чином, об'єднання популярності таксі як засобу транспорту та зростаючої популярності месенджерів, зокрема Telegram, відкриває шлях до новаторських рішень у сфері обслуговування пасажирів. Розробка Telegram-бота для управління таксі відповідає сучасним тенденціям у розвитку інформаційних технологій та може відкрити нові перспективи у сфері транспортних послуг.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є покращення процесу управління служби таксі шляхом розроблення спеціалізованого Telegram-бота.

Відповідно до поставленої мети потрібно вирішити такі **завдання**:

– здійснити аналіз стану питання та методів розв’язання задачі;
– спроектувати модель даних та структуру Telegram-бота;
– розробити метод і алгоритм оптимізації процесу видачі замовлень водіям;
– розробити метод і алгоритм автоматичного призначення замовлень водіям;
– розробити загальний алгоритм роботи програми;
– розробити графічний інтерфейс для зручної взаємодії з користувачами;
– програмно реалізувати модуль Telegram-бота, який забезпечить взаємодію з користувачами та обробку їх запитів;
– провести тестування програмного застосунку;
– розробити інструкцію користувача та описати системні вимоги програми для відстеження читання.

Об’єктом дослідження є процес управління служби таксі.

Предметом дослідження є методи та засоби реалізації програмного забезпечення для розробки ефективного Telegram-бота для управління служби таксі.

Методи дослідження. У процесі проведення дослідження

використовувалися різні методи, зокрема порівняльний аналіз програм-аналогів для виявлення їх недоліків і постановки завдань розробки; застосування теорії алгоритмів для створення та візуалізації програмного забезпечення; комп'ютерне моделювання для аналізу та перевірки теоретичних концепцій.

Новизна отриманих результатів.

1. Удосконалено метод оптимізації процесу видачі замовлень водіям, який на відміну від відомих, дає водіям можливість переглянути список активних замовлень в невеликому радіусі від водія та пропонує брати замовлення, які довше знаходяться без відповіді, що дозволило підвищити ефективність та швидкість реагування на замовлення клієнтів.

2. Вперше запропоновано метод автоматичного призначення замовлень водіям, у якому враховується їхня доступність та рейтинг, що дозволило підвищити ефективність розподілу замовлень та задоволення клієнтів.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для удосконалення системи управління служби таксі.

Особистий внесок здобувача. Усі наукові результати, що викладені у бакалаврській дипломній роботі, отримано автором самостійно. У наукових працях, опублікованих у співавторстві, автору належать такі результати: переваги та недоліки телеграм боту порівняно з мобільним застосунком [3]; розробка Telegram-бота для системи управління роботою служби таксі [4].

Апробація матеріалів бакалаврської дипломної роботи.

Результати роботи доповідались на:

– LIII Науково-технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2024 р.) [3];

– Молодь в науці: дослідження, проблеми, перспективи (МН-2024) [4].

Публікації. За тематикою дослідження опубліковано 2 наукові праці у збірниках матеріалів конференцій.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз розвитку телеграм-ботів для систем управління роботою служб таксі

З розвитком технологій та популярністю мобільних застосунків, системи управління служби таксі стають все більш ефективними та зручними для користувачів у різних галузях. Зокрема, інтеграція технологій чат-ботів в платформу Telegram виявляється дуже перспективною для оптимізації процесів управління та взаємодії з клієнтами в сфері таксі.

Однією з ключових переваг використання телеграм-ботів для управління службами таксі є їхня доступність та зручність використання для клієнтів та власників таксопарку. Вони надають можливість клієнтам швидко замовити таксі, вказавши необхідні маршрут та додаткові умови, а власникам таксопарку – ефективно керувати флотом машин та контролювати робочий процес.

Розробка програмних модулів для впровадження телеграм-ботів у системи управління таксі є важливим завданням у цьому плані, оскільки дозволяє створювати інноваційні рішення, що покращують якість обслуговування та надають конкурентні переваги бізнесу у сфері таксі. Такі модулі можуть включати в себе функціонал замовлення таксі, відстеження руху машин, обробку платежів та звітність про роботу флоту.

Крім того, інтеграція телеграм-ботів у системи управління таксі дозволяє розширити функціонал за допомогою різних модулів та можливостей. Наприклад, можливість інтеграції з системами навігації, сповіщеннями про стан замовлення, а також забезпечення високого рівня безпеки та конфіденційності даних користувачів.

Отже, розробка програмних модулів для впровадження телеграм-ботів у системи управління служби таксі є критично важливим завданням у сучасному цифровому ландшафті. Використовуючи досвід та ресурси розробників програмного забезпечення, компанії можуть створювати інноваційні рішення,

що покращують якість обслуговування та забезпечують конкурентні переваги. Зважаючи на швидкий розвиток технологій та зміни вимог ринку, важливо постійно вдосконалювати та адаптувати телеграм-боти для оптимального задоволення потреб користувачів у сфері таксі.

1.2 Порівняльний аналіз аналогів

У сучасному цифровому світі розробка та використання програмних ботів для надання послуг стає все більш поширеним. Розглянемо деякі аналогічні рішення, які можуть слугувати прикладом для розробки Telegram-бота для управління служби таксі:

- Uber;
- Grab;
- Bolt;

Одним із прикладів застосунку для замовлення таксі є найпопулярніший застосунок Uber (рис. 1.1).

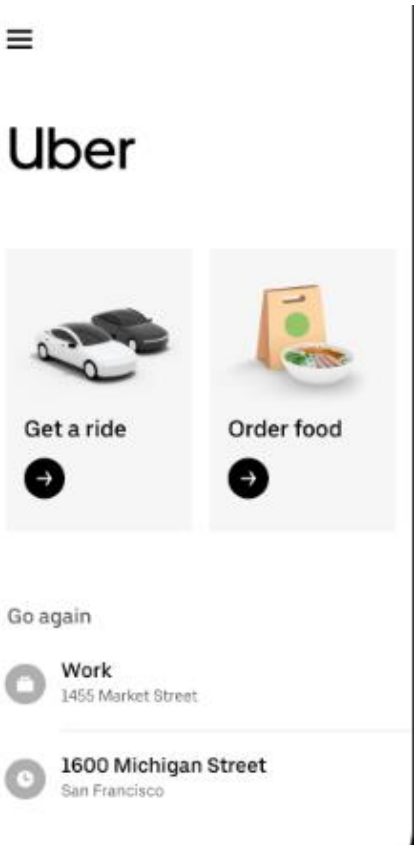


Рисунок 1.1 – Приклад роботи застосунку Uber

Uber [5] – це один з найбільш популярних сервісів таксі у світі, який надає можливість замовити та оплатити поїздку через мобільний застосунок. За допомогою Uber, користувачі можуть вибирати тип автомобіля, вказувати маршрут, відстежувати місцезнаходження та оцінювати якість обслуговування.

Що стосується переваг, Uber відомий своєю широкою доступністю у багатьох країнах світу, високою надійністю та зручністю користуванням. Однак, серед можливих недоліків можна відзначити високу вартість послуг у деяких регіонах та можливість затримок через транспортні затори або великий попит у пік години.

Grab [6] – це азіатський сервіс, який надає різноманітні послуги, включаючи замовлення таксі, доставку їжі, кур'єрські послуги та інші. За допомогою Grab, користувачі можуть користуватися широким спектром послуг через один мобільний застосунок, що робить процес замовлення та оплати послуг більш зручним (рис 1.2).

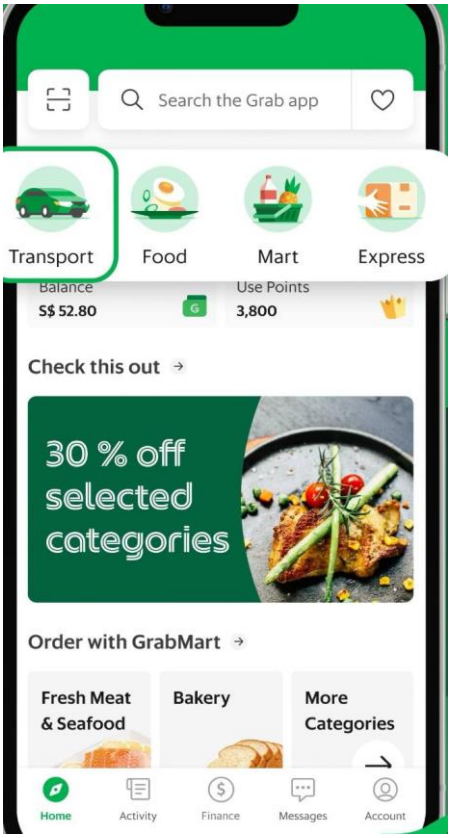


Рисунок 1.2 – Приклад роботи застосунку Grab

Додатково, Grab відрізняється своєю великою популярністю в азіатських країнах, таких як Сінгапур, Малайзія, Індонезія та Філіппіни. Його послуги дуже популярні серед місцевого населення та туристів, оскільки вони пропонують швидкий та зручний спосіб переміщення та доставки.

Однак, вартість послуг Grab може варіюватися в залежності від популярності та попиту, інфраструктури та інших факторів. Це може призвести до високих цін у певних районах або часи доби, що може зробити сервіс менш привабливим для деяких користувачів. Також, деякі користувачі можуть стикатися з проблемами надійності та якості обслуговування, особливо в пік часи або в часи поганої погоди.

Bolt (раніше відомий як Taxify) [7] – це сервіс таксі, який надає послуги замовлення та оплати таксі через мобільний застосунок. Bolt пропонує широкий вибір типів автомобілів і додаткові послуги в різних регіонах, такі як замовлення їжі та оренда автомобілів (рис 1.3).

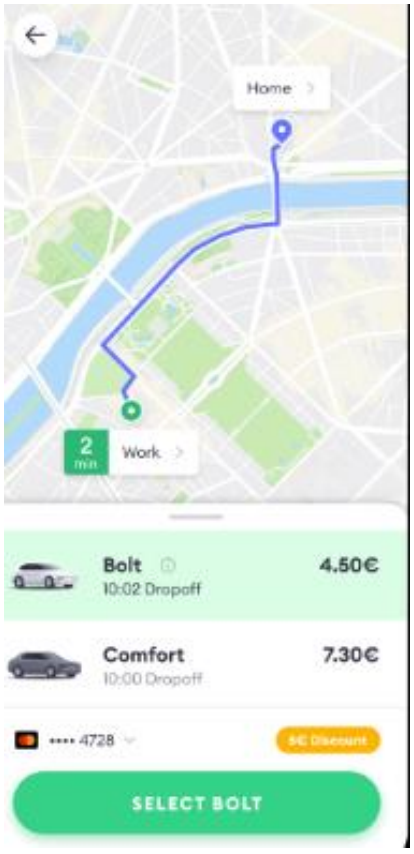


Рисунок 1.3 – Приклад роботи застосунку Bolt

Однією з його переваг є широкий вибір типів автомобілів та додаткових послуг, таких як доставка їжі та аренда автомобілів. Зокрема, Bolt активно розвивається у багатьох країнах Європи та Африки. Проте, серед можливих недоліків можна відзначити обмежену доступність у деяких регіонах та можливість виникнення проблем з якістю обслуговування, особливо у години пік або в погіршувальних погодних умовах.

Розробка програмних модулів для Telegram-бота системи управління служби таксі вимагає глибоких знань у програмуванні та інтеграції з платформою Telegram. Основні аспекти, які необхідно врахувати, включають інтеграцію з Telegram API для взаємодії з користувачами, розробку функціоналу для прийому та обробки замовлень, відстеження маршрутів автомобілів, оплату послуг через бота, захист персональних даних та оптимізацію користувацького інтерфейсу. Особлива увага буде приділена створенню зручного та інтуїтивно зрозумілого інтерфейсу, який забезпечить зручну взаємодію з ботом. Цей процес розробки дозволить покращити обслуговування клієнтів та оптимізувати робочі процеси служби таксі через ефективне використання Telegram-бота.

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльні характеристика онлайн платформ

Критерії	Uber	Grab	Bolt	Власний модуль
Простота використання	+	+	+	+
Інтуїтивно зрозумілий інтерфейс	+	-	+	+
Відсутність проблем з підтримкою клієнтів	-	+	-	+
Кросплатформність	+	-	-	+
Вбудовані функції спілкування з водіями	+	-	+	+
Можливість використання без необхідності завантаження мобільного застосунку	-	-	-	+
Можливість користуватися сервісом без наявності кредитної картки	-	-	+	+
Загальна оцінка	57%	28.5%	57%	100%

Зважаючи на аналіз порівняльних характеристик та ринкових тенденцій у сфері таксі, а також враховуючи, що на даний момент не існує подібного Telegram-бота для служби таксі можна зробити висновок про доцільність розробки та впровадження Telegram-бота для системи управління служби таксі. Створення такого бота може значно полегшити процеси замовлення та відстеження таксі, забезпечуючи високий рівень зручності для користувачів.

Крім того, враховуючи зріст популярності месенджерів і зручність їх використання, Telegram-бот може бути ефективним інструментом для замовлення таксі без необхідності завантаження додаткового мобільного застосунку. Це може привернути більше користувачів та забезпечити зручність використання послуг таксі.

Отже, розробка та впровадження Telegram-бота для системи управління служби таксі може виявитися стратегічно важливим кроком для підвищення конкурентоспроможності та задоволення потреб сучасних користувачів. З врахуванням зазначених переваг та потенціалу для подальшого розвитку, такий бот може стати не лише інноваційним рішенням, але й ключовим конкурентним перевагою на ринку таксі-послуг.

1.3 Аналіз методів розв’язання задачі

Розробка програмного модуля для Telegram-бота системи управління служби таксі потребує виваженого підходу до вибору методів розробки. Нижче розглянуті основні методи реалізації такого модуля та їх переваги та недоліки.

Один з можливих підходів – це розробка окремої бібліотеки або модуля, який може бути інтегрований у вже існуючі системи управління таксі. Цей метод може забезпечити швидкий старт роботи та зручність для користувачів, але може вимагати додаткових зусиль для інтеграції та може бути обмеженим у функціоналі.

Інший підхід полягає в розробці окремого сервісу для інтеграції Telegram-бота у вже існуючі системи управління таксі. Це дозволить зосередитися на розробці самого бота, а не на повній системі, а також може зменшити ресурси,

необхідні для інтеграції, порівняно з іншими методами. Однак цей підхід може вимагати значних апаратних ресурсів у разі інтеграції з багатьма системами.

Найбільш оптимальним варіантом може бути розробка власного Telegram-бота для управління служби таксі. Це дозволить уникнути адаптації під різні потреби сторонніх платформ і забезпечить повний контроль над функціоналом. Однак такий підхід може вимагати значних зусиль і часу на розробку та підтримку платформи.

У зв’язку з обговореними перевагами та недоліками кожного методу, вирішено використати метод власного Telegram-бота. Це забезпечить повний контроль над функціоналом та можливістю адаптації до конкретних потреб користувачів без зайвих обмежень.

1.4 Постановка задач для Telegram-бота

Після аналізу порівняльних характеристик існуючих систем управління служби таксі та методів їх реалізації, було визначено наступні завдання, які потрібно виконати для успішної реалізації Telegram-бота для таксі:

- здійснити аналіз стану питання та методів розв’язання задачі;
- спроектувати модель даних та структуру Telegram-бота;
- розробити метод та алгоритм оптимізації процесу видачі замовлень водіям
- розробити метод та алгоритм автоматичного призначення замовлень водіям;
- розробити загальний алгоритм роботи програми;
- розробити графічний інтерфейс для зручної взаємодії з користувачами;
- програмно реалізувати модуль Telegram-бота, який забезпечить взаємодію з користувачами та обробку їх запитів;
- провести тестування програмного застосунку;
- розробити інструкцію користувача та описати системні вимоги програми для відстеження читання.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У даному розділі було проведено порівняльний аналіз існуючих платформ для управління служби таксі, таких як Uber, Grab, Bolt, а також власний модуль.

Аналіз показав, що кожна з розглянутих платформ має свої переваги та недоліки. Наприклад, Uber і Bolt відзначаються високою простотою використання та наявністю вбудованих функцій замовлення та відстеження, але мають обмежену кроссплатформенність. Grab має деякі обмеження у функціоналі та доступності, але також має популярність серед користувачів. З іншого боку, власний модуль для системи управління служби таксі отримав найвищу загальну оцінку, оскільки повністю відповідає потребам користувачів.

На підставі цього аналізу було прийнято рішення про доцільність розробки Telegram-бота для системи управління служби таксі. Це дозволить забезпечити максимальну зручність та доступність для користувачів, а також підвищить конкурентоспроможність системи на ринку послуг таксі.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних даних системи

Для програмної реалізації Telegram-бота для системи управління служби таксі будуть використані Java, Spring Boot та Telegram API. Java – це потужна та універсальна мова програмування, розроблена компанією Sun Microsystems (пізніше придбаною Oracle) для створення різноманітних програм. Java є однією з найпопулярніших мов програмування у світі завдяки своїй переносності, надійності та простоті використання [8]. Spring Boot – це потужний фреймворк для розробки Java-додатків. Він надає рішення для швидкого старту і розгортання Java-додатків, використовуючи конвенції над конфігурацією. Spring Boot спрощує розробку за допомогою вбудованих функцій, таких як автоматичне конфігурування та управління залежностями [9]. Telegram API – це інтерфейс програмування застосунків (API), який надає доступ до функцій Telegram, таких як обмін повідомленнями, створення ботів, робота з групами тощо. Telegram API дозволяє розробникам створювати різноманітні застосунки, які використовують можливості Telegram для спілкування та автоматизації різних процесів [10]. За допомогою Java та Spring Boot в поєднанні з Telegram API можна створювати різноманітні боти для Telegram, які можуть взаємодіяти з користувачами, обробляти повідомлення, надсилати інформацію та виконувати автоматичні дії на основі отриманих даних. Це відкриває широкі можливості для розробки автоматизованих сервісів, інформаційних систем та інших додатків, які використовують комунікаційні можливості Telegram.

Розробка програмних модулів для реалізації Telegram-бота для системи управління служби таксі передбачає кілька етапів. По-перше, потрібно створити модуль, який може розпізнавати команди користувача. Цей модуль буде відповідати за розпізнавання команд користувача та надання необхідних даних для застосунку. Далі буде розроблений модуль для обробки отриманих даних та

встановлення необхідної дії користувачу. Цей модуль буде запам’ятовувати на якій дії знаходиться користувач для подальшої взаємодії з ботом.

Розробка цих програмних модулів вимагає глибокого розуміння програмування на Java, використання фреймворку Spring Boot та інтеграції з Telegram API. Для реалізації цих модулів використовуватимуться різноманітні інструменти та бібліотеки програмування, зокрема IntelliJ IDEA для написання коду на Java та роботи з фреймворком Spring Boot, а також використання Telegram API для взаємодії з месенджером Telegram.

Отже, розробка програмних модулів для реалізації Telegram-бота для системи управління служби таксі є комплексним завданням, яке включає кілька етапів. Модулі повинні бути ретельно спроектовані, реалізовані та протестовані, щоб забезпечити якісний досвід роботи з ботом для користувачів.

2.2 Розробка моделі системи

Модель системи Telegram-бота для служби таксі включає різні компоненти, такі як користувацький інтерфейс, базу даних (БД), API-інтеграції та зовнішні бібліотеки. Ці компоненти співпрацюють для забезпечення ефективного функціонування бота [11].

Користувацький інтерфейс відповідає за взаємодію з користувачами, приймання вхідних даних і надсилання вихідних даних. Він взаємодіє з базою даних для зберігання інформації про замовлення, маршрути, статуси та інші дані.

Інтеграція з зовнішніми сервісами через API дозволяє боту отримувати доступ до зовнішніх сервісів, таких як Open Street Map. Це дозволяє використовувати додаткову функціональність.

Зовнішні бібліотеки можна використовувати для спрощення розробки та додавання додаткових функцій у застосунок. Наприклад, бібліотека Telegram Bot API може бути використана для зчитування вхідних даних користувача та відправки вихідних даних, а також Liquibase для спрощення роботи з базою даних. Ці бібліотеки слід ретельно вибирати, щоб мінімізувати розмір застосунку та оптимізувати продуктивність.

Локальне сховище є важливою частиною моделі програми для відстежування читання. Вона відповідає за зберігання всієї інформації, пов'язаної з замовленнями, інформацією та квитками. Локальна БД дозволяє застосунку надавати швидкий і оперативний доступ до збережених даних без необхідності підключення до віддаленого сервера.

База даних може бути реалізована за допомогою різних технологій і бібліотек. Для запланованої розробки було обрано бібліотеку для керування БД під назвою Spring Data. Це бібліотека, яка забезпечує надійне та масштабоване рішення управління великими обсягами даних .

Для реалізації локальної бази даних для Telegram-бота слід розробити схему, яка визначатиме таблиці, стовпці та зв'язки між ними. Схема повинна відображати модель даних застосунку і забезпечувати логічну структуру для зберігання і пошуку даних. На рисунку 2.1 зображено схему бази даних розроблюваного застосунку у вигляді UML-діаграми класів.

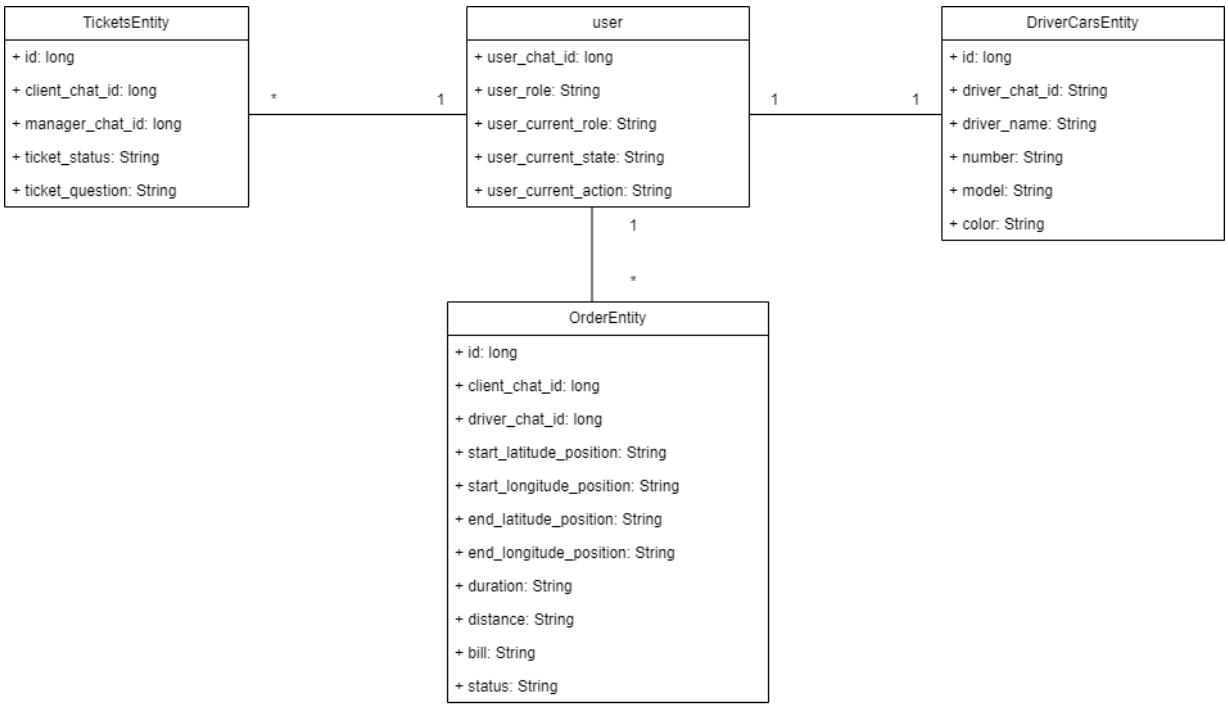


Рисунок 0.1 – Діаграма класів

Класи даних на діаграмі використовуються для забезпечення взаємодії між програмним кодом і базою даних шляхом створення простого та зрозумілого

інтерфейсу для операцій з даними. Змінні в цих класах відображають інформацію, яка зберігається в таблицях бази даних або обчислюється під час виконання SQL-запитів. Методи також повертають обчислені значення, використовуючи дані, що зберігаються в базі даних.

Тип зв'язку між сутністю юзера та замовлень представлено як асоціація «один до багатьох». Цей тип зв'язку використовується, оскільки замовлення не може існувати незалежно від юзера. Крім того, один юзер може мати багато замовлень, але одне замовлення може бути пов'язанним лише з одним юзером. Зв'язок між сутністю юзер та квитки такий же, як уже описано.

Тип зв'язку між сутністю юзера та автомобілем водія показано як асоціація «один до одного». Цей тип зв'язку використовується тому, що один юзер, який є водієм може мати лише одне авто, але авто не може існувати без водія.

2.3 Розробка загального алгоритму роботи Telegram-бота

При розробці складної програмної системи, такої як Telegram-бот, для системи управління служби таксі, дуже важливо мати чітке уявлення про її загальну роботу. Це включає в себе визначення функціональності програми, взаємодію різних компонентів між собою та загальний потік даних і процесів. Один із способів досягти цього – розробка загального алгоритму.

Загальний алгоритм може допомогти отримати комплексне уявлення про функціональність застосунку, а також визначити потенційні вузькі місця і проблеми, які можуть виникнути в процесі розробки. Таким чином, можна оптимізувати дизайн програми, підвищити її ефективність та результативність.

Для розробки Telegram-бота, для системи управління служби таксі, необхідно розробити загальний алгоритм, згідно якого буде працювати Telegram-бот (рис. 2.2).

Розглянувши даний алгоритм, можна зрозуміти, що розроблений застосунок дає змогу користуватися службою таксі в телеграмі. Алгоритм дозволяє зробити систему замовлення таксі максимально простою для клієнта.

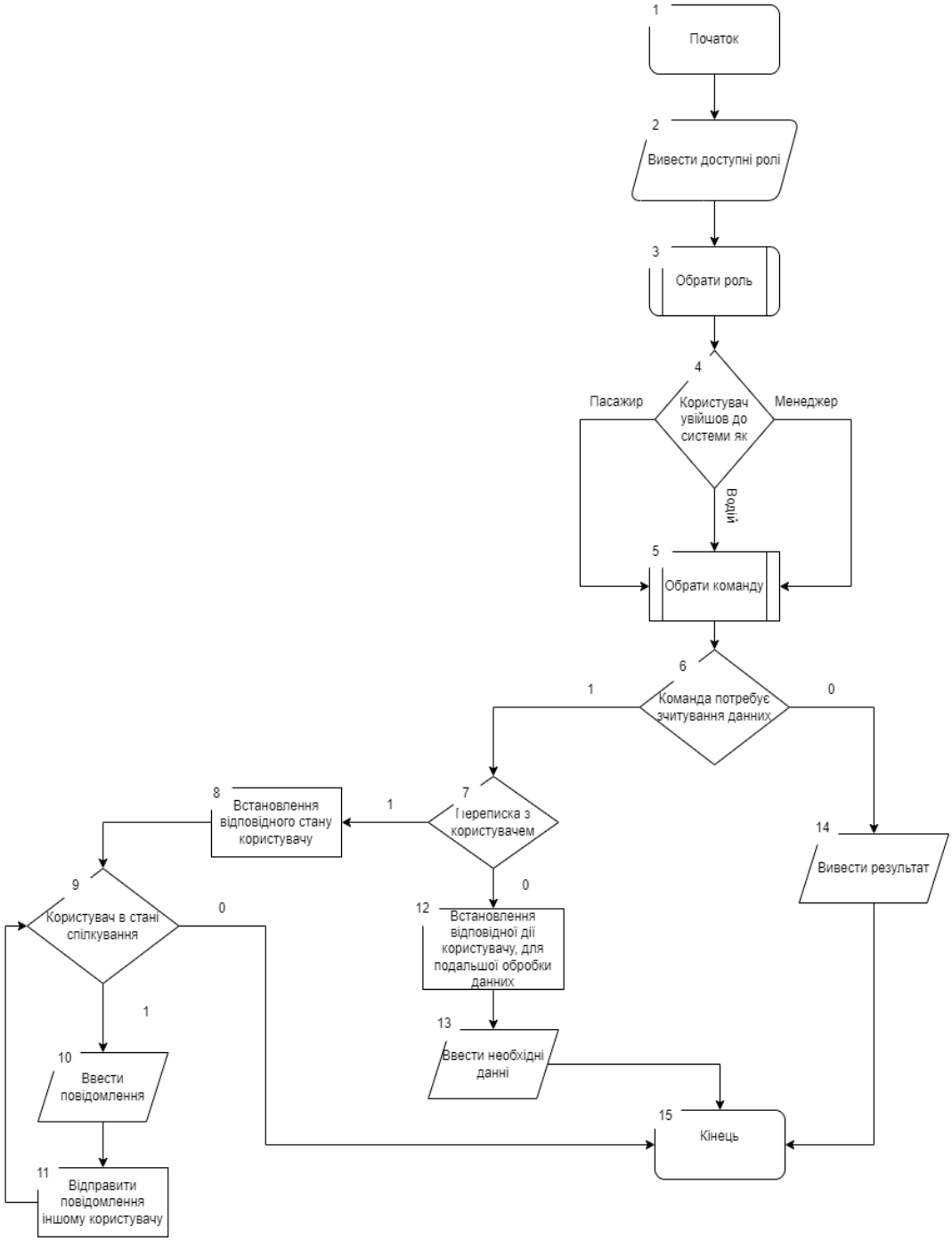


Рисунок 2.2 – Блок-схема загального алгоритму роботи Telegram-бота

Нижче представлено пояснення кроків виконання загального алгоритму:

Крок 1. Початок.

Крок 2. Виведення доступних ролей користувачу, після перевірки чи існує користувач в базі і має доступні ролі для використання.

Крок 3. Вибір ролі користувача.

Крок 4. Перевірка з якою роллю входить користувач.

Крок 5. Очікування вибору команди від користувача.

Крок 6. Перевірка чи потрібно вводити данні користувача для конкретної команди.

Крок 7. Якщо команда потребує даних від користувача, перевірка чи користувач обрав переписку з іншим користувачем.

Крок 8. Встановлення стану переписки користувачу, для того щоб його наступні повідомлення дублювалися іншому користувачу.

Крок 9. Перевірка чи користувач в стані спілкування.

Крок 10. Очікуємо повідомлення від користувача.

Крок 11. Перенаправлення повідомлення.

Крок 12. Встановлення необхідного екшену користувачу для коректної обробки даних.

Крок 13. Очікування даних.

Крок 14. Виводимо результат.

Крок 15. Кінець.

Отже, було розроблено алгоритм взаємодії користувача з системою управління службою таксі, який дозволяє користувачам обирати роль, виконувати відповідні команди, вводити необхідні дані для обробки, та спілкуватись з іншим користувачем.

2.4 Розробка методу та алгоритму автоматичного призначення замовлень водіям

Вперше запропоновано метод автоматичного призначення замовлень водіям, який був розроблений для зменшення часу очікування клієнтів. Метод враховує рейтинг водіїв та пріоритезує замовлення з найдовшим часом очікування в межах міста.

Принципи роботи методу:

1) автоматичне призначення замовлень водіям з рейтингом нижче 4: Для підвищення ефективності обслуговування клієнтів, водіям, рейтинг яких нижче 4, замовлення присвоюються автоматично. Це дозволяє уникнути

ситуацій, коли водії з низьким рейтингом обирають тільки зручні замовлення, що може призводити до збільшення часу очікування для інших клієнтів.

2) пріоритезація замовлень з найдовшим часом очікування: метод призначає замовлення, які очікують найбільший проміжок часу. Це сприяє зменшенню часу очікування клієнтів, забезпечуючи швидше обслуговування тих, хто довше чекає.

3) обмеження призначення замовлень межами міста: Щоб гарантувати, що водії не отримують замовлення з віддалених місць, метод обмежує призначення замовлень межами міста. Це дозволяє забезпечити більш рівномірний розподіл замовлень.

На рисунку 2.3 зображено алгоритм автоматичного призначення замовлень водіям.

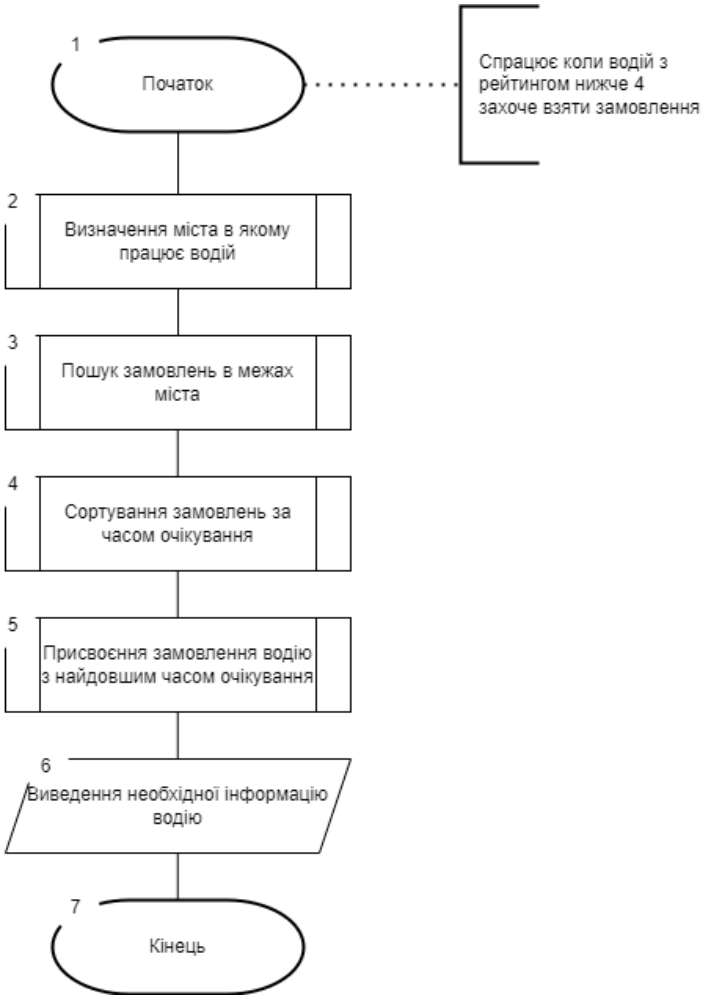


Рисунок 2.3 – Блок-схема алгоритму автоматичного призначення замовлень водіям

Нижче представлено пояснення кроків виконання алгоритму автоматичного призначення замовлень водіям:

Крок 1. Початок.

Крок 2. Визначення місто праці водія.

Крок 3. Пошук замовлень в межах міста.

Крок 4. Сортування замовлень за часом очікування від найдовшого до найменшого.

Крок 5. Присвоєння замовлення з найдовшим очікуванням водію.

Крок 6. Виведення необхідної інформації про поїздку водію.

Крок 7. Кінець.

Отже, було розроблено метод автоматичного призначення замовлень водіям, який дозволяє підвищити ефективність розподілу замовлень та задоволення клієнтів.

2.5 Розробка методу та алгоритму видачі замовлень водіям

Для системи управління служби таксі важливо розробити метод і алгоритм, який оптимізує процес видачі замовлень водіям, забезпечуючи ефективність та швидкість реагування на замовлення клієнтів.

Метод видачі замовлень водіям базується на використанні їх географічної локації. Після того як водій стає доступним для прийому нового замовлення, система використовує його географічне положення для пошуку доступних замовлень в межах одного кілометра від нього. Це допомагає уникнути надлишкового часу на переміщення водія до місця замовлення та покращує швидкість обслуговування. Якщо в цьому радіусі знаходиться лише одне доступне замовлення, воно автоматично призначається водієві без його участі в виборі.

У разі, якщо в радіусі одного кілометра від водія не знайдено жодного доступного замовлення, система розширює радіус пошуку на ще один кілометр. Це дозволяє збільшити шанси на знаходження замовлення для водія та забезпечує більш ефективне використання його часу.

Потім знайдені замовлення сортуються за часом очікування клієнтів. Замовлення, які чекають найбільше часу, розміщуються на верхніх позиціях списку. Це дозволяє водіям спочатку бачити та вибирати ті замовлення, які вже довший час чекають на обслуговування, щоб максимально оптимізувати їх час та задоволення клієнта.

На рисунку 2.4 алгоритм оптимізації процесу видачі замовлень подано у вигляді блок-схеми.

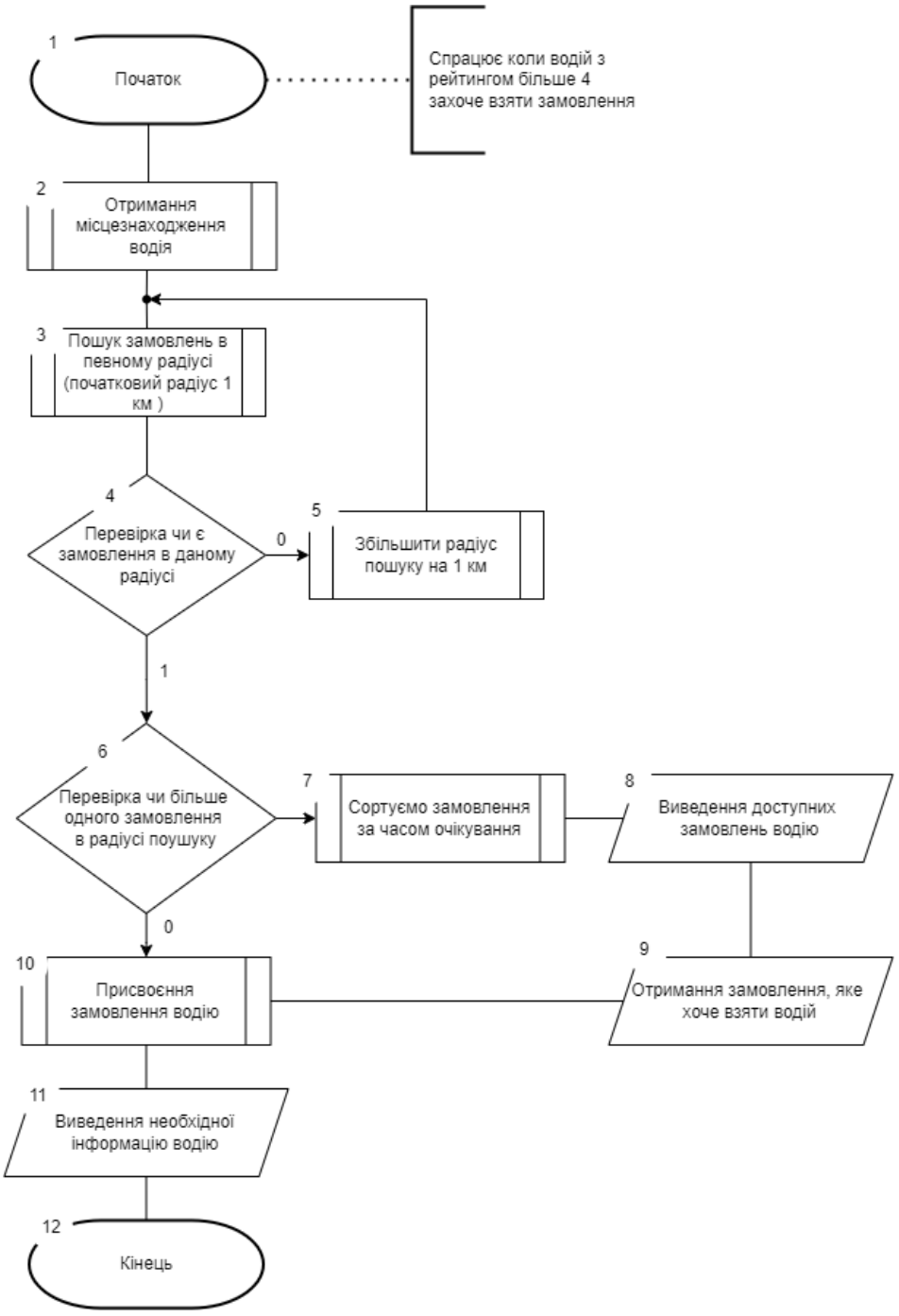


Рисунок 2.4 – Блок-схема алгоритму оптимізації видачі замовлень водіям

Нижче представлено пояснення кроків виконання загального алгоритму:

Крок 1. Початок.

Крок 2. Отримання місцезнаходження водія.

Крок 3. Пошук замовлень в певному радіусі (початковий пошук в радіусі 1км).

Крок 4. Перевірка чи є замовлення в радіусі.

Крок 5. Якщо замовлень в радіусі немає то збільшуємо радіус на 1 км.

Крок 6. Перевірка чи більше одного замовлення в певному радіусі.

Крок 7. Сортуємо замовлення за часом очікування від старих до нових.

Крок 8. Виводимо водію замовлення в певному радіусі.

Крок 9. Приймаємо замовлення, яке водій хоче взяти.

Крок 10. Призначаємо замовлення водію.

Крок 11. Виводимо водію інформацію про замовлення.

Крок 12. Кінець

Отже удосконалено метод видачі замовлень водіям, що дозволяє забезпечити ефективне та швидке призначення замовлень водіям, зменшуючи час очікування клієнтів та покращуючи якість обслуговування.

2.6 Висновки

У другому розділі було проведено аналіз вхідних та вихідних даних програмного продукту. Також було розроблено основний алгоритм роботи Telegram-бота, модель системи з діаграмою класів, метод та алгоритм автоматичного призначення замовлень водіям, метод та алгоритм оптимізації процесу видачі замовлень водіям.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Перед тим як приступити до розробки Telegram-бота, важливо обдумати вибір мови програмування, оскільки від цього буде залежати якість та продуктивність роботи програмного застосунку. Кожна мова має свої переваги та недоліки, тому потрібно ретельно проаналізувати кожен варіант перед прийняттям рішення. Нижче наведено кілька основних мов програмування, які зазвичай використовуються для розробки Telegram-ботів, та порівнюються їх особливості.

1. Java – це об'єктно-орієнтована мова програмування з високою платформонезалежністю, що дозволяє створювати Telegram-ботів, використовуючи Telegram Bot API. Для розробки ботів на Java зазвичай використовуються зручні бібліотеки, такі як telegrambots або telegram-api, які спрощують взаємодію з Telegram API. Хоча мова може бути складною для початківців через строгу типізацію, вона залишається популярним вибором для створення програм з великою універсальністю та потенціалом для розширення.

2. C++ – це потужна мова програмування загального призначення, яка надає великий контроль над ресурсами системи. Використання C++ для створення Telegram-ботів може бути корисним для тих, хто шукає максимальну продуктивність та оптимізацію ресурсів. Однак розробка на C++ може бути більш складною через його складну синтаксичну структуру та вимоги щодо керування пам'яттю.

3. C# – це мова програмування, розроблена компанією Microsoft, яка є частиною платформи .NET. Вона відома своєю простотою використання та широким спектром можливостей. Для створення Telegram-ботів на C# можна використовувати .NET бібліотеки, наприклад, Telegram.Bot, які дозволяють легко взаємодіяти з Telegram API.

Порівняльний аналіз описаних мов програмування наведено в таблиці 0..

Таблиця 0.1 – Порівняння мов програмування

Особливість	**Java**	**C++**	**C#**
Платформонезалежність	+	-	-
Автоматичне управління пам'яттю	+	-	+
Наявність бібліотек для Telegram API	+	+	+
Інтеграція з іншими сервісами	+	-	+
Широкий вибір бібліотек	+	-	-
Контроль ресурсів пристрою	-	+	-
Підсумок	5	2	3

Для запланованої розробки Java є кращим вибором з кількох причин. По-перше, ця мова має кращу читабельність і більш компактний синтаксис порівняно з C++ та C#. Це зробить процес розробки швидшим та ефективнішим. По-друге, Java має більше різноманітних бібліотек, що дозволить полегшити розробку. По-третє, Java має платформонезалежність, що дозволить розробляти проект на різних пристроях.

Інтегровані середовища розробки (IDE) – це програмні системи, які надають комплексне середовище для розробки, тестування та розгортання програм. Коли мова йде про розробку застосунків на Java, існує кілька IDE, які можна використовувати.

1. IntelliJ IDEA – це IDE, яке було розроблено для програмування на різних мовах, зокрема Java, Kotlin, Scala, Groovy, і багатьох інших. Воно відоме своєю високою продуктивністю, багатим функціоналом, великою кількістю плагінів і підтримкою різних технологій розробки.

2. Eclipse – це IDE, яке було розроблено для мови Java. Воно відоме своєю відкритістю та розширюваністю, а також широким спектром інструментів розробки.

3. NetBeans – це IDE, яке також спеціалізується на роботі з Java та іншими мовами програмування. Воно відоме своєю простотою використання, дружнім інтерфейсом користувача, а також потужними інструментами для розробки та підтримки проектів..

Порівняльний аналіз на основі можливостей згаданих IDE наведено в таблиці 0..

Таблиця 0.2 – Порівняння середовищ розробки

Особливість	**IntelliJ IDEA**	**Eclipse**	**NetBeans**
Швидкодія	+	–	–
Інтеграція зі всіма потрібними для розробки зовнішніми сервісами	+	–	-
Підтримка функцій рефакторингу	+	-	-
Візуальний редактор	+	+	+
Відлагодження	+	+	+
Миттєваий запуск для тестування	+	–	+
Система плагінів та розширень	+	+	+
Підсумок	7	3	6

Як видно з таблиці, Intellij IDEA має перевагу над Eclipse та NetBeans, що робить її кращим вибором для запланованої розробки.

Розробляючи програмний модуль для системи управління служби таксі, важливо обирати технології, що гармонійно поєднуються і забезпечують ефективність і надійність рішення. Java є популярною мовою програмування завдяки своїй платформенній незалежності та великій спільноті розробників. Spring Boot є потужним фреймворком для розробки застосунків на Java, який забезпечує швидку і просту розробку за допомогою вбудованих конфігурацій і стандартів. Бібліотека Telegram надає інструменти для розробки ботів, що використовують цей популярний месенджер для взаємодії з користувачами.

Spring Boot [12] – це потужний фреймворк для розробки застосунків на Java, який спрощує процес створення веб-додатків, мікросервісів та інших застосунків. Він надає стандартні конфігурації за замовчуванням, що дозволяє швидко створювати працюючі застосунки з меншим обсягом коду. Spring Boot також має багато вбудованих бібліотек і компонентів, що спрощують роботу з базами даних, безпекою, обробкою HTTP-запитів та іншими аспектами розробки.

Бібліотека Telegram (Telegram Bot API) [13] – це набір інструментів, що надається Telegram для створення ботів. Вона дозволяє розробникам створювати інтерактивних ботів, які можуть взаємодіяти з користувачами через чати Telegram. Бібліотека надає доступ до різних функцій API Telegram, таких як обробка повідомлень, відправлення повідомлень, робота з клавіатурою чату та багато іншого. Завдяки цій бібліотеці розробники можуть швидко і просто інтегрувати функціонал чат-бота в свої застосунки.

Поєднання цих технологій дозволить створити ефективний і зручний Telegram-бот для системи управління служби таксі. Spring Boot забезпечить швидкість розробки і надійність серверної частини, в той час як Java дозволить створювати розширені функції обробки даних та бізнес-логіки. Бібліотека Telegram спростить інтеграцію з платформою месенджера, дозволяючи боту взаємодіяти з клієнтами таксі зручним способом.

Таким чином, вибір Java, Spring Boot і бібліотеки Telegram для розробки Telegram-бота для системи управління служби таксі є ключовим для створення

функціонального та ефективного рішення, яке задовольнить потреби користувачів і оптимізує роботу служби таксі.

3.2 Розробка модулів зчитування команд користувача

Розуміння та обробка команд користувача у телеграм-боті є ключовою функцією для забезпечення коректної взаємодії із клієнтами [14]. Процес обробки команд користувача охоплює такі аспекти, як ідентифікація та аналіз отриманих повідомлень, встановлення потрібного стану або дії на основі змісту повідомлення, а також відповідна обробка цих даних для подальшої дії (рис 3.1).

```
private static final List<String> firstPriorityCommand = List.of("Відмінити замовлення",
        "Назад ←\uFE0F",
        "Закрити тікет",
        "Закінчити замовлення",
        "Закінчити розмову з водієм");

public void processTextCommand(Update update) {
    Long chatId = update.getMessage().getChatId();
    UserAction userAction = userService.getUserAction(chatId);
    UserState userState = userService.getUserState(chatId);

    if (update.getMessage().getText() !=null && firstPriorityCommand.contains(update.getMessage().getText())){
        commandControllerForText.processCommand(update.getMessage().getText(), chatId, update);
    }
    else if (!UserAction.DEFAULT.equals(userAction)){
        userActionController.processUserActionCommand(userAction, chatId, update);
    } else if (!UserState.DEFAULT.equals(userState)){
        userStateController.processUserStateCommand(userState, chatId, update);
    } else commandControllerForText.processCommand(update.getMessage().getText(), chatId, update);
}
```

Рисунок 3.1 – TextCommandService

Коли клієнт взаємодіє з ботом, він може надіслати різні команди чи запити, що потребують спеціалізованої обробки, за це відповідає «CommandControllerForText» та «CommandControllerForCallBackQuery», що видно на рисунку 3.1. Це може включати встановлення контексту, визначення очікуваного результату, а також взаємодію з іншими системами або сервісами для отримання додаткової інформації чи обробки запитів.

Крім того, обробка стану або дії відображається у взаємодії з користувачем через чат-бот, за це відповідає «UserActionController» та

«UserStateController» , що видно на рисунку 3.1. Наприклад, коли клієнт надсилає запит на конкретну інформацію чи виконання певної операції, бот повинен аналізувати цей запит, встановлювати необхідний контекст і відповідати відповідним чином.

Процес обробки команд та взаємодії з клієнтами вимагає належного розуміння потреб користувачів, визначення оптимальних шляхів вирішення запитів і впровадження ефективної логіки для забезпечення зручної та продуктивної взаємодії з ботом.

Такий підхід дозволяє створити бота, який відповідає вимогам клієнтів і забезпечує позитивний досвід використання через інтуїтивно зрозумілу та ефективну обробку команд та запитів.

3.3 Розробка сервісу для отримання інформації про маршрут

Основною задачею модуля є повернення інформації маршруту між вибраними місцями (рис 3.2). При розробці даного модуля першочергово було імплементовано сервіс для пошуку місць за допомогою OpenStreet Map API [15]. Сервіс отримує в якості параметрів рядок запиту та обробник завершення. Обробник завершення повертає JSON об’єкт [16], що містить інформацію про маршрут.

```
public JSONObject getRouteInfo(double startLon, double startLat, double endLon, double endLat) {
    String apiUrl = String.format(ROUTE_INFO_URL, startLon, startLat, endLon, endLat);

    try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
        HttpGet request = new HttpGet(apiUrl);
        try (CloseableHttpResponse response = httpClient.execute(request)) {
            String responseBody = EntityUtils.toString(response.getEntity());
            JSONObject json = new JSONObject(responseBody);
            JSONArray routesArray = json.getJSONArray( key: "routes");
            return routesArray.getJSONObject( index: 0);
        }
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Рисунок 3.2 – GeolocationService

В модулі було реалізовано метод «getRouteInfo», який призначений для отримання детальної інформації про маршрут між двома вказаними точками з використанням зовнішнього сервісу маршрутизації. Цей метод здійснює запит до API за допомогою вказаних координат стартової та кінцевої точок маршруту. Після обробки відповіді в форматі JSON [17], метод повертає деталі знайденого маршруту.

Також було реалізовано метод «getStreetName» (рис 3.3), який використовується для визначення назви вулиці за вказаними географічними координатами. Він здійснює запит до відповідного сервісу з використанням координат точки. Після обробки відповіді в форматі JSON, метод видобуває назву вулиці та номер будинку і повертає дані.

```
public String getStreetName(double latitude, double longitude) {
    String apiUrl = String.format(URL_FOR_STREET_NAME, latitude, longitude);
    try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
        HttpGet request = new HttpGet(apiUrl);
        try (CloseableHttpResponse response = httpClient.execute(request)) {
            String responseBody = EntityUtils.toString(response.getEntity());
            JSONObject json = new JSONObject(responseBody);
            String houseNumber = getHouseNumber(json);
            return json.getJSONObject( key: "address").get("road").toString() + " " + houseNumber;
        }
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

Рисунок 3.3 – GeolocationService

Додатково був створений окремий метод «getHouseNumber» (рис. 3.4) для обробки номерів будинків, оскільки не всі адреси містять номери будинків, що може призвести до виникнення помилок.

```
private static String getHouseNumber (JSONObject jsonObject) {
    try {
        return jsonObject.getJSONObject( key: "address").get("house_number").toString();
    } catch (Exception e){
        return "";
    }
```

Рисунок 3.4 – GeolocationService

Ці методи демонструють важливий компонент функціональності програмного модуля для роботи з географічними даними і використання зовнішніх сервісів для отримання необхідної інформації.

3.4 Розробка інтерфейсу програмного застосунку

Розробляючи інтерфейс Telegram-бота вирішено зробити його максимально простим та дружнім для використання користувачем.

При пошуку, користувача зустрічає бот із логотипом та назвою (рис. 3.5), з якою користувач буде асоціювати враження та досвід під час користування.

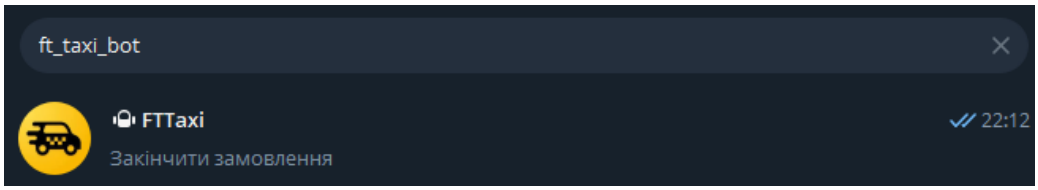


Рисунок 3.5 – Пошук бота в системі Telegram

Коли користувач відвідує бота, він бачить меню авторизації з доступними кнопками, які відрізняються в залежності від його ролі. Кожен користувач має свою унікальну роль, яка визначає доступні кнопки. Наприклад, звичайний користувач може використовувати бота лише у режимі пасажира (рис. 3.6 а), водій може ввійти як пасажир або водій (рис. 3.6 б), а менеджер має можливість використовувати бота як менеджер або пасажир (рис. 3.7 в).

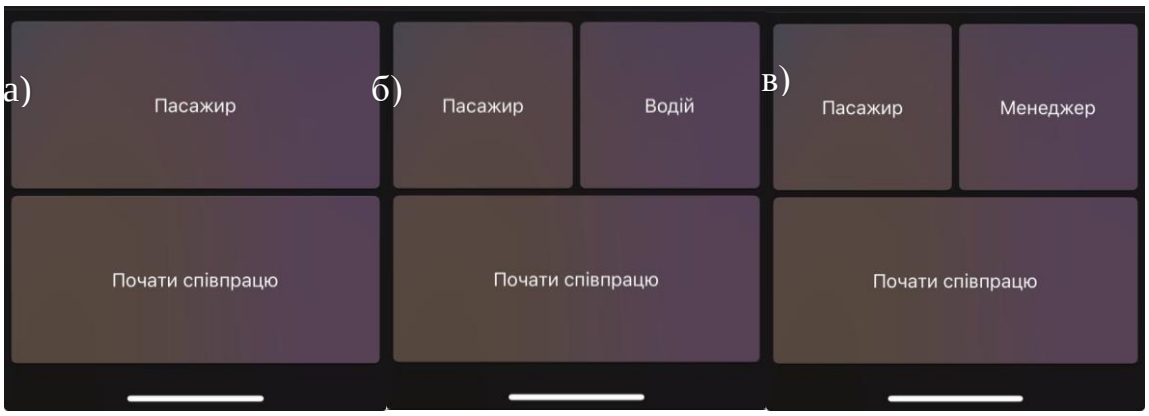


Рисунок 3.6 – Меню авторизації клієнта, для звичайного користувача (а), для водія (б), для менджера (в)

Після обрання ролі користувачеві стають доступні різні функції в залежності від обраної ролі. Для пасажира це можливість замовити таксі, переглянути історію поїздок та зв'язатись з менеджером (рис. 3.7 а). Для водія це активні замовлення, перегляд історії замовлень, можливість прийняти замовлення та зв'язатись з менеджером (рис. 3.7 б). Для менеджера доступні активні тікети та можливість вибору тікету для подальшої роботи (рис. 3.7 в).

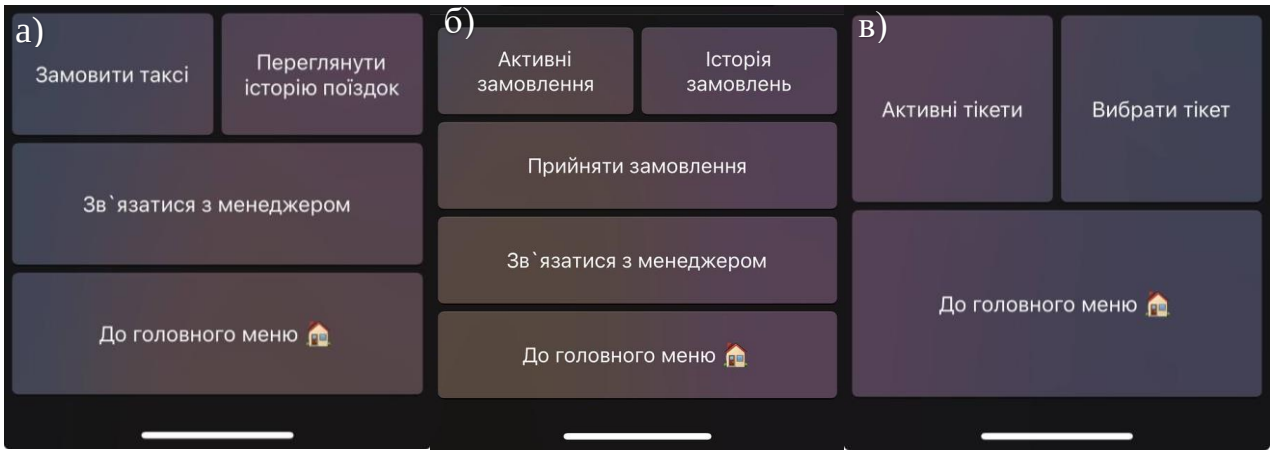


Рисунок 3.7 – Меню дій клієнта, для пасажира (а), для водія (б), для менеджера (в)

Розглянемо дії пасажира, якщо користувач натискає "замовити таксі". По-перше, йому потрібно визначити стартову та кінцеву точки поїздки. Це можна зробити за допомогою мапи (рис. 3.8 а), яка дозволяє вибрати місце розташування на карті шляхом перетягування маркерів або за допомогою пошуку (рис. 3.8 б), де користувач може ввести адресу або назву місця. Після визначення маршруту, користувач побачить уточнюючу інформацію щодо його кінцевої та стартової позицій, дистанції поїздки, ціни та приблизного часу поїздки (рис. 3.9). Ознайомившись з цією інформацією, користувач має можливість викликати таксі або відхилити його. Цей етап дозволяє користувачеві перевірити всю необхідну інформацію перед тим, як прийняти остаточне рішення щодо замовлення таксі.

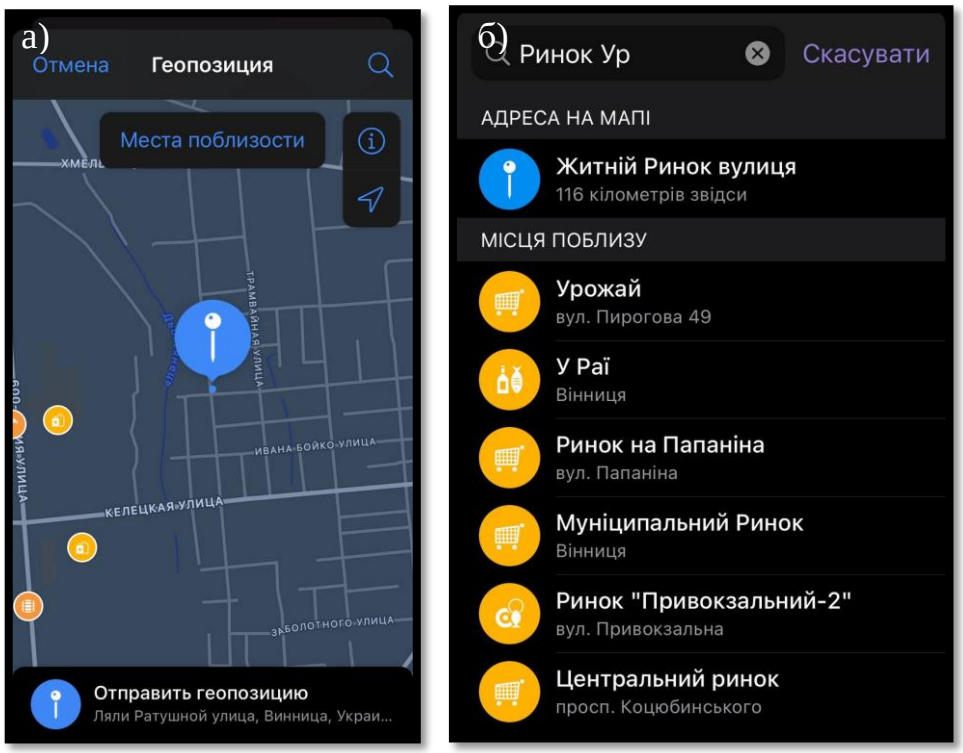


Рисунок 3.8 – Вибір стратової/кінцевої позиції поїздки. За допомогою мапи (а), за допомогою текстового пошуку (б).

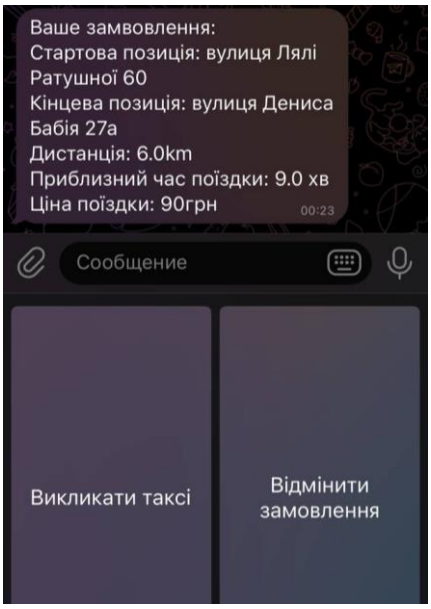


Рисунок 3.9 – Інформація про замовлення

Також пасажир має можливість переглянути історію своїх поїздок, щоб відстежити попередні маршрути та витрати. Крім того, він може зв'язатись з менеджером для отримання поради, вирішення скарг або надання пропозицій та

ідей щодо поліпшення сервісу. Це забезпечує пасажиру зручний доступ до підтримки та можливість активно взаємодіяти з сервісом таксі.

Тепер розглянемо дії водія, коли він натискає на активні замовлення. Водію надсилається список замовлень, до яких на даний момент не призначено жодного водія. В кожному замовленні він бачитиме деталі, такі як номер замовлення, початкову та кінцеву позиції, статус замовлення, що підтверджує його активність, відстань поїздки, час та ціну (рис. 3.10). Ця інформація дозволяє водіям ефективно керувати своїми замовленнями та приймати рішення про їх прийняття на основі різних факторів.

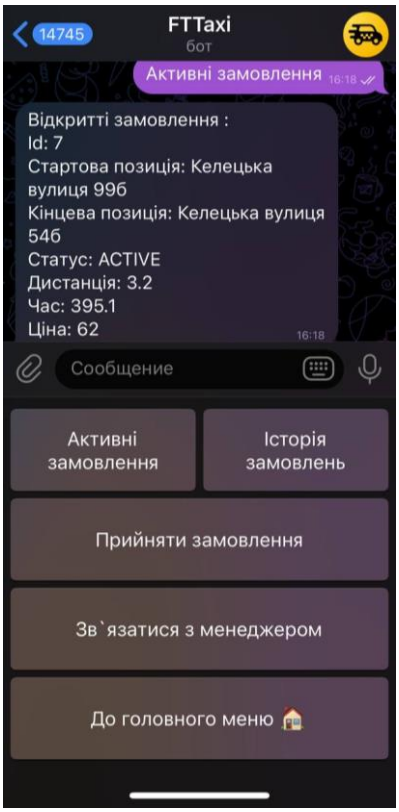


Рисунок 3.10 – Приклад списку активних замовлень

Після прийняття замовлення водію надходить інформація щодо стартової та кінцевої позиції маршруту на карті, щоб водій міг скористатися навігатором та ефективно прокладати маршрут. Крім того, водій має можливість зв'язатися з клієнтом для уточнення деталей або вирішення будь-яких питань. Також водій може переглянути історію своїх виконаних замовлень (рис. 3.11), що дозволяє

йому відстежувати свою роботу та аналізувати свою продуктивність. Ці функції допомагають водіям ефективно керувати своїм часом та ресурсами під час виконання замовлень.

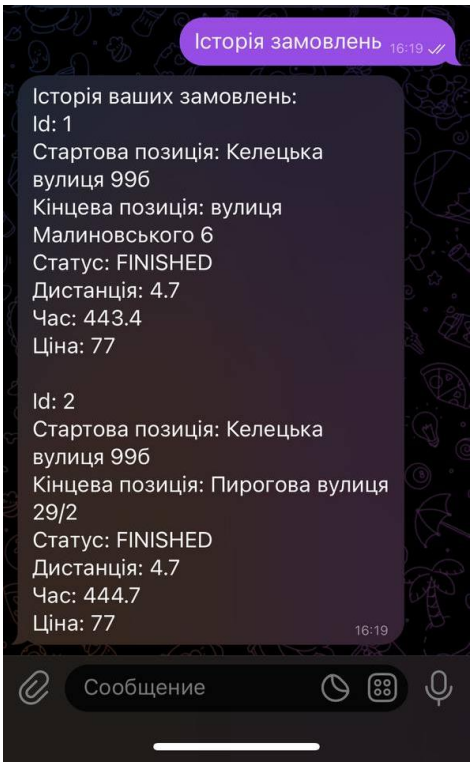


Рисунок 3.11 – Приклад історії замовлень водія

Далі розглянемо перегляд активних квитків для менеджера. При доступі до платформи менеджер має можливість переглянути список активних квитків перед тим, як починати на них відповідати (рис 3.12). Ця функція дозволяє менеджеру оцінити поточну кількість та характер питань або проблем, що виникають в системі, та прийняти рішення про найбільш ефективний спосіб їх вирішення.

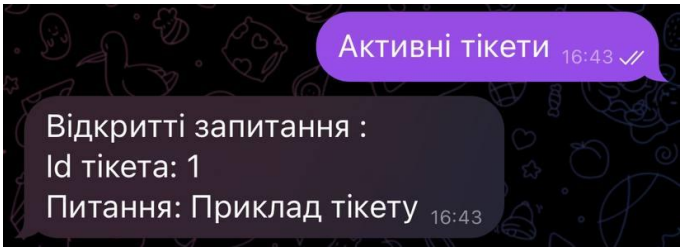


Рисунок 3.12 – Приклад списку активних тікетів

Розробка графічного інтерфейсу Telegram-бота була проведена у середовищі розробки IntelliJ IDEA з використанням бібліотеки Telegram. Було розглянуто можливості кожного типу користувача та їх взаємодію з Telegram-ботом таксі. Описано кілька ключових функцій для кожного з них, які є найбільш важливими в контексті користування платформою.

Пасажир може легко та зручно замовити таксі через бот, обравши початкову та кінцеву точки маршруту, отримавши інформацію про ціну та час поїздки, а також маючи можливість зв'язатися з менеджером у разі необхідності. Водій має доступ до активних замовлень, можливість перегляду деталей маршруту та контактування з клієнтом для уточнень. Менеджер, у свою чергу, може ефективно керувати активними заявками та вирішувати проблеми користувачів.

3.5 Висновки

У розділі розглянуто вибір технологій для розробки Telegram-бота для системи управління служби таксі. Проаналізувавши потреби програмного продукту, було вирішено використовувати Java для своєї платформенної незалежності та широких можливостей. Spring Boot обрано для спрощення розробки серверної частини за допомогою вбудованих конфігурацій і стандартів. Бібліотека Telegram обрана для забезпечення взаємодії з користувачами через месенджер Telegram з використанням чат-бота. Також було розроблено модулі зчитування команд користувача, сервісу для отримання інформації про маршрути та інтерфейс програмного застосунку.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Модульне тестування системи

Модульне тестування, також відоме як юніт-тестування, є одним з основних методів тестування програмного забезпечення. Цей процес полягає в перевірці окремих модулів або компонентів програми на коректність їхньої роботи. Модульні тести створюються для кожного окремого модуля програми, таких як функції, методи класів або навіть окремі компоненти програмного забезпечення.

Основна мета модульного тестування полягає в виявленні та виправленні помилок на ранніх етапах розробки програми. Це дозволяє зменшити витрати на виправлення помилок у майбутньому та забезпечити високу якість програмного забезпечення. Під час модульного тестування окремі частини програми перевіряються ізольовано від інших компонентів, що дозволяє ефективно виявляти та виправляти проблеми [18].

Ще однією важливою метою модульного тестування є забезпечення коректної роботи окремих модулів або функцій. Це дозволяє переконатися, що кожен компонент програми виконує свої функції правильно та очікувано.

Крім того, модульні тести полегшують підтримку та розширення програмного забезпечення. Під час внесення змін або додавання нового функціоналу, модульні тести допомагають переконатися, що зміни не вплинули на роботу існуючих компонентів.

Також варто відзначити, що використання модульних тестів сприяє покращенню співпраці в команді розробників. Якщо кожен розробник пише модульні тести для своїх функцій або класів, це дозволяє швидше виявляти та виправляти проблеми, а також забезпечує високу якість коду та програмного забезпечення в цілому.

Використання модульного тестування допомагає підвищити якість коду, зменшити кількість помилок та забезпечити стабільність програмного забезпечення.

Приклад модульного тестування зображено на рисунку 4.1.

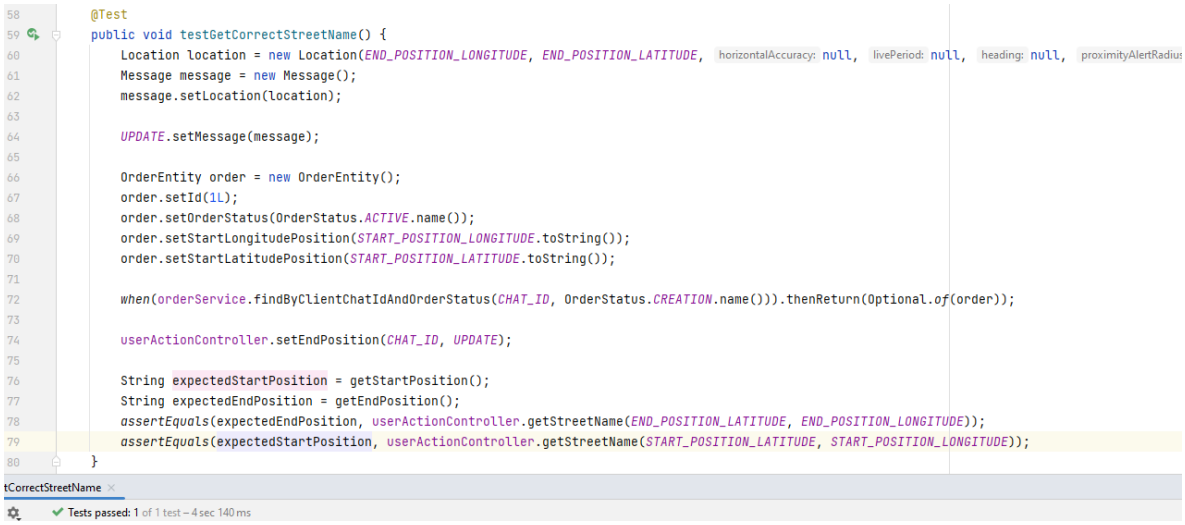


Рисунок 4.1 – Приклад модульного тестування

Було проведено ретельне тестування модулів системи результат тестування зображено на рисунку 4.2

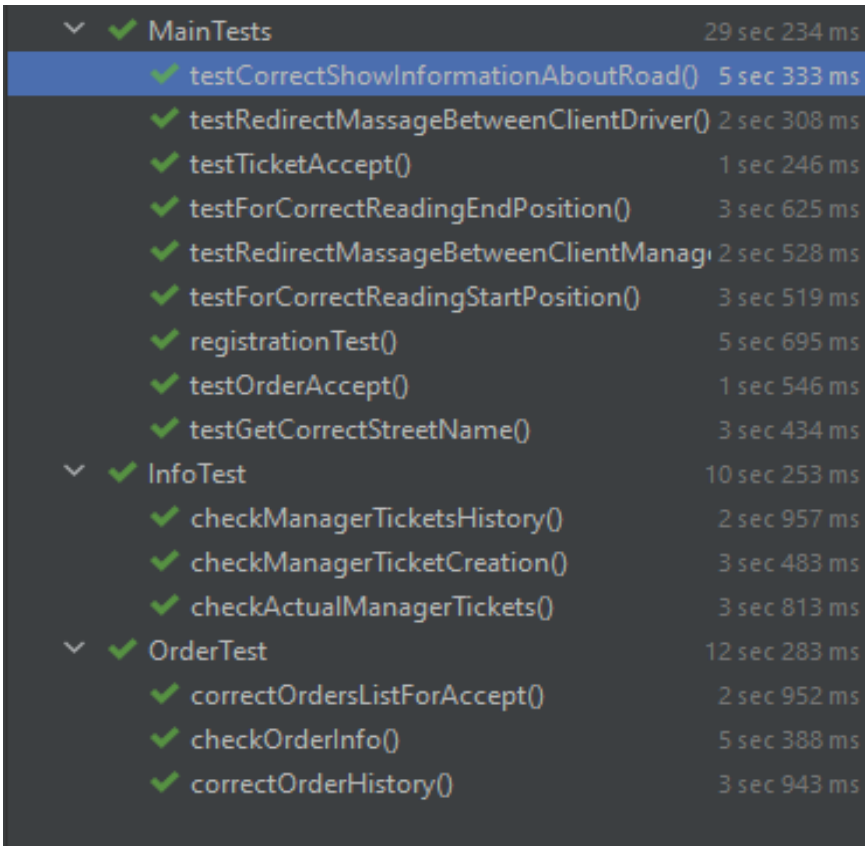


Рисунок 4.2 – Результат модульного тестування

Для модульного тестування було використано: фреймворк Junit та бібліотеки Mockito і AssertJ.

JUnit – це фреймворк для тестування програмного забезпечення в мові програмування Java. Він надає набір інструментів та бібліотек для автоматизації тестування, зокрема, для написання та виконання юніт-тестів. Юніт-тести, у свою чергу, є тестами, які перевіряють правильність роботи окремих модулів або компонентів програмного забезпечення, таких як методи класів або окремі функції.

Бібліотека Mockito, є інструментом для створення мок-об'єктів, або імітацій об'єктів, для використання у тестах. Мок-об'єкти використовуються для ізоляції тестованих компонентів від їхніх залежностей, що дозволяє ефективно та незалежно перевіряти їхню поведінку.

AssertJ – це бібліотека, яка надає зручні засоби для написання тестів в мові Java. Вона включає в себе набір методів для перевірки умов та структури об'єктів у тестах, що дозволяє створювати зрозумілі та ефективні тести.

Використання цих інструментів разом з JUnit допомагає забезпечити ефективне та надійне модульне тестування програмного забезпечення, що в свою чергу сприяє підвищенню якості коду та робочої стабільності системи.

4.2 Мануальне тестування системи

Для забезпечення надійності та коректної роботи Telegram-бота для замовлення таксі було проведено ряд мануальних тестів [19]. Основні аспекти, які було перевірено в процесі мануального тестування, включали:

- проведена перевірка відображення інтерфейсу для нового користувача;
- тестування функціональності замовлення таксі;
- перевірено коректність помилки, якщо користувач обере однакову адресу для початкової та кінцевої точки маршруту;
- перевірка відображення історії поїздок;
- перевірка коректного відображення історії замовлень водія;
- перевірка прийнята замовлення водієм;

- тестування створення квитка для менеджера;
- перевірка коректного відображення активних тікетів для менеджера;
- перевірка спілкування користувача з водієм;
- перевірка спілкування користувача з менеджером.

Нижче наведено приклад тестування деяких аспектів:

Під час мануального тестування було проведено перевірку відображення інтерфейсу для нового користувача. Під час цього тесту перевірялося, чи відображається інтерфейс коректно та з усіма необхідними елементами і кнопками меню авторизації (рис 4.3). Впевнено, що новий користувач може взаємодіяти з інтерфейсом та виконувати необхідні дії для подальшого використання системи.

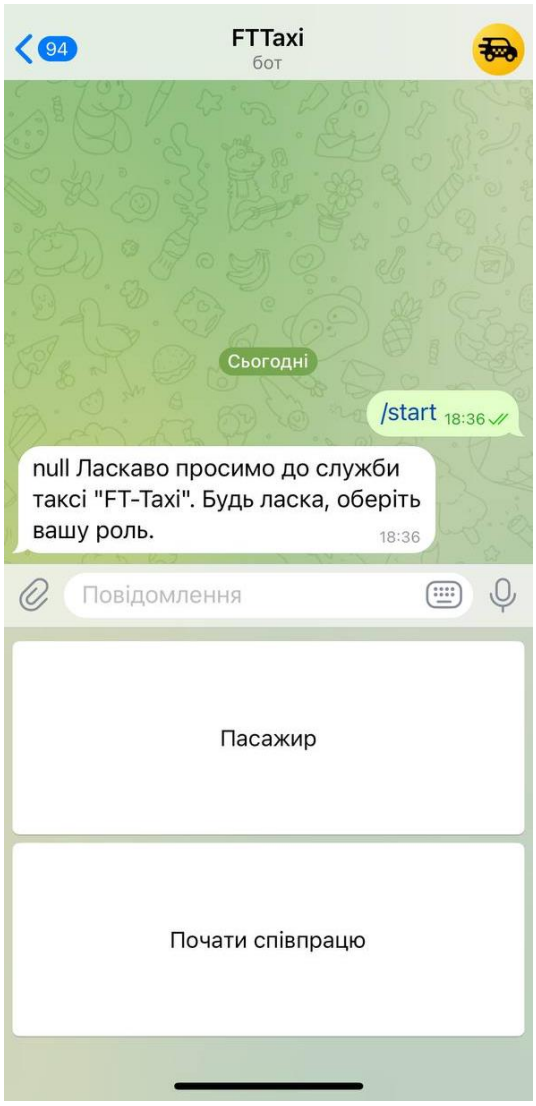


Рисунок 4.3 – Відображення інтерфейсу для нового користувача

Проведено тести на різноманітні сценарії замовлення таксі через відправлення повідомлень у бота. Під час тестування особлива увага приділялася перевірці правильності розпізнавання місця призначення. Було перевірено, чи коректно бот інтерпретує та обробляє введені дані щодо місця, до якого слід прибути. Результати тестування було здійснено відповідно до заздалегідь визначених сценаріїв, зокрема у ситуаціях, коли вказане місце призначення було невідомим або неоднозначним (рис 4.4).

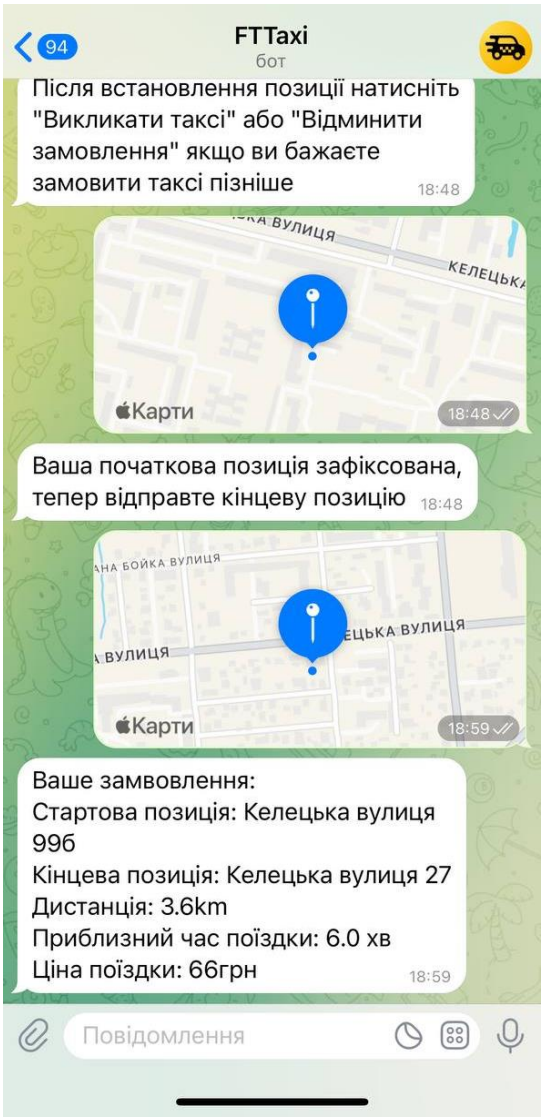


Рисунок 4.4 – Перевірка коректної інформації про замовлення

Під час тестування також була перевірена коректність повідомлення про помилку у випадку, якщо користувач обере однакову адресу для початкової та

кінцевої точки маршруту. Метою цього тесту було переконатися, що система адекватно реагує на такий випадок та надсилає зрозуміле та інформативне повідомлення про помилку, що допомагає користувачу усвідомити причину невдачі та виправити її (рис 4.5).

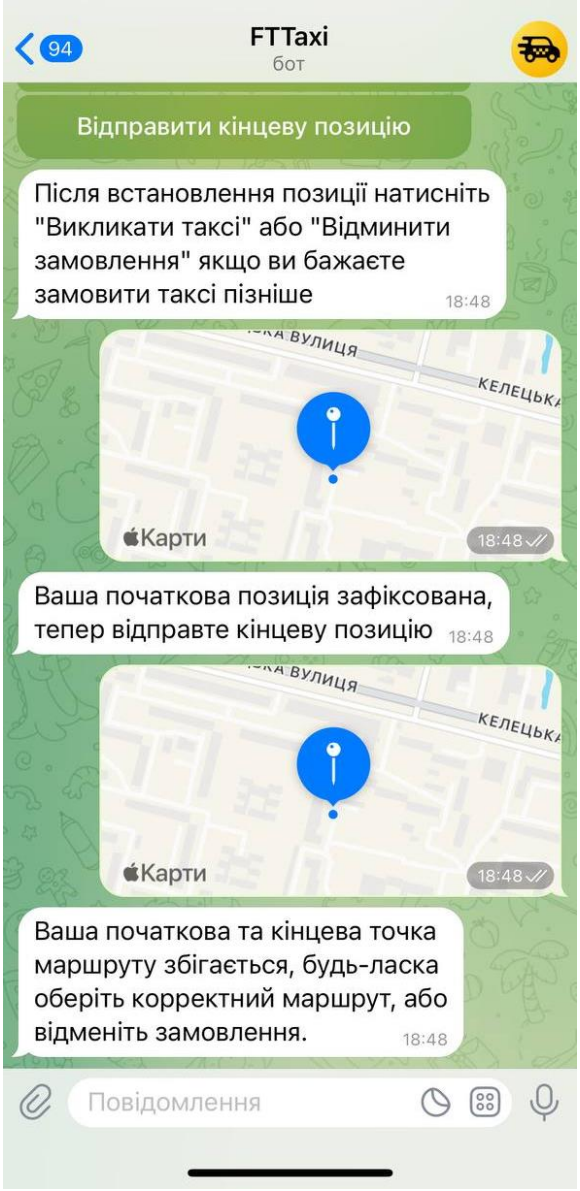


Рисунок 4.5 – Повідомлення про некоректно встановленні точки маршруту

Було проведено перевірку коректності відображення інформації після того, як водій прийняв замовлення. Метою цього тесту було переконатися, що система відображає актуальну та достовірну інформацію про замовлення після того, як водій підтвердив свою готовність виконати його. Перевірка включала

аналіз різних аспектів інформації, таких як номер замовлення, маршрут, час очікування та інші важливі дані, що стосуються замовлення. Результатом тестування було переконання у тому, що вся інформація коректно відображається та оновлюється після підтвердження водієм замовлення (рис 4.6).

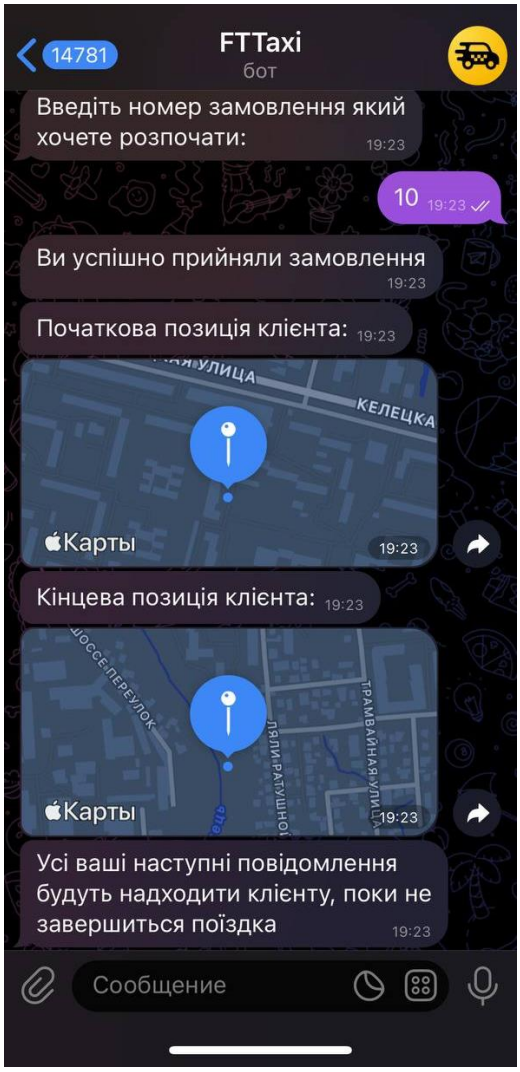


Рисунок 4.6 – Перевірка надходження необхідної інформації водію після прийняття замовлення

Було проведено перевірку того, чи надходить необхідна інформація до клієнта після прийняття замовлення. Метою цього тесту було переконатися, що система коректно повідомляє клієнта про підтвердження його замовлення водієм. Під час перевірки аналізувалася доставка повідомлення клієнту, що містить інформацію про прийняття його замовлення водієм, включаючи номер

замовлення, можливість клієнта зв’язатись з водієм. Результатом тесту було переконання у тому, що клієнт отримує необхідну інформацію після того, як його замовлення було прийнято водієм (рис 4.7).

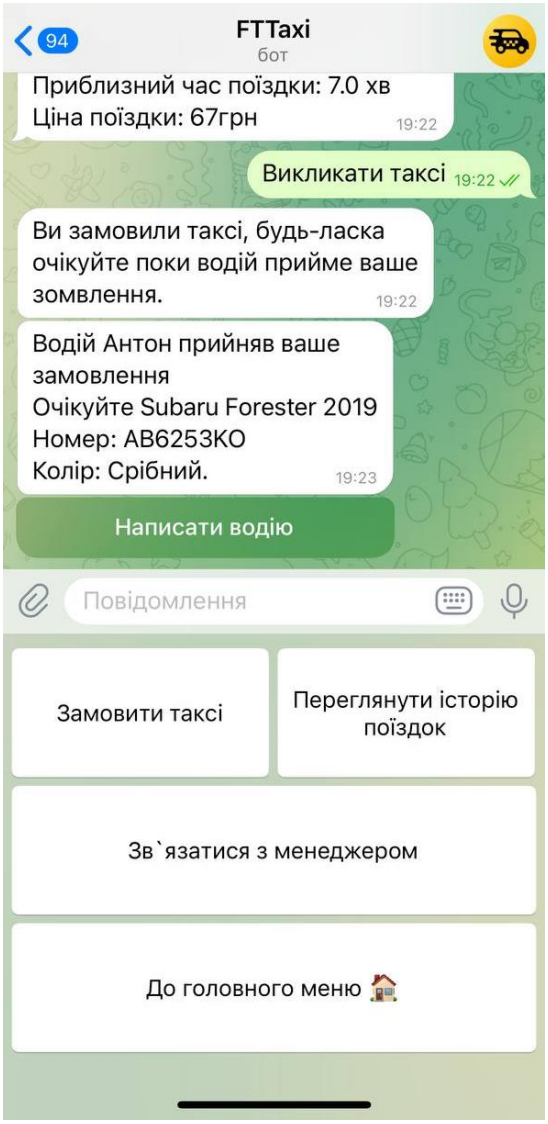


Рисунок 4.7 – Перевірка надходження необхідної інформації клієнту після прийняття замовлення водієм

Під час проведення цього тесту було перевірено процес створення запитів для менеджерів у системі. Метою тесту було переконатися, що функціонал створення квитків працює належним чином і надає клієнтам можливість ефективно створювати нові запити. Під час перевірки аналізувалася правильність процесу створення квитка, включаючи заповнення необхідної

інформації. Результатом тесту було переконання у тому, що функціонал створення квитків працює коректно і надає необхідну інформацію для подальшого вирішення проблеми або обробки запиту менеджером (рис 4.8).

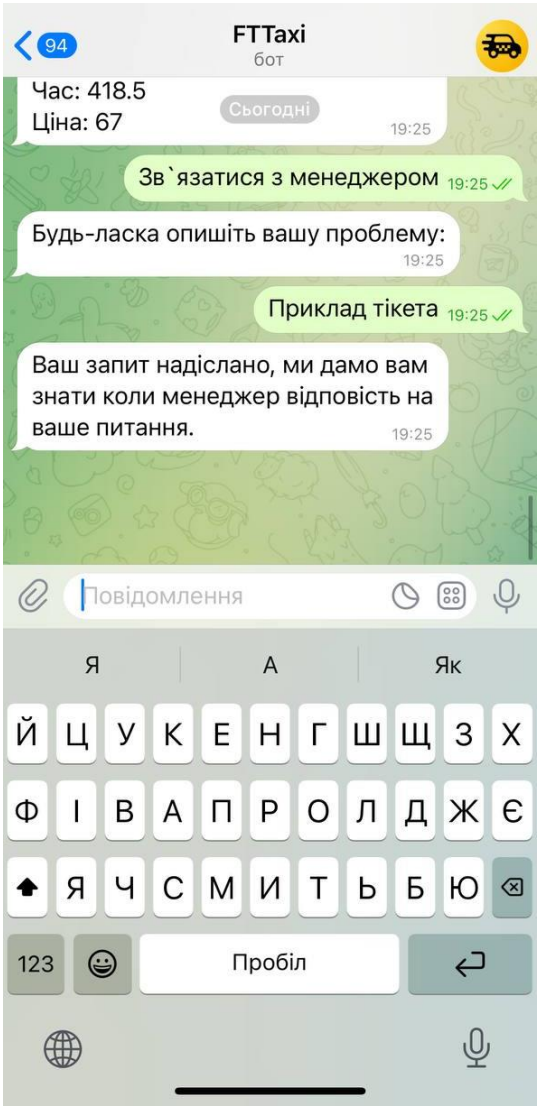


Рисунок 4.8 – Перевірка створення квитка для менеджера

Під час цього тесту проводилася перевірка того, чи правильно програма відображає активні квитки для менеджерів і чи забезпечує їм можливість ефективно взаємодіяти з ними. Основною метою тесту було переконатися, що менеджери можуть легко знаходити активні квитки, оцінювати їх стан та вживати необхідні дії. Під час перевірки аналізувалася правильність відображення інформації про кожен квиток, доступність необхідних опцій для взаємодії з квитками та загальна зручність інтерфейсу для менеджерів.

Результатом тесту було переконання у тому, що програма надає зручний та інтуїтивно зрозумілий інтерфейс для роботи з активними квитками, що сприяє ефективній роботі менеджерів з системою (рис 4.9).

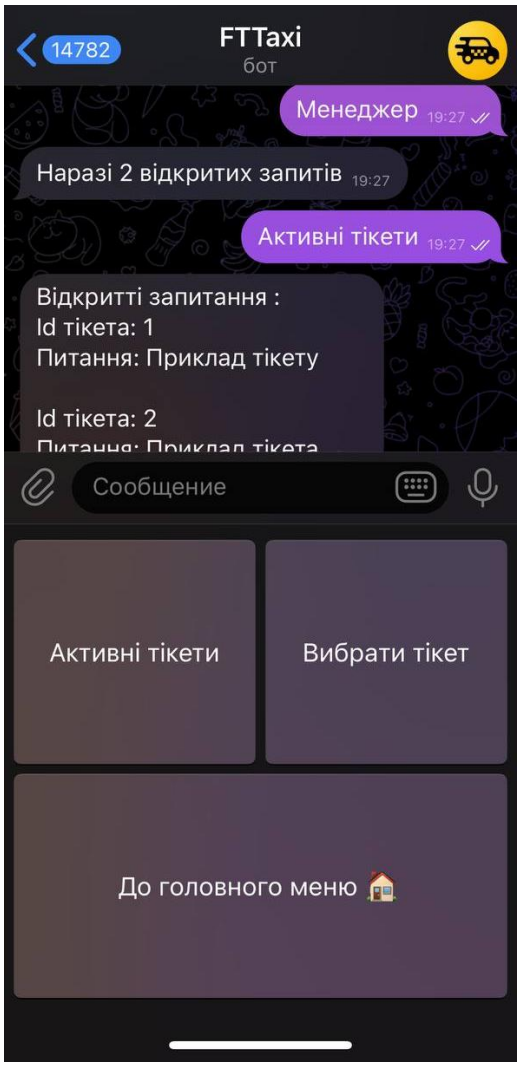


Рисунок 4.9 – Перевірка коректного відображення активних тікетів для менеджера

Під час цього тесту була проведена перевірка функціональності спілкування між користувачами та водіями через програмний інтерфейс. Основною метою тестування було впевнитися, що користувачі можуть ефективно зв'язуватися з водіями за допомогою інтерфейсу бота, а також щоб перевірити правильність відображення та обробки повідомлень водіями. Під час тестування аналізувалася швидкість та надійність доставки повідомлень,

правильність відображення інформації про статус та деталі замовлення, а також загальна зручність інтерфейсу для користувачів під час спілкування з водіями. Результатом тесту було переконання у тому, що функціональність спілкування користувачів з водіями працює належним чином і забезпечує зручну та ефективну комунікацію між сторонами (рис 4.10).

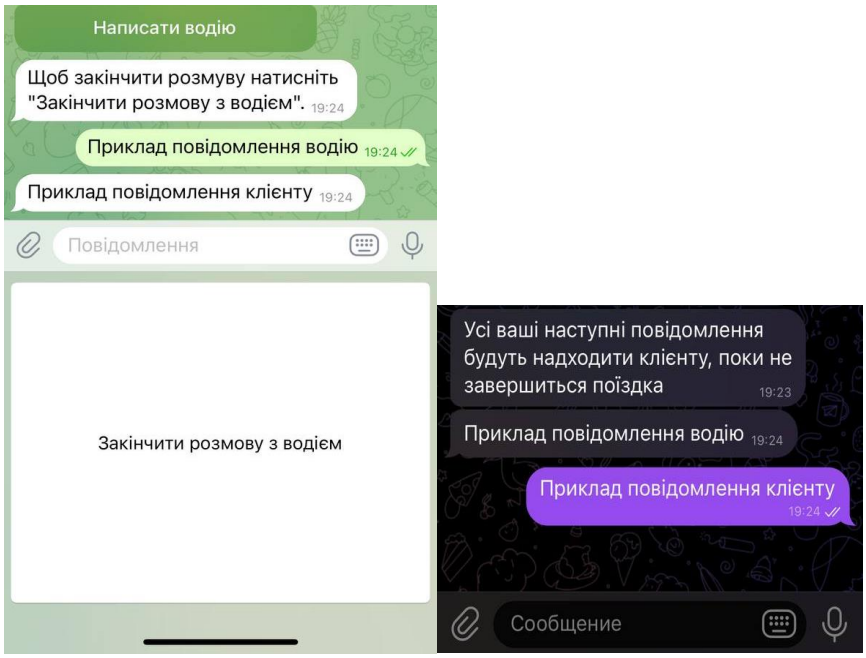


Рисунок 4.10 – Перевірка спілкування користувача з водієм

Проведене модульне тестування відіграло ключову роль у забезпеченні якості та надійності системи. Цей етап випробувань дозволив виявити та усунути помилки та неузгодженості на рівні окремих модулів та компонентів програмного забезпечення. Результатом було поліпшення функціональності системи та підвищення її загальної продуктивності.

4.3 Розробка інструкції користувача

Інструкція користувача передбачає, що користувач має встановленний застосунок «Telegram» та визначає технічних вимог для його запуску. Деталі щодо мінімальної та рекомендованої конфігурації смартфону можна знайти в таблицях 4.1 та 4.2.

Таблиця 4.1 – Мінімальна конфігурація

Операційна система	Android 4.1 або iOS 9.0 і вище
Процесор	1 ГГц
Оперативна пам'ять	1 ГБ
Місце на диску	Вільне місце для встановлення застосунку

Таблиця 4.2 – Рекомендована конфігурація

Операційна система	Android 5.0 або iOS 12.0 і вище
Процесор	2 ГГц
Оперативна пам'ять	2 ГБ і більше
Місце на диску	Рекомендовано більше вільного місця для зберігання даних

Для початку користувачу потрібно відкрити Telegram і у полі пошуку ввести назву бота «ft_taxi_bot». Після пошуку користувач повинен обрати бота з логотипом та назвою (рис. 4.11).

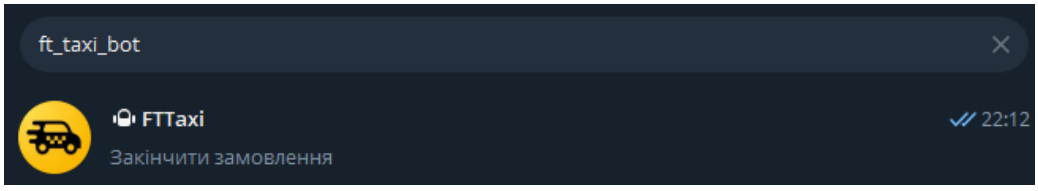


Рисунок 4.11 – Пошук бота в системі Telegram

Після входу до бота, користувач побачить меню авторизації з кнопками, що залежать від його ролі. Далі потрібно обрати свою роль і подальші дії зі списку опцій. Якщо обрати «замовити таксі», потрібно вибрати на карті початкову та кінцеву позицію подорожі. Після цього користувач побачить інформацію про поїздку, таку як стартова позиція, кінцева позиція, дистанція, час поїздки та ціна. Коли водій погодить замовлення, користувач побачить

інформацію про нього і зможе з ним зв'язатися. Також можна переглянути історію своїх поїздок та зв'язатися з менеджером для будь-яких запитань чи пропозицій

4.4 Висновки

У четвертому розділі було проведено модульне та мануальне тестування Telegram-бота для системи управління служби таксі. У процесі модульного тестування було протестовано: перевірка правильного відображення інформації про маршрут, перевірка правильності прийняття квитка, тестування коректного зчитування кінцевої позиції, перевірка коректного зчитування початкової позиції, тестування процесу реєстрації користувачів, перевірка процесу прийняття замовлення, тестування отримання правильних назв вулиць. У процесі мануального тестування було проведено перевірку функціональності: замовлення таксі, взаємодію бота з користувачем на різні команди та запити, перевірка безпеки системи, перевірка коректної роботи переписки між користувачами у боті. Також було розроблено інструкцію користувача по початковій експлуатації програмного продукту.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено Telegram-бот для системи управління служби таксі. Розроблений бот має на меті удосконалення процесу замовлення та управління таксі за рахунок автоматизації та спрощення взаємодії між клієнтами та водіями. Це сприятиме підвищенню ефективності роботи служби таксі та її конкурентоспроможності серед подібних рішень. Важливо зазначити, що наразі на ринку відсутні подібні Telegram-боти, що робить цей проект інноваційним і перспективним.

Було розглянуто стан розвитку телеграм-ботів для систем управління роботою служб таксі, проведено порівняльний аналіз додатків-аналогів, проаналізовано методи розв’язання задачі, та поставлені задачі для успішної реалізації Telegram-бота.

Було спроектовано модель даних та розроблено структуру програмного застосунку. Було розроблено загальний алгоритм роботи застосунку; оптимізовано процес видачі замовлень водіям, забезпечуючи ефективність та швидкість реагування на замовлення клієнтів; уперше запропоновано автоматичне призначення пріоритетів замовлень водіям з урахуванням їхньої доступності та рейтингу, що підвищує ефективність розподілу замовлень та задоволення клієнтів.

Було розглянуто переваги й недоліки популярних мов програмування та середовищ розробки. На основі аналізу для розробки обрано середовище Intellij IDEA [20] та мову програмування Java. Описано процес розробки основних модулів програми: зчитування команд користувача, отримання інформації про маршрут, отримання номеру будинку.

Було створено ряд тест-кейсів для тестування програми. За результатами тестування проблем не виявлено. Також було розроблено інструкцію користувача застосунку та визначено системні вимоги.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Telegram: корисна добірка ботів [Електронний ресурс] – Режим доступу до ресурсу: www.kiwiagency.com.ua/blog/pomoschniki-v-telegram (дата звернення: 15.03.2024).

2. Популярність месенджерів [Електронний ресурс] - Режим доступу до ресурсу: https://gerabot.com/article/populyarnist_mesendzheriv (дата звернення: 15.03.2024).

3. Юрченко А. О. Переваги та недоліки телеграм боту порівняно з мобільним застосунком / А. О. Юрченко, О. В. Романюк // Матеріали LIII Науково-технічної конференції підрозділів Вінницького національного технічного університету, Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20453/16974

4. Юрченко А. О. Розробка telegram-бота для системи управління роботою служби таксі / А. О. Юрченко, О. В. Романюк // Молодь в науці: дослідження, проблеми, перспективи (МН-2024) [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/mn2024/paper/view/21641

5. Uber [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Uber (дата звернення: 17.03.2024).

6. Grab [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Grab (дата звернення: 17.03.2024).

7. Bolt [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Bolt (дата звернення: 17.03.2024).

8. Java tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/tutorial/ (дата звернення: 19.03.2024).

9. Spring boot [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/guides/gs/spring-boot (дата звернення: 20.03.2024).

10. Telegram Bot API [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/api (дата звернення: 21.03.2024).

11. What is a UML diagram [Електронний ресурс] – Режим доступу до ресурсу: https://miro.com/diagramming/what-is-a-uml-diagram/ (дата звернення: 30.03.2024).

12. Переваги та недоліки використання Spring Boot [Електронний ресурс] – Режим доступу до ресурсу: https://javarush.com/ua/groups/posts/uk (дата звернення: 01.04.2024).

13. Будуємо телеграм чат-бот [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/38358/ (дата звернення: 14.04.2024).

14. Telegram Bot Features [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/features (дата звернення: 14.04.2024).

15. OpenStreeaMap API [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.openstreetmap.org/wiki/API (дата звернення: 17.04.2024).

16. JSON Object Literals [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/js/js_json_objects.asp (дата звернення: 17.04.2024).

17. What is JSON [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/whatis/whatis_json.asp (дата звернення: 17.04.2024).

18. Effective Unit Testing: A guide for Java developers. Сполучені Штати Америки : Manning Publications, 2020. 248 с. (дата звернення: 15.05.2024).

19. Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing. Сполучені Штати Америки : Elisabeth Hendrickson, 2013. 256 с. (дата звернення: 17.05.2024).

20. IntelliJ IDEA [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/IntelliJ_IDEA (дата звернення: 20.05.2024).

ДОДАТКИ

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., проф. О. Н. Романюк
" 12 " березня 2024 р.

Технічне завдання
на бакалаврську дипломну роботу
«Розробка Telegram-бота для системи управління служби таксі»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
к.т.н., доц. О.В.Романюк
" 12 " березня 2024 р.

Виконав:
студент гр. 1ПІ-20Б А.О. Юрченко
" 12 " березня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка Telegram-бота для системи управління служби таксі».

Галузь застосування – Telegram-бот.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від «11» березня 2024 р. ректора по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою роботи є покращення процесу управління служби таксі шляхом розроблення спеціалізованого Telegram-бота.

Призначення роботи – розробка та реалізація програмного забезпечення Telegram-бота для системи управління служби таксі.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР:

1. Effective Java. Сполучені Штати Америки : Joshua Bloch, 2017р. 412 с.
2. Building Telegram Bots: Develop Bots in 12 Programming Languages using the Telegram Bot API. Сполучені Штати Америки : Nicolas Modrzyk, 2017р. 210с.
3. Spring in Action. Сполучені Штати Америки : Craig Walls, 2021р. 500с.

5. Технічні вимоги

Тип програми – Telegram-бот; модель розробки – ітеративна; засоби організації дних програми – фреймворк Spring та бібліотека TelegramAPI; вхідні данні: команда користувача, початкова та кінцева координата поїздки,

повідомлення між користувачами; середовище розробки Intellij IDEA; мова програмування – Java.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БДР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз стану питання та постановка задач дослідження	14.03.24 – 17.03.24
2	Розробка структури та моделі застосунку	18.03.24 – 28.03.24
3	Розробка загального алгоритму роботи програми	29.03.24 – 01.04.24
4	Розробка методу та алгоритму видачі замовлень водіям	02.04.24 – 04.04.24
5	Розробка методу та алгоритму автоматичного призначення замовлень водіям	05.04.24 – 07.04.24
6	Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення	08.04.24 – 12.04.24
7	Розробка інтерфейсу програмного застосунку	13.04.24 – 15.04.24
8	Програмна реалізація Telegram-боту	16.04.24 – 12.05.24
9	Тестування програмного застосунку	13.05.24 – 20.05.24
10	Оформлення матеріалів до захисту БДР	21.05.24 – 30.05.24

10. Порядок контролю та прийняття

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б

Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка Telegram-бота для системи управління служби таксі»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доцент кафедри ПЗ Романюк О.В.

Оригінальність	92,9%
Схожість	7,1%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Юрченко А.О.

Керівник роботи _____ Романюк. О.В.

ДОДАТОК В
Лістинг програми

```
package com.example.kursach;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class KursachApplication {

    public static void main(String[] args) {
        SpringApplication.run(KursachApplication.class, args);
    }

}

package com.example.kursach.bot;

import com.example.kursach.configuration.TelegramBotConfiguration;
import com.example.kursach.services.command.CallBackQueryCommandService;
import com.example.kursach.services.command.TextCommandService;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.bots.TelegramLongPollingBot;
import org.telegram.telegrambots.meta.api.methods.send.SendLocation;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;

import javax.annotation.Resource;

@Component
public class TelegramBot extends TelegramLongPollingBot {

    @Resource
    private TelegramBotConfiguration telegramBotConfiguration;
    @Resource
    private TextCommandService textCommandService;
    @Resource
    private CallBackQueryCommandService callBackQueryCommandService;

    public TelegramBot(TelegramBotConfiguration telegramBotConfiguration) {
        super(telegramBotConfiguration.getBotToken());
    }

    @Override
    public String getBotUsername() {
        return telegramBotConfiguration.getBotName();
    }

    @Override
    public void onUpdateReceived(Update update) {
        if (!update.hasCallbackQuery()) {
            textCommandService.processTextCommand(update);
        } else callBackQueryCommandService.processCallBackQueryCommand(update);
    }

    public void executeMessage(SendMessage sendMessage) {
        try {
            execute(sendMessage);
        } catch (TelegramApiException e) {
            throw new RuntimeException(e);
```

```
        }
    }

    public void executeLocation(SendLocation sendLocation) {
        try {
            execute(sendLocation);
        } catch (TelegramApiException e) {
            throw new RuntimeException(e);
        }
    }

    public void notSupportedCommand(Long chatId) {
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText("Невідома команда");
        executeMessage(sendMessage);
    }
}
package com.example.kursach.buttons.authorization;

import com.example.kursach.enums.UserRole;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;

import java.util.List;
import java.util.Map;

@Component
public class AuthorizationButtons {

    private final Map<UserRole, List<KeyboardButton>> userRoleButtonsMap = Map.of(
            UserRole.Administrator, List.of(addPassengerRoleButton(), addDriverRoleButton(),
addManagerRoleButton()),
            UserRole.Manager, List.of(addPassengerRoleButton(), addManagerRoleButton()),
            UserRole.Driver, List.of(addPassengerRoleButton(), addDriverRoleButton()),
            UserRole.Passenger, List.of(addPassengerRoleButton())
    );

    public KeyboardRow showAuthorizationButtons (UserRole userRole) {
        return new KeyboardRow(userRoleButtonsMap.get(userRole));
    }

    private KeyboardButton addPassengerRoleButton () {
        return KeyboardButton.builder()
                .text("Пасажир")
                .build();
    }
    private KeyboardButton addDriverRoleButton () {
        return KeyboardButton.builder()
                .text("Водій")
                .build();
    }
    private KeyboardButton addManagerRoleButton () {
        return KeyboardButton.builder()
                .text("Менеджер")
                .build();
    }
}
package com.example.kursach.buttons.authorization;

import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardButton;
```

```
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;

import java.util.List;

@Component
public class DriverButtons {

    public KeyboardRow showDriverButtons () {
        return new KeyboardRow(List.of(addOrderButton(), addPassengerHistoryButton()));
    }
    public KeyboardRow showDriverManagerButton () {
        return new KeyboardRow(List.of(addPassengerWithCSButton()));
    }
    public KeyboardRow showDriverAcceptButton () {
        return new KeyboardRow(List.of(addAcceptButton()));
    }

    private KeyboardButton addOrderButton () {
        return KeyboardButton.builder()
                .text("Активні замовлення")
                .build();
    }
    private KeyboardButton addPassengerHistoryButton () {
        return KeyboardButton.builder()
                .text("Історія замовлень")
                .build();
    }
    private KeyboardButton addPassengerWithCSButton () {
        return KeyboardButton.builder()
                .text("Зв`язатися з менеджером")
                .build();
    }
    private KeyboardButton addAcceptButton () {
        return KeyboardButton.builder()
                .text("Прийняти замовлення")
                .build();
    }

}
package com.example.kursach.buttons.authorization;

import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;

import java.util.List;

@Component
public class MainMenuButton {

    public KeyboardRow showMainMenuButton () {
        return new KeyboardRow(List.of(mainMenu()));
    }

    private KeyboardButton mainMenu () {
        return KeyboardButton.builder()
                .text("До головного меню \uD83C\uDFE0")
                .build();
    }
}
package com.example.kursach.buttons.authorization;

import org.springframework.stereotype.Component;
```

```
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;

import java.util.List;

@Component
public class ManagerButtons {

   public KeyboardRow showManagerButtons () {
      return new KeyboardRow(List.of(addActiveTicketsButton(), chooseTicket()));
   }

   private KeyboardButton addActiveTicketsButton () {
      return KeyboardButton.builder()
            .text("Активні тікети")
            .build();
   }

   private KeyboardButton chooseTicket () {
      return KeyboardButton.builder()
            .text("Вибрати тікет")
            .build();
   }
}
package com.example.kursach.buttons.authorization;

import com.example.kursach.enums.UserRole;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;

import java.awt.*;
import java.util.List;

@Component
public class NewEmployeesButton {

   public KeyboardRow showRegistrationButton () {
      return new KeyboardRow(List.of(registrationEmployeesButton()));
   }

   private KeyboardButton registrationEmployeesButton () {
      return KeyboardButton.builder()
            .text("Почати співпрацю")
            .build();
   }
}
package com.example.kursach.buttons.authorization;

import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;

import java.util.List;

@Component
public class PassengerButtons {
   public KeyboardRow showPassengerButtons () {
      return new KeyboardRow(List.of(addOrderButton(), addPassengerHistoryButton()));
   }
   public KeyboardRow showPassengerManagerButton () {
      return new KeyboardRow(List.of(addPassengerWithCSButton()));
   }
```

```
    private KeyboardButton addOrderButton () {
        return KeyboardButton.builder()
                .text("Замовити таксі")
                .build();
    }
    private KeyboardButton addPassengerHistoryButton () {
        return KeyboardButton.builder()
                .text("Переглянути історію поїздок")
                .build();
    }
    private KeyboardButton addPassengerWithCSButton () {
        return KeyboardButton.builder()
                .text("Зв`язатися з менеджером")
                .build();
    }
}
package com.example.kursach.command.controller;

import com.example.kursach.bot.TelegramBot;
import com.example.kursach.enums.UserAction;
import com.example.kursach.enums.UserState;
import com.example.kursach.services.entity.UserService;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;

import javax.annotation.Resource;
import javax.inject.Inject;
import javax.inject.Singleton;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.function.BiConsumer;

@Component
@Singleton
public class CommandControllerForCallBackQuery {

    private final Map<String, BiConsumer<Long, Update>> commandTextMap = new HashMap<>();

    @Resource
    private TelegramBot telegramBot;

    @Resource
    private UserService userService;

    @Inject
    public CommandControllerForCallBackQuery() {
        commandTextMap.put("/startPosition", this::setStartPositionAction);
        commandTextMap.put("/endPosition", this::setEndPositionAction);
        commandTextMap.put("/answerToManager", this::answerToManager);
        commandTextMap.put("/answerToClient", this::answerManagerToClient);
        commandTextMap.put("/sendMassageToDriver", this::sendMassageToDriver);
    }

    public void processCommand(String command, Long chatId, Update update) {
        if (commandTextMap.get(command) == null) {
            telegramBot.notSupportedCommand(chatId);
        } else {
```

```
            commandTextMap.get(command).accept(chatId, update);
        }
    }

    private void setStartPositionAction(Long chatId, Update update){
        userService.updateCurrentAction(chatId, UserAction.SET_START_POSITION.name());
    }

    private void setEndPositionAction(Long chatId, Update update){
        userService.updateCurrentAction(chatId, UserAction.SET_END_POSITION.name());
    }

    private void answerToManager(Long chatId, Update update){
        userService.updateCurrentState(chatId, UserState.CHAT_WITH_MANAGER.name());
    }

    private void answerManagerToClient(Long chatId, Update update){
        userService.updateCurrentState(chatId, UserState.CHAT_MANAGER_WITH_PASSENGER.name());
    }

    private void sendMassageToDriver(Long chatId, Update update){
        userService.updateCurrentState(chatId, UserState.CHAT_WITH_DRIVER.name());

        SendMessage chatWithDriver = new SendMessage();
        chatWithDriver.setChatId(chatId);
        chatWithDriver.setText("Щоб закінчити розмуву натисніть \"Закінчити розмову з водієм\".");
        ReplyKeyboardMarkup inlineKeyboardMarkup = ReplyKeyboardMarkup.builder()
                .keyboard(List.of(
                        new KeyboardRow(List.of(closeChat()))
                ))
                .build();
        chatWithDriver.setReplyMarkup(inlineKeyboardMarkup);
        telegramBot.executeMessage(chatWithDriver);

    }

    private KeyboardButton closeChat () {
        return KeyboardButton.builder()
                .text("Закінчити розмову з водієм")
                .build();
    }
}
package com.example.kursach.command.controller;

import com.example.kursach.buttons.authorization.*;
import com.example.kursach.bot.TelegramBot;
import com.example.kursach.entities.OrderEntity;
import com.example.kursach.entities.TicketsEntity;
import com.example.kursach.entities.UserEntity;
import com.example.kursach.enums.*;
import com.example.kursach.services.entity.OrderService;
import com.example.kursach.services.entity.TicketsService;
import com.example.kursach.services.entity.UserService;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.InlineKeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;
```

```
import javax.annotation.Resource;
import javax.inject.Inject;
import javax.inject.Singleton;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.function.BiConsumer;

@Component
@Singleton
public class CommandControllerForText {

    private final Map<String, BiConsumer<Long, Update>> commandTextMap = new HashMap<>();

    private final Map<UserRole, BiConsumer<Long, Update>> userCurrentRoleShow = new HashMap<>();

    @Resource
    private TelegramBot telegramBot;
    @Resource
    private AuthorizationButtons authorizationButtons;
    @Resource
    private NewEmployeesButton newEmployeesButton;
    @Resource
    private PassengerButtons passengerButtons;
    @Resource
    private DriverButtons driverButtons;
    @Resource
    private ManagerButtons managerButtons;
    @Resource
    private MainMenuButton mainMenuButton;
    @Resource
    private UserService userService;
    @Resource
    private OrderService orderService;
    @Resource
    private TicketsService ticketsService;
    @Resource
    private UserActionController userActionController;

    @Inject
    public CommandControllerForText() {
        commandTextMap.put("/start", this::getAuthorizationPage);
        commandTextMap.put("Пасажир", this::defaultPassengerButton);
        commandTextMap.put("Переглянути історію поїздок", this::userHistory);
        commandTextMap.put("Зв`язатися з менеджером", this::callManager);
        commandTextMap.put("Замовити таксі", this::createOrder);
        commandTextMap.put("Викликати таксі", this::acceptOrderPassengerButton);
        commandTextMap.put("Відмінити замовлення", this::dismissOrderPassengerButton);
        commandTextMap.put("Закінчити розмову з водієм", this::closeChatWithDriver);
        commandTextMap.put("До головного меню \uD83C\uDFE0", this::getAuthorizationPage);
        commandTextMap.put("Водій", this::defaultDriverButton);
        commandTextMap.put("Історія замовлень", this::userHistory);
        commandTextMap.put("Активні замовлення", this::activeOrders);
        commandTextMap.put("Прийняти замовлення", this::chooseDriverOrder);
        commandTextMap.put("Закінчити замовлення", this::closeOrder);
        commandTextMap.put("Менеджер", this::defaultManagerButton);
        commandTextMap.put("Активні тікети", this::activeManagerTickets);
        commandTextMap.put("Вибрати тікет", this::chooseManagerTicket);
        commandTextMap.put("Назад ⬅\uFE0F", this::back);
        commandTextMap.put("Закрити тікет", this::closeTicket);
        userCurrentRoleShow.put(UserRole.Passenger, this::defaultPassengerButton);
        userCurrentRoleShow.put(UserRole.Driver, this::defaultDriverButton);
```

```
        userCurrentRoleShow.put(UserRole.Manager, this::defaultManagerButton);
    }

    public void processCommand(String command, Long chatId, Update update) {
        if (commandTextMap.get(command) == null) {
            telegramBot.notSupportedCommand(chatId);
        } else {
            commandTextMap.get(command).accept(chatId, update);
        }
    }

    private void getAuthorizationPage(Long chatId, Update update) {
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText(update.getMessage().getFrom().getUserName() + " Ласкаво просимо до служби
таксі \"FT-Taxi\". Будь ласка, оберіть вашу роль.");

        Optional<UserEntity> userOpt = userService.getUserByChatId(chatId);
        UserEntity user;
        if (userOpt.isEmpty()) {
            UserEntity newUser = new UserEntity();
            newUser.setUserChatId(chatId);
            newUser.setUserRole(UserRole.Passenger.name());
            newUser.setUserCurrentState(UserState.DEFAULT.name());
            newUser.setUserCurrentAction(UserAction.DEFAULT.name());
            newUser.setUserCurrentRole(UserRole.Passenger.name());
            userService.saveUser(newUser);
            user = newUser;
        } else user = userOpt.get();
        user.setUserCurrentAction(UserAction.DEFAULT.name());
        user.setUserCurrentState(UserState.DEFAULT.name());
        userService.saveUser(user);
        ReplyKeyboardMarkup inlineKeyboardMarkup = ReplyKeyboardMarkup.builder()
                .keyboard(List.of(
                        authorizationButtons.showAuthorizationButtons(UserRole.valueOf(user.getUserRole())),
                        newEmployeesButton.showRegistrationButton()
                ))
                .build();

        sendMessage.setReplyMarkup(inlineKeyboardMarkup);
        telegramBot.executeMessage(sendMessage);
    }

    // PASSENGER

    private void defaultPassengerButton(Long chatId, Update update) {
        UserEntity user = userService.getUserByChatId(chatId).orElseThrow();
        user.setUserCurrentRole(UserRole.Passenger.name());
        user.setUserCurrentState(UserState.DEFAULT.name());
        userService.saveUser(user);
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText("Вітаємо вас у службі таксі \"FT-Taxi\". Бажаєте зробити замовлення ? ");
        ReplyKeyboardMarkup inlineKeyboardMarkup = ReplyKeyboardMarkup.builder()
                .keyboard(List.of(
                        passengerButtons.showPassengerButtons(),
                        passengerButtons.showPassengerManagerButton(),
                        mainMenuButton.showMainMenuButton()
                ))
                .build();

        sendMessage.setReplyMarkup(inlineKeyboardMarkup);
```

```
    telegramBot.executeMessage(sendMessage);
}

private void userHistory(Long chatId, Update update) {
    UserEntity user = userService.getUserByChatId(chatId).orElseThrow();
    List<OrderEntity> orderEntities = user.getUserCurrentRole().equals(UserRole.Passenger.name())
            ? orderService.findUserHistory(chatId, OrderStatus.CANCELLED.name())
            : orderService.findDriverHistory(chatId, OrderStatus.CANCELLED.name());
    SendMessage sendMessage = new SendMessage();
    sendMessage.setChatId(chatId);
    sendMessage.setText(getOrders(orderEntities, "Історія ваших замовлень:"));
    telegramBot.executeMessage(sendMessage);
}

private void callManager(Long chatId, Update update) {
    SendMessage sendMessage = new SendMessage();
    sendMessage.setChatId(chatId);
    sendMessage.setText("Будь-ласка опишіть вашу проблему: ");
    userService.updateCurrentState(chatId, UserState.FIRST_MASSAGE_TO_MANAGER.name());

    ReplyKeyboardMarkup inlineKeyboardMarkup = ReplyKeyboardMarkup.builder()
            .keyboard(List.of(
                    new KeyboardRow(List.of(backButton()))
            ))
            .build();

    sendMessage.setReplyMarkup(inlineKeyboardMarkup);
    telegramBot.executeMessage(sendMessage);
}

// PASSENGER ORDER

private void createOrder(Long chatId, Update update) {
    SendMessage sendMessage = new SendMessage();
    sendMessage.setChatId(chatId);
    sendMessage.setText("Будь-ласка встановіть стартову та кінцеву позицію маршута");

    InlineKeyboardButton startPosition = new InlineKeyboardButton();
    startPosition.setCallbackData("/startPosition");
    startPosition.setText("Відправити стартову позицію");

    InlineKeyboardButton endPosition = new InlineKeyboardButton();
    endPosition.setCallbackData("/endPosition");
    endPosition.setText("Відправити кінцеву позицію");

    List<InlineKeyboardButton> startKeyboardButtons = List.of(startPosition);
    List<InlineKeyboardButton> endKeyboardButtons = List.of(endPosition);


    InlineKeyboardMarkup inlineKeyboardMarkup1 = new InlineKeyboardMarkup();
    inlineKeyboardMarkup1.setKeyboard(List.of(startKeyboardButtons, endKeyboardButtons));

    sendMessage.setReplyMarkup(inlineKeyboardMarkup1);

    try {
        telegramBot.execute(sendMessage);
        createOrderSeparateButtons(chatId);
    } catch (TelegramApiException e) {
        throw new RuntimeException(e);
    }
}

private void createOrderSeparateButtons(Long chatId) {
```

```
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText("Після встановлення позиції натисніть \"Викликати таксі\" або \"Відминити
замовлення\" якщо ви бажаєте замовити таксі пізніше");
        KeyboardButton endOrder = KeyboardButton.builder()
                .text("Викликати таксі")
                .build();
        KeyboardButton cancelOrder = KeyboardButton.builder()
                .text("Відмінити замовлення")
                .build();

        List<KeyboardButton> keyboardButtons = List.of(endOrder, cancelOrder);
        KeyboardRow keyboardRows = new KeyboardRow(keyboardButtons);

        ReplyKeyboardMarkup inlineKeyboardMarkup = ReplyKeyboardMarkup.builder()
                .keyboard(List.of(keyboardRows))
                .build();

        sendMessage.setReplyMarkup(inlineKeyboardMarkup);

        try {
            telegramBot.execute(sendMessage);
        } catch (TelegramApiException e) {
            throw new RuntimeException(e);
        }
    }

    private void acceptOrderPassengerButton(Long chatId, Update update) {
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText("Ви замовили таксі, будь-ласка очікуйте поки водій прийме ваше
зомвлення.");
        ReplyKeyboardMarkup inlineKeyboardMarkup = ReplyKeyboardMarkup.builder()
                .keyboard(List.of(
                        passengerButtons.showPassengerButtons(),
                        passengerButtons.showPassengerManagerButton(),
                        mainMenuButton.showMainMenuButton()
                ))
                .build();

        sendMessage.setReplyMarkup(inlineKeyboardMarkup);

        orderService.updateOrderStatus(chatId, OrderStatus.CREATION.name(),
OrderStatus.ACTIVE.name());
        userService.updateCurrentAction(chatId, UserAction.DEFAULT.name());

        telegramBot.executeMessage(sendMessage);
    }

    private void dismissOrderPassengerButton(Long chatId, Update update) {
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText("Ви відхилили замовлення.");
        ReplyKeyboardMarkup inlineKeyboardMarkup = ReplyKeyboardMarkup.builder()
                .keyboard(List.of(
                        passengerButtons.showPassengerButtons(),
                        passengerButtons.showPassengerManagerButton(),
                        mainMenuButton.showMainMenuButton()
                ))
                .build();

        sendMessage.setReplyMarkup(inlineKeyboardMarkup);
```

```
        orderService.updateOrderStatus(chatId, OrderStatus.CREATION.name(),
OrderStatus.CANCELLED.name());
        userService.updateCurrentAction(chatId, UserAction.DEFAULT.name());

        telegramBot.executeMessage(sendMessage);
    }

    private void closeChatWithDriver(Long chatId, Update update) {
        userService.updateCurrentState(chatId, UserState.DEFAULT.name());
        userService.updateCurrentAction(chatId, UserAction.DEFAULT.name());

        back(chatId, update);
    }

    // DRIVER

    private void defaultDriverButton(Long chatId, Update update) {
        UserEntity user = userService.getUserByChatId(chatId).orElseThrow();
        user.setUserCurrentRole(UserRole.Driver.name());
        user.setUserCurrentState(UserState.DEFAULT.name());
        userService.saveUser(user);
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText("Вітаємо вас у службі таксі \"FT-Taxi\". Бажаєте взяти замовлення ? ");
        ReplyKeyboardMarkup inlineKeyboardMarkup = ReplyKeyboardMarkup.builder()
                .keyboard(List.of(
                        driverButtons.showDriverButtons(),
                        driverButtons.showDriverAcceptButton(),
                        driverButtons.showDriverManagerButton(),
                        mainMenuButton.showMainMenuButton()
                ))
                .build();

        sendMessage.setReplyMarkup(inlineKeyboardMarkup);
        telegramBot.executeMessage(sendMessage);
    }

    private void chooseDriverOrder(Long chatId, Update update) {
        userService.updateCurrentAction(chatId, UserAction.CHOOSE_DRIVER_ACTION.name());
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText("Введіть номер замовлення який хочете розпочати:");
        telegramBot.executeMessage(sendMessage);
    }

    private void activeOrders(Long chatId, Update update) {
        List<OrderEntity> orders = orderService.findAllByOrderStatus(OrderStatus.ACTIVE.name());
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText(getOrders(orders, "Відкритті замовлення :"));
        telegramBot.executeMessage(sendMessage);
    }

    private void closeOrder(Long chatId, Update update) {
        OrderEntity orderEntity = orderService.findByDriverChatIdAndOrderStatus(chatId,
OrderStatus.PROCESSING.name()).orElseThrow();
        orderEntity.setOrderStatus(OrderStatus.FINISHED.name());
        orderService.saveOrder(orderEntity);

        userService.updateCurrentState(chatId, UserState.DEFAULT.name());
        userService.updateCurrentAction(chatId, UserAction.DEFAULT.name());

        back(chatId, update);
```

```
    }

    // MANAGER

    private void defaultManagerButton(Long chatId, Update update) {
        UserEntity user = userService.getUserByChatId(chatId).orElseThrow();
        user.setUserCurrentRole(UserRole.Manager.name());
        user.setUserCurrentState(UserState.DEFAULT.name());
        userService.saveUser(user);

        List<TicketsEntity> tickets = ticketsService.findAllByStatus(TicketStatus.Active.name());
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText("Наразі " + tickets.size() + " відкритих запитів");
        ReplyKeyboardMarkup inlineKeyboardMarkup = ReplyKeyboardMarkup.builder()
                .keyboard(List.of(
                        managerButtons.showManagerButtons(),
                        mainMenuButton.showMainMenuButton()
                ))
                .build();

        sendMessage.setReplyMarkup(inlineKeyboardMarkup);
        telegramBot.executeMessage(sendMessage);
    }


    private void activeManagerTickets(Long chatId, Update update) {
        List<TicketsEntity> tickets = ticketsService.findAllByStatus(TicketStatus.Active.name());
        StringBuilder textBuilder = new StringBuilder("Відкритті запитання :").append("\n");
        for (TicketsEntity ticket : tickets) {
            String ticketInfo = String.format(
                    "Id тікета: %s\n" +
                            "Питання: %s\n" +
                            "\n",
                    ticket.getId().toString(),
                    ticket.getTicketQuestion()
            );

            textBuilder.append(ticketInfo);
        }
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText(textBuilder.toString());
        telegramBot.executeMessage(sendMessage);
    }

    private void chooseManagerTicket(Long chatId, Update update) {
        UserEntity user = userService.getUserByChatId(chatId).orElseThrow();
        user.setUserCurrentAction(UserAction.CHOOSE_USER_ACTION.name());
        userService.saveUser(user);
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText("Введіть номер тікету який хочете ропочати:");
        telegramBot.executeMessage(sendMessage);
    }


    private void closeTicket(Long chatId, Update update) {
        TicketsEntity ticketsEntity = ticketsService.findByManagerChatIdAndTicketStatus(chatId,
TicketStatus.Processing.name()).orElseThrow();
        ticketsEntity.setTicketStatus(TicketStatus.Closed.name());
        ticketsService.save(ticketsEntity);
```

```
        userService.updateCurrentState(chatId, UserState.DEFAULT.name());
        userService.updateCurrentState(ticketsEntity.getClientChatId(), UserState.DEFAULT.name());

        back(chatId, update);
    }

    private void back(Long chatId, Update update) {
        UserEntity user = userService.getUserByChatId(chatId).orElseThrow();
        userCurrentRoleShow.get(UserRole.valueOf(user.getUserCurrentRole())).accept(chatId, update);
    }

    private KeyboardButton backButton() {
        return KeyboardButton.builder()
                .text("Назад ⬅\uFE0F")
                .build();
    }

    private String getOrders(List<OrderEntity> orders, String text) {
        StringBuilder textBuilder = new StringBuilder(text).append("\n");
        for (OrderEntity orderEntity : orders) {
            String startPosition =
userActionController.getStreetName(Double.parseDouble(orderEntity.getStartLatitudePosition()),
Double.parseDouble(orderEntity.getStartLongitudePosition()));
            String endPosition =
userActionController.getStreetName(Double.parseDouble(orderEntity.getEndLatitudePosition()),
Double.parseDouble(orderEntity.getEndLongitudePosition()));
            String orderInfo = String.format(
                    "Id: %s\n" +
                    "Стартова позиція: %s\n" +
                            "Кінцева позиція: %s\n" +
                            "Статус: %s\n" +
                            "Дистанція: %s\n" +
                            "Час: %s\n" +
                            "Ціна: %s\n" +
                            "\n",
                    orderEntity.getId().toString(),
                    startPosition,
                    endPosition,
                    orderEntity.getOrderStatus(),
                    orderEntity.getDistance(),
                    orderEntity.getDuration(),
                    orderEntity.getBill()
            );

            textBuilder.append(orderInfo);
        }

        return textBuilder.toString();
    }
}
package com.example.kursach.command.controller;

import com.example.kursach.bot.TelegramBot;
import com.example.kursach.entities.DriverCarsEntity;
import com.example.kursach.entities.OrderEntity;
import com.example.kursach.entities.TicketsEntity;
import com.example.kursach.entities.UserEntity;
import com.example.kursach.enums.OrderStatus;
import com.example.kursach.enums.TicketStatus;
import com.example.kursach.enums.UserAction;
import com.example.kursach.enums.UserState;
import com.example.kursach.services.entity.DriverCarsService;
import com.example.kursach.services.entity.OrderService;
```

```
import com.example.kursach.services.entity.TicketsService;
import com.example.kursach.services.entity.UserService;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import org.json.JSONArray;
import org.json.JSONObject;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendLocation;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Location;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.InlineKeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;

import javax.annotation.Resource;
import javax.inject.Singleton;
import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.function.BiConsumer;

@Component
@Singleton
public class UserActionController {

    private static final String DISTANCE = "distance";
    private static final String DURATION = "duration";
    private static final Integer BASE_COST = 30;
    private static final Integer COST_PER_KM = 10;

    private final Map<UserAction, BiConsumer<Long, Update>> userActionCommandMap = new
HashMap<>();

    @Resource
    private TelegramBot telegramBot;
    @Resource
    private OrderService orderService;
    @Resource
    private UserService userService;
    @Resource
    private TicketsService ticketsService;
    @Resource
    private DriverCarsService driverCarsService;

    public UserActionController() {
        userActionCommandMap.put(UserAction.SET_START_POSITION, this::setStartPosition);
        userActionCommandMap.put(UserAction.SET_END_POSITION, this::setEndPosition);
        userActionCommandMap.put(UserAction.CHOOSE_USER_ACTION, this::chooseTicketNumber);
        userActionCommandMap.put(UserAction.CHOOSE_DRIVER_ACTION, this::chooseDriveNumber);
    }

    public void processUserActionCommand(UserAction userAction, Long chatId, Update update) {
        userActionCommandMap.get(userAction).accept(chatId, update);
    }
```

```
    // POSITION

    private void setStartPosition(Long chatId, Update update){
        Location location = update.getMessage().getLocation();
        OrderEntity orderEntity = new OrderEntity();
        orderEntity.setClientChatId(chatId);
        orderEntity.setStartLatitudePosition(location.getLatitude().toString());
        orderEntity.setStartLongitudePosition(location.getLongitude().toString());
        orderEntity.setOrderStatus(OrderStatus.CREATION.name());
        orderService.saveOrder(orderEntity);

        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText("Ваша початкова позиція зафіксована, тепер відправте кінцеву позицію");

        telegramBot.executeMessage(sendMessage);
    }

    private void setEndPosition(Long chatId, Update update){
        Location location = update.getMessage().getLocation();
        Double endLatitude = location.getLatitude();
        Double endLongitude = location.getLongitude();

        OrderEntity orderEntity = orderService.findByClientChatIdAndOrderStatus(chatId,
OrderStatus.CREATION.name()).orElseThrow();
        orderEntity.setClientChatId(chatId);
        orderEntity.setEndLatitudePosition(endLatitude.toString());
        orderEntity.setEndLongitudePosition(endLongitude.toString());

        double startLatitude = Double.parseDouble(orderEntity.getStartLatitudePosition());
        double startLongitude = Double.parseDouble(orderEntity.getStartLongitudePosition());
        JSONObject routeInfo = getRouteInfo(
                startLongitude, startLatitude, endLongitude, endLatitude);

        Double distance = Math.round(routeInfo.getDouble(DISTANCE)/1000 * 10.0) / 10.0;
        String duration = String.valueOf(routeInfo.getDouble(DURATION));
        Long bill = Math.round(BASE_COST + distance * COST_PER_KM);

        orderEntity.setDistance(String.valueOf(distance));
        orderEntity.setDuration(duration);
        orderEntity.setBill(String.valueOf(bill));

        orderService.saveOrder(orderEntity);

        userService.updateCurrentAction(chatId, UserAction.DEFAULT.name());

        String textToClient = "Ваше замвовлення:" + "\n" +
                "Стартова позиція: " + getStreetName(startLatitude, startLongitude) + "\n" +
                "Кінцева позиція: " + getStreetName(endLatitude, endLongitude) + "\n" +
                "Дистанція: " + distance + "km" + "\n" +
                "Приблизний час поїздки: " + Math.round(((Double.parseDouble(duration) % 3600) /60)) * 100.0
/ 100.0 + " хв" + "\n" +
                "Ціна поїздки: " + bill + "грн";

        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText(textToClient);

        telegramBot.executeMessage(sendMessage);
    }

    private JSONObject getRouteInfo(double startLon, double startLat, double endLon, double endLat) {
        String apiUrl = "http://router.project-osrm.org/route/v1/driving/"
```

```
            + startLon + "," + startLat + ";" + endLon + "," + endLat
            + "?steps=true&geometries=geojson";

        try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
            HttpGet request = new HttpGet(apiUrl);
            try (CloseableHttpResponse response = httpClient.execute(request)) {
                String responseBody = EntityUtils.toString(response.getEntity());
                JSONObject json = new JSONObject(responseBody);
                JSONArray routesArray = json.getJSONArray("routes");
                return routesArray.getJSONObject(0);
            }
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    public String getStreetName(double latitude, double longitude) {
        String apiUrl = "https://nominatim.openstreetmap.org/reverse?lat=" + latitude + "&lon=" + longitude +
"&format=json";
        try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
            HttpGet request = new HttpGet(apiUrl);
            try (CloseableHttpResponse response = httpClient.execute(request)) {
                String responseBody = EntityUtils.toString(response.getEntity());
                JSONObject json = new JSONObject(responseBody);
                String houseNumber = getHouseNumber(json);
                return json.getJSONObject("address").get("road").toString() + " " + houseNumber;
            }
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }


    private static String getHouseNumber (JSONObject jsonObject) {
        try {
            return jsonObject.getJSONObject("address").get("house_number").toString();
        } catch (Exception e){
            return "";
        }
    }

    // MANAGER
    private void chooseTicketNumber (Long chatId, Update update) {
        String ticketId = update.getMessage().getText();
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        Optional<TicketsEntity> ticketsEntityOpt = ticketsService.findById(Long.parseLong(ticketId));
        if (ticketsEntityOpt.isEmpty()){
            sendMessage.setText("Тікета з таким номером не існує");
            telegramBot.executeMessage(sendMessage);
        }
        TicketsEntity ticketsEntity = ticketsEntityOpt.get();
        ticketsEntity.setManagerChatId(chatId);
        ticketsEntity.setTicketStatus(TicketStatus.Processing.name());
        ticketsService.save(ticketsEntity);
        UserEntity user = userService.getUserByChatId(chatId).get();
        user.setUserCurrentState(UserState.CHAT_MANAGER_WITH_PASSENGER.name());
        user.setUserCurrentAction(UserAction.DEFAULT.name());
        userService.saveUser(user);

        sendMessage.setText("Ви успішно взяли тікет, ваші наступні повідомлення буде бачити клієнт.");
```

```
        ReplyKeyboardMarkup inlineKeyboardMarkup = ReplyKeyboardMarkup.builder()
                .keyboard(List.of(
                        new KeyboardRow(List.of(closeTicket()))
                ))
                .build();

        sendMessage.setReplyMarkup(inlineKeyboardMarkup);
        telegramBot.executeMessage(sendMessage);
    }

    private KeyboardButton closeTicket () {
        return KeyboardButton.builder()
                .text("Закрити тікет")
                .build();
    }

    // Driver
    private void chooseDriveNumber (Long chatId, Update update) {
        String orderId = update.getMessage().getText();
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        Optional<OrderEntity> orderOpt = orderService.findById(Long.parseLong(orderId));
        if (orderOpt.isEmpty()){
            sendMessage.setText("Замовлення з таким номером не існує");
            telegramBot.executeMessage(sendMessage);
        }
        OrderEntity order = orderOpt.get();
        order.setDriverChatId(chatId);
        order.setOrderStatus(OrderStatus.PROCESSING.name());
        orderService.saveOrder(order);

        driverMassageAfterAccept(order);
        passengerMassageAfterDriverAccept(order);

        userService.updateCurrentAction(chatId, UserAction.DEFAULT.name());
    }

    private void driverMassageAfterAccept (OrderEntity order){
        Long chatId = order.getDriverChatId();
        SendMessage acceptOrderMessage = new SendMessage();
        acceptOrderMessage.setChatId(chatId);
        acceptOrderMessage.setText("Ви успішно прийняли замовлення");
        telegramBot.executeMessage(acceptOrderMessage);

        SendMessage firstClientPositionMassage = new SendMessage();
        firstClientPositionMassage.setChatId(chatId);
        firstClientPositionMassage.setText("Початкова позиція клієнта:");
        telegramBot.executeMessage(firstClientPositionMassage);
        sendLocationForDriver(chatId, Double.parseDouble(order.getStartLatitudePosition()),
Double.parseDouble(order.getStartLongitudePosition()));

        SendMessage secondClientPositionMassage = new SendMessage();
        secondClientPositionMassage.setChatId(chatId);
        secondClientPositionMassage.setText("Кінцева позиція клієнта:");
        telegramBot.executeMessage(secondClientPositionMassage);
        sendLocationForDriver(chatId, Double.parseDouble(order.getEndLatitudePosition()),
Double.parseDouble(order.getEndLongitudePosition()));

        SendMessage chatWithClient = new SendMessage();
        chatWithClient.setChatId(chatId);
        chatWithClient.setText("Усі ваші наступні повідомлення будуть надходити клієнту, поки не
завершиться поїздка");
```

```
        ReplyKeyboardMarkup inlineKeyboardMarkup = ReplyKeyboardMarkup.builder()
                .keyboard(List.of(
                        new KeyboardRow(List.of(closeOrder()))
                ))
                .build();
        chatWithClient.setReplyMarkup(inlineKeyboardMarkup);
        telegramBot.executeMessage(chatWithClient);

        userService.updateCurrentState(chatId, UserState.CHAT_DRIVER_WITH_PASSENGER.name());
    }

    private void sendLocationForDriver (Long chatId, Double latitude, Double longitude){
        SendLocation sendLocation = new SendLocation();
        sendLocation.setChatId(chatId);
        sendLocation.setLatitude(latitude);
        sendLocation.setLongitude(longitude);
        telegramBot.executeLocation(sendLocation);
    }
    private void passengerMassageAfterDriverAccept (OrderEntity order){
        Long chatId = order.getClientChatId();
        DriverCarsEntity driverCarsEntity =
driverCarsService.findByDriver(order.getDriverChatId()).orElseThrow();

        String text = "Водій " + driverCarsEntity.getDriverName() + " прийняв ваше замовлення" + "\n"
                + "Очікуйте " + driverCarsEntity.getModel() + "\n"
                + "Номер: " + driverCarsEntity.getNumber() + "\n"
                + "Колір: " + driverCarsEntity.getColor() + ".";

        SendMessage acceptOrderMessage = new SendMessage();
        acceptOrderMessage.setChatId(chatId);
        acceptOrderMessage.setText(text);

        InlineKeyboardButton sendMassageToDriver = new InlineKeyboardButton();
        sendMassageToDriver.setCallbackData("/sendMassageToDriver");
        sendMassageToDriver.setText("Написати водію");

        List<InlineKeyboardButton> sendMassageToDriverB = List.of(sendMassageToDriver);


        InlineKeyboardMarkup inlineKeyboardMarkup = new InlineKeyboardMarkup();
        inlineKeyboardMarkup.setKeyboard(List.of(sendMassageToDriverB));

        acceptOrderMessage.setReplyMarkup(inlineKeyboardMarkup);
        telegramBot.executeMessage(acceptOrderMessage);
    }

    private KeyboardButton closeOrder () {
        return KeyboardButton.builder()
                .text("Закінчити замовлення")
                .build();
    }
}
package com.example.kursach.command.controller;

import com.example.kursach.bot.TelegramBot;
import com.example.kursach.entities.OrderEntity;
import com.example.kursach.entities.TicketsEntity;
import com.example.kursach.entities.UserEntity;
import com.example.kursach.enums.OrderStatus;
import com.example.kursach.enums.TicketStatus;
import com.example.kursach.enums.UserState;
import com.example.kursach.services.entity.OrderService;
import com.example.kursach.services.entity.TicketsService;
```

```
import com.example.kursach.services.entity.UserService;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.InlineKeyboardButton;

import javax.annotation.Resource;
import javax.inject.Singleton;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.function.BiConsumer;

@Component
@Singleton
public class UserStateController {

    private final Map<UserState, BiConsumer<Long, Update>> userStateCommandMap = new
HashMap<>();

    @Resource
    private TicketsService ticketsService;
    @Resource
    private UserService userService;
    @Resource
    private OrderService orderService;
    @Resource
    private TelegramBot telegramBot;

    public UserStateController() {
        userStateCommandMap.put(UserState.FIRST_MASSAGE_TO_MANAGER,
this::firstMassageToManager);
        userStateCommandMap.put(UserState.CHAT_WITH_MANAGER, this::chatWithManager);
        userStateCommandMap.put(UserState.CHAT_MANAGER_WITH_PASSENGER,
this::chatManagerWithClient);
        userStateCommandMap.put(UserState.CHAT_DRIVER_WITH_PASSENGER,
this::chatDriverWithClient);
        userStateCommandMap.put(UserState.CHAT_WITH_DRIVER, this::chatWithDriver);
    }

    public void processUserStateCommand(UserState userState, Long chatId, Update update) {
        userStateCommandMap.get(userState).accept(chatId, update);
    }

    private void firstMassageToManager(Long chatId, Update update){
        TicketsEntity ticketsEntity = new TicketsEntity();
        ticketsEntity.setClientChatId(chatId);
        ticketsEntity.setTicketStatus(TicketStatus.Active.name());
        ticketsEntity.setTicketQuestion(update.getMessage().getText());
        ticketsService.save(ticketsEntity);

        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText("Ваш запит надіслано, ми дамо вам знати коли менеджер відповість на ваше
питання.");

        telegramBot.executeMessage(sendMessage);

        UserEntity user = userService.getUserByChatId(chatId).orElseThrow();

        user.setUserCurrentState(UserState.CHAT_WITH_MANAGER.name());
        userService.saveUser(user);
```

```
    }

    private void chatWithManager(Long chatId, Update update){
        TicketsEntity ticketsEntity = ticketsService.findByClientChatIdAndTicketStatus(chatId,
TicketStatus.Processing.name()).orElseThrow();
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(ticketsEntity.getManagerChatId());
        sendMessage.setText(update.getMessage().getText());

        InlineKeyboardButton answerToClient = new InlineKeyboardButton();
        answerToClient.setCallbackData("/answerToClient");
        answerToClient.setText("Відповісти клієнту");
        List<InlineKeyboardButton> answerToManagers = List.of(answerToClient);
        InlineKeyboardMarkup inlineKeyboardMarkup = new InlineKeyboardMarkup();
        inlineKeyboardMarkup.setKeyboard(List.of(answerToManagers));

        sendMessage.setReplyMarkup(inlineKeyboardMarkup);

        telegramBot.executeMessage(sendMessage);
    }

    private void chatManagerWithClient(Long chatId, Update update){
        TicketsEntity ticketsEntity = ticketsService.findByManagerChatIdAndTicketStatus(chatId,
TicketStatus.Processing.name()).orElseThrow();
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(ticketsEntity.getClientChatId());
        sendMessage.setText(update.getMessage().getText());

        InlineKeyboardButton answerToManager = new InlineKeyboardButton();
        answerToManager.setCallbackData("/answerToManager");
        answerToManager.setText("Відповісти менеджеру");
        List<InlineKeyboardButton> answerToManagers = List.of(answerToManager);
        InlineKeyboardMarkup inlineKeyboardMarkup = new InlineKeyboardMarkup();
        inlineKeyboardMarkup.setKeyboard(List.of(answerToManagers));

        sendMessage.setReplyMarkup(inlineKeyboardMarkup);

        telegramBot.executeMessage(sendMessage);
    }

    private void chatDriverWithClient(Long chatId, Update update){
        OrderEntity orderEntity = orderService.findByDriverChatIdAndOrderStatus(chatId,
OrderStatus.PROCESSING.name()).orElseThrow();
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(orderEntity.getClientChatId());
        sendMessage.setText(update.getMessage().getText());
        telegramBot.executeMessage(sendMessage);
    }


    private void chatWithDriver(Long chatId, Update update){
        OrderEntity orderEntity = orderService.findByClientChatIdAndOrderStatus(chatId,
OrderStatus.PROCESSING.name()).orElseThrow();
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(orderEntity.getDriverChatId());
        sendMessage.setText(update.getMessage().getText());
        telegramBot.executeMessage(sendMessage);
    }
}
package com.example.kursach.configuration;

import lombok.Data;
import org.springframework.beans.factory.annotation.Value;
```

```
import org.springframework.context.annotation.Configuration;

@Configuration
@Data
public class TelegramBotConfiguration {

    @Value("${bot.name}")
    private String botName;

    @Value("${bot.token}")
    private String botToken;

}
package com.example.kursach.entities;

import lombok.Data;

import javax.persistence.*;

@Entity
@Table(name = "driver_cars")
@Data
public class DriverCarsEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "driver_chat_id")
    private Long driverChatId;

    @Column(name = "driver_name")
    private String driverName;

    @Column(name = "number")
    private String number;

    @Column(name = "model")
    private String model;

    @Column(name = "color")
    private String color;

}
package com.example.kursach.entities;

import lombok.Data;

import javax.persistence.*;

@Entity
@Table(name = "taxi_order")
@Data
public class OrderEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "client_chat_id")
    private Long clientChatId;
```

```
    @Column(name = "driver_chat_id")
    private Long driverChatId;

    @Column(name = "start_latitude_position")
    private String startLatitudePosition;

    @Column(name = "start_longitude_position")
    private String startLongitudePosition;

    @Column(name = "end_latitude_position")
    private String endLatitudePosition;

    @Column(name = "end_longitude_position")
    private String endLongitudePosition;

    @Column(name = "duration")
    private String duration;

    @Column(name = "distance")
    private String distance;

    @Column(name = "bill")
    private String bill;

    @Column(name = "status")
    private String orderStatus;
}
package com.example.kursach.entities;

import lombok.Data;

import javax.persistence.*;

@Entity
@Table(name = "tickets")
@Data
public class TicketsEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "client_chat_id")
    private Long clientChatId;

    @Column(name = "manager_chat_id")
    private Long managerChatId;

    @Column(name = "ticket_status")
    private String ticketStatus;

    @Column(name = "ticket_question")
    private String ticketQuestion;
}
package com.example.kursach.entities;

import lombok.Data;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
```

```
import javax.persistence.Table;

@Entity
@Table(name = "user")
@Data
public class UserEntity {

    @Id
    @Column(name = "user_chat_id")
    private Long userChatId;

    @Column(name = "user_role")
    private String userRole;

    @Column(name = "user_current_role")
    private String userCurrentRole;

    @Column(name = "user_current_state")
    private String userCurrentState;

    @Column(name = "user_current_action")
    private String userCurrentAction;
}
package com.example.kursach.enums;

public enum OrderStatus {
    CREATION,
    ACTIVE,
    PROCESSING,
    FINISHED,
    CANCELLED
}
package com.example.kursach.enums;

public enum TicketStatus {
    Active,
    Processing,
    Closed
}
package com.example.kursach.enums;

public enum UserAction {
    DEFAULT,
    SET_START_POSITION,
    SET_END_POSITION,
    CHOOSE_USER_ACTION,
    CHOOSE_DRIVER_ACTION
}
package com.example.kursach.enums;

public enum UserRole {
    Passenger,
    Driver,
    Manager,
    Administrator
}
package com.example.kursach.enums;

public enum UserState {
    DEFAULT,
    CHAT_WITH_DRIVER,
    CHAT_DRIVER_WITH_PASSENGER,
    CHAT_MANAGER_WITH_PASSENGER,
```

```
    CHAT_WITH_MANAGER,
    FIRST_MASSAGE_TO_MANAGER,
    CHAT_WITCH_CLIENT
}
package com.example.kursach.repositrories;

import com.example.kursach.entities.DriverCarsEntity;
import com.example.kursach.entities.OrderEntity;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface DriverCarsRepository extends JpaRepository<DriverCarsEntity, Long> {
    Optional<DriverCarsEntity> findByDriverChatId (Long chatId);
}
package com.example.kursach.repositrories;

import com.example.kursach.entities.OrderEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

public interface OrderRepository extends JpaRepository<OrderEntity, Long> {
    Optional<OrderEntity> findByClientChatIdAndOrderStatus(Long chatId, String orderStatus);
    Optional<OrderEntity> findByDriverChatIdAndOrderStatus(Long chatId, String orderStatus);
    Optional<OrderEntity> findById(Long id);

    List<OrderEntity> findByClientChatIdAndOrderStatusNot(Long chatId, String orderStatus);
    List<OrderEntity> findByDriverChatIdAndOrderStatusNot(Long chatId, String orderStatus);

    List<OrderEntity> findAllByOrderStatus(String orderStatus);

    @Transactional
    @Modifying
    @Query("UPDATE OrderEntity o SET o.orderStatus = :newStatus WHERE o.clientChatId = :chatId AND
o.orderStatus = :oldStatus")
    void updateOrderStatusByChatIdAndStatus(Long chatId, String oldStatus, String newStatus);

}
package com.example.kursach.repositrories;

import com.example.kursach.entities.TicketsEntity;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;
import java.util.Optional;

public interface TicketsRepository extends JpaRepository<TicketsEntity, Long> {
    List<TicketsEntity> findAllByTicketStatus(String ticketStatus);
    Optional<TicketsEntity> findById(Long id);

    Optional<TicketsEntity> findByManagerChatIdAndTicketStatus(Long managerId, String ticketStatus);
    Optional<TicketsEntity> findByClientChatIdAndTicketStatus(Long managerId, String ticketStatus);
}
package com.example.kursach.repositrories;

import com.example.kursach.entities.UserEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
```

```
import org.springframework.data.jpa.repository.Query;
import org.springframework.transaction.annotation.Transactional;

import java.util.Optional;

public interface UserRepository extends JpaRepository<UserEntity, Long> {
   Optional<UserEntity> findByUserChatId (Long chatId);

   @Transactional
   @Modifying
   @Query("UPDATE UserEntity u SET u.userCurrentRole = :newCurrentRole WHERE u.userChatId =
:chatId")
   void updateUserCurrentRoleByUserChatId(Long chatId, String newCurrentRole);

   @Transactional
   @Modifying
   @Query("UPDATE UserEntity u SET u.userCurrentAction = :newCurrentAction WHERE u.userChatId =
:chatId")
   void updateUserCurrenActionByUserChatId(Long chatId, String newCurrentAction);

   @Transactional
   @Modifying
   @Query("UPDATE UserEntity u SET u.userCurrentState = :newCurrentState WHERE u.userChatId =
:chatId")
   void updateUserCurrentStateByUserChatId(Long chatId, String newCurrentState);
}
package com.example.kursach.services.command;

import com.example.kursach.command.controller.CommandControllerForCallBackQuery;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.objects.Update;

@Service
@AllArgsConstructor
public class CallBackQueryCommandService {

   private CommandControllerForCallBackQuery commandControllerForCallBackQuery;

   public void processCallBackQueryCommand(Update update) {
      commandControllerForCallBackQuery.processCommand(update.getCallbackQuery().getData(),
update.getCallbackQuery().getMessage().getChatId(), update);
   }
}
package com.example.kursach.services.command;

import com.example.kursach.command.controller.CommandControllerForText;
import com.example.kursach.command.controller.UserActionController;
import com.example.kursach.command.controller.UserStateController;
import com.example.kursach.enums.UserAction;
import com.example.kursach.enums.UserState;
import com.example.kursach.services.entity.UserService;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.objects.Update;

import java.util.List;

@Service
@AllArgsConstructor
public class TextCommandService {

   private CommandControllerForText commandControllerForText;
```

```
    private UserService userService;
    private UserActionController userActionController;
    private UserStateController userStateController;

    private static final List<String> firstPriorityCommand = List.of("Відмінити замовлення",
            "Назад ⬅\uFE0F",
            "Закрити тікет",
            "Закінчити замовлення",
            "Закінчити розмову з водієм");

    public void processTextCommand(Update update) {
        Long chatId = update.getMessage().getChatId();
        UserAction userAction = userService.getUserAction(chatId);
        UserState userState = userService.getUserState(chatId);

        if (update.getMessage().getText() !=null &&
firstPriorityCommand.contains(update.getMessage().getText())){
            commandControllerForText.processCommand(update.getMessage().getText(), chatId, update);
        }
        else if (!UserAction.DEFAULT.equals(userAction)){
            userActionController.processUserActionCommand(userAction, chatId, update);
        } else if (!UserState.DEFAULT.equals(userState)){
            userStateController.processUserStateCommand(userState, chatId, update);
        } else commandControllerForText.processCommand(update.getMessage().getText(), chatId, update);
    }
}
package com.example.kursach.services.entity;

import com.example.kursach.entities.DriverCarsEntity;
import com.example.kursach.repositrories.DriverCarsRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service
@AllArgsConstructor
public class DriverCarsService {

    private final DriverCarsRepository driverCarsRepository;

    public Optional<DriverCarsEntity> findByDriver (Long chatId){
        return driverCarsRepository.findByDriverChatId(chatId);
    }
}
package com.example.kursach.services.entity;

import com.example.kursach.entities.OrderEntity;
import com.example.kursach.enums.OrderStatus;
import com.example.kursach.repositrories.OrderRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
@AllArgsConstructor
public class OrderService {

    private OrderRepository orderRepository;

    public void saveOrder(OrderEntity orderEntity) {
```

```
        orderRepository.save(orderEntity);
    }

    public Optional<OrderEntity> findByClientChatIdAndOrderStatus(Long chatId, String orderStatus) {
        return orderRepository.findByClientChatIdAndOrderStatus(chatId, orderStatus);
    }

    public Optional<OrderEntity> findByDriverChatIdAndOrderStatus(Long chatId, String orderStatus) {
        return orderRepository.findByDriverChatIdAndOrderStatus(chatId, orderStatus);
    }

    public List<OrderEntity> findUserHistory(Long chatId, String orderStatus) {
        return orderRepository.findByClientChatIdAndOrderStatusNot(chatId, orderStatus);
    }

    public List<OrderEntity> findDriverHistory(Long chatId, String orderStatus) {
        return orderRepository.findByDriverChatIdAndOrderStatusNot(chatId, orderStatus);
    }

    public void updateOrderStatus(Long chatId, String orderStatus, String newOrderStatus) {
        orderRepository.updateOrderStatusByChatIdAndStatus(chatId, orderStatus, newOrderStatus);
    }

    public List<OrderEntity> findAllByOrderStatus(String orderStatus) {
        return orderRepository.findAllByOrderStatus(orderStatus);
    }

    public Optional<OrderEntity> findById(Long id) {
        return orderRepository.findById(id);
    }

}
package com.example.kursach.services.entity;

import com.example.kursach.entities.TicketsEntity;
import com.example.kursach.repositrories.TicketsRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
@AllArgsConstructor
public class TicketsService {

    private TicketsRepository ticketsRepository;

    public void save(TicketsEntity ticketsEntity){
        ticketsRepository.save(ticketsEntity);
    }

    public List<TicketsEntity> findAllByStatus (String ticketStatus){
        return ticketsRepository.findAllByTicketStatus(ticketStatus);
    }

    public Optional<TicketsEntity> findById (Long id){
        return ticketsRepository.findById(id);
    }

    public Optional<TicketsEntity> findByManagerChatIdAndTicketStatus (Long id, String ticketStatus){
        return ticketsRepository.findByManagerChatIdAndTicketStatus(id, ticketStatus);
    }
```

```
    public Optional<TicketsEntity> findByClientChatIdAndTicketStatus (Long id, String ticketStatus){
        return ticketsRepository.findByClientChatIdAndTicketStatus(id, ticketStatus);
    }
}
package com.example.kursach.services.entity;

import com.example.kursach.entities.UserEntity;
import com.example.kursach.enums.UserAction;
import com.example.kursach.enums.UserState;
import com.example.kursach.repositrories.UserRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service
@AllArgsConstructor
public class UserService {

    private final UserRepository userRepository;

    public void saveUser(UserEntity userEntity){
        userRepository.save(userEntity);
    }

    public Optional<UserEntity> getUserByChatId(Long chatId){
        return userRepository.findByUserChatId(chatId);
    }

    public UserAction getUserAction(Long chatId){
        Optional<UserEntity> userOpt = getUserByChatId(chatId);
        return userOpt.map(userEntity ->
UserAction.valueOf(userEntity.getUserCurrentAction())).orElse(UserAction.DEFAULT);
    }

    public UserState getUserState(Long chatId){
        Optional<UserEntity> userOpt = getUserByChatId(chatId);
        return userOpt.map(userEntity ->
UserState.valueOf(userEntity.getUserCurrentState())).orElse(UserState.DEFAULT);
    }

    public void updateCurrentRole(Long chatId, String currentState){
        userRepository.updateUserCurrentRoleByUserChatId(chatId, currentState);
    }

    public void updateCurrentState(Long chatId, String currentState){
        userRepository.updateUserCurrentStateByUserChatId(chatId, currentState);
    }

    public void updateCurrentAction(Long chatId, String currentAction){
        userRepository.updateUserCurrenActionByUserChatId(chatId, currentAction);
    }
}
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
```

```
    <version>2.5.2</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.example</groupId>
<artifactId>kursach</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>kursach</name>
<description>kursach</description>
<properties>
    <java.version>17</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-amqp</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.amqp</groupId>
        <artifactId>spring-rabbit-test</artifactId>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.telegram</groupId>
        <artifactId>telegrambots-spring-boot-starter</artifactId>
        <version>6.5.0</version>
    </dependency>

    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.28</version>
    </dependency>

    <dependency>
        <groupId>org.liquibase</groupId>
```

```
            <artifactId>liquibase-core</artifactId>
        </dependency>

        <dependency>
            <groupId>org.apache.httpcomponents</groupId>
            <artifactId>httpclient</artifactId>
            <version>4.5.13</version>
        </dependency>
        <dependency>
            <groupId>org.json</groupId>
            <artifactId>json</artifactId>
            <version>20210307</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <image>
                        <builder>paketobuildpacks/builder-jammy-base:latest</builder>
                    </image>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

ДОДАТОК Г

Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА TELEGRAM-БОТА ДЛЯ СИСТЕМИ УПРАВЛІННЯ СЛУЖБИ ТАКСІ

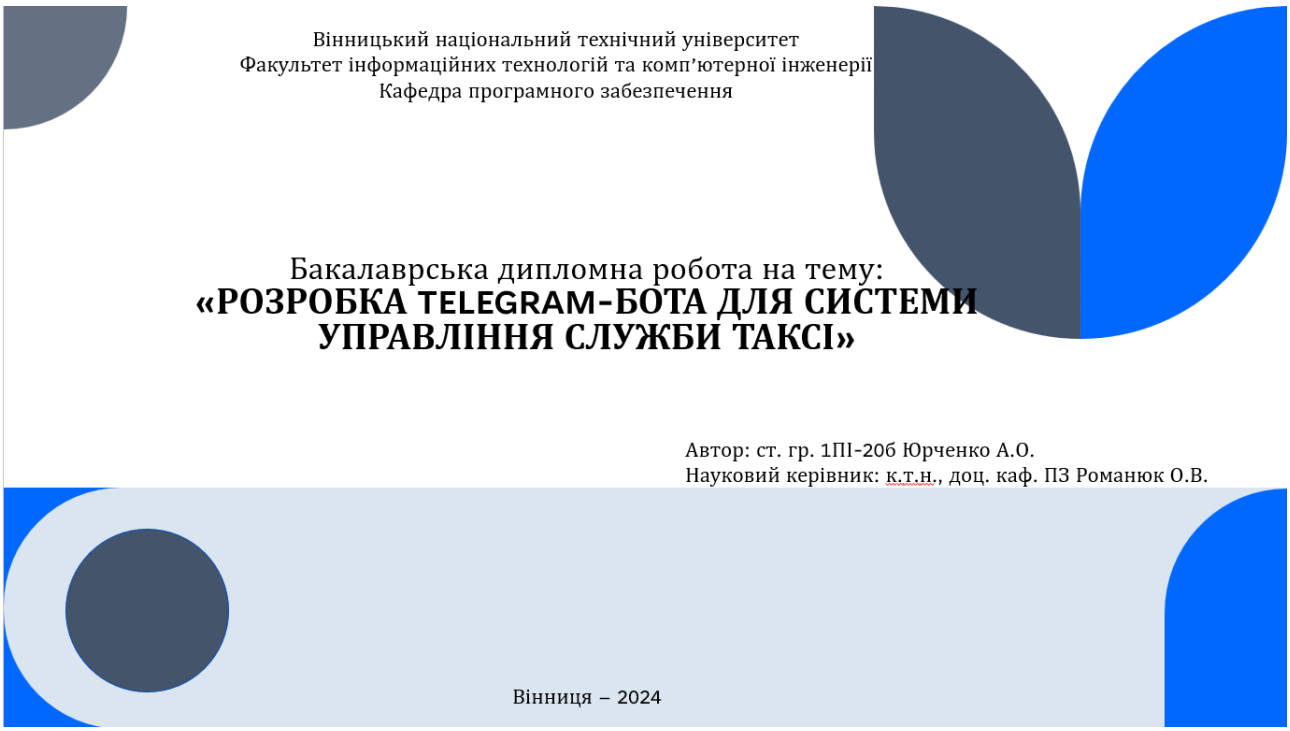


Рисунок Г.1 – Титульний слайд

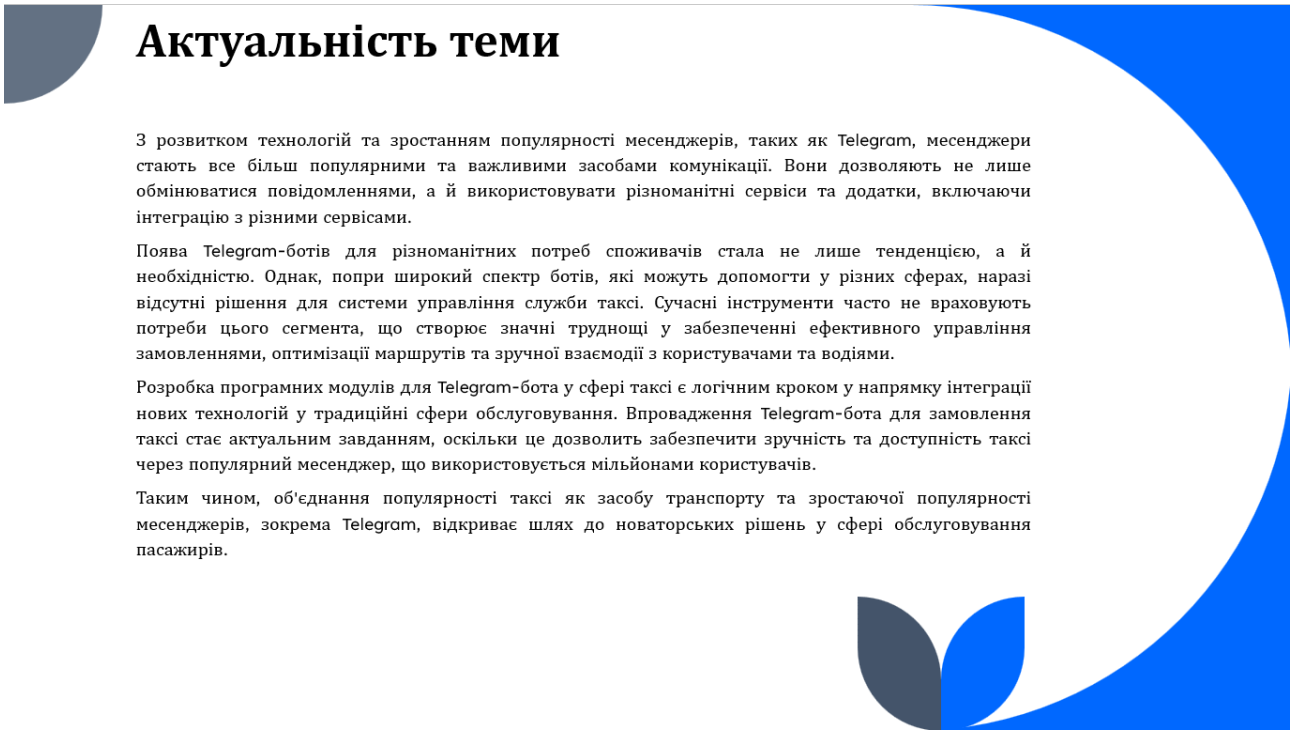


Рисунок Г.2 – Актуальність теми

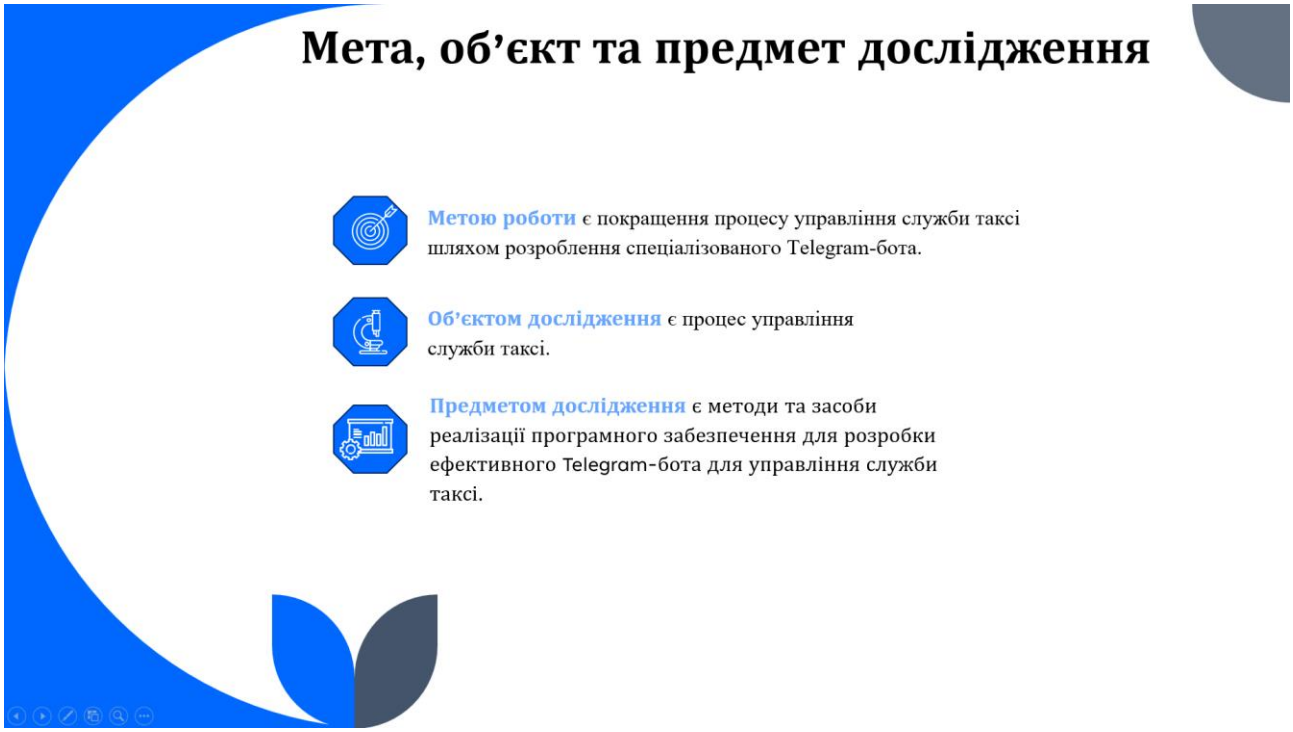


Рисунок Г.3 – Мета, об’єкт та предмет дослідження

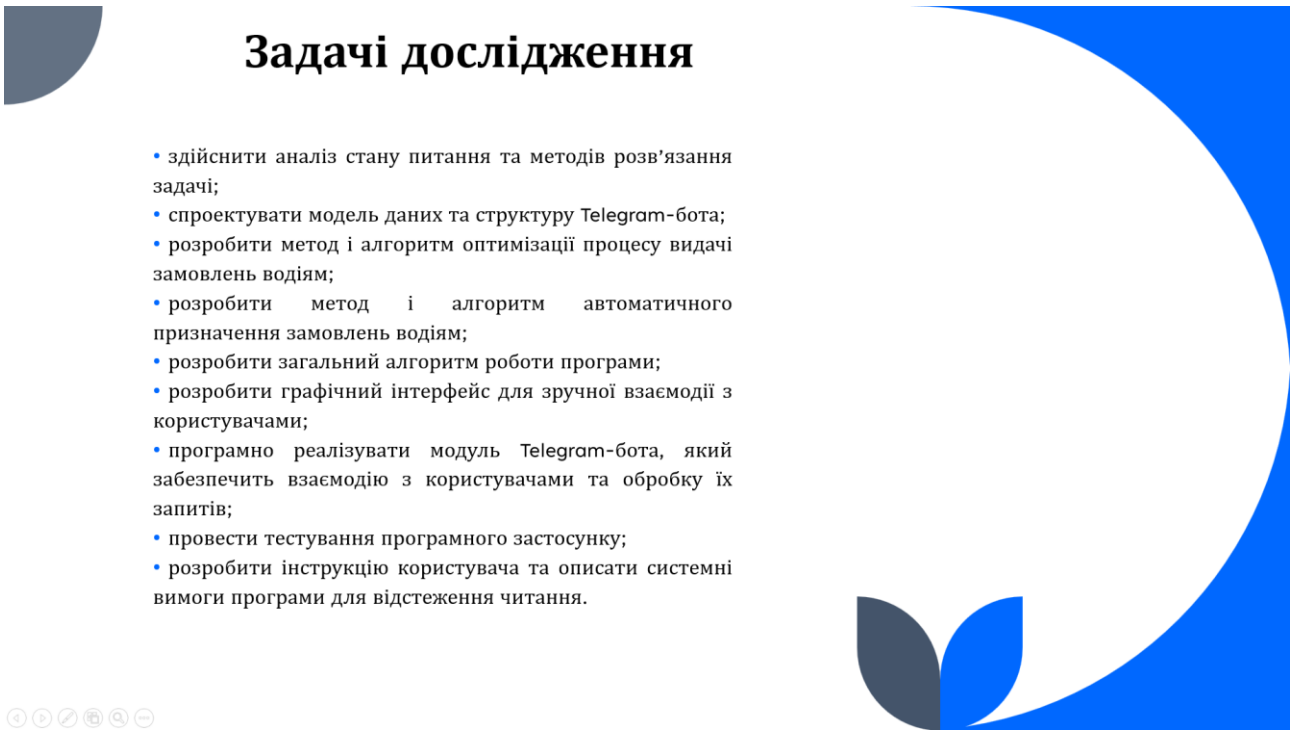


Рисунок Г.4 – Задачі дослідження

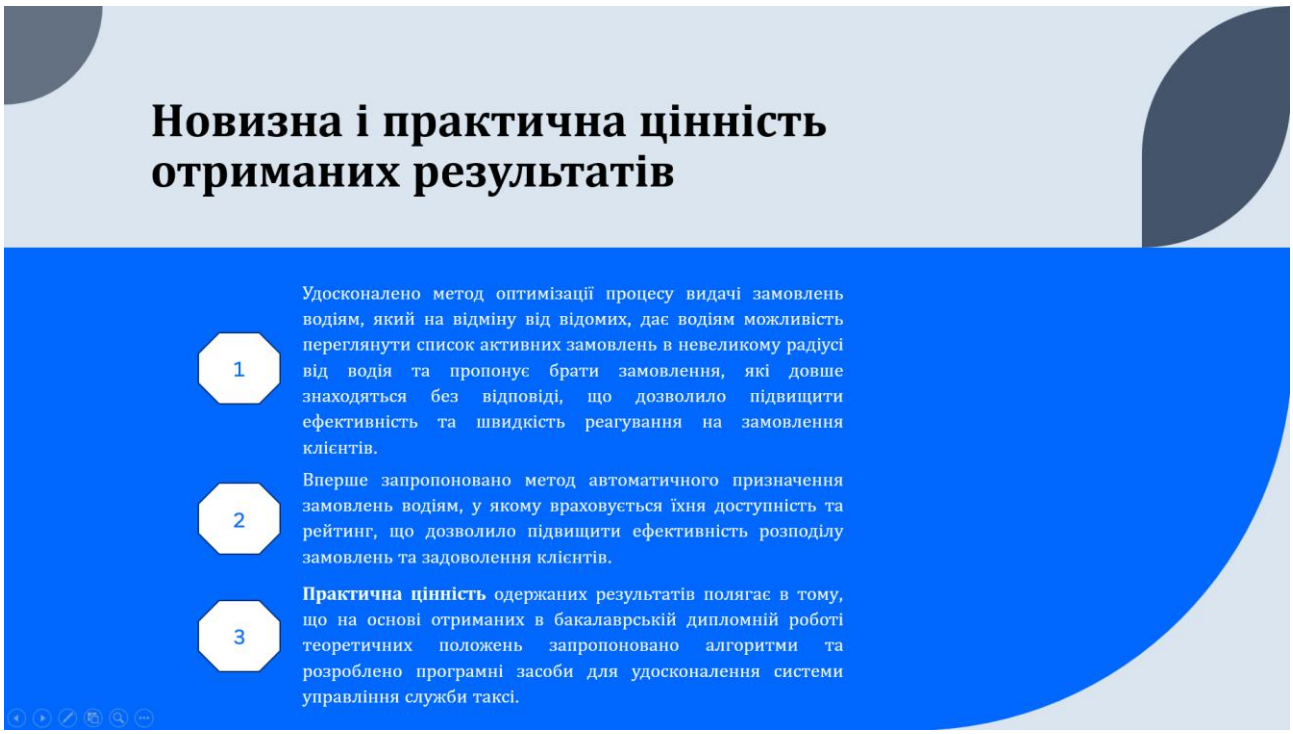


Рисунок Г.5 – Новизна і практична цінність отриманих результатів

Функція	Uber	Grab	Bolt	Власна реалізація
Простота використання	+	+	+	+
Інтуїтивно зрозумілий інтерфейс	+	–	+	+
Відсутність проблем з підтримкою клієнтів	–	+	–	+
Кросплатформність	+	–	–	+
Вбудовані функції спілкування з водіями	+	–	+	+
Можливість використання без необхідності завантаження мобільного додатку	–	–	–	+
Можливість користуватися сервісом без наявності кредитної картки	–	–	+	+
Підсумок	**4**	**2**	**4**	**7**

Рисунок Г.6 – Порівняльний аналіз аналогів

Рисунок Г.7 – Діаграма класів

Рисунок Г.8 – Загальний алгоритм роботи Telegram-бота

Рисунок Г.9 – Метод і блок-схема алгоритму автоматичного призначення замовлень водіям

Рисунок Г.10 – Метод і блок-схема алгоритму оптимізації процесу видачі замовлень

Меню авторизації застосунку

В меню авторизації користувачеві доступно:

- вибір ролі кристувача (Користувач бачить доступні опції в залежності від його ролі);
- початок співпраці з службою таксі

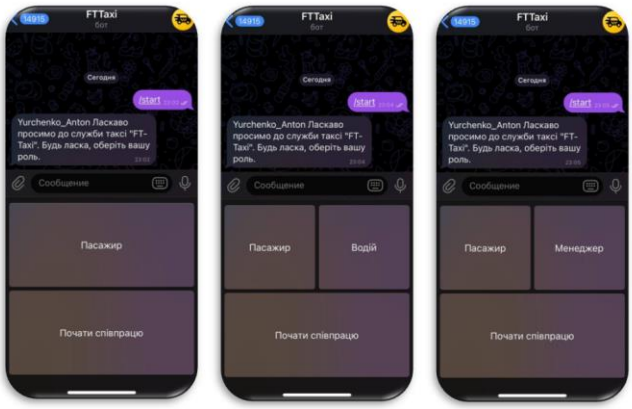


Рисунок Г.11 – Меню авторизації застосунку

Вибір стартової і кінцевої позиції маршруту

Вибір стартової і кінцевої позиції маршруту дозволяє:

- обрати стартову та кінцеву позицію на мапі;
- ввести назву вулиці в пошуку

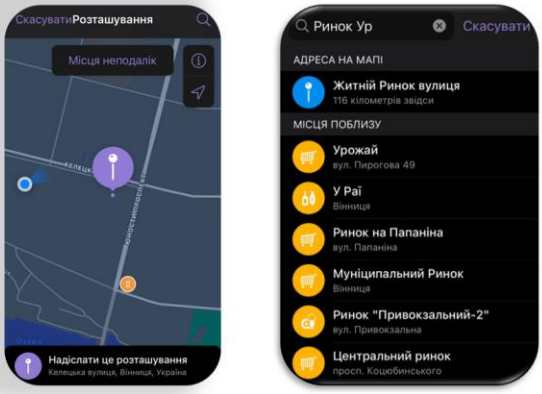


Рисунок Г.12 – Вибір позицій маршруту

Фінальний етап замовлення

Фінальний етап замовлення дозволяє:

- перегляд інформації про поїздку;
- підтвердження замовлення;
- відхилення замовлення.

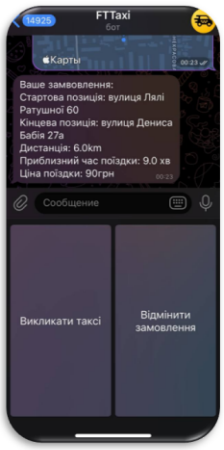
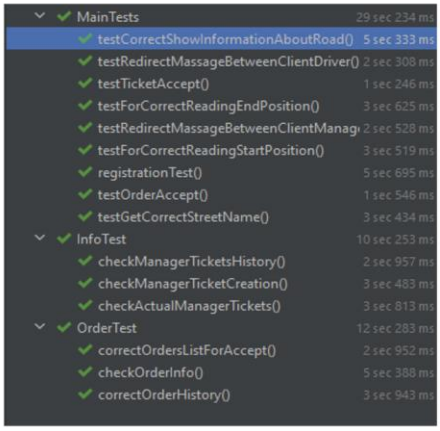


Рисунок Г.13 – Фінальний етап замовлення

Тестування програми



Перевірка коректного зчитування даних користувача та відображення інформації про поїздку

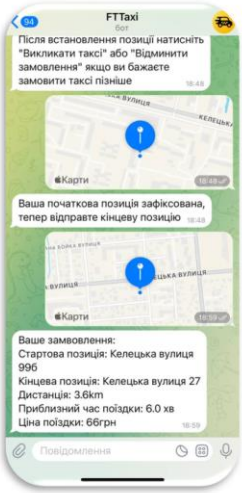


Рисунок Г.14 – Тестування програми

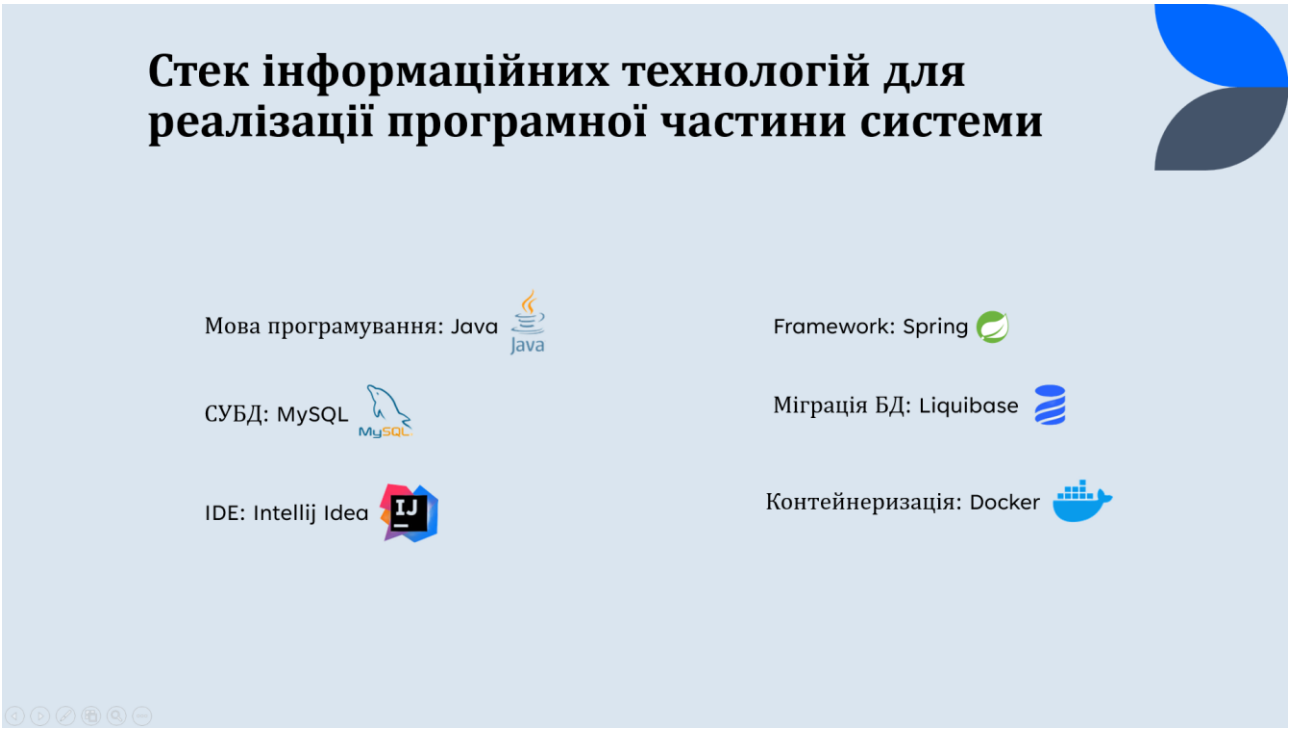


Рисунок Г.15 – Стек технологій для реалізації програмної частини системи

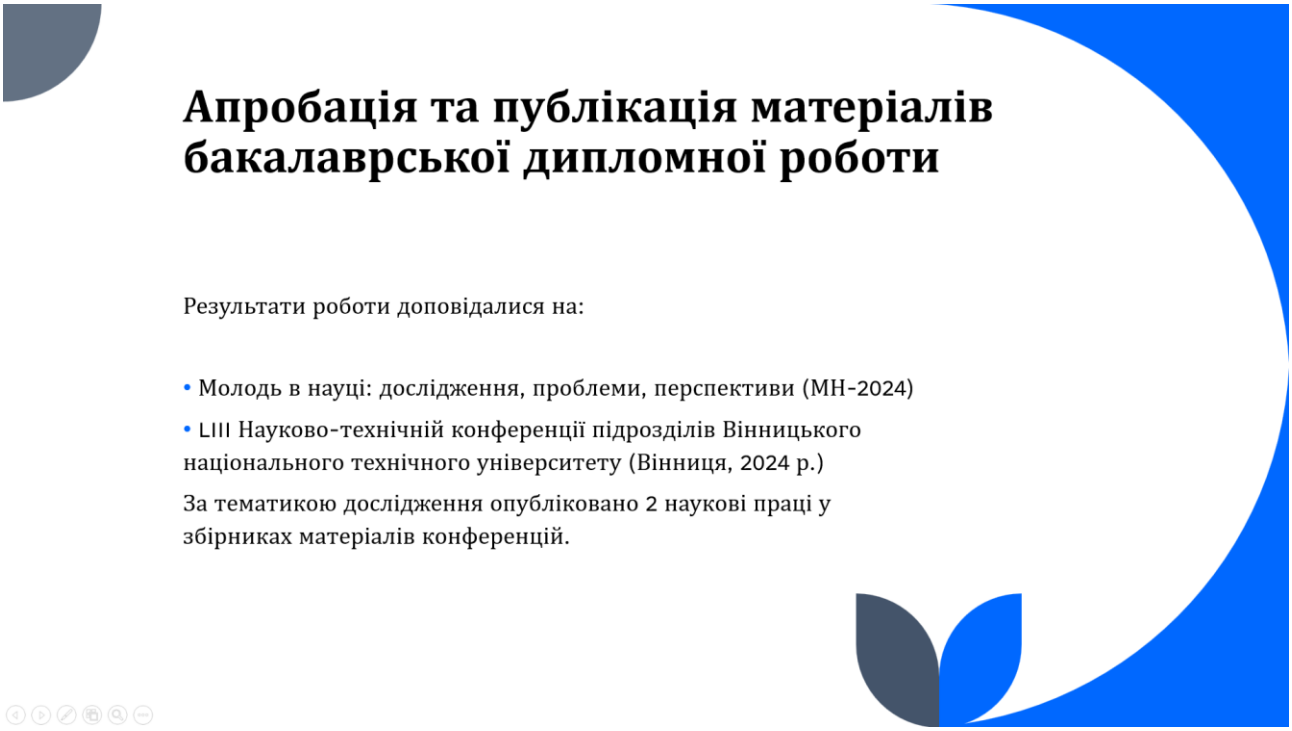


Рисунок Г.16 – Апробація та публікація матеріалів бакалаврської дипломної роботи

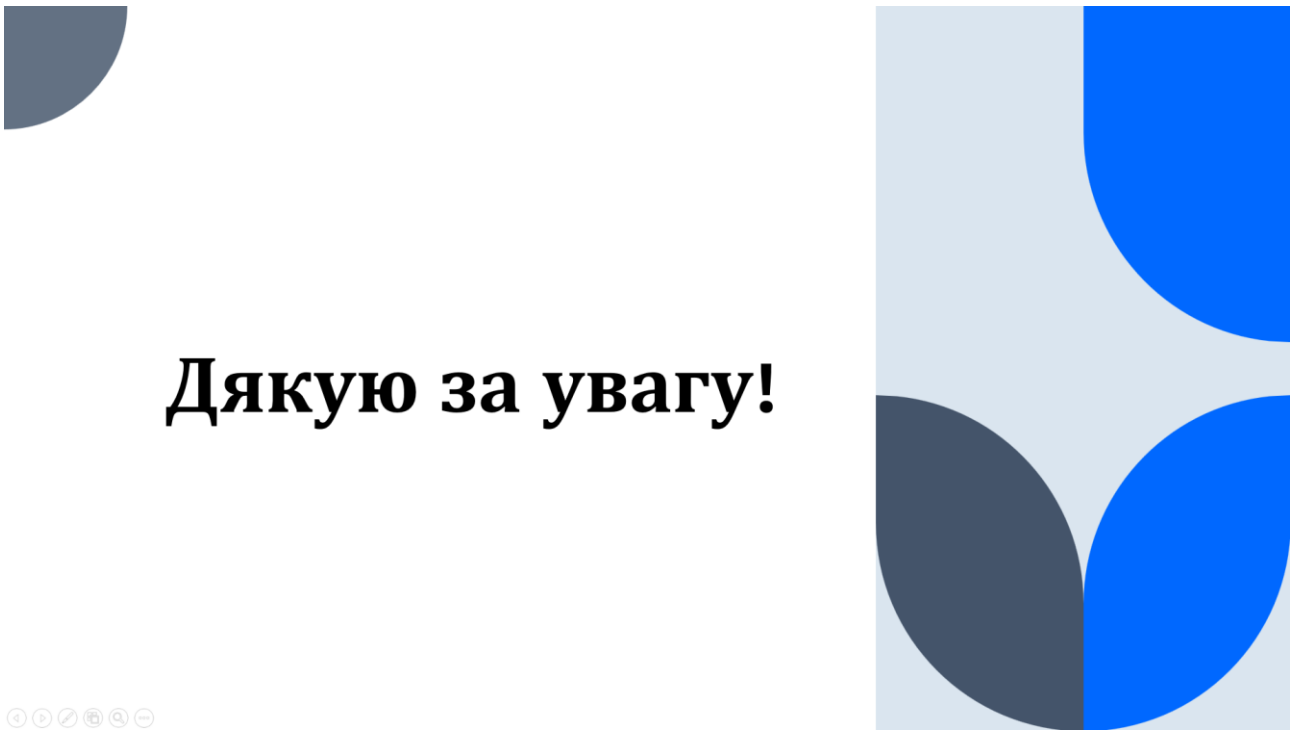


Рисунок Г.17 – Фінальний слайд