МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ТЕЛЕГРАМ-БОТА ДЛЯ ЦИФРОВОГО МАРКЕТИНГУ»**

Виконала: студентка 4 курсу, групи 6.1220-з
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
Т. М. Прошак
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, д.е.н., доцент Полуектова Н.Р.,
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії ЗНУ, к.ф.-м.н., доцент Лісняк А.О.,
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М.
(підпис)

" ____ " _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Прошак Тетяні Миколаївни
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка телеграм-бота для цифрового маркетингу

керівник роботи Полуектова Наталія Робертівна, д.е.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » 12 2023 року № 2181-с

2. Строк подання студентом роботи 15.05.2024 р

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Вивчити сутність та проблеми цифрового маркетингу, вивчити призначення чат-ботів та інструменти їх створення, розробити чат-бот для Telegram, який підвищує ефек-. тивність маркетингу на підприємстві.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.12.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	10.01.2024	
2.	Збір вихідних даних.	10.02.2024	
3.	Обробка методичних та теоретичних	1.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	10.04.2024	
5.	Розробка третього розділу.	1.05.2024	
6.	Оформлення та нормоконтроль	14.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	30.05.2024	

Студент _____ Т.М. Прошак
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.Р. Полуектова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка телеграм-бота для цифрового маркетингу»: 64 с., 6 рис., 1 табл., 5 джерел, 1 додаток.

ЕФЕКТИВНІСТЬ, МАРКЕТИНГ, РОЗРОБКА, ТЕЛЕГРАМ-БОТ, ЦИФРОВІ ІНСТРУМЕНТИ.

Об’єкт дослідження – цифрові інструменти підвищення ефективності маркетингу. Предмет дослідження: методи т інструменти створення телеграм-бота для підвищення ефективності маркетингу в організації.

Мета роботи: проектування та розробка телеграм-бота для організації виробництва і збуту продукції, та більш ефективного задоволення потреб споживачів..

Метод дослідження – опитування та спостереження (визначення потреб споживачів та проблем цифрового маркетингу); узагальнення (висновок, в чому проблема маркетингу, після вдалого спостереження); порівняння (для вибору інструментів).

Результатами кваліфікаційної роботи є визначення проблем сучасного маркетингу та знаходження цифрових інструментів для підвищення ефективності маркетингу, зокрема за рахунок створення та впровадження телеграм-бота.

Розроблений додаток може бути корисним споживачам та співробітникам підприємства, для якого він розроблявся, а, також інших підприємств ресторанного бізнесу.

SUMMARY

Bachelor's qualifying paper «Development of a Telegram bot for digital marketing»: 64 pages, 6 figures, 1 tables, 5 references, 1 supplement.

EFFICIENCY, MARKETING, DEVELOPMENT, TELEGRAM BOT, DIGITAL TOOLS.

The object of research is digital tools for improving marketing efficiency. The subject of research is methods and tools for creating a Telegram bot to enhance marketing effectiveness in an organization.

The aim of the work is to design and develop a Telegram bot for production and sales organizations to better meet consumer needs.

Research method: survey and observation (identifying consumer needs and digital marketing problems); generalization (conclusion on marketing issues after successful observation); comparison (for tool selection).

The results of the thesis include identifying problems in modern marketing and finding digital tools to improve marketing efficiency, particularly through the creation and implementation of a Telegram bot.

The developed application can be beneficial to consumers and employees of the enterprise for which it was developed, as well as other businesses in the restaurant industry.

ЗМІСТ

ВСТУП

У сучасному світі інформаційних технологій та цифрової економіки, маркетинг стає все більш стратегічною складовою для успішного функціонування підприємств. Однак, разом із зростанням можливостей цифрового маркетингу з'являються нові виклики та проблеми, які потребують ефективних рішень та інноваційних підходів.

Цифровий маркетинг в сучасному бізнес-середовищі вимагає постійного вдосконалення та адаптації до швидкозмінних технологічних та конкурентних умов. Одним з перспективних напрямків вдосконалення маркетингових стратегій є використання телеграм-ботів.

У цій кваліфікаційній роботі досліджується можливість розробки телеграм-бота для підвищення ефективності маркетингу в організації. Цей проект має на меті вивчення сутності та задач маркетингу на підприємстві, виявлення проблем сучасного маркетингу та аналіз цифрових інструментів підвищення ефективності маркетингу.

У наш час, коли комунікація з клієнтами стає все більш цифровою та змінюється споживча поведінка, розробка телеграм-бота може стати важливим інструментом для підтримки та покращення маркетингових стратегій компанії. У нашій роботі ми розглянемо та проаналізуємо можливості використання телеграм-ботів у маркетингових цілях та розробимо практичний приклад застосування такого бота для підвищення ефективності маркетингових зусиль організації.

До виконання кваліфікаційної роботи поставлено такі задачі: розглянути використання в бізнес-процесі веб-ресурсу, розглянути питання розробки телеграм-бота за допомогою мови програмування Python для організації виробництва і збуту продукції, та більш ефективного задоволення потреб споживачів.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ЦИФРОВОГО МАРКЕТИНГУ

Сучасне бізнес-середовище, насичене конкуренцією та стрімкими змінами, вимагає від підприємств ефективного використання інструментів маркетингу для забезпечення стабільного розвитку та здобуття конкурентних переваг [1]. Залучення цільової аудиторії та збільшення конверсії, перетворення відвідувачів сайту чи соціальних мереж на клієнтів або покупців, досягається за допомогою різних стратегій, таких як контент-маркетинг, пошукова оптимізація (SEO), соціальна медіа-реклама, електронна комерція та інші. Все це об’єднує цифровий маркетинг.

1.1 Поняття цифрового маркетингу

Вперше поняття «цифрового маркетингу» використовували як проекцію традиційного маркетингу, його інструментів та стратегій, в інтернеті. Однак, особливості цифрового світу та його використання для маркетингу сприяли розвитку каналів, форматів та мов, що призвели до інструментів та стратегій, які неможливі в офлайні.

Сьогодні «цифровий маркетинг» - це більше не підтип традиційного маркетингу, а нове явище, що поєднує індивідуалізацію та масову дистрибуцію для досягнення маркетингових цілей. Технологічна збіжність та збільшення кількості пристроїв привели до розширення способів, якими ми користуємося в маркетингу в інтернеті, та допомогли визначити нове поняття «цифрового маркетингу»: він спрямований на користувача, більш вимірюваний, всезагальний та інтерактивний[2].

Маркетинг, як складова стратегічного управління підприємством, зосереджений на задоволенні потреб споживачів. Основними завданнями маркетингу є визначення цільового сегменту аудиторії, розробка та реалізація

продуктової політики, ціноутворення, просування на ринку та управління комунікаціями з клієнтами[3].

1.2 Стратегії цифрового маркетингу

Розвиток стратегій цифрового маркетингу надає великий потенціал для брендів та організацій. За допомогою стратегії-брендингу, платформи та сервіси дають можливість побудувати імідж бренду в мережі та охопити нових користувачів, постійно оновлюючись. Можливості поширення інформації через посилання дають споживачам можливість звертатися до організації більш широко та індивідуалізовано – це називається «комплексність».

Розвиток стратегій обіцяє зручність та функціональність: веб-платформи 2.0 пропонують прості та зручні інструменти для всіх, поліпшують користувацький досвід та дозволяють їм активно діяти.

У відповідності до візуального мислення, цифровий маркетинг пропонує маркетологам різноманітні інструменти на основі зображень та відео. Це привабливий спосіб досягнення аудиторії, що може призвести до більшого залучення.

Також, в контексті, коли організації намагаються встановити довгострокові відносини зі своєю аудиторією, Інтернет відкриває можливість спілкування, а це сприяє створенню позитивного досвіду з брендом. Інтернет - це унікальна можливість з'єднати організації з їх аудиторією та користувачами між собою. Ця зв'язаність може покращити їх досвід та покращити відносини з продуктом, брендом чи організацією. Цифровий маркетинг створює зв’язки з спільнотою, тісно переплітаючи аудиторії, користувачів, бренди, створюючи всесвітню павутину. Це побуджує організації поліпшувати маркетинг, використовуючи нові технології та стратегії.

Слід зазначити, що максимізують виходи легка сегментація та індивідуалізація реклами в Інтернеті. Крім того, вільність від обмежень інших медіа дозволяє більш привабливу рекламу. Суть Інтернету як мережі взаємопов'язаних вузлів робить експоненційне розповсюдження будь-якого контенту можливим. Виходячи з моделі комунікації "з вуст в уста", вірусна комунікація стає більш актуальною завдяки з'єднаності, миттєвості та можливості обміну віртуальних платформ, що полегшують поширення контенту.

У будь-якому випадку, для отримання максимальної вигоди від усіх цих можливостей, організації повинні забезпечити, щоб їхня присутність в Інтернеті або їхня присутність на різних каналах 2.0 слідувала стратегії з конкретними цілями, відповідно до їхнього бренду чи організаційного іміджу. Бути в Інтернеті без належного планування може не лише означати втрату можливостей з ресурсів та потенціалу, але також може справді негативно позначитися на організації, оскільки аудиторія, їхні потреби та сприйняття щодо організації невідомі.

Оцінка результатів досягнених цілей – одна з перших функцій, цифрового маркетингу. Онлайн-платформи першими виступають у доступності опцій слідування та можливості оцінки результатів: це допомагає організаціям та компаніям проводити моніторинг ефективності тої чи іншої стратегії цифрового маркетингу. Вимірювання результативності допомагає визначати проблеми маркетингу та шукати методи їх вирішення[4].

1.3 Проблеми сучасного цифрового маркетингу

У контексті цифрового маркетингу стають актуальними проблеми, які обмежують його ефективність та вимагають впровадження нових підходів та методів їх рішення.

Серед ключових проблем сучасного маркетингу слід виділити насиченість ринку, скорочення терміну життя продуктів, зміна у споживчому підході, труднощі у вимірюванні ефективності кампаній та нестабільність економічної ситуації [5]. Деякі з найпоширеніших проблем сучасного маркетингу включають:

- розрив між споживачами та брендами: швидка зміна споживчих уподобань та підходів до покупок може створювати розрив між споживачами та брендами. Компанії повинні знайти способи взаємодії зі своєю аудиторією та реагувати на їхні потреби та очікування;
- зменшення уваги споживачів: велика кількість інформації та реклами, з якою зіштовхуються споживачі, може призводити до зменшення уваги до маркетингових повідомлень. Компанії повинні знаходити способи привернення уваги своєї аудиторії та відзначатися серед конкурентів;
- зміни в медіа-споживанні: швидке розширення цифрових технологій та соціальних медіа змінює спосіб, яким споживачі взаємодіють з медіа-контентом. це може ускладнювати завдання маркетологів у досягненні своєї аудиторії та взаємодії з нею;
- конкуренція на ринку: зростаюча конкуренція на ринку може ускладнювати завдання просування товарів або послуг. Компанії повинні розробляти стратегії маркетингу, які дозволять їм виділятися серед конкурентів і привертати увагу аудиторії;
- технологічні зміни: швидкі технологічні зміни можуть призводити до того, що певні маркетингові стратегії стають застарілими або неефективними. Компанії повинні бути готові адаптуватися до нових технологій та трендів, щоб залишатися конкурентоспроможними;
- недостатня довіра споживачів: в умовах зростаючої кількості онлайн шахрайства та порушень конфіденційності, споживачі стають більш обережними щодо взаємодії з маркетинговими повідомленнями та

рекламою. Збільшення довіри може виявитися складною задачею для компаній;

- недоліки в зборі та аналізі даних: велика кількість даних, що генерується в інтернеті, може бути важкою управляти та аналізувати. Брак адекватних інструментів для збору, обробки та аналізу даних може ускладнювати розробку ефективних маркетингових стратегій;
- проблеми з каналами комунікації: швидко змінюючийся ландшафт медіа та соціальних мереж може призводити до того, що певні канали комунікації втрачають свою ефективність або стають застарілими. Компанії повинні постійно оцінювати та адаптувати свої стратегії комунікації;
- проблеми з конверсією: помилки у веб-дизайні, недоліки у дослідженнях споживчого поведінки та інші фактори можуть призводити до низької конверсії на веб-сайтах та інших цифрових платформах. Вирішення цих проблем може вимагати ретельного тестування та оптимізації;
- недоліки у креативному підході: іноді компанії можуть мати важкості у створенні привабливого та ефективного контенту для своїх маркетингових кампаній. Недоліки у креативному підході можуть призводити до того, що повідомлення стають непривабливими або непривертаючими увагу аудиторії.

Тому, спеціалісти цифрового маркетингу «не стоять на місці», а щоденно розвиваються, приймаючи виклики; відповідаючи на потреби користувачів; знаходячи шляхи вирішення тих чи інших проблем.

1.4 Цифрові інструменти підвищення ефективності маркетингу

Враховуючи ці виклики, важливим стає використання цифрових інструментів. Автоматизація маркетингових процесів, використання

аналітики для прийняття управлінських рішень, впровадження стратегій контент-маркетингу та персоналізації пропозицій сприяють підвищенню ефективності маркетингових заходів.

Цифрові інструменти, правильно інтегровані в маркетингову стратегію, дозволяють підприємствам подолати сучасні виклики та досягти більшої конкурентоспроможності. Розвиток та впровадження таких інструментів може стати ключовим фактором для підвищення ефективності маркетингу в сучасних умовах.

Цифрові інструменти відкривають широкі можливості для підвищення ефективності маркетингу. Ось деякі з них:

- соціальні медіа: платформи соціальних медіа, такі як facebook, instagram, telegram, twitter і linkedin, надають компаніям можливість зв'язуватися зі своєю аудиторією, створювати спільноти, розміщувати рекламу та отримувати відгуки.
- електронна пошта: електронна розсилка залишається одним з найефективніших інструментів маркетингу. Вона дозволяє компаніям надсилати персоналізовані повідомлення своїм клієнтам та підписникам, що сприяє збільшенню конверсії.
- веб-аналітика: засоби веб-аналітики, дозволяють компаніям відстежувати та аналізувати поведінку користувачів на своєму веб-сайті. Це дозволяє вимірювати ефективність рекламних кампаній та вдосконалювати їх для досягнення кращих результатів.
- пошукова оптимізація (seo): seo є важливим інструментом для підвищення видимості веб-сайту в пошукових системах. Оптимізація контенту та технічних аспектів сайту допомагає залучати більше органічного трафіку та покращувати його позиції в пошукових результатів[6].
- контент-маркетинг: створення цікавого та корисного контенту є ключовим для привертання уваги аудиторії та збільшення відвідуваності веб-сайту. Блоги, відео, інфографіка та інші формати

контенту можуть допомогти компаніям залучити та утримати свою аудиторію.

– рекламні платформи: рекламні платформи, такі як google ads, facebook ads, linkedin ads та інші, дозволяють компаніям розміщувати таргетовану рекламу перед своєю аудиторією та вимірювати її ефективність.

Використання цих інструментів допомагає компаніям підвищити свою конкурентоспроможність та досягнути більшого успіху в цифровому середовищі[7].

2 РОЗРОБКА ТЕЛЕГРАМ-БОТУ ЯК ЗАСОБУ ЦИФРОВОГО МАРКЕТИНГУ

2.1 Телеграм-боти як вирішення проблем цифрового маркетингу

Телеграм – це популярний месенджер, який також може бути потужним інструментом для цифрового маркетингу.

2.1.1 Можливості телеграму, як інструменту цифрового маркетингу

Телеграм надає можливість безпосереднього контакту з аудиторією через особисті повідомлення, групи та канали. Це дозволяє підтримувати взаємодію з клієнтами, відповідати на їх запитання та надавати індивідуальну підтримку: мати прямий контакт з аудиторією.

Канали і групи у Телеграмі можуть бути використані для просування продуктів, послуг або бренду, стати маркетинговими. Вони дозволяють розміщувати анонси, новини, акції та інші матеріали для привертання уваги аудиторії.

Телеграм дозволяє надсилати масові розсилки повідомлень своїм підписникам через канали або групи. Це може бути використано для інформування клієнтів про новини, акції, знижки та інші події.

Телеграм надає інструменти для аналізу та відстеження статистики активності аудиторії в каналах та групах. Це дозволяє оцінити ефективність маркетингових кампаній та вносити відповідні корективи.

Телеграм доступний на різних платформах, включаючи веб, мобільні пристрої та настільні комп'ютери. Це робить його доступним для широкого кола користувачів і дозволяє досягати цільової аудиторії незалежно від їх пристрою. контексті цифрового маркетингу стають актуальними проблеми,

які обмежують його ефективність та вимагають впровадження нових підходів та методів їх рішення.

Слід зазначити, що одною з найефективніших та унікальною можливістю месенджера Телеграм є Телеграм-боти. Вони можуть бути використані для автоматизації різних завдань, таких як надсилання повідомлень, відповіді на запитання або обробка замовлень. Вони дозволяють зменшити ручну працю та оптимізувати процеси обслуговування клієнтів[8].

2.1.2 Види телеграм-ботів та їхня роль у маркетингових стратегіях

Телеграм-боти – це програмні агенти, які автоматизовано взаємодіють з користувачами у месенджері Телеграм. Вони можуть виконувати різноманітні функції залежно від їх призначення та програмування. За типовими видами та функціями телеграм-боти поділяють на: інформаційні; «клієнтську підтримку»; боти для збору даних та опитування, аналітики даних і так далі.

Інформаційні боти призначені для надання користувачам інформації про продукти, послуги або бренд. Вони можуть відповідати на питання, надавати корисні статті, новини або іншу інформацію, що допомагає користувачам краще розуміти ваш бренд.

Боти «клієнтської підтримки» можуть використовуватися для надання технічної або консультаційної підтримки користувачам. Вони можуть відповідати на запитання, допомагати вирішувати проблеми та надавати поради щодо використання продуктів або послуг.

Також телеграм-боти можуть збирати дані від користувачів, проводити опитування та збирати відгуки. Це допомагає компаніям зрозуміти потреби та побажання своєї аудиторії та адаптувати свою стратегію маркетингу відповідно до цих даних.

Крім того, боти можуть бути використані для запуску маркетингових кампаній, розсилки рекламних повідомлень або проведення промоакцій. Вони

можуть надсилати спеціальні пропозиції, знижки або розіграші для залучення уваги користувачів.

В столітті передових технологій, телеграм-боти можуть допомагати в автоматизації різних бізнес-процесів, таких як замовлення товарів, запис на прийом або реєстрація на події. Це полегшує взаємодію з користувачами та оптимізує робочі процеси.

Деякі боти можуть забезпечувати аналітичні звіти та відстежувати результати маркетингових кампаній. Це допомагає компаніям оцінювати ефективність своєї стратегії та вносити відповідні корективи, тим самим відповідаючи на проблеми, знаходячи їхнє рішення.

Телеграм-боти можуть стати потужним інструментом у вашій маркетинговій стратегії, допомагаючи взаємодіяти з аудиторією, просувати продукти та послуги, а також забезпечувати підтримку користувачів.

2.1.3 Аналіз сучасного стану телеграм-ботів

Аналізуючи сучасний стан використання телеграм-ботів у маркетингових кампаніях, можна оглянути їхню роль та вплив на результативність стратегій просування. Це допоможе дослідити потреби та очікування цільової аудиторії стосовно використання телеграм-ботів, а також визначити ключові функції та можливості, які ці боти можуть надавати для підвищення ефективності маркетингу.

Огляд використання телеграм-ботів у маркетингових кампаніях дає змогу розглянути широкий спектр можливостей, які ці інтерактивні інструменти надають для підвищення ефективності маркетингу. Телеграм-боти можуть бути використані для автоматизації багатьох аспектів маркетингових стратегій, починаючи від обробки замовлень і відповідей на запитання клієнтів до створення інтерактивних промо-кампаній та аналізу даних.

Багато компаній у різних галузях використовують телеграм-ботів для реалізації своїх маркетингових стратегій та поліпшення спілкування з клієнтами. Ось декілька прикладів компаній, які активно використовують телеграм-ботів для маркетингу:

KLM Royal Dutch Airlines: Авіакомпанія KLM запустива телеграм-бота, який надає клієнтам інформацію про статус рейсів, можливість замовлення квитків та інші послуги[9].

The Wall Street Journal: Видання The Wall Street Journal використовує телеграм-бота для надсилання коротких новинних оглядів, статей та оновлень своїм підписникам[10].

BBC News: Телеграм-бот BBC News надає користувачам доступ до останніх новин, аналітики та інформації про найважливіші події.

Це лише кілька прикладів компаній, які успішно використовують телеграм-ботів у своїх маркетингових стратегіях.

2.2 Реалізація телеграм-бота

2.2.1 Постановка задачі кваліфікаційної роботи

Темою даної кваліфікаційної роботи є «Розробка телеграм-бота для підвищення ефективності маркетингу в організації». Бот буде призначений для збільшення взаємодії з клієнтами. Бот буде використовуватись для збору замовлень та відповідей на запитання клієнтів.

Основною функцією буде можливість клієнтів здійснювати замовлення через бота; а користувач з доступом адміна може отримувати список замовлень; додавати/змінювати товари; обробляти інформацію від користувачів.

Область застосування даного боту: ресторанний бізнес.

2.2.2 Обґрунтування технології реалізації боту

Мова програмування даної кваліфікаційної роботи – Python. Це обумовлено наступними причинами. По-перше, Python відомий своєю простотою та зрозумілістю синтаксису, що дозволяє швидко розробляти та модифікувати код. Крім того, наявність великої кількості бібліотек та фреймворків у Python сприяє швидкому розвитку функціоналу та підтримці необхідних функцій та інтеграцій. Python також має велику спільноту розробників, що сприяє вирішенню проблем та забезпечує доступ до багатьох корисних ресурсів і матеріалів для вивчення. Вибір Python для цього проєкту дозволить нам швидко та ефективно реалізувати функціональність телеграм-бота для підвищення ефективності маркетингу в організації.

Python має широкі можливості для роботи з телеграм API (отримання повідомлень, відправлення повідомлень, керування чатами, робота з мультимедіа, реакції на події і т.д.).

Фреймворк (framework) – це структурована або скомпонована колекція коду, яка надає загальну архітектурну основу для розробки програмного забезпечення. Фреймворк, який використовуватиму для створення телеграм-боту – aiogram (фреймворк для розробки ботів для месенджера Telegram в середовищі мови програмування Python). Він надає розробникам зручний інтерфейс для взаємодії з Telegram API та реалізації різноманітних функцій бота. Aiogram спрощує розробку та підтримку Телеграм-ботів, надаючи потужні інструменти для взаємодії з Телеграм API та реалізації різноманітних функцій для мого бота[11].

Архітектура програмного забезпечення (Рисунок 2.1 – Схема архітектури) – загальна структура системи, до складу якої входить Frontend, Backend та Database. Фронтендом для мого проєкту є інтерфейс Телеграм, завдяки якому користувачі будуть взаємодіяти з ботом. Бот буде обробляти вхідні повідомлення від користувачів та надсилати відповіді, реалізовані за допомогою функціоналу, розробленого з використанням Python, та вчасності,

фреймворку aiogram. Бекендом мого проєкту є логіка програми, яка оброблятиме вхідні запити від користувачів; взаємодіятиме з базою даних та відправлятиме відповіді користувачам. Це Python-код, написаний з використанням aiogram. Базою даних мого проєкту є SQLite, що використовується для зберігання даних. Також з цієї бази даних надається інформація для функціонування мого бота.

Отже, у моєму проєкті можна виокремити три основні компоненти: фронтенд (інтерфейс Телеграму), бекенд (Python-код з aiogram) та базу даних (SQLite), які спільно працюють для забезпечення функціональності бота.

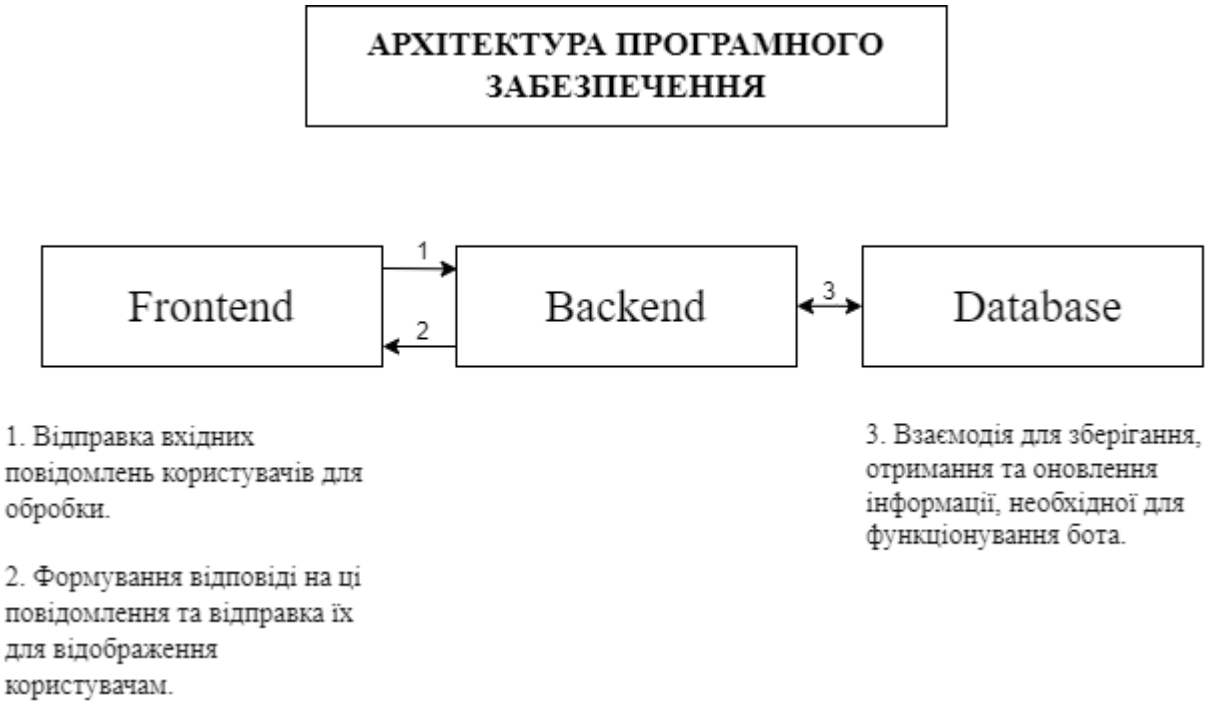

Рисунок 2.1 – Схема архітектури програмного забезпечення

2.2.3 Опис основних програмних модулів та функцій

Основні програмні модулі в aiogram, які використовуються для розробки Телеграм-ботів, включають такі: Диспетчер (dispatcher), бот (bot), типи (types), класи(class), проміжне програмне забезпечення(middleware), фільтри(filters), обробники (handlers) та інші.

Головним файлом мого Телеграм-Боту є файл «app.py», в якому реалізовані основна логіка та налаштування. Цей файл відповідає за запуск бота та основний цикл роботи. Для цього використовуються програмні модулі фреймворка aiogram.

Почнемо з модулей, які відповідають за взаємодію з Телеграм API та відправлення повідомлень користувачам. Налаштування модуля бот (bot) включають в себе парсер повідомлень (ParseMode), що є перерахуванням в бібліотеці aiogram. Це дозволяє вказати Телеграму, як повинен оброблятися текст повідомлень, які відправляються ботові або користувачам. Диспетчер (dispatcher) відповідає за обробку вхідних повідомлень та подій від Телеграм та маршрутизацію до відповідних обробників. Диспетчер дозволяє реєструвати обробники (handlers) для різних типів повідомлень, команд, кнопок та ін.

Слід зазначити, що в процесі створення боту використовувались HTML-теги для форматування тексту.

Модуль «asyncio» використовується для роботи з асинхронними функціями у Python. Також, при створенні боту використовувався модуль «os», що надає функції для взаємодії з операційною системою. В моєму проєкті цей модуль використовувався для отримання доступу до змінних середовища, зокрема токену бота, що зберігається в окремому файлі «.env». Допомагає завантажити змінні середовища з цього фалу бібліотека «dotenv». Я використовую цю бібліотеку для безпечного зберігання конфіденційних даних.

Модуль обробників (Handlers) містить класи та функції, які використовуються для обробки різних типів повідомлень та подій. Обробники визначають логіку, яка виконується при отриманні певного типу повідомлень. Саме в моєму проєкті я створила окрему папку handlers, у вміст якої увійшли наступні файли: admin_private.py(обробники для панелі керування адміна), menu_processing.py, user_group.py(список команд, які може отримати бот в групових чатах), user_private.py(налаштування обробників повідомлень

користувачів). Отже, handlers відповідають за обробку різних типів повідомлень та команд, що отримує бот.

Модуль проміжного ПЗ «middleware» використовується для відкриття файлу проєкту «middlewares.db.py», що відповідає за роботу з базою даних. У файлі «db.py» (Рисунок 2.2 – Програмний код class), після імпорту необхідних типів, класів та генераторів; визначається новий клас, який пізніше визначає основну логіку middleware, отримуючи обробники, події та дані; додаючи до них об’єкт сесії бази даних.

Сумуючи вищезазначене, можна прийти висновку, що middleware призначений для забезпечення доступу до сесії бази даних у кожному handlers Телеграм-бота. Це і дозволить здійснювати взаємодію з базою даних під час обробки повідомлень та подій.

```python
class DataBaseSession(BaseMiddleware):
    def __init__(self, session_pool: async_sessionmaker):
        self.session_pool = session_pool


    async def __call__(
        self,
        handler: Callable[[TelegramObject, Dict[str, Any]], Awaitable[Any]],
        event: TelegramObject,
        data: Dict[str, Any],
    ) -> Any:
        async with self.session_pool() as session:
            data['session'] = session
            return await handler(event, data)
```

Рисунок 2.2 – Програмний код class DataBaseSession у файлі db.py

Ще один модуль, що містить функції для взаємодії з «двигуном» бази даних – database.engine. Якщо узагальнити дії у файлі «engine.py», то можна описати їх так:

– підключення до бази даних за допомогою встановлення асинхронного з’єднання;

- створення асинхронного генератора сесій для взаємодії з базою даних;
- опис функцій для створення та видалення бази даних, а також для ініціалізації початкових даних;
- створення/видалення таблиць, заповнення даних.

Модуль engine забезпечує правильну роботу з даними.

Для представлення таблиць в базі даних, створюються класи. Опис моделей даних для мого проєкту містяться в файлі models.py. Кожен клас представляє таблицю у БД, а атрибути класу відображають стовпці відповідної таблиці.

Кожен клас моделі оголошується за допомогою класу, що успадковує DeclarativeBase (базовий клас, що надає функціональність SQLAlchemy для оголошення моделей даних). Також, для кожного створеного стовпця в таблиці оголошуються атрибути класу, які представляють стовпці бази даних. За допомогою об'єктів Mapped визначаються структура та типи даних, які зберігаються в таблицях бази даних. Крім того, вони дозволяють встановлювати різноманітні параметри для стовпців, такі як унікальність, обмеження на значення, зовнішні ключі тощо. Щоб визначити відношення між таблицями в моїй базі даних, я використовувала атрибути relationship. Так класи можуть взаємодіяти між собою; можуть бути відношення між категоріями та класами. ForeignKey – об’єкт, що використовується для визначення зовнішніх ключів. Цей об’єкт вказує на зв’язок між стовпцями у різних таблицях: дані надсилаються для зовнішнього використання і можуть бути задіяні в іншому класі. В моєму випадку, створюється сім класів для збереження даних про користувача, страви, категорії, корзини замовлень та налаштувань інтерфейсу бота. Отже, маючи на увазі вищезазначений опис файлу models.py, можна прийти до висновку, що цей файл допомагає визначити структуру бази даних; включаючи таблиці, стовпці та відношення між ними. Він є частиною модуля database і використовується для організації та опису структури даних.

Реалізація мого телеграм-бота потребує набір асинхронних функцій для виконання запитів до бази даних за допомогою SQLAlchemy (бібліотеки для роботи з реляційними (табличними) базами даних в Python). Ці всі функції я помістила у файл orm_query.py в директорії database. Нижче наведена таблиця основних функцій та їхнє призначення (– Основні функцій файлу orm_query.pyТаблиця .1).

Таблиця 2.1 – Основні функцій файлу orm_query.py

Операції з банерами (Banner)	orm_add_banner_description	додавання описів банерів до бази даних, якщо вони ще не існують (за допомогою сесії)
	orm_change_banner_image	оновлення зображення банера за вказаним ім’ям
	orm_get_banner	отримання даних конкретного банера за його ім’ям
	orm_get_info_pages	отримання всіх банерів з бд
Операції з категоріями (Category)	orm_get_categories	отримання всіх категорій з бази даних
	orm_create_categories	додавання категорій до бд, якщо вони відсутні
Операції з стравами (Dish)	orm_add_dish	додавання нової страви до бд
	orm_get_dishes	отримання всіх страв певної категорії
	orm_get_dish	отримання даних про конкретну страву
	orm_update_dish	оновлення даних страви
	orm_delete_dish	видалення страви з бд
	orm_get_all_orders	отримання всіх замовлень з інформацією про користувачів та страви
Операції з користувачами (User)	orm_add_user	додавання нового користувача до бази даних, якщо він відсутній
Операції з кошиком (Cart)	orm_add_to_cart	додавання страви до кошика користувача
	orm_get_user_carts	отримання всіх страв в кошику користувача
	orm_delete_from_cart	видалити страви з кошика
	orm_reduce_dish_in_cart	зменшення кількості страв в кошику користувача або

		видалення їх, якщо кількість стала = 0

Повертаючись до файлу app.py, слід зазначити декілька функцій, що там використовуються. Функції on_startup та om_shutdown виконуються під час запуску та завершенню роботи бота відповідною Основна функція – main. Вона викликається для запуску бота та включає в себе налаштування функцій запуску та завершення роботи, а також запуск основного циклу бота.

Отже, файл «app.py» відповідає за налаштування, ініціалізацію та запуск бота, а також за підключення та налаштування обробників для обробки вхідних повідомлень та команд. Повний код файлу можна подивитися в Додатках.

Для будь-якого проєкту діять якісь модулі фільтрів. Мій проєкт не є виключенням, тому маю файл chat_types.py, який містить два класи фільтрів для повідомлень у чаті. Перший фільтр ChatTypeFilter призначений для фільтрації повідомлень за типом чату. Для того, щоб конструктор класу приймав список типів чату, які дозволені, створюється список заборонених типів. Наступний фільтр необхідний для адміністрування. Не вся інформація має бути видна користувачу, тому для створення адмінської частини проєкту, призначається фільтр IsAdmin; який і перевіряє, чи є користувач, що надсилає повідомлення, адміністратором бота. Файл фільтрів надає функціональність для фільтрації повідомлень за типом чату та перевірки статусу адміністратора для виконання певних дій у боті.

В телеграм-боті використовуються клавіатури для взаємодії з користувачем. Клавіатури дозволяють користувачам швидко та зручно вибирати варіанти відповіді або взаємодіяти з функціями бота, не вводячи текст вручну. У моєму проєкті я використовую клавіатури для надання користувачам можливості вибору певної інформації, якою вони цікавляться; для виклику певних дій, зокрема замовленню продукції. В адмін-панелі є можливість додавати/видаляти товари, передивлятися асортимент та замовлення клієнтів. Все це робиться завдяки клавіатурам. В моїй директорії

keyboards є два файли: inline.py та reply.py. В файлі inline.py створюються функції для генерації клавіатур для взаємодії з користувачами телеграм-боту (рис. 2.3.). Ось перелік функцій, що використовувались у даному файлі:

- menuCallBack – клас підкласу CallBackData, що використовується для створення об’єктів з інформацією про вибір користувача у меню. Даний клас, як і всі класи, містить поля(атрибути).
- функція get_user_main_btns генерує клавіатуру з основними кнопками головного меню
- функція get_user_catalog_btns генерує клавіатуру з кнопками категорій товарів для користувача: Страви або Напої.
- get_dishes_btns – функція, що створює клавіатуру з кнопками для вибору страв, додавання до корзини та навігації між сторінками.
- функція get_user_cart допомагає в корзині користувача змінювати кількість товарів. Для цього було створено клас Paginator, який призначений для розділення списку елементів на сторінки та навігації між ними.

Всі ці функції допомагають створювати різні типи клавіатур, забезпечуючи зручний та зрозумілий інтерфейс телеграм-бота.

Рисунок 2.3 – Інлайнова клавіатура 1

Наступний файл – reply.py. Цей файл відрізняється від попереднього тим, що містить лише одну функцію get_keyboard, яка призначена для створення клавіатур із звичайними кнопками, які розташовуються під

текстовим полем введення в чаті. Ці кнопки з’являються в адмін-інтерфейсі (рис. 2.4). На відміну від цього файлу, inline.py містить функції та класи, які пов’язані з інлайн-клавіатурами, тобто, з клавіатурами, які відображаються під повідомленнями та дозволяють користувачу швидко відповісти, вибравши один з варіантів. Інлайн-кнопки не залишають поточної бесіди з ботом.

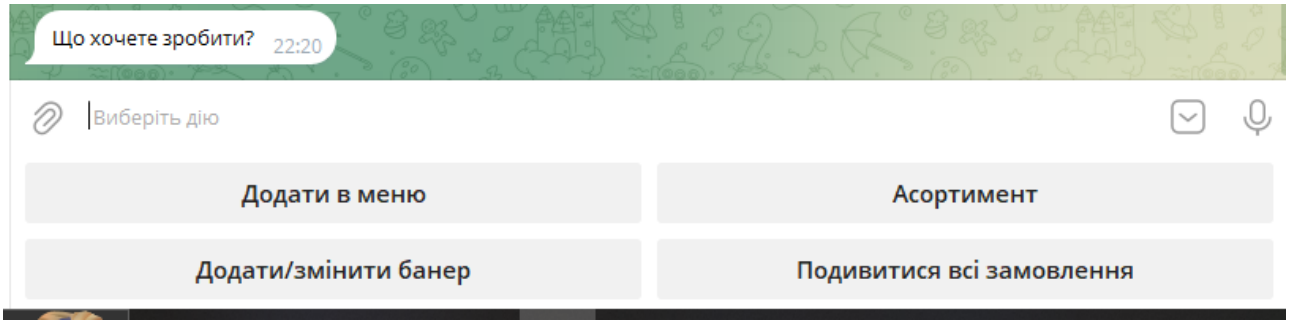

Рисунок 2.4 – Текстова клавіатура

Отже, основна відмінність між файлами у видах клавіатур, які вони створюють.

Хочу звернути увагу на клас Paginator, який призначений для розділення списку елементів на сторінки та навігації між ними. Основною метою пагінатора є уникання перенавантаження сторінки великою кількістю інформації. Основними характеристиками пагінатора є: розбиття на сторінки, навігаційні кнопки, зручність для користувача. Цей клас допомагає покращити користувацький досвід та зробити навігацію більш ефективною.

Як я і зазначала на початку розділу реалізації телеграм-бота, одним з основних модулей в моєму проєкті, як і в будь-якому, є модулі обробників, вони ж handlers. Адміністративні функції зазначені у файлі admin_private.py. Код, що записаний в цьому файлі імпортує необхідні залежності; визначає клавіатури для адміністративних функцій; реалізовує обробники команд, повідомлень, callback-запитів, які викликаються при взаємодії користувачів з ботом; визначає стани та реалізує машину станів (FSM) для додавання нової інформації адміністратором, реалізовує команди «скасувати» та «назад», які дозволяють адміністратору скасувати дії або повернутися на попередній крок.

Також в цьому файлі присутні handlers, які дозволяють обробляти зображення. В цілому, цей файл містить логіку, необхідну для адміністрування бота, додавання нових страв та керування замовленнями через взаємодію з адміністратором.

Наступний, не менш важливий файл-хендлер – user_private.py. Цей файл відповідає за початок дій (команда start); обробку повідомлень та взаємодію з користувачами в приватних чатах, що дозволяє їм переглядати меню та робити замовлення.

Функції для отримання контенту з різних меню користувача зберігаються в файлі menu_processing.py. Крім того, цей файл містить функції для роботи з пагінацією, щоб дозволити користувачам переглядати великі набори даних.

Файл user_group.py, що міститься в цій же директорії handlers, містить обробники для роботи з повідомленнями у групах. Саме завдяки цьому файлові перевіряється адмін-статус користувача та зберігається список адміністраторів в боті. Також цей файл відповідає за безпеку в групі: якщо будуть написати небажані заборонені слова/фрази – з’явиться попередження для користувача. Якщо користувач декілька разів порушує правила спільноти – він, за його user_id, додається в чорний список користувачів.

Отже, основною метою проєкту є: можливість надавати користувачам перегляд меню ресторану, додавання страв до кошика та здійснювати замовлення. Все це звершується за допомогою представлених вище функцій та модулів.

3 ВИКОРИСТАННЯ РОЗРОБЛЕНОГО ТЕЛЕГРАМ-БОТА

Для використання розробленого телеграм-бота нам необхідно встановити Desktop-версію Телеграму на наш пристрій. Це допоможе встановити та налаштувати бот з легкістю.

3.1 Встановлення та налаштування

Щоб створити бот в Телеграмі, необхідно, першим чином, звернутися до офіційного BotFather в Телеграм та створити нового бота за допомогою команди /newbot. Пізніше, треба надати назву нашому боту, username. Таким чином ми зможемо отримати TOKEN, який і розмістимо в файлі .env нашого проєкту, тим самим «підключивши» нашу програму до нашого новоствореного бота (Рисунок 3.1 – Створення нового бота).

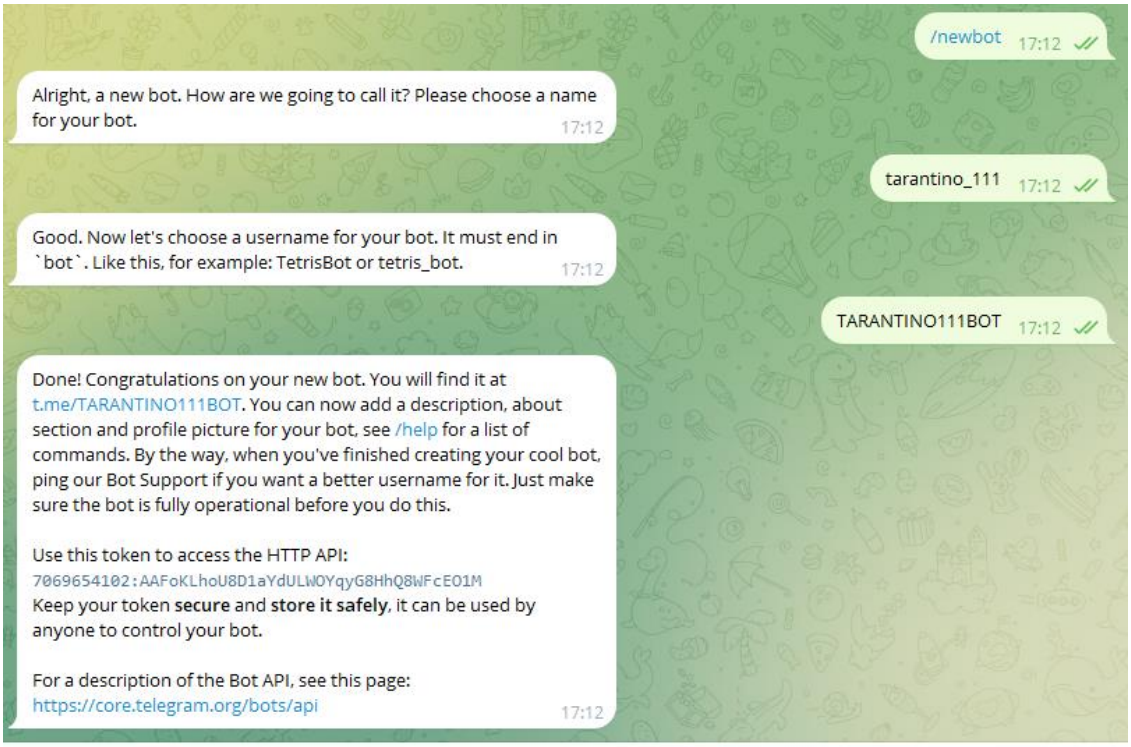

Рисунок 3.1 – Створення нового бота

Щоб запустити проєкт, необхідно відкрити код в обраному середовищі розробки (в моєму випадку це Visual Studio Code), та запустити мій телеграм-бот, виконавши головний скрипт, що записаний у файлі app.py.

Середовищем розробки (IDE) я вибрала саме Visual Studio Code тому, що це середовище підтримує різноманітні мови програмування, зокрема і Python, та має багато корисних функцій. VS Code має простий інтерфейс, може поліпшувати функціональність завдяки широкому вибору розширень, а також одразу інтегрується з Git.

Після вдалої перевірки запуску бота, можна заявити, що проєкт працює.

3.2 Керівництво користувача

Для початку роботи щ ботом, необхідно його знайти в Телеграмі. Для цього варто використовувати ім’я або логін. Розпочинаючи роботу, слід натиснути на кнопку «start» або відправити відповідну команду «/start». Після цього можна починати взаємодію з ботом (Рисунок 3.2 – Команда /start).

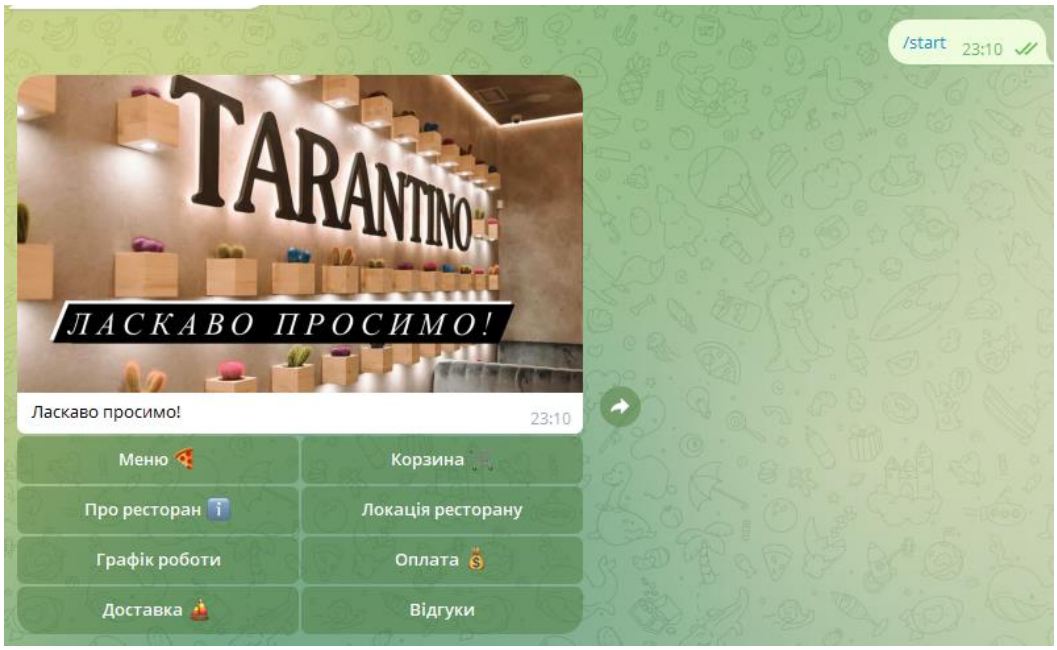

Рисунок 3.2 – Команда /start

Навігація в боті здійснюється за допомогою inline-кнопок. Щоб створити замовлення, слід виконувати інструкції, які буде надавати бот.

Щоб зайти в інтерфейс, як адміністратор, з правами змінювати ту чи іншу інформацію, необхідно авторизуватись в групі, в які бот буде адміністратором. Усі, хто в цій групі також має статус «адмін» або «суперадмін», матимуть доступ до адмін-боту, якщо буде надіслано команду /admin в групу. Після цього, адміністратор може надіслати цю ж команду боту, і тоді відкриється адмін-сторінка з можливістю додавати страви, видаляти їх, передивлятися замовлення, додавати або змінювати банери, передивитись увесь асортимент та інші.

ВИСНОВКИ

Отже, у ході дослідження теми кваліфікаційної роботи «Розробка телеграм-бота для підвищення ефективності маркетингу в організації», я досліджувала цифрові інструменти підвищення ефективності маркетингу, зокрема телеграм-боти.

Визначивши проблеми цифрового маркетингу, я намагалась знайти методи їх рішення. Результатами кваліфікаційної роботи, які я досягнула, є : визначення сутності та задачі сучасного маркетингу, а також знаходження цифрових інструментів для підвищення ефективності маркетингу та задовільнення потреб користувачів.

На підставі виконаної роботи можна зробити наступні висновки щодо даного проєкту розробки телеграм-боту: мова програмування, що використовувалась для розробки боту – Python. Потужна бібліотека, що застосовувалась в даному проєкті – aiogram. Вона працює з Телеграм API в асинхронному режимі у Python. Для роботи з базою даних використовувався інструмент для роботи з реляційними базами даних у Python, а саме SQLAlchemy. Це дозволяло створювати та взаємодіяти з об’єктами бази даних у стилі «об’єктно-реляційного відображення»(ORM), що робить роботу з базами даних більш зручною та ефективною.

З проєкту видно, що реалізований бот має різноманітні можливості. Це означає, що бот може виконувати різні завдання (такі як перегляд асортименту, додавання товарів, оформлення замовлень, робота з банерами та інше). Дивлячись на цей проєкт, можна побачити, що бот надає користувачам досить широкий функціонал. Отже, такі компоненти, як фронтенд (інтерфейс Телеграму), бекенд (Python-код з aiogram) та база даних (SQLite), спільно працюють для забезпечення функціональності бота.

Слід зазначити, що в коді використовується машина для управління станами користувача у боті FSM (Finite State Machine). Це дозволяє

реалізовувати складну логіку обробки повідомлень та взаємодії з користувачами.

Також, для розділення довгих списків товарів або повідомлень на сторінки, в проєкті реалізовано використання пагінатора, що полегшує навігацію користувача.

У цілому, проєкт добре організований та має потенціал для розвитку та розширення функціоналу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Казакова Л. О., Мойсюк Ю. В.. Конкурентне Середовище Міжнародного Бізнесу. URL: http://www.visnyk-econom.uzhnu.uz.ua/archive/49_2023ua/18.pdf (Дата звернення: 16.04.2023)

2. Писаренко Н. Діджитал Маркетинг: навч. посіб. Київ: КПІ ім.Сікорського, 2020. 100 с.

3. Carolina Machado. J.Paulo Davim. MBA Theory and Application of Business and Management Principles. Springer International Publishing Switzerland, 2016. 45 pages.

4. Саймон Кингснорт. Стратегия цифрового маркетинга: интегрированный подход к онлайн-маркетингу. Олимп-Бизнес, 2019. 416 сторінок.

5. Гноєвий В. Г., О. М. Корень. Сучасні тенденції цифрового маркетингу та їх вплив на формування маркетингової стратегії // Академічний огляд. 2021. №1 (54). С.49-55.

6. Netmark`s 2016 Guide to The 6 Fundamentals of Digital Marketing. 15 pages.

7. Jeganathan Gomathi Sankar. Digital Marketing and its challenges. Article, August 2017. 18 pages.

8. Мельничук А. Чат-бот. URL: https://sendpulse.ua/support/glossary/chatbot (дата звернення: 13.05.2024).

9. Чат-бот KLM розповість пасажирам, що їм взяти в поїздку | BYTSKO TRAVEL GROUP. Новини. URL: https://bytsko.com/chat-bot-klm-rozpov--st-pasagiram--shcho---m-vzyati-v-po--zdku/ (Дата звернення: 13.05.2024).

10. Quickly connect Telegram to The Wall Street Journal – IFTTT. Новини. URL: https://ifttt.com/connect/telegram/the_wall_street_journal (Дата звернення: 13.05.2024).

11. Aiogram documentation. URL: https://docs.aiogram.dev/uk-ua/dev-3.x/ (дата звернення: 11.04.2024).

ДОДАТОК А

Лістинг файлу «app.py»

```
import asyncio
import os

from aiogram import Bot, Dispatcher, types
from aiogram.enums import ParseMode

from dotenv import find_dotenv, load_dotenv

load_dotenv(find_dotenv())

from middlewares.db import DataBaseSession

from database.engine import create_db, drop_db, session_maker

from handlers.user_private import user_private_router
from handlers.user_group import user_group_router
from handlers.admin_private import admin_router

bot = Bot(token=os.getenv('TOKEN'), parse_mode=ParseMode.HTML)
bot.my_admins_list = []

dp = Dispatcher()

dp.include_router(user_private_router)
dp.include_router(user_group_router)
dp.include_router(admin_router)

async def on_startup(bot):

    #await drop_db()

    await create_db()

async def on_shutdown(bot):
    print('бот лег')

async def main():
    dp.startup.register(on_startup)
    dp.shutdown.register(on_shutdown)

    dp.update.middleware(DataBaseSession(session_pool=session_maker))

    await bot.delete_webhook(drop_pending_updates=True)
    await bot.delete_my_commands(scope=types.BotCommandScopeAllPrivateChats())
```

```
    # await bot.set_my_commands(commands=private,
scope=types.BotCommandScopeAllPrivateChats())
    await dp.start_polling(bot, allowed_updates=dp.resolve_used_update_types())

asyncio.run(main())
```

ДОДАТОК Б

Лістинг файлів директорії common

restricted_words.py

```
restricted_words = {'росія', 'ідіот', 'сука'}
```

texts_for_db.py

```
from aiogram.utils.formatting import Bold, as_list, as_marked_section


categories = ['Страви', 'Напої']

description_for_info_pages = {
    "main": "Ласкаво просимо!",

    "about": "Тарантіно - ресторан з затишним інтер'єром та з широким асортиментом страв, з високим рівнем обслуговування",

    "location":"https://maps.app.goo.gl/gtshbkFfbLCmeNM16",

    "work_time": as_marked_section(
        Bold("Працюємо:"),
        "Понеділок - Неділя.🗓",
        "9.00 - 23.00.🕓",
        "Без перерви, без вихідних.🔄",
        marker="- "
    ).as_html(),

    "payment": as_marked_section(
        Bold("Варіанти оплати:"),
        "При отриманні карта/готівка",
        "В закладі",
        marker="✅ ",
    ).as_html(),

    "shipping": as_list(
        as_marked_section(
            Bold("Варіанти доставки/замовлення:"),
            "Кур'єр",
            "Самовивіз",
            "У закладі",
            marker="✅ "
        ),
        as_marked_section(
```

```
            Bold("Неможливо:"),
            "Пошта",
            "Голуби",
            marker="❌ "
        ),
        sep='\n----------------------\n'
    ).as_html(),

    "feedback":"Будь ласка, Залишайте свої відгуки в групі (за посиланням в описі боту)",

    "catalog": "Категорії",
    "cart": "В корзині нічого немає!",

    "order":"Ваше замовлення надіслане. Чекайте телеграм-дзвінка оператора для
підтвердження."

}
```

ДОДАТОК В

Лістинг файлів директорії database

engine.py

```
import os
from sqlalchemy.ext.asyncio import AsyncSession, async_sessionmaker, create_async_engine

from database.models import Base
from database.orm_query import orm_add_banner_description, orm_create_categories

from common.texts_for_db import categories, description_for_info_pages

engine = create_async_engine(os.getenv('DB_URL'), echo=True)

session_maker = async_sessionmaker(bind=engine, class_=AsyncSession,
expire_on_commit=False)

async def create_db():
    async with engine.begin() as conn:
        await conn.run_sync(Base.metadata.create_all)

    async with session_maker() as session:
        await orm_create_categories(session, categories)
        await orm_add_banner_description(session, description_for_info_pages)


async def drop_db():
    async with engine.begin() as conn:
        await conn.run_sync(Base.metadata.drop_all)
```

models.py

```
from sqlalchemy import DateTime, ForeignKey, Numeric, String, Text, BigInteger, func, Float,
Column, Integer
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column, relationship

class Base(DeclarativeBase):
    created: Mapped[DateTime] = mapped_column(DateTime, default=func.now())
    updated: Mapped[DateTime] = mapped_column(DateTime, default=func.now(),
onupdate=func.now())

class Banner(Base):
    __tablename__ = 'banner'

    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    name: Mapped[str] = mapped_column(String(15), unique=True)
```

```
    image: Mapped[str] = mapped_column(String(150), nullable=True)
    description: Mapped[str] = mapped_column(Text, nullable=True)

class Category(Base):
    __tablename__ = 'category'

    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    name: Mapped[str] = mapped_column(String(150), nullable=False)

class Dish (Base):
    __tablename__= 'dish'

    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    name: Mapped[str] = mapped_column(String(150), nullable=False)
    description: Mapped[str] = mapped_column(Text)
    weight: Mapped[str] = mapped_column(String(20), nullable=False)
    price: Mapped[float] = mapped_column(Float(asdecimal=True), nullable=False)
    image: Mapped[str] = mapped_column(String(150))
    category_id: Mapped[int] = mapped_column(ForeignKey('category.id',
ondelete='CASCADE'), nullable=False)

    category: Mapped['Category'] = relationship(backref='dish')
    carts: Mapped['Cart'] = relationship("Cart", back_populates="dish")
    cart_items: Mapped['CartItems'] = relationship("CartItems", back_populates="dish")


class User(Base):
    __tablename__ = 'user'

    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    user_id: Mapped[int] = mapped_column(BigInteger, unique=True)
    first_name: Mapped[str] = mapped_column(String(150), nullable=True)
    last_name: Mapped[str]  = mapped_column(String(150), nullable=True)
    phone: Mapped[str]  = mapped_column(String(13), nullable=True)

class Cart(Base):
    __tablename__ = 'cart'

    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    user_id: Mapped[int] = mapped_column(ForeignKey('user.user_id', ondelete='CASCADE'),
nullable=False)
    dish_id: Mapped[int] = mapped_column(ForeignKey('dish.id', ondelete='CASCADE'),
nullable=False)
    quantity: Mapped[int]

    user: Mapped['User'] = relationship(backref='cart')
    dish: Mapped['Dish'] = relationship(backref='cart')


class CartItems(Base):
    __tablename__ = "cart_items"
```

```
    id = Column(Integer, primary_key=True, autoincrement=True)
    user_id = Column(Integer, ForeignKey('user.user_id', ondelete='CASCADE'), nullable=False)
    dish_id = Column(Integer, ForeignKey('dish.id', ondelete='CASCADE'), nullable=False)
    quantity = Column(Integer)

    dish: Mapped['Dish'] = relationship("Dish", back_populates="cart_items")
```

orm_query.py

```
import math
from sqlalchemy import select, update, delete
from sqlalchemy.ext.asyncio import AsyncSession
from sqlalchemy.orm import joinedload

from database.models import Banner, Cart, Category, Dish, User, CartItems


#BANNER

async def orm_add_banner_description(session: AsyncSession, data: dict):
    query = select(Banner)
    result = await session.execute(query)
    if result.first():
        return
    session.add_all([Banner(name=name, description=description) for name, description in
data.items()])
    await session.commit()

async def orm_change_banner_image(session: AsyncSession, name: str, image: str):
    query = update(Banner).where(Banner.name == name).values(image=image)
    await session.execute(query)
    await session.commit()

async def orm_get_banner(session: AsyncSession, page: str):
    query = select(Banner).where(Banner.name == page)
    result = await session.execute(query)
    return result.scalar()


async def orm_get_info_pages(session: AsyncSession):
    query = select(Banner)
    result = await session.execute(query)
    return result.scalars().all()

#CATEGORIES

async def orm_get_categories(session: AsyncSession):
    query = select(Category)
    result = await session.execute(query)
    return result.scalars().all()

async def orm_create_categories(session: AsyncSession, categories: list):
```

```
    query = select(Category)
    result = await session.execute(query)
    if result.first():
        return
    session.add_all([Category(name=name) for name in categories])
    await session.commit()



#DISH_admin

async def orm_add_dish(session: AsyncSession, data: dict):
    obj = Dish(
        name=data['name'],
        description=data['description'],
        weight=data['weight'],
        price=float(data['price']),
        image=data['image'],
        category_id=int(data["category"]),
    )
    session.add(obj)
    await session.commit()

async def orm_get_dishes(session: AsyncSession, category_id):
    query = select(Dish).where(Dish.category_id == int(category_id))
    result = await session.execute(query)
    return result.scalars().all()

async def orm_get_dish(session: AsyncSession, dish_id: int):
    query = select(Dish).where(Dish.id == dish_id)
    result = await session.execute(query)
    return result.scalar()

async def orm_update_dish(session: AsyncSession, dish_id: int, data):
    query = (
        update(Dish)
        .where(Dish.id == dish_id)
        .values(
            name=data["name"],
            description=data["description"],
            weight=data["weight"],
            price=float(data["price"]),
            image=data["image"],
            category_id=int(data["category"]),
        )
    )
    await session.execute(query)
    await session.commit()

async def orm_delete_dish(session: AsyncSession, dish_id: int):
    query = delete(Dish).where(Dish.id == dish_id)
```

```
    await session.execute(query)
    await session.commit()


async def orm_get_all_orders(session: AsyncSession):
    query = select(CartItems, User, Dish).join(User).join(Dish)
    result = await session.execute(query)
    orders = result.fetchall()
    return orders


#USER

async def orm_add_user(
    session: AsyncSession,
    user_id: int,
    first_name: str | None = None,
    last_name: str | None = None,
    phone: str | None = None,
):
    query = select(User).where(User.user_id == user_id)
    result = await session.execute(query)
    if result.first() is None:
        session.add(
            User(user_id=user_id, first_name=first_name, last_name=last_name, phone=phone)
        )
        await session.commit()


#CART

async def orm_add_to_cart(session: AsyncSession, user_id: int, dish_id: int):
    query = select(Cart).where(Cart.user_id == user_id, Cart.dish_id ==
dish_id).options(joinedload(Cart.dish))
    cart = await session.execute(query)
    cart = cart.scalar()
    if cart:
        cart.quantity += 1
        await session.commit()
        return cart
    else:
        session.add(Cart(user_id=user_id, dish_id=dish_id, quantity=1))
        await session.commit()


async def orm_get_user_carts(session: AsyncSession, user_id):
    query = select(Cart).filter(Cart.user_id == user_id).options(joinedload(Cart.dish))
    result = await session.execute(query)
    return result.scalars().all()


async def orm_delete_from_cart(session: AsyncSession, user_id: int, dish_id: int):
```

```
    query = delete(Cart).where(Cart.user_id == user_id, Cart.dish_id == dish_id)
    await session.execute(query)
    await session.commit()


async def orm_reduce_dish_in_cart(session: AsyncSession, user_id: int, dish_id: int):
    query = select(Cart).where(Cart.user_id == user_id, Cart.dish_id ==
dish_id).options(joinedload(Cart.dish))
    cart = await session.execute(query)
    cart = cart.scalar()

    if not cart:
        return
    if cart.quantity > 1:
        cart.quantity -= 1
        await session.commit()
        return True
    else:
        await orm_delete_from_cart(session, user_id, dish_id)
        await session.commit()
        return False
```

ДОДАТОК Г

Лістинг файлів директорії filters

chat_types.py

```
from aiogram.filters import Filter
from aiogram import Bot, types


class ChatTypeFilter(Filter):
    def __init__(self, chat_types: list[str]) -> None:
        self.chat_types = chat_types

    async def __call__(self, message: types.Message) -> bool:
        return message.chat.type in self.chat_types


class IsAdmin(Filter):
    def __init__(self) -> None:
        pass

    async def __call__(self, message: types.Message, bot: Bot) -> bool:
        return message.from_user.id in bot.my_admins_list
```

ДОДАТОК Д

Лістинг файлів директорії handlers

admin_private.py

```
from aiogram import F, Router, types
from aiogram.filters import Command, StateFilter, or_f
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup

from sqlalchemy import select

from sqlalchemy.ext.asyncio import AsyncSession

from database.orm_query import (
    orm_change_banner_image,
    orm_get_categories,
    orm_add_dish,
    orm_delete_dish,
    orm_get_info_pages,
    orm_get_dish,
    orm_get_dishes,
    orm_update_dish,
    orm_get_all_orders,
)

from filters.chat_types import ChatTypeFilter, IsAdmin

from keyboards.inline import get_callback_btns
from keyboards.reply import get_keyboard

admin_router = Router()
admin_router.message.filter(ChatTypeFilter(["private"]), IsAdmin())

ADMIN_KB = get_keyboard(
    "Додати в меню",
    "Асортимент",
    "Додати/змінити банер",
    "Подивитися всі замовлення",
    placeholder="Виберіть дію",
    sizes=(2,),
)

@admin_router.message(Command("admin"))
async def admin_features(message: types.Message):
    await message.answer("Що хочете зробити?", reply_markup=ADMIN_KB)
```

```
#замовлення

async def send_order_to_admin(session: AsyncSession, user_id: int, dish_id: int):
    # Отримати інформацію про замовлення
    user = await orm_get_user(session, user_id)
    dish = await orm_get_dish(session, dish_id)

    # Скласти текст повідомлення
    order_info = f"Нове замовлення від користувача {user_id}:\nСтрава: {dish.name}\nЦіна:
{dish.price}"

    # Відправити повідомлення адміністратору
    # Наприклад, через Telegram бота
    await bot.send_message(admin_chat_id, order_info)

@admin_router.message(F.text == "Подивитися всі замовлення")
async def view_all_orders(message: types.Message, session: AsyncSession):
    orders = await orm_get_all_orders(session)
    for order in orders:
        user_id = order.User.user_id
        dishes_ordered = ", ".join([item.Dish.name for item in order.CartItems])
        await message.answer(
            f"Користувач: {user_id}, Замовлення: {dishes_ordered}"
        )
    if not orders:
        await message.answer("Наразі немає замовлень.")
        return


#асортимент

@admin_router.message(F.text == 'Асортимент')
async def admin_menu(message: types.Message, session: AsyncSession):
    categories = await orm_get_categories(session)
    btns = {category.name : f'category_{category.id}' for category in categories}
    await message.answer("Виберіть категорію", reply_markup=get_callback_btns(btns=btns))


@admin_router.callback_query(F.data.startswith('category_'))
async def starring_at_dish(callback: types.CallbackQuery, session: AsyncSession):
    category_id = callback.data.split('_')[-1]
    for dish in await orm_get_dishes(session, int(category_id)):
        await callback.message.answer_photo(
            dish.image,
            caption=f"<strong>{dish.name}\
                  </strong>\n{dish.description}\nВартість: {round(dish.price, 2)}",
            reply_markup=get_callback_btns(
                btns={
                    "Видалити": f"delete_{dish.id}",
                    "Змінити": f"change_{dish.id}",
                },
```

```
            sizes=(2,)
        ),
    )
    await callback.answer()
    await callback.message.answer("ОК, ось меню: ")

#видалити

@admin_router.callback_query(F.data.startswith("delete_"))
async def delete_dish_callback(callback: types.CallbackQuery, session: AsyncSession):
    dish_id = callback.data.split("_")[-1]
    await orm_delete_dish(session, int(dish_id))

    await callback.answer("Страву/напій видалено")
    await callback.message.answer("Страву/напій видалено!")

#micro-FSM для банерів

class AddBanner(StatesGroup):
    image = State()

@admin_router.message(StateFilter(None), F.text == 'Додати/змінити банер')
async def add_image2(message: types.Message, state: FSMContext, session: AsyncSession):
    pages_names = [page.name for page in await orm_get_info_pages(session)]
    await message.answer(f"Завантажте фото банера.\nВ описі вкажіть для якої сторінки:\
                \n{', '.join(pages_names)}")
    await state.set_state(AddBanner.image)

@admin_router.message(AddBanner.image, F.photo)
async def add_banner(message: types.Message, state: FSMContext, session: AsyncSession):
    image_id = message.photo[-1].file_id
    for_page = message.caption.strip()
    pages_names = [page.name for page in await orm_get_info_pages(session)]
    if for_page not in pages_names:
        await message.answer(f"Введіть нормальну назву сторінки, наприклад:\
                \n{', '.join(pages_names)}")
        return
    await orm_change_banner_image(session, for_page, image_id,)
    await message.answer("Банер додано/змінено.")
    await state.clear()

@admin_router.message(AddBanner.image)
async def add_banner2(message: types.Message, state: FSMContext):
    await message.answer("Завантажте фото банера або скасувати.")




#FSM для додавання/зміни меню адміном

class AddDish(StatesGroup):
```

```
    name = State()
    description = State()
    category = State()
    weight = State()
    price = State()
    image = State()

    dish_for_change = None

    texts = {
        'AddDish:name': 'Введіть назву страви/напою знову: ',
        'AddDish:description': 'Введіть опис страви/напою знову: ',
        'AddDish:category': 'Введіть категорію страви/напою знову: ',
        'AddDish:weight': 'Введіть вагу/мл страви/напою знову: ',
        'AddDish:price': 'Введіть ціну страви/напою знову: ',
        'AddDish:image': 'Завантажте фото страви/напою знову: ',
    }


#змінити

@admin_router.callback_query(StateFilter(None), F.data.startswith("change_"))
async def change_dish_callback(
    callback: types.CallbackQuery, state: FSMContext, session: AsyncSession
):
    dish_id = callback.data.split("_")[-1]

    dish_for_change = await orm_get_dish(session, int(dish_id))

    AddDish.dish_for_change = dish_for_change
    await callback.answer()
    await callback.message.answer(
        "Введіть назву страви/напою", reply_markup=types.ReplyKeyboardRemove()
    )
    await state.set_state(AddDish.name)


#додати в меню

@admin_router.message(StateFilter(None), F.text == "Додати в меню")
async def add_dish(message: types.Message, state: FSMContext):
    await message.answer(
        "Введіть назву страви/напою", reply_markup=types.ReplyKeyboardRemove()
    )
    await state.set_state(AddDish.name)

#скасувати

@admin_router.message(StateFilter("*"), Command("Скасувати"))
@admin_router.message(StateFilter("*"), F.text.casefold() == "скасувати")
async def cancel_handler(message: types.Message, state: FSMContext) -> None:
    current_state = await state.get_state()
```

```
    if current_state is None:
        return
    if AddDish.dish_for_change:
        AddDish.dish_for_change = None
    await state.clear()
    await message.answer("Дії скасовані.", reply_markup=ADMIN_KB)

#назад

@admin_router.message(StateFilter("*"), Command("назад"))
@admin_router.message(StateFilter("*"), F.text.casefold() == "назад")
async def back_step_handler(message: types.Message, state: FSMContext) -> None:
    current_state = await state.get_state()

    if current_state == AddDish.name:
        await message.answer(
            'Попередньої дії немає або введіть назву страви/напою та напишіть "відміна"'
        )
        return

    previous = None
    for step in AddDish.__all_states__:
        if step.state == current_state:
            await state.set_state(previous)
            await message.answer(
                f"Ок, ви повернулись до попередньої дії \n {AddDish.texts[previous.state]}"
            )
            return
        previous = step

#name

@admin_router.message(AddDish.name, or_f(F.text, F.text == "."))
async def add_name(message: types.Message, state: FSMContext):
    if message.text == ".":
        await state.update_data(name=AddDish.dish_for_change.name)
    else:
            if len(message.text) >= 100:
                await message.answer(
                    "Назва страви/напою <= 100 символів.\n Введіть знову"
                )
                return

            await state.update_data(name=message.text)
    await message.answer("Введіть опис страви/напою: ")
    await state.set_state(AddDish.description)

@admin_router.message(AddDish.name)
async def add_name2(message: types.Message, state: FSMContext):
    await message.answer("Ви ввели неприпустиме значення, введіть текст назви
страви/напою: ")
```

```
#description

@admin_router.message(AddDish.description, F.text)
async def add_description(message: types.Message, state: FSMContext, session: AsyncSession):
    if message.text == "." and AddDish.dish_for_change:
        await state.update_data(description=AddDish.dish_for_change.description)
    else:
        if 4 >= len(message.text):
            await message.answer(
                "Занадто короткий опис. \n Введіть знову"
            )
            return
        await state.update_data(description=message.text)

    categories = await orm_get_categories(session)
    btns = {category.name : str(category.id) for category in categories}
    await message.answer("Виберіть категорію", reply_markup=get_callback_btns(btns=btns))
    await state.set_state(AddDish.category)



@admin_router.message(AddDish.description)
async def add_description2(message: types.Message, state: FSMContext):
    await message.answer("Ви ввели неприпустиме значення, введіть текст опису
страви/напою: ")

#callback_category

@admin_router.callback_query(AddDish.category)
async def category_choice(callback: types.CallbackQuery, state: FSMContext , session:
AsyncSession):
    if int(callback.data) in [category.id for category in await orm_get_categories(session)]:
        await callback.answer()
        await state.update_data(category=callback.data)
        await callback.message.answer('Введіть вагу/мл страви/напою.')
        await state.set_state(AddDish.weight)
    else:
        await callback.message.answer('Виберіть категорію.')
        await callback.answer()

@admin_router.message(AddDish.category)
async def category_choice2(message: types.Message, state: FSMContext):
    await message.answer("Виберіть категорію.")

#weight

@admin_router.message(AddDish.weight, or_f(F.text, F.text == "."))
async def add_weight(message: types.Message, state: FSMContext):
    if message.text == ".":
        await state.update_data(weight=AddDish.dish_for_change.weight)
    else:
        await state.update_data(weight=message.text)
```

```
    await message.answer("Введіть ціну страви/напою: ")
    await state.set_state(AddDish.price)

@admin_router.message(AddDish.weight)
async def add_weight2(message: types.Message, state: FSMContext):
    await message.answer("Ви ввели неприпустиме значення, введіть вагу/мл страви/напою:
")

#price

@admin_router.message(AddDish.price, or_f(F.text, F.text == "."))
async def add_price(message: types.Message, state: FSMContext):
    if message.text == ".":
        await state.update_data(price=AddDish.dish_for_change.price)
    else:
        try:
            float(message.text)
        except ValueError:
            await message.answer("Введіть коректне значення ціни страви/напою")
            return

        await state.update_data(price=message.text)
    await message.answer("Завантажте зображення страви/напою")
    await state.set_state(AddDish.image)

@admin_router.message(AddDish.price)
async def add_price2(message: types.Message, state: FSMContext):
    await message.answer("Введіть ціну страви/напою правильно.")

#image

@admin_router.message(AddDish.image, or_f(F.photo, F.text == "."))
async def add_image(message: types.Message, state: FSMContext, session: AsyncSession):
    if message.text and message.text == ".":
        await state.update_data(image=AddDish.dish_for_change.image)

    else:
        await state.update_data(image=message.photo[-1].file_id)
    data = await state.get_data()
    try:
        if AddDish.dish_for_change:
            await orm_update_dish(session, AddDish.dish_for_change.id, data)
        else:
            await orm_add_dish(session, data)
        await message.answer("Страву/напій додано/змінено", reply_markup=ADMIN_KB)
        await state.clear()

    except Exception as e:
        await message.answer(
            f"Помилка: \n{str(e)}\nЗверніться до програміста",
            reply_markup=ADMIN_KB
        )
```

```
        await state.clear()

    AddDish.dish_for_change = None

@admin_router.message(AddDish.image)
async def add_image2(message: types.Message, state: FSMContext):
    await message.answer("Завантажте зображення страви/напою: ")
```

menu_processing.py

```
from aiogram.types import InputMediaPhoto
from sqlalchemy.ext.asyncio import AsyncSession

from database.orm_query import (
    orm_add_to_cart,
    orm_delete_from_cart,
    orm_get_banner,
    orm_get_categories,
    orm_get_dishes,
    orm_get_user_carts,
    orm_reduce_dish_in_cart,
)
from keyboards.inline import (
    get_dishes_btns,
    get_user_cart,
    get_user_catalog_btns,
    get_user_main_btns,
)

from database.models import CartItems

from utils.paginator import Paginator


async def main_menu(session, level, menu_name):
    banner = await orm_get_banner(session, menu_name)
    image = InputMediaPhoto(media=banner.image, caption=banner.description)

    keyboards = get_user_main_btns(level=level)

    return image, keyboards


async def catalog(session, level, menu_name):
    banner = await orm_get_banner(session, menu_name)
    image = InputMediaPhoto(media=banner.image, caption=banner.description)

    categories = await orm_get_categories(session)
    keyboards = get_user_catalog_btns(level=level, categories=categories)

    return image, keyboards
```

```
def pages(paginator: Paginator):
    btns = dict()
    if paginator.has_previous():
        btns["◂ Попередня"] = "previous"

    if paginator.has_next():
        btns["Наступна ▸"] = "next"

    return btns


async def dishes(session, level, category, page):
    dishes = await orm_get_dishes(session, category_id=category)

    paginator = Paginator(dishes, page=page)
    dish = paginator.get_page()[0]

    image = InputMediaPhoto(
        media=dish.image,
        caption=f"<strong>{dish.name}\
                </strong>\n{dish.description}\nВартість: {round(dish.price, 2)}\n\
                <strong>Товар {paginator.page} з {paginator.pages}</strong>",
    )

    pagination_btns = pages(paginator)

    keyboards = get_dishes_btns(
        level=level,
        category=category,
        page=page,
        pagination_btns=pagination_btns,
        dish_id=dish.id,
    )

    return image, keyboards


async def carts(session, level, menu_name, page, user_id, dish_id):
    if menu_name == "order":
        try:
            async with session.begin():
                # Створити об'єкт замовлення
                order = CartItems(user_id=user_id, dish_id=dish_id, quantity=1)
                # Додати об'єкт замовлення до сесії
                session.add(order)
                # Зберегти зміни в базі даних
                await session.commit()

                # Надіслати інформацію про замовлення адміністратору
                await send_order_to_admin(session, user_id, dish_id)
        except Exception as e:
```

```
            print(f"Помилка під час збереження замовлення: {e}")

    elif menu_name == "delete":
        await orm_delete_from_cart(session, user_id, dish_id)
        if page > 1:
            page -= 1
    elif menu_name == "decrement":
        is_cart = await orm_reduce_dish_in_cart(session, user_id, dish_id)
        if page > 1 and not is_cart:
            page -= 1
    elif menu_name == "increment":
        await orm_add_to_cart(session, user_id, dish_id)



    #оновити вміст корзини
    carts = await orm_get_user_carts(session, user_id)

    if not carts:
        banner = await orm_get_banner(session, "cart")
        image = InputMediaPhoto(
            media=banner.image, caption=f"<strong>{banner.description}</strong>"
        )

        keyboards = get_user_cart(
            level=level,
            page=None,
            pagination_btns=None,
            dish_id=None,
        )

    else:
        paginator = Paginator(carts, page=page)

        cart = paginator.get_page()[0]

        cart_price = round(cart.quantity * cart.dish.price, 2)
        total_price = round(
            sum(cart.quantity * cart.dish.price for cart in carts), 2
        )
        image = InputMediaPhoto(
            media=cart.dish.image,
            caption=f"<strong>{cart.dish.name}</strong>\n{cart.dish.price}$ x {cart.quantity} =
{cart_price}$\
                \nТовар {paginator.page} з {paginator.pages} в корзині.\nЗагальна вартість
товарів в корзині {total_price}",
        )

        pagination_btns = pages(paginator)

        keyboards = get_user_cart(
            level=level,
```

```
            page=page,
            pagination_btns=pagination_btns,
            dish_id=cart.dish.id,
        )

    return image, keyboards


async def get_menu_content(
    session: AsyncSession,
    level: int,
    menu_name: str,
    category: int | None = None,
    page: int | None = None,
    dish_id: int | None = None,
    user_id: int | None = None,
):
    if level == 0:
        return await main_menu(session, level, menu_name)
    elif level == 1:
        return await catalog(session, level, menu_name)
    elif level == 2:
        return await dishes(session, level, category, page)
    elif level == 3:
        return await carts(session, level, menu_name, page, user_id, dish_id)
```

user_group.py

```
from string import punctuation

from aiogram import F, Bot, types, Router
from aiogram.filters import Command

from filters.chat_types import ChatTypeFilter
from common.restricted_words import restricted_words


user_group_router = Router()
user_group_router.message.filter(ChatTypeFilter(["group", "supergroup"]))
user_group_router.edited_message.filter(ChatTypeFilter(["group", "supergroup"]))


@user_group_router.message(Command("admin"))
async def get_admins(message: types.Message, bot: Bot):
    chat_id = message.chat.id
    admins_list = await bot.get_chat_administrators(chat_id)

    admins_list = [
        member.user.id
        for member in admins_list
        if member.status == "creator" or member.status == "administrator"
    ]
```

```
    bot.my_admins_list = admins_list
    if message.from_user.id in admins_list:
        await message.delete()
    #print(admins_list)


def clean_text(text: str):
    return text.translate(str.maketrans("", "", punctuation))


@user_group_router.edited_message()
@user_group_router.message()
async def cleaner(message: types.Message):
    if restricted_words.intersection(clean_text(message.text.lower()).split()):
        await message.answer(
            f"{message.from_user.first_name}, дотримуйтесь правил спільноти!"
        )
        await message.delete()
        # await message.chat.ban(message.from_user.id)
```

user_private.py

```
from aiogram import F, types, Router
from aiogram.filters import CommandStart

from sqlalchemy.ext.asyncio import AsyncSession
from database.orm_query import (
    orm_add_to_cart,
    orm_add_user,
)

from filters.chat_types import ChatTypeFilter
from handlers.menu_processing import get_menu_content
from keyboards.inline import MenuCallBack, get_callback_btns

from database.models import Cart

user_private_router = Router()
user_private_router.message.filter(ChatTypeFilter(["private"]))


#@user_private_router.message(CommandStart())
#@user_private_router.message(or_f(F.text.lower().contains("почат")),
(F.text.lower().contains("старт")))
#async def start_cmd(message: types.Message):
#    await message.answer(
#        "Вітаємо! Раді, що ви приєднались до нас! В TARANTINO представлені страви з
європейської, української та паназійської кухні в авторському виконанні!",
#        reply_markup=get_callback_btns(btns={
#            'Натисніть сюди': 'some_1'
#            }))
```

```
@user_private_router.message(CommandStart())
async def start_cmd(message: types.Message, session: AsyncSession):
    media, reply_markup = await get_menu_content(session, level=0, menu_name="main")

    await message.answer_photo(media.media, caption=media.caption,
reply_markup=reply_markup)

#add_to_cart
async def add_to_cart(callback: types.CallbackQuery, callback_data: MenuCallBack, session:
AsyncSession):
    user = callback.from_user
    await orm_add_user(
        session,
        user_id=user.id,
        first_name=user.first_name,
        last_name=user.last_name,
        phone=None,
    )
    await orm_add_to_cart(session, user_id=user.id, dish_id=callback_data.dish_id)
    await callback.answer("Страву/напій додано в корзину.")



@user_private_router.callback_query(MenuCallBack.filter())
async def user_menu(callback: types.CallbackQuery, callback_data: MenuCallBack, session:
AsyncSession):

    if callback_data.menu_name == "add_to_cart":
        await add_to_cart(callback, callback_data, session)
        return

    media, reply_markup = await get_menu_content(
        session,
        level=callback_data.level,
        menu_name=callback_data.menu_name,
        category=callback_data.category,
        page=callback_data.page,
        dish_id=callback_data.dish_id,
        user_id=callback.from_user.id,
    )

    await callback.message.edit_media(media=media, reply_markup=reply_markup)
    await callback.answer()


#orders

@user_private_router.callback_query(F.data.startswith("order_"))
async def order_callback(callback: types.CallbackQuery, session: AsyncSession):
    #ідентифікатор страви з даних кнопки
    dish_id = int(callback.data.split("_")[-1])
```

```
    #інформація про користувача з повідомлення
    user_id = callback.message.chat.id
    # Збереження даних про замовлення в базу даних
    try:
        async with session.begin():
            new_order = Cart(user_id=user_id, dish_id=dish_id, quantity=1)
            session.add(new_order)
        await callback.answer("Ваше замовлення надіслане. Чекайте телеграм-дзвінка
оператора для підтвердження.")
    except Exception as e:
        await callback.answer(f"Помилка під час збереження замовлення: {str(e)}")



async def create_order_from_cart(session: AsyncSession, user_id: int, cart_items: list):
    for item in cart_items:
        order_item = CartItems(user_id=user_id, dish_id=item['dish_id'], quantity=item['quantity'])
        session.add(order_item)
    await session.commit()
    await clear_cart(session, user_id)

async def clear_cart(session: AsyncSession, user_id: int):
    # Видаляємо дані з корзини після створення замовлення
    await session.execute(delete(Cart).where(Cart.user_id == user_id))
    await session.commit()
```

ДОДАТОК Е

Лістинг файлів директорії keyboards

inline.py

```
from aiogram.filters.callback_data import CallbackData
from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup
from aiogram.utils.keyboard import InlineKeyboardBuilder

class MenuCallBack(CallbackData, prefix="menu"):
    level: int
    menu_name: str
    category: int | None = None
    page: int = 1
    dish_id: int | None = None


def get_user_main_btns(*, level: int, sizes: tuple[int] = (2,)):
    keyboard = InlineKeyboardBuilder()
    btns = {
        "Меню 🍕": "catalog",
        "Корзина 🛒": "cart",
        "Про ресторан ℹ️": "about",
        "Локація ресторану": "location",
        "Графік роботи": "work_time",
        "Оплата 💰": "payment",
        "Доставка ⛵": "shipping",
        "Відгуки" : "feedback",
    }
    for text, menu_name in btns.items():
        if menu_name == 'catalog':
            keyboard.add(InlineKeyboardButton(text=text,
                callback_data=MenuCallBack(level=level+1, menu_name=menu_name).pack()))
        elif menu_name == 'cart':
            keyboard.add(InlineKeyboardButton(text=text,
                callback_data=MenuCallBack(level=3, menu_name=menu_name).pack()))
        else:
            keyboard.add(InlineKeyboardButton(text=text,
                callback_data=MenuCallBack(level=level, menu_name=menu_name).pack()))

    return keyboard.adjust(*sizes).as_markup()


def get_user_catalog_btns(*, level: int, categories: list, sizes: tuple[int] = (2,)):
    keyboard = InlineKeyboardBuilder()
```

```
    keyboard.add(InlineKeyboardButton(text='Назад',
                 callback_data=MenuCallBack(level=level-1, menu_name='main').pack()))
    keyboard.add(InlineKeyboardButton(text='Корзина 🛒',
                 callback_data=MenuCallBack(level=3, menu_name='cart').pack()))

    for c in categories:
        keyboard.add(InlineKeyboardButton(text=c.name,
                 callback_data=MenuCallBack(level=level+1, menu_name=c.name,
category=c.id).pack()))

    return keyboard.adjust(*sizes).as_markup()


def get_dishes_btns(
    *,
    level: int,
    category: int,
    page: int,
    pagination_btns: dict,
    dish_id: int,
    sizes: tuple[int] = (2, 1)
):
    keyboard = InlineKeyboardBuilder()

    keyboard.add(InlineKeyboardButton(text='Назад',
                 callback_data=MenuCallBack(level=level-1, menu_name='catalog').pack()))
    keyboard.add(InlineKeyboardButton(text='Корзина 🛒',
                 callback_data=MenuCallBack(level=3, menu_name='cart').pack()))
    keyboard.add(InlineKeyboardButton(text='Купити 💷',
                 callback_data=MenuCallBack(level=level, menu_name='add_to_cart',
dish_id=dish_id).pack()))

    keyboard.adjust(*sizes)

    row = []
    for text, menu_name in pagination_btns.items():
        if menu_name == "next":
            row.append(InlineKeyboardButton(text=text,
                    callback_data=MenuCallBack(
                        level=level,
                        menu_name=menu_name,
                        category=category,
                        page=page + 1).pack()))

        elif menu_name == "previous":
            row.append(InlineKeyboardButton(text=text,
                    callback_data=MenuCallBack(
                        level=level,
                        menu_name=menu_name,
                        category=category,
                        page=page - 1).pack()))
```

```
    return keyboard.row(*row).as_markup()


def get_user_cart(
    *,
    level: int,
    page: int | None,
    pagination_btns: dict | None,
    dish_id: int | None,
    sizes: tuple[int] = (3,)
):
    keyboard = InlineKeyboardBuilder()
    if page:
        keyboard.add(InlineKeyboardButton(text='Видалити',
                callback_data=MenuCallBack(level=level, menu_name='delete', dish_id=dish_id,
page=page).pack()))
        keyboard.add(InlineKeyboardButton(text='-1',
                callback_data=MenuCallBack(level=level, menu_name='decrement',
dish_id=dish_id, page=page).pack()))
        keyboard.add(InlineKeyboardButton(text='+1',
                callback_data=MenuCallBack(level=level, menu_name='increment',
dish_id=dish_id, page=page).pack()))

        keyboard.adjust(*sizes)

        row = []
        for text, menu_name in pagination_btns.items():
            if menu_name == "next":
                row.append(InlineKeyboardButton(text=text,
                    callback_data=MenuCallBack(level=level, menu_name=menu_name, page=page
+ 1).pack()))
            elif menu_name == "previous":
                row.append(InlineKeyboardButton(text=text,
                    callback_data=MenuCallBack(level=level, menu_name=menu_name, page=page
- 1).pack()))

        keyboard.row(*row)

        row2 = [
        InlineKeyboardButton(text='На головну 🏠',
                callback_data=MenuCallBack(level=0, menu_name='main').pack()),
        InlineKeyboardButton(text='Замовити',
                callback_data=MenuCallBack(level=0, menu_name='order').pack()),
        ]
        return keyboard.row(*row2).as_markup()
    else:
        keyboard.add(
            InlineKeyboardButton(text='На головну 🏠',
                callback_data=MenuCallBack(level=0, menu_name='main').pack()))

        return keyboard.adjust(*sizes).as_markup()
```

```
def get_callback_btns(*, btns: dict[str, str], sizes: tuple[int] = (2,)):
    keyboard = InlineKeyboardBuilder()

    for text, data in btns.items():
        keyboard.add(InlineKeyboardButton(text=text, callback_data=data))

    return keyboard.adjust(*sizes).as_markup()
```

reply.py

```
from aiogram.types import KeyboardButton
from aiogram.utils.keyboard import ReplyKeyboardBuilder


def get_keyboard(
    *btns: str,
    placeholder: str = None,
    request_contact: int = None,
    request_location: int = None,
    sizes: tuple[int] = (2,),
):
    keyboard = ReplyKeyboardBuilder()

    for index, text in enumerate(btns, start=0):

        if request_contact and request_contact == index:
            keyboard.add(KeyboardButton(text=text, request_contact=True))

        elif request_location and request_location == index:
            keyboard.add(KeyboardButton(text=text, request_location=True))
        else:
            keyboard.add(KeyboardButton(text=text))

    return keyboard.adjust(*sizes).as_markup(
            resize_keyboard=True, input_field_placeholder=placeholder)
```

ДОДАТОК Ж

Лістинг файлів директорії middlewares

db.py

```
from typing import Any, Awaitable, Callable, Dict

from aiogram import BaseMiddleware
from aiogram.types import Message, TelegramObject

from sqlalchemy.ext.asyncio import async_sessionmaker


class DataBaseSession(BaseMiddleware):
    def __init__(self, session_pool: async_sessionmaker):
        self.session_pool = session_pool


    async def __call__(
        self,
        handler: Callable[[TelegramObject, Dict[str, Any]], Awaitable[Any]],
        event: TelegramObject,
        data: Dict[str, Any],
    ) -> Any:
        async with self.session_pool() as session:
            data['session'] = session
            return await handler(event, data)
```

ДОДАТОК К

Лістинг файлів директорії utils)

paginator.py

```
import math

class Paginator:
    def __init__(self, array: list | tuple, page: int=1, per_page: int=1):
        self.array = array
        self.per_page = per_page
        self.page = page
        self.len = len(self.array)
        self.pages = math.ceil(self.len / self.per_page)

    def __get_slice(self):
        start = (self.page - 1) * self.per_page
        stop = start + self.per_page
        return self.array[start:stop]

    def get_page(self):
        page_items = self.__get_slice()
        return page_items

    def has_next(self):
        if self.page < self.pages:
            return self.page + 1
        return False

    def has_previous(self):
        if self.page > 1:
            return self.page - 1
        return False

    def get_next(self):
        if self.page < self.pages:
            self.page += 1
            return self.get_page()
        raise IndexError(f'Next page does not exist. Use has_next() to check before.')

    def get_previous(self):
        if self.page > 1:
            self.page -= 1
            return self.__get_slice()
        raise IndexError(f'Previous page does not exist. Use has_previous() to check before.')
```

ДОДАТОК Л

Лістинг файлу «.env»

```
TOKEN=7069654102:AAFoKLhoU8D1aYdULWOYqyG8HhQ8WFcEO1M
DB_URL=sqlite+aiosqlite:///my_base.db
#PAYMENTS_TOKEN = "..."
```