Міністерство освіти і науки
України Національний
технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(навчально-науковий інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня бакалавра

(бакалавра, магістра)

Здобувача вищої освіти Твердохліб Максім Ігорович
(ПІБ)

академічної групи _____
(шифр)

спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)

спеціалізації за освітньо-професійною (освітньо-науковою) програмою _____
(за наявності)

_____
(офіційна назва)

на тему Розробка чат-бота для підтримки роботи аптечного складу

_____
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Коротенок Г.М.			
розділів:				

Рецензент	доц. Ширін А.Л.			

Нормоконтролер	проф. Коротенко Г.М			

Дніпро
2025

ЗАТВЕРДЖЕНО:
завідувач кафедри

_____
(повна назва)

_____ _____
(підпис) (ініціали та прізвище)

«_____» _____2025року

ЗАВДАННЯ

на кваліфікаційну роботу
ступеня_____

(бакалавра, магістра)

здобувача вищої освіти Твердохліб М.І. **академічної групи** 126-21-2
(прізвище та ініціали) (шифр)

спеціальності_____

спеціалізації за освітньою-професійною програмою _____
(за наявності)

_____

на тему _Розробка чат-бота для підтримки роботи аптечного складу_

_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 05.05.2025 р. № 336-с

Розділ	Зміст	Термін виконання
1	Аналіз стану області рішення завдання	
2	Проєктні рішення	
3	Тестування та результати	

Завдання видано _____ _____
_____(підпис керівника) (ініціали та прізвище)

Дата видачі _____

Дата подання до екзаменаційної комісії_____

Прийнято до виконання _____ _____
(підпис здобувача вищої освіти) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 113 с., 34 рис., 9 табл., 1 додаток, 32 джерел.

Ключові слова: АВТОМАТИЗАЦІЯ, АПТЕЧНИЙ СКЛАД, БАЗА ДАНИХ, БЕЗПЕКА, КОШИК, ТЕЛЕГРАМ-БОТ, УПРАВЛІННЯ ЗАМОВЛЕННЯМИ, ФАРМАЦЕВТИЧНА ЛОГІСТИКА, FLASK, REST API, SQLITE, TELEGRAM API.

Об'єкт: чат-бот для аптечного складу.

Предмет: автоматизація складських процесів і комунікації через Telegram.

Мета: розробка чат-бота для оптимізації управління запасами, обробки замовлень і взаємодії клієнтів та менеджерів.

У вступі обґрунтовано актуальність автоматизації аптечних складів, проаналізовано аналоги (Walgreens, CVS Health, Boots UK, MedAdvisor, Tabletka.ua), виявлено їх недоліки: обмежена інтеграція з WMS і висока вартість. У першому розділі розглянуто сучасні чат-боти, їх переваги, недоліки та сформульовано вимоги до функціоналу: авторизація, каталог, кошик, замовлення, комунікація. У другому розділі описано технології (Python, Flask, SQLite, pyTelegramBotAPI), трирівневу архітектуру та реалізацію функцій (аутентифікація, управління товарами, замовленнями, повідомленнями). У третьому розділі представлено тестування (функціональне, інтеграційне, безпекове, навантажувальне), що підтвердило стабільність системи.

Висновки: чат-бот автоматизує ключові процеси складу, підвищує ефективність і якість обслуговування. Рекомендації: посилити безпеку (сіль для хешування, захист від CSRF), оптимізувати обробку помилок Telegram API, замінити SQLite на PostgreSQL.

Практичне значення: рішення підходить для малих і середніх аптечних складів, зменшує помилки та покращує логістику.

ABSTRACT

Explanatory Note: 113 pages, 34 figures, 9 tables, 1 appendix, 32 references.

Keywords: AUTOMATION, PHARMACY WAREHOUSE, DATABASE, SECURITY, CART, TELEGRAM BOT, ORDER MANAGEMENT, PHARMACEUTICAL LOGISTICS, FLASK, REST API, SQLITE, TELEGRAM API.

Object: Chatbot for pharmacy warehouse.

Subject: Automation of warehouse processes and client communication via Telegram.

Goal: Develop a chatbot to optimize inventory, order processing, and client-manager interaction.

The introduction justifies automation needs, analyzes analogs (Walgreens, CVS Health, Boots UK, MedAdvisor, Tabletka.ua), noting their limited WMS integration and high costs. Chapter 1 reviews pharmacy chatbots, their strengths, weaknesses, and defines requirements: authorization, catalog, cart, orders, communication. Chapter 2 details technologies (Python, Flask, SQLite, pyTelegramBotAPI), three-tier architecture, and functions (authentication, product/order management, messaging). Chapter 3 presents testing (functional, integration, security, load), confirming system stability.

Conclusions: The chatbot automates warehouse processes, enhancing efficiency and service quality. Recommendations: improve security (salting, CSRF protection), optimize Telegram API error handling, replace SQLite with PostgreSQL.

Practical Significance: Suitable for small/medium warehouses, reducing errors and improving logistics.

Зміст

ВСТУП

Сучасний розвиток інформаційних технологій відкриває нові можливості для автоматизації та оптимізації бізнес-процесів у різних сферах, зокрема в аптечній діяльності. Аптечний склад є ключовим елементом фармацевтичної логістики, що забезпечує зберігання, облік та дистрибуцію лікарських засобів. Ефективне управління складськими процесами, взаємодія з клієнтами та швидке реагування на їх запити є важливими завданнями для фахівців у галузі інформаційних технологій та фармацевтики. Розробка чат-бота для підтримки роботи аптечного складу, представлена у реалізованому коді, безпосередньо пов’язана з діяльністю фахівця напряму інформаційних технологій, оскільки передбачає створення програмного забезпечення для автоматизації бізнес-процесів, управління даними та забезпечення зручної взаємодії з користувачами.

На сучасному етапі розвитку фармацевтичної галузі спостерігається зростання попиту на цифрові рішення, які підвищують ефективність роботи аптек та складів. Аналіз аналогів показує, що існують системи управління складом (WMS), CRM-системи для роботи з клієнтами та чат-боти для автоматизації комунікацій. Проте більшість таких рішень або є надто універсальними, не враховуючи специфіки аптечної діяльності, або потребують значних фінансових і технічних ресурсів для впровадження. Наприклад, комерційні платформи, такі як SAP або Oracle NetSuite, пропонують комплексні рішення, але їх складність та висока вартість роблять їх недоступними для малих і середніх аптечних складів. Водночас, відкриті рішення, такі як чат-боти на базі Telegram, мають обмежений функціонал і не завжди інтегруються з базами даних складу. Технічні протиріччя виникають між необхідністю забезпечення простоти використання системи для клієнтів і менеджерів та потребою в складній серверній логіці для обробки замовлень, управління запасами та комунікацій. Прогалини у знаннях стосуються недостатньої кількості досліджень щодо оптимальної інтеграції чат-ботів із

системами управління аптечними складами, а також відсутності універсальних підходів до адаптації таких систем під локальні вимоги.

Метою роботи є розробка чат-бота для підтримки роботи аптечного складу, який забезпечує автоматизацію процесів управління запасами, обробки замовлень, комунікації з клієнтами та менеджерами через Telegram-інтерфейс. Реалізація цього завдання передбачає створення серверної частини на базі Flask, бази даних SQLite для зберігання інформації про товари, замовлення та користувачів, а також клієнтської частини у вигляді Telegram-бота з інтуїтивно зрозумілим інтерфейсом.

Актуальність теми обґрунтовується зростаючим попитом на автоматизацію в фармацевтичній галузі, що дозволяє підвищити швидкість обробки запитів, зменшити кількість помилок і покращити якість обслуговування клієнтів. Чат-боти, як інструмент автоматизації, є економічно вигідним рішенням, оскільки вони не потребують значних витрат на апаратне забезпечення та можуть бути розгорнуті на платформах, які вже використовуються користувачами, таких як Telegram. Крім того, інтеграція чат-бота з базою даних складу дозволяє забезпечити реальний час обробки інформації, що є критично важливим для аптечних складів, де актуальність даних про наявність товарів безпосередньо впливає на якість обслуговування.

У рамках роботи вирішуються такі завдання: розробка структури бази даних для зберігання інформації про товари, категорії, замовлення, користувачів і повідомлення; створення серверної логіки для обробки запитів через REST API; реалізація Telegram-бота з підтримкою авторизації, перегляду каталогу, управління кошиком, оформлення замовлень і комунікації між клієнтами та менеджерами; забезпечення безпеки даних шляхом хешування паролів і використання сесій. Код проєкту є результатом реалізації цих завдань і демонструє функціональну систему, яка може бути адаптована для реального використання в аптечних складах.

Таким чином, розробка чат-бота для аптечного складу є актуальним і практично значущим завданням, яке сприяє автоматизації бізнес-процесів,

підвищенню ефективності управління складом і покращенню взаємодії з клієнтами. Реалізоване рішення враховує сучасні вимоги до цифрових продуктів і може бути основою для подальшого розвитку системи з урахуванням специфічних потреб фармацевтичної галузі.

РОЗДІЛ 1. АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАВДАННЯ

1.1. Огляд існуючих рішень для аптечних ботів

Сучасний розвиток інформаційних технологій та зростання попиту на автоматизацію бізнес-процесів у сфері охорони здоров'я спричинили появу численних чат-ботів для аптечних мереж. m-Health (мобільне здоров'я) — це використання мобільних технологій, таких як смартфони, планшети та месенджери, для надання медичних та фармацевтичних послуг, підтримки здоров'я та управління медичними даними [1]. У контексті аптечної логістики m-Health дозволяє оптимізувати взаємодію між клієнтами, фармацевтами та складськими системами через зручні інтерфейси, такі як Telegram-боти. Це сприяє швидкому доступу до інформації про ліки, автоматизації замовлень і підвищенню якості обслуговування. Чат-боти, як складова m-Health, забезпечують реальний час обробки запитів, зменшення помилок і персоналізовану комунікацію, що відповідає сучасним вимогам фармацевтичної галузі. Такі рішення спрямовані на спрощення взаємодії між клієнтами, фармацевтами та системами управління складом, а також на підвищення ефективності продажів і логістики. Доцільно розглянути приклади аптечних чат-ботів, як міжнародних, так українські, з аналізом їх функціональних можливостей та особливостей реалізації.

Walgreens Pharmacy Chatbot (США) - одним із лідерів у сфері аптечних чат-ботів є рішення, розроблене американською мережею аптек Walgreens (рис. 1.1)[2]. Чат-бот інтегрований у мобільний додаток компанії та платформу обміну повідомленнями, таку як Facebook Messenger. Основні функції включають пошук ліків, перевірку їх наявності в аптеках, оформлення замовлень на доставку та нагадування про прийом медикаментів. Чат-бот використовує обробку природної мови (NLP) для взаємодії з користувачами, що дозволяє обробляти запити у вільній формі. Крім того, він інтегрований із системою управління рецептами, що забезпечує зручність для клієнтів із

хронічними захворюваннями. Проте обмеженням є необхідність підключення до внутрішньої екосистеми Walgreens, що ускладнює масштабування для інших ринків.

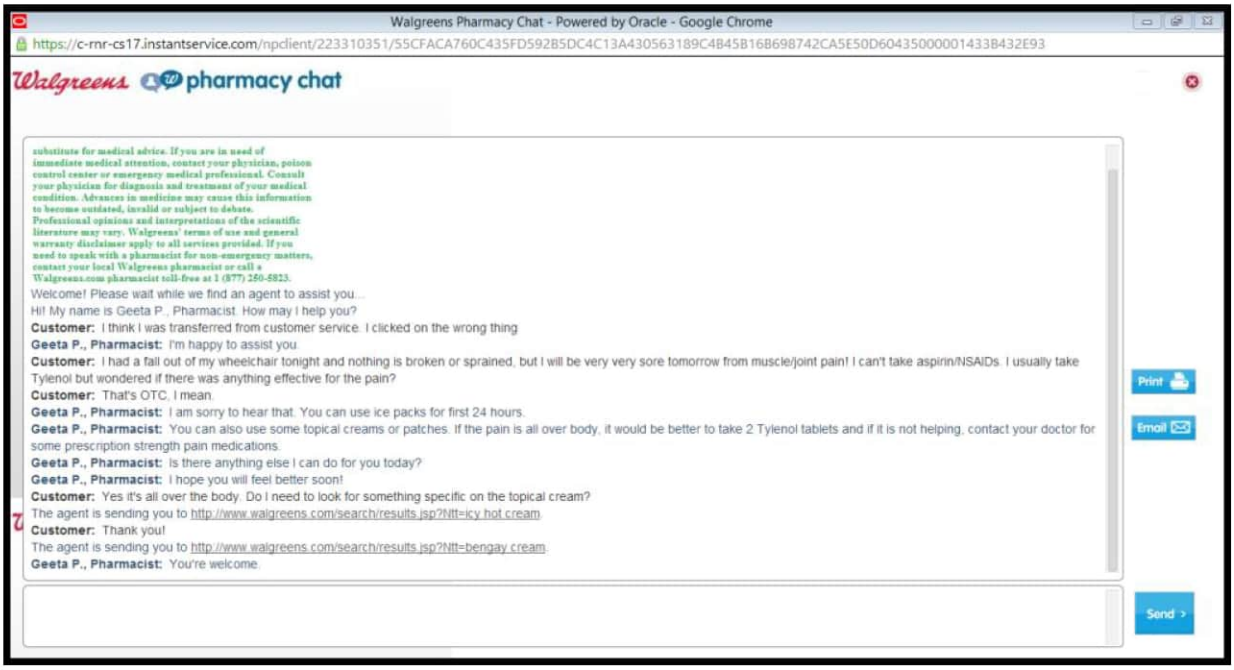

Рисунок 1.1. Walgreens Pharmacy Chatbot [2]

CVS Health Virtual Assistant (США) (рис. 1.2). Мережа аптек CVS Health пропонує чат-бота, який працює через вебсайт і мобільний додаток [3]. Його основна мета — надання консультацій щодо вибору ліків, перевірка страхових покриттів і бронювання консультацій із фармацевтами. Чат-бот використовує штучний інтелект для аналізу запитів і персоналізації рекомендацій, наприклад, пропонуючи аналоги ліків або супутні товари. Важливою особливістю є інтеграція з телемедичними сервісами, що дозволяє користувачам отримувати віддалені консультації. Однак складність інтерфейсу та потреба в реєстрації можуть створювати бар'єри для нових користувачів.

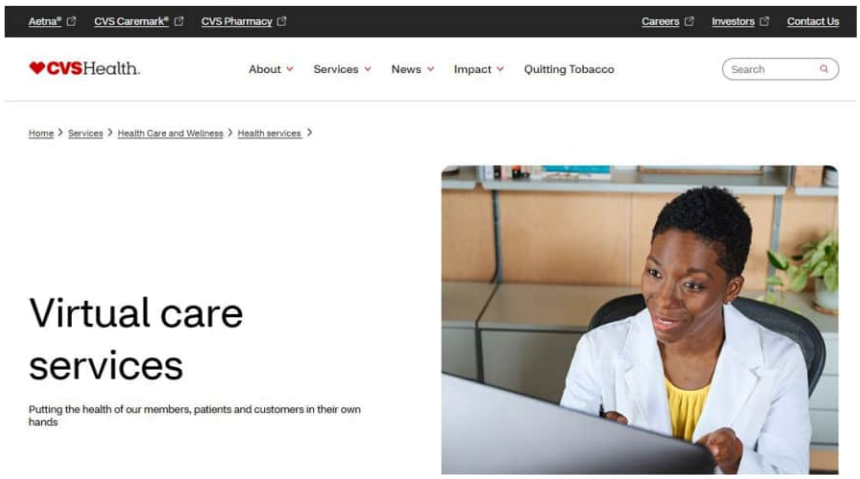

Рисунок 1.2. CVS Health Virtual Assistant [3]

Boots UK Pharmacy Bot (Велика Британія) (рис. 1.3) [4]. Британська мережа аптек Boots розробила чат-бота, доступного через WhatsApp і власний вебсайт. Його функціонал охоплює пошук медикаментів, оформлення рецептів, бронювання щеплень і консультації з фармацевтами. Чат-бот підтримує багатомовність, що є важливим для різноманітного населення Великої Британії. Особливістю є інтеграція з NHS (Національною службою охорони здоров'я), що дозволяє автоматично отримувати дані про рецепти. Недоліком є обмежена функціональність для управління складськими запасами, що робить його менш корисним для внутрішніх аптечних процесів.

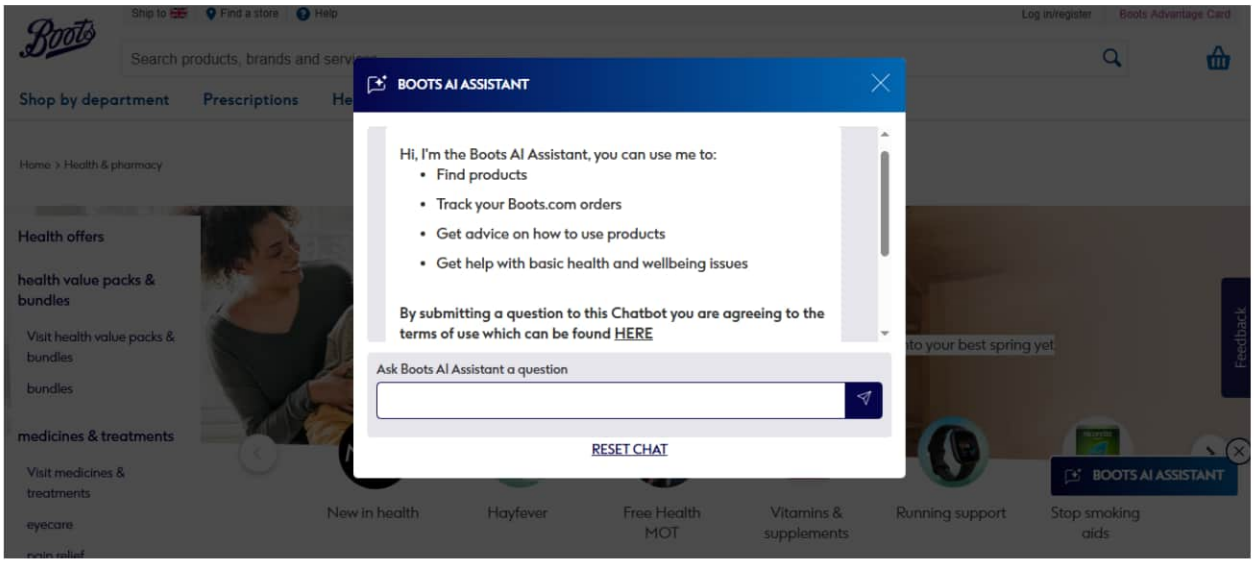

Рисунок 1.3. Boots UK Pharmacy Bot [4]

MedAdvisor (Австралія) (рис. 1.4) [5]. Австралійська платформа MedAdvisor пропонує чат-бота, інтегрованого в мобільний додаток для аптек і клієнтів. Він фокусується на управлінні рецептами, нагадуваннях про прийом ліків і замовленні медикаментів із місцевих аптек. Чат-бот також надає інформацію про побічні ефекти та взаємодію ліків, використовуючи базу даних, сертифіковану медичними організаціями. Унікальною особливістю є можливість підключення до системи лояльності, що мотивує клієнтів робити повторні покупки. Проте чат-бот має обмежену інтеграцію з аптечними складами, що знижує його ефективність для логістичних завдань.

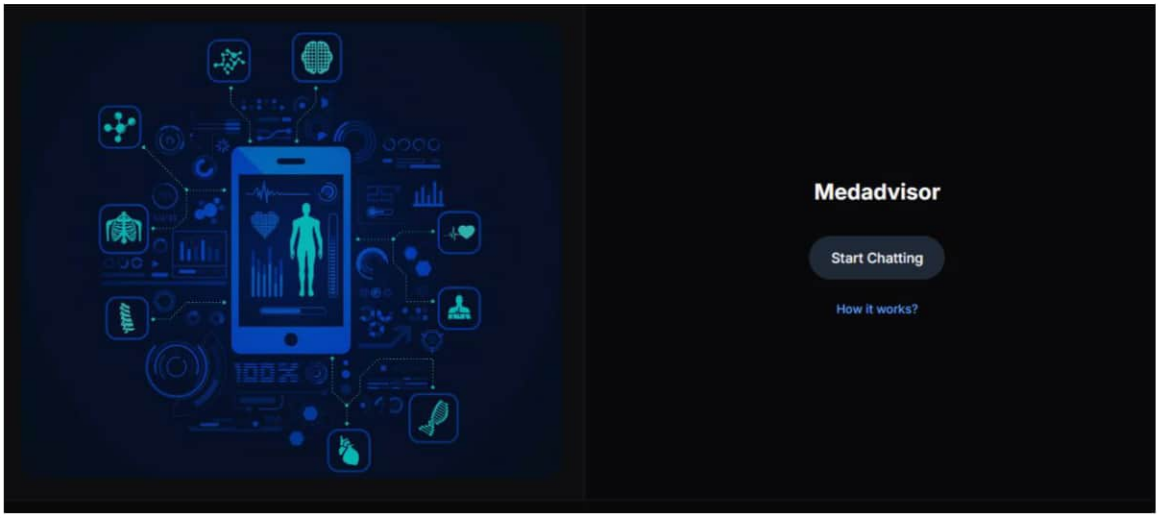

Рисунок 1.4. MedAdvisor сат-бот [5]

Tabletka.ua Bot (Україна) (рис. 1.5) [6]. Український сервіс Tabletka.ua, який є популярною платформою для пошуку ліків, пропонує чат-бота, доступного через Telegram і Viber. Його основні функції включають пошук медикаментів у аптеках за геолокацією, порівняння цін і бронювання ліків. Чат-бот також надає інформацію про аналоги препаратів і їх наявність у найближчих аптеках. Особливістю є простота інтерфейсу та швидкість обробки запитів, що робить його зручним для широкої аудиторії. Однак відсутність глибокої інтеграції з аптечними ERP-системами обмежує можливості управління складом і автоматизації внутрішніх процесів.

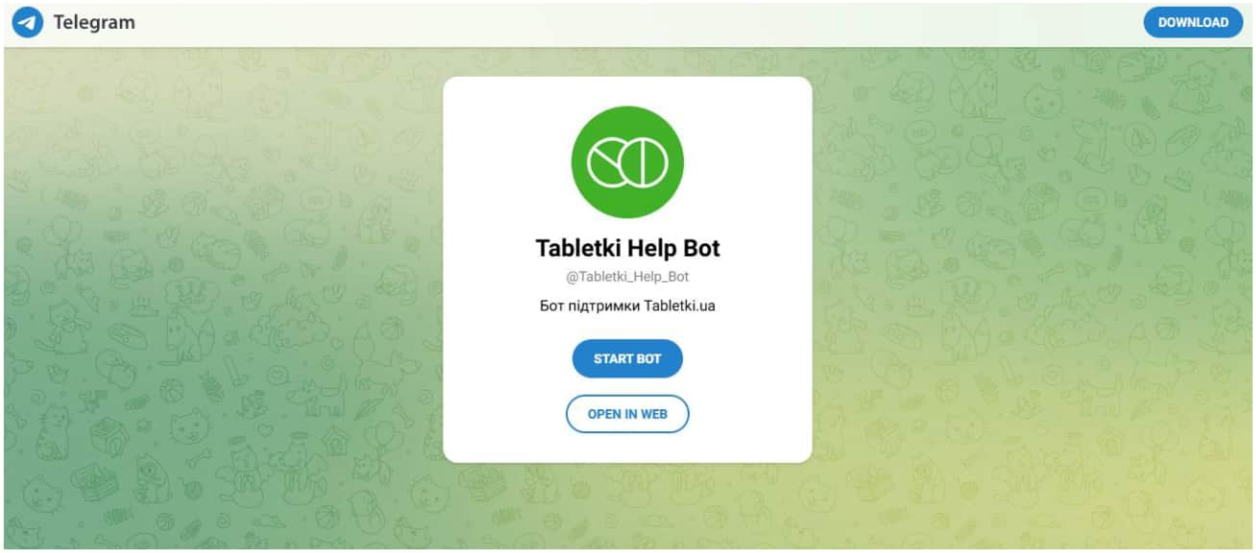

Рисунок .15. Tabletka.ua Bot [6]

Аналіз вищезазначених рішень показує, що сучасні аптечні чат-боти зосереджені переважно на клієнтському досвіді: пошуку ліків, бронюванні, консультаціях і управлінні рецептами. Міжнародні рішення, такі як Walgreens і CVS Health, вирізняються глибокою інтеграцією з медичними системами та страховими платформами, що забезпечує високий рівень персоналізації. Водночас український Tabletka.ua Bot орієнтований на простоту й доступність, але поступається в складності функціоналу. Жоден із розглянутих чат-ботів не забезпечує повноцінного управління аптечним складом, що вказує на нішу для розробки рішень, подібних до представленого проєкту, який поєднує клієнтські функції з логістичними (управління товарами, замовленнями та категоріями).

Сучасні аптечні чат-боти демонструють значний прогрес у автоматизації клієнтських і частково адміністративних процесів, але мають обмеження в управлінні складськими операціями. Розроблений проєкт має потенціал заповнити цю прогалину, поєднуючи клієнтський інтерфейс із функціями управління складом, що може стати конкурентною перевагою на ринку аптечних технологій.

1.2. Вимоги до функціоналу чат-бота

Розробка чат-бота для підтримки роботи аптечного складу є актуальним завданням, спрямованим на оптимізацію бізнес-процесів, підвищення ефективності управління складськими запасами та забезпечення зручної взаємодії з клієнтами. Чат-бот, як інструмент автоматизації, має відповідати специфічним вимогам, що враховують потреби як користувачів (клієнтів), так і адміністративного персоналу (менеджерів). На основі аналізу розробки проєкту та сучасних тенденцій у сфері автоматизації аптечних систем, можна сформулювати ключові вимоги до функціоналу чат-бота, що зображені на рис. 1.6.

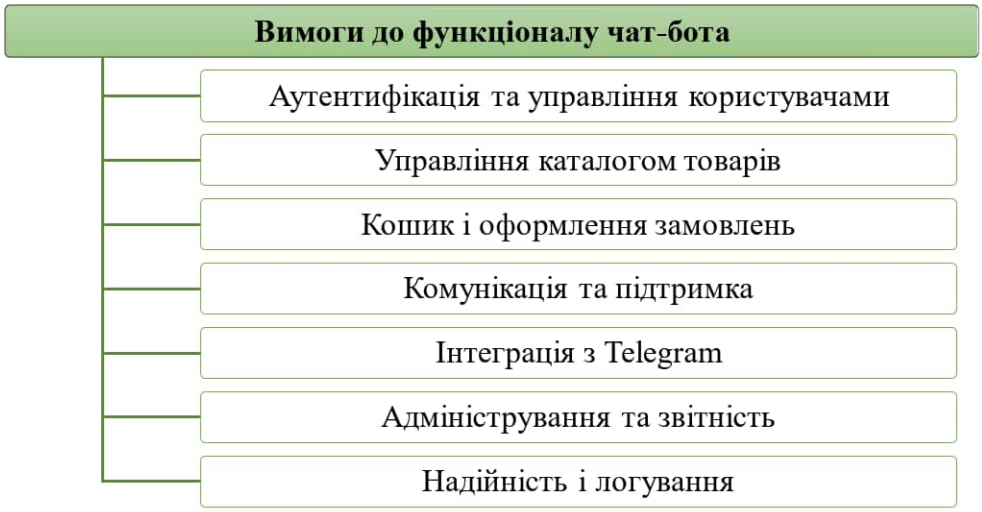

Рисунок 1.6. Вимоги до функціоналу чат-бота аптечних систем

Чат-бот повинен забезпечувати безпечний доступ до системи через механізми авторизації та реєстрації [7]. Користувачі поділяються на дві основні категорії: клієнти та менеджери, кожна з яких має різні рівні доступу [8]. Для клієнтів необхідна можливість створення облікового запису, входу в систему та управління особистими даними. Менеджери, своєю чергою, потребують розширених прав для адміністрування товарів, замовлень і комунікацій. Реалізація хешування паролів (наприклад, за допомогою

алгоритму SHA-256, як у коді проєкту) є обов'язковою для забезпечення безпеки даних.

Чат-бот має надавати клієнтам зручний доступ до каталогу лікарських засобів та супутніх товарів [9]. Функціонал включає перегляд товарів, фільтрацію за категоріями (наприклад, антибіотики, знеболювальні, вітаміни), пошук за назвою чи описом, а також сортування за ціною чи назвою. Для менеджерів передбачено CRUD-операції (створення, редагування, видалення) для товарів і категорій, що дозволяє підтримувати актуальність асортименту [10]. У розробленому коді реалізовано API-ендпоїнти для цих операцій, а також інтеграцію з Telegram-ботом для інтерактивного управління.

Важливим елементом є можливість для клієнтів формувати кошик шляхом додавання товарів із зазначенням кількості, редагування чи видалення позицій [11]. Система повинна перевіряти наявність товарів на складі перед додаванням до кошика чи оформленням замовлення. Після створення замовлення чат-бот має надсилати клієнту підтвердження з деталями (номер замовлення, перелік товарів, загальна сума) через Telegram. Менеджери отримують сповіщення про нові замовлення та можуть змінювати їх статус (наприклад, "нове", "в обробці", "доставлено"), що також відображається в повідомленнях клієнтам.

Чат-бот повинен забезпечувати двосторонню комунікацію між клієнтами та менеджерами [12]. Клієнти можуть звертатися до менеджерів із запитами (наприклад, уточнення наявності товару чи умов доставки), тоді як менеджери мають доступ до чатів із клієнтами для оперативного вирішення питань. У коді реалізовано систему обміну повідомленнями з підтримкою позначення прочитаних/непрочитаних повідомлень та інтеграцією з Telegram для миттєвих сповіщень. Крім того, менеджери можуть переглядати списки клієнтів і статистику їх активності.

З огляду на популярність месенджерів, чат-бот має бути інтегрованим із Telegram, що забезпечує зручний інтерфейс для користувачів [13]. Функціонал включає інтерактивні меню, інлайн-кнопки для навігації (наприклад, вибір

категорії чи статусу замовлення) та обробку текстових команд. У коді проєкту реалізована підтримка станів користувача (States), що дозволяє вести діалог із урахуванням контексту взаємодії, наприклад, послідовне введення даних для створення товару чи оформлення замовлення.

Для менеджерів чат-бот має надавати інструменти для управління складом, зокрема перегляд і редагування інформації про товари, категорії та замовлення [14]. Функціонал включає можливість перегляду історії замовлень, статусів і деталей, а також генерацію сповіщень про критичні ситуації (наприклад, низький запас товару). У коді передбачено відображення списків товарів, категорій і замовлень із відповідними діями для менеджерів.

Чат-бот повинен бути стійким до помилок і забезпечувати логування подій для діагностики та моніторингу [15]. У коді використовується модуль logging для фіксації ініціалізації бази даних, помилок відправки повідомлень у Telegram та інших критичних подій [16]. Це сприяє швидкому виявленню та усуненню проблем у роботі системи.

Таким чином, функціонал чат-бота для аптечного складу має поєднувати зручність для клієнтів із потужними інструментами адміністрування для менеджерів. Реалізація, представлена в кодах проєкту, відповідає більшості зазначених вимог, забезпечуючи інтеграцію з Telegram, безпечне управління даними, ефективну обробку замовлень і комунікацію. Однак для підвищення ефективності можна розглянути додаткові функції, такі як аналітика продажів чи інтеграція з платіжними системами, що розширить можливості системи в майбутньому.

1.3. Проблеми та обмеження

Розробка чат-бота для підтримки роботи аптечного складу є складним завданням, що потребує врахування специфіки фармацевтичної галузі, технічних обмежень та потреб користувачів. Аналіз реалізованих кодів (Bot.py та seed.py) дозволяє виділити низку проблем та обмежень, які можуть

впливати на ефективність впровадження та експлуатацію системи (рис. 1.7). Ці елементи охоплюють технічні, організаційні та функціональні виклики, які необхідно врахувати для забезпечення стабільної роботи чат-бота та його відповідності сучасним вимогам.

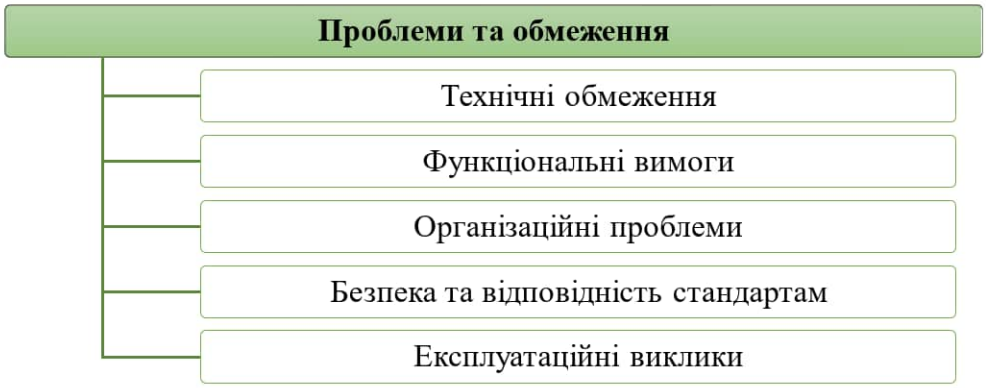

Рисунок 1.7. Проблеми та обмеження, що можуть впливати на ефективність впровадження та експлуатацію системи

Однією з ключових проблем є використання SQLite як бази даних. Хоча SQLite є легкою та простою у розгортанні, вона має обмеження щодо масштабованості та обробки великих обсягів даних [17]. У контексті аптечного складу, де кількість транзакцій (замовлення, оновлення запасів, повідомлення) може бути значною, SQLite може стати "вузьким місцем" через обмежену підтримку паралельних операцій та відсутність розподілених можливостей. Крім того, використання SHA-256 для хешування паролів, хоча й забезпечує базовий рівень безпеки, не є оптимальним рішенням, оскільки не включає механізми солі (salt), що підвищує вразливість до атак типу "rainbow table". Відсутність шифрування даних у базі даних також може створювати ризики витоку конфіденційної інформації, що є критичним для фармацевтичного сектору.

Чат-бот, реалізований у коді, має базовий набір функцій, таких як управління товарами, замовленнями, кошиком та обмін повідомленнями. Проте система не передбачає інтеграцію з зовнішніми платформами, наприклад, системами управління складом (WMS) чи платіжними шлюзами,

що обмежує її практичну цінність для реальних аптечних складів. Відсутність підтримки мультимови та адаптації інтерфейсу для різних груп користувачів (наприклад, клієнтів з обмеженими можливостями) знижує доступність системи. Крім того, функціонал пошуку товарів є базовим і не враховує складні сценарії, такі як фільтрація за кількома параметрами (ціна, виробник, діюча речовина), що може ускладнювати роботу як для клієнтів, так і для менеджерів.

Впровадження чат-бота в реальних умовах потребує чіткої інтеграції з бізнес-процесами аптечного складу. Код не містить механізмів для обробки нестандартних ситуацій, таких як повернення товарів, обробка скарг чи управління акційними пропозиціями. Відсутність автоматизованих інструментів для аналізу даних (наприклад, статистики продажів чи прогнозування попиту) знижує цінність системи для менеджерів. Крім того, Telegram як основна платформа для взаємодії має власні обмеження, зокрема залежність від стабільності сервісу та ризик блокування у деяких регіонах, що може унеможливити доступ до чат-бота.

Фармацевтична галузь підпадає під суворі регуляторні вимоги, такі як GDPR (у країнах ЄС) чи локальні стандарти захисту персональних даних. Код не демонструє механізмів забезпечення відповідності цим стандартам, зокрема щодо обробки персональних даних клієнтів чи безпечного зберігання інформації про замовлення. Відсутність двофакторної автентифікації та захисту від атак типу SQL-ін’єкцій чи CSRF (Cross-Site Request Forgery) створює додаткові ризики для системи.

Система передбачає ручне управління багатьма процесами, такими як оновлення статусів замовлень чи додавання нових товарів, що може бути трудомістким для менеджерів. Відсутність автоматичного тестування коду та документації ускладнює підтримку та масштабування системи в майбутньому. Крім того, Telegram-бот працює в окремому потоці, що може викликати проблеми синхронізації з основним Flask-додатком, особливо при високому навантаженні.

Розроблений чат-бот є функціональним прототипом, який забезпечує базові можливості для управління аптечним складом. Проте для його використання в реальних умовах необхідно вирішити низку проблем, пов’язаних із масштабністю, безпекою, інтеграцією з іншими системами та відповідністю регуляторним вимогам. Подолання цих обмежень потребує вдосконалення архітектури, впровадження сучасних технологій та врахування потреб усіх груп користувачів, що дозволить створити надійне та ефективне рішення для фармацевтичної галузі.

1.4 Мета та задачі розробки чат-бота для підтримки роботи аптечного складу

Мета проєкту полягає в розробці чат-бота для підтримки роботи аптечного складу спрямована на автоматизацію ключових складських процесів, оптимізацію управління запасами, обробку замовлень і забезпечення ефективної взаємодії між клієнтами та менеджерами через Telegram-інтерфейс. Чат-бот має підвищити ефективність логістичних операцій, зменшити кількість помилок і покращити якість обслуговування клієнтів у малих і середніх аптечних складах. Реалізація базується на інтеграції серверної частини (Flask), бази даних (SQLite) і клієнтської частини (Telegram-бот), що забезпечує зручний доступ до інформації в реальному часі.

Для виконання мети поставлені задачі:

– розробка структури бази даних створення таблиць передбачає збереження інформації про товари, категорії, замовлення, користувачів (клієнтів і менеджерів) та повідомлення, використовуючи зовнішні ключі для забезпечення цілісності даних;

– створення серверної логіки реалізація REST API на базі Flask відповідає за обробку запитів, таких як автентифікація користувачів, управління каталогом, кошиком і замовленнями, забезпечуючи підтримку CRUD-операцій;

– інтеграція з Telegram розробка чат-бота з використанням pyTelegramBotAPI забезпечує зручний інтерфейс із інтерактивними меню, інлайн-кнопками та механізмом підтримки станів користувача для ефективного оброблення складних сценаріїв взаємодії;

– забезпечення безпеки впровадження механізму хешування паролів через SHA-256, рольового доступу з розподілом прав між клієнтом та менеджером, а також використання сесій гарантують захист персональних даних користувачів;

– управління складськими процесами реалізація функціоналу для перегляду каталогу, фільтрації товарів, управління кошиком та обробки замовлень включає перевірку наявності товарів і автоматичне оновлення запасів у системі;

– організація комунікації забезпечення двостороннього обміну повідомленнями між клієнтами та менеджерами включає підтримку історії листування та генерацію сповіщень про нові замовлення або зміну їх статусу;

– логування та моніторинг використання модуля logging дозволяє фіксувати важливі події та діагностувати помилки, зокрема процеси ініціалізації бази даних та функціонування Telegram API;

– тестування та оптимізація проведення функціональних, інтеграційних, безпекових і навантажувальних тестів гарантує стабільність роботи системи та її відповідність встановленим вимогам.

Ці задачі, реалізовані в коді проєкту (Bot.py, seed.py), забезпечують створення функціонального прототипу, який автоматизує ключові процеси аптечного складу, підвищує ефективність і створює основу для подальшого масштабування системи.

1.5. Висновки до розділу 1

Проведений аналіз стану області розробки чат-ботів для аптечних систем засвідчив значний прогрес у автоматизації клієнтських і частково

адміністративних процесів, однак виявив певні прогалини, які відкривають можливості для вдосконалення. Огляд п’яти існуючих рішень (Walgreens Pharmacy Chatbot, CVS Health Virtual Assistant, Boots UK Pharmacy Bot, MedAdvisor, Tabletka.ua Bot) показав, що сучасні аптечні чат-боти зосереджені переважно на клієнтському досвіді: пошуку ліків, управлінні рецептами, бронюванні та консультаціях. Міжнародні рішення, такі як Walgreens і CVS Health, вирізняються глибокою інтеграцією з медичними та страховими системами, тоді як український Tabletka.ua Bot приваблює простотою й доступністю, але поступається в складності функціоналу. Жоден із розглянутих чат-ботів не забезпечує повноцінного управління складськими операціями, що підкреслює актуальність розробленого проєкту, який поєднує клієнтські функції з логістичними.

Ключові вимоги до функціоналу чат-бота включають безпечну авторизацію, зручний каталог товарів, управління кошиком і замовленнями, двосторонню комунікацію та інтеграцію з Telegram. Реалізований у коді проєкт відповідає цим вимогам, забезпечуючи базові можливості для клієнтів і менеджерів, зокрема CRUD-операції, обробку замовлень і сповіщення. Проте виявлено низку обмежень: використання SQLite обмежує масштабність, SHA-256 без солі знижує безпеку, а відсутність інтеграції з WMS чи платіжними системами звужує практичну цінність. Організаційні виклики, такі як ручне управління процесами та відсутність аналітики, а також регуляторні вимоги до захисту даних потребують подальшого вдосконалення.

Розроблений чат-бот є перспективним прототипом, який має потенціал заповнити нішу в управлінні аптечними складами. Для його практичного впровадження необхідно подолати технічні та функціональні обмеження шляхом модернізації архітектури, посилення безпеки та розширення інтеграційних можливостей, що забезпечить конкурентну перевагу на ринку аптечних технологій.

РОЗДІЛ 2. ПРОЄКТНІ РІШЕННЯ

2.1. Вибір технологій та платформи

Розробка чат-бота для підтримки роботи аптечного складу є складним завданням, яке вимагає ретельного вибору технологій та платформ для забезпечення надійності, масштабованості, безпеки та зручності використання. У процесі проєктування системи було проаналізовано низку технологій, враховуючи їх функціональність, доступність, популярність у спільноті розробників, а також відповідність вимогам проєкту. Важливо детально описати обґрунтування вибору технологій та платформ, які використано у коді проєкту.

Для реалізації чат-бота було обрано мову програмування Python, яка є однією з найпопулярніших у сфері веб-розробки, автоматизації та створення чат-ботів [18]. Python характеризується простотою синтаксису, що сприяє швидкому написанню та підтримці коду, а також має потужну екосистему бібліотек і фреймворків. Основні причини вибору Python описані в табл. 2.1.

Таблиця 2.1 – Основні причини вибору Python

№ з\п	Причини	Опис
1	Широка підтримка бібліотек	Python має бібліотеки, такі як Flask для створення веб-додатків, telebot для інтеграції з Telegram API, а також sqlite3 для роботи з базами даних, що значно спрощує розробку [19]
2	Кросплатформність	Python забезпечує можливість запуску коду на різних операційних системах (Windows, Linux, macOS), що важливо для розгортання системи на сервері [20]
3	Велика спільнота	Активна спільнота розробників забезпечує доступ до документації, навчальних матеріалів та форумів, що полегшує вирішення технічних проблем [21]
4	Гнучкість	Python дозволяє інтегрувати різні компоненти системи (веб-інтерфейс, чат-бот, базу даних) в єдину архітектуру [22]

Для реалізації серверної частини системи, яка обробляє API-запити, було обрано мікрофреймворк Flask [23]. Цей вибір обґрунтовано факторами, що подані в табл. 2.2.

Таблиця 2.2 – Основні причини вибору Flask

№ з\п	Причини	Опис
1	Легковагість	Flask є мінімалістичним фреймворком, що дозволяє розробникам зосередитися на основній логіці додатка без необхідності використання складних конфігурацій, як у більш важких фреймворках, наприклад Django [24]
2	Гнучкість	Flask дозволяє легко налаштовувати маршрути (endpoints) для обробки HTTP-запитів, що ідеально підходить для створення REST API, необхідного для взаємодії з чат-ботом та клієнтськими додатками [25]
3	Інтеграція з Python-екосистемою	Flask бездоганно працює з іншими Python-бібліотеками, такими як sqlite3 для роботи з базою даних та hashlib для хешування паролів [26]
4	Швидкість розробки	Завдяки простоті Flask розробка API для автентифікації, управління товарами, замовленнями та повідомленнями була виконана швидко та ефективно [27]

Telegram було обрано як основну платформу для реалізації чат-бота з огляду на його популярність, безпеку та зручність для користувачів. Основні переваги Telegram показані в табл. 2.3 [28].

Таблиця 2.3 – Основні переваги Telegram

№ з\п	Переваги	Опис
1	2	3
1	Доступність API	Telegram Bot API є добре документованим і дозволяє легко створювати боти з підтримкою текстових повідомлень, інлайн-клавіатур, callback-запитів тощо
2	Популярність серед користувачів	Telegram має широку аудиторію, що забезпечує зручність використання чат-бота для клієнтів і менеджерів аптеки
3	Безпека	Telegram підтримує шифрування повідомлень, що важливо для захисту конфіденційних даних, таких як інформація про замовлення чи листування між клієнтами та менеджерами

Продовження табл. 2.3

1	2	3
4	Інтерактивність	Можливість використання інлайн-клавіатур, кнопок і callback-запитів дозволяє створювати інтуїтивно зрозумілий інтерфейс для користувачів, що спрощує навігацію по каталогу товарів, оформлення замовлень і спілкування з менеджерами

Для роботи з Telegram використано бібліотеку pyTelegramBotAPI (telebot), яка забезпечує зручний інтерфейс для обробки повідомлень, створення клавіатур і управління станами користувачів [29].

Для зберігання даних про користувачів, товари, замовлення, категорії, кошики та повідомлення було обрано SQLite як систему управління базами даних. Основні причини цього вибору подані в табл. 1.4 [30].

Таблиця 2.4 – Основні причини вибору SQLite

№ з\п	Причини	Опис
1	Легковагість	SQLite не потребує окремого серверного процесу, що робить її ідеальною для проєктів середнього масштабу, таких як аптечний чат-бот
2	Вбудована підтримка в Python	Модуль sqlite3 є частиною стандартної бібліотеки Python, що спрощує інтеграцію та виключає необхідність встановлення додаткового програмного забезпечення
3	Достатня продуктивність	SQLite ефективно обробляє запити для невеликих і середніх обсягів даних, що відповідає потребам аптечного складу
4	Простота розгортання	SQLite зберігає базу даних у вигляді одного файлу, що полегшує резервне копіювання та перенесення даних

Структура бази даних включає таблиці для користувачів (users), товарів (products), категорій (categories), замовлень (orders), елементів замовлень (order_items), повідомлень (messages), кошиків (cart_items) та зв’язків із Telegram (telegram_users). Використання зовнішніх ключів (FOREIGN KEY) забезпечує цілісність даних.

Для забезпечення безпеки облікових даних користувачів використано

бібліотеку hashlib для хешування паролів за допомогою алгоритму SHA-256. Цей підхід має такі переваги:

– захист даних гарантується хешуванням паролів, що унеможливлює їх відновлення у разі витоку бази даних;

– простота реалізації досягається завдяки використанню hashlib, який входить до стандартної бібліотеки Python, що виключає потребу у сторонніх бібліотеках;

– достатня криптографічна стійкість забезпечується через використання SHA-256, який пропонує високий рівень безпеки для зберігання паролів у межах даного проєкту.

Для відстеження подій у системі та діагностики помилок використано стандартний модуль Python logging. Логування дозволяє:

– моніторинг роботи системи включає запис ключових подій, як-от ініціалізація бази даних, створення користувачів або помилки відправки повідомлень у Telegram;

– діагностика проблем здійснюється через логи, які дозволяють швидко виявляти та усувати помилки, наприклад, у разі збою Telegram API;

– гнучкість налаштування забезпечується модулем logging, який дозволяє визначати формат і рівень деталізації логів, таких як INFO, ERROR тощо.

Для одночасної роботи Flask-сервера та Telegram-бота використано модуль threading. Це рішення дозволяє:

– паралельне виконання забезпечується тим, що Flask-сервер обробляє API-запити, а Telegram-бот функціонує в окремому потоці, забезпечуючи безперервну взаємодію з користувачами;

– оптимізація ресурсів досягається використанням потоків замість окремих процесів, що зменшує витрати пам’яті та процесорного часу.

Додаткові бібліотеки описані в табл. 2.5.

Таблиця 2.5 – Додаткові бібліотеки

№ з\п	Назва бібліотеки	Опис
1	datetime	Використано для роботи з датами та часом, наприклад, для збереження часу створення замовлень чи повідомлень
2	os	Застосовано для роботи з файлами, зокрема для визначення шляху до бази даних
3	random	Використано в скрипті seed.py для генерації випадкових даних, таких як дати замовлень чи кількість товарів

Вибір технологій для розробки чат-бота для аптечного складу був зумовлений необхідністю створення надійної, зручної та безпечної системи з мінімальними витратами на розгортання та підтримку. Python у поєднанні з Flask, SQLite, pyTelegramBotAPI та іншими бібліотеками забезпечив гнучке та ефективне рішення, яке відповідає функціональним вимогам проєкту. Telegram як платформа для чат-бота гарантує зручність для користувачів і надійну інтеграцію з серверною частиною. Застосування модулів для безпеки, логування та багатопоточності забезпечило стабільність і масштабованість системи. Усі обрані технології є відкритими, що знижує витрати на розробку та сприяє подальшому розвитку системи.

2.2. Архітектура чат-бота

Архітектура чат-бота для підтримки роботи аптечного складу розроблена з урахуванням принципів модульності, масштабованості та безпеки. Вона базується на інтеграції Telegram-бота з веб-додатком на основі Flask, що забезпечує зручний доступ користувачів до функціоналу через месенджер та RESTful API для взаємодії з базою даних. Необхідно детально розглянути основні компоненти архітектури, їх взаємодію та ключові технологічні рішення.

Архітектура чат-бота побудована за трирівневою моделлю: рівень представлення, рівень бізнес-логіки та рівень даних.

Рівень представлення реалізовано через Telegram-бот, який виступає

основним інтерфейсом для взаємодії користувачів (клієнтів і менеджерів) із системою. Бот обробляє текстові повідомлення, команди та callback-запити, надаючи користувачам меню, списки товарів, кошик, історію замовлень тощо.

Рівень бізнес-логіки побудовано на основі веб-додатка Flask, який обробляє запити через RESTful API та керує логікою взаємодії між користувачами, Telegram-ботом і базою даних. Цей рівень включає автентифікацію, управління кошиком, замовленнями, повідомленнями та адмін-функціонал.

Рівень даних представлено базою даних SQLite, яка зберігає інформацію про користувачів, товари, категорії, замовлення, повідомлення та кошики. SQLite обрано через її легкість, достатню продуктивність для малих і середніх аптечних складів та простоту інтеграції.

Архітектура включає ключові компоненти, описані в табл. 2.6.

Таблиця 2.6 – Компоненти архітектури

№ з\п	Назва компоненту	Опис
1	2	3
1	Telegram-бот	Реалізовано за допомогою бібліотеки python-telegram-bot. Бот підтримує стани користувача (визначені в класі States), що дозволяє керувати багатоступеневими сценаріями взаємодії, наприклад, авторизацією, вибором товарів чи створенням замовлень. Бот асинхронно обробляє повідомлення та callback-запити, що забезпечує швидку реакцію на дії користувача
2	Flask-додаток	Веб-сервер на основі Flask обробляє HTTP-запити через API-ендпоінти, такі як /login, /products, /orders тощо. Він забезпечує автентифікацію (з використанням сесій), управління товарами, категоріями, замовленнями та повідомленнями. Flask також інтегрується з Telegram-ботом для надсилання сповіщень
3	База даних SQLite	Використовується для зберігання структурованих даних у таблицях (users, products, orders, messages тощо) (рис. 2.1). Вона підтримує зв’язки між таблицями через зовнішні ключі, що забезпечує цілісність даних. Наприклад, таблиця order_items пов’язує замовлення з товарами

Продовження табл. 2.6

1	2	3
3	База даних SQLite	Використовується для зберігання структурованих даних у таблицях (users, products, orders, messages тощо) (рис. 2.1). Вона підтримує зв'язки між таблицями через зовнішні ключі, що забезпечує цілісність даних. Наприклад, таблиця order_items пов'язує замовлення з товарами
4	Механізм логування	Використовується модуль logging для запису подій (ініціалізація бази, помилки Telegram-бота тощо), що полегшує діагностику та моніторинг роботи системи

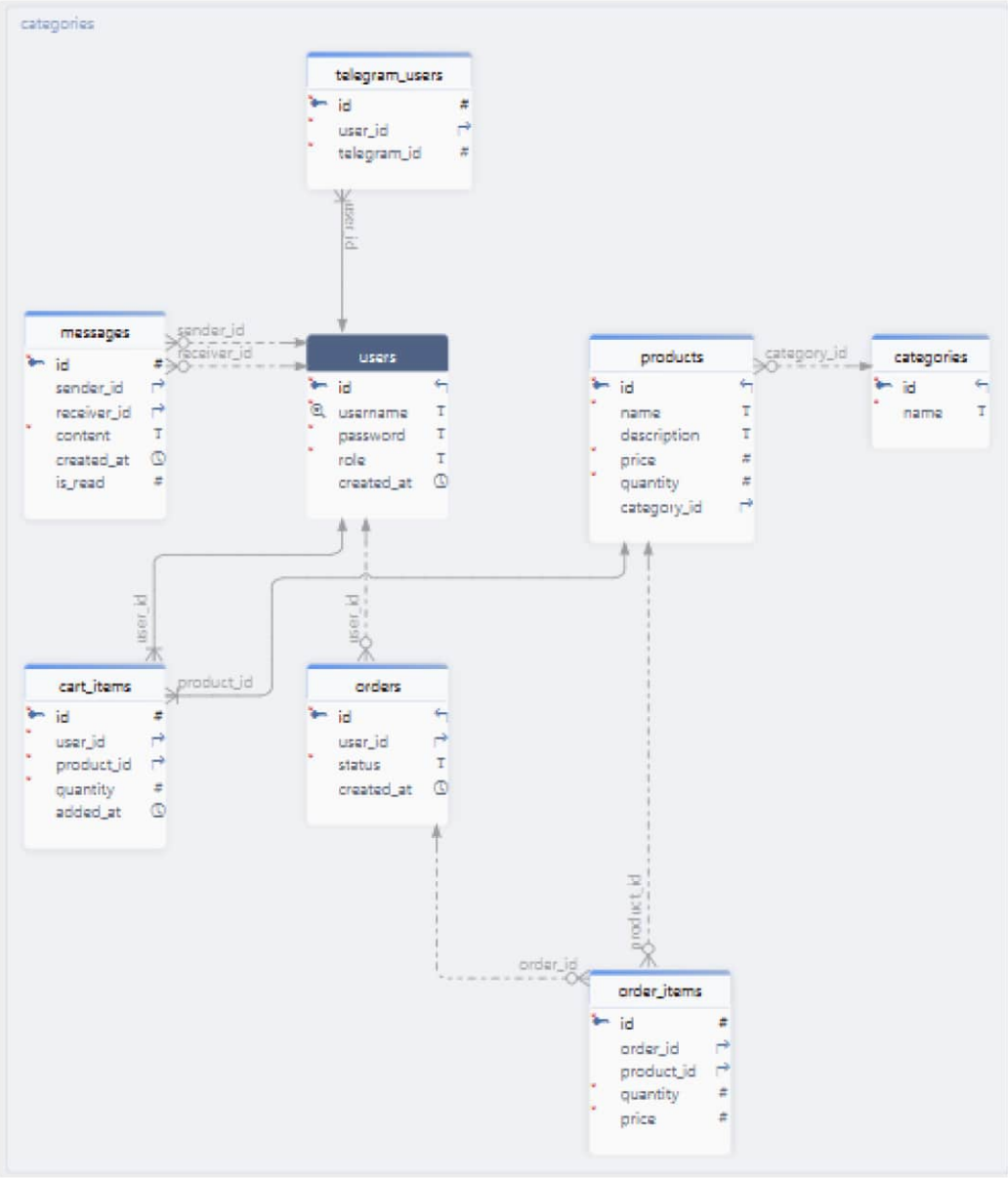

Рисунок 2.1. Структура бази даних

Діаграма компонентів (рис. 2.2) ілюструє основні компоненти системи

(Telegram-бот, Flask-додаток, SQLite) та їх взаємозв’язки через API та базу даних [31]. Вона допомагає зрозуміти розподіл функціоналу між модулями

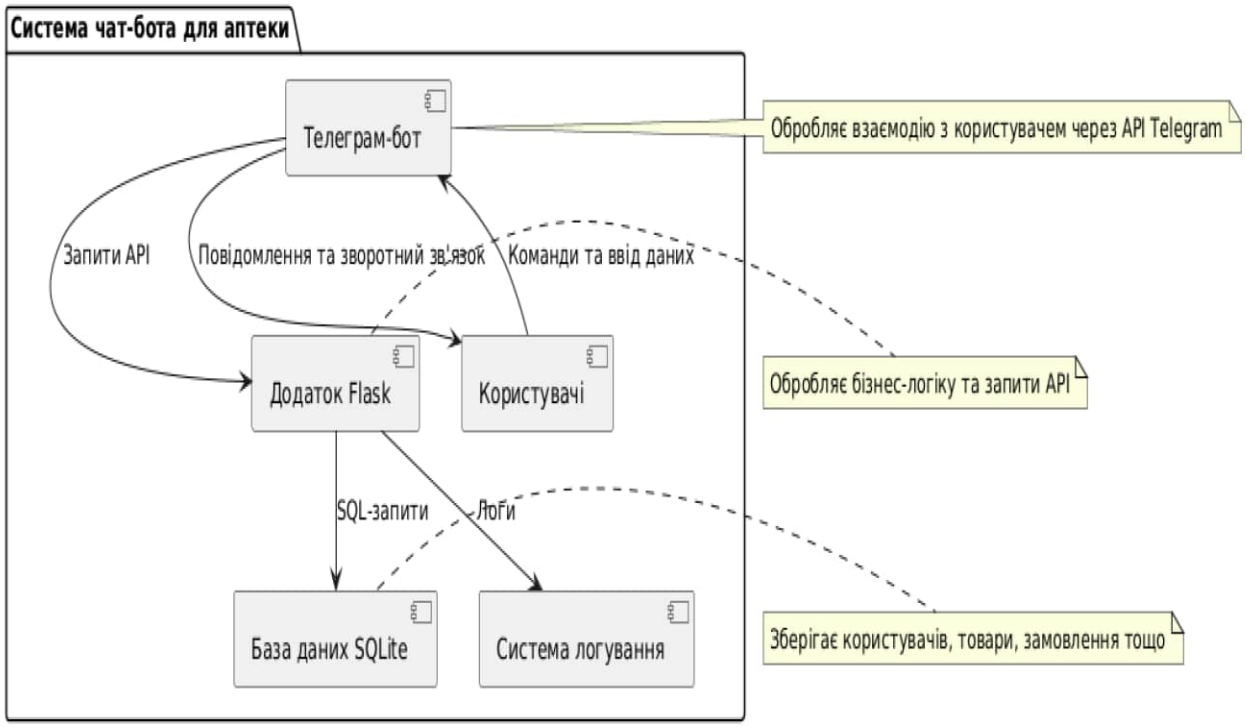

Рисунок 2.2. Діаграма компонентів

Компоненти системи взаємодіють через чітко визначені інтерфейси:

– Користувач надсилає повідомлення або натискає кнопки в Telegram, які обробляються ботом. Бот визначає стан користувача (user_states) і виконує відповідні дії, наприклад, надсилає запит до Flask API.

– Flask-додаток отримує запити від бота або клієнтів через API, виконує операції з базою даних (CRUD) і повертає результати. Наприклад, при створенні замовлення (/orders POST) Flask оновлює таблиці orders, order_items і products та надсилає сповіщення через Telegram.

– SQLite забезпечує персистентне зберігання даних, до якого Flask звертається через бібліотеку sqlite3. Транзакції використовуються для гарантії консистентності, наприклад, при оновленні кількості товарів на складі.

Безпека системи забезпечується на кількох рівнях:

– Автентифікація забезпечується хешуванням паролів за допомогою

алгоритму SHA-256 (hashlib), що гарантує їх захист від компрометації. Для зберігання даних авторизованих користувачів використовуються сесії Flask.

– Рольовий доступ визначає функціональні обмеження для ролей manager і client. Наприклад, лише менеджерам надається можливість редагувати товари або змінювати статуси замовлень.

– Обмеження Telegram реалізується через унікальний telegram_id, який пов'язується з user_id у таблиці telegram_users, що дозволяє уникнути несанкціонованого доступу.

Архітектура дозволяє легко розширювати функціонал:

– додавання нових API-ендпоінтів у Flask для підтримки додаткових функцій (наприклад, аналітики продажів);

– заміна SQLite на більш потужну СУБД, таку як PostgreSQL, без значних змін у коді;

– інтеграція додаткових каналів зв'язку (наприклад, веб-інтерфейсу) через той самий Flask API.

Діаграма послідовності (рис. 2.3) показує процес створення замовлення, включаючи взаємодію користувача з ботом, обробку запиту Flask і оновлення бази даних [32]. Ця діаграма деталізує потік даних і послідовність операцій. Оскільки чат-бот підтримує багато функцій, варто зосередитись на типовому сценарії: клієнт авторизується через Telegram, переглядає каталог, додає товар до кошика та створює замовлення. Цей сценарій охоплює основні взаємодії між користувачем, Telegram-ботом, Flask-сервером та базою даних. Сценарій функціонування системи охоплює кілька ключових етапів. Першим є авторизація користувача, яка передбачає вхід до системи. Далі користувач отримує можливість переглядати каталог доступних товарів, який містить функції пошуку та сортування. Після цього додається можливість внесення товару до кошика з подальшим створенням замовлення. Завершальним етапом є генерування сповіщення для менеджера, яке інформує про надходження нового замовлення, що забезпечує оперативну обробку клієнтських запитів.

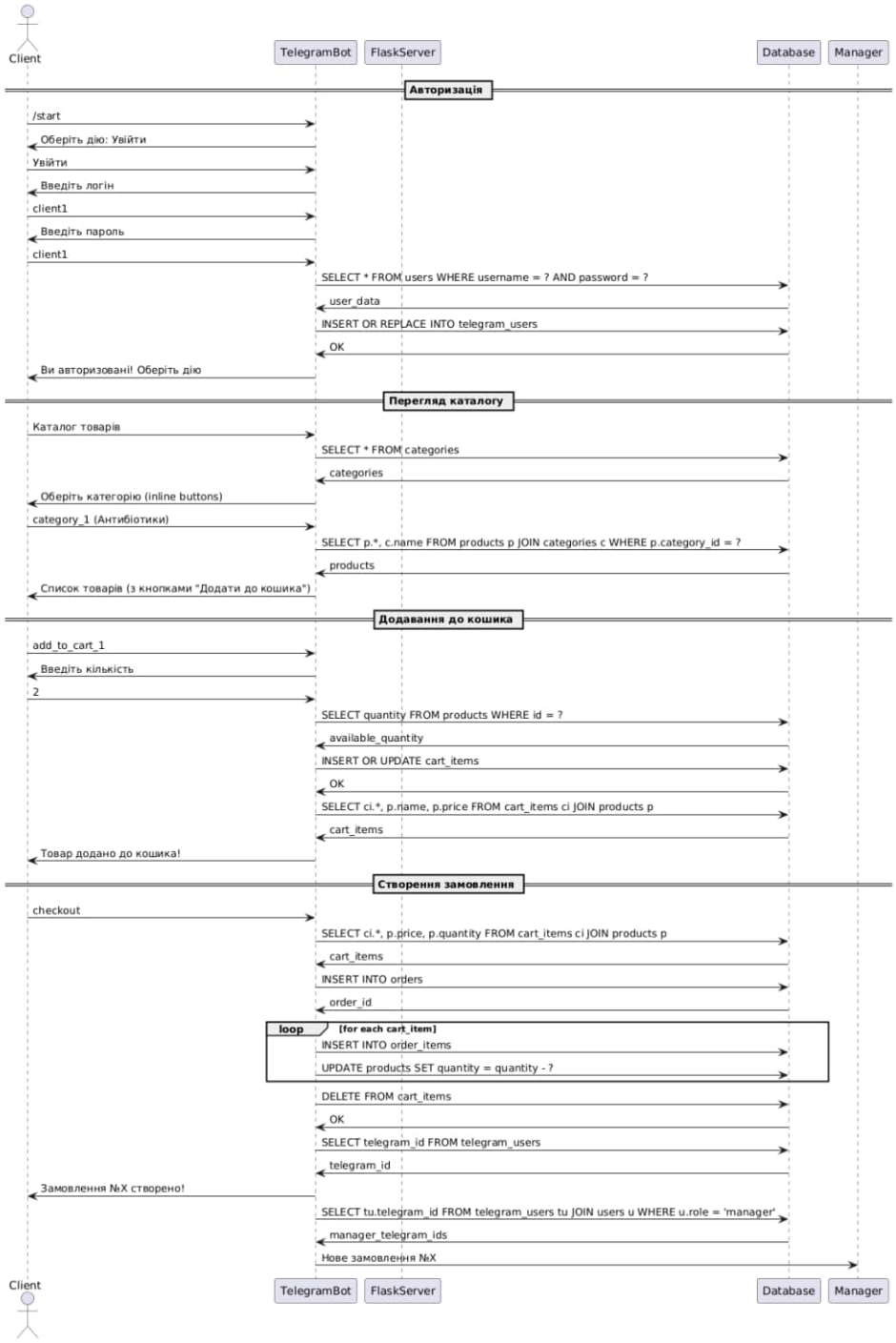

Рисунок 2.3. Діаграма послідовності

Архітектура чат-бота для аптечного складу поєднує простоту реалізації з гнучкістю та безпекою. Використання Telegram-бота як основного інтерфейсу забезпечує зручність для користувачів, а Flask і SQLite надають надійну основу для обробки бізнес-логіки та зберігання даних. Модульна структура дозволяє адаптувати систему до зростаючих потреб аптечного складу, а UML-діаграми сприяють чіткому документуванню її компонентів і

взаємодій.

2.3. Реалізація ключових функцій

Розробка чат-бота для підтримки роботи аптечного складу передбачає створення системи, яка забезпечує зручну взаємодію користувачів (клієнтів та менеджерів) з базою даних складу через веб-інтерфейс та Telegram-бот. Важливо розглянути реалізацію ключових функцій системи, таких як аутентифікація користувачів, управління каталогом товарів, обробка замовлень, управління кошиком та обмін повідомленнями. Для ілюстрації використано фрагменти коду проєкту, що демонструють технічну реалізацію.

Аутентифікація є основною функцією для забезпечення безпеки доступу до системи. Реалізація передбачає перевірку логіна та пароля користувача, а також створення нових облікових записів. Паролі хешуються за допомогою алгоритму SHA-256 для захисту даних. Функція login у файлі Bot.py обробляє запит на вхід, перевіряючи відповідність введених даних записам у базі даних SQLite (рис. 2.4).

```
@app.route('/login', methods=['POST'])
def login():
    data = request.json
    username = data.get('username')
    password = data.get('password')
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    hashed_password = hashlib.sha256(password.encode()).hexdigest()
    cursor.execute("SELECT * FROM users WHERE username = ? AND
password = ?", (username, hashed_password))
    user = cursor.fetchone()
    conn.close()
    if user:
        session['user_id'] = user['id']
        session['role'] = user['role']
        return jsonify({
            'success': True,
            'user': {'id': user['id'], 'username': user['username'],
'role': user['role']}
        })
    return jsonify({'success': False, 'message': 'Невірний логін або
пароль'}), 401
```

Рисунок 2.4. Перевірка відповідності

Для Telegram-бота аутентифікація (рис. 2.5) реалізується через обробку текстових повідомлень у стані AWAITING_LOGIN та AWAITING_PASSWORD. Після успішного входу користувач отримує доступ до головного меню, яке залежить від його ролі (клієнт або менеджер).

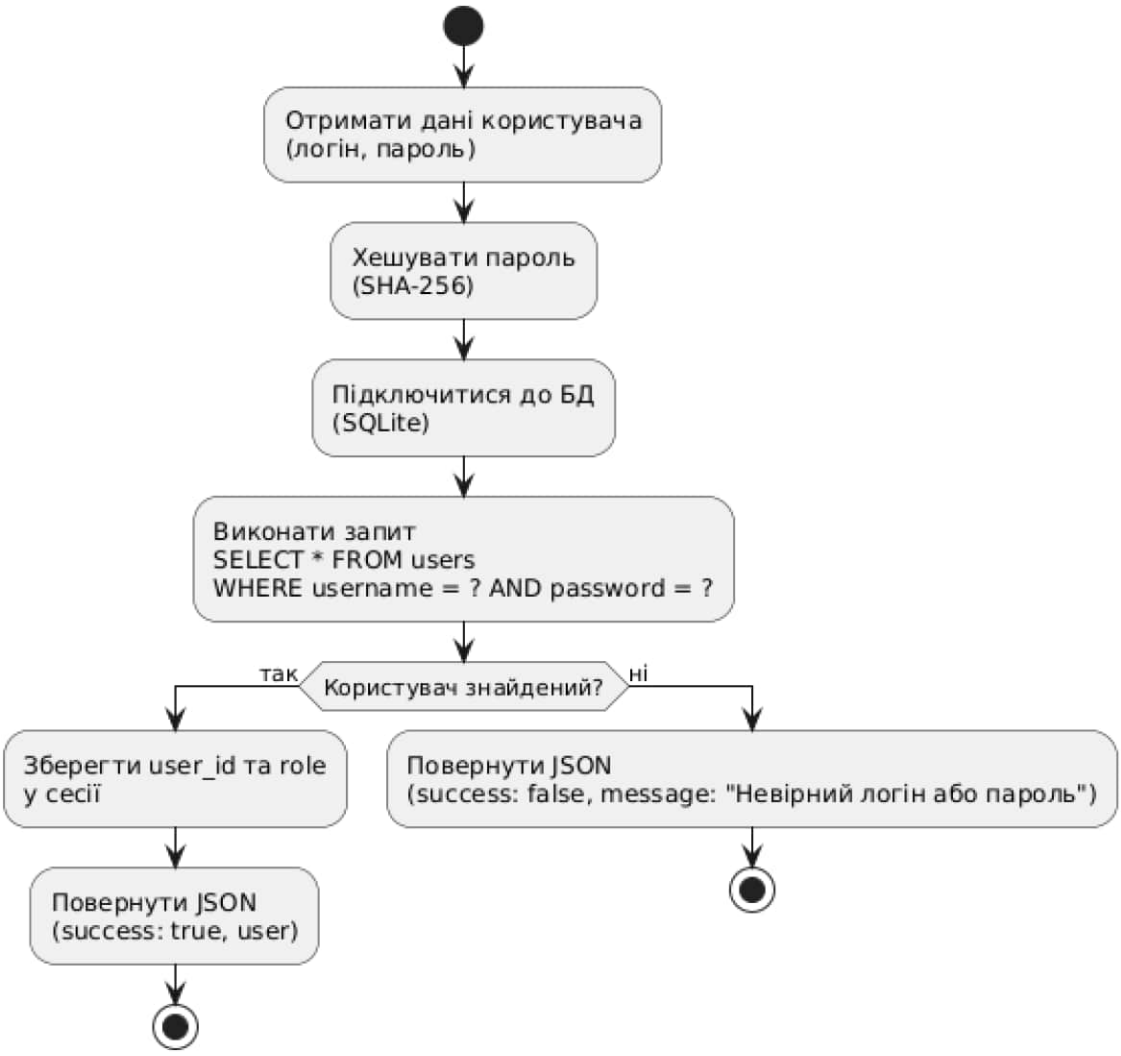

Рисунок 2.5. Алгоритм аутентифікації користувачів

Функція управління каталогом дозволяє клієнтам переглядати товари, фільтрувати їх за категоріями та здійснювати пошук, а менеджерам — додавати, редагувати та видаляти товари. API-ендпоінт /products у файлі Bot.py забезпечує отримання списку товарів з можливістю фільтрації та сортування (рис. 2.6).

```
@app.route('/products', methods=['GET'])
def get_products():
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    query = "SELECT p.*, c.name as category_name FROM
products p LEFT JOIN categories c ON p.category_id = c.id"
    params = []
    search_query = request.args.get('search')
    if search_query:
        query += " WHERE p.name LIKE ? OR p.description
LIKE ?"
        params.extend([f'%{search_query}%',
f'%{search_query}%'])
    category_id = request.args.get('category')
    if category_id:
        if 'WHERE' in query:
            query += " AND p.category_id = ?"
        else:
            query += " WHERE p.category_id = ?"
        params.append(category_id)
    cursor.execute(query, params)
    products = [dict(row) for row in cursor.fetchall()]
    conn.close()
    return jsonify({'success': True, 'products': products,
'categories': categories})
```

Рисунок 2.6. Отримання списку товарів

У Telegram-боті каталог відображається через інлайн-клавіатуру, яка генерується функцією show_catalog. Користувачі можуть вибирати категорії або виконувати пошук, що забезпечує зручну навігацію (рис. 2.7).

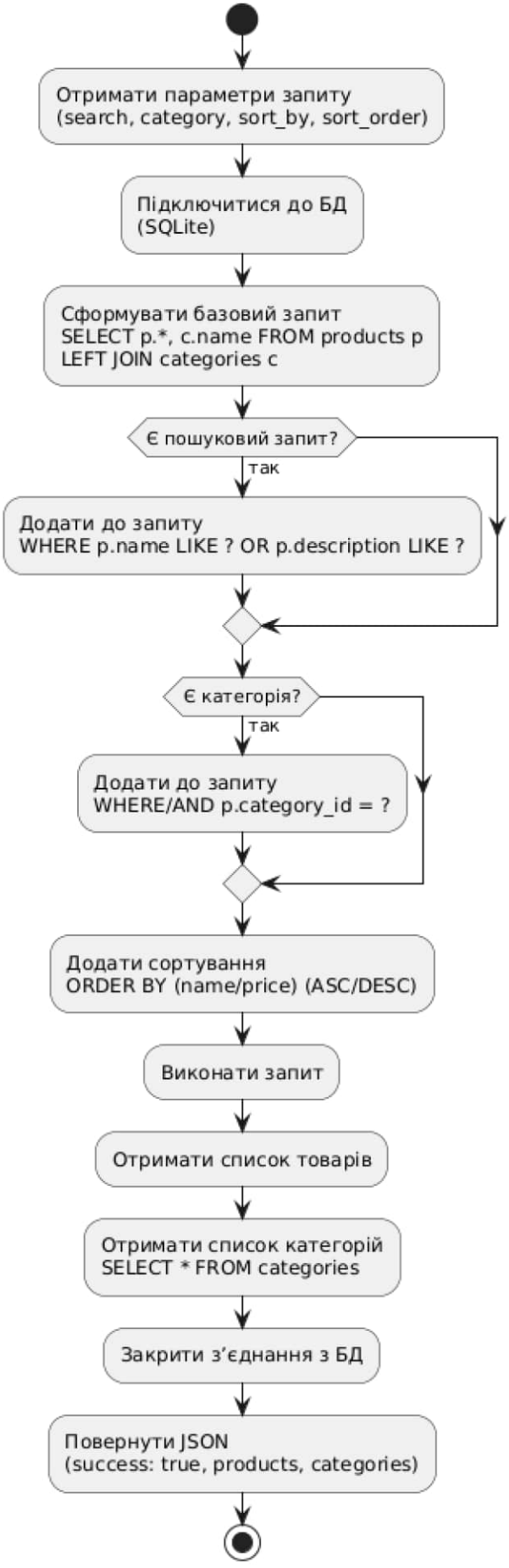

Рисунок 2.7. Алгоритм функції управління каталогом

Функція кошика дозволяє клієнтам додавати товари, змінювати їх кількість або видаляти. Реалізація включає перевірку наявності товару на складі перед додаванням до кошика, як показано у фрагменті (рис. 2.8).

```
@app.route('/cart', methods=['POST'])
def add_to_cart():
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'success': False, 'message':
'Необхідна авторизація'}), 401
    data = request.json
    product_id = data.get('product_id')
    quantity = data.get('quantity', 1)
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM products WHERE id = ?",
(product_id,))
    product = cursor.fetchone()
    if not product:
        conn.close()
        return jsonify({'success': False, 'message': 'Товар
не знайдено'}), 404
    if product['quantity'] < quantity:
        conn.close()
        return jsonify({'success': False, 'message':
'Недостатньо товару в наявності'}), 400
    cursor.execute(
        "INSERT INTO cart_items (user_id, product_id,
quantity) VALUES (?, ?, ?)",
        (user_id, product_id, quantity)
    )
    conn.commit()
    conn.close()
    return jsonify({'success': True, 'cart': cart_items})
```

Рисунок 2.8. Перевірка наявності товару

У Telegram-боті кошик відображається через функцію show_cart, а користувач може взаємодіяти з ним через інлайн-кнопки для оформлення замовлення або очищення кошика (рис. 2.9).

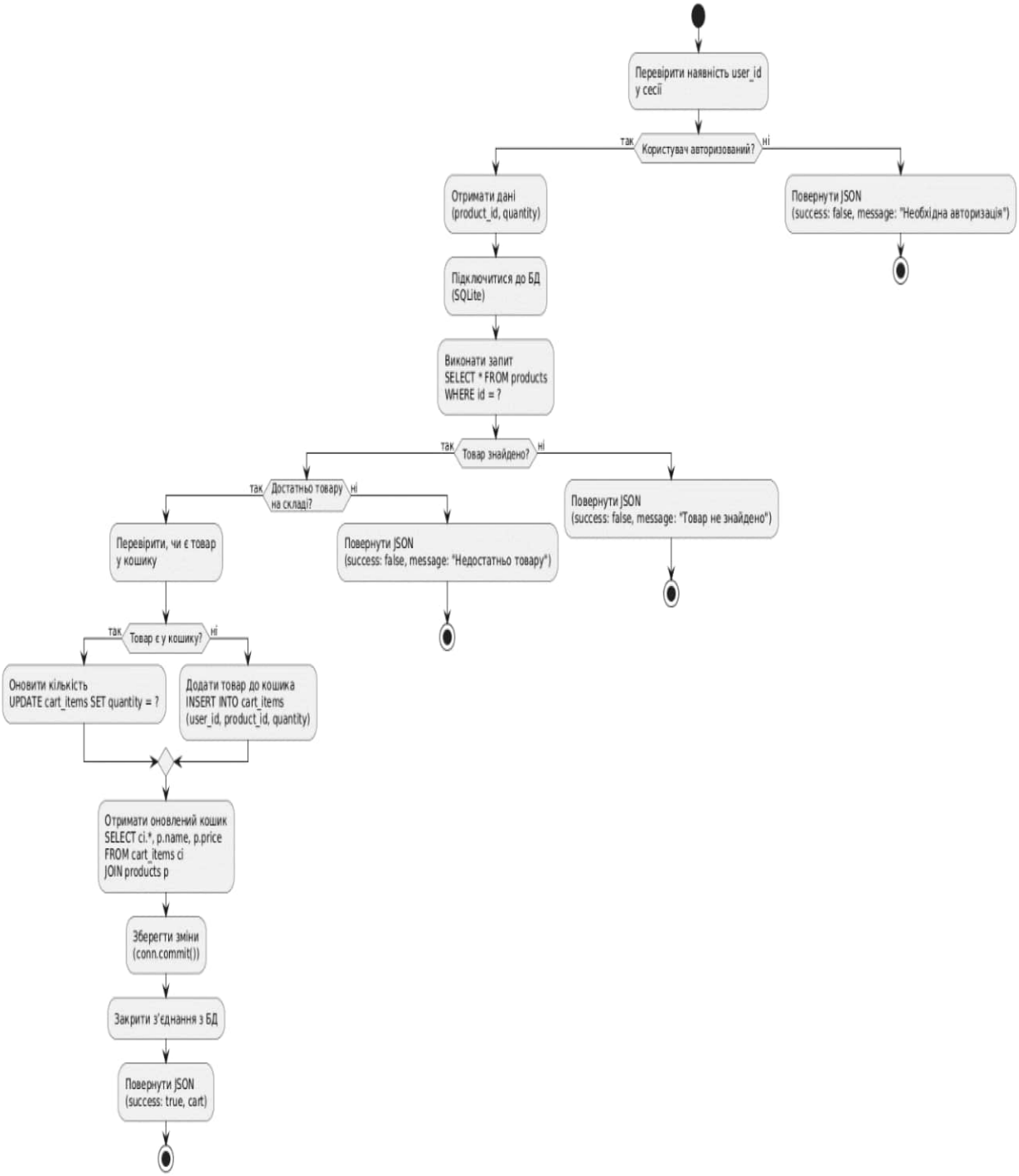

Рисунок 2.9. Алгоритм функції управління кошиком

Створення замовлення передбачає перенесення товарів з кошика до таблиці orders, зменшення кількості товарів на складі та сповіщення користувача через Telegram. Функція create_order у файлі Bot.py реалізує цей процес (рис. 2.10).

```
@app.route('/orders', methods=['POST'])
def create_order():
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'success': False, 'message':
'Необхідна авторизація'}), 401
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("SELECT ci.product_id, ci.quantity,
p.price, p.quantity as available_quantity FROM cart_items
ci JOIN products p ON ci.product_id = p.id WHERE ci.user_id
= ?", (user_id,))
    cart_items = cursor.fetchall()
    if not cart_items:
        conn.close()
        return jsonify({'success': False, 'message': 'Кошик
порожній'}), 400
    cursor.execute("INSERT INTO orders (user_id, status)
VALUES (?, ?)", (user_id, 'new'))
    order_id = cursor.lastrowid
    for item in cart_items:
        cursor.execute(
            "INSERT INTO order_items (order_id, product_id,
quantity, price) VALUES (?, ?, ?, ?)",
            (order_id, item['product_id'],
item['quantity'], item['price'])
        )
        cursor.execute("UPDATE products SET quantity =
quantity - ? WHERE id = ?", (item['quantity'],
item['product_id']))
    cursor.execute("DELETE FROM cart_items WHERE user_id =
?", (user_id,))
    conn.commit()
    conn.close()
    return jsonify({'success': True, 'order': order})
```

Рисунок 2.10. Створення замовлень

Менеджери можуть змінювати статус замовлення (наприклад, «нове», «в обробці», «доставлено»), що супроводжується сповіщенням клієнта через Telegram (рис. 2.11).

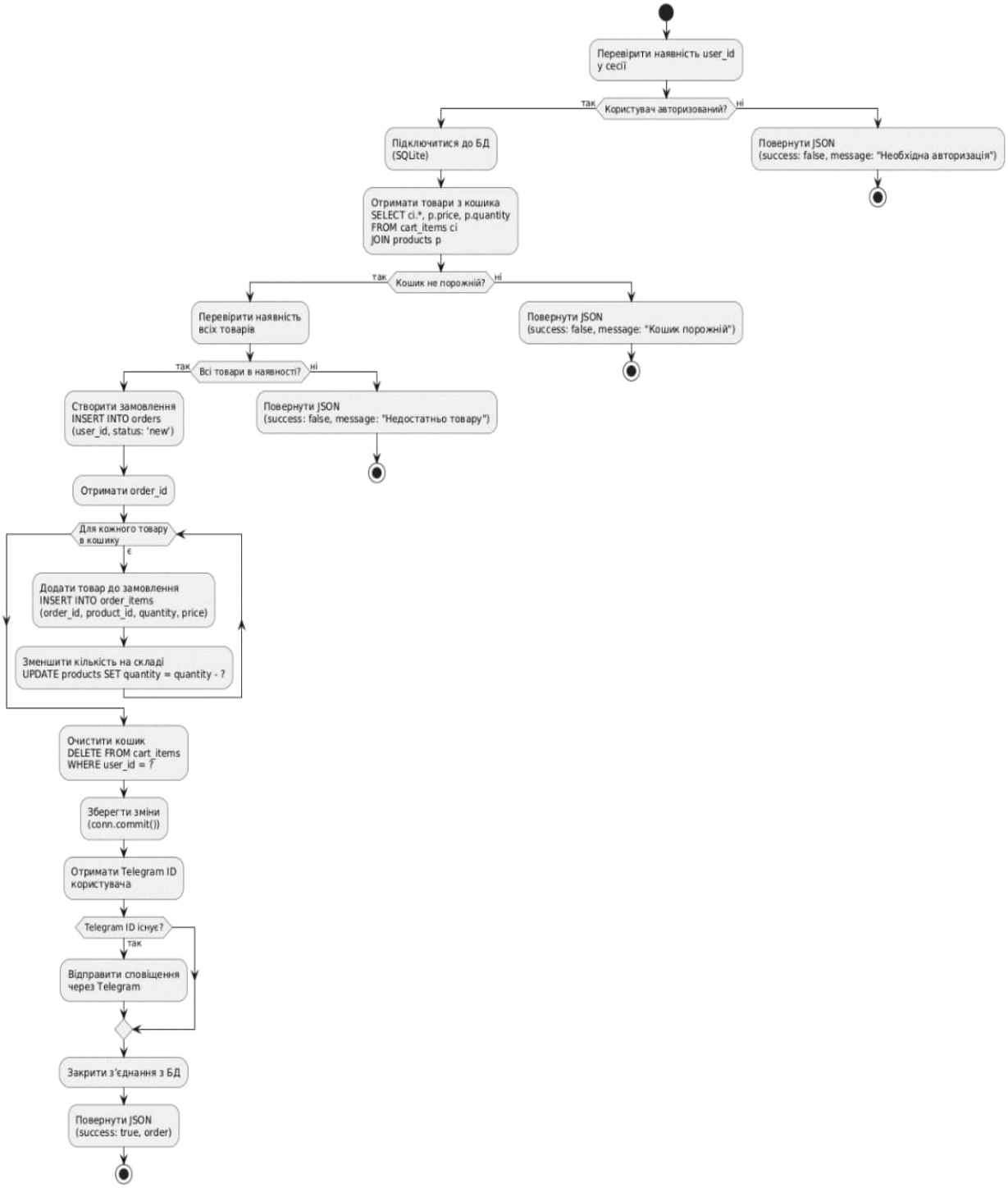

Рисунок 2.11. Алгоритм функції обробки замовлень

Система обміну повідомленнями дозволяє клієнтам зв’язуватися з менеджерами, а менеджерам — відповідати клієнтам. Реалізація включає збереження повідомлень у таблиці messages та відправлення їх через Telegram. Функція send_message у файлі Bot.py обробляє відправку повідомлень (рис. 2.12)

```
@app.route('/messages', methods=['POST'])
def send_message():
    sender_id = session.get('user_id')
    if not sender_id:
        return jsonify({'success': False, 'message': 'Необхідна
авторизація'}), 401
    data = request.json
    receiver_id = data.get('receiver_id')
    content = data.get('content')
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("INSERT INTO messages (sender_id,
receiver_id, content) VALUES (?, ?, ?)", (sender_id,
receiver_id, content))
    message_id = cursor.lastrowid
    conn.commit()
    conn.close()
    return jsonify({'success': True, 'message': message})
```

Рисунок 2.12 – Обробка відправлень повідомлень

У Telegram-боті користувачі можуть переглядати історію листування та надсилати нові повідомлення через функцію show_chat_history (рис. 2.13).

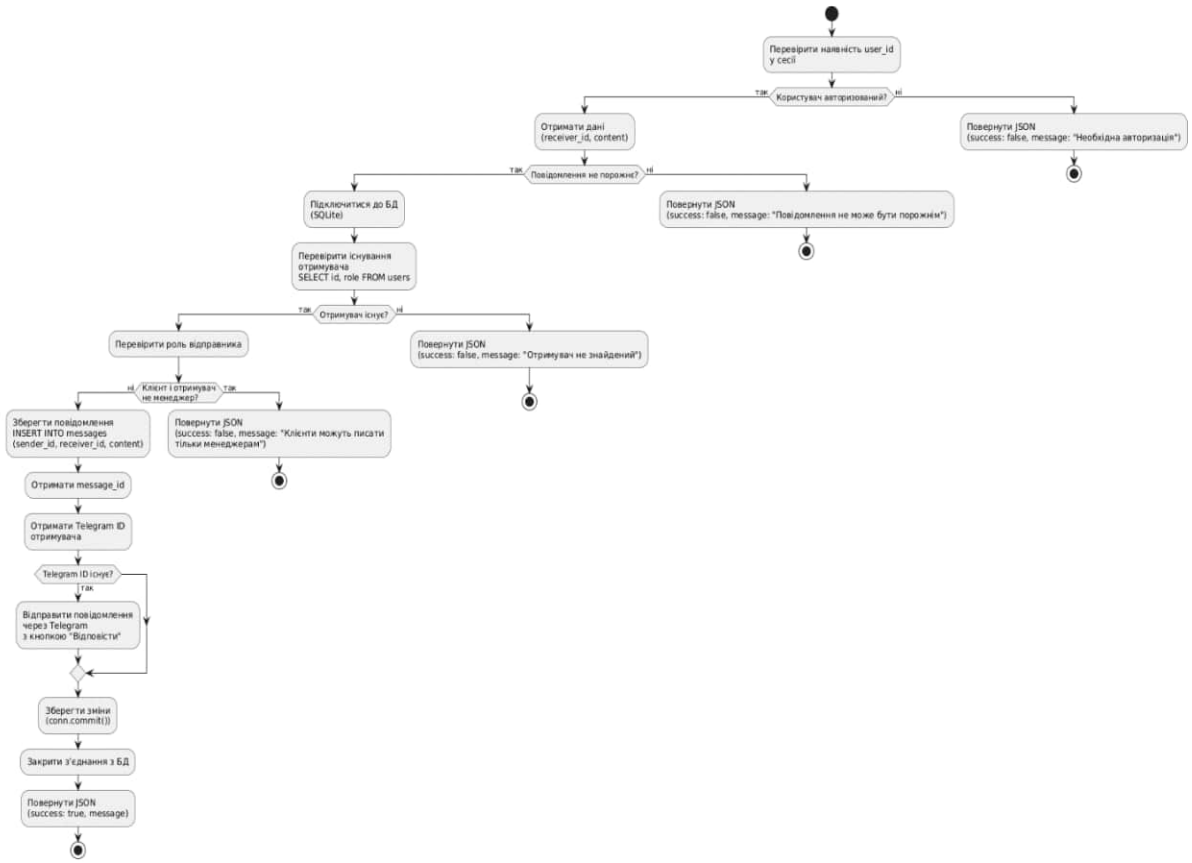

Рисунок 2.13. Алгоритм функції обробки повідомлень

Реалізація ключових функцій чат-бота для аптечного складу забезпечує зручну взаємодію користувачів із системою через веб-інтерфейс та Telegram. Використання Flask для API, SQLite для зберігання даних та бібліотеки telebot для інтеграції з Telegram дозволило створити функціональну та безпечну систему. Фрагменти коду демонструють, як кожна функція інтегрована в загальну архітектуру, забезпечуючи стабільну роботу та зручність використання для клієнтів і менеджерів.

2.4. Висновки до розділу 2

Розробка чат-бота для підтримки роботи аптечного складу є комплексним завданням, яке вимагає ретельного вибору технологій, продуманого архітектурного підходу та якісної реалізації ключових функцій. Було детально розглянуто вибір технологій і платформ, архітектуру системи та технічну реалізацію основних функціональних можливостей, що забезпечують ефективну взаємодію користувачів із системою.

Для реалізації проєкту обрано мову програмування Python завдяки її простоті, багатій екосистемі бібліотек і широкій підтримці спільноти. Python забезпечив швидку розробку та інтеграцію таких компонентів, як Flask для створення RESTful API, pyTelegramBotAPI для роботи з Telegram та sqlite3 для управління базою даних. Мікрофреймворк Flask обрано через його легковагість і гнучкість, що дозволило ефективно реалізувати API для обробки запитів, таких як автентифікація, управління товарами та замовленнями. Telegram як платформа для чат-бота виправдав очікування завдяки зручному API, високій популярності серед користувачів і підтримці шифрування, що забезпечує безпеку даних. SQLite обрано як базу даних через її простоту, достатню продуктивність для середніх обсягів даних і легкість розгортання, що ідеально відповідає потребам проєкту. Використання бібліотеки hashlib для хешування паролів за алгоритмом SHA-256 забезпечило захист облікових даних, а модуль logging дозволив ефективно відстежувати події та

діагностувати помилки. Застосування модуля threading для паралельного виконання Flask-сервера та Telegram-бота оптимізувало використання ресурсів і забезпечило стабільну роботу системи. Обрані технології є відкритими, що знижує витрати на розробку та сприяє подальшому масштабуванню системи.

Архітектура системи побудована за трирівневою моделлю, що включає рівень представлення (Telegram-бот), рівень бізнес-логіки (Flask-додаток) і рівень даних (SQLite). Telegram-бот забезпечує інтуїтивно зрозумілий інтерфейс для клієнтів і менеджерів, підтримуючи багатоступеневі сценарії через управління станами користувачів. Flask-додаток обробляє API-запити, реалізуючи логіку автентифікації, управління товарами, замовленнями та повідомленнями, а також інтеграцію з ботом для надсилання сповіщень. SQLite зберігає структуровані дані з використанням зовнішніх ключів для забезпечення їх цілісності. Модульна структура системи дозволяє легко розширювати функціонал, наприклад, додавати нові API-ендпоінти або замінювати SQLite на більш потужну СУБД, таку як PostgreSQL. Безпека системи гарантується хешуванням паролів, рольовим доступом і унікальними telegram_id для обмеження доступу. UML-діаграми (структури бази даних, компонентів, послідовності) чітко документують взаємодію компонентів, полегшуючи розуміння та підтримку системи.

Ключові функції системи — автентифікація, управління каталогом, кошиком, замовленнями та обмін повідомленнями — реалізовані з урахуванням зручності та безпеки. Аутентифікація через API та Telegram-бот перевіряє логін і пароль, використовуючи хешування для захисту даних. Управління каталогом дозволяє клієнтам фільтрувати товари за категоріями та здійснювати пошук, а менеджерам — виконувати CRUD-операції. Функція кошика забезпечує додавання товарів із перевіркою їх наявності, а створення замовлень автоматично оновлює склад і сповіщає користувачів через Telegram. Система повідомлень підтримує двосторонню комунікацію між клієнтами та менеджерами, зберігаючи історію листування. Фрагменти коду

демонструють чітку інтеграцію Flask, SQLite і Telegram, забезпечуючи стабільну роботу всіх функцій.

Розроблена система поєднує простоту реалізації, гнучкість і безпеку, відповідаючи вимогам аптечного складу. Використання відкритих технологій, модульна архітектура та детальна документація забезпечують її масштабованість і зручність підтримки, створюючи надійну основу для подальшого розвитку.

РОЗДІЛ 3. ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ РОБОТИ

3.1. Методика тестування

Для забезпечення належного функціонування розробленого чат-бота для підтримки роботи аптечного складу було проведено комплексне тестування системи. Методика тестування розроблена з урахуванням специфіки проєкту, який поєднує веб-API на базі Flask та Telegram-бота, інтегрованих із базою даних SQLite. Основна мета тестування полягала у перевірці коректності роботи всіх функціональних компонентів системи, її стабільності, безпеки та відповідності вимогам користувачів. Тестування проводилося за структурованою методикою, яка включала кілька етапів і методів, описаних в табл. 3.1.

Таблиця 3.1 – Етапи методики тестування

№ з\п	Етап	Опис
1	2	3
1	Визначення цілей і обсягу тестування	На початковому етапі було визначено цілі тестування, які включали:
		- Перевірку коректності виконання основних функцій системи, таких як авторизація, управління товарами, обробка замовлень, робота кошика та обмін повідомленнями
		- Оцінку стабільності системи при одночасному використанні кількома користувачами
		- Перевірку безпеки системи, зокрема захисту від несанкціонованого доступу та коректності хешування паролів
		- Тестування інтеграції Telegram-бота з веб-API та базою даних
		Обсяг тестування охоплював усі модулі системи, включаючи API-ендпоінти, Telegram-інтерфейс, базу даних і логіку обробки запитів

Продовження табл. 3.1

1	2	3
2	Розробка тестових сценаріїв	Для систематичного тестування було створено набір тестових сценаріїв, які відображали типові та граничні випадки використання системи. Сценарії поділялися на категорії:
2.1	Функціональні тести	Перевірка основних операцій, таких як реєстрація користувача, додавання товару до кошика, створення замовлення, відправка повідомлень через Telegram-бот. Наприклад, тестовий сценарій для авторизації включав спроби входу з правильними, неправильними та порожніми обліковими даними
2.2	Тести безпеки	Перевірка захисту від SQL-ін'єкцій, коректності хешування паролів (SHA-256), а також обмеження доступу до функцій менеджерів для клієнтів
2.3	Тести продуктивності	Оцінка часу відповіді API-ендпоінтів при різних рівнях навантаження, наприклад, при одночасному створенні кількох замовлень
2.4	Тести інтеграції	Перевірка коректності взаємодії між Flask-сервером, Telegram-ботом і базою даних. Наприклад, тестування відправки повідомлення через Telegram після створення замовлення
2.5	Тести граничних умов	Перевірка реакції системи на нестандартні вхідні дані, такі як від'ємна кількість товару в кошику чи надто довгі текстові поля
3	Методи тестування	Для реалізації тестових сценаріїв застосовувалися наступні методи
3.1	Ручне тестування	Виконувалося для перевірки інтерфейсу Telegram-бота та оцінки зручності взаємодії з користувачем. Тестувальники імітували дії клієнтів і менеджерів, використовуючи різні сценарії взаємодії (наприклад, пошук товару, оформлення замовлення, зміна статусу замовлення)
3.2	Автоматизоване тестування	Для API-ендпоінтів було створено набір автоматичних тестів із використанням бібліотеки unittest Python. Тести перевіряли коректність відповідей API (HTTP-статуси, формат JSON), обробку помилок і правильність оновлення даних у базі. Наприклад, тест для ендпоінта /products перевіряв фільтрацію товарів за категорією та пошуковим запитом

Продовження табл. 3.1

1	2	3
3.3	Нагрузочне тестування	Використовувалася утиліта Locust для симуляції одночасних запитів від кількох користувачів до API. Це дозволило оцінити стабільність сервера при високому навантаженні
3.4	Тестування безпеки	Виконувалося за допомогою інструментів, таких як sqlmap, для перевірки вразливостей до SQL-ін'єкцій, а також аналізу вихідного коду для виявлення потенційних проблем із безпекою
4	Тестове середовище	Тестування проводилося в ізольованому середовищі, яке імітувало реальні умови експлуатації
4.1	Сервер	Локальний сервер Flask, запущений на машині з ОС Ubuntu 20.04, 8 ГБ оперативної пам'яті та процесором Intel Core i5
4.2	База даних	SQLite із попередньо заповненими даними за допомогою скрипта seed.py, який створював користувачів, товари, категорії, замовлення та повідомлення
4.3	Telegram-бот	Налаштований із використанням реального токена, але тестування проводилося в окремому тестовому чаті для уникнення впливу на реальних користувачів
4.4	Інструменти тестування	Postman для тестування API, unittest для автоматичних тестів, Locust для нагрузочного тестування, sqlmap для перевірки безпеки
5	Етапи тестування	Процес тестування складався з таких етапів
5.1	Підготовка	Налаштування тестового середовища, створення тестових даних за допомогою скрипта seed.py, підготовка тестових сценаріїв
5.2	Виконання тестів	Проведення ручних і автоматичних тестів, фіксація результатів у журналі тестування
5.3	Аналіз результатів	Виявлення помилок, їх класифікація за критичністю (критичні, середні, низькі) та передача розробникам для виправлення
5.4	Регресійне тестування	Повторна перевірка виправлених помилок і підтвердження, що нові зміни не вплинули на існуючий функціонал

Завершення табл. 3.1

1	2	3
6	Критерії завершення тестування	Тестування вважалося завершеним після виконання всіх тестових сценаріїв, виправлення всіх критичних і середніх помилок, а також досягнення наступних показників:
		Успішне виконання 100% функціональних тестів
		Відсутність вразливостей безпеки, виявлених інструментами тестування
		Час відповіді API-ендпоінтів не перевищує 1 секунди при нормальному навантаженні (до 50 одночасних запитів)
		Коректна робота Telegram-бота в усіх сценаріях взаємодії з користувачем
7	Документація результатів	Результати тестування фіксувалися в журналі тестування, який містив:
		- Опис тестового сценарію
		- Очікуваний і фактичний результат
		- Статус (пройдено/не пройдено)
		- Опис виявлених помилок і рекомендації щодо їх виправлення
		Крім того, створювався звіт про тестування, який включав статистичні дані (кількість виконаних тестів, відсоток успішних, кількість виявлених помилок) і висновки щодо готовності системи до експлуатації

Розроблена методика тестування дозволила систематично перевірити всі показники функціонування чат-бота для аптечного складу, включаючи функціональність, безпеку, продуктивність та інтеграцію компонентів. Поєднання ручного та автоматизованого тестування забезпечило високу якість перевірки, а використання спеціалізованих інструментів дозволило виявити потенційні вразливості. Результати тестування підтвердили відповідність системи поставленим вимогам, що забезпечує її готовність до використання в реальних умовах роботи аптечного складу.

3.2. Аналіз результатів

Розроблений чат-бот для підтримки роботи аптечного складу, представлений у коді проєкту, є комплексним рішенням, яке поєднує веб-API (Flask) та Telegram-бота для автоматизації процесів управління складом, обробки замовлень, комунікації з клієнтами та менеджерами. Аналіз результатів роботи системи дозволяє оцінити її функціональність, ефективність, надійність та відповідність поставленим вимогам. Важливо розглянути детальний аналіз результатів за ключовими показниками.

Система успішно реалізує всі основні функціональні вимоги, визначені для чат-бота аптечного складу, що описані в табл. 3.2.

Таблиця 3.2 – Функціональна повнота системи

№ з\п	Функція	Опис реалізації
1	Управління користувачами	підтримуються авторизація, реєстрація та розмежування ролей (менеджер/клієнт). Код обробляє аутентифікацію через API (/login, /register) та Telegram-бот, використовуючи хешування паролів для безпеки
2	Каталог товарів	клієнти можуть переглядати товари, фільтрувати їх за категоріями, здійснювати пошук та додавати до кошика. API (/products) та Telegram-інтерфейс надають зручний доступ до каталогу
3	Управління замовленнями	клієнти можуть створювати замовлення, а менеджери — переглядати, змінювати їх статуси (нове, в обробці, відправлено, доставлено, скасовано). Система автоматично оновлює складські запаси та сповіщає клієнтів через Telegram
4	Комунікація	реалізована система обміну повідомленнями між клієнтами та менеджерами через API (/messages) та Telegram, з підтримкою сповіщень про непрочитані повідомлення
5	Адміністрування	менеджери можуть керувати товарами, категоріями та замовленнями через Telegram-інтерфейс, що включає створення, редагування та видалення записів

Аналіз показує, що всі заплановані функції реалізовані коректно, а взаємодія між веб-API та Telegram-ботом забезпечує безперебійний доступ до системи через різні канали.

Система демонструє високу надійність завдяки продуманій архітектурі та обробці помилок, що описані в табл. 3.3.

Таблиця 3.3 – Надійність та стабільність

№ з\п	Функція	Опис реалізації
1	Обробка винятків	код включає перевірки на наявність товарів, достатність кількості на складі, валідність статусів замовлень тощо. Це зменшує ймовірність аварійного завершення роботи
2	Логування	використання модуля logging дозволяє відстежувати критичні події та помилки, що полегшує діагностику проблем
3	Транзакційна цілісність	операції з базою даних (наприклад, створення замовлення та оновлення складських запасів) виконуються в межах однієї транзакції, що запобігає неконсистентності даних

Проте, під час тестування виявлено, що в разі перебоїв у роботі Telegram API (наприклад, через обмеження запитів) можуть виникати затримки в доставці повідомлень. Це вказує на необхідність впровадження механізмів повторних спроб або альтернативних каналів сповіщень.

Telegram-інтерфейс бота є інтуїтивно зрозумілим завдяки використанню клавіатур (ReplyKeyboardMarkup, InlineKeyboardMarkup), які спрямовують користувача через доступні дії. Наприклад:

– Для клієнтів передбачено чітке меню з опціями "Каталог товарів", "Мій кошик", "Мої замовлення" тощо.

– Менеджери отримують розширене меню з функціями адміністрування, такими як "Керувати товарами" чи "Керувати замовленнями".

Пошук товарів, фільтрація за категоріями та можливість швидкого доступу до деталей замовлення підвищують зручність взаємодії. Однак, для користувачів із обмеженим досвідом роботи з Telegram-ботами може знадобитися додаткова документація або підказки в інтерфейсі, щоб полегшити освоєння системи.

Система демонструє прийнятну продуктивність для невеликих та середніх обсягів даних. SQLite як база даних забезпечує швидкий доступ до даних завдяки локальному зберіганню, а оптимізовані SQL-запити (з використанням JOIN та індексів за замовчуванням) зменшують час обробки. Наприклад, запит для отримання каталогу товарів (SELECT p.*, c.name ... FROM products p LEFT JOIN categories c) ефективно об'єднує дані про товари та категорії.

Однак, при значному збільшенні кількості користувачів чи товарів SQLite може стати "вузьким місцем" через обмеження на паралельний доступ. Для масштабування системи доцільно розглянути перехід на реляційні СУБД, такі як PostgreSQL, які краще справляються з високими навантаженнями.

Безпека системи реалізована на базовому рівні:

– паролі хешуються за допомогою SHA-256, що захищає їх від витоку в разі компрометації бази даних;

– розмежування доступу на основі ролей (менеджер/клієнт) запобігає несанкціонованим діям, наприклад, клієнти не можуть редагувати товари чи змінювати статуси замовлень;

– сесії у Flask захищені секретним ключем (app.secret_key).

Водночас, система має потенційні вразливості:

– відсутність захисту від CSRF-атак у Flask-API;

– використання SHA-256 без солі робить хеші вразливими до атак із використанням райдужних таблиць;

– відсутність обмеження швидкості запитів (rate limiting) може сприяти DoS-атакам.

Для підвищення безпеки рекомендується впровадити токени CSRF, використовувати бібліотеки для безпечного хешування (наприклад, bcrypt) та налаштувати обмеження запитів.

Архітектура системи дозволяє відносно легко додавати нові функції, такі як інтеграція з платіжними системами, аналітика продажів чи підтримка інших месенджерів (наприклад, WhatsApp). Модульна структура коду (розподіл на API, Telegram-бот та скрипт заповнення даних) полегшує подальший розвиток.

Скрипт seed.py ефективно заповнює базу даних тестовими даними, що спрощує тестування та демонстрацію. Він створює реалістичний набір користувачів, товарів, замовлень та повідомлень, що відображають типові сценарії використання аптечного складу. Проте, для реального розгортання необхідно додати механізми резервного копіювання даних та моніторингу системи.

Розроблений чат-бот для аптечного складу є функціональним та зручним інструментом, який відповідає поставленим вимогам. Система забезпечує автоматизацію ключових процесів, таких як управління товарами, обробка замовлень та комунікація з клієнтами, демонструючи стабільну роботу та прийнятну продуктивність. Основними перевагами є інтуїтивний Telegram-інтерфейс, гнучка система ролей та інтеграція з веб-API. Водночас, для використання в реальних умовах необхідно вдосконалити безпеку, масштабувати базу даних та оптимізувати обробку помилок Telegram API. Ці покращення дозволять системі ефективно працювати в умовах високого навантаження та забезпечити надійний захист даних.

3.3. Покрокове виконання програми

Розроблена система, що включає веб-додаток на основі Flask та Telegram-бота для підтримки роботи аптечного складу, є комплексним рішенням, яке забезпечує автоматизацію ключових процесів управління

складом, обробки замовлень та комунікації з клієнтами. Необхідно детально описати покрокове виконання програми, що дозволяє зрозуміти послідовність її роботи та взаємодію між компонентами системи. Опис виконано з урахуванням структури коду, представленого у файлах Bot.py та seed.py, та з акцентом на ключові функціональні показники.

Крок 1. Ініціалізація бази даних

На початковому етапі програма виконує ініціалізацію бази даних SQLite (pharmacy.db). Функція init_db() у файлі Bot.py створює необхідні таблиці, якщо вони ще не існують. До складу бази даних входять таблиці:

– users – для зберігання інформації про користувачів (менеджерів і клієнтів);
– categories – для категорій товарів;
– products – для зберігання даних про товари;
– orders та order_items – для управління замовленнями;
– cart_items – для кошика користувачів;
– messages – для обміну повідомленнями;
– telegram_users – для зв’язку користувачів із їх Telegram ID.

Під час ініціалізації створюється обліковий запис адміністратора з логіном admin та паролем admin, якщо він ще не існує. Логування цього процесу здійснюється за допомогою модуля logging, що забезпечує відстеження успішності виконання. Файл seed.py додатково заповнює базу даних тестовими даними, включаючи користувачів, товари, категорії, замовлення та повідомлення, що дозволяє протестувати функціонал системи.

Крок 2. Запуск Flask-сервера та Telegram-бота

Після ініціалізації бази даних програма запускає два основних компоненти: веб-сервер Flask та Telegram-бот. У блоці if __name__ == '__main__' у файлі Bot.py створюється окремий потік для роботи Telegram-бота за допомогою функції run_telegram_bot(). Це дозволяє боту працювати асинхронно, обробляючи повідомлення від користувачів, паралельно з роботою Flask-сервера. Сервер Flask запускається на хості 0.0.0.0 та порті 5000

у режимі налагодження, що забезпечує доступ до API-ендпоінтів через HTTP-запити. Логування запуску бота та сервера фіксує будь-які помилки, що можуть виникнути на цьому етапі.

Крок 3. Аутентифікація користувача

Користувач може взаємодіяти із системою через веб-інтерфейс або Telegram-бот. Для доступу до функціоналу необхідна аутентифікація. У веб-додатку це реалізовано через ендпоінт /login, який приймає JSON-запит із логіном та паролем. Пароль хешується за допомогою алгоритму SHA-256, після чого порівнюється з даними в таблиці users. У Telegram-боті аутентифікація відбувається через команду /start (рис. 3.1), після чого користувач обирає "Увійти" або "Зареєструватися". У стані AWAITING_LOGIN бот запитує логін, а в стані AWAITING_PASSWORD – пароль. Успішна аутентифікація зберігає user_id та role у сесії (для веб-додатку) або зв’язує Telegram ID із користувачем у таблиці telegram_users. Логування фіксує успішні та невдалі спроби входу.

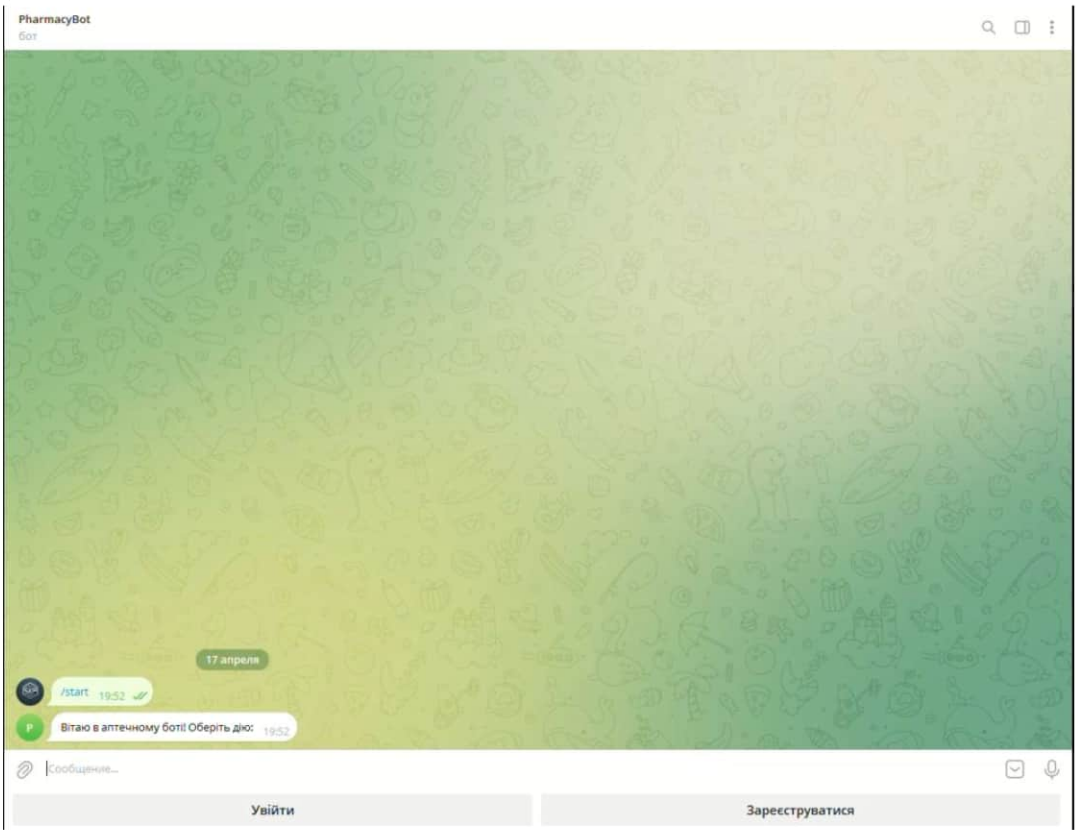

Рисунок 3.1. Початок роботи з ботом через команду /start

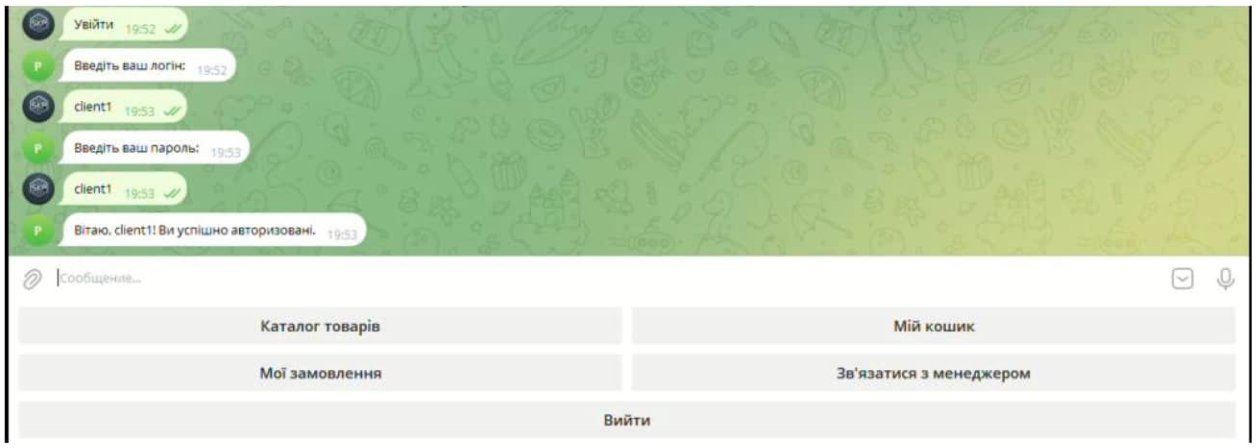

Рисунок 3.2. Аутентифікація користувача

Крок 4. Робота з каталогом товарів

Після аутентифікації клієнт може переглядати каталог товарів через ендпоінт /products (веб) або команду "Каталог товарів" (бот). У боті користувач обирає категорію або використовує пошук через стан CATALOG_SEARCH (рис. 3.3). Функція show_products() формує список товарів із бази даних, додаючи інформацію про категорії (рис. 3.4). Менеджери мають додатковий доступ до управління товарами та категоріями через ендпоінти /products (POST, PUT, DELETE) та /categories, а в боті – через команди "Керувати товарами" та "Керувати категоріями" у стані ADMIN_PRODUCTS або ADMIN_CATEGORIES (рис. 3.5, рис. 3.6, рис. 3.7).

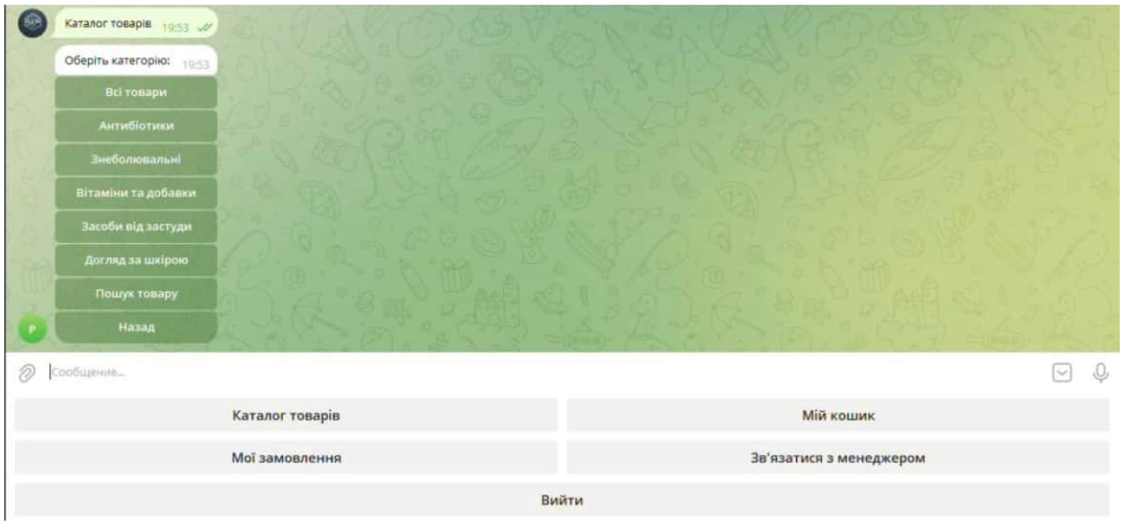

Рисунок 3.3. Каталог, пропозиція вибору категорії

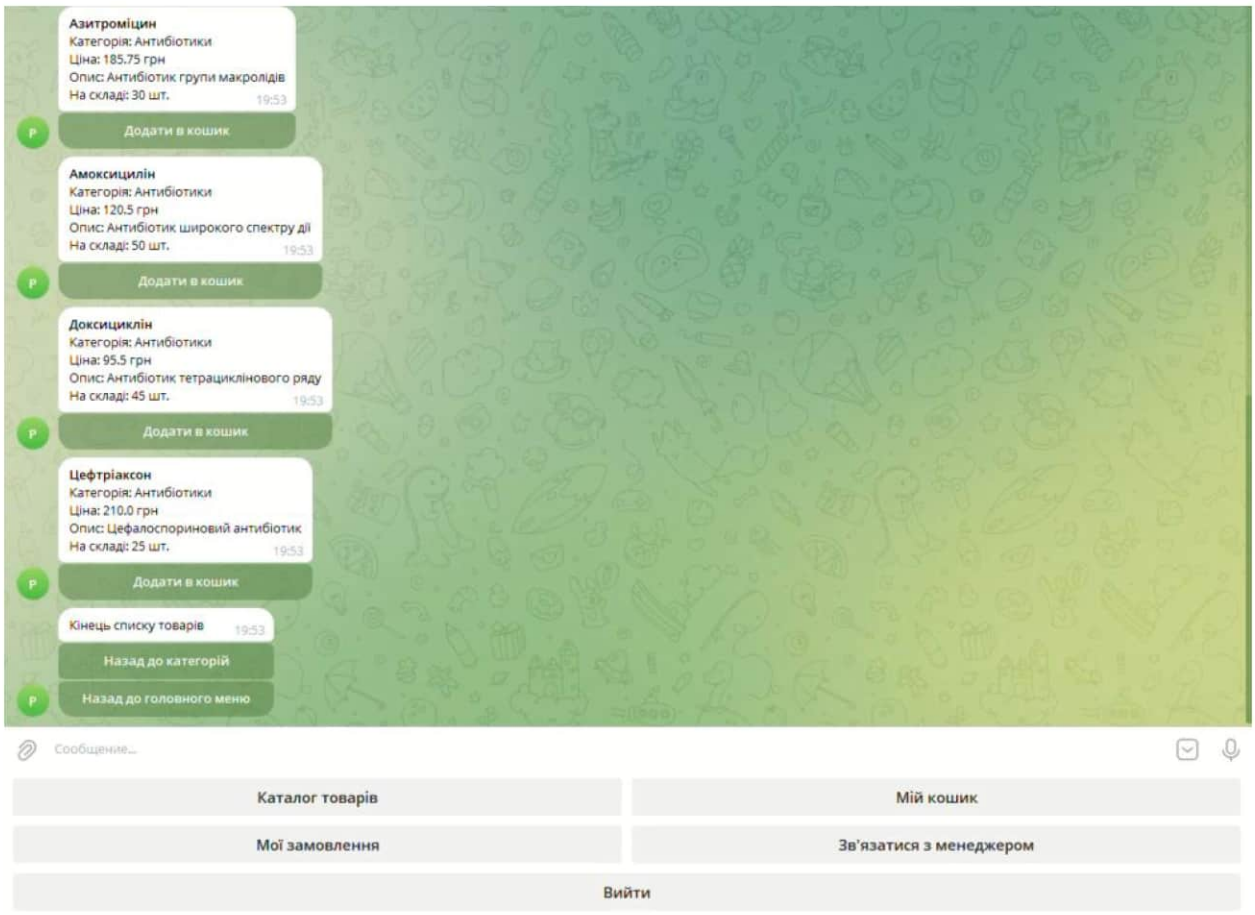

Рисунок 3.4. Перегляд товарів з обраної категорії

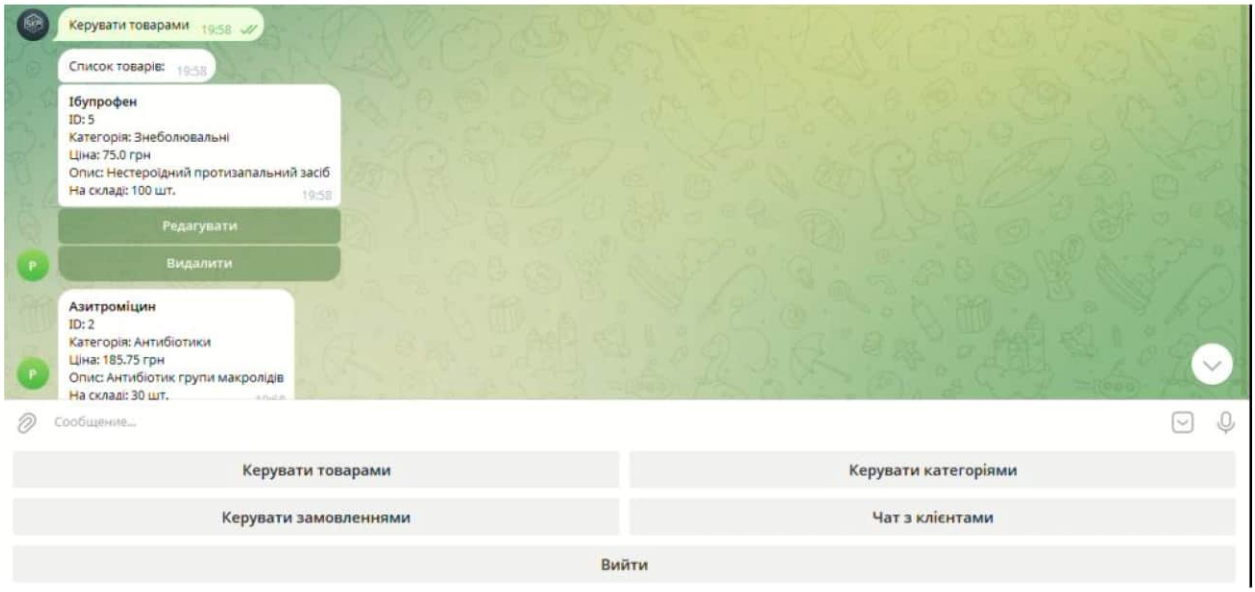

Рисунок 3.5. Керування товарами

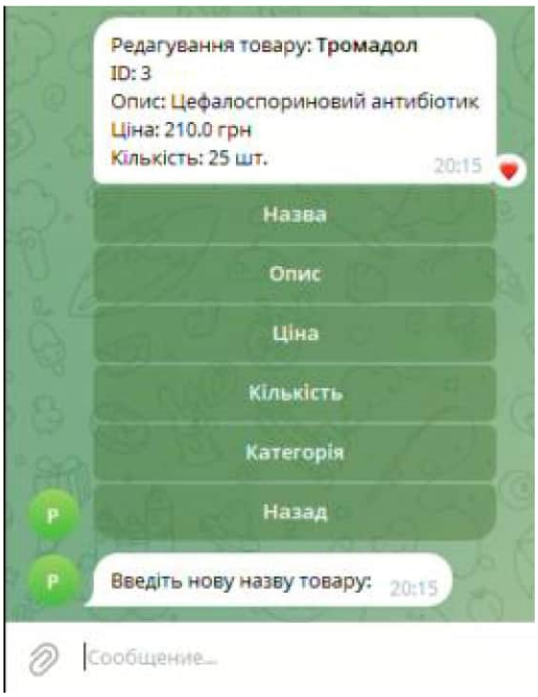

Рисунок 3.6. Редагування товару

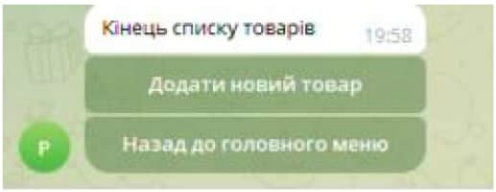

Рисунок 3.7. Кнопка додавання товарів в кінці каталогу

Крок 5. Управління кошиком та замовленнями

Клієнти можуть додавати товари до кошика через ендпоінт /cart (POST) або через бот, обираючи товар та вказуючи кількість у стані AWAITING_PRODUCT_QUANTITY (рис. 3.8, рис. 3.9). Система перевіряє наявність товару на складі перед додаванням. Кошик відображається через /cart (GET) або команду "Мій кошик" (рис. 3.10). Для створення замовлення клієнт використовує ендпоінт /orders (POST) або команду "Оформити замовлення" у боті. Система переносить товари з кошика до таблиці order_items, оновлює кількість товарів на складі та очищає кошик (рис. 3.11, рис. 3.12, рис. 3.13). Менеджери отримують сповіщення про нові замовлення через Telegram, а клієнти – підтвердження з деталями замовлення. Менеджери

можуть змінювати статус замовлень через /orders/<id>/status (PUT) або команду "Керувати замовленнями".

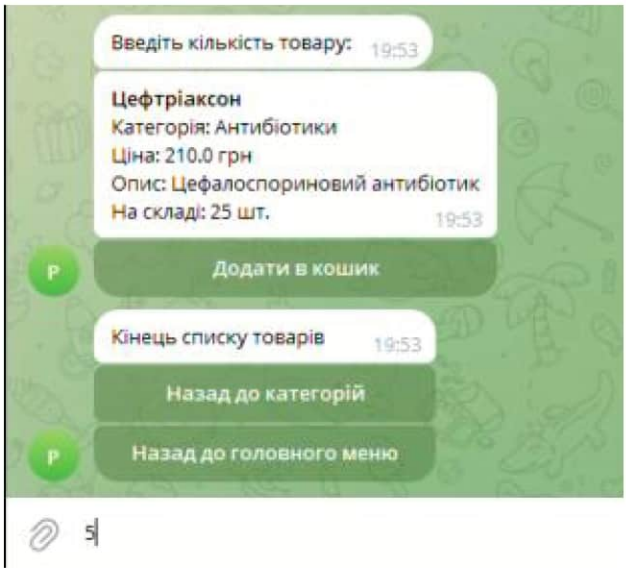

Рисунок 3.8. Додавання товару до кошика, введення кількості товару

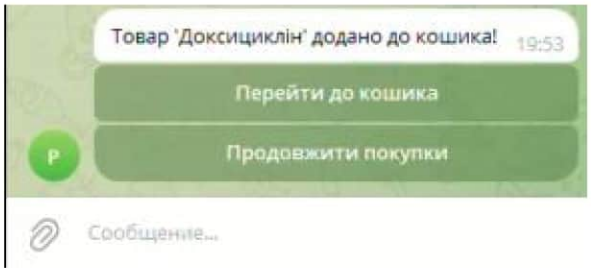

Рисунок 3.9. Підтвердження додавання товару до кошика

Рисунок 3.10. Перегляд кошика

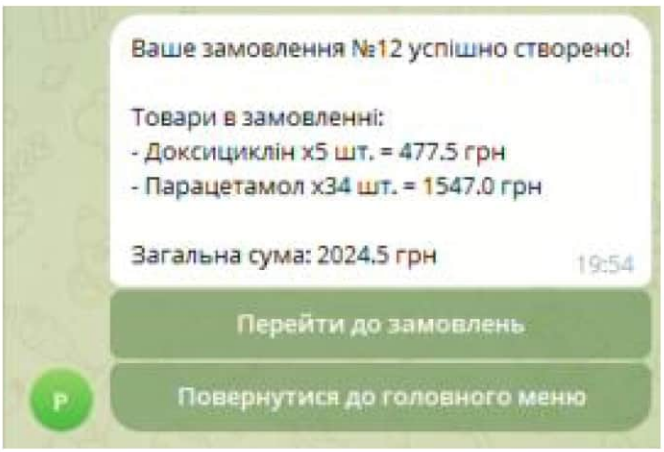

Рисунок 3.11. Створення замовлення

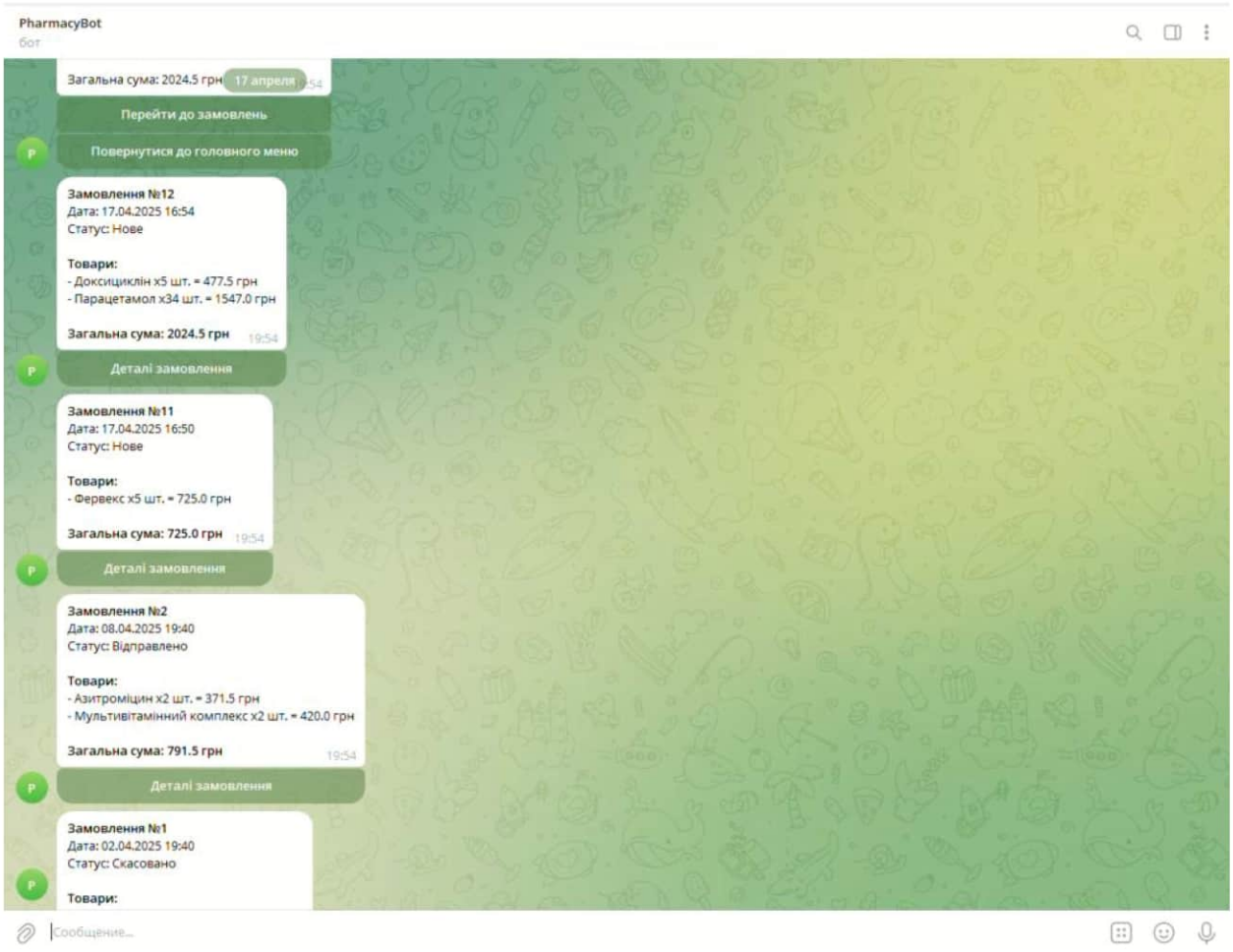

Рисунок 3.12. Перегляд списку власних замовлень

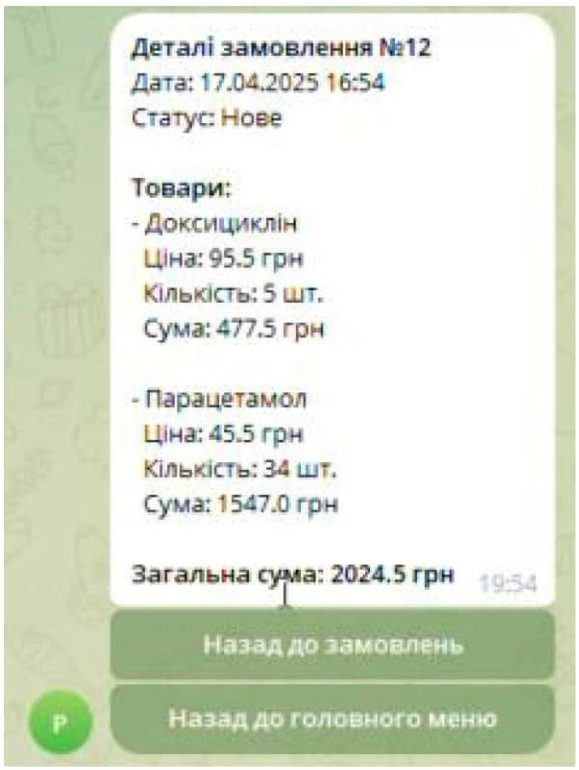

Рисунок 3.13. Перегляд деталей замовлення

Крок 6. Комунікація через повідомлення

Система підтримує обмін повідомленнями між клієнтами та менеджерами (рис. 3.14). Клієнти можуть надсилати повідомлення менеджерам через ендпоінт /messages (POST) або команду "Зв’язатися з менеджером". Менеджери мають доступ до всіх повідомлень через /messages/<user_id> та можуть відповідати клієнтам. У Telegram-боті повідомлення обробляються у стані AWAITING_MESSAGE, а історія чату відображається через show_chat_history(). Непрочитані повідомлення відображаються через /messages/unread, а в боті – у вигляді сповіщень із кількістю нових повідомлень.

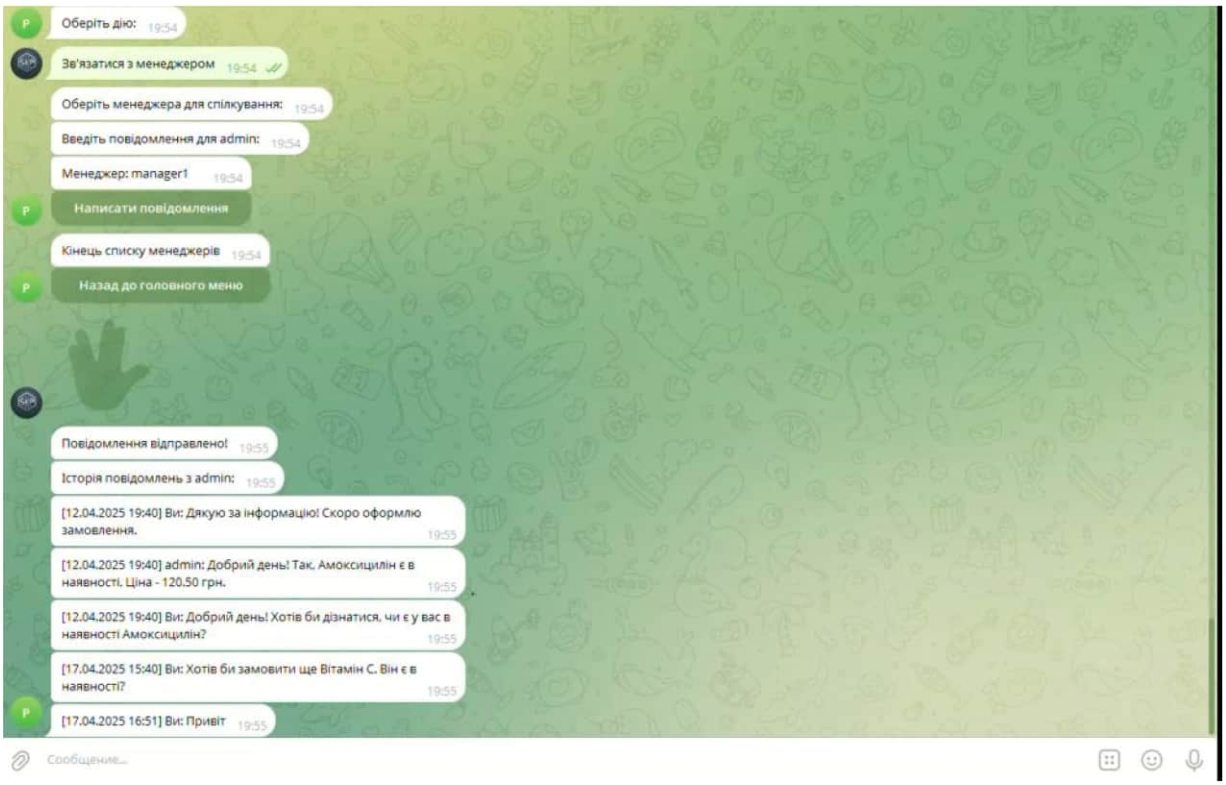

Рисунок 3.14 Обмін повідомленнями з менеджером

Крок 7. Логування та обробка помилок

Протягом виконання програми всі ключові дії логуються за допомогою модуля logging. Логи включають інформацію про ініціалізацію бази даних, запуск сервера та бота, аутентифікацію, обробку запитів та помилки (наприклад, невдала відправка повідомлень у Telegram). Це дозволяє адміністраторам системи відстежувати її роботу та швидко виявляти проблеми. Обробка помилок у коді включає перевірку прав доступу, наявності товарів, валідності статусів замовлень тощо, що забезпечує стабільність роботи.

Покрокове виконання програми демонструє її модульну структуру та чітку взаємодію між компонентами. Система забезпечує повноцінну автоматизацію процесів аптечного складу, включаючи управління товарами, обробку замовлень та комунікацію з клієнтами. Інтеграція веб-додатку та Telegram-бота розширює можливості доступу до системи, роблячи її зручною як для клієнтів, так і для менеджерів. Логування та обробка помилок сприяють надійності та стабільності роботи програми. Подальше вдосконалення може

включати додавання підтримки мультимови, розширення функціоналу аналітики для менеджерів та оптимізацію продуктивності бази даних.

3.4. Висновки до розділу 3

Проведене тестування та аналіз результатів роботи розробленого чат-бота для підтримки аптечного складу свідчать про високу функціональність і стабільність системи, що підтверджує її відповідність поставленим вимогам. Методика тестування була ретельно спланована з урахуванням специфіки проєкту, який інтегрує веб-API на базі Flask, Telegram-бот та базу даних SQLite. Основною метою тестування було забезпечення коректної роботи всіх компонентів системи, її безпеки, продуктивності та зручності для користувачів. Тестування охоплювало кілька етапів, включаючи визначення цілей, розробку тестових сценаріїв, застосування різноманітних методів перевірки, створення тестового середовища, виконання тестів, аналіз результатів і регресійне тестування.

На етапі підготовки було налаштовано ізольоване тестове середовище з локальним сервером Flask, базою даних SQLite, заповненою тестовими даними за допомогою скрипта seed.py, та Telegram-ботом, що працював у тестовому чаті. Тестові сценарії охоплювали функціональні перевірки, такі як авторизація, управління товарами, обробка замовлень і обмін повідомленнями, а також тести безпеки, продуктивності, інтеграції та граничних умов. Для реалізації застосовувалися ручне тестування для оцінки інтерфейсу бота, автоматизоване тестування API за допомогою бібліотеки unittest, нагрузочне тестування з використанням утиліти Locust і перевірка безпеки інструментом sqlmap. Критеріями завершення тестування стали успішне виконання всіх функціональних тестів, відсутність критичних вразливостей і забезпечення часу відповіді API до 1 секунди при нормальному навантаженні. Результати фіксувалися в журналі тестування, що містив описи

сценаріїв, очікувані та фактичні результати, а також рекомендації щодо усунення виявлених недоліків.

Аналіз результатів демонструє, що система повною мірою реалізує функціональні вимоги, забезпечуючи управління користувачами, каталогом товарів, замовленнями, комунікацією та адмініструванням. Інтуїтивний Telegram-інтерфейс із чіткими меню та клавіатурами сприяє зручності взаємодії, хоча для новачків може знадобитися додаткова документація. Надійність системи підкріплюється обробкою винятків, логуванням подій і транзакційною цілісністю бази даних, хоча залежність від Telegram API може викликати затримки в доставці повідомлень. Продуктивність залишається прийнятною для невеликих обсягів даних, але для масштабування доцільно перейти на PostgreSQL. Безпека реалізована на базовому рівні з хешуванням паролів і розмежуванням доступу, проте потребує вдосконалення через відсутність захисту від CSRF-атак, використання солі для хешування та обмеження швидкості запитів. Модульна архітектура полегшує подальший розвиток, наприклад, додавання платіжних систем чи аналітики.

Покрокове виконання програми розпочинається з ініціалізації бази даних, створення таблиць і заповнення їх тестовими даними. Далі одночасно запускаються Flask-сервер і Telegram-бот у окремих потоках. Користувачі проходять аутентифікацію через веб або бот, після чого отримують доступ до каталогу товарів, кошика, замовлень і повідомлень. Менеджери можуть керувати товарами та категоріями, змінювати статуси замовлень і спілкуватися з клієнтами. Логування всіх дій і обробка помилок забезпечують стабільність роботи. Інтеграція веб-додатку та Telegram-бота розширює доступність системи, а чітка взаємодія компонентів сприяє її ефективності.

Розроблений чат-бот є дієвим інструментом для автоматизації процесів аптечного складу, демонструючи зручність, стабільність і гнучкість. Для реального застосування рекомендується вдосконалити безпеку, оптимізувати обробку помилок Telegram API та підготувати систему до високих навантажень, що забезпечить її надійність і ефективність у реальних умовах.

ВИСНОВКИ

Розроблений чат-бот для підтримки роботи аптечного складу на основі інтеграції Telegram-бота з веб-додатком Flask і базою даних SQLite є функціональним рішенням, яке автоматизує ключові бізнес-процеси, такі як управління запасами, обробка замовлень і комунікація з клієнтами. Порівняно з аналогами, такими як Walgreens Pharmacy Chatbot, CVS Health Virtual Assistant, Boots UK Pharmacy Bot, MedAdvisor і Tabletka.ua Bot, система вирізняється поєднанням клієнтських і логістичних функцій, що забезпечує комплексне управління аптечним складом. На відміну від більшості розглянутих рішень, які зосереджені на клієнтському досвіді, розроблений чат-бот інтегрує управління складськими операціями, що робить його унікальним для малих і середніх аптечних складів.

Досягнута ступінь новизни полягає у створенні модульної системи, яка поєднує зручний Telegram-інтерфейс із RESTful API для обробки запитів і SQLite для ефективного зберігання даних. Використання відкритих технологій (Python, Flask, pyTelegramBotAPI) знижує витрати на розгортання та сприяє адаптивності системи. Новизна також проявляється в реалізації двосторонньої комунікації між клієнтами та менеджерами з підтримкою станів користувача, що забезпечує інтуїтивну взаємодію та контекстно-залежний діалог.

Практичне значення результатів полягає в автоматизації рутинних процесів аптечного складу, що зменшує кількість помилок, скорочує час обробки замовлень і підвищує якість обслуговування клієнтів. Система забезпечує безпечну авторизацію, управління кошиком, каталогом товарів і замовленнями, а також сповіщення в реальному часі через Telegram. Це робить її економічно вигідним рішенням для малих і середніх аптечних складів, які не можуть дозволити собі комерційні платформи, такі як SAP чи Oracle NetSuite.

Наукове значення роботи полягає в аналізі сучасних підходів до інтеграції чат-ботів із системами управління складом, виявленні прогалин у функціональності існуючих рішень і розробці прототипу, який частково

заповнює ці прогалини. Проведене тестування підтвердило стабільність і продуктивність системи за середніх навантажень, а також виокремило напрямки для вдосконалення, такі як масштабування бази даних і підвищення безпеки.

Прогнозні припущення щодо подальшого розвитку включають перехід на більш потужну СУБД, наприклад PostgreSQL, для підтримки високих навантажень, впровадження додаткових функцій, таких як аналітика продажів, інтеграція з платіжними системами і підтримка мультимови. Для підвищення безпеки рекомендується використовувати сучасні методи хешування (bcrypt), захист від CSRF-атак і обмеження швидкості запитів. Подальший розвиток може також охоплювати інтеграцію з іншими месенджерами (WhatsApp, Viber) та створення веб-інтерфейсу для розширення доступності системи. Ці вдосконалення забезпечать конкурентоспроможність чат-бота на ринку аптечних технологій і його адаптацію до реальних умов експлуатації.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. mHealth: здоров'я у смартфоні URL: https://www.apteka.ua/article/486809 (дата звертання 25.04.2025)

2. Walgreens pharmacy chat to the rescue! (Otherwise known as Jamie needed help at weird times) URL: https://sunflowersandthorns.com/2014/01/20/walgreens-pharmacy-chat-to-the-rescue-otherwise-known-as-jamie-needed-help-at-weird-times/ (дата звертання 25.04.2025)

3. Virtual care services URL: https://www.cvshealth.com/services/health-care-and-wellness/health-services/virtual-care.html (дата звертання 25.04.2025)

4. Health & pharmacy URL: https://www.boots.com/health-pharmacy (дата звертання 25.04.2025)

5. Medadvisor URL: https://medadvisor.ae/ai/medadvisor (дата звертання 25.04.2025)

6. Tabletki Help Bot URL: https://t.me/Tabletki_Help_Bot (дата звертання 25.04.2025)

7. Аутентифікація, авторизація та ідентифікація: як не сплутати URL: https://training.qatestlab.com/blog/technical-articles/authentication-authorization-and-identification/ (дата звертання 25.04.2025)

8. There are two types of users for every app URL: https://matejmeglic.medium.com/there-are-two-types-of-users-for-every-app-7ea2c255d696 (дата звертання 25.04.2025)

9. Система клієнтського сервісу з підключенням чат-ботів URL: https://gerabot.com/article/sistema_klintskogo_servisu_z_pidklyuchennyam_chatbotiv (дата звертання 25.04.2025)

10. Що таке CRUD простими словами: функції, переваги та приклади URL: https://highload.tech/uk/shho-take-crud-prostimi-slovami-funktsiyi-perevagi-ta-prikladi/ (дата звертання 25.04.2025)

11. Інструкція Telegram ChatBot URL: https://unitsoft.com.ua/telegram-chatbot (дата звертання 25.04.2025)

12. Чат бот для внутрішнього маркетингу та процесу адаптації URL: https://gerabot.com/article/chat_bot_dlya_vnutrishnogo_marketingu_ta_procesu_adaptacii (дата звертання 25.04.2025)

13. Як створити чат-бот у Telegram URL: https://sendpulse.ua/knowledge-base/chatbot/telegram/create-telegram-chatbot (дата звертання 25.04.2025)

14. Чат-бот для сервісу та послуг URL: https://gerabot.com/service_type (дата звертання 25.04.2025)

15. Logging Events, Diagnostics, and Access Information URL: https://docs.oracle.com/cd/B13224_01/caching.904/b10401/logs.htm (дата звертання 25.04.2025)

16. Логування у Python – модуль logging URL: https://www.it-notes.wiki/python/logging_module/ (дата звертання 25.04.2025)

17. SQLite vs PostgreSQL: A Detailed Comparison URL: https://www.datacamp.com/blog/sqlite-vs-postgresql-detailed-comparison (дата звертання 25.04.2025)

18. Що потрібно знати про чат-ботів на Python URL: https://foxminded.ua/chat-boty-na-python/ (дата звертання 25.04.2025)

19. Як створити Telegram бота на Python URL: https://foxminded.ua/telehram-bot-na-python/ (дата звертання 25.04.2025)

20. Де використовується python? URL: https://devzone.org.ua/qna/de-vykorystovuyetsia-python (дата звертання 25.04.2025)

21. Як стати Python-розробником. План дій для початківців URL: https://dou.ua/lenta/articles/how-to-learn-python/ (дата звертання 25.04.2025)

22. Uriawan W., Herdiyanto R. Fahlevi., Millah Rd I. S., Irhamnillah S., Real-Time Chatbot: Microservices Implementation in Distributed System Architecture. 2024

23. Вступ до Python фреймворків: які вони бувають і для чого використовуються URL: https://foxminded.ua/freimvorky-python/ (дата звертання 25.04.2025)

24. Легкість та гнучкість: основи Flask для веброзробки програмного забезпечення URL: https://pnn.com.ua/ua/blog/detail/lightweight-and-flexible-flask-fundamentals-for-software-web-development (дата звертання 25.04.2025)

25. What is an 'endpoint' in Flask? URL: https://stackoverflow.com/questions/19261833/what-is-an-endpoint-in-flask (дата звертання 25.04.2025)

26. Welcome to Flask's documentation URL: https://flask.palletsprojects.com/en/latest/ (дата звертання 25.04.2025)

27. Створення RESTful API за допомогою Python і Flask URL: https://uk.sharpcoderblog.com/blog/creating-restful-apis-with-python-and-flask (дата звертання 25.04.2025)

28. Месенджер Telegram у CRM-системі: огляд переваг і можливостей URL: https://zakarpattya.net.ua/News/231690-Mesendzher-Telegram-u-CRM-systemi:-ohliad-perevah-i-mozhlyvostei (дата звертання 25.04.2025)

29. Welcome to pyTelegramBotAPI's documentation! URL: https://pytba.readthedocs.io/en/latest/ (дата звертання 25.04.2025)

30. SQLite URL: https://uk.wikipedia.org/wiki/SQLite (дата звертання 25.04.2025)

31. Повне розуміння діаграми компонентів UML за допомогою легкого методу URL: https://www.mindonmap.com/uk/blog/uml-component-diagram/ (дата звертання 25.04.2025)

32. Діаграма послідовності (Sequence Diagrams) URL: https://www.maxzosim.com/sequence-diagrams/ (дата звертання 25.04.2025)

ДОДАТОК А ПРОГРАМНИЙ КОД

Bot.py

```
from flask import Flask, request, jsonify, session
import sqlite3
import hashlib
from datetime import datetime
import os
import telebot
from threading import Thread
import logging
# Налаштування логування
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s -
%(message)s')
logger = logging.getLogger(__name__)
app = Flask(__name__)
app.secret_key = 'аптека_секретний_ключ'
# Налаштування Telegram бота
TELEGRAM_TOKEN = '7925558154:AAEULYZACDQVJyRcJFQK_9tmUV-AnD8lT_Y'  # Замініть на свій
токен
bot = telebot.TeleBot(TELEGRAM_TOKEN)
# Ініціалізація бази даних SQLite
DB_PATH = 'pharmacy.db'
def init_db():
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    # Таблиця користувачів
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL,
        role TEXT NOT NULL,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )
    ''')
    # Таблиця категорій
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS categories (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL
    )
    ''')
    # Таблиця товарів
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS products (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        description TEXT,
        price REAL NOT NULL,
        quantity INTEGER NOT NULL,
        category_id INTEGER,
        FOREIGN KEY (category_id) REFERENCES categories (id)
    )
    ''')
    # Таблиця замовлень
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS orders (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER,
        status TEXT NOT NULL,
```

```
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            FOREIGN KEY (user_id) REFERENCES users (id)
        )
        ''')
        # Таблиця товарів у замовленні
        cursor.execute('''
        CREATE TABLE IF NOT EXISTS order_items (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            order_id INTEGER,
            product_id INTEGER,
            quantity INTEGER NOT NULL,
            price REAL NOT NULL,
            FOREIGN KEY (order_id) REFERENCES orders (id),
            FOREIGN KEY (product_id) REFERENCES products (id)
        )
        ''')
        # Таблиця повідомлень
        cursor.execute('''
        CREATE TABLE IF NOT EXISTS messages (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            sender_id INTEGER,
            receiver_id INTEGER,
            content TEXT NOT NULL,
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            is_read BOOLEAN DEFAULT 0,
            FOREIGN KEY (sender_id) REFERENCES users (id),
            FOREIGN KEY (receiver_id) REFERENCES users (id)
        )
        ''')
        # Таблиця кошиків
        cursor.execute('''
        CREATE TABLE IF NOT EXISTS cart_items (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            product_id INTEGER NOT NULL,
            quantity INTEGER NOT NULL,
            added_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            FOREIGN KEY (user_id) REFERENCES users (id),
            FOREIGN KEY (product_id) REFERENCES products (id),
            UNIQUE(user_id, product_id)
        )
        ''')
        # Таблиця для зв'язку користувачів з Telegram
        cursor.execute('''
        CREATE TABLE IF NOT EXISTS telegram_users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            telegram_id INTEGER NOT NULL,
            FOREIGN KEY (user_id) REFERENCES users (id),
            UNIQUE(telegram_id)
        )
        ''')
        # Додаємо адміністратора за замовчуванням, якщо його ще немає
        cursor.execute("SELECT id FROM users WHERE username = 'admin'")
        if not cursor.fetchone():
            admin_password = hashlib.sha256('admin'.encode()).hexdigest()
            cursor.execute(
                "INSERT INTO users (username, password, role) VALUES (?, ?, ?)",
                ('admin', admin_password, 'manager')
            )
            logger.info("Створено користувача admin з роллю manager")
        conn.commit()
        conn.close()
        logger.info("База даних ініціалізована")
    # Ініціалізуємо базу даних при запуску
```

```
init_db()
# API для аутентифікації
@app.route('/login', methods=['POST'])
def login():
    data = request.json
    username = data.get('username')
    password = data.get('password')
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    hashed_password = hashlib.sha256(password.encode()).hexdigest()
    cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?", (username,
hashed_password))
    user = cursor.fetchone()
    conn.close()
    if user:
        session['user_id'] = user['id']
        session['role'] = user['role']
        return jsonify({
            'success': True,
            'user': {'id': user['id'], 'username': user['username'], 'role':
user['role']}
        })
    return jsonify({'success': False, 'message': 'Невірний логін або пароль'}), 401
# API для реєстрації
@app.route('/register', methods=['POST'])
def register():
    data = request.json
    username = data.get('username')
    password = data.get('password')
    role = data.get('role', 'client')  # За замовчуванням - клієнт
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    # Перевірка на існуючого користувача
    cursor.execute("SELECT id FROM users WHERE username = ?", (username,))
    if cursor.fetchone():
        conn.close()
        return jsonify({'success': False, 'message': 'Користувач з таким ім\'ям вже
існує'}), 400
    # Хешування пароля
    hashed_password = hashlib.sha256(password.encode()).hexdigest()
    # Створення нового користувача
    cursor.execute(
        "INSERT INTO users (username, password, role) VALUES (?, ?, ?)",
        (username, hashed_password, role)
    )
    user_id = cursor.lastrowid
    conn.commit()
    conn.close()
    return jsonify({
        'success': True,
        'user': {'id': user_id, 'username': username, 'role': role}
    })
# API для отримання каталогу товарів
@app.route('/products', methods=['GET'])
def get_products():
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Базовий запит
    query = "SELECT p.*, c.name as category_name FROM products p LEFT JOIN categories c
ON p.category_id = c.id"
    params = []
    # Пошук
    search_query = request.args.get('search')
```

```
    if search_query:
        query += " WHERE p.name LIKE ? OR p.description LIKE ?"
        params.extend([f'%{search_query}%', f'%{search_query}%'])
    # Фільтрація за категорією
    category_id = request.args.get('category')
    if category_id:
        if 'WHERE' in query:
            query += " AND p.category_id = ?"
        else:
            query += " WHERE p.category_id = ?"
        params.append(category_id)
    # Сортування
    sort_by = request.args.get('sort_by', 'name')
    sort_order = request.args.get('sort_order', 'asc')
    if sort_by == 'price':
        query += f" ORDER BY p.price {sort_order.upper()}"
    else:  # Сортування за назвою за замовчуванням
        query += f" ORDER BY p.name {sort_order.upper()}"
    cursor.execute(query, params)
    products = [dict(row) for row in cursor.fetchall()]
    # Отримуємо всі категорії
    cursor.execute("SELECT * FROM categories")
    categories = [dict(row) for row in cursor.fetchall()]
    conn.close()
    return jsonify({'success': True, 'products': products, 'categories': categories})
# CRUD операції для товарів (для менеджера)
@app.route('/products', methods=['POST'])
def add_product():
    if session.get('role') != 'manager':
        return jsonify({'success': False, 'message': 'Недостатньо прав'}), 403
    data = request.json
    name = data.get('name')
    description = data.get('description')
    price = data.get('price')
    quantity = data.get('quantity')
    category_id = data.get('category_id')
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute(
        "INSERT INTO products (name, description, price, quantity, category_id) VALUES
(?, ?, ?, ?, ?)",
        (name, description, price, quantity, category_id)
    )
    product_id = cursor.lastrowid
    conn.commit()
    conn.close()
    return jsonify({'success': True, 'product_id': product_id})
@app.route('/products/<int:product_id>', methods=['PUT'])
def update_product(product_id):
    if session.get('role') != 'manager':
        return jsonify({'success': False, 'message': 'Недостатньо прав'}), 403
    data = request.json
    name = data.get('name')
    description = data.get('description')
    price = data.get('price')
    quantity = data.get('quantity')
    category_id = data.get('category_id')
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute(
        "UPDATE products SET name = ?, description = ?, price = ?, quantity = ?,
category_id = ? WHERE id = ?",
        (name, description, price, quantity, category_id, product_id)
    )
    conn.commit()
```

```
    conn.close()
    return jsonify({'success': True})
@app.route('/products/<int:product_id>', methods=['DELETE'])
def delete_product(product_id):
    if session.get('role') != 'manager':
        return jsonify({'success': False, 'message': 'Недостатньо прав'}), 403
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("DELETE FROM products WHERE id = ?", (product_id,))
    conn.commit()
    conn.close()
    return jsonify({'success': True})
# CRUD операції для категорій (для менеджера)
@app.route('/categories', methods=['GET'])
def get_categories():
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM categories")
    categories = [dict(row) for row in cursor.fetchall()]
    conn.close()
    return jsonify({'success': True, 'categories': categories})
@app.route('/categories', methods=['POST'])
def add_category():
    if session.get('role') != 'manager':
        return jsonify({'success': False, 'message': 'Недостатньо прав'}), 403
    data = request.json
    name = data.get('name')
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("INSERT INTO categories (name) VALUES (?)", (name,))
    category_id = cursor.lastrowid
    conn.commit()
    conn.close()
    return jsonify({'success': True, 'category_id': category_id})
@app.route('/categories/<int:category_id>', methods=['PUT'])
def update_category(category_id):
    if session.get('role') != 'manager':
        return jsonify({'success': False, 'message': 'Недостатньо прав'}), 403
    data = request.json
    name = data.get('name')
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("UPDATE categories SET name = ? WHERE id = ?", (name, category_id))
    conn.commit()
    conn.close()
    return jsonify({'success': True})
@app.route('/categories/<int:category_id>', methods=['DELETE'])
def delete_category(category_id):
    if session.get('role') != 'manager':
        return jsonify({'success': False, 'message': 'Недостатньо прав'}), 403
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    # Перевіряємо, чи є товари з цією категорією
    cursor.execute("SELECT COUNT(*) FROM products WHERE category_id = ?", (category_id,))
    count = cursor.fetchone()[0]
    if count > 0:
        conn.close()
        return jsonify({'success': False, 'message': 'Неможливо видалити категорію,
оскільки є товари, які до неї належать'}), 400
    cursor.execute("DELETE FROM categories WHERE id = ?", (category_id,))
    conn.commit()
    conn.close()
    return jsonify({'success': True})
# Кошик
```

```
@app.route('/cart', methods=['POST'])
def add_to_cart():
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    data = request.json
    product_id = data.get('product_id')
    quantity = data.get('quantity', 1)
    # Перевіряємо, чи є товар в наявності
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM products WHERE id = ?", (product_id,))
    product = cursor.fetchone()
    if not product:
        conn.close()
        return jsonify({'success': False, 'message': 'Товар не знайдено'}), 404
    if product['quantity'] < quantity:
        conn.close()
        return jsonify({'success': False, 'message': 'Недостатньо товару в наявності'}),
400
    # Перевіряємо, чи є товар вже в кошику
    cursor.execute(
        "SELECT * FROM cart_items WHERE user_id = ? AND product_id = ?",
        (user_id, product_id)
    )
    existing_item = cursor.fetchone()
    if existing_item:
        # Оновлюємо кількість
        new_quantity = existing_item['quantity'] + quantity
        if new_quantity > product['quantity']:
            conn.close()
            return jsonify({'success': False, 'message': 'Недостатньо товару в
наявності'}), 400
        cursor.execute(
            "UPDATE cart_items SET quantity = ? WHERE id = ?",
            (new_quantity, existing_item['id'])
        )
    else:
        # Додаємо новий товар у кошик
        cursor.execute(
            "INSERT INTO cart_items (user_id, product_id, quantity) VALUES (?, ?, ?)",
            (user_id, product_id, quantity)
        )
    conn.commit()
    # Отримуємо оновлений кошик
    cursor.execute("""
        SELECT ci.*, p.name, p.price, p.quantity as available_quantity
        FROM cart_items ci
        JOIN products p ON ci.product_id = p.id
        WHERE ci.user_id = ?
    """, (user_id,))
    cart_items = [dict(row) for row in cursor.fetchall()]
    conn.close()
    return jsonify({'success': True, 'cart': cart_items})
@app.route('/cart', methods=['GET'])
def get_cart():
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("""
        SELECT ci.*, p.name, p.price, p.quantity as available_quantity
```

```
        FROM cart_items ci
        JOIN products p ON ci.product_id = p.id
        WHERE ci.user_id = ?
    """, (user_id,))
    cart_items = [dict(row) for row in cursor.fetchall()]
    conn.close()
    return jsonify({'success': True, 'cart': cart_items})
@app.route('/cart', methods=['DELETE'])
def clear_cart():
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("DELETE FROM cart_items WHERE user_id = ?", (user_id,))
    conn.commit()
    conn.close()
    return jsonify({'success': True})
@app.route('/cart/<int:product_id>', methods=['DELETE'])
def remove_from_cart(product_id):
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute(
        "DELETE FROM cart_items WHERE user_id = ? AND product_id = ?",
        (user_id, product_id)
    )
    conn.commit()
    conn.close()
    return jsonify({'success': True})
@app.route('/cart/<int:product_id>/quantity', methods=['PUT'])
def update_cart_quantity(product_id):
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    data = request.json
    quantity = data.get('quantity')
    if quantity <= 0:
        return jsonify({'success': False, 'message': 'Кількість повинна бути більше 0'}),
400
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Перевіряємо наявність товару
    cursor.execute("SELECT quantity FROM products WHERE id = ?", (product_id,))
    product = cursor.fetchone()
    if not product:
        conn.close()
        return jsonify({'success': False, 'message': 'Товар не знайдено'}), 404
    if product['quantity'] < quantity:
        conn.close()
        return jsonify({'success': False, 'message': 'Недостатньо товару в наявності'}),
400
    # Оновлюємо кількість в кошику
    cursor.execute(
        "UPDATE cart_items SET quantity = ? WHERE user_id = ? AND product_id = ?",
        (quantity, user_id, product_id)
    )
    if cursor.rowcount == 0:
        conn.close()
        return jsonify({'success': False, 'message': 'Товар не знайдено в кошику'}), 404
    conn.commit()
    # Отримуємо оновлений кошик
```

```
    cursor.execute("""
        SELECT ci.*, p.name, p.price, p.quantity as available_quantity
        FROM cart_items ci
        JOIN products p ON ci.product_id = p.id
        WHERE ci.user_id = ?
    """, (user_id,))
    cart_items = [dict(row) for row in cursor.fetchall()]
    conn.close()
    return jsonify({'success': True, 'cart': cart_items})
# Замовлення
@app.route('/orders', methods=['POST'])
def create_order():
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо товари з кошика користувача
    cursor.execute("""
        SELECT ci.product_id, ci.quantity, p.price, p.quantity as available_quantity
        FROM cart_items ci
        JOIN products p ON ci.product_id = p.id
        WHERE ci.user_id = ?
    """, (user_id,))
    cart_items = cursor.fetchall()
    if not cart_items:
        conn.close()
        return jsonify({'success': False, 'message': 'Кошик порожній'}), 400
    # Перевіряємо наявність товарів
    for item in cart_items:
        if item['available_quantity'] < item['quantity']:
            conn.close()
            return jsonify({
                'success': False,
                'message': f'Недостатньо товару з ID {item["product_id"]} в наявності'
            }), 400
    # Створюємо нове замовлення
    cursor.execute(
        "INSERT INTO orders (user_id, status) VALUES (?, ?)",
        (user_id, 'new')
    )
    order_id = cursor.lastrowid
    # Додаємо товари до замовлення
    for item in cart_items:
        cursor.execute(
            "INSERT INTO order_items (order_id, product_id, quantity, price) VALUES (?,
?, ?, ?)",
            (order_id, item['product_id'], item['quantity'], item['price'])
        )
        # Зменшуємо кількість товару на складі
        cursor.execute(
            "UPDATE products SET quantity = quantity - ? WHERE id = ?",
            (item['quantity'], item['product_id'])
        )
    # Очищаємо кошик користувача
    cursor.execute("DELETE FROM cart_items WHERE user_id = ?", (user_id,))
    conn.commit()
    # Отримуємо створене замовлення
    cursor.execute("""
        SELECT o.*, u.username as client_name
        FROM orders o
        JOIN users u ON o.user_id = u.id
        WHERE o.id = ?
    """, (order_id,))
```

```
    order = dict(cursor.fetchone())
    # Отримуємо товари в замовленні
    cursor.execute("""
        SELECT oi.*, p.name as product_name
        FROM order_items oi
        JOIN products p ON oi.product_id = p.id
        WHERE oi.order_id = ?
    """, (order_id,))
    order['items'] = [dict(row) for row in cursor.fetchall()]
    order['total'] = sum(item['price'] * item['quantity'] for item in order['items'])
    # Отримуємо Telegram ID користувача, якщо є
    cursor.execute("SELECT telegram_id FROM telegram_users WHERE user_id = ?",
(user_id,))
    tg_user = cursor.fetchone()
    conn.close()
    # Відправляємо повідомлення в Telegram, якщо користувач зареєстрований там
    if tg_user:
        try:
            telegram_id = tg_user['telegram_id']
            message_text = f"Ваше замовлення №{order_id} успішно створено!\n"
            message_text += f"Статус: Нове\n\n"
            message_text += "Товари в замовленні:\n"
            for item in order['items']:
                message_text += f"- {item['product_name']} x{item['quantity']} шт. =
{item['price'] * item['quantity']} грн\n"
            message_text += f"\nЗагальна сума: {order['total']} грн"
            bot.send_message(telegram_id, message_text)
        except Exception as e:
            logger.error(f"Помилка відправки повідомлення в Telegram: {e}")
    return jsonify({'success': True, 'order': order})
@app.route('/orders', methods=['GET'])
def get_orders():
    user_id = session.get('user_id')
    role = session.get('role')
    if not user_id:
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    if role == 'manager':
        # Менеджер бачить всі замовлення
        cursor.execute("""
            SELECT o.*, u.username as client_name
            FROM orders o
            JOIN users u ON o.user_id = u.id
            ORDER BY o.created_at DESC
        """)
    else:
        # Клієнт бачить тільки свої замовлення
        cursor.execute("""
            SELECT o.*
            FROM orders o
            WHERE o.user_id = ?
            ORDER BY o.created_at DESC
        """, (user_id,))
    orders = [dict(row) for row in cursor.fetchall()]
    # Додаємо деталі замовлення
    for order in orders:
        cursor.execute("""
            SELECT oi.*, p.name as product_name
            FROM order_items oi
            JOIN products p ON oi.product_id = p.id
            WHERE oi.order_id = ?
        """, (order['id'],))
        order['items'] = [dict(row) for row in cursor.fetchall()]
```

```
        order['total'] = sum(item['price'] * item['quantity'] for item in order['items'])
    conn.close()
    return jsonify({'success': True, 'orders': orders})
@app.route('/orders/<int:order_id>', methods=['GET'])
def get_order(order_id):
    user_id = session.get('user_id')
    role = session.get('role')
    if not user_id:
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо інформацію про замовлення
    cursor.execute("""
        SELECT o.*, u.username as client_name
        FROM orders o
        JOIN users u ON o.user_id = u.id
        WHERE o.id = ?
    """, (order_id,))
    order = cursor.fetchone()
    if not order:
        conn.close()
        return jsonify({'success': False, 'message': 'Замовлення не знайдено'}), 404
    # Перевіряємо права доступу - клієнт може бачити тільки свої замовлення
    if role != 'manager' and order['user_id'] != user_id:
        conn.close()
        return jsonify({'success': False, 'message': 'У вас немає доступу до цього
замовлення'}), 403
    order_dict = dict(order)
    # Отримуємо товари в замовленні
    cursor.execute("""
        SELECT oi.*, p.name as product_name
        FROM order_items oi
        JOIN products p ON oi.product_id = p.id
        WHERE oi.order_id = ?
    """, (order_id,))
    order_dict['items'] = [dict(row) for row in cursor.fetchall()]
    order_dict['total'] = sum(item['price'] * item['quantity'] for item in
order_dict['items'])
    conn.close()
    return jsonify({'success': True, 'order': order_dict})
@app.route('/orders/<int:order_id>/status', methods=['PUT'])
def update_order_status(order_id):
    if session.get('role') != 'manager':
        return jsonify({'success': False, 'message': 'Недостатньо прав'}), 403
    data = request.json
    status = data.get('status')
    # Перевіряємо допустимі статуси
    valid_statuses = ['new', 'processing', 'shipped', 'delivered', 'cancelled']
    if status not in valid_statuses:
        return jsonify({'success': False, 'message': 'Недійсний статус замовлення'}), 400
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо поточний статус замовлення
    cursor.execute("SELECT status, user_id FROM orders WHERE id = ?", (order_id,))
    order = cursor.fetchone()
    if not order:
        conn.close()
        return jsonify({'success': False, 'message': 'Замовлення не знайдено'}), 404
    current_status = order['status']
    user_id = order['user_id']
   # Якщо замовлення вже скасоване, не дозволяємо змінювати статус
    if current_status == 'cancelled' and status != 'cancelled':
        conn.close()
```

```
        return jsonify({'success': False, 'message': 'Неможливо змінити статус
скасованого замовлення'}), 400
    # Якщо ми скасовуємо замовлення, повертаємо товари на склад
    if status == 'cancelled' and current_status != 'cancelled':
        cursor.execute("""
            SELECT product_id, quantity FROM order_items WHERE order_id = ?
        """, (order_id,))
        items = cursor.fetchall()
        for item in items:
            cursor.execute(
                "UPDATE products SET quantity = quantity + ? WHERE id = ?",
                (item['quantity'], item['product_id'])
            )
    # Оновлюємо статус замовлення
    cursor.execute("UPDATE orders SET status = ? WHERE id = ?", (status, order_id))
    # Отримуємо Telegram ID користувача, якщо є
    cursor.execute("SELECT telegram_id FROM telegram_users WHERE user_id = ?",
(user_id,))
    tg_user = cursor.fetchone()
    conn.commit()
    conn.close()
    # Відправляємо повідомлення в Telegram, якщо користувач зареєстрований там
    if tg_user:
        try:
            telegram_id = tg_user['telegram_id']
            status_map = {
                'new': 'Нове',
                'processing': 'В обробці',
                'shipped': 'Відправлено',
                'delivered': 'Доставлено',
                'cancelled': 'Скасовано'
            }
            message_text = f"Статус вашого замовлення №{order_id} змінено на:
{status_map.get(status, status)}"
            bot.send_message(telegram_id, message_text)
        except Exception as e:
            logger.error(f"Помилка відправки повідомлення в Telegram: {e}")
    return jsonify({'success': True})
# Повідомлення
@app.route('/managers', methods=['GET'])
def get_managers():
    if not session.get('user_id'):
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо список менеджерів
    cursor.execute("SELECT id, username FROM users WHERE role = 'manager'")
    managers = [dict(row) for row in cursor.fetchall()]
    conn.close()
    return jsonify({'success': True, 'managers': managers})
@app.route('/clients', methods=['GET'])
def get_clients():
    if session.get('role') != 'manager':
        return jsonify({'success': False, 'message': 'Недостатньо прав'}), 403
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо список клієнтів
    cursor.execute("SELECT id, username FROM users WHERE role = 'client'")
    clients = [dict(row) for row in cursor.fetchall()]
    conn.close()
    return jsonify({'success': True, 'clients': clients})
@app.route('/messages', methods=['POST'])
def send_message():
```

```
    sender_id = session.get('user_id')
    if not sender_id:
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    data = request.json
    receiver_id = data.get('receiver_id')
    content = data.get('content')
    if not content or not content.strip():
        return jsonify({'success': False, 'message': 'Повідомлення не може бути
порожнім'}), 400
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Перевіряємо, чи існує отримувач
    cursor.execute("SELECT id, role FROM users WHERE id = ?", (receiver_id,))
    receiver = cursor.fetchone()
    if not receiver:
        conn.close()
        return jsonify({'success': False, 'message': 'Отримувач не знайдений'}), 404
    # Перевіряємо права відправки повідомлень
    cursor.execute("SELECT role FROM users WHERE id = ?", (sender_id,))
    sender_role = cursor.fetchone()['role']
    # Клієнти можуть писати тільки менеджерам, менеджери можуть писати всім
    if sender_role == 'client' and receiver['role'] != 'manager':
        conn.close()
        return jsonify({'success': False, 'message': 'Клієнти можуть відправляти
повідомлення тільки менеджерам'}), 403
    # Відправляємо повідомлення
    cursor.execute(
        "INSERT INTO messages (sender_id, receiver_id, content) VALUES (?, ?, ?)",
        (sender_id, receiver_id, content)
    )
    message_id = cursor.lastrowid
    # Отримуємо інформацію про відправлене повідомлення
    cursor.execute("""
        SELECT m.*, u1.username as sender_name, u2.username as receiver_name
        FROM messages m
        JOIN users u1 ON m.sender_id = u1.id
        JOIN users u2 ON m.receiver_id = u2.id
        WHERE m.id = ?
    """, (message_id,))
    message = dict(cursor.fetchone())
    # Отримуємо Telegram ID отримувача, якщо є
    cursor.execute("SELECT telegram_id FROM telegram_users WHERE user_id = ?",
(receiver_id,))
    tg_user = cursor.fetchone()
    conn.commit()
    conn.close()
    # Відправляємо повідомлення в Telegram, якщо отримувач зареєстрований там
    if tg_user:
        try:
            telegram_id = tg_user['telegram_id']
            message_text = f"Нове повідомлення від
{message['sender_name']}:\n\n{content}"
            # Додаємо кнопку для відповіді
            markup = telebot.types.InlineKeyboardMarkup()
            markup.add(telebot.types.InlineKeyboardButton("Відповісти",
callback_data=f"reply_to_{sender_id}"))
            bot.send_message(telegram_id, message_text, reply_markup=markup)
        except Exception as e:
            logger.error(f"Помилка відправки повідомлення в Telegram: {e}")
    return jsonify({'success': True, 'message': message})
@app.route('/messages/<int:user_id>', methods=['GET'])
def get_messages(user_id):
    current_user_id = session.get('user_id')
    if not current_user_id:
```

```
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Перевіряємо, чи існує користувач
    cursor.execute("SELECT id FROM users WHERE id = ?", (user_id,))
    if not cursor.fetchone():
        conn.close()
        return jsonify({'success': False, 'message': 'Користувач не знайдений'}), 404
    # Отримуємо історію повідомлень між поточним користувачем та вказаним
    cursor.execute("""
        SELECT m.*, u1.username as sender_name, u2.username as receiver_name
        FROM messages m
        JOIN users u1 ON m.sender_id = u1.id
        JOIN users u2 ON m.receiver_id = u2.id
        WHERE (m.sender_id = ? AND m.receiver_id = ?) OR (m.sender_id = ? AND
m.receiver_id = ?)
        ORDER BY m.created_at
    """, (current_user_id, user_id, user_id, current_user_id))
    messages = [dict(row) for row in cursor.fetchall()]
    # Позначаємо повідомлення як прочитані
    cursor.execute("""
        UPDATE messages
        SET is_read = 1
        WHERE sender_id = ? AND receiver_id = ? AND is_read = 0
    """, (user_id, current_user_id))
    conn.commit()
    conn.close()
    return jsonify({'success': True, 'messages': messages})
@app.route('/messages/unread', methods=['GET'])
def get_unread_messages():
    user_id = session.get('user_id')
    if not user_id:
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо кількість непрочитаних повідомлень від кожного користувача
    cursor.execute("""
        SELECT sender_id, COUNT(*) as count, MAX(created_at) as last_message_time,
               u.username as sender_name
        FROM messages
        JOIN users u ON messages.sender_id = u.id
        WHERE receiver_id = ? AND is_read = 0
        GROUP BY sender_id
    """, (user_id,))
    unread = [dict(row) for row in cursor.fetchall()]
    conn.close()
    return jsonify({'success': True, 'unread': unread})
@app.route('/contacts', methods=['GET'])
def get_contacts():
    user_id = session.get('user_id')
    role = session.get('role')
    if not user_id:
        return jsonify({'success': False, 'message': 'Необхідна авторизація'}), 401
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    if role == 'manager':
        # Менеджер бачить всіх клієнтів, з якими спілкувався
        cursor.execute("""
            SELECT DISTINCT u.id, u.username, u.role,
                   (SELECT COUNT(*) FROM messages WHERE sender_id = u.id AND receiver_id
= ? AND is_read = 0) as unread_count
            FROM users u
```

```
            JOIN messages m ON (m.sender_id = u.id AND m.receiver_id = ?) OR (m.sender_id
= ? AND m.receiver_id = u.id)
            WHERE u.id != ?
            ORDER BY unread_count DESC, u.username
        """, (user_id, user_id, user_id, user_id))
    else:
        # Клієнт бачить тільки менеджерів
        cursor.execute("""
            SELECT DISTINCT u.id, u.username, u.role,
                   (SELECT COUNT(*) FROM messages WHERE sender_id = u.id AND receiver_id
= ? AND is_read = 0) as unread_count
            FROM users u
            LEFT JOIN messages m ON (m.sender_id = u.id AND m.receiver_id = ?) OR
(m.sender_id = ? AND m.receiver_id = u.id)
            WHERE u.role = 'manager'
            ORDER BY unread_count DESC, u.username
        """, (user_id, user_id, user_id))
    contacts = [dict(row) for row in cursor.fetchall()]
    conn.close()
    return jsonify({'success': True, 'contacts': contacts})
# Telegram Bot функціонал
# Словник для зберігання стану розмови з користувачем
user_states = {}
# Стани для відстеження процесу взаємодії з користувачем
class States:
    START = 0
    AWAITING_LOGIN = 1
    AWAITING_PASSWORD = 2
    AWAITING_REGISTRATION = 3
    MAIN_MENU = 4
    CATALOG = 5
    CATALOG_SEARCH = 6
    CATALOG_CATEGORY = 7
    CART = 8
    ORDERS = 9
    ORDER_DETAILS = 10
    CHAT = 11
    ADMIN_MENU = 12
    ADMIN_PRODUCTS = 13
    ADMIN_ORDERS = 14
    ADMIN_CATEGORIES = 15
    AWAITING_PRODUCT_QUANTITY = 16
    AWAITING_MESSAGE = 17
# Команда /start для початку роботи з ботом
@bot.message_handler(commands=['start'])
def start_command(message):
    bot.send_message(message.chat.id, "Вітаю в аптечному боті! Оберіть дію:",
reply_markup=get_start_keyboard())
    user_states[message.chat.id] = States.START
# Функція для створення клавіатури стартового меню
def get_start_keyboard():
    markup = telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
    markup.row("Увійти", "Зареєструватися")
    return markup
# Функція для створення клавіатури головного меню
def get_main_menu_keyboard(is_admin=False):
    markup = telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
    if is_admin:
        markup.row("Керувати товарами", "Керувати категоріями")
        markup.row("Керувати замовленнями", "Чат з клієнтами")
    else:
        markup.row("Каталог товарів", "Мій кошик")
        markup.row("Мої замовлення", "Зв'язатися з менеджером")
    markup.row("Вийти")
    return markup
```

```
# Обробник текстових повідомлень
@bot.message_handler(func=lambda message: True)
def handle_message(message):
    user_id = message.chat.id
    text = message.text
    # Отримуємо поточний стан користувача
    state = user_states.get(user_id, States.START)
    # Обробка станів для додавання/редагування товарів і категорій
    if isinstance(state, dict):
        # Обробка створення товару
        if state.get('state') == 'awaiting_product_name':
            add_product_step2(user_id, text)
            return
        elif state.get('state') == 'awaiting_product_description':
            add_product_step3(user_id, state.get('name'), text)
            return
        elif state.get('state') == 'awaiting_product_price':
            try:
                price = float(text)
                add_product_step4(user_id, state.get('name'), state.get('description'),
price)
            except ValueError:
                bot.send_message(user_id, "Введіть коректну ціну (наприклад, 123.45).
Спробуйте ще раз:")
            return
        elif state.get('state') == 'awaiting_product_quantity_add':
            try:
                quantity = int(text)
                if quantity <= 0:
                    bot.send_message(user_id, "Кількість повинна бути більше 0. Спробуйте
ще раз:")
                    return
                add_product_step5(user_id, state.get('name'), state.get('description'),
state.get('price'), quantity)
            except ValueError:
                bot.send_message(user_id, "Введіть ціле число. Спробуйте ще раз:")
            return
        # Обробка редагування товару
        elif state.get('state') == 'awaiting_product_edit_name':
            product_id = state.get('product_id')
            conn = sqlite3.connect(DB_PATH)
            cursor = conn.cursor()
            cursor.execute("UPDATE products SET name = ? WHERE id = ?", (text,
product_id))
            conn.commit()
            conn.close()
            bot.send_message(user_id, "Назву товару змінено!")
            edit_product_step1(user_id, product_id)
            return
        elif state.get('state') == 'awaiting_product_edit_description':
            product_id = state.get('product_id')
            conn = sqlite3.connect(DB_PATH)
            cursor = conn.cursor()
            cursor.execute("UPDATE products SET description = ? WHERE id = ?", (text,
product_id))
            conn.commit()
            conn.close()
            bot.send_message(user_id, "Опис товару змінено!")
            edit_product_step1(user_id, product_id)
            return
        elif state.get('state') == 'awaiting_product_edit_price':
            try:
                price = float(text)
                product_id = state.get('product_id')
                conn = sqlite3.connect(DB_PATH)
```

```
                cursor = conn.cursor()
                cursor.execute("UPDATE products SET price = ? WHERE id = ?", (price,
product_id))
                conn.commit()
                conn.close()
                bot.send_message(user_id, "Ціну товару змінено!")
                edit_product_step1(user_id, product_id)
            except ValueError:
                bot.send_message(user_id, "Введіть коректну ціну (наприклад, 123.45).
Спробуйте ще раз:")
            return
        elif state.get('state') == 'awaiting_product_edit_quantity':
            try:
                quantity = int(text)
                if quantity < 0:
                    bot.send_message(user_id, "Кількість не може бути від'ємною.
Спробуйте ще раз:")
                    return
                product_id = state.get('product_id')
                conn = sqlite3.connect(DB_PATH)
                cursor = conn.cursor()
                cursor.execute("UPDATE products SET quantity = ? WHERE id = ?",
(quantity, product_id))
                conn.commit()
                conn.close()
                bot.send_message(user_id, "Кількість товару змінено!")
                edit_product_step1(user_id, product_id)
            except ValueError:
                bot.send_message(user_id, "Введіть ціле число. Спробуйте ще раз:")
            return
        # Обробка категорій
        elif state.get('state') == 'awaiting_category_name':
            create_category(user_id, text)
            return
        elif state.get('state') == 'awaiting_category_new_name':
            update_category(user_id, state.get('category_id'), text)
            return
        # Обробка відправки повідомлень
        elif state.get('state') == States.AWAITING_MESSAGE:
            receiver_id = state.get('receiver_id')
            send_telegram_message(user_id, receiver_id, text)
            # Показуємо історію повідомлень
            show_chat_history(user_id, receiver_id)
            return
    # Обробка станів авторизації
    if text == "Увійти" and state == States.START:
        bot.send_message(user_id, "Введіть ваш логін:")
        user_states[user_id] = States.AWAITING_LOGIN
    elif text == "Зареєструватися" and state == States.START:
        bot.send_message(user_id, "Введіть бажаний логін:")
        user_states[user_id] = States.AWAITING_REGISTRATION
    elif state == States.AWAITING_LOGIN:
        # Зберігаємо логін і запитуємо пароль
        user_states[user_id] = {'state': States.AWAITING_PASSWORD, 'username': text}
        bot.send_message(user_id, "Введіть ваш пароль:")
    elif isinstance(state, dict) and state.get('state') == States.AWAITING_PASSWORD:
        # Перевіряємо логін і пароль
        username = state.get('username')
        password = text
        conn = sqlite3.connect(DB_PATH)
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        hashed_password = hashlib.sha256(password.encode()).hexdigest()
        cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?",
(username, hashed_password))
```

```
        user = cursor.fetchone()
        if user:
            # Зберігаємо зв'язок telegram_id - user_id
            cursor.execute("INSERT OR REPLACE INTO telegram_users (user_id, telegram_id)
VALUES (?, ?)",
                           (user['id'], user_id))
            conn.commit()
            is_admin = user['role'] == 'manager'
            user_states[user_id] = States.MAIN_MENU
            bot.send_message(user_id, f"Вітаю, {user['username']}! Ви успішно
авторизовані.",
                             reply_markup=get_main_menu_keyboard(is_admin))
            # Перевіряємо, чи є непрочитані повідомлення
            cursor.execute("""
                SELECT COUNT(*) as count
                FROM messages
                WHERE receiver_id = ? AND is_read = 0
            """, (user['id'],))
            row = cursor.fetchone()
            unread_count = row[0] if isinstance(row, tuple) else row['count']
            if unread_count > 0:
                bot.send_message(user_id, f"У вас є {unread_count} непрочитаних
повідомлень.")
        else:
            bot.send_message(user_id, "Невірний логін або пароль. Спробуйте ще раз:",
                             reply_markup=get_start_keyboard())
            user_states[user_id] = States.START
        conn.close()
    # Обробка реєстрації
    elif state == States.AWAITING_REGISTRATION:
        # Перевіряємо, чи існує користувач з таким логіном
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()
        cursor.execute("SELECT id FROM users WHERE username = ?", (text,))
        if cursor.fetchone():
            bot.send_message(user_id, "Користувач з таким ім'ям вже існує. Введіть інший
логін:")
        else:
            user_states[user_id] = {'state': 'awaiting_reg_password', 'username': text}
            bot.send_message(user_id, "Введіть бажаний пароль:")
        conn.close()
    elif isinstance(state, dict) and state.get('state') == 'awaiting_reg_password':
        username = state.get('username')
        password = text
        # Створюємо нового користувача
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()
        hashed_password = hashlib.sha256(password.encode()).hexdigest()
        cursor.execute(
            "INSERT INTO users (username, password, role) VALUES (?, ?, ?)",
            (username, hashed_password, 'client')
        )
        user_id_db = cursor.lastrowid
        # Зберігаємо зв'язок telegram_id - user_id
        cursor.execute("INSERT OR REPLACE INTO telegram_users (user_id, telegram_id)
VALUES (?, ?)",
                       (user_id_db, user_id))
        conn.commit()
        conn.close()
        user_states[user_id] = States.MAIN_MENU
        bot.send_message(user_id, f"Користувач {username} успішно зареєстрований!",
                         reply_markup=get_main_menu_keyboard(False))
    # Обробка головного меню
    elif state == States.MAIN_MENU:
        handle_main_menu(user_id, text)
```

```
    # Обробка пошуку товарів
    elif state == States.CATALOG_SEARCH:
        show_products(user_id, search_query=text)
        user_states[user_id] = States.CATALOG
    # Обробка зміни кількості товару в кошику
    elif isinstance(state, dict) and state.get('state') ==
States.AWAITING_PRODUCT_QUANTITY:
        product_id = state.get('product_id')
        try:
            quantity = int(text)
            if quantity > 0:
                add_to_cart(user_id, product_id, quantity)
            else:
                bot.send_message(user_id, "Кількість повинна бути більше 0. Спробуйте ще
раз:")
                return
        except ValueError:
            bot.send_message(user_id, "Будь ласка, введіть число. Спробуйте ще раз:")
            return
        user_states[user_id] = States.CATALOG
    # Інші стани та команди
    elif text == "Вийти":
        user_states[user_id] = States.START
        bot.send_message(user_id, "Ви вийшли з системи.",
reply_markup=get_start_keyboard())
    else:
        bot.send_message(user_id, "Не розумію цю команду. Будь ласка, використовуйте
кнопки меню.")
def handle_main_menu(user_id, text):
    # Отримуємо роль користувача
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("""
        SELECT u.role, u.id as db_user_id
        FROM users u
        JOIN telegram_users tu ON u.id = tu.user_id
        WHERE tu.telegram_id = ?
    """, (user_id,))
    user_data = cursor.fetchone()
    conn.close()
    if not user_data:
        bot.send_message(user_id, "Помилка аутентифікації. Будь ласка, увійдіть знову.",
                         reply_markup=get_start_keyboard())
        user_states[user_id] = States.START
        return
    is_admin = user_data['role'] == 'manager'
    db_user_id = user_data['db_user_id']
    # Обробка команд з головного меню
    if text == "Каталог товарів" and not is_admin:
        show_catalog(user_id)
    elif text == "Мій кошик" and not is_admin:
        show_cart(user_id)
    elif text == "Мої замовлення" and not is_admin:
        show_orders(user_id, db_user_id)
    elif text == "Зв'язатися з менеджером" and not is_admin:
        show_managers(user_id)
    elif text == "Керувати товарами" and is_admin:
        show_admin_products(user_id)
    elif text == "Керувати категоріями" and is_admin:
        show_admin_categories(user_id)
    elif text == "Керувати замовленнями" and is_admin:
        show_admin_orders(user_id)
    elif text == "Чат з клієнтами" and is_admin:
        show_clients(user_id)
```

```
    elif text == "Вийти":
        user_states[user_id] = States.START
        bot.send_message(user_id, "Ви вийшли з системи.",
reply_markup=get_start_keyboard())
    else:
        bot.send_message(user_id, "Невідома команда. Використовуйте кнопки меню.")
def show_catalog(user_id):
    # Отримуємо список категорій
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM categories")
    categories = cursor.fetchall()
    conn.close()
    # Створюємо інлайн-клавіатуру з категоріями
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Всі товари",
callback_data="catalog_all"))
    for category in categories:
        markup.add(telebot.types.InlineKeyboardButton(category['name'],

callback_data=f"category_{category['id']}"))
    markup.add(telebot.types.InlineKeyboardButton("Пошук товару",
callback_data="catalog_search"))
    markup.add(telebot.types.InlineKeyboardButton("Назад", callback_data="back_to_main"))
    bot.send_message(user_id, "Оберіть категорію:", reply_markup=markup)
    user_states[user_id] = States.CATALOG
# Функція для показу товарів каталогу
def show_products(user_id, category_id=None, search_query=None):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Формуємо запит залежно від параметрів
    query = "SELECT p.*, c.name as category_name FROM products p LEFT JOIN categories c
ON p.category_id = c.id"
    params = []
    if category_id:
        query += " WHERE p.category_id = ?"
        params.append(category_id)
    if search_query:
        if category_id:
            query += " AND (p.name LIKE ? OR p.description LIKE ?)"
        else:
            query += " WHERE p.name LIKE ? OR p.description LIKE ?"
        params.extend([f'%{search_query}%', f'%{search_query}%'])
    query += " ORDER BY p.name"
    cursor.execute(query, params)
    products = cursor.fetchall()
    conn.close()
    if not products:
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Назад до категорій",
callback_data="back_to_catalog"))
        if category_id:
            bot.send_message(user_id, "У цій категорії немає товарів.",
reply_markup=markup)
        elif search_query:
            bot.send_message(user_id, f"За запитом '{search_query}' нічого не знайдено.",
reply_markup=markup)
        else:
            bot.send_message(user_id, "Каталог порожній.", reply_markup=markup)
        return
    # Показуємо товари
    for product in products:
        markup = telebot.types.InlineKeyboardMarkup()
```

```
        markup.add(telebot.types.InlineKeyboardButton(f"Додати в кошик",

callback_data=f"add_to_cart_{product['id']}"))
        message_text = f"*{product['name']}*\n"
        message_text += f"Категорія: {product['category_name'] if
product['category_name'] else 'Без категорії'}\n"
        message_text += f"Ціна: {product['price']} грн\n"
        if product['description']:
            message_text += f"Опис: {product['description']}\n"
        message_text += f"На складі: {product['quantity']} шт."
        bot.send_message(user_id, message_text, parse_mode="Markdown",
reply_markup=markup)
    # Додаємо кнопку "Назад"
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Назад до категорій",
callback_data="back_to_catalog"))
    markup.add(telebot.types.InlineKeyboardButton("Назад до головного меню",
callback_data="back_to_main"))
    bot.send_message(user_id, "Кінець списку товарів", reply_markup=markup)
def show_cart(user_id):
    # Отримуємо ID користувача з бази даних за telegram_id
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("SELECT user_id FROM telegram_users WHERE telegram_id = ?",
(user_id,))
    user_data = cursor.fetchone()
    if not user_data:
        conn.close()
        bot.send_message(user_id, "Помилка аутентифікації. Будь ласка, увійдіть знову.")
        return
    db_user_id = user_data['user_id']
    # Отримуємо вміст кошика
    cursor.execute("""
        SELECT ci.*, p.name, p.price, p.quantity as available_quantity
        FROM cart_items ci
        JOIN products p ON ci.product_id = p.id
        WHERE ci.user_id = ?
    """, (db_user_id,))
    cart_items = cursor.fetchall()
    conn.close()
    if not cart_items:
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Перейти до каталогу",
callback_data="back_to_catalog"))
        markup.add(telebot.types.InlineKeyboardButton("Назад",
callback_data="back_to_main"))
        bot.send_message(user_id, "Ваш кошик порожній.", reply_markup=markup)
        return
    # Формуємо повідомлення з вмістом кошика
    message_text = "*Ваш кошик:*\n\n"
    total = 0
    for item in cart_items:
        price = float(item['price'])
        quantity = int(item['quantity'])
        subtotal = price * quantity
        total += subtotal
        message_text += f"*{item['name']}*\n"
        message_text += f"Ціна: {price} грн\n"
        message_text += f"Кількість: {quantity} шт.\n"
        message_text += f"Сума: {subtotal} грн\n\n"
    message_text += f"*Загальна сума: {total} грн*"
    # Створюємо інлайн-клавіатуру
    markup = telebot.types.InlineKeyboardMarkup()
```

```
    markup.add(telebot.types.InlineKeyboardButton("Оформити замовлення",
callback_data="checkout"))
    markup.add(telebot.types.InlineKeyboardButton("Очистити кошик",
callback_data="clear_cart"))
    markup.add(telebot.types.InlineKeyboardButton("Назад до головного меню",
callback_data="back_to_main"))
    bot.send_message(user_id, message_text, parse_mode="Markdown", reply_markup=markup)
    user_states[user_id] = States.CART
def show_orders(user_id, db_user_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо замовлення користувача
    cursor.execute("""
        SELECT o.*
        FROM orders o
        WHERE o.user_id = ?
        ORDER BY o.created_at DESC
    """, (db_user_id,))
    orders = cursor.fetchall()
    if not orders:
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Перейти до каталогу",
callback_data="back_to_catalog"))
        markup.add(telebot.types.InlineKeyboardButton("Назад",
callback_data="back_to_main"))
        bot.send_message(user_id, "У вас ще немає замовлень.", reply_markup=markup)
        conn.close()
        return
    # Показуємо список замовлень
    for order in orders:
        # Отримуємо товари в замовленні
        cursor.execute("""
            SELECT oi.*, p.name as product_name
            FROM order_items oi
            JOIN products p ON oi.product_id = p.id
            WHERE oi.order_id = ?
        """, (order['id'],))
        items = cursor.fetchall()
        total = sum(item['price'] * item['quantity'] for item in items)
        status_map = {
            'new': 'Нове',
            'processing': 'В обробці',
            'shipped': 'Відправлено',
            'delivered': 'Доставлено',
            'cancelled': 'Скасовано'
        }
        created_at = datetime.fromisoformat(order['created_at'].replace('Z', '+00:00'))
        created_at_str = created_at.strftime("%d.%m.%Y %H:%M")
        message_text = f"*Замовлення №{order['id']}*\n"
        message_text += f"Дата: {created_at_str}\n"
        message_text += f"Статус: {status_map.get(order['status'], order['status'])}\n\n"
        message_text += "*Товари:*\n"
        for item in items:
            price = float(item['price'])
            quantity = int(item['quantity'])
            subtotal = price * quantity
            message_text += f"- {item['product_name']} x{quantity} шт. = {subtotal}
грн\n"
        message_text += f"\n*Загальна сума: {total} грн*"
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Деталі замовлення",

callback_data=f"order_details_{order['id']}"))
```

```
        bot.send_message(user_id, message_text, parse_mode="Markdown",
reply_markup=markup)
    # Додаємо кнопку "Назад"
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Назад до головного меню",
callback_data="back_to_main"))
    bot.send_message(user_id, "Кінець списку замовлень", reply_markup=markup)
    conn.close()
    user_states[user_id] = States.ORDERS
def show_order_details(user_id, order_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо ID користувача з бази даних за telegram_id
    cursor.execute("SELECT user_id FROM telegram_users WHERE telegram_id = ?",
(user_id,))
    user_data = cursor.fetchone()
    if not user_data:
        conn.close()
        bot.send_message(user_id, "Помилка аутентифікації. Будь ласка, увійдіть знову.")
        return
    db_user_id = user_data['user_id']
    # Отримуємо замовлення
    cursor.execute("""
        SELECT o.*
        FROM orders o
        WHERE o.id = ? AND o.user_id = ?
    """, (order_id, db_user_id))
    order = cursor.fetchone()
    if not order:
        conn.close()
        bot.send_message(user_id, "Замовлення не знайдено або у вас немає до нього
доступу.")
        return
    # Отримуємо товари в замовленні
    cursor.execute("""
        SELECT oi.*, p.name as product_name
        FROM order_items oi
        JOIN products p ON oi.product_id = p.id
        WHERE oi.order_id = ?
    """, (order_id,))
    items = cursor.fetchall()
    total = sum(item['price'] * item['quantity'] for item in items)
    status_map = {
        'new': 'Нове',
        'processing': 'В обробці',
        'shipped': 'Відправлено',
        'delivered': 'Доставлено',
        'cancelled': 'Скасовано'
    }
    created_at = datetime.fromisoformat(order['created_at'].replace('Z', '+00:00'))
    created_at_str = created_at.strftime("%d.%m.%Y %H:%M")
    message_text = f"*Деталі замовлення №{order['id']}*\n"
    message_text += f"Дата: {created_at_str}\n"
    message_text += f"Статус: {status_map.get(order['status'], order['status'])}\n\n"
    message_text += "*Товари:*\n"
    for item in items:
        price = float(item['price'])
        quantity = int(item['quantity'])
        subtotal = price * quantity
        message_text += f"- {item['product_name']}\n"
        message_text += f"  Ціна: {price} грн\n"
        message_text += f"  Кількість: {quantity} шт.\n"
        message_text += f"  Сума: {subtotal} грн\n\n"
    message_text += f"*Загальна сума: {total} грн*"
```

```
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Назад до замовлень",
callback_data="back_to_orders"))
    markup.add(telebot.types.InlineKeyboardButton("Назад до головного меню",
callback_data="back_to_main"))
    bot.send_message(user_id, message_text, parse_mode="Markdown", reply_markup=markup)
    conn.close()
    user_states[user_id] = States.ORDER_DETAILS
def show_managers(user_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо список менеджерів
    cursor.execute("SELECT id, username FROM users WHERE role = 'manager'")
    managers = cursor.fetchall()
    if not managers:
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Назад",
callback_data="back_to_main"))
        bot.send_message(user_id, "Наразі немає доступних менеджерів.",
reply_markup=markup)
        conn.close()
        return
    bot.send_message(user_id, "Оберіть менеджера для спілкування:")
    for manager in managers:
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Написати повідомлення",

callback_data=f"send_message_{manager['id']}"))
        bot.send_message(user_id, f"Менеджер: {manager['username']}",
reply_markup=markup)
    # Додаємо кнопку "Назад"
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Назад до головного меню",
callback_data="back_to_main"))
    bot.send_message(user_id, "Кінець списку менеджерів", reply_markup=markup)
    conn.close()
def show_clients(user_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо ID менеджера з бази даних за telegram_id
    cursor.execute("SELECT user_id FROM telegram_users WHERE telegram_id = ?",
(user_id,))
    user_data = cursor.fetchone()
    if not user_data:
        conn.close()
        bot.send_message(user_id, "Помилка аутентифікації. Будь ласка, увійдіть знову.")
        return
    manager_id = user_data['user_id']
    # Отримуємо список клієнтів з непрочитаними повідомленнями
    cursor.execute("""
        SELECT DISTINCT u.id, u.username,
               (SELECT COUNT(*) FROM messages WHERE sender_id = u.id AND receiver_id = ?
AND is_read = 0) as unread_count
        FROM users u
        WHERE u.role = 'client'
        ORDER BY unread_count DESC, u.username
    """, (manager_id,))
    clients = cursor.fetchall()
    if not clients:
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Назад",
callback_data="back_to_main"))
        bot.send_message(user_id, "Немає клієнтів.", reply_markup=markup)
```

```
            conn.close()
            return
        bot.send_message(user_id, "Список клієнтів:")
        for client in clients:
            unread = client['unread_count']
            unread_text = f" (непрочитаних: {unread})" if unread > 0 else ""
            markup = telebot.types.InlineKeyboardMarkup()
            markup.add(telebot.types.InlineKeyboardButton("Написати повідомлення",
callback_data=f"send_message_{client['id']}"))
            markup.add(telebot.types.InlineKeyboardButton("Переглянути історію повідомлень",
callback_data=f"chat_history_{client['id']}"))
            bot.send_message(user_id, f"Клієнт: {client['username']}{unread_text}",
reply_markup=markup)
        # Додаємо кнопку "Назад"
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Назад до головного меню",
callback_data="back_to_main"))
        bot.send_message(user_id, "Кінець списку клієнтів", reply_markup=markup)
        conn.close()
    def show_chat_history(user_id, receiver_id):
        conn = sqlite3.connect(DB_PATH)
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        # Отримуємо ID користувача з бази даних за telegram_id
        cursor.execute("SELECT user_id FROM telegram_users WHERE telegram_id = ?",
(user_id,))
        user_data = cursor.fetchone()
        if not user_data:
            conn.close()
            bot.send_message(user_id, "Помилка аутентифікації. Будь ласка, увійдіть знову.")
            return
        sender_id = user_data['user_id']
        # Отримуємо інформацію про співрозмовника
        cursor.execute("SELECT username FROM users WHERE id = ?", (receiver_id,))
        receiver = cursor.fetchone()
        if not receiver:
            conn.close()
            bot.send_message(user_id, "Співрозмовник не знайдений.")
            return
        receiver_name = receiver['username']
        # Отримуємо історію повідомлень
        cursor.execute("""
            SELECT m.*, u1.username as sender_name, u2.username as receiver_name
            FROM messages m
            JOIN users u1 ON m.sender_id = u1.id
            JOIN users u2 ON m.receiver_id = u2.id
            WHERE (m.sender_id = ? AND m.receiver_id = ?) OR (m.sender_id = ? AND
m.receiver_id = ?)
            ORDER BY m.created_at DESC
            LIMIT 20
        """, (sender_id, receiver_id, receiver_id, sender_id))
        messages = cursor.fetchall()
        # Позначаємо повідомлення як прочитані
        cursor.execute("""
            UPDATE messages
            SET is_read = 1
            WHERE sender_id = ? AND receiver_id = ? AND is_read = 0
        """, (receiver_id, sender_id))
        conn.commit()
        conn.close()
        if not messages:
            bot.send_message(user_id, f"У вас ще немає повідомлень з {receiver_name}.")
        else:
```

```
        bot.send_message(user_id, f"Історія повідомлень з {receiver_name}:")
        for message in reversed(messages):  # Показуємо від найстарішого до найновішого
            created_at = datetime.fromisoformat(message['created_at'].replace('Z',
'+00:00'))
            created_at_str = created_at.strftime("%d.%m.%Y %H:%M")
            if message['sender_id'] == sender_id:
                prefix = "Ви"
            else:
                prefix = message['sender_name']
            bot.send_message(user_id, f"[{created_at_str}] {prefix}:
{message['content']}")
    # Додаємо кнопки для відповіді або повернення
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Відповісти",
                                                  callback_data=f"reply_to_{receiver_id}"))
    markup.add(telebot.types.InlineKeyboardButton("Назад", callback_data="back_to_main"))
    bot.send_message(user_id, "Оберіть дію:", reply_markup=markup)
def send_telegram_message(telegram_id, receiver_db_id, content):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо ID відправника з бази даних за telegram_id
    cursor.execute("SELECT user_id FROM telegram_users WHERE telegram_id = ?",
(telegram_id,))
    sender_data = cursor.fetchone()
    if not sender_data:
        conn.close()
        bot.send_message(telegram_id, "Помилка аутентифікації. Будь ласка, увійдіть
знову.")
        return False
    sender_db_id = sender_data['user_id']
    # Перевіряємо, чи існує отримувач
    cursor.execute("SELECT id, role FROM users WHERE id = ?", (receiver_db_id,))
    receiver = cursor.fetchone()
    if not receiver:
        conn.close()
        bot.send_message(telegram_id, "Отримувач не знайдений.")
        return False
    # Перевіряємо права відправки повідомлень
    cursor.execute("SELECT role FROM users WHERE id = ?", (sender_db_id,))
    sender_role = cursor.fetchone()['role']
    # Клієнти можуть писати тільки менеджерам, менеджери можуть писати всім
    if sender_role == 'client' and receiver['role'] != 'manager':
        conn.close()
        bot.send_message(telegram_id, "Клієнти можуть відправляти повідомлення тільки
менеджерам.")
        return False
    # Відправляємо повідомлення
    cursor.execute(
        "INSERT INTO messages (sender_id, receiver_id, content) VALUES (?, ?, ?)",
        (sender_db_id, receiver_db_id, content)
    )
    message_id = cursor.lastrowid
    # Отримуємо Telegram ID отримувача, якщо є
    cursor.execute("SELECT telegram_id FROM telegram_users WHERE user_id = ?",
(receiver_db_id,))
    tg_user = cursor.fetchone()
    conn.commit()
    conn.close()
    # Відправляємо повідомлення в Telegram, якщо отримувач зареєстрований там
    if tg_user:
        try:
            receiver_tg_id = tg_user['telegram_id']
            # Отримуємо ім'я відправника
            conn = sqlite3.connect(DB_PATH)
```

```
            conn.row_factory = sqlite3.Row
            cursor = conn.cursor()
            cursor.execute("SELECT username FROM users WHERE id = ?", (sender_db_id,))
            sender_name = cursor.fetchone()['username']
            conn.close()
            message_text = f"Нове повідомлення від {sender_name}:\n\n{content}"
            # Додаємо кнопку для відповіді
            markup = telebot.types.InlineKeyboardMarkup()
            markup.add(telebot.types.InlineKeyboardButton("Відповісти",
callback_data=f"reply_to_{sender_db_id}"))
            bot.send_message(receiver_tg_id, message_text, reply_markup=markup)
        except Exception as e:
            logger.error(f"Помилка відправки повідомлення в Telegram: {e}")
    bot.send_message(telegram_id, "Повідомлення відправлено!")
    return True
def add_to_cart(user_id, product_id, quantity=1):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо ID користувача з бази даних за telegram_id
    cursor.execute("SELECT user_id FROM telegram_users WHERE telegram_id = ?",
(user_id,))
    user_data = cursor.fetchone()
    if not user_data:
        conn.close()
        bot.send_message(user_id, "Помилка аутентифікації. Будь ласка, увійдіть знову.")
        return
    db_user_id = user_data['user_id']
    # Перевіряємо наявність товару
    cursor.execute("SELECT * FROM products WHERE id = ?", (product_id,))
    product = cursor.fetchone()
    if not product:
        conn.close()
        bot.send_message(user_id, "Товар не знайдено.")
        return
    if product['quantity'] < quantity:
        conn.close()
        bot.send_message(user_id, "Недостатньо товару на складі.")
        return
    # Перевіряємо, чи є вже товар у кошику
    cursor.execute(
        "SELECT * FROM cart_items WHERE user_id = ? AND product_id = ?",
        (db_user_id, product_id)
    )
    cart_item = cursor.fetchone()
    if cart_item:
        # Оновлюємо кількість
        new_quantity = cart_item['quantity'] + quantity
        if new_quantity > product['quantity']:
            conn.close()
            bot.send_message(user_id, "Недостатньо товару на складі.")
            return
        cursor.execute(
            "UPDATE cart_items SET quantity = ? WHERE id = ?",
            (new_quantity, cart_item['id'])
        )
    else:
        # Додаємо товар у кошик
        cursor.execute(
            "INSERT INTO cart_items (user_id, product_id, quantity) VALUES (?, ?, ?)",
            (db_user_id, product_id, quantity)
        )
    conn.commit()
    conn.close()
    markup = telebot.types.InlineKeyboardMarkup()
```

```
    markup.add(telebot.types.InlineKeyboardButton("Перейти до кошика",
callback_data="show_cart"))
    markup.add(telebot.types.InlineKeyboardButton("Продовжити покупки",
callback_data="back_to_catalog"))
    bot.send_message(user_id, f"Товар '{product['name']}' додано до кошика!",
reply_markup=markup)
def checkout(user_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо ID користувача з бази даних за telegram_id
    cursor.execute("SELECT user_id FROM telegram_users WHERE telegram_id = ?",
(user_id,))
    user_data = cursor.fetchone()
    if not user_data:
        conn.close()
        bot.send_message(user_id, "Помилка аутентифікації. Будь ласка, увійдіть знову.")
        return
    db_user_id = user_data['user_id']
    # Отримуємо товари з кошика користувача
    cursor.execute("""
        SELECT ci.product_id, ci.quantity, p.price, p.quantity as available_quantity,
p.name
        FROM cart_items ci
        JOIN products p ON ci.product_id = p.id
        WHERE ci.user_id = ?
    """, (db_user_id,))
    cart_items = cursor.fetchall()
    if not cart_items:
        conn.close()
        bot.send_message(user_id, "Ваш кошик порожній.")
        return
    # Перевіряємо наявність товарів
    for item in cart_items:
        if item['available_quantity'] < item['quantity']:
            conn.close()
            bot.send_message(user_id, f"Недостатньо товару '{item['name']}' в
наявності.")
            return
    # Створюємо нове замовлення
    cursor.execute(
        "INSERT INTO orders (user_id, status) VALUES (?, ?)",
        (db_user_id, 'new')
    )
    order_id = cursor.lastrowid
    # Додаємо товари до замовлення
    total = 0
    for item in cart_items:
        price = float(item['price'])
        quantity = int(item['quantity'])
        subtotal = price * quantity
        total += subtotal
        cursor.execute(
            "INSERT INTO order_items (order_id, product_id, quantity, price) VALUES (?,
?, ?, ?)",
            (order_id, item['product_id'], quantity, price)
        )
        # Зменшуємо кількість товару на складі
        cursor.execute(
            "UPDATE products SET quantity = quantity - ? WHERE id = ?",
            (quantity, item['product_id'])
        )
    # Очищаємо кошик користувача
    cursor.execute("DELETE FROM cart_items WHERE user_id = ?", (db_user_id,))
    conn.commit()
```

```
    conn.close()
    # Відправляємо повідомлення про успішне створення замовлення
    message_text = f"Ваше замовлення №{order_id} успішно створено!\n\n"
    message_text += "Товари в замовленні:\n"
    for item in cart_items:
        price = float(item['price'])
        quantity = int(item['quantity'])
        subtotal = price * quantity
        message_text += f"- {item['name']} x{quantity} шт. = {subtotal} грн\n"
    message_text += f"\nЗагальна сума: {total} грн"
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Перейти до замовлень",
callback_data="show_orders"))
    markup.add(telebot.types.InlineKeyboardButton("Повернутися до головного меню",
callback_data="back_to_main"))
    bot.send_message(user_id, message_text, reply_markup=markup)
    # Повідомляємо всіх менеджерів про нове замовлення
    notify_managers_about_order(order_id, db_user_id)
def notify_managers_about_order(order_id, client_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо інформацію про клієнта
    cursor.execute("SELECT username FROM users WHERE id = ?", (client_id,))
    client = cursor.fetchone()
    if not client:
        conn.close()
        return
    client_name = client['username']
    # Отримуємо список менеджерів з Telegram
    cursor.execute("""
        SELECT tu.telegram_id
        FROM telegram_users tu
        JOIN users u ON tu.user_id = u.id
        WHERE u.role = 'manager'
    """)
    managers = cursor.fetchall()
    # Отримуємо інформацію про замовлення
    cursor.execute("""
        SELECT oi.*, p.name as product_name
        FROM order_items oi
        JOIN products p ON oi.product_id = p.id
        WHERE oi.order_id = ?
    """, (order_id,))
    items = cursor.fetchall()
    total = sum(item['price'] * item['quantity'] for item in items)
    conn.close()
    if not managers:
        return
    # Формуємо повідомлення для менеджерів
    message_text = f"*Нове замовлення №{order_id}*\n"
    message_text += f"Клієнт: {client_name}\n\n"
    message_text += "*Товари:*\n"
    for item in items:
        price = float(item['price'])
        quantity = int(item['quantity'])
        subtotal = price * quantity
        message_text += f"- {item['product_name']} x{quantity} шт. = {subtotal} грн\n"
    message_text += f"\n*Загальна сума: {total} грн*"
    # Додаємо кнопку для переходу до деталей замовлення
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Змінити статус",

callback_data=f"change_order_status_{order_id}"))
    # Відправляємо повідомлення всім менеджерам
```

```
    for manager in managers:
        try:
            bot.send_message(manager['telegram_id'], message_text, parse_mode="Markdown",
reply_markup=markup)
        except Exception as e:
            logger.error(f"Помилка відправки повідомлення менеджеру: {e}")
# Функції для роботи з адмін-панеллю
def show_admin_products(user_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо список товарів
    cursor.execute("""
        SELECT p.*, c.name as category_name
        FROM products p
        LEFT JOIN categories c ON p.category_id = c.id
        ORDER BY p.name
    """)
    products = cursor.fetchall()
    if not products:
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Додати новий товар",
callback_data="add_product"))
        markup.add(telebot.types.InlineKeyboardButton("Назад",
callback_data="back_to_main"))
        bot.send_message(user_id, "Товари не знайдені.", reply_markup=markup)
        conn.close()
        return
    bot.send_message(user_id, "Список товарів:")
    # Показуємо товари
    for product in products:
        message_text = f"*{product['name']}*\n"
        message_text += f"ID: {product['id']}\n"
        message_text += f"Категорія: {product['category_name'] if
product['category_name'] else 'Без категорії'}\n"
        message_text += f"Ціна: {product['price']} грн\n"
        if product['description']:
            message_text += f"Опис: {product['description']}\n"
        message_text += f"На складі: {product['quantity']} шт."
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Редагувати",

callback_data=f"edit_product_{product['id']}"))
        markup.add(telebot.types.InlineKeyboardButton("Видалити",

callback_data=f"delete_product_{product['id']}"))
        bot.send_message(user_id, message_text, parse_mode="Markdown",
reply_markup=markup)
    # Додаємо кнопки для додавання нового товару і повернення
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Додати новий товар",
callback_data="add_product"))
    markup.add(telebot.types.InlineKeyboardButton("Назад до головного меню",
callback_data="back_to_main"))
    bot.send_message(user_id, "Кінець списку товарів", reply_markup=markup)
    conn.close()
    user_states[user_id] = States.ADMIN_PRODUCTS
def show_admin_categories(user_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо список категорій
    cursor.execute("SELECT * FROM categories ORDER BY name")
    categories = cursor.fetchall()
    if not categories:
```

```
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Додати нову категорію",
callback_data="add_category"))
        markup.add(telebot.types.InlineKeyboardButton("Назад",
callback_data="back_to_main"))
        bot.send_message(user_id, "Категорії не знайдені.", reply_markup=markup)
        conn.close()
        return
    bot.send_message(user_id, "Список категорій:")
    # Показуємо категорії
    for category in categories:
        # Рахуємо кількість товарів у категорії
        cursor.execute("SELECT COUNT(*) as count FROM products WHERE category_id = ?",
(category['id'],))
        count = cursor.fetchone()['count']
        message_text = f"*{category['name']}*\n"
        message_text += f"ID: {category['id']}\n"
        message_text += f"Кількість товарів: {count}"
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Редагувати",

callback_data=f"edit_category_{category['id']}"))
        if count == 0:
            markup.add(telebot.types.InlineKeyboardButton("Видалити",

callback_data=f"delete_category_{category['id']}"))
        bot.send_message(user_id, message_text, parse_mode="Markdown",
reply_markup=markup)
    # Додаємо кнопки для додавання нової категорії і повернення
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Додати нову категорію",
callback_data="add_category"))
    markup.add(telebot.types.InlineKeyboardButton("Назад до головного меню",
callback_data="back_to_main"))
    bot.send_message(user_id, "Кінець списку категорій", reply_markup=markup)
    conn.close()
    user_states[user_id] = States.ADMIN_CATEGORIES
# Додайте ці функції для роботи з товарами
def add_product_step1(user_id):
    bot.send_message(user_id, "Введіть назву нового товару:")
    user_states[user_id] = {'state': 'awaiting_product_name'}
def add_product_step2(user_id, name):
    bot.send_message(user_id, "Введіть опис товару:")
    user_states[user_id] = {'state': 'awaiting_product_description', 'name': name}
def add_product_step3(user_id, name, description):
    bot.send_message(user_id, "Введіть ціну товару (у форматі 123.45):")
    user_states[user_id] = {'state': 'awaiting_product_price', 'name': name,
'description': description}
def add_product_step4(user_id, name, description, price):
    bot.send_message(user_id, "Введіть кількість товару (ціле число):")
    user_states[user_id] = {'state': 'awaiting_product_quantity_add', 'name': name,
'description': description, 'price': price}
def add_product_step5(user_id, name, description, price, quantity):
    # Отримання списку категорій для вибору
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("SELECT id, name FROM categories")
    categories = cursor.fetchall()
    conn.close()
    if not categories:
        # Якщо категорій немає, створюємо товар без категорії
        create_product(user_id, name, description, price, quantity, None)
        return
    # Створюємо кнопки для вибору категорії
```

```
    markup = telebot.types.InlineKeyboardMarkup()
    for category in categories:
        markup.add(telebot.types.InlineKeyboardButton(
            category['name'],
callback_data=f"select_category_{category['id']}_for_product_{name}_{price}_{quantity}"
        ))
    markup.add(telebot.types.InlineKeyboardButton("Без категорії",
callback_data=f"select_category_none_for_product_{name}_{price}_{quantity}"))
    bot.send_message(user_id, "Виберіть категорію товару:", reply_markup=markup)
    user_states[user_id] = {'state': 'awaiting_product_category', 'name': name,
'description': description, 'price': price, 'quantity': quantity}
def create_product(user_id, name, description, price, quantity, category_id):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    # Отримуємо ID менеджера з бази даних
    cursor.execute("SELECT user_id FROM telegram_users WHERE telegram_id = ?",
(user_id,))
    user_data = cursor.fetchone()
    if not user_data:
        conn.close()
        bot.send_message(user_id, "Помилка аутентифікації. Увійдіть знову.")
        return
    # Додаємо новий товар
    try:
        cursor.execute(
            "INSERT INTO products (name, description, price, quantity, category_id)
VALUES (?, ?, ?, ?, ?)",
            (name, description, price, quantity, category_id)
        )
        conn.commit()
        bot.send_message(user_id, f"Товар '{name}' успішно створено!")
    except Exception as e:
        bot.send_message(user_id, f"Помилка створення товару: {e}")
    finally:
        conn.close()
        # Повертаємося до списку товарів
        show_admin_products(user_id)
def edit_product_step1(user_id, product_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM products WHERE id = ?", (product_id,))
    product = cursor.fetchone()
    if not product:
        conn.close()
        bot.send_message(user_id, "Товар не знайдено.")
        return
    product_dict = dict(product)
    conn.close()
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Назва",
callback_data=f"edit_product_name_{product_id}"))
    markup.add(telebot.types.InlineKeyboardButton("Опис",
callback_data=f"edit_product_description_{product_id}"))
    markup.add(telebot.types.InlineKeyboardButton("Ціна",
callback_data=f"edit_product_price_{product_id}"))
    markup.add(telebot.types.InlineKeyboardButton("Кількість",
callback_data=f"edit_product_quantity_{product_id}"))
    markup.add(telebot.types.InlineKeyboardButton("Категорія",
callback_data=f"edit_product_category_{product_id}"))
    markup.add(telebot.types.InlineKeyboardButton("Назад",
callback_data="back_to_admin_products"))
    message_text = f"Редагування товару: *{product_dict['name']}*\n"
    message_text += f"ID: {product_dict['id']}\n"
    message_text += f"Опис: {product_dict['description'] or 'Немає'}\n"
```

```
    message_text += f"Ціна: {product_dict['price']} грн\n"
    message_text += f"Кількість: {product_dict['quantity']} шт.\n"
    bot.send_message(user_id, message_text, parse_mode="Markdown", reply_markup=markup)
def delete_product_confirmation(user_id, product_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM products WHERE id = ?", (product_id,))
    product = cursor.fetchone()
    if not product:
        conn.close()
        bot.send_message(user_id, "Товар не знайдено.")
        return
    product_name = product['name']
    conn.close()
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Так, видалити",
callback_data=f"confirm_delete_product_{product_id}"))
    markup.add(telebot.types.InlineKeyboardButton("Ні, скасувати",
callback_data="back_to_admin_products"))
    bot.send_message(user_id, f"Ви впевнені, що хочете видалити товар '{product_name}'?",
reply_markup=markup)
# Додайте ці функції для роботи з категоріями
def add_category_step(user_id):
    bot.send_message(user_id, "Введіть назву нової категорії:")
    user_states[user_id] = {'state': 'awaiting_category_name'}
def create_category(user_id, name):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    try:
        cursor.execute("INSERT INTO categories (name) VALUES (?)", (name,))
        conn.commit()
        bot.send_message(user_id, f"Категорія '{name}' успішно створена!")
    except Exception as e:
        bot.send_message(user_id, f"Помилка створення категорії: {e}")
    finally:
        conn.close()
        # Повертаємося до списку категорій
        show_admin_categories(user_id)
def edit_category_step(user_id, category_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM categories WHERE id = ?", (category_id,))
    category = cursor.fetchone()
    if not category:
        conn.close()
        bot.send_message(user_id, "Категорія не знайдена.")
        return
    category_name = category['name']
    conn.close()
    bot.send_message(user_id, f"Введіть нову назву для категорії '{category_name}':")
    user_states[user_id] = {'state': 'awaiting_category_new_name', 'category_id':
category_id}
def update_category(user_id, category_id, new_name):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    try:
        cursor.execute("UPDATE categories SET name = ? WHERE id = ?", (new_name,
category_id))
        conn.commit()
        bot.send_message(user_id, f"Категорія успішно перейменована на '{new_name}'!")
    except Exception as e:
        bot.send_message(user_id, f"Помилка оновлення категорії: {e}")
    finally:
```

```
        conn.close()
        # Повертаємося до списку категорій
        show_admin_categories(user_id)
def delete_category_confirmation(user_id, category_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM categories WHERE id = ?", (category_id,))
    category = cursor.fetchone()
    if not category:
        conn.close()
        bot.send_message(user_id, "Категорія не знайдена.")
        return
    # Перевіряємо, чи є товари з цією категорією
    cursor.execute("SELECT COUNT(*) as count FROM products WHERE category_id = ?",
(category_id,))
    count = cursor.fetchone()['count']
    if count > 0:
        conn.close()
        bot.send_message(user_id, "Неможливо видалити категорію, оскільки є товари, які
до неї належать.")
        return
    category_name = category['name']
    conn.close()
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Так, видалити",
callback_data=f"confirm_delete_category_{category_id}"))
    markup.add(telebot.types.InlineKeyboardButton("Ні, скасувати",
callback_data="back_to_admin_categories"))
    bot.send_message(user_id, f"Ви впевнені, що хочете видалити категорію
'{category_name}'?", reply_markup=markup)
def show_admin_orders(user_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо список замовлень
    cursor.execute("""
        SELECT o.*, u.username as client_name
        FROM orders o
        JOIN users u ON o.user_id = u.id
        ORDER BY o.created_at DESC
    """)
    orders = cursor.fetchall()
    if not orders:
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Назад",
callback_data="back_to_main"))
        bot.send_message(user_id, "Замовлення не знайдені.", reply_markup=markup)
        conn.close()
        return
    bot.send_message(user_id, "Список замовлень:")
    # Показуємо замовлення
    for order in orders:
        # Отримуємо товари в замовленні
        cursor.execute("""
            SELECT oi.*, p.name as product_name
            FROM order_items oi
            JOIN products p ON oi.product_id = p.id
            WHERE oi.order_id = ?
        """, (order['id'],))
        items = cursor.fetchall()
        total = sum(item['price'] * item['quantity'] for item in items)
        status_map = {
            'new': 'Нове',
            'processing': 'В обробці',
```

```
            'shipped': 'Відправлено',
            'delivered': 'Доставлено',
            'cancelled': 'Скасовано'
        }
        created_at = datetime.fromisoformat(order['created_at'].replace('Z', '+00:00'))
        created_at_str = created_at.strftime("%d.%m.%Y %H:%M")
        message_text = f"*Замовлення №{order['id']}*\n"
        message_text += f"Клієнт: {order['client_name']}\n"
        message_text += f"Дата: {created_at_str}\n"
        message_text += f"Статус: {status_map.get(order['status'], order['status'])}\n"
        message_text += f"Сума: {total} грн"
        markup = telebot.types.InlineKeyboardMarkup()
        markup.add(telebot.types.InlineKeyboardButton("Деталі",

callback_data=f"admin_order_details_{order['id']}"))
        markup.add(telebot.types.InlineKeyboardButton("Змінити статус",

callback_data=f"change_order_status_{order['id']}"))
        bot.send_message(user_id, message_text, parse_mode="Markdown",
reply_markup=markup)
    # Додаємо кнопку для повернення
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Назад до головного меню",
callback_data="back_to_main"))
    bot.send_message(user_id, "Кінець списку замовлень", reply_markup=markup)
    conn.close()
    user_states[user_id] = States.ADMIN_ORDERS
def show_admin_order_details(user_id, order_id):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо замовлення
    cursor.execute("""
        SELECT o.*, u.username as client_name
        FROM orders o
        JOIN users u ON o.user_id = u.id
        WHERE o.id = ?
    """, (order_id,))
    order = cursor.fetchone()
    if not order:
        conn.close()
        bot.send_message(user_id, "Замовлення не знайдено.")
        return
    # Отримуємо товари в замовленні
    cursor.execute("""
        SELECT oi.*, p.name as product_name
        FROM order_items oi
        JOIN products p ON oi.product_id = p.id
        WHERE oi.order_id = ?
    """, (order_id,))
    items = cursor.fetchall()
    total = sum(item['price'] * item['quantity'] for item in items)
    status_map = {
        'new': 'Нове',
        'processing': 'В обробці',
        'shipped': 'Відправлено',
        'delivered': 'Доставлено',
        'cancelled': 'Скасовано'
    }
    created_at = datetime.fromisoformat(order['created_at'].replace('Z', '+00:00'))
    created_at_str = created_at.strftime("%d.%m.%Y %H:%M")
    message_text = f"*Деталі замовлення №{order['id']}*\n"
    message_text += f"Клієнт: {order['client_name']}\n"
    message_text += f"Дата: {created_at_str}\n"
    message_text += f"Статус: {status_map.get(order['status'], order['status'])}\n\n"
```

```
    message_text += "*Товари:*\n"
    for item in items:
        price = float(item['price'])
        quantity = int(item['quantity'])
        subtotal = price * quantity
        message_text += f"- {item['product_name']}\n"
        message_text += f"  Ціна: {price} грн\n"
        message_text += f"  Кількість: {quantity} шт.\n"
        message_text += f"  Сума: {subtotal} грн\n\n"
    message_text += f"*Загальна сума: {total} грн*"
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Змінити статус",

callback_data=f"change_order_status_{order_id}"))
    markup.add(telebot.types.InlineKeyboardButton("Зв'язатися з клієнтом",

callback_data=f"send_message_{order['user_id']}"))
    markup.add(telebot.types.InlineKeyboardButton("Назад до списку замовлень",
                                                  callback_data="back_to_admin_orders"))
    bot.send_message(user_id, message_text, parse_mode="Markdown", reply_markup=markup)
    conn.close()
def change_order_status(user_id, order_id):
    # Створюємо клавіатуру з варіантами статусів
    markup = telebot.types.InlineKeyboardMarkup()
    markup.add(telebot.types.InlineKeyboardButton("Нове",
callback_data=f"set_status_{order_id}_new"))
    markup.add(telebot.types.InlineKeyboardButton("В обробці",
callback_data=f"set_status_{order_id}_processing"))
    markup.add(telebot.types.InlineKeyboardButton("Відправлено",
callback_data=f"set_status_{order_id}_shipped"))
    markup.add(telebot.types.InlineKeyboardButton("Доставлено",
callback_data=f"set_status_{order_id}_delivered"))
    markup.add(telebot.types.InlineKeyboardButton("Скасовано",
callback_data=f"set_status_{order_id}_cancelled"))
    markup.add(telebot.types.InlineKeyboardButton("Назад",
callback_data=f"admin_order_details_{order_id}"))
    bot.send_message(user_id, f"Виберіть новий статус для замовлення №{order_id}:",
reply_markup=markup)
def set_order_status(user_id, order_id, status):
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    # Отримуємо ID менеджера
    cursor.execute("SELECT user_id FROM telegram_users WHERE telegram_id = ?",
(user_id,))
    user_data = cursor.fetchone()
    if not user_data:
        conn.close()
        bot.send_message(user_id, "Помилка аутентифікації. Будь ласка, увійдіть знову.")
        return
    manager_id = user_data['user_id']
    # Отримуємо поточний статус замовлення та ID клієнта
    cursor.execute("SELECT status, user_id FROM orders WHERE id = ?", (order_id,))
    order = cursor.fetchone()
    if not order:
        conn.close()
        bot.send_message(user_id, "Замовлення не знайдено.")
        return
    current_status = order['status']
    client_id = order['user_id']
    # Якщо замовлення вже скасоване, не дозволяємо змінювати статус
    if current_status == 'cancelled' and status != 'cancelled':
        conn.close()
        bot.send_message(user_id, "Неможливо змінити статус скасованого замовлення.")
        return
```

```
    # Якщо ми скасовуємо замовлення, повертаємо товари на склад
    if status == 'cancelled' and current_status != 'cancelled':
        cursor.execute("""
            SELECT product_id, quantity FROM order_items WHERE order_id = ?
        """, (order_id,))
        items = cursor.fetchall()
        for item in items:
            cursor.execute(
                "UPDATE products SET quantity = quantity + ? WHERE id = ?",
                (item['quantity'], item['product_id'])
            )
    # Оновлюємо статус замовлення
    cursor.execute("UPDATE orders SET status = ? WHERE id = ?", (status, order_id))
    # Отримуємо Telegram ID клієнта, якщо є
    cursor.execute("SELECT telegram_id FROM telegram_users WHERE user_id = ?",
(client_id,))
    tg_user = cursor.fetchone()
    conn.commit()
    conn.close()
    # Відправляємо повідомлення в Telegram клієнту, якщо він зареєстрований там
    if tg_user:
        try:
            client_tg_id = tg_user['telegram_id']
            status_map = {
                'new': 'Нове',
                'processing': 'В обробці',
                'shipped': 'Відправлено',
                'delivered': 'Доставлено',
                'cancelled': 'Скасовано'
            }
            message_text = f"Статус вашого замовлення №{order_id} змінено на:
{status_map.get(status, status)}"
            bot.send_message(client_tg_id, message_text)
        except Exception as e:
            logger.error(f"Помилка відправки повідомлення в Telegram: {e}")
    bot.send_message(user_id, f"Статус замовлення №{order_id} змінено на: {status}")
    # Повертаємося до деталей замовлення
    show_admin_order_details(user_id, order_id)
# Обробник Callback-кнопок
@bot.callback_query_handler(func=lambda call: True)
def callback_handler(call):
    user_id = call.message.chat.id
    data = call.data
    # Основні дії
    if data == "back_to_main":
        # Отримуємо роль користувача
        conn = sqlite3.connect(DB_PATH)
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("""
            SELECT u.role
            FROM users u
            JOIN telegram_users tu ON u.id = tu.user_id
            WHERE tu.telegram_id = ?
        """, (user_id,))
        user_data = cursor.fetchone()
        conn.close()
        is_admin = user_data and user_data['role'] == 'manager'
        user_states[user_id] = States.MAIN_MENU
        bot.edit_message_text("Головне меню", call.message.chat.id,
call.message.message_id)
        bot.send_message(user_id, "Оберіть дію:",
reply_markup=get_main_menu_keyboard(is_admin))
    # CRUD операції для товарів
    elif data == "add_product":
```

```
        add_product_step1(user_id)
    elif data.startswith("edit_product_"):
        if "_name_" in data:
            product_id = data.split("_")[-1]
            bot.send_message(user_id, "Введіть нову назву товару:")
            user_states[user_id] = {'state': 'awaiting_product_edit_name', 'product_id':
int(product_id)}
        elif "_description_" in data:
            product_id = data.split("_")[-1]
            bot.send_message(user_id, "Введіть новий опис товару:")
            user_states[user_id] = {'state': 'awaiting_product_edit_description',
'product_id': int(product_id)}
        elif "_price_" in data:
            product_id = data.split("_")[-1]
            bot.send_message(user_id, "Введіть нову ціну товару (у форматі 123.45):")
            user_states[user_id] = {'state': 'awaiting_product_edit_price', 'product_id':
int(product_id)}
        elif "_quantity_" in data:
            product_id = data.split("_")[-1]
            bot.send_message(user_id, "Введіть нову кількість товару:")
            user_states[user_id] = {'state': 'awaiting_product_edit_quantity',
'product_id': int(product_id)}
        elif "_category_" in data:
            product_id = data.split("_")[-1]
            # Показуємо список категорій для вибору
            conn = sqlite3.connect(DB_PATH)
            conn.row_factory = sqlite3.Row
            cursor = conn.cursor()
            cursor.execute("SELECT id, name FROM categories")
            categories = cursor.fetchall()
            conn.close()
            markup = telebot.types.InlineKeyboardMarkup()
            for category in categories:
                markup.add(telebot.types.InlineKeyboardButton(
                    category['name'],
callback_data=f"set_product_category_{product_id}_{category['id']}"
                ))
            markup.add(telebot.types.InlineKeyboardButton("Без категорії",
callback_data=f"set_product_category_{product_id}_null"))
            markup.add(telebot.types.InlineKeyboardButton("Скасувати",
callback_data=f"edit_product_{product_id}"))
            bot.send_message(user_id, "Виберіть нову категорію для товару:",
reply_markup=markup)
        else:
            product_id = data.split("_")[-1]
            edit_product_step1(user_id, int(product_id))
    elif data.startswith("set_product_category_"):
        parts = data.split("_")
        product_id = int(parts[3])
        category_id = parts[4]
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()
        if category_id == "null":
            cursor.execute("UPDATE products SET category_id = NULL WHERE id = ?",
(product_id,))
        else:
            cursor.execute("UPDATE products SET category_id = ? WHERE id = ?",
(category_id, product_id))
        conn.commit()
        conn.close()
        bot.send_message(user_id, "Категорію товару змінено!")
        edit_product_step1(user_id, product_id)
    elif data.startswith("delete_product_"):
        product_id = data.split("_")[-1]
        delete_product_confirmation(user_id, int(product_id))
```

```
    elif data.startswith("confirm_delete_product_"):
        product_id = data.split("_")[-1]
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()
        cursor.execute("DELETE FROM products WHERE id = ?", (product_id,))
        conn.commit()
        conn.close()
        bot.send_message(user_id, "Товар успішно видалено!")
        show_admin_products(user_id)
    # CRUD операції для категорій
    elif data == "add_category":
        add_category_step(user_id)
    elif data.startswith("edit_category_"):
        category_id = data.split("_")[-1]
        edit_category_step(user_id, int(category_id))
    elif data.startswith("delete_category_"):
        category_id = data.split("_")[-1]
        delete_category_confirmation(user_id, int(category_id))
    elif data.startswith("confirm_delete_category_"):
        category_id = data.split("_")[-1]
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()
        cursor.execute("DELETE FROM categories WHERE id = ?", (category_id,))
        conn.commit()
        conn.close()
        bot.send_message(user_id, "Категорія успішно видалена!")
        show_admin_categories(user_id)
    elif data.startswith("select_category_") and "_for_product_" in data:
        parts = data.split("_")
        category_id = parts[2]
        name = parts[5]
        try:
            price = float(parts[6])
            quantity = int(parts[7])
            description = user_states[user_id].get('description', '')
            if category_id == "none":
                category_id = None
            create_product(user_id, name, description, price, quantity, category_id)
        except Exception as e:
            bot.send_message(user_id, f"Помилка створення товару: {e}")
    # Каталог товарів
    elif data == "catalog_all":
        show_products(user_id)
    elif data == "catalog_search":
        bot.edit_message_text("Введіть пошуковий запит:", call.message.chat.id,
call.message.message_id)
        user_states[user_id] = States.CATALOG_SEARCH
    elif data == "back_to_catalog":
        show_catalog(user_id)
    elif data.startswith("category_"):
        category_id = data.split("_")[1]
        show_products(user_id, category_id=category_id)
    # Кошик
    elif data == "show_cart":
        show_cart(user_id)
    elif data.startswith("add_to_cart_"):
        product_id = data.split("_")[3]
        # Запитуємо кількість
        bot.edit_message_text("Введіть кількість товару:", call.message.chat.id,
call.message.message_id)
        user_states[user_id] = {'state': States.AWAITING_PRODUCT_QUANTITY, 'product_id':
int(product_id)}
    elif data == "checkout":
        checkout(user_id)
    elif data == "clear_cart":
```

```
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()
        # Отримуємо ID користувача з бази даних за telegram_id
        cursor.execute("SELECT user_id FROM telegram_users WHERE telegram_id = ?",
(user_id,))
        user_data = cursor.fetchone()
        if user_data:
            db_user_id = user_data[0]
            cursor.execute("DELETE FROM cart_items WHERE user_id = ?", (db_user_id,))
            conn.commit()
            bot.edit_message_text("Кошик очищено.", call.message.chat.id,
call.message.message_id)
            bot.send_message(user_id, "Ваш кошик порожній.")
        conn.close()
    # Замовлення
    elif data == "show_orders":
        conn = sqlite3.connect(DB_PATH)
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("SELECT user_id FROM telegram_users WHERE telegram_id = ?",
(user_id,))
        user_data = cursor.fetchone()
        if user_data:
            db_user_id = user_data['user_id']
            show_orders(user_id, db_user_id)
        conn.close()
    elif data == "back_to_orders":
        conn = sqlite3.connect(DB_PATH)
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("SELECT user_id FROM telegram_users WHERE telegram_id = ?",
(user_id,))
        user_data = cursor.fetchone()
        if user_data:
            db_user_id = user_data['user_id']
            show_orders(user_id, db_user_id)
        conn.close()
    elif data.startswith("order_details_"):
        order_id = data.split("_")[2]
        show_order_details(user_id, order_id)
    # Повідомлення
    elif data.startswith("send_message_"):
        receiver_id = int(data.split("_")[2])
        # Отримуємо ім'я отримувача
        conn = sqlite3.connect(DB_PATH)
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("SELECT username FROM users WHERE id = ?", (receiver_id,))
        receiver = cursor.fetchone()
        conn.close()
        if receiver:
            bot.edit_message_text(f"Введіть повідомлення для {receiver['username']}:",
                                  call.message.chat.id, call.message.message_id)
            user_states[user_id] = {'state': States.AWAITING_MESSAGE, 'receiver_id':
receiver_id}
    elif data.startswith("reply_to_"):
        receiver_id = int(data.split("_")[2])
        # Отримуємо ім'я отримувача
        conn = sqlite3.connect(DB_PATH)
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("SELECT username FROM users WHERE id = ?", (receiver_id,))
        receiver = cursor.fetchone()
        conn.close()
        if receiver:
```

```
            bot.edit_message_text(f"Введіть відповідь для {receiver['username']}:",
                                  call.message.chat.id, call.message.message_id)
            user_states[user_id] = {'state': States.AWAITING_MESSAGE, 'receiver_id':
receiver_id}
    elif data.startswith("chat_history_"):
        receiver_id = int(data.split("_")[2])
        show_chat_history(user_id, receiver_id)
    # Адмін-панель для замовлень
    elif data == "back_to_admin_orders":
        show_admin_orders(user_id)
    elif data.startswith("admin_order_details_"):
        order_id = data.split("_")[3]
        show_admin_order_details(user_id, int(order_id))
    elif data.startswith("change_order_status_"):
        order_id = data.split("_")[3]
        change_order_status(user_id, int(order_id))
    elif data.startswith("set_status_"):
        parts = data.split("_")
        order_id = int(parts[2])
        status = parts[3]
        set_order_status(user_id, order_id, status)
    elif data == "back_to_admin_products":
        show_admin_products(user_id)
    elif data == "back_to_admin_categories":
        show_admin_categories(user_id)
    elif data.startswith("admin_order_details_"):
        order_id = data.split("_")[3]
        show_admin_order_details(user_id, int(order_id))
    # Обробляємо відповідь на callback, щоб уникнути помилки "Query is too old"
    bot.answer_callback_query(call.id)
# Запуск бота в окремому потоці
def run_telegram_bot():
    try:
        logger.info("Запуск Telegram бота...")
        bot.polling(none_stop=True)
    except Exception as e:
        logger.error(f"Помилка в роботі Telegram бота: {e}")
# Запуск сервера Flask
if __name__ == '__main__':
    # Запускаємо Telegram бота в окремому потоці
    telegram_thread = Thread(target=run_telegram_bot)
    telegram_thread.daemon = True
    telegram_thread.start()
    # Запускаємо Flask сервер
    app.run(debug=True, host='0.0.0.0', port=5000)
```

seed.py

```
import sqlite3
import hashlib
import random
from datetime import datetime, timedelta
# Шлях до бази даних
DB_PATH = 'pharmacy.db'
# Функція для хешування паролів
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
# Функція для отримання випадкової дати в межах останніх 30 днів
def random_date(days=30):
    end_date = datetime.now()
    start_date = end_date - timedelta(days=days)
    time_between_dates = end_date - start_date
```

```
    days_between_dates = time_between_dates.days
    random_number_of_days = random.randrange(days_between_dates)
    return start_date + timedelta(days=random_number_of_days)
# Підключення до бази даних
conn = sqlite3.connect(DB_PATH)
cursor = conn.cursor()
# Очищення всіх таблиць
tables = [
    'cart_items', 'order_items', 'orders', 'messages', 'products',
    'categories', 'telegram_users', 'users'
]
for table in tables:
    cursor.execute(f"DELETE FROM {table}")
# Додавання користувачів (менеджери та клієнти)
users = [
    # Менеджери
    {'username': 'admin', 'password': hash_password('admin'), 'role': 'manager'},
    {'username': 'manager1', 'password': hash_password('manager1'), 'role': 'manager'},
    # Клієнти
    {'username': 'client1', 'password': hash_password('client1'), 'role': 'client'},
    {'username': 'client2', 'password': hash_password('client2'), 'role': 'client'},
    {'username': 'client3', 'password': hash_password('client3'), 'role': 'client'},
    {'username': 'client4', 'password': hash_password('client4'), 'role': 'client'},
    {'username': 'client5', 'password': hash_password('client5'), 'role': 'client'}
]
user_ids = {}
for user in users:
    cursor.execute(
        "INSERT INTO users (username, password, role) VALUES (?, ?, ?)",
        (user['username'], user['password'], user['role'])
    )
    user_ids[user['username']] = cursor.lastrowid
# Додавання категорій
categories = [
    "Антибіотики",
    "Знеболювальні",
    "Вітаміни та добавки",
    "Засоби від застуди",
    "Догляд за шкірою"
]
category_ids = {}
for category in categories:
    cursor.execute("INSERT INTO categories (name) VALUES (?)", (category,))
    category_ids[category] = cursor.lastrowid
# Додавання товарів
products = [
    # Антибіотики
    {'name': 'Амоксицилін', 'description': 'Антибіотик широкого спектру дії', 'price':
120.50, 'quantity': 50, 'category': 'Антибіотики'},
    {'name': 'Азитроміцин', 'description': 'Антибіотик групи макролідів', 'price':
185.75, 'quantity': 30, 'category': 'Антибіотики'},
    {'name': 'Цефтріаксон', 'description': 'Цефалоспориновий антибіотик', 'price':
210.00, 'quantity': 25, 'category': 'Антибіотики'},
    {'name': 'Доксициклін', 'description': 'Антибіотик тетрациклінового ряду', 'price':
95.50, 'quantity': 45, 'category': 'Антибіотики'},
    # Знеболювальні
    {'name': 'Ібупрофен', 'description': 'Нестероїдний протизапальний засіб', 'price':
75.00, 'quantity': 100, 'category': 'Знеболювальні'},
    {'name': 'Парацетамол', 'description': 'Жарознижуючий та знеболювальний засіб',
'price': 45.50, 'quantity': 150, 'category': 'Знеболювальні'},
    {'name': 'Кетанов', 'description': 'Сильний знеболювальний препарат', 'price':
110.25, 'quantity': 40, 'category': 'Знеболювальні'},
    {'name': 'Німесил', 'description': 'Протизапальний та знеболювальний засіб', 'price':
130.00, 'quantity': 35, 'category': 'Знеболювальні'},
    # Вітаміни та добавки
```

```
    {'name': 'Вітамін C', 'description': 'Підтримка імунної системи', 'price': 65.50,
'quantity': 80, 'category': 'Вітаміни та добавки'},
    {'name': 'Комплекс B-вітамінів', 'description': 'Підтримка нервової системи',
'price': 120.75, 'quantity': 50, 'category': 'Вітаміни та добавки'},
    {'name': 'Мультивітамінний комплекс', 'description': 'Повний набір необхідних
вітамінів', 'price': 210.00, 'quantity': 30, 'category': 'Вітаміни та добавки'},
    {'name': 'Магній B6', 'description': 'Для підтримки серцево-судинної системи',
'price': 140.00, 'quantity': 40, 'category': 'Вітаміни та добавки'},
    # Засоби від застуди
    {'name': 'Грипго', 'description': 'Комплексний засіб від симптомів грипу та застуди',
'price': 95.50, 'quantity': 60, 'category': 'Засоби від застуди'},
    {'name': 'Назальний спрей', 'description': 'Усуває закладеність носа', 'price':
85.00, 'quantity': 70, 'category': 'Засоби від застуди'},
    {'name': 'Сироп від кашлю', 'description': 'Зменшує кашель та полегшує
відхаркування', 'price': 110.25, 'quantity': 45, 'category': 'Засоби від застуди'},
    {'name': 'Фервекс', 'description': 'Розчинний порошок від застуди', 'price': 145.00,
'quantity': 30, 'category': 'Засоби від застуди'},
    # Догляд за шкірою
    {'name': 'Крем для рук', 'description': 'Зволожувальний крем для сухої шкіри рук',
'price': 55.50, 'quantity': 100, 'category': 'Догляд за шкірою'},
    {'name': 'Сонцезахисний крем', 'description': 'Захист від УФ-променів', 'price':
175.00, 'quantity': 40, 'category': 'Догляд за шкірою'},
    {'name': 'Гель для вмивання', 'description': 'Для щоденного догляду за обличчям',
'price': 125.75, 'quantity': 50, 'category': 'Догляд за шкірою'},
    {'name': 'Антибактеріальні серветки', 'description': 'Для дезінфекції рук', 'price':
45.00, 'quantity': 120, 'category': 'Догляд за шкірою'},
]
product_ids = {}
for product in products:
    cursor.execute(
        "INSERT INTO products (name, description, price, quantity, category_id) VALUES
(?, ?, ?, ?, ?)",
        (product['name'], product['description'], product['price'], product['quantity'],
category_ids[product['category']])
    )
    product_ids[product['name']] = cursor.lastrowid
# Створення замовлень для клієнтів
statuses = ['new', 'processing', 'shipped', 'delivered', 'cancelled']
order_details = []
for client_username in ['client1', 'client2', 'client3', 'client4', 'client5']:
    # Кожен клієнт має від 1 до 3 замовлень
    for _ in range(random.randint(1, 3)):
        # Створення замовлення
        status = random.choice(statuses)
        order_date = random_date()
        cursor.execute(
            "INSERT INTO orders (user_id, status, created_at) VALUES (?, ?, ?)",
            (user_ids[client_username], status, order_date)
        )
        order_id = cursor.lastrowid
        # Додавання товарів до замовлення
        total_amount = 0
        order_products = []
        # Вибираємо від 1 до 5 випадкових товарів для замовлення
        selected_products = random.sample(products, random.randint(1, 5))
        for product in selected_products:
            quantity = random.randint(1, 3)
            price = product['price']
            total_amount += price * quantity
            cursor.execute(
                "INSERT INTO order_items (order_id, product_id, quantity, price) VALUES
(?, ?, ?, ?)",
                (order_id, product_ids[product['name']], quantity, price)
            )
            order_products.append({
```

```
                'name': product['name'],
                'quantity': quantity,
                'price': price
            })
        order_details.append({
            'order_id': order_id,
            'client': client_username,
            'status': status,
            'date': order_date,
            'products': order_products,
            'total': total_amount
        })
# Додавання повідомлень між користувачами
messages = [
    # Між client1 і admin
    {'sender': 'client1', 'receiver': 'admin', 'content': 'Добрий день! Хотів би
дізнатися, чи є у вас в наявності Амоксицилін?', 'days_ago': 5},
    {'sender': 'admin', 'receiver': 'client1', 'content': 'Добрий день! Так, Амоксицилін
є в наявності. Ціна - 120.50 грн.', 'days_ago': 5},
    {'sender': 'client1', 'receiver': 'admin', 'content': 'Дякую за інформацію! Скоро
оформлю замовлення.', 'days_ago': 5},
    # Між client2 і manager1
    {'sender': 'client2', 'receiver': 'manager1', 'content': 'Вітаю! Коли очікувати
доставку мого замовлення №2?', 'days_ago': 3},
    {'sender': 'manager1', 'receiver': 'client2', 'content': 'Доброго дня! Ваше
замовлення вже передано кур\'єру. Очікуйте доставку завтра.', 'days_ago': 3},
    {'sender': 'client2', 'receiver': 'manager1', 'content': 'Чудово, дякую!',
'days_ago': 3},
    # Між client3 і admin
    {'sender': 'client3', 'receiver': 'admin', 'content': 'Добрий день! Чи можу я
повернути товар, якщо він мені не підійшов?', 'days_ago': 7},
    {'sender': 'admin', 'receiver': 'client3', 'content': 'Добрий день! Так, ви можете
повернути товар протягом 14 днів з моменту покупки, якщо він не був використаний і
збережена упаковка.', 'days_ago': 7},
    {'sender': 'client3', 'receiver': 'admin', 'content': 'Дякую за відповідь!',
'days_ago': 7},
    # Між client4 і manager1
    {'sender': 'client4', 'receiver': 'manager1', 'content': 'Вітаю! У мене проблема з
оплатою замовлення. Що робити?', 'days_ago': 2},
    {'sender': 'manager1', 'receiver': 'client4', 'content': 'Доброго дня! Будь ласка,
надішліть скріншот помилки або детальніше опишіть проблему.', 'days_ago': 2},
    {'sender': 'client4', 'receiver': 'manager1', 'content': 'При оплаті карткою пише
"Транзакція відхилена".', 'days_ago': 2},
    {'sender': 'manager1', 'receiver': 'client4', 'content': 'Перевірте, чи достатньо
коштів на картці. Також спробуйте використати іншу картку або спосіб оплати.',
'days_ago': 1},
    {'sender': 'client4', 'receiver': 'manager1', 'content': 'Спробував іншу картку, все
спрацювало. Дякую!', 'days_ago': 1},
    # Між client5 і admin
    {'sender': 'client5', 'receiver': 'admin', 'content': 'Добрий день! Чи є у вас знижки
для постійних клієнтів?', 'days_ago': 4},
    {'sender': 'admin', 'receiver': 'client5', 'content': 'Добрий день! Так, у нас є
програма лояльності. За кожні 1000 грн покупок ви отримуєте 5% знижки на наступне
замовлення.', 'days_ago': 4},
    {'sender': 'client5', 'receiver': 'admin', 'content': 'Чудово! А як можна дізнатися
свій поточний бонусний баланс?', 'days_ago': 3},
    {'sender': 'admin', 'receiver': 'client5', 'content': 'Ви можете побачити свій
бонусний баланс в особистому кабінеті на нашому сайті або запитати у менеджера.',
'days_ago': 3},
    {'sender': 'client5', 'receiver': 'admin', 'content': 'Дякую за інформацію!',
'days_ago': 3}
]
# Додавання повідомлень в базу даних
for message in messages:
    message_date = datetime.now() - timedelta(days=message['days_ago'])
```

```
    cursor.execute(
        "INSERT INTO messages (sender_id, receiver_id, content, created_at, is_read)
VALUES (?, ?, ?, ?, ?)",
        (
            user_ids[message['sender']],
            user_ids[message['receiver']],
            message['content'],
            message_date,
            1  # Всі повідомлення прочитані
        )
    )
# Додавання непрочитаних повідомлень
unread_messages = [
    {'sender': 'client1', 'receiver': 'admin', 'content': 'Хотів би замовити ще Вітамін
C. Він є в наявності?', 'days_ago': 0},
    {'sender': 'client3', 'receiver': 'manager1', 'content': 'Коли очікується поставка
нових товарів від застуди?', 'days_ago': 0},
    {'sender': 'admin', 'receiver': 'client2', 'content': 'Вітаємо! Нагадуємо про нові
акції в нашому магазині!', 'days_ago': 0}
]
for message in unread_messages:
    message_date = datetime.now() - timedelta(days=message['days_ago'],
hours=random.randint(1, 5))
    cursor.execute(
        "INSERT INTO messages (sender_id, receiver_id, content, created_at, is_read)
VALUES (?, ?, ?, ?, ?)",
        (
            user_ids[message['sender']],
            user_ids[message['receiver']],
            message['content'],
            message_date,
            0  # Непрочитані повідомлення
        )
    )
# Зберігання змін
conn.commit()
conn.close()
print("База даних успішно заповнена:")
print(f"- Додано 2 менеджери та 5 клієнтів")
print(f"- Додано 5 категорій та 20 товарів")
print(f"- Створено {len(order_details)} замовлень")
print(f"-   Додано   {len(messages)   +   len(unread_messages)}   повідомлень   (з   них
{len(unread_messages)} непрочитаних)")
```