Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Інститут комп’ютерних систем
Кафедра інформаційних систем

Кара Микита,
студент групи AI-221

ДИСЦИПЛІНА
Об’єктно-орієнтоване програмування

КУРСОВА РОБОТА
Розробка телеграм-бота для покупки здорового харчування

Спеціальність:
122 Комп’ютерні науки

Освітня програма:
Комп’ютерні науки

Керівник:
Годовиченко Микола Анатолійович,
кандидат технічних наук, доцент

Одеса – 2024

ЗМІСТ

Анотація

Ця курсова робота націлена на створення телеграм-бота, спрямованого на покупку здорового харчування. Основною метою проекту є розробка зручного та ефективного інструменту, який дозволить користувачам замовляти здорову їжу безпосередньо через месенджер Телеграм. Це надасть можливість швидко та без зайвих зусиль отримати здорові продукти прямо на робочому місці або вдома.

Використовуючи мову програмування Java, Spring Framework та Telegram API, цей проєкт спрямований на створення простого та інтуїтивно зрозумілого інтерфейсу для замовлення їжі. Застосування Java Persistence API для зв'язку з базою даних дозволить зберігати та управляти інформацією про замовлення та користувачів.

Отже, розробка цього телеграм-бота відповідає сучасним вимогам до зручності та швидкості обслуговування, допомагаючи користувачам ефективно організувати своє харчування без зайвих зусиль.

ABSTRACT

This coursework is dedicated to developing a Telegram bot aimed at facilitating the purchase of healthy food. The primary objective of the project is to create a convenient and efficient tool that allows users to order food directly through the Telegram messenger. This will enable them to quickly and effortlessly obtain their favorite dishes at their workplace or home.

Utilizing Java programming language, Spring Framework, and Telegram API, this project focuses on creating a simple and intuitive interface for food ordering. The application of Java Persistence API for database connectivity will enable the storage and management of information regarding orders and users.

Thus, the development of this Telegram bot meets contemporary demands for convenience and speed of service, assisting users in efficiently organizing their meals without unnecessary effort.

Вступ

У сучасному світі тенденція до здорового способу життя та правильного харчування набуває все більшої популярності. Однак, зі зростанням обсягів роботи та різноманіттям щоденних зайняттів, час на приготування здорових страв часто стає розкішшю. У такому контексті виникає потреба в швидкому та зручному доступі до здорового харчування.

Користування месенджерами, такими як Telegram, стало не лише засобом спілкування, але й платформою для різноманітних послуг та сервісів. Саме тут виникає ідея створення телеграм-бота, який надасть можливість замовити здорову їжу безпосередньо через месенджер.

Мета цієї роботи полягає в розробці телеграм-бота для покупки здорового харчування. Основна ідея полягає в тому, щоб навіть у розірваний графік дня будь-яка людина могла легко та швидко замовити здорові продукти без надмірних зусиль. Для досягнення цієї мети необхідно вивчити та використати принципи роботи Telegram API, а також володіти навичками програмування, зокрема використовуючи Java та інші технології, що дозволять створити ефективного та інтуїтивно зрозумілого бота.

Таким чином, розробка телеграм-бота для покупки здорового харчування відповідає сучасним потребам і вимогам до зручності та ефективності, сприяючи здоровому способу життя та полегшуючи доступ до якісної їжі навіть у найзайнятіші часи.

1 Проектування телеграм-бота для покупки здорового харчування

1.1 Мета та задачі телеграм-бота

Створення зручної та ефективної платформи для замовлення здорової їжі через Telegram з метою сприяння здоровому способу життя та забезпечення зручності для користувачів.

Основні функції:
Перегляд меню: надання користувачам можливості переглядати різноманітні здорові продукти у зручному форматі.
Додавання в кошик: можливість додавати обрані продукти до кошика для подальшого оформлення замовлення.
Оформлення замовлення: забезпечення можливості користувачам зручно та швидко оформлювати свої замовлення, вказавши необхідні дані.
Вибір методу оплати: надання різних методів оплати для зручності користувачів.
Перегляд історії замовлень: можливість перегляду користувачами своєї історії замовлень для контролю та повторного замовлення.
Зміна та видалення продуктів з кошика: можливість коригувати кошик шляхом зміни кількості або видалення продуктів.
Збереження даних: автоматичне збереження даних користувача для подальшого використання та прискорення процесу замовлення у майбутньому.

Застосування телеграм-бота для замовлення здорової їжі дозволяє користувачам отримати доступ до різноманітних та смачних продуктів, зберігати час та зусилля, які зазвичай витрачаються на пошук та замовлення їжі, та сприяти їхньому здоровому способу життя.

1.2 Визначення функціональних вимог до телеграм-бота

Визначення функціональних вимог є важливим та невід’ємним етапом у процесі створення телеграм-бота для покупки здорового харчування. Функціональні вимоги визначають, які можливості, опції та функції мають

бути реалізовані в чат-боті. Завдяки цьому встановлюється чітка спрямованість розробки, що зменшує кількість непорозумінь.

Також, визначення функціональних вимог дозволяє сфокусуватися на потребах та вимогах користувачів. Це допомагає зрозуміти, які функції та можливості є найбільш корисними для користувачів.

Крім того, функціональні вимоги є основою для комунікації між усіма учасниками проєкту, оскільки вони сприяють розумінню того, що саме потрібно реалізувати та що очікується в результаті.

Таким чином, встановлення функціональних вимог дозволяє визначити загальний обсяг роботи, а також ефективно планувати робочий час, бюджет та необхідні ресурси для проєкту.

Для того, щоб визначити, як користувачі будуть використовувати телеграм-бота та які вимоги можна для нього сформувати, була створена діаграма сценаріїв для нього (рис. 1.1). Діаграма сценаріїв UML (Unified Modeling Language) - це спосіб графічно показати функції системи та їх зв’язок з користувачами. На діаграмі зображені актори, сценарії та їх взаємодія. Ця діаграма дозволяє проаналізувати потреби акторів, можливості системи та визначити вимоги до неї. Також ця діаграма наочно демонструє розробникам основні можливості та функції системи.

Телеграм-бот для покупки здорового харчування має лише одного актора, і це «користувач». Користувач має можливість переглядати меню здорових продуктів, обирати з них, отримувати більш детальну інформацію про кожен продукт, додавати їх до кошика та оформлювати замовлення для доставки, тобто він має доступ до повного функціонала бота.

Рисунок 1.1 – Діаграма варіантів використання телеграм-бота

1.3 Формування користувацьких історій телеграм-бота

За допомогою спроектованої діаграми варіантів використання телеграм-бота, можна визначити такі користувацькі історії чат-бота для замовлення їжі.

US1 Як користувач, я можу переглядати меню, щоб обрати продукт для замовлення.

Сценарій користувацької історії:

- користувач відкриває спілкування з ботом в месенджері;

- користувач натискає на кнопку «Меню»;
- після цього на екрані з’являються різні категорії меню;
- обравши необхідну категорію, з’являються продукти, що належать до цієї категорії;
- обравши продукт, користувач може переглянути інформацію про нього.

US2 Як користувач, я можу переглядати детальну інформацію про продукти, щоб краще розуміти чи варто її замовляти.

Сценарій користувацької історії:

- після обрання категорії продуктів в меню, на екрані з’являється перелік продуктів, що належать до цієї категорії;
- натиснувши на кнопку з назвою продукта, користувач може переглянути детальну інформацію про неї: опис, фото, ціна тощо.

US3 Як користувач, я можу додати обраний продукт до кошику, щоб мати змогу оформити замовлення.

Сценарій користувацької історії:

- користувач обрав продукт з меню та отримав детальну інформацію про неї;
- користувач натискає на кнопку «Додати до кошику» під описом продукта;
- Обраний продукт додається до кошику користувача.

US4 Як користувач, я можу переглянути вміст кошику.

Сценарій користувацької історії:

- після додавання продукта до кошику, з’являється кнопка «Переглянути кошик»;
- натиснувши на неї користувач може переглянути продукти, які знаходяться в його кошику.

Інший можливий сценарій користувацької історії:

- натиснувши на кнопку «Повернутися в головне меню», користувач переходить до головного меню;
- в головному меню користувач натискає на кнопку «Кошик»;
- на екран виводиться вміст кошику користувача.

US5 Як користувач, я можу оформити замовлення.

Сценарій користувацької історії:

- після перегляду кошику, користувач натискає на кнопку «Оформити замовлення»;
- користувач вводить свої особисті дані, такі як ім’я, адреса доставки, номер телефону для зв’язку;
- після чого на екрані з’являється повідомлення з переліком обраних продуктів та введеною особистою інформацією, з метою перевірки коректності інформації;
- користувач натискає кнопку «Оформити замовлення»;
- замовлення оформлено та відправляється на обробку.

US6 Як користувач, я можу обрати спосіб оплати замовлення, щоб оплатити зручним для мене методом.

Сценарій користувацької історії:

- під час оформлення замовлення, користувачеві пропонується обрати зручний для нього метод оплати замовлення;
- натиснувши на кнопку «Картка», користувач обирає сплату замовлення банківською картою;
- натиснувши на кнопку «Готівка», користувач обирає сплату замовлення готівкою.

US7 Як користувач, я можу переглядати історію своїх замовлень та стан свого замовлення, щоб мати можливість повторно робити замовлення або відстежувати на якому етапі готовності знаходиться поточне замовлення.

Сценарій користувацької історії:

- в головному меню бота, користувач натискає на кнопку «Попередні замовлення»;
- серед запропонованих номерів замовлень, клієнт обирає потрібний, натиснувши на нього;
- на екран виводиться інформація про обране замовлення користувача.

US8 Як користувач, я можу вказати додаткову інформацію про замолення, щоб уточнити деяку інформацію стосовно замовлення або його замолення.

Сценарій користувацької історії:

- при оформленні замовлення, користувачу пропонується надати додаткову інформацію до замовлення;
- користувач натискає на кнопку «Пропустити» та пропускає цей етап;
- або користувач вказує додаткову інформацію та переходить до підтвердження замовлення.

US9 Як користувач, я можу здійснити пошук продукта за назвою, щоб швидше знайти необхідний продукт.

Сценарій користувацької історії:

- користувач, знаходячись в меню, вводить назву продукта, інформацію про яку бажає переглянути;
- після чого виводиться інформація про обраний продукт та з’являється можливість додати її до кошику;
- якщо назва продукта введена некоректно або продукт з такою назвою не наявна в боті, виводиться повідомлення про помилку.

US10 Як користувач, я можу змінювати кількість продуктів в кошику, щоб отримати необхідну для мене кількість продуктів.

Сценарій користувацької історії:

- користувач, переглянувши вміст кошику, натискає на кнопку «Внести зміни до кошику»;
- користувач натискає на кнопку «Змінити кількість продуктів»;

- з наведеного переліку користувач обирає продукт, кількість якого бажає змінити;
- обравши продукт, виводиться інформація про нього, та його кількість в кошику;
- натискаючи на кнопку «+», користувач може додати ще один продукт до кошику;
- натискаючи на кнопку «–», користувач може зменшити кількість цього продукту в кошику.

US11 Як користувач, я можу видалити продукт кошику, щоб не отримати непотрібний для мене продукт.

Сценарій користувацької історії:

- користувач, переглянувши вміст кошику, натискає на кнопку «Внести зміни до кошику»;
- користувач натискає на кнопку «Видалити продукт з кошику»;
- з наведеного переліку користувач обирає продукт, який бажає видалити;
- обраний продукт видаляється з кошику.

Таким чином, було сформовано основні користувацькі історії для телеграм-ботудля покупки здорового харчування. Розроблені користувацькі історії допоможуть при проектуванні та розробці бота.

1.4 Проектування користувацького інтерфейсу телеграм-бота

На рисунку 1.2 представлено макет головного меню телеграм-бота. Він містить три кнопки: Меню, Кошик та Попередні замовлення. За допомогою цієї клавіатури користувач може перейти до перегляду меню, кошика або своїх попередніх замовлень.

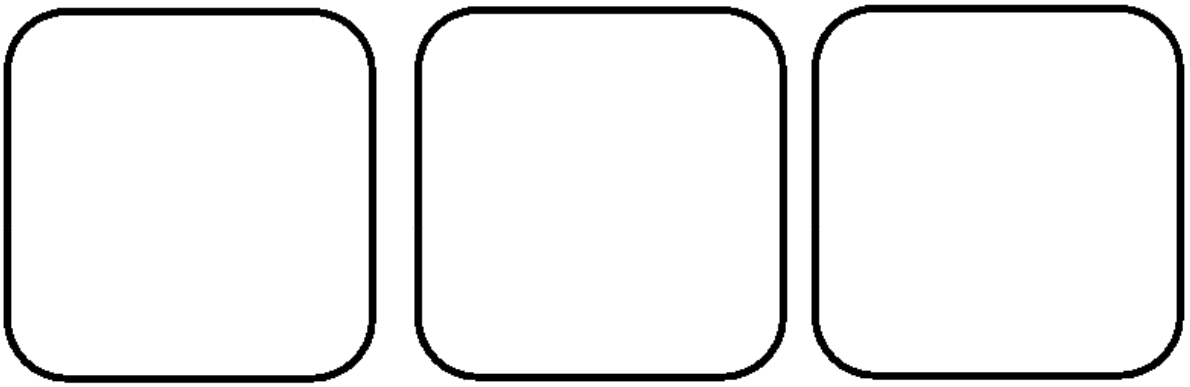

Рисунок 1.2 – Макет клавіатури головного меню бота

На рисунку 1.3 показано макет вибору категорій продуктів, який з’являється після переходу до перегляду меню. Макет включає кнопки для кожної категорії меню та кнопку повернення до головного меню. За допомогою цієї клавіатури користувач може обрати категорію, яка його цікавить, та перейти до перегляду продуктів цієї категорії.

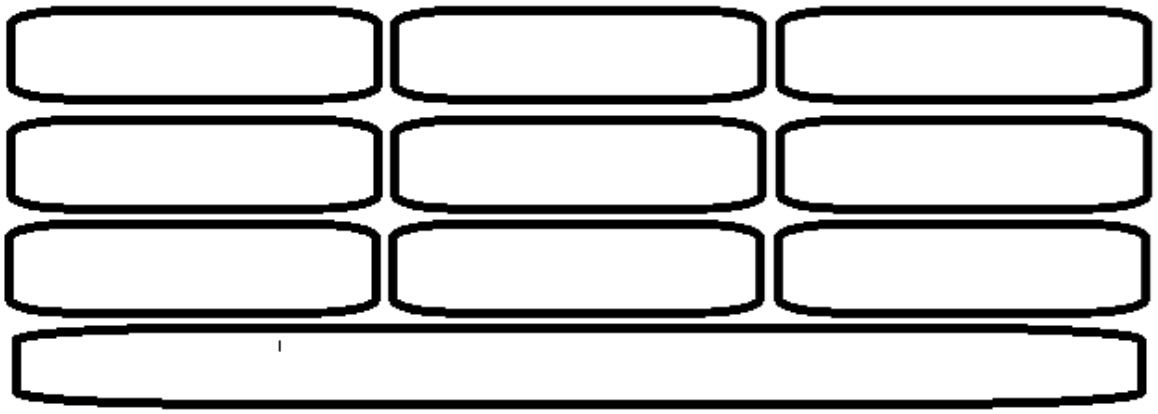

Рисунок 1.3 – Макет клавіатури вибору категорій меню

На рисунку 1.4 показано макет вибору продукта, доступний після вибору категорії меню. Макет включає кнопки для кожного продукта, що належить до обраної категорії, а також кнопку повернення до перегляду меню. За допомогою цієї клавіатури користувач може обрати продукт, який його зацікавив, для перегляду детальної інформації про нього та подальшого додавання до кошика.

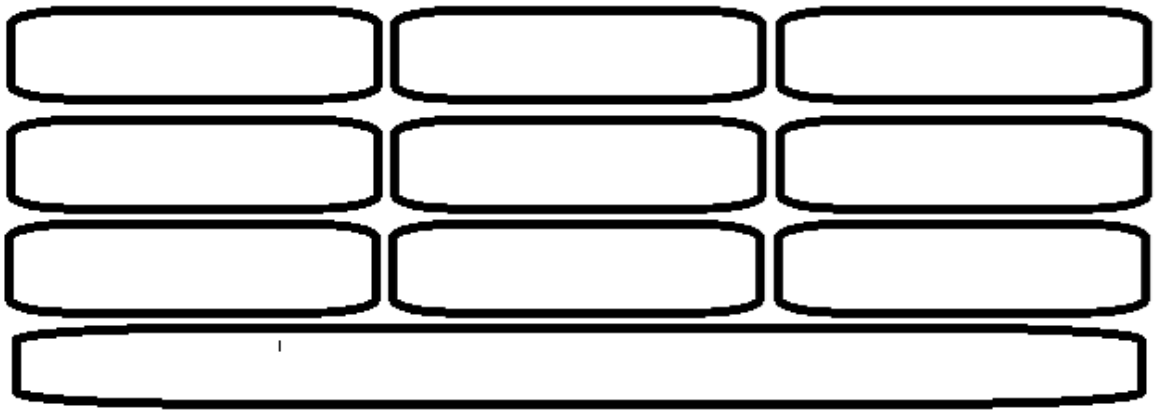

Рисунок 1.4 – Макет клавіатури вибору продукта з меню

На рисунку 1.5 зображено макет повідомлення, яке отримує користувач після вибору продукта, який його зацікавив. Макет містить зображення продукта, її детальний опис, що включає ціну, вагу, опис тощо, а також кнопку для додавання цього продукта до кошика. Це повідомлення надає користувачу всю необхідну інформацію про продукт, що допомагає йому вирішити, чи бажає він його замовити, і дає можливість додати продукт до кошика.

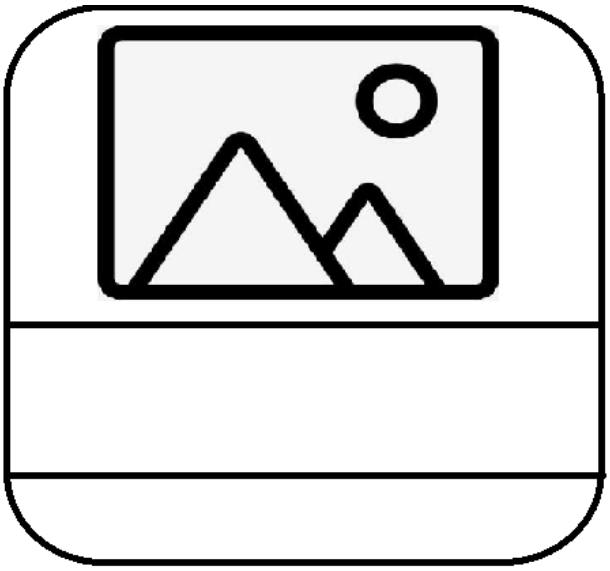

Рисунок 1.5 – Макет повідомлення з інформацією про продукт

На рисунку 1.6 зображено макет повідомлення, яке отримує користувач при прийнятті рішення про зміну кількості продуктів в кошику. Макет складається з зображення продукта, його опису, в якому також зазначається її кількість в кошику, та кнопок зменшення та збільшення кількості обраної продуктів. Таким чином, з цього повідомлення користувач може остаточно переконатися в тому чи потрібно йому змінювати кількість цього продукта, а також, при необхідності, додати ще один такий продукт або видалити зайвий.

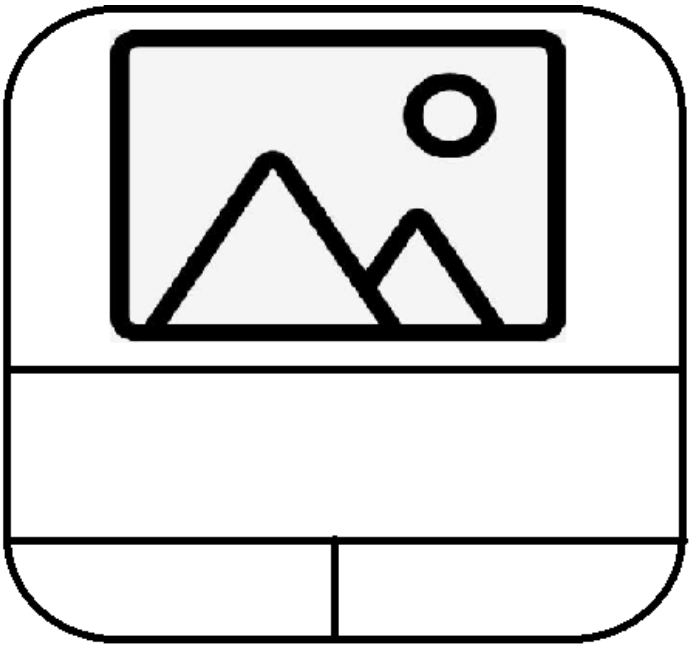

Рисунок 1.6 – Макет повідомлення для зміни кількості продуктів в кошику

На рисунку 1.7 зображено макет повідомлення, яке отримує користувач при оформленні замовлення, а саме перед його остаточним підтвердженням. Макет складається з опису замовлення, куди включена надана користувачем інформація про доставку замовлення та перелік обраних продуктів, а також під повідомленням розташована кнопка підтвердження замовлення. Таким чином, це повідомлення надає можливість користувачу перевірити коректність замовлення та підтвердити його.

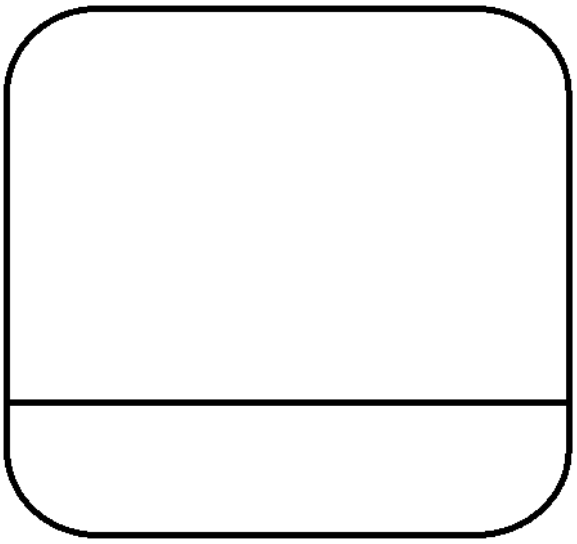

Рисунок 1.7 – Макет повідомлення про підтвердження замовлення

Отже, було створено макет основних сценаріїв взаємодії користувача з телеграм-ботом, що зробить розробку бота більш цілеспрямованою завдяки чіткому розумінню кінцевого вигляду чат-бота. Інші сценарії взаємодії будуть будуватися на цих засадах. Наприклад, меню налаштування кошика матиме подібну клавіатуру до головного меню, але кнопки виконуватимуть інші функції.

2 Програмна реалізація телеграм-бота для покупки здорового харчування

2.1 Структура серверного програмного проекту

Пакет програмного проекту Spring для телеграм-бота покупки здорового харчування може складатися з різних компонентів та класів. Визначимо компоненти та класи, які буде необхідно реалізувати при розробці.

Головним класом проекту є TelegramBot, він відповідає за взаємодію між користувачем та ботом. Його основними задачами є обробка повідомлень від користувача, їх аналіз, формування та надсилання відповідей на повідомлення та запити користувача, а також створення інтерфейсу у вигляді різних клавіатур та кнопок під повідомленнями для користувача.

До моделей проекту, тобто сутностей, які використовуються в системі та, зазвичай, мають зв’язок з базою даних, можна віднести два класи: Products та Order. Клас Products є сутністю продукта, дані про яку отримуються з бази даних та після чого оброблюються ботом. Клас Order є сутністю замовлення, телеграм-бот формує дані про нього та зберігає їх у базу даних для подальшої обробки менеджерами закладу. Також, дані про замовлення можуть бути отримані з бази даних, наприклад, для надання можливості користувачу переглядати попередні замовлення.

До створених моделей необхідно створити репозиторії, саме вони відповідають за збереження та отримання даних з бази даних. В нашому випадку необхідно створити два репозиторії: ProductsRepository – репозиторій продуктів, OrdersRepository – репозиторій замовлень.

Клас проекту Cart є кошиком користувача. До основного функціоналу цього класу належить додавання та видалення продуктів з кошику, отримання інформації про те, які продукти знаходяться в кошику, формування списку продуктів для оформлення замовлення, а також зміна кількості продуктів в кошику.

Також, проект містить такі класи як BotInitializer та BotConfig, за допомогою цих класів виконується налаштування телеграм-бота для його коректної роботи з сервісом Telegram. Крім того, проект містить конфігураційні файли, за допомогою яких відбувається конфігурування та налаштування фреймворку Spring, а також зв'язку з базою даних, що сприяє коректній роботі всіх елементів проекту.

2.2 Функціональне тестування розробленого телеграм-бота

Функціональне тестування телеграм-бота для покупки здорового харчування є важливим етапом його розробки, оскільки воно дозволяє перевірити правильність роботи всіх функцій бота.

Це тестування допомагає переконатися в коректності виконання розроблених функцій бота та гарантує, що весь функціонал чат-бота працює як очікувалося, без помилок, які можуть заважати роботі сервісу.

Крім того, функціональне тестування допомагає виявити помилки в роботі бота, що можуть негативно впливати на користувацький досвід. Після виявлення таких помилок проводиться їх аналіз і виправлення, що забезпечує надійність та стабільність сервісу для користувачів.

Для проведення функціонального тестування необхдно скласти протокол тестування. Протокол тестування - це документ, що описує кроки, процедури та результати тестування програмного продукту, системи або її окремих компонентів. Він призначений для систематичного документування процесу тестування та його результатів, забезпечуючи максимальну об'єктивність отриманих даних.

Для функціонального тестування телеграм-бота для покупки здорового харчування був розроблений наступний протокол тестування.

ТС1 Тест-кейс для відображення категорій меню:

- відкрити чат з ботом в месенджері;
- натиснути на кнопку «Меню»;

- переконатися в тому, що всі додані категорії меню відображаються.

ТС2 Тест-кейс для відображення продуктів в меню:

- відкрити чат з ботом в месенджері;
- натиснути на кнопку «Меню»;
- натиснути на кнопку з категорією продуктів, які цікавлять;
- переконатися в тому, що всі продукти, які належать до обраної категорії, відображаються коректно.

ТС3 Тест-кейс для відображення інформації про продукт:

- відкрити чат з ботом в месенджері;
- натиснути на кнопку «Меню»;
- натиснути на кнопку з категорією продуктів, які цікавлять;
- натиснути на кнопку з назвою продукта, для отримання інформації про нього;
- переконатися в тому, шо відображення інформації про продукт відбувається коректно.

ТС4 Тест-кейс для додавання продукта до кошика:

- відкрити чат з ботом в месенджері;
- натиснути на кнопку «Меню»;

- натиснути на кнопку з категорією продуктів, які цікавлять;
- натиснути на кнопку з назвою продукт, який бажаємо замовити;
- натиснути на кнопку «Додати до кошику», під описом продукта;
- натиснути на кнопку «Переглянути кошик»;
- переконатися в тому, що обраний продукт був у кошику.

ТС5 Тест-кейс для перегляду вмісту кошику:

- після додавання товарів до кошику, натиснути на кнопку «Переглянути кошик»;
- переконатися в коректності інформації про вміст кошика.

ТС6 Тест-кейс для видалення продукту з кошика:

- після додавання продукту до кошика, натиснути кнопку «Переглянути кошик»;
- в меню кошика натиснути на кнопку «Внести зміни до кошика»;
- натиснути на кнопку «Видалити продукт з кошику»;
- натиснути на кнопку з назвою продукта, який бажаємо видалити;

- натиснути на кнопку «Переглянути кошик»;
- переконатися в тому, що продукт був видалений з кошику.

ТС7 Тест-кейс для зміни кількості продуктів у кошику:

- після додавання продукта до кошика, натиснути кнопку «Переглянути кошик»;
- в меню кошика натиснути на кнопку «Внести зміни до кошика»;
- натиснути на кнопку «Змінити кількість продуктів»;
- натиснути на кнопку з назвою продукта, кількість якого бажаємо змінити;
- натиснути на кнопку «–», тим самим зменшити кількість продуктів, або натиснути на кнопку «+» для збільшення кількості продуктів;
- натиснути на кнопку «Переглянути кошик»;
- переконатися в тому, що кількість продуктів була змінена.

ТС8 Тест-кейс для оформлення замовлення:

- після додавання продуктів до кошика, натиснути кнопку «Переглянути кошик»;
- в меню кошика натиснути на кнопку «Оформити замовлення»;

- ввести всі необхідні дані, такі як ім’я отримувача, номер телефону для зв’язку та адреса доставки замовлення;
- натиснути на кнопку «Готівка», для обрання оплати замовлення готівкою, або натиснути на кнопку «Картка», для сплати замовлення банківською картою;
- вказати додаткову інформацію до замовлення або натиснути на кнопку «Пропустити», щоб пропустити цей етап;
- перевірити коректність інформації, яка надається ботом перед підтвердженням замовлення;
- натиснути на кнопку «Оформити замовлення»;
- натиснути на кнопку «Перегляд минулих замовлень»;
- натиснути на кнопку з номером створеного замовлення;
- перевірити коректність збереження інформації про замовлення.

ТС9 Тест-кейс для перегляду попередніх замовлень:

- після оформлення замовлення натиснути на кнопку «Перегляд минулих замовлень»;
- натиснути на кнопку з номером створеного замовлення;
- перевірити коректність інформації про створене замовлення.

ТС10 Тест-кейс для застосування підказок під час оформлення замовлення:

- після додавання продукта до кошика, натиснути кнопку «Переглянути кошик»;
- в меню кошика натиснути на кнопку «Оформити замовлення»;
- під час прохання ввести особисту інформацію, переконатися в тому, що відображається кнопки з варіантами, які були введені при попередніх замовленнях;
- натиснути на кнопку з одним з варіантів;
- перед підтвердженням замовлення переконатися в тому, що дані відображені коректно.

Таблиця 2.1 – Протокол функціонального тестування телеграм-бота

Номер тест-кейсу	Очікуваний результат	Фактичний результат	Результат тестування
TC1	Коректне відображення категорій меню	Всі додані категорії меню відображаються коректно	Успішно
TC2	Коректне відображення продуктів в меню	Відображаються тільки ті продукти, які належать до обраної категорії	Успішно
TC3	Коректне відображення інформації про продукт	Інформація про обраний продукт надається користувачу	Успішно
TC4	Успішне додавання продукта до кошику	Продукт додається до кошику	Успішно
TC5	Коректний перегляд вмісту кошика	Отримання вірної інформація про вміст кошика	Успішно
TC6	Успішне видалення продукта з кошика	Продукт видаляється з кошика	Успішно
TC7	Успішна зміна кількості продуктів	Кількість продуктів в кошику змінюється	Успішно

ТС8	Успішне оформлення замовлення	Нове замовлення додається до бази даних	Успішно
ТС9	Коректний перегляд попередніх замовлень	Отримана вірна інформація про попередні замовлення	Успішно
ТС10	Виведення підказок під час оформлення замовлення	Під час оформлення замовлення виводиться дані, які були надані при оформленні попередніх замовлень	Успішно

Результати функціонального тестування телеграм-бота для покупки здорового харчування в таблиці 2.1.

З результатів тестування можна побачити, що всі функціональні тести пройдені успішно, з чого можна зробити висновок, що фактична поведінка телеграм-бота відповідає очікуваній поведінці, яка було визначена у функціональних вимогах.

2.3 Інструкція користувача телеграм-бота

З метою забезпечення зручності використання розробленого телеграм-бота для продажу здорового харчування, необхідно скласти інструкцію користувача, яка включає в себе знімки екранів та пояснювальний текст до кожного з них. Це допоможе користувачам легко орієнтуватися у функціоналі та особливостях створеного бота та отримати успішний досвід використання чат-бота.

На рисунку 2.2 зображено головне меню телеграм-бота, яке зустрічає користувача одразу на початку роботи з ботом. Головне меню складається з трьох кнопок: Меню, Кошик та Перегляд минулих замовлень. За допомогою цього меню відбувається навігація серед основних напрямів: переглядом продуктів з меню, переглядом вмісту кошика та його зміни, а також переглядом попередніх замовлень користувача.

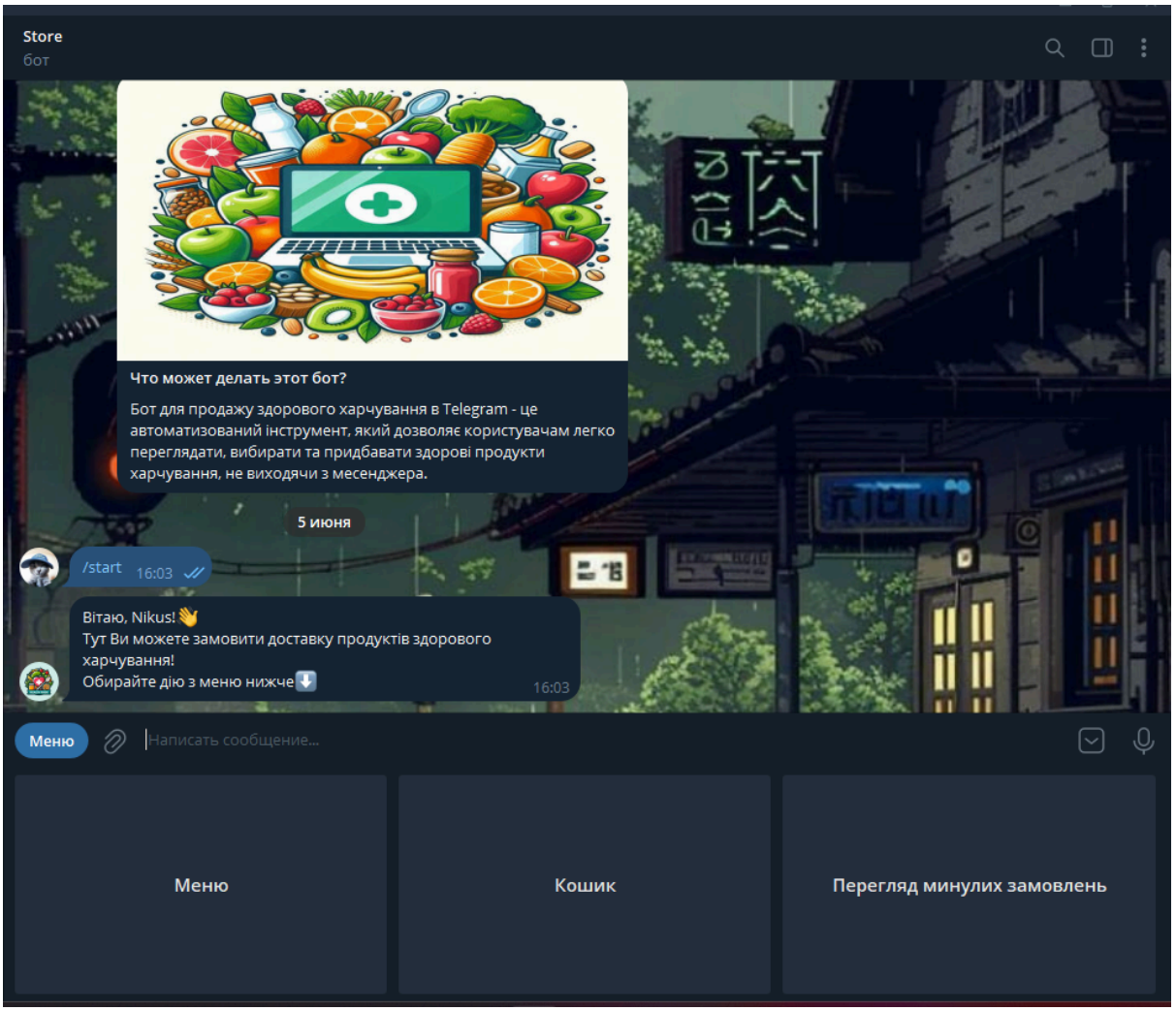

Рисунок 2.2 – Головне меню телеграм-бота

На рисунку 2.3 зображено категорії меню, які з’являються перед користувачем після обрання перегляду меню бота. Категорії представлені окремими кнопками, натискання на кожну з яких призведе до перегляду продуктів цієї категорії. Також присутня кнопка повернення до головного меню та кнопка перегляду всього меню, якщо користувач не бажає переглядати окремий продукт. Таким чином, це дозволяє зручно та зрозуміло орієнтуватися в меню, щоб обрати необхідний продукт.

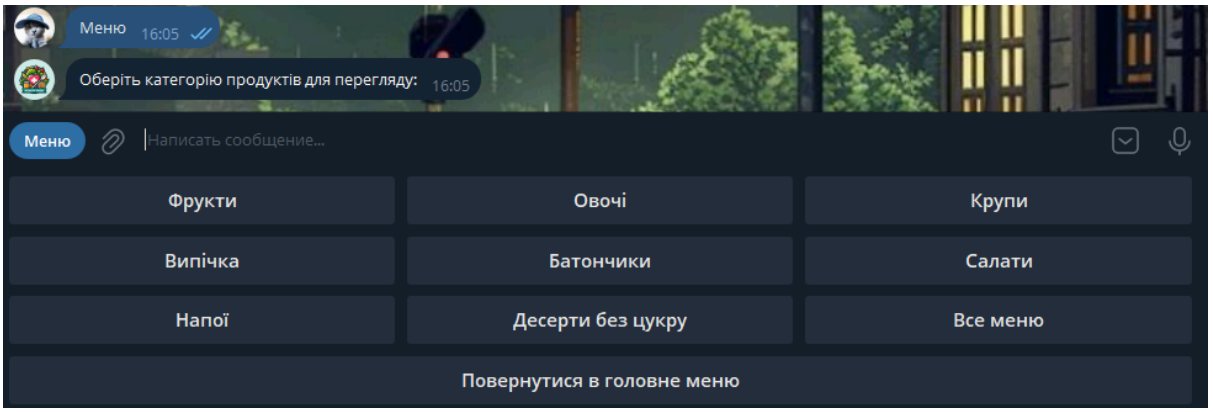

Рисунок 2.3 – Обрання категорії меню

На рисунку 2.4 зображено перелік продуктів, які відносять до обраної категорії, наприклад Фрукти, а також демонстрація інформації про обраний продукт. Натискаючи на назву продукта, яку бажає переглянути користувач, з’являється повідомлення з описом обраного продукта, її фотографією, а також кнопкою додавання цього продукта до кошику. Таким чином, обравши продукт, можна додати її до кошику, або, якщо хочеться переглянути інший продукт, можна натиснути на його назву внизу та подивитися інформацію про нього. Також присутня кнопка для повернення в меню перегляду категорій продуктів. Таким чином, можна переглянути всі продукти які належать до обраної категорії, переглянути детальну інформацію про кожний з них, при бажанні, додати їх до кошику та повернутися в меню, при необхідності.

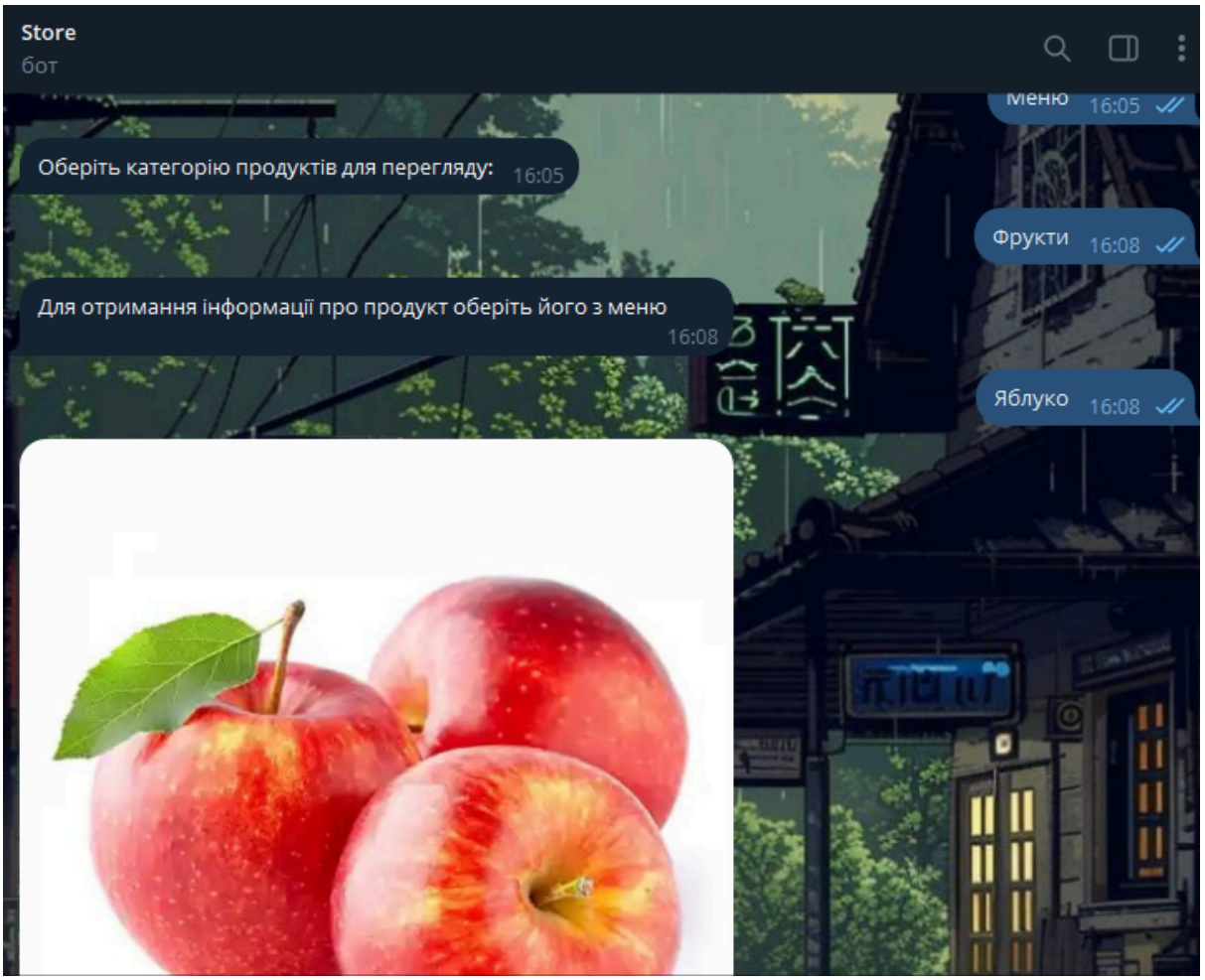

Рисунок 2.4(1) – Перегляд продуктів з меню

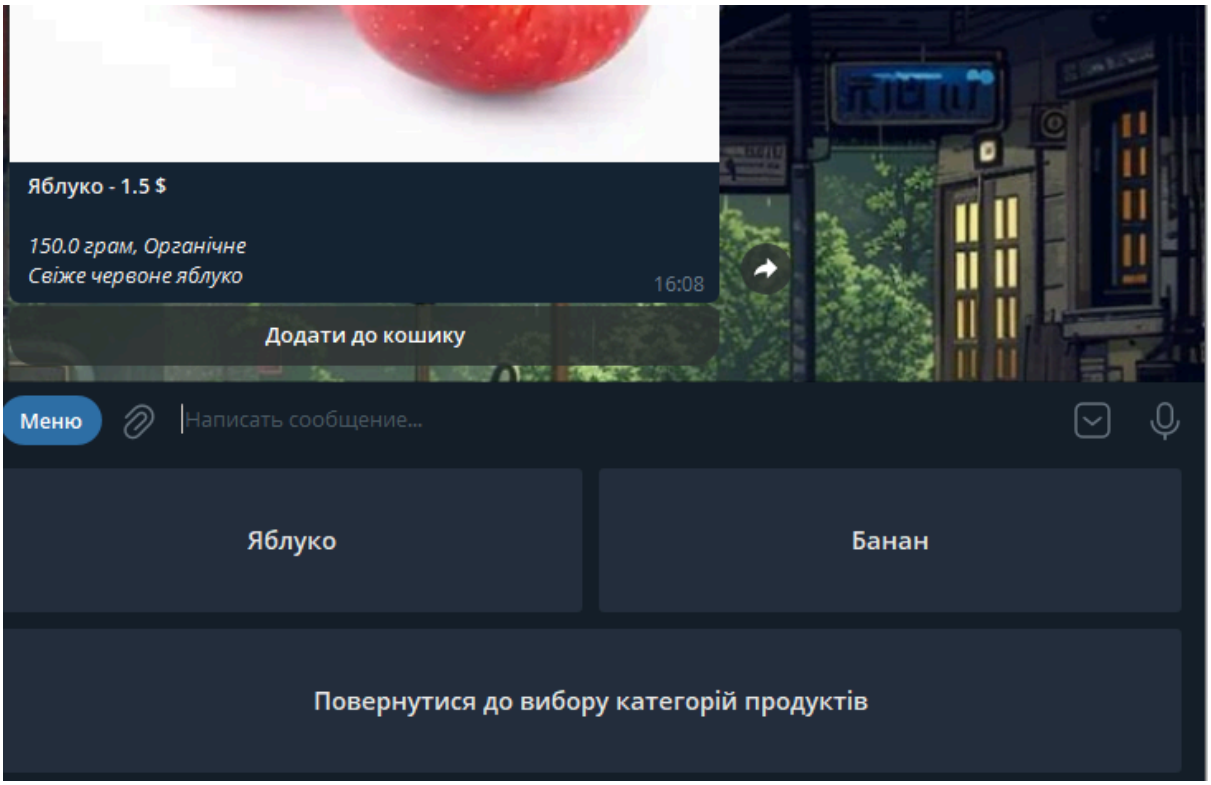

Рисунок 2.4(2) – Перегляд продуктів з меню

На рисунку 2.5 зображено додавання продукт до кошику. Після натискання на кнопку «Додати до кошику» продукт додається до кошику. Після цього користувач може натиснути на кнопку «Переглянути кошик». На екрані з’явиться меню кошику, таке саме, як при переході з головного меню до кошику. Користувачу демонструється перелік продуктів, які на поточний момент знаходяться в його кошику. Також, користувач має можливість, шляхом натискання на кнопки розташовані знизу, перейти до оформлення замовлення, внести зміни до кошику, повторно переглянути його вміст, а також повернутися в головне меню.

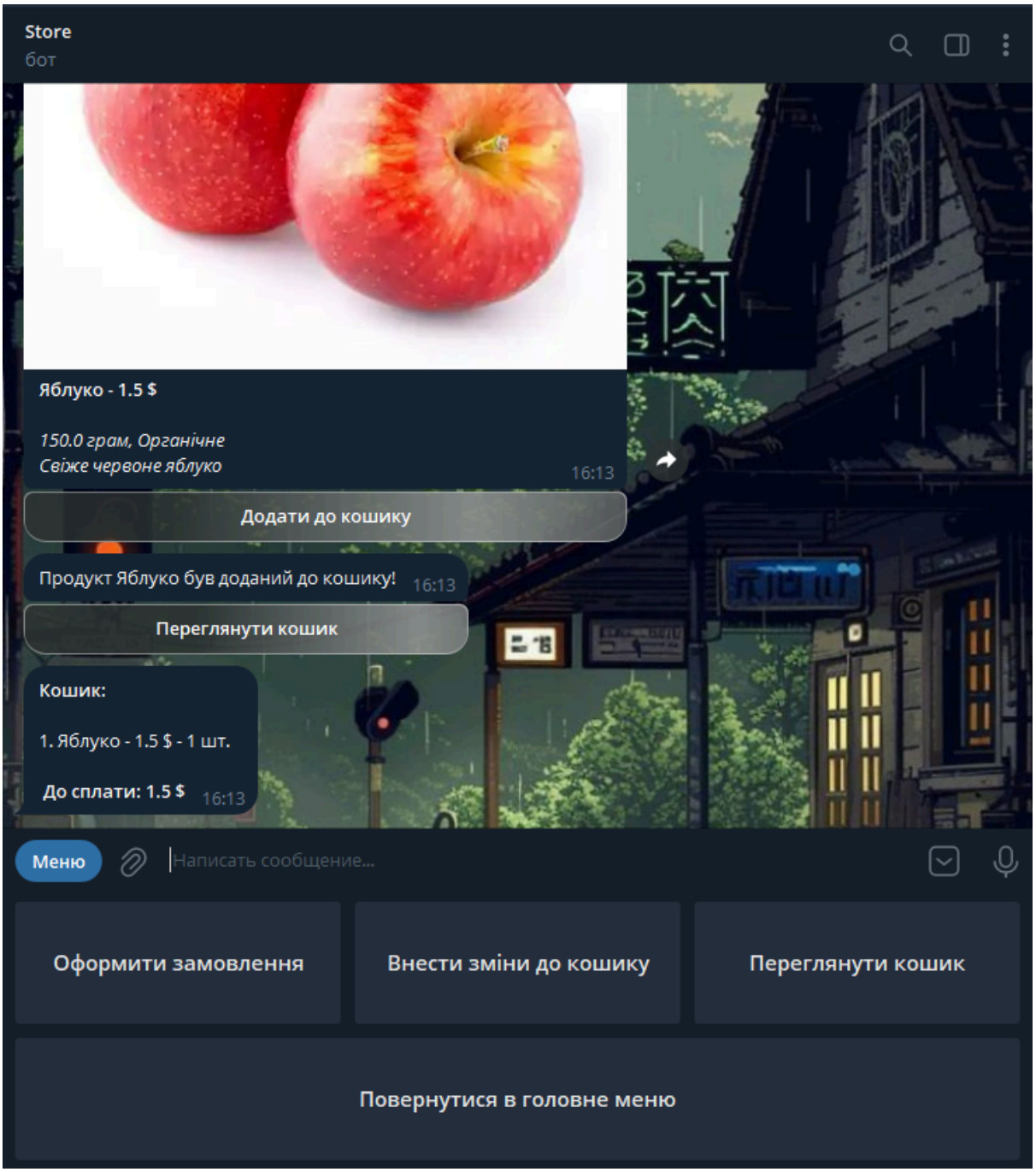

Рисунок 2.5 – Додавання продукта до кошику та перегляд його вмісту

На рисунку 2.6 зображено меню внесення змін до кошику. Воно складається з кнопок зміни кількості продуктів, видалення продуктів з кошику, очищення кошику та повернення до меню кошика. За допомогою цього меню користувач може виконувати основні операції зі вмістом кошика, змінюючи його.

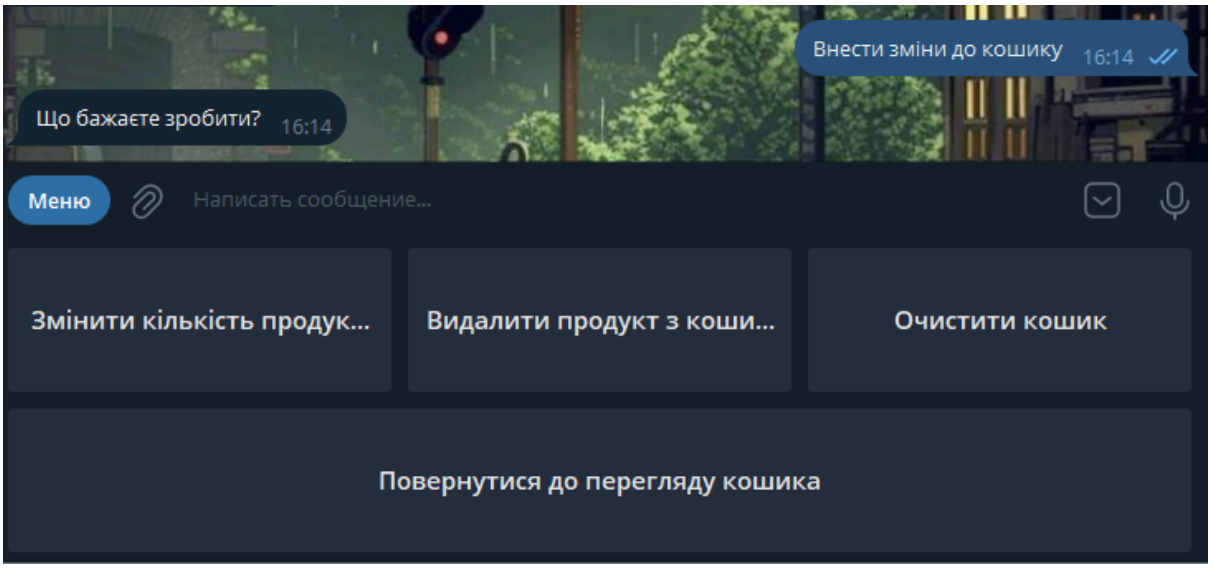

Рисунок 2.6 – Внесення змін до кошику

На рисунку 2.7 зображено видалення продукта з кошику. Меню видалення продуктів складається з кнопок з назвою кожного продукта з кошика. Для видалення продукта необхідно натиснути на кнопку з її назвою внизу, після чого продукт буде видалений, а користувача буде перенаправлено до меню кошика.

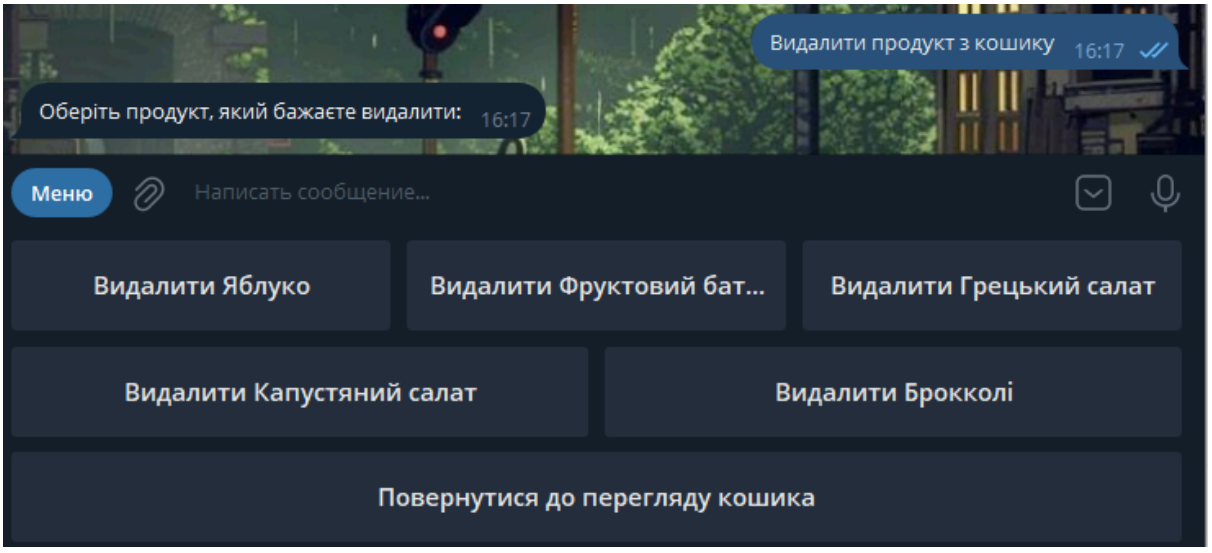

Рисунок 2.7 – Видалення продукту з кошику

На рисунку 2.8 зображено процес оформлення замовлення. Після того, як користувач визначився з продуктами, які бажає замовити, він може перейти до оформлення замовлення. Обравши в меню кошика оформлення замовлення, користувачу необхідно ввести свої контактні дані та адресу доставки, після чого, натиснувши на відповідну кнопку, обрати спосіб оплати замовлення, та, при необхідності, вказати додаткову інформацію до замовлення. Після цього буде сформовано повідомлення для підтвердження замовлення, в якому користувачу потрібно перевірити коректність наданої інформації, після чого оформити замовлення, натиснувши на відповідну кнопку. Таким чином замовлення буде успішно оформлено та відправлено на обробку до менеджеру закладу.

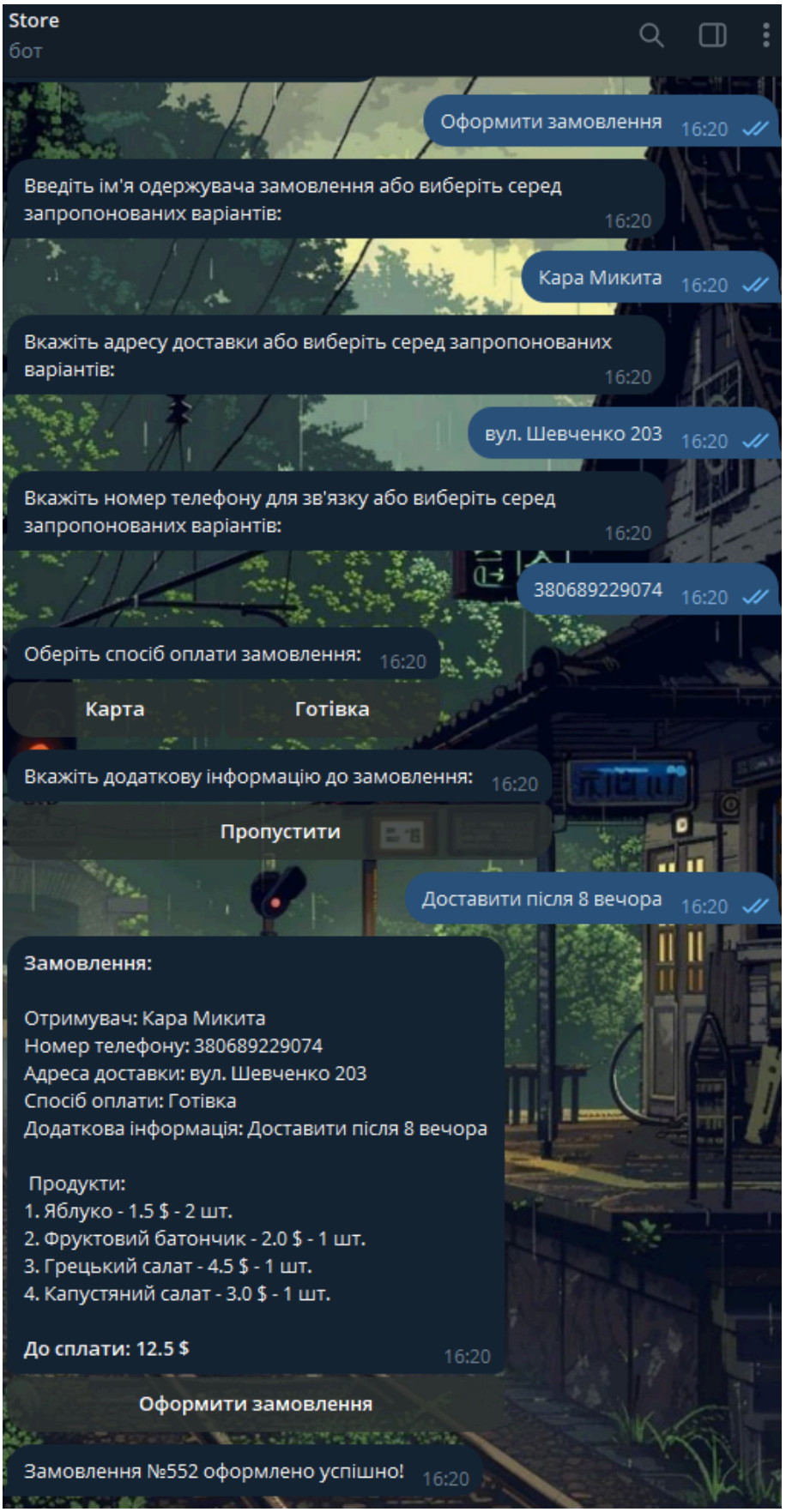

Рисунок 2.8 – Оформлення замовлення

На рисунку 2.9 зображено перегляд попередніх замовлень користувача. Для перегляду попередніх замовлень, користувачу необхідно зайти у відповідне меню. Воно складається з номерів замовлень, які робив цей користувач. Натиснувши на відповідний номер замовлення, користувач може переглянути всю інформацію про нього, включно з його станом готовності.

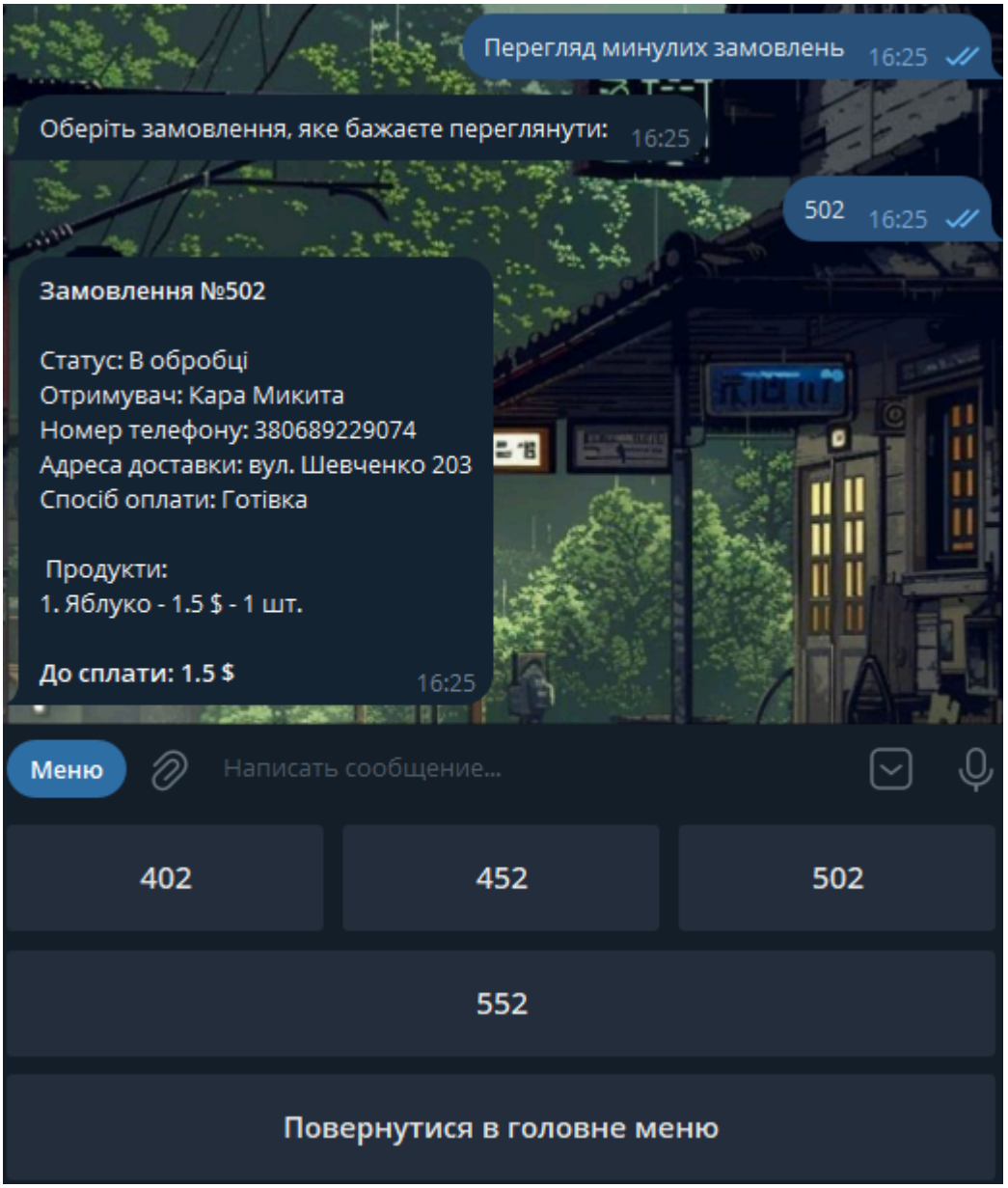

Рисунок 2.9 – Перегляд попередніх замовлень

Таким чином, було розглянуто основні меню телеграм-бота та можливі сценарії взаємодії користувача з ним. Виходячи з цього можна зробити висновок, що користувач може зручно та всього за декілька хвилин переглянути різні продукти з меню, додати їх до кошику та оформити замовлення, а також, при необхідності, видаляти продукти з кошику та змінювати їх кількість.

2.4 Код телеграм-бота

Код телеграм-бота для зручності знаходиться в кінці ПЗ.

2.5 Висновки розділу

В третьому розділі курсової роботи було виконано програмну реалізацію телеграм-бота для продажу здорового харчування. Для успішного виконання програмної реалізації бота було використано розроблений у першому розділі цієї роботи проект телеграм-бота.

Була розглянута структура серверного програмного проекту. Як результат, було визначено основні компоненти та класи, з яких складається проект телеграм-бота для продажу здорового харчування.

Крім того, було проведено функціональне тестування розробленого телеграм-бота. Був розроблений протокол тестування, який складався з множини тест-кейсів, за яким було проведено тестування розробленого чат-бота. В результаті чого, всі тест-кейси були успішно виконані. Це дозволяє переконатися в тому, що фактична поведінка телеграм-бота відповідає очікуваній поведінці, яка було визначена у функціональних вимогах.

Завершальним етапом, стала розробка інструкції користувача, яка складається з множини знімків екрану та пояснювального тексту до них. В поясненні розглядаються можливі дії користувача та функціонал телеграм-бота при конкретних сценаріях взаємодії користувача з чат-ботом. Також був наданий вихідний код програмних класів розробленого телеграм-бота.

Висновки

У цій курсовій роботі було розроблено телеграм-бот для продажу здорового харчування. Створений бот дозволяє користувачам швидко та зручно замовляти доставку корисних продуктів, не відволікаючись від важливих спілкувань у чатах. Користувачі мають можливість переглядати різні продукти в меню, додавати їх до кошика, оформлювати замовлення, змінювати склад кошика та переглядати свої попередні замовлення. Таким чином, основна мета цієї курсової роботи була успішно досягнута, а для цього було вирішено низку завдань.

У першому розділі курсової роботи була детально досліджена предметна область розробки телеграм-бота для продажу здорового харчування. Було сформульовано та детально описано основні вимоги до телеграм-бота. Також було проведено огляд сучасних інформаційних технологій для розробки телеграм-бота, таких як фреймворк Spring та Telegram API.

Далі, у першому розділі, було здійснено проектування телеграм-бота. Визначено головну мету та задачі, які має виконувати бот. Було встановлено основні функціональні та нефункціональні вимоги, а також сформовано користувацькі історії з відповідними сценаріями дій. Крім того, була розроблена діаграма варіантів використання телеграм-бота та макети основних сценаріїв взаємодії користувача з ботом.

У другому розділі було здійснено програмну реалізацію телеграм-бота. Розглянуто структуру серверного програмного проекту, визначено основні компоненти та класи, з яких складається бот. Було проведено функціональне тестування, яке підтвердило успішність реалізації. Також створено інструкцію користувача, що включає множину знімків екрана та

пояснювальний текст. У результаті реалізації надано вихідний код розробленого телеграм-бота.

Цей систематичний підхід до розробки та тестування забезпечив створення надійного та функціонального телеграм-бота, що відповідає всім встановленим вимогам та забезпечує високу якість користувацького досвіду.

Перелік використаних джерел

1. Лекційний матеріал та записи пар з курсу ООП.
2. Spring. URL: https://spring.io
3. Фреймворк Spring та його особливості. URL: https://highload.today/uk/frejmvork-spring-ta-jogo-osoblivosti/
4. Spring Framework пояснення за 5 хвилин. URL: https://techukraine.net/spring-framework-пояснюється-за-5-хвилин-або-менше/
5. Telegram Bot API. URL: https://core.telegram.org/bots/api
6. Bots: An introduction for developers. URL: https://core.telegram.org/bots
7. Introducing Bot API 2.0. URL: https://core.telegram.org/bots/2-0-intro
8. Порівнюємо Spring Data JDBC та JPA. URL: https://dou.ua/forums/topic/36109/
9. Будуємо телеграм чат-бот на Java: від ідеї до деплою. URL: https://dou.ua/forums/topic/38358/
10. Telegram Bot: How to Build a Telegram Chatbot with Java. URL: https://www.baeldung.com/spring-telegram-bot
11. Java Concurrency in Practice by Brian Goetz. URL: https://www.oreilly.com/library/view/java-concurrency-in

Додаток

КОД ПРОГРАМИ

Клас TelegramBot.java:

```
package com.example.Shopbot;
package com.example.Shopbot;

import com.example.Shopbot.config.BotConfig;
import com.example.Shopbot.model.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.bots.TelegramLongPollingBot;
import org.telegram.telegrambots.meta.api.methods.ParseMode;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.methods.send.SendPhoto;
import org.telegram.telegrambots.meta.api.objects.InputFile;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.InlineKeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;


import java.util.ArrayList;
import java.util.List;
import java.util.Objects;


@Component
public class TelegramBot extends TelegramLongPollingBot {


   public static final String START = "Старт";
   public static final String MAIN_MENU = "Головне меню";
   public static final String CATEGORY_OF_DISHES = "Категорії продуктів";
   public static final String CART = "Кошик";
   public static final String ALL_DISHES = "Все меню";
   public static final String MAKE_ORDER = "Оформити замовлення";
   public static final String CART_CHANGES = "Зміни у кошику";
   public static final String REMOVE_FROM_CART = "Видалення з кошику";
   public static final String CHANGE_NUMBER_OF_DISHES = "Змінити кількість
продуктів";
   public static final String ADDED_TO_CART = "Додано в кошик";
```

```
    public static final String CART_REVIEW = "Перегляд кошику";
    public static final String ADD_TO_CART = "Додавання до кошику";
    public static final String DELIVERY_ADDRESS = "Вказати адресу доставки";
    public static final String PHONE_NUMBER = "Вказати номер телефону";
    public static final String PAYMENT_METHOD = "Вказати спосіб оплати";
    public static final String ADDITIONAL_INFO = "Вказати додаткову інформацію";
    public static final String CONFIRM_ORDER = "Підтвердити замовлення";
    public static final String PREVIOUS_ORDERS_REVIEW = "Перегляд минулих
замовлень";


    @Autowired
    private final ProductsRepository productsRepository;
    @Autowired
    private final OrdersRepository ordersRepository;
    Cart cart;
    Order order;
    private final BotConfig botConfig;
    private String state;


    public TelegramBot(ProductsRepository productsRepository, OrdersRepository
ordersRepository, Cart cart, BotConfig botConfig) {
        this.productsRepository = productsRepository;
        this.ordersRepository = ordersRepository;
        this.cart = cart;
        this.order = new Order();
        this.botConfig = botConfig;
        this.state = START;
    }


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


    @Override
    public String getBotToken() {
        return botConfig.getToken();
    }


    @Override
    public void onUpdateReceived(Update update) {
        if (update.hasMessage() && update.getMessage().hasText()) {
            messageHandler(update);
```

```
    } else if (update.hasCallbackQuery()) {
        callbackQueryHandler(update);
    }
  }


  private void messageHandler(Update update) {
    String textFromUser = update.getMessage().getText();
    Long chatId = update.getMessage().getChatId();
    String userFirstName = update.getMessage().getFrom().getFirstName();


    if (textFromUser.equals("/start") || textFromUser.equals("Старт")) {
        cart.setChatId(chatId);
        state = MAIN_MENU;
        String message = "Вітаю, " + userFirstName + "!\uD83D\uDC4B" + "\n" +
                "Тут Ви можете замовити доставку продуктів здорового харчування!" + "\n" +
                "Обирайте дію з меню нижче⬇";
        sendMessage(chatId, message);
    } else if (textFromUser.equals("Меню") || textFromUser.equals("Повернутися до вибору
категорій продуктів")) {
        state = CATEGORY_OF_DISHES;
        sendMessage(chatId, "Оберіть категорію продуктів для перегляду: ");
    } else if (textFromUser.equals("Кошик") || textFromUser.equals("Переглянути кошик")) {
        state = CART;
        sendMessage(chatId, cart.getCartInfo());
    } else if (getDishCategories().contains(textFromUser)) {
        state = textFromUser;
        sendMessage(chatId, "Для отримання інформації про продукт оберіть його з
меню");
    } else if (getDishNames(getCategoryOfDish(textFromUser)).contains(textFromUser)) {
        sendDishInfo(chatId, textFromUser);
    } else if (textFromUser.equals("Повернутися в головне меню")) {
        state = MAIN_MENU;
        sendMessage(chatId, "Повернулися в головне меню. \nЩо бажаєте зробити?");
    } else if (textFromUser.equals("Все меню")) {
        state = ALL_DISHES;
        sendMessage(chatId, "Для отримання інформації про продукт оберіть його з
меню");
    } else if (textFromUser.equals("Оформити замовлення")) {
        if (cart.isEmpty()) {
            sendMessage(chatId, "Кошик порожній, тому оформити замовлення
неможливо!");
        } else {
            state = MAKE_ORDER;
            sendMessage(chatId, "Введіть ім'я одержувача замовлення або виберіть серед
запропонованих варіантів:");
        }
```

```
    } else if (state.equals(MAKE_ORDER)) {
        order.setChatId(chatId);
        order.setUserName(textFromUser);
        order.setOrderList(cart.getOrderList());
        order.setOrderPrice(cart.getTotalPrice());
        state = DELIVERY_ADDRESS;
        sendMessage(chatId, "Вкажіть адресу доставки або виберіть серед
запропонованих варіантів:");
    } else if (state.equals(DELIVERY_ADDRESS)) {
        order.setDeliveryAddress(textFromUser);
        state = PHONE_NUMBER;
        sendMessage(chatId, "Вкажіть номер телефону для зв'язку або виберіть серед
запропонованих варіантів:");
    } else if (state.equals(PHONE_NUMBER)) {
        order.setPhoneNumber(Long.parseLong(textFromUser));
        state = PAYMENT_METHOD;
        sendMessage(chatId, "Оберіть спосіб оплати замовлення:");
    } else if (state.equals(ADDITIONAL_INFO)) {
        order.setAdditionalInfo(textFromUser);
        state = CONFIRM_ORDER;
        sendMessage(chatId, getOrderInfo(order));
    } else if (textFromUser.equals("Внести зміни до кошику")) {
        if (cart.isEmpty()) {
            sendMessage(chatId, "Кошик порожній, тому внесення змін неможливе!");
        } else {
            state = CART_CHANGES;
            sendMessage(chatId, "Що бажаєте зробити?");
        }
    } else if (textFromUser.equals("Видалити продукт з кошику")) {
        state = REMOVE_FROM_CART;
        sendMessage(chatId, "Оберіть продукт, який бажаєте видалити:");
    } else if (cart.getDishNamesToChange("Видалення з кошику").contains(textFromUser))
{
        state = CART;
        textFromUser = textFromUser.replaceFirst("Видалити ", "");
        cart.removeAllDishWithName(textFromUser);
        sendMessage(chatId, "Продукт " + textFromUser + " видалено з кошику.");
    } else if (textFromUser.equals("Змінити кількість продуктів")) {
        state = CHANGE_NUMBER_OF_DISHES;
        sendMessage(chatId, "Оберіть продукт, кількість якого бажаєте змінити:");
    } else if (cart.getDishNamesToChange("Змінити кількість
продукта").contains(textFromUser)) {
        state = CART_CHANGES;
        textFromUser = textFromUser.replaceFirst("Змінити ", "");
        sendDishPhoto(chatId, cart.getDishPhoto(textFromUser),
cart.getDishInfo(textFromUser), textFromUser);
    } else if (textFromUser.equals("Очистити кошик")) {
        state = MAIN_MENU;
```

```
            cart.removeAllFromCart();
            sendMessage(chatId, "Всі продукти з кошику було видалено.");
        } else if (textFromUser.equals("Повернутися до перегляду кошика")) {
            state = CART;
            sendMessage(chatId, "Повернулися до перегляду кошика.");
        } else if (textFromUser.equals("Перегляд минулих замовлень")) {
            state = PREVIOUS_ORDERS_REVIEW;
            sendMessage(chatId, "Оберіть замовлення, яке бажаєте переглянути:");
        } else if (state.equals(PREVIOUS_ORDERS_REVIEW) &&
getElementFromOrdersByChatId(chatId).contains(textFromUser)) {
            sendMessage(chatId,
getOrderInfo(Objects.requireNonNull(getOrderById(Long.parseLong(textFromUser)))));
        } else {
            state = MAIN_MENU;
            sendMessage(chatId, "Введена команда не підтримується.");
        }
    }


    private void callbackQueryHandler(Update update) {
        String callbackData = update.getCallbackQuery().getData();
        long chatId = update.getCallbackQuery().getMessage().getChatId();


        if (getDishNames(getCategoryOfDish(callbackData)).contains(callbackData) &&
state.equals(ADD_TO_CART)) {
            cart.addDishToCart(callbackData);
            state = ADDED_TO_CART;
            sendMessage(chatId, "Продукт " + callbackData + " був доданий до кошику!");
        } else if (callbackData.equals("Кошик") && (state.equals(ADDED_TO_CART) ||
state.equals(CART_REVIEW))) {
            state = CART;
            sendMessage(chatId, cart.getCartInfo());
        } else if (cart.getDishNamesToChange("Decrease").contains(callbackData) &&
state.equals(CART_CHANGES)) {
            callbackData = callbackData.replaceFirst("Decrease ", "");
            cart.removeDish(callbackData);
            state = CART_REVIEW;
            sendMessage(chatId, "Кількість була зменшена!");
        } else if (cart.getDishNamesToChange("Increase").contains(callbackData) &&
state.equals(CART_CHANGES)) {
            callbackData = callbackData.replaceFirst("Increase ", "");
            cart.addDishToCart(callbackData);
            state = ADDED_TO_CART;
            sendMessage(chatId, "Кількість була збільшена!");
        } else if ((callbackData.equals("Карта") || callbackData.equals("Готівка")) &&
state.equals(PAYMENT_METHOD)) {
            order.setPaymentMethod(callbackData);
```

```
            state = ADDITIONAL_INFO;
            sendMessage(chatId, "Вкажіть додаткову інформацію до замовлення:");
        } else if (callbackData.equals("Пропустити") && state.equals(ADDITIONAL_INFO)) {
            state = CONFIRM_ORDER;
            sendMessage(chatId, getOrderInfo(order));
        } else if (callbackData.equals("Оформити замовлення") &&
state.equals(CONFIRM_ORDER)) {
            order.setStatus("В обробці");
            ordersRepository.save(order);
            String message = "Замовлення №" + order.getId() + " оформлено успішно!";
            cart.removeAllFromCart();
            order = new Order();
            state = MAIN_MENU;
            sendMessage(chatId, message);
        }
    }


    private void sendMessage(Long chatId, String textToSend) {
        SendMessage message = new SendMessage();
        message.setChatId(String.valueOf(chatId));
        message.setText(textToSend);
        message.setParseMode(ParseMode.HTML);


        keyboardBuilder(message);


        try {
            execute(message);
        } catch (TelegramApiException e) {
            System.out.println("Error while sending message!");
        }
    }


    private void sendDishPhoto(Long chatId,String imageToSend, String textToSend, String
dishName) {
        SendPhoto photo = new SendPhoto();
        photo.setChatId(String.valueOf(chatId));
        photo.setPhoto(new InputFile(imageToSend));
        photo.setParseMode(ParseMode.HTML);
        photo.setCaption(textToSend);


        if (state.equals(ADD_TO_CART)) {
            photo.setReplyMarkup(inlineKeyboardBuilder("Додати до кошику", dishName));
        } else if (state.equals(CART_CHANGES)) {
```

```
            photo.setReplyMarkup(inlineKeyboardBuilder("-", "Decrease " + dishName, "+",
"Increase " + dishName));
        }


        try {
            execute(photo);
        } catch (TelegramApiException e) {
            System.out.println("Error while sending photo!");
        }
    }


    private InlineKeyboardMarkup inlineKeyboardBuilder(String ButtonText, String
callbackData) {
        InlineKeyboardMarkup inlineMarkup = new InlineKeyboardMarkup();
        List<List<InlineKeyboardButton>> inlineRows = new ArrayList<>();
        List<InlineKeyboardButton> inlineRow = new ArrayList<>();


        InlineKeyboardButton inlineButton = new InlineKeyboardButton();


        inlineButton.setText(ButtonText);
        inlineButton.setCallbackData(callbackData);
        inlineRow.add(inlineButton);


        inlineRows.add(inlineRow);
        inlineMarkup.setKeyboard(inlineRows);


        return inlineMarkup;
    }


    private InlineKeyboardMarkup inlineKeyboardBuilder(String firstButtonText, String
firstButtonCallbackData, String secondButtonText, String secondButtonCallbackData) {
        InlineKeyboardMarkup inlineMarkup = new InlineKeyboardMarkup();
        List<List<InlineKeyboardButton>> inlineRows = new ArrayList<>();
        List<InlineKeyboardButton> inlineRow = new ArrayList<>();


        InlineKeyboardButton firstButton = new InlineKeyboardButton();
        firstButton.setText(firstButtonText);
        firstButton.setCallbackData(firstButtonCallbackData);
        inlineRow.add(firstButton);
```

```
    InlineKeyboardButton secondButton = new InlineKeyboardButton();
    secondButton.setText(secondButtonText);
    secondButton.setCallbackData(secondButtonCallbackData);
    inlineRow.add(secondButton);


    inlineRows.add(inlineRow);
    inlineMarkup.setKeyboard(inlineRows);


    return inlineMarkup;
}


private void keyboardBuilder(SendMessage message) {
    List<String> buttons = new ArrayList<>();


    if (state.equals(START)) {
        buttons.add("Старт");
        message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
    } else if (state.equals(MAIN_MENU)) {
        buttons.add("Меню");
        buttons.add("Кошик");
        buttons.add("Перегляд минулих замовлень");
        message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
    } else if (state.equals(CATEGORY_OF_DISHES)) {
        buttons.add("Фрукти");
        buttons.add("Овочі");
        buttons.add("Крупи");
        buttons.add("Випічка");
        buttons.add("Батончики");
        buttons.add("Салати");
        buttons.add("Напої");
        buttons.add("Десерти без цукру");
        buttons.add("Все меню");
        buttons.add("Повернутися в головне меню");
        message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
    } else if (getDishCategories().contains(state) || state.equals(ALL_DISHES)) {
        buttons = getDishNames(state);
        buttons.add("Повернутися до вибору категорій продуктів");
        message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
    } else if (state.equals(CART)) {
        buttons.add("Оформити замовлення");
        buttons.add("Внести зміни до кошику");
        buttons.add("Переглянути кошик");
        buttons.add("Повернутися в головне меню");
```

```
            message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
        } else if (state.equals(CART_CHANGES)) {
            buttons.add("Змінити кількість продуктів");
            buttons.add("Видалити продукт з кошику");
            buttons.add("Очистити кошик");
            buttons.add("Повернутися до перегляду кошика");
            message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
        } else if (state.equals(REMOVE_FROM_CART) ||
state.equals(CHANGE_NUMBER_OF_DISHES)) {
            buttons = cart.getDishNamesToChange(state);
            buttons.add("Повернутися до перегляду кошика");
            message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
        } else if (state.equals(PREVIOUS_ORDERS_REVIEW) || state.equals(MAKE_ORDER)
|| state.equals(PHONE_NUMBER) || state.equals(DELIVERY_ADDRESS)) {
            buttons = getElementFromOrdersByChatId(Long.parseLong(message.getChatId()));
            buttons.add("Повернутися в головне меню");
            message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
        } else if (state.equals(ADDED_TO_CART) || state.equals(CART_REVIEW)) {
            message.setReplyMarkup(inlineKeyboardBuilder("Переглянути кошик", "Кошик"));
        } else if (state.equals(PAYMENT_METHOD)) {
            message.setReplyMarkup(inlineKeyboardBuilder("Карта", "Карта", "Готівка",
"Готівка"));
        } else if (state.equals(ADDITIONAL_INFO)) {
            message.setReplyMarkup(inlineKeyboardBuilder("Пропустити", "Пропустити"));
        } else if (state.equals(CONFIRM_ORDER)) {
            message.setReplyMarkup(inlineKeyboardBuilder("Оформити замовлення",
"Оформити замовлення"));
        }
    }


    private List<String> getDishNames(String dishCategory) {
        var dishes = productsRepository.findAll();


        List<String> dishNames = new ArrayList<>();


        for (Products dish : dishes) {
            if (dishCategory.equals(dish.getCategory()) || dishCategory.equals(ALL_DISHES))
                dishNames.add((dish.getName()).trim());
        }
        return dishNames;
    }


    private List<String> getDishCategories() {
        var dishes = productsRepository.findAll();
```

```
        List<String> dishCategories = new ArrayList<>();


        for (Products dish : dishes) {
            if (!dishCategories.contains(dish.getCategory()))
                dishCategories.add(dish.getCategory());
        }
        return dishCategories;
    }


    private String getCategoryOfDish(String dishName) {
        Products dish = getDishByName(dishName);


        if (dish != null)
            return dish.getCategory();


        return "none";
    }


    private void sendDishInfo(Long chatId, String dishName) {
        Products dish = getDishByName(dishName);
        String message = "Помилка";


        if (dish == null) {
            sendMessage(chatId, message);
            return;
        }


        if (dish.getCategory().equals("Напої")) {
            message = "<b>" + dish.getName() + " - " + dish.getPrice() + " $</b>\n\n<i>" +
dish.getWeight() + " л, " +
                    dish.getAdditional() + "\n" + dish.getDescription() + "</i>";
        } else {
            message = "<b>" + dish.getName() + " - " + dish.getPrice() + " $</b>\n\n<i>" +
dish.getWeight() + " грам, " +
                    dish.getAdditional() + "\n" + dish.getDescription() + "</i>";
        }


        if (!state.equals(CART_CHANGES))
```

```
        state = ADD_TO_CART;


    sendDishPhoto(chatId, dish.getImage(), message, dishName);
}


private Products getDishByName(String dishName) {
    var dishes = productsRepository.findAll();


    for (Products dish : dishes) {
        if (dishName.equals((dish.getName().trim())))
            return dish;
    }


    return null;
}


private List<Order> getOrdersByChatId(long chatId) {
    var orders = ordersRepository.findAll();


    List<Order> userOrders = new ArrayList<>();


    for (Order order : orders) {
        if (order.getChatId() == chatId)
            userOrders.add(order);
    }


    return userOrders;
}


private Order getOrderById(long orderId) {
    var orders = ordersRepository.findAll();


    for (Order order : orders) {
        if (order.getId() == orderId)
            return order;
    }
```

```
    return null;
}

private List<String> getElementFromOrdersByChatId(long chatId) {
    List<Order> orders = getOrdersByChatId(chatId);

    List<String> result = new ArrayList<>();

    switch (state) {
        case PREVIOUS_ORDERS_REVIEW:
            for (Order order : orders) {
                result.add("" + order.getId());
            }
            break;
        case MAKE_ORDER:
            for (Order order : orders) {
                if (!result.contains(order.getUserName()))
                    result.add(order.getUserName());
            }
            break;
        case PHONE_NUMBER:
            for (Order order : orders) {
                if (!result.contains("" + order.getPhoneNumber()))
                    result.add("" + order.getPhoneNumber());
            }
            break;
        case DELIVERY_ADDRESS:
            for (Order order : orders) {
                if (!result.contains(order.getDeliveryAddress()))
                    result.add(order.getDeliveryAddress());
            }
            break;
    }

    return result;
}

private String getOrderInfo(Order order) {
    String orderInfo;

    if (order.getId() != 0) {
```

```
            orderInfo = "<b>Замовлення №" + order.getId() + "</b>\n\nСтатус: " +
order.getStatus();
        } else {
            orderInfo = "<b>Замовлення:</b>\n";
        }


        orderInfo = orderInfo + "\nОтримувач: " + order.getUserName() +
                "\nНомер телефону: " + order.getPhoneNumber() + "\nАдреса доставки: " +
order.getDeliveryAddress() +
                "\nСпосіб оплати: " + order.getPaymentMethod();


        if (order.getAdditionalInfo() != null) {
            orderInfo = orderInfo + "\nДодаткова інформація: " + order.getAdditionalInfo();
        }
        orderInfo = orderInfo + "\n\n Продукти: \n" + order.getOrderList() + "\n<b>До сплати: " +
order.getOrderPrice() + " $ </b>";


        return orderInfo;
    }


    private ReplyKeyboardMarkup ReplyKeyboardBuilder(List<String> list) {
        ReplyKeyboardMarkup keyboardMarkup = new ReplyKeyboardMarkup();
        List<KeyboardRow> keyboardRows = new ArrayList<>();
        KeyboardRow row = new KeyboardRow();


        for (int i = 0; i < list.size(); i++) {
            row.add(list.get(i));
            if ((1 + i) % 3 == 0 || i == list.size() - 1 || (i == list.size() - 2 &&
!state.equals(MAIN_MENU))) {
                keyboardRows.add(row);
                row = new KeyboardRow();
            }
        }


        keyboardMarkup.setKeyboard(keyboardRows);
        return keyboardMarkup;
    }


}

import com.example.Shopbot.config.BotConfig;
```

```
import com.example.Shopbot.model.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.bots.TelegramLongPollingBot;
import org.telegram.telegrambots.meta.api.methods.ParseMode;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.methods.send.SendPhoto;
import org.telegram.telegrambots.meta.api.objects.InputFile;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.InlineKeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;


import java.util.ArrayList;
import java.util.List;
import java.util.Objects;


@Component
public class TelegramBot extends TelegramLongPollingBot {


   public static final String START = "Старт";
   public static final String MAIN_MENU = "Головне меню";
   public static final String CATEGORY_OF_DISHES = "Категорії продуктів";
   public static final String CART = "Кошик";
   public static final String ALL_DISHES = "Все меню";
   public static final String MAKE_ORDER = "Оформити замовлення";
   public static final String CART_CHANGES = "Зміни у кошику";
   public static final String REMOVE_FROM_CART = "Видалення з кошику";
   public static final String CHANGE_NUMBER_OF_DISHES = "Змінити кількість
продуктів";
   public static final String ADDED_TO_CART = "Додано в кошик";
   public static final String CART_REVIEW = "Перегляд кошику";
   public static final String ADD_TO_CART = "Додавання до кошику";
   public static final String DELIVERY_ADDRESS = "Вказати адресу доставки";
   public static final String PHONE_NUMBER = "Вказати номер телефону";
   public static final String PAYMENT_METHOD = "Вказати спосіб оплати";
   public static final String ADDITIONAL_INFO = "Вказати додаткову інформацію";
   public static final String CONFIRM_ORDER = "Підтвердити замовлення";
   public static final String PREVIOUS_ORDERS_REVIEW = "Перегляд минулих
замовлень";
```

```
    @Autowired
    private final ProductsRepository productsRepository;
    @Autowired
    private final OrdersRepository ordersRepository;
    Cart cart;
    Order order;
    private final BotConfig botConfig;
    private String state;


    public TelegramBot(ProductsRepository productsRepository, OrdersRepository
ordersRepository, Cart cart, BotConfig botConfig) {
        this.productsRepository = productsRepository;
        this.ordersRepository = ordersRepository;
        this.cart = cart;
        this.order = new Order();
        this.botConfig = botConfig;
        this.state = START;
    }


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


    @Override
    public String getBotToken() {
        return botConfig.getToken();
    }


    @Override
    public void onUpdateReceived(Update update) {
        if (update.hasMessage() && update.getMessage().hasText()) {
            messageHandler(update);
        } else if (update.hasCallbackQuery()) {
            callbackQueryHandler(update);
        }
    }


    private void messageHandler(Update update) {
        String textFromUser = update.getMessage().getText();
        Long chatId = update.getMessage().getChatId();
        String userFirstName = update.getMessage().getFrom().getFirstName();
```

```
    if (textFromUser.equals("/start") || textFromUser.equals("Старт")) {
        cart.setChatId(chatId);
        state = MAIN_MENU;
        String message = "Вітаю, " + userFirstName + "!\uD83D\uDC4B" + "\n" +
                "Тут Ви можете замовити доставку продуктів здорового харчування!" + "\n" +
                "Обирайте дію з меню нижче⬇";
        sendMessage(chatId, message);
    } else if (textFromUser.equals("Меню") || textFromUser.equals("Повернутися до вибору
категорій продуктів")) {
        state = CATEGORY_OF_DISHES;
        sendMessage(chatId, "Оберіть категорію продуктів для перегляду: ");
    } else if (textFromUser.equals("Кошик") || textFromUser.equals("Переглянути кошик")) {
        state = CART;
        sendMessage(chatId, cart.getCartInfo());
    } else if (getDishCategories().contains(textFromUser)) {
        state = textFromUser;
        sendMessage(chatId, "Для отримання інформації про продукт оберіть його з
меню");
    } else if (getDishNames(getCategoryOfDish(textFromUser)).contains(textFromUser)) {
        sendDishInfo(chatId, textFromUser);
    } else if (textFromUser.equals("Повернутися в головне меню")) {
        state = MAIN_MENU;
        sendMessage(chatId, "Повернулися в головне меню. \nЩо бажаєте зробити?");
    } else if (textFromUser.equals("Все меню")) {
        state = ALL_DISHES;
        sendMessage(chatId, "Для отримання інформації про продукт оберіть його з
меню");
    } else if (textFromUser.equals("Оформити замовлення")) {
        if (cart.isEmpty()) {
            sendMessage(chatId, "Кошик порожній, тому оформити замовлення
неможливо!");
        } else {
            state = MAKE_ORDER;
            sendMessage(chatId, "Введіть ім'я одержувача замовлення або виберіть серед
запропонованих варіантів:");
        }
    } else if (state.equals(MAKE_ORDER)) {
        order.setChatId(chatId);
        order.setUserName(textFromUser);
        order.setOrderList(cart.getOrderList());
        order.setOrderPrice(cart.getTotalPrice());
        state = DELIVERY_ADDRESS;
        sendMessage(chatId, "Вкажіть адресу доставки або виберіть серед
запропонованих варіантів:");
    } else if (state.equals(DELIVERY_ADDRESS)) {
        order.setDeliveryAddress(textFromUser);
        state = PHONE_NUMBER;
```

```
            sendMessage(chatId, "Вкажіть номер телефону для зв'язку або виберіть серед
запропонованих варіантів:");
        } else if (state.equals(PHONE_NUMBER)) {
            order.setPhoneNumber(Long.parseLong(textFromUser));
            state = PAYMENT_METHOD;
            sendMessage(chatId, "Оберіть спосіб оплати замовлення:");
        } else if (state.equals(ADDITIONAL_INFO)) {
            order.setAdditionalInfo(textFromUser);
            state = CONFIRM_ORDER;
            sendMessage(chatId, getOrderInfo(order));
        } else if (textFromUser.equals("Внести зміни до кошику")) {
            if (cart.isEmpty()) {
                sendMessage(chatId, "Кошик порожній, тому внесення змін неможливе!");
            } else {
                state = CART_CHANGES;
                sendMessage(chatId, "Що бажаєте зробити?");
            }
        } else if (textFromUser.equals("Видалити продукт з кошику")) {
            state = REMOVE_FROM_CART;
            sendMessage(chatId, "Оберіть продукт, який бажаєте видалити:");
        } else if (cart.getDishNamesToChange("Видалення з кошику").contains(textFromUser))
{
            state = CART;
            textFromUser = textFromUser.replaceFirst("Видалити ", "");
            cart.removeAllDishWithName(textFromUser);
            sendMessage(chatId, "Продукт " + textFromUser + " видалено з кошику.");
        } else if (textFromUser.equals("Змінити кількість продуктів")) {
            state = CHANGE_NUMBER_OF_DISHES;
            sendMessage(chatId, "Оберіть продукт, кількість якого бажаєте змінити:");
        } else if (cart.getDishNamesToChange("Змінити кількість
продукта").contains(textFromUser)) {
            state = CART_CHANGES;
            textFromUser = textFromUser.replaceFirst("Змінити ", "");
            sendDishPhoto(chatId, cart.getDishPhoto(textFromUser),
cart.getDishInfo(textFromUser), textFromUser);
        } else if (textFromUser.equals("Очистити кошик")) {
            state = MAIN_MENU;
            cart.removeAllFromCart();
            sendMessage(chatId, "Всі продукти з кошику було видалено.");
        } else if (textFromUser.equals("Повернутися до перегляду кошика")) {
            state = CART;
            sendMessage(chatId, "Повернулися до перегляду кошика.");
        } else if (textFromUser.equals("Перегляд минулих замовлень")) {
            state = PREVIOUS_ORDERS_REVIEW;
            sendMessage(chatId, "Оберіть замовлення, яке бажаєте переглянути:");
        } else if (state.equals(PREVIOUS_ORDERS_REVIEW) &&
getElementFromOrdersByChatId(chatId).contains(textFromUser)) {
```

```
            sendMessage(chatId,
getOrderInfo(Objects.requireNonNull(getOrderById(Long.parseLong(textFromUser)))));
        } else {
            state = MAIN_MENU;
            sendMessage(chatId, "Введена команда не підтримується.");
        }
    }


    private void callbackQueryHandler(Update update) {
        String callbackData = update.getCallbackQuery().getData();
        long chatId = update.getCallbackQuery().getMessage().getChatId();


        if (getDishNames(getCategoryOfDish(callbackData)).contains(callbackData) &&
state.equals(ADD_TO_CART)) {
            cart.addDishToCart(callbackData);
            state = ADDED_TO_CART;
            sendMessage(chatId, "Продукт " + callbackData + " був доданий до кошику!");
        } else if (callbackData.equals("Кошик") && (state.equals(ADDED_TO_CART) ||
state.equals(CART_REVIEW))) {
            state = CART;
            sendMessage(chatId, cart.getCartInfo());
        } else if (cart.getDishNamesToChange("Decrease").contains(callbackData) &&
state.equals(CART_CHANGES)) {
            callbackData = callbackData.replaceFirst("Decrease ", "");
            cart.removeDish(callbackData);
            state = CART_REVIEW;
            sendMessage(chatId, "Кількість була зменшена!");
        } else if (cart.getDishNamesToChange("Increase").contains(callbackData) &&
state.equals(CART_CHANGES)) {
            callbackData = callbackData.replaceFirst("Increase ", "");
            cart.addDishToCart(callbackData);
            state = ADDED_TO_CART;
            sendMessage(chatId, "Кількість була збільшена!");
        } else if ((callbackData.equals("Карта") || callbackData.equals("Готівка")) &&
state.equals(PAYMENT_METHOD)) {
            order.setPaymentMethod(callbackData);
            state = ADDITIONAL_INFO;
            sendMessage(chatId, "Вкажіть додаткову інформацію до замовлення:");
        } else if (callbackData.equals("Пропустити") && state.equals(ADDITIONAL_INFO)) {
            state = CONFIRM_ORDER;
            sendMessage(chatId, getOrderInfo(order));
        } else if (callbackData.equals("Оформити замовлення") &&
state.equals(CONFIRM_ORDER)) {
            order.setStatus("В обробці");
            ordersRepository.save(order);
            String message = "Замовлення №" + order.getId() + " оформлено успішно!";
```

```
            cart.removeAllFromCart();
            order = new Order();
            state = MAIN_MENU;
            sendMessage(chatId, message);
        }
    }


    private void sendMessage(Long chatId, String textToSend) {
        SendMessage message = new SendMessage();
        message.setChatId(String.valueOf(chatId));
        message.setText(textToSend);
        message.setParseMode(ParseMode.HTML);


        keyboardBuilder(message);


        try {
            execute(message);
        } catch (TelegramApiException e) {
            System.out.println("Error while sending message!");
        }
    }


    private void sendDishPhoto(Long chatId,String imageToSend, String textToSend, String
dishName) {
        SendPhoto photo = new SendPhoto();
        photo.setChatId(String.valueOf(chatId));
        photo.setPhoto(new InputFile(imageToSend));
        photo.setParseMode(ParseMode.HTML);
        photo.setCaption(textToSend);


        if (state.equals(ADD_TO_CART)) {
            photo.setReplyMarkup(inlineKeyboardBuilder("Додати до кошику", dishName));
        } else if (state.equals(CART_CHANGES)) {
            photo.setReplyMarkup(inlineKeyboardBuilder("-", "Decrease " + dishName, "+",
"Increase " + dishName));
        }


        try {
            execute(photo);
        } catch (TelegramApiException e) {
            System.out.println("Error while sending photo!");
        }
```

```
  }

  private InlineKeyboardMarkup inlineKeyboardBuilder(String ButtonText, String
callbackData) {
    InlineKeyboardMarkup inlineMarkup = new InlineKeyboardMarkup();
    List<List<InlineKeyboardButton>> inlineRows = new ArrayList<>();
    List<InlineKeyboardButton> inlineRow = new ArrayList<>();

    InlineKeyboardButton inlineButton = new InlineKeyboardButton();

    inlineButton.setText(ButtonText);
    inlineButton.setCallbackData(callbackData);
    inlineRow.add(inlineButton);

    inlineRows.add(inlineRow);
    inlineMarkup.setKeyboard(inlineRows);

    return inlineMarkup;
  }

  private InlineKeyboardMarkup inlineKeyboardBuilder(String firstButtonText, String
firstButtonCallbackData, String secondButtonText, String secondButtonCallbackData) {
    InlineKeyboardMarkup inlineMarkup = new InlineKeyboardMarkup();
    List<List<InlineKeyboardButton>> inlineRows = new ArrayList<>();
    List<InlineKeyboardButton> inlineRow = new ArrayList<>();

    InlineKeyboardButton firstButton = new InlineKeyboardButton();
    firstButton.setText(firstButtonText);
    firstButton.setCallbackData(firstButtonCallbackData);
    inlineRow.add(firstButton);

    InlineKeyboardButton secondButton = new InlineKeyboardButton();
    secondButton.setText(secondButtonText);
    secondButton.setCallbackData(secondButtonCallbackData);
    inlineRow.add(secondButton);

    inlineRows.add(inlineRow);
    inlineMarkup.setKeyboard(inlineRows);
```

```
        return inlineMarkup;
    }


    private void keyboardBuilder(SendMessage message) {
        List<String> buttons = new ArrayList<>();


        if (state.equals(START)) {
            buttons.add("Старт");
            message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
        } else if (state.equals(MAIN_MENU)) {
            buttons.add("Меню");
            buttons.add("Кошик");
            buttons.add("Перегляд минулих замовлень");
            message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
        } else if (state.equals(CATEGORY_OF_DISHES)) {
            buttons.add("Фрукти");
            buttons.add("Овочі");
            buttons.add("Крупи");
            buttons.add("Випічка");
            buttons.add("Батончики");
            buttons.add("Салати");
            buttons.add("Напої");
            buttons.add("Десерти без цукру");
            buttons.add("Все меню");
            buttons.add("Повернутися в головне меню");
            message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
        } else if (getDishCategories().contains(state) || state.equals(ALL_DISHES)) {
            buttons = getDishNames(state);
            buttons.add("Повернутися до вибору категорій продуктів");
            message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
        } else if (state.equals(CART)) {
            buttons.add("Оформити замовлення");
            buttons.add("Внести зміни до кошику");
            buttons.add("Переглянути кошик");
            buttons.add("Повернутися в головне меню");
            message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
        } else if (state.equals(CART_CHANGES)) {
            buttons.add("Змінити кількість продуктів");
            buttons.add("Видалити продукт з кошику");
            buttons.add("Очистити кошик");
            buttons.add("Повернутися до перегляду кошика");
            message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
        } else if (state.equals(REMOVE_FROM_CART) ||
state.equals(CHANGE_NUMBER_OF_DISHES)) {
            buttons = cart.getDishNamesToChange(state);
```

```
            buttons.add("Повернутися до перегляду кошика");
            message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
        } else if (state.equals(PREVIOUS_ORDERS_REVIEW) || state.equals(MAKE_ORDER)
|| state.equals(PHONE_NUMBER) || state.equals(DELIVERY_ADDRESS)) {
            buttons = getElementFromOrdersByChatId(Long.parseLong(message.getChatId()));
            buttons.add("Повернутися в головне меню");
            message.setReplyMarkup(ReplyKeyboardBuilder(buttons));
        } else if (state.equals(ADDED_TO_CART) || state.equals(CART_REVIEW)) {
            message.setReplyMarkup(inlineKeyboardBuilder("Переглянути кошик", "Кошик"));
        } else if (state.equals(PAYMENT_METHOD)) {
            message.setReplyMarkup(inlineKeyboardBuilder("Карта", "Карта", "Готівка",
"Готівка"));
        } else if (state.equals(ADDITIONAL_INFO)) {
            message.setReplyMarkup(inlineKeyboardBuilder("Пропустити", "Пропустити"));
        } else if (state.equals(CONFIRM_ORDER)) {
            message.setReplyMarkup(inlineKeyboardBuilder("Оформити замовлення",
"Оформити замовлення"));
        }
    }


    private List<String> getDishNames(String dishCategory) {
        var dishes = productsRepository.findAll();


        List<String> dishNames = new ArrayList<>();


        for (Products dish : dishes) {
            if (dishCategory.equals(dish.getCategory()) || dishCategory.equals(ALL_DISHES))
                dishNames.add((dish.getName()).trim());
        }
        return dishNames;
    }


    private List<String> getDishCategories() {
        var dishes = productsRepository.findAll();


        List<String> dishCategories = new ArrayList<>();


        for (Products dish : dishes) {
            if (!dishCategories.contains(dish.getCategory()))
                dishCategories.add(dish.getCategory());
        }
        return dishCategories;
```

```
    }

    private String getCategoryOfDish(String dishName) {
        Products dish = getDishByName(dishName);

        if (dish != null)
            return dish.getCategory();

        return "none";
    }

    private void sendDishInfo(Long chatId, String dishName) {
        Products dish = getDishByName(dishName);
        String message = "Помилка";

        if (dish == null) {
            sendMessage(chatId, message);
            return;
        }

        if (dish.getCategory().equals("Напої")) {
            message = "<b>" + dish.getName() + " - " + dish.getPrice() + " $</b>\n\n<i>" +
dish.getWeight() + " л, " +
                    dish.getAdditional() + "\n" + dish.getDescription() + "</i>";
        } else {
            message = "<b>" + dish.getName() + " - " + dish.getPrice() + " $</b>\n\n<i>" +
dish.getWeight() + " грам, " +
                    dish.getAdditional() + "\n" + dish.getDescription() + "</i>";
        }

        if (!state.equals(CART_CHANGES))
            state = ADD_TO_CART;

        sendDishPhoto(chatId, dish.getImage(), message, dishName);
    }

    private Products getDishByName(String dishName) {
        var dishes = productsRepository.findAll();
```

```
    for (Products dish : dishes) {
        if (dishName.equals((dish.getName().trim())))
            return dish;
    }


    return null;
}


private List<Order> getOrdersByChatId(long chatId) {
    var orders = ordersRepository.findAll();


    List<Order> userOrders = new ArrayList<>();


    for (Order order : orders) {
        if (order.getChatId() == chatId)
            userOrders.add(order);
    }


    return userOrders;
}


private Order getOrderById(long orderId) {
    var orders = ordersRepository.findAll();


    for (Order order : orders) {
        if (order.getId() == orderId)
            return order;
    }


    return null;
}


private List<String> getElementFromOrdersByChatId(long chatId) {
    List<Order> orders = getOrdersByChatId(chatId);


    List<String> result = new ArrayList<>();
```

```
        switch (state) {
            case PREVIOUS_ORDERS_REVIEW:
                for (Order order : orders) {
                    result.add("" + order.getId());
                }
                break;
            case MAKE_ORDER:
                for (Order order : orders) {
                    if (!result.contains(order.getUserName()))
                        result.add(order.getUserName());
                }
                break;
            case PHONE_NUMBER:
                for (Order order : orders) {
                    if (!result.contains("" + order.getPhoneNumber()))
                        result.add("" + order.getPhoneNumber());
                }
                break;
            case DELIVERY_ADDRESS:
                for (Order order : orders) {
                    if (!result.contains(order.getDeliveryAddress()))
                        result.add(order.getDeliveryAddress());
                }
                break;
        }


        return result;
    }


    private String getOrderInfo(Order order) {
        String orderInfo;


        if (order.getId() != 0) {
            orderInfo = "<b>Замовлення №" + order.getId() + "</b>\n\nСтатус: " +
order.getStatus();
        } else {
            orderInfo = "<b>Замовлення:</b>\n";
        }


        orderInfo = orderInfo + "\nОтримувач: " + order.getUserName() +
                "\nНомер телефону: " + order.getPhoneNumber() + "\nАдреса доставки: " +
order.getDeliveryAddress() +
                "\nСпосіб оплати: " + order.getPaymentMethod();
```

```
        if (order.getAdditionalInfo() != null) {
            orderInfo = orderInfo + "\nДодаткова інформація: " + order.getAdditionalInfo();
        }
        orderInfo = orderInfo + "\n\n Продукти: \n" + order.getOrderList() + "\n<b>До сплати: " +
order.getOrderPrice() + " $ </b>";


        return orderInfo;
    }


    private ReplyKeyboardMarkup ReplyKeyboardBuilder(List<String> list) {
        ReplyKeyboardMarkup keyboardMarkup = new ReplyKeyboardMarkup();
        List<KeyboardRow> keyboardRows = new ArrayList<>();
        KeyboardRow row = new KeyboardRow();


        for (int i = 0; i < list.size(); i++) {
            row.add(list.get(i));
            if ((1 + i) % 3 == 0 || i == list.size() - 1 || (i == list.size() - 2 &&
!state.equals(MAIN_MENU))) {
                keyboardRows.add(row);
                row = new KeyboardRow();
            }
        }


        keyboardMarkup.setKeyboard(keyboardRows);
        return keyboardMarkup;
    }


}
```

Клас Cart.java:

```
package com.example.Shopbot.model;

import lombok.Getter;
import lombok.Setter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;
```

```
@Getter
@Setter
@Component
public class Cart {

  @Autowired
  private ProductsRepository productsRepository;

  private long chatId;
  private List<Products> dishes;

  public Cart(ProductsRepository productsRepository) {
     this.productsRepository = productsRepository;
     this.dishes = new ArrayList<>();
  }

  public void addDishToCart(String dishName) {
     if (findDishInMenu(dishName) != null)
        dishes.add(findDishInMenu(dishName));
  }

  public Products findDishInMenu(String dishName) {
     var dishes = productsRepository.findAll();

     for (Products dish : dishes) {
        if (dishName.equals(dish.getName()))
           return dish;
     }
     return null;
  }

  public float getTotalPrice() {
     float totalPrice = 0;

     if (!dishes.isEmpty()) {
        for (Products dish : dishes) {
           totalPrice += dish.getPrice();
        }
     }

     return totalPrice;
  }

  public String getCartInfo() {
     StringBuilder message = new StringBuilder("Кошик порожній");

     if (!dishes.isEmpty()) {
```

```
            message = new StringBuilder("<b>Кошик:</b>\n\n" + getOrderList() + "\n<b> До
сплати: " + getTotalPrice() + " $</b>");
        }

        return message.toString();
    }

    public int getDishQuantity(Products dish) {
        int dishQuantity = 0;

        for (Products value : dishes) {
            if (value.getId() == dish.getId())
                dishQuantity++;
        }

        return dishQuantity;
    }

    public void removeAllDishWithName(String dishName) {
        if (findDishInMenu(dishName) != null && !dishes.isEmpty()) {
            dishes.removeIf(dish -> dish.getName().equals(dishName));
        }
    }

    public List<String> getDishNamesToChange(String change) {
        List<String> dishNames = new ArrayList<>();

        switch (change) {
            case "Видалення з кошику" -> change = "Видалити ";
            case "Змінити кількість продуктів" -> change = "Змінити ";
            case "Decrease" -> change = "Decrease ";
            case "Increase" -> change = "Increase ";
        }

        for (Products dish : getDishesWithoutRepeat()) {
            dishNames.add(change + dish.getName());
        }
        return dishNames;
    }

    public List<Products> getDishesWithoutRepeat() {
        List<Products> uniqueDishes = new ArrayList<>();
        for (Products dish : dishes) {
            if (!uniqueDishes.contains(dish)) {
                uniqueDishes.add(dish);
            }
        }
        return uniqueDishes;
```

```
    }

    public Products getDishFromCart(String dishName) {
        List<Products> dishes = getDishesWithoutRepeat();

        for (Products dish : dishes) {
            if (dish.getName().equals(dishName))
                return dish;
        }

        return null;
    }

    public String getDishInfo(String dishName) {
        Products dish = getDishFromCart(dishName);
        String message = "Помилка";

        if (dish == null) {
            return message;
        }

        if (dish.getCategory().equals("Напої")) {
            message = "<b>" + dish.getName() + " - " + dish.getPrice() + " $ - " +
getDishQuantity(dish) + " шт.</b>\n\n<i>" +
                    dish.getWeight() + " л, " + dish.getAdditional() + "\n" + dish.getDescription() +
"</i>";
        } else {
            message = "<b>" + dish.getName() + " - " + dish.getPrice() + " $ - " +
getDishQuantity(dish) + " шт.</b>\n\n<i>" +
                    dish.getWeight() + " грам, " + dish.getAdditional() + "\n" + dish.getDescription() +
"</i>";
        }

        return message;
    }

    public String getDishPhoto(String dishName) {
        Products dish = getDishFromCart(dishName);

        if (dish == null)
            return null;

        return dish.getImage();
    }

    public void removeDish(String dishName) {
        if (findDishInMenu(dishName) != null && !dishes.isEmpty()) {
            for (int i = dishes.size() - 1; i >= 0; i--) {
```

```
            if (dishes.get(i).getName().equals(dishName)) {
                dishes.remove(i);
                return;
            }
        }
    }
  }

  public void removeAllFromCart() {
    dishes.clear();
  }

  public boolean isEmpty() {
    return dishes.isEmpty();
  }

  public String getOrderList() {
    StringBuilder message = new StringBuilder();

    if (!dishes.isEmpty()) {

      List<Products> dishesWithoutRepeat = new ArrayList<>(getDishesWithoutRepeat());

      for (Products dish : dishesWithoutRepeat) {
        message.append(dishesWithoutRepeat.indexOf(dish) + 1).append(".
").append(dish.getName()).append(" - ").append(dish.getPrice()).append(" $ -
").append(getDishQuantity(dish)).append(" шт.\n");
      }
    }

    return message.toString();
  }
}
```

Клас Products.java

```
package com.example.Shopbot.model;


import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@Entity(name = "product")
public class Products {
```

```
    @Id
    private int id;

    private String category;

    private String name;

    private String description;

    private String additional;

    private float weight;

    private float price;

    private String image;


    @Override
    public boolean equals(Object o) {
        if (this == o)
            return true;
        if (o == null || getClass() != o.getClass())
            return false;

        Products dish = (Products) o;

        return id == dish.getId() && name.equals(dish.getName());
    }
}
```

Клас Order.java:

```
package com.example.Shopbot.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@Entity(name = "orders")
public class Order {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private long id;

    private long chatId;

    private String userName;

    private String status;

    private String orderList;

    private float orderPrice;

    private long phoneNumber;

    private String deliveryAddress;

    private String paymentMethod;

    private String additionalInfo;

}
```

Інтерфейс ProductsRepository.java:

```
package com.example.Shopbot.model;

import org.springframework.data.repository.CrudRepository;

public interface ProductsRepository extends CrudRepository<Products, Integer> {
}
```

Інтерфейс OrdersRepository.java:

```
package com.example.Shopbot.model;

import org.springframework.data.repository.CrudRepository;

public interface OrdersRepository extends CrudRepository<Order, Integer> {

}
```

Клас BotConfig.java:

```
package com.example.Shopbot.config;

import lombok.Data;
```

```
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

@Configuration
@Data
@PropertySource("application.properties")
public class BotConfig {

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

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

}
```

Клас BotInitializer.java:

```
package com.example.Shopbot.config;

import com.example.Shopbot.TelegramBot;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.event.ContextRefreshedEvent;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.TelegramBotsApi;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;
import org.telegram.telegrambots.updatesreceivers.DefaultBotSession;

@Component
public class BotInitializer {
  private final TelegramBot telegramBot;

  @Autowired
  public BotInitializer(TelegramBot telegramBot) {
    this.telegramBot = telegramBot;
  }

  @EventListener({ContextRefreshedEvent.class})
  public void init()throws TelegramApiException {
    TelegramBotsApi telegramBotsApi = new TelegramBotsApi(DefaultBotSession.class);
    try{
      telegramBotsApi.registerBot(telegramBot);
    } catch (TelegramApiException e){
      System.out.println("Error while registering Bot!");
    }
```

```
  }
}
```

Клас ShopbotApplication:

```
package com.example.Shopbot.config;

import com.example.Shopbot.TelegramBot;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.event.ContextRefreshedEvent;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.TelegramBotsApi;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;
import org.telegram.telegrambots.updatesreceivers.DefaultBotSession;

@Component
public class BotInitializer {
  private final TelegramBot telegramBot;

  @Autowired
  public BotInitializer(TelegramBot telegramBot) {
    this.telegramBot = telegramBot;
  }

  @EventListener({ContextRefreshedEvent.class})
  public void init()throws TelegramApiException {
    TelegramBotsApi telegramBotsApi = new TelegramBotsApi(DefaultBotSession.class);
    try{
      telegramBotsApi.registerBot(telegramBot);
    } catch (TelegramApiException e){
      System.out.println("Error while registering Bot!");
    }
  }
}
```

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
```

```
  <version>3.2.5</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.example</groupId>
<artifactId>Shopbot</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>Shopbot</name>
<description>Demo project for Spring Boot</description>
<properties>
  <java.version>21</java.version>
  <telemetry.version>5.6.0</telemetry.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.telegram</groupId>
    <artifactId>telegrambots</artifactId>
    <version>${telemetry.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>

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

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

  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
  </dependency>

  <dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
```

```
      <version>2.3.1</version>
    </dependency>

    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <version>42.6.2</version>
    </dependency>

  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

application.properties:

```
bot.name=Store_cursovaya_bot
bot.token=6786755502:AAEaVbsBzMX4_5dufZqFpeqXXQDJysnEHRI

spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3307/tg-bot
spring.datasource.username=root
spring.datasource.password=12345
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.show-sql=true
```