МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: Онлайн сервіс замовлення і доставки безалкогольних напоїв і сервірованих страв на основі Telegram-бота

Здобувача(ки) групи ІТ-03 Перепелиці Дениса Віталійовича
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

____ _____ Денис ПЕРЕПЕЛИЦЯ
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник к. т. н., доц. Едуард КУЗНЄЦОВ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В. о. зав. кафедри ІТ

_____Світлана ВАЩЕНКО
«_____»_____ 2024 р.

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

Перепелиці Денису Віталійовичу

1 **Тема роботи** *Онлайн сервіс замовлення і доставки безалкогольних напоїв і сервірованих страв на основі Telegram-бота*

керівник роботи *Кузнєцов Едуард Геннадійович, к.т.н., доцент*,

затверджені наказом по університету від « 7 » травня 2024 р. №0482-VI

2 Строк подання студентом роботи « 26 » травня 2024 р.

3 Вхідні дані до роботи технічне завдання, спектр послуг сервісу та меню

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області, постановка задачі, моделювання та проектування, розробка, використання програмного продукту, висновки

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) порівняння продуктів-аналогів, архітектура програмного продукту, діаграма у нотації IDEF0, діаграма декомпозиції у нотації IDEF0, діаграма варіантів використання сервісу, діаграма бази даних, архітектура чат-боту, приклади роботи програмного продукту.

6 Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7 Дата видачі завдання 08.04.2024

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підготовка специфікації та визначення властивостей бота	08.04.24 – 12.04.24	
2	Розробка меню бота	15.04.24 – 19.04.24	
3	Створення функціоналу боту	22.04.24 – 26.04.24	
4	Створення та наповнення бази даних	29.04.24 – 03.05.24	
5	Перевірка працездатності боту	06.05.24 – 10.05.24	
6	Реліз боту	13.05.24 – 17.05.24	
7	Написання супровідної документації	20.05.24 – 26.05.24	

Студент _____ Денис ПЕРЕПЕЛИЦЯ
(підпис)

Керівник роботи _____ к.т.н., доц. Едуард КУЗНЄЦОВ
(підпис)

Анотація

Тема кваліфікаційної роботи бакалавра “Онлайн сервіс замовлення і доставки безалкогольних напоїв і сервірованих страв на основі Telegram-бота”.

Пояснювальна записка складається зі вступу, 3 розділів, висновків, списку використаних джерел із 20 найменувань, додатків. Загальний обсяг роботи – 70 сторінок, у тому числі 38 сторінки основного тексту, 3 сторінки використаних джерел, 29 сторінок додатків.

Кваліфікаційну роботу бакалавра присвячено розробці чат-боту для месенджеру Telegram по замовленню та доставки їжі.

У першому розділі було визначено актуальність роботи, сформовано мету проекту, було досліджено літературні джерела, які стосуються теми роботи, провівся аналіз аналогів продукту.

У другому розділі виконане структурно-функціональне моделювання, моделювання варіантів використання продукту, а також інші діаграми.

У третьому розділі описані основні етапи створення телеграм-боту та бази даних, наведено приклади використання створеного сервісу.

Результатом є розроблений онлайн сервіс замовлення і доставки безалкогольних напоїв і сервірованих страв на основі Telegram-бота.

Ключові слова: чат-бот, Telegram, доставка, їжа, напої, база даних, сервіс.

Зміст

Вступ

У сучасному світі, де цифрові технології стрімко розвиваються, важливість зручного та швидкого доступу до різноманітних послуг стає все більш очевидною. Особливо це стосується сфери обслуговування, де зручність і швидкість замовлення є ключовими факторами для забезпечення задоволення клієнтів. Враховуючи ці тенденції, ми розпочали проект, метою якого є розробка інтернет-магазину для ресторану, що спеціалізується на продажу страв та безалкогольних напоїв.

Основна ідея проекту полягає у створенні простого та інтуїтивно зрозумілого сервісу, який дозволить користувачам легко ознайомлюватися з асортиментом страв та напоїв, переглядати їх ціни і здійснювати замовлення безпосередньо через Telegram-бота. Такий підхід не лише підвищить зручність для клієнтів, але й дозволить ресторану оптимізувати процеси обробки замовлень та покращити взаємодію з аудиторією.

Для досягнення цієї мети ми плануємо виконати ряд важливих етапів, серед яких:

- Проведення детального аналізу ринку та вимог користувачів, що допоможе зрозуміти їхні потреби та очікування;
- Визначення необхідного функціоналу бота та шляхів взаємодії з користувачами для забезпечення максимальної зручності;
- Розробка Telegram-бота, який стане основним інструментом для користувачів у процесі замовлення страв та напоїв;
- Створення бази даних для зберігання інформації про користувачів та їхні замовлення;
- Розробка веб-інтерфейсу для адміністраторів, що дозволить ефективно керувати замовленнями та аналізувати діяльність;

- Проведення функціонального тестування для виявлення та виправлення можливих недоліків;
- Запуск сервісу та перевірка його працездатності в реальних умовах.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБРАСТІ

1.1 Огляд останніх досліджень і публікацій

Згідно з дослідженням McKinsey & Company, ринок онлайн-замовлень їжі зріс на 20% у 2023 році порівняно з попереднім роком. Важливим фактором цього зростання є зміна споживчих звичок під час пандемії COVID-19, коли люди стали більше замовляти їжу онлайн для зменшення контактів та зручності.

Дослідження, проведене компанією Juniper Research, показало, що до 2025 року більше 70% замовлень в онлайн-ресторанах здійснюватимуться за допомогою чат-ботів. Це пояснюється тим, що чат-боти забезпечують швидку та ефективну взаємодію з клієнтами, допомагаючи їм знайти потрібні страви, оформити замовлення та отримати необхідну інформацію без затримок.

Опитування, проведене компанією Deloitte, виявило, що користувачі очікують від онлайн-замовлень високого рівня персоналізації, можливості швидко знаходити потрібні страви та напої, а також отримувати актуальну інформацію про акції та спеціальні пропозиції. Крім того, важливими факторами є зручність оплати та швидка доставка.

Згідно з даними дослідження Gartner, використання ефективних систем управління базами даних (СУБД) є ключовим аспектом успішної роботи інтернет-магазинів їжі. Вони дозволяють забезпечити швидкий доступ до інформації про користувачів та їхні замовлення, що, у свою чергу, покращує обслуговування клієнтів та оптимізує внутрішні процеси.

Дослідження, проведене компанією IBM, показало, що наявність зручного веб-інтерфейсу для адміністраторів є критично важливою для успішного управління онлайн-ресторанами. Зручний інтерфейс дозволяє ефективно керувати замовленнями, аналізувати продажі та ефективність роботи персоналу, а також швидко реагувати на зміни ринку та впроваджувати нові стратегії. Це

допомагає підвищити продуктивність, зменшити витрати часу та покращити загальну ефективність управління системою.

Отже, останні дослідження підтверджують, що розвиток інтернет-магазинів для замовлення їжі є перспективним напрямом, який відповідає сучасним вимогам споживачів та бізнесу. Використання чат-ботів, ефективних баз даних та зручних веб-інтерфейсів для адміністраторів сприяє підвищенню ефективності роботи ресторанів та задоволенню потреб клієнтів. Ці знання будуть використані для розробки нашого сервісу, щоб забезпечити його успішне впровадження та функціонування.

1.2 Аналіз існуючих продуктів-аналогів

1. **Ikura** – це мережа онлайн ресторанів, представлена в багатьох містах України. Вони відкрилися 12 вересня 2016 року у місті Сєвєродонецьку, а з 11 жовтня 2018 року готуємо та доставляємо страви і по лівому березі Києва. Ikura спеціалізується на таких стравах: суші, піци, закуски, супи, бургери, італійські пасти, салати, боули і поке, десерти, напої та інше.

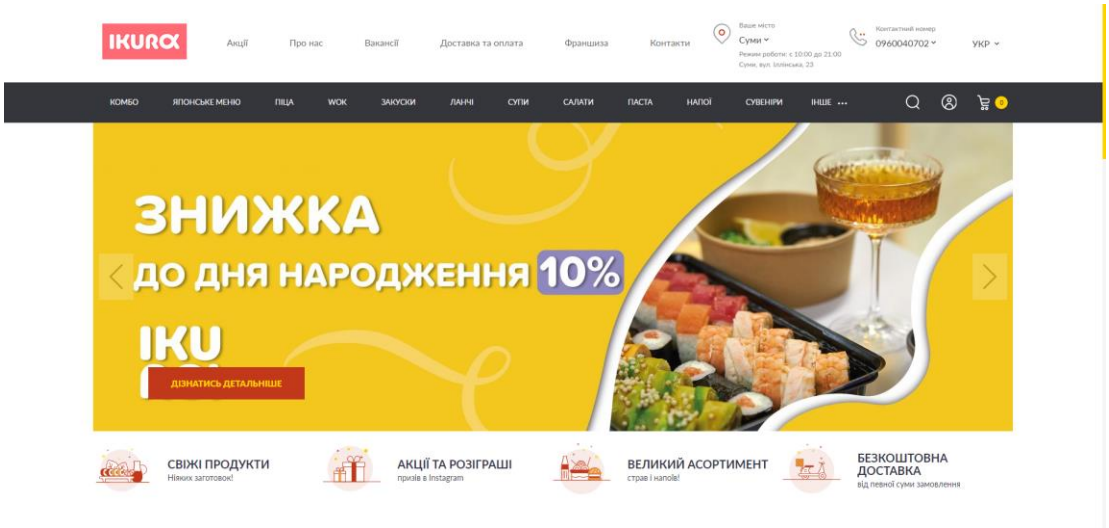


Рисунок 1. Демонстрація роботи сервісу Ikura

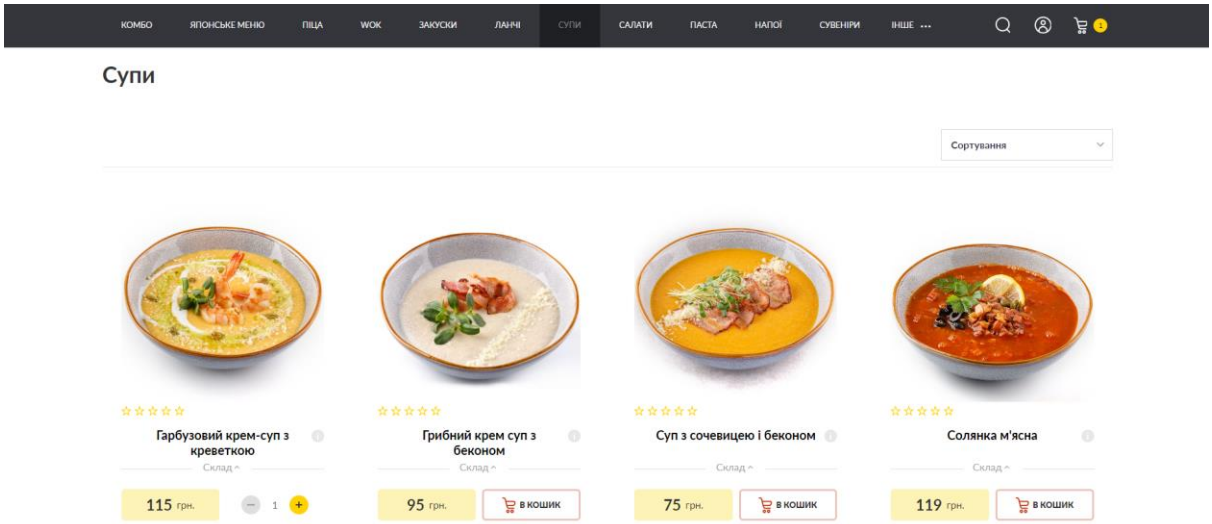


Рисунок 2. Розділ Супи сервісу Ikura

Рисунок 3. Кошик в сервісі Ikura

2. **New York** – це атмосферний і стильний заклад, який спеціалізується на доставці їжі, а також має власні кафе, в які люди можуть прийти та насолодитися стравами, або напоями. В асортимент страв входять: піца, суші, салати, перші страви, другі страви, хлібні вироби, млинці, десерти, торти (на замовлення) та інше.

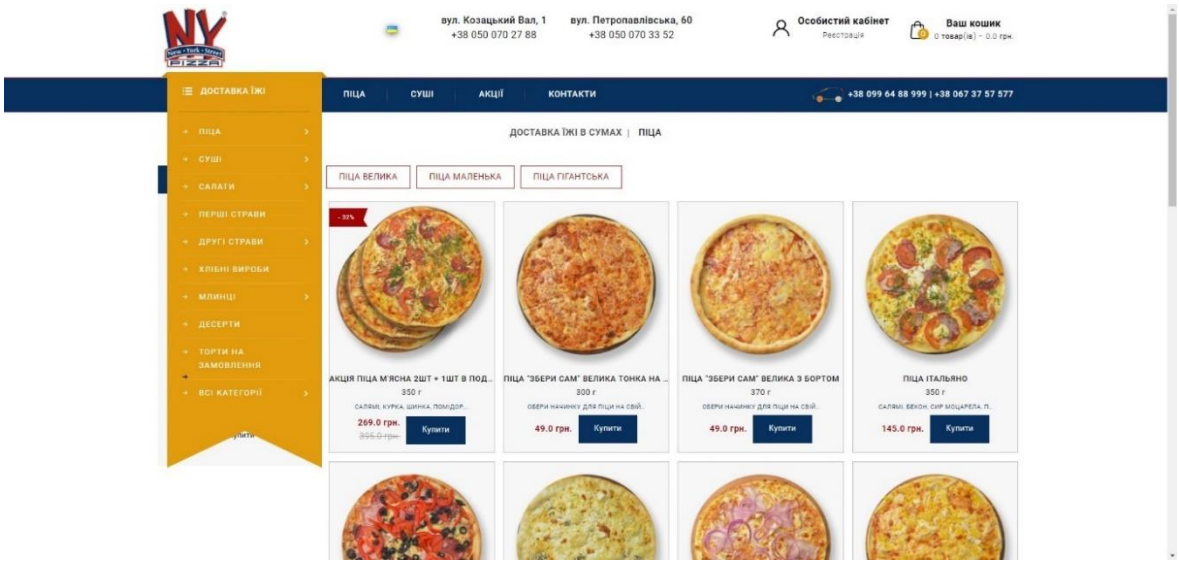


Рисунок 4. Демонстрація роботи сервісу New York

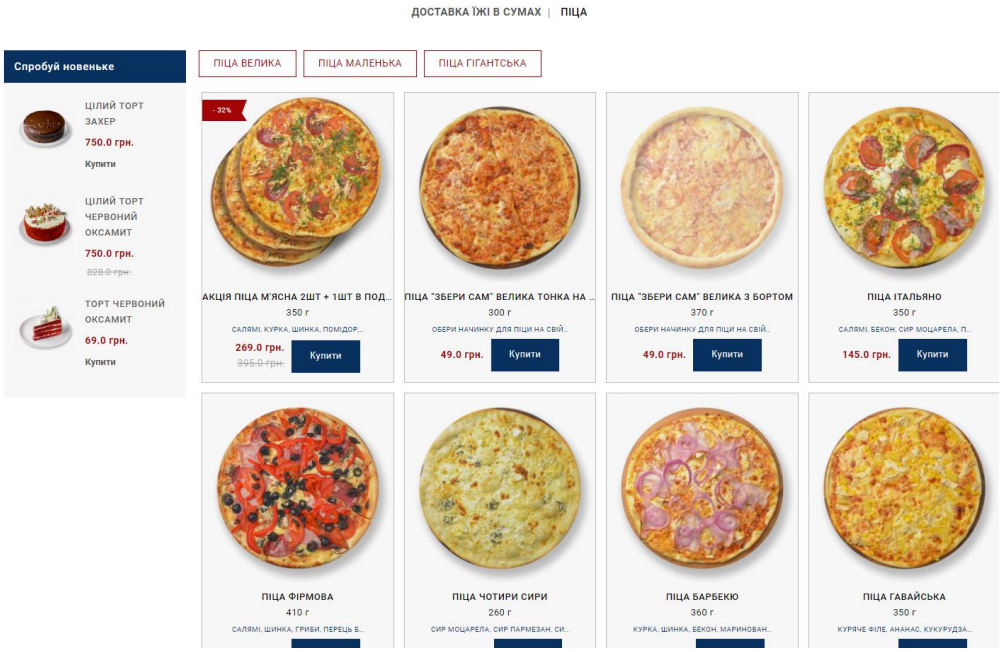


Рисунок 5. Розділ Піца в сервісі New York

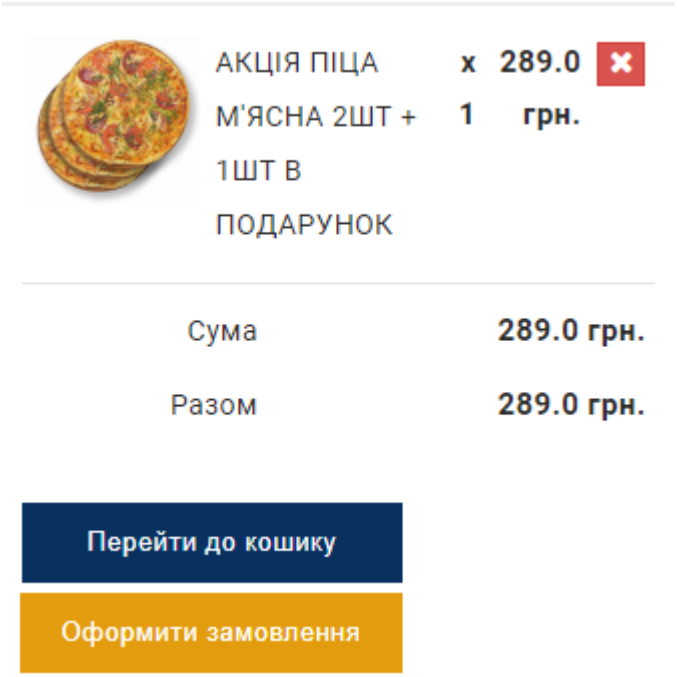


Рисунок 6. Кошик сервісу New York

3. **MISTER.AM** – це сервіс з доставки страв та напоїв з багатьох ресторанів та кафе в містах України. Користуватися сервісом може будь-хто. Потрібен просто доступ до Інтернету. Зручна навігація та простий вибір необхідного закладу та самої страви прискорює та полегшує процес замовлення. У багатьох випадках доставка обідів та вечерь, сніданків та ланчів здійснюється безкоштовно. Це вагоме та дуже приємне доповнення до того, що всі напої та їжу ресторани-партнери надають без націнки. Отже, найзручніше та найвигідніше замовлення їжі на будинок можуть здійснювати за допомогою цього сервісу.

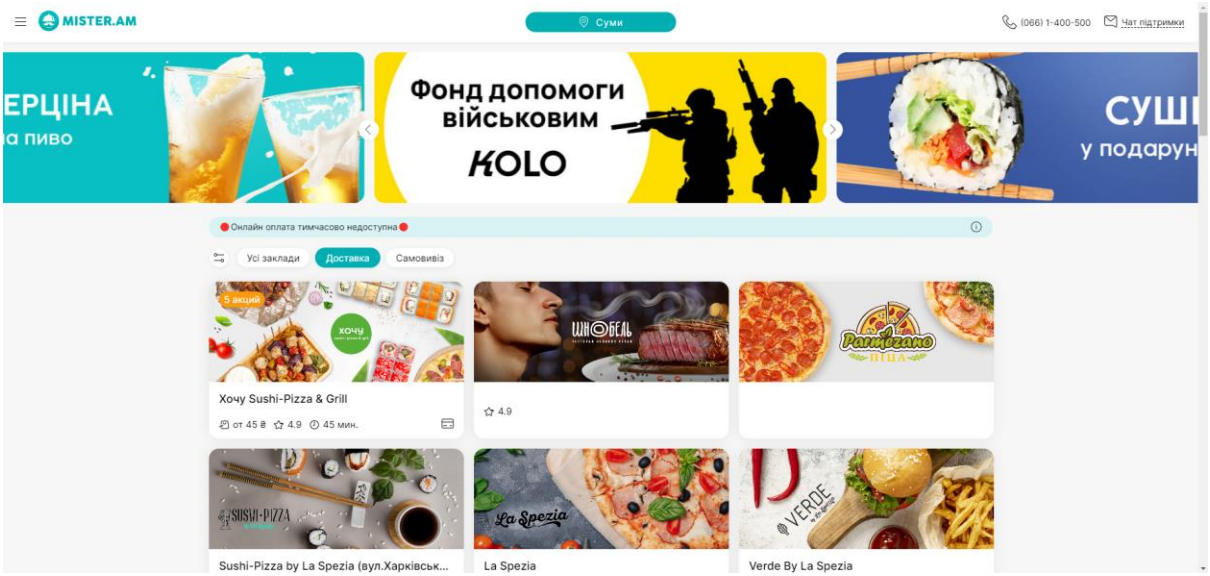


Рисунок 7. Демонстрація роботи сервісу MISTER.AM

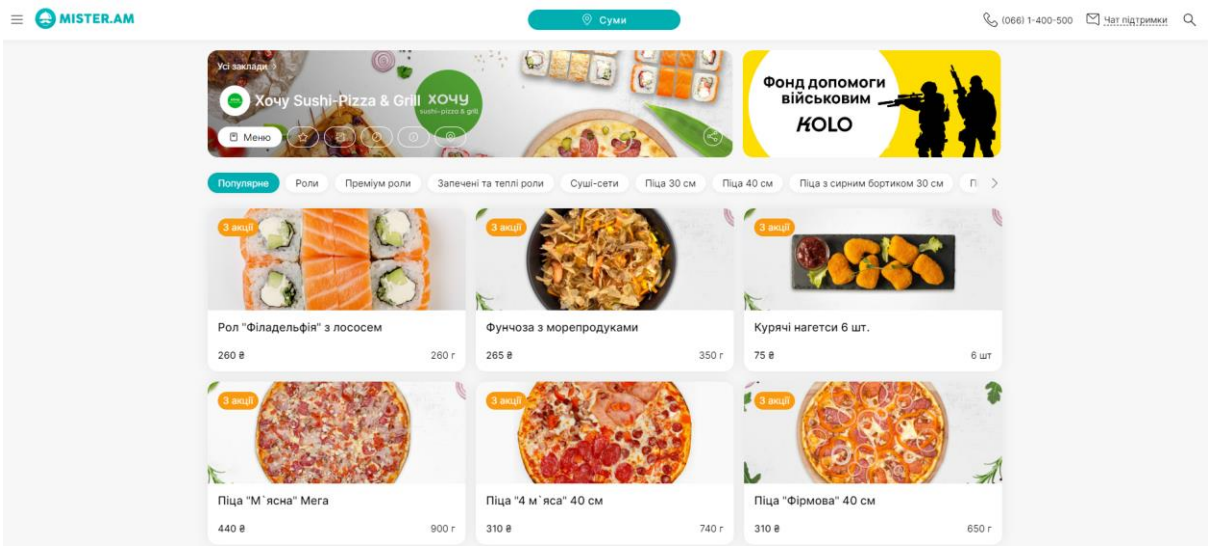


Рисунок 8. Перегляд меню в сервісі MISTER.AM

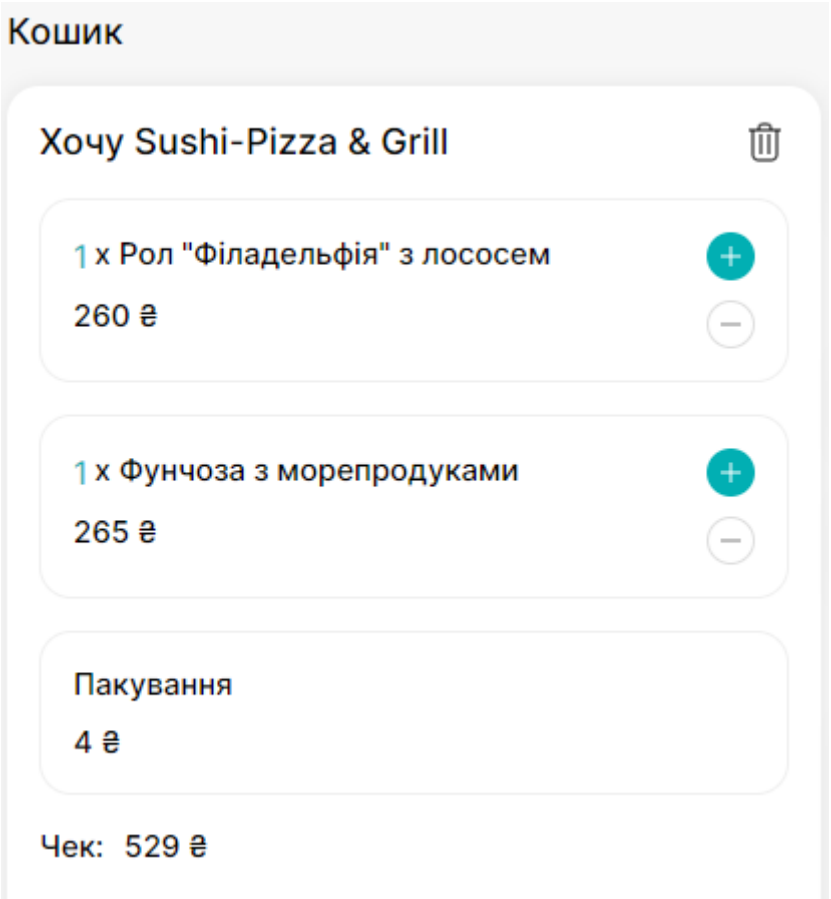


Рисунок 9. Кошик в сервісі MISTER.AM

Таблиця 2.1. Проведемо порівняння між додатками:

Параметр/сервіс	**IKURA**	**NEW YORK**	**MISTER.AM**	**GUSTO DELIGHT**
Асортимент	Ресторан, фаст-фуд, японська їжа	Ресторан, фаст-фуд, японська їжа	Ресторани, кафе, фаст-фуд	Ресторан, японська їжа, фаст-фуд
Інтерфейс користувача	Інтуїтивно зрозумілий	Інтуїтивно зрозумілий	Інтуїтивно зрозумілий	Інтуїтивно зрозумілий

Продовження таблиці 2.1 Проведемо порівняння між додатками

Час доставки	Залежить від завантаженості працівників	Залежить від завантаженості працівників	Залежить від завантаженості працівників	Залежить від завантаженості працівників
Вартість доставки	Залежить від тарифу доставки	Залежить від тарифу доставки	Залежить від тарифу доставки	Залежить від тарифу доставки
Платіжні методи	Готівка, онлайн-оплата	Готівка, онлайн-оплата	Готівка, онлайн-оплата	Готівка, онлайн-оплата

Отже, після аналізу додатків, можна зробити такі висновки, що кожен додаток зручний, містить широкий асортимент та зручну навігацію по додатку, але, на мою думку, можна виділити MISTER.AM, так як там найбільший асортимент, через те що це сервіс, який містить багато різноманітних кафе з їх меню.

1.3 Постановка задачі

Мета цього проекту – є розробка телеграм-бота з продажу страв у ресторані та безалкогольних напоїв, з можливістю ознайомлення з товаром, його ціною. Розроблюваний сервіс має бути простим, для охоплення аудиторії.

Для випускної кваліфікаційної роботи сформовані наступні завдання для роботи:

- Провести аналіз ринку та вимог користувачів;

- Визначити функціонал бота та взаємодію з користувачем;
- Розробити Telegram-бота та його функціонал;
- Розробити базу даних та веб-інтерфейс для адміністраторів;
- Функціональне тестування та запуск сервісу

Цільовою аудиторією сервісу будуть люди, які зацікавлені в простому та бистрому замовленні їжі та напоїв.

Для реалізації було використано такі засоби, як мова програмування Python, база даних SQLite3, середовище розробки Pycharm, фреймворк Django.

Python було обрано через його простоту і зрозумілістю синтаксису, що значно полегшує процес розробки та підтримки коду. Це дозволяє швидше розробляти і впроваджувати нові функціональні можливості. А також через такі переваги:

- Багатство бібліотек та фреймворків;
- швидка інтеграція з базами даних;
- гнучкість і масштабованість;
- інтеграція з API;

SQLite3 - це легка реляційна система керування базами даних, яка є популярною завдяки своїй простоті у використанні та потужним можливостям. Головні переваги у використанні SQLite3:

- Створення файлів бази даних;
- створення, визначення, зміна та видалення таблиць;
- перегляд, редагування, додавання та видалення записів;
- імпорт та експорт даних;
- вбудованість і портативність;
- швидкість та ефективність

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ЧАТ-БОТУ

2.1 Структурно-функціональне моделювання

IDEF0 - це методологія графічного опису систем і процесів діяльності організації як безлічі взаємозалежних функцій. Інформаційні та матеріальні потоки, які перетворюються в бізнес-процесі, позначаються стрілками зліва та є входами. Матеріальні та інформаційні потоки, які перетворюються на процеси, позначаються стрілками зверху та є управлінням. Інструменти та ресурси, за допомогою яких бізнес-процес реалізується, позначаються стрілками знизу та є механізмами. Вихід бізнес-процесу, описаного в стандарті IDEF0, повністю відповідає за змістом виходу процесу.

Процес зазначений як «Замовлення в Gusto Delight».

Вхід: Запит на замовлення.

Керування: Асортимент, Правила користування.

Механізми: Користувач, Месенджер Telegram, GustoDelightBot, апаратне забезпечення.

Результат: Сформоване замовлення.

Діаграма IDEF0 для чат-бота для месенджеру Telegram по замовленню доставки їжі в Gusto Delight зображена на рисунку 2.1.

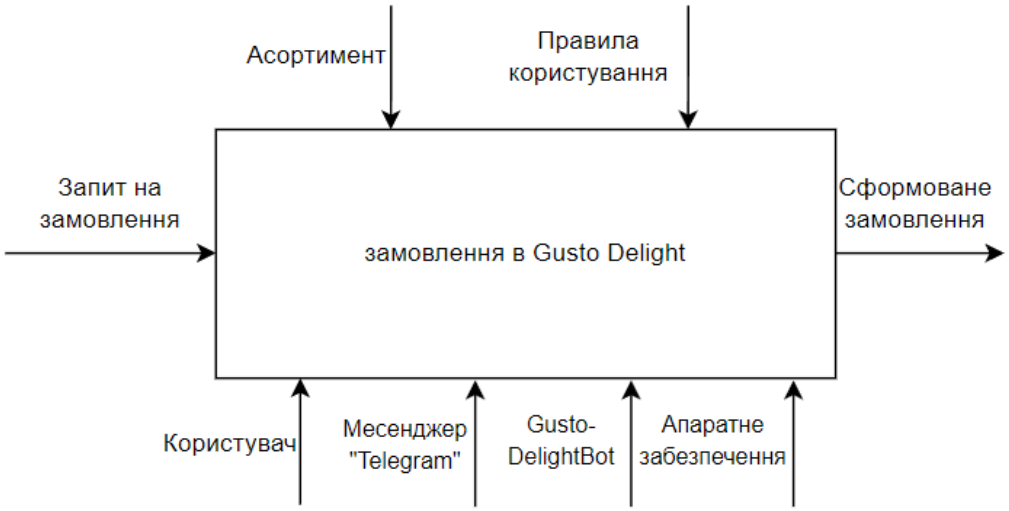


Рисунок 2.1 – Діаграма у нотації IDEF0

Таблиця 2.1 – Дані з діаграми декомпозиції IDEF0

Підпроцес	Вхід	Управління	Механізм	Вихід
Початок роботи з ботом	Запит на замовлення	Правила користування	Месенджер “Telegram”, Апаратне забезпечення, користувач, Gusto-DelightBot	Вибір розділу (меню, кошик, контакти)
Вибір розділу (меню, кошик, контакти)	Початок роботи з ботом	Асортимент, правила користування	Апаратне забежпечення, користувач, Gusto-DelightBot	Додавання продукції в кошик

Продовження таблиці 2.1 – Дані з діаграми декомпозиції IDEF0

Додавання продукції в кошик	Вибір розділу (меню, кошик, контакти)	Асортимент, правила користування	Користувач, Gusto-DelightBot	Перегляд списку замовлення, Вибір розділу (меню, кошик, контакти)
Перегляд списку замовлення	Додавання продукції в кошик	Асортимент, правила користування	Користувач, Gusto-DelightBot	Сформоване замовлення

Для деталізації процесів було виконано діаграму декомпозиції – рисунок 2.2

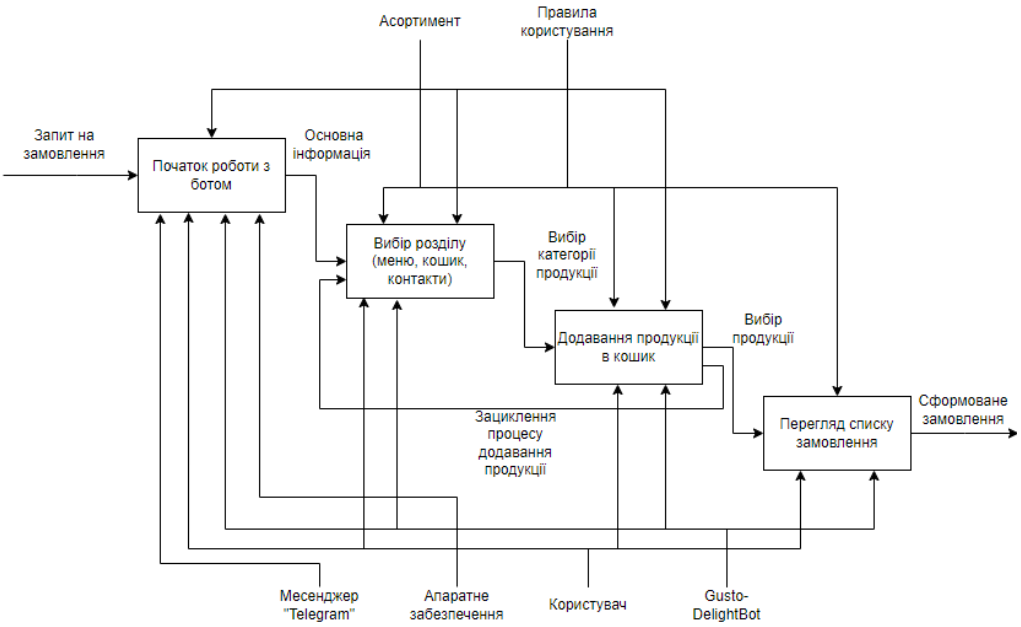


Рисунок 2.2 – Діаграма декомпозиції у нотації IDEF0

2.2 Моделювання варіантів використання

Діаграма варіантів використання (Use Case Diagram) — це тип діаграми який використовується для візуалізації функціональних вимог системи. Вона показує, як різні типи користувачів (актори) взаємодіють з системою, виконуючи різні дії (варіанти використання). Це допомагає зрозуміти, які функції система повинна виконувати та хто ними користується.

Діаграма поділяється на такі складові:

- Система - представляється прямокутником, всередині якого розташовані варіанти використання.
- Актор (Actor) - це зовнішня сутність, яка взаємодіє із системою. Це може бути користувач, інша система чи зовнішній пристрій. Актори зображуються у вигляді людських фігур або паличкових людей.
- Варіант використання (Use Case) — це конкретна функція або послуга, яку система надає актору. Варіанти використання зображуються у вигляді овальних фігур і розташовуються всередині системи.
- Взаємозв'язки - лінії, які з'єднують акторів із варіантами використання, показуючи, які дії можуть виконувати актори

На рисунку 2.3 зображено діаграму використання сервісу.

Рисунок 2.3 – Діаграма використання сервісу

2.3 Моделювання бази даних

Таблиця OrderCache містить кєш замовлень, тобто тут зберігаються сформовані замовлення деякий час.

Таблиця Button – містить розділи на кнопки.

Таблиця Product – містить назву, опис, ціну, вагу та картинку продукції.

Таблиця OrderItem – містить номер телефону, ім’я користувача, ціну, кількість продукції, загальну ціну замовлення, дату та перевірку на виконання замовлення.

Таблиця Order – це таблиця з основною інформацією про замовлення, в якій зберігається айді користувача, його номер, дата замовлення, замовлені позиції та перевірка на виконання.

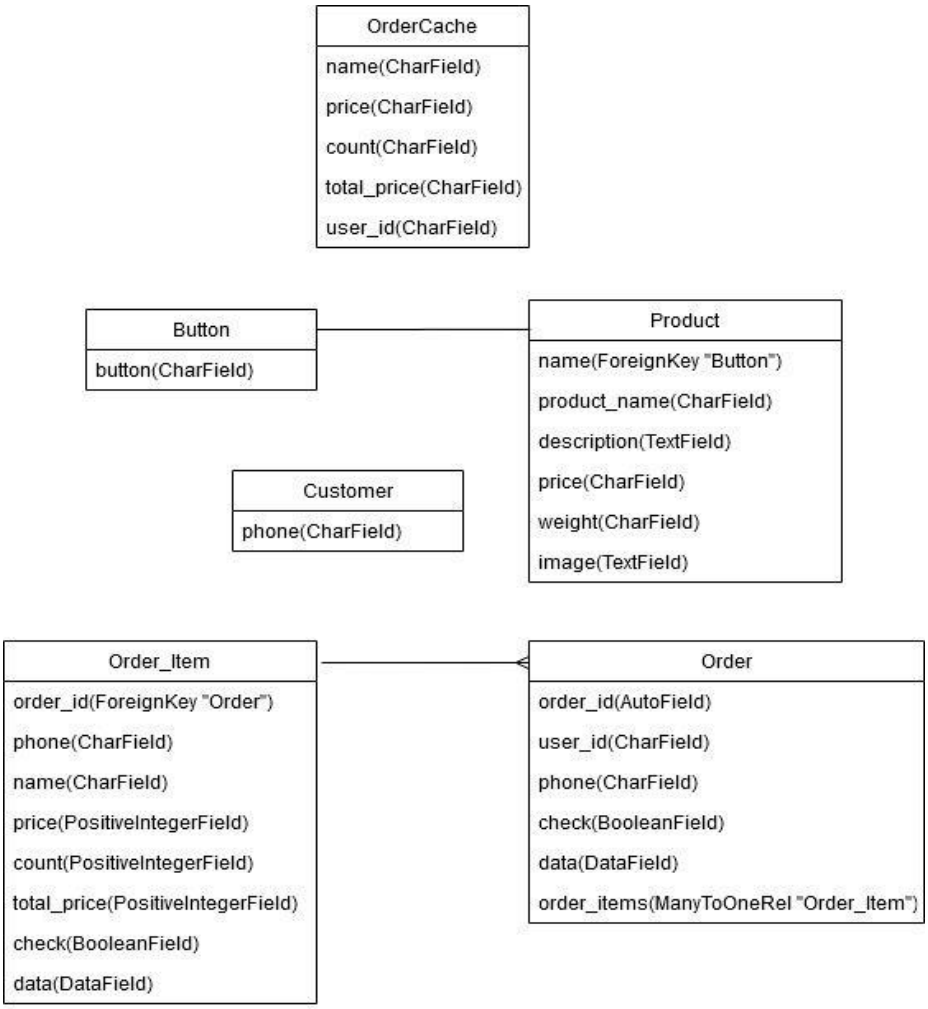


Рисунок 2.4 – Діаграма база даних

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Архітектура програмного продукту

Як ми можемо бачити, інтерфейс дуже простий та зрозумілий для користувача, що добре сприяє для подальшого користування сервісом.

На початку роботи користувач подає запит на початок роботи з ботом, бот відповідає йому привітанням та пропонує обрати дію, яка цікава користувачу, а саме – Переглянути меню, переглянути контактну інформацію, або переглянути кошик (після формування замовлення), якщо в кошик не було додано ніякої продукції, бот повідомить про це та запропонує звернутись до Меню. Після перегляду Меню, користувачу буде представлено розділи з продукцією, до яких входять: Напої, Супи, Салати, Піца, Роли та Ланчі. Після додавання продукції до кошику користувач зможе оформити замовлення, з’явиться повідомлення про те, що замовлення було сформоване, та з ним зв’яжуться за номером.

Чат-боти в Telegram працюють за допомогою Telegram Bot API, яка дозволяє створювати та керувати чат-ботами. Ці боти можуть взаємодіяти з користувачами через повідомлення, команди та інтерактивні кнопки. Для створення чат-боту треба звернутися для вже розробленого бота – Telegram BotFather. За допомогою цього бота можна створити наш телеграм-бот, задати йому назву, інформацію про нього, додати картинку та інші налаштування.

SQLite3 - це вбудована Система Управління Базами Даних, яка надає можливість зберігати та керувати базами даних у вигляді одного файлу бази даних без потреби встановлення окремого сервера.

На рисунку 3.1 продемонстровано процес роботи чат-боту.

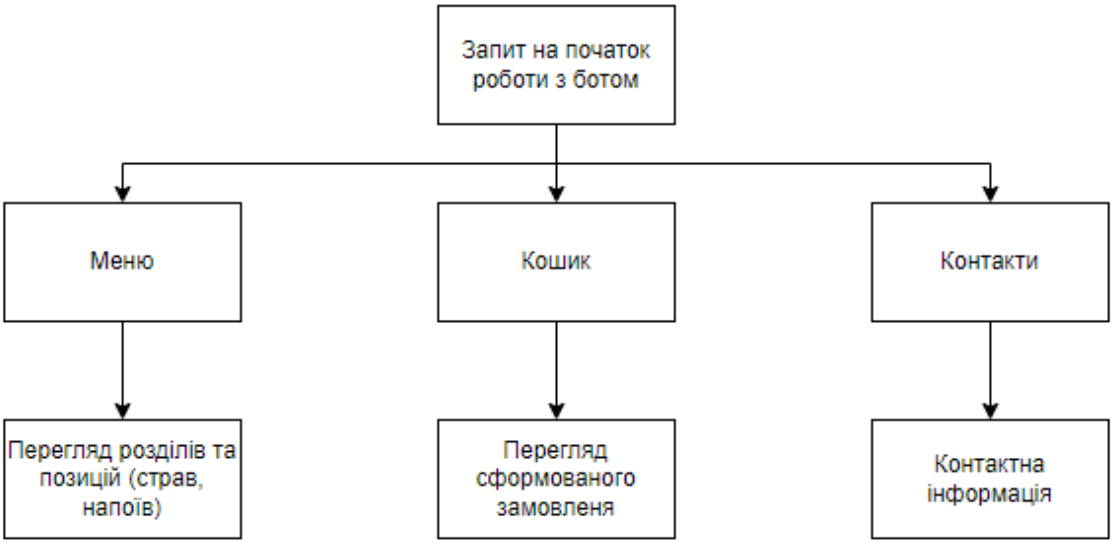


Рисунок 3.1 Архітектура чат-боту

3.2 Реалізація бази даних

Всі запити містяться в проекті, в файлі models.py. Для перегляду та редагування таблиць можна використати адміністративну панель. Вигляд таблиць зображено на рисунках нижче.

Всі створені таблиці відображено на рисунку 3.2

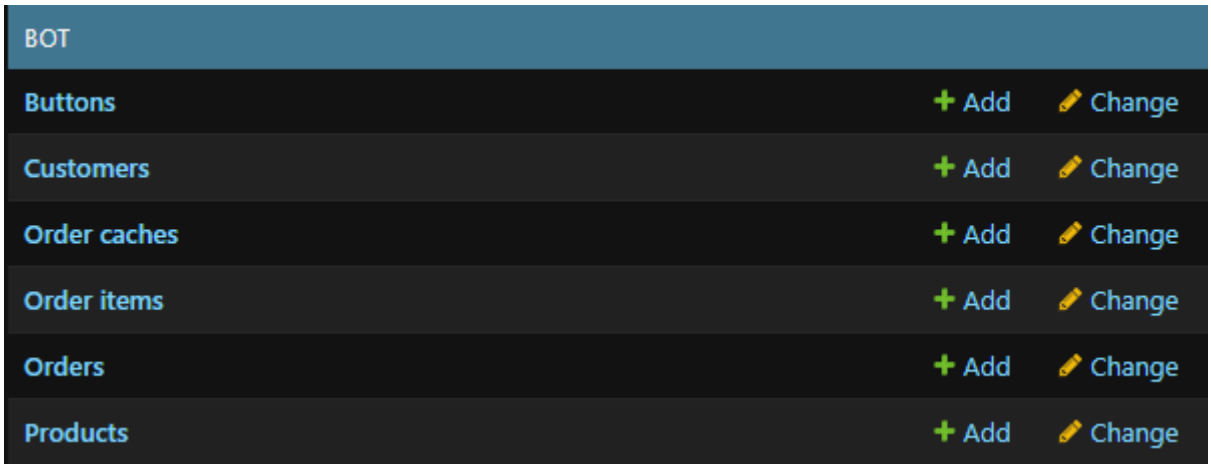


Рисунок 3.2 Створені таблиці бази даних

Розділи які містяться в таблиці Buttons зображено на рисунку 3.3

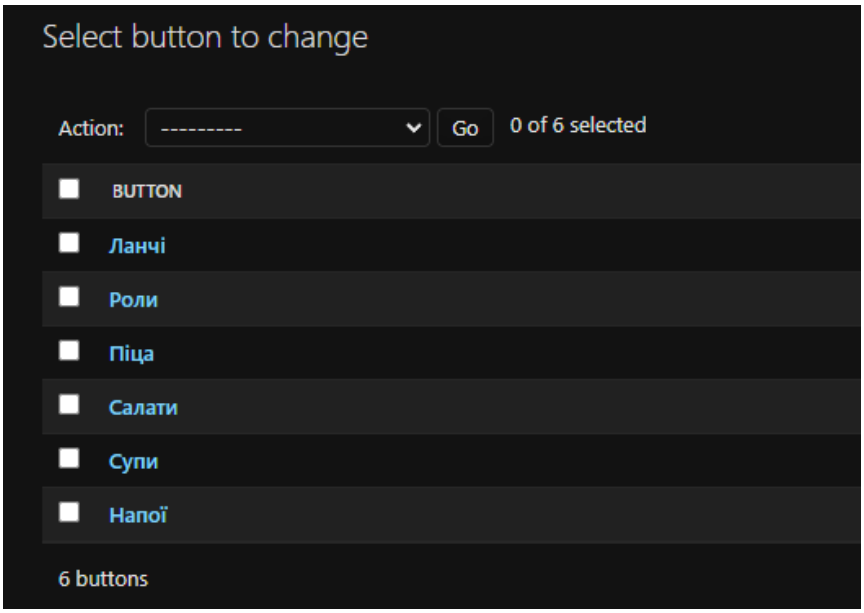


Рисунок 3.3 Дані таблиці Buttons

Дані з усією продукцією, яка міститься в таблиці Products відображено на рисунку 3.4

НАЗВА ПРОДУКТУ	ВАРТІСТЬ ПРОДУКТУ В ГРН
Картопляні діпи з беконом	125.0
Свинячий ошийок на грилі з картоплею	185.0
Чікен Фреш	155.0
Рол Сирний з лососем гриль	175.0
Рол Філадельфія Лайт	175.0
Рол Філадельфія з тунцем	239.0
Піца Царська	165.0
Піца Пепероні з томатами	165.0
Піца «4 Сири»	185.0
Салат з копченою куркою	125.0
Салат з морепродуктами	185.0
Салат Цезар із лососем	215.0
Том Ям з морепродуктами	169.0
Солянка м'ясна	119.0
Грибний крем суп з беконом	95.0
Fanta (0.75 л)	35.0
Sprite (0.75 л)	35.0
Coca-Cola zero (0,5 л)	25.0

Рисунок 3.4 Дані таблиці Buttons

Приклад відображення інформації про продукт зображено на рисунку 3.5

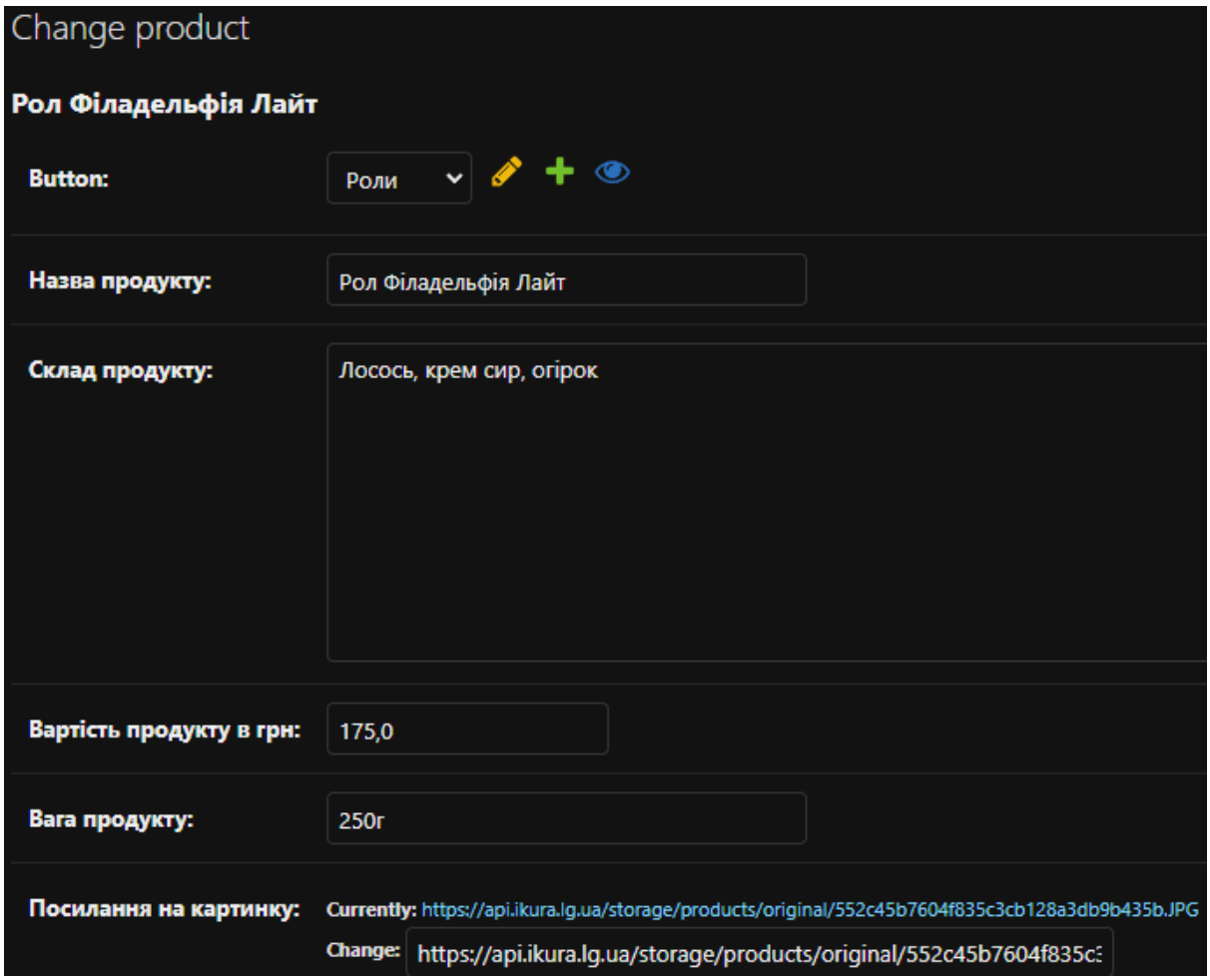


Рисунок 3.5 Дані про продукт

Дані які містяться в таблиці Order items зображено на рисунку 3.6

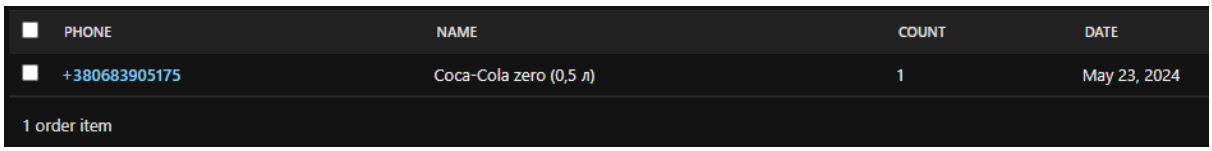


Рисунок 3.6 Дані про сформоване замовлення

А також інформація про замовника, яка зберігається в таблиці Orders зображена на рисунку 3.7

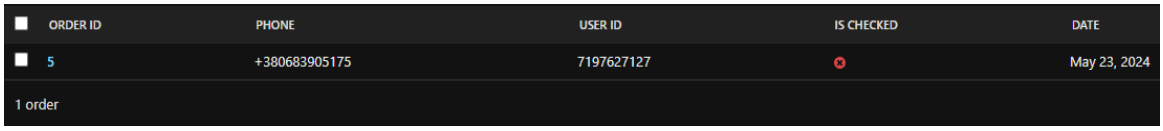


Рисунок 3.7 Дані про замовника

3.3 Програмна реалізація

Для того, щоб створити телеграм-бота потрібно скористатися спеціально створеним для цього ботом – BotFather.

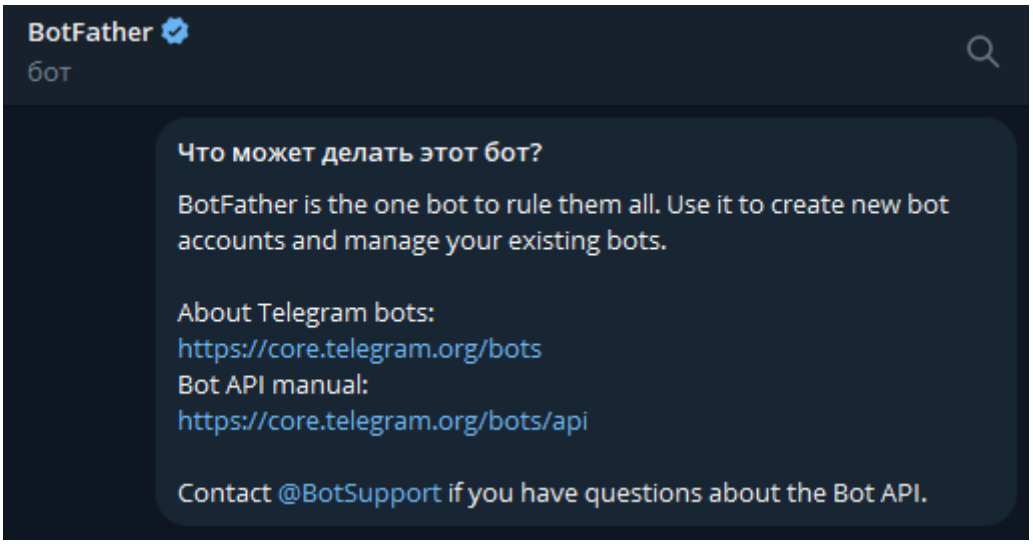


Рисунок 3.8 Опис BotFather

Після початку роботи з цим ботом він видає повідомлення з переліком команд для створення та керування ботом. Для створення використовується команда /newbot та задається ім’я нашого бота. Після цього задається id бота, а також можна вказати опис та загрузити картинку. Перелік команд зображено на рисунку 3.9.

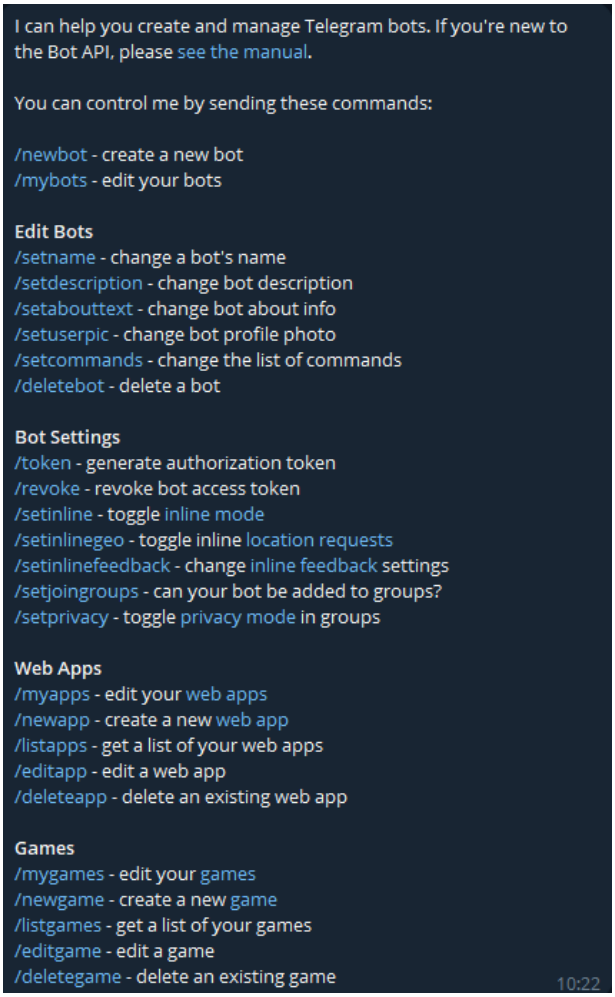


Рисунок 3.9 Перелік та опис команд бота FatherBot

Після створення бота, нам видається унікальний токен HTTP API нашого бота.

Перелік функцій, які використовувались для розробки:

- def start_def – функція для початку роботи з ботом;
- def hello_def – функція привітання з користувачем;
- def get_button_def – функція яка служить для показу розділів Меню, Кошик, Контакти;
- def get_button_def – функція для виводу меню;
- def query – функція для виводу позицій в меню;

- def add_to_order_menu_def – функція додавання продукції до кошику;
- def get_garbage_def – функція для перегляду позицій в кошику;
- def get_geophone_def – функція відправки номера телефону боту;
- def contact – функція формування замовлення;
- def get_garbage_def – функція для перегляду контактної інформації.

3.4 Використання програмного додатку

На початку роботи з ботом користувача зустрічає Стартова сторінка бота, яка зображена на рисунку 3.10.

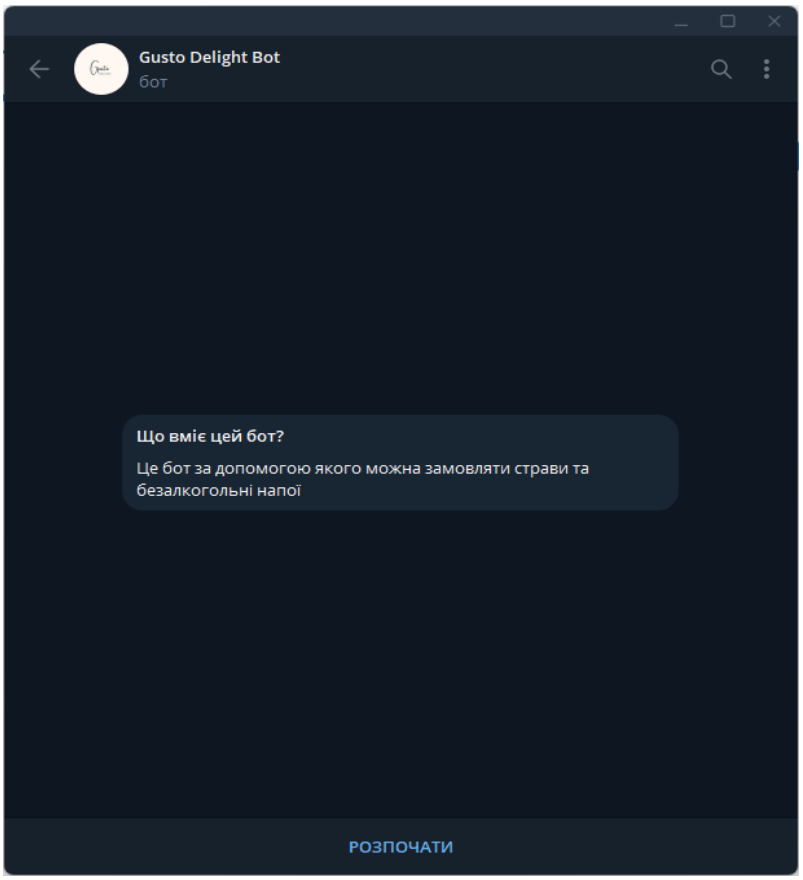


Рисунок 3.10 Стартова сторінка боту GustoDelightBot

Тут користувач може переглянути інформацію про бот, натиснувши зверху на панель “Gusto Delight Bot”. Інформація про бот зображена на рисунку 3.11.

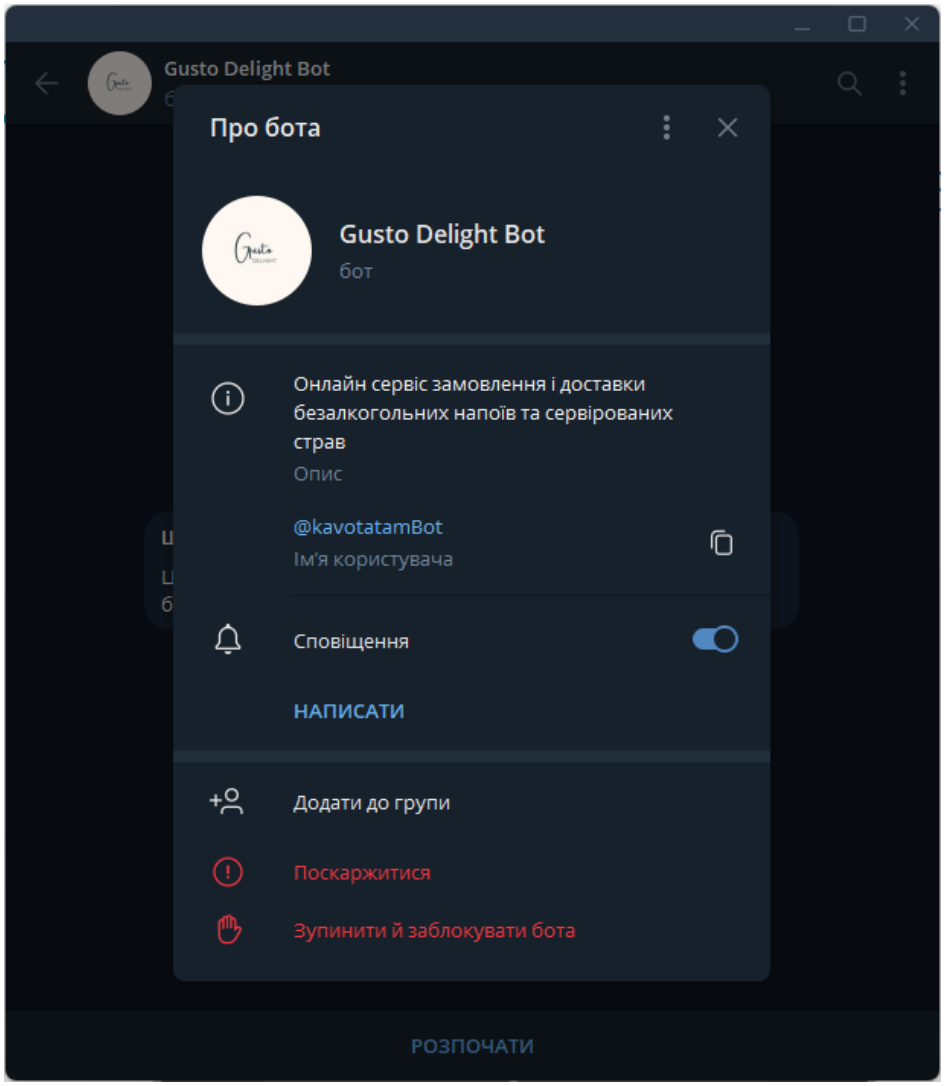


Рисунок 3.11 Інформація про бот GustoDelightBot

Далі, після початку роботи, бот виводить повідомлення-привітання та пропонує перейти до меню для вибору страв та напоїв. Процес зображено на рисунку 3.12

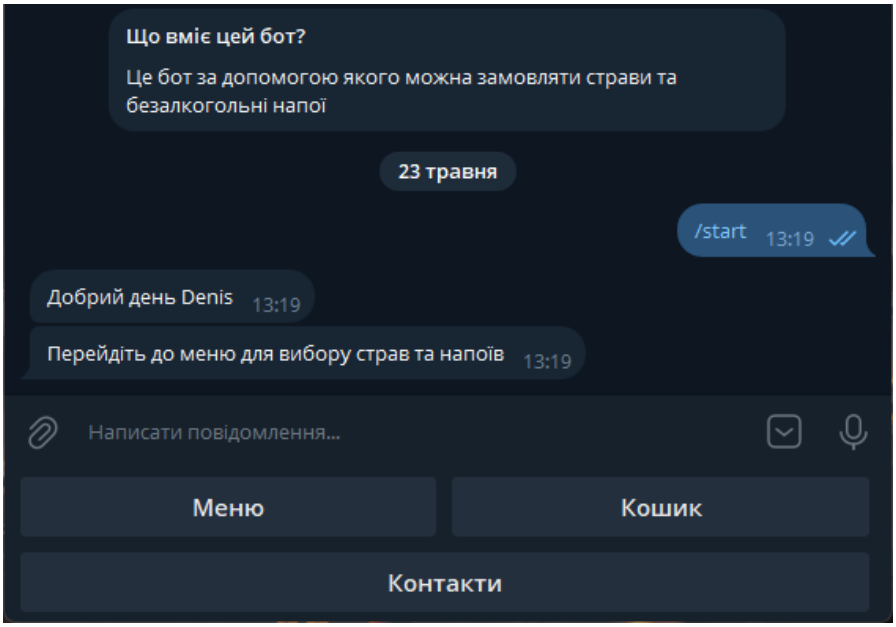


Рисунок 3.11 Початок роботи з ботом GustoDelightBot

Після натискання кнопки “Меню” з’являється перелік розділів меню. Процес зображено на рисунку 3.12.

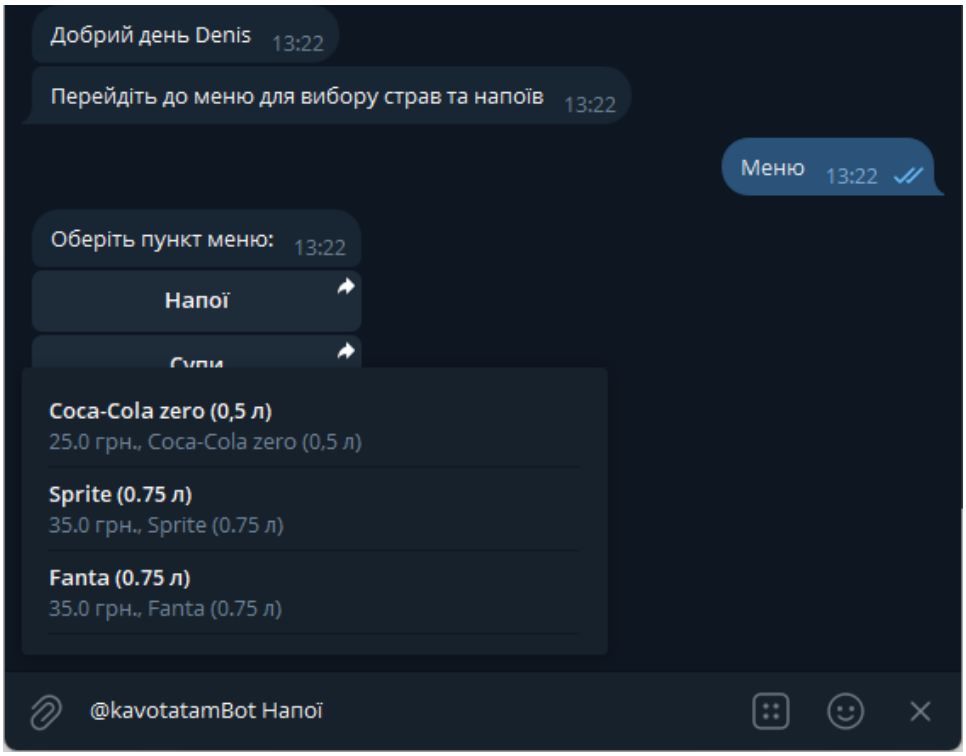


Рисунок 3.12 Меню боту GustoDelightBot

Тут користувач може обрати цікаву для нього продукцію, та на екрані з’явиться інформація про нього та кнопка “Додати до кошику”. Процес зображено на рисунку 3.13.

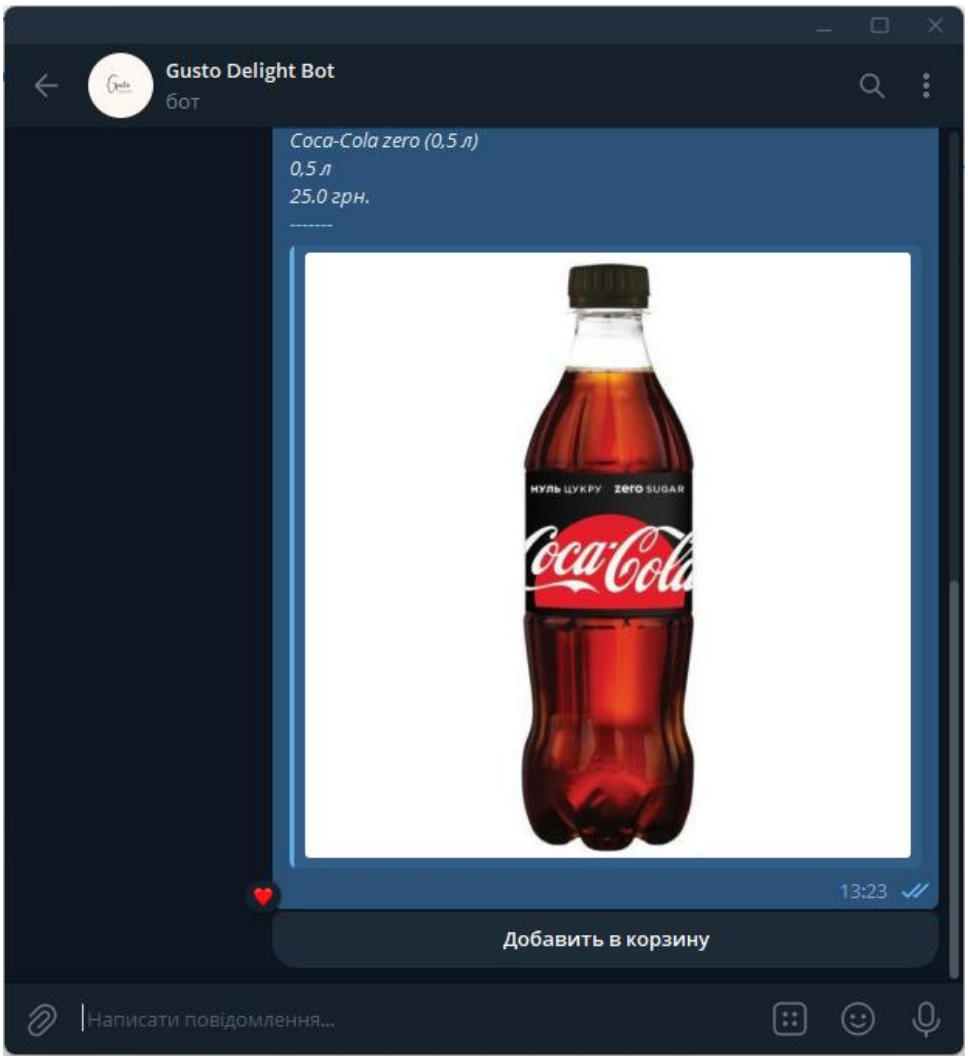


Рисунок 3.13 Інформація про позицію

Після успішного додавання позицій до кошику, користувач може переглянути інформацію про замовлене в Кошику. Процес зображено на рисунку 3.14.

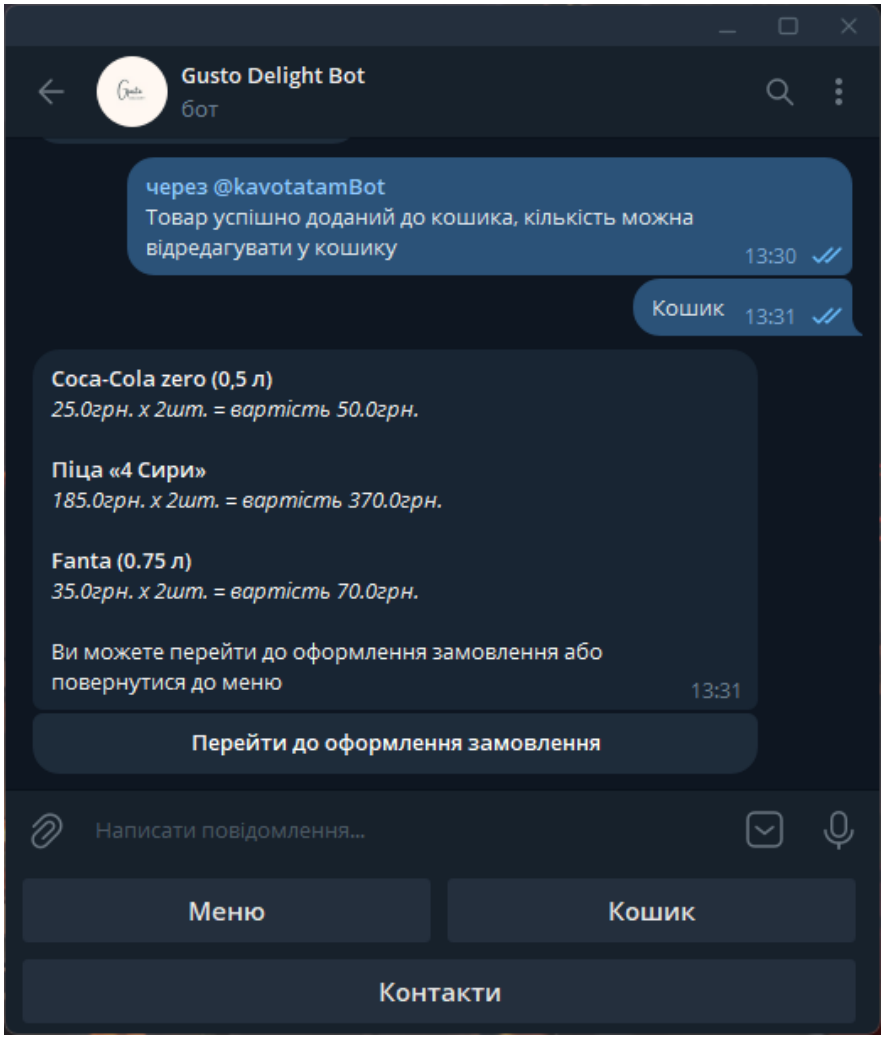


Рисунок 3.14 Інформація про замовлене

Після цього користувачу пропонується або повернутися до меню, або перейти до оформлення замовлення. Якщо користувач обирає “Перейти до оформлення замовлення”, то бот видає інформацію про загальну вартість замовлення та просить поділитися номером телефону. Процес зображено на рисунку 3.15.

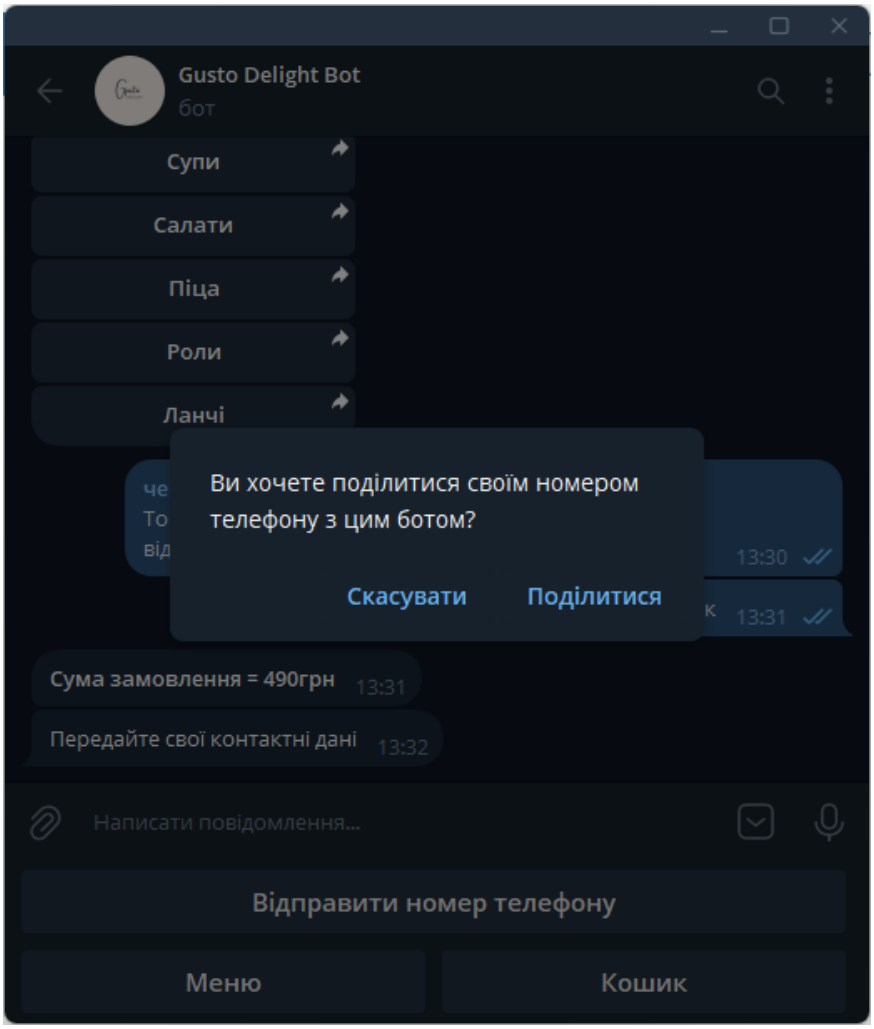


Рисунок 3.15 Запит бота на отримання номеру телефону

Після того як користувач надає номер телефону, бот сповіщає, що замовлення відправлено в обробку і треба дочекатися повідомлення за номером. Процес зображено на рисунку 3.16.

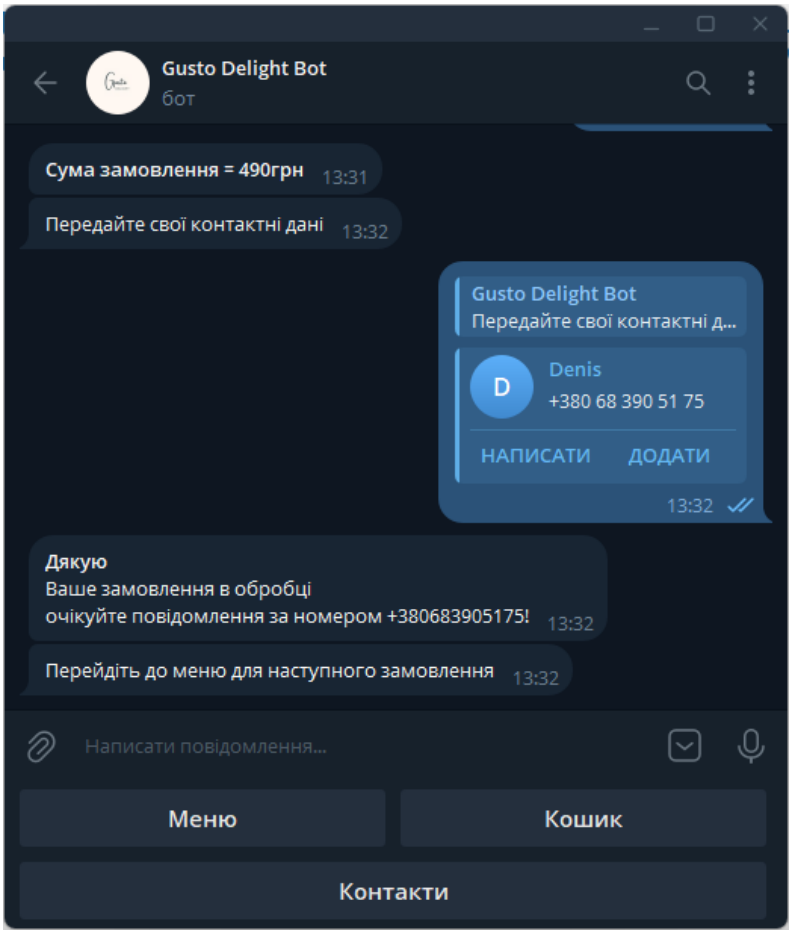


Рисунок 3.16 Повідомлення про обробку замовлення

На рисунках 3.17 – 3.18 буде зображений процес додавання нової позиції до меню, а також перегляд наявних товарів. В ході додавання позиції адміністратору потрібно обрати розділ, в який саме він хоче додати продукцію, задати назву продукту, склад продукту, вартість в грн, вагу та фото продукту.

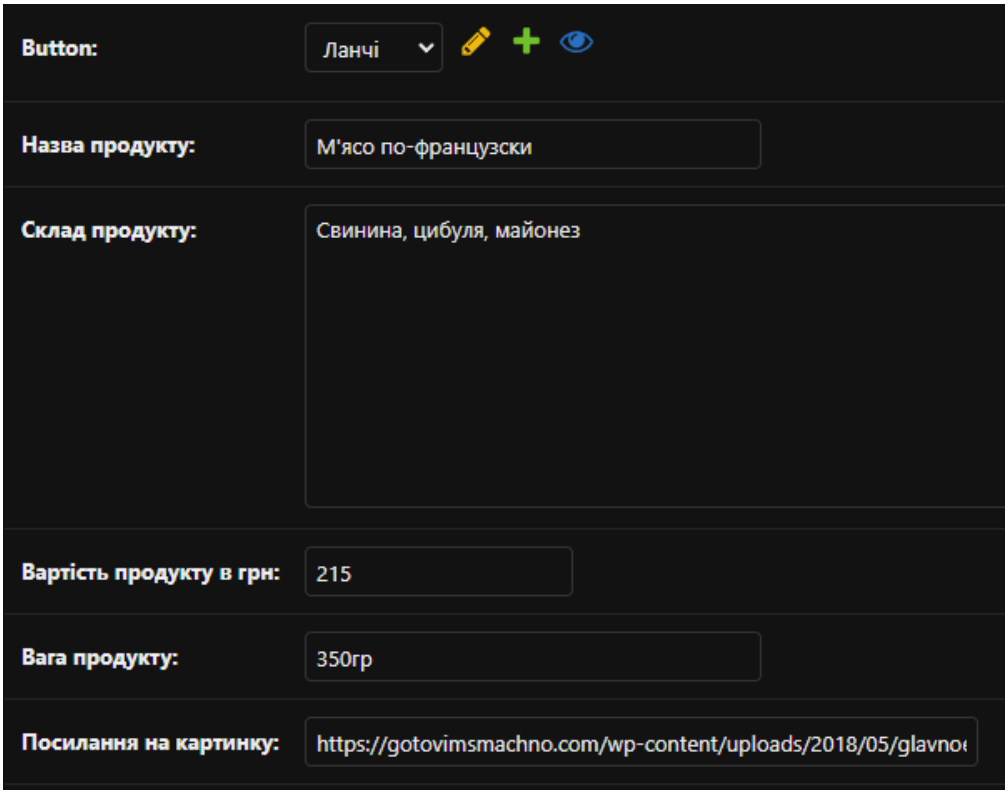
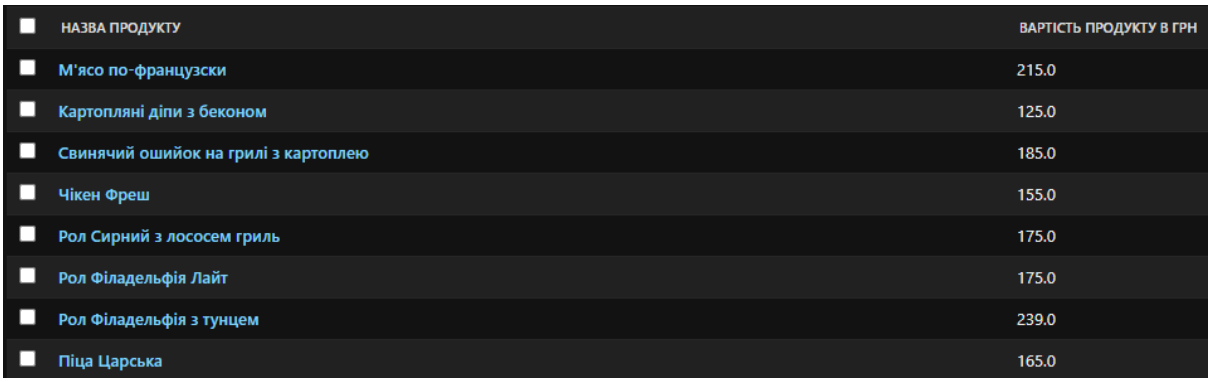


Рисунок 3.17 – Процес додавання продукту в розділ Ланчі



НАЗВА ПРОДУКТУ	ВАРТІСТЬ ПРОДУКТУ В ГРН
М'ясо по-французски	215.0
Картопляні діпи з беконом	125.0
Свинячий ошийок на грилі з картоплею	185.0
Чікен Фреш	155.0
Рол Сирний з лососем гриль	175.0
Рол Філадельфія Лайт	175.0
Рол Філадельфія з тунцем	239.0
Піца Царська	165.0

Рисунок 3.18 – Перегляд всієї продукції з меню

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра було розроблено онлайн сервіс замовлення і доставки безалкогольних напоїв і сервірованих страв на основі Telegram-бота

Під час реалізації роботи було виконане наступне:

- проведено аналіз сучасного ринку сервісів для замовлення їжі;
- визначено функціонал боту та взаємодії з користувачем;
- розроблено Telegram-бота та його функціонал;
- розроблено базу даних та веб-інтерфейс для адміністраторів;
- проведено функціональне тестування та запуск сервісу

Інтеграція сервісу замовлення і доставки безалкогольних напоїв та сервірованих страв через Telegram-бот є стратегічним кроком, який може суттєво покращити як користувацький досвід, так і бізнес-процеси. Цей підхід поєднує сучасні технології з практичними потребами споживачів, відкриваючи нові можливості для росту та розвитку.

Такий сервіс має на меті бути легким в доступі, адже мобільні пристрої завжди під рукою, боти можуть пропонувати простий і зрозумілий інтерфейс для вибору напоїв і страв, що робить процес замовлення швидким і зручним.

Також це позитивно впливає і на бізнес, адже Telegram популярний у багатьох країнах, що дозволяє залучити велику аудиторію користувачів, використання бота для реклами нових продуктів, акцій і подій може бути ефективним інструментом маркетингу. Це знижує потребу в ручному обробленні замовлень, що скорочує витрати на персонал.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. McKinsey & Company - "How COVID-19 has pushed companies over the technology tipping point—and transformed business forever"
   URL: https://www.mckinsey.com/capabilities/strategy-and-corporate-finance/our-insights/how-covid-19-has-pushed-companies-over-the-technology-tipping-point-and-transformed-business-forever
2. Juniper Research - "Chatbots in Online Food Ordering"
   URL: https://www.juniperresearch.com/research/telecoms-connectivity/messaging/chatbots-trends-research-report/
3. Deloitte - "The Restaurant of the Future: A Vision Evolves"
   URL: https://www2.deloitte.com/content/dam/Deloitte/us/Documents/consumer-business/us-cb-restaurant-of-the-future.pdf
4. Gartner - "The Importance of Database Management Systems in Online Retail"
   URL: https://www.gartner.com/en/documents/4594399
5. IBM – “The Impact of User-Friendly Interfaces on Restaurant Management”
   URL: https://www.researchgate.net/publication/220146876_The_impact_of_interface_usability_on_trust_in_Web_retailers
6. Стратегії та інструменти побудови інтернет-бізнесу.
   URL: https://sprava.ua/blog/z-chogo-skladaetysya-internet-marketing
7. Використання Telegram-ботів у бізнесі: переваги та можливості
   URL: https://www.promodo.ua/blog/yak-onlayn-magazinu-prodavati-cherez-telegram
8. Європейський досвід розвитку онлайн-сервісів замовлення та доставки їжі: аналіз ринку та тенденції // European Food Delivery Market Report 2024

URL: https://www.statista.com/outlook/emo/online-food-delivery/europe#users

9. Особливості розвитку онлайн-сервісів замовлення їжі в Україні // Сучасні проблеми торгівлі та послуг.
URL: https://hub.kyivstar.ua/articles/stan-rinku-fud-dostavki-v-ukrayini-yak-zrostaye-popit-na-dostavku-yizhi-ta-napoyiv
10. Роль Telegram-ботів у підтримці малого бізнесу // Мале підприємництво: економіка, управління, інновації.
URL: https://conf.ztu.edu.ua/wp-content/uploads/2021/11/118.pdf
11. Python [Електронний ресурс]
URL: https://www.python.org/
12. The official site of the Django Software Foundation [Електронний ресурс]
URL: www.djangoproject.com/foundation
13. E-commerce Trends - "Top E-commerce Trends for 2023"
URL: https://www.shopify.com/blog/ecommerce-trends
14. Food Delivery Services - "The Growth of Food Delivery Services Post-Pandemic"
URL: https://www.statista.com/statistics/1190703/firewood-production-volume-brazil/
15. Online AI Chatbots Transforming Customer Service – LeadSQL
URL: https://leadsql.ai/blog/online-ai-chatbots-transforming-customer-service-in-the-digital-age/
16. 10 Stats That Show How COVID-19 Impacted Food Delivery Services
URL: https://www.routific.com/10-stats-that-show-how-covid-19-impacted-food-delivery-services
17. How automation emprowers small business
URL: https://www.linkedin.com/pulse/how-automation-empowers-small-businesses-agile-automations-03nqe/?trk=public_post

18. The Power of Social Media for E-commerce: Leveraging Platforms for Business Growth
URL: https://www.linkedin.com/pulse/power-social-media-e-commerce-leveraging-platforms-business-eba4c/
19. A Guide to Digital Customer Engagement: Strategies, Examples & Trends
URL: https://vwo.com/customer-engagement/digital-customer-engagement/
20. Online Food Delivery – Ukraine
URL: https://www.statista.com/outlook/emo/online-food-delivery/ukraine

ДОДАТОК А. Технічне завдання

ТЕХНІЧНЕ ЗАВДАННЯ

Онлайн сервіс замовлення і доставки безалкогольних напоїв і сервірованих страв на основі Telegram-бота

1 Призначення й мета створення чат-боту

1.1 Призначення чат-боту

До призначень чат-боту можна віднести доступність для користувачів у будь-який час доби, оперативність обробки запитів, безперервний доступ до боту, зручність використання, можливість дистанційного замовлення їжі, адже бот працює і з телефону.

1.2 Мета створення

Мета цього проекту – є розробка сервісу з продажу страв у ресторані та безалкогольних напоїв, з можливістю ознайомлення з товаром, його ціною. Розроблюваний сервіс має бути простим, для охоплення більшої аудиторії.

1.3 Цільова аудиторія

Чіткої цільової аудиторії немає, адже бот може використовувати будь-яка людина, яка зацікавлена в дистанційному замовленні їжі та напоїв.

2 Вимоги до чат-боту

2.1 Вимоги в цілому

2.1.1 Вимоги до стуктури та функціоналу чат-боту

Після аналізу сучасного ринку сервісів та вимог користувачів, було сформовані наступні вимоги: інтерфейс боту повинен бути інтуїтивно

зрозумілим та простим в користуванні, текст повинен бути читабельним, функціонал боту конкретним та простим для клієнтів.

Кінцевим продуктом є чат-бот для Telegram-додатку по замовленню сервірованих страв та напоїв.

2.1.2 Вимоги до персоналу

Додаткових та особливих вимог до користувачів та адміністраторів не передбачено, адже користування ботом є легким та зрозумілим. Від адміністратора потребується знання користування адміністративною панеллю, але там також все зрозуміло, що дає змогу дуже швидко знайти людину на це місце роботи.

2.1.3 Вимоги до збереження інформації

Уся інформація зберігається в базі даних, яка підключена до телеграм-боту. Базою даних було обрано SQLite3, через її простоту використання, швидкість та автономність.

2.1.4 Вимоги до розмежування доступу

Чат-бот доступний для звичайних користувачів за пошуком, або за посиланням на нього.

Для адміністратора же створений окремий веб-інтерфейс, за допомогою якого можна додавати, видаляти, змінювати інформацію про страви, напої та розділи а також переглядати інформацію про сформовані замовлення.

2.2 Структура чат-боту

2.2.1 Загальна інформація про структуру додатку

Зверху чату, можна переглянути інформацію про бот, для цього потрібно клікнути на профіль боту, після чого з’явиться вікно з описом. Після запуску роботи на екрані з’являються кнопки знизу чату, за допомогою яких можна користуватися сервісом. В нижній частині чату присутня сама кнопка “Старт”, після чого з’являються основні кнопки.

2.2.2 Навігація

Навігація в боті досить легка та практична. Користувач взаємодіє з ботом за допомогою кнопок та перегляду розділів, інформація з’являється послідовно, що дає змогу легко користуватися функціоналом.

2.2.3 Контент

Здебільшого весь контент в телеграм-боті представлений у вигляді тексту та кнопок, але також присутні і зображення страв та напоїв.

2.2.4 Дизайн та структура бота

Дизайн телеграм-боту обмежується додатком “Telegram”, розробник має змогу лише на зручне розташування кнопок та їх назву. Усі інші види дизайну вже залежать від самого користувача, який налаштовує всі чати в додатку. Весь дизайн сервісу зображено на прикладах роботи в розділі 3.4 Використання програмного продукту.

2.2.5 Система навігації

Діаграма системи навігації чат-боту зображена на рисунку А.1.

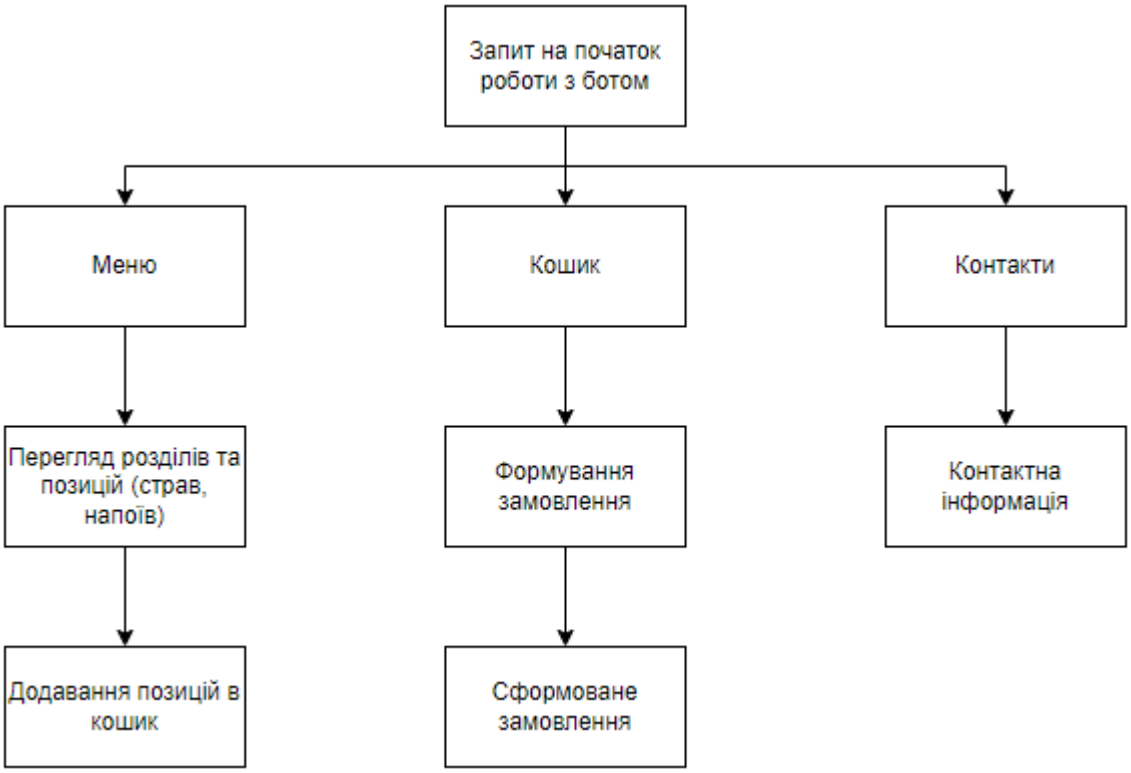


Рисунок А.1. Навігація чат-боту

2.3 Вимоги до функціонування системи

2.3.1 Потреби користувача

Усі потреби користувача зазначено в таблиці А.1.

Таблиця А.1. – Потреби користувача

Потреби користувача	Джерело

Продовження таблиці А.1 – Потреби користувача

Перегляд розділів меню	Користувач
Перегляд позицій в меню	Користувач
Формування замовлення	Користувач
Перегляд сформованого замовлення	Користувач, адміністратор
Перегляд контактної інформації	Користувач

2.3.2 Функціональні вимоги

Функціональні вимоги були сформовані на основі потреб користувача, а саме:

- авторизація;
- перегляд меню ресторану;
- навігація по категоріям;
- додавання страв до кошику;
- можливість коригувати замовлення в кошику;
- оформлення замовлення;
- зворотній зв’язок;
- адміністративна панель

2.3.3 Системні вимоги

Таблиця А.2 – Системні вимоги та їх опис

Системні вимоги	Опис
База даних з розділами	Надає можливість відображати, змінювати, додавати та видаляти інформацію про розділи

Продовження таблиці А.2 – Системні вимоги та їх опис

База даних з інформацією про користувача	Надає можливість переглянути номер телефону користувача, який зробив замовлення
База даних з кешем	Надає можливість переглянути інформацію про замовлення, яка зберігається деякий час
База даних з інформацією про замовлені страви та напої	Надає можливість для перегляду замовленої продукції
База даних з замовленням	Надає можливість для перегляду інформації про замовлення (позиції, кількість)
База даних з продукцією	Містить в собі перелік всієї продукції з меню

2.4 Вимоги до видів забезпечення

2.4.1 Вимоги до інформаційного забезпечення

Для реалізації телеграм-бота було використано:

- Мова програмування Python;
- PyCharm;
- SQLite3;
- Telegram;

- Django

2.4.2 Вимоги до лінгвістичного забезпечення

Вся інформація яка міститься в базі даних та використовується в боті була написано українською.

2.4.3 Вимоги до програмного забезпечення

До вимог програмного забезпечення відноситься:

- Стабільний інтернет
- Додаток “Telegram”
- Операційна система апарату, на якій використовується додаток (Android, IOS, Windows, Linux та інші)

3 Склад і зміст робіт зі створення боту

Опис етапів створення бота наведено в таблиці А.3.

Таблиця А.3 – Етапи розробки

№	Склад і зміст робіт	Строк розробки (у робочих днях)
1	Підготовка специфікації	4 дні
2	Визначення проблем, які вирішать використання бота	4 дні
3	Визначення цілей і задач	3 дні
4	Визначення властивостей бота	4 дні
5	Розробка технічного завдання	6 днів
6	Редагування та обговорення технічного завдання	4 дні
7	Затвердження технічного завдання	3 дні

Продовження таблиці А.3 – Етапи розробки

8	Процес написання	12 днів
9	Розробка меню бота	7 днів
10	Створення та наповнення бази даних	2 дні
11	Перевірка додатку і виправлення помилок	1 день
12	Фінальна перевірка додатку і оптимізація	6 днів
13	Впровадження в дію	4 дні
14	Підготовка до релізу і запуск додатку	4 дні
15	Загальна тривалість робіт	64 дні

4 Вимоги до складу й змісту робіт із введення в експлуатацію

Для початку роботи з сервісом потрібно знайти бот в Telegram, це можна зробити за пошуком по назві в телеграм – GustoDelightBot, або перейти за посиланням https://t.me/kavotatamBot. Для запуску використовується команда “/start”. Для запуску телеграм-бота потрібно запустити проект.

ДОДАТОК Б. Планування робіт

Метою роботи є розробка та впровадження онлайн-сервісу замовлення і доставки безалкогольних напоїв і сервірованих страв на основі Telegram-бота. Для досягнення мети проєкту необхідно виконати наступні задачі:

– Провести аналіз ринку та вимог користувачів

– Визначити функціонал бота та взаємодію з користувачем;

– Розробити Telegram-бота та його функціонал;

– Розробити базу даних та веб-інтерфейсу для адміністраторів;

– Функціональне тестування та запуск сервісу.

Деталізація мети проекту методом SMART

Метою даної роботи є створення зручного та ефективного способу замовлення і доставки безалкогольних напоїв та сервірованих страв через Telegram-бот.

Отже, можемо сформулювати мету нашого проекту за цими п’ятьма факторами. Результати наведені у таблиці Б.1.

Таблиця Б.1 – Деталізація мети проекту методом SMART

Specific	Розробка та впровадження онлайн-сервісу замовлення і доставки безалкогольних напоїв і сервірованих страв на основі Telegram-бота
Measurable	Збільшення кількості замовлень на 20% протягом першого місяця після впровадження сервісу
Achievable	Наявне затверджене технічне завдання від замовника, команда програмістів і тестувальників, безкоштовне ПЗ для розробки PyCharm, адміністратор, бібліотеки для роботи з Telegram-ботами

Продовження таблиці Б.1 – Деталізація мети проекту методом SMART

Relevant	Підвищення рівня обслуговування клієнтів, зручність використання сервісу, зростання лояльності користувачів
Time-bound	Наявний конкретний термін – до кінця 4 курсу (06 червня 2024 р.)

WBS – це перше завдання, яке потрібно виконати на етапі планування проекту. WBS – це графічне подання згрупованих елементів проекту у вигляді пакетів робіт, які ієрархічно пов’язані з продуктом проекту. Вона необхідна для забезпечення ефективного управління проектом, визначення та структурування переліку робіт, створення структури звітності, а також для розуміння задач виконавцями. WBS є базовим засобом для створення організаційної структури (OBS) і системи управління проектом, оскільки дозволяє виявити проблеми організації робіт, визначити ієрархію проектних завдань (етапів робіт), підзавдань і пакетів робіт на всіх подальших фазах життєвого циклу проекту.

Діаграма WBS зазначена на рисунку Б.1.

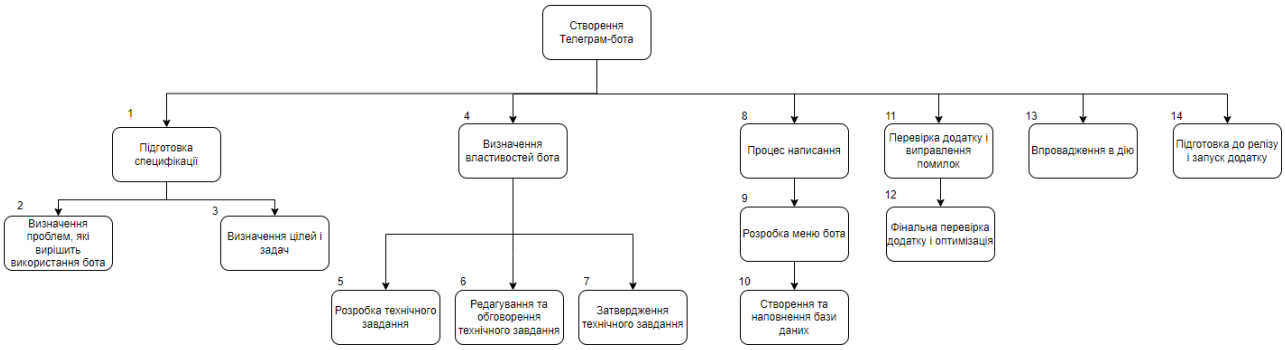


Рисунок Б.1. Діаграма WBS

OBS – це друге завдання, яке необхідно виконати на етапі планування проекту. Вона являє собою графічне відображення учасників проекту (фізичних та юридичних осіб) та їхніх відповідальних осіб, залучених до реалізації проекту. Елементами OBS можуть бути: окремі виконавці (керівники, фахівці, службовці); організації, структурні підрозділи і служби, в яких зайнята певна кількість фахівців, що виконують конкретні функціональні обов'язки; зовнішні постачальники обладнання та послуг.

На рисунку Б.2. представлено діаграму OBS.

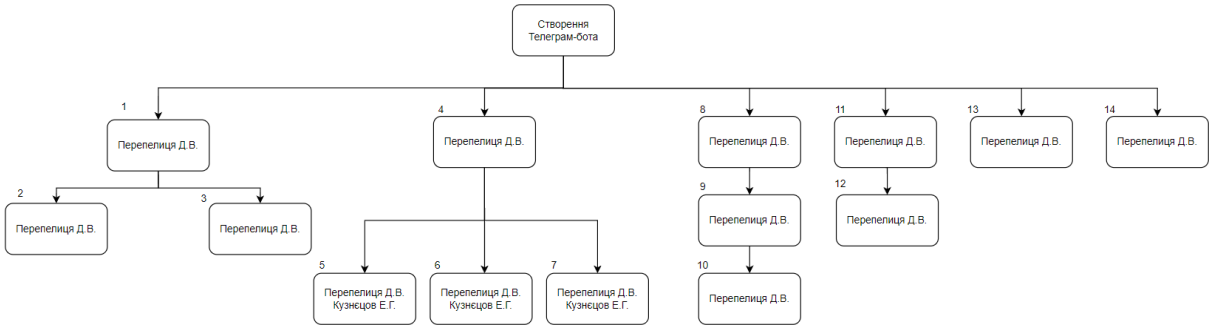


Рисунок Б.2 – Діаграма OBS

Таблиця Б.2 – Виконавці проєкту

Роль	ПІБ	Проєктна роль
Розробник	Перепелиця Д.В.	Виконує front-end та back-end розробку
Проєктувальник	Перепелиця Д.В.	Виконує проєктування бази даних та розробляє структуру онлайн-сервісу
Тестувальник	Перепелиця Д.В.	Відповідає за тестування функціоналу та дизайну сервісу
Керівник проєкту	Кузнєцов Е.Г.	Формує завдання на розробку проєкту

Діаграма Ганта — це інструмент управління проектами, що використовується для планування і контролю виконання завдань у часі. Вона являє собою графік, на якому відображено завдання проекту у вигляді горизонтальних смуг вздовж осі часу. Діаграма Ганта дозволяє візуалізувати старт, тривалість та завершення кожного завдання, а також їх взаємозв'язки.

Діаграма ганта представлено на рисунку Б.3.

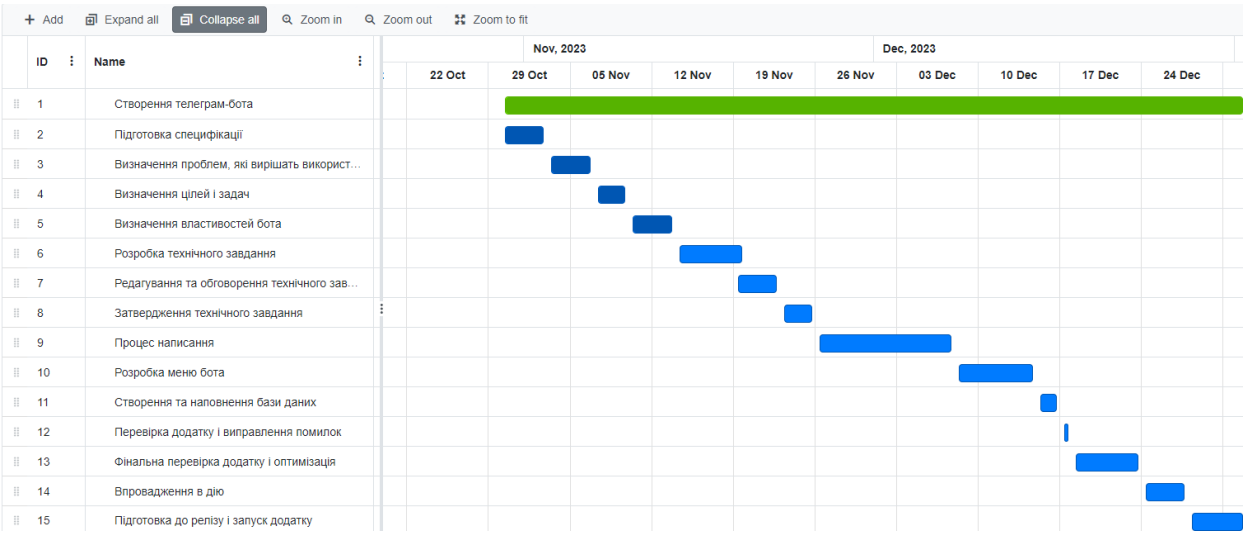


Рисунок Б.3. Діаграма Ганта

Управління ризиками проекту

Управління ризиками проекту — це процес ідентифікації, аналізу та реагування на ризики, що можуть вплинути на проект. Метою управління ризиками є мінімізація негативного впливу ризиків і використання можливостей для покращення результатів проекту.

Ризик проекту — це будь-яка подія або умова, яка, якщо відбудеться, може негативно або позитивно вплинути на досягнення цілей проекту. Ризики можуть бути внутрішніми (пов'язаними з самим проектом або організацією) або

зовнішніми (пов'язаними з оточенням проекту, такими як ринкові умови, законодавчі зміни тощо).

Оцінюваня виконується за показниками, що описані в таблиці Б.3.

Таблиця Б.3 – шкала оцінювання ризиків за ймовірністю виникнення та величиною впливу

Оцінка	Ймовірність виникнення	Вплив ризику	Тип ризику	Межі
1	Низька	Низький	Прийнятні	1≤R≤2
2	Середня	Середній	Виправдані	3≤R≤4
3	Висока	Високий	Недопустимі	6≤R≤9

Було виконано оцінку проекту та визначені ймовірності, вприв та тип кожного ризику. Отримані результати записані в таблицю Б.4.

Таблиця Б.4 – Назва, ймовірність, вплив та ранг ризику

№ ризику	Назва ризику	Ймовірність (0,1 – 0,9)	Вплив (0,05 – 0,8)	Ранг
1	Відключення світла	0,9	0,8	0,7
2	Загроза обстрілів	0,6	0,8	0,42
3	Хвороба персоналу	0,4	0,4	0,13
4	Низька кваліфікація розробників	0,1	0,2	0,02

Продовження таблиці Б.4 – Назва, ймовірність, вплив та ранг ризику

5	Неоптимальний розподіл часу	0,5	0,5	0,25
6	Часте внесення змін у ТЗ	0,2	0,1	0,05

За зелений колір в таблиці відповідають прийнятні ризики, за жовтий – виправдані, за червоні – недопустимі.

Таблиця Б.5 – Матриця ймовірності ризику та впливу.

Ймовірність ризику	Вплив загрози (ризику)				
	Дуже малий	Малий	Середній	Великий	Дуже великий
	0,05	0,1	0,2	0,4	0,8
0,9					
0,7					
0,5				R2(0,15)	R6(0,20)
0,3					R7(0,10)
0,1			R1(0,02), R3(0,02)		R4(0,05), R5(0,10)

Класифікація ризиків за рівнем, відповідно до отриманого значення індексу, представлена у таблиці Б.6.

Таблиця Б.6 – Шкала оцінювання ризику за рівнем

№	Назва	Межі	Ризики, які входять(номера)
1	Прийнятні	0,005<R<0,05	4, 6
2	Виправдані	0,05<R<0,14	5, 3

Продовження таблиці Б.6 – Шкала оцінювання ризику за рівнем

3	Недопустимі	0,14<R≤0,72	1, 2

У таблиці Б.7 описано ризики та стратегії реагування на кожен з них

Таблиця Б.7 – Ризики та стратегії реагування

ID	Статус ризику	Опис ризику	Ймовірність	Вплив	Ранг	План А	Тип стратегії реагування	План Б
RS_1	Open	Відключення світла	0,9	0,8	0,7	Мати аккумулятори або паверабанки	Ухилення	Адаптувати графік роботи під відключення світла
RS_2	Open	Загроза обстрілів	0,6	0,8	0,42	Працювати у безпечному приміщенні	Зменшення	Працювати подалі від територій, на який ведуться бойові дії
RS_3	Open	Хвороба персоналу	0,4	0,4	0,13	Зменшити навантаження	Зменшення	Мати декілька робітників, які можуть підміняти один одного

Продовження таблиці Б.7 – Ризики та стратегії реагування

RS_4	Open	Низька кваліфікація розробників	0,1	0,2	0,02	Набирати людей з необхідними знаннями та навичками	Попередження	Мати кваліфікаційну людину, яка буде вчити і допомагати співробітникам
RS_5	Open	Неоптимальний розподіл часу	0,5	0,5	0,25	Спланувати роботу згідно дедлайну	Зменшення	Задіяти більшу кількість робітників
RS_6	Open	Часте внесення змін у ТЗ	0,2	0,1	0,05	Поставити нову мету та ціль проекту	Ухилення	Зміна ТЗ

ДОДАТОК В. Лістинг коду

Код Telegram-боту

```
import telebot
import requests
import json
import time
import urllib
import re
from telebot import types

import logging
logging.basicConfig(level=logging.INFO)
logger = telebot.logger
telebot.logger.setLevel(logging.DEBUG)

bot = telebot.TeleBot(' ')
ip = ['localhost:8000', '127.0.0.1:8000']


@bot.message_handler(commands=['start'])
def start_def(message):
    hello_def(message)

def hello_def(message):
    text = "Добрий день {}".format(message.from_user.first_name)
    bot.send_message(message.chat.id, text)
    get_button_def(message)

def get_button_def(message):
    key = types.ReplyKeyboardMarkup(True, False)
    key.row("Меню", "Кошик")
    key.row("Контакти")
    bot.send_message(message.chat.id, "Перейдіть до меню для вибору
страв та напоїв", reply_markup=key)

def get_again_button_def(message):
    key = types.ReplyKeyboardMarkup(True, False)
    key.row("Меню", "Кошик")
    key.row("Контакти")
    bot.send_message(message.chat.id, "Перейдіть до меню для
наступного замовлення", reply_markup=key)
```

```
@bot.message_handler(func=lambda message: message.text == "Меню")
def get_menu_def(message):
    try:
        link = requests.get(f'http://{ip}/api/button')
        data = json.loads(link.text)
        in_key = types.InlineKeyboardMarkup()
        for item in data['list']:
            call_name = item['name']
            button = types.InlineKeyboardButton(text=call_name,
switch_inline_query_current_chat=call_name)
            in_key.add(button)
        bot.send_message(message.chat.id, 'Оберіть пункт меню:',
reply_markup=in_key)
    except Exception as e:
        print(e)

@bot.inline_handler(func=lambda query: True)
def query(query):
    try:
        link = f'http://{ip}/api/products'
        text = {'text': query.query}
        req = requests.post(link, data=json.dumps(text))
        data = json.loads(req.text)
        caller = []
        for item in data['list']:
            call_id = item['id']
            call_title = item['name']
            call_desc = item['desc']
            price = item['price']
            weight = item['weight']
            image_url = item['image']
            in_key = types.InlineKeyboardMarkup()
            add_button = types.InlineKeyboardButton(text="Додати в
кошик", callback_data=f'add_product_{call_title}')
            in_key.add(add_button)
            try:
                caller.append(types.InlineQueryResultArticle(
                    id=call_id,
                    title=call_title,
                    description=f'{price} грн., {call_desc}',
                    thumb_url=image_url, thumb_width=48,
thumb_height=48,

input_message_content=types.InputTextMessageContent(
```

```
                        message_text=f'<b>{call_title}</b>'

f'&#010;&#010;<i>{call_desc}</i>'
                                        f'&#010;<i>{weight}</i>'
                                        f'&#010;<i>{price} грн.</i>'
                                        f'&#010;<a
href="{image_url}">-------</a>',
                        parse_mode='html',
disable_web_page_preview=False),
                    reply_markup=in_key))
            except Exception as e:
                print(e)
        bot.answer_inline_query(query.id, caller, cache_time=0)
    except Exception as e:
        print(e)

@bot.callback_query_handler(func=lambda call: 'add_product_' in
call.data)
def add_to_order_menu_def(call):
    if call.inline_message_id:
        product = re.match(r"add_product_(.+)", call.data).group(1)
        link = f'http://{ip}/api/send_order_cache'
        text = {'text': product, 'user_id': call.from_user.id}
        req = requests.post(link, data=json.dumps(text))

bot.edit_message_text(inline_message_id=call.inline_message_id,
text="Товар успішно доданий до кошику, кількість можна
відредагувати в кошику")

@bot.message_handler(func=lambda message: message.text == "Кошик")
def get_garbage_def(message):
    try:
        link = f'http://{ip}/api/get_order_cache'
        text = {'user_id': message.from_user.id}
        req = requests.post(link, data=json.dumps(text))
        data = json.loads(req.text)
        if not data['list']:
            bot.send_message(chat_id=message.chat.id, text='Ви
нічого не обрали\nНатисніть кнопку: Меню для перегляду страв та
напоїв')
        else:
            for item in data['list']:
                call_name = item['name']
                call_price = item['price']
```

```
                call_count = item['count']
                call_total_price = item['total_price']
                in_key = types.InlineKeyboardMarkup()
                add_button_minus =
types.InlineKeyboardButton(text="-", callback_data=f'count_change_-
_{call_name}')
                add_button_plus =
types.InlineKeyboardButton(text="+",
callback_data=f'count_change_+_{call_name}')
                in_key.add(add_button_minus, add_button_plus)
                bot.send_message(message.chat.id,
text=f'<b>{call_name}</b>'

f'&#010;<b>{call_price}грн. x {call_count}шт. = вартість
{call_total_price}грн.</b>',
                                 parse_mode='html',
disable_web_page_preview=False, reply_markup=in_key)
            key = types.InlineKeyboardMarkup()
            ad_button = types.InlineKeyboardButton(text="Перейти до
оформлення замовлення", callback_data='make_order')
            key.add(ad_button)
            bot.send_message(message.chat.id, "Ви можете перейти до
оформлення замовлення або повернутися до меню", reply_markup=key)
    except Exception as e:
        print(e)

@bot.callback_query_handler(func=lambda call: 'count_change_-_' in
call.data)
def count_change_minus_def(call):
    product = re.match(r"count_change_-_(.+)", call.data).group(1)
    try:
        link = f'http://{ip}/api/del_order_cache'
        text = {'text': product, 'user_id': call.from_user.id}
        req = requests.post(link, data=json.dumps(text))
        link = f'http://{ip}/api/get_product_in_order_cache'
        text = {'text': product, 'user_id': call.from_user.id}
        r = requests.post(link, data=json.dumps(text))
        data = json.loads(r.text)
        if not data['list']:
            bot.edit_message_text(chat_id=call.message.chat.id,
message_id=call.message.message_id, text="Товар видалено з кошика")
        else:
            for item in data['list']:
                call_name = item['name']
```

```
                call_price = item['price']
                call_count = item['count']
                call_total_price = item['total_price']
                in_key = types.InlineKeyboardMarkup()
                add_button_minus =
types.InlineKeyboardButton(text="-", callback_data=f'count_change_-
_{call_name}')
                add_button_plus =
types.InlineKeyboardButton(text="+",
callback_data=f'count_change_+_{call_name}')
                in_key.add(add_button_minus, add_button_plus)
                bot.edit_message_text(chat_id=call.message.chat.id,
message_id=call.message.message_id, text=f'<b>{call_name}</b>'

f'&#010;<b>{call_price}грн. x {call_count}шт. = вартість
{call_total_price}грн.</b>',
                                      parse_mode='html',
disable_web_page_preview=False, reply_markup=in_key)
    except Exception as e:
        print(e)

@bot.callback_query_handler(func=lambda call: 'count_change_+_' in
call.data)
def count_change_plus_def(call):
    product = re.match(r"count_change_\+_(.+)", call.data).group(1)
    try:
        link = f'http://{ip}/api/send_order_cache'
        text = {'text': product, 'user_id': call.from_user.id}
        req = requests.post(link, data=json.dumps(text))
        link = f'http://{ip}/api/get_product_in_order_cache'
        text = {'text': product, 'user_id': call.from_user.id}
        r = requests.post(link, data=json.dumps(text))
        data = json.loads(r.text)
        for item in data['list']:
            call_name = item['name']
            call_price = item['price']
            call_count = item['count']
            call_total_price = item['total_price']
            in_key = types.InlineKeyboardMarkup()
            add_button_minus = types.InlineKeyboardButton(text="-",
callback_data=f'count_change_-_{call_name}')
            add_button_plus = types.InlineKeyboardButton(text="+",
callback_data=f'count_change_+_{call_name}')
            in_key.add(add_button_minus, add_button_plus)
```

```
            bot.edit_message_text(chat_id=call.message.chat.id,
message_id=call.message.message_id, text=f'<b>{call_name}</b>'

f'&#010;<b>{call_price}грн. x {call_count}шт. = вартість
{call_total_price}грн.</b>',
                                  parse_mode='html',
disable_web_page_preview=False, reply_markup=in_key)
    except Exception as e:
        print(e)

@bot.callback_query_handler(func=lambda call: call.data ==
'make_order')
def get_geophone_def(call):
    key = types.ReplyKeyboardMarkup(row_width=1,
resize_keyboard=True)
    button_phone = types.KeyboardButton(text="Відправити номер
телефону", request_contact=True)
    key.row(button_phone)
    key.row("Меню", "Кошик")
    link = f'http://{ip}/api/get_order_cache'
    text = {'user_id': call.from_user.id}
    r = requests.post(link, data=json.dumps(text))
    data = json.loads(r.text)
    total_order_price = 0
    for item in data['list']:
        total_order_price += int(item['total_price'])
    bot.edit_message_text(chat_id=call.message.chat.id,
message_id=call.message.message_id, text=f'<b> Сума замовлення =
{total_order_price}</b>',
                          parse_mode='html',
disable_web_page_preview=False)
    bot.send_message(call.from_user.id, "Передайте свої контактні
дані", reply_markup=key)

@bot.message_handler(content_types=['contact'])
def contact(message):
    if message.contact is not None:
        link = f'http://{ip}/api/get_order_cache'
        text = {'user_id': message.from_user.id}
        r = requests.post(link, data=json.dumps(text))
        data = json.loads(r.text)
        link = f'http://{ip}/api/send_order'
        order = {'user_id': message.from_user.id, 'phone':
message.contact.phone_number, 'list': data['list']}
```

```
        r = requests.post(link, data=json.dumps(order))
        bot.send_message(message.from_user.id, text=f'<b>Дякую</b>'
                                                    f'&#010;Ваше
замовлення в обробці'

f'&#010;очікуйте повідомлення за номером
{message.contact.phone_number}!',
                         parse_mode='html',
disable_web_page_preview=False)
        get_again_button_def(message)

@bot.message_handler(func=lambda message: message.text ==
"Контакти")
def contacts_def(message):
    try:
        bot.send_message(message.chat.id, 'г. Суми\nПокровська
площа, 15А\nGusto Delight\n380993680000\nПН-ВС\n8:00-
21:00\ngustodelight@mail.com')
    except Exception as e:
        print(e)

while True:
    try:
        bot.polling()
    except Exception as e:
        print(e)
        time.sleep(10)
```

views.py:

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from .models import Button, Product, OrderCache, Customer, Order,
OrderItem
import json


class Menu_Button(APIView):
    def get(self, _):
        button = Button.objects.all()
        data = {'list': []}
        for i in button:
            data['list'].append({'name': i.button})
        return Response(data)
```

```
class Get_Product(APIView):
    def get(self, request):
        products = Product.objects.all()
        serialized_products = []
        for product in products:
            serialized_product = {
                'name': product.product_name,
                'desc': product.description,
                'price': product.price,
                'weight': product.weight,
                'image': product.image
            }
            serialized_products.append(serialized_product)
        return Response({'list': serialized_products},
status=status.HTTP_200_OK)

    def post(self, request):
        try:
            data = json.loads(request.body)
            get_button = Button.objects.get(button=data['text'])
            name = Product.objects.filter(button=get_button)
            products = {'list': []}
            for i in name:
                products['list'].append({
                    'name': i.product_name,
                    'desc': i.description,
                    'price': i.price,
                    'weight': i.weight,
                    'image': i.image
                })
            return Response(products, status=status.HTTP_200_OK)
        except Button.DoesNotExist:
            return Response({'error': 'Button not found'},
status=status.HTTP_404_NOT_FOUND)
        except json.JSONDecodeError:
            return Response({'error': 'Invalid JSON'},
status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return Response({'error': str(e)},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

```
class Send_Order_Cache(APIView):
    def post(self, request):
        try:
            data = json.loads(request.body)
            name =
OrderCache.objects.all().filter(user_id=data['user_id'],
name=data['text'])
            if not name:
                name =
Product.objects.filter(product_name=data['text'])
                products = {'list': []}
                for i in name:
                    products['list'].append({'name':
i.product_name, 'price': i.price, 'count': 1,
                                             'total_price':
i.price, 'user_id': data['user_id']})
                    for j in range(len(products['list'])):
                        name = products['list'][j]['name']
                        price = products['list'][j]['price']
                        count = products['list'][j]['count']
                        total_price =
products['list'][j]['total_price']
                        user_id = products['list'][j]['user_id']
                        OrderCache.objects.create(name=name,
price=price, count=count, total_price=total_price,
                                                  user_id=user_id)
            else:
                products = {'list': []}
                name =
OrderCache.objects.filter(user_id=data['user_id'],
name=data['text'])
                for o in name:
                    products['list'].append(
                        {'name': o.name, 'price': o.price, 'count':
o.count, 'user_id': data['user_id']})
                    for j in range(len(products['list'])):
                        name = products['list'][j]['name']
                        price = int(products['list'][j]['price'])
                        count = int(products['list'][j]['count']) +
1
                        total_price = price * count
                        user_id = products['list'][j]['user_id']

OrderCache.objects.get(user_id=data['user_id'],
```

```
name=data['text']).delete()
                        OrderCache.objects.create(name=name,
price=price, count=count, total_price=total_price,
                                                  user_id=user_id)
            return Response(status=status.HTTP_200_OK)
        except json.JSONDecodeError:
            return Response({'error': 'Invalid JSON'},
status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return Response({'error': str(e)},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)


class Get_Order_Cache(APIView):
    def post(self, request):
        try:
            data = json.loads(request.body)
            name =
OrderCache.objects.all().filter(user_id=data['user_id'])
            products = {'list': []}
            for i in name:
                products['list'].append({'name': i.name, 'price':
i.price,
                                         'count': i.count,
'total_price': i.total_price})
            return Response(products, status=status.HTTP_200_OK)
        except json.JSONDecodeError:
            return Response({'error': 'Invalid JSON'},
status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return Response({'error': str(e)},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)


class Get_Product_in_Order_Cache(APIView):
    def post(self, request):
        try:
            data = json.loads(request.body)
            name =
OrderCache.objects.all().filter(name=data['text'],
user_id=data['user_id'])
            products = {'list': []}
            for i in name:
                products['list'].append({'name': i.name, 'price':
```

```
i.price,
                                            'count': i.count,
'total_price': i.total_price})
            return Response(products, status=status.HTTP_200_OK)
        except json.JSONDecodeError:
            return Response({'error': 'Invalid JSON'},
status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return Response({'error': str(e)},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)


class Del_Order_Cache(APIView):
    def post(self, request):
        try:
            data = json.loads(request.body)
            products = {'list': []}
            name =
OrderCache.objects.filter(user_id=data['user_id'],
name=data['text'])
            for o in name:
                products['list'].append(
                    {'name': o.name, 'price': o.price, 'count':
o.count, 'user_id': data['user_id']})
                for j in range(len(products['list'])):
                    name = products['list'][j]['name']
                    price = int(products['list'][j]['price'])
                    count = int(products['list'][j]['count']) - 1
                    total_price = price * count
                    user_id = products['list'][j]['user_id']
                    if count >= 1:

OrderCache.objects.get(user_id=data['user_id'],
name=data['text']).delete()
                        OrderCache.objects.create(name=name,
price=price, count=count, total_price=total_price,
                                                  user_id=user_id)
                    elif count <= 0:

OrderCache.objects.get(user_id=data['user_id'],
name=data['text']).delete()
            return Response(status=status.HTTP_200_OK)
        except json.JSONDecodeError:
            return Response({'error': 'Invalid JSON'},
```

```
status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return Response({'error': str(e)},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)


class Send_Order(APIView):
    def post(self, request):
        try:
            data = json.loads(request.body)
            order = Order.objects.create(user_id=data['user_id'],
phone=data['phone'])
            name =
Customer.objects.all().filter(phone=data['phone'])
            if not name:
                Customer.objects.create(phone=data['phone'])

            for i in range(len(data['list'])):
                phone = data['phone']
                name = data['list'][i]['name']
                price = data['list'][i]['price']
                count = data['list'][i]['count']
                OrderItem.objects.create(order=order, phone=phone,
name=name, price=price, count=count)


OrderCache.objects.filter(user_id=data['user_id']).delete()
            return Response(status=status.HTTP_201_CREATED)
        except json.JSONDecodeError:
            return Response({'error': 'Invalid JSON'},
status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return Response({'error': str(e)},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

База даних:

```
from django.db import models

class Button(models.Model):
     button = models.CharField(max_length=300)

     def __str__(self):
```

```
     return self.button

class Product(models.Model):
     name = models.ForeignKey(Button, on_delete=models.CASCADE)
     product_name = models.CharField('назва продукту',
max_length=1000)
     description = models.TextField('короткий опис',
max_length=1000)
     price = models.CharField('вартість продукту в грн',
max_length=30) #price = models.FloatField(max_length=100)
     weight = models.CharField('вага продукту', max_length=30)
     image = models.TextField('посилання на картинку',
max_length=300)

     def __str__(self):
     return self.product_name

class OrderCache(models.Model):
     name = models.CharField(max_length=300)
     price = models.CharField(max_length=30) #price =
models.FloatField(max_length=100)
     count = models.CharField(max_length=30) #count =
models.PositiveIntegerField(max_length=100)
     total_price = models.CharField(max_length=10000) #total_price
= models.PositiveIntegerField(max_length=100, editable=False)
     user_id = models.CharField(max_length=100)

     def __str__(self):
     return self.name

     #def total_price(self):
     #    self.total_price = self.price * self.count

class Order_Item(models.Model):
     order_id = models.ForeignKey('Order',
on_delete=models.CASCADE)
     #order_id = models.CharField(max_length=100)
     phone = models.CharField(max_length=100)
     name = models.CharField(max_length=100)
     price = models.PositiveIntegerField() #price =
models.FloatField(max_length=100)
     count = models.PositiveIntegerField()
     total_price = models.PositiveIntegerField()
     check = models.BooleanField(default=False)
```

```
    data = models.DateField(auto_now=True)

    def total_price(self):
    self.total_price = self.price * self.count
    return self.total_price

    def __str__(self):
    return self.phone

class Order(models.Model):
    order_id = models.AutoField(primary_key=True)
    user_id = models.CharField(max_length=100)
    phone = models.CharField(max_length=100)
    #total_price = models.PositiveIntegerField()
    check = models.BooleanField(default=False)
    data = models.DateField(auto_now=True)
    order_items = models.ManyToOneRel(Order_Item, to=Order_Item,
field_name='order_id')

    def __str__(self):
    return self.phone

class Customer(models.Model):
    phone = models.CharField(max_length=100, null=True)

    def __str__(self):
    return self.phone
```