

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
“ ” _____ 2024 р.

Дипломний проєкт
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення
інформаційних систем»
спеціальності «121 Інженерія програмного забезпечення»

на тему: Вебзастосунок підвищення ефективності діяльності служби
доставки води

Виконав студент IV курсу, групи ІТ-04
(шифр групи)
Мартишенко Павло Сергійович
(прізвище, ім'я, по батькові) _____ (підпис)

Керівник доцент, к.т.н., доц., Лісовиченко О.І.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) _____ (підпис)

Рецензент професор, д.т.н., професор каф. ІСТ Корнага Я.І.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) _____ (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших
авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення
інформаційних систем

ЗАТВЕРДЖУЮ

Завідувач кафедри

(підпис) Едуард ЖАРІКОВ
(ім'я прізвище)

“ ____ ” _____ 2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Мартишенку Павлу Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Вебзастосунок підвищення ефективності діяльності служби
доставки води

керівник проєкту Лісовиченко Олег Іванович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 27 » травня 2024 р. № 2112.с

2. Термін подання студентом проєкту « 17 » червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: загальні положення, змістовний опис і аналіз предметної області, аналіз існуючих технологій та успішних ІТ-проєктів, аналіз вимог до програмного забезпечення, постановка задачі.

2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних.

3) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, опис процесів тестування, опис контрольного прикладу.

4) Впровадження та супровід програмного забезпечення: розгортання програмного забезпечення, підтримка програмного забезпечення.

5. Перелік графічного матеріалу
- 1) Схема структурна варіантів використань
- 2) Схема бази даних
- 3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «11» березня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	18.03.2024	
3	Постановка та формалізація задачі	20.03.2024	
4	Розробка інформаційного забезпечення	28.03.2024	
5	Алгоритмізація задачі	03.04.2024	
6	Обґрунтування вибору використаних технічних засобів	06.04.2024	
7	Розробка програмного забезпечення	04.05.2024	
8	Налагодження програми	10.05.2024	
9	Виконання графічних документів	15.05.2024	
10	Оформлення пояснювальної записки	29.05.2024	
11	Подання ДП на попередній захист	03.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____
(підпис)

Павло МАРТИШЕНКО _____
(ініціали, прізвище)

Керівник _____
(підпис)

Олег ЛІСОВИЧЕНКО _____
(ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проекту складається з п'ятих розділів, містить 57 таблиць, 45 рисунків та 13 джерел – загалом 95 сторінок.

Дипломний проект присвячений розробці вебзастосунку підвищення ефективності діяльності служби доставки води.

Метою розробки є підвищення ефективності роботи служби доставки води за рахунок автоматизації.

Об'єкт дослідження: технології реалізації вебзастосунку.

Предмет дослідження: протоколи обміну даними, API, інтерфейс у Telegram та база даних.

У розділі «Аналіз вимог до програмного забезпечення» описані загальні положення, аналіз предметної області, аналіз існуючих технологій та успішних ІТ-проектів, аналіз вимог до програмного забезпечення, постановка задачі.

Розділ «Моделювання та конструювання програмного забезпечення» присвячений аналізу та моделюванню програмного забезпечення, опису архітектури, конструюванню програмного забезпечення та аналізу безпеки даних.

У розділі «Аналіз якості та тестування програмного забезпечення» проведений аналіз якості ПЗ, описані процеси тестування та контрольний приклад.

Останній розділ «Впровадження та супровід програмного забезпечення» містить опис розгортання та підтримки програмного забезпечення.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУВАННЯ, ДОСТАВКА, ЗАМОВЛЕННЯ, БОТ, ТЕЛЕГРАМ, АНАЛІЗ, БАЗА ДАНИХ, ТЕСТИ.

ABSTRACT

The explanatory note of the diploma project consists of five sections, containing 57 tables, 45 figures, and 13 sources – a total of 95 pages.

The diploma project is dedicated to the development of a web application to enhance the efficiency of a water delivery service.

The goal of the development is to increase the efficiency of the water delivery service through automation.

The object of the study: technologies for implementing a web application.

The subject of the study: data exchange protocols, API, Telegram interface, and database.

The section "Analysis of Software Requirements" describes general provisions, domain analysis, analysis of existing technologies and successful IT projects, analysis of software requirements, and problem statement.

The section "Software Modeling and Design" is devoted to software analysis and modeling, description of the architecture, software design, and data security analysis.

In the section "Quality Analysis and Software Testing," software quality analysis is conducted, and the testing processes and a control example are described.

The last section, "Software Implementation and Maintenance," contains a description of the deployment and support of the software.

KEYWORDS: WEB APPLICATION, DELIVERY, ORDER, BOT, TELEGRAM, ANALYSIS, DATABASE, TESTS.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2024 р.

**ВЕБЗАСТОСУНОК ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ДІЯЛЬНОСТІ
СЛУЖБИ ДОСТАВКИ ВОДИ**

Технічне завдання

КП.ІТ-0411.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олег ЛІСОВИЧЕНКО

Нормоконтроль:

_____ Павло РОДІОНОВ

Виконавець:

_____ Павло МАРТИШЕНКО

Київ – 2024

ЗМІСТ

1	НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ	3
2	ПІДСТАВА ДЛЯ РОЗРОБКИ	4
3	ПРИЗНАЧЕННЯ РОЗРОБКИ.....	5
4	ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	6
4.1	Вимоги до функціональних характеристик	6
4.1.1	Користувацького інтерфейсу.....	6
4.1.2	Для користувача:	16
4.1.3	Для адміністратора системи:	17
4.1.4	Додаткові вимоги:	17
4.2	Вимоги до надійності.....	17
4.3	Умови експлуатації	17
4.3.1	Вид обслуговування.....	17
4.3.2	Обслуговуючий персонал	18
4.4	Вимоги до складу і параметрів технічних засобів	18
4.5	Вимоги до інформаційної та програмної сумісності	18
4.5.1	Вимоги до вхідних даних	19
4.5.2	Вимоги до вихідних даних	19
4.5.3	Вимоги до мови розробки.....	19
4.5.4	Вимоги до середовища розробки	19
4.5.5	Вимоги до представленню вихідних кодів	19
4.6	Вимоги до маркування та пакування	19
4.7	Вимоги до транспортування та зберігання	19
4.8	Спеціальні вимоги.....	19
5	ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ	20
5.1	Попередній склад програмної документації.....	20
5.2	Спеціальні вимоги до програмної документації	20
6	СТАДІЇ І ЕТАПИ РОЗРОБКИ.....	21
7	ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ	22

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Вебзастосунок підвищення ефективності діяльності служби доставки води.

Галузь застосування:

Наведене технічне завдання поширюється на розробку вебзастосунку підвищення ефективності діяльності служби доставки води “Water Delivery”, у якому буде впроваджено електронний каталог товарів з актуальними цінами, можливість створення електронної заявки в системі, а також адміністрування замовлень користувачів та ведення обліку доставок для адміністратора.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки вебзастосунку підвищення ефективності діяльності служби доставки води є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для зменшення кількості ручних процесів у роботі служби доставки води, що призводять до помилок, затримок та неефективності, і вказують на проблему відсутності автоматизованих процесів.

Метою розробки є підвищення ефективності роботи служби доставки води за рахунок автоматизації ручних процесів.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Користувацького інтерфейсу

- прототип головного меню зображений на рисунку 4.1;

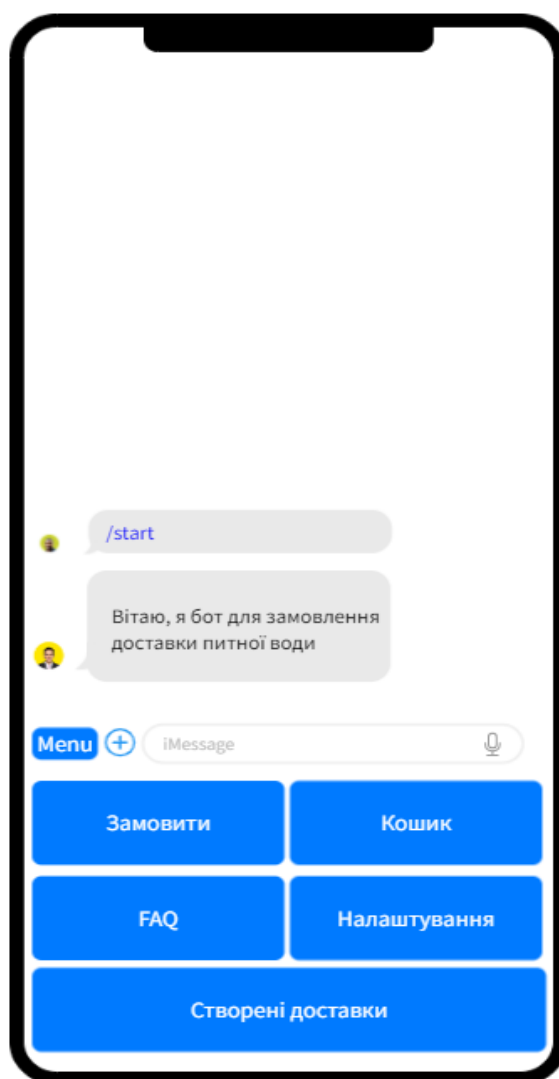


Рисунок 4.1 – Головне меню

- прототип реєстрації зображений на рисунку 4.2;

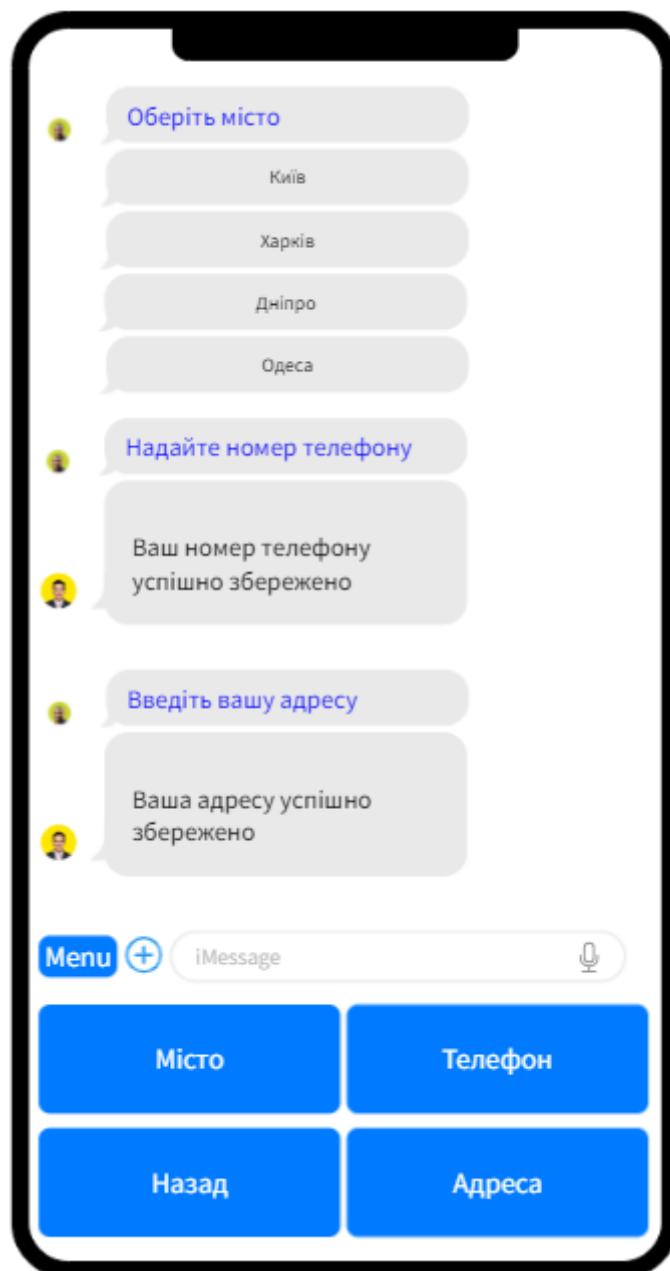


Рисунок 4.2 – Реєстрація

- прототип отримання відповідей на поширені питання зображений на рисунку 4.3;

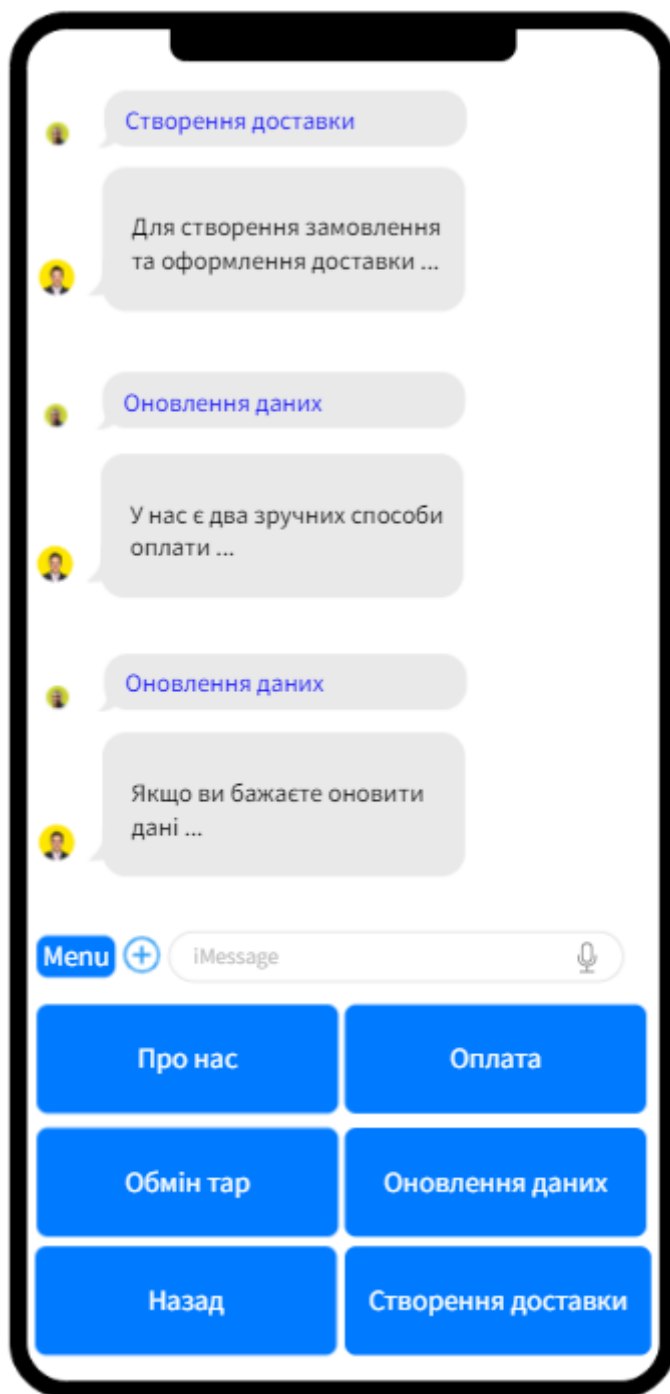


Рисунок 4.3 – Відповіді на питання

- прототип створення замовлення зображений на рисунку 4.4;

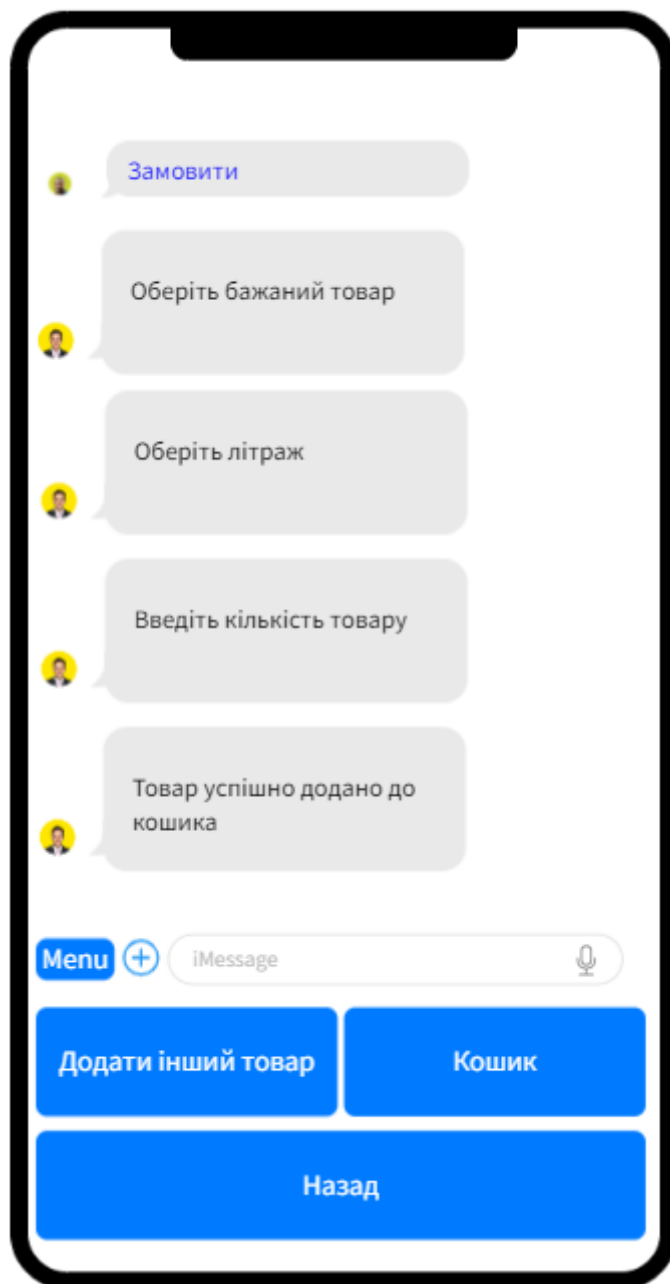


Рисунок 4.4 – Створення замовлення

- прототип перегляду кошику зображений на рисунку 4.5;

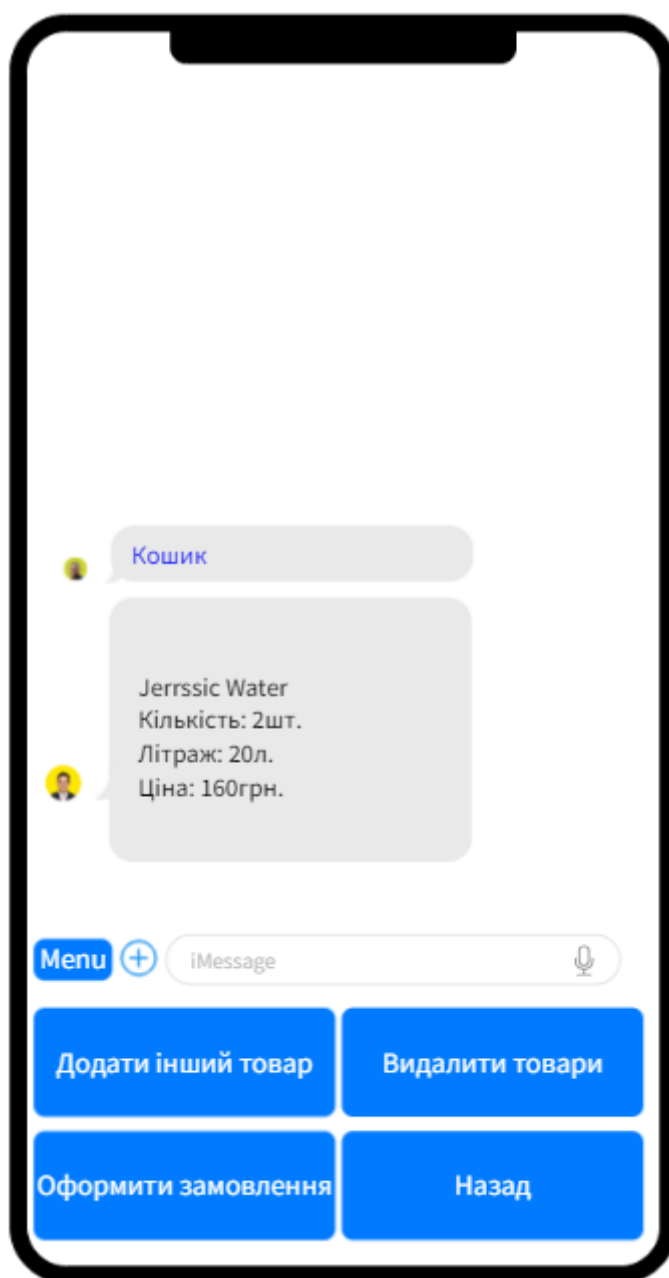


Рисунок 4.5 – Перегляд кошику

- прототип оформлення доставки зображений на рисунку 4.6;

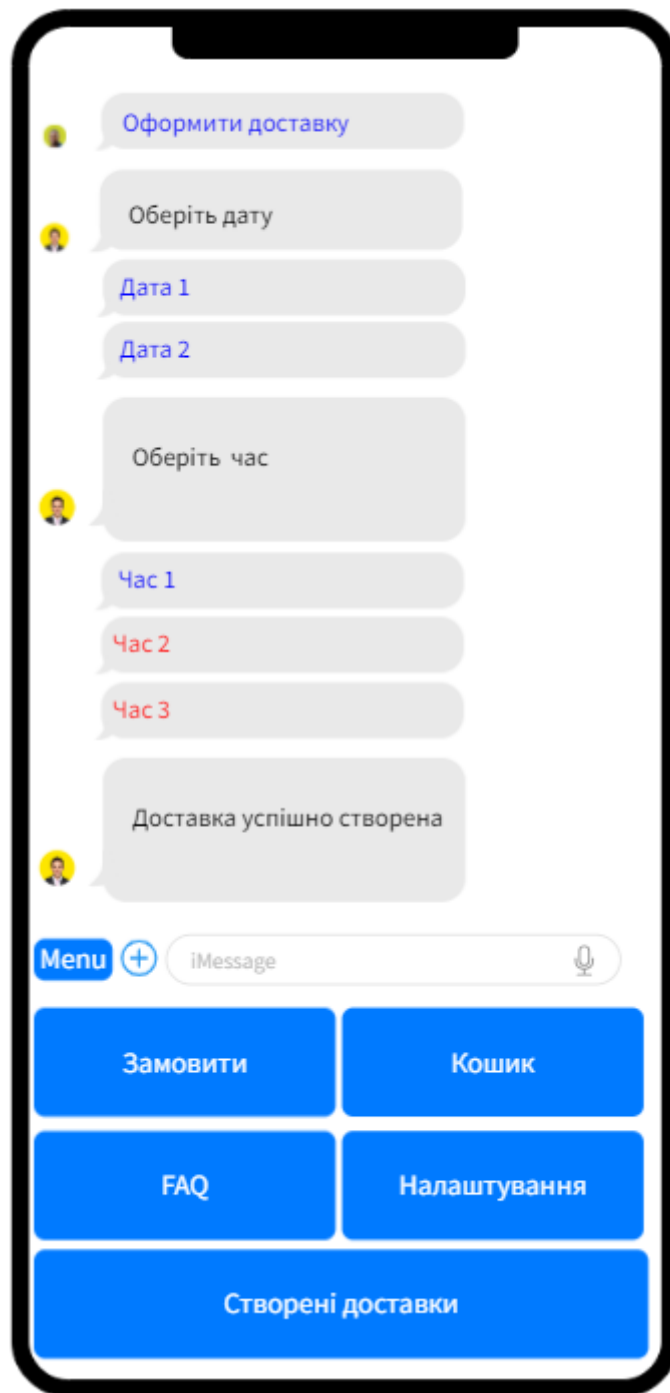


Рисунок 4.6 – Оформлення доставки

- прототип перегляду створеної доставки зображений на рисунку 4.7;

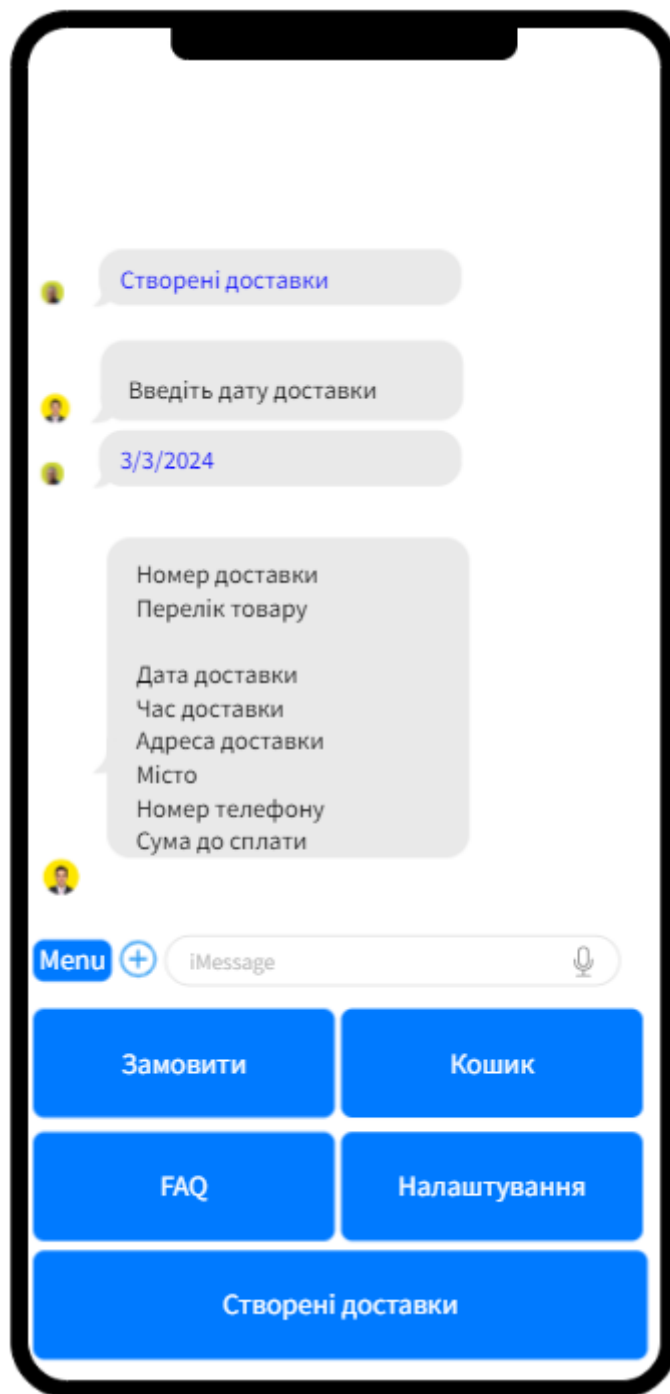


Рисунок 4.7 – Перегляд створеної доставки

- прототип перегляду активних доставок зображений на рисунку 4.8;

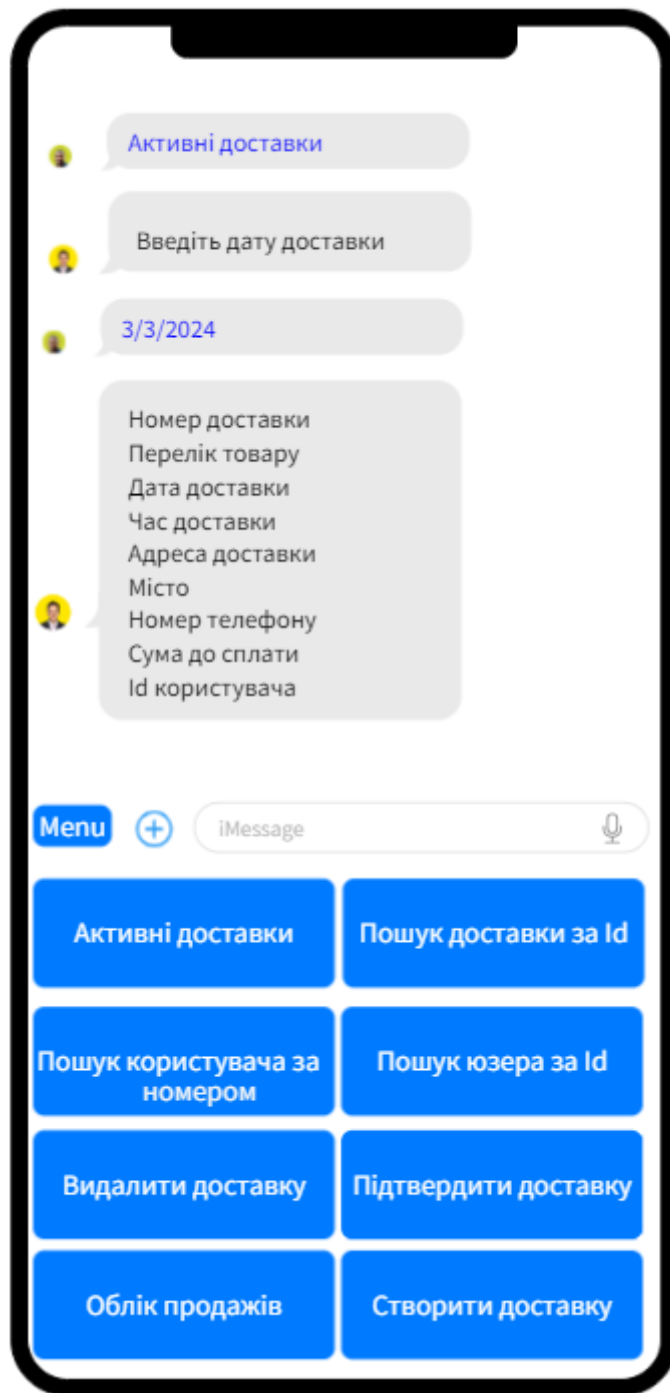


Рисунок 4.8 – Активні доставки

- прототип пошуку доставки за Id зображений на рисунку 4.9;

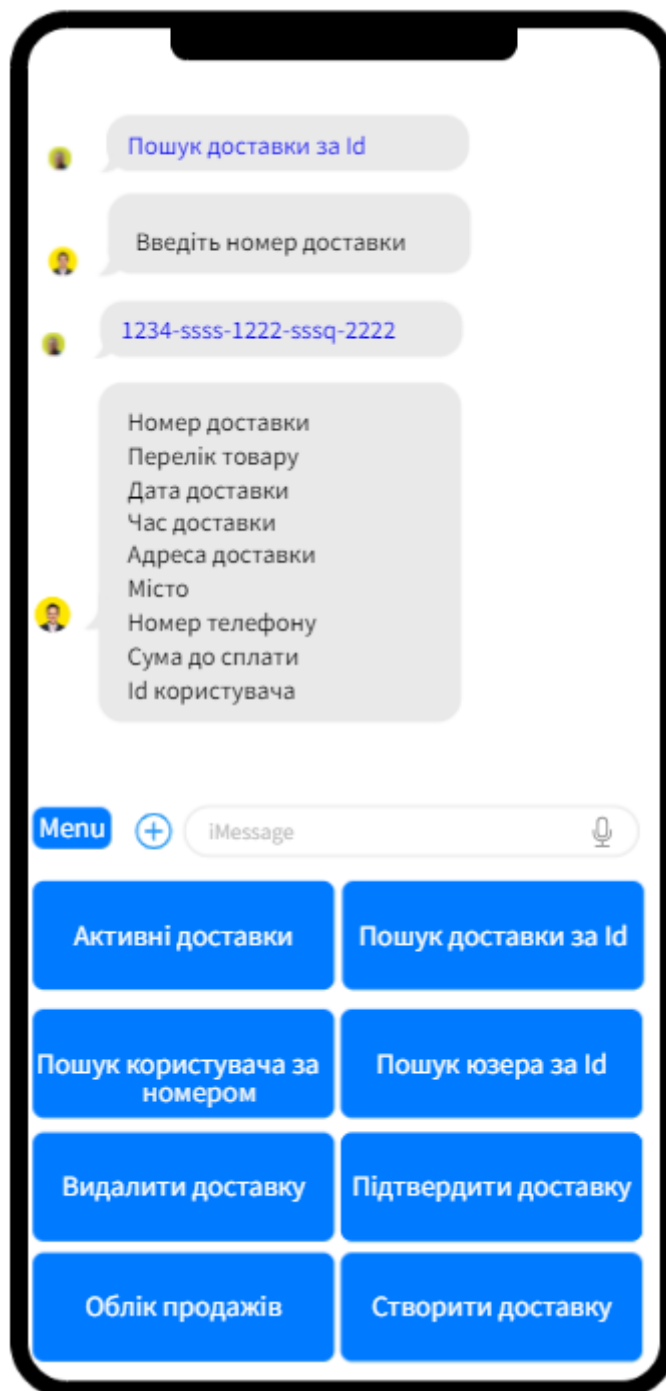


Рисунок 4.9 – Пошук доставки за Id

- прототип пошуку користувача за номером або Id зображений на рисунку 4.10;

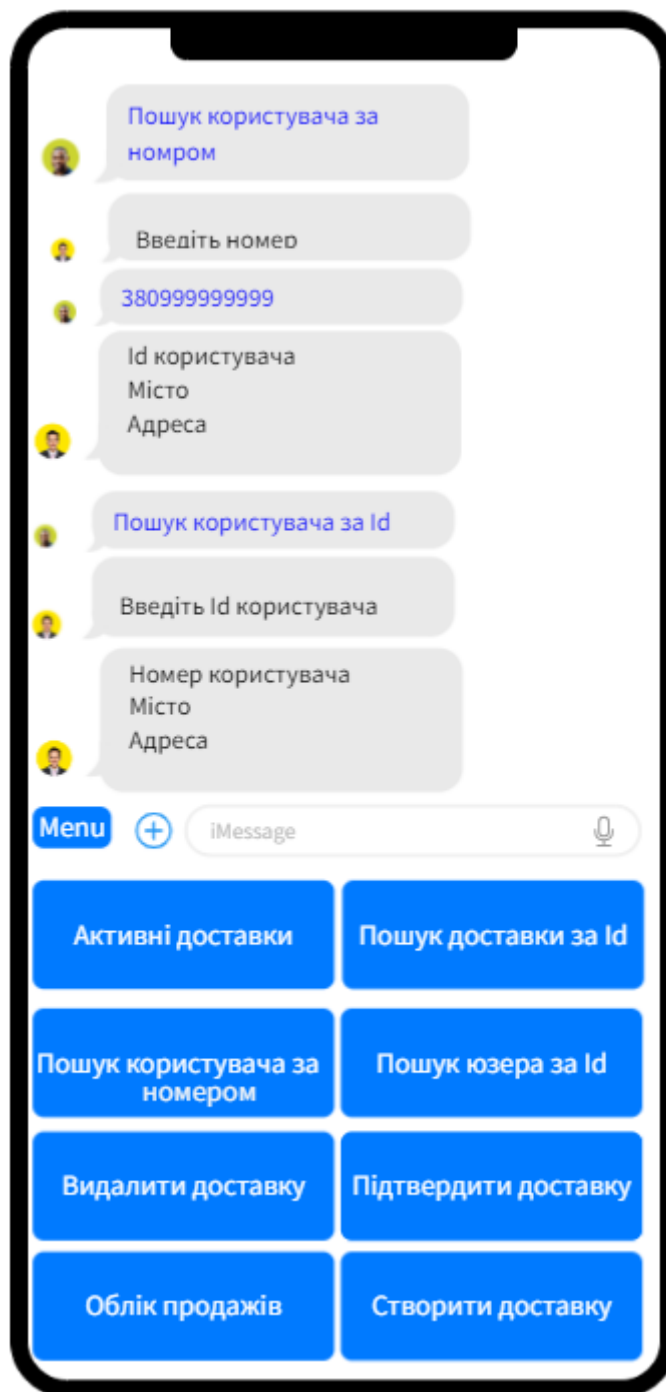


Рисунок 4.10 – Пошук користувача за номером або Id

- прототип обліку продажів зображений на рисунку 4.11;

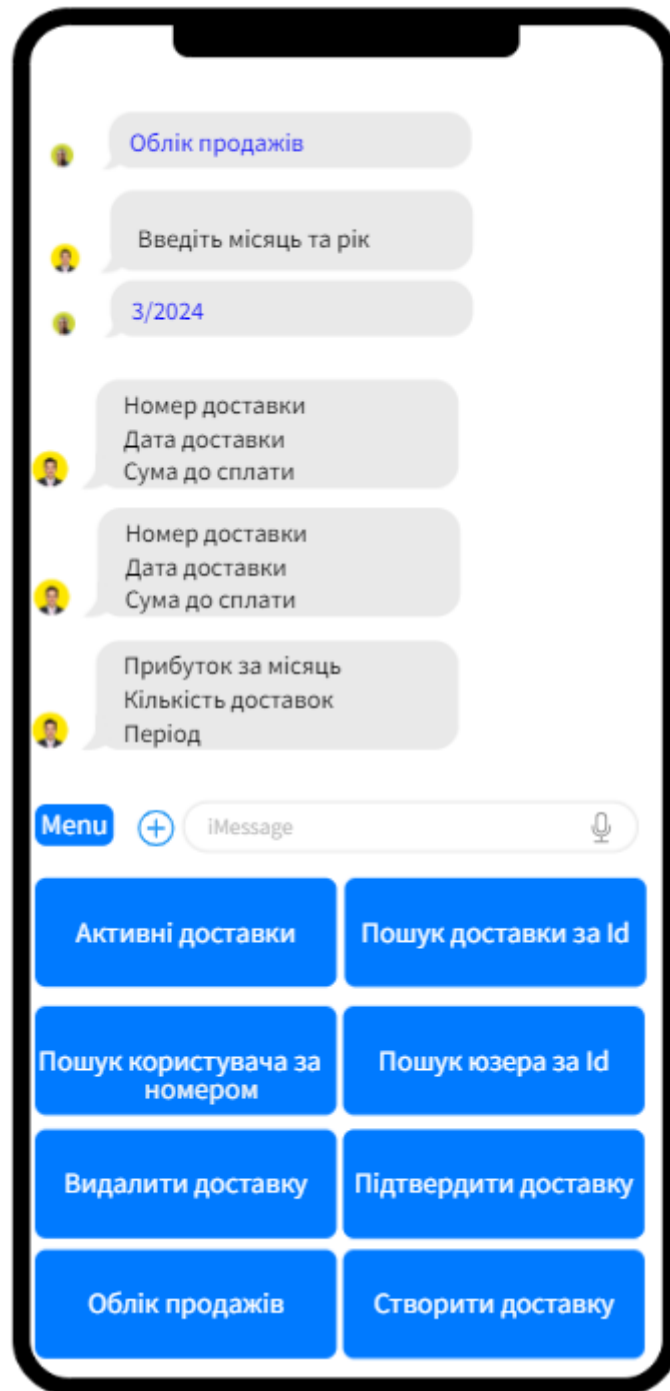


Рисунок 4.11 – Облік продажів

4.1.2 Для користувача:

- можливість додати або змінити номер телефону;
- можливість додати або оновити місто;
- можливість додати або оновити адресу;

- можливість переглянути відповіді на найбільш поширені питання;
- можливість створити та видалити замовлення;
- можливість переглянути додані товари в корзину;
- можливість змінити замовлення та додати інші товари;
- можливість оформити доставку;
- можливість обрати дату та час доставки;
- можливість переглянути інформацію про оформлену доставку.

4.1.3 Для адміністратора системи:

- можливість переглянути активні доставки;
- можливість знайти доставку за Id;
- можливість знайти користувача за номером телефону або Id;
- можливість підтвердити або видалити доставку;
- можливість створити нову доставку;
- можливість обліку продажів.

4.1.4 Додаткові вимоги:

Не висуваються.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Адміністратор системи повинен слідкувати за актуальністю інформації у телеграм боті, а також за актуальністю даних доставки.

4.3.2 Обслуговуючий персонал

Адміністратор, який вмiє аналізувати інформацію та обов'язками якого буде робота з телеграм ботом, модерація доставок, перевірка актуальності інформації у боті та ведення обліку продажів.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп'ютерах.

Мінімальна конфігурація технічних засобів:

- Intel Pentium 4 чи пізніше, що підтримує SSE3.
- Підключення до мережі інтернет.
- Об'єм ОЗП 2Гб.

Або

- наявність мобільного пристрою з операційною системою на базі Android з версією 6.0 та вище або IOS 7 та вище;
- наявність доступу до Інтернету;
- для встановлення додатку Телеграм на мобільному пристрої повинно бути не менше 200 МБ вільної пам'яті

Рекомендована конфігурація технічних засобів:

- тип процесору: Intel Core i5;
- об'єм ОЗП: 16 Гб;
- підключення до мережі Інтернет;

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows 7, Windows 8, Windows 8.1, Windows 10 чи пізніше), MacOS або Unix або на операційних системах Android чи iOS.

4.5.1 Вимоги до вхідних даних

Відсутні.

4.5.2 Вимоги до вихідних даних

Відсутні.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування C# за допомогою фреймворку ASP.Net Core.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі IDE Microsoft Visual Studio.

4.5.5 Вимоги до представлення вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторія на платформі GitHub.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

- пояснювальна записка;
- технічне завдання;
- текст програми;
- програма та методика тестування;
- керівництво користувача;
- керівництво адміністратора.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

- схема структурна варіантів використання;
- схема бази даних;
- креслення вигляду екранних форм;

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

**Пояснювальна записка
до дипломного проєкту**

на тему: **Вебзастосунок підвищення ефективності діяльності служби
доставки води**

КП.ІТ-0411.045440.02.81

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	4
ВСТУП	5
1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ.....	6
1.1 Аналіз предметної області.....	6
1.2 Аналіз існуючих рішень	7
1.2.1 Аналіз відомих програмних продуктів.....	7
1.2.2 Аналіз відомих алгоритмічних та технічних рішень.....	11
1.3 Опис бізнес-процесів	13
1.4 Постановка задачі	17
Висновки до розділу	17
2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	19
2.1 Варіанти використання програмного забезпечення.....	19
2.2 Аналіз системних вимог	30
2.3 Розроблення функціональних вимог	31
2.4 Розроблення нефункціональних вимог	38
Висновки до розділу	38
3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	40
3.1 Архітектура програмного забезпечення	40
3.2 Обґрунтування вибору засобів розробки.....	43
3.3 Конструювання програмного забезпечення	48
3.4 Аналіз безпеки даних.....	54
Висновки до розділу	55
4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	56
4.1 Аналіз якості ПЗ.....	56
4.2 Опис процесів тестування	57
4.3 Опис контрольного прикладу.....	65
Висновки до розділу	78

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	79
5.1 Розгортання програмного забезпечення	79
5.2 Супровід програмного забезпечення	90
Висновки до розділу	91
ВИСНОВКИ	93
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	95
ДОДАТОК А ЗВІТ ПОДІБНОСТІ.....	97

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	– Integrated Development Environment – інтегроване середовище розробки.
API	– Application programming interface – прикладний програмний Інтерфейс
REST	– Representational State Transfer – передача репрезентативного стану
TCP	– Transmission Control Protocol – протокол керування передачею
HTML	– HyperText Markup Language – мова гіпертекстової розмітки
HTTPS	– HyperText Transfer Protocol Secure – захищений протокол передачі гіпертексту
SOAP	– Simple Object Access Protocol – простий протокол доступу до об'єктів
IP	– Internet Protocol –інтернет-протокол
URL	– Uniform Resource Locator – єдиний вказівник на ресурс
XML	– Extensible Markup Language – розширювана мова розмітки
JSON	– JavaScript Object Notation – запис об'єктів JavaScript
NoSQL	– Not Only Structured Query Language – не тільки структурована мова запитів
DB	– Database – база даних
AWS	– Amazon Web Services – веб-сервіси Amazon
IT	– Інформаційні технології
ЦП	– Центральний процесор
Гб	– Гігабайт
ПЗ	– Програмне забезпечення
БД	– База даних.

ВСТУП

З розвитком та широким розповсюдженням інтернету багато сервісів перейшли в онлайн-простір. Крім можливостей для віртуального спілкування, інтернет надає доступ до різноманітних джерел корисної інформації для користувачів. Зростає кількість людей, які обирають інтернет-сервіси замість звичних стаціонарних точок обслуговування. Такі сервіси стають дедалі популярнішими завдяки зручності та простоті використання, адже вони доступні прямо з месенджерів. Так звані чат-боти, які спілкуються з користувачами через повідомлення, виконуючи численні спеціалізовані функції, мають значні переваги. Одна з основних переваг полягає в тому, що відповіді на запити користувачі отримують майже миттєво, оскільки обробка запитів відбувається швидше, ніж у людини.

На теперішній час в суспільстві з'являється все більше нових, небажаних але неминучих глобальних проблем. Доставка води на замовлення є дуже актуальною темою у сьогоденні, адже обмеження інколи унеможливають пересування людей по місту та знаходження у різноманітних точках продажу будь-яких товарів. Проте найбільш важливим чинником є саме велика кількість ручних процесів. У свою чергу це створює навантаження на персонал, що призводить до помилок в роботі, затримок та неефективності.

Основним завданням є розробка вебзастосунку, який забезпечує підвищення ефективності, за рахунок впровадження електронного каталогу товарів з актуальними цінами, створення електронної заявки в системі, а також адміністрування замовлень користувачів та ведення обліку доставок.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Інформаційні технології останнім часом відкривають значні можливості, пов'язані з чат-ботами. Вони настільки міцно увійшли в повсякденне життя, що використовуються у всіх сферах діяльності, і їхня роль постійно зростає. Це пояснюється тим, що люди проводять значну частину свого часу зі смартфонами, використовуючи e-mail клієнти та месенджери. Боти – це програми, які виконують різні завдання для користувачів у месенджерах. Залежно від поставленого завдання, можна виділити різні типи ботів[1]. Наприклад, це можуть бути боти, що взаємодіють із різними сервісами, такими як управління розумним будинком, або боти-утиліти, які показують погоду чи перекладають тексти.

Чат-боти – це спеціально створені акаунти, які не керуються людиною, а повідомлення, відправлені з них або на них, обробляються зовнішньою системою[2]. Спілкування з ботом для користувача виглядає як звичайне листування з реальною людиною. Існують такі типи чат-ботів за механізмом взаємодії з користувачем: кнопковий, текстовий і вбудований. Кнопковий бот використовує кнопки з варіантами дій для спілкування з користувачем. Текстовий бот розпізнає запити користувачів та надає релевантні відповіді. Вбудований бот з'являється всередині діалогу та пропонує варіанти дій.

Для створення вебзастосунку важливо вміти працювати з API, протоколами обміну даних, Telegram Bot API та базою даних. Одним із основних протоколів є HTTP, який використовується для передачі інформації в Інтернеті. HTTP є гнучким, але не дуже безпечним протоколом, оскільки дані передаються у вигляді звичайного тексту. Для безпечної комунікації існує HTTPS, який забезпечує аутентифікацію і шифрування даних.

Ботів можна класифікувати на три основні типи: самонавчаючі, скриптові та гібридні. Самонавчаючі боти використовують обробку природної мови та машинне навчання для формування відповідей. Скриптові боти працюють за

заздалегідь визначеним сценарієм, а гібридні боти комбінують обидва підходи, що дозволяє їм ефективно обробляти запити користувачів[3].

В рамках даного дипломного проєкту було обрано створення скриптового бота, завдяки якому можна буде замовити доставку води в одному із запропонованих міст України, переглянути створене замовлення, дізнатися його поточну ціну, отримати більше інформації про саму службу доставки, методи оплати та інше. Також надається можливість адмініструвати замовлення в системі та ведення обліку самих доставок.

1.2 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні технічні рішення у даній області, що допоможуть у реалізації вебзастосунку іпідвищення ефективності діяльності служби доставки води “Water Delivery”. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.2.1 Аналіз відомих програмних продуктів

Сфера послуг та виробничі компанії стрімко мігрують до Telegram. Ми продовжуємо спостерігати, наскільки успішно український бізнес автоматизує продаж за допомогою простого та ефективного засобу — ботів[4].

Тому для того щоб створити веб-застосунок потрібно визначити основні можливості готових програмних продуктів за предметною областю.

Наведемо декілька прикладів застосування саме Телеграм-ботів та розглянемо їхнє призначення:

1) Назва: «Delivery of water».

Ім'я облікового запису: @deliverywater_bot (рисунок 1.1).

Призначення: замовлення доставки води.

Функції: бот надає можливість швидко та ефективно замовити воду в певне місто України.

Ціль додатку: підвищення ефективності роботи служби доставки води за рахунок автоматизації.

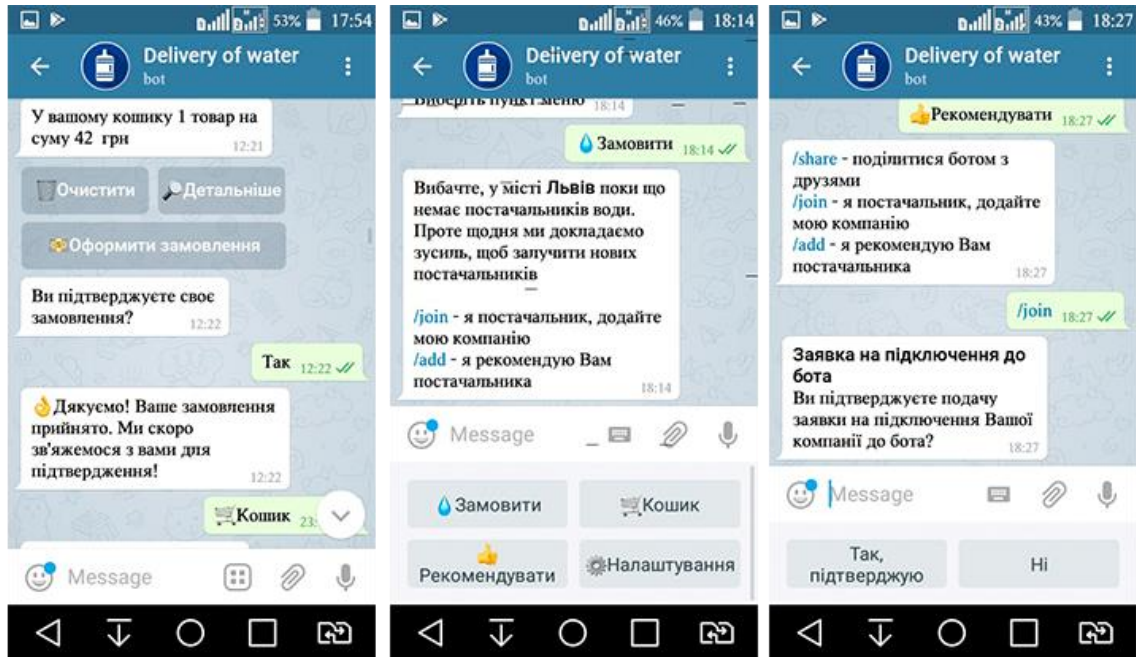


Рис. 1.1. Демонстрація роботи бота @deliverywater_bot

2) Назва: «Хвиля Здоров'я».

Ім'я облікового запису: @hz_voda_bot (рисунок 1.2).

Призначення: замовлення доставки води.

Функції: бот надає можливість швидко та ефективно замовити воду в певне місто України.

Ціль додатку: підвищення ефективності роботи служби доставки води за рахунок автоматизації.

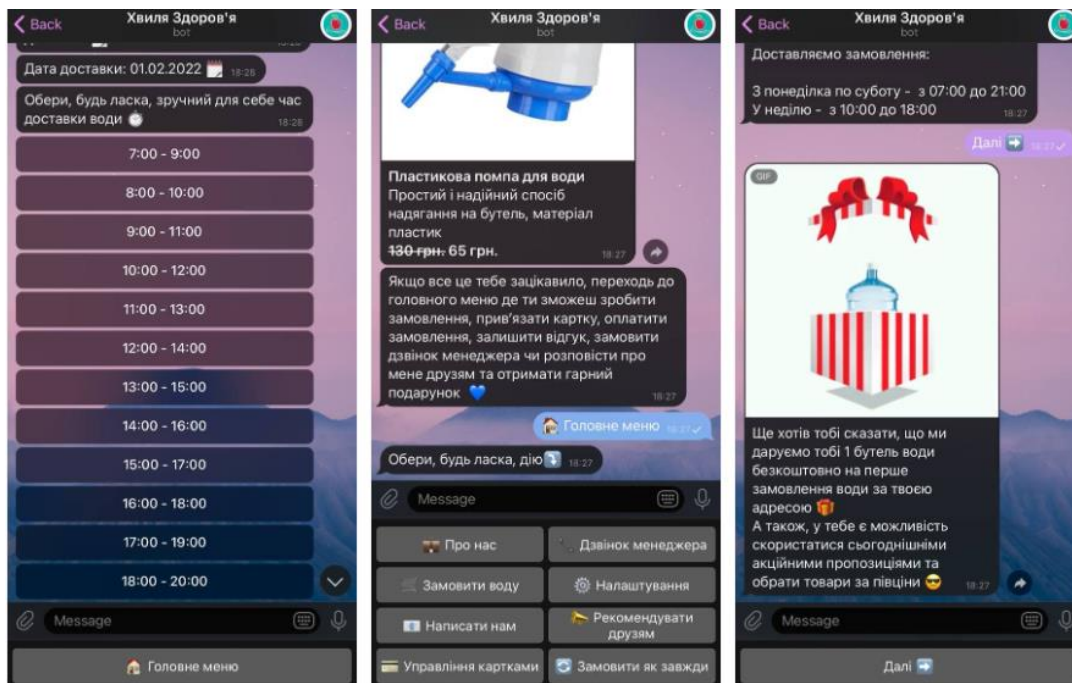


Рис. 1.2. Демонстрація роботи бота @hz_voda_bot

Для порівняння дипломного проєкту з аналогами, скористаємося таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	WDelivery_Bot	Хвиля Здоров'я	Delivery of water	Пояснення
Номер телефону	+	+	+	Додавання та зміна номеру телефону
Місто	+	-	+	Додавання та зміна міста
Адреса	+	+	-	Додавання та зміна адреси
Вибір води	+	-	-	Перелік доступних брендів води

Продовження таблиці 1.1

Кількість води	+	+	+	Вибір бажаної кількості бутлів
Літраж	+	-	+	Вибір бажаного літражу
Кошик	+	-	+	Додавання та перегляд товарів в кошику
Видалення з кошика	+	-	+	Видалення товарів з кошику
Дата	+	+	-	Вибір дати доставки
Час	+	+	-	Вибір часу доставки
Інформація про доставку	+	-	+	Перегляд інформації про доставку
FAQ	+	-	-	Інструкція користування ботом
Попередні доставки	+	-	-	Перегляд попередніх доставок
Інформація користувача	+	-	-	Отримання інформації про користувача (адміністратором)

Продовження таблиці 1.1

Адміністрування доставки	+	-	-	Підтвердження або видалення замовлення в системі (адміністратором)
Облік продажів	+	-	-	Інформація про отриманий прибуток (адміністратором)
Створення доставки	+	-	-	Створення доставки за запитом користувача (адміністратором)

Скориставшись порівняльною таблицею можна зробити висновок, що запропонований функціонал вебзастосунку підвищення ефективності діяльності служби доставки води, набагато ширший, аніж той, який наданий в інших програмних продуктах. Наявність ширшого функціоналу в поєднанні з інтуїтивно зрозумілим інтерфейсом, позитивно впливає на діяльність служби доставки води, збільшуючи потік клієнтів, а також підвищує ефективність роботи самої доставки, за рахунок мінімізації ручних процесів.

1.2.2 Аналіз відомих алгоритмічних та технічних рішень

Вибираючи платформу для створення бота, важливо враховувати її можливості. Розглянемо функціонал чат-ботів на найбільш популярних платформах.

На платформі Facebook Messenger чат-боти можуть використовувати текстові повідомлення, кнопки для вибору дій, структурні елементи і рахунки

на оплату. Щоб створити чат-бот на Facebook, необхідно мати публічну сторінку в цій соціальній мережі.

У Viber чат-боти не поступаються по функціоналу. Вони можуть відправляти розсилки повідомлень, оформляти пости у вигляді каруселі з товарами, вести взаємодію з аудиторією з публічного облікового запису в месенджері.

Telegram, як один із перших месенджерів, що надав можливість створювати ботів, має розширений функціонал. Чат-боти в Telegram можуть інтегруватися з іншими сервісами, працювати в інлайн-режимі, розв'язувати різні завдання (наприклад, передбачати погоду, перекладати тексти тощо), взаємодіяти з користувачами у формі гри, відправляти текстові повідомлення та коментарі, виконувати команди. Важливою особливістю є можливості для заміни інтерфейсу та налаштування клавіатури.

Кожна з цих платформ надає різноманітні можливості для розробки чат-ботів, що дозволяє підбирати оптимальний варіант для конкретного проекту залежно від його потреб та цілей.

Провівши аналіз було обрано створення чат-бота на платформі Telegram, тому подальший аналіз відомих алгоритмів для розв'язання задач чи підзадач у розробці направлений саме на цю платформу.

Робота з Bot API

Bot API – це інтерфейс, створений для розробників, що прагнуть створювати ботів. Він представляє собою набір методів, за допомогою яких одна програма може взаємодіяти з іншою. Програмні компоненти взаємодіють між собою за допомогою API, утворюючи ієрархію – високорівневі використовують API низькорівневих[5].

Документація пропонує два способи отримання оновлень: періодичні запити та встановлення веб-хуків. Оновлення зберігаються на сервері понад 24 години і повинні бути оброблені протягом цього часу. Спосіб періодичних запитів є доволі простим – проводиться періодичне опитування сервера Telegram. Інший спосіб – це веб-хуки, коли повідомлення від користувача

надходить на сервер, а месенджер сповіщає про це. Недоліком цього методу є необхідність розгортання повноцінного веб-сервера та наявність SSL-сертифіката, адже веб-хуки працюють лише через HTTPS.

Існують ще такі способи роботи з Bot API:

AIOGram – це асинхронний фреймворк для Telegram Bot API, написаний на Python 3.7. Він підходить для великих проєктів з високою відвідуваністю та тривалою підтримкою. Хоча бібліотека має проблеми з документацією, існує чат у самому месенджері, де можна поспілкуватися з розробником.

TeleBot – це оболонка над запитами до Telegram Bot API, яка спрощує написання коду та відповідає за його мінімізацію. Типи TeleBot повністю відповідають визначенням типів API Telegram, за винятком поля Message, яке названо `from_user` (оскільки `from` – це зарезервоване слово в Python). При створенні програм, які не вимагають високої швидкості виконання, слід зауважити, що атрибут `message.chat` може належати як конкретному користувачеві, так і чату спільноти.

Провівши аналіз в рамках даного дипломного проєкту, було обрано створення скриптового бота за допомогою Telegram Bot API з використанням періодичних запитів спрямованих на сервер, адже цей спосіб розробки є доволі простим, не потребує складних налаштувань, не вимагає розгортання повноцінного веб-сервера на пристрої, де запускається бот, що спрощує інфраструктуру та знижує витрати. А також даний метод розробки є зручним для налаштування та обслуговування, що дозволяє швидко впроваджувати зміни та оновлення.

1.3 Опис бізнес-процесів

Для опису бізнес процесу програмного забезпечення використовується BPMN модель (рисунки 1.3, 1.4, 1.5 та 1.6).

Опис послідовності створення замовлення для користувача:

- користувач натискає на кнопку «Замовити»;

- сервер, отримавши GET запит, перевіряє, чи користувач ввів номер телефону, місто та адресу;
- якщо користувач не ввів дані, з'явиться відповідне повідомлення та відобразиться меню налаштувань;
- якщо користувач ввів дані, з'явиться панель вибору бренду води, літражу та бажаної кількості;
- після вибору товару, сервер отримає POST запит, відобразиться меню кошику та повідомлення про доданий товар;

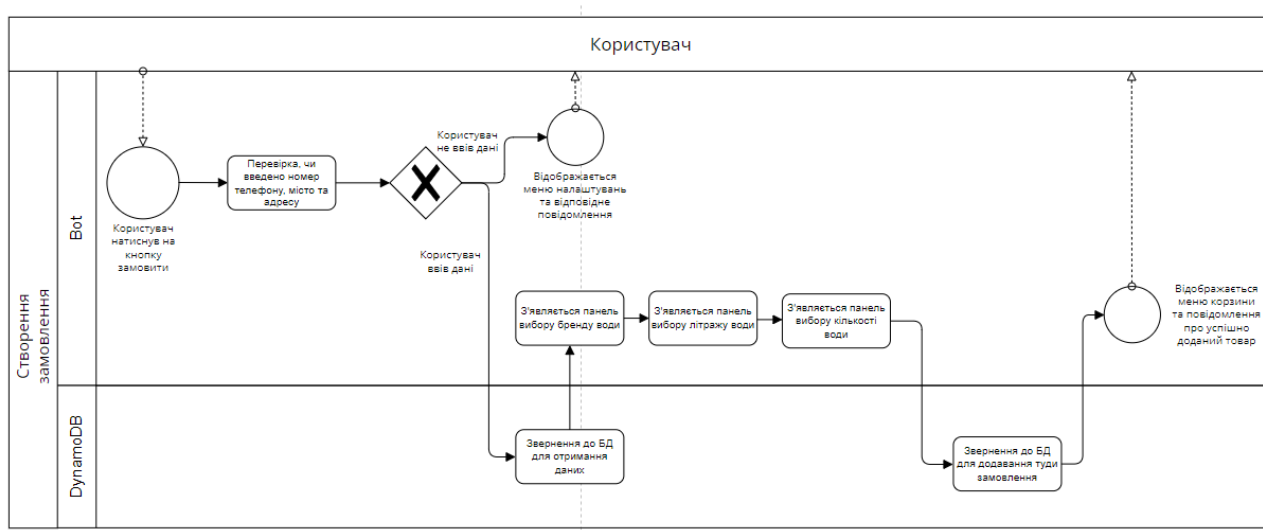


Рисунок 1.3 – BPMN модель створення замовлення

Опис послідовності оформлення доставки для користувача:

- користувач натискає на кнопку «Кошик»;
- сервер, отримавши GET запит, перевіряє, чи є товари в кошику;
- якщо товарів немає, з'явиться відповідне повідомлення та відобразиться меню налаштувань;
- якщо товари є, з'явиться меню кошика та кнопка «Оформити замовлення»;
- після чого з'явиться панель з вибором дати та часу доставки;
- сервер отримує PATCH запит, відображається головне меню та повідомлення про оформлене замовлення;

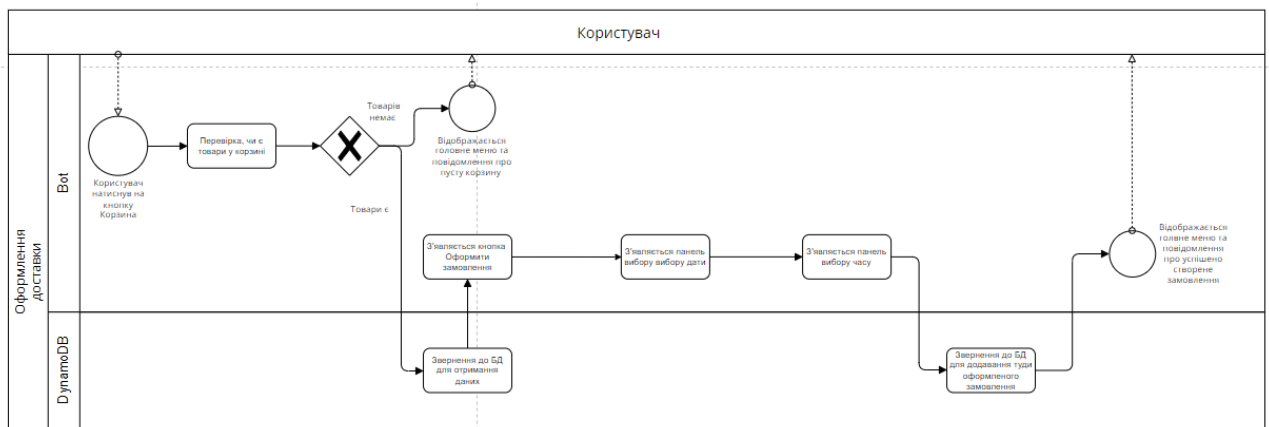


Рисунок 1.4 – BPMN модель оформлення доставки

Опис послідовності адміністрування доставки (для адміністратора):

- адміністратор натискає на кнопку «Активні доставки»;
- з’являється поле для вводу дати;
- адміністратор вводить необхідну дату;
- сервер, отримавши GET запит, перевіряє, чи є активні доставки на вказану дату;
 - якщо доставок немає, з’явиться відповідне повідомлення та відобразиться меню налаштувань;
 - якщо доставки є, вони виведуться в повідомленні, після чого з’явиться меню, де будуть кнопки «Підтвердити доставку» та «Видалити доставку»;
 - адміністратор натискає на одну з двох кнопок відповідно;
 - з’являється поле для вводу ID доставки;
 - адміністратор вводить необхідний ID;
 - сервер, отримавши GET запит, перевіряє, чи є доставки з вказаним ID;
 - якщо доставки не існують, з’явиться відповідне повідомлення та відобразиться меню налаштувань;
 - якщо доставка існує, вона буде успішно підтверджена/видалена, з’явиться відповідне повідомлення та відобразиться меню налаштувань;

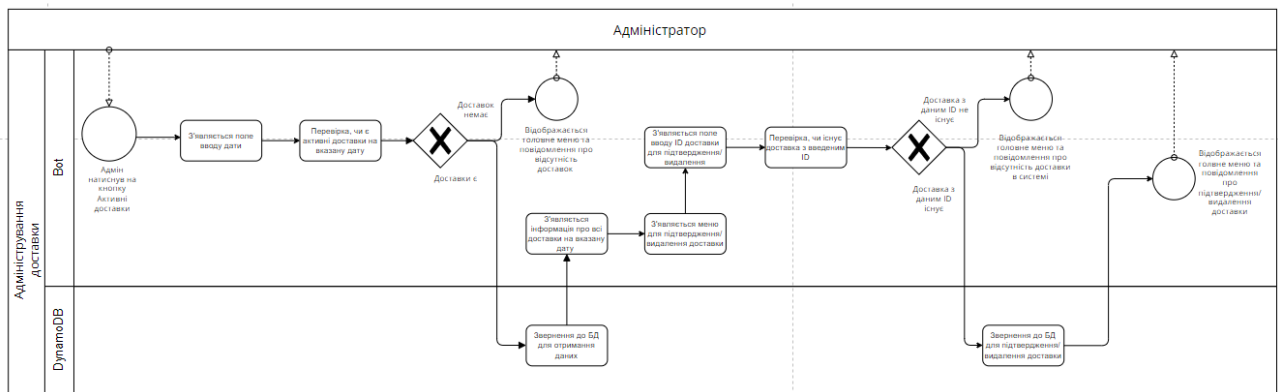


Рисунок 1.5 – BPMN модель адміністрування доставки

Опис послідовності обліку продаж (для адміністратора):

- адміністратор натискає на кнопку «Облік продажів»;
- з'являється поле для вводу дати за конкретний період;
- адміністратор вводить необхідну дату;
- сервер, отримавши GET запит, перевіряє, чи були доставки на вказаний період;
 - якщо доставок не було, з'явиться відповідне повідомлення та відобразиться меню налаштувань;
 - якщо доставки були, вони виведуться в повідомленні, після чого з'явиться головне меню та інформація про прибуток служби доставки;

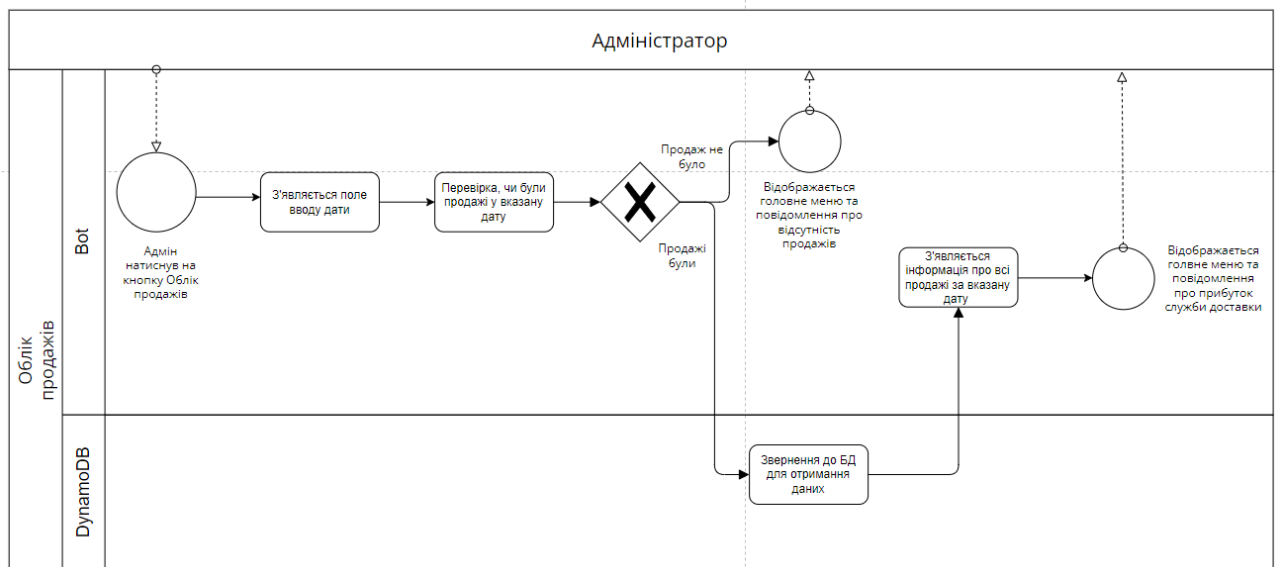


Рисунок 1.6 – BPMN модель обліку продажів

1.4 Постановка задачі

Метою розробки є підвищення ефективності роботи служби доставки води за рахунок впровадження електронного каталогу товарів з актуальними цінами та створення електронної заявки в системі. А також адміністрування замовлень користувачів та ведення обліку доставок, що у свою чергу автоматизує певні процеси роботи діяльності служби доставки води.

Користувач матиме змогу замовити доставку води в одному із запропонованих міст України, переглянути створене замовлення, дізнатися його поточну ціну та отримати більше інформації про саму службу доставки, а також інструкцію по користуванню самим ботом. В свою чергу адміністратор може переглянути та обробити створене замовлення, створити замовлення в системі за користувача та провести облік продажів для отримання інформації про прибуток компанії.

Висновки до розділу

Щодня месенджери стають невід'ємною складовою життя все більшої кількості людей. Вони використовуються не тільки для спілкування і обміну файлами, а й для підвищення ефективності роботи служб за допомогою

автоматизації. Оскільки, Telegram у сьогодні є одним із месенджерів, який має найбільшу аудиторію користувачів, то створення і розміщення чат-боту в його середовищі є актуальним.

Чат-боти Telegram не тільки автоматизовують ручні процеси, які доволі часто призводять до помилок, затримок та неефективності, але й збільшують кількість продажів. Завдяки цим додаткам в короткий термін можливо отримати цільову аудиторію. Як показує практика, використання чат-ботів для підтримки клієнтів може зменшити до 35% часу консультантів в онлайн-чатах. Крім того, більшість запитів від користувачів надходять у неробочий час. На відміну від живої людини, бот здатний одночасно давати відповіді декільком користувачам. Більш того, відповідь дається миттєво. Великою перевагою чат-ботів є робота на різних платформах.

У розділі надано поняття «чат-боту», проведено аналіз відомих технічних та алгоритмічних рішень, наведено переваги взаємодії користувача з додатком над взаємодією з оператором, описано бізнес-процеси. Також було розглянуто існуючі рішення на прикладі схожих вебзастосунків.

Результатом проведеного дослідження стало рішення про створення вебзастосунку підвищення ефективності роботи служби доставки води. Система є сучасною та зручною, а також задовольняє всім вимогам з точки зору функціональності, використання та логіки.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Головною метою програмного забезпечення є підвищення ефективності діяльності служби доставки води, більше функцій можна побачити на рисунках 2.1-2.2.

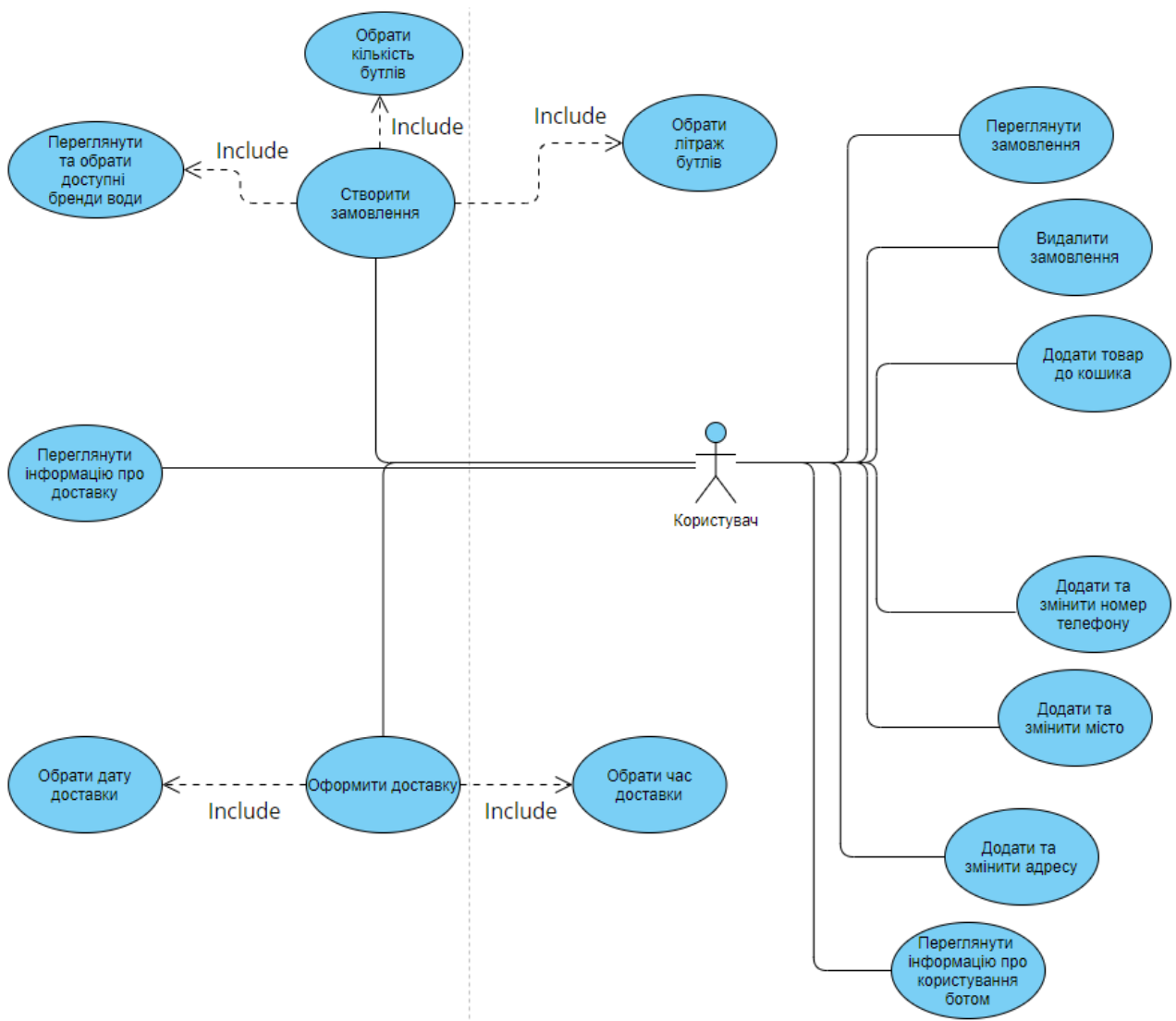


Рисунок 2.1 – Діаграма варіантів використання користувача

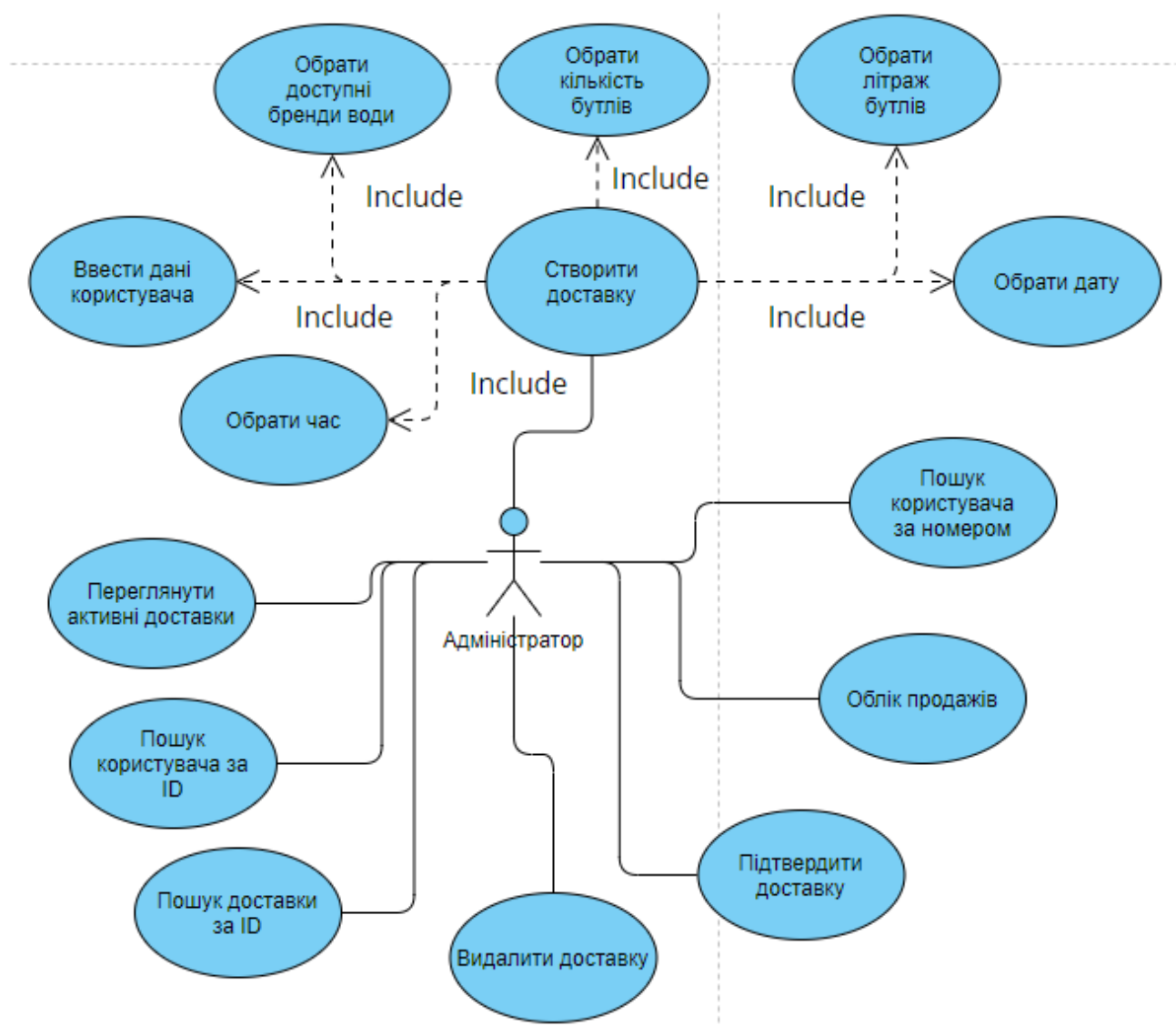


Рисунок 2.2 – Діаграма варіантів використання адміністратора

В таблицях 2.1 - 2.18 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 - Варіант використання UC-1

Use case name	Додати або оновити номер телефону
Use case ID	UC-01
Goals	Додати або оновити номер телефону користувача в базі даних

Продовження таблиці 2.1

Actors	Користувач
Trigger	Користувач бажає додати або оновити номер телефону
Pre-conditions	-
Flow of Events	Користувач натискає на кнопку «Налаштування» та вибирає пункт «Номер телефону». Номер користувача автоматично підтягується за допомогою вбудованого функціоналу Telegram і користувач отримує повідомлення, про підтвердження надання свого номеру. Після цього з'являється повідомлення про успішне додавання номеру телефону.
Extension	Якщо користувач відмовиться від підтвердження надання свого номеру, то його поверне до меню налаштування.
Post-Condition	Користувач успішно додав або оновив номер телефону

Таблиця 2.2 - Варіант використання UC-2

Use case name	Додати або оновити місто
Use case ID	UC-02
Goals	Додати або оновити місто користувача в базі даних
Actors	Користувач
Trigger	Користувач бажає додати або оновити місто
Pre-conditions	-
Flow of Events	Користувач натискає на кнопку «Налаштування» та вибирає пункт «Місто». Після чого з'являється кнопковий перелік доступних міст, юзер обирає потрібне для нього. Після цього з'являється повідомлення про успішне додавання міста.
Extension	Якщо користувач не вибере жодне місто або введе некоректні дані, то його поверне до меню налаштування.
Post-Condition	Користувач успішно додав або оновив місто

Таблиця 2.3 - Варіант використання UC-3

Use case name	Додати або оновити адресу
Use case ID	UC-03
Goals	Додати або оновити адресу користувача в базі даних
Actors	Користувач
Trigger	Користувач бажає додати або оновити адресу

Продовження таблиці 2.3

Pre-conditions	-
Flow of Events	Користувач натискає на кнопку «Налаштування» та вибирає пункт «Адреса». Після чого з'являється повідомлення із зразком, як правильно ввести адресу доставки. Користувач додає свою. Після цього з'являється повідомлення про успішне додавання адреси.
Extension	Якщо користувач відмовиться вводити адресу або введе некоректні дані, то його поверне до меню налаштування.
Post-Condition	Користувач успішно додав або оновив адресу

Таблиця 2.4 - Варіант використання UC-4

Use case name	Перегляд інструкції користувача
Use case ID	UC-04
Goals	Отримати інформацію по користуванню ботом
Actors	Користувач
Trigger	Користувач бажає отримати інформацію по користуванню ботом
Pre-conditions	-
Flow of Events	Користувач за допомогою команди /menu переходить до основного меню та натискає на кнопку «FAQ». Після чого з'являється меню з кнопками «Про нас», «Оплата», «Оновлення даних» та інші. Натиснувши на потрібну кнопку користувач побачить повідомлення з детальною інформацією по потрібному питанню. Також надається контакт адміністратора на випадок виникнення проблем.
Extension	Некоректних дій не передбачено.
Post-Condition	Користувач успішно отримав інформацію про службу доставки

Таблиця 2.5 - Варіант використання UC-5

Use case name	Створення замовлення
Use case ID	UC-05
Goals	Створити замовлення
Actors	Користувач
Trigger	Користувач бажає створити замовлення

Продовження таблиці 2.5

Pre-conditions	Користувач повинен ввести дані номеру телефону, місто та адресу доставки
Flow of Events	Користувач з головного меню натискає на кнопку «Замовити» після чого з'являється перелік брендів води та помпи і користувач обирає один із запропонованих. Далі обирає літраж та кількість бутлів і замовлення додається в кошик. Після чого з'являється повідомлення про те, що товар було успішно додано до кошика та користувача направляє до меню кошика, де він зможе додати інший товар, переглянути активні товари в кошику або ж повернутися до головного меню.
Extension	Якщо користувач не вибере бренд, літраж та кількість води, або введе некоректні дані, замовлення не буде створене і його поверне до головного меню, а також з'явиться відповідне повідомлення.
Post-Condition	Користувач успішно створив замовлення

Таблиця 2.6 - Варіант використання UC-6

Use case name	Перегляд замовлення
Use case ID	UC-06
Goals	Переглянути замовлення
Actors	Користувач
Trigger	Користувач бажає переглянути замовлення
Pre-conditions	Користувач повинен додати хоча б один товар до кошика та має бути зареєстрованим в системі
Flow of Events	Користувач з головного меню натискає на кнопку «Кошик» після чого з'являється повідомлення з переліком товарів, які користувач додав раніше. Далі користувач може оформити замовлення, видалити товари з кошика або ж повернутися до головного меню.
Extension	Якщо користувач не натисне на потрібну кнопку або введе некоректні дані, товари з кошику не відобразяться і його поверне до головного меню.
Post-Condition	Користувач успішно переглянув замовлення

Таблиця 2.7 - Варіант використання UC-7

Use case name	Видалення замовлення
Use case ID	UC-07
Goals	Видалити замовлення
Actors	Користувач
Trigger	Користувач бажає видалити замовлення
Pre-conditions	Користувач повинен додати хоча б один товар до кошика
Flow of Events	Користувач з головного меню натискає на кнопку «Кошик» після чого з'являється повідомлення з переліком товарів, які користувач додав раніше. Далі користувач обирає кнопку «Видалити товари», після чого з'являється меню, де користувач обирає кнопку «Підтвердити видалення». Користувачу надсилається повідомлення про те, що товар успішно видалено.
Extension	Якщо користувач не натисне на потрібну кнопку або введе некоректні дані, товари з кошику не видаляються і його поверне до головного меню.
Post-Condition	Користувач успішно видалив замовлення

Таблиця 2.8 - Варіант використання UC-8

Use case name	Додавання іншого товару до кошика
Use case ID	UC-08
Goals	Додати товар до кошика
Actors	Користувач
Trigger	Користувач бажає додати інший товар до кошика
Pre-conditions	Користувач повинен додати хоча б один товар до кошика
Flow of Events	Користувач з головного меню натискає на кнопку «Замовити» після чого з'являється перелік брендів води та помпи і користувач обирає один з запропонованих. Далі обирає літраж та кількість бутлів і товар додається в кошик. Після чого з'являється повідомлення про те, що товар було успішно додано до кошика та користувача направляє до меню кошика, де він зможе додати інший товар, переглянути активні товари в кошику або ж повернутися до головного меню.

Продовження таблиці 2.8

Extension	Якщо користувач не вибере бренд, літраж та кількість води, або введе некоректні дані, замовлення не буде створене і його поверне до головного меню, а також з'явиться відповідне повідомлення.
Post-Condition	Користувач успішно додав інший товар до кошика

Таблиця 2.9 - Варіант використання UC-9

Use case name	Оформлення доставки
Use case ID	UC-09
Goals	Оформити доставку
Actors	Користувач
Trigger	Користувач бажає оформити доставку
Pre-conditions	Користувач повинен додати хоча б один товар до кошика
Flow of Events	Користувач з головного меню натискає на кнопку «Кошик» після чого з'являється повідомлення з переліком товарів, які користувач додав раніше. Далі користувач обирає кнопку «Оформити замовлення», після чого з'являється вибір дня та часу. Далі висвітиться повідомлення, що замовлення успішно створено.
Extension	Якщо користувач не вибере день та час, або введе некоректні дані, доставка не буде оформлена.
Post-Condition	Користувач успішно оформив доставку

Таблиця 2.10 - Варіант використання UC-10

Use case name	Перегляд інформації про доставку
Use case ID	UC-10
Goals	Переглянути інформацію про активну доставку
Actors	Користувач
Trigger	Користувач бажає переглянути інформацію про активну доставку
Pre-conditions	Користувач повинен оформити доставку

Продовження таблиці 2.10

Flow of Events	Користувач з головного меню натискає на кнопку «Створені доставки», після чого з'являється повідомлення з інформацією про доставку, де вказано перелік товару, кількість, літраж, ціну за кожен товар окремо і сумарно, дату та час доставки, номер телефону користувача, а також місто та адресу доставки.
Extension	Якщо користувач натисне не на потрібні кнопки або введе некоректні дані, інформація про доставку не відобразиться.
Post-Condition	Користувач переглянув інформацію про доставку

Таблиця 2.11 - Варіант використання UC-11

Use case name	Перегляд активних доставок
Use case ID	UC-11
Goals	Отримати список активних доставок
Actors	Адміністратор
Trigger	Адміністратор бажає отримати список активних доставок
Pre-conditions	-
Flow of Events	Адміністратор в головному меню натискає на кнопку «Активні доставки» і вводить потрібну дату. Після цього з'являється повідомлення з активними доставками.
Extension	Якщо адміністратор введе некоректні дані, то з'явиться відповідне повідомлення і його поверне до головного меню.
Post-Condition	Адміністратор отримав список активних доставок

Таблиця 2.12 - Варіант використання UC-12

Use case name	Пошук доставки за Id
Use case ID	UC-12
Goals	Знайти доставку за Id
Actors	Адміністратор
Trigger	Адміністратор бажає знайти доставку за Id
Pre-conditions	-
Flow of Events	Адміністратор в головному меню натискає на кнопку «Пошук доставки за Id» і вводить потрібний Id. Після цього з'являється повідомлення з даними про доставку.

Продовження таблиці 2.12

Extension	Якщо адміністратор введе некоректні дані, то з'явиться відповідне повідомлення і його поверне до головного меню.
Post-Condition	Адміністратор отримав доставку за Id

Таблиця 2.13 - Варіант використання UC-13

Use case name	Пошук користувача за номером
Use case ID	UC-13
Goals	Знайти користувача за номером
Actors	Адміністратор
Trigger	Адміністратор бажає знайти користувача за номером
Pre-conditions	-
Flow of Events	Адміністратор в головному меню натискає на кнопку «Пошук користувача за номером» і вводить потрібний номер. Після цього з'являється повідомлення з даними про користувача.
Extension	Якщо адміністратор введе некоректні дані, то з'явиться відповідне повідомлення і його поверне до головного меню.
Post-Condition	Адміністратор знайшов користувача за номером

Таблиця 2.14 - Варіант використання UC-14

Use case name	Пошук користувача за Id
Use case ID	UC-14
Goals	Знайти користувача за Id
Actors	Адміністратор
Trigger	Адміністратор бажає знайти користувача за Id
Pre-conditions	-
Flow of Events	Адміністратор в головному меню натискає на кнопку «Пошук користувача за Id» і вводить потрібний Id. Після цього з'являється повідомлення з даними про користувача.
Extension	Якщо адміністратор введе некоректні дані, то з'явиться відповідне повідомлення і його поверне до головного меню.
Post-Condition	Адміністратор знайшов користувача за Id

Таблиця 2.15 - Варіант використання UC-15

Use case name	Видалити доставку
Use case ID	UC-15
Goals	Видалити доставку
Actors	Адміністратор
Trigger	Адміністратор бажає видалити доставку
Pre-conditions	-
Flow of Events	Адміністратор в головному меню натискає на кнопку «Видалити доставку» і вводить потрібний Id. Після цього з'являється повідомлення з даними про успішне видалення доставки.
Extension	Якщо адміністратор введе некоректні дані, то з'явиться відповідне повідомлення і його поверне до головного меню.
Post-Condition	Адміністратор успішно видалив доставку

Таблиця 2.16 - Варіант використання UC-16

Use case name	Підтвердити доставку
Use case ID	UC-16
Goals	Підтвердити доставку
Actors	Адміністратор
Trigger	Адміністратор бажає підтвердити доставку
Pre-conditions	-
Flow of Events	Адміністратор в головному меню натискає на кнопку «Підтвердити доставку» і вводить потрібний Id. Після цього з'являється повідомлення з даними про успішне підтвердження доставки.
Extension	Якщо адміністратор введе некоректні дані, то з'явиться відповідне повідомлення і його поверне до головного меню.
Post-Condition	Адміністратор успішно підтвердив доставку

Таблиця 2.17 - Варіант використання UC-17

Use case name	Створення нової доставки
Use case ID	UC-17

Продовження таблиці 2.17

Goals	Створити нову доставку
Actors	Адміністратор
Trigger	Адміністратор бажає створити нову доставку
Pre-conditions	-
Flow of Events	Адміністратор в головному меню натискає на кнопку «Створення доставки» після чого вводить дані користувача, а саме адресу, місто та номер телефону. Далі з'являється перелік брендів води та помпи., літраж, кількість бутлів. Також потрібно вибрати дату та час доставки. Адміністратор обирає відповідно до отриманого запиту від користувача. Після цього з'являється повідомлення про те, що доставка була успішно створена та відображається головне меню.
Extension	Якщо адміністратор введе некоректні дані, то з'явиться відповідне повідомлення і його поверне до головного меню.
Post-Condition	Адміністратор створив нову доставку

Таблиця 2.18 - Варіант використання UC-18

Use case name	Облік продажів
Use case ID	UC-18
Goals	Переглянути інформацію про виконані доставки
Actors	Адміністратор
Trigger	Адміністратор бажає отримати інформацію про виконані доставки
Pre-conditions	В базі даних має бути хоча б одна виконана доставка
Flow of Events	Адміністратор в головному меню натискає на кнопку «Облік продажів» і вводить потрібну дату. Після цього з'являється повідомлення з виконаними доставками за певний місяць та підраховується прибуток самої служби доставки.
Extension	Якщо адміністратор введе некоректні дані, то з'явиться відповідне повідомлення і його поверне до головного меню.
Post-Condition	Адміністратор отримав інформацію про виконані доставки

2.2 Аналіз системних вимог

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп'ютерах.

Рекомендовані характеристики можуть варіюватись в залежності від очікуваного навантаження, проте через відсутність складних операцій із даними, особливих змін не буде навіть при великій кількості одночасних користувачів.

Мінімальна конфігурація технічних засобів:

- Intel Pentium 4 чи пізніше, що підтримує SSE3;
- об'єм ОЗП: 2 Гб;
- підключення до мережі Інтернет;

Рекомендована конфігурація технічних засобів:

- Intel Pentium 4 чи пізніше, що підтримує SSE3;
- об'єм ОЗП: 4 Гб;
- підключення до мережі Інтернет;

Програмне забезпечення повинно працювати під управлінням операційних систем WIN32 (Windows 7, Windows 8, Windows 8.1, Windows 10 чи пізніше), MacOS, Unix, Android або iOS.

Intel Pentium 4 – це процесор з підтримкою базових інструкцій для обробки даних, достатній для виконання основних операцій бота. SSE3 (Streaming SIMD Extensions 3) дозволяє оптимізувати роботу з мультимедійними та числовими даними, що може бути корисним навіть у простих додатках.

2 Гб оперативної пам'яті достатньо для роботи Telegram бота, оскільки він не виконує складних обчислювальних задач і не вимагає великих обсягів пам'яті для зберігання даних. Більшість операцій, таких як обробка повідомлень та взаємодія з API, не є ресурсомісткими.

Обрані системні вимоги забезпечують оптимальний баланс між доступністю, продуктивністю і стабільністю роботи Telegram бота. Вони

враховують можливості різних апаратних та програмних платформ, забезпечуючи ефективну роботу бота у широкому спектрі умов та середовищ.

2.3 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.19 наведено загальну модель вимог, а в таблицях 2.20 – 2.37 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.3.

Таблиця 2.19 – Модель вимог у загальному вигляді

Id	Full description	Code	Priority	Workload	Risk	Status
1.	Вибір або зміна міста	REQ-1	1	0	-	Approved
2.	Додавання або зміна номера телефону	REQ-2	1	0	-	Approved
3.	Ввід або зміна адреси	REQ-3	1	0	-	Approved
4.	Створення замовлення	REQ-4	2	0	-	Approved
4.1	Перевірка, чи додано номер телефону, місто та адресу користувача	REQ-5	-	0	-	Approved
4.2	Вибір бренда води	REQ-6	-	0	-	Approved
4.3	Вибір літражу	REQ-7		0	-	Approved
4.4	Введення бажаної кількості води	REQ-8	-	0	-	Approved

Продовження таблиці 2.19

4.4.1	Перевірка на коректність введення	REQ-9	-	0	-	Approved
4.4.2	Перевірка, чи не перевищено ліміт кількості води	REQ-10	-	0	-	Approved
5.	Додавання іншого товару в кошик	REQ-11	3	0	-	Approved
6.	Видалення товарів з кошика	REQ-12	3	0	-	Approved
6.1	Перевірка, чи є товари в кошику	REQ-13	-	0	-	Approved
6.2	Підтвердження видалення товарів	REQ-14	-	0	-	Approved
7.	Перегляд товарів в кошику	REQ-15	-	0	-	Approved
7.1	Перевірка, чи є товари в кошику	REQ-16	-	0	-	Approved
8.	Оформлення доставки	REQ-17	2	0	-	Approved
8.1	Вибір дати	REQ-18	-	0	-	Approved
8.2	Вибір часу	REQ-19	-	0	-	Approved
9.	Перегляд інформації про доставку	REQ-20	3	0	-	Approved
9.1	Перевірка, чи створена доставка	REQ-21	-	0	-	Approved

Продовження таблиці 2.19

10.	Перегляд інформації по користуванню ботом	REQ-22	3	0	-	Approved
11.	Перегляд активних доставок	REQ-23	-	0	-	Approved
11.1	Перевірка, чи є створені доставки	REQ-24	-	0	-	Approved
12.	Пошук доставки за Id	REQ-25	-	0	-	Approved
12.1	Перевірка, чи існує доставка з веденим Id	REQ-26	-	0	-	Approved
13.	Пошук користувача за номером	REQ-27	-	0	-	Approved
13.1	Перевірка, чи існує користувач з веденим номером	REQ-28	-	0	-	Approved
14.	Пошук користувача за Id	REQ-29	-	0	-	Approved
14.1	Перевірка, чи існує користувач з веденим Id	REQ-30	-	0	-	Approved
15.	Видалення доставки	REQ-31	2	0	-	Approved
15.1	Перевірка, чи існує доставка з веденим Id	REQ-32	-	0	-	Approved
16.	Підтвердження доставки	REQ-33	2	0	-	Approved
16.1	Перевірка, чи існує доставка з веденим Id	REQ-34	-	0	-	Approved

Продовження таблиці 2.19

17.	Створення доставки	REQ-35	2	0	-	Approved
17.1	Введення даних про місто, адресу та номер телефону	REQ-36	-	0	-	Approved
17.2	Вибір бренда води	REQ-37	-	0	-	Approved
17.3	Вибір літражу	REQ-38		0	-	Approved
17.4	Введення бажаної кількості води	REQ-39	-	0	-	Approved
17.5	Вибір дати доставки	REQ-40	-	0	-	Approved
17.6	Вибір часу доставки	REQ-41	-	0	-	Approved
18.	Облік продажів	REQ-42	2	0	-	Approved
18.1	Перевірка чи були виконані доставки за період часу	REQ-43	-	0	-	Approved

Таблиця 2.20 – Функціональна вимога FR-1

Назва	Додавання або зміна номеру телефону
Опис	Система повинна надавати можливість додати або змінити номер телефону користувачеві.

Таблиця 2.21 – Функціональна вимога FR-2

Назва	Додавання або зміна міста
Опис	Система повинна надавати можливість додати або змінити місто користувачеві.

Таблиця 2.22 – Функціональна вимога FR-3

Назва	Додавання або зміна адреси
Опис	Система повинна надавати можливість додати або змінити адресу користувачеві.

Таблиця 2.23 – Функціональна вимога FR-4

Назва	Перегляд інформації по користуванню ботом
Опис	Система повинна надавати можливість переглядати інформацію про службу доставки користувачу, а також інформацію по користуванню ботом.

Таблиця 2.24 – Функціональна вимога FR-5

Назва	Створення замовлення
Опис	Система повинна надавати можливість користувачеві створити замовлення, шляхом вибору бренду води, літражу та кількості.

Таблиця 2.25 – Функціональна вимога FR-6

Назва	Перегляд замовлення
Опис	Система повинна надавати можливість переглянути створене замовлення користувачеві.

Таблиця 2.26 – Функціональна вимога FR-7

Назва	Видалення замовлення
Опис	Система повинна надавати можливість видалити замовлення користувачеві.

Таблиця 2.27 – Функціональна вимога FR-8

Назва	Додавання іншого товару до кошика
Опис	Система повинна надавати можливість користувачеві додати інший товар до кошика.

Таблиця 2.28 – Функціональна вимога FR-9

Назва	Оформлення доставки
Опис	Система повинна надавати можливість користувачеві оформити доставку, шляхом вибору дати та часу.

Таблиця 2.29 – Функціональна вимога FR-10

Назва	Перегляд інформації про доставку
Опис	Система повинна надавати можливість користувачеві переглянути інформацію про створену доставку.

Таблиця 2.30 – Функціональна вимога FR-11

Назва	Перегляд активних доставок
Опис	Система повинна надавати можливість переглядати активні доставки адміністратору.

Таблиця 2.31 – Функціональна вимога FR-12

Назва	Пошук доставки за Id
Опис	Система повинна надавати можливість адміністратору пошуку доставки за Id.

Таблиця 2.32 – Функціональна вимога FR-13

Назва	Пошук користувача за номером телефону
Опис	Система повинна надавати можливість адміністратору пошуку користувача за номером телефону.

Таблиця 2.33 – Функціональна вимога FR-14

Назва	Пошук користувача за Id
Опис	Система повинна надавати можливість адміністратору пошуку користувача за Id.

Таблиця 2.34 – Функціональна вимога FR-15

Назва	Видалення доставки
Опис	Система повинна надавати можливість адміністратору видалити доставку за Id.

Таблиця 2.35 – Функціональна вимога FR-16

Назва	Підтвердження доставки
Опис	Система повинна надавати можливість адміністратору підтвердити доставку за Id.

Таблиця 2.36 – Функціональна вимога FR-17

Назва	Створення доставки
Опис	Система повинна надавати можливість адміністратору створити доставку за запитом користувача.

Таблиця 2.37 – Функціональна вимога FR-18

Назва	Облік продажів
Опис	Система повинна надавати можливість адміністратору переглядати виконані доставки, а також відображати інформацію про прибуток компанії.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18
UC-1	+																	
UC-2		+																
UC-3			+															
UC-4				+														
UC-5					+													
UC-6						+												
UC-7							+											
UC-8								+										
UC-9									+									
UC-10										+								
UC-11											+							
UC-12												+						
UC-13													+					
UC-14														+				
UC-15															+			
UC-16																+		
UC-17																	+	
UC-18																		+

Рисунок 2.3 – Матриця трасування вимог

2.4 Розроблення нефункціональних вимог

Основна мета – гарантувати користувачам повну прозорість і забезпечити їм повний контроль над власними даними. У веб-застосунку передбачено використання користувацьких аккаунтів та адміністраторських прав, тому треба гарантувати їх безпеку та конфіденційність.

Також, важливо, щоб СУБД могла впоратись із можливим великим навантаженням, оскільки веб-застосунок передбачає постійну взаємодію із БД, тому треба переконатися, що вона буде задовольняти потреби усіх користувачів.

Висновки до розділу

В даному розділі було описано варіанти використання телеграм боту, проведено аналіз системних вимог, розроблено функціональні та нефункціональні вимоги. Детально розглянуто мінімальні та рекомендовані конфігурації технічних засобів для забезпечення стабільної та ефективної роботи Telegram бота служби доставки води.

Окрім цього, було проаналізовано необхідність підтримки різних операційних систем, що дозволяє охопити максимальну аудиторію користувачів. Зокрема, програмне забезпечення повинно функціонувати на платформі Windows, MacOS, Unix, Android та iOS, що забезпечує універсальність та зручність для користувачів з різними технічними засобами.

Було визначено ключові функції вебзастосунку, такі як створення замовлень, оформлення доставки, управління даними клієнтів та ведення обліку продажів. Ці функціональні можливості були розроблені з метою підвищення ефективності роботи служби доставки води, а також покращення взаємодії з клієнтами.

Не менш важливим аспектом є нефункціональні вимоги, що включають безпеку, продуктивність, надійність та масштабованість системи. Важливість цих вимог обумовлена необхідністю забезпечити захист даних клієнтів, швидке оброблення запитів та стабільну роботу системи навіть при високих навантаженнях.

Таким чином, було надано всебічний огляд системних вимог та функціональних можливостей Telegram бота, що є основою для його подальшої розробки та впровадження. Описані вимоги та аналіз забезпечують фундамент для створення надійного, ефективного та зручного для користувачів програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Для даного вебзастосунку було обрано клієнт-серверну архітектуру. Клієнт-серверна архітектура – це модель взаємодії між комп'ютерами, в якій один пристрій (клієнт) надсилає запити на обробку даних або послуги до іншого пристрою (сервера), який відповідає на ці запити. Ця модель зазвичай використовується для розподіленої обробки даних та надання послуг через мережу, де клієнти можуть бути будь-якими пристроями (комп'ютерами, мобільними телефонами, планшетами тощо), що з'єднані з сервером через мережу. На рисунку 3.1 зображено зв'язок між клієнтом та сервером, реалізованим у самому боті.

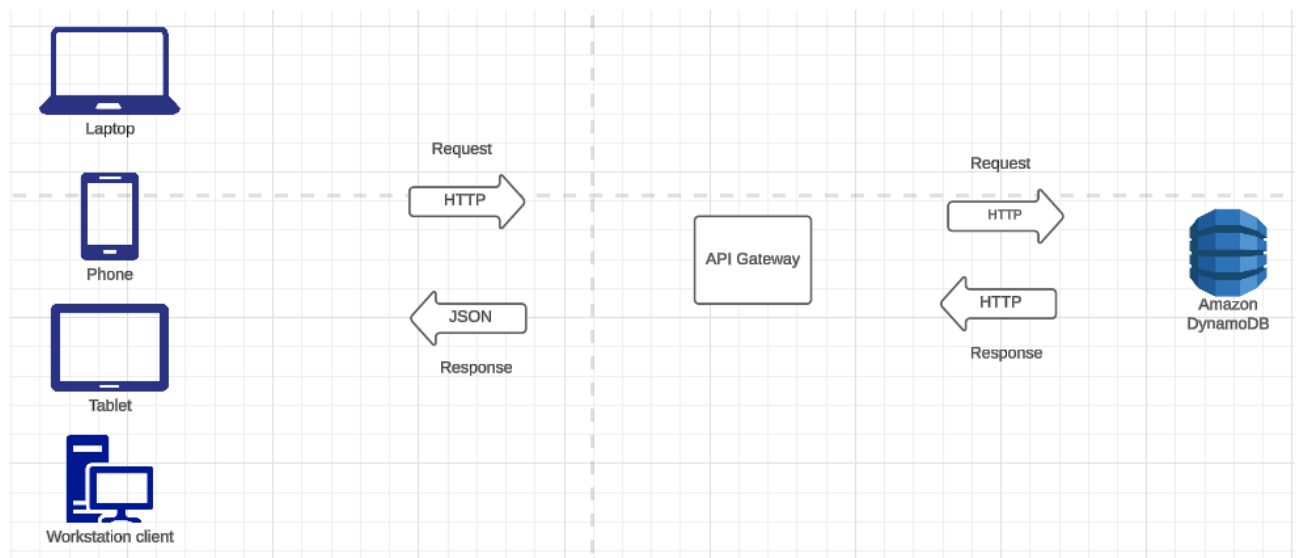


Рисунок 3.1 – Клієнт-серверна архітектура телеграм боту

Архітектура була побудована з використанням архітектурного патерну MVC (Model-View-Controller). Така архітектура забезпечує чітке розділення логіки додатку, що полегшує його розробку, підтримку та масштабування. Патерн MVC розділяє додаток на три основні компоненти:

- Model: Відповідає за управління даними та бізнес-логікою додатку.
- View: Відповідає за відображення даних користувачу.
- Controller: Обробляє вхідні запити, взаємодіє з моделлю для обробки даних та повертає відповідний вид для відображення.

На рисунку 3.2 нижче зображено патерн проектування, який використовувався для побудови архітектури.

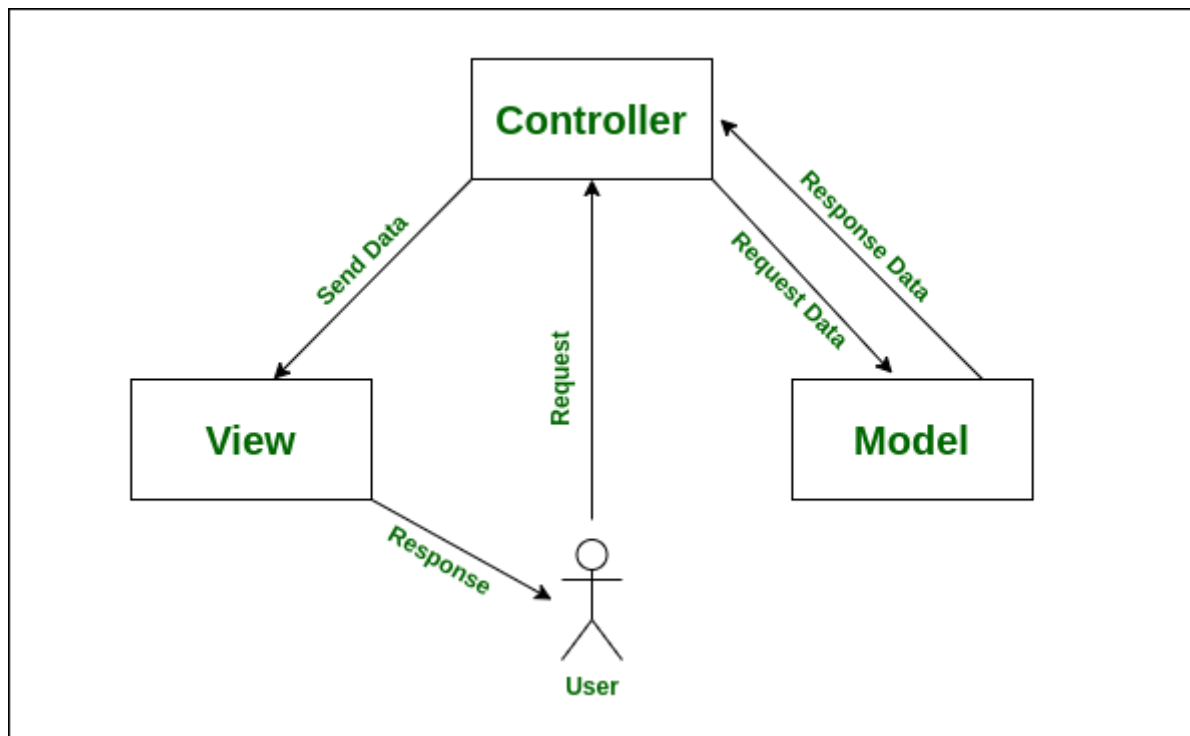


Рисунок 3.2 – MVC патерн

Модель (Model):

Відповідає за управління даними, логіку додатку та управління його станом. Модель обробляє запити, пов'язані з даними, такі як збереження, оновлення, видалення, а також здійснює валідацію та перетворення даних для подальшого використання. Цей компонент не залежить від інших частин додатку і може бути повторно використаний в інших частинах програми.

Вид (View):

Відповідає за відображення даних користувачеві та забезпечення інтерфейсу користувача для взаємодії з додатком. Він візуально відображає

інформацію користувачеві та надає засоби для взаємодії з додатком, такі як введення даних або натискання кнопок. Вид не має логіки взаємодії з даними і отримує необхідні дані з моделі для відображення користувачу.

Контролер (Controller):

Відповідає за прийом запитів від користувача, обробку цих запитів та відправку відповідей. Контролер взаємодіє з моделлю для отримання необхідних даних та з видом для передачі цих даних користувачеві. Він забезпечує взаємодію між моделлю та видом, реагуючи на події від користувача та виконуючи відповідні дії.

Розробку боту було виконано у стилі REST. Стиль REST означає використання підходу до взаємодії компонентів програмного забезпечення через мережу, що базується на узгодженому наборі обмежень, відомих як REST (Representational State Transfer). Цей підхід дозволяє спростити архітектуру розподіленого додатку та підвищити його продуктивність, що особливо корисно для інтернет-магазинів і пошукових систем.

У контексті REST, компоненти взаємодіють між собою, нагадуючи взаємодію клієнта і сервера в Інтернеті. Виклик віддаленої процедури, який може бути здійснений через мережу Інтернет, представляє собою звичайний HTTP-запит, такий як «GET» або «POST», із передачею необхідних даних у вигляді параметрів запиту[6].

Термін "RESTful" використовується для опису веб-служб, які були розроблені з дотриманням принципів REST. На відміну від веб-сервісів на основі SOAP, REST не має єдиного офіційного стандарту, оскільки це архітектурний стиль, а не протокол. Проте, більшість RESTful-реалізацій використовують відомі стандарти, такі як HTTP, URL, JSON і XML, для забезпечення взаємодії між компонентами системи[7].

3.2 Обґрунтування вибору засобів розробки

Вибір мови програмування

C# – проста, сучасна, гнучка та універсальна об'єктно-орієнтована мова програмування. Вона була розроблена у 2001 році компанією Microsoft. Застосовується розробниками для створення ПЗ для роботи у теперішніх і майбутніх реаліях завдяки своїй гнучкості[8].

Переваги *C#*:

- подібність синтаксису до *C++*, *Java*, *Basic*;
- об'єктно-орієнтований підхід до розробки проєктів;
- широка сфера застосування;
- адаптація під різні платформи;
- безпека, в порівнянні з *C* та *C++*;
- швидкість роботи;
- зручність при написанні коду;

Недоліки *C#*:

- відсутність багато потокової компіляція;
- базується під операційну систему Windows;

Python – це об'єктно-орієнтована мова програмування загального призначення високого рівня з інтегрованою динамічною структурою, спрямована насамперед на підвищення продуктивності самого програміста, ніж коду, який він пише. Часто використовується для підтримки ПЗ у всесвітній павутині та розробки різноманітних додатків.

Переваги *Python*:

- простота та швидкість у написанні коду;
- велика кількість бібліотек для спрощення розробки проєкту;
- зручний для роботи з великою кількістю даних;
- доступний у відкритому виді код;
- можливість розробки під різні платформи.

Недоліки *Python*:

- відсутність статичної типізації;
- відсутність створення приватних змінних (створення відбувається за допомогою додавання символу “_” перед змінною, наприклад: “_privat_var”);
- низька продуктивність.

В рамках дипломного проєкту для розробки чат-бота в месенджері Telegram, було обрано мову програмування C#, оскільки вона забезпечує високу продуктивність, зручність у розробці, безпеку та широкий спектр можливостей для масштабування і підтримки проєкту.

Visual Studio

Відповідно до обраної мови програмування, було обрано середовище розробки – Visual Studio.

Visual Studio є продуктом компанії Microsoft і має безліч інструментів та функцій, які значно спрощують процес розробки. Використовуючи Visual Studio, розробники можуть швидко створювати, налагоджувати та тестувати свої програми завдяки інтегрованим засобам розробки та потужному налагоджувачу[9].

Однією з головних переваг Visual Studio є можливість використання шаблонів, таких як Bot Application, Bot Framework Emulator, Bot Dialog і Bot Controller. Ці шаблони забезпечують готову структуру проєкту, що дозволяє зекономити час і зусилля, необхідні для написання великої кількості коду з нуля. Замість цього можна зосередитися на налаштуванні та адаптації цих шаблонів під конкретні вимоги свого проєкту.

Visual Studio очевидно підтримує мову програмування C#, яка була обрана для розробки чат-бота. Завдяки тісній інтеграції з .NET Framework, Visual Studio забезпечує високу продуктивність та зручність у написанні коду, дозволяючи використовувати всі переваги об'єктно-орієнтованого програмування[10].

Крім того, Visual Studio пропонує потужні засоби для управління версіями, інтеграцію з різними системами контролю версій, такими як Git, що забезпечує надійність та контроль над процесом розробки.

Таким чином, обрання Visual Studio для розробки чат-бота в рамках дипломного проекту є обґрунтованим і раціональним рішенням, що забезпечує високу ефективність, зручність і продуктивність роботи.

Вибір фреймворку

Для розробки веб-застосунку було обрано фреймворк ASP.NET Core. Цей вибір обумовлений кількома ключовими факторами, що роблять фреймворк ідеальним рішенням для створення високопродуктивного та масштабованого Telegram бота для служби доставки води.

Ключові переваги ASP.NET Core:

- кросплатформність (ASP.NET Core є кросплатформним фреймворком, що дозволяє розробляти та розгортати додатки на різних операційних системах, включаючи Windows, MacOS та Linux. Це забезпечує гнучкість у виборі серверного середовища та дозволяє використовувати існуючу інфраструктуру без обмежень на конкретну операційну систему);

- висока продуктивність (ASP.NET Core відомий своєю високою продуктивністю, здатністю обробляти велику кількість запитів з низькими затримками. Для Telegram бота, який повинен швидко реагувати на запити користувачів та забезпечувати безперебійну роботу, висока продуктивність є критичним фактором);

- масштабованість (фреймворк підтримує горизонтальну та вертикальну масштабованість, дозволяючи легко розширювати додаток у міру зростання навантаження. Масштабованість важлива для забезпечення безперебійної роботи сервісу при збільшенні кількості користувачів та обсягу даних);

- велика спільнота та підтримка (ASP.NET Core має велику спільноту розробників та потужну підтримку від Microsoft, включаючи регулярні оновлення та вдосконалення. У свою чергу наявність активної спільноти та підтримка забезпечують швидке вирішення проблем, доступ до численних ресурсів та інструментів, що полегшує розробку та усунення неполадок)[11].

Порівняльний аналіз інших фреймворків

При виборі ASP.NET Core також були розглянуті інші популярні фреймворки:

Node.js

Переваги: Швидке виконання завдань вводу/виводу, велика кількість доступних бібліотек через npm, популярний для реальних додатків.

Недоліки: Потребує більше часу для налаштування безпеки, особливо для захисту від поширених веб-атак.

Django (Python)

Переваги: Швидкий старт розробки, вбудовані адміністративні інтерфейси, велика кількість плагінів.

Недоліки: Менша продуктивність у порівнянні з ASP.NET Core, може бути складним у налаштуванні для високонавантажених систем.

На основі порівняльного аналізу, ASP.NET Core було обрано як фреймворк, що забезпечує найкращий баланс між продуктивністю, безпекою, масштабованістю та зручністю розробки.

Вибір бази даних

Для зберігання даних у проекті було обрано базу даних DynamoDB від Amazon Web Services (AWS). Цей вибір обґрунтований численними перевагами, які забезпечують ефективне управління даними, високу продуктивність та масштабованість. Нижче наведено основні переваги DynamoDB у порівнянні з іншими базами даних.

- повністю керована база даних (DynamoDB є повністю керованим сервісом, що означає, що AWS відповідає за всі аспекти управління базою даних, включаючи налаштування, конфігурацію, масштабування, резервне копіювання та відновлення);

- автоматичне масштабування (DynamoDB автоматично масштабується для обробки будь-якого обсягу трафіку без необхідності ручного втручання. Це забезпечує постійну продуктивність незалежно від зростання кількості запитів

або даних, що є критичним для Telegram бота з непередбачуваним навантаженням);

- висока продуктивність (DynamoDB забезпечує низьку затримку обробки запитів (милісекунди), що робить її ідеальною для додатків реального часу. Висока швидкість відповіді бази даних гарантує швидке оброблення запитів користувачів Telegram бота, забезпечуючи безперебійну роботу сервісу);

- підтримка різних моделей даних (DynamoDB підтримує документи і ключ-значення моделі даних, що робить її гнучкою для різних типів додатків. Ця гнучкість дозволяє легко моделювати дані для різних вимог Telegram бота, включаючи зберігання інформації про користувачів, замовлення та інші дані);

- інтеграція з іншими сервісами AWS (DynamoDB тісно інтегрується з іншими сервісами AWS, такими як AWS Lambda, Amazon S3, та Amazon Kinesis. Це дозволяє створювати складні серверні додатки з мінімальними зусиллями, використовуючи інші сервіси AWS для обробки подій, зберігання файлів та аналізу даних);

- вбудоване резервне копіювання та відновлення (DynamoDB забезпечує автоматичне резервне копіювання та відновлення даних, що мінімізує ризик втрати даних).

MySQL / PostgreSQL (реляційні бази даних)

Переваги: Підтримка складних запитів SQL, транзакції ACID.

Недоліки: Менша продуктивність і масштабованість при обробці великих обсягів даних і високого навантаження, необхідність ручного масштабування.

MongoDB (NoSQL документна база даних)

Переваги: Гнучка схема, хороша продуктивність для зберігання документів.

Недоліки: Відсутність вбудованого автоматичного масштабування та управління, потреба в додаткових зусиллях для забезпечення відмовостійкості.

DynamoDB було обрано як основну базу даних для Telegram бота служби доставки води завдяки її повній керованості, високій продуктивності,

автоматичному масштабуванню, відмовостійкості та гнучкості. Інтеграція з іншими сервісами AWS, підтримка різних моделей даних та вбудовані засоби для забезпечення безпеки та резервного копіювання роблять DynamoDB оптимальним вибором для забезпечення надійної та ефективної роботи бота.

3.3 Конструювання програмного забезпечення

Використання Telegram Bot API надає розробникам широкі можливості для створення інтерактивних чат-ботів, які ефективно взаємодіють з користувачами, обробляють їх запити та виконують різноманітні завдання. Один з методів реалізації цього – опитування (polling), що дозволяє ботам регулярно отримувати оновлення від серверів Telegram, обробляти їх у реальному часі та забезпечувати безперервну роботу без необхідності в додаткових серверах для обробки вебхуків. Це особливо корисно на початкових етапах розробки та тестування бота.

Telegram Bot API підтримує використання стандартних методів HTTP, таких як GET та POST, що спрощує інтеграцію бота з іншими веб-сервісами та API. Це забезпечує гнучкість та масштабованість рішення, що дозволяє використовувати ботів у різних сценаріях, від простих чатів до складних бізнес-процесів.

Окрім того, Telegram Bot API дозволяє відправляти та обробляти різноманітні типи повідомлень, включаючи текст, фотографії, документи та інші медіафайли. Це дає можливість створювати багатофункціональні боти, які можуть відповідати на запити користувачів з використанням різних форматів даних.

Також Telegram Bot API має інтерактивні можливості, такі як кнопки `InlineKeyboardButton` та `ReplyKeyboardMarkup`, що дозволяють створювати інтерактивні повідомлення. Це дозволяє створювати зручні та інтуїтивно зрозумілі інтерфейси користувача, що покращує ефективність взаємодії з ботом. Додатково, є можливість обробки команд, які починаються зі слешу, що забезпечує зручність для користувачів при взаємодії з ботом.

Загалом, Telegram Bot API надає потужний інструментарій для створення сучасних та функціональних чат-ботів. Його простота використання, широкі можливості налаштування та інтерактивні функції роблять його відмінним вибором для покращення бізнесу та особистих потреб.

Як було зазначено раніше, в якості бази даних було обрано нереляційну базу даних від Amazon DynamoDB.

База даних серверу призначена для зберігання користувачів, даних про їх них та їх замовлення, зв'язки між ними та їх статуси у кожного з користувачів, а також для ведення обліку продажів. Опис таблиць бази даних наведено у таблицях 3.1-3.4. Модель бази даних наведена на рисунку 3.3.

Таблиця 3.1 – Опис таблиці wd-db

Таблиця	Назва поля	Тип даних	Опис
wd-db	User_Id	string	Ідентифікаційний номер користувача
	Address	string	Адреса доставки
	Checker	string	Перевірка на активне замовлення у користувача
	City	string	Місто доставки
	Phone	string	Номер телефону користувача

Таблиця 3.2 – Опис таблиці temp-db

Таблиця	Назва поля	Тип даних	Опис
temp-db	User_Id	string	Ідентифікаційний номер користувача
	Num_Water	string	Кількість бутлів води
	Volume	string	Літраж води
	Water_Brand	string	Бренд води

Таблиця 3.3 – Опис таблиці od-db

Таблиця	Назва поля	Тип даних	Опис
od-db	Order_Id	string	Ідентифікаційний номер замовлення
	Num_Water	string	Кількість бутлів води
	Price	string	Ціна
	Volume	string	Літраж води
	Water_Brand	string	Бренд води
	User_Id	string	Ідентифікаційний номер користувача

Таблиця 3.4 – Опис таблиці asso-db

Таблиця	Назва поля	Тип даних	Опис
asso-db	Order_Id	string	Ідентифікаційний номер замовлення
	Summary	string	Опис всього замовлення
	Price	string	Ціна
	Date_Acco	string	Дата доставки
	Time_Acco	string	Час доставки
	User_Id	string	Ідентифікаційний номер користувача
	City	string	Місто доставки
	Address	string	Адреса доставки
	Phone	string	Номер телефону користувача

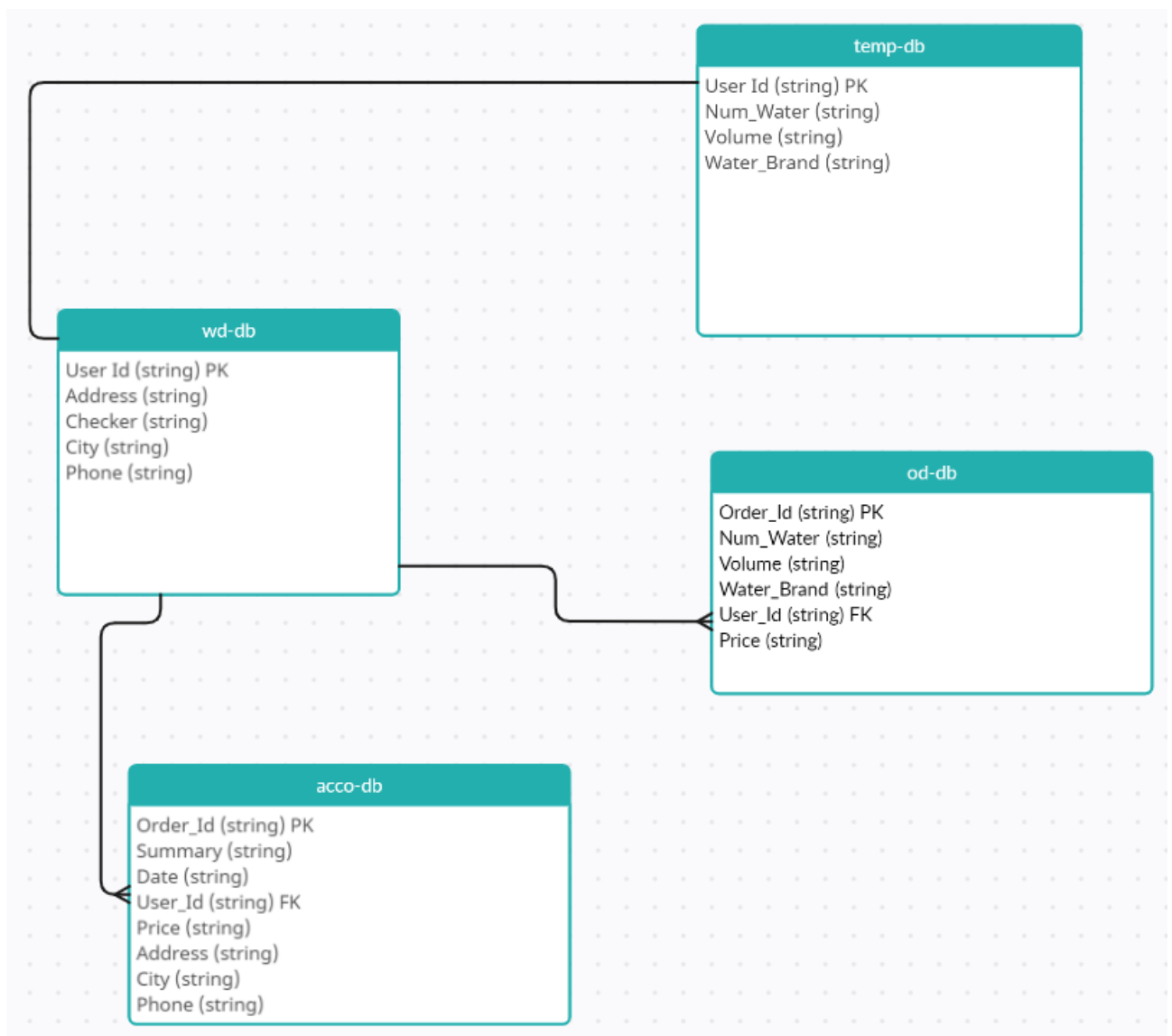


Рисунок 3.3 – Модель бази даних

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.5.

Таблиця 3.5 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Visual Studio	Головне середовище розробки програмного забезпечення серверної частини проєктної роботи.
2	Postman	Програмне забезпечення необхідне для тестування rest запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів.
3	Lucidchart	Веб-додаток для створення діаграм, який дозволяє візуально співпрацювати над малювання, переглядом і обміном діаграмами. Покращує процеси системи та організаційна структури.
4	Prometheus	Клієнтські бібліотеки для отримання метрик якості ПЗ на мові програмування C#.
5	AWSSDK	Бібліотека, яка забезпечує зручний інтерфейс для взаємодії з Amazon DynamoDB – швидкою і гнучкою базою даних NoSQL, яка забезпечує однозначну затримку при будь-якому масштабуванні.
6	Swashbuckle.AspNetCore	Бібліотека, яка дозволяє інтегрувати Swagger в ASP.NET Core проєкти для автоматичної генерації документації до API.

Продовження таблиці 3.5

7	Newtonsoft.Json	Популярна бібліотека для роботи з JSON в .NET. Забезпечує високопродуктивну серіалізацію і десеріалізацію JSON даних.
8	Visual Paradigm	Інструмент для моделювання, що надає засоби для розробки та візуалізації програмного забезпечення, бізнес-процесів і систем.
9	Telegram	Сервіс миттєвих повідомлень, який дозволяє користувачам обмінюватися текстовими повідомленнями, медіафайлами, документами, здійснювати голосові та відеодзвінки, а також користуватися ботами як в цілях ведення бізнесу, так і в цілях отримання сервісу.

3.4 Аналіз безпеки даних

Функції безпеки в ASP.NET Core

ASP.NET Core надає безліч засобів і бібліотек для захисту програм, включаючи вбудовані постачальники посвідчень та сторонні посвідчення, такі як Facebook, LinkedIn або Twitter. ASP.NET Core надає кілька підходів до зберігання секретів додатків.

Безпека в Amazon DynamoDB

Хмарна безпека в AWS є найвищим пріоритетом. Безпека хмари – AWS відповідає за захист інфраструктури, яка запускає служби AWS у хмарі AWS. AWS також надає послуги, якими можна безпечно користуватися. Ефективність безпеки регулярно тестується та перевіряється сторонніми аудиторями в рамках програм відповідності AWS.

Amazon DynamoDB забезпечує надійну інфраструктуру зберігання, призначену для зберігання критично важливих і первинних даних. Дані з

резервуванням зберігаються на кількох пристроях у кількох об'єктах у регіоні Amazon DynamoDB.

DynamoDB захищає дані користувача, які зберігаються в стані спокою, а також дані, що передаються між локальними клієнтами та DynamoDB, а також між DynamoDB та іншими ресурсами AWS у тому самому регіоні AWS.

Висновки до розділу

У даному розділі було всебічно розглянуто ключові аспекти розробки програмного забезпечення для телеграм бота служби доставки води. Детальний аналіз охоплював кілька важливих підрозділів, що допомогло сформувати цілісне уявлення про процес створення, архітектурні рішення, вибір інструментів розробки, реалізацію програмного забезпечення та забезпечення безпеки даних.

Обґрунтовано вибір мови програмування, середовище розробки, фреймворки, а також описано архітектурний патерн, обраний для розробки бота. Основою архітектури є клієнт-серверна модель, яка забезпечує ефективну взаємодію між клієнтською частиною (користувачами Telegram) та серверною частиною (API і базою даних).

Розглянуто питання безпеки даних, що є критично важливим для будь-якої сучасної системи. Визначено, що Amazon DynamoDB забезпечує надійну інфраструктуру зберігання, призначену для зберігання важливих і первинних даних. Проведений аналіз та опис допомагають зрозуміти всі аспекти розробки програмного забезпечення, що забезпечує його ефективність, надійність та безпеку у використанні.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Для отримання метрик якості ПЗ використовувалася вбудована функція середовища розробки. Для цього потрібно на панелі меню вибрати «Аналіз», «Обчислити показники коду», «Для рішення». Метрики, за котрими буде оцінюватись ПЗ будуть наступні:

Індекс супроводжуваності є мірою, яка оцінює відносну легкість підтримки коду та поділяється на діапазон від 0 до 100. Високе значення індексу вказує на кращу ремонтпридатність. Цей індекс можна представити в різних кольорах для швидкого визначення проблемних місць у коді: зелений рейтинг від 20 до 100 свідчить про добру підтримку коду, жовтий рейтинг від 10 до 19 - про помірну підтримку, а червоний рейтинг від 0 до 9 - про низьку ремонтпридатність.

Цикломатична складність вимірює структурну складність коду, обчислюючи кількість різних шляхів в потоці програми. Програма з вищою складністю керування потоком потребує більше тестів для забезпечення хорошого покриття коду та є менш зручною для обслуговування.

Глибина успадкування вказує на кількість класів, які успадковують один від одного, аж до базового класу. Це схоже на зчеплення класів, оскільки зміна в базовому класі може вплинути на всі його успадковані класи. Низьке значення глибини успадкування вважається бажаним, тоді як високе значення може призвести до складних модифікацій базового класу.

Зв'язок класів – це міра зв'язку з унікальними класами через параметри, локальні змінні, типи повернення, виклики методів та інші фактори. Хороший дизайн програмного забезпечення передбачає високу когезію та низький зв'язок між класами. Великий зв'язок може свідчити про складний проект, що важко підтримувати та повторно використовувати.

Рядки вихідного коду вказують на кількість рядків у вихідному файлі, включаючи порожні рядки.

Рядки виконуваного коду показують приблизну кількість рядків або операцій у виконуваному коді. Це може бути корисною мірою складності програми та її роботи.

Аналіз загальних метрик Web Api та телеграм боту можна побачити на рисунках 4.1-4.2. відповідно.

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
WD_API (Debug)	91	260	2	102	1,486	314
WD_API	84	8	1	42	89	26
WD_API.Clients	84	73	1	31	717	118
WD_API.Controllers	68	46	2	39	484	160
WD_API.Extensions	61	5	1	6	29	10
WD_API.Model	100	128	1	0	167	0

Рисунок 4.1 – Аналіз загальних метрик Web Api

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
WDelivery_Bot (Debug)	84	295	1	59	3,195	975
WDelivery_Bot	73	7	1	21	61	19
WDelivery_Bot.Commands	11	175	1	50	2,755	941
WDelivery_Bot.Keyboards	81	9	1	8	236	15
WDelivery_Bot.Model	100	104	1	0	143	0

Рисунок 4.2 – Аналіз загальних метрик Telegram боту

Web Api та Телеграм бот покриває усі вимоги, які були описані в технічному завданні, тому можна визначити, що він є функціонально придатним, зручним для внесення змін та перевірки, адже присутні задачі, що відповідають за зручність супроводу. Також він відповідає характеристикам продуктивності, адже час завантаження доволі малий та здатний для вирішення задач з використанням певних об'ємів ресурсів конкретних видів. Бот відповідає умові практичності через зрозумілість для користування.

4.2 Опис процесів тестування

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 – 4.14.

Таблиця 4.1 – Тест 1.1 Створення замовлення незареєстрованим користувачем

Початковий стан системи	Користувач не зареєструвався в системі та знаходиться у головному меню
Вхідні данні	-
Опис проведення тесту	Користувач натискає на панелі меню кнопку «Замовити».
Очікуваний результат	З'являється повідомлення про те, що користувач не додав усі дані та його перенаправляє до меню налаштувань.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Створення замовлення зареєстрованим користувачем

Початковий стан системи	Користувач зареєструвався в системі та знаходиться у головному меню
Вхідні данні	Вибір бренду води, літражу та кількості води
Опис проведення тесту	Користувач натискає на панелі меню кнопку «Замовити», після чого з'являється панель з вибором бренду, літражу та кількості води, які користувач обирає.
Очікуваний результат	Товар додається до кошика та користувачу видається повідомлення, про успішно доданий товар до кошика, а також перенаправляє до меню кошика.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.3 Оформлення доставки користувачем без доданих товарів до кошика

Початковий стан системи	Користувач не додав товарів до кошику та знаходиться у головному меню
Вхідні данні	-
Опис проведення тесту	Користувач натискає у головному меню кнопку «Кошик».
Очікуваний результат	З'являється повідомлення про те, що користувач не додав ніяких товарів і кошик пустий та його перенаправляє до головного меню.
Фактичний результат	Збігається з очікуванням.

Таблиця 4.4 – Тест 1.4 Оформлення доставки користувачем із доданими товарами до кошику

Початковий стан системи	Користувач додав товари до кошику та знаходиться у головному меню
Вхідні данні	Перегляд замовлення, вибір дати та часу доставки
Опис проведення тесту	Користувач натискає у головному меню кнопку «Кошик», переглядає додані товари, після чого у меню кошика обирає кнопку «Оформити доставку», з'являється панель із вибором дати та часу. Користувач обирає бажану дату та час.
Очікуваний результат	З'являється повідомлення про успішне оформлення доставки та користувача перенаправляє до головного меню.
Фактичний результат	Збігається з очікуванням.

Таблиця 4.5 – Тест 1.5 Перегляд не існуючої доставки

Початковий стан системи	Користувач не оформлював доставки на вказану дату та знаходиться у головному меню
Вхідні данні	Дата створеної доставки
Опис проведення тесту	Користувач натискає у головному меню кнопку «Створені доставки», після чого вводить дату, для перегляду оформленої доставки.
Очікуваний результат	З'являється відповідне повідомлення про те, що за наданим запитом доставки не знайдено.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.6 Перегляд існуючої доставки

Початковий стан системи	Користувач оформив доставку на вказану дату та знаходиться у головному меню
Вхідні данні	Дата створеної доставки
Опис проведення тесту	Користувач натискає у головному меню кнопку «Створені доставки», після чого вводить дату, для перегляду оформленої доставки.
Очікуваний результат	З'являється повідомлення про доставку: номер доставки, перелік товарів, обрані дата та час доставки, адреса доставки, місто, номер телефону для зв'язку з користувачем, а також сума до сплати.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 1.7 Підтвердження доставки адміністратором, із некоректно введеним Id користувача

Початковий стан системи	Користувач оформив доставку, адміністратор знаходиться в головному меню управління
Вхідні данні	Id користувача
Опис проведення тесту	Адміністратор в головному меню натискає на кнопку «Підтвердити доставку», після чого вводить потрібний Id користувача.
Очікуваний результат	З'являється відповідне повідомлення про те, що користувача з даним Id не існує та адміністратора перенаправляє до головного меню.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 1.8 Підтвердження доставки адміністратором, із правильно введеним Id користувача

Початковий стан системи	Користувач оформив доставку, адміністратор знаходиться в головному меню управління
Вхідні данні	Id користувача
Опис проведення тесту	Адміністратор в головному меню натискає на кнопку «Підтвердити доставку», після чого вводить потрібний Id користувача.
Очікуваний результат	З'являється відповідне повідомлення про те, що доставка була успішно підтверджена та адміністратора перенаправляє до головного меню.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.9 Перегляд активних доставок

Початковий стан системи	В системі є створені доставки, адміністратор знаходиться в головному меню управління
Вхідні данні	Дата оформлених доставок
Опис проведення тесту	Адміністратор в головному меню натискає на кнопку «Активні доставки», після чого вводить потрібний дату.
Очікуваний результат	З'являється відповідне повідомлення з переліком всіх доставок на вказану дату. Надається інформація про номер доставки, перелік товарів, дату та час доставки, адресу, місто, номер телефону для зв'язку з користувачем, сума до сплати, а також Id користувача. Адміністратора перенаправляє до головного меню.
Фактичний результат	Збігається з очікуванням.

Таблиця 4.10 – Тест 1.10 Пошук не існуючої доставки

Початковий стан системи	Адміністратор знаходиться в головному меню управління
Вхідні данні	Номер доставки
Опис проведення тесту	Адміністратор в головному меню натискає на кнопку «Пошук доставки за Id», після чого вводить потрібний Id.
Очікуваний результат	З'являється відповідне повідомлення, що доставки з наданим номером в системі не існує, а також адміністратора перенаправляє до головного меню.
Фактичний результат	Збігається з очікуванням.

Таблиця 4.11 – Тест 1.11 Пошук існуючої доставки

Початковий стан системи	Створена доставка в системі, адміністратор знаходиться в головному меню управління
Вхідні данні	Номер доставки
Опис проведення тесту	Адміністратор в головному меню натискає на кнопку «Пошук доставки за Id», після чого вводить потрібний Id.
Очікуваний результат	Надається інформація про номер доставки, перелік товарів, дату та час доставки, адресу, місто, номер телефону для зв'язку з користувачем, сума до сплати, а також Id користувача. Адміністратора перенаправляє до головного меню
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест 1.12 Створення доставки адміністратором

Початковий стан системи	Адміністратор знаходиться в головному меню управління
Вхідні данні	Місто, адреса, номер телефону користувача, вибір бренду води, літражу та кількості води, дати та часу доставки

Продовження таблиці 4.12

Опис проведення тесту	Адміністратор в головному меню натискає на кнопку «Створити доставку», після чого його перекидає в меню для оформлення замовлень. Після цього адміністратор натискає на кнопку «Налаштування» та вводить інформацію про місто, адресу та номер телефону користувача. Далі повертається до головного меню, натискає на кнопку «Замовити» та обирає товари за запитом. Далі переходить до корзини, натискає на кнопку «Оформити доставку» та обирає бажану дату та час для доставки.
Очікуваний результат	З'являється відповідне повідомлення, що доставка була успішно створена, а також адміністратора перенаправляє до головного меню створення доставок.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13 – Тест 1.13 Видалення доставки

Початковий стан системи	Створена доставка в системі, адміністратор знаходиться в головному меню управління
Вхідні данні	Номер доставки
Опис проведення тесту	Адміністратор в головному меню натискає на кнопку «Видалити доставку», після чого вводить потрібний номер самої доставки, яку бажає видалити з системи.
Очікуваний результат	З'являється відповідне повідомлення, що доставка була успішно видалена з системи, а також адміністратора перенаправляє до головного меню управління.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.14 – Тест 1.14 Облік продажів

Початковий стан системи	Створені доставки в системі, адміністратор знаходиться в головному меню управління
Вхідні данні	Місяць та рік для обліку
Опис проведення тесту	Адміністратор в головному меню натискає на кнопку «Облік продажів», після чого вводить потрібний місяць та рік, за який бажає отримати інформацію.
Очікуваний результат	Надається інформація про усі створені доставки за вказаний місяць, конкретна дата виконання доставки, сума до сплати. Окремим повідомленням виводиться кількість виконаних доставок за місяць, прибуток компанії та період, за який виконано підрахунки.
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

Початково користувач натискає на кнопку «start». Для створення замовлення користувач має бути зареєстрований в системі. Натискаємо на кнопку «Налаштування» обираємо місто доставки, вводимо адресу та надаємо номер телефону. На рисунку 4.3 відображено реєстрацію користувача в системі.

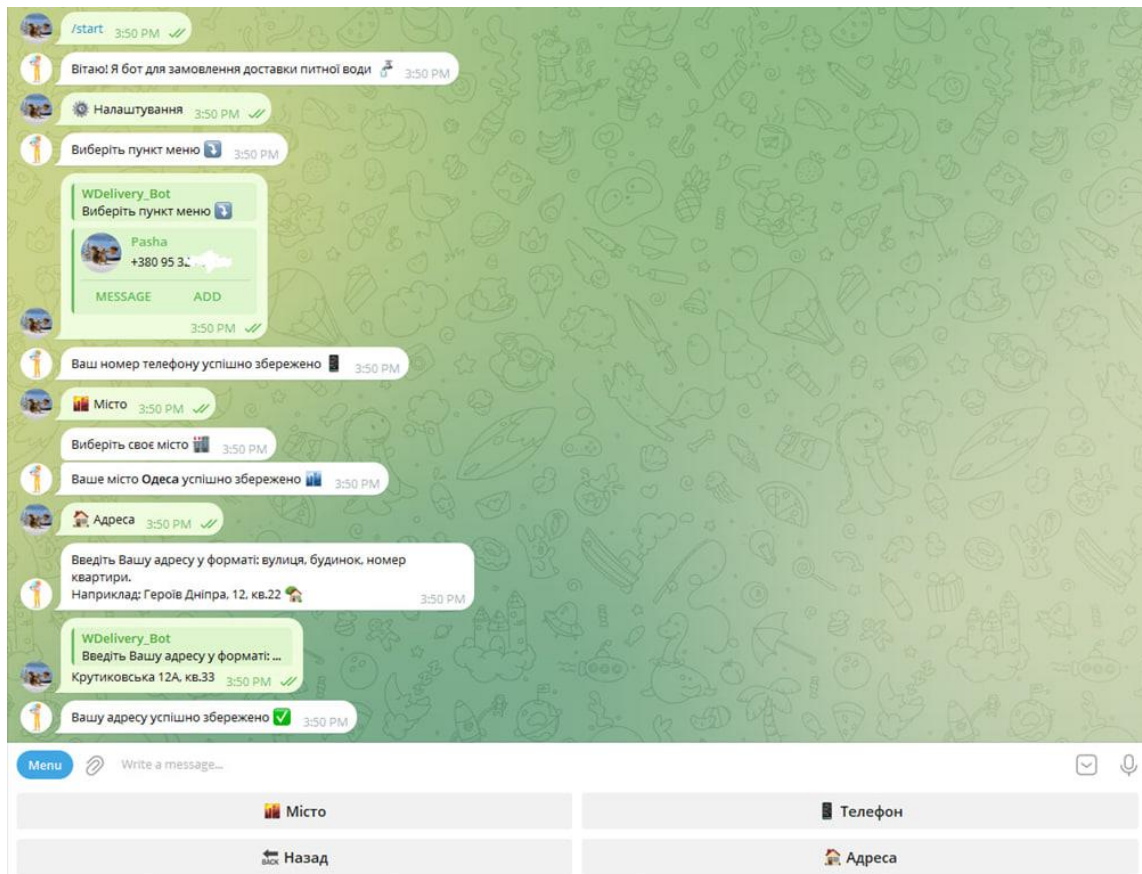


Рисунок 4.3 – Реєстрація користувача

Було обрано місто Одесу та надано адресу Крутиківська 12А, кв.33.

Для перегляду інформації про створення доставки, оновлення даних, обмін тар, оплату та саму службу доставки, користувач натискає на кнопку «FAQ». Після чого користувачу надається можливість вибрати питання, яке його цікавить та отримати відповідь на них. На рисунку 4.4 відображено перегляд самої інформації.

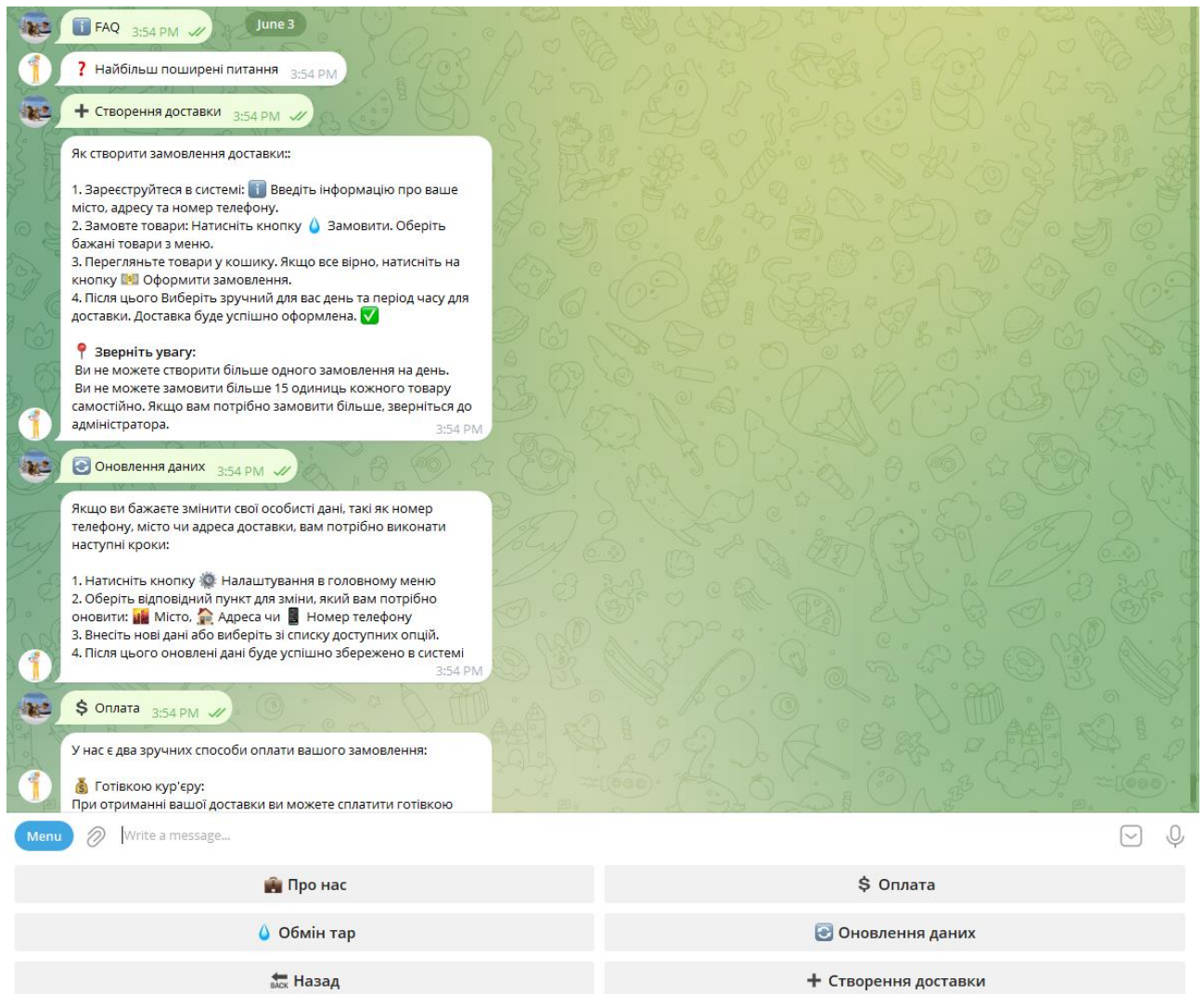


Рисунок 4.4 – Перегляд відповідей на найбільш поширені запитання

Після перегляду інформації користувач повертається до головного меню та натискає на кнопку «Замовити». Обирає бажаний товар зі списку доступних, літраж та вводить кількість товару на замовлення. Для користувача встановлено обмеження на максимальну кількість одиниць кожного товару 15 штук. Для прикладу було додано до замовлення 5 помп на 20 літрів кожна. На рисунку 4.5 відображено додавання помпи до кошику.

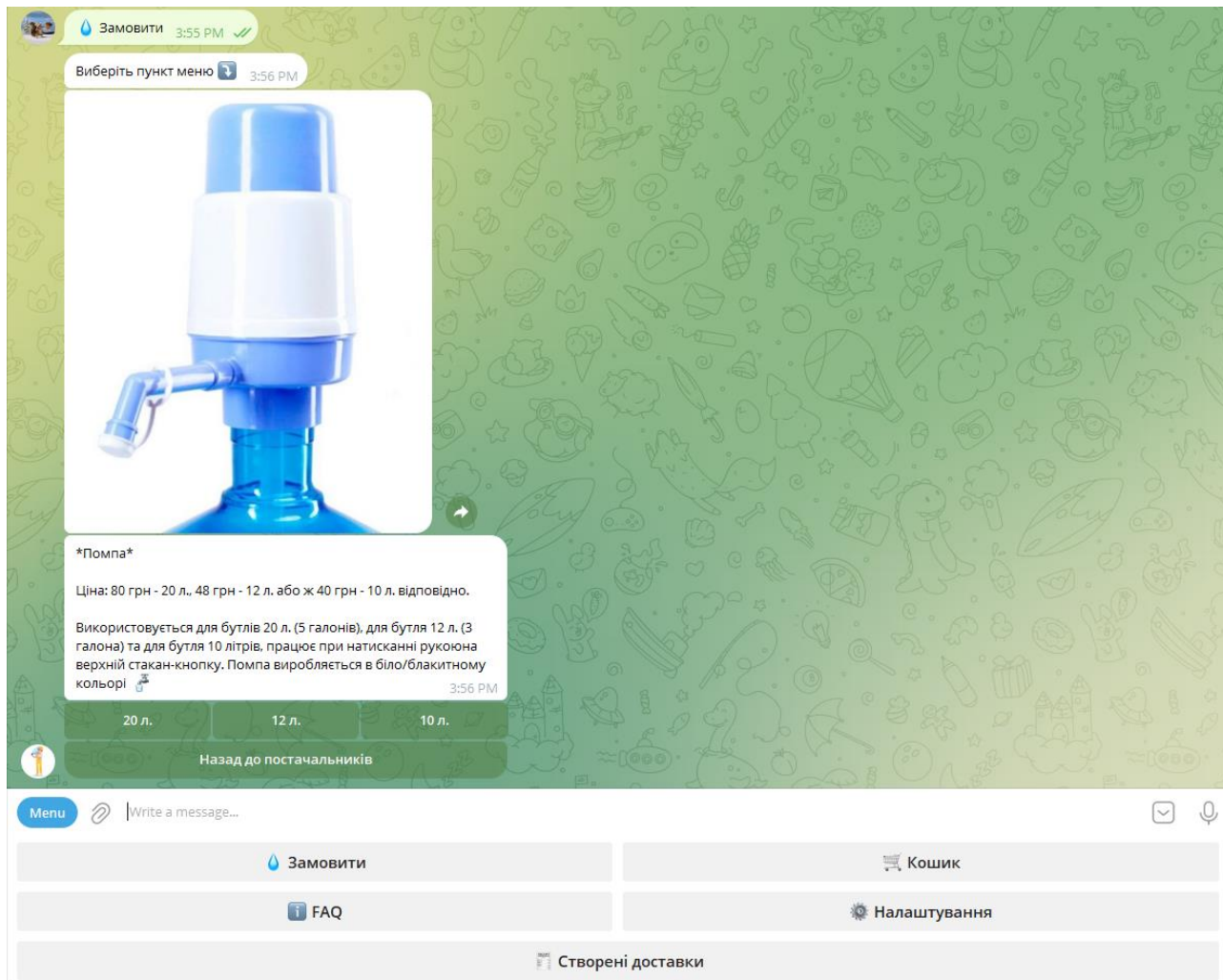


Рисунок 4.5 – Додавання помпи в кошик

Після додавання товару в кошик, користувачу видається меню з опцією додати інші товари. Користувач натискає на кнопку «Додати інший товар», обирає бажаний товар зі списку доступних, літраж та вводить кількість товару на замовлення.. В даному прикладі було додано до замовлення воду, кількістю 12 штук та літражем 20 літрів. На рисунку 4.6 відображено додавання іншого товару до кошику.

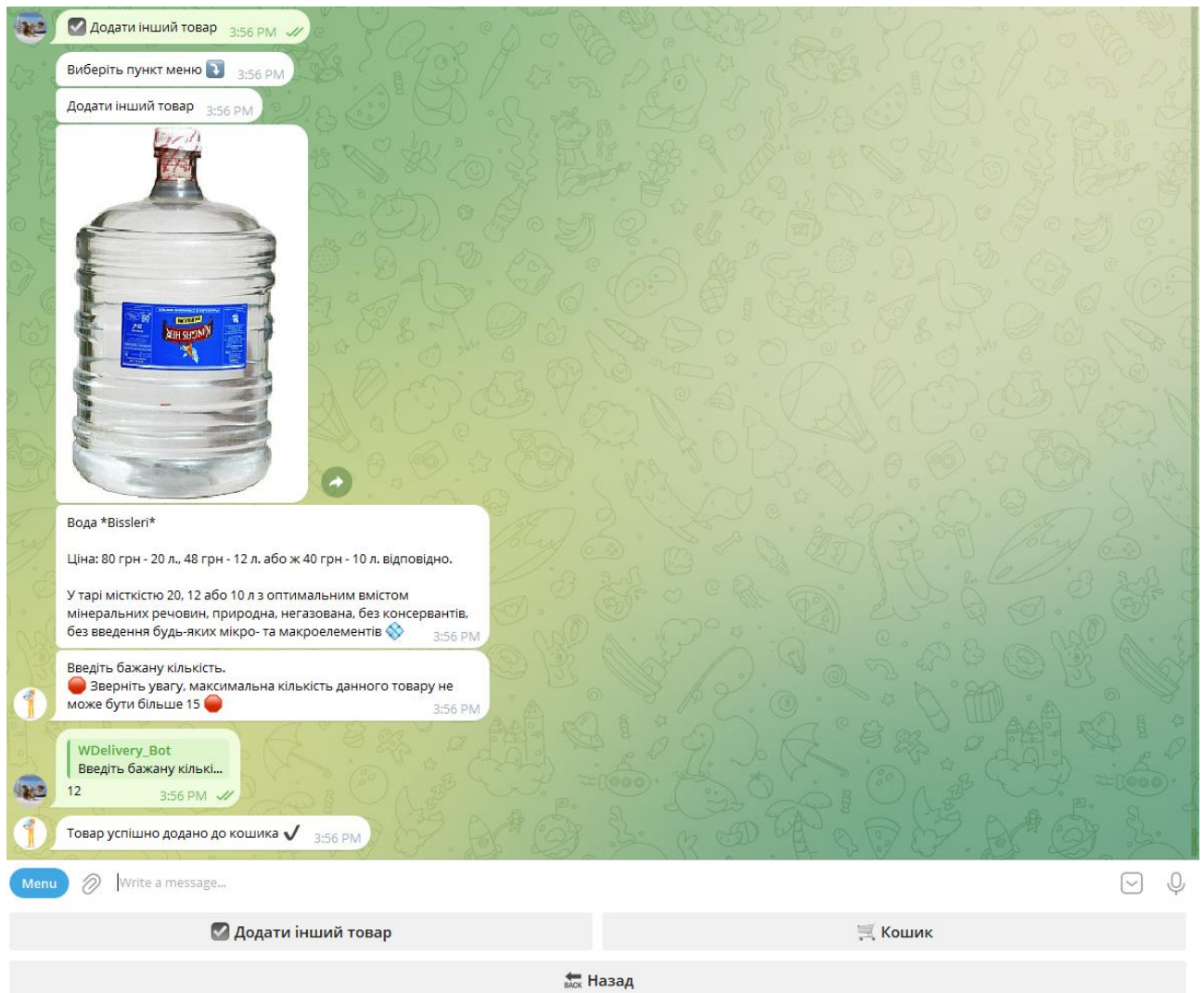


Рисунок 4.6 – Додавання іншого товару

Для перегляду товарів в кошику користувач в головному меню натискає на кнопку «Кошик». Після цього на екрані з'являється інформація з переліком товарів у кошику, їх кількістю, ціною за кожний товар та остаточну суму. На рисунку 4.7 відображено перегляд інформації про товари в кошику.

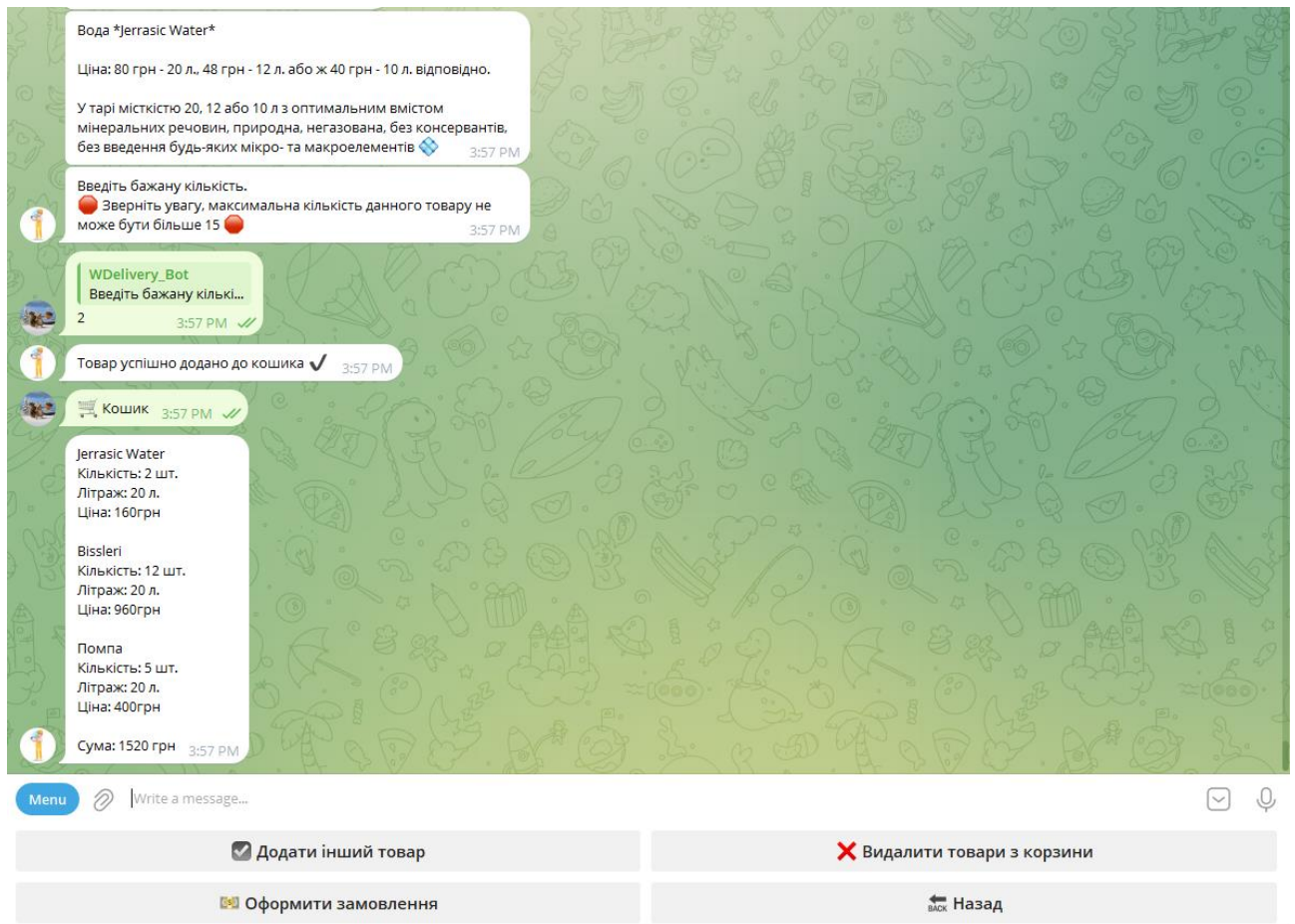


Рисунок 4.7 – Перегляд кошика

Із самого меню кошика користувач може оформити доставку. Для цього потрібно натиснути на кнопку «Оформити замовлення», після чого на екрані з’явиться повідомлення з вибором дати та часу відповідно. На рисунках 4.8-4.9 відображено оформлення доставки користувачем. Для прикладу було обрано дату доставки 6/4/2024, а період доставки з 12:00 до 15:00 відповідно.

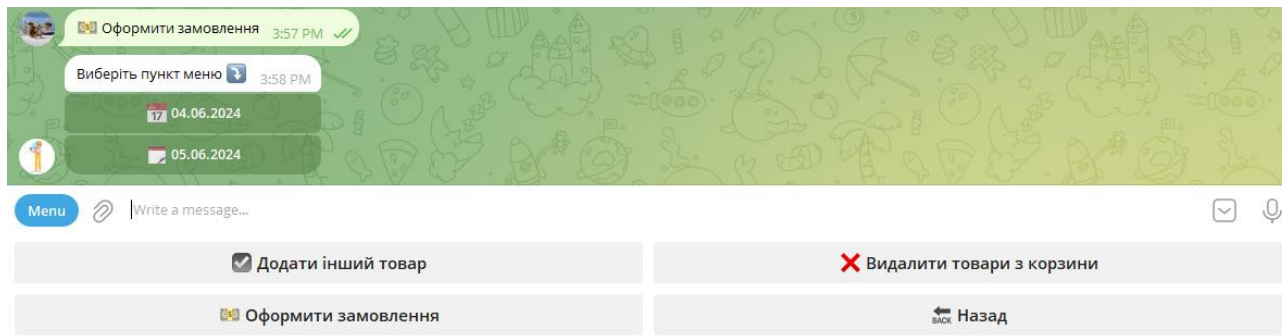


Рисунок 4.8 – Вибір дати доставки

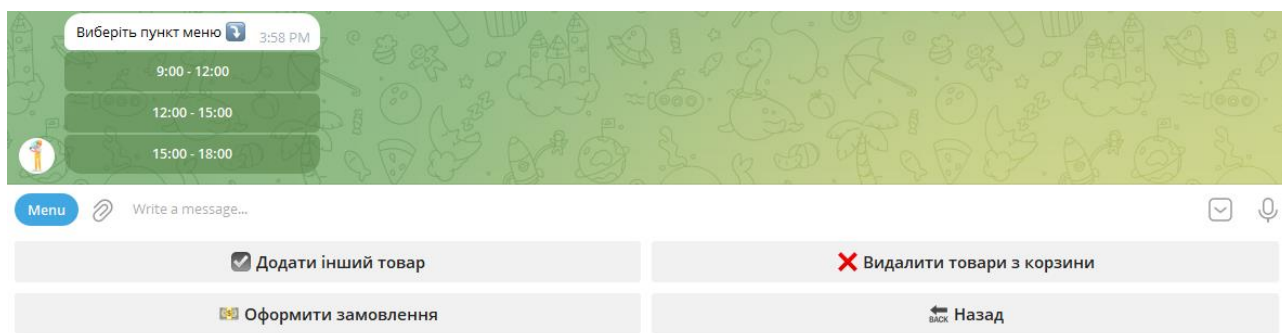


Рисунок 4.9 – Вибір часу доставки

Після оформлення доставки користувач перенаправляється в головне меню. Для перегляду інформації про оформлені доставки натискає на кнопку «Створені доставки». Вводить потрібну дату, на яку було створено замовлення та отримує відповідний результат. В інформації про доставку користувачу надається номер доставки, перелік замовлених товарів, дата доставки, час, адреса та місто доставки, номер телефону користувача і сума до сплати. На рисунку 4.10 відображено перегляд інформації про доставки. У прикладі переглядається доставка, оформлена вище, тобто на 6/4/2024.

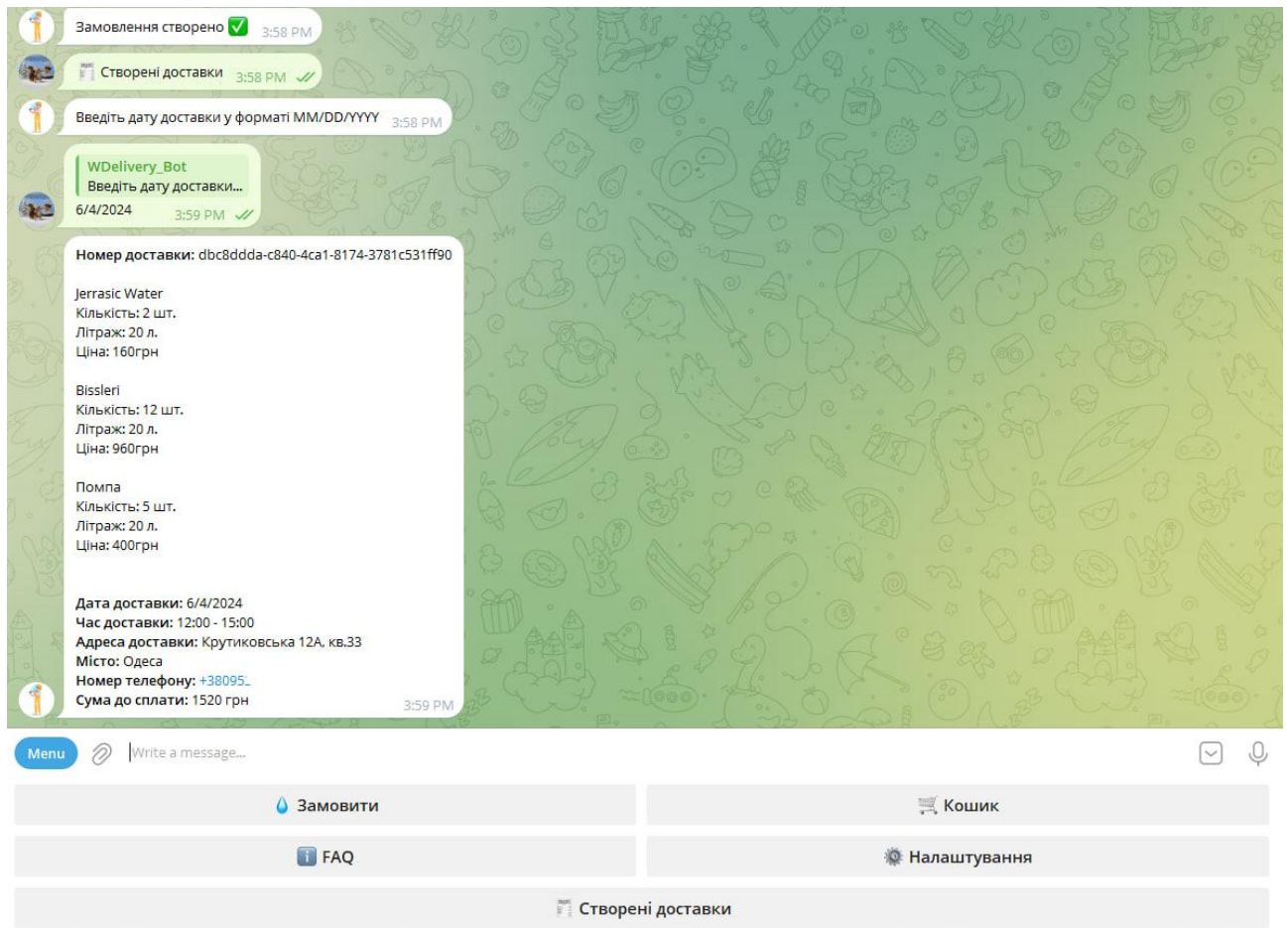


Рисунок 4.10 – Перегляд створеної доставки користувачем

Панель адміністратора значно відрізняється від панелі користувача. Адміністратор переглядає активні доставки за допомогою натискання кнопки «Активні доставки» та введення бажаної дати. В інформації про доставку адміністратору надається номер доставки, перелік замовлених товарів, дата доставки, час, адреса та місто доставки, номер телефону користувача, сума до сплати, а також Id користувача. Для прикладу було введено 6/3/2024. На рисунку 4.11 відображено перегляд активних доставок адміністратором.

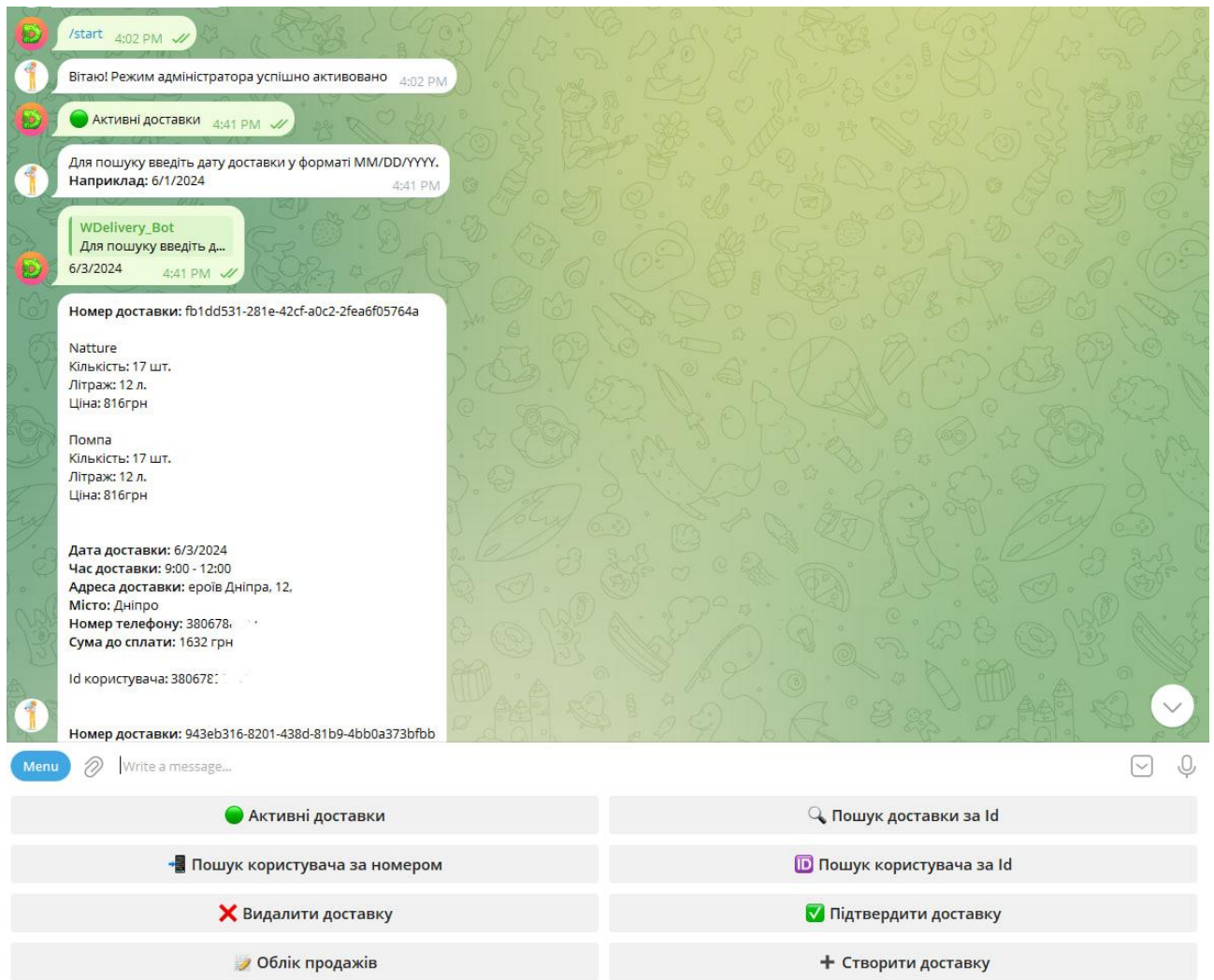


Рисунок 4.11 – Перегляд активних доставок

Для пошуку користувача за номером телефону або Id, потрібно натиснути на кнопку «Пошук користувача за номером» або «Пошук користувача за Id» відповідно та ввести бажані дані. Для прикладу початково було знайдено користувача за номером, а потім за Id. У відповідь адміністратору надається додатково місто та адреса користувача. На рисунку 4.12 відображено пошук користувача за одним з параметрів.

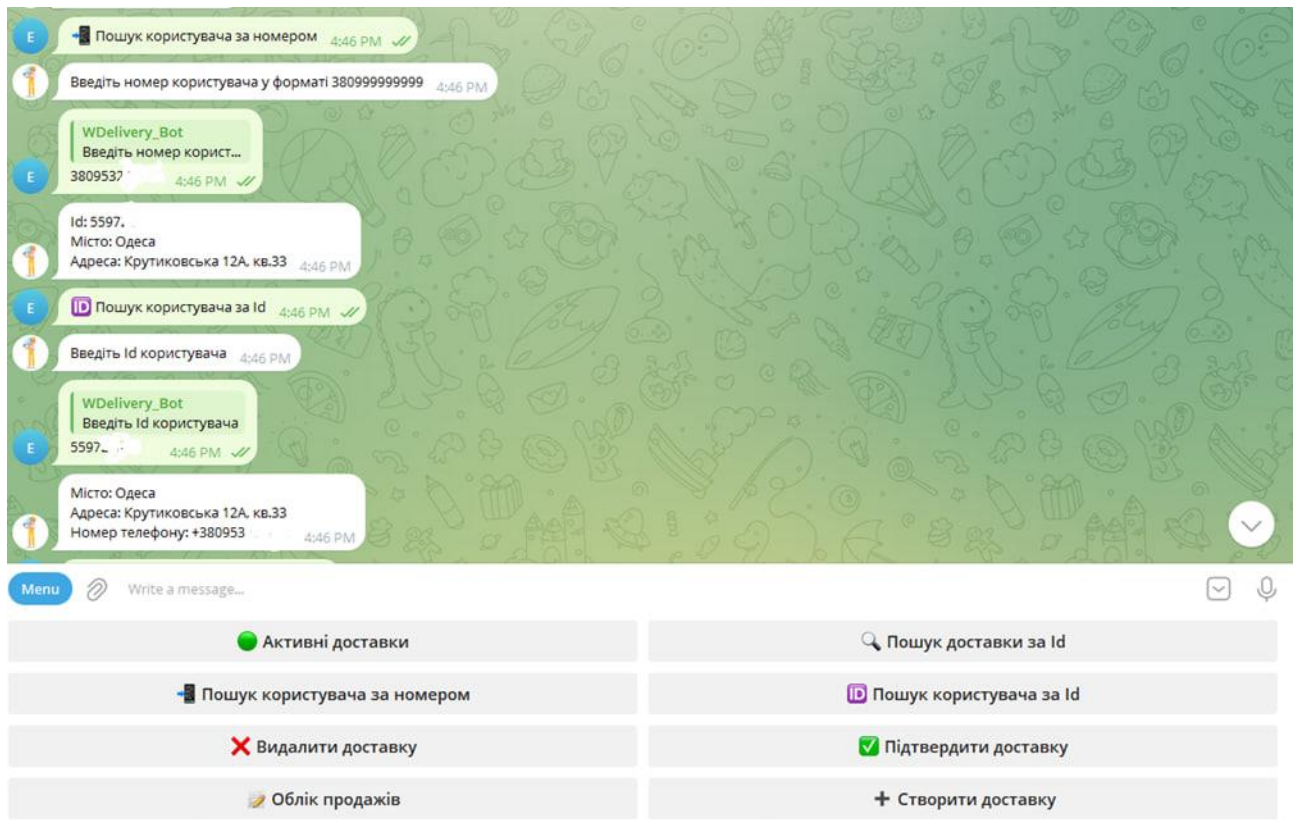


Рисунок 4.12 – Пошук користувача за номером телефону та Id

Для пошуку доставки за Id потрібно натиснути на відповідну кнопку «Пошук доставки за Id» з головного меню, після чого ввести Id доставки. Для прикладу було введено Id оформленої доставки, а саме dbc8ddda-c840-4ca1-8174-3781c531ff90 та успішно отримано відповідь. В інформації про доставку адміністратору надається номер доставки, перелік замовлених товарів, дата доставки, час, адреса та місто доставки, номер телефону користувача, сума до сплати, а також Id користувача. На рисунку 4.13 відображено пошук доставки за Id.

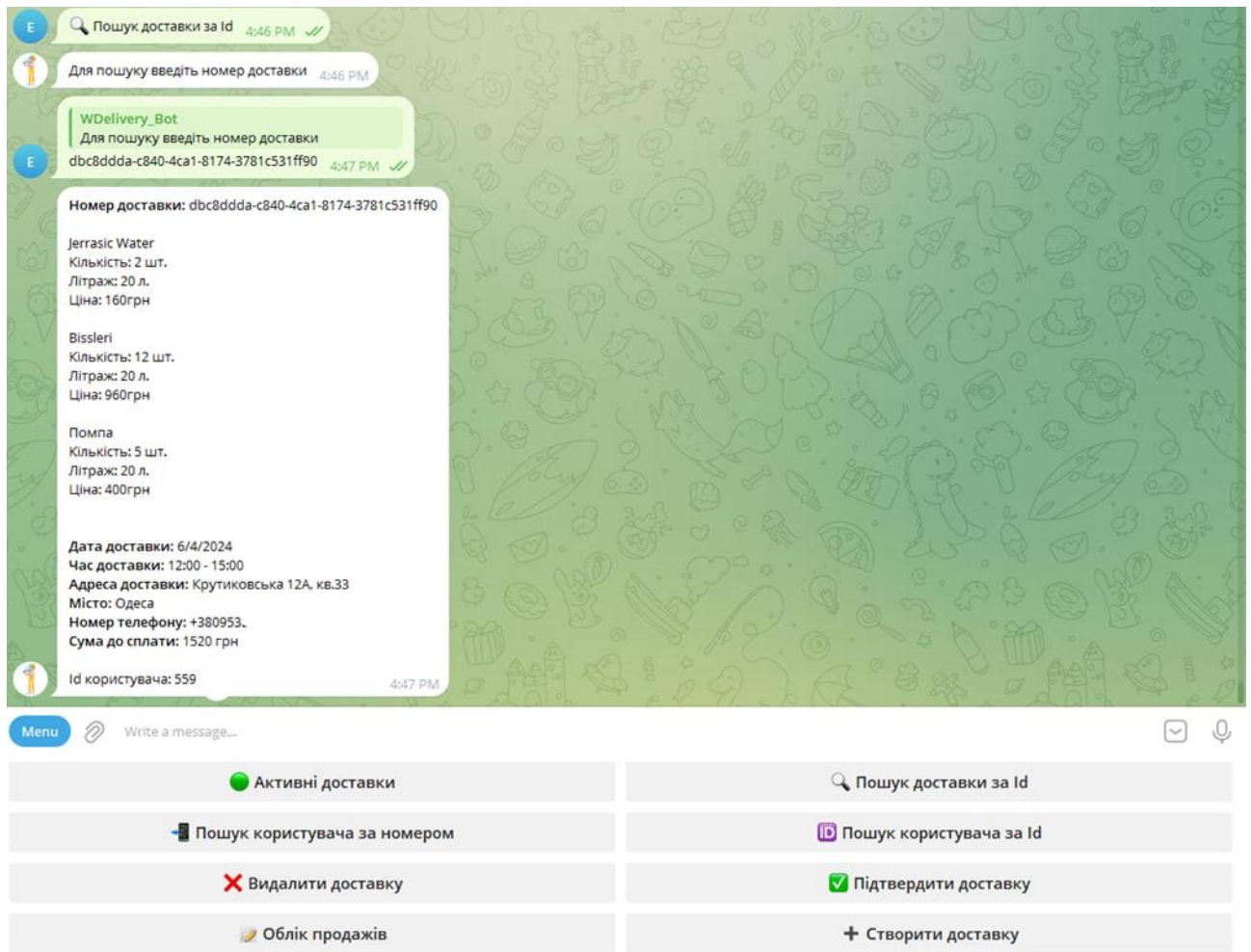


Рисунок 4.13 – Пошук доставки за Id

Для обліку продажів та перегляду інформації про виконані доставки, потрібно натиснути на кнопку «Облік продажів», після чого ввести місяць та дату для обліку. Для прикладу було введено 6/2024 та успішно отримано результат. Початково виводиться короткий опис усіх виконаних доставок за обраний місяць, із вказаними Id номером доставки, датою та сумою. Після чого на основі виконаних доставок обробляється та виводиться результат про кількість доставок, прибуток компанії, та період, за який було проведено підрахунки. На рисунку 4.14 відображено перегляд обліку продажів.

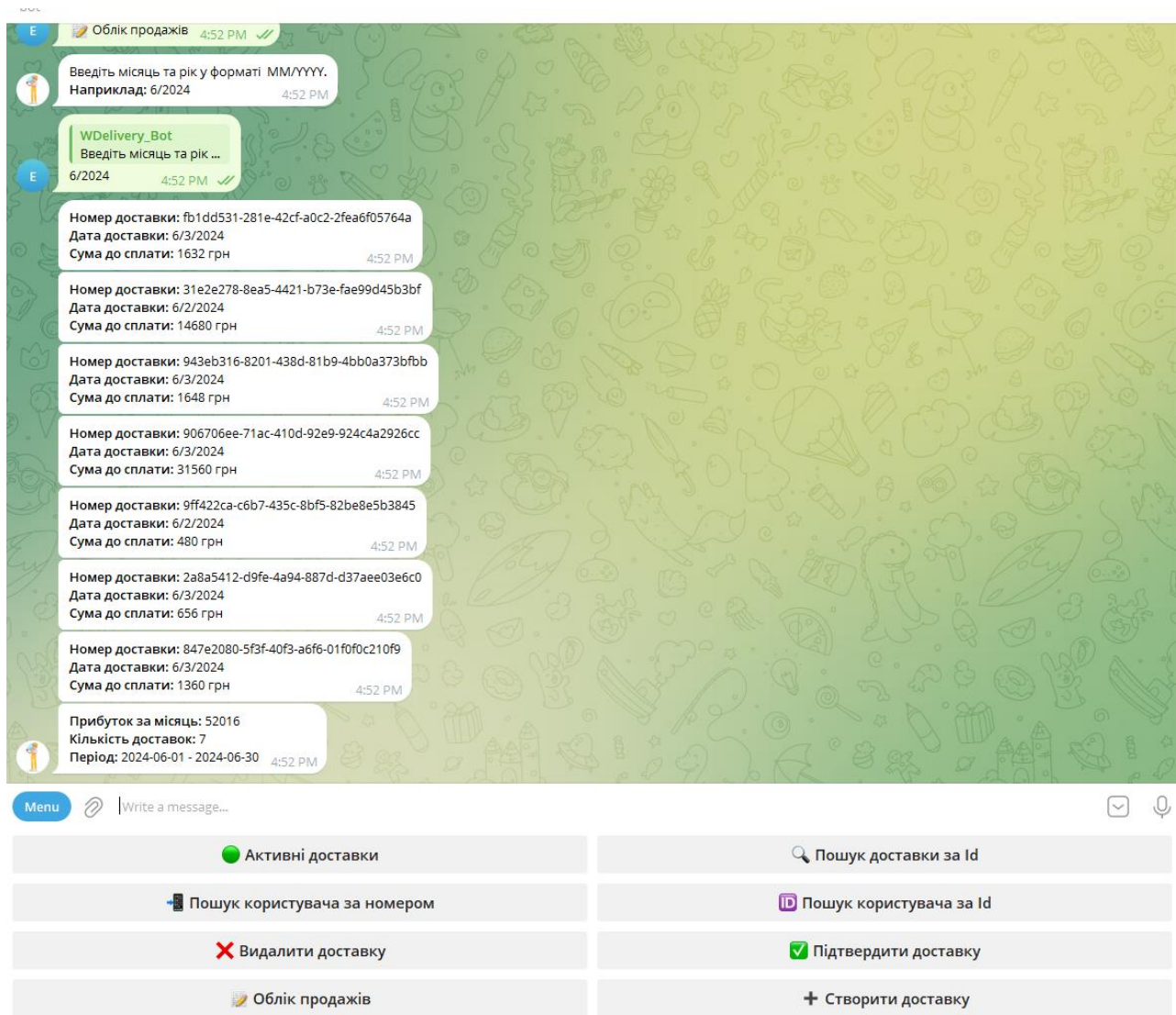


Рисунок 4.14 – Облік продажів

Також для адміністратора було додано функціонал створення доставок без обмежень, які є у користувачів. Адміністратор має натиснути на кнопку «Створити доставку», після чого його перекине в меню користувача і потрібно буде пройти усі кроки, які проходив користувач. Тобто початкова реєстрація, із вибором міста, введенням адреси та номеру телефону. Далі вибір самих товарів і відповідно оформлення доставки, з вибором дати та часу. Для прикладу було додано до замовлення 5 помп на 20 літрів кожна. На рисунку 4.15 відображено функціонал адміністратора, для створення доставки, який є повною копією меню користувача, проте без обмежень.

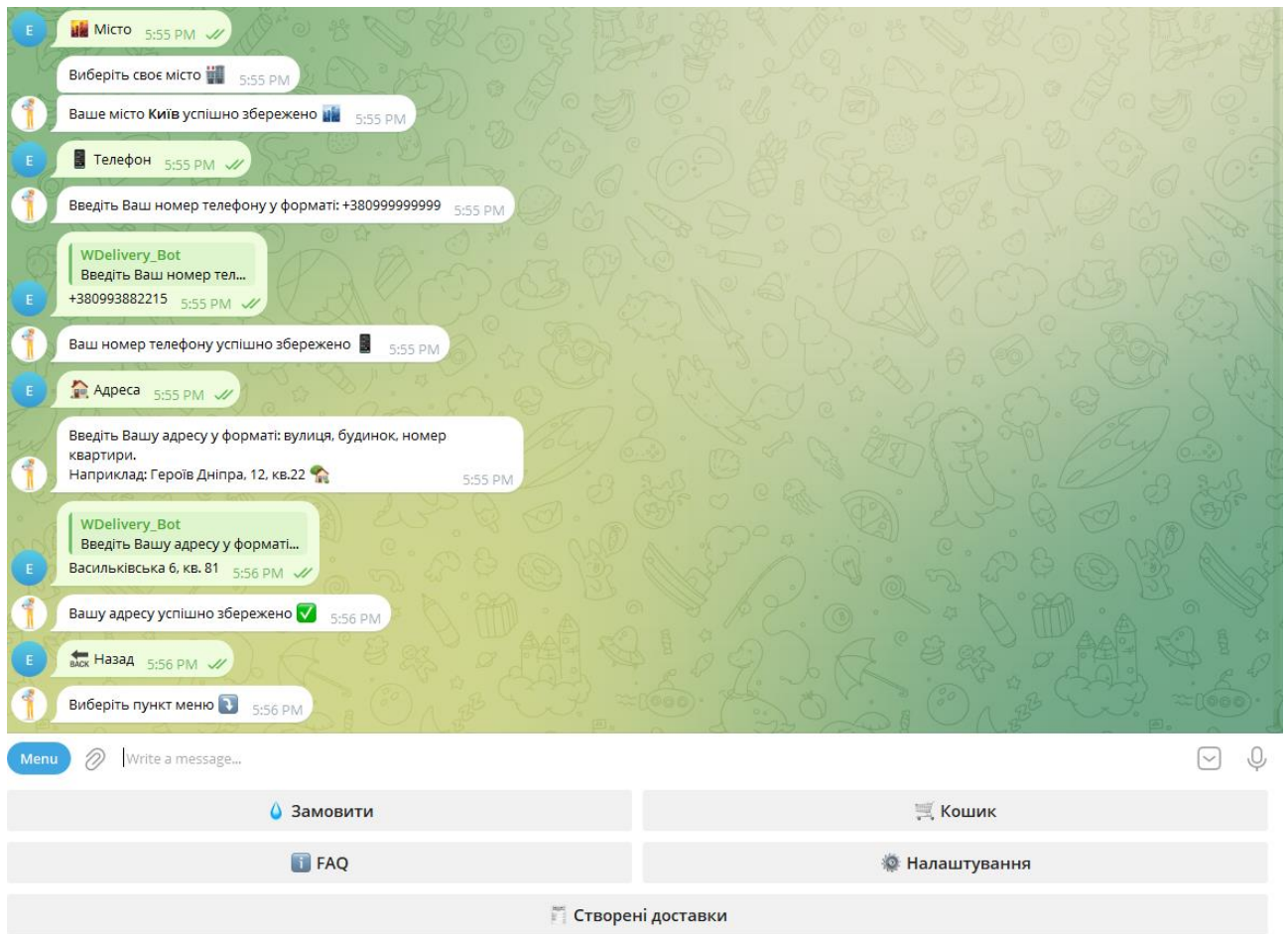


Рисунок 4.15 – Створення доставки адміністратором

Для видалення доставки з системи, потрібно натиснути на «Видалити доставку» та ввести бажаний номер доставки. Для прикладу було взято один з номерів створених доставок та видалено його з системи. На рисунку 4.16 відображено видалення доставки з системи.

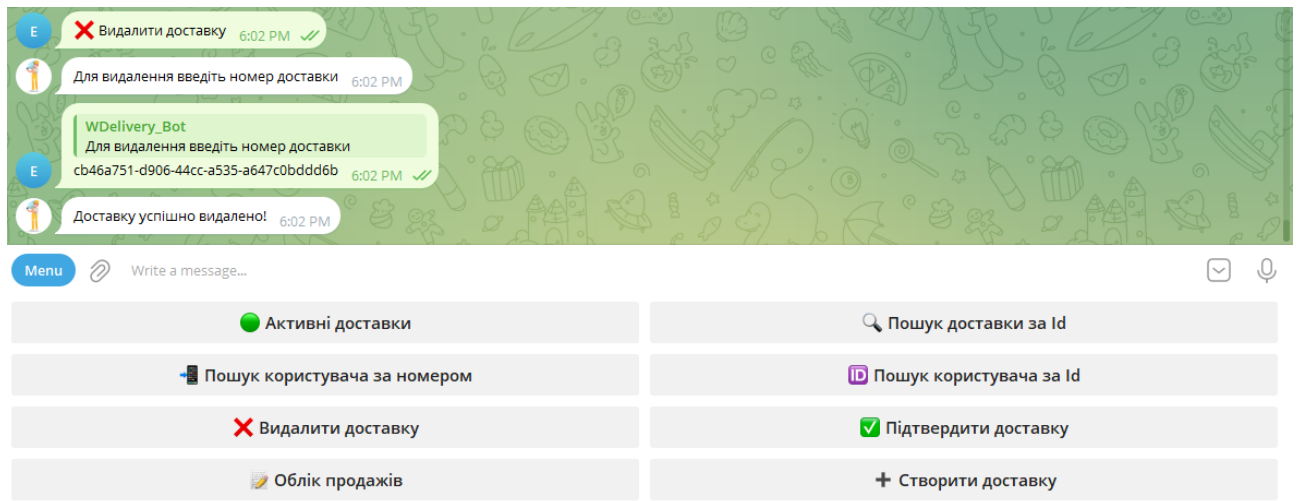


Рисунок 4.16 – Видалення доставки з системи

Висновки до розділу

У даному розділі було оцінено якість ПЗ за певними метриками. За допомогою вбудованого функціоналу в середовище розробки, було визначено індекс супроводжуваності, цикломатичну складність, глибину успадкування, зв'язок класів, рядки вихідного коду та рядки виконуваного коду. Проведено опис процесів тестування із наданими прикладами тестів та їх описом, а також проведено повний опис контрольного прикладу з основними можливостями.

Визначено, що створений вебзастосунок відповідає характеристикам продуктивності, адже час завантаження доволі малий та здатний для вирішення задач з використанням певних об'ємів ресурсів конкретних видів. Бот відповідає умові практичності через зрозумілість для користування, надані інструкції для користувачів та інтуїтивно зрозумілий інтерфейс, як для самого користувача, так і для адміністратора.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Клієнтську і серверну частини програмного забезпечення було вирішено розгорнути на платформі Microsoft Azure на операційній системі Windows.

Microsoft Azure - це інноваційна хмарна платформа від провідної компанії Microsoft, яка надає широкі можливості для розробки, виконання та зберігання додатків та даних у хмарному середовищі. Основна перевага Azure полягає у гнучкості та розширюваності, що дозволяє підприємствам будувати та масштабувати свої рішення відповідно до змінних потреб бізнесу.

Платформа реалізує дві основні моделі хмарних послуг - платформу як сервіс (PaaS) та інфраструктуру як сервіс (IaaS). Крім того, вона підтримує модель програмного забезпечення як сервісу (SaaS), що дозволяє користувачам використовувати як сторонні, так і власні сервіси Microsoft у форматі плати за використання.

Однією з ключових особливостей Microsoft Azure є глобальна мережа розподілених дата-центрів, яка забезпечує надійність та доступність послуг на всій планеті. Крім базових функцій операційних систем, платформа має додаткові можливості, такі як автоматична реплікація даних для підвищення надійності та обробка відмов інфраструктури для забезпечення стійкості та доступності.

Модель надання інфраструктури (IaaS) дозволяє користувачам орендувати різноманітні ресурси, такі як сервери та мережеве обладнання, а управління цими ресурсами здійснюється постачальником. Користувачі мають доступ до різних операційних систем для своїх віртуальних машин, що дозволяє вибрати оптимальне середовище для своїх потреб.

Багато з сервісів Azure мають API для взаємодії, що базується на обмеженнях для розподілених систем REST, що робить їх доступними з будь-якої операційної системи та платформи. Крім того, користувачі можуть

керувати своїми сервісами за допомогою візуального веб-інтерфейсу порталу Azure, що спрощує налаштування, моніторинг та управління ресурсами[12].

Першим кроком відкриємо портал Azure і пройдемо початкову реєстрацію. Для цього потрібно буде ввести ім'я, прізвище, номер телефону, адресу та поштовий індекс. Після реєстрації ми опинимося на головній сторінці порталу, на якій розташована стрічка пошука, з якої зможемо знайти усе потрібне для розгортання програмного забезпечення. На рисунку 5.1 можна відображено головну панель пошуку.

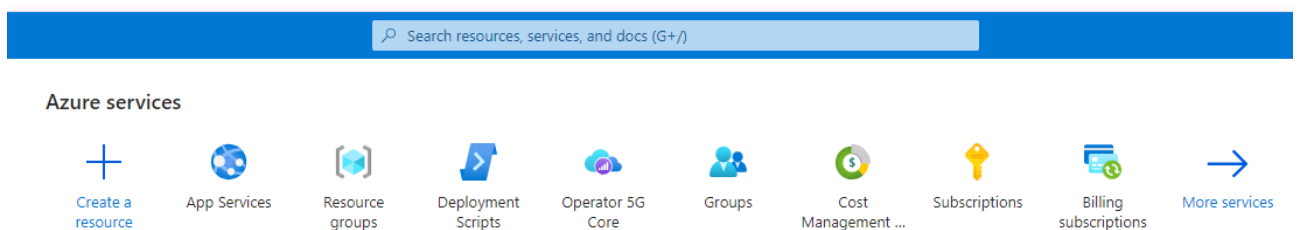


Рисунок 5.1 – Головна панель пошуку на порталі Azure

Далі нам потрібно створити нову службу додатків, для цього потрібно ввести «Служба додатків» у рядку пошуку, після чого створити нову. Після цього потрібно обрати параметри для хостингу. Виберемо підписку, групу ресурсів (якщо її немає, можна натиснути створити нову та вказати назву), назву програми, програмну технологію, ОС для служби програми, вказати, де ми хочемо розмістити свою програму та вибрати план Windows, а також обрати розмір служби програми (рисунок 5.2).

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ Azure subscription 1 ▼

Resource Group * ⓘ (New) apiwaterdelivery_group ▼

[Create new](#)

Instance Details

Name * apiwaterdelivery ✓ .azurewebsites.net

Publish * ☒ Code ☐ Container ☐ Static Web App

Runtime stack * .NET 6 (LTS) ▼

Operating System * ☐ Linux ☒ Windows

Region * East US ▼

i Not finding your App Service Plan? Try a different region or select your App Service Environment.

Pricing plans

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. [Learn more](#) ⓘ

Windows Plan (East US) * ⓘ (New) ASP-apiwaterdeliverygroup-8798 ▼

[Create new](#)

Pricing plan Standard S1 (100 total ACU, 1.75 GB memory, 1 vCPU) ▼

[Explore pricing plans](#)

Рисунок 5.2 – Розгортання програми в службі програм Azure

Далі натиснемо кнопку «Переглянути + створити». Після натискання кнопки Azure перевірить налаштування та поверне помилки, якщо їх знайде.

Натиснемо «Створити» та зачекаймо деякий час, доки служба додатків буде створена . Якщо розгортання пройде успішно, то висвітиться відповідне повідомлення (рисунок 5.3).

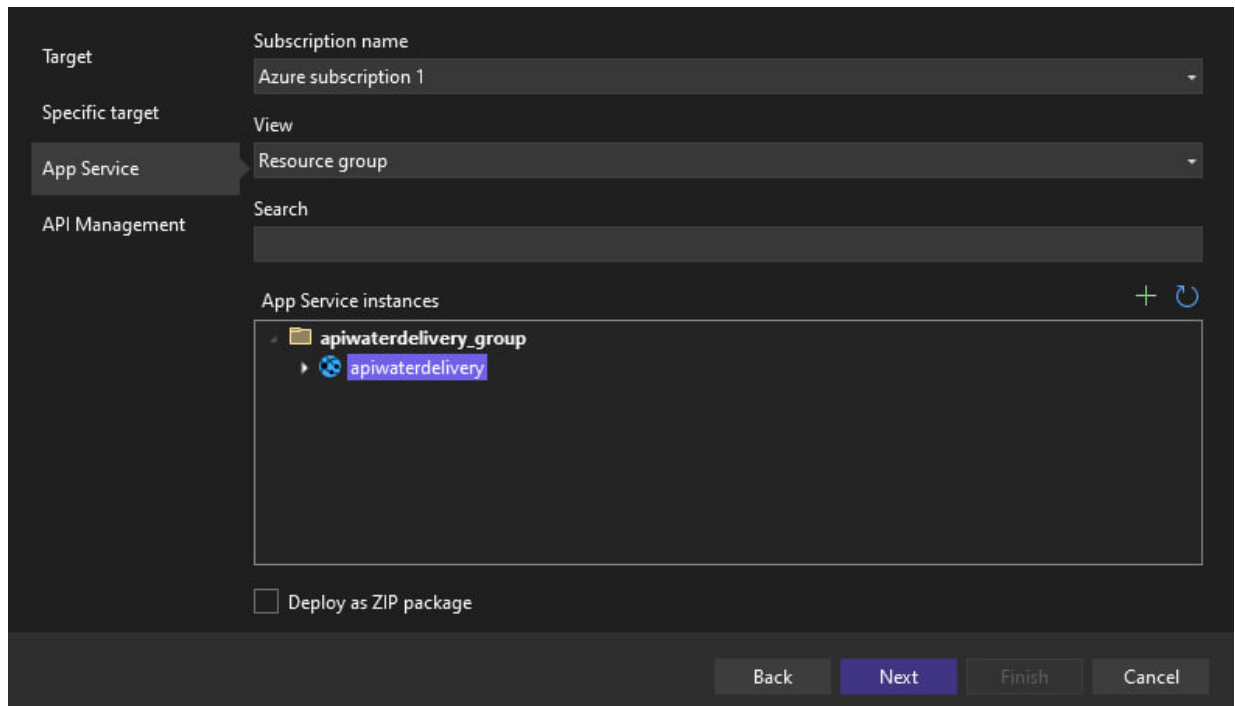


Рисунок 5.5 – Публікація програми в середовищі розробки

Залишається лише перевірити, чи всі налаштування були обрані вірно і натиснути на кнопку «Опублікувати», після чого розгортання буде успішно завершено. (рисунок 5.6).

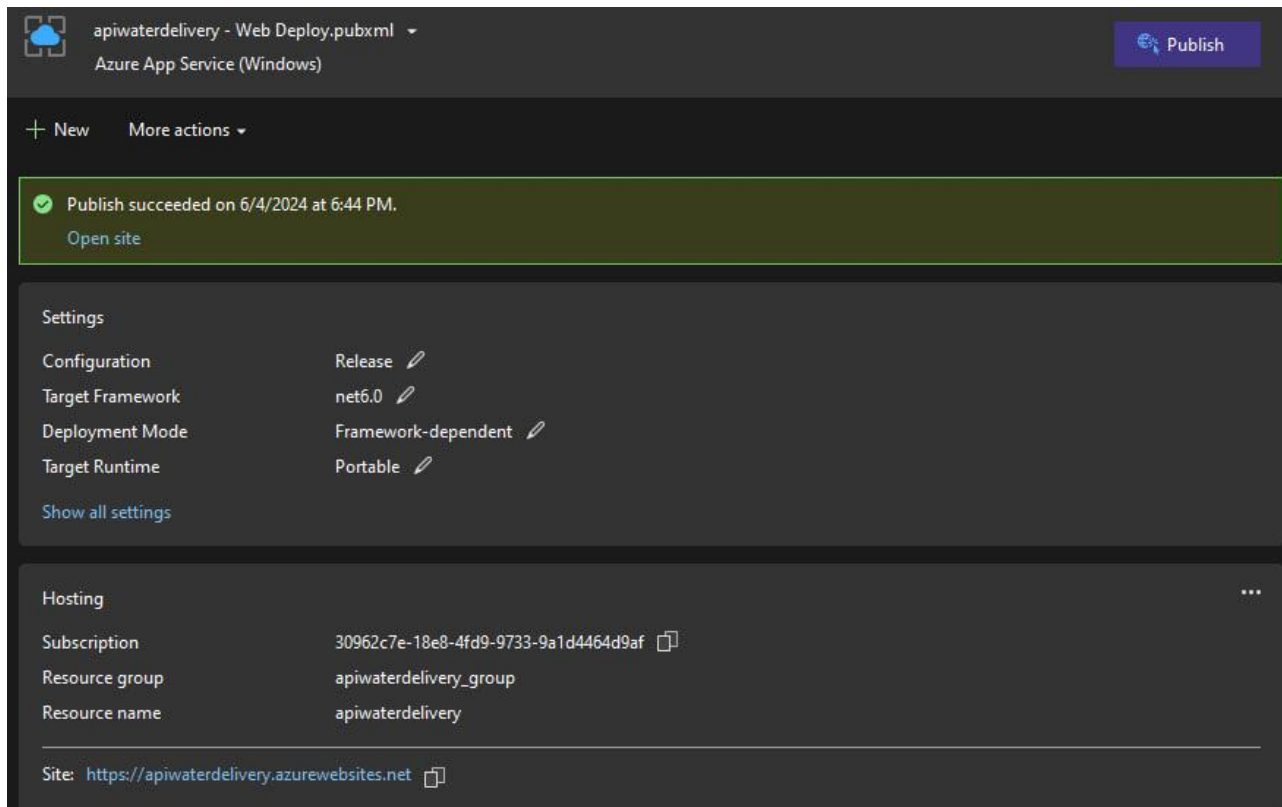


Рисунок 5.6 – Перевірка та публікація програми на Microsoft Azure

Оскільки інтерфейс виконувався за допомогою Telegram Арі, розгортання потрібно виконати як для API, так і для бота окремо. Отже, як було продемонстровано вище, відкриємо портал Azure, увійдемо і створимо нову службу додатків для самого телеграм бота. В даному випадку зміниться тільки назва. Після цього потрібно обрати параметри для хостингу. Виберемо підписку, групу ресурсів (якщо її немає, можна натиснути створити нову та вказати назву), назву програми, програмну технологію, ОС для служби програми, вказати, де ми хочемо розмістити свою програму та вибрати план Windows, а також обрати розмір служби програми. Далі натиснемо кнопку «Переглянути + створити». Після натискання кнопки Azure перевірить налаштування та поверне помилку, якщо знайде.

Натиснемо «Створити» та почекаймо деякий час, доки служба додатків буде створена та отримаємо повідомлення про успішне розгортання (рисунок 5.7). Після цього переглянемо основну інформацію про створену програму (рисунок 5.8).

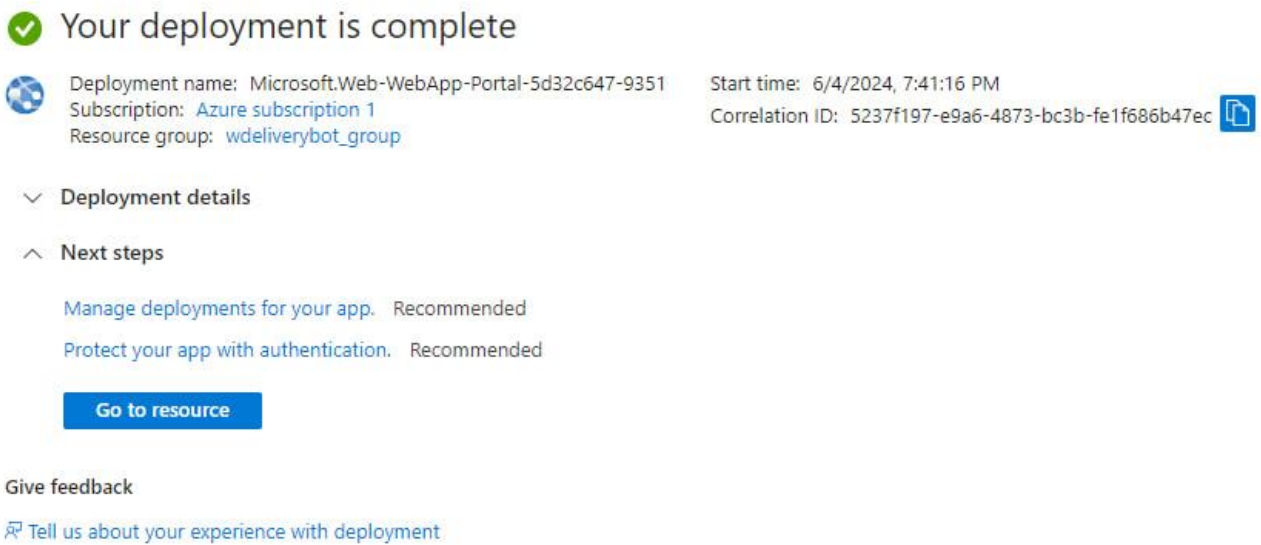


Рисунок 5.7 – Повідомлення про успішне розгортання

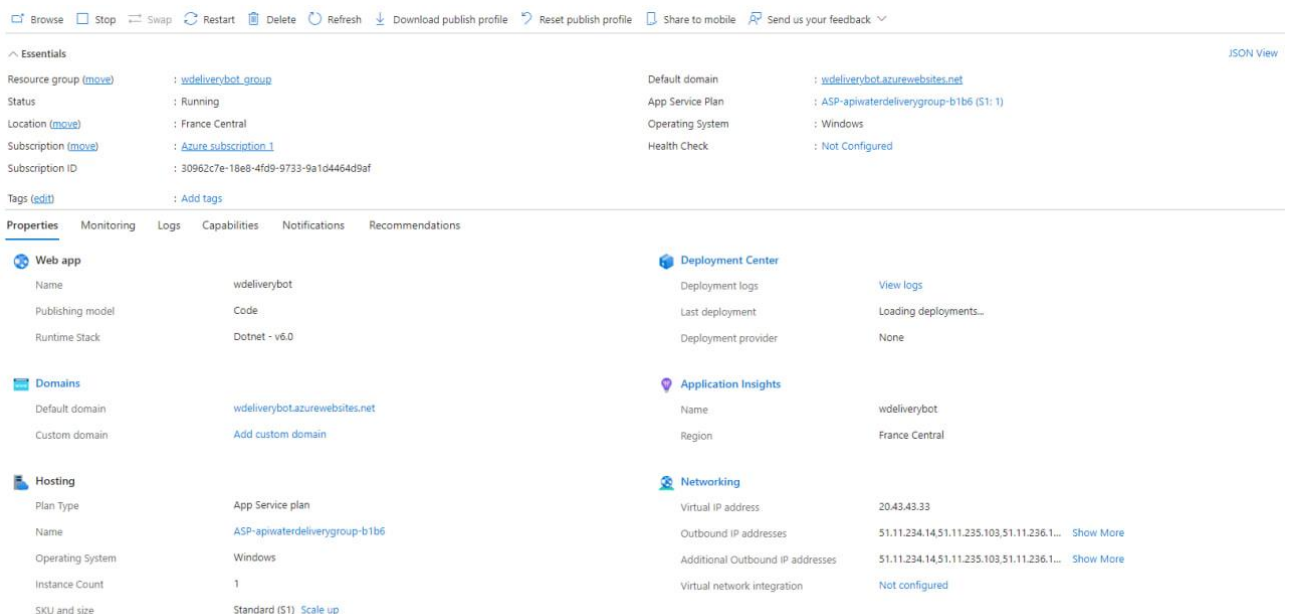


Рисунок 5.8 – Огляд програми в службі програм Azure

Далі, для того, щоб виконати розгортання на Azure телеграм бота, потрібно створити Web Job, який допоможе захостити бота. Спочатку перейдемо у середовище розробки, виберемо нашу програму та створимо архів, який ми завантажимо у Web Job. Для цього відкриємо нашу програму у файловому провіднику, знайдемо потрібний файл та створимо архів (рисунок 5.9).

This PC > Documents > visual studio 2015 > Projects > WebJobNotification > WebJobNotification > bin > Debug

Name	Date modified	Type	Size
ru	7/29/2017 11:04 PM	File folder	
zh-Hans	7/29/2017 11:04 PM	File folder	
zh-Hant	7/29/2017 11:04 PM	File folder	
Microsoft.Azure.WebJobs.Host.dll	4/22/2016 4:05 PM	Application extens...	585 KB
Microsoft.Azure.WebJobs.Host	4/22/2016 4:05 PM	XML Document	281 KB
Microsoft.Data.Edm.dll	11/20/2015 4:32 PM	Application extens...	646 KB
Microsoft.Data.Edm	11/20/2015 4:32 PM	XML Document	640 KB
Microsoft.Data.OData.dll	11/20/2015 4:33 PM	Application extens...	1,492 KB
Microsoft.Data.OData	11/20/2015 4:33 PM	XML Document	3,639 KB
Microsoft.Data.Services.Client.dll	11/20/2015 4:34 PM	Application extens...	653 KB
Microsoft.Data.Services.Client	11/20/2015 4:34 PM	XML Document	1,428 KB
Microsoft.WindowsAzure.Configuration.dll	2/3/2016 4:22 PM	Application extens...	28 KB
Microsoft.WindowsAzure.Configuration	2/3/2016 4:22 PM	XML Document	6 KB
Microsoft.WindowsAzure.Storage.dll	4/6/2016 3:02 PM	Application extens...	1,028 KB
Microsoft.WindowsAzure.Storage.pdb	4/6/2016 3:02 PM	PDB File	2,160 KB
Microsoft.WindowsAzure.Storage	4/6/2016 3:02 PM	XML Document	3,133 KB
Newtonsoft.Json.dll	1/11/2015 7:54 PM	Application extens...	502 KB
Newtonsoft.Json	1/11/2015 7:54 PM	XML Document	483 KB
System.Spatial.dll	11/20/2015 4:35 PM	Application extens...	116 KB
System.Spatial	11/20/2015 4:35 PM	XML Document	359 KB
webjob	7/30/2017 12:56 AM	Compressed (zipp...	4,162 KB
WebJobNotification	7/29/2017 11:04 PM	Application	5 KB
WebJobNotification.exe	7/29/2017 11:03 PM	XML Configuratio...	3 KB
WebJobNotification.pdb	7/29/2017 11:04 PM	PDB File	12 KB
WebJobNotification.vshost	7/29/2017 11:04 PM	Application	23 KB
WebJobNotification.vshost.exe	7/29/2017 11:03 PM	XML Configuratio...	3 KB
WebJobNotification.vshost.exe.manifest	3/19/2017 2:30 AM	MANIFEST File	1 KB

Рисунок 5.9 – Файл програми у провіднику

Тепер завантажимо цей zip-файл у Web Job. Для цього перейдемо до порталу Azure та відкривмо раніше створену веб-програму (рисунок 5.10).

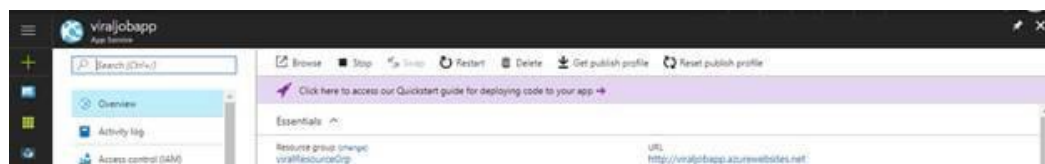


Рисунок 5.10 – Раніше створена веб-програма

Введемо Web Job у вікні пошуку (рисунок 5.11).

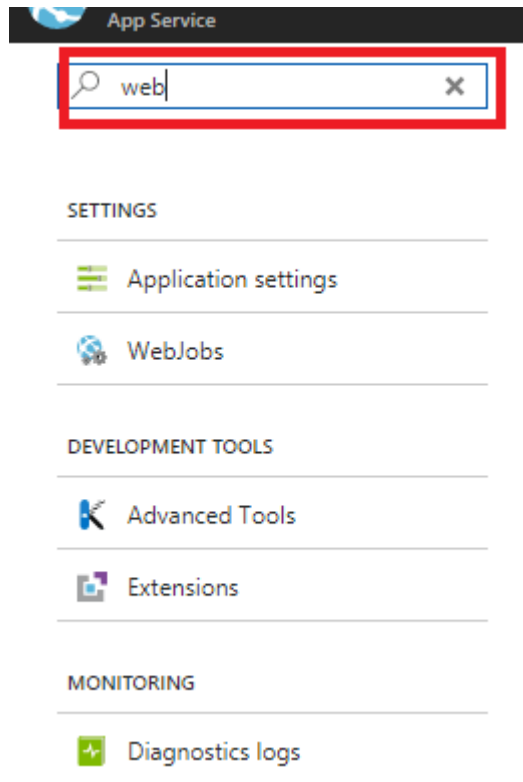


Рисунок 5.11 – Вікно пошуку

Потім натиснемо на кнопку «Веб-завдання» і натиснемо «ДОДАТИ» знак «+» (рисунок 5.12).

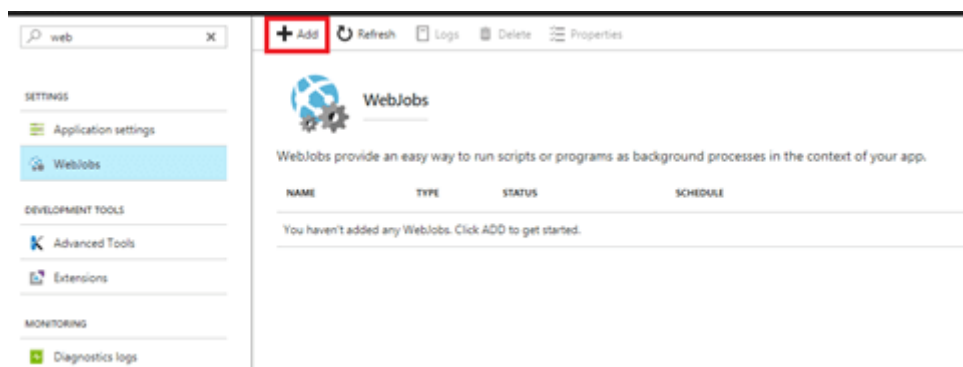


Рисунок 5.12 – Створення нового Web Job

У відкритому вікні потрібно ввести назву, завантажити архів програми, який ми створили раніше, вибрати Тип – «Ініційований або Безперервний»

відповідно до наших потреб, в даному випадку ми обрали безперервний, та натиснути на кнопку «ОК» (рисунок 5.13).

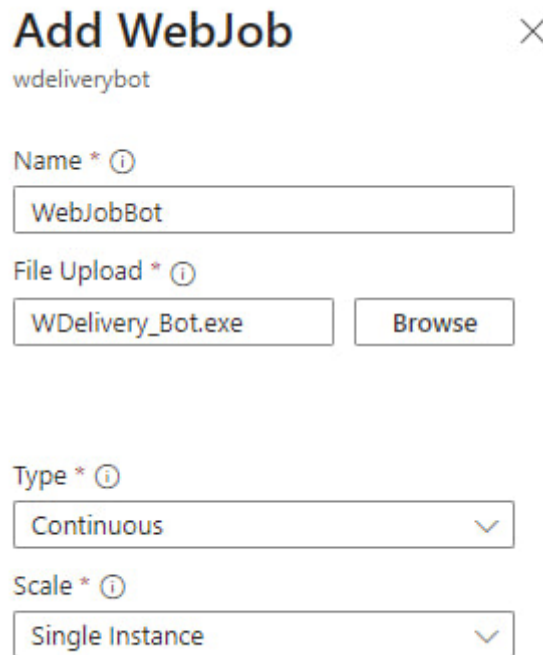
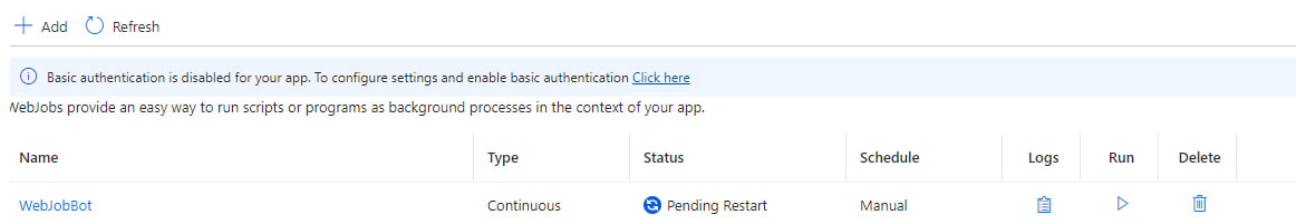


Рисунок 5.13 – Додавання Web Job

Після цього ми побачимо статус доданого Web Job, як зображено на рисунку 5.14.



Name	Type	Status	Schedule	Logs	Run	Delete
WebJobBot	Continuous	 Pending Restart	Manual			

Рисунок 5.14 – Статус доданого WebJobBot

Далі перейдемо у середовище розробки, виберемо нашу програму та натиснемо «Опублікувати». Після чого слід обрати потрібні параметри, а саме підписку, пошукову групу та сам сервіс, який ми вже створили (рисунок 5.15).

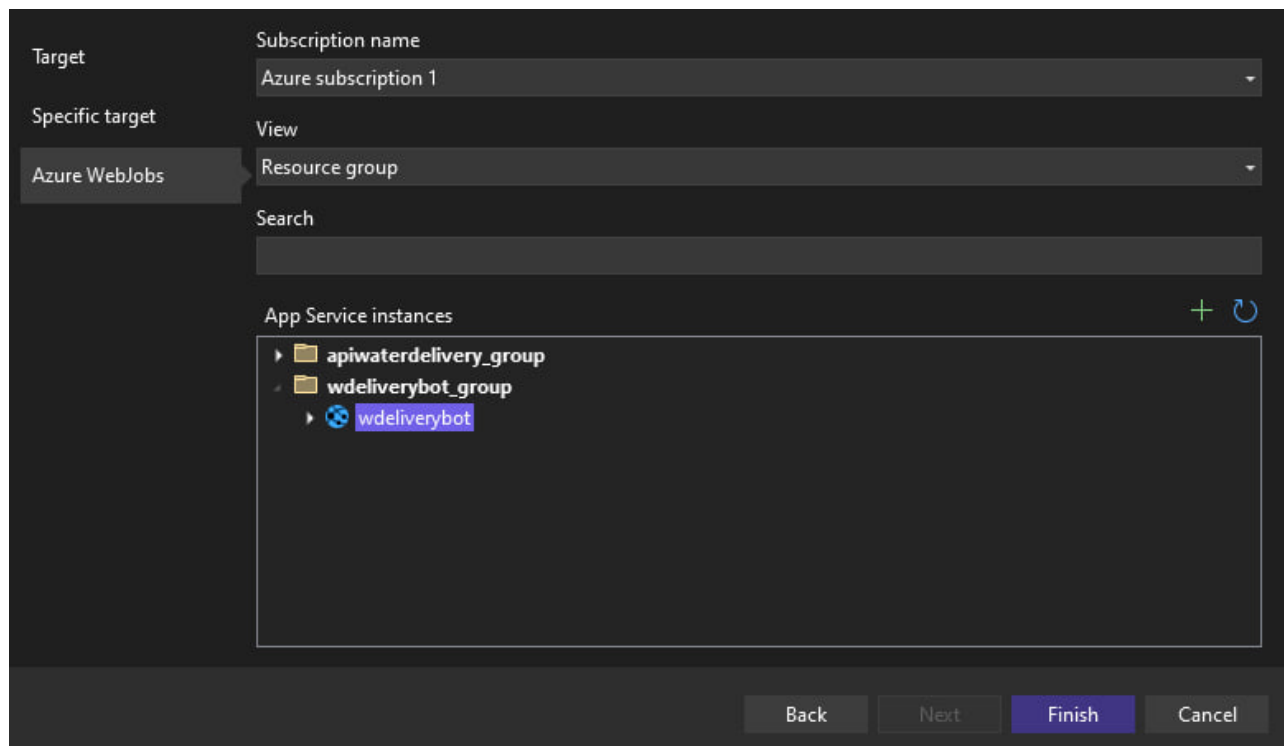


Рисунок 5.15 – Публікація програми в середовищі розробки

Залишається лише перевірити, чи всі налаштування були обрані вірно і натиснути на кнопку «Опублікувати», після чого розгортання буде успішно завершено. (рисунок 5.16).

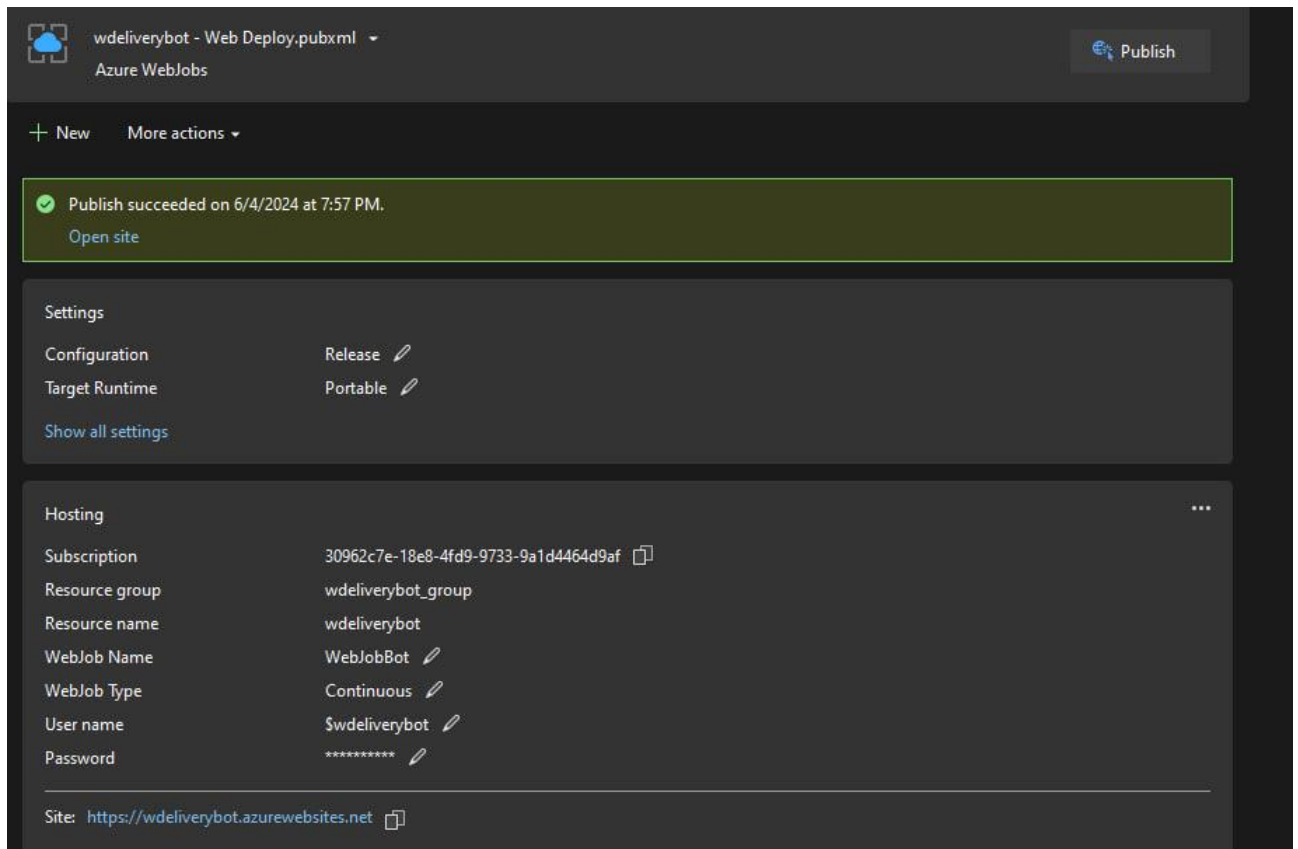


Рисунок 5.16 – Перевірка та публікація програми на Microsoft Azure

На цьому розгортання програмного забезпечення успішно завершено.

5.2 Супровід програмного забезпечення

Користувачі повинні мати можливість отримати нову версію консольного застосунку з кожною версією. Як було описане вище, для розгортання програмного забезпечення було використано платформу Microsoft Azure. Для підтримки оновлень на даній платформі, потрібно лише перейти в середовище розробки, натиснути на кнопку «Опублікувати», як для API, так і для Telegram боту. Оновлення автоматично підтягнуться на платформу. Для цього у середовищі розробки виберемо наш файл та натиснемо на кнопку «Publish». Після чого на екрані з'явиться повідомлення про успішну публікацію нової версії (рисунок 5.17).

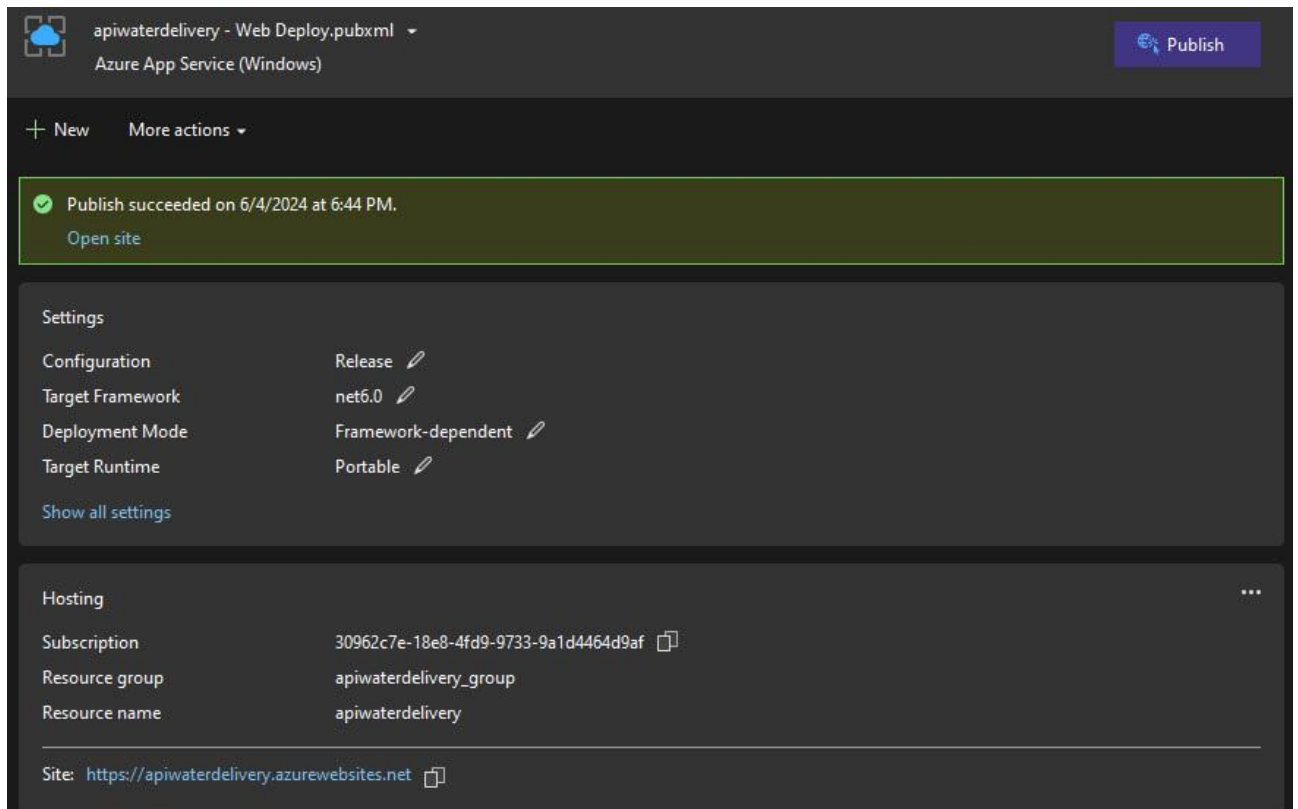


Рисунок 5.17 –Публікація оновлень

Висновки до розділу

У даному розділі було наведено повний опис покрокового розгортання ПЗ з ілюстраціями, а також те, як буде виконуватись підтримка програмного забезпечення. В рамках даної роботи клієнтську та серверну частини програмного забезпечення було вирішено розгорнути на платформі Microsoft Azure на операційній системі Windows. Дана хмарна платформа від компанії Microsoft. Надає можливість розробки, виконання додатків і зберігання даних на серверах, розташованих в розподілених дата-центрах.

Одна з основних причин вибору даної платформи це те, що практично всі сервіси Microsoft Azure мають інтерфейс взаємодії API, побудований на основі обмежень для розподілених систем REST, що дозволяє використовувати хмарні сервіси з будь-якою операційною системою, пристроєм і платформою.

Також, вагомою причиною стало те, що для супровід та підтримки проводиться доволі просто. Потрібно лише перейти в середовище розробки,

натиснути на кнопку «Опублікувати» та оновлення автоматично підтягнуться на платформу.

ВИСНОВКИ

З кожним днем месенджери входять у життя все більшої кількості людей. Вони використовуються не тільки для спілкування і обміну файлами, а й для підвищення ефективності роботи служб за допомогою автоматизації. Оскільки, Telegram у сьогоденні є одним із месенджерів, який має найбільшу аудиторію користувачів, то створення і розміщення чат-боту в його середовищі є актуальним. Телеграм підтримується на всіх операційних системах для смартфонів та десктопів. При роботі над проектом надано поняття «чат-боту», дана класифікація за форматом взаємодії, наведено переваги взаємодії користувача з ботом над спів-дією з оператором. Розглянуто існуючі рішення на прикладі телеграм ботів.

Було проведено детальний аналіз, який охоплював кілька важливих підрозділів, що допомогло сформуванню цілісного уявлення про процес створення, архітектурні рішення, вибір інструментів розробки, реалізацію програмного забезпечення та забезпечення безпеки даних.

Обґрунтовано вибір обраної мови програмування, середовище розробки, фреймворки, а також описано архітектурний патерн, обраний для розробки бота. Основою архітектури є клієнт-серверна модель, яка забезпечує ефективну взаємодію між клієнтською частиною (користувачами Telegram) та серверною частиною (API і базою даних). Розглянуто питання безпеки даних, що є критично важливим для будь-якої сучасної системи.

В результаті виконання дипломного проекту було спроектовано вебзастосунок для підвищення ефективності служби доставки води, відповідно до функціональних вимог. За рахунок впровадження електронного каталогу товарів з актуальними цінами, створення електронної заявки в системі, а також адміністрування замовлень користувачів та ведення обліку доставок, було автоматизовано ручні процеси, які раніше виконувалися співробітниками служби доставки, що дозволило значно зменшити навантаження на персонал, запобігти помилкам в роботі, затримкам та неефективності. Даний бот не

тільки підвищує ефективність діяльності служби доставки води, але й підвищує рівень задоволеності клієнтів завдяки швидкому та зручному сервісу.

Після реалізації вебзастосунку, він був протестований на пристроях з різними версіями операційних систем, з різними розмірами екранів щоб переконатися, що телеграм бот акуратно відображається на різних пристроях. В ході тестування перевірено відповідність системи встановленим функціональним вимогам та коректну роботу самого боту – заплановані дії виконувалися коректно, без збоїв. Отже, розроблений вебзастосунок є надійним і стійким та підвищує ефективність діяльності служби доставки води.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1) Технологии создания и применения чат ботов / Новые технологии
URL: <https://cyberleninka.ru/article/v/tehnologii-sozdaniya-i-primeneniya-chatbotov>
- 2) Антонов С. Что такое чат-боты и зачем они нужны? // Inform Бюро
2018. [Электронный ресурс]. Режим доступа: <https://informburo.kz/cards/chto-takoe-chat-boty-i-zachem-oninuzhny.html/>
- 3) Андриянова С.С. Использование мессенджера Telegram для продвижения бренда/ С.С.Андриянова, А.А.Веретено// Economics. – 2018. – с. 54-56.
- 4) Стефанова Н.А. Мессенджеры как цифровой бизнес – инструмент/ Н.А.Стефанова, К.О.Шматок// Карельский научный журнал. – 2018. – Т.7. №2(23). – С. 127-129.
- 5) Справочник по Bot API. [Электронный ресурс]. – Режим доступа: <https://tlgrm.ru/docs/bots/api>
- 6) REST API [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/351890/>
- 7) JSON [Электронный ресурс]. – Режим доступа: <https://www.json.org/json-ru.html>
- 8) Топ-10 мов програмування в Україні, 2010–2021. Динамічна інфографіка [Електронний ресурс] Режим доступу: <https://dou.ua/lenta/articles/top-10-lang-in-ukraine/>
- 9) Популярные среды разработки и их недостатки [Электронный ресурс] Режим доступа: https://gb.ru/posts/ide_negative
- 10) Среда разработки в программировании [Электронный ресурс] Режим доступа: <https://it-black.ru/sredy-razrabotki-v-programmirovanii/>
- 11) What is ASP.NET Core? [Электронный ресурс]. – Режим доступа: <https://umbraco.com/knowledge-base/asp-dot-net-core/>
- 12) Кухаренко А.А. Облачные вычисления. Платформа Windows Azure: реферат / Кухаренко А.А. – Гомель, 2012. — 59 с.

13) Репозиторій застосунку: <https://github.com/Martysenko11/Water-Delivery-Service>

ДОДАТОК А ЗВІТ ПОДІБНОСТІ



Ім'я користувача:
Лісовиченко Олег Іванович

Дата перевірки:
06.06.2024 01:54:52 EEST

Дата звіту:
06.06.2024 08:24:42 EEST

ID перевірки:
1016324492

Тип перевірки:
Doc vs Internet + Library

ID користувача:
76913

Назва документа: IT-04_Мартишенко_ПЗ

Кількість сторінок: 88 Кількість слів: 12109 Кількість символів: 94015 Розмір файлу: 9.36 MB ID файлу: 1016123222

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

13.2%
Схожість

Найбільша схожість: 3.22% з джерелом з Бібліотеки (ID файлу: 1016114034)

5.92% Джерела з Інтернету 237

Сторінка 90

12.9% Джерела з Бібліотеки 633

Сторінка 92

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування 26 сторінок

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2024 р.

**ВЕБЗАСТОСУНОК ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ДІЯЛЬНОСТІ
СЛУЖБИ ДОСТАВКИ ВОДИ**

Текст програми

КПІ. ІТ-0411.045440.03.12

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олег ЛІСОВИЧЕНКО

Нормоконтроль:

_____ Павло РОДІОНОВ

Виконавець:

_____ Павло МАРТИШЕНКО

Київ – 2024

Файл DynamoDbClient.cs

```
namespace WD_API.Clients
{
    public class DynamoDbClient : IDynamoDbClient
    {
        public string _tableName;
        public string _ordertableName;
        public string _temptableName;
        public string _accountingtableName;

        private readonly IAmazonDynamoDB _dynamoDb;

        public DynamoDbClient(IAmazonDynamoDB dynamoDB)
        {
            _dynamoDb = dynamoDB;
            _tableName = Constants.TableName;
            _ordertableName = Constants.Order_TableName;
            _temptableName = Constants.Temp_TableName;
            _accountingtableName = Constants.Accounting_TableName;
        }

        public async Task<bool> PostOrderToDb(OrderDbRepository data)
        {
            var request = new PutItemRequest
            {
                TableName = _ordertableName,
                Item = new Dictionary<string, AttributeValue>
                {
                    {"Order_Id", new AttributeValue { S = data.Order_Id } },
                    {"User_Id", new AttributeValue { S = data.User_Id } },
                    {"Water_Brand", new AttributeValue { S = data.Water_Brand } },
                    {"Volume", new AttributeValue { S = data.Volume } },
                    {"Num_Water", new AttributeValue { S = data.Num_Water } },
                    {"Price", new AttributeValue { S = data.Price } }
                }
            };

            try
            {
                var response = await _dynamoDb.PutItemAsync(request);
                return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
            }
            catch (Exception e)
            {
            }
        }
    }
}
```

```

{
Console.WriteLine("Here is your error \n" + e);
return false;
}
}

public async Task UpdatePhoneToDb(UserDbRepository data)
{
var request = new UpdateItemRequest
{
TableName = _tableName,
Key = new Dictionary<string, AttributeValue>
{
{"User_Id", new AttributeValue { S = data.User_Id } }
},
ExpressionAttributeValues = new Dictionary<string, AttributeValue>
{
{":newphone", new AttributeValue { S = data.Phone } }
},
UpdateExpression = "SET Phone = :newphone",
ReturnValues = "ALL_NEW"
};
var response = await _dynamoDb.UpdateItemAsync(request);
}

public async Task UpdateDeliveryToDb(UserDbRepository data)
{
var request = new UpdateItemRequest
{
TableName = _tableName,
Key = new Dictionary<string, AttributeValue>
{
{ "User_Id", new AttributeValue { S = data.User_Id } }
},
ExpressionAttributeValues = new Dictionary<string, AttributeValue>
{
{":newchecker", new AttributeValue { S = data.Checker } }
},
UpdateExpression = "SET Checker = :newchecker",
ReturnValues = "ALL_NEW"
};
var response = await _dynamoDb.UpdateItemAsync(request);
}

public async Task UpdateDataIntoDb(UserDbRepository data)

```

```

{
var request = new UpdateItemRequest
{
    TableName = _tableName,
    Key = new Dictionary<string, AttributeValue>
    {
        { "User_Id", new AttributeValue { S = data.User_Id } }
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>
    {
        { ":newcity", new AttributeValue { S = data.City } }
    },
    UpdateExpression = "SET City = :newcity",
    ReturnValues = "ALL_NEW"
};
var response = await _dynamoDb.UpdateItemAsync(request);
}

public async Task UpdateAddressIntoDb(UserDbRepository data)
{
var request = new UpdateItemRequest
{
    TableName = _tableName,
    Key = new Dictionary<string, AttributeValue>
    {
        { "User_Id", new AttributeValue { S = data.User_Id } }
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>
    {
        { ":newaddress", new AttributeValue { S = data.Address } }
    },
    UpdateExpression = "SET Address = :newaddress",
    ReturnValues = "ALL_NEW"
};

var response = await _dynamoDb.UpdateItemAsync(request);
}

public async Task UpdateBrandToDb(TempDbRepository data)
{
var request = new UpdateItemRequest
{
    TableName = _temptableName,
    Key = new Dictionary<string, AttributeValue>

```

```

{
    { "User_Id", new AttributeValue { S = data.User_Id } }
},
ExpressionAttributeValues = new Dictionary<string, AttributeValue>
{
    { ":newwaterbrand", new AttributeValue { S = data.Water_Brand } }
},
UpdateExpression = "SET Water_Brand = :newwaterbrand",
ReturnValues = "ALL_NEW"
};
var response = await _dynamoDb.UpdateItemAsync(request);
}

public async Task UpdateVolumeToDb(TempDbRepository data)
{
    var request = new UpdateItemRequest
    {
        TableName = _temptableName,
        Key = new Dictionary<string, AttributeValue>
        {
            { "User_Id", new AttributeValue { S = data.User_Id } }
        },
        ExpressionAttributeValues = new Dictionary<string, AttributeValue>
        {
            { ":newvolume", new AttributeValue { S = data.Volume } }
        },
        UpdateExpression = "SET Volume = :newvolume",
        ReturnValues = "ALL_NEW"
    };
    var response = await _dynamoDb.UpdateItemAsync(request);
}

public async Task UpdateNumberToDb(TempDbRepository data)
{
    var request = new UpdateItemRequest
    {
        TableName = _temptableName,
        Key = new Dictionary<string, AttributeValue>
        {
            { "User_Id", new AttributeValue { S = data.User_Id } }
        },
        ExpressionAttributeValues = new Dictionary<string, AttributeValue>
        {
            { ":newnum_water", new AttributeValue { S = data.Num_Water } }
        },
    };
    var response = await _dynamoDb.UpdateItemAsync(request);
}

```

```

    },
    UpdateExpression = "SET Num_Water = :newnum_water",
    ReturnValues = "ALL_NEW"
};

var response = await _dynamoDb.UpdateItemAsync(request);
}

public async Task UpdateDateIntoOrder(OrderDbRepository data)
{
    var request = new UpdateItemRequest
    {
        TableName = _ordertableName,
        Key = new Dictionary<string, AttributeValue>
        {
            { "Order_Id", new AttributeValue { S = data.Order_Id } }
        },
        ExpressionAttributeValues = new Dictionary<string, AttributeValue>
        {
            { ":neworderdate", new AttributeValue { S = data.Order_Date } }
        },
        UpdateExpression = "SET Order_Date = :neworderdate",
        ReturnValues = "ALL_NEW"
    };

    var response = await _dynamoDb.UpdateItemAsync(request);
}

public async Task UpdateTimeIntoOrder(OrderDbRepository data)
{
    var request = new UpdateItemRequest
    {
        TableName = _ordertableName,
        Key = new Dictionary<string, AttributeValue>
        {
            { "Order_Id", new AttributeValue { S = data.Order_Id } }
        },
        ExpressionAttributeValues = new Dictionary<string, AttributeValue>
        {
            { ":newordertime", new AttributeValue { S = data.Order_Time } }
        },
        UpdateExpression = "SET Order_Time = :newordertime",
        ReturnValues = "ALL_NEW"
    };
}

```

```

var response = await _dynamoDb.UpdateItemAsync(request);
}

public async Task UpdateTotalOrderForAccounting(AccountingDbRepository data)
{
    var request = new UpdateItemRequest
    {
        TableName = _accountingtableName,
        Key = new Dictionary<string, AttributeValue>
        {
            { "Order_Id", new AttributeValue { S = data.Order_Id } }
        },
        ExpressionAttributeValues = new Dictionary<string, AttributeValue>
        {
            { ":newsummary", new AttributeValue { S = data.Summary } },
            { ":newprice", new AttributeValue { S = data.Price } },
            { ":newdateacco", new AttributeValue { S = data.Date_Acco } },
            { ":newtimeacco", new AttributeValue { S = data.Time_Acco } },
            { ":newaddress", new AttributeValue { S = data.Address } },
            { ":newcityaddress", new AttributeValue { S = data.City } },
            { ":newphone", new AttributeValue { S = data.Phone } },
            { ":newuserid", new AttributeValue { S = data.User_Id } }
        },
        UpdateExpression = "SET Summary = :newsummary, Price = :newprice, Date_Acco = :newdateacco, Time_Acco = :newtimeacco, Address = :newaddress, " +
        "City = :newcityaddress, Phone = :newphone, User_Id = :newuserid",
        ReturnValues = "ALL_NEW"
    };
    var response = await _dynamoDb.UpdateItemAsync(request);
}

public async Task DeleteTrashOrder(string Order_Id)
{
    var request = new DeleteItemRequest
    {
        TableName = _accountingtableName,
        Key = new Dictionary<string, AttributeValue>
        {
            { "Order_Id", new AttributeValue { S = Order_Id } }
        }
    };
    //ExpressionAttributeValues = new Dictionary<string, AttributeValue>
    //{

```

```

// { ":userid", new AttributeValue { S = User_Id } }
//},
//ConditionExpression = "User_Id = :userid"
};
var response = await _dynamoDb.DeleteItemAsync(request);
}
public async Task DeleteFromTrashOrder(string Order_Id)
{
var request = new DeleteItemRequest
{
TableName = _ordertableName,

Key = new Dictionary<string, AttributeValue>
{
{"Order_Id", new AttributeValue { S = Order_Id }}
}
//ExpressionAttributeValues = new Dictionary<string, AttributeValue>
//{
// { ":userid", new AttributeValue { S = User_Id } }
//},
//ConditionExpression = "User_Id = :userid"
};
var response = await _dynamoDb.DeleteItemAsync(request);
}
public async Task<List<OrderTrashResponse>> GetOrderInfoByUser(string User_Id)
{
var request = new ScanRequest
{
TableName = _ordertableName,
ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
{":userid", new AttributeValue { S = User_Id }}
},
FilterExpression = "User_Id = :userid"
};
var response = await _dynamoDb.ScanAsync(request);
var db_items = response.Items;
var result = new List<OrderTrashResponse>();
foreach (var item in db_items)
{
var record = item.ToClass<OrderTrashResponse>();
result.Add(record);
}
}

```

```

//if (response.Item == null || !response.IsItemSet)
//    return null;

//var result = response.Item.ToClass<OrderDbRepository>();
return result;
}

public async Task<TempDbRepository> GetTempInfoByUser(string User_Id)
{
    var item = new GetItemRequest
    {
        TableName = _temptableName,
        Key = new Dictionary<string, AttributeValue>
        {
            {"User_Id", new AttributeValue{S = User_Id} }
        }
    };
    var response = await _dynamoDb.GetItemAsync(item);
    if (response.Item == null || !response.IsItemSet)
        return null;
    var result = response.Item.ToClass<TempDbRepository>();
    return result;
}

public async Task<List<OrderTrashResponse>> GetCheckIsOrderInfoByUser(string User_Id, string Water_Brand,
string Volume)
{
    var request = new ScanRequest
    {
        TableName = _ordertableName,
        ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
            {":userid", new AttributeValue { S = User_Id } },
            {":waterbrand", new AttributeValue { S = Water_Brand } },
            {":volume", new AttributeValue { S = Volume } }
        },
        FilterExpression = "User_Id = :userid AND Water_Brand = :waterbrand AND Volume = :volume"
    };
    var response = await _dynamoDb.ScanAsync(request);
    var db_items = response.Items;
    var result = new List<OrderTrashResponse>();
    foreach (var item in db_items)
    {
        var record = item.ToClass<OrderTrashResponse>();
        result.Add(record);
    }
}

```

```

    }
    }
    public async Task<List<AccountingDbRepository>> GetOrderByDate(string Date_Acco, string User_Id)
    {
        var request = new ScanRequest
        {
            TableName = _accountingtableName,
            ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
                {":dateacco", new AttributeValue { S = Date_Acco } },
                {":newuserid", new AttributeValue { S = User_Id } }

            },
            FilterExpression = "Date_Acco = :dateacco AND User_Id = :newuserid"
        };

        var response = await _dynamoDb.ScanAsync(request);
        var db_items = response.Items;
        var result = new List<AccountingDbRepository>();
        foreach (var item in db_items)
        {
            var record = item.ToClass<AccountingDbRepository>();
            result.Add(record);
        }
        return result;
    }
    public async Task<UserAddressResponse> GetAddress(string User_Id)
    {
        var item = new GetItemRequest
        {
            TableName = _tableName,
            Key = new Dictionary<string, AttributeValue>
            {
                {"User_Id", new AttributeValue{S = User_Id} }
            }
        };
        var response = await _dynamoDb.GetItemAsync(item);
        if (response.Item == null || !response.IsItemSet)
            return null;
        var result = response.Item.ToClass<UserAddressResponse>();
        return result;
    }

```

```

public async Task<UserCityResponse> GetCityAddress(string User_Id)
{

var item = new GetItemRequest
{
    TableName = _tableName,
    Key = new Dictionary<string, AttributeValue>
    {
        {"User_Id", new AttributeValue{S = User_Id} }
    }
};

var response = await _dynamoDb.GetItemAsync(item);
if (response.Item == null || !response.IsItemSet)
return null;
var result = response.Item.ToClass<UserCityResponse>();
return result;
}

public async Task<AccountingDbRepository> GetInfoByOrderId(string Order_Id)
{
var item = new GetItemRequest
{
    TableName = _accountingtableName,
    Key = new Dictionary<string, AttributeValue>
    {
        {"Order_Id", new AttributeValue{S = Order_Id} }
    }
};

var response = await _dynamoDb.GetItemAsync(item);
if (response.Item == null || !response.IsItemSet)
return null;

var result = response.Item.ToClass<AccountingDbRepository>();
return result;
}

public async Task<UserDbRepository> GetVerifyInfoByUser(string User_Id)
{
var item = new GetItemRequest
{
    TableName = _tableName,
    Key = new Dictionary<string, AttributeValue>
    {

```

```

{"User_Id", new AttributeValue{S = User_Id} }
}
};

var response = await _dynamoDb.GetItemAsync(item);
if (response.Item == null || !response.IsItemSet)
return null;
var result = response.Item.ToClass<UserDbRepository>();
return result;
}

public async Task<UserDbRepository> GetStatusInfoByUser(string User_Id)
{
var item = new GetItemRequest
{
TableName = _tableName,
Key = new Dictionary<string, AttributeValue>
{
{"User_Id", new AttributeValue{S = User_Id} }
}
};

var response = await _dynamoDb.GetItemAsync(item);

if (response.Item == null || !response.IsItemSet)
return null;
var result = response.Item.ToClass<UserDbRepository>();
return result;
}

public async Task<List<AccountingDbRepository>> GetAllOrdersByDate(string Date_Acco)
{
var request = new ScanRequest
{
TableName = _accountingtableName,
ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
{" :dateacco", new AttributeValue { S = Date_Acco } }
},
FilterExpression = "Date_Acco = :dateacco"
};

var response = await _dynamoDb.ScanAsync(request);
var db_items = response.Items;
var result = new List<AccountingDbRepository>();

foreach (var item in db_items)

```

```

{
var record = item.ToClass<AccountingDbRepository>();
result.Add(record);
}
//if (response.Item == null || !response.IsItemSet)
//    return null;

//var result = response.Item.ToClass<OrderDbRepository>();

return result;
}
public async Task<List<AccountingDbRepository>> GetDataForAccounting(string startDate, string endDate)
{
var request = new ScanRequest
{
TableName = _accountingtableName,
ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
{"startDate", new AttributeValue { S = startDate } },
{"endDate", new AttributeValue { S = endDate } }
},
FilterExpression = "Date_Acco BETWEEN :startDate AND :endDate"
};
var response = await _dynamoDb.ScanAsync(request);
var db_items = response.Items;
var result = new List<AccountingDbRepository>();
foreach (var item in db_items)
{
var record = item.ToClass<AccountingDbRepository>();
result.Add(record);
}
return result;
}
public async Task<UserDbRepository> GetUserInfoById(string User_Id)
{
var item = new GetItemRequest
{
TableName = _tableName,
Key = new Dictionary<string, AttributeValue>
{
{"User_Id", new AttributeValue { S = User_Id } }
}
};
};

```

```

var response = await _dynamoDb.GetItemAsync(item);
if (response.Item == null || !response.IsItemSet)
return null;
var result = response.Item.ToClass<UserDbRepository>();
return result;
}

public async Task<List<UserDbRepository>> GetUserInfoByPhone(string Phone)
{
var request = new ScanRequest
{
TableName = _tableName,
ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
{":phone", new AttributeValue { S = Phone }}
},
FilterExpression = "Phone = :phone"
};

var response = await _dynamoDb.ScanAsync(request);
var db_items = response.Items;
var result = new List<UserDbRepository>();

foreach (var item in db_items)
{
var record = item.ToClass<UserDbRepository>();
result.Add(record);
}
return result;
}
}
}
}

```

Файл IDynamoDbClient.cs

```

namespace WD_API.Clients
{
public interface IDynamoDbClient
{

public Task<bool> PostOrderToDb(OrderDbRepository data);

public Task UpdatePhoneToDb(UserDbRepository data);

public Task UpdateDeliveryToDb(UserDbRepository data);

public Task UpdateDataIntoDb(UserDbRepository data);

```

```

public Task UpdateAddressIntoDb(UserDbRepository data);

public Task UpdateBrandToDb(TempDbRepository data);

public Task UpdateVolumeToDb(TempDbRepository data);

public Task UpdateNumberToDb(TempDbRepository data);

public Task UpdateDateIntoOrder(OrderDbRepository data);

public Task UpdateTimeIntoOrder(OrderDbRepository data);

public Task UpdateTotalOrderForAccounting(AccountingDbRepository data);

public Task DeleteTrashOrder(string Order_Id);

public Task DeleteFromTrashOrder(string User_Id);

public Task<List<OrderTrashResponse>> GetOrderInfoByUser(string User_Id);

public Task<TempDbRepository> GetTempInfoByUser(string User_Id);

public Task<List<OrderTrashResponse>> GetCheckIsOrderInfoByUser(string User_Id, string Water_Brand, string
Volume);

public Task<AccountingDbRepository> GetInfoByOrderId(string Order_Id);

public Task<UserDbRepository> GetVerifyInfoByUser(string User_Id);

public Task<UserDbRepository> GetStatusInfoByUser(string User_Id);

public Task<List<AccountingDbRepository>> GetAllOrdersByDate(string Date_Acco);

public Task<List<AccountingDbRepository>> GetDataForAccounting(string startDate, string endDate);

public Task<List<AccountingDbRepository>> GetOrderByDate(string Date_Acco, string User_Id);

public Task<UserAddressResponse> GetAddress(string User_Id);

public Task<UserCityResponse> GetCityAddress(string User_Id);

public Task<UserDbRepository> GetUserInfoById(string User_Id);

public Task<List<UserDbRepository>> GetUserInfoByPhone(string Order_Date);

}
}

```

Файл AdminController.cs

```

namespace WD_API.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class AdminController : ControllerBase
    {
        private readonly ILogger<AdminController> _logger;
        private readonly IDynamoDbClient _dynamoDbClient;

        public AdminController(ILogger<AdminController> logger, IAmazonDynamoDB dynamoDb, IDynamoDbClient
dynamoDbClient)
        {
            _logger = logger;

```

```

_dynamoDbClient = dynamoDbClient;
}

[HttpGet("get/allordersbydate")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetAllOrdersInfoByDate(string Date_Acco)
{
    var result = await _dynamoDbClient.GetAllOrdersByDate(Date_Acco);

    if (result == null)
        return NotFound("Record doesn't exist in database");

    return Ok(result);
}

[HttpGet("get/ordersbyperiod")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetDataForAccounting(string startDate, string endDate)
{
    var result = await _dynamoDbClient.GetDataForAccounting(startDate, endDate);

    if (result == null || result.Count == 0)
        return NotFound("No records found in the specified period");

    return Ok(result);
}

[HttpGet("get/infobyorderid")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetInfoByOrderId([FromQuery] string Order_Id)
{
    var result = await _dynamoDbClient.GetInfoByOrderId(Order_Id);

    if (result == null)
        return NotFound("Record doesn't exist in database");

    var orderInfoResponse = new AccountingDbRepository
    {
        Order_Id = result.Order_Id,
        Summary = result.Summary,
        Price = result.Price,
        Date_Acco = result.Date_Acco,
        Time_Acco = result.Time_Acco,
        Address = result.Address,
        City = result.City,
        Phone = result.Phone,
        User_Id = result.User_Id
    };

    return Ok(orderInfoResponse);
}

[HttpGet("get/userinfobyid")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetUserById(string User_Id)
{
    var result = await _dynamoDbClient.GetUserById(User_Id);

    if (result == null)

```

```

return NotFound("Record doesn't exist in database");

var orderInfoResponse = new UserDbRepository
{
    Phone = result.Phone,
    City = result.City,
    Address = result.Address,
    Checker = result.Checker
};

return Ok(orderInfoResponse);
}

[HttpGet("get/userinfobyphone")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetUserByPhone(string Phone)
{
    var result = await _dynamoDbClient.GetUserInfoByPhone(Phone);

    if (result == null)
        return NotFound("Record doesn't exist in database");

    return Ok(result);
}

[HttpDelete("delete/orderinfo")]
public async Task<IActionResult> Delete([FromQuery] string Order_Id)
{
    var result = await _dynamoDbClient.GetInfoById(Order_Id);
    if (result == null)
        return NotFound("Record doesn't exist in database");
    await _dynamoDbClient.DeleteTrashOrder(Order_Id);

    return Ok();
}

[HttpPatch("update/checker")]
public async Task<IActionResult> AddToDelivery([FromBody] UserStatusResponse mainOrder)
{
    var data = new UserDbRepository
    {
        User_Id = mainOrder.User_Id,
        Checker = mainOrder.Checker
    };

    await _dynamoDbClient.UpdateDeliveryToDb(data);

    return Ok();
}

[HttpPatch("update/accounting")]
public async Task<IActionResult> UpdateTotalOrderForAccounting([FromBody] AccountingDbRepository
accountingOrder)
{
    var data = new AccountingDbRepository
    {
        Order_Id = accountingOrder.Order_Id,
        Summary = accountingOrder.Summary,
        Price = accountingOrder.Price,
        Date_Acco = accountingOrder.Date_Acco,
        Time_Acco = accountingOrder.Time_Acco,
    }
}

```

```

Address = accountingOrder.Address,
City = accountingOrder.City,
Phone = accountingOrder.Phone,
User_Id = accountingOrder.User_Id
};
await _dynamoDbClient.UpdateTotalOrderForAccounting(data);
return Ok();
}
}
}

```

Файл MainController.cs

```

namespace WD_API.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class MainController : ControllerBase
    {
        private readonly ILogger<MainController> _logger;
        private readonly IAmazonDynamoDB _dynamoDb;
        private readonly IDynamoDbClient _dynamoDbClient;

        public MainController(ILogger<MainController> logger, IAmazonDynamoDB dynamoDb, IDynamoDbClient dynamoDbClient)
        {
            _logger = logger;
            _dynamoDb = dynamoDb;
            _dynamoDbClient = dynamoDbClient;
        }

        [HttpPatch("update/phone")]
        public async Task<IActionResult> UpdatePhone([FromBody] UserPhoneResponse mainUser)
        {
            var data = new UserDbRepository()
            {
                User_Id = mainUser.User_Id,
                Phone = mainUser.Phone
            };

            await _dynamoDbClient.UpdatePhoneToDb(data);

            return Ok();
        }

        [HttpPatch("update/city")]
        public async Task<IActionResult> UpdateData([FromBody] UserCityResponse mainUser)
        {
            var data = new UserDbRepository()
            {
                User_Id = mainUser.User_Id,
                City = mainUser.City
            };
            await _dynamoDbClient.UpdateDataIntoDb(data);
            return Ok();
        }

        [HttpPatch("update/address")]
        public async Task<IActionResult> UpdateAddress([FromBody] UserAddressResponse mainUser)
        {
            var data = new UserDbRepository()
            {

```

```

User_Id = mainUser.User_Id,
Address = mainUser.Address
};
await _dynamoDbClient.UpdateAddressIntoDb(data);
return Ok();
}
[HttpGet("get/userverify")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetVerify([FromQuery] string User_Id)
{
var result = await _dynamoDbClient.GetVerifyInfoByUser(User_Id);

if (result == null)
return NotFound("Record doesn't exist in database");

var orderInfoResponse = new UserVerifyResponse
{
Phone = result.Phone,
City = result.City,
Address = result.Address
};
return Ok(orderInfoResponse);
}

[HttpGet("get/statuschecker")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetChecker([FromQuery] string User_Id)
{
var result = await _dynamoDbClient.GetStatusInfoByUser(User_Id);
if (result == null)
return NotFound("Record doesn't exist in database");
var orderInfoResponse = new UserStatusResponse
{
Checker = result.Checker
};
return Ok(orderInfoResponse);
}
}
}
}

```

Файл OrderController.cs

```

namespace WD_API.Controllers
{
[ApiController]
[Route("[controller]")]
public class OrderController : ControllerBase
{
private readonly ILogger<OrderController> _logger;
private readonly IDynamoDbClient _dynamoDbClient;

public OrderController(ILogger<OrderController> logger, IDynamoDbClient dynamoDbClient)
{
_logger = logger;
_dynamoDbClient = dynamoDbClient;
}

[HttpPost("add/order")]
public async Task<IActionResult> AddToOrder([FromBody] OrderMainResponse mainOrder)
{
var data = new OrderDbRepository

```

```

{
    Order_Id = mainOrder.Order_Id,
    User_Id = mainOrder.User_Id,
    Water_Brand = mainOrder.Water_Brand,
    Volume = mainOrder.Volume,
    Num_Water = mainOrder.Num_Water,
    Price = mainOrder.Price
};
var result = await _dynamoDbClient.PostOrderToDb(data);

if (result == false)
{
    return BadRequest("Cannot insert value to database. Please see console log");
}

return Ok("Value has been successfully added to DB");
}

[HttpGet("get/totalorderinfo")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetOrderInfo([FromQuery] string User_Id)
{
    var result = await _dynamoDbClient.GetOrderInfoByUser(User_Id);

    if (result == null)
        return NotFound("Record doesn't exist in database");
    return Ok(result);
}

[HttpGet("get/checkisorderinfo")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetCheckIsOrderInfo([FromQuery] string User_Id, string Water_Brand, string Volume)
{
    var result = await _dynamoDbClient.GetCheckIsOrderInfoByUser(User_Id, Water_Brand, Volume);
    if (result == null)
        return NotFound("Record doesn't exist in database");

    return Ok(result);
}

[HttpGet("get/tempinfo")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetTempInfo([FromQuery] string User_Id)
{
    var result = await _dynamoDbClient.GetTempInfoByUser(User_Id);

    if (result == null)
        return NotFound("Record doesn't exist in database");
    var orderInfoResponse = new OrderTempResponse
    {
        Water_Brand = result.Water_Brand,
        Volume = result.Volume,
        Num_Water = result.Num_Water
    };

    return Ok(orderInfoResponse);
}

[HttpPatch("update/water_brand")]

```

```

public async Task<IActionResult> UpdateBrandOrder ([FromBody] OrderBrandResponse mainOrder)
{
    var data = new TempDbRepository
    {
        User_Id = mainOrder.User_Id,
        Water_Brand = mainOrder.Water_Brand
    };
    await _dynamoDbClient.UpdateBrandToDb(data);
    return Ok();
}

[HttpPatch("update/volume")]
public async Task<IActionResult> UpdateVolumeOrder([FromBody] OrderVolumeResponse mainOrder)
{
    var data = new TempDbRepository
    {
        User_Id = mainOrder.User_Id,
        Volume = mainOrder.Volume
    };
    await _dynamoDbClient.UpdateVolumeToDb(data);
    return Ok();
}

[HttpPatch("update/number")]
public async Task<IActionResult> UpdateNumberOrder([FromBody] OrderNumberResponse mainOrder)
{
    var data = new TempDbRepository
    {
        User_Id = mainOrder.User_Id,
        Num_Water = mainOrder.Num_Water,
    };
    await _dynamoDbClient.UpdateNumberToDb(data);
    return Ok();
}

[HttpPatch("update/date")]
public async Task<IActionResult> UpdateDateOrder([FromBody] OrderDateResponse mainOrder)
{
    var data = new OrderDbRepository
    {
        Order_Id = mainOrder.Order_Id,
        Order_Date = mainOrder.Order_Date
    };
    await _dynamoDbClient.UpdateDateIntoOrder(data);
    return Ok();
}

[HttpPatch("update/time")]
public async Task<IActionResult> UpdateTimeOrder([FromBody] OrderTimeResponse mainOrder)
{
    var data = new OrderDbRepository
    {
        Order_Id = mainOrder.Order_Id,
        Order_Time = mainOrder.Order_Time
    };
    await _dynamoDbClient.UpdateTimeIntoOrder(data);
    return Ok();
}

[HttpDelete("delete/orderinfo")]
public async Task<IActionResult> Delete([FromQuery] string User_Id)
{
    var result = await _dynamoDbClient.GetOrderInfoByUser(User_Id);
}

```

```

if (result == null)
return NotFound("Record doesn't exist in database");
await _dynamoDbClient.DeleteFromTrashOrder(User_Id);

return Ok();
}

[HttpGet("get/orderbydate")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetOrderByDate(string Date_Acco, string User_Id)
{
var result = await _dynamoDbClient.GetOrderByDate(Date_Acco, User_Id);
if (result == null)
return NotFound("Record doesn't exist in database");
return Ok(result);
}

[HttpGet("get/useraddress")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetAddress(string User_Id)
{
var result = await _dynamoDbClient.GetAddress(User_Id);
if (result == null)
return NotFound("Record doesn't exist in database");
var orderInfoResponse = new UserAddressResponse
{
Address = result.Address
};
return Ok(orderInfoResponse);
}

[HttpGet("get/usercityaddress")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetCityAddress(string User_Id)
{
var result = await _dynamoDbClient.GetCityAddress(User_Id);
if (result == null)
return NotFound("Record doesn't exist in database");
var orderInfoResponse = new UserCityResponse
{
City = result.City
};

return Ok(orderInfoResponse);
}
}
}
}

```

Файл OnHandle.cs

```

namespace WDelivery_Bot.Commands
{
class OnHandle
{
public static async void OnMessageHandler(object sender, MessageEventArgs e)
{
string apiAddress = $"https://apiwaterdelivery.azurewebsites.net";/"https://localhost:44347";

```

```

var client = new HttpClient();
client.BaseAddress = new Uri(apiAddress);
var msg = e.Message;
if (msg.Contact != null)
{
    Console.WriteLine($"Id: {e.Message.From.Id} Phone number: {msg.Contact.PhoneNumber}");
    var person = new UserPhoneResponse();
    person.User_Id = Convert.ToString(e.Message.From.Id);
    person.Phone = Convert.ToString(msg.Contact.PhoneNumber);
    var json = JsonConvert.SerializeObject(person);
    var data = new StringContent(json, Encoding.UTF8, "application/json");
    var patch = await client.PatchAsync("Main/update/phone", data);
    var patchcontent = patch.Content.ReadAsStringAsync().Result;
    Console.WriteLine(patchcontent);

    await Program.client.SendTextMessageAsync(
        chatId: msg.Chat.Id,
        text: "Ваш номер телефону успішно збережено 📞",
        replyMarkup: MainButtons.GetSettingsButtons());
}

if (e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Введіть Вашу адресу")
|| e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Введіть коректні дані ✖
\n\nВведіть Вашу адресу у форматі:" +
"вулиця, будинок, номер квартири.\nНаприклад: Героїв Дніпра, 12, кв.22 🏠"))
{
    var person = new UserAddressResponse();
    person.User_Id = Convert.ToString(e.Message.From.Id);
    person.Address = $"{e.Message.Text}";
    if (e.Message.Text != null)
    {
        var json = JsonConvert.SerializeObject(person);
        var data = new StringContent(json, Encoding.UTF8, "application/json");
        var patch = await client.PatchAsync("Main/update/address", data);
        var patchcontent = patch.Content.ReadAsStringAsync().Result;
        Console.WriteLine(patchcontent);
        if (e.Message.From.Id != 1)
        {
            await Program.client.SendTextMessageAsync(
                chatId: msg.Chat.Id,
                text: "Вашу адресу успішно збережено ✅",

```

```

replyMarkup: MainButtons.GetSettingsButtons());
}
else
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Вашу адресу успішно збережено ✓",
replyMarkup: MainButtons.GetSettingsButtonsForAdmin());
}
}
else
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Введіть коректні дані ✕ \n\nВведіть Вашу адресу у форматі:" +
"вулиця, будинок, номер квартири.\nНаприклад: Героїв Дніпра, 12, кв.22 🏠",
replyMarkup: new ForceReplyMarkup() { Selective = true });
}
}

else if (e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Введіть бажану
кількість.\n" +
"🔴 Зверніть увагу, максимальна кількість данного товару не може бути більше 15 🔴"))
|| e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Введіть коректні дані ✕
\n\nВведіть бажану кількість.\n" +
"🔴 Зверніть увагу, максимальна кількість данного товару не може бути більше 15 🔴"))
{
var person = new OrderNumberResponse();
person.User_Id = Convert.ToString(e.Message.From.Id);
person.Num_Water = $"{e.Message.Text}";
string userInput = e.Message.Text;

try
{
if (Convert.ToInt32(e.Message.Text) < 16 && Convert.ToInt32(e.Message.Text) > 0 && userInput.TrimStart('0')
== e.Message.Text.ToString())
{

var json = JsonConvert.SerializeObject(person);

```

```

var data = new StringContent(json, Encoding.UTF8, "application/json");
var patch = await client.PatchAsync("Order/update/number", data);
var patchcontent = patch.Content.ReadAsStringAsync().Result;
Console.WriteLine(patchcontent);

string User_Id = Convert.ToString(e.Message.From.Id);
var result = await client.GetAsync($"Order/get/tempinfo?User_Id={User_Id}");
result.EnsureSuccessStatusCode();
var content = result.Content.ReadAsStringAsync().Result;

var info = JsonConvert.DeserializeObject<OrderTempResponse>(content);
var check_result = await
client.GetAsync($"Order/get/checkisorderinfo?User_Id={User_Id}&Water_Brand={info.Water_Brand}&Volume=
{info.Volume}");
check_result.EnsureSuccessStatusCode();
var check_content = check_result.Content.ReadAsStringAsync().Result;

var check_info = JsonConvert.DeserializeObject<List<OrderTrashResponse>>(check_content);

if (check_info.Count == 0)
{
string orderNumber = Guid.NewGuid().ToString();
Console.WriteLine(orderNumber);
var person_add = new OrderMainResponse();

person_add.Order_Id = orderNumber;
person_add.User_Id = Convert.ToString(e.Message.From.Id);
person_add.Water_Brand = info.Water_Brand;
person_add.Volume = info.Volume;
person_add.Num_Water = info.Num_Water;
person_add.Price = Convert.ToString(Convert.ToInt32(person_add.Volume) *
Convert.ToInt32(person_add.Num_Water) * 4);

var json_add = JsonConvert.SerializeObject(person_add);
var data_add = new StringContent(json_add, Encoding.UTF8, "application/json");
var patch_add = await client.PostAsync("Order/add/order", data_add);
var patchcontent_add = patch_add.Content.ReadAsStringAsync().Result;
Console.WriteLine(patchcontent_add);
}

else
{

```

```

var checked_order = "";
foreach (var item in check_info)
{
checked_order = item.Order_Id;
}

var person_add = new OrderMainResponse();
person_add.Order_Id = checked_order;
person_add.User_Id = Convert.ToString(e.Message.From.Id);
person_add.Water_Brand = info.Water_Brand;
person_add.Volume = info.Volume;
person_add.Num_Water = info.Num_Water;
person_add.Price = Convert.ToString(Convert.ToInt32(person_add.Volume) *
Convert.ToInt32(person_add.Num_Water) * 4);

var json_add = JsonConvert.SerializeObject(person_add);
var data_add = new StringContent(json_add, Encoding.UTF8, "application/json");
var patch_add = await client.PostAsync("Order/add/order", data_add);
var patchcontent_add = patch_add.Content.ReadAsStringAsync().Result;
Console.WriteLine(patchcontent_add);
}

await Program.client.SendTextMessageAsync(msg.Chat.Id, "Товар успішно додано до кошика ✓",
replyMarkup: MainButtons.SecondMenuItems());
}

else
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Введіть коректні дані ✕ \n\nВведіть бажану кількість.\n" +
"🔴 Зверніть увагу, максимальна кількість данного товару не може бути більше 15 🔴",
replyMarkup: new ForceReplyMarkup() { Selective = true });
}

}

catch (System.FormatException)
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Введіть коректні дані ✕ \n\nВведіть бажану кількість.\n" +
"🔴 Зверніть увагу, максимальна кількість данного товару не може бути більше 15 🔴",
replyMarkup: new ForceReplyMarkup() { Selective = true });
}

```

```

    }
    }
    else if (e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Введіть дату доставки"))
    {
        try
        {
            var result_totalinfo = await
            client.GetAsync($"Order/get/orderbydate?Date_Acco={e.Message.Text}&User_Id={e.Message.From.Id}");
            result_totalinfo.EnsureSuccessStatusCode();

            var content_totalinfo = result_totalinfo.Content.ReadAsStringAsync().Result;
            try
            {
                var info_totalinfo = JsonConvert.DeserializeObject<List<AccountingDbRepository>>(content_totalinfo);

                var result_items_info = "";
                if (info_totalinfo != null)
                foreach (var item in info_totalinfo)
                {
                    result_items_info = $"*Номер доставки:* " + item.Order_Id + "\n\n" + item.Summary + "\n*Дата доставки:* " +
                    item.Date_Acco +
                    "\n*Час доставки:* " + item.Time_Acco +
                    "\n*Адреса доставки:* " + item.Address +
                    "\n*Місто:* " + item.City +
                    "\n*Номер телефону:* " + item.Phone +
                    "\n*Сума до сплати:* " + item.Price + " грн\n\n";
                }

                await Program.client.SendTextMessageAsync(
                    chatId: msg.Chat.Id,
                    text: $"{result_items_info}",
                    replyMarkup: MainButtons.GetButtons(),
                    parseMode: Telegram.Bot.Types.Enums.ParseMode.Markdown);
            }
            catch (Newtonsoft.Json.JsonReaderException)
            {
                await Program.client.SendTextMessageAsync(
                    chatId: msg.Chat.Id,
                    text: $"Виберіть пункт меню ",
                    replyMarkup: MainButtons.GetButtons());
            }
        }
    }

```

```

    }
    catch (Exception ex)
    {
        await Program.client.SendTextMessageAsync(
            chatId: msg.Chat.Id,
            text: $"На цю дату немає доставок",
            replyMarkup: MainButtons.GetButtons());
    }
}

else if (e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Введіть Ваш номер телефону у форматі: +380999999999"))
    || e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Введіть коректні дані ✖ \n\nВведіть Ваш номер телефону у форматі: +380999999999"))
    {
        var person = new UserPhoneResponse();

        person.User_Id = Convert.ToString(e.Message.From.Id);
        person.Phone = $"{e.Message.Text}"; ;
        if (e.Message.Text != null)
        {
            var json = JsonConvert.SerializeObject(person);
            var data = new StringContent(json, Encoding.UTF8, "application/json");
            var patch = await client.PatchAsync("Main/update/phone", data);
            var patchcontent = patch.Content.ReadAsStringAsync().Result;
            Console.WriteLine(patchcontent);

            await Program.client.SendTextMessageAsync(
                chatId: msg.Chat.Id,
                text: "Ваш номер телефону успішно збережено 📞",
                replyMarkup: MainButtons.GetSettingsButtonsForAdmin());
        }
        else
        {
            await Program.client.SendTextMessageAsync(
                chatId: msg.Chat.Id,
                text: "Введіть коректні дані ✖ \n\nВведіть Ваш номер телефону у форматі: +380999999999",
                replyMarkup: new ForceReplyMarkup() { Selective = true });
        }
    }

if (msg.Text != null)
{

```

```
Console.WriteLine($"Username: {e.Message.From.Username} Id: {e.Message.From.Id} Message: {msg.Text}");
```

```
if (e.Message.From.Id != 1)
{
    switch (msg.Text)
    {
        case "/start":
            await Program.client.SendTextMessageAsync(
                chatId: msg.Chat.Id,
                text: "Вітаю! Я бот для замовлення доставки питної води 🚰",
                replyMarkup: MainButtons.GetButtons());
            break;
```

```
        case "/menu":
            await Program.client.SendTextMessageAsync(
                chatId: msg.Chat.Id,
                text: "Головне меню ▼",
                replyMarkup: MainButtons.GetButtons());
            break;
```

```
        case "⚙️ Налаштування":
            await Program.client.SendTextMessageAsync(
                chatId: msg.Chat.Id,
                text: "Виберіть пункт меню 1",
                replyMarkup: MainButtons.GetSettingsButtons());
            break;
```

```
        case "📞 Телефон":
            await Program.client.SendTextMessageAsync(
                chatId: msg.Chat.Id,
                text: "Введіть Ваш номер телефону",
                replyMarkup: new ForceReplyMarkup() { Selective = true });
            break;
    }
}
```

```
switch (msg.Text)
{
    case "📞 Замовити":
        try
        {
```

```

string User_Id_Check = Convert.ToString(e.Message.From.Id);
var result_check = await client.GetAsync($"Main/get/statuschecker?User_Id={User_Id_Check}");
result_check.EnsureSuccessStatusCode();
var content_check = result_check.Content.ReadAsStringAsync().Result;
var info_check = JsonConvert.DeserializeObject<UserStatusResponse>(content_check);
if (info_check.Checker != "OK")
{
string User_Id_Verify = Convert.ToString(e.Message.From.Id);
var result_verify = await client.GetAsync($"Main/get/userverify?User_Id={User_Id_Verify}");
try
{
result_verify.EnsureSuccessStatusCode();
var content_verify = result_verify.Content.ReadAsStringAsync().Result;
var info_verify = JsonConvert.DeserializeObject<UserVerifyResponse>(content_verify);

if (info_verify.Address != null && info_verify.City != null && info_verify.Phone != null && info_verify.Address
!= "null"
&& info_verify.City != "null" && info_verify.Phone != "null")
{
await Program.SendInlineKeyboard(
chatId: msg.Chat.Id,
text: "Виберіть пункт меню ☐ ",
inlineKeyboard: Inline.MainOrderKeyboard);
break;
}
else
{
if (e.Message.From.Id != 1)
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Будь-ласка, введіть усі дані ☐ ",
replyMarkup: MainButtons.GetSettingsButtons());
}
else
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Будь-ласка, введіть усі дані ☐ ",
replyMarkup: MainButtons.GetSettingsButtonsForAdmin());
}
}
}
}

```

```

}
catch (System.Net.Http.HttpRequestException)
{
if (e.Message.From.Id != 1)
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Для створення замовлення зареєструйтеся в системі ⚠ \nДля реєстрації потрібно надати " +
"інформацію про: \nНомер телефону 📞 \nМісто 📍 \nАдресу, за якою буде доставлено воду 🏠",
replyMarkup: MainButtons.GetSettingsButtons());
}
else
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Для створення замовлення зареєструйтеся в системі ⚠ \nДля реєстрації потрібно надати " +
"інформацію про: \nНомер телефону 📞 \nМісто 📍 \nАдресу, за якою буде доставлено воду 🏠",
replyMarkup: MainButtons.GetSettingsButtonsForAdmin());
}
}
}
else
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"У вас вже є активна доставка",
replyMarkup: MainButtons.GetButtons());
}
}
catch (System.Net.Http.HttpRequestException)
{
if (e.Message.From.Id != 1)
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Для створення замовлення зареєструйтеся в системі ⚠ \nДля реєстрації потрібно надати " +
"інформацію про: \nНомер телефону 📞 \nМісто 📍 \nАдресу, за якою буде доставлено воду 🏠",
replyMarkup: MainButtons.GetSettingsButtons());
}
else
{

```

```

await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Для створення замовлення зареєструйтеся в системі ⚠️ \nДля реєстрації потрібно надати " +
"інформацію про: \nНомер телефону 📞 \nМісто 🏠 \nАдресу, за якою буде доставлено воду 🚰",
replyMarkup: MainButtons.GetSettingsButtonsForAdmin());
}
}
break;

case "🏠 Місто":
await Program.SendInlineKeyboard(
chatId: msg.Chat.Id,
text: "Виберіть своє місто 📍",
inlineKeyboard: Inline.CityKeyboard);
break;

case "🏠 Адреса":
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Введіть Вашу адресу у форматі: вулиця, будинок, номер квартири.\nНаприклад: Героїв Дніпра, 12, кв.22 🏠",
replyMarkup: new ForceReplyMarkup() { Selective = true });
break;

case "⬅️ Назад":
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Виберіть пункт меню ⬇️",
replyMarkup: MainButtons.GetButtons());
break;

case "🛒 Кошик":
try
{
string User_Id_Check_Trash = Convert.ToString(e.Message.From.Id);
var result_check_trash = await client.GetAsync($"Main/get/statuschecker?User_Id={User_Id_Check_Trash}");

result_check_trash.EnsureSuccessStatusCode();
var content_check_trash = result_check_trash.Content.ReadAsStringAsync().Result;

var info_check_trash = JsonConvert.DeserializeObject<UserStatusResponse>(content_check_trash);

```

```

if (info_check_trash.Checker != "OK")
{
string User_Id = Convert.ToString(e.Message.From.Id);
var result = await client.GetAsync($"Order/get/totalorderinfo?User_Id={User_Id}");
result.EnsureSuccessStatusCode();
var content = result.Content.ReadAsStringAsync().Result;
var info = JsonConvert.DeserializeObject<List<OrderTrashResponse>>(content);
var result_items = "";
var total_sum = 0;

var result_verify = await client.GetAsync($"Main/get/userverify?User_Id={User_Id_Check_Trash}");
result_verify.EnsureSuccessStatusCode();
var content_verify = result_verify.Content.ReadAsStringAsync().Result;

var info_verify = JsonConvert.DeserializeObject<UserVerifyResponse>(content_verify);
if (info_verify.Address != null && info_verify.City != null && info_verify.Phone != null && info_verify.Address
!= "null"
&& info_verify.City != "null" && info_verify.Phone != "null")
{
foreach (var item in info)
{
result_items += item.Water_Brand + "\nКількість: " + item.Num_Water + " шт." + "\nЛітраж: " + item.Volume + "
л." +
"\nЦіна: " + item.Price + " грн\n\n";
total_sum += Convert.ToInt32(item.Price);
}

if (total_sum > 0)
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $" {result_items} Сума: {total_sum} грн\n",
replyMarkup: MainButtons.DeleteTrashItems());
}
else
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"Кошик порожній 🗑️",
replyMarkup: MainButtons.GetButtons());
}
}
}

```

```

else
{
if (e.Message.From.Id != 5678883851)
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Будь-ласка, введіть усі дані 📄",
replyMarkup: MainButtons.GetSettingsButtons());
}
else
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Будь-ласка, введіть усі дані 📄",
replyMarkup: MainButtons.GetSettingsButtonsForAdmin());
}
}
}
else
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"У вас вже є активна доставка",
replyMarkup: MainButtons.GetButtons());
}
}
catch (System.Net.Http.HttpRequestException)
{
if (e.Message.From.Id != 5678883851)
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Для створення замовлення зареєструйтеся в системі ⚠️ \nДля реєстрації потрібно надати " +
"інформацію про: \nНомер телефону 📞 \nМісто 📍 \nАдресу, за якою буде доставлено воду 🏠",
replyMarkup: MainButtons.GetSettingsButtons());
}
else
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Для створення замовлення зареєструйтеся в системі ⚠️ \nДля реєстрації потрібно надати " +

```

"інформацію про: \nНомер телефону 📞 \nМісто 📍 \nАдресу, за якою буде доставлено воду 🏠",

replyMarkup: MainButtons.GetSettingsButtonsForAdmin());

}

}

break;

case "❓ FAQ":

await Program.client.SendTextMessageAsync(

chatId: msg.Chat.Id,

text: "🔗 Найбільш поширені питання ",

replyMarkup: MainButtons.MenuWithInfo());

break;

case "📁 Про нас":

await Program.client.SendTextMessageAsync(

chatId: msg.Chat.Id,

text: \$"Компанія WDelivery - це інноваційний сервіс з доставки води 💧 у багатьох містах України 🇺🇦 \n\n" +

"Доставка натуральної артезіанської води із скважини глибиною 335 метрів. Не додаємо жодних консервантів та домішок! \n\n" +

"- понад 11 років роботи;\n- доставили понад 5 мільйонів бутлів;\n- маємо понад 60 000 вдячних клієнтів;\n" +

"- щоранку 50 автомобілів виїжджають у всі мікрорайони міст України.\n\nНаше виробництво сертифіковано" +

"за міжнародними стандартами якості ISO 9001 та ISO 22000, які працюють на базі принципів HACCP уа \n\n" +

"У разі виникнення будь-яких питань або проблем, звертайтеся до нашого адміністратора @admin_wdelivery або за номером телефону +380999999900 📞",

replyMarkup: MainButtons.MenuWithInfo());

break;

case "💰 Оплата":

await Program.client.SendTextMessageAsync(

chatId: msg.Chat.Id,

text: "У нас є два зручних способи оплати вашого замовлення: \n\n" +

"💵 Готівкою кур'єру: \nПри отриманні вашої доставки ви можете сплатити готівкою кур'єру. \n\n" +

"💳 Картою кур'єру: \nМи також приймаємо безготівкову оплату за допомогою кредитних або дебетових карток при доставці. \n\n" +

"📍 При отриманні замовлення, повідомте нашого кур'єра, яким способом оплати бажаєте скористатися, і він забезпечить безпечну оплату прямо на місці.",

```
replyMarkup: MainButtons.MenuWithInfo());
```

```
break;
```

```
case "🔄 Оновлення даних":
```

```
await Program.client.SendTextMessageAsync(
```

```
chatId: msg.Chat.Id,
```

```
text: "Якщо ви бажаєте змінити свої особисті дані, такі як номер телефону, місто чи адреса доставки, вам  
потрібно виконати наступні кроки:\n\n" +
```

```
"1. Натисніть кнопку ⚙️ Налаштування в головному меню\n" +
```

```
"2. Оберіть відповідний пункт для зміни, який вам потрібно оновити: 📍 Місто, 🏠 Адреса чи 📞 Номер  
телефону\n" +
```

```
"3. Внесіть нові дані або виберіть зі списку доступних опцій.\n" +
```

```
"4. Після цього оновлені дані буде успішно збережено в системі",
```

```
replyMarkup: MainButtons.MenuWithInfo());
```

```
break;
```

```
case "🔄 Обмін тар":
```

```
await Program.client.SendTextMessageAsync(
```

```
chatId: msg.Chat.Id,
```

```
text: "Ви можете обміняти тару у кур'єра під час доставки за ціною 40 грн за одиницю. Кур'єр огляне тару,  
щоб переконатися, що вона у належному стані." +
```

```
`${"\n\n"} Зверніть увагу: * кількість тар на обмін не може перевищувати кількості замовлених одиниць.",
```

```
replyMarkup: MainButtons.MenuWithInfo(),
```

```
parseMode: Telegram.Bot.Types.Enums.ParseMode.Markdown);
```

```
break;
```

```
case "➕ Створення доставки":
```

```
await Program.client.SendTextMessageAsync(
```

```
chatId: msg.Chat.Id,
```

```
text: "Як створити замовлення доставки:\n\n" +
```

```
"1. Зареєструйтеся в системі: 📍 Введіть інформацію про ваше місто, адресу та номер телефону.\n" +
```

```
"2. Замовте товари: Натисніть кнопку 🛒 Замовити. Оберіть бажані товари з меню.\n" +
```

```
"3. Перегляньте товари у кошику. Якщо все вірно, натисніть на кнопку 📦 Оформити замовлення.\n" +
```

```
"4. Після цього Виберіть зручний для вас день та період часу для доставки. Доставка буде успішно  
оформлена. ✅\n\n" +
```

```
`**📢 Зверніть увагу:*\n Ви не можете створити більше одного замовлення на день.\n " +
```

```
"Ви не можете замовити більше 15 одиниць кожного товару самостійно. " +
```

```
"Якщо вам потрібно замовити більше, зверніться до адміністратора.",
```

```
replyMarkup: MainButtons.MenuWithInfo(),
```

```
parseMode: Telegram.Bot.Types.Enums.ParseMode.Markdown);
```

break;

case "📅 Створені доставки":

```
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Введіть дату доставки у форматі MM/DD/YYYY",
replyMarkup: new ForceReplyMarkup() { Selective = true });
break;
```

case "🛒 Оформити замовлення":

```
await Program.SendInlineKeyboard(
chatId: msg.Chat.Id,
text: "Виберіть пункт меню 📋",
inlineKeyboard: Inline.GetDeliveryDayKeyboard());
break;
```

case "✕ Видалити товари з кошика":

```
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Підтвердіть видалення ▼",
replyMarkup: MainButtons.DeleteItemsAgree());
break;
```

case "🔙 Скасувати видалення":

```
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "🛒 Кошик",
replyMarkup: MainButtons.DeleteTrashItems());
break;
```

case "✅ Підтвердити видалення":

```
string User_Id_delete = Convert.ToString(e.Message.From.Id);
var result_delete = await client.GetAsync($"Order/get/totalorderinfo?User_Id={User_Id_delete}");
result_delete.EnsureSuccessStatusCode();
var content_delete = result_delete.Content.ReadAsStringAsync().Result;
var info_delete = JsonConvert.DeserializeObject<List<OrderTrashResponse>>(content_delete);
foreach (var item in info_delete)
{
var delete = await client.DeleteAsync($"Order/delete/orderinfo?User_Id={item.Order_Id}");
}
```

```

await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Товари успішно видалено ☒",
replyMarkup: MainButtons.GetButtons());
break;

case "☒ Додати інший товар":
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Виберіть пункт меню ☐",
replyMarkup: MainButtons.GetButtons());

await Program.SendInlineKeyboard(
chatId: msg.Chat.Id,
text: "Додати інший товар",
inlineKeyboard: Inline.MainOrderKeyboard);
break;
}
}

if (msg.Text != null && e.Message.From.Id == 1)
{
if (e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Для пошуку введіть дату
доставки у форматі MM/DD/YYYY"))
{
try
{
if (e.Message.Text != null) {
var result_totalinfo = await client.GetAsync($"Admin/get/allordersbydate?Date_Acco={e.Message.Text}");

result_totalinfo.EnsureSuccessStatusCode();
var content_totalinfo = result_totalinfo.Content.ReadAsStringAsync().Result;
var info_totalinfo = JsonConvert.DeserializeObject<List<AccountingDbRepository>>(content_totalinfo);

var result_info = "";
foreach (var item in info_totalinfo)
{
result_info += $"*Номер доставки:* " + item.Order_Id + "\n\n" + item.Summary + "\n*Дата доставки:* " +
item.Date_Acco +
"\n*Час доставки:* " + item.Time_Acco +
"\n*Адреса доставки:* " + item.Address +
"\n*Місто:* " + item.City +

```

```

$"\n*Номер телефону:* " + item.Phone +
$"\n*Сума до сплати:* " + item.Price + " грн\n\n" +
"Id користувача: " + item.User_Id + "\n\n\n";
}

await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"{result_info}",
replyMarkup: MainButtons.AdminKeyboard(),
parseMode: Telegram.Bot.Types.Enums.ParseMode.Markdown);
}

else
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"На цю дату немає доставок",
replyMarkup: MainButtons.AdminKeyboard());
}
}

catch (Exception ex)
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"На цю дату немає доставок",
replyMarkup: MainButtons.AdminKeyboard());
}
}

else if (e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Для пошуку введіть номер доставки"))
{
try
{
var result_temp = await client.GetAsync($"Admin/get/infobyorderid?Order_Id={e.Message.Text}");
result_temp.EnsureSuccessStatusCode();
var content_temp = result_temp.Content.ReadAsStringAsync().Result;
var info_temp = JsonConvert.DeserializeObject<AccountingDbRepository>(content_temp);

await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"*Номер доставки:* " + info_temp.Order_Id + "\n\n" + info_temp.Summary + "\n*Дата доставки:* " +
info_temp.Date_Acco +
$"\n*Час доставки:* " + info_temp.Time_Acco +

```

```

"\n*Адреса доставки:* " + info_temp.Address +
"\n*Місто:* " + info_temp.City +
$" \n*Номер телефону:* " + info_temp.Phone +
$" \n*Сума до сплати:* " + info_temp.Price + " грн\n\n" +
"Id користувача: " + info_temp.User_Id,
replyMarkup: MainButtons.AdminKeyboard(),
parseMode: Telegram.Bot.Types.Enums.ParseMode.Markdown);
}
catch (System.Net.Http.HttpRequestException)
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"Доставки з даним номером в системі немає",
replyMarkup: MainButtons.AdminKeyboard());
}
}
else if (e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Введіть Id
користувача"))
{
try
{
var result = await client.GetAsync($"Admin/get/userinfo?id=User_Id={e.Message.Text}");
result.EnsureSuccessStatusCode();
var content = result.Content.ReadAsStringAsync().Result;
var info = JsonConvert.DeserializeObject<UserVerifyResponse>(content);

await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"Місто: {info.City} \nАдреса: {info.Address} \nНомер телефону: {info.Phone}",
replyMarkup: MainButtons.AdminKeyboard());
}
catch (System.Net.Http.HttpRequestException)
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"Користувача з даним Id в системі немає",
replyMarkup: MainButtons.AdminKeyboard());
}
}
else if (e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Введіть номер
користувача"))
{

```

```

try
{
var result_totalinfo = await client.GetAsync($"Admin/get/userinfobyphone?Phone=%2B{e.Message.Text}");
result_totalinfo.EnsureSuccessStatusCode();
var content_totalinfo = result_totalinfo.Content.ReadAsStringAsync().Result;
var info_totalinfo = JsonConvert.DeserializeObject<List<UserDbRepository>>(content_totalinfo);
var result_items_info = "";

foreach (var item in info_totalinfo)
{
result_items_info += "Id: " + item.User_Id + "\nMicro: " + item.City + "\nАдреса: " + item.Address;
}

await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"{result_items_info}",
replyMarkup: MainButtons.AdminKeyboard());
}
catch (Telegram.Bot.Exceptions.ApiRequestException)
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"Користувача з даним номером в системі немає",
replyMarkup: MainButtons.AdminKeyboard());
}
}
else if (e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Для видалення
введіть номер доставки"))
{
try
{
var result_temp = await client.GetAsync($"Admin/get/infobyorderid?Order_Id={e.Message.Text}");
result_temp.EnsureSuccessStatusCode();
var content_temp = result_temp.Content.ReadAsStringAsync().Result;
var info_temp = JsonConvert.DeserializeObject<AccountingDbRepository>(content_temp);
var delete = await client.DeleteAsync($"Admin/delete/orderinfo?Order_Id={e.Message.Text}");

var result_verify = await client.GetAsync($"Main/get/userverify?User_Id={info_temp.User_Id}");

result_verify.EnsureSuccessStatusCode();
var content_verify = result_verify.Content.ReadAsStringAsync().Result;
var info_verify = JsonConvert.DeserializeObject<UserVerifyResponse>(content_verify);

```

```

if (info_verify.Address != null && info_verify.City != null && info_verify.Phone != null)
{
    var person_checker = new UserStatusResponse();
    person_checker.User_Id = info_temp.User_Id;
    person_checker.Checker = "COMPLETED";
    var json_checker = JsonConvert.SerializeObject(person_checker);
    var data_checker = new StringContent(json_checker, Encoding.UTF8, "application/json");
    var patch_checker = await client.PatchAsync("Admin/update/checker", data_checker);
    var patchcontent_checker = patch_checker.Content.ReadAsStringAsync().Result;
    Console.WriteLine(patchcontent_checker);
}

await Program.client.SendTextMessageAsync(
    chatId: msg.Chat.Id,
    text: "Доставку успішно видалено!",
    replyMarkup: MainButtons.AdminKeyboard());
}
catch (System.Net.Http.HttpRequestException)
{
    await Program.client.SendTextMessageAsync(
        chatId: msg.Chat.Id,
        text: $"Доставку успішно видалено!",
        replyMarkup: MainButtons.AdminKeyboard());
}
}

else if (e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Для підтвердження
доставки введіть Id користувача"))
{
    try
    {
        string User_Id_Verify = Convert.ToString(e.Message.Text);
        var result_verify = await client.GetAsync($"Main/get/userverify?User_Id={User_Id_Verify}");
        result_verify.EnsureSuccessStatusCode();
        var content_verify = result_verify.Content.ReadAsStringAsync().Result;
        var info_verify = JsonConvert.DeserializeObject<UserVerifyResponse>(content_verify);

        if (info_verify.Address != null && info_verify.City != null && info_verify.Phone != null)
        {
            var person_checker = new UserStatusResponse();
            person_checker.User_Id = e.Message.Text;
            person_checker.Checker = "COMPLETED";

```

```

var json_checker = JsonConvert.SerializeObject(person_checker);
var data_checker = new StringContent(json_checker, Encoding.UTF8, "application/json");
var patch_cheker = await client.PatchAsync("Admin/update/checker", data_checker);
var patchcontent_checker = patch_cheker.Content.ReadAsStringAsync().Result;

Console.WriteLine(patchcontent_checker);

await Program.client.SendTextMessageAsync(
    chatId: msg.Chat.Id,
    text: "Доставку підтверджено!",
    replyMarkup: MainButtons.AdminKeyboard());
}
}
catch (System.Net.Http.HttpRequestException)
{
    await Program.client.SendTextMessageAsync(
        chatId: msg.Chat.Id,
        text: $"Користувача з даним Id в системі немає",
        replyMarkup: MainButtons.AdminKeyboard());
}
}

else if (e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Введіть місяць та рік у форматі MM/YYYY"))
{
    try
    {
        if (e.Message.Text != null)
        {
            var input = e.Message.Text;
            var parts = input.Split('/');

            if (parts.Length == 2 && int.TryParse(parts[0], out int month) && int.TryParse(parts[1], out int year))
            {
                var tempDate = new DateTime(year, month, 1);
                var startDate = tempDate.AddDays(-1);

                var endDate = tempDate.AddMonths(1).AddDays(-1); // Останній день місяця
                var result = await client.GetAsync($"Admin/get/ordersbyperiod?startDate={startDate}&endDate={endDate}");
                result.EnsureSuccessStatusCode();
                var content = await result.Content.ReadAsStringAsync();
            }
        }
    }
}

```

```

var info = JsonConvert.DeserializeObject<List<AccountingDbRepository>>(content);
var total_sum = 0;
var counter = 0;

foreach (var item in info)
{
    await Program.client.SendTextMessageAsync(
        chatId: msg.Chat.Id,
        text: $"*Номер доставки:* " + item.Order_Id + "\n" + "*Дата доставки:* " + item.Date_Acco +
            "\n*Сума до сплати:* " + item.Price + " грн\n\n",
        replyMarkup: MainButtons.AdminKeyboard(),
        parseMode: Telegram.Bot.Types.Enums.ParseMode.Markdown);
    total_sum += Convert.ToInt32(item.Price);
    counter++;
}

await Program.client.SendTextMessageAsync(
    chatId: msg.Chat.Id,
    text: $"*Прибуток за місяць:* " + total_sum + "\n" + "*Кількість доставок:* " + counter +
        "\n*Період:* {startDate.AddDays(+1):yyyy-MM-dd} - {endDate:yyyy-MM-dd}",
    replyMarkup: MainButtons.AdminKeyboard(),
    parseMode: Telegram.Bot.Types.Enums.ParseMode.Markdown);
}
else
{
    await Program.client.SendTextMessageAsync(
        chatId: msg.Chat.Id,
        text: $"За наданим запитом інформації не знайдено",
        replyMarkup: MainButtons.AdminKeyboard());
}
}

else
{
    await Program.client.SendTextMessageAsync(
        chatId: msg.Chat.Id,
        text: $"За наданим запитом інформації не знайдено",
        replyMarkup: MainButtons.AdminKeyboard());
}
}

catch (Exception ex)

```

```

{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: $"За наданим запитом інформації не знайдено",
replyMarkup: MainButtons.AdminKeyboard());
}
}

else if (e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Введіть бажану
кількість."))
|| e.Message.ReplyToMessage != null && e.Message.ReplyToMessage.Text.Contains("Введіть коректні дані ✕
\nВведіть бажану кількість.\n"))
{
var person = new OrderNumberResponse();

person.User_Id = Convert.ToString(e.Message.From.Id);
person.Num_Water = $"{e.Message.Text}";

string userInput = e.Message.Text;

try
{
if (Convert.ToInt32(e.Message.Text) > 0 && userInput.TrimStart('0') == e.Message.Text.ToString())
{
var json = JsonConvert.SerializeObject(person);
var data = new StringContent(json, Encoding.UTF8, "application/json");
var patch = await client.PatchAsync("Order/update/number", data);
var patchcontent = patch.Content.ReadAsStringAsync().Result;
Console.WriteLine(patchcontent);

string User_Id = Convert.ToString(e.Message.From.Id);
var result = await client.GetAsync($"Order/get/tempinfo?User_Id={User_Id}");
result.EnsureSuccessStatusCode();
var content = result.Content.ReadAsStringAsync().Result;
var info = JsonConvert.DeserializeObject<OrderTempResponse>(content);

var check_result = await
client.GetAsync($"Order/get/checkisorderinfo?User_Id={User_Id}&Water_Brand={info.Water_Brand}&Volume=
{info.Volume}");
check_result.EnsureSuccessStatusCode();

```

```

var check_content = check_result.Content.ReadAsStringAsync().Result;
var check_info = JsonConvert.DeserializeObject<List<OrderTrashResponse>>(check_content);

if (check_info.Count == 0)
{

string orderNumber = Guid.NewGuid().ToString();
Console.WriteLine(orderNumber);
var person_add = new OrderMainResponse();

person_add.Order_Id = orderNumber;
person_add.User_Id = Convert.ToString(e.Message.From.Id);
person_add.Water_Brand = info.Water_Brand;
person_add.Volume = info.Volume;
person_add.Num_Water = info.Num_Water;
person_add.Price = Convert.ToString(Convert.ToInt32(person_add.Volume) *
Convert.ToInt32(person_add.Num_Water) * 4);
var json_add = JsonConvert.SerializeObject(person_add);
var data_add = new StringContent(json_add, Encoding.UTF8, "application/json");
var patch_add = await client.PostAsync("Order/add/order", data_add);
var patchcontent_add = patch_add.Content.ReadAsStringAsync().Result;
Console.WriteLine(patchcontent_add);
}
else
{
var checked_order = "";
foreach (var item in check_info)
{
checked_order = item.Order_Id;
}
var person_add = new OrderMainResponse();
person_add.Order_Id = checked_order;
person_add.User_Id = Convert.ToString(e.Message.From.Id);
person_add.Water_Brand = info.Water_Brand;
person_add.Volume = info.Volume;
person_add.Num_Water = info.Num_Water;
person_add.Price = Convert.ToString(Convert.ToInt32(person_add.Volume) *
Convert.ToInt32(person_add.Num_Water) * 4);
var json_add = JsonConvert.SerializeObject(person_add);
var data_add = new StringContent(json_add, Encoding.UTF8, "application/json");
var patch_add = await client.PostAsync("Order/add/order", data_add);
var patchcontent_add = patch_add.Content.ReadAsStringAsync().Result;

```

```

Console.WriteLine(patchcontent_add);
}
await Program.client.SendTextMessageAsync(msg.Chat.Id, "Товар успішно додано", replyMarkup:
MainButtons.SecondMenuItems());
}
else
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Введіть коректні дані ✕ \nВведіть бажану кількість.\n",
replyMarkup: new ForceReplyMarkup() { Selective = true });
}
}
catch (System.FormatException)
{
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Введіть коректні дані ✕ \nВведіть бажану кількість.\n",
replyMarkup: new ForceReplyMarkup() { Selective = true });
}
}
}
if (msg.Text != null && e.Message.From.Id == 5678883851)
{
switch (msg.Text)
{
case "/start":
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Вітаю! Режим адміністратора успішно активовано",
replyMarkup: MainButtons.AdminKeyboard());
break;

case "/menu":
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Головне меню",
replyMarkup: MainButtons.AdminKeyboard());
break;

case "⚙️ Налаштування":
await Program.client.SendTextMessageAsync(

```

```

chatId: msg.Chat.Id,
text: "Виберіть пункт меню ,
replyMarkup: MainButtons.GetSettingsButtonsForAdmin());
break;

case " Телефон":
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Введіть Ваш номер телефону у форматі: +380999999999",
replyMarkup: new ForceReplyMarkup() { Selective = true });
break;

case " Активні доставки":

await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Для пошуку введіть дату доставки у форматі ММ/ДД/YYYY. \n*Наприклад:* 6/1/2024",
replyMarkup: new ForceReplyMarkup() { Selective = true },
parseMode: Telegram.Bot.Types.Enums.ParseMode.Markdown);
break;

case " Пошук користувача за Id":
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Введіть Id користувача",
replyMarkup: new ForceReplyMarkup() { Selective = true });
break;

case " Пошук користувача за номером":
await Program.client.SendTextMessageAsync(
chatId: msg.Chat.Id,
text: "Введіть номер користувача у форматі 380999999999",
replyMarkup: new ForceReplyMarkup() { Selective = true });
break;

case "
```

```

case "🔍 Пошук доставки за Id":
    await Program.client.SendTextMessageAsync(
        chatId: msg.Chat.Id,
        text: "Для пошуку введіть номер доставки",
        replyMarkup: new ForceReplyMarkup() { Selective = true });
    break;

case "✅ Підтвердити доставку":
    await Program.client.SendTextMessageAsync(
        chatId: msg.Chat.Id,
        text: "Для підтвердження доставки введіть Id користувача",
        replyMarkup: new ForceReplyMarkup() { Selective = true });
    break;

case "📅 Облік продажів":
    await Program.client.SendTextMessageAsync(
        chatId: msg.Chat.Id,
        text: "Введіть місяць та рік у форматі ММ/YYYY. \n*Наприклад:* 6/2024",
        replyMarkup: new ForceReplyMarkup() { Selective = true },
        parseMode: Telegram.Bot.Types.Enums.ParseMode.Markdown);
    break;

case "✚ Створити доставку":
    await Program.client.SendTextMessageAsync(
        chatId: msg.Chat.Id,
        text: "Виберіть пункт меню 📄",
        replyMarkup: MainButtons.GetButtons());
    break;
}
}
}
}

```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2024 р.

**ВЕБЗАСТОСУНОК ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ДІЯЛЬНОСТІ
СЛУЖБИ ДОСТАВКИ ВОДИ**

Програма та методика тестування

КП.ІТ-0411.045440.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олег ЛІСОВИЧЕНКО

Нормоконтроль:

_____ Павло РОДІОНОВ

Виконавець:

_____ Павло МАРТИШЕНКО

Київ – 2024

ЗМІСТ

1	ОБ'ЄКТ ВИПРОБУВАНЬ	3
2	МЕТА ТЕСТУВАННЯ	4
3	МЕТОДИ ТЕСТУВАННЯ	5
4	ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ	6

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є результат виконання дипломного проекту – вебзастосунок ьпідвищення ефективності діяльності служби доставки води “WDelivery_Bot”. Тестується весь функціонал розробленої системи, що виконується на серверній частині. Також тестуванню підлягає користувацька частина застосунку та частина, яка відноситься до адміністратора.

Так як вебзастосунок є веб орієнтованим та розповсюджується як відкритий об’єкт з віддаленого серверу, використовувати його можна з самого застосунку Телеграм, будь-якого веб-браузера та на будь-якій операційній системі.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
 - перевірка збереження даних;
 - перевірка сумісності веб-додатку з останніми версіями сучасних браузерів (Chrome, Opera, Firefox, ...);
 - знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- динамічне тестування – застосовується в процесі виконання програми. Коректність застосунку перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- системне тестування – перевіряється усе програмне забезпечення в цілому;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;
- тестування «сірої скриньки» – об'єктом тестування тут є деякі особливості внутрішньої поведінки програми. Перевіряється коректність вихідних даних при заданих вхідних, застосовується для тестування окремих алгоритмів (функцій).

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення, так і в зручності користування, відповідності інтерфейсу зазначеним вимогам. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- тестування на мобільних пристроях з різною версією операційної системи;
- тестування зміни орієнтації екрану;
- динамічне тестування на відповідність функціональним вимогам;
- тестування інтерфейсу користувача;
- тестування зручності використання;
- тестування на виведення повідомлень про помилку, коли це необхідно;
- тестування позитивного, негативного виконання функцій

тестування на різних веб-браузерах.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2024 р.

**ВЕБЗАСТОСУНОК ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ДІЯЛЬНОСТІ
СЛУЖБИ ДОСТАВКИ ВОДИ**

Керівництво користувача

КП.ІТ-0411.045440.05.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олег ЛІСОВИЧЕНКО

Нормоконтроль:

_____ Павло РОДІОНОВ

Виконавець:

_____ Павло МАРТИШЕНКО

Київ – 2024

ЗМІСТ

1	Призначення програми	3
2	ПІДГОТОВКА ДО РОБОТИ з програмним забезпеченням	4
2.1	Системні вимоги для коректної роботи	4
2.2	Завантаження застосунку	4
2.3	Перевірка коректної роботи	4
3	Виконання програми	5

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Вебзастосунок підвищення ефективності діяльності служби доставки води “Water Delivery” – це Telegram бот, завдяки якому можна замовити доставку води в одному із запропонованих міст України, переглянути створене замовлення, дізнатися його поточну ціну, отримати більше інформації про саму службу доставки, методи оплати та інше. А також для адміністратора надається можливість адмініструвати замовлення в системі та вести облік самих доставок. Основним завданням є автоматизація ручних процесів.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Мінімальна конфігурація технічних засобів:

- Intel Pentium 4 чи пізніше, що підтримує SSE3;
- об'єм ОЗП: 2 Гб;
- підключення до мережі Інтернет.

Рекомендована конфігурація технічних засобів:

- Intel Pentium 4 чи пізніше, що підтримує SSE3;
- об'єм ОЗП: 4 Гб;
- підключення до мережі Інтернет.

Або

- наявність мобільного пристрою з операційною системою на базі Android з версією 6.0 та вище або IOS 7 та вище;
- наявність доступу до Інтернету;
- для встановлення додатку Телеграм на мобільному пристрої повинно бути не менше 200 МБ вільної пам'яті.

2.2 Завантаження застосунку

Оскільки вебзастосунок підвищення ефективності діяльності служби доставки води було реалізовано у вигляді Telegram бота, для користування потрібно встановити додаток Telegram на пристрій, після чого знайти бота, ввівши у пошуковому вікні «@WrDeliVery_Bot».

2.3 Перевірка коректної роботи

Для перевірки коректності процесу установки, потрібно вибрати бота, який був запропонований після введення назви у пошуковому вікні та натиснути кнопку «start». Після натиснення кнопки, бот має привітатися, після чого з'явиться головне меню.

3 ВИКОНАННЯ ПРОГРАМИ

При запуску програмного застосунку користувачу буде відображено привітальне повідомлення та з'явиться головне меню (рисунок 3.1). Початково користувач натискає на кнопку «start».

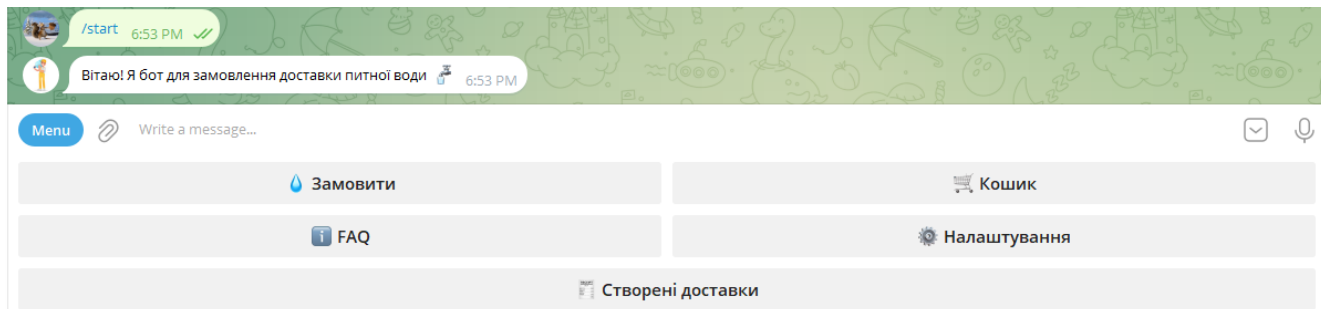


Рисунок 3.1 – Головне меню

Для створення замовлення користувач має бути зареєстрований в системі. Для цього він натискає на кнопку «Налаштування» обираємо місто доставки, вводить адресу та надаємо номер телефону. На рисунку 3.2 відображено реєстрацію користувача в системі.

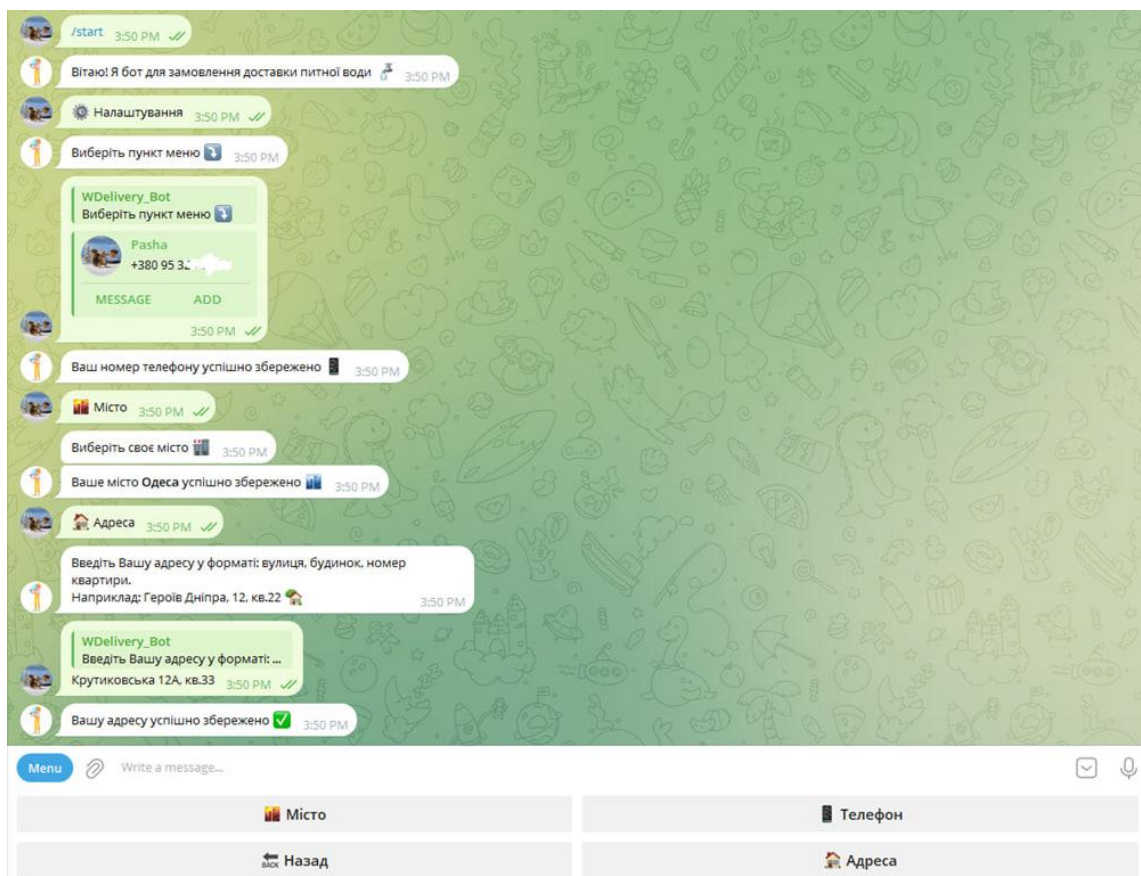


Рисунок 3.2 – Реєстрація користувача

Для перегляду інформації про створення доставки, оновлення даних, обмін тар, оплату та саму службу доставки, користувач натискає на кнопку «FAQ». Після чого користувачу надається можливість вибрати питання, яке його цікавить та отримати відповідь на нього. На рисунку 3.3 відображено перегляд відповідей на обрані питання.

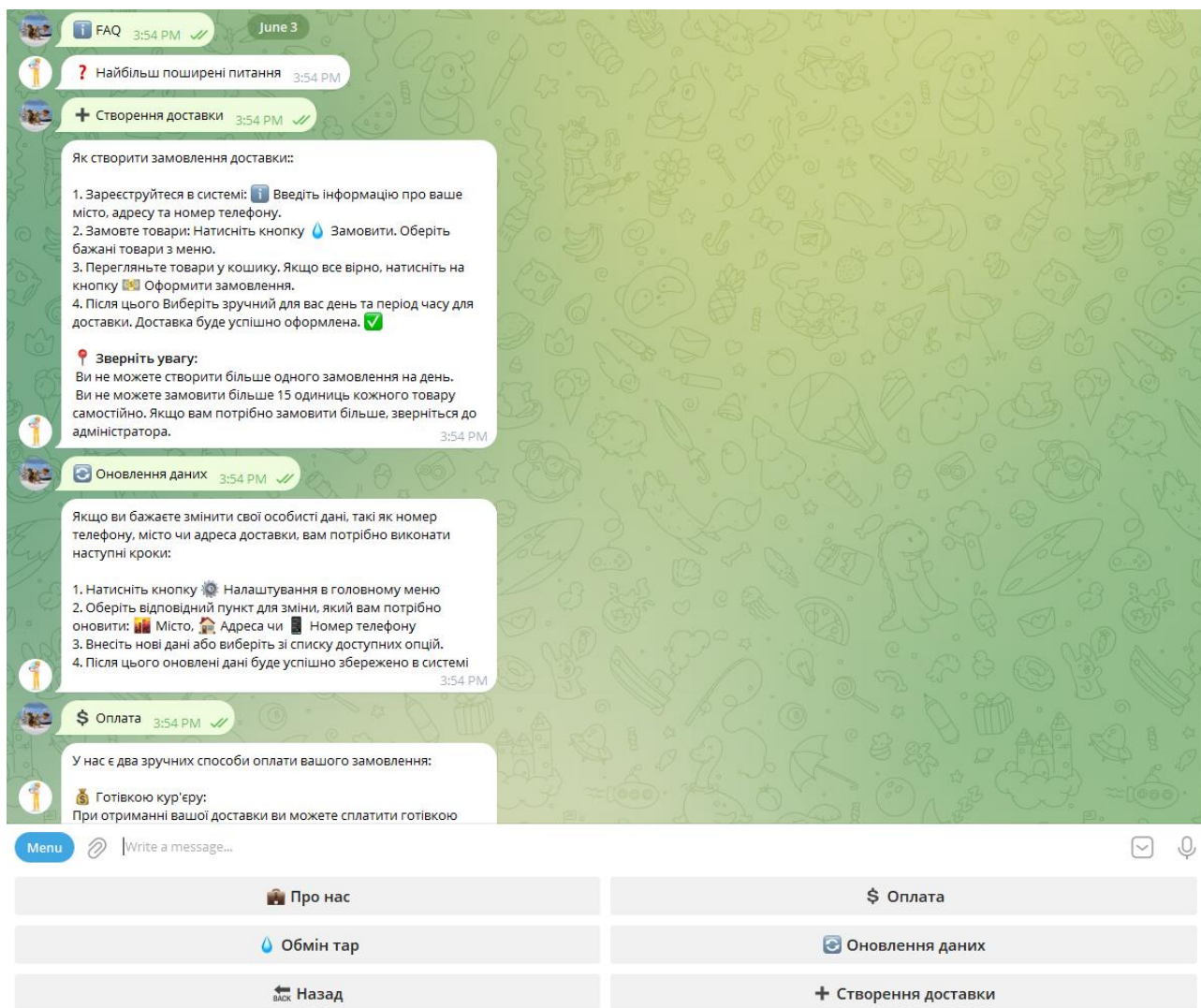


Рисунок 3.3 – Перегляд відповідей на найбільш поширені запитання

Для створення замовлення користувач має натиснути на кнопку «Замовити». Обрати бажаний товар зі списку доступних, літраж та ввести кількість товару на замовлення. Для користувача встановлено обмеження на максимальну кількість одиниць кожного товару 15 штук. Якщо користувач введе некоректні дані або перевищить ліміт, йому буде надіслано відповідне повідомлення. Якщо ж замовлення товару до кошика відбудеться успішно, висвітиться відповідне повідомлення. На рисунку 3.4 відображено додавання помпи до кошику.

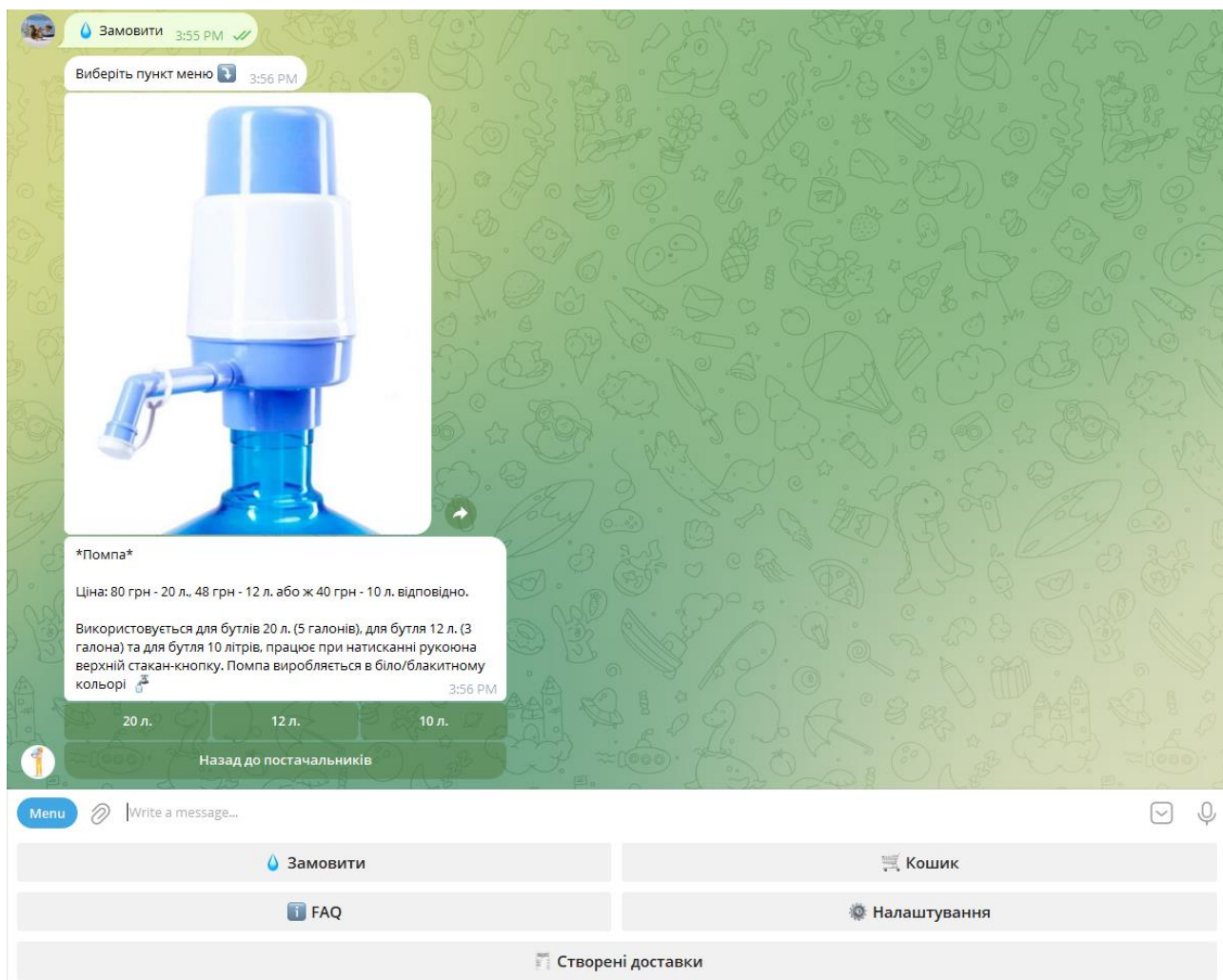


Рисунок 3.4 – Додавання помпи в кошик

Після додавання товару в кошик, користувачу видається меню з опцією додати інші товари. Користувач може натиснути на кнопку «Додати інший товар», обрати бажаний товар зі списку доступних, літраж та ввести кількість товару на замовлення. На рисунку 3.5 відображено додавання іншого товару до кошику.

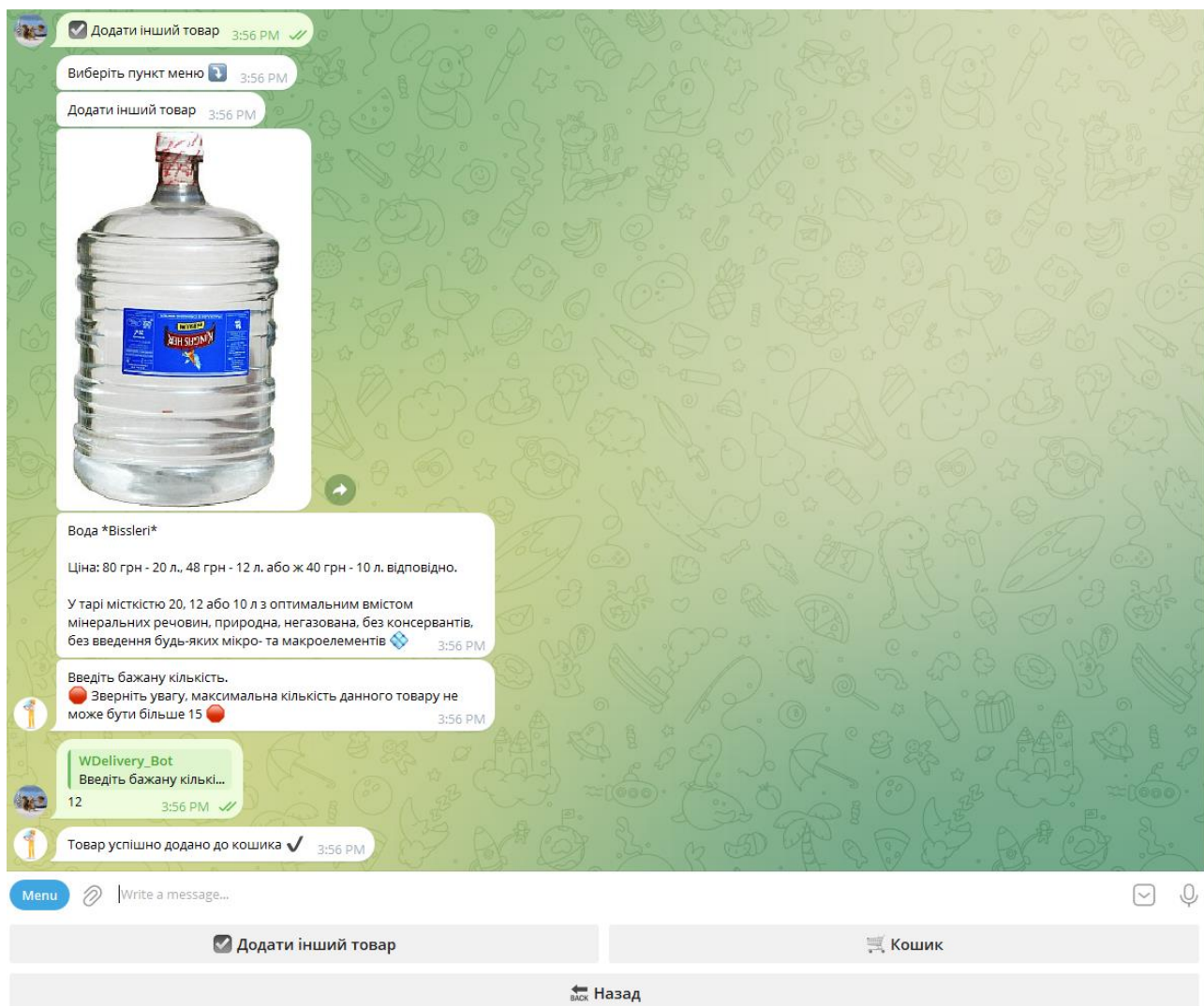


Рисунок 3.5 – Додавання іншого товару

Для перегляду товарів в кошику користувач в головному меню натискає на кнопку «Кошик». Після цього на екрані з'являється інформація з переліком товарів у кошику, їх кількістю, ціною за кожний товар та остаточну суму. На рисунку 3.6 відображено перегляд інформації про товари в кошику.

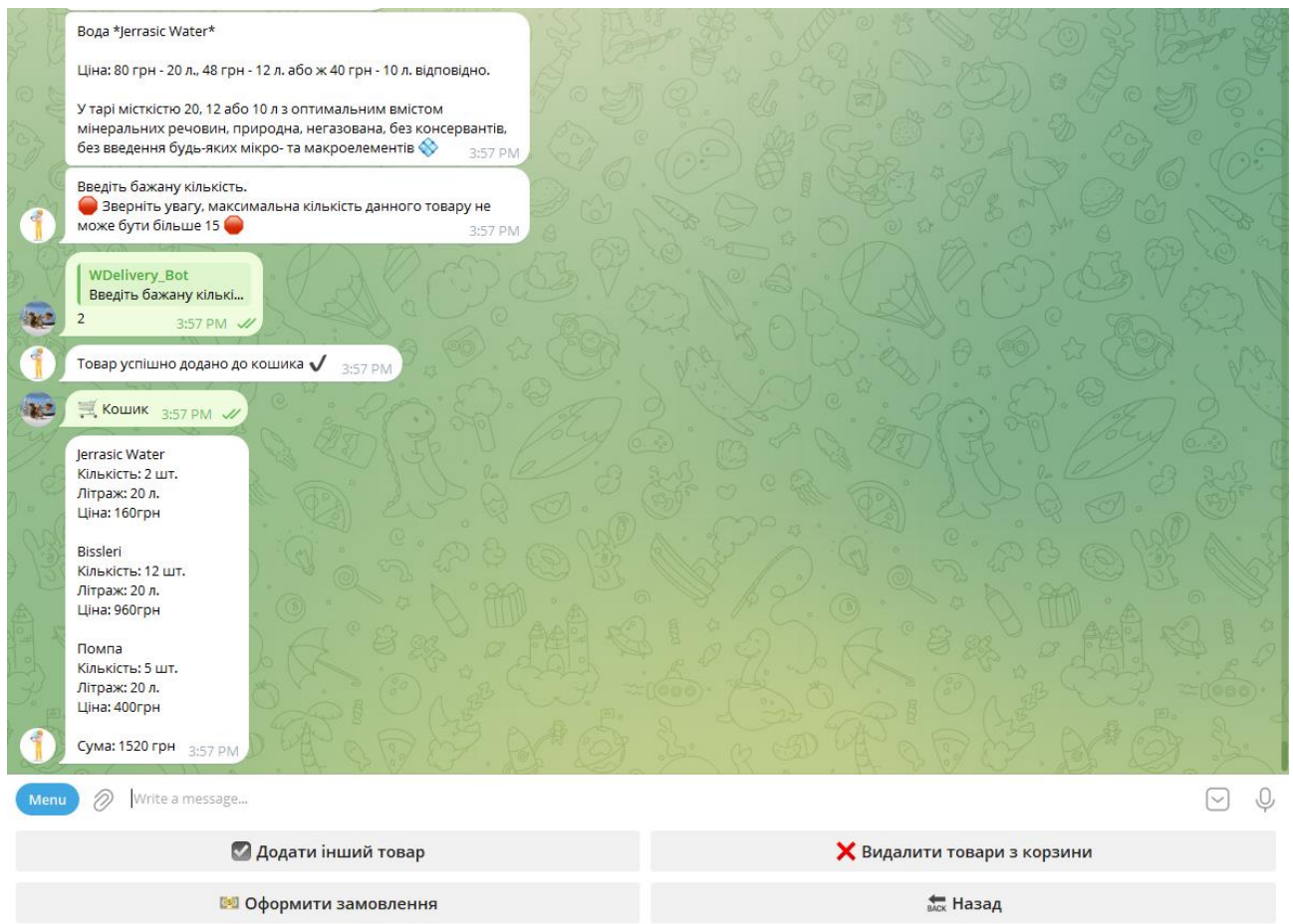


Рисунок 3.6 – Перегляд кошика

Із самого меню кошика користувач може видалити товари з кошика, оформити доставку або додати інший товар. Для оформлення доставки потрібно натиснути на кнопку «Оформити замовлення», після чого на екрані з’явиться повідомлення з вибором дати та часу відповідно. На рисунках 3.7-3.8 відображено оформлення доставки користувачем. Якщо ж користувач не обере потрібну дату або час, замовлення не буде створено і його поверне до головного меню.

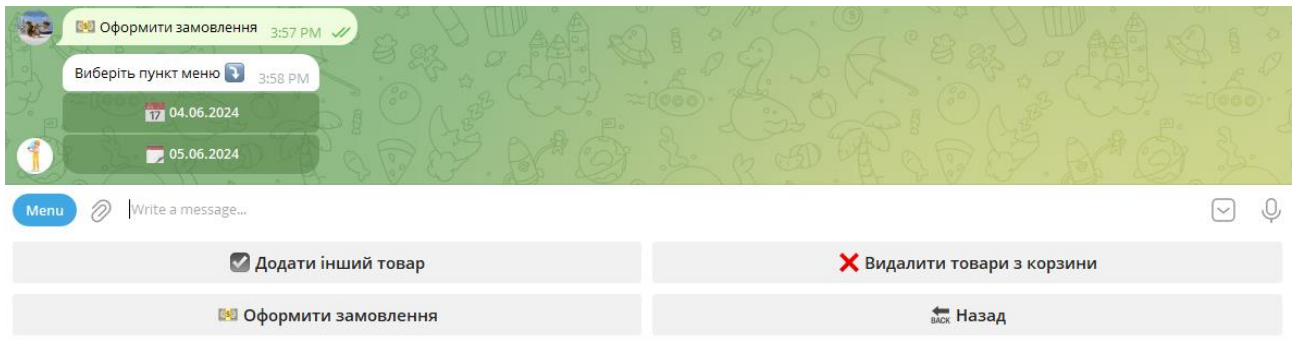


Рисунок 3.7 – Вибір дати доставки

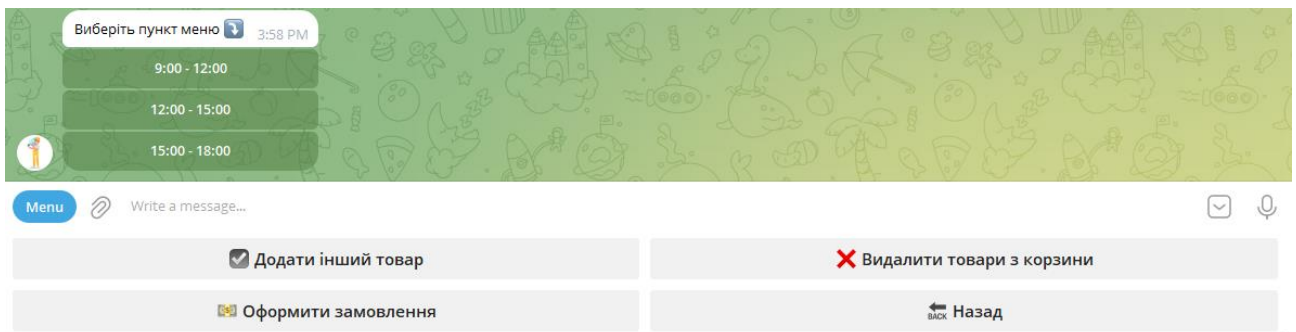


Рисунок 3.8 – Вибір часу доставки

Після оформлення доставки видається відповідне повідомлення і користувач перенаправляється в головне меню.

Для перегляду інформації про оформлені доставки потрібно натиснути на кнопку «Створені доставки», ввести потрібну дату, на яку було створено замовлення, після чого отримати відповідний результат. В інформації про доставку користувачу надається номер доставки, перелік замовлених товарів, дата доставки, час, адреса та місто доставки, номер телефону користувача і сума до сплати. Якщо ж користувач введе некоректні дані або дату, на яку не було оформлено доставку, йому висвітиться відповідне повідомлення. На рисунку 3.9 відображено перегляд інформації про доставку.

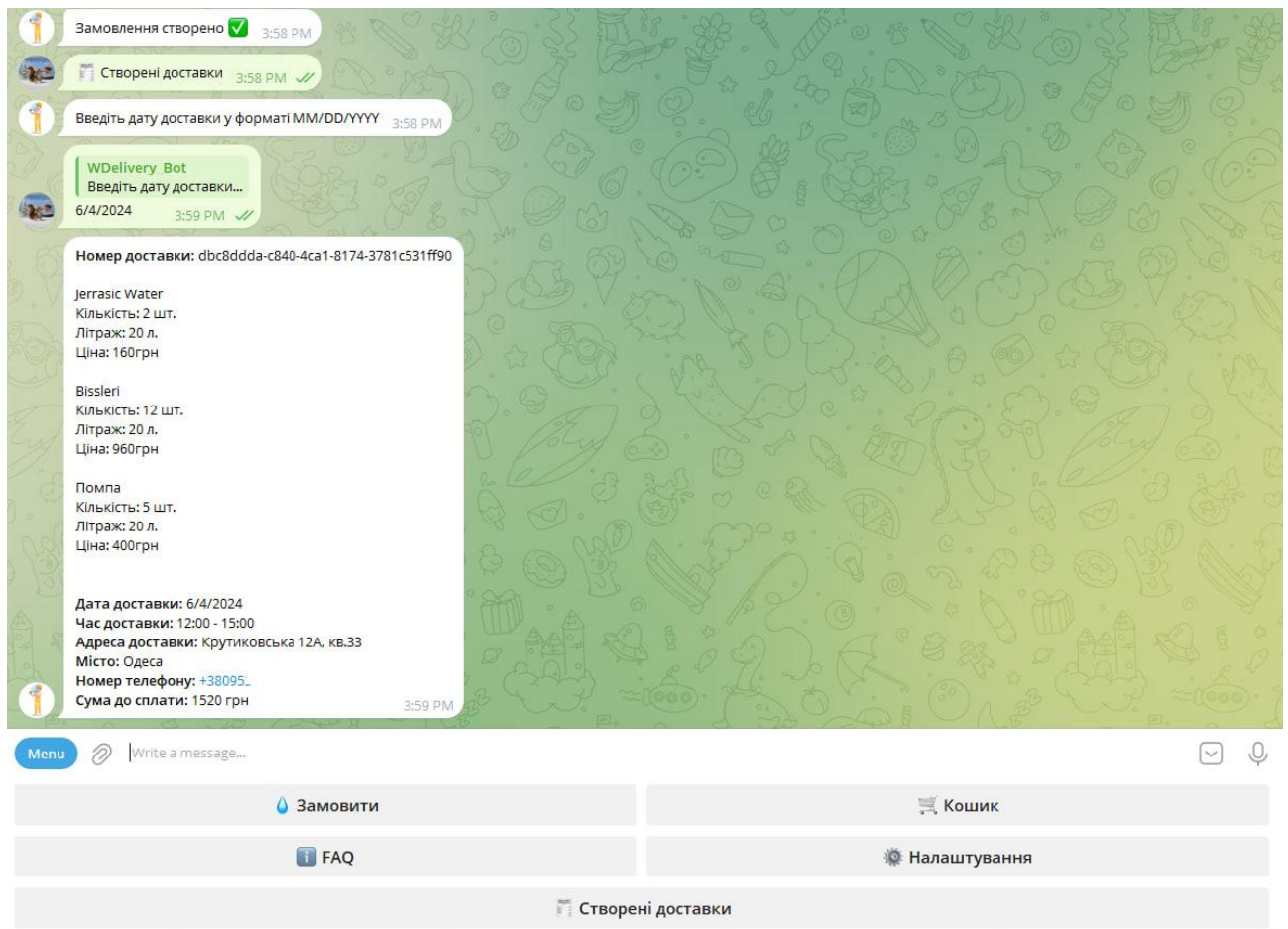


Рисунок 3.9 – Перегляд створеної доставки користувачем

Панель адміністратора значно відрізняється від панелі користувача. Адміністратор переглядає активні доставки за допомогою натискання кнопки «Активні доставки» та введення бажаної дати. В інформації про доставку адміністратору надається номер доставки, перелік замовлених товарів, дата доставки, час, адреса та місто доставки, номер телефону користувача, сума до сплати, а також Id користувача. Якщо ж адміністратор введе некоректні дані, йому висвітиться відповідне повідомлення. На рисунку 3.10 відображено перегляд активних доставок адміністратором.

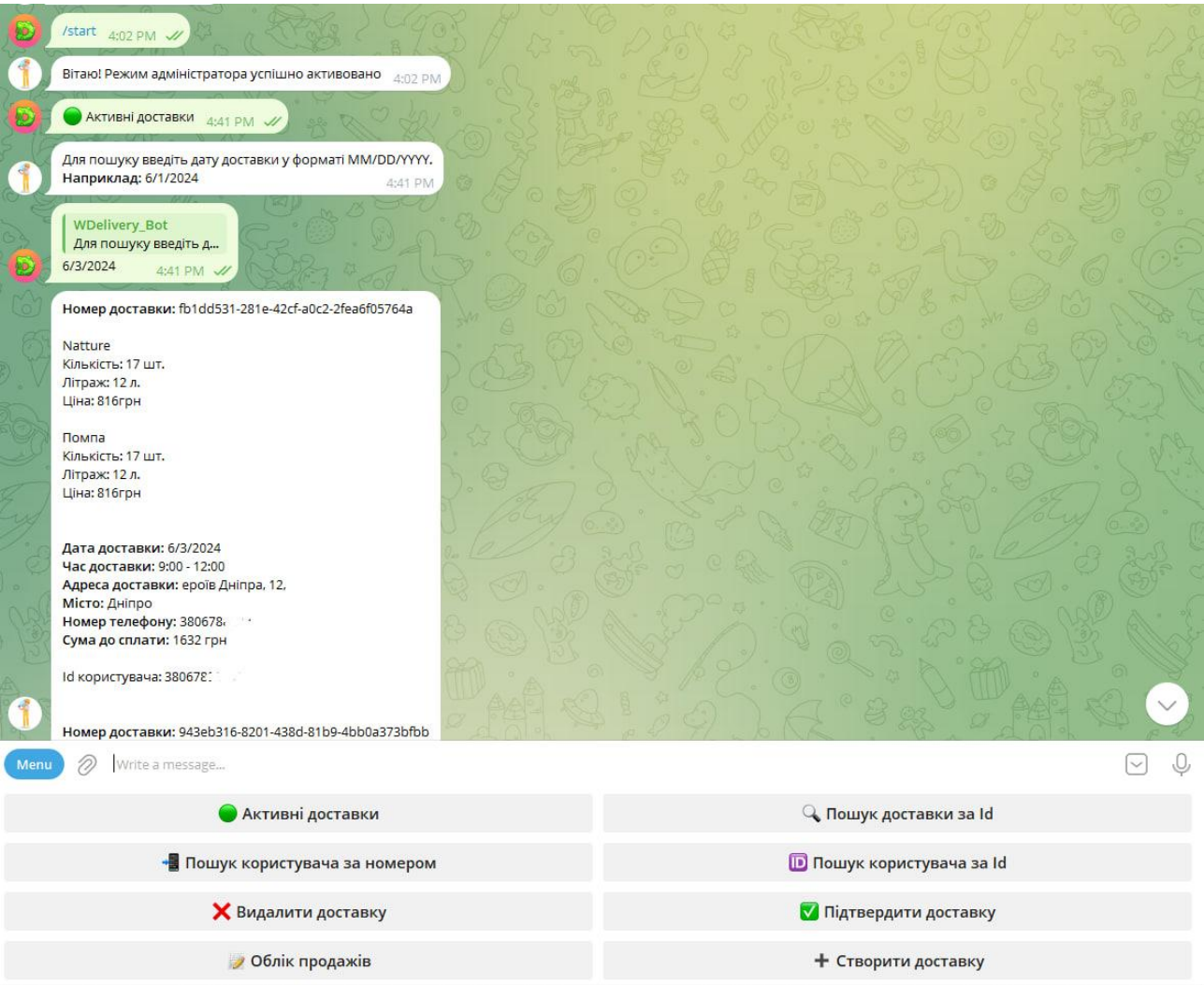


Рисунок 3.10 – Перегляд активних доставок

Для пошуку користувача за номером телефону або Id, потрібно натиснути на кнопку «Пошук користувача за номером» або «Пошук користувача за Id» відповідно та ввести бажані дані. Для прикладу початково було знайдено користувача за номером, а потім за Id. У відповідь адміністратору надається додатково місто та адреса користувача. Якщо ж адміністратор введе некоректні дані, йому висвітиться відповідне повідомлення. На рисунку 3.11 відображено пошук користувача за номером та Id.

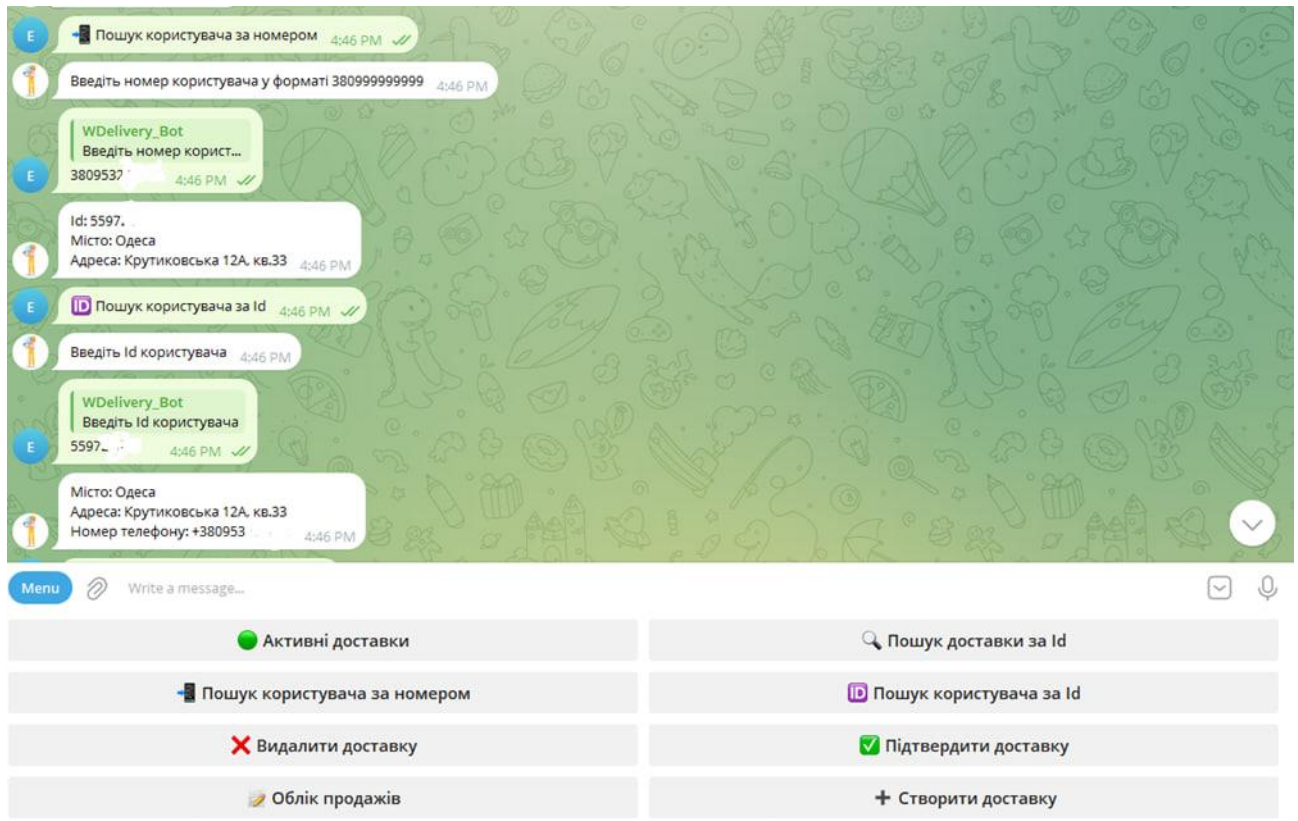


Рисунок 3.11 – Пошук користувача за номером телефону та Id

Для пошуку доставки за Id потрібно натиснути на відповідну кнопку «Пошук доставки за Id» з головного меню, після чого ввести Id доставки. Для прикладу було введено Id оформленої доставки, а саме dbc8ddda-c840-4ca1-8174-3781c531ff90 та успішно отримано відповідь. В інформації про доставку адміністратору надається номер доставки, перелік замовлених товарів, дата доставки, час, адреса та місто доставки, номер телефону користувача, сума до сплати, а також Id користувача. Якщо ж адміністратор введе некоректні дані, йому висвітиться відповідне повідомлення. На рисунку 3.12 відображено пошук доставки за Id.

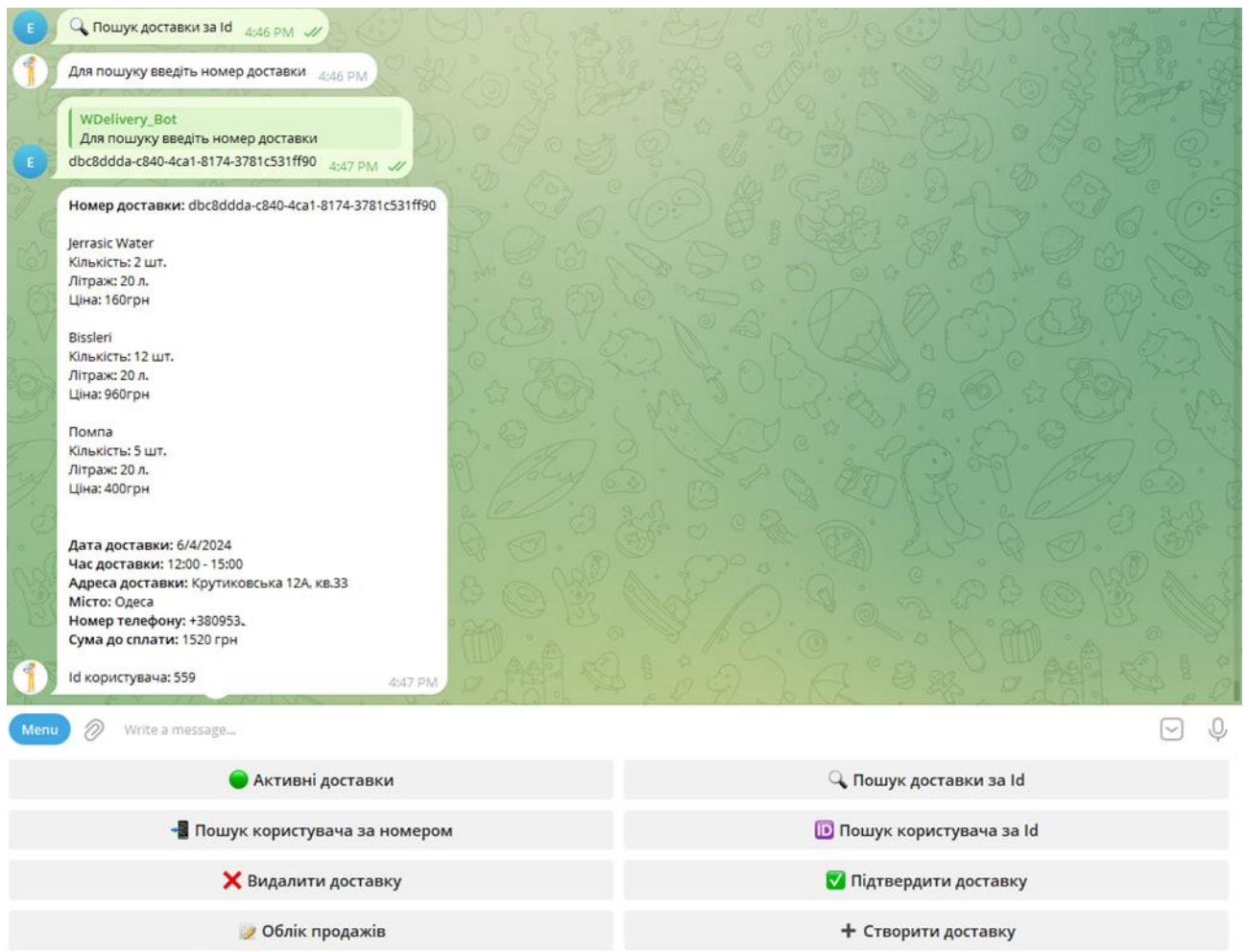


Рисунок 3.12 – Пошук доставки за Id

Для обліку продажів та перегляду інформації про виконані доставки, потрібно натиснути на кнопку «Облік продажів», після чого ввести місяць та дату для обліку. Початково виводиться короткий опис усіх виконаних доставок за обраний місяць, із вказаними Id номером доставки, датою та сумою. Після чого на основі виконаних доставок обробляється та виводиться результат про кількість доставок, прибуток компанії, та період, за який було проведено підрахунки. Якщо ж адміністратор введе некоректні дані, йому висвітиться відповідне повідомлення. На рисунку 3.13 відображено перегляд обліку продажів.

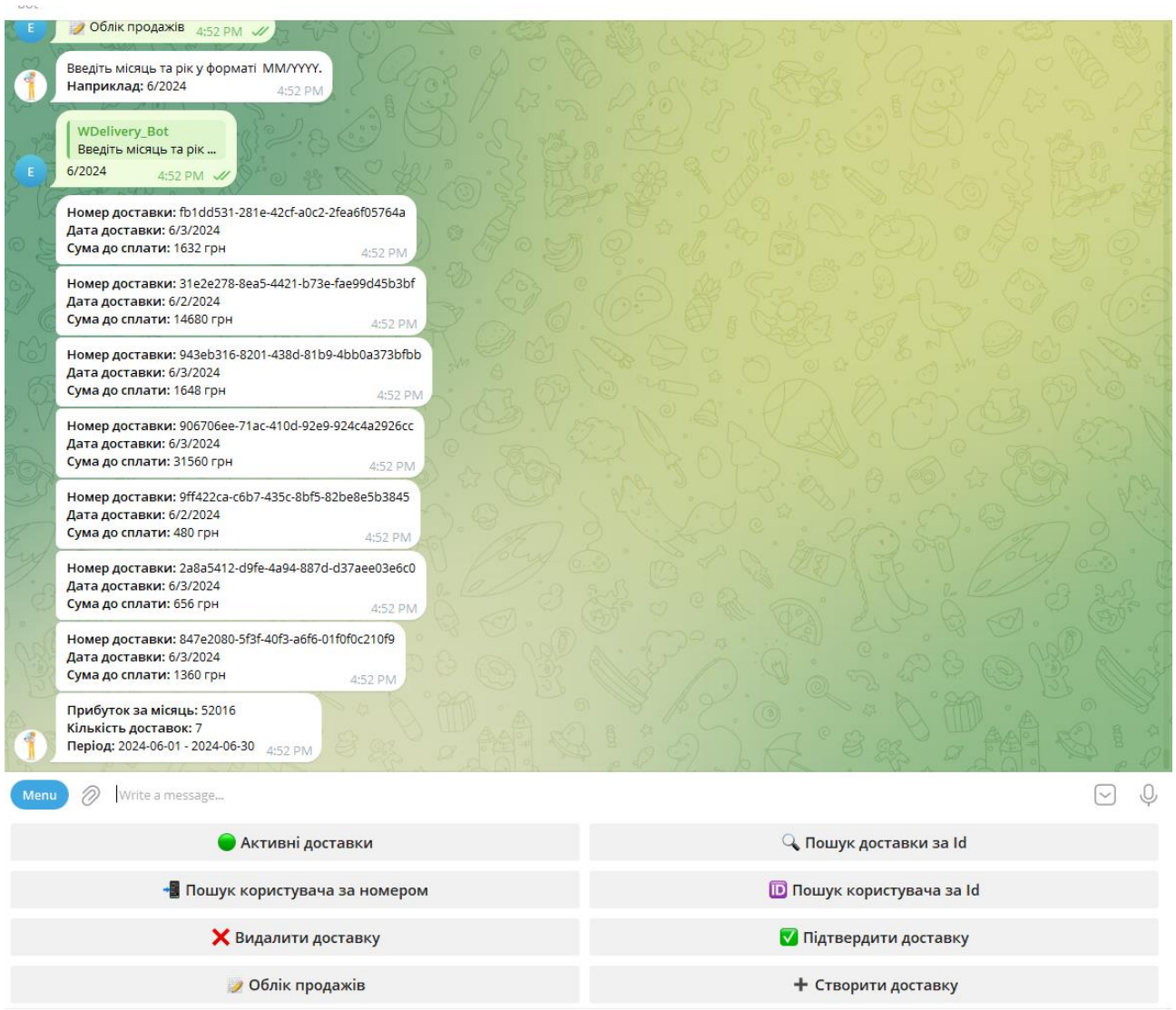


Рисунок 3.13 – Облік продажів

Також для адміністратора було додано функціонал створення доставок без обмежень, які є у користувачів. Адміністратор має натиснути на кнопку «Створити доставку», після чого його перекине в меню користувача і потрібно буде пройти усі кроки, які проходив користувач. Тобто початкова реєстрація, із вибором міста, введенням адреси та номеру телефону. Далі вибір самих товарів і відповідно оформлення доставки, з вибором дати та часу. Для прикладу було додано до замовлення 5 помп на 20 літрів кожна. Якщо ж адміністратор введе некоректні дані або надасть не повноцінні дані, йому висвітиться відповідне повідомлення. На рисунку 3.14 відображено

функціонал адміністратора, для створення доставки, який є повною копією меню користувача, проте без обмежень.

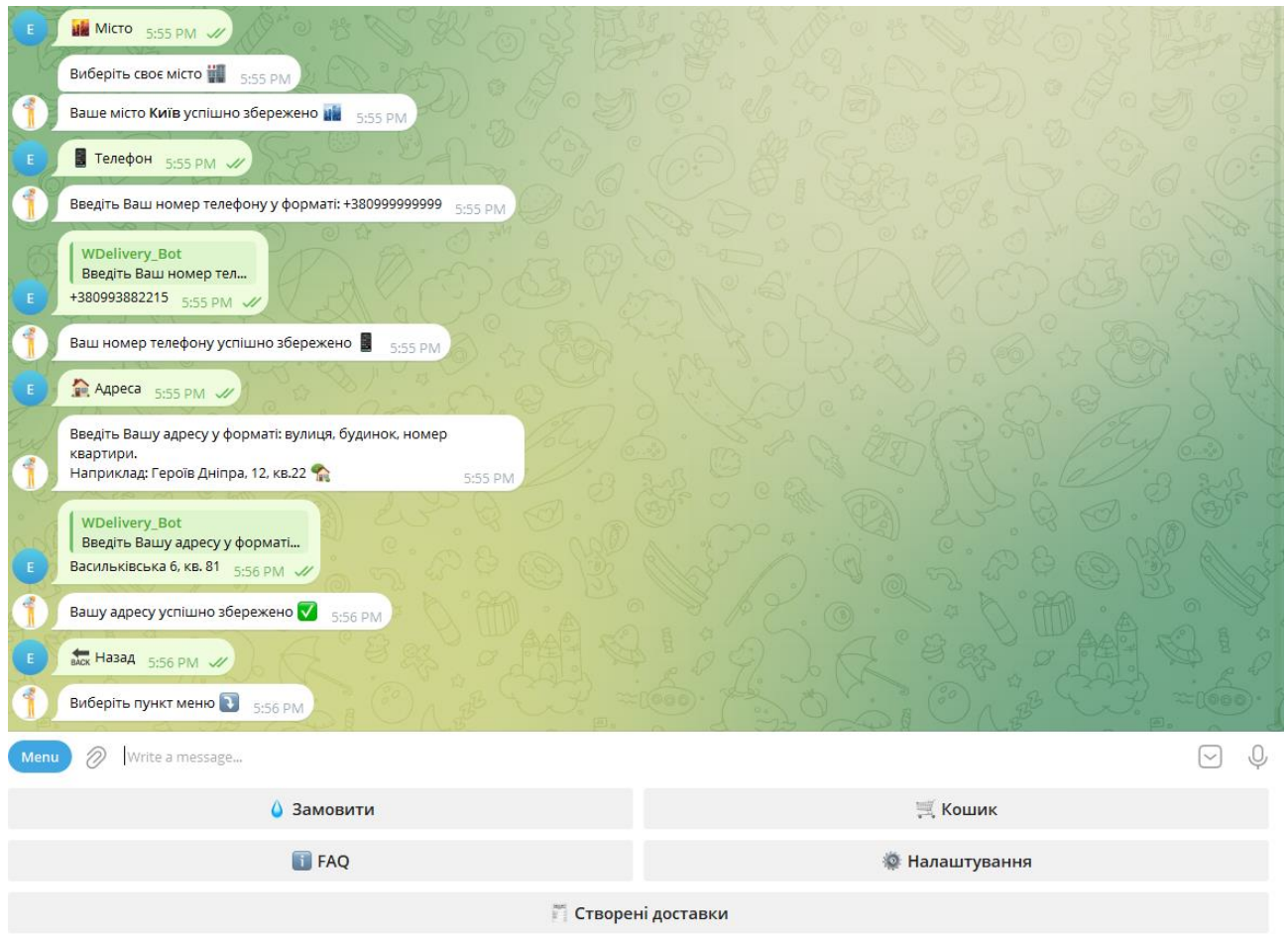


Рисунок 3.14 – Створення доставки адміністратором

Для видалення доставки з системи, потрібно натиснути на «Видалити доставку» та ввести бажаний номер доставки. Для прикладу було взято один з номерів створених доставок та видалено його з системи. Якщо ж адміністратор введе некоректні дані, йому висвітиться відповідне повідомлення. На рисунку 3.15 відображено видалення доставки з системи.

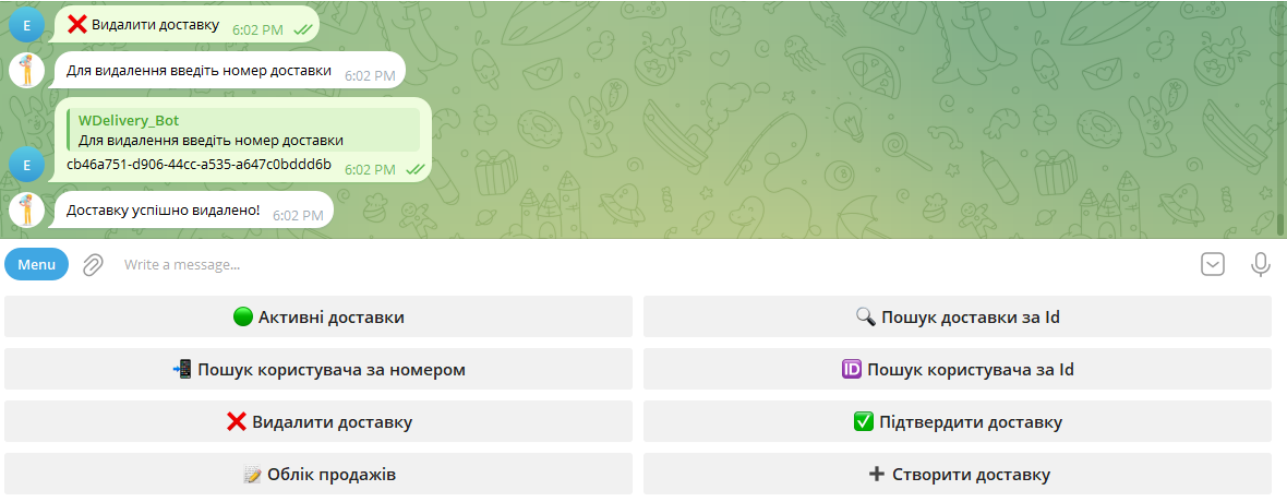


Рисунок 3.15 – Видалення доставки з системи

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2024 р.

**ВЕБЗАСТОСУНОК ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ДІЯЛЬНОСТІ
СЛУЖБИ ДОСТАВКИ ВОДИ**

Графічний матеріал

КПІ.ІТ-0411.045440.06.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олег ЛІСОВИЧЕНКО

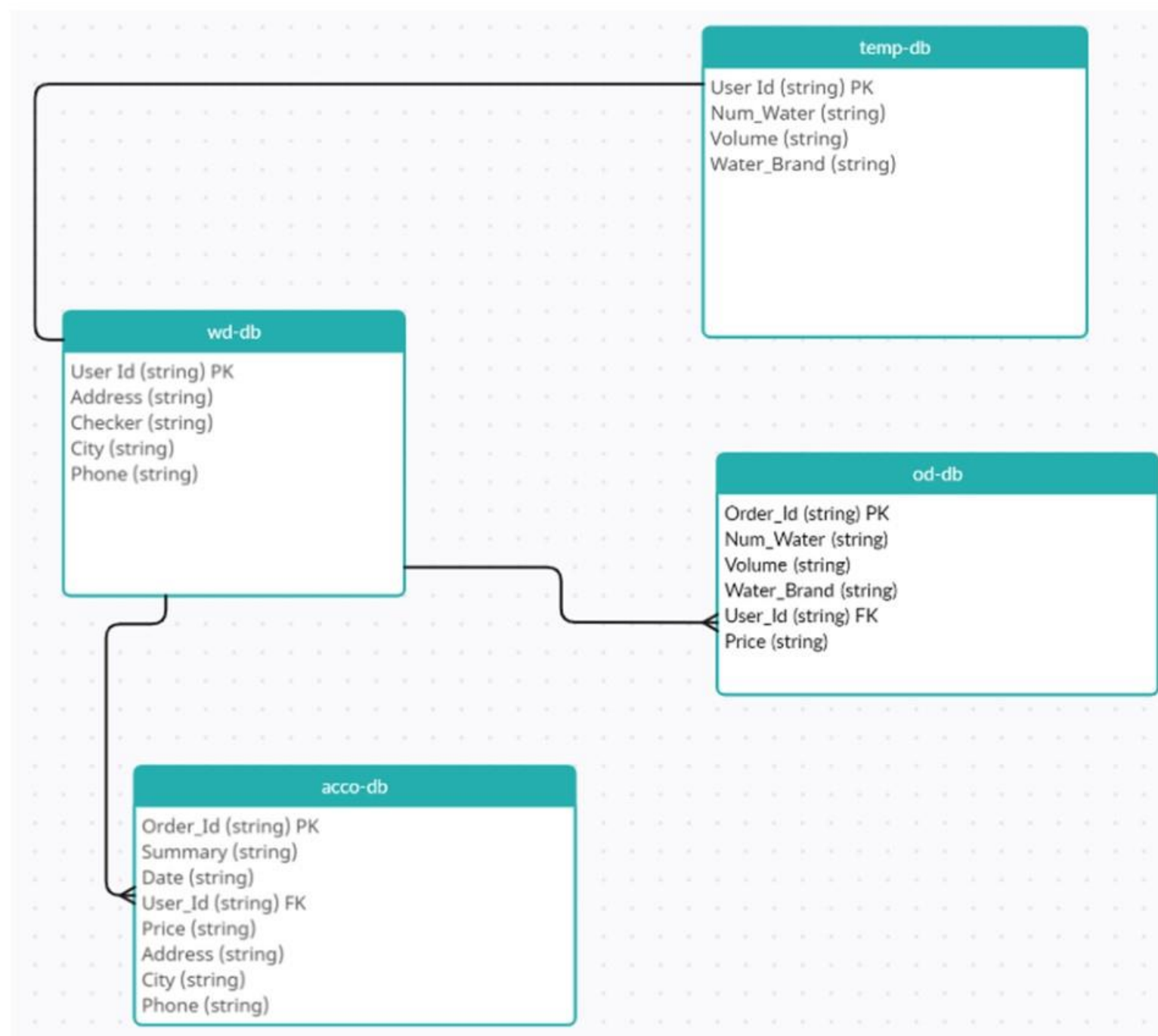
Нормоконтроль:

_____ Павло РОДІОНОВ

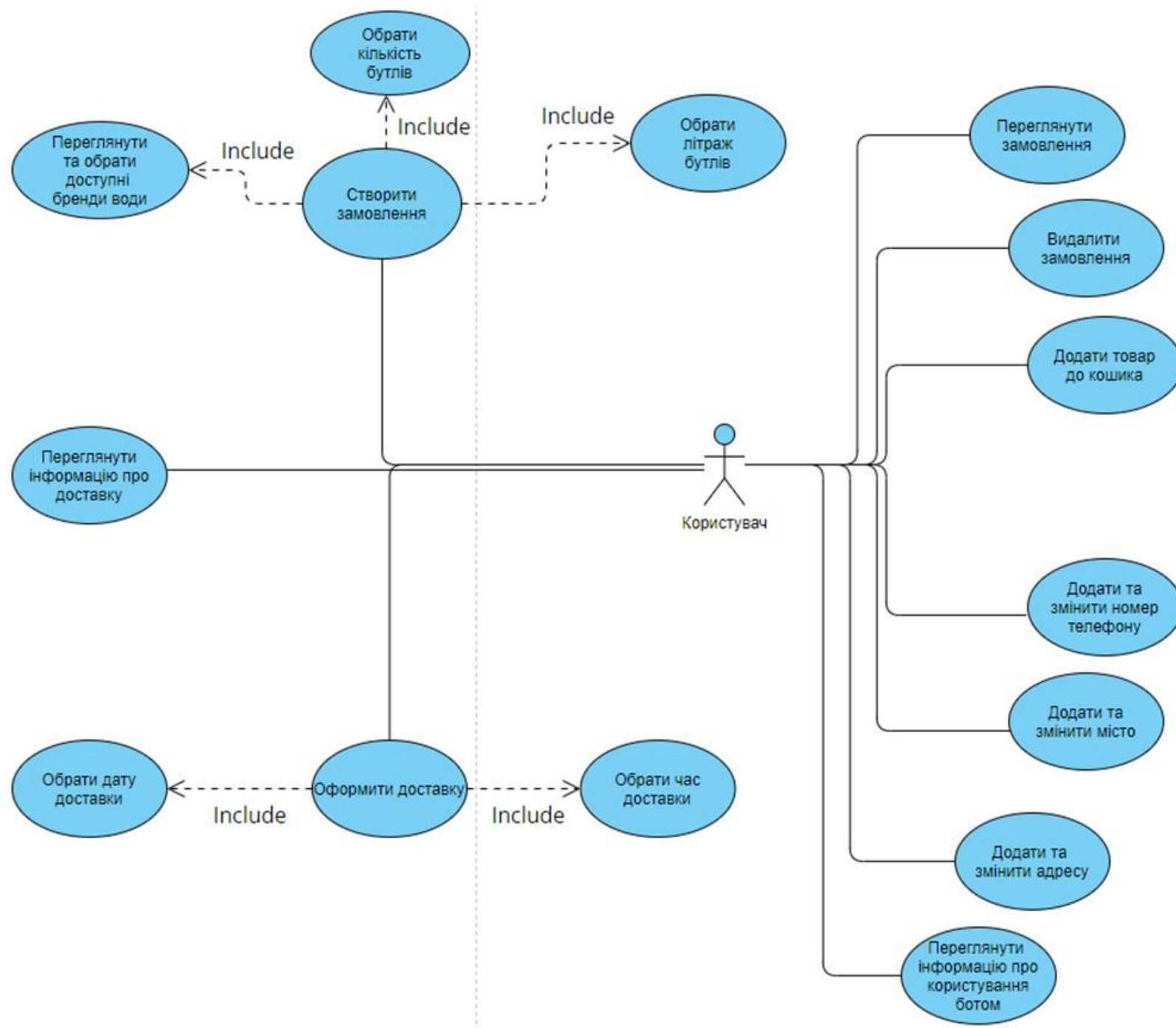
Виконавець:

_____ Павло МАРТИШЕНКО

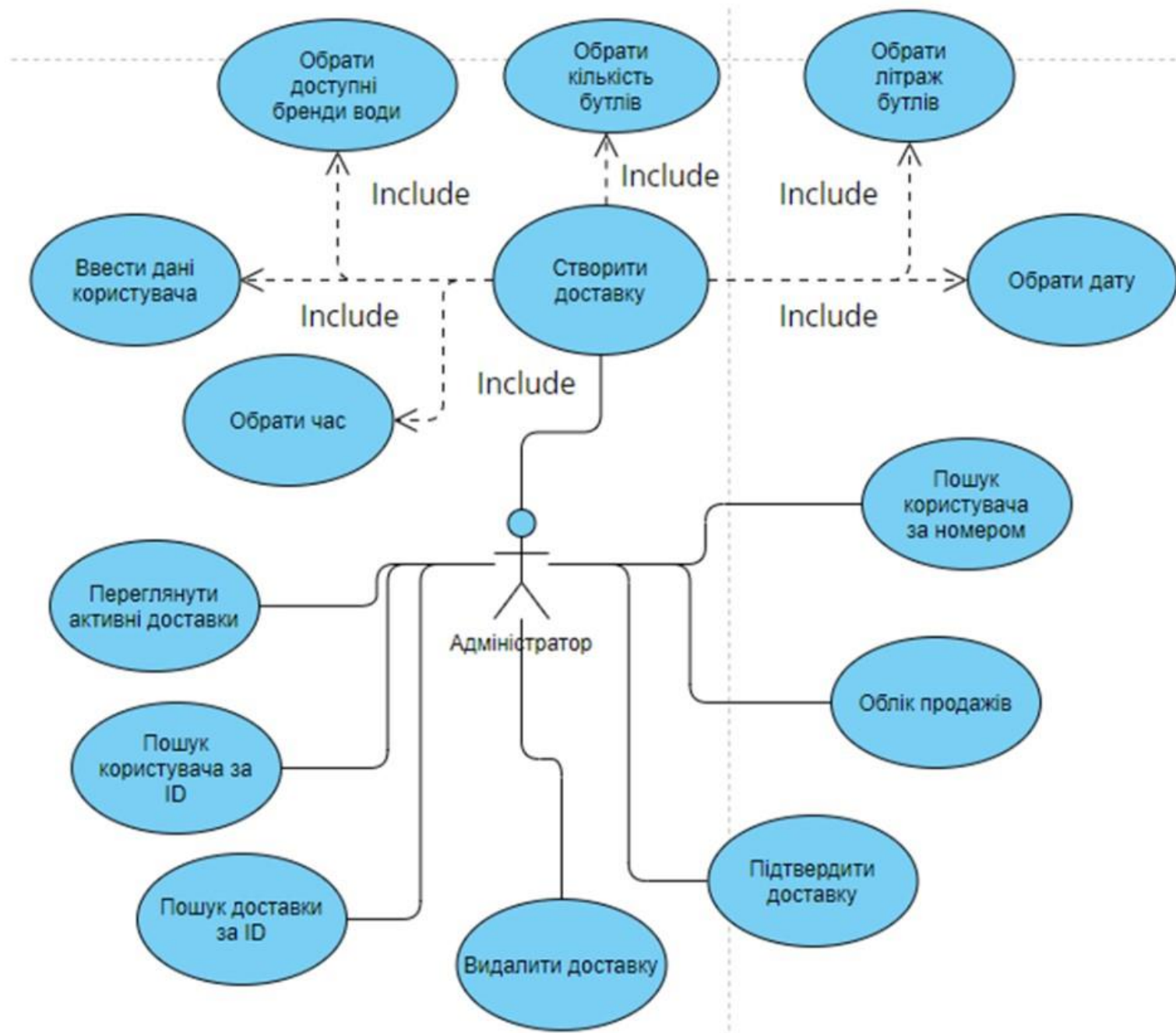
Київ – 2024



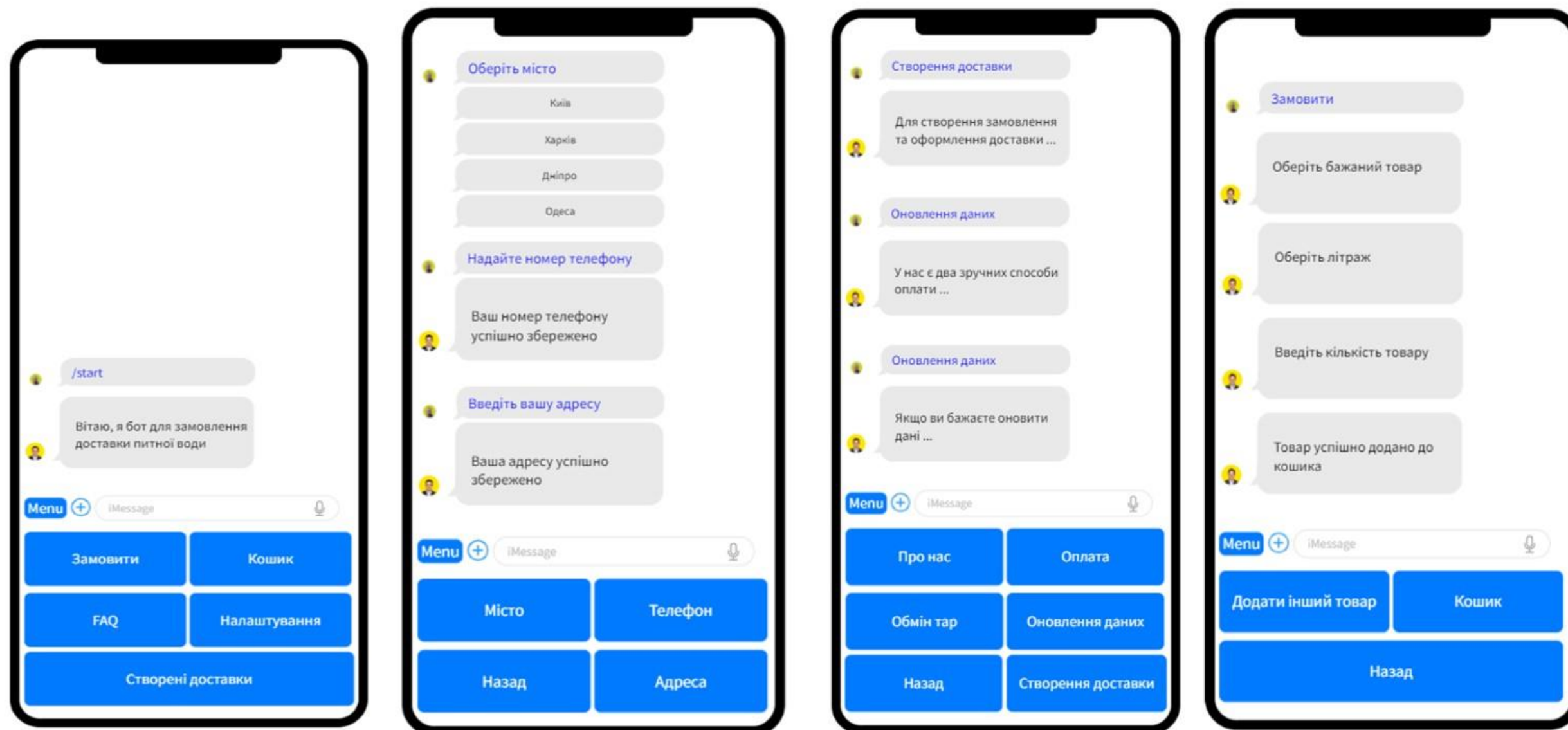
						КПІ.ІТ-0411.045440.06.99.СБД			
						Схема бази даних	Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата					
Розробив		Мартищенко П.С.							
Перевішив		Лісовиченко О.І.							
Т. контр.						Вебзастосунок підвищення ефективності діяльності служби доставки води	Аркуш		Аркушів
Н. контр.		Родіонов П.Ю.					КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-04		
Затвердив		Жаріков Е.В.							



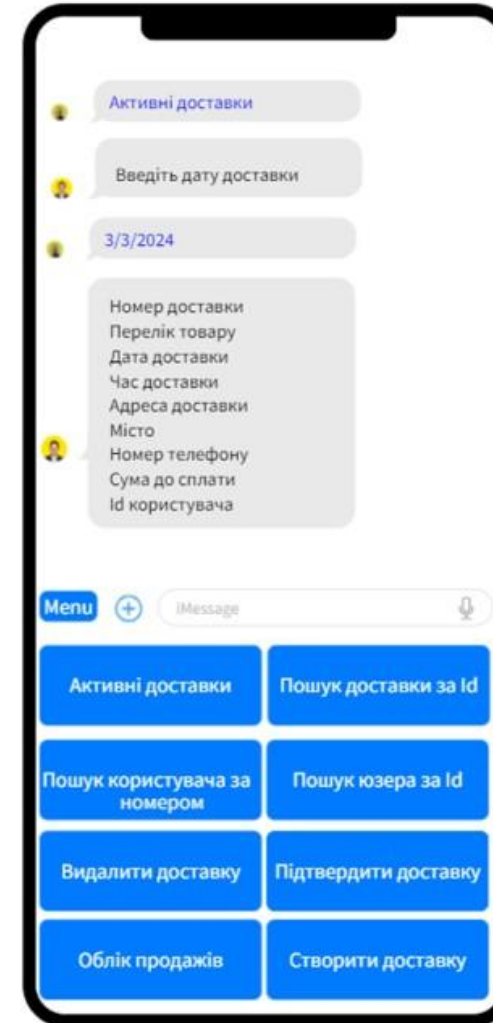
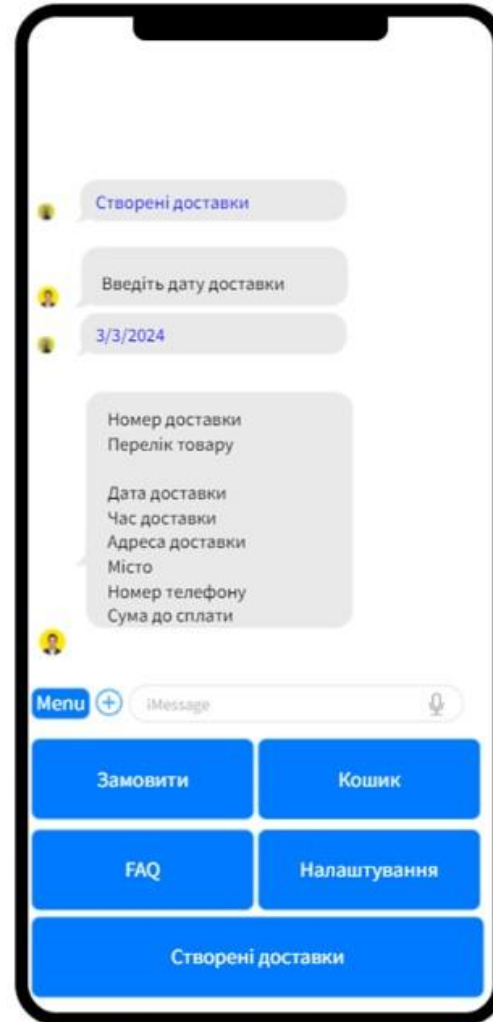
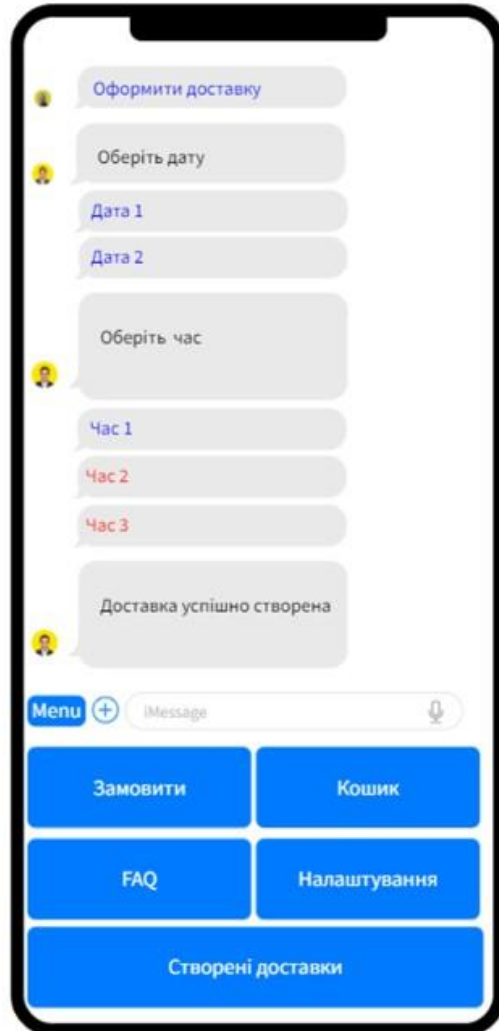
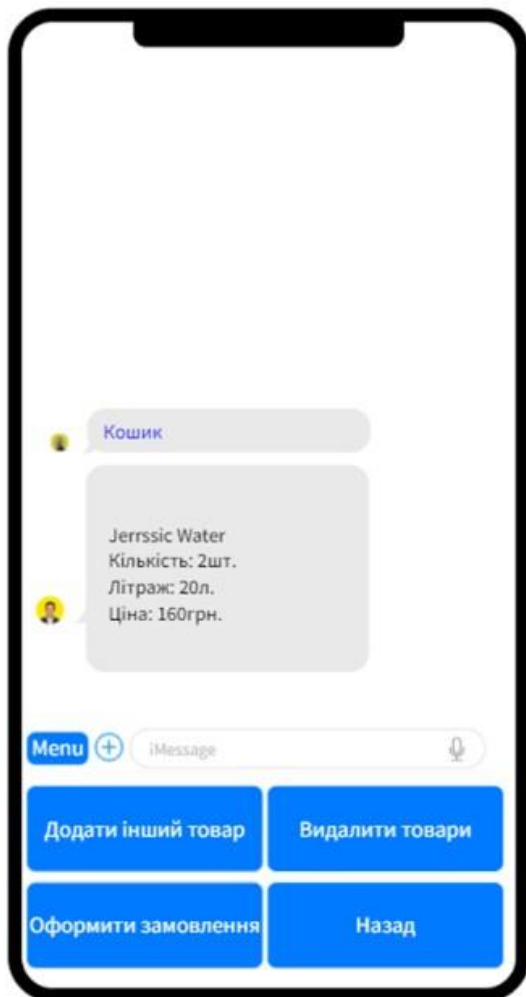
					КПІ.ІТ-0411.045440.06.99.CCB						
					Діаграма варіантів використання користувача	Літера		Маса		Масштаб	
Зм.	Арк.	№ документа	Підпис	Дата							
Розробив		Мартищенко П.С.									
Перевішив		Лісовиченко О.І.									
Т. контр.											
					Вебзастосунок підвищення ефективності діяльності служби доставки води	Аркуш		Аркушів			
Н. контр.		Родіонов П.Ю.				КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-04					
Затвердив		Жаріков Е.В.									



					КПІ.ІТ-0411.045440.06.99.CCB						
					Діаграма варіантів використання адміністратора	Літера		Маса		Масштаб	
Зм.	Арк.	№ документа	Підпис	Дата							
Розробив		Мартищенко П.С.									
Перевішив		Лісовиченко О.І.									
Т. контр.											
					Вебзастосунок підвищення ефективності діяльності служби доставки води	Аркуш		Аркушів			
Н. контр.		Родіонов П.Ю.				КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-04					
Затвердив		Жаріков Е.В.									



					КП.ІТ-0411.045440.06.99.КЕ				
					Креслення вигляду екранних форм	Літера		Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата					
Розробив	Мартищенко П.С.								
Перевішив	Лісовиченко О.І.								
Т. контр.						Аркуш		Аркушів	
					Вебзастосунок підвищення ефективності діяльності служби доставки води	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-04			
Н. контр.	Родіонов П.Ю.								
Затвердив	Жаріков Е.В.								



						КПІ.ІТ-0411.045440.06.99.КЕ						
						Креслення вигляду екранних форм	Літера		Маса		Масштаб	
Зм.	Арк.	№ документа	Підпис	Дата								
Розробив		Мартищенко П.С.										
Перевішив		Лісовиченко О.І.										
Т. контр.												
							Аркуш		Аркушів			
Н. контр.		Родіонов П.Ю.				Вебзастосунок підвищення ефективності діяльності служби доставки води	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-04					
Затвердив		Жаріков Е.В.										

Пошук доставки за Іd

Введіть номер доставки

1234-ssss-1222-sssq-2222

Номер доставки
Перелік товару
Дата доставки
Час доставки
Адреса доставки
Місто
Номер телефону
Сума до сплати
Іd користувача

Menu + iMessage

Активні доставки Пошук доставки за Іd

Пошук користувача за номером Пошук юзера за Іd

Видалити доставку Підтвердити доставку

Облік продажів Створити доставку

Пошук користувача за номером

Введіть номер

380999999999

Іd користувача
Місто
Адреса

Пошук користувача за Іd

Введіть Іd користувача

Номер користувача
Місто
Адреса

Menu + iMessage

Активні доставки Пошук доставки за Іd

Пошук користувача за номером Пошук юзера за Іd

Видалити доставку Підтвердити доставку

Облік продажів Створити доставку

Облік продажів

Введіть місяць та рік

3/2024

Номер доставки
Дата доставки
Сума до сплати

Номер доставки
Дата доставки
Сума до сплати

Прибуток за місяць
Кількість доставок
Період

Menu + iMessage

Активні доставки Пошук доставки за Іd

Пошук користувача за номером Пошук юзера за Іd

Видалити доставку Підтвердити доставку

Облік продажів Створити доставку

					КПІ.ІТ-0411.045440.06.99.КЕ				
					Креслення вигляду екранних форм	Літера		Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата					
Розробив		Мартишенко П.С.							
Перевішив		Лісовиченко О.І.							
Т. контр.						Аркуш		Аркушів	
Н. контр.		Родіонов П.Ю.			Вебзастосунок підвищення ефективності діяльності служби доставки води	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-04			
Затвердив		Жаріков Е.В.							