ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА
Навчально-науковий інститут енергетичної, інформаційної
та транспортної інфраструктури

Кафедра комп'ютерних наук та інформаційних технологій

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Розробка та впровадження Telegram-бота для обслуговування клієнтів кав'ярні»

Виконав: здобувач вищої освіти,
групи КН 2021-1
спеціальності
122 Комп'ютерні науки
(шифр і назва спеціальності)

Рубен МАРТІРОСЯН
(ім'я та прізвище)

Керівник: Наталія БРАТЕРСЬКА
(ім'я та прізвище)

Рецензент Марина НОВОЖИЛОВА
(ім'я та прізвище)

м. Харків – 2025 рік

Харківський національний університет міського господарства імені О. М. Бекетова
(повне найменування закладу вищої освіти)
Навчально-науковий Інститут енергетичної, інформаційної та транспортної інфраструктури
Кафедра комп'ютерних наук та інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 Комп'ютерні науки
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КНтаІТ

Марина НОВОЖИЛОВА
«23» 06 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Мартіросяна Рубена Кареновича
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка та впровадження Telegram-бота для обслуговування клієнтів кав'ярні»

керівник роботи Братерська Наталія Миколаївна, асистент кафедри КН та ІТ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від 09 травня 2025 р. №341-03
2. Термін подання студентом роботи 23 червня 2025 р.
3. Вихідні дані до роботи: проєктування та реалізація Telegram-бота для обслуговування клієнтів кав'ярні з використанням Python.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): 1. Аналітична частина 2. Інформаційно-математична частина 3. Програмно-технічна частина 4. Охорона праці

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):

Презентація – 15 слайдів

6. Консультанти розділів роботи

Розділ	Ім’я та Прізвище, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Наталія БРАТЕРСЬКА, асистент кафедри КН та ІТ	12.05.2025	20.05.2025
2	Наталія БРАТЕРСЬКА, асистент кафедри КН та ІТ	20.05.2025	28.05.2025
3	Наталія БРАТЕРСЬКА, асистент кафедри КН та ІТ	28.05.2025	05.06.2025
4	Вікторія МАЛИШЕВА, к. т. н., доцент кафедри ОП та БЖ	09.06.2025	14.06.2025

7. Дата видачі завдання 12 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми дипломної роботи	05.05.2025	виконано
2	Затвердження тем, наукових керівників, завдань та календарного плану підготовки кваліфікаційної роботи	09.05.2025	виконано
3	Написання І розділу	20.05.2025	виконано
4	Написання ІІ розділу	28.05.2025	виконано
5	Написання ІІІ розділу	05.06.2025	виконано
6	Написання IV розділу	14.06.2025	виконано
7	Подання дипломної роботи керівнику	16.06.2025	виконано
8	Робота по усуненню зауважень керівника, уточнення і доповнення практичного матеріалу, оформлення додатків до роботи	18.06.2025	виконано
9	Подання доопрацьованого варіанту роботи керівнику	20.06.2025	виконано
10	Захист матеріалів дипломної роботи на засіданні кафедри	21.06.2025	виконано
11	Офіційний захист матеріалів дипломної роботи на засіданні екзаменаційної комісії	24.06.2025	виконано

Студент _____ (підпис) Рубен МАРТІРОСЯН (прізвище та ініціали)

Керівник роботи _____ (підпис) Наталія БРАТЕРСЬКА (прізвище та ініціали)

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка кваліфікаційної роботи бакалавра студента групи КН 2021-1 спеціальності 122 Комп’ютерні науки Мартіросяна Рубена Кареновича за темою «Розробка та впровадження Telegram-бота для обслуговування клієнтів кав'ярні» складається з 4 розділів, містить 114 рисунків, 15 джерел та 2 додатки.

Кваліфікаційну роботу бакалавра присвячено розробці та впровадженню Telegram-бота для обслуговування клієнтів кав'ярні.

У першому розділі «Загальні положення» розглянуто особливості предметної області, обґрунтовано актуальність тематики, проаналізовано існуючі аналоги та визначено мету і завдання розробки Telegram-бота для автоматизації замовлень у кав’ярні.

Другий розділ містить опис інформаційного та математичного забезпечення системи, що включає структуру та логічну схему взаємодії користувача з ботом, а також модель бонусної системи лояльності.

У розділі «Програмне та технічне забезпечення» описано практичне застосування мови Python та бібліотеки Aiogram для реалізації Telegram-бота, включаючи структуру коду, механізми керування станами, логіку основних функцій – від реєстрації користувача до оформлення замовлення та обробки оплати.

Розділ «Охорона праці» присвячено аналізу потенційних ризиків для працівників кав’ярні, нормативно-правовим актам у сфері охорони праці, а також заходам щодо забезпечення безпеки персоналу.

Ключові слова: TELEGRAM-БОТ, ЧАТ-БОТ, КАВ’ЯРНЯ, КАВА, КЛІЄНТ, КОРИСТУВАЧ, ЗАМОВЛЕННЯ

ANNOTATION

Structure and Scope of the Work. The explanatory note of the bachelor's qualification paper by Martirosian Ruben, a student of group KN 2021-1, specialty 122 Computer Science, on the topic "Development and Implementation of a Telegram Bot for Coffee Shop Customer Service" consists of 4 sections, contains 114 figures, 15 sources, and 2 appendices.

The bachelor's qualification paper is dedicated to the development and implementation of a Telegram bot for coffee shop customer service.

The first section, "General Provisions," examines the specifics of the subject area, substantiates the relevance of the topic, analyzes existing counterparts, and defines the goal and objectives of developing a Telegram bot for automating orders in a coffee shop.

The second section describes the information and mathematical support of the system, which includes the structure and logical interaction scheme between the user and the bot, as well as a bonus loyalty system model.

The "Software and Hardware" section describes the practical application of Python and the Aiogram library for implementing the Telegram bot, including code structure, state management mechanisms, and the logic of key functions – from user registration to order placement and payment processing.

The "Labor Protection" section is dedicated to analyzing potential risks for coffee shop employees, regulatory legal acts in the field of labor protection, and measures to ensure personnel safety.

Keywords: TELEGRAM BOT, CHATBOT, COFFEE SHOP, COFFEE, CLIENT, USER, ORDER

ЗМІСТ

ВСТУП

В умовах стрімкого розвитку цифрових технологій і зростання вимог до якості обслуговування клієнтів особливої актуальності набуває впровадження сучасних інформаційних технологій у сферу малого та середнього бізнесу. Одним із найперспективніших напрямів впровадження цих технологій є використання інтелектуальних «помічників», а саме – чат-ботів, які дають змогу автоматизувати рутинні процеси взаємодії з клієнтами, поліпшити користувацький досвід і підвищити ефективність обслуговування.

Особливого значення такі технології набувають у сфері громадського харчування, де оперативність, точність і зручність взаємодії з клієнтом безпосередньо впливають на конкурентоспроможність підприємства. Кав'ярні, як один із найпопулярніших форматів закладів швидкого обслуговування, особливо зацікавлені в інструментах, що здатні поліпшити процес обробки замовлень, а також зібрати зворотний зв'язок для аналізу якості сервісу.

Однією з найпопулярніших платформ для розробки та використання чат-ботів на сьогоднішній день є месенджер Telegram. Завдяки відкритому API, широкій призначеній для користувача базі та підтримці безлічі функціональних можливостей, Telegram є зручним і доступним середовищем для створення автоматизованих помічників, які легко інтегруються в повсякденні процеси.

Метою даної кваліфікаційної роботи є розробка та впровадження Telegram-бота, призначеного для обслуговування клієнтів кав'ярні. У рамках проєкту передбачається створення програмного рішення, здатного надавати користувачеві інформацію про меню, приймати попередні замовлення, інформувати про акції та спеціальні пропозиції, а також збирати відгуки про якість обслуговування.

У рамках цієї роботи об'єктом виступає система організації обслуговування клієнтів у закладах громадського харчування, а саме – у контексті роботи сучасної кав'ярні.

Предметом дослідження є програмне забезпечення на базі месенджера Telegram, призначене для автоматизації ключових бізнес-процесів: представлення асортименту продукції, оформлення та оплати замовлень, отримання клієнтських відгуків, а також реалізація механізму лояльності.

Завдання дослідження включають:

- аналіз наявних рішень і технологій у сфері чат-ботів;
- розробку архітектури програмного продукту:
- реалізацію його основних функцій;
- проведення тестування;
- оцінку ефективності впровадження розробленого рішення в діяльність кав'ярні.

Значимість роботи полягає в демонстрації можливостей застосування сучасних інформаційних технологій у малому бізнесі, а також у розробці конкретного інструменту, здатного поліпшити клієнтський сервіс, знизити навантаження на персонал і підвищити загальний рівень цифровізації підприємства.

Таким чином, розробка Telegram-бота для обслуговування клієнтів кав'ярні є актуальним і затребуваним завданням, вирішення якого може стати прикладом успішної інтеграції сучасних IT-рішень у повсякденну практику обслуговування споживачів.

РОЗДІЛ 1
ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Сучасна кав'ярня, будучи популярним форматом закладу громадського харчування, орієнтована не тільки на надання якісного продукту, а й на створення комфортних умов обслуговування для клієнтів. Предметна сфера цього дослідження охоплює процеси, пов'язані з обслуговуванням відвідувачів класичної кав'ярні, в якій передбачено точку видачі замовлень на винос.

Асортимент кав'ярні охоплює різні категорії продукції, як-от гарячі та холодні напої (кава, чай, лимонади тощо), десерти, сендвічі та інші легкі закуски. Обслуговування клієнтів у поточній моделі здійснюється виключно в очному форматі – під час особистого звернення до бариста біля стійки, без використання веб-сайтів, мобільних застосунків або месенджерів.

Середнє навантаження на кав'ярню становить від 50 до 100 замовлень на день, що створює певну потребу в автоматизації низки процесів для підвищення швидкості обслуговування і розвантаження персоналу. Незважаючи на наявність постійних клієнтів, програма лояльності в закладі на поточний момент не реалізована. Інформування відвідувачів про акції, новинки та зміни в меню відбувається за допомогою фізичних інформаційних матеріалів (вивіски, банери всередині кав'ярні), що обмежує охоплення аудиторії та оперативність передачі інформації.

Пропоноване в рамках кваліфікаційної роботи рішення спрямоване на впровадження Telegram-бота, який виконуватиме низку функцій, що сприятимуть поліпшенню якості обслуговування та взаємодії з клієнтами. Зокрема, бот дасть змогу:

- переглядати актуальне меню кав'ярні;
- оформляти замовлення з можливістю вибору часу отримання (негайно або в зазначений проміжок часу);

- здійснювати оплату замовлень безпосередньо через інтерфейс бота (оформлення замовлення можливе лише після успішної оплати);
- залишати відгуки про якість обслуговування та продукції;
- у перспективі – брати участь у програмі лояльності з нарахуванням балів за замовлення, які можна використовувати для отримання безкоштовної продукції.

Таким чином, предметне оточення дослідження охоплює процеси приймання та обробки замовлень, інформування клієнтів, збору зворотного зв'язку та підвищення клієнтської лояльності за допомогою автоматизованого засобу – Telegram-бота. Це рішення дасть змогу не тільки поліпшити клієнтський сервіс, а й оптимізувати внутрішні процеси кав'ярні без необхідності впровадження складних і дорогих IT-систем.

1.2 Огляд наявних аналогів

На сьогодні Telegram-боти активно використовуються в ресторанах і службах доставки їжі. Їхнє основне призначення – спростити процедуру оформлення замовлень, надати клієнтам інформацію про меню, забезпечити зручну оплату й автоматизувати комунікацію. Однак серед закладів формату «кав'ярня» використання таких рішень залишається рідкістю. Серед доступної інформації не вдалося знайти прикладів кав'ярень, що використовують Telegram-бот із можливістю повноцінного оформлення замовлення та оплати, що свідчить про наявність вільної ніші та потенціалу для інноваційного впровадження.

Існуючі Telegram-боти для ресторанів і доставок зазвичай мають стандартний набір функцій: перегляд меню, оформлення замовлення, вибір адреси доставки, автоматична оплата і повідомлення про статус.

Було розглянуто наступні помічники в месенджерах:

- Telegram-бот від компанії «ideil.», розроблений для замовлення їжі з ресторану традиційної тайської кухні Lemongrass;

- Telegram-бот від компанії «Artjoker» для служби доставки їжі Roll Club.

Telegram-бот для ресторану Lemongrass, розроблений вітчизняною компанією «ideil.» має дуже цікавий функціонал. Їхній продукт є нічим іншим, як підпискою на обіди (рис. 1.1).

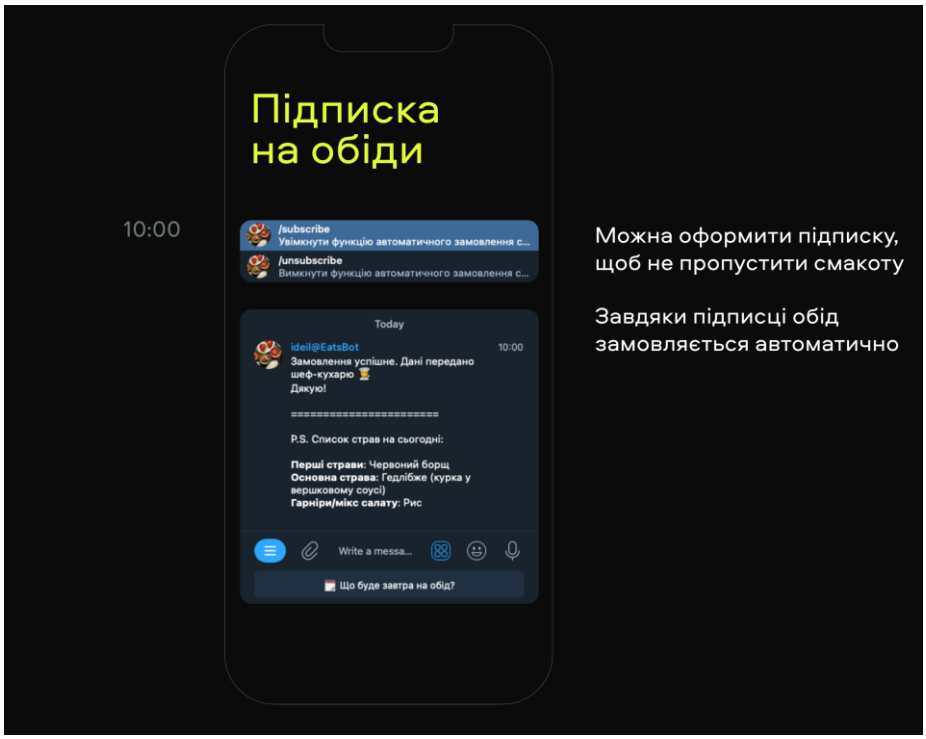


Рисунок 1.1 – Можливість оформити/скасувати підписку на обіди

Телеграм-бот щоранку сповіщає користувачів про доступні страви на обід. Що дуже цікаво, цей бот так само має функціонал у вигляді системи сповіщення, який нагадує користувачеві про можливість замовити цей самий обід. Якщо користувач ігнорує і дане сповіщення, то в певний час він отримує повідомлення про те, що можливість замовлення обід на сьогодні закрита (рис. 1.2)

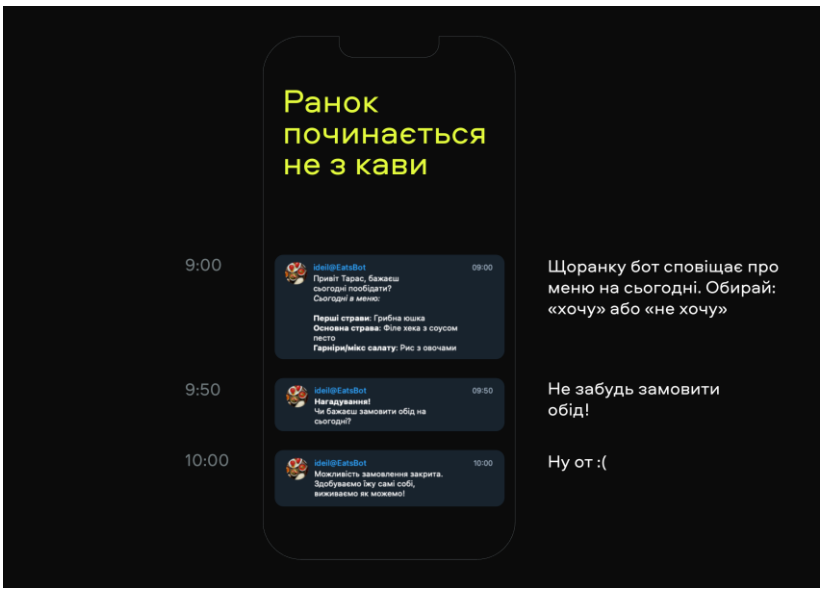


Рисунок 1.2 – Щоранкове сповіщення юзеру щодо можливості замовити обід

Окрему увагу треба приділити функціоналу керівника. Як видно, він дає можливість побачити меню, кількість замовлень за місяць (рис. 1.3) та список усіх, хто замовив обіди, в подальшому який буде передаватися кухарю (рис. 1.4).

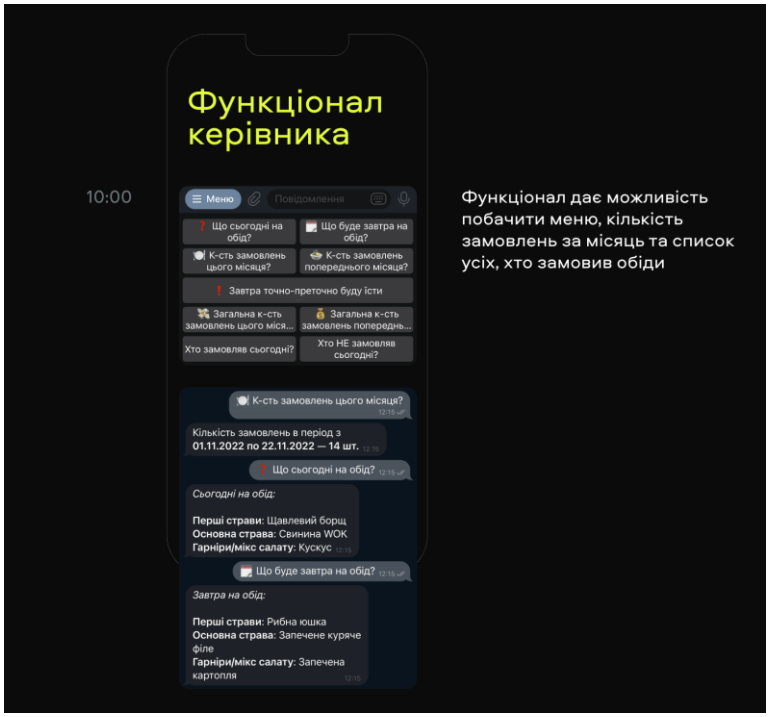


Рисунок 1.3 – Функціонал керівника

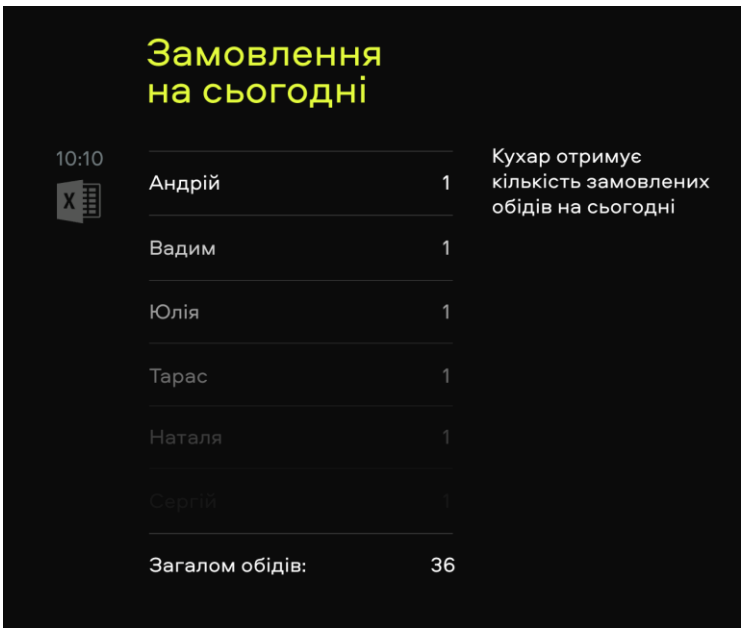


Рисунок 1.4 – Кількість замовлень, яку отримає кухар

Також, кожен користувач має змогу передивитись всі обіди, які плануються готуватись протягом всього тижня (рис. 1.5)

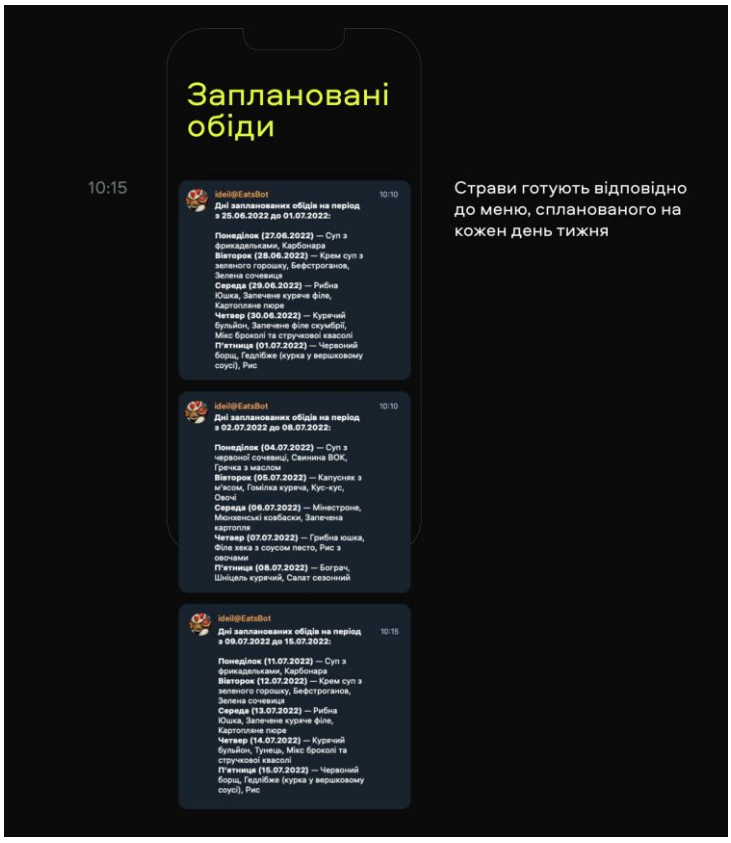


Рисунок 1.5 – Заплановані обіди

Після кожного обіду наприкінці дня можна надати оцінку кожній страві з обіду [1] (рис. 1.6).

Рисунок 1.6 – Оцінка кожної страви з обіду

Отже, цей Telegram-помічник має великий, ефективний функціонал, але все ще має куди розвиватися:

- значним мінусом є те, що обід можна замовити тільки до окремої години, а не на той час, коли захоче сам користувач;
- відсутність зображень страв робить користування ботом не зручним, оскільки користувачеві все рівно треба переходити на сайт ресторану, щоб побачити як виглядатиме страва;
- бот не надає інформації стосовно страв (їх складу, ваги і т.п.);
- немає можливості залишити коментар для кухара, якщо, наприклад, якийсь інгредієнт треба прибрати зі страви.

Що стосується Roll Club, то він є прикладом ефективного цифрового інструменту автоматизації клієнтського сервісу у сфері доставки їжі. Бот являє собою повнофункціональний канал для оформлення замовлень без необхідності відвідування сайту або дзвінка оператору (рис. 1.7).

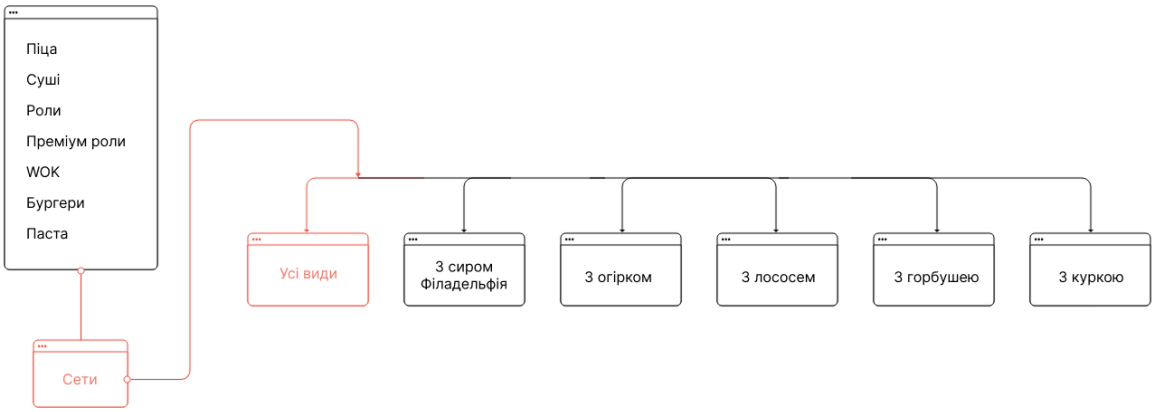


Рисунок 1.7 – Функціональність приймання замовлень

На початку роботи і знайомстві з чат-ботом вони продумали, щоб користувач отримав повідомлення з усіма інструкціями з користування та можливостями цього бота (рис. 1.8).

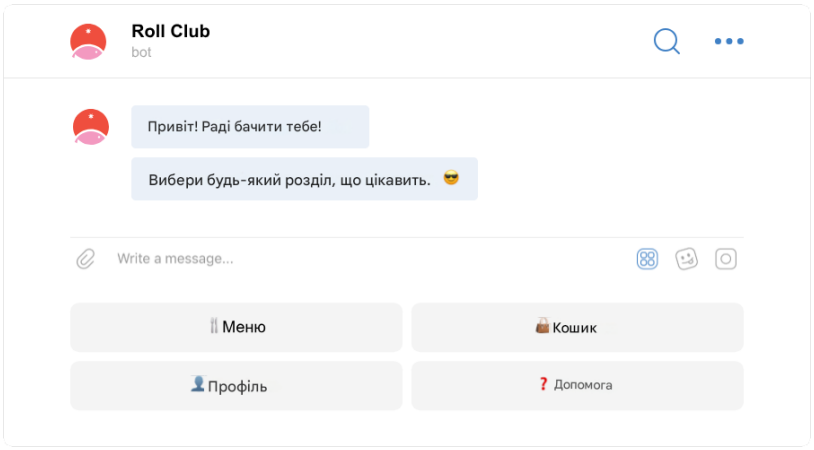


Рисунок 1.8 – Функціональність приймання замовлень

Після запуску бот надає користувачеві структуроване меню, розбите на категорії: суші, роли, піца, бургери, десерти та напої (рис. 1.9).

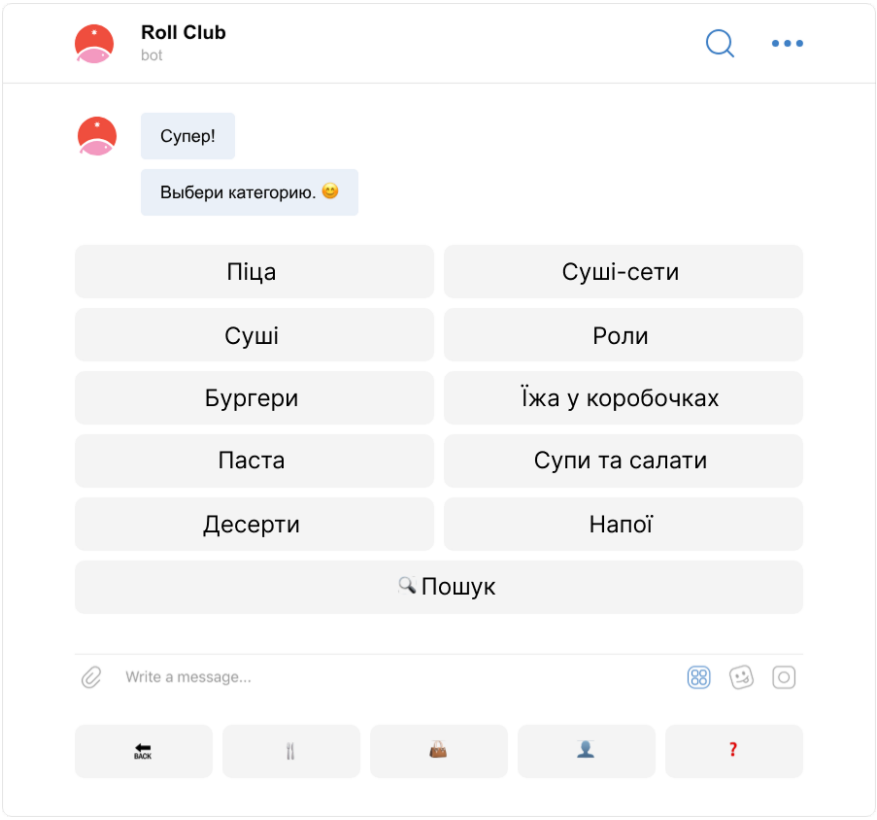


Рисунок 1.9 – Функціональність приймання замовлень

Кожна картка містить у собі назву страви, її складові, розмір порції, ціну та можливість одразу ж додати її до замовлення (рис. 1.10).

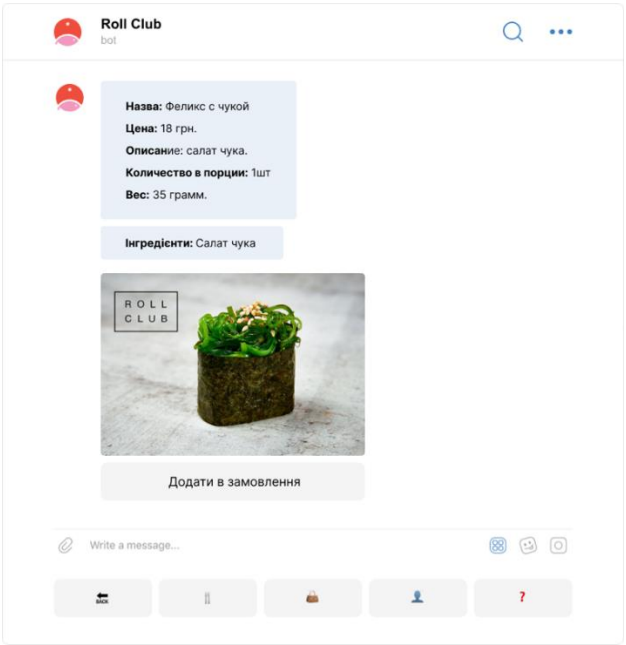


Рисунок 1.10 – Картка товара

Після того, як користувач вибрав усі необхідні позиції і додав їх до замовлення, він отримує загальну вартість замовлення (рис. 1.11).

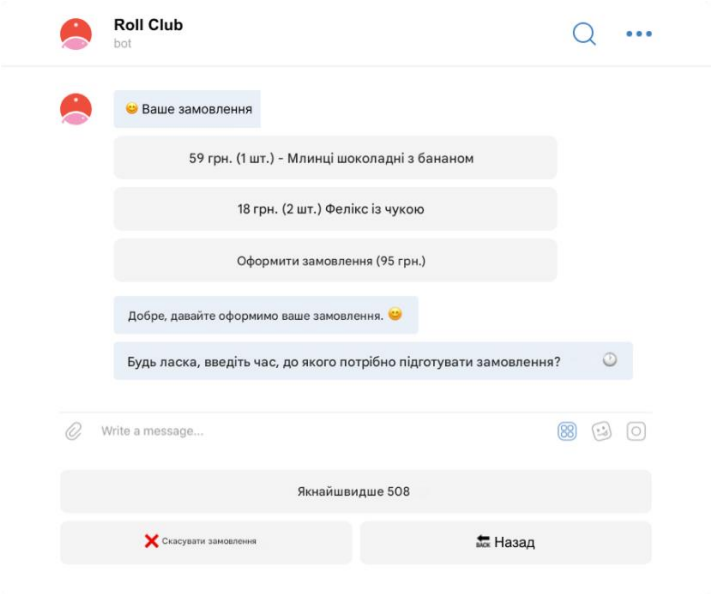


Рисунок 1.11 – Віртуальний кошик

У цьому боті для замовлення їжі онлайн ви можете самі вибрати, коли хочете, щоб ваше замовлення було доставлено. І скільки коштуватиме доставка за певної суми замовлення (рис. 1.12).

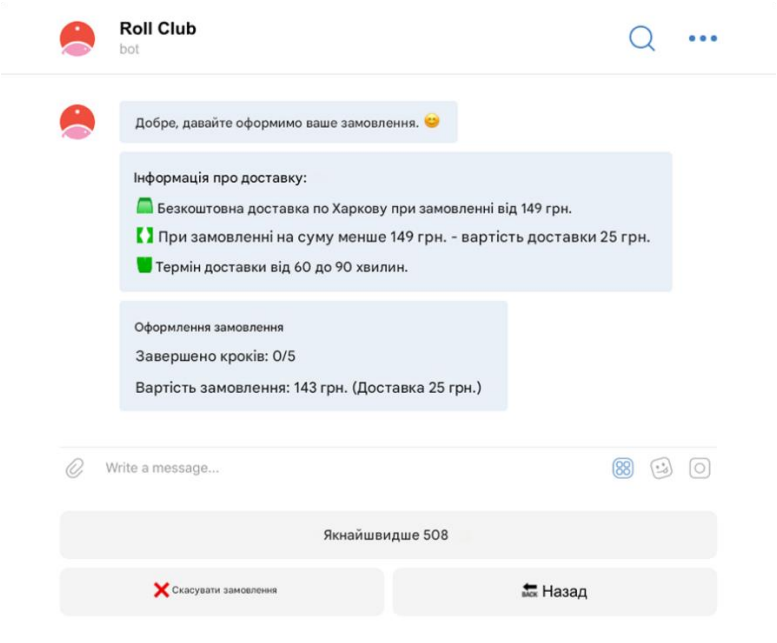


Рисунок 1.12 – Інформація щодо доставки

Бот підтримує кілька способів оплати, включно з оплатою онлайн і готівкою при отриманні [2] (рис. 1.13).

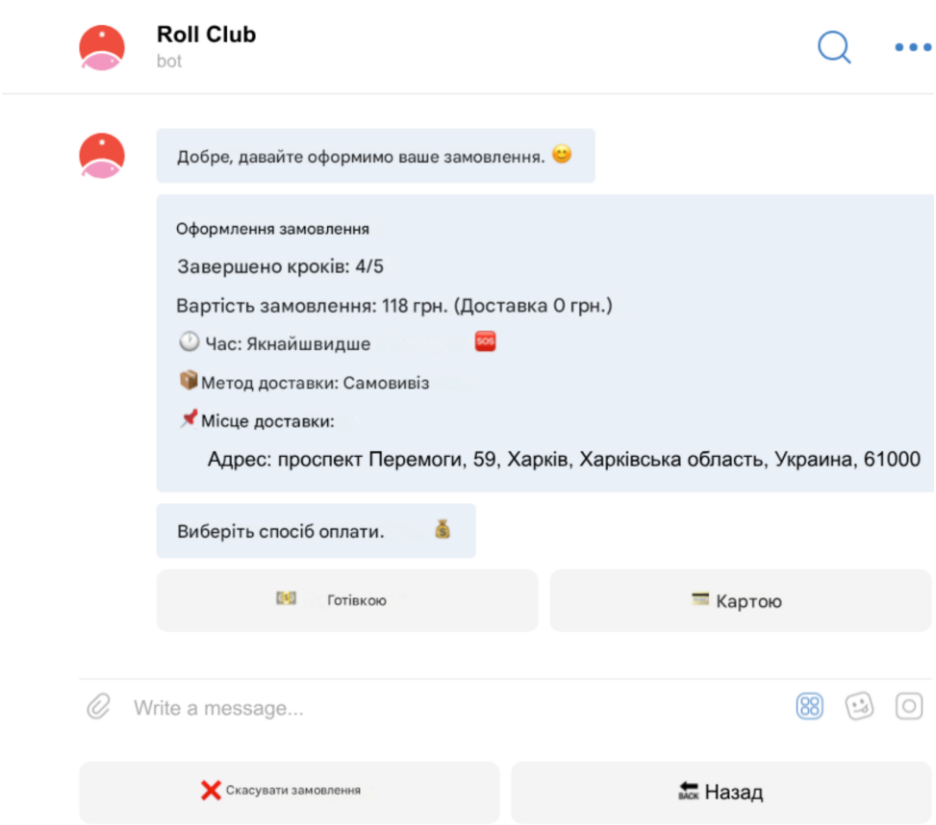


Рисунок 1.13 – Способи оплати замовлення

Roll Club бот є гарним прикладом Telegram-ботів для замовлення їжі. Попри це, в нього не вистачає важливого функціоналу, як для подібних ботів, а саме:

- відсутність змоги залишити відгук;
- як і в чат-боті для Lemongrass, немає можливості залишити коментар для кухара.

Рішення, що розробляється в рамках цієї кваліфікаційної роботи, відрізнятиметься від наявних аналогів низкою ключових особливостей. По-перше, особливу увагу буде приділено зручності користувацького інтерфейсу і простоті взаємодії. По-друге, планується реалізація функції попереднього замовлення з можливістю зазначення точного часу отримання. Це особливо

важливо для клієнтів, які бажають забрати замовлення дорогою або під час перерви, без очікування.

Крім того, унікальним елементом пропонованого рішення стане система лояльності: користувачі отримуватимуть бонусні бали за кожну покупку (у розмірі 5% від суми замовлення), які зможуть використовувати для оплати майбутніх замовлень. Бали будуть округлятися до цілого числа і еквівалентно прирівнюватися до грошового еквівалента (1 бал = 0,5 грн). Користувачеві буде надано вибір способу оплати: банківською карткою, готівкою або накопиченими балами - за умови, що їх кількість достатня для покриття вартості замовлення.

Таким чином, на відміну від існуючих рішень, Telegram-бот, що розробляється в даній роботі, буде адаптований під специфічні потреби кав'ярні та забезпечить вищий рівень персоналізації, зручності та клієнтської залученості.

1.3 Постановка задачі

З метою автоматизації процесів обслуговування клієнтів у закладі громадського харчування, що функціонує у форматі класичної кав’ярні з можливістю продажу напоїв та їжі, було поставлено завдання розробити Telegram-бот.

Користувач має пройти обов'язкову реєстрацію під час першої взаємодії з ботом, вказавши ім'я та номер телефону. Після реєстрації користувачеві стають доступні всі основні функції.

Бот має забезпечувати користувачам зручний доступ до актуального меню кав'ярні, що включає такі категорії, як кава, лимонади, сніданки та їжа формату «to-go». Важливою функцією є оформлення замовлення з можливістю вибору часу його отримання – як на найближчий час, так і на конкретний момент. Після оформлення та успішної оплати замовлення автоматично передається бариста із зазначенням усіх необхідних даних:

складу замовлення, часу, контактного номера і, за наявності, коментаря користувача.

Передбачається можливість оплати замовлення трьома способами: готівкою при отриманні, за допомогою банківської карти, а також бонусними балами. У рамках вбудованої системи лояльності користувачеві нараховуються бонусні бали в розмірі 5% від вартості кожного замовлення. Один бал еквівалентний 0,5 гривні. Бали не згорають і можуть використовуватися для повної оплати майбутніх замовлень. Можливість часткової оплати (комбінування балів і картки/готівки) не передбачена. Після оформлення кожного замовлення бот автоматично повідомляє користувача про нараховані бали та його поточний бонусний баланс.

Також передбачена система сповіщень, за допомогою якої користувач отримує сповіщення про готовність замовлення. Особистий кабінет користувача включає можливості перегляду балансу балів, зміни контактного номера, залишення відгуку, а також зв'язку з бариста.

Розробка бота здійснюватиметься з використанням мови програмування Python. Щодо бібліотеки, за допомогою якої буде реалізовано цього бота, то вибір припав між «python-telegram-bot» та «aiogram». Обидві бібліотеки є хорошими інструментами реалізації подібного проєкту. Однак, «python-telegram-bot» виділяється своєю простотою і стабільністю, що робить її чудовим вибором для старту і швидкого прототипування. [5, 6] З іншого боку, «aiogram» має архітектурні переваги, що роблять її більш придатною для високонавантажених, модульних і гнучких рішень. Тому пропонується реалізація даного боту за допомогою бібліотеки «aiogram». [16]

На поточному етапі розгортання проєкту на сервері або в хмарі не планується.

Висновки до розділу

У першому розділі було розглянуто загальні положення, які формують основу для подальшого дослідження і розробки Telegram-бота, призначеного для обслуговування клієнтів кав'ярні. Проведено аналіз предметної області, в якій функціонує підприємство, визначено ключові потреби користувачів, а також внутрішні бізнес-процеси, що підлягають автоматизації.

Особливу увагу було приділено огляду наявних аналогів програмних рішень у суміжних сферах. Проведений аналіз показав, що використання Telegram-ботів у сфері обслуговування закладів харчування на сьогоднішній день є обмеженим, що додатково підкреслює актуальність обраної теми і перспективність її реалізації.

Також було сформульовано постановку задачі, яка полягає у створенні функціонального та зручного у використанні інструменту для взаємодії між кав'ярнею та її клієнтами.

Розробка цього Telegram-бота дасть змогу оптимізувати процеси обслуговування клієнтів, знизити навантаження на персонал і підвищити рівень клієнтського сервісу, зберігаючи водночас індивідуальний підхід і оперативність взаємодії.

РОЗДІЛ 2
ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз предметної області

Як було зазначено раніше, розглянута кав'ярня функціонує в класичному форматі – обслуговує клієнтів переважно на винос. Середньостатистична успішна кав'ярня подібного типу щодня приймає від 50 до 100 відвідувачів, що забезпечує стабільний дохід і підтримує бізнес на базовому рівні. Однак в умовах постійного зростання конкуренції в сегменті малого громадського харчування, особливо в містах з високою щільністю населення і активним діловим життям, виживання і розвиток нових закладів стають все більш складними.

Звичайний ранок типового мешканця міста – це поспіх, брак часу і боротьба за кожну хвилину. Кава вже давно стала невід'ємною частиною ранкової рутини багатьох людей – студентів, офісних працівників, підприємців. Ранковий ритм життя часто не залишає часу на тривале перебування в закладі, тому швидкість обслуговування і зручність стають ключовими факторами вибору. У цьому контексті вибір між двома кав'ярнями, навіть якщо вони пропонують ідентичний за якістю напій, може бути зроблений на користь тієї, що пропонує більший комфорт. Отже, для молодих кавових брендів вкрай важливо впроваджувати додаткові сервісні рішення, які можуть виділити їх на тлі конкурентів і забезпечити стійкий потік клієнтів.

Розглянемо гіпотетичну, але реалістичну ситуацію: клієнт збирається на роботу, поспішає і усвідомлює, що час – критично важливий ресурс. При цьому він відчуває необхідність в кофеїні, щоб «включитися» в робочий день. По дорозі у нього дві кав'ярні: кав'ярня «А», в яку він ходить регулярно, і кав'ярня «Б», яку нещодавно відкрив для себе. Обидві пропонують аналогічний за якістю продукт, проте кав'ярня «Б» надає можливість заздалегідь оформити замовлення через Telegram-бот – буквально за пару

кліків з дому або в ліфті. Клієнту залишається лише вчасно підійти і забрати напій без черг і очікування. Більш того, система лояльності, реалізована в тому ж боті, дозволяє клієнту отримувати бонусні бали, які можна обмінювати на безкоштовні напої. У подібній ситуації вибір на користь кав'ярні «Б» стає очевидним. В умовах динамічного міського ритму саме такі, здавалося б, незначні деталі формують лояльність і стимулюють повторні покупки.

Цей приклад наочно демонструє цінність автоматизованого рішення, такого як Telegram-бот, для бізнесу. Його впровадження здатне:

- підвищити кількість замовлень;
- розширити клієнтську базу за рахунок зручності та гнучкості сервісу;
- створити ефект унікальності на тлі конкурентів;
- зміцнити клієнтську лояльність за рахунок бонусної програми;
- збільшити загальну конкурентоспроможність кав'ярні на ринку.

Таким чином, основними проблемами, з якими стикаються кав'ярні в сучасних умовах, є висока конкуренція і необхідність залучення нових клієнтів при збереженні поточних. Telegram-бот виступає в даній моделі як ефективний інструмент цифровізації обслуговування, здатний не тільки вирішити зазначені проблеми, але і стати драйвером подальшого зростання і масштабування бізнесу.

2.2 Проєктування системи

Блок-схема, яка представлена на рисунку 2.1, відображає основну логіку функціонування Telegram-бота, розробленого для автоматизації процесів обслуговування клієнтів кав'ярні.

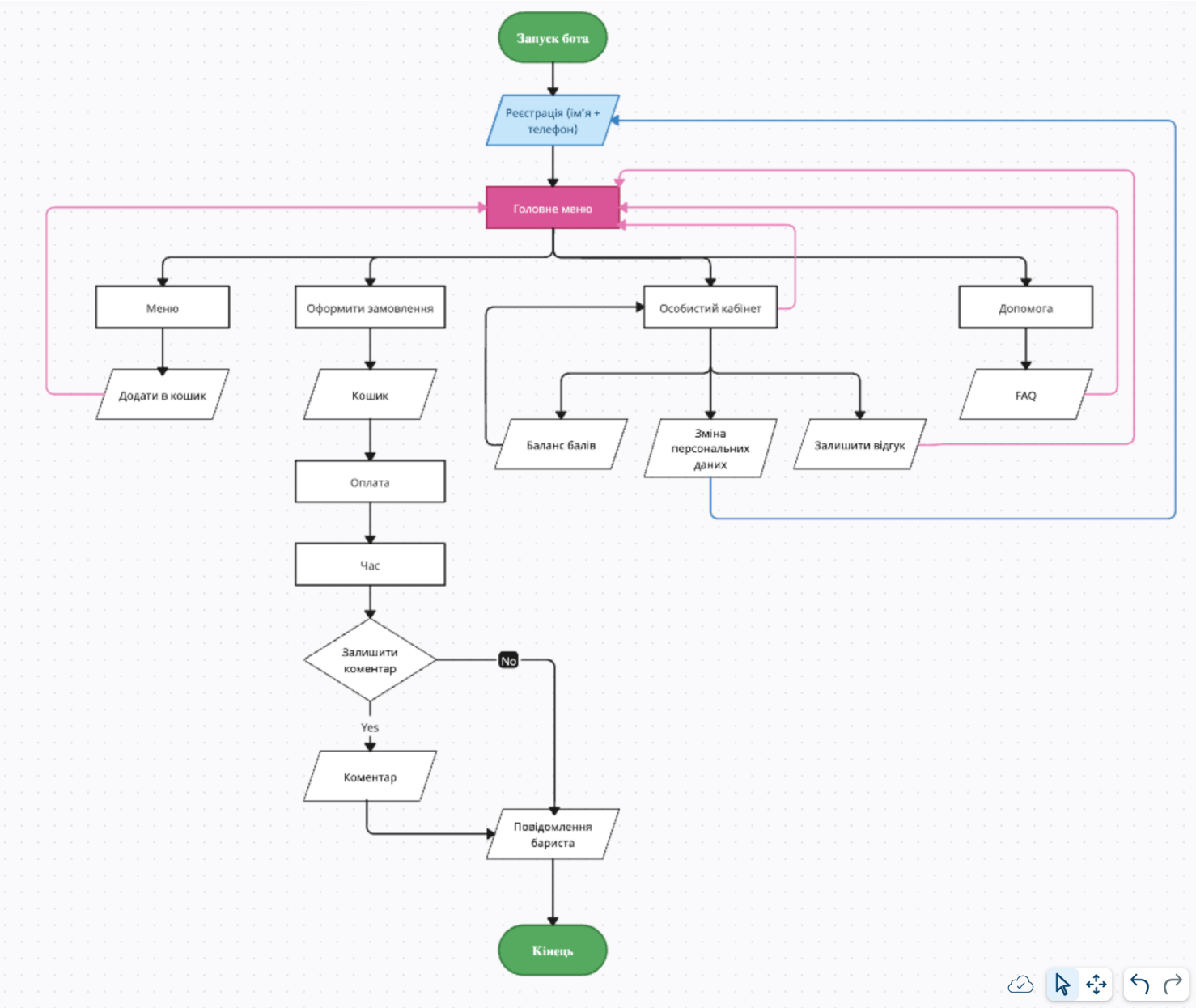


Рисунок 2.1 – Блок-схема телеграм-бота для кав’ярні

Для кращого розуміння функціонування боту, розглянемо кожен елемент схеми окремо.

Взаємодія користувача з ботом починається з його запуску. Відразу після активації бот ініціює процес реєстрації, в ході якого користувач надає базову інформацію – ім'я та номер телефону. Реєстрація є обов'язковим етапом, оскільки ці дані необхідні для ідентифікації клієнта, персоналізації сервісу, ведення історії замовлень, роботи системи бонусів, а також для зв'язку з бариста в разі необхідності.

У користувача буде можливість переглянути все меню завдяки блоку з меню. В цьому блоці також виконується формування кошику.

Одним з ключових функціональних блоків є можливість оформлення замовлення. Користувач вибирає відповідний пункт («оформлення замовлення») і переглядає всі товари, які додав завдяки попередньому блоку.

Після цього клієнт переходить до етапу оплати. Бот надає вибір способу оплати: готівкою, банківською карткою, або бонусними балами, які можна накопичувати за програмою лояльності.

Далі бот задає питання –чи потрібно приготувати замовлення на найближчий час або користувач хоче вказати конкретний час, чи має бажання клієнт залишити відгук і т.д. Якщо вибирається найближчий час, замовлення відразу передається в роботу. У разі вибору певного часу, з'являється можливість його вказати вручну. Що стосується коментаря, то у разі бажання залишити коментар, користувач має можливість вручну прописати коментар до бариста.

Після успішного платежу, обраного часу та надання/не надання коментаря замовлення вважається оформленим, і бот автоматично відправляє повідомлення баристу з деталями: ім'я клієнта, контактний номер, зміст замовлення, обраний час і коментар, якщо такий є.

Зареєстрованому користувачеві доступний особистий кабінет, який включає в себе наступні розділи:

- баланс бонусів – відображається поточна кількість накопичених бонусних балів, які нараховуються в розмірі 5% від суми кожного замовлення (один бал еквівалентний 0,5 гривні);

- зміна персональних даних – можливість змінити ім'я або номер телефону, вказані при реєстрації;

- залишити відгук – після використання сервісу клієнт може залишити відгук про замовлення або обслуговування.

Telegram-бот також містить розділ «Допомога», в якому зібрані поширені запитання та короткі інструкції з використання бота. Цей розділ сприяє зниженню навантаження на персонал кав'ярні, дозволяючи користувачам самостійно знаходити відповіді на базові запитання.

Після надсилання повідомлення бариста, бот завершує активну сесію з користувачем. Клієнт може повернутися до головного меню і продовжити взаємодію (наприклад, переглянути бонуси, залишити відгук або оформити нове замовлення) або завершити використання бота до наступного звернення.

Таким чином, представлена блок-схема наочно демонструє послідовну і логічно побудовану архітектуру Telegram-бота, призначеного суттєво спростити і прискорити процес взаємодії клієнта з кав'ярнею.

2.3 Математичне та алгоритмічне забезпечення

Розробка програмного забезпечення, зокрема Telegram-бота, що виконує роль цифрового помічника для клієнтів кав'ярні, вимагає чіткого математичного та алгоритмічного обґрунтування. Це забезпечує не тільки коректну обробку даних користувачів, але й оптимізацію всіх процесів взаємодії між клієнтом, системою та персоналом закладу. У цьому підрозділі розглянуто ключові алгоритми функціонування системи, математичні моделі нарахування та списання бонусних балів, а також логіку обробки замовлень.

Кожна взаємодія користувача з ботом починається з етапу реєстрації. Система пропонує користувачеві ввести ім'я та номер телефону. Після цього

відбувається перевірка існування даного користувача в базі даних. Якщо запис відсутній – створюється новий. Надалі для ідентифікації використовується унікальний ідентифікатор Telegram-користувача і його номер телефону. Це дозволяє забезпечити персоналізовану взаємодію, ведення історії замовлень, накопичення бонусів і обробку відгуків.

Меню являє собою динамічно оновлюваний список товарних позицій, кожна з яких містить наступні параметри:

- назва товару;
- категорія (кава, лимонади, десерти, to-go їжа і т.п.);
- ціна;
- можливі модифікатори (наприклад, об'єм напою, добавки тощо).

Математично, меню може бути представлено у вигляді масиву структур:

$$M = \{m_1, m_2, \ldots, m_n\}, m_i = (name_i, price_i, category_i, options_i) \quad (2.1)$$

При оформленні замовлення користувач послідовно вибирає позиції, що його цікавлять. Кожна позиція додається до тимчасової структури даних – «кошика». Після завершення вибору бот розраховує загальну вартість замовлення:

$$S = \sum_{i=1}^{k} p_i \quad (2.2)$$

де p – вартість кожної з обраних позицій, k – кількість позицій у замовленні.

Користувач вибирає один з двох варіантів виконання:

- якнайшвидше;
- на конкретний час (в межах робочого дня кав’ярні).

У разі вибору конкретного часу, здійснюється перевірка часового інтервалу на допустимість (наприклад, від поточного моменту має пройти мінімум 10 хвилин).

Після оплати замовлення бот нараховує бонусні бали. Розрахунок бонусів проводиться за такою формулою:

$$B = \lfloor S * r \rfloor \quad (2.3)$$

де:

- B – кількість нарахованих бонусів;
- S – сума замовлення;
- r – коефіцієнт нарахування (наприклад, 5% або 0,05).

Нараховані бали додаються на бонусний рахунок користувача. Використовувати бонуси можна при оформленні наступного замовлення. При цьому 1 бонусний бал еквівалентний 0,5 грн.

Після успішної оплати (незалежно від методу), бот формує повідомлення, яке містить:

- ім'я користувача;
- номер телефону;
- повний склад замовлення;
- коментар користувача (якщо був доданий);
- час, на який заплановано замовлення.

Ця інформація передається в Telegram-аккаунт бариста у вигляді повідомлення від бота.

Алгоритм відгуків передбачає, що відгук можна залишити тільки після завершеного замовлення. Це запобігає фальсифікації даних і дає адекватний зворотний зв'язок бізнесу.

Telegram-бот реалізований на основі подійно-орієнтованої моделі і побудований за принципом кінцевого автомата, де кожен стан відповідає етапу взаємодії, а переходи між ними здійснюються в залежності від дій користувача.

Висновки до розділу

Проведено всебічну аналітичну, проєктну та алгоритмічну опрацювання майбутньої системи – Telegram-бота для обслуговування клієнтів кав'ярні.

На етапі аналізу предметної області були виявлені ключові характеристики функціонування сучасної кав'ярні, орієнтованої на формат «на винос», а також визначені конкурентні фактори, що впливають на розвиток бізнесу. Встановлено, що використання цифрових інструментів, зокрема Telegram-бота, здатне підвищити ефективність обслуговування, розширити клієнтську базу та зміцнити конкурентні позиції підприємства на ринку.

В рамках проєктування системи була описана загальна архітектура Telegram-бота, розроблена блок-схема його функціонування, а також визначені основні компоненти користувальницького інтерфейсу і бізнес-логіки. Запропоновані проєктні рішення забезпечують інтуїтивну навігацію, простоту оформлення замовлення, а також наявність ключових функцій: особистий кабінет, бонусна система, можливість вибору часу отримання замовлення, оплата різними способами і передача інформації баристі.

На етапі математичного та алгоритмічного забезпечення були розроблені та обґрунтовані ключові формули, що лежать в основі реалізації бонусної програми, запропоновані алгоритми нарахування балів.

Таким чином, інформаційне, проєктне та алгоритмічне наповнення Telegram-бота створює надійну основу для його практичної реалізації, задовольняючи потреби бізнесу в автоматизації та підвищенні якості обслуговування клієнтів.

РОЗДІЛ 3
ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки

Для створення програмного забезпечення, що забезпечує автоматизацію процесів обслуговування клієнтів кав'ярні з використанням Telegram-бота, був обраний комплекс сучасних засобів розробки, що поєднує в собі надійність, зручність і широкі функціональні можливості. Основним завданням при виборі інструментів було забезпечення гнучкості, масштабованості, а також легкості в підтримці та подальшому розвитку програмного продукту.

В якості основної платформи розробки було обрано мову програмування Python – одну з найбільш популярних і універсальних мов програмування в світі. Його основні переваги полягають у простоті синтаксису, лаконічності вираження логіки, а також багатій екосистемі бібліотек, що охоплює широкий спектр завдань: від роботи з базами даних до машинного навчання і створення web-інтерфейсів. Python ідеально підходить для швидкого прототипування та реалізації додатків, що вимагають інтенсивної взаємодії із зовнішніми API, як у випадку з Telegram.

Для інтеграції з Telegram API використовувалась бібліотека Aiogram, яка є одним з найбільш актуальних рішень для асинхронної розробки Telegram-ботів. Aiogram побудована на базі асинхронного програмування (asyncio), що дозволяє обробляти безліч запитів користувачів одночасно, не блокуючи виконання програми. Це робить її особливо придатною для завдань, пов'язаних з високим навантаженням і необхідністю миттєвої реакції на дії користувача. Бібліотека надає зручний інструментарій для маршрутизації команд, створення інтерфейсів, роботи з клавіатурами і управління станами діалогу з користувачем. [3]

Безпосередньо розробка проєкту буде вестись в професійному середовищі програмування PyCharm від компанії JetBrains. Даний інструмент

забезпечує широкий спектр можливостей: інтелектуальне автодоповнення коду, налагодження в реальному часі, інтеграція з системами контролю версій (наприклад, Git), управління залежностями, а також повноцінна підтримка роботи з віртуальними середовищами. Завдяки PyCharm процес розробки бота стане більш впорядкованим, контрольованим і продуктивним. [8]

В якості системи управління базами даних була обрана SQLite – легка і вбудована реляційна база даних, яка не вимагає окремого сервера. Основним аргументом на користь вибору SQLite послужила простота її налаштування та інтеграції, а також відсутність необхідності в складній архітектурі для зберігання даних. База даних проєкту зберігає ключову інформацію, таку як: ім’я користувачів, номери телефонів, зміст замовлень, коментарі та суми оплати. Така структура забезпечує достатній рівень функціональності на початковому етапі експлуатації бота і може бути в подальшому масштабована або замінена на більш потужне рішення при необхідності.

Що стосується реалізації функціоналу оплати, в поточній версії проєкту вона буде виконана в демонстраційному режимі. Користувачеві надається можливість вибору способу оплати (готівкою, карткою або бонусними балами), проте технічна інтеграція з реальними платіжними системами (наприклад, Apple Pay або Google Pay) поки не реалізована. Це обумовлено етапом розробки і тестування прототипу. Разом з тим, архітектура проєкту передбачає можливість подальшої інтеграції з платіжними шлюзами і банківськими API.

Окремо варто відзначити, що взаємодія з користувачем в Telegram буде реалізована через сценарії, написані з використанням механізмів станів і контекстів, що робить роботу бота інтуїтивно зрозумілою і персоналізованою. З огляду на особливості предметної області, в розробці також застосовуватимуться підходи, спрямовані на забезпечення надійності обробки даних, стійкості до помилок і зручності масштабування логіки взаємодії.

У сукупності, обрані засоби розробки дозволять створити надійну основу для реалізації інтелектуального помічника у вигляді Telegram-бота, здатного ефективно виконувати функції автоматизації обслуговування

клієнтів кав'ярні. Їх гнучкість, сучасність і простота використання роблять ці інструменти оптимальними для проєктів подібного масштабу і напрямку.

3.2 Вимоги до технічного та програмного забезпечення

Розробка та впровадження програмного забезпечення, призначеного для обслуговування клієнтів кав'ярні за допомогою Telegram-бота, вимагає попереднього аналізу та формулювання відповідних вимог як до технічної, так і до програмної складової. Ці вимоги визначаються як особливостями архітектури реалізованого програмного рішення, так і умовами його функціонування в реальному операційному середовищі.

На поточному етапі проєкт реалізується і тестується в локальному середовищі на персональному комп'ютері, що функціонує під управлінням операційної системи macOS. Використання локального середовища на етапі розробки дозволяє гнучко і швидко вносити необхідні зміни в структуру коду, оперативно виявляти і усувати помилки, а також адаптувати інтерфейс взаємодії бота з користувачем. Однак у перспективі розглядається можливість розміщення Telegram-бота на сервері, що дозволить забезпечити його цілодобову доступність, стабільну роботу при збільшенні кількості користувачів, а також масштабованість системи.

Технічні вимоги до обладнання в локальному середовищі залишаються досить помірними. Мінімально достатньою є конфігурація, що включає в себе сучасний багатоядерний процесор (наприклад, Intel Core i5 або аналогічний за продуктивністю чіп Apple Silicon), обсяг оперативної пам'яті від 4 ГБ, а також не менше 500 МБ вільного простору на диску. В даному випадку, конфігурація складається з наступних комплектуючих:

- 8/10 ядерний процесор Apple M1 Pro;
- 16 ГБ ОЗУ;
- більше 40 ГБ вільної пам'яті на диску.

Така конфігурація здатна забезпечити стабільну роботу програмного забезпечення при реалізації стандартної бізнес-логіки, а також при обробці запитів від користувачів в межах обмеженого кола тестування. У разі переходу до серверної моделі розміщення, слід передбачити наявність високошвидкісного інтернет-з'єднання, стабільної мережевої інфраструктури і, при необхідності, можливості резервного копіювання даних і відмовостійкості.

Таким чином, сформульовані вимоги до технічного та програмного забезпечення на початковому етапі розробки є обґрунтованими і достатніми для реалізації ключового функціоналу Telegram-бота в локальному середовищі. Водночас вони закладають основу для подальшого масштабування проєкту з урахуванням можливого зростання кількості користувачів, розширення функціональних можливостей і переходу на серверну інфраструктуру. Подібний підхід дозволяє забезпечити гнучкість системи, її адаптивність до умов експлуатації та відповідність сучасним вимогам до цифрових сервісів у сфері обслуговування клієнтів.

3.3 Опис програмної реалізації

Як зазначалось раніше, для розробки Telegram-бота кав'ярні було обрано інтегроване середовище розробки PyCharm, яке забезпечує комплексні інструменти для створення Python-додатків. На рисунку 3.1 представлений процес ініціалізації нового проєкту з назвою «coffee-telegram-bot». При створенні проєкту були встановлені наступні параметри: використання інтерпретатора Python версії 3.11.5, створення ізольованого віртуального середовища для управління залежностями, а також автоматична генерація файлу main.py в якості точки входу в додаток. Вибір даної конфігурації обумовлений необхідністю забезпечення стабільної роботи бота і можливістю легкого управління зовнішніми бібліотеками, такими як aiogram, які будуть використані в процесі розробки.

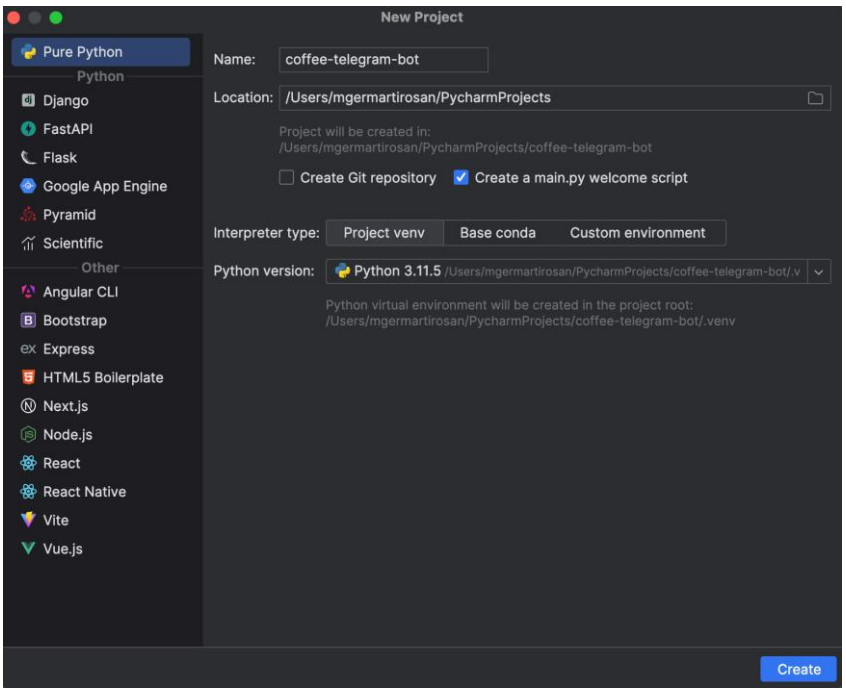


Рисунок 3.1 – Створення проєкту в PyCharm

Найпершим етапом розробки є встановлення ключових бібліотек. Однією з них є aiogram, що представляє собою сучасний фреймворк для створення Telegram-ботів на мові Python. Команда «pip install aiogram», що виконується в терміналі інтегрованого середовища розробки (рис. 3.2), ініціює процес завантаження та встановлення бібліотеки з офіційного репозиторію Python Package Index (PyPI) [4]. Дана бібліотека забезпечує асинхронну архітектуру обробки запитів, що критично важливо для забезпечення високої продуктивності бота при роботі з множинними сесіями користувачів.

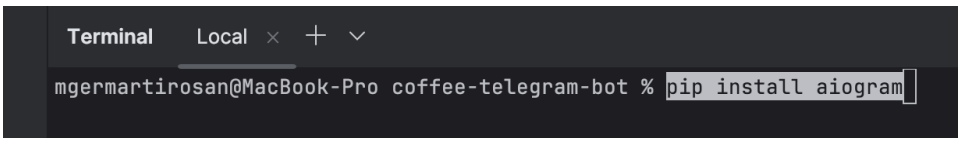


Рисунок 3.2 – Встановлення бібліотеки aiogram

Іншою ключовою бібліотекою, яка буде необхідна під час розробки бота, є бібліотека DateTime, що служить в даній роботі для забезпечення коректної обробки тимчасових даних в системі замовлень кав'ярні. Команда «pip install DateTime» встановлює спеціалізований пакет, призначений для

роботи з датами, часом і часовими інтервалами. Дана бібліотека необхідна для реалізації функцій планування замовлень, відстеження часу приготування напоїв, формування часових міток для замовлень і забезпечення коректної роботи системи повідомлень. [7]

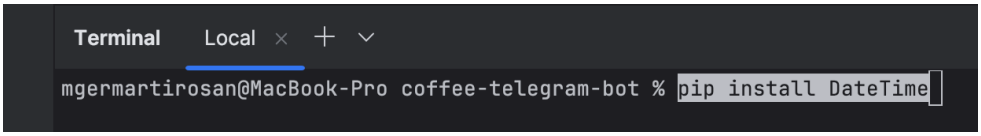


Рисунок 3.3 – Встановлення бібліотеки DateTime

Останнім етапом підготовки створення бота є імпорт необхідних модулів і компонентів для функціонування Telegram-бота кав'ярні. Набір ключових компонентів (кожен з яких має свою функціональність в системі) (рис. 3.4) виглядає наступним чином:

- модуль asyncio (вбудований в Python, тому не було необхідності в pip установці), який реалізує асинхронну модель програмування, що дозволяє обробляти множинні запити користувачів без блокування основного потоку виконання [9];
- з бібліотеки aiogram імпортуються фундаментальні класи Bot і Dispatcher, де перший забезпечує взаємодію з Telegram API, а другий здійснює маршрутизацію і обробку вхідних повідомлень;
- механізм кінцевих автоматів представлений класами State, StatesGroup і FSMContext, які необхідні для реалізації складних багатоетапних діалогів в процесі оформлення замовлень;
- фільтр Command являє собою клас для обробки команд, що починаються з символу «/»;
- компоненти користувальницького інтерфейсу включають класи Message, ReplyKeyboardMarkup, ReplyKeyboardRemove і KeyboardButton, що забезпечують створення інтерактивних елементів взаємодії;
- імпорт тимчасових модулів datetime і timedelta надає інструментарій для роботи з тимчасовими даними в контексті управління замовленнями.

```
import asyncio
from aiogram import Bot, Dispatcher
from aiogram.fsm.state import State, StatesGroup
from aiogram.fsm.context import FSMContext
from aiogram.filters import Command
from aiogram.types import Message, ReplyKeyboardMarkup, ReplyKeyboardRemove, KeyboardButton
from datetime import datetime, timedelta
```

Рисунок 3.4 – Імпорт необхідних модулів і бібліотек

Тепер можна приступати до розробки безпосередньо самого бота. Перше, що треба зробити – це створити екземпляр класу Bot, в якому обов'язковим аргументом є token. Паралельно створюється об'єкт dp – диспетчер. Він відповідає за обробку всіх вхідних повідомлень від користувачів і перенаправлення їх до відповідних функцій-обробників (рис. 3.5)

```
API_TOKEN = ''

bot = Bot(token=API_TOKEN)
dp = Dispatcher()
```

Рисунок 3.5 – Створення боту

На відміну від диспетчера, bot має обов'язковий аргумент у вигляді token. Значення токена не може бути порожнім. Щоб отримати токен, який з'єднає нашого бота і код в одне ціле, потрібно звернутися до найголовнішого бота в телеграмі – BotFather («батько ботів»). Для початку роботи з даним ботом, необхідно знайти його через ID – @botfather (рис. 3.6).

Рисунок 3.6 – Дані найголовнішого боту в телеграмі

Для того, щоб отримати необхідний нам токен, необхідно запустити бота і дотримуватися інструкцій, які він дає (рис. 3.7 - 3.10):

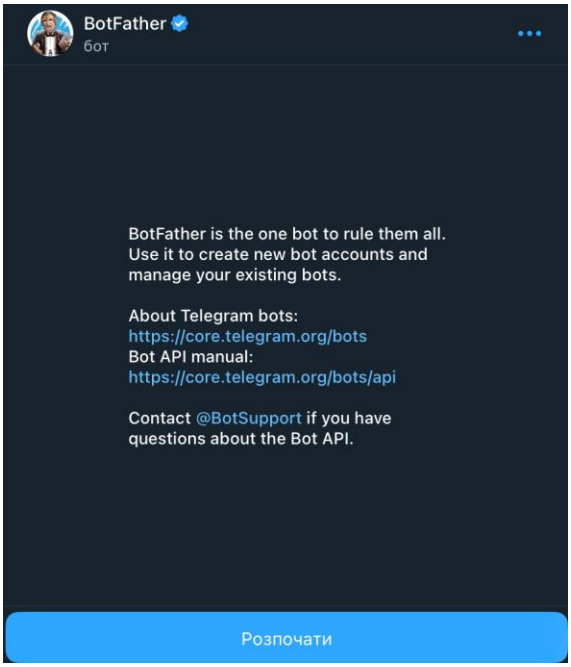


Рисунок 3.7 – Початковий екран взаємодії з BotFather

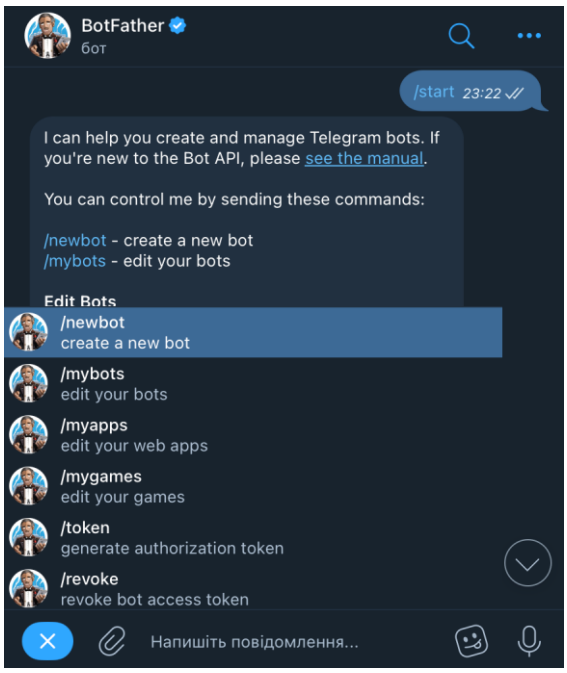


Рисунок 3.8 – Список команд, серед яких обираємо «/newbot»

Рисунок 3.9 – Назва телеграм-боту

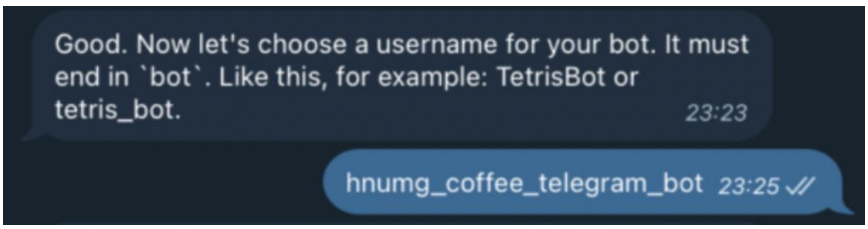


Рисунок 3.10 – Ім’я користувача (username) для телеграм-боту

Після того, як ми виконали ці кроки, ми отримуємо повідомлення з привітанням щодо створення бота, в якому якраз і знаходиться наш телеграм-бот токен (рис. 3.11):

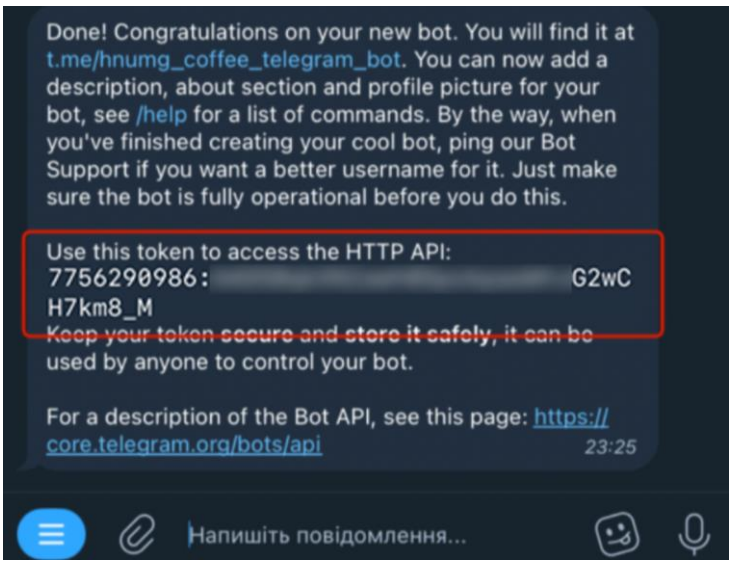


Рисунок 3.11 – Отримання токену

Налаштуємо одразу візуальну складову нашого боту, а саме: поставимо аватар. Для цього, виконуємо кожний крок за інструкцією (рис. 3.12 – 3.15):

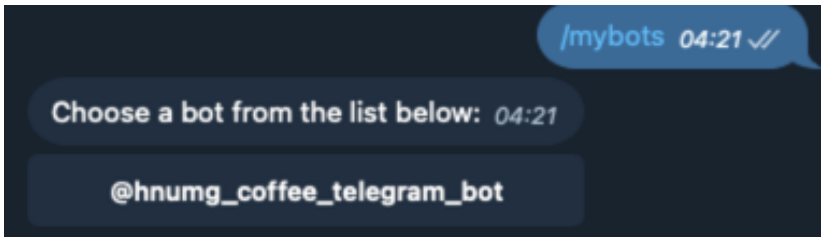


Рисунок 3.12 – Викликаємо команду /mybots

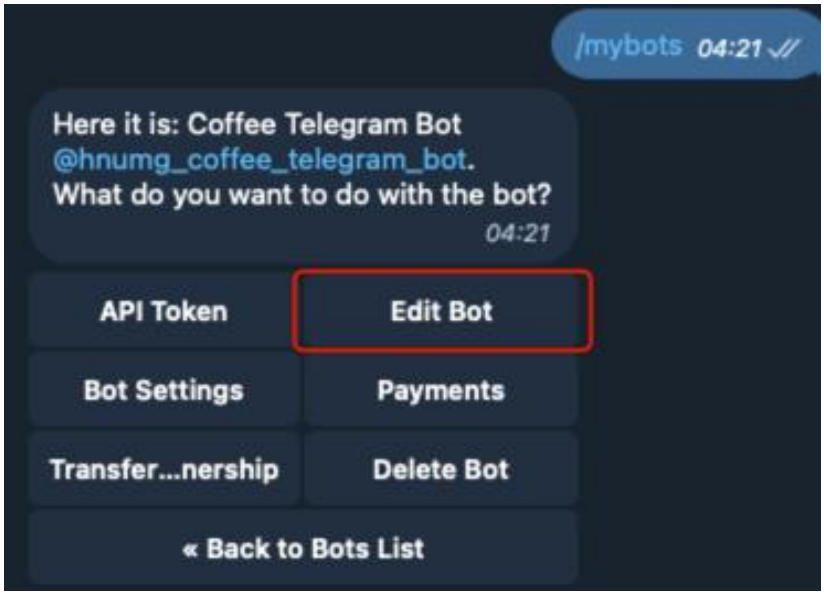


Рисунок 3.13 – Переходимо у розділ «Edit Bot»

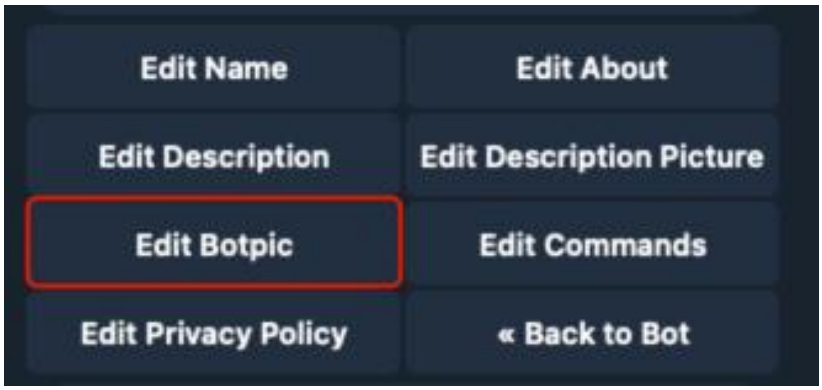


Рисунок 3.14 – Обираємо пункт «Edit Botpic»

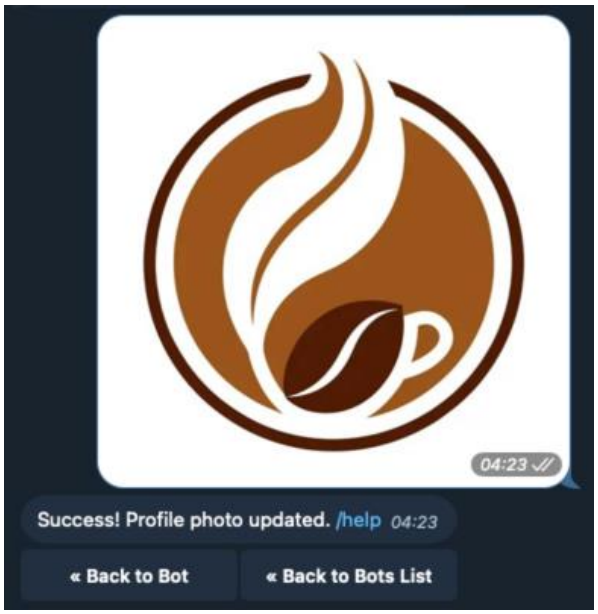


Рисунок 3.15 – Завантажуємо фото, яке бажаємо бачити на аватарі бота

Отриманий токен вставляємо в змінну API_TOKEN, яку позначаємо як константа за рахунок великих літер (рис. 3.16):

```
API_TOKEN = '7756290986                    G2wCH7km8_M'

bot = Bot(token=API_TOKEN)
dp = Dispatcher()
```

Рисунок 3.16 – Додавання токену

Додаємо частини коду, які відповідатиме за запуск бота (рис. 3.16). Асинхронна функція main() містить виклик методу start_polling(), який запускає процес безперервного опитування серверів Telegram для отримання нових повідомлень і подій від користувачів. Конструкція if __name__ == “__main__”: забезпечує виконання коду тільки при безпосередньому запуску файлу. Функція asyncio.run(main()) створює асинхронний цикл подій і запускає основну функцію бота, забезпечуючи його постійну роботу і готовність до обробки запитів користувачів.

```
async def main():
    await dp.start_polling(bot)


if __name__ == '__main__':
    asyncio.run(main())
```

Рисунок 3.17 – Реалізація основного циклу виконання бота

Використаємо ось такий фрагмент коду, щоб переконатися в правильності виконаних дій (рис. 3.18). Представлений код демонструє реалізацію обробника команди /start, яка традиційно використовується для ініціалізації взаємодії користувача з Telegram-ботом. Декоратор @dp.message(Command(“start”)) реєструє асинхронну функцію start() як обробник, який активується при отриманні команди /start від користувача. Функція приймає параметр message типу Message, що містить об'єкт вхідного повідомлення з метаданими про користувача і чат. Метод message.answer() здійснює відправку відповідного повідомлення в той же чат, звідки була отримана команда.

```python
@dp.message(Command('start'))
async def start(message: Message):
    await message.answer(
        'Вітаємо у Coffee Telegram Bot! ☕ \n\n'
        'Для початку, будь ласка, пройдіть реєстрацію.'
    )
```

Рисунок 3.18 – Фрагмент коду для перевірки

Переходимо до нашого бота і натискаємо кнопку «Почати», щоб запустити його (рис. 3.19). Що й потрібно було довести: на початку нашого спілкування з ботом ми отримуємо вітальне повідомлення, яке прописали кроком раніше (рис. 3.20)

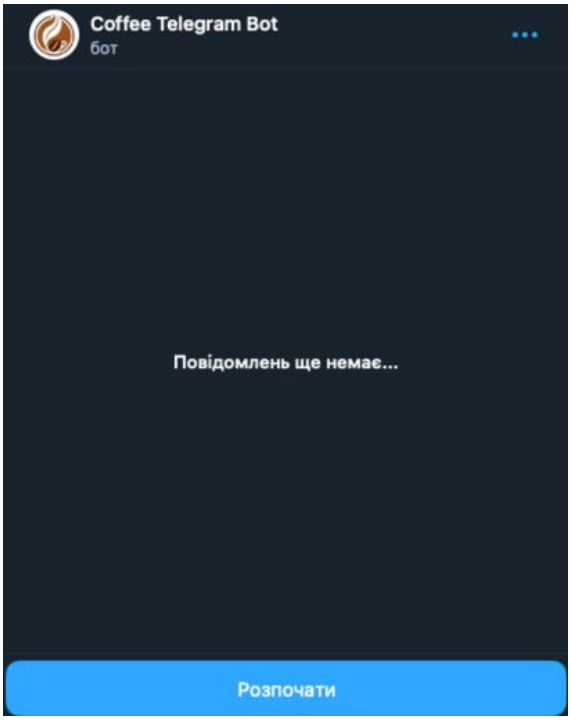


Рисунок 3.19 – Початковий екран Coffee Telegram Bot

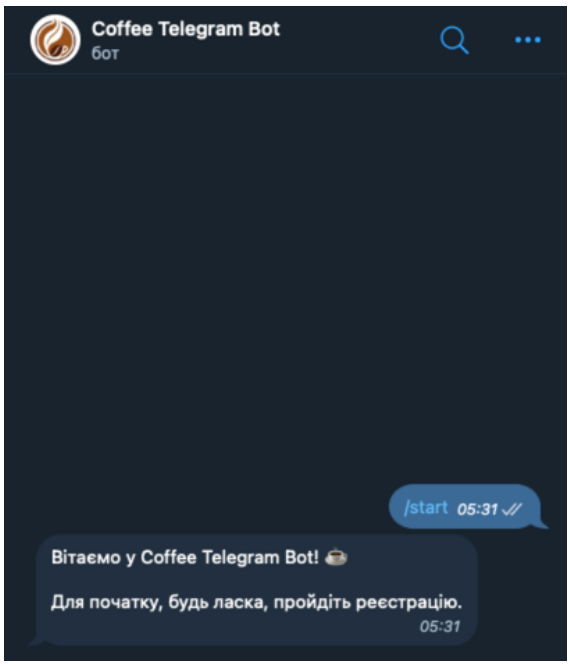


Рисунок 3.20 – Бот надсилає вітальне повідомлення

На цьому етапі розробки можна вважати, що бот вже створений. Далі – це вже безпосередньо його налаштування, визначення функціоналу та його реакція на повідомлення. Тому, почнемо його «вдосконалювати» відповідно до сформованого технічного завдання.

Як і будь-якому підприємству, яке працює з клієнтами, нам треба збирати інформацію від наших відвідувачів. Тому постає питання щодо збору інформації. У даному випадку імені та номера телефону буде цілком достатньо. Для реалізації цього, скористаємося імпортованими класами State, StatesGroup та FSMContext. Але, щоб ними скористатися, необхідно створити клас, який буде обробляти стани реєстрації. Назвемо його RegisterUser, і зробимо його спадкоємним від StatesGroup. Клас буде містити два стани: waiting_for_name і waiting_for_phone, кожен з яких створюється як екземпляр класу State(). Стан waiting_for_name буде активуватися, коли система очікує від користувача введення імені, а waiting_for_phone – коли буде вимагати вказання номера телефону.

```
class RegisterUser(StatesGroup):
    waiting_for_name = State()
    waiting_for_phone = State()
```

Рисунок 3.21 – Клас RegisterUser, який відповідає за стани реєстрації

Удосконалимо той код, який ми використовували для перевірки працездатності бота. Тепер, замість того, щоб просто виводити одне повідомлення, разом з ним бот виводить додаткове повідомлення, в якому просить користувача ввести своє ім'я. Після цього, за допомогою state.set_state(RegisterUser.waiting_for_name) ми встановлюємо стан «в очікуванні імені» (рис. 3.22)

```
@dp.message(Command('start'))
async def start(message: Message, state: FSMContext):
    await message.answer(
        'Вітаємо у Coffee Telegram Bot! ☕\n\n'
        'Для початку, будь ласка, пройдіть реєстрацію.'
    )
    await message.answer('Введіть ваше ім\'я 👤:')
    await state.set_state(RegisterUser.waiting_for_name)
```

Рисунок 3.22 – Запит імені користувача

Далі, за допомогою @dp.message(RegisterUser.waiting_for_name) обробляємо дані стосовно імені та встановлюємо новий стан – «в очікуванні номеру» (рис. 3.23)

```
@dp.message(RegisterUser.waiting_for_name)
async def get_name(message: Message, state: FSMContext):
    await state.update_data(name=message.text.strip())
    await message.answer('Тепер введіть ваш номер телефону у форматі +380XXXXXXXXX 📞:')
    await state.set_state(RegisterUser.waiting_for_phone)
```

Рисунок 3.23 – Обробка імені користувача та запит номеру телефону

Таку ж саму роль відіграє @dp.message(RegisterUser.waiting_for_phone), тільки вже не для імені, а для номеру телефона. Після того, як необхідні дані для реєстрації внесені, клієнт отримує відповідне повідомлення (рис. 3.24).

```python
@dp.message(RegisterUser.waiting_for_phone)
async def get_phone(message: Message, state: FSMContext):
    phone = message.text.strip()
    if not phone.startswith('+380') or not phone[1:].isdigit() or len(phone) != 13:
        await message.answer('❗ Будь ласка, введіть коректний номер телефону у форматі +380XXXXXXXXX ❗')
        return

    await state.update_data(phone=phone)
    data = await state.get_data()

    name = data['name']
    phone = data['phone']

    await message.answer( text: f'{name}, дякуємо, що обрали нас!🎉\n\n'
                          f'Ваш акаунт зареєстровано під номером: {phone}', reply_markup=main_menu_keyboard())

    await state.clear()
```

Рисунок 3.24 – Обробка номеру телефона користувача

Після створення аккаунту, клієнт не тільки отримує повідомлення, але й для нього стає активним меню навігації в чат-боті, яке складається з наступних пунктів: «Меню», «Оформити замовлення», «Особистий кабінет» та «Допомога» (рис. 3.25 – 3.26)

```python
def main_menu_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='📋 Меню'), KeyboardButton(text='🛒 Оформити замовлення')],
            [KeyboardButton(text='👤 Особистий кабінет'), KeyboardButton(text='❓ Допомога')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

Рисунок 3.25 – Реалізація головного меню в боті

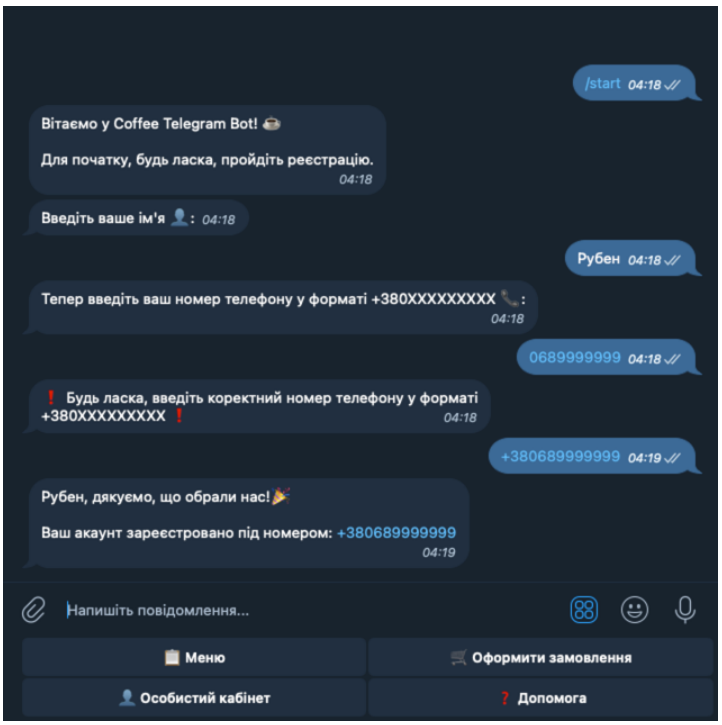


Рисунок 3.26 – Тестування етапу реєстрації

Використання механізму кінцевих автоматів дозволяє контролювати послідовність діалогу з користувачем, забезпечуючи коректний збір необхідних даних для реєстрації.

Розглянемо кожний пункт окремо. Почнемо з кнопки «Меню». Ця кнопка повинна показувати повне меню кав'ярні, розділивши товари на такі категорії: «Кава», «Лимонади», «Сендвічі» та «Випічка». Для цього, за допомогою нашого диспетчера та lambda-функції робимо перевірку: якщо повідомлення, яке надіслав клієнт, є «📋 Меню» (тобто тією ж кнопкою з головного меню), то йому надається меню з вибору підкатегорій (рис. 3.27 – 3.29):

```
@dp.message(lambda message: message.text == '📋 Меню')
async def show_products_menu(message: Message):
    await message.answer( text: 'Оберіть категорію:', reply_markup=products_menu_keyboard())
```

Рисунок 3.27 – Перевірка на натиск кнопки «📋 Меню»

```python
def products_menu_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='☕ Кава'), KeyboardButton(text='🍷 Лимонади')],
            [KeyboardButton(text='🥪 Сендвічі'), KeyboardButton(text='🥐 Випічка')],
            [KeyboardButton(text='◀️ До головного меню')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

Рисунок 3.28 – Меню категорій

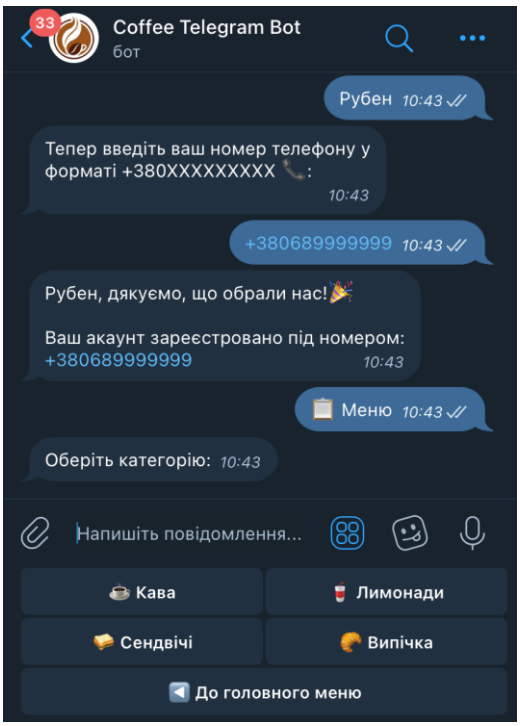


Рисунок 3.29 – Реалізація меню кав’ярні в телеграм боті

Серед кнопок меню кав'ярні також є кнопка «◀️ До головного меню». Ця кнопка має однакову функціональність у будь-якому фрагменті коду, а саме – повертає користувача до головного меню.

```python
@dp.message(lambda message: message.text == '◀️ До головного меню')
async def back_to_main_menu(message: Message):
    await message.answer( text: 'Ви повернулись до головного меню:', reply_markup=main_menu_keyboard())
```

Рисунок 3.30 – Обробник повернення до головного меню

Створимо клас Order, який успадковує від StatesGroup і визначає послідовність станів для процесу оформлення замовлення в системі бота

кав'ярні. Кожен стан представлений як екземпляр класу State() і відповідає певному етапу взаємодії з користувачем: choosing_type активується при виборі категорії продукту, choosing_name – при виборі конкретної назви товару, choosing_size – при вказівці розміру порції, choosing_price – при підтвердженні вартості, choosing_comment – при додаванні коментарів до замовлення, і choosing_review – при фінальному перегляді замовлення перед підтвердженням (рис. 3.31)

```
class Order(StatesGroup):
    choosing_type = State()
    choosing_name = State()
    choosing_size = State()
    choosing_price = State()
    choosing_comment = State()
    choosing_review = State()
```

Рисунок 3.31 – Визначення станів для оформлення замовлення

В окремих змінних (константах) записуємо дані для товарів у вигляді словника, в якому ключ – назва товару, а значення – вкладений словник з розмірами та ціною товару.

```
COFFEE_PRICES = {
    "Капучіно": {"S": 45, "M": 55, "L": 65},
    "Американо": {"S": 35, "M": 45, "L": 55},
    "Латте": {"S": 50, "M": 60, "L": 70},
    "Макіато": {"S": 48, "M": 58, "L": 68},
    "Мокачино": {"S": 52, "M": 62, "L": 72}
}

LEMONADES_PRICES = {
    "Мохіто": {"S": 60, "M": 75, "L": 80},
    "Лимонад Цитрус": {"S": 60, "M": 75, "L": 80},
    "Оранж": {"S": 50, "M": 60, "L": 70},
    "Лимонад маракуя": {"S": 70, "M": 80, "L": 90}
}

SANDWICHES_PRICES = {
    "Сендвіч з куркою": {"S": 39, "M": 55, "L": 70},
    "Сендвіч з шинкою": {"S": 45, "M": 69, "L": 90},
    "Сендвіч з салямі": {"S": 30, "M": 50, "L": 70},
    "Сендвіч Веганський": {"S": 25, "M": 40, "L": 50}
}

BAKERY_PRICES = {
    "Круасан": {"S": 10, "M": 15, "L": 20},
    "Штрудель": {"S": 10, "M": 15, "L": 25},
    "Пиріг": {"S": 40, "M": 50, "L": 60},
    "Чизкейк": {"S": 50, "M": 70, "L": 100}
}
```

Рисунок 3.32 – Визначення цінової структури продукції

Оскільки кнопку меню ми вже реалізували, то пора вже зайнятися реалізацією товарів в даному меню, а конкретно – реалізацією товарів в

розділах Кава, Лимонади, Сендвічі та Випічка. Для цього з відповідного прайс-листу з попереднього кроку, «витягуємо» значення ціни для кожного розміру кожного товару, що потім демонструємо користувачеві. Далі – встановлюємо state.set_state(Order.choosing_name) для обов'язкового вибору назви товару в наступному кроці. Повторюємо цей фрагмент коду для всіх 4-х категорій меню (рис. 3.33 – 3.44):

```python
@dp.message(lambda message: message.text == '☕ Кава')
async def show_coffee_menu(message: Message, state: FSMContext):
    await state.update_data(product_type='Кава')
    await state.set_state(Order.choosing_name)
    prices_text = ''
    for coffee, sizes in COFFEE_PRICES.items():
        prices_text += coffee
        for size, price in sizes.items():
            prices_text += f' // {size} - {price}грн'
        prices_text += '\n'

    await message.answer( text: 'Оберіть каву, яку бажаєте замовити:\n\n'
                         f'{prices_text}', reply_markup=coffee_menu_keyboard())
```

Рисунок 3.33 – Прайс-лист для кави

```python
def coffee_menu_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='Капучіно')],
            [KeyboardButton(text='Американо ')],
            [KeyboardButton(text='Латте')],
            [KeyboardButton(text='Макіато')],
            [KeyboardButton(text='Мокачино')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

Рисунок 3.34 – Клавіатура для вибору товару

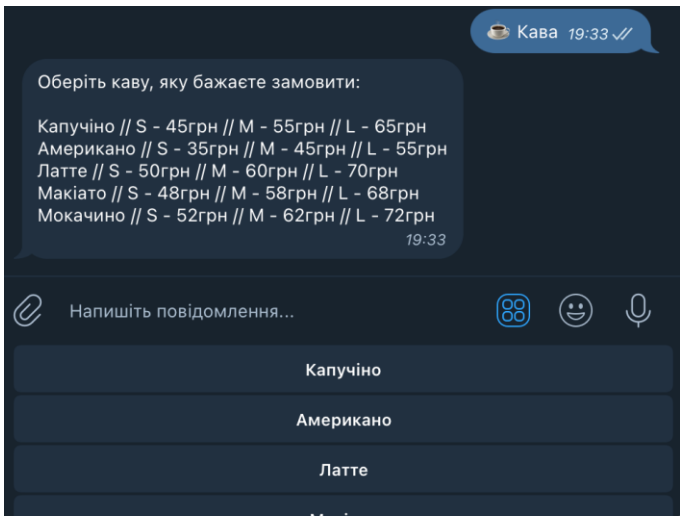


Рисунок 3.35 – Меню товару

```python
@dp.message(lambda message: message.text == '🍹 Лимонади')
async def show_lemonade_menu(message: Message, state: FSMContext):
    await state.update_data(product_type='Лимонад')
    await state.set_state(Order.choosing_name)
    prices_text = ''
    for lemonade, sizes in LEMONADES_PRICES.items():
        prices_text += lemonade
        for size, price in sizes.items():
            prices_text += f' // {size} - {price}грн'
        prices_text += '\n'

    await message.answer( text: 'Оберіть лимонад, який бажаєте замовити:\n\n'
                         f'{prices_text}', reply_markup=lemonade_menu_keyboard())
```

Рисунок 3.36 – Прайс-лист для лимонаду

```python
def lemonade_menu_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='Мохіто')],
            [KeyboardButton(text='Лимонад Цитрус')],
            [KeyboardButton(text='Оранж')],
            [KeyboardButton(text='Лимонад маракуя')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

Рисунок 3.37 – Клавіатура для вибору товару

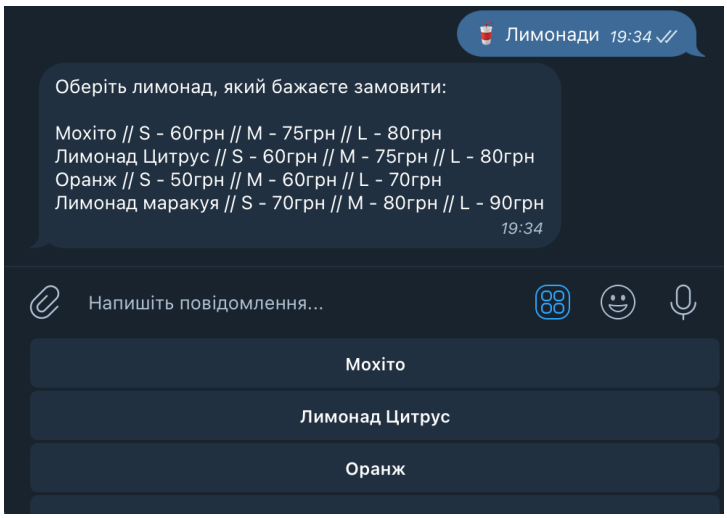


Рисунок 3.38 – Меню товару

```python
@dp.message(lambda message: message.text == '🥪 Сендвічі')
async def show_sandwich_menu(message: Message, state: FSMContext):
    await state.update_data(product_type='Сендвіч')
    await state.set_state(Order.choosing_name)
    prices_text = ''
    for sandwich, sizes in SANDWICHES_PRICES.items():
        prices_text += sandwich
        for size, price in sizes.items():
            prices_text += f' // {size} - {price}грн'
        prices_text += '\n'

    await message.answer( text: 'Оберіть сендвіч, який бажаєте замовити:\n\n'
                         f'{prices_text}', reply_markup=sandwich_menu_keyboard())
```

Рисунок 3.39 – Прайс-лист для сендвічів

```python
def sandwich_menu_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='Сендвіч з куркою')],
            [KeyboardButton(text='Сендвіч з шинкою')],
            [KeyboardButton(text='Сендвіч з салямі')],
            [KeyboardButton(text='Сендвіч Веганський')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

Рисунок 3.40 – Клавіатура для вибору товару

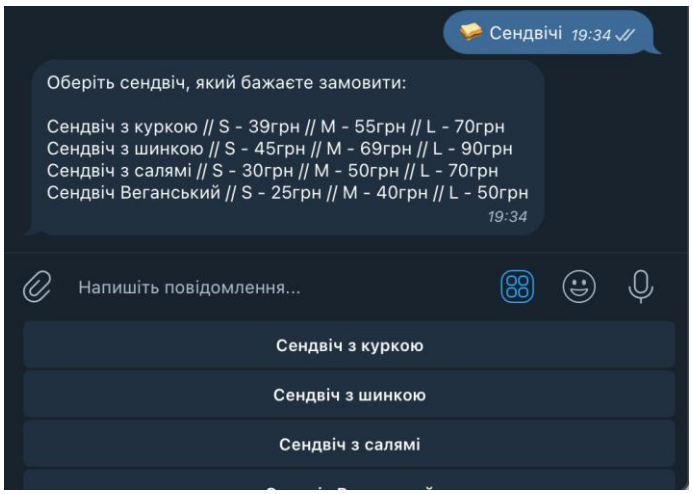


Рисунок 3.41 – Меню товару

```python
@dp.message(lambda message: message.text == '🥐 Випічка')
async def show_bakery_menu(message: Message, state: FSMContext):
    await state.update_data(product_type='Випічка')
    await state.set_state(Order.choosing_name)
    prices_text = ''
    for bakery, sizes in BAKERY_PRICES.items():
        prices_text += bakery
        for size, price in sizes.items():
            prices_text += f' // {size} - {price}грн'
        prices_text += '\n'

    await message.answer(text: 'Оберіть випічку, яку бажаєте замовити:\n\n'
                         f'{prices_text}', reply_markup=bakery_menu_keyboard())
```

Рисунок 3.42 – Прайс-лист для випічки

```python
def bakery_menu_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='Круасан')],
            [KeyboardButton(text='Штрудель')],
            [KeyboardButton(text='Пиріг')],
            [KeyboardButton(text='Чизкейк')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

Рисунок 3.43 – Клавіатура для вибору товару

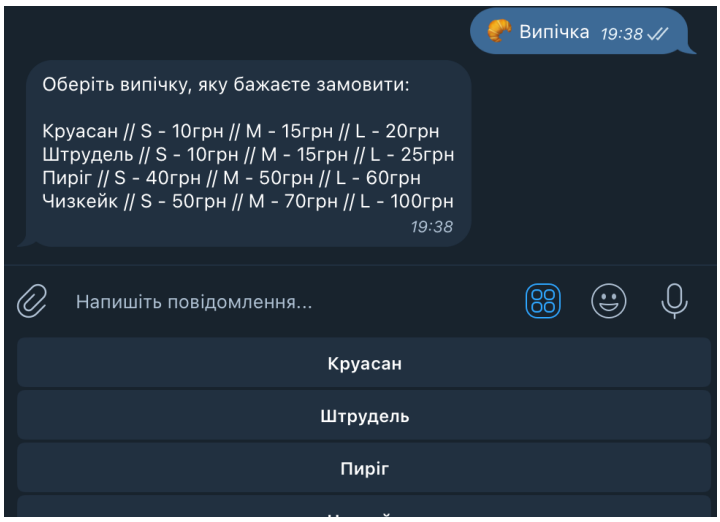


Рисунок 3.44 – Меню товару

Після того, як отримали ім’я товару, необхідно отримати його розмір (рис. 3.45 – 3.47):

```
@dp.message(Order.choosing_name)
async def choosing_product_name(message: Message, state: FSMContext):
    await state.update_data(product_name=message.text)
    await message.answer( text: f'Якого розміру {message.text} ви бажаєте:', reply_markup=size_keyboard())
    await state.set_state(Order.choosing_size)
```

Рисунок 3.45 – Сценарій вибору розміру

```
def size_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='S'), KeyboardButton(text='M'), KeyboardButton(text='L')],
            [KeyboardButton(text='❌ Скасувати')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

Рисунок 3.46 – Клавіатура для вибору розміру

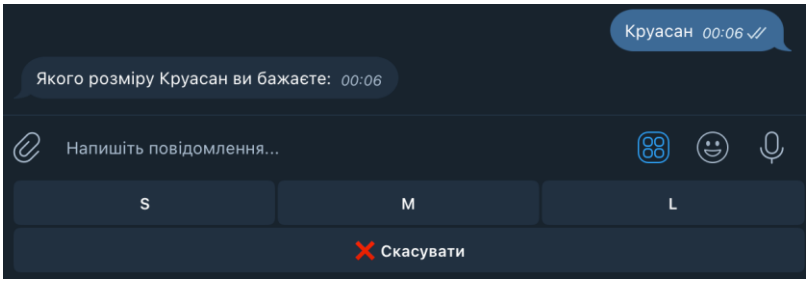


Рисунок 3.47 – Вибір розміру товару

Одним з найголовніших етапів розробки бота є кошик. Для того, щоб реалізувати даний функціонал в боті, нам необхідно створити змінну checkout, яка буде словником (рис. 3.48)

```
checkout = {}
```

Рисунок 3.48 – Оголошення змінної checkout

Враховуючи, що ціна товару залежить від його розміру (S, M або L), нам необхідно відштовхуватися саме від цього. Тому, у функції choosing_product_size() ми на основі отриманих даних щодо назви та розміру товару, отримуємо дані про ціну, звертаючись до прайс-листів (рис. 3.49 – 3.50)

```python
@dp.message(Order.choosing_size)
async def choosing_product_size(message: Message, state: FSMContext):
    if message.text == '❌ Скасувати':
        await state.clear()
        await message.answer( text: 'Вибір скасовано.', reply_markup=main_menu_keyboard())
        return
    if message.text not in ['S', 'M', 'L']:
        await message.answer("Будь ласка, оберіть розмір із запропонованих варіантів.")
        return
    if message.text in ['S', 'M', 'L']:
        await state.update_data(product_size=message.text)

    data = await state.get_data()

    product_type = data.get('product_type')
    product = data.get('product_name')
    product_size = data.get('product_size')
    price = 0

    if product_type == 'Кава':
        price = COFFEE_PRICES.get(product).get(product_size)
    elif product_type == 'Лимонад':
        price = LEMONADES_PRICES.get(product).get(product_size)
    elif product_type == 'Сендвіч':
        price = SANDWICHES_PRICES.get(product).get(product_size)
    elif product_type == 'Випічка':
        price = BAKERY_PRICES.get(product).get(product_size)

    user_id = message.from_user.id

    if user_id not in checkout:
        checkout[user_id] = []

    checkout[user_id].append({'product': product, 'size': product_size, 'price': price})

    await message.answer( text: '✅ Додано до кошика. Щоб оформити замовлення, перейдіть в розділ 🛒 Оформити замовлення.'
                         reply_markup=main_menu_keyboard())

    await state.clear()
```

Рисунок 3.49 – Реалізація додавання товару в кошик

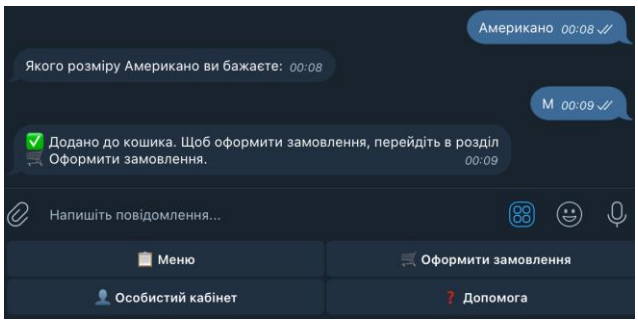


Рисунок 3.50 – Повідомлення про успішне додавання товару в кошик

Функціонал з вибором товарів готовий. Наразі, треба реалізувати оформлення замовлення. Відразу необхідно задуматися і про здійснення бальної системи, тому оголошуємо змінну bonuses у вигляді словника (рис. 3.51):

```
bonuses = {}
```

Рисунок 3.51 – Оголошення змінною bonuses

У цій функції ми виводимо клієнту повний кошик з усіма товарами, які він до нього додав. І розраховуємо кількість потенційних бонусних балів (bonuses[message.from_user.id][0][“potential bonuses”] = bonus), які він може отримати в разі, якщо оплатить замовлення.

```
@dp.message(lambda message: message.text == '🛒 Оформити замовлення')
async def start_checkout(message: Message):
    total_price = 0
    if not checkout or not checkout[message.from_user.id]:
        await message.answer( text: '🛒 Ваш кошик порожній. Додайте товари перед оформленням замовлення.',
                             reply_markup=main_menu_keyboard())
        return

    checkout_text = ''
    for product in checkout[message.from_user.id]:
        total_price += product['price']
        for product_info in product.values():
            if product_info != '':
                checkout_text += f'{product_info} '
        checkout_text += '\n'

    checkout_text += f'\n💰 До сплати: {total_price} грн'

    if int(total_price * 0.05) < 1:
        bonus = 1
    else:
        bonus = int(total_price * 0.05)

    bonuses[message.from_user.id][0]['potential bonuses'] = bonus

    await message.answer( text: f'Ваш кошик:\n\n{checkout_text}', reply_markup=checkout_keyboard())
```

Рисунок 3.52 – Інформація стосовно кошику та розрахунок потенційних бонусних балів

Надалі користувачеві надається вибір з наступних кнопок (рис. 3.53 – 3.54):

```
def checkout_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='💳 Сплатити'), KeyboardButton(text='❌ Скасувати замовлення'),
             KeyboardButton(text='✏ Редагувати')],
            [KeyboardButton(text='◀ До головного меню')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

Рисунок 3.53 – Меню для сплати/скасування/редагування кошику

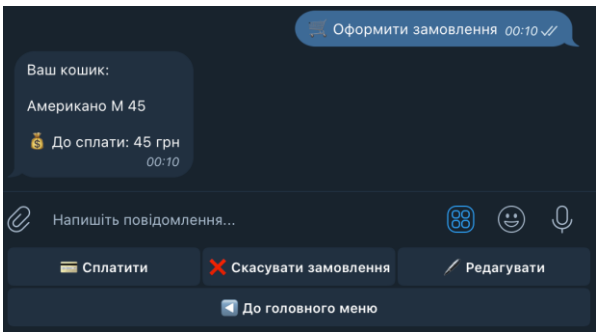


Рисунок 3.54 – Меню для сплати/скасування/редагування кошику

У разі, якщо користувач обирає варіант оплати, то йому стає доступним меню з варіантами способів оплати (рис. 3.55 – 3.57):

```
@dp.message(lambda message: message.text == '💳 Сплатити')
async def payment(message: Message):
    await message.answer( text: 'Оберіть спосіб оплати:', reply_markup=payment_keyboard())
```

Рисунок 3.55 – Перевірка того, чи натиснув користувач на кнопку « 💳 Сплатити»

```
def payment_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='💳 Картка'), KeyboardButton(text='💵 Готівка'), KeyboardButton(text='🎁 Бонусні бали')],
            [KeyboardButton(text='◀ До головного меню')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

Рисунок 3.56 – Варіанти способів оплати

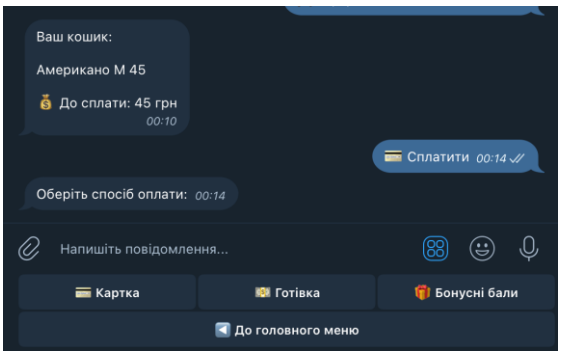


Рисунок 3.57 – Меню зі способами оплати

Розглянемо кожен спосіб окремо (рис. 3.58). У разі, якщо користувач вибирає оплату карткою, то його повинно переадресовувати на платіжну систему, яка буде підключена до боту. На даному етапі розробки бота, платіжна система не була підключена, а була просто зімітована. Тому, коли обрано спосіб оплати карткою, відразу приходить повідомлення про успішну оплату.

При оплаті готівкою, клієнт просто отримує повідомлення про те, що його замовлення прийнято.

Що стосується оплати балами, то замовлення ними можна оплатити тільки в тому випадку, якщо на рахунку користувача така ж сума балів або ж більше. Тобто кількість_балів * 0,5 >= загальної ціни.

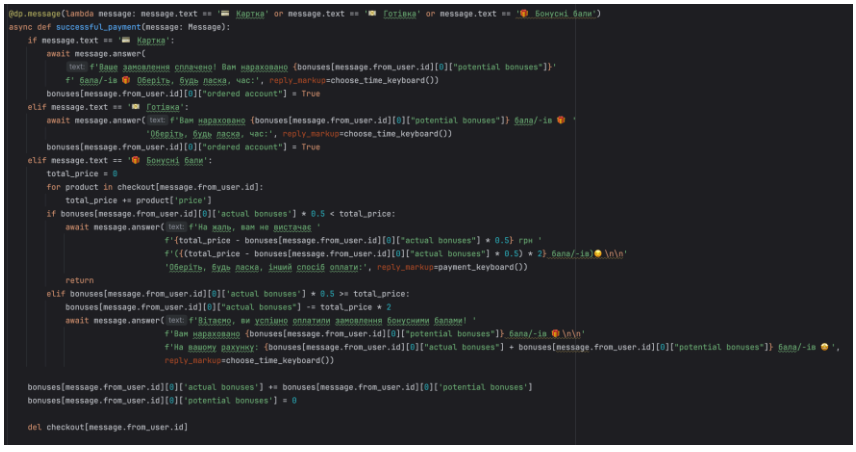


Рисунок 3.58 – Способи оплати замовлення

Як при оплаті карткою, так і при оплаті готівкою, відбувається переадресація на вибір часу отримання замовлення (рис. 3.59)

```python
def choose_time_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='⚡ Найближчим часом')],
            [KeyboardButton(text='⏰ На конкретний час')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

Рисунок 3.59 – Кнопки для вибору часу

При натисканні кнопки «найближчим часом», бот переходить у наступний стан, де йому буде надана можливість залишити коментар для бариста (рис. 3.60). У випадку, якщо замовлення потрібне на конкретну годину, завдяки функції generate_time_buttons(), яка генерує 3 варіанти часових кнопок за принципом, що замовлення як мінімум повинне замовлятися за 10 хвилин до прибуття клієнта в кав'ярню (рис. 3.61), надається можливість обрати 1 з 3 запропонованих варіантів часу, який більш за все буде влаштовувати клієнта (рис. 3.62):

```python
@dp.message(lambda message: message.text == '⚡ Найближчим часом' or message.text == '⏰ На конкретний час')
async def choosing_time(message: Message, state: FSMContext):
    if message.text == '⚡ Найближчим часом':
        await state.update_data(time='На найближчий час')
        await message.answer( text: f'Чи бажаєте залишити коментар для бариста? 💬', reply_markup=comment_keyboard())
    elif message.text == '⏰ На конкретний час':
        times = generate_time_buttons()
        await message.answer( text: 'Оберіть, будь ласка, час:', reply_markup=ReplyKeyboardMarkup(
            keyboard=[
                [KeyboardButton(text=f'{times[0]}'), KeyboardButton(text=f'{times[1]}'),
                 KeyboardButton(text=f'{times[2]}')]
            ],
            resize_keyboard=True
        ))
```

Рисунок 3.60 – Обирання часу

```python
def generate_time_buttons():
    now = datetime.now()

    total_minutes = now.hour * 60 + now.minute

    next_slot = ((total_minutes // 10) + 2) * 10

    times = []
    for i in range(3):
        future_minutes = next_slot + i * 10
        future_time = (datetime.combine(now.date(), datetime.min.time()) +
                       timedelta(minutes=future_minutes))
        times.append(future_time.strftime("%H:%M"))

    return times
```

Рисунок 3.61 – Функція generate_time_buttons()

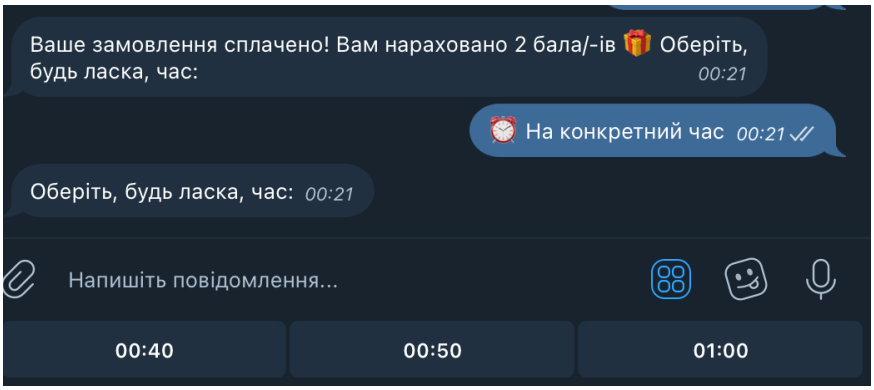


Рисунок 3.62 – Етап вибору часу

Останнім кроком в оформленні замовлення є коментар для бариста (рис. 3.63 – 3.66):

```python
@dp.message(lambda message: message.text in generate_time_buttons())
async def check(message: Message, state: FSMContext):
    await state.update_data(time=message.text)
    await message.answer(text: f'Чи бажаєте залишити коментар для бариста? 💬 ', reply_markup=comment_keyboard())
```

Рисунок 3.63 – Запит користувача чи бажає він залишити коментар

```python
def comment_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='💬 Залишити коментар'), KeyboardButton(text='😐 Не залишати коментар')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

Рисунок 3.64 – Меню для вибору

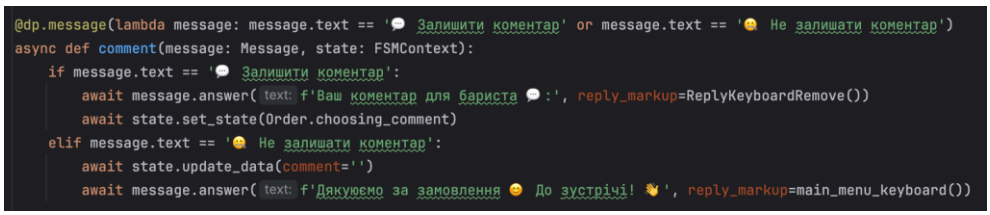


Рисунок 3.65 – Вибір між тим, щоб залишити коментар, та тим, щоб не залишати

```python
@dp.message(Order.choosing_comment)
async def give_comment(message: Message, state: FSMContext):
    await state.update_data(comment=message.text)
    await state.clear()
```

Рисунок 3.66 – Отримання коментарю

Оформлення замовлення тепер повністю готове. Тепер, нам його треба передати до бариста, щоб він/вона почали готувати замовлення. Робити ми це будемо через телеграм групу. Для цього, створимо змінну barista_message у вигляді словника (рис. 3.67):

```python
barista_message = {}
```

Рисунок 3.67 – Змінна для передавання замовлення бариста

Протягом реалізації даного бота, ми отримали всі необхідні дані для повідомлення бариста. Тому, у відповідних фрагментах коду збираємо відповідні дані (рис. 3.68 – 3.74):

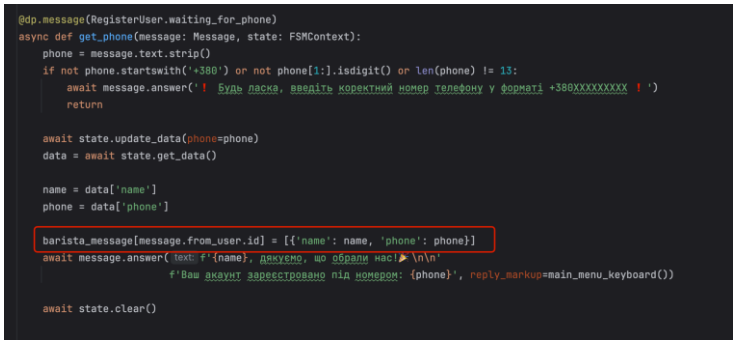
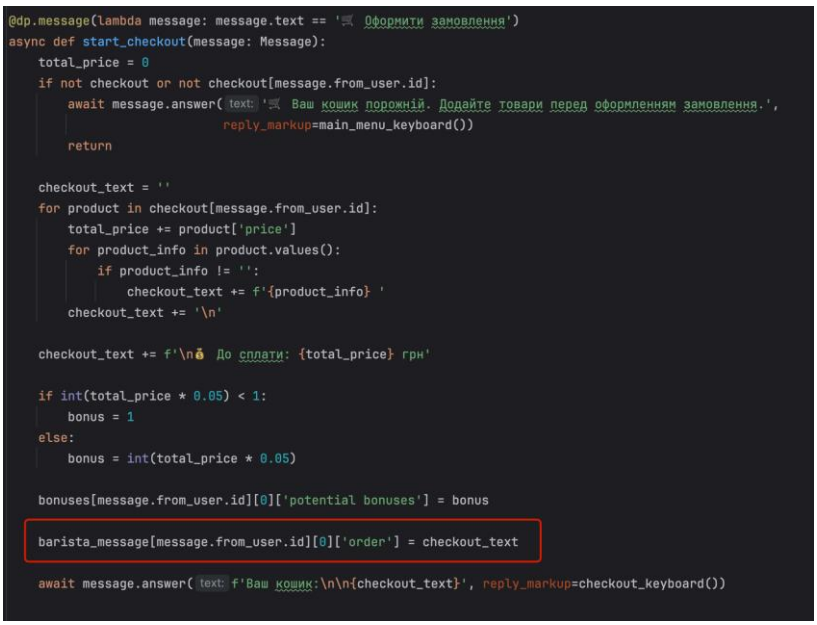
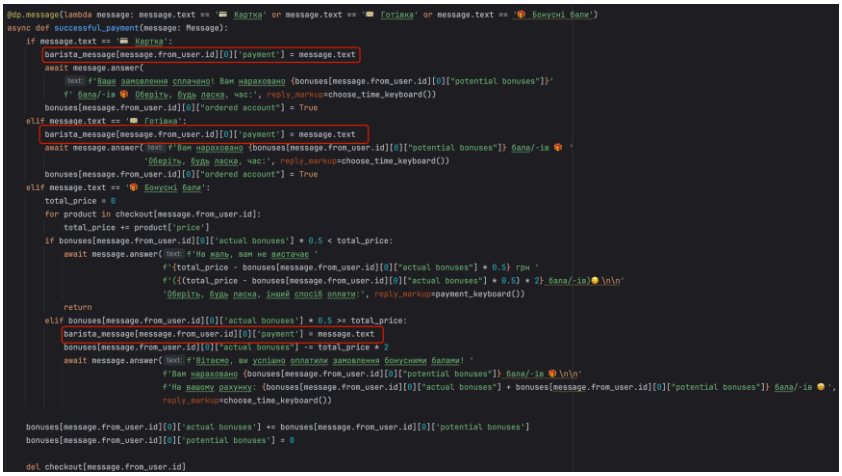


Рисунок 3.68 – На етапі реєстрації отримуємо ім’я та номер телефону



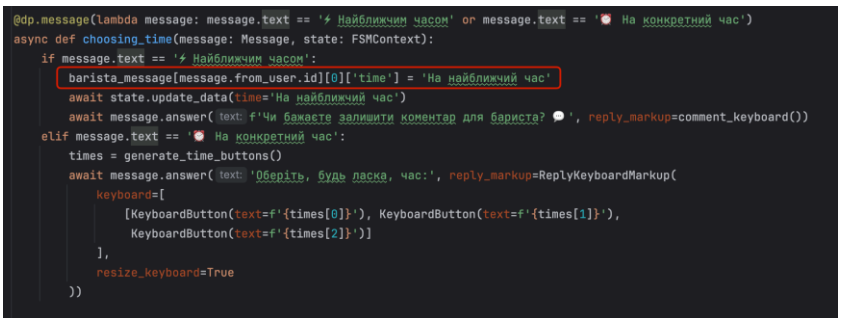
```python
@dp.message(lambda message: message.text == '🛒 Оформити замовлення')
async def start_checkout(message: Message):
    total_price = 0
    if not checkout or not checkout[message.from_user.id]:
        await message.answer( text: '🛒 Ваш кошик порожній. Додайте товари перед оформленням замовлення.',
                             reply_markup=main_menu_keyboard())
        return

    checkout_text = ''
    for product in checkout[message.from_user.id]:
        total_price += product['price']
        for product_info in product.values():
            if product_info != '':
                checkout_text += f'{product_info} '
        checkout_text += '\n'

    checkout_text += f'\n💰 До сплати: {total_price} грн'

    if int(total_price * 0.05) < 1:
        bonus = 1
    else:
        bonus = int(total_price * 0.05)

    bonuses[message.from_user.id][0]['potential bonuses'] = bonus

    barista_message[message.from_user.id][0]['order'] = checkout_text

    await message.answer( text: f'Ваш кошик:\n\n{checkout_text}', reply_markup=checkout_keyboard())
```

Рисунок 3.69 – На етапі надсилання повного кошику отримуємо інформацію про склад замовлення



```python
@dp.message(lambda message: message.text == '💳 Картка' or message.text == '💵 Готівка' or message.text == '🎁 Бонусні бали')
async def successful_payment(message: Message):
    if message.text == '💳 Картка':
        barista_message[message.from_user.id][0]['payment'] = message.text
        await message.answer(
            text: f'Ваше замовлення сплачено! Вам нараховано {bonuses[message.from_user.id][0]["potential bonuses"]}'
            f' бала/-ів 🎁. Оберіть, будь ласка, час:', reply_markup=choose_time_keyboard())
        bonuses[message.from_user.id][0]["ordered account"] = True
    elif message.text == '💵 Готівка':
        barista_message[message.from_user.id][0]['payment'] = message.text
        await message.answer( text: f'Вам нараховано {bonuses[message.from_user.id][0]["potential bonuses"]} бала/-ів 🎁 '
                             'Оберіть, будь ласка, час:', reply_markup=choose_time_keyboard())
        bonuses[message.from_user.id][0]["ordered account"] = True
    elif message.text == '🎁 Бонусні бали':
        total_price = 0
        for product in checkout[message.from_user.id]:
            total_price += product['price']
        if bonuses[message.from_user.id][0]['actual bonuses'] * 0.5 < total_price:
            await message.answer( text: f'На жаль, вам не вистачає '
                                 f'{total_price - bonuses[message.from_user.id][0]["actual bonuses"] * 0.5} грн '
                                 f'({(total_price - bonuses[message.from_user.id][0]["actual bonuses"] * 0.5) * 2} бала/-ів)😔\n\n'
                                 'Оберіть, будь ласка, інший спосіб оплати:', reply_markup=payment_keyboard())
            return
        elif bonuses[message.from_user.id][0]['actual bonuses'] * 0.5 >= total_price:
            barista_message[message.from_user.id][0]['payment'] = message.text
            bonuses[message.from_user.id][0]["actual bonuses"] -= total_price * 2
            await message.answer( text: f'Вітаємо, ви успішно оплатили замовлення бонусними балами! '
                                 f'Вам нараховано {bonuses[message.from_user.id][0]["potential bonuses"]} бала/-ів 🎁\n\n'
                                 f'На вашому рахунку: {bonuses[message.from_user.id][0]["actual bonuses"] + bonuses[message.from_user.id][0]["potential bonuses"]} бала/-ів 🎁.',
                                 reply_markup=choose_time_keyboard())

    bonuses[message.from_user.id][0]['actual bonuses'] += bonuses[message.from_user.id][0]['potential bonuses']
    bonuses[message.from_user.id][0]['potential bonuses'] = 0

    del checkout[message.from_user.id]
```

Рисунок 3.70 – На етапі оплати отримуємо спосіб оплати



```python
@dp.message(lambda message: message.text == '⚡ Найближчим часом' or message.text == '⏰ На конкретний час')
async def choosing_time(message: Message, state: FSMContext):
    if message.text == '⚡ Найближчим часом':
        barista_message[message.from_user.id][0]['time'] = 'На найближчий час'
        await state.update_data(time='На найближчий час')
        await message.answer( text: f'Чи бажаєте залишити коментар для бариста? 💬 ', reply_markup=comment_keyboard())
    elif message.text == '⏰ На конкретний час':
        times = generate_time_buttons()
        await message.answer( text: 'Оберіть, будь ласка, час:', reply_markup=ReplyKeyboardMarkup(
            keyboard=[
                [KeyboardButton(text=f'{times[0]}'), KeyboardButton(text=f'{times[1]}'),
                 KeyboardButton(text=f'{times[2]}')]
            ],
            resize_keyboard=True
        ))
```

Рисунок 3.71 – На етапі обирання часу отримаємо час, на котрий треба приготувати замовлення

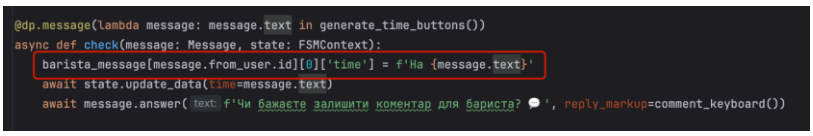


Рисунок 3.72 – На етапі обирання часу отримаємо час, на котрий треба приготувати замовлення

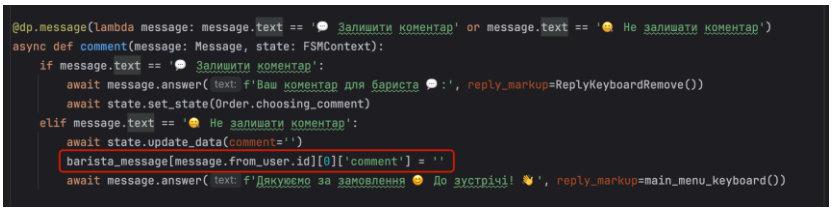


Рисунок 3.73 – На етапі залишення коментаря у разі, якщо немає коментаря, отримуємо пусте значення

Рисунок 3.74 – На етапі додавання коментаря отримуємо коментар

Також, на рис. 3.74 можна побачити, що за допомогою методу bot.send_message() замовлення надсилається бариста в групу. Надсилання в группу реалізовано за допомогою параметру chat_id, значення для якого ми отримали в самій групі, куди будуть надсилатись повідомлення.

Рисунок 3.75 – Значення константи COFFEE_ORDERS_GROUP

Для того, щоб отримати chat_id групи, необхідно додати бота як адміністратора в групу (рис. 3.76), після чого створити функцію get_chat_id()

(рис. 3.77), яка після надсилання будь-якого повідомлення в цю групу, надасть нам chat_id від неї (рис. 3.78):

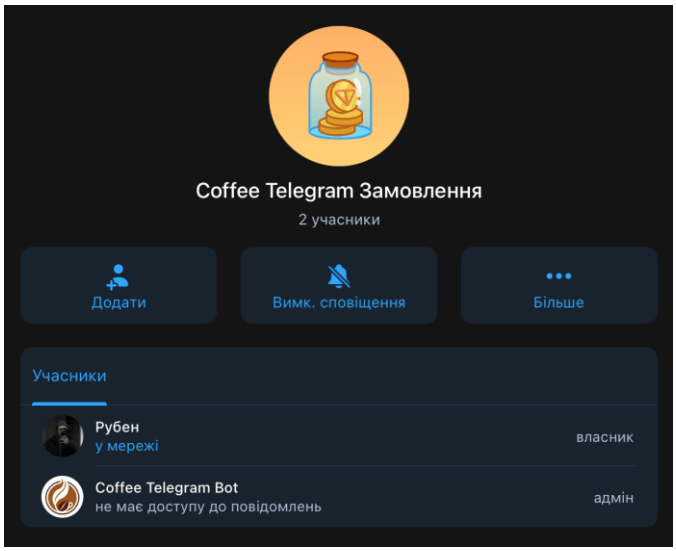


Рисунок 3.76 – Група для отримання та фіксації замовлень

```
@dp.message()
async def get_chat_id(message: Message):
    await bot.send_message(chat_id=message.chat.id, text=f'Чат ID цієї групи: {message.chat.id}')
```

Рисунок 3.77 – Функція для отримання chat_id

Рисунок 3.78 – chat_id від групи «Coffee Telegram Замовлення»

Після того, як ми отримали chat_id та форматували повідомлення для бариста, на виході в групі будуть фіксуватися замовлення в такому форматі (рис. 3.79):

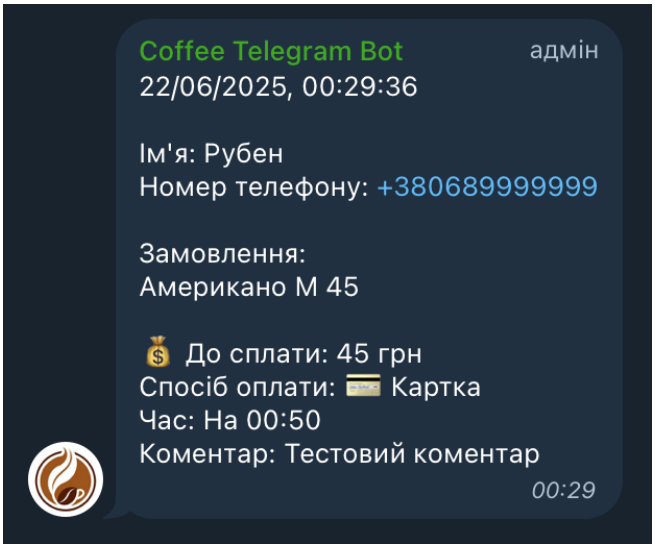


Рисунок 3.79 – Повідомлення про оформлення замовлення

Також, необхідно реалізувати можливість скасовувати замовлення та редагувати його на етапі формування замовлення. У випадку з скасування все легко – якщо користувач надсилає повідомлення про скасування замовлення, всі дані про замовлення, які були надіслані за його user_id просто видаляються (рис. 3.80):

```
@dp.message(lambda message: message.text == '❌ Скасувати замовлення')
async def cancel_checkout(message: Message):
    del checkout[message.from_user.id]
    await message.answer( text: 'Замовлення скасовано! ❌ Ваш кошик очищений 🛒 ', reply_markup=main_menu_keyboard())
```

Рисунок 3.80 – Скасування замовлення

Якщо ж клієнт хоче видалити саме якийсь один товар зі всього замовлення, то він може скористуватися функцією редагування замовлення, за допомогою якого можна видалити окремі товари зі всього замовлення (рис. 3.81 – 3.82):

```python
@dp.message(lambda message: message.text == '✏ Редагувати')
async def edit_checkout(message: Message):
    if not checkout[message.from_user.id]:
        await message.answer(text='🛒 Ваш кошик порожній. Додайте товари перед оформленням замовлення.',
                             reply_markup=main_menu_keyboard())
    buttons = []
    products = checkout.get(message.from_user.id)
    for i in range(len(products)):
        buttons.append([KeyboardButton(text=f'Видалити позицію # {str(i + 1)}')])

    buttons.append([KeyboardButton(text='◀ До головного меню')])

    keyboard = ReplyKeyboardMarkup(
        keyboard=buttons,
        resize_keyboard=True
    )

    await message.answer( text: 'Оберіть позицію, яку бажаєте видалити:', reply_markup=keyboard)


@dp.message(lambda message: 'Видалити позицію #' in message.text)
async def remove_from_checkout(message: Message):
    removed_product = checkout[message.from_user.id].pop(int(message.text.split(' ')[-1]) - 1)
    await message.answer( text: f'{removed_product["product"]} {removed_product["size"]} видалено з кошику ✅ ',
                         reply_markup=main_menu_keyboard())
```

Рисунок 3.81 – Редагування кошику

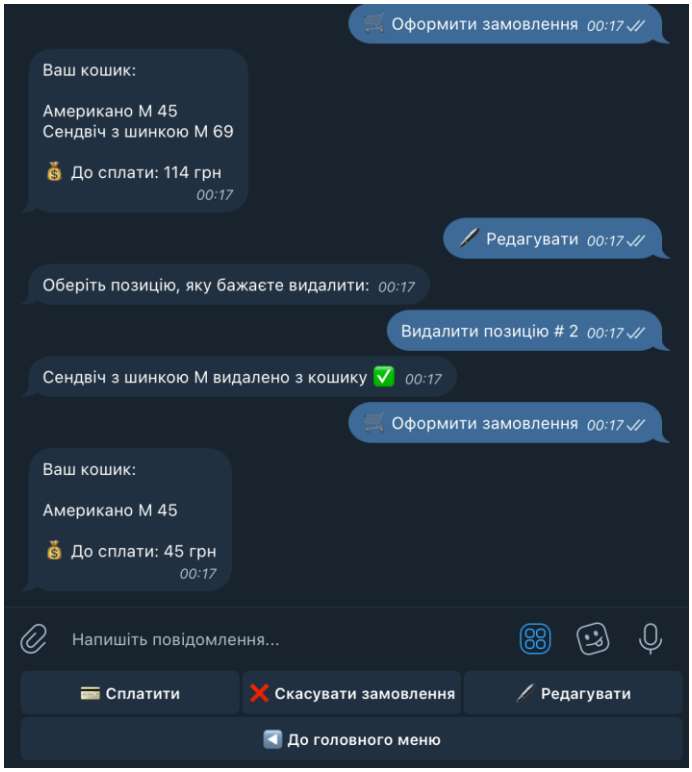


Рисунок 3.82 – Редагування кошику

Справа доходить до особистого кабінету. У ньому користувач повинен мати можливість переглянути бали, змінити персональну інформацію та залишити відгук. Тому відображаємо ці кнопки в меню навігації в особистому кабінеті (рис. 3.83 – 3.84):

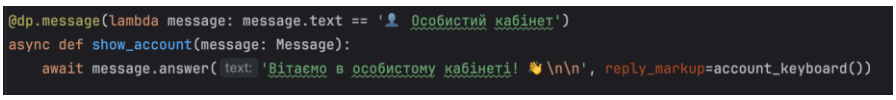


Рисунок 3.83 – Вхід в особистий кабінет

Рисунок 3.84 – Меню навігації в особистому кабінеті

Щоб переглянути баланс бонусних балів, нам достатньо було знову звернутися до змінної bonuses, яка як раз і містить ці дані. Якщо користувач хоче змінити свої персональні дані, йому просто необхідно натиснути відповідну кнопку, яка буде знову запускати процес реєстрації користувача (рис. 3.85)

Залишити відгук, як і планувалося, користувач зможу тільки в тому випадку, якщо він колись робив замовлення. Це реалізовано за допомогою змінної bonuses та ключу «ordered account», яке за замовчуванням має значення False, але після оформлення будь-якого замовлення перетворюється на True. Залишений відгук, як і замовлення, відправляється у відповідну групу (рис. 3.85 – 3.88).

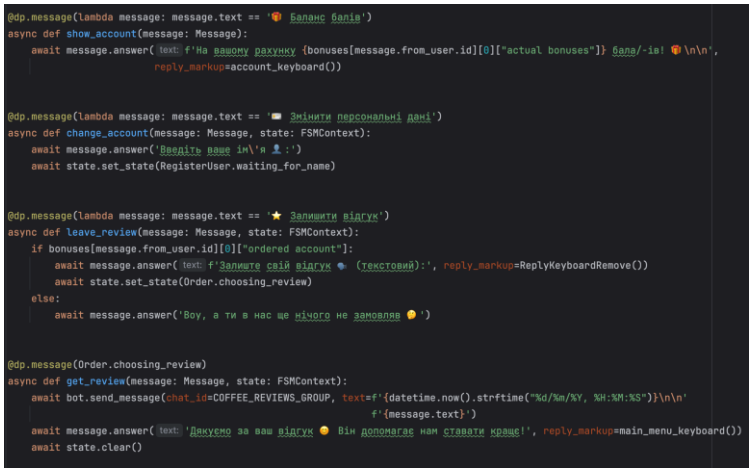


Рисунок 3.85 – Реалізація особистого кабінету

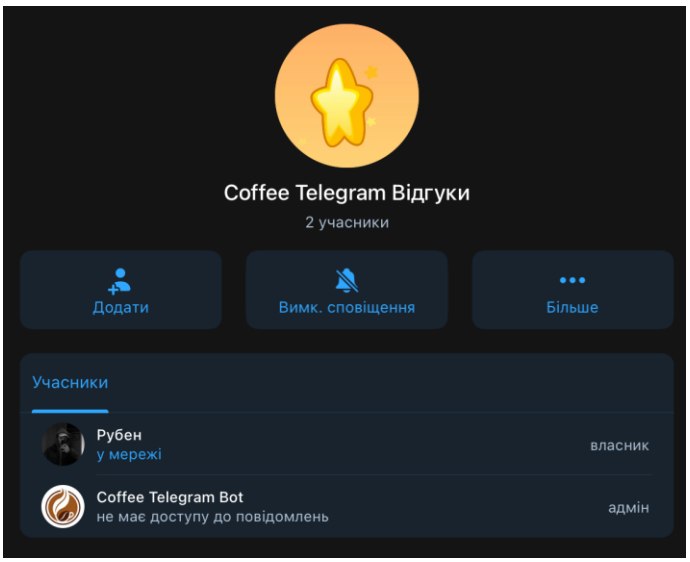


Рисунок 3.86 – Створена група для відгуків

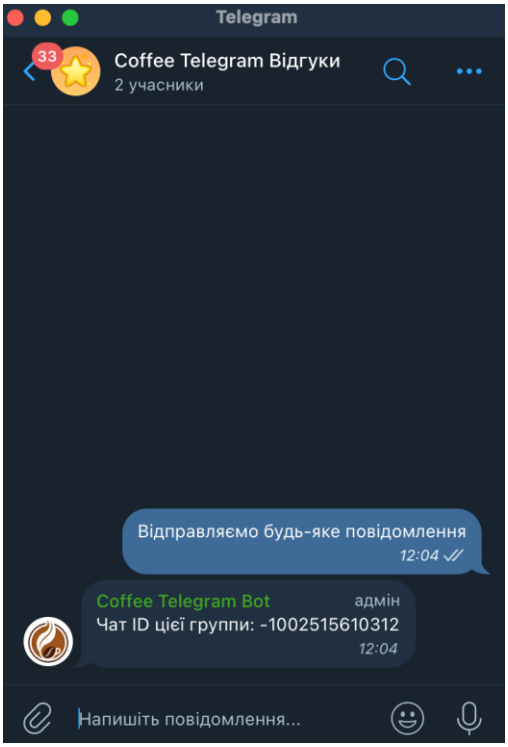


Рисунок 3.87 – Отримання chat_id групи для відгуків

```
COFFEE_REVIEWS_GROUP = -1002515610312
```

Рисунок 3.88 – chat_id групи «Coffee Telegram Відгуки»

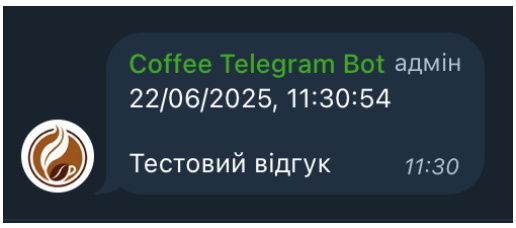


Рисунок 3.89 – Відгук в групі

Останнім етапом в розробці боту є кнопка «Допомога», яка надсилатиме клієнту інформацію з питаннями, які найчастіше виникають у відвідувачів кав'ярні та користувачів боту (рис. 3.90)

```
@dp.message(lambda message: message.text == '❓ Допомога')
async def show_help(message: Message):
    await message.answer('Виникли питання? 🤔\n'
                         'Ось список найчастіших питань та відповіді на них:\n\n'

                         '❓ Що буде, якщо я не заберу своє замовлення три рази поспіль?\n\n'
                         'Якщо ви не забираєте замовлення три рази поспіль без попередження, ваш обліковий запис може '
                         'бути тимчасово заблокований в боті, щоб уникнути можливих зловживань. Рекомендуємо завжди '
                         'повідомляти про неможливість забрати замовлення заздалегідь.\n\n'

                         '❓ Що буде, якщо я не заберу своє замовлення три рази поспіль?\n\n'
                         'Так, ви з легкістю можете це зробити, вказавши всю необхідну інформацію в коментарі на етапі '
                         'оформлення замовлення.\n\n'

                         '❓ Які у вас години роботи?\n\n'
                         'Ми працюємо щодня з 8:00 до 22:00.\n\n'

                         '❓ Чи можна оплатити замовлення балами + своїми коштами?\n\n'
                         'На жаль, оплата бонусними балами доступна тільки в тому випадку, якщо вона покриває повну'
                         'вартість замовлення.\n\n'

                         '❓ Що робити, якщо у мене виникли проблеми з оплатою?\n\n'
                         'Якщо оплата не пройшла або виникли проблеми з оплатою, зверніться до бариста.\n\n'

                         'Не знайшов відповідь на своє питання? Звернись до бариста!')
```

Рисунок 3.90 – Список питань, що задають найчастіше

3.4 Керівництво користувача

Щоб розпочати роботу з ботом, достатньо в пошуку прописати @hnumg_coffee_telegram_bot та натиснути кнопку «Розпочати» (рис. 3.91):

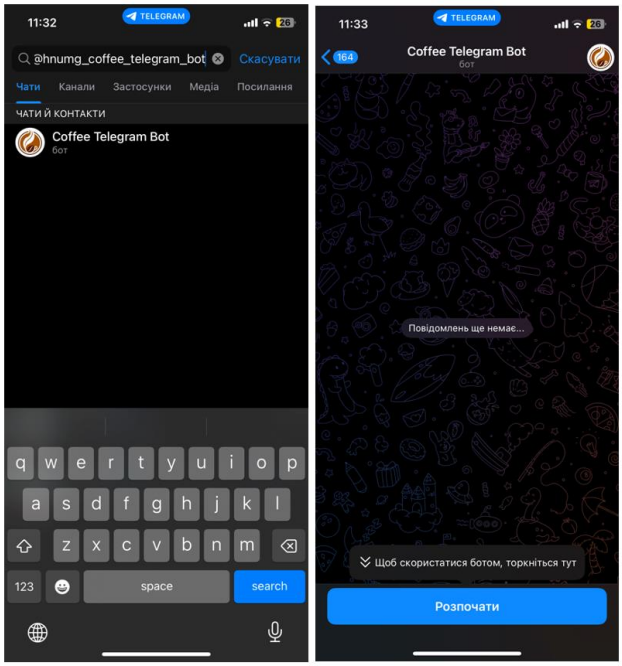


Рисунок 3.91 – Початок роботи з ботом

Далі, наступає етап реєстрація, після якого доступне меню з основними кнопка навігації в боті (рис. 3.92):

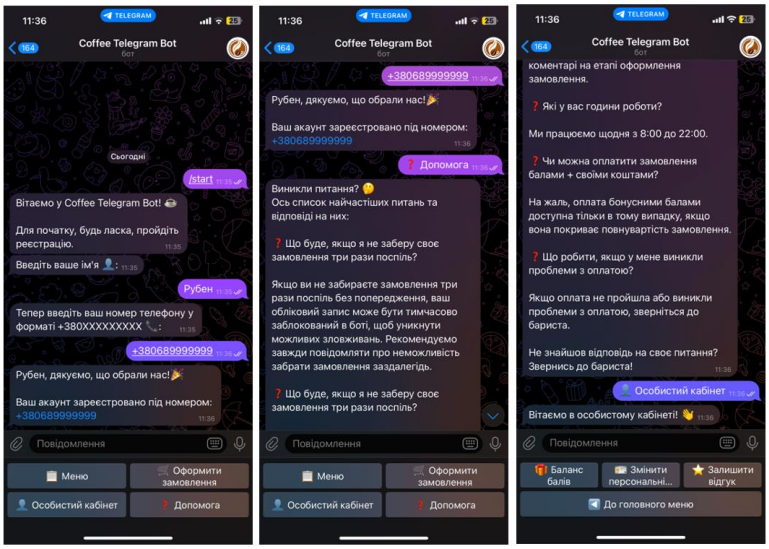


Рисунок 3.92 – Реєстрація, розділ «Допомога» та вхід в особистий кабінет

Після натискання кнопки меню, стає можливим вибір між категоріями, а потім – між товарами. Також, є можливість обрати розмір, вслід за чим товар додається у кошик (рис. 3.93):

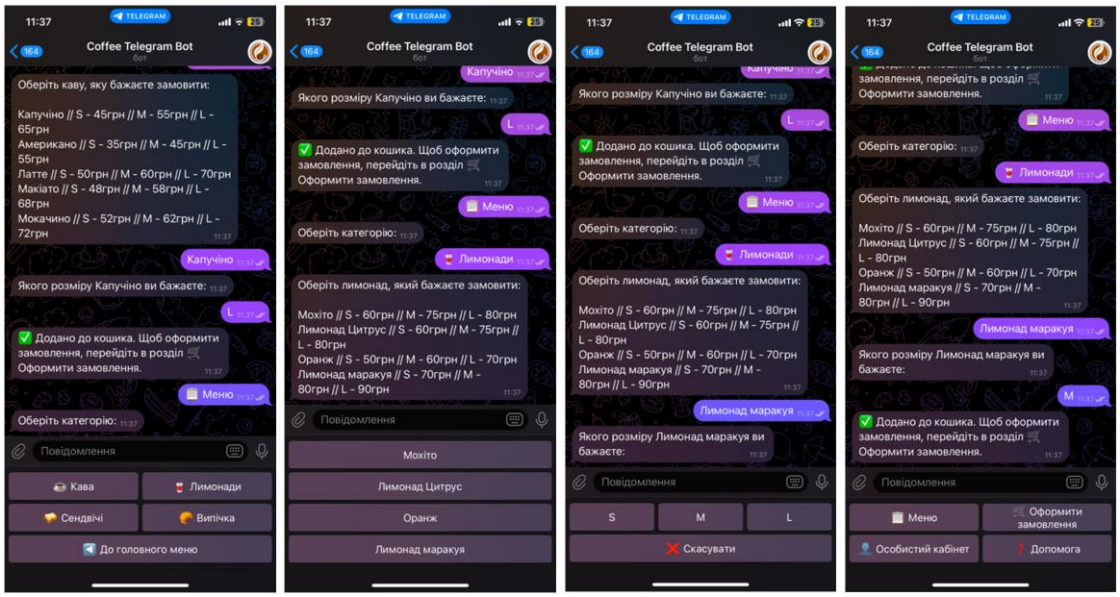


Рисунок 3.93 – Меню, процес додавання товару в замовлення

В оформлені замовлення (кошику) можна як скасовувати все замовлення, так і редагувати його, видаляючи окремі товари з замовлення (рис. 3.94):

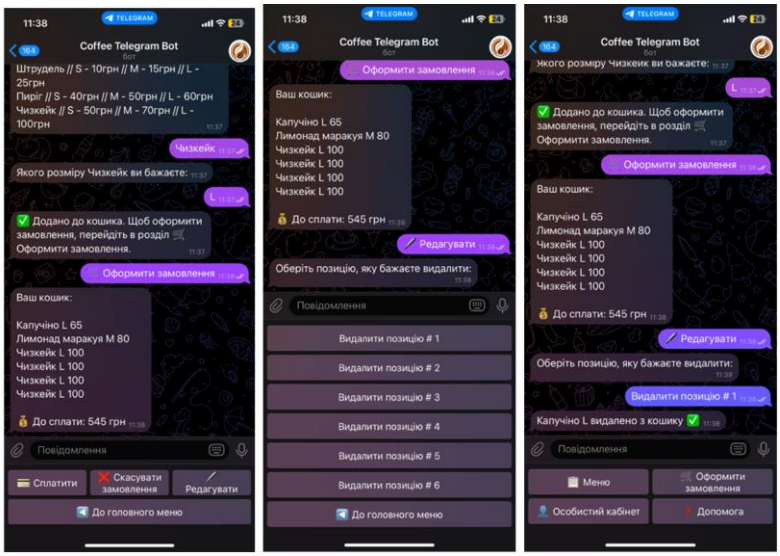


Рисунок 3.94 – Оформлення замовлення (кошик), редагування кошику

Додавання всіх необхідних товарів, використовуючи кнопку «Сплатити», можна перейти на етап обирання способу оплати:

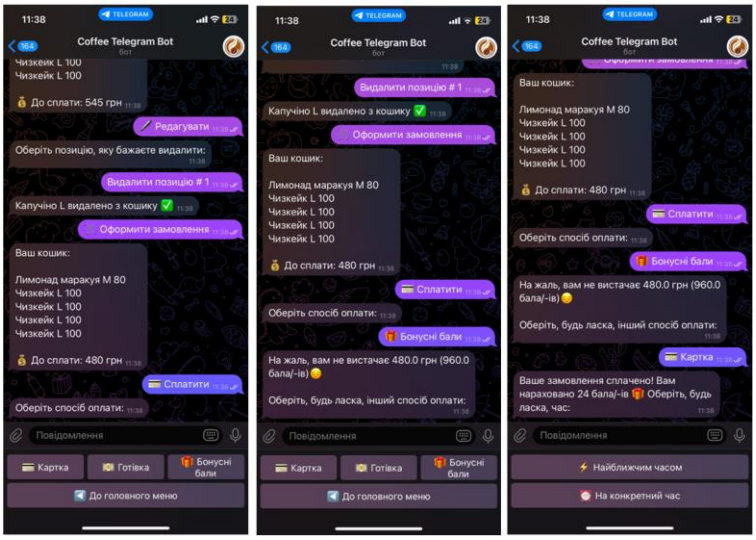


Рисунок 3.95 – Вибір способу оплати

Далі надається вибір часу та можливість надати коментар до замовлення (рис. 3.96):

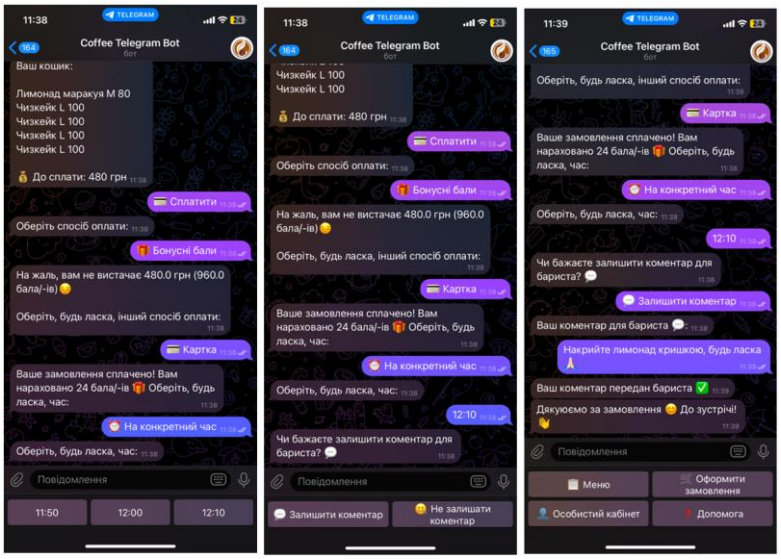


Рисунок 3.96 – Оформлення замовлення, вибір часу, коментар для бариста

Після оформлення замовлення нараховуються бали, які в майбутньому можна використати для сплати замовлення (рис. 3.97):

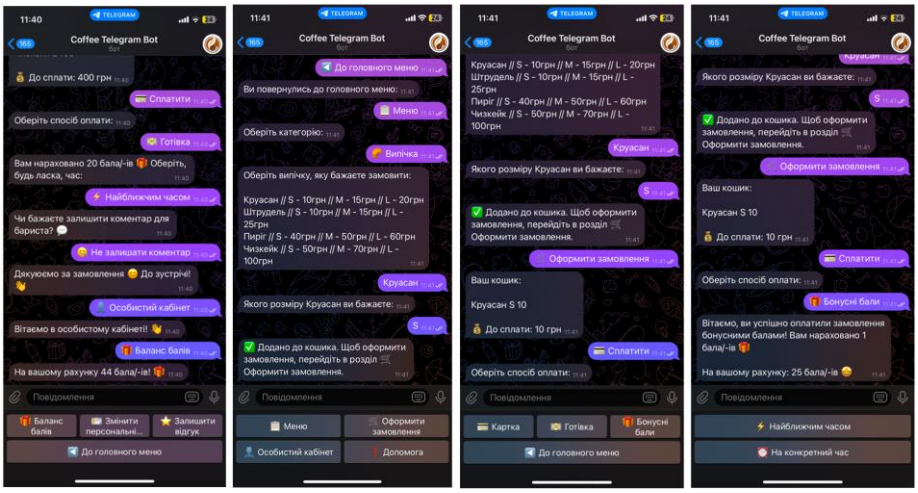


Рисунок 3.97 – Нарахування балів, оформлення замовлення бонусними балами

В особистому кабінеті надається можливість змінити персональні дані, перевірити баланс бонусів та написати відгук про кав’ярню (рис. 3.98):

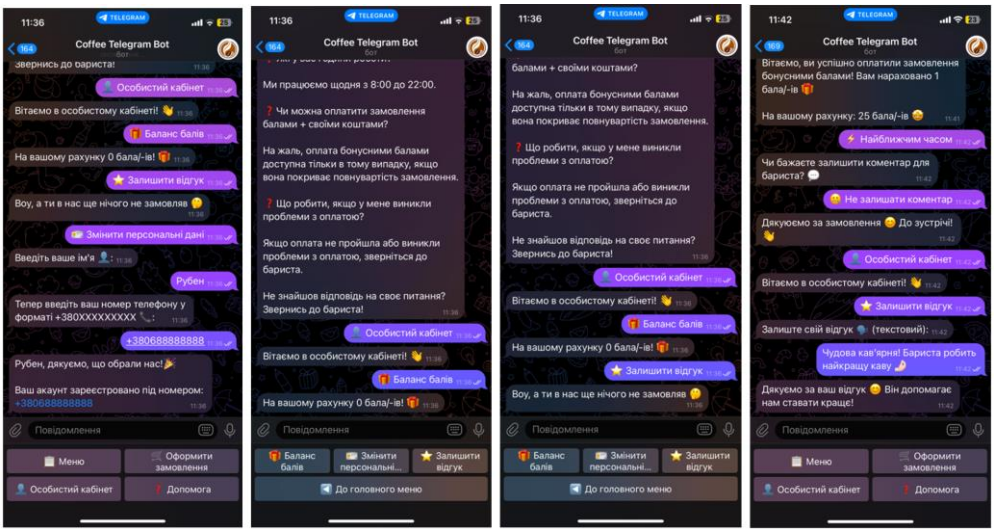


Рисунок 3.98 – Зміна персональних даних, баланс бонусів, запис відгуку

Висновки до розділу

Було проведено детальну характеристику програмної реалізації Telegram-бота, призначеного для автоматизації процесу оформлення замовлень у кав'ярні. Представлений програмний продукт розроблений з використанням мови програмування Python і бібліотеки Aiogram, що

обумовлено високим ступенем інтеграції останньої з Telegram Bot API, а також широкими можливостями асинхронної обробки подій і гнучкою архітектурою.

Запропонований підхід дозволяє забезпечити ефективну, інтуїтивно зрозумілу і послідовну взаємодію з користувачем. Використання механізму кінцевого автомата станів (FSM) дозволило організувати покрокову логіку спілкування, що особливо актуально для багатоетапних процесів, таких як реєстрація, оформлення замовлення, вибір способу оплати та уточнення часу отримання.

Було реалізовано безліч функцій, кожна з яких відповідає за конкретний етап або елемент бізнес-логіки. Користувач має можливість вибрати продукцію з декількох категорій (напої, випічка, сендвічі тощо), вказати бажаний розмір, залишити коментар до замовлення, вибрати час отримання, а також переглянути або змінити особисті дані в особистому кабінеті. Особлива увага приділена зручності та доступності інтерфейсу, що реалізовано через клавіатури з кнопками, що забезпечують швидкий доступ до основних функцій бота.

Також варто відзначити реалізацію бонусної системи лояльності, що дозволяє заохочувати клієнтів за оформлення замовлень. Це сприяє формуванню постійної клієнтської бази і стимулює повторні покупки. Можливість використання бонусних балів для часткової або повної оплати робить взаємодію з ботом не тільки функціональною, але і привабливою для користувачів.

Важливим аспектом є і автоматична відправка даних замовлення в Telegram-групу бариста, що забезпечує оперативне отримання інформації і мінімізує людський фактор. Таким чином, досягається синхронізація між системою оформлення замовлення і безпосереднім виконавцем, що підвищує швидкість обслуговування і зменшує ймовірність помилок. практичний досвід у розробці прикладних телекомунікаційних рішень.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ

4.1 Регулювання питань охорони праці на законодавчому рівні

Охорона праці на українських підприємствах являє собою комплекс правових норм і вимог, які регулюються чинним законодавством. Ці норми встановлюють обов'язки роботодавця в питаннях забезпечення безпечних умов праці. Вони можуть бути різними залежно від сфери діяльності, особливостей підприємства та інших факторів.

Організація грамотної системи охорони праці безпосередньо впливає на облаштування робочих місць, організацію виробничого процесу та загальну безпеку співробітників. В Україні існують фундаментальні правові принципи, на яких будується підхід до охорони праці, і саме вони служать основою для формування відповідної політики на будь-якому підприємстві. [15]

- пріоритетне значення має захист життя і здоров'я працівників порівняно з економічними інтересами підприємств;
- роботодавець несе повну відповідальність за забезпечення безпечних, відповідних санітарним і гігієнічним вимогам умов праці;
- передбачена державна гарантія соціальної підтримки працівників, включаючи повне відшкодування збитків особам, які постраждали внаслідок професійних захворювань або нещасних випадків на виробництві;
- діяльність із забезпечення охорони праці повинна здійснюватися на професійній та компетентній основі;
- дотримання вимог охорони праці є обов'язковим як для роботодавця, так і для працівника, що підкреслює двосторонню відповідальність сторін трудового процесу;
- передбачені механізми захисту трудових прав працівників, у тому числі гарантії реалізації права на безпечні та здорові умови праці;

- обов'язок щодо компенсації шкоди, заподіяної працівникові в процесі виконання ним трудових функцій, є невід'ємною частиною системи охорони праці.

Основу правового регулювання охорони праці в Україні складають наступні нормативно-правові акти:

- Кодекс законів про працю України (КЗпП) [11];
- Закон України «Про охорону праці» [12];
- Державні санітарні правила і норми;
- правила пожежної безпеки, норми охорони праці та ін.

Закон України «Про охорону праці» визначає обов'язки всіх суб'єктів трудових відносин. Зокрема, роботодавець зобов'язаний забезпечити належні умови праці, поінформувати працівників про існуючі ризики, організувати навчання з охорони праці та забезпечити засобами індивідуального захисту. Фахівець з охорони праці або відповідальна посадова особа контролює виконання вимог законодавства, проводить інструктажі та моніторинг стану робочих місць.

Працівник, у свою чергу, зобов'язаний дотримуватися норм охорони праці, використовувати засоби індивідуального захисту, проходити інструктаж і повідомляти про виникнення небезпечних ситуацій. Крім того, державні органи, такі як Державна служба України з питань праці, здійснюють контроль і нагляд за дотриманням законодавства в цій сфері.

4.2 Аналіз можливих джерел небезпеки в рамках об'єкта проєктування

Об'єктом проєктування є інформаційний продукт – Telegram-бот, призначений для обслуговування клієнтів кав'ярні. Незважаючи на віддалений характер розробки, охорона праці в сфері інформаційних технологій залишається актуальною, а також слід враховувати ризики, що виникають в самій кав'ярні – кінцевій точці реалізації проєкту.

Потенційні небезпеки на робочому місці розробника:

- тривале перебування за комп'ютером (зір, опорно-руховий апарат);
- електромагнітне випромінювання від обладнання;
- електротравми при несправності техніки;
- стресові та розумові перевантаження, монотонність праці;
- несприятливі умови мікроклімату в приміщенні. [10]

Слід зазначити, що крім потенційних загроз, пов'язаних з професійною діяльністю програміста, який, як правило, здійснює свою роботу дистанційно і не знаходиться безпосередньо на території кав'ярні, існує значна кількість факторів ризику, властивих самому виробничому середовищу кавового закладу.

Умови праці в кав'ярні супроводжуються впливом безлічі несприятливих факторів, які можуть становити небезпеку як для бариста, так і для інших співробітників, які знаходяться в безпосередній близькості до обладнання, джерел тепла, електромереж та інших технологічних елементів.

Таким чином, при розгляді питань охорони праці важливо приділяти увагу не тільки умовам праці розробників програмного забезпечення, але і більш вираженим і безпосереднім ризикам, властивим самій кав'ярні як виробничій і обслуговувальній одиниці. Серед потенційних небезпек на об'єкті застосування (в кав'ярні) слід зазначити:

- опіки при роботі з гарячими рідинами та обладнанням;
- порізи при використанні інвентарю;
- пожежна небезпека через використання нагрівальних приладів;
- слизькі підлоги, падіння і травми;
- хімічні речовини (миючі засоби);
- загальні загрози: пожежа, стихійні лиха, військові дії. [10, 13]

Додатково, для бариста – основної категорії працівників кав'ярні – характерні такі види професійних ризиків:

- частий контакт з гарячими напоями та парою, що збільшує ймовірність опіків;

- повторювані рухи та тривале стояння, що сприяє розвитку захворювань суглобів та опорно-рухового апарату;
- шум від обладнання (кавомолки, кавомашини), що впливає на слух і загальне самопочуття;
- ризик психоемоційного вигорання через високу щільність взаємодії з клієнтами;
- недостатня вентиляція, особливо в години пік, що погіршує мікроклімат і підвищує стомлюваність.

4.3 Оцінка ризиків виникнення небезпек та шляхи їх запобігання

Оцінка професійних ризиків – це систематичний і послідовний процес, спрямований на виявлення, аналіз і мінімізацію можливих загроз здоров'ю та життю працівників. Цей процес дозволяє завчасно визначити існуючі та потенційні небезпеки, оцінити ймовірність їх реалізації, а також спрогнозувати можливі наслідки.

Основні цілі оцінки ризику:

- підвищення рівня безпеки на робочому місці;
- зниження кількості нещасних випадків і професійних захворювань;
- обґрунтування заходів щодо запобігання та зниження впливу небезпечних факторів;
- поліпшення умов праці та підвищення загальної ефективності діяльності.

Серед найбільш застосовуваних методів оцінки ризику – матричний підхід і метод «дерева відмов». Матричний метод дозволяє класифікувати рівень небезпеки на основі поєднання двох параметрів: ймовірності реалізації та тяжкості наслідків. Метод «дерева відмов» допомагає візуалізувати та логічно простежити можливі шляхи розвитку інцидентів і запобігти їх реалізації.

Одним з найбільш поширених професійних ризиків в умовах роботи кав'ярні є отримання термічних опіків бариста при взаємодії з гарячим обладнанням і рідинами. З огляду на високу інтенсивність праці, обмежений простір і постійну експлуатацію термонагрівальних приладів (кавомашини, парогенератори тощо), ймовірність подібних інцидентів залишається значною. Для системного аналізу даної небезпеки доцільно використовувати метод «дерева відмов», що дозволяє наочно відобразити сукупність факторів, що сприяють виникненню даної ситуації, а також визначити точки контролю і можливі превентивні заходи (рис. 4.1):

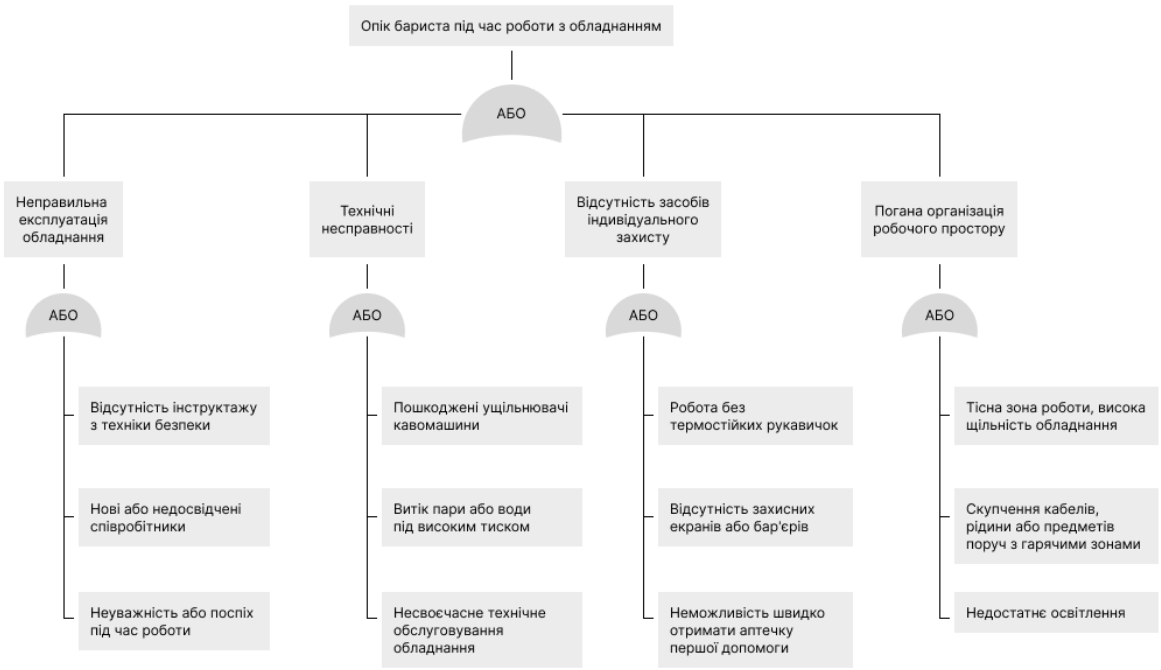


Рисунок 4.1 – Аналіз небезпеки опіку бариста під час роботи з обладнанням методом «дерева відмов»

Проведений аналіз з використанням методу «дерева відмов» дозволяє систематизувати основні джерела ризику і визначити, які саме елементи робочого середовища та організаційного процесу вимагають особливої уваги. Зокрема, очевидна необхідність регулярного технічного обслуговування обладнання, підвищення рівня поінформованості персоналу, а також суворого дотримання правил безпеки при поводженні з гарячими рідинами. Тільки комплексний підхід, що включає як технічні, так і організаційні заходи,

дозволить значно знизити ймовірність отримання опіків і забезпечити безпечні умови праці для співробітників кав'ярні.

Таким чином, дерево відмов виникнення пожежі дозволяє виділити ключові технічні та організаційні фактори, що сприяють даній загрозі, і формує основу для розробки запобіжних заходів. Однак при забезпеченні безпеки працівників важливо враховувати не тільки фізичні, але й психоемоційні аспекти трудової діяльності, особливо в умовах сфери обслуговування, де персонал постійно взаємодіє з клієнтами і працює в динамічній обстановці.

Наступним значущим фактором професійного ризику є психоемоційне напруження, яке має істотний вплив на працездатність, емоційний стан і загальне благополуччя працівників кав'ярні. Його природа, на відміну від технічних загроз, носить комплексний характер і визначається поєднанням організаційних, соціальних, психологічних і фізичних факторів. Для більш глибокого аналізу даного ризику доцільно використовувати метод «дерева відмов», який представлений нижче (рис. 4.2)

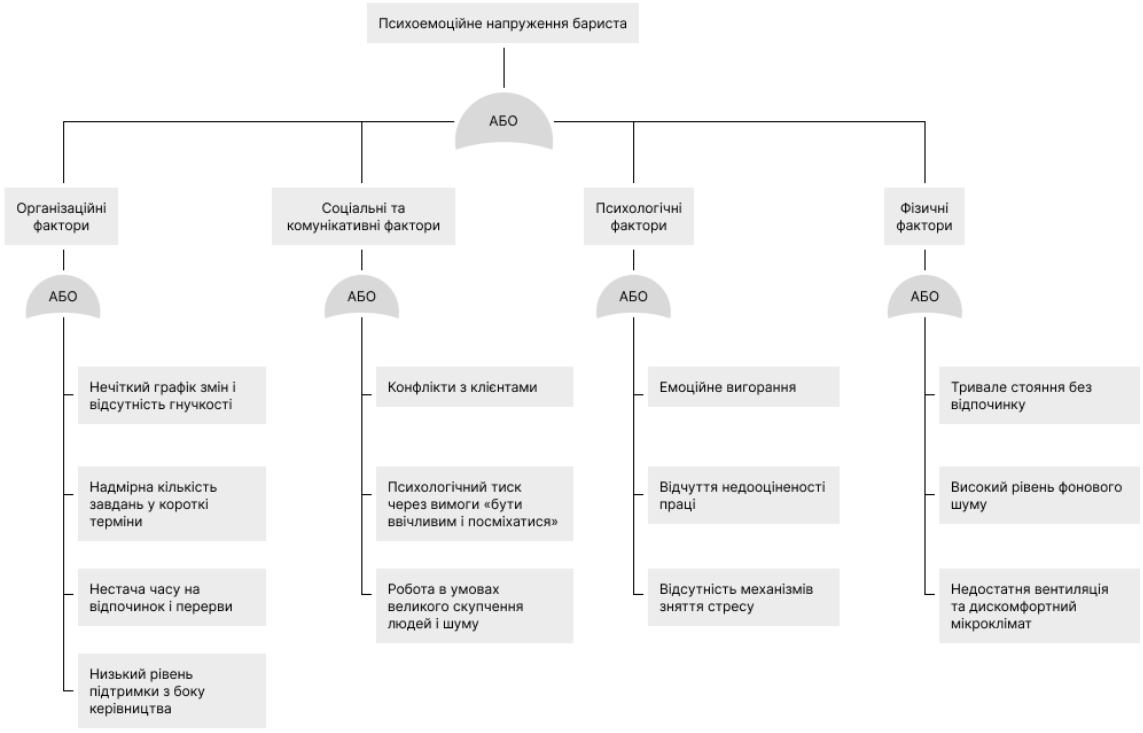


Рисунок 4.2 – Аналіз психоемоційного напруження бариста методом «дерева відмов»

Проблема психоемоційного напруження бариста обумовлена не тільки інтенсивністю та умовами праці, але й якістю організаційної культури та рівнем підтримки персоналу. Високе навантаження, брак часу на відпочинок, відсутність емоційної підтримки з боку керівництва та конфліктні ситуації з клієнтами – все це створює стійке стресове середовище, здатне призвести до зниження продуктивності, підвищеної плинності кадрів і погіршення загального психофізіологічного стану працівників. Таким чином, своєчасне виявлення та усунення цих факторів є ключовим елементом комплексної системи охорони праці, спрямованої не тільки на фізичну, але й на психологічну безпеку персоналу.

Висновки до розділу

Були розглянуті ключові аспекти охорони праці в контексті розробки інформаційного продукту та його застосування в сфері громадського харчування. Проведено аналіз нормативної бази, охарактеризовано обов'язки суб'єктів трудових відносин, виділено потенційні небезпеки як для розробників, так і для працівників кав'ярні.

Було застосовано метод дерева відмов для аналізу можливого розвитку аварійних ситуацій. З огляду на поточні реалії, охорона праці залишається найважливішим елементом сталого функціонування як інформаційних, так і фізичних робочих процесів, і вимагає системного підходу на всіх етапах проєктування та експлуатації продукту.

ВИСНОВКИ

В ході роботи було вирішено завдання розробки Telegram-бота, призначеного для автоматизації процесу прийому замовлень в кав'ярні. Актуальність даної теми обумовлена стрімким зростанням використання месенджерів в повсякденному житті, а також необхідністю підвищення швидкості обслуговування та оптимізації взаємодії між закладом і його клієнтами.

Було проведено аналіз предметної області, в рамках якого розглянуто особливості роботи закладів громадського харчування та виявлено основні проблеми, з якими стикаються співробітники і відвідувачі кав'ярні. Також були розглянуті існуючі рішення – аналогічні боти – що дозволило виділити їх ключові переваги і недоліки та сформулювати вимоги до власної розробки.

Також, були визначені логічна структура системи, її інформаційне та математичне забезпечення. Особлива увага була приділена розробці структури бота, зберіганню інформації про замовлення, розрахунку бонусної системи лояльності, а також формалізації станів користувача при взаємодії з ботом. Це дозволило забезпечити послідовну і коректну обробку запитів користувачів на всіх етапах оформлення замовлення.

Послідовно були розглянуті створення логіки взаємодії з користувачем, формування інтерфейсу, реалізації функціоналу від реєстрації і до надсилання оформленого замовлення до бариста. Додатково реалізована можливість відправки даних про замовлення в Telegram-групу бариста, що забезпечує миттєве отримання інформації та оперативне виконання замовлення.

Розроблений бот може бути легко адаптований під реальні умови конкретного закладу. Таким чином, поставлена мета була досягнута, а результати роботи підтверджують практичну значимість запропонованого рішення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Eats bot: Telegram-бот для замовлення їжі. ideil. URL: https://ideil.com/project/eats-bot

2. Roll club - чат-бот для замовлення їжі через Телеграм. Artjoker. URL: https://artjoker.ua/portfolio/chat-bot/roll-club/

3. Aiogram 3.19.0 documentation. aiogram. URL: https://docs.aiogram.dev/en/v3.19.0/

4. Aiogram. PyPI. URL: https://pypi.org/project/aiogram/

5. Python-telegram-bot. PyPI. URL: https://pypi.org/project/python-telegram-bot/

6. Python-telegram-bot. python-telegram-bot. URL: https://docs.python-telegram-bot.org/en/stable/

7. datetime – Базові типи дати і часу. Python documentation. URL: https://docs.python.org/3/library/datetime.html

8. JetBrains. PyCharm: The only Python IDE you need. JetBrains. URL: https://www.jetbrains.com/pycharm/

9. asyncio – Asynchronous I/O. Python documentation. URL: https://docs.python.org/3/library/asyncio.html

10. Інструкція з охорони праці для баристи. Бухгалтерські послуги. Аутсорсинг, аутстафінг | Компанія Вікторія. URL: https://www.victorija.ua/blanki-ta-formi-dokumentiv/prymirna-instruktsiya-z-ohorony-pratsi-dlya-barysty.html

11. Кодекс законів про працю України. Офіційний вебпортал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/322-08#Text

12. Про охорону праці. Головна - Законодавство України 2019 рік. URL: https://kodeksy.com.ua/pro_ohoronu_pratsi283_new.htm

13. Охорона праці в закладах громадського харчування - Охорона праці і пожежна безпека. Охорона праці і пожежна безпека. URL: https://oppb.com.ua/news/ohorona-praci-v-zakladah-gromadskogo-harchuvannya

14. Про затвердження Державних санітарних норм і правил "Гігієнічні вимоги до друкованої продукції для дітей". Офіційний вебпортал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/z0077-07#Text

15. Про охорону праці. Офіційний вебпортал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text

16. Мартіросян Р., Братерська Н. «Python-telegram-bot vs. Aiogram: яка бібліотека краща для написання телеграм ботів». Стан, досягнення і перспективи інформаційних систем і технологій / Матеріали XXV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. Одеса, 17-18 квітня 2025 р. - Одеса, Видавництво ОНТУ, 2025 р. – 287 с. – 132-133

ДОДАТКИ

Додаток А

Лістинг коду

```
import asyncio
from aiogram import Bot, Dispatcher
from aiogram.fsm.state import State, StatesGroup
from aiogram.fsm.context import FSMContext
from aiogram.filters import Command
from aiogram.types import Message, ReplyKeyboardMarkup, ReplyKeyboardRemove,
KeyboardButton
from datetime import datetime, timedelta

API_TOKEN = '7756290986:AAG50qkXN2awh0SpokpaeWtxG2wCH7km8_M'

bot = Bot(token=API_TOKEN)
dp = Dispatcher()

COFFEE_ORDERS_GROUP = -1002832214057
COFFEE_REVIEWS_GROUP = -1002515610312


COFFEE_PRICES = {
    "Капучіно": {"S": 45, "M": 55, "L": 65},
    "Американо": {"S": 35, "M": 45, "L": 55},
    "Латте": {"S": 50, "M": 60, "L": 70},
    "Макіато": {"S": 48, "M": 58, "L": 68},
    "Мокачино": {"S": 52, "M": 62, "L": 72}
}

LEMONADES_PRICES = {
    "Мохіто": {"S": 60, "M": 75, "L": 80},
    "Лимонад Цитрус": {"S": 60, "M": 75, "L": 80},
    "Оранж": {"S": 50, "M": 60, "L": 70},
    "Лимонад маракуя": {"S": 70, "M": 80, "L": 90}
}

SANDWICHES_PRICES = {
    "Сендвіч з куркою": {"S": 39, "M": 55, "L": 70},
    "Сендвіч з шинкою": {"S": 45, "M": 69, "L": 90},
    "Сендвіч з салямі": {"S": 30, "M": 50, "L": 70},
    "Сендвіч Веганський": {"S": 25, "M": 40, "L": 50}
}

BAKERY_PRICES = {
    "Круасан": {"S": 10, "M": 15, "L": 20},
    "Штрудель": {"S": 10, "M": 15, "L": 25},
    "Пиріг": {"S": 40, "M": 50, "L": 60},
    "Чизкейк": {"S": 50, "M": 70, "L": 100}
}


class RegisterUser(StatesGroup):
    waiting_for_name = State()
    waiting_for_phone = State()


class Order(StatesGroup):
    choosing_type = State()
    choosing_name = State()
    choosing_size = State()
```

```
    choosing_price = State()
    choosing_comment = State()
    choosing_review = State()


checkout = {}
bonuses = {}
barista_message = {}


def main_menu_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text=' Меню'), KeyboardButton(text=' Оформити
замовлення')],
            [KeyboardButton(text=' Особистий кабінет'),
KeyboardButton(text=' ? Допомога')]
        ],
        resize_keyboard=True
    )
    return keyboard


def products_menu_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text=' Кава'), KeyboardButton(text='
Лимонади')],
            [KeyboardButton(text=' Сендвічі'), KeyboardButton(text='
Випічка')],
            [KeyboardButton(text=' До головного меню')]
        ],
        resize_keyboard=True
    )
    return keyboard


def coffee_menu_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='Капучіно')],
            [KeyboardButton(text='Американо ')],
            [KeyboardButton(text='Латте')],
            [KeyboardButton(text='Макіато')],
            [KeyboardButton(text='Мокачино')]
        ],
        resize_keyboard=True
    )
    return keyboard


def lemonade_menu_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='Мохіто')],
            [KeyboardButton(text='Лимонад Цитрус')],
            [KeyboardButton(text='Оранж')],
            [KeyboardButton(text='Лимонад маракуя')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

```
def sandwich_menu_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='Сендвіч з куркою')],
            [KeyboardButton(text='Сендвіч з шинкою')],
            [KeyboardButton(text='Сендвіч з салямі')],
            [KeyboardButton(text='Сендвіч Веганський')]
        ],
        resize_keyboard=True
    )
    return keyboard


def bakery_menu_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='Круасан')],
            [KeyboardButton(text='Штрудель')],
            [KeyboardButton(text='Пиріг')],
            [KeyboardButton(text='Чизкейк')]
        ],
        resize_keyboard=True
    )
    return keyboard


def size_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='S'), KeyboardButton(text='M'),
KeyboardButton(text='L')],
            [KeyboardButton(text='❌ Скасувати')]
        ],
        resize_keyboard=True
    )
    return keyboard


def checkout_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='💳 Сплатити'), KeyboardButton(text='❌
Скасувати замовлення'),
             KeyboardButton(text='✏️ Редагувати')],
            [KeyboardButton(text='◀️ До головного меню')]
        ],
        resize_keyboard=True
    )
    return keyboard


def payment_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text='💳 Картка'), KeyboardButton(text='💵
Готівка'), KeyboardButton(text='🎁 Бонусні бали')],
            [KeyboardButton(text='◀️ До головного меню')]
        ],
        resize_keyboard=True
    )
    return keyboard
```

```
def choose_time_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text=' Найближчим часом')],
            [KeyboardButton(text=' На конкретний час')]
        ],
        resize_keyboard=True
    )
    return keyboard


def generate_time_buttons():
    now = datetime.now()

    total_minutes = now.hour * 60 + now.minute

    next_slot = ((total_minutes // 10) + 2) * 10

    times = []
    for i in range(3):
        future_minutes = next_slot + i * 10
        future_time = (datetime.combine(now.date(), datetime.min.time()) +
                       timedelta(minutes=future_minutes))
        times.append(future_time.strftime("%H:%M"))

    return times


def account_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text=' Баланс балів'), KeyboardButton(text='
Змінити персональні дані'),
             KeyboardButton(text=' Залишити відгук')],
            [KeyboardButton(text=' До головного меню')]
        ],
        resize_keyboard=True
    )
    return keyboard


def comment_keyboard():
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text=' Залишити коментар'),
KeyboardButton(text=' Не залишати коментар')]
        ],
        resize_keyboard=True
    )
    return keyboard


@dp.message(Command('start'))
async def start(message: Message, state: FSMContext):
    bonuses[message.from_user.id] = [{'actual bonuses': 0, 'potential
bonuses': 0, 'ordered account': False}]
    await message.answer(
        'Вітаємо у Coffee Telegram Bot! \n\n'
        'Для початку, будь ласка, пройдіть реєстрацію.'
    )
```

```
    await message.answer('Введіть ваше ім\'я 👤:')
    await state.set_state(RegisterUser.waiting_for_name)


@dp.message(RegisterUser.waiting_for_name)
async def get_name(message: Message, state: FSMContext):
    await state.update_data(name=message.text.strip())
    await message.answer('Тепер введіть ваш номер телефону у форматі
+380XXXXXXXXX 📞:')
    await state.set_state(RegisterUser.waiting_for_phone)


@dp.message(RegisterUser.waiting_for_phone)
async def get_phone(message: Message, state: FSMContext):
    phone = message.text.strip()
    if not phone.startswith('+380') or not phone[1:].isdigit() or len(phone)
!= 13:
        await message.answer('❗ Будь ласка, введіть коректний номер телефону
у форматі +380XXXXXXXXX ❗')
        return

    await state.update_data(phone=phone)
    data = await state.get_data()

    name = data['name']
    phone = data['phone']

    barista_message[message.from_user.id] = [{'name': name, 'phone': phone}]
    await message.answer(f'{name}, дякуємо, що обрали нас!🎉\n\n'
                         f'Ваш акаунт зареєстровано під номером: {phone}',
reply_markup=main_menu_keyboard())

    await state.clear()


@dp.message(lambda message: message.text == '📋 Меню')
async def show_products_menu(message: Message):
    await message.answer('Оберіть категорію:',
reply_markup=products_menu_keyboard())


@dp.message(lambda message: message.text == '☕ Кава')
async def show_coffee_menu(message: Message, state: FSMContext):
    await state.update_data(product_type='Кава')
    await state.set_state(Order.choosing_name)
    prices_text = ''
    for coffee, sizes in COFFEE_PRICES.items():
        prices_text += coffee
        for size, price in sizes.items():
            prices_text += f' // {size} - {price}грн'
        prices_text += '\n'

    await message.answer('Оберіть каву, яку бажаєте замовити:\n\n'
                         f'{prices_text}',
reply_markup=coffee_menu_keyboard())


@dp.message(lambda message: message.text == '🥤 Лимонади')
async def show_lemonade_menu(message: Message, state: FSMContext):
    await state.update_data(product_type='Лимонад')
    await state.set_state(Order.choosing_name)
    prices_text = ''
```

```
    for lemonade, sizes in LEMONADES_PRICES.items():
        prices_text += lemonade
        for size, price in sizes.items():
            prices_text += f' // {size} - {price}грн'
        prices_text += '\n'

    await message.answer('Оберіть лимонад, який бажаєте замовити:\n\n'
                         f'{prices_text}',
reply_markup=lemonade_menu_keyboard())


@dp.message(lambda message: message.text == '🥪 Сендвічі')
async def show_sandwich_menu(message: Message, state: FSMContext):
    await state.update_data(product_type='Сендвіч')
    await state.set_state(Order.choosing_name)
    prices_text = ''
    for sandwich, sizes in SANDWICHES_PRICES.items():
        prices_text += sandwich
        for size, price in sizes.items():
            prices_text += f' // {size} - {price}грн'
        prices_text += '\n'

    await message.answer('Оберіть сендвіч, який бажаєте замовити:\n\n'
                         f'{prices_text}',
reply_markup=sandwich_menu_keyboard())


@dp.message(lambda message: message.text == '🥨 Випічка')
async def show_bakery_menu(message: Message, state: FSMContext):
    await state.update_data(product_type='Випічка')
    await state.set_state(Order.choosing_name)
    prices_text = ''
    for bakery, sizes in BAKERY_PRICES.items():
        prices_text += bakery
        for size, price in sizes.items():
            prices_text += f' // {size} - {price}грн'
        prices_text += '\n'

    await message.answer('Оберіть випічку, яку бажаєте замовити:\n\n'
                         f'{prices_text}',
reply_markup=bakery_menu_keyboard())


@dp.message(Order.choosing_name)
async def choosing_product_name(message: Message, state: FSMContext):
    await state.update_data(product_name=message.text)
    await message.answer(f'Якого розміру {message.text} ви бажаєте:',
reply_markup=size_keyboard())
    await state.set_state(Order.choosing_size)


@dp.message(Order.choosing_size)
async def choosing_product_size(message: Message, state: FSMContext):
    if message.text == '❌ Скасувати':
        await state.clear()
        await message.answer('Вибір скасовано.',
reply_markup=main_menu_keyboard())
        return
    if message.text not in ['S', 'M', 'L']:
        await message.answer("Будь ласка, оберіть розмір із запропонованих
варіантів.")
        return
    if message.text in ['S', 'M', 'L']:
```

```
        await state.update_data(product_size=message.text)

    data = await state.get_data()

    product_type = data.get('product_type')
    product = data.get('product_name')
    product_size = data.get('product_size')
    price = 0

    if product_type == 'Кава':
        price = COFFEE_PRICES.get(product).get(product_size)
    elif product_type == 'Лимонад':
        price = LEMONADES_PRICES.get(product).get(product_size)
    elif product_type == 'Сендвіч':
        price = SANDWICHES_PRICES.get(product).get(product_size)
    elif product_type == 'Випічка':
        price = BAKERY_PRICES.get(product).get(product_size)

    user_id = message.from_user.id

    if user_id not in checkout:
        checkout[user_id] = []

    checkout[user_id].append({'product': product, 'size': product_size,
'price': price})

    await message.answer('☑ Додано до кошика. Щоб оформити замовлення,
перейдіть в розділ 🛒 Оформити замовлення.',
                         reply_markup=main_menu_keyboard())

    await state.clear()


@dp.message(lambda message: message.text == '🛒 Оформити замовлення')
async def start_checkout(message: Message):
    total_price = 0
    if not checkout or not checkout[message.from_user.id]:
        await message.answer('🛒 Ваш кошик порожній. Додайте товари перед
оформленням замовлення.',
                             reply_markup=main_menu_keyboard())
        return

    checkout_text = ''
    for product in checkout[message.from_user.id]:
        total_price += product['price']
        for product_info in product.values():
            if product_info != '':
                checkout_text += f'{product_info} '
        checkout_text += '\n'

    checkout_text += f'\n💰 До сплати: {total_price} грн'

    if int(total_price * 0.05) < 1:
        bonus = 1
    else:
        bonus = int(total_price * 0.05)

    bonuses[message.from_user.id][0]['potential bonuses'] = bonus

    barista_message[message.from_user.id][0]['order'] = checkout_text

    await message.answer(f'Ваш кошик:\n\n{checkout_text}',
reply_markup=checkout_keyboard())
```

```
@dp.message(lambda message: message.text == '💳 Сплатити')
async def payment(message: Message):
    await message.answer('Оберіть спосіб оплати:',
reply_markup=payment_keyboard())


@dp.message(lambda message: message.text == '❌ Скасувати замовлення')
async def cancel_checkout(message: Message):
    del checkout[message.from_user.id]
    await message.answer('Замовлення скасовано! ❌ Ваш кошик очищений 🛒',
reply_markup=main_menu_keyboard())


@dp.message(lambda message: message.text == '🖊 Редагувати')
async def edit_checkout(message: Message):
    if not checkout[message.from_user.id]:
        await message.answer(text='🛒 Ваш кошик порожній. Додайте товари
перед оформленням замовлення.',
                             reply_markup=main_menu_keyboard())
    buttons = []
    products = checkout.get(message.from_user.id)
    for i in range(len(products)):
        buttons.append([KeyboardButton(text=f'Видалити позицію # {str(i +
1)}')])

    buttons.append([KeyboardButton(text='◀️ До головного меню')])

    keyboard = ReplyKeyboardMarkup(
        keyboard=buttons,
        resize_keyboard=True
    )

    await message.answer('Оберіть позицію, яку бажаєте видалити:',
reply_markup=keyboard)


@dp.message(lambda message: 'Видалити позицію #' in message.text)
async def remove_from_checkout(message: Message):
    removed_product =
checkout[message.from_user.id].pop(int(message.text.split(' ')[-1]) - 1)
    await message.answer(f'{removed_product["product"]}
{removed_product["size"]} видалено з кошику ☑️',
                         reply_markup=main_menu_keyboard())


@dp.message(lambda message: message.text == '💳 Картка' or message.text ==
'💵 Готівка' or message.text == '🎁 Бонусні бали')
async def successful_payment(message: Message):
    if message.text == '💳 Картка':
        barista_message[message.from_user.id][0]['payment'] = message.text
        await message.answer(
            f'Ваше замовлення сплачено! Вам нараховано
{bonuses[message.from_user.id][0]["potential bonuses"]}'
            f' бала/-ів 🎁 Оберіть, будь ласка, час:',
reply_markup=choose_time_keyboard())
        bonuses[message.from_user.id][0]["ordered account"] = True
    elif message.text == '💵 Готівка':
        barista_message[message.from_user.id][0]['payment'] = message.text
        await message.answer(f'Вам нараховано
```

```
{bonuses[message.from_user.id][0]["potential bonuses"]} бала/-ів 🎁 '
                                'Оберіть, будь ласка, час:',
reply_markup=choose_time_keyboard())
        bonuses[message.from_user.id][0]["ordered account"] = True
    elif message.text == '🎁 Бонусні бали':
        total_price = 0
        for product in checkout[message.from_user.id]:
            total_price += product['price']
        if bonuses[message.from_user.id][0]['actual bonuses'] * 0.5 <
total_price:
            await message.answer(f'На жаль, вам не вистачає '
                                 f'{total_price -
bonuses[message.from_user.id][0]["actual bonuses"] * 0.5} грн '
                                 f'({(total_price -
bonuses[message.from_user.id][0]["actual bonuses"] * 0.5) * 2} бала/-
ів)😔\n\n'
                                 'Оберіть, будь ласка, інший спосіб оплати:',
reply_markup=payment_keyboard())
            return
        elif bonuses[message.from_user.id][0]['actual bonuses'] * 0.5 >=
total_price:
            barista_message[message.from_user.id][0]['payment'] =
message.text
            bonuses[message.from_user.id][0]["actual bonuses"] -= total_price
* 2
            await message.answer(f'Вітаємо, ви успішно оплатили замовлення
бонусними балами! '
                                 f'Вам нараховано
{bonuses[message.from_user.id][0]["potential bonuses"]} бала/-ів 🎁\n\n'
                                 f'На вашому рахунку:
{bonuses[message.from_user.id][0]["actual bonuses"] +
bonuses[message.from_user.id][0]["potential bonuses"]} бала/-ів 🤩',
                                 reply_markup=choose_time_keyboard())

    bonuses[message.from_user.id][0]['actual bonuses'] +=
bonuses[message.from_user.id][0]['potential bonuses']
    bonuses[message.from_user.id][0]['potential bonuses'] = 0

    del checkout[message.from_user.id]


@dp.message(lambda message: message.text == '⚡ Найближчим часом' or
message.text == '⏰ На конкретний час')
async def choosing_time(message: Message, state: FSMContext):
    if message.text == '⚡ Найближчим часом':
        barista_message[message.from_user.id][0]['time'] = 'На найближчий
час'
        await state.update_data(time='На найближчий час')
        await message.answer(f'Чи бажаєте залишити коментар для бариста? 💬',
reply_markup=comment_keyboard())
    elif message.text == '⏰ На конкретний час':
        times = generate_time_buttons()
        await message.answer('Оберіть, будь ласка, час:',
reply_markup=ReplyKeyboardMarkup(
            keyboard=[
                [KeyboardButton(text=f'{times[0]}'),
KeyboardButton(text=f'{times[1]}'),
                 KeyboardButton(text=f'{times[2]}')]
            ],
            resize_keyboard=True
        ))
```

```
@dp.message(lambda message: message.text == '💬 Залишити коментар' or
message.text == '🤐 Не залишати коментар')
async def comment(message: Message, state: FSMContext):
    if message.text == '💬 Залишити коментар':
        await message.answer(f'Ваш коментар для бариста 💬:',
reply_markup=ReplyKeyboardRemove())
        await state.set_state(Order.choosing_comment)
    elif message.text == '🤐 Не залишати коментар':
        await state.update_data(comment='')
        barista_message[message.from_user.id][0]['comment'] = ''
        await message.answer(f'Дякуємо за замовлення 😊 До зустрічі! 👋',
reply_markup=main_menu_keyboard())


@dp.message(lambda message: message.text in generate_time_buttons())
async def check(message: Message, state: FSMContext):
    barista_message[message.from_user.id][0]['time'] = f'На {message.text}'
    await state.update_data(time=message.text)
    await message.answer(f'Чи бажаєте залишити коментар для бариста? 💬',
reply_markup=comment_keyboard())


@dp.message(Order.choosing_comment)
async def give_comment(message: Message, state: FSMContext):
    await state.update_data(comment=message.text)
    barista_message[message.from_user.id][0]['comment'] = message.text
    if message.text:
        await message.answer('Ваш коментар передан бариста ☑')
    await message.answer(f'Дякуємо за замовлення 😊 До зустрічі! 👋',
reply_markup=main_menu_keyboard())

    await bot.send_message(chat_id=COFFEE_ORDERS_GROUP,
text=f'{datetime.now().strftime("%d/%m/%Y, %H:%M:%S")}\n\n'
                                                    f'Ім\'я:
{barista_message[message.from_user.id][0]["name"]}\n'
                                                    f'Номер
телефону: {barista_message[message.from_user.id][0]["phone"]}\n\n'

f'Замовлення:\n{barista_message[message.from_user.id][0]["order"]}\n'
                                                    f'Спосіб оплати:
{barista_message[message.from_user.id][0]["payment"]}\n'
                                                    f'Час:
{barista_message[message.from_user.id][0]["time"]}\n'
                                                    f'Коментар:
{barista_message[message.from_user.id][0]["comment"]}\n')

    await state.clear()


@dp.message(lambda message: message.text == '👤 Особистий кабінет')
async def show_account(message: Message):
    await message.answer('Вітаємо в особистому кабінеті! 👋\n\n',
reply_markup=account_keyboard())


@dp.message(lambda message: message.text == '🎁 Баланс балів')
async def show_account(message: Message):
    await message.answer(f'На вашому рахунку
{bonuses[message.from_user.id][0]["actual bonuses"]} бала/-ів! 🎁\n\n',
                         reply_markup=account_keyboard())
```

```
@dp.message(lambda message: message.text == '📇 Змінити персональні дані')
async def change_account(message: Message, state: FSMContext):
    await message.answer('Введіть ваше ім\'я 👤:')
    await state.set_state(RegisterUser.waiting_for_name)


@dp.message(lambda message: message.text == '⭐ Залишити відгук')
async def leave_review(message: Message, state: FSMContext):
    if bonuses[message.from_user.id][0]["ordered account"]:
        await message.answer(f'Залиште свій відгук 🗣 (текстовий):',
reply_markup=ReplyKeyboardRemove())
        await state.set_state(Order.choosing_review)
    else:
        await message.answer('Воу, а ти в нас ще нічого не замовляв 🤔')


@dp.message(Order.choosing_review)
async def get_review(message: Message, state: FSMContext):
    await bot.send_message(chat_id=COFFEE_REVIEWS_GROUP,
text=f'{datetime.now().strftime("%d/%m/%Y, %H:%M:%S")}\n\n'

f'{message.text}')
    await message.answer('Дякуємо за ваш відгук 😊 Він допомагає нам ставати
кращє!', reply_markup=main_menu_keyboard())
    await state.clear()


@dp.message(lambda message: message.text == '❓ Допомога')
async def show_help(message: Message):
    await message.answer('Виникли питання? 🤔\n'
                         'Ось список найчастіших питань та відповіді на
них:\n\n'

                         '❓Що буде, якщо я не заберу своє замовлення три
рази поспіль?\n\n'
                         'Якщо ви не забираєте замовлення три рази поспіль
без попередження, ваш обліковий запис може '
                         'бути тимчасово заблокований в боті, щоб уникнути
можливих зловживань. Рекомендуємо завжди '
                         'повідомляти про неможливість забрати замовлення
заздалегідь.\n\n'

                         '❓Що буде, якщо я не заберу своє замовлення три
рази поспіль?\n\n'
                         'Так, ви з легкістю можете це зробити, вказавши всю
необхідну інформацію в коментарі на етапі '
                         'оформлення замовлення.\n\n'

                         '❓Які у вас години роботи?\n\n'
                         'Ми працюємо щодня з 8:00 до 22:00.\n\n'

                         '❓Чи можна оплатити замовлення балами + своїми
коштами?\n\n'
                         'На жаль, оплата бонусними балами доступна тільки в
тому випадку, якщо вона покриває повну'
                         'вартість замовлення.\n\n'

                         '❓Що робити, якщо у мене виникли проблеми з
оплатою?\n\n'
```

```
                            'Якщо оплата не пройшла або виникли проблеми з
оплатою, зверніться до бариста.\n\n'

                            'Не знайшов відповідь на своє питання? Звернись до
бариста!')


@dp.message(lambda message: message.text == '◀️ До головного меню')
async def back_to_main_menu(message: Message):
    await message.answer('Ви повернулись до головного меню:',
reply_markup=main_menu_keyboard())


# # Фрагмент коду для отримання чат ID
# @dp.message()
# async def get_chat_id(message: Message):
#     await bot.send_message(chat_id=message.chat.id, text=f'Чат ID цієї
групи: {message.chat.id}')

async def main():
    await dp.start_polling(bot)


if __name__ == '__main__':
    asyncio.run(main())
```

Додаток Б

Апробація результатів роботи

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Одеський національний технологічний університет
ННІ Комп`ютерної інженерії, автоматизації робототехніки та програмування ім. П.М.Платонова
Національний технічний університет України «Київський політехнічний інститут»
Університет Інформатики і прикладних знань м. Лодзь, Польща

XXV Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів

«СТАН, ДОСЯГНЕННЯ І ПЕРСПЕКТИВИ ІНФОРМАЦІЙНИХ СИСТЕМ І ТЕХНОЛОГІЙ»

Матеріали конференції

Одеса

17-18 квітня 2025 р.

УДК 004.43

PYTHON-TELEGRAM-BOT VS. AIOGRAM: ЯКА БІБЛІОТЕКА КРАЩА ДЛЯ НАПИСАННЯ ТЕЛЕГРАМ БОТІВ

МАРТІРОСЯН Р. К., БРАТЕРСЬКА Н. М.
(mrmartirosyan1@gmail.com, nataliia.braterska@kname.edu.ua)
Харківський національний університет міського господарства ім. О. М. Бекетова

Розробка Telegram-ботів на Python стала популярним завданням, для якого існує кілька ефективних бібліотек, серед яких найвідомішими є python-telegram-bot і aiogram. Кожна з бібліотек має свої переваги: python-telegram-bot вирізняється простотою та стабільністю, тоді як aiogram пропонує більш сучасний підхід та більшу гнучкість. Вибір між ними залежить від потреб конкретного проєкту. Загалом, хоча обидві бібліотеки є ефективними інструментами, саме aiogram завдяки своїм архітектурним особливостям може вважатися більш професійним рішенням.

Розробка Telegram-ботів мовою програмування Python – затребуване завдання, для якого існує низка спеціалізованих бібліотек. Серед них найбільш популярними є python-telegram-bot і aiogram. На перший погляд може здатися, що вибір між цими двома рішеннями – справа смаку або звички. Однак у реальності кожна з бібліотек має як переваги, так і обмеження, що робить вибір більш комплексним.

З метою виявлення найбільш універсального інструменту для створення Telegram-ботів, проведемо детальний огляд кожної бібліотеки, їхніх ключових особливостей, сфер застосування, а також порівняльний аналіз за основними критеріями.

Python-telegram-bot – це одна з найбільш ранніх і стабільних бібліотек для взаємодії з Telegram Bot API. Вона побудована навколо об'єктно-орієнтованого підходу і синхронної моделі виконання, що робить її зрозумілою для початківців-розробників. Бібліотека ідеально підходить для невеликих і середніх проєктів, де не потрібне високе навантаження або асинхронна обробка подій. Часто використовується в навчальних проєктах, MVP-прототипах, а також у ботах із простою логікою. [1]

Ключові особливості:

- повна обгортка над Telegram Bot API;
- підтримка webhook і polling;
- проста і зрозуміла архітектура;
- велика документація і велика спільнота. [2]

Aiogram – це сучасна та асинхронна бібліотека для розробки Telegram-ботів. Вона побудована на asyncio, що дає змогу ефективно обробляти безліч одночасних запитів, особливо під час великих навантажень. aiogram рекомендується для створення масштабованих і продуктивних ботів, у яких передбачається велика кількість користувачів, складна логіка та інтеграція із зовнішніми сервісами. [3]

Ключові особливості:

- повністю асинхронна архітектура;
- підтримка FSM (finite-state machine) для управління станами;