ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ

Навчально-науковий інститут денної
освіти Форма навчання денна

Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри

___Олена ОЛХОВСЬКА

«_____»_202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

«РОЗРОБКА ТА ВПРОВАДЖЕННЯ ІНТЕРНЕТ-МАГАЗИНУ НА ПЛАТФОРМІ TELEGRAM»

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Мордик Ярослав Олегович

_____«____»_____202_ р.

Науковий керівник к. ф.-м. н. Ольховська Олена Володимирівна

_____«____»_____202_ р.

Рецензент _____

ПОЛТАВА 2025

Завідувач кафедри Олена ОЛЬХОВСЬКА

«____»_____ 202_ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК

ВИКОНАННЯ КВАЛІФАКАЦІЙНОЇ РОБОТИ

на тему «Розробка та впровадження інтернет-магазину на платформі Telegram»
зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Мордик Ярослав Олегович
Затверджена наказом ректора № 177-Н від «10» вересня 2024 р.
Термін подання студентом роботи «___» _____ 20__ р.
Вихідні дані до кваліфікаційної роботи: публікації та лекційний матеріал з теми, навчальні відеоматеріали, технічна документація.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
1.1 Постановка задачі проект телеграм магазин
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Розвиток електронної комерції
2.2. Використання месенджерів для бізнесу
2.3. Інструменти для розробки Telegram-магазинів
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Вимоги для створення телеграм бота
3.2. Вимоги до структури та інтеграції
3.3. Вимоги до бази даних
3.4. Обмеження
4. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РОЗРОБКИ
4.1. Вибір мови програмування
4.2. Додаткові бібліотеки та інструменти
4.3. Обгрунтування вибору Telegram як платформи для магазину
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТКИ
А. Лістинг коду.

Перелік графічного матеріалу: Необхідні ілюстрації.

Консультанти розділів кваліфікаційної роботи

Розділ	ПІП, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Ольховська О.В.		
Інформаційний огляд	Ольховська О.В		
Теоретична частина	Ольховська О.В.		
Практична частина	Ольховська О.В.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання «____»_____ 202_ р.

Здобувач вищої освіти Мордик Ярослав Олегович

Науковий керівник к. ф.-м. н. Олена ОЛХОВСЬКА

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____

Протокол засідання ЕК № ____ від «____» _____ 202_ р.

Секретар ЕК_____ _____

Затверджую	**Погоджено**
Зав. кафедрою	Науковий керівник
_____	_____
к.ф.-м.н. Олена ОЛЬХОВСЬКА	к. ф.-м. н. Олена ОЛЬХОВСЬКА
«____» _____ 202_ р.	«____» _____ 202_ р.

План

кваліфікаційної роботи на тему
«Розробка та впровадження інтернет-магазину на платформі Telegram»
зі спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
ступеня бакалавра
Мордик Ярослав Олегович
Прізвище, ім'я, по батькові

Здобувач вищої освіти _____ Мордик Ярослав
«____» _____ 202_ р.

РЕФЕРАТ

Записка: 107с., 11 рис., 20 джерел

ПРОЕКТУВАННЯ ТА ВПРОВАДЖЕННЯ ІНТЕРНЕТ-МАГАЗИНУ НА ПЛАТФОРМІ TELEGRAM

Об’єкт розробки – процеси автоматизації інтернет-торгівлі на платформі Telegram із використанням програмних рішень на Python.

Мета роботи – спроектувати та реалізувати інтернет-магазин на платформі Telegram мовою програмування Python, розробити архітектуру рішення, реалізувати основні функціональні можливості (каталог товарів, кошик, оформлення замовлення, підтримка користувачів, адміністрування), протестувати працездатність системи та проаналізувати отримані результати.

Методи дослідження – методи проектування, розробки та впровадження інтернет-магазинів на основі Telegram-ботів.

Практична цінність роботи полягає у створенні готового Telegram-бота інтернет-магазину в Telegram, який може бути адаптований під різноманітні бізнес-потреби.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології займають важливе місце у житті кожної людини. Особливо активно розвивається сфера електронної комерції(e-commerce), що дозволяє здійснювати покупки товарів або послуг онлайн, не виходячи з дому. В останні роки популярності набули інтернет-магазини у месенджерах, зокрема у Telegram, який надає зручний інтерфейс для спілкування та взаємодії з ботами.

Створення магазину на платформі Telegram дозволяє автоматизувати багато основних процесів, розширити користувачів та забезпечити їм зручний спосіб взаємодії із магазином. Такий підхід відкриває нові можливості для підприємців, зменшує витрати на підтримку повноцінних сайтів або фізичних магазинів і пропонує сучасний підхід до онлайн-бізнесу.

Мета цієї роботи – розробити повноцінний інтернет-магазин на платформі Telegram мовою програмування Python, реалізувати основні функції(каталог товарів, кошик, оформлення замовлення, підтримка користувачів, адмін-панель), протестувати працездатність функцій та виправити помилки.

Об’єкт дослідження – процеси автоматизації інтернет-магазину на платформі Telegram із використанням мови Python.

Предмет дослідження – методи проектування, розробки та впровадження інтернет-магазинів на основі Telegram-ботів.

Завдання дипломної роботи:

- провести аналіз предметної області та вимог до інтернет-магазинів у Telegram;
- обгрунтувати вибір інструментальних засобів і програмних технологій для розробки;
- спроектувати архітектуру програмного рішення, структуру даних і логіку бізнес-процесів;
- реалізувати основні функції інтернет-магазину (каталог, кошик, замовлення, адмін-панель);

- здійснити тестування реалізованих функцій бота та оцінити результати розробки, виправити всі помилки;

1. ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка задачі проект телеграм магазин

Розробка проекту телеграм магазину передбачає наступні задачі:

- Збір вимог до проекту – визначення основних функцій, які будуть реалізовані в магазині
- Привітальне повідомлення користувачу при введенні команди /start
- Зручне меню для покупця та адміністратора магазину
- Кнопка «Каталог» - дозволяє переглянути доступні товари в магазині з описом, назвою, фото та ціною
- «Кошик» - перегляд доданих товарів в кошик, можливість подальшого оформлення замовлення
- «Мої замовлення» - перегляд замовлень, які були здійснені в магазині
- «Підтримка» - саппорт для покупців, вирішення проблемних ситуацій
- «Адмін-панель» - кнопка для адміністраторів магазину, яка дозволяє керувати ним
- «Управління товарами» - додати товар, видалити, редагувати
- «Замовлення» - нові замовлення від покупців, підтвердження/відхилення
- «Архів замовлень» - перегляд всіх покупок в магазині
- «Налаштування» - додавати/видаляти адмінів, кнопка доступна для власника магазину, для інших можливість переглянути всіх адміністраторів
- Сповіщення для адміністраторів при надходженні нового замовлення
- Реалізація бази даних для збереження всієї інформації для коректної роботи магазину

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Розвиток електронної комерції

На сучасному ринку онлайн продажів все частіше зустрічаються інтернет-магазини, які працюють не лише як повноцінні вебсайти, а й у вигляді ботів у месенджерах. Telegram є одним із найпопулярніших месенджерів завдяки своїй відкритості, підтримці ботів та великій аудиторії. Класичні інтернет-магазини зазвичай реалізуються у вигляді веб-сайтів, які мають складну архітектуру, вимагають розробки окремого front-end й back-end, захищеного хостингу та постійної підтримки. Водночас споживачі дедалі частіше віддають перевагу мобільним платформам та месенджерам, що стимулює розвиток альтернативних каналів взаємодії між бізнесом і клієнтами. Telegram-боти дозволяють автоматизувати обробку замовлень, інформувати клієнтів про наявність товару, а також забезпечують зручний інтерфейс для спілкування із покупцем. Відомі платформи, як BotStore, ManyBot, ShopBot, надають готові конструктори для створення ботів-магазинів, але часто мають обмежений функціонал і не дозволяють гнучко змінювати логіку роботи.

Власна розробка Telegram-бота для магазину дозволяє реалізувати саме той функціонал, який потрібен бізнесу: від налаштування асортименту та прийому замовлень до інтеграції з платіжними системами й аналітикою, без обмежень типових конструкторів.

2.2. Використання месенджерів для бізнесу

Месенджери, такі як Telegram, Viber, WhatsApp, Instagram, Threads, стали не тільки засобами особистого спілкування, а й платформами для створення бізнес-інструментів: ботів, онлайн-сервісів, систем підтримки клієнтів, каналів продажу товарів та послуг, залучення нових клієнтів.

Використання месенджерів у бізнесі має ряд переваг. По-перше, це прямий канал зв’язку зі споживачем у звичному для нього середовищі — чаті на смартфоні. По-друге, месенджери забезпечують високу швидкість і персоналізацію взаємодії: бот може миттєво відповісти на запит клієнта, надати інформацію про товар чи допомогти з оформленням замовлення. По-третє, розвиток офіційних API (Application Programming Interface) у популярних месенджерах, зокрема Telegram, дозволяє розробникам втілювати різноманітні бізнес-логіки без необхідності створювати власні додатки з нуля. Telegram особливо виділяється розвиненим Bot API, що дає змогу легко створювати власних ботів, автоматизувати діалоги, інтегрувати зовнішні системи, приймати платежі і фактично реалізовувати повноцінні інтернет-магазини всередині месенджера. Завдяки цьому месенджери стали привабливою платформою для електронної комерції, маркетингу та клієнтської підтримки.

Класичний web-магазин — це зазвичай сайт із каталогом товарів, кошиком, особистим кабінетом користувача та системою оплати. Telegram-магазин — це бот, з яким користувач взаємодіє через чат. Основні переваги Telegram-магазину:

- Швидкий старт: створити Telegram-бота можна за кілька годин без необхідності в складній інфраструктурі.
- Мобільність: користувачі Telegram можуть замовляти товари прямо з телефону, не встановлюючи додаткових застосунків.
- Зручність спілкування: бот спілкується з користувачем у звичному для нього чат-інтерфейсі.
- Можливості автоматизації: бот може працювати 24/7, автоматично обробляти замовлення, відповідати на запитання, повідомляти про статуси.

Простота адміністрування: адміністраторам легко додавати/видаляти товари, переглядати замовлення, відповідати на питання клієнтів прямо у Telegram.

Серед можливих недоліків Telegram-магазину можна відзначити певні обмеження у дизайні інтерфейсу (вигляд діалогів задається месенджером) та

залежність від політик і правил платформи Telegram. Проте переваги у швидкості запуску та зручності часто переважують ці недоліки, особливо для малого і середнього бізнесу.

Типовий Telegram-бот для інтернет-магазину повинен реалізовувати такий набір функцій:

- Перегляд каталогу товарів: показ списку товарів із назвою, описом, фотографією та ціною.
- Додавання товару до кошика: можливість обрати товар і помістити його до кошика покупця для подальшого оформлення замовлення.
- Оформлення замовлення: збір необхідних контактних даних користувача, вибір способу доставки та оплати, підтвердження замовлення.
- Адміністрування товарів і замовлень: інтерфейс для адміністратора, який дозволяє додавати/редагувати/видаляти товари, переглядати список замовлень, змінювати статуси замовлень.
- Підтримка користувачів: функція для зв’язку користувача з представником підтримки (наприклад, кнопка "Підтримка", яка надсилає повідомлення адміністратору.
- Інтеграція з платіжними системами (необов’язково): можливість приймати онлайн-оплату через платіжні системи, підтримувані Telegram (LiqPay, WayForPay тощо).
- Ведення історії замовлень та архів: зберігання всіх замовлень користувача, доступ користувача до історії покупок, а адміністратора — до архіву виконаних замовлень.

2.3. Інструменти для розробки Telegram-магазинів

«Для реалізації Telegram-бота магазину використовуються різні інструментальні засоби. Мова програмування найчастіше обирається Python, яка є однією з найпопулярніших мов для створення Telegram-ботів. Це зумовлено

простим і зрозумілим синтаксисом Python, наявністю широкої спільноти розробників та великою кількістю готових бібліотек і фреймворків. Зокрема, існують спеціалізовані Python-бібліотеки для роботи з Telegram Bot API (наприклад, python-telegram-bot, Telebot (pyTelegramBotAPI), aiogram тощо), що значно спрощують процес розробки бота. Використання цих бібліотек дозволяє зосередитися на бізнес-логіці, не витрачаючи часу на низькорівневу реалізацію HTTP-запитів до Telegram.

Для зберігання даних типовими є реляційні бази даних. Залежно від масштабу проєкту, можна використовувати вбудовану СКБД SQLite або серверні СКБД на кшталт PostgreSQL, MySQL. У процесі розробки часто застосовують ORM (Object-Relational Mapping) бібліотеки (наприклад, SQLAlchemy) для спрощення роботи з базою даних на рівні об’єктів мови Python. Це дозволяє маніпулювати записами у БД як об’єктами, уникати прямого написання SQL-запитів та підвищує швидкість розробки.

На практиці багато малих і середніх бізнесів вже використовують Telegram-боти для продажу одягу, квитків, їжі, електроніки, надання освітніх та інших послуг. Деякі компанії запускають ботів як додатковий канал збуту паралельно з веб-сайтом, а інколи бот стає основним інструментом продажів. Існують навіть платформи, що пропонують готові рішення для запуску комерційних ботів. Наприклад, платіжна система LiqPay надає інструменти для інтеграції прийому платежів через Telegram, що спрощує монетизацію ботів-магазинів.

Більшість кастомних (індивідуально розроблених) Telegram-магазинів мають відносно просту архітектуру. Типовий цикл роботи виглядає так: бот отримує повідомлення або команду від користувача, обробляє її (визначає, яку дію треба виконати), звертається до бази даних при необхідності (наприклад, щоб отримати список товарів чи зберегти замовлення) та надсилає користувачеві відповідне повідомлення або меню з кнопками. Така послідовність дій виконується для кожної взаємодії користувача з ботом. Подібна структура дозволяє обслуговувати багато користувачів одночасно та достатньо легко масштабувати рішення, додаючи нові функції.

Інтернет-магазини на платформі Telegram – це сучасний, ефективний та доступний спосіб автоматизації електронної комерції. Використання месенджера як платформи для магазину забезпечує швидкий старт бізнесу та зручний досвід для кінцевого користувача. Проведений огляд показав, що для реалізації такого рішення доцільно обрати мову Python та спеціалізований фреймворк (наприклад, aiogram). Такий вибір дозволяє легко масштабувати проєкт, інтегрувати нові функції та забезпечити високу якість сервісу для користувачів. Отже, Telegram-бот є перспективним інструментом для підприємців, що прагнуть розширити канали збуту і автоматизувати продажі з мінімальними витратами» [https://github.blog/developer-skills/programming-languages-and-frameworks/why-python-keeps-growing-explained/]

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Вимоги для створення телеграм бота

Для розробки повноцінного інтернет-магазину на базі Telegram-бота необхідно визначити перелік основних функцій, які він повинен виконувати. Виходячи з аналізу предметної області та типового функціоналу (див. розділ 1.3), до функціональних вимог бот-системи відносяться такі можливості:

- Реєстрація та ідентифікація користувачів: автоматичне розпізнавання користувача на основі його Telegram ID та імені користувача; збір додаткових контактних даних при оформленні першого замовлення за потреби.
- Перегляд каталогу товарів: надання користувачу списку доступних товарів із їх назвою, описом, фотографією та ціною; можливість переглянути детальну інформацію про кожен товар.
- Пошук та фільтрація товарів: забезпечення простого пошуку товарів за назвою або категорією; опціонально – фільтрація за ціною, наявністю тощо, для зручності навігації по каталогу.
- Додавання товарів до кошика: можливість вибрати товар із каталогу та додати його в кошик; формування переліку вибраних товарів, який користувач може переглянути перед підтвердженням замовлення.
- Оформлення замовлення: збір необхідної інформації для замовлення – вибір способу доставки, введення адреси та контактного номера телефону; підтвердження замовлення користувачем і створення запису про замовлення в системі.
- Перегляд історії замовлень: надання користувачу доступу до списку його попередніх замовлень та їх статусів (наприклад, "Очікує оплату", "Виконано", "Скасовано").

- Відображення статусу замовлення: інформування користувача про актуальний статус його замовлення (через повідомлення від бота або через команду "Мої замовлення").
- Підтримка зворотного зв’язку: забезпечення можливості швидкого зв’язку користувача з менеджером або адміністратором (через кнопку "Підтримка" чи команду, яка надсилає повідомлення адміністраторам про запит від користувача).
- Адміністрування магазину: спеціальні функції для користувачів з роллю адміністратора, що дозволяють:
    - Керування товарами: додавати нові товари, редагувати їхні описи, ціни, наявність; видаляти товари або приховувати їх (наприклад, змінюючи статус на "недоступний").
    - Керування замовленнями: переглядати всі замовлення, змінювати статус замовлення (підтверджувати, відправляти, скасовувати тощо), переглядати деталі кожного замовлення.
    - Обробка звернень користувачів: відповідати на запитання та повідомлення, що надходять через систему підтримки бота.
    - Управління адміністраторами: додавати або видаляти облікові записи інших адміністраторів (наприклад, через окремий список довірених Telegram ID).

Окрім функціоналу, система повинна відповідати ряду нефункціональних вимог, які забезпечують зручність, безпеку та ефективність її використання:

- Зручність інтерфейсу: бот повинен мати простий та інтуїтивно зрозумілий інтерфейс у чаті. Використовуються зрозумілі кнопки та меню, мінімізується введення тексту користувачем для швидкої навігації. Вся необхідна інформація подається стисло і вчасно.
- Мінімізація людського фактору: більшість рутинних операцій (формування повідомлень, перевірка даних, нагадування) виконується автоматично ботом, знижуючи ризик помилок з боку оператора чи користувача.

- Надійність зберігання даних: всі дані магазину (список товарів, інформація про користувачів, історія замовлень тощо) зберігаються в базі даних із резервним копіюванням (за потреби) та захистом від втрати або пошкодження.
- Безпека: доступ до адміністративних функцій має бути обмежений лише перевіреними користувачами (списком адміністративних Telegram ID). Потрібна валідація всіх даних, що вводяться користувачами (наприклад, номер телефону у правильному форматі), щоб уникнути некоректних або шкідливих даних. Передача конфіденційної інформації (токенів, паролів до БД) здійснюється через безпечні механізми (наприклад, зберігання в .env файлі).
- Масштабованість: архітектура рішення має передбачати можливість додавання нових функцій або розширення асортименту без суттєвих змін у вже написаному коді. Наприклад, повинна легко додаватися нова категорія товарів або новий модуль (такий як система лояльності) без переписування всієї системи.
- Сумісність: бот має коректно працювати у різних офіційних клієнтах Telegram – як на мобільних пристроях (Android, iOS), так і у десктопних додатках чи веб-версії. Інтерфейс (кнопки, повідомлення) повинен відображатися і реагувати однаково передбачувано на всіх платформах.
- Продуктивність: система повинна забезпечувати швидку обробку запитів. Час відправки користувачем команди до отримання відповіді від бота не повинен перевищувати 2–3 секунд навіть при значному навантаженні. Це потребує оптимізації коду і запитів до бази даних.
- Простота розгортання: розроблений бот має бути легко розгорнутий як локально, так і на сервері. Бажано, щоб для запуску достатньо було встановити необхідне програмне забезпечення (Python, залежні бібліотеки) та запустити скрипт, без складної конфігурації. В ідеалі – підтримка запуску в Docker-контейнері для швидкого деплою.

3.2. Вимоги до структури та інтеграції

При проектуванні системи враховуються ключові вимоги до архітектури бота та можливостей інтеграції з зовнішніми сервісами:

- **Модульна структура проєкту:** код бота повинен бути поділений на окремі модулі, кожен з яких відповідає за свою функціональність. Наприклад, окремі модулі для роботи з базою даних, для логіки користувацьких команд, для адміністративних команд, для опрацювання замовлень, тощо. Це підвищує читаність коду і спрощує подальше супроводження та розширення системи.
- **Логування та моніторинг:** передбачити механізми ведення журналу подій (логування) – щоб відслідковувати дії користувачів, замовлення, а також виявляти та усувати помилки в роботі бота. Бажано передбачити логування ключових подій (наприклад, створення нового замовлення, помилки під час звернення до БД) та можливість моніторингу стану бота.
- **Збереження історії:** система повинна зберігати історичні дані, зокрема історію усіх замовлень, для можливості подальшого аналізу продажів, формування звітів тощо. Бот має надавати інструменти для перегляду архівних даних (наприклад, архів замовлень для адміністратора).
- **Гнучкість налаштування:** рішення має бути конфігурованим – тобто, певні параметри роботи бота (такі як список категорій товарів, тексти повідомлень, валюта цін) повинні змінюватися через налаштування, а не шляхом втручання в код. Зокрема, асортимент товарів коректніше змінювати шляхом оновлення записів у базі даних або файлах конфігурації, без потреби перепрограмування бота.

3.3. Вимоги до бази даних

База даних Telegram-магазину має бути спроєктована таким чином, щоб забезпечити зберігання всієї необхідної інформації. Основні вимоги до БД включають наявність таких даних:

- Товари: таблиця для зберігання інформації про товари (назва, опис, ціна, фото або шлях до зображення, категорія, статус наявності чи доступності).
- Користувачі: таблиця користувачів, що взаємодіяли з ботом (Telegram ID, username, ім’я, контактні дані за потреби, роль/статус – наприклад, клієнт чи адміністратор).
- Замовлення: таблиця замовлень, де кожен запис відображає зроблене користувачем замовлення (структура замовлення – які товари і в якій кількості, сума, дата, статус виконання, пов’язані ідентифікатори користувача та товарів).
- Деталі замовлень: окрема таблиця або структура для позицій замовлення (товари в межах конкретного замовлення: назва товару, ціна на момент замовлення, кількість тощо), пов’язана з таблицею замовлень.
- Звернення до підтримки: якщо реалізовано систему підтримки, потрібне зберігання історії звернень (наприклад, повідомлення користувача та відповіді адміністратора, час звернення, статус – вирішене/нове).
- Адміністратори: список ідентифікаторів або облікових записів адмінів, яким дозволено доступ до адмін-функцій (може бути окрема таблиця або поле в таблиці користувачів).
- Налаштування: таблиця або файл налаштувань, де зберігаються конфігураційні параметри бота (токени, налаштування оплати, дозволені ID адміністраторів, тощо).

3.4. Обмеження

При розробці рішення необхідно брати до уваги такі потенційні обмеження системи:

- Бот не обробляє самостійно платіжні транзакції – усі платежі реалізуються через сторонні платіжні системи, інтегровані в Telegram (наприклад, через інструмент Telegram Payments API з підключенням провайдера LiqPay, Stripe чи ін.). Таким чином, відповідальність за безпечність платежів покладається на перевірені платіжні шлюзи, а бот лише ініціює оплату і отримує підтвердження.
- Основний режим роботи – це індивідуальна взаємодія з користувачем у приватному чаті. Робота бота в групах або каналах не передбачається як головна функція магазину (можливе лише оповіщення або консультації, але оформлення замовлень – тільки в особистому чаті).
- Кількість товарів у кошику або кількість одночасних замовлень може бути обмежена відповідно до бізнес-правил. Наприклад, за одну сесію користувач може оформити не більше певної кількості замовлень, або в кошик можна додати обмежену кількість позицій, щоб уникнути перевантаження інтерфейсу. Такі обмеження мають бути визначені і задокументовані.
- Будь-які зміни у структурі даних (наприклад, додавання нових полів до товару чи нового типу сутності) виконуються через оновлення коду та міграцію бази даних розробником/адміністратором. Кінцеві користувачі та навіть адміністратори магазину не повинні змінювати структуру БД безпосередньо через бот.

Формалізовані вище функціональні та нефункціональні вимоги є основою для проєктування архітектури Telegram-магазину. Чітке визначення вимог до системи гарантує, що під час реалізації будуть враховані всі необхідні можливості та умови роботи бота. Дотримання цих вимог забезпечить створення сучасного, безпечного, зручного та масштабованого рішення для електронної

комерції у месенджері. Визначені обмеження допомагають звузити область проєктування до реалістичних меж і врахувати особливості платформи Telegram при реалізації магазину.

4. ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1. Вибір мови програмування

Для розробки Telegram-бота, що реалізує функції інтернет-магазину, обрано мову програмування Python. Основні причини такого вибору наступні:

- Простота і швидкість розробки: Python має простий, зрозумілий синтаксис та низький поріг входження. Це дозволяє швидко створювати прототипи й реалізовувати ідеї, що важливо в умовах обмеженого часу на розробку дипломного проєкту. Код на Python легко читається і супроводжується, що знижує ймовірність допущення помилок.
- Широка спільнота і екосистема: мільйони розробників використовують Python, існує велика кількість форумів, документації та готових рішень. У разі виникнення проблем чи питань завжди можна знайти приклади реалізації схожих задач. Активна спільнота забезпечує регулярне оновлення бібліотек та підтримку актуальності інструментів.
- Наявність спеціалізованих бібліотек: для Python доступні численні бібліотеки, які суттєво спрощують розробку Telegram-ботів. Замість того, щоб працювати з HTTP-запитами до Telegram Bot API вручну, розробник може скористатися готовими фреймворками (як-от python-telegram-bot, Telebot, aiogram тощо), які беруть на себе більшу частину низькорівневої роботи. Це прискорює розробку та підвищує надійність.
- Кросплатформеність і простота деплою: Python-код може бути запущений на різних платформах (Windows, Linux, macOS) без значних змін. Багато хостинг-провайдерів підтримують розгортання Python-додатків, у тому числі у вигляді Docker-контейнерів. Таким чином, бот на Python можна легко розгорнути як на власному сервері, так і на хмарних платформах.

Telegram надає відкритий Bot API, який дозволяє стороннім розробникам створювати ботів і інтегрувати їх з месенджером. Bot API працює через надсилання HTTP-запитів до серверів Telegram: бот може отримувати оновлення (повідомлення від користувачів) методом *long polling* або через *webhook*, та відповідати, надсилаючи повідомлення назад користувачам. Безпосередньо працювати з HTTP-запитами можливо, проте це досить рутинно і незручно. Для спрощення взаємодії з Telegram Bot API існують спеціальні бібліотеки-фреймворки під різні мови програмування, зокрема й Python.

Було проаналізовано декілька популярних Python-бібліотек для роботи з Telegram Bot API, а саме:

- PyTelegramBotAPI (Telebot): проста у використанні бібліотека з базовим синхронним підходом. Вона підходить для невеликих проєктів та ботів із мінімальним навантаженням. Перевагою є легкий для розуміння синтаксис, проте для масштабованих рішень Telebot може бути менш зручним, оскільки не підтримує асинхронність і складніше управляти великою кількістю обробників.
- python-telegram-bot: потужна бібліотека з багатим функціоналом. Вона підтримує роботу як у синхронному, так і в асинхронному режимах, має розширену систему обробки повідомлень, але є дещо складнішою у налаштуванні. Цей фреймворк добре підходить для серйозних проєктів, проте для початку роботи потребує більше часу на освоєння.
- aiogram: сучасна асинхронна бібліотека, побудована на базі asyncio. Відзначається високою продуктивністю та гнучкістю. Підтримує концепцію скінченних автоматів (FSM) для керування станами діалогу, має модульну структуру проєкту, що полегшує підтримку коду. Aiogram активно розвивається і має спільноту, яка підтримує оновлення (актуальна версія 3.x).

4.2. Додаткові бібліотеки та інструменти

Для зберігання інформації обрано SQLite — вбудовану реляційну базу даних, яка ідеально підходить для невеликих та середніх проєктів.

Переваги SQLite:

- Не потребує окремого сервера — база зберігається у вигляді одного файлу.
- Простота підключення та використання з Python (стандартна бібліотека sqlite3).
- Достатня продуктивність для середніх навантажень.
- Легко переноситься між платформами.

За потреби можна легко мігрувати на PostgreSQL або іншу СУБД із мінімальними змінами у коді.

- SQLAlchemy *(опційно)* — ORM для складніших моделей БД.
- Pillow — для роботи з фотографіями, якщо потрібна обробка зображень.
- openpyxl, pandas — для експорту даних у Excel (наприклад, архів замовлень).
- logging — для ведення журналу подій та помилок.
- dotenv — для зберігання токенів і конфіденційної інформації у .env-файлі.
- pytest — модульне тестування основних функцій.
- ngrok — для створення тунелів під час локальної розробки (webhook-режим).
- Docker — для розгортання бота у контейнері (опційно).
- Git — для керування версіями коду.

4.3. Обгрунтування вибору Telegram як платформи для магазину

Варто окремо зазначити, чому саме Telegram обрано платформою для реалізації інтернет-магазину. Telegram сьогодні є сучасною, надійною і масовою платформою з мільйонами активних користувачів. На користь Telegram свідчать такі фактори:

- Безкоштовний хостинг ботів: Telegram не вимагає оплати за використання своїх серверів для пересилання повідомлень ботам. Це означає, що розробник може зосередитися лише на утриманні свого серверу з логікою бота, а вся інфраструктура доставки повідомлень на сторону клієнта — на боці Telegram і надається безкоштовно.
- Проста реєстрація бота: для створення нового бота не потрібно проходити складні процедури – достатньо скористатися офіційним ботом @BotFather, який за кілька кроків видасть унікальний токен. Все, що потрібно для старту, – це цей токен і обліковий запис у Telegram.
- Підтримка платежів: Telegram на рівні Bot API підтримує інтеграцію з платіжними системами (Telegram Payments). Це дозволяє реалізувати прийом оплати прямо у чаті, використовуючи перевірені провайдери, що значно розширює можливості комерційних ботів.
- Розширені можливості ботів: платформа Telegram постійно розвивається, пропонуючи нові функції для ботів – такі як інтерактивні клавіатури, кнопки опитувань, меню команд, автозаповнення для введення даних (наприклад, запит локації або контакту). Все це можна використовувати для поліпшення користувацького досвіду у нашому магазині.
- Безпека та авторизація: Telegram підтримує OAuth-авторизацію через бота та різні рівні доступу, що можна використати для реалізації

адміністративної панелі з більш захищеним входом. Крім того, весь трафік між клієнтом і серверами Telegram шифрується, що забезпечує конфіденційність переданих даних.

- Аудиторія: велика частка цільової аудиторії багатьох бізнесів вже використовує Telegram. Це знижує поріг входження клієнтів до користування ботом-магазином – їм не треба встановлювати новий додаток, достатньо відкрити чат у знайомому месенджері.

На основі аналізу інструментів, для реалізації поставленого завдання обрано стек технологій, що оптимально відповідає вимогам. Поєднання **Python** із фреймворком **aiogram** та базою даних **SQLite** дозволяє швидко створювати гнучкі, масштабовані рішення, які легко підтримувати й розширювати у майбутньому. Додаткові бібліотеки (для роботи з зображеннями, таблицями, логами тощо) та інструменти (для тестування і деплою) покликані забезпечити якість розробки та надійну роботу системи. Таким чином, вибір засобів розробки обґрунтовано їх відповідністю потребам проєкту і спрямований на досягнення максимальної ефективності при розробці Telegram-магазину.

4.4 Загальна архітектура Telegram-магазину

Телеграм-магазин будується за принципом клієнт-серверної архітектури, де користувачі взаємодіють із телеграм ботом, а бот із базою даних, або зовнішніми сервісами.

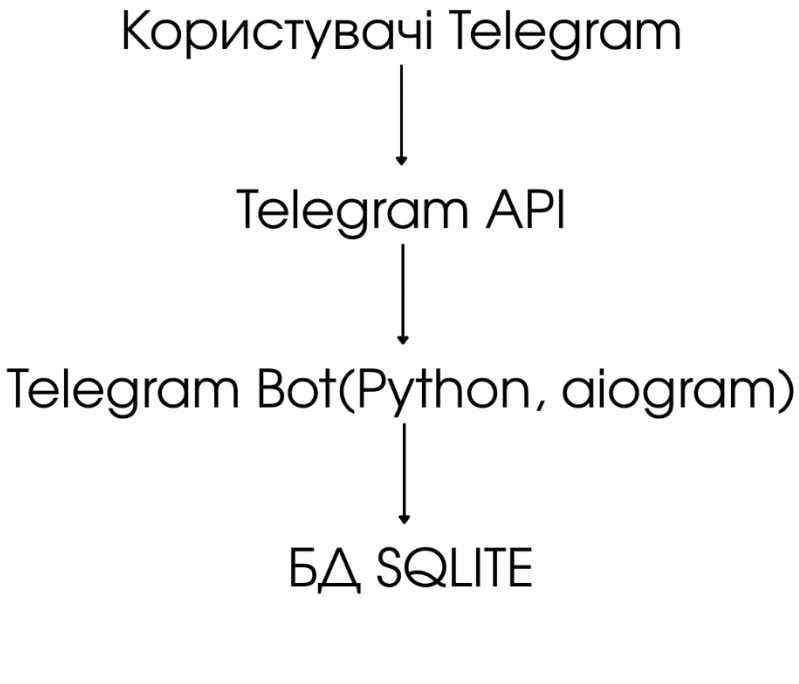

Рисунок 5.1 – Архітектура Telegram-бота

Основні компоненти Telegram-магазину на Python з aiogram:

- main.py — точка входу, ініціалізація бота, підключення модулів.
- handlers/ — обробники повідомлень і callback-кнопок (user.py, admin.py, support.py тощо).
- keyboards/ — модулі з описом кнопок та клавіатур.
- database/ — робота з БД (queries.py, models.py, fixes.py).
- states.py — визначення станів FSM для складних діалогів.

- config.py / settings.py — зберігання налаштувань, токенів.
- utils/ — додаткові функції (логування, форматування повідомлень).

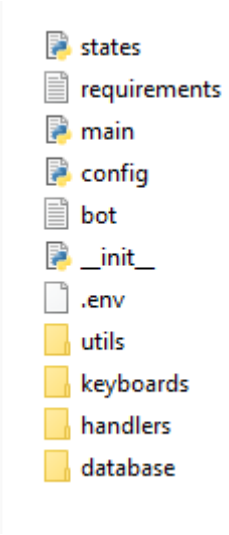

Рисунок 5.2 – Структура проєкту

- users — інформація про користувачів (id, Telegram ID, username, ім’я, роль)
- products — товари (id, назва, опис, ціна, фото, категорія, is_deleted)
- orders — замовлення (id, user_id, дата, статус, контактні дані)
- order_items — товари в замовленні (id, order_id, product_id, name, price, quantity)
- admins — адміністратори (user_id)
- settings — різні налаштування бота, наприклад, додавання нових адміністраторів

Оформлення замовлення покупцем:

- Користувач натискає “Каталог”
- Обирає товар – Додати до кошика
- Вводить контакти (ПІБ, телефон, адресу)
- Бот формує замовлення, додає до таблиці orders

- Адміністратор отримує сповіщення про нове замовлення

1. main.py приймає запит – передає у handlers/
2. handlers/user.py — функції для користувача
3. handlers/admin.py — функції для адмінів
4. handlers/support.py — запити до підтримки
5. database/queries.py — всі запити до бази даних
6. keyboards/ — всі кнопки
7. states.py — FSM для діалогів із кількома кроками

Продумана архітектура, модульна структура та чітко визначені сценарії забезпечують масштабованість, простоту підтримки й розширення Telegram-магазину. Всі компоненти системи логічно взаємодіють між собою, а структура БД забезпечує збереження історії покупок, гнучкість та надійність рішення.

4.5. Основні етапи реалізації

1. Створення Telegram-бота та отримання токена
2. Налаштування середовища розробки
3. Розробка основної структури проєкту
4. Розробка бази даних
5. Створення handler-ів для користувачів і адміністраторів
6. Оформлення інтерфейсу через inline-клавіатури
7. Реалізація логіки замовлень, корзини, підтримки
8. Адмін-панель та архів замовлень
9. Тестування та деплой

Крок 1. Створення бота через @BotFather

1. Відкрити додаток телеграм.
2. У пошуку знайти оф.бота @BotFather
3. Написати /start
4. Написати команду /newbot для створення нового бота
5. Вказати назву(наприклад, shopbot)
6. Вказати унікальне ім’я(наприклад, tgbotmagazin_bot
7. Отримати API токен
8. Перейти в @getmyid_bot та отримати id головного адміна

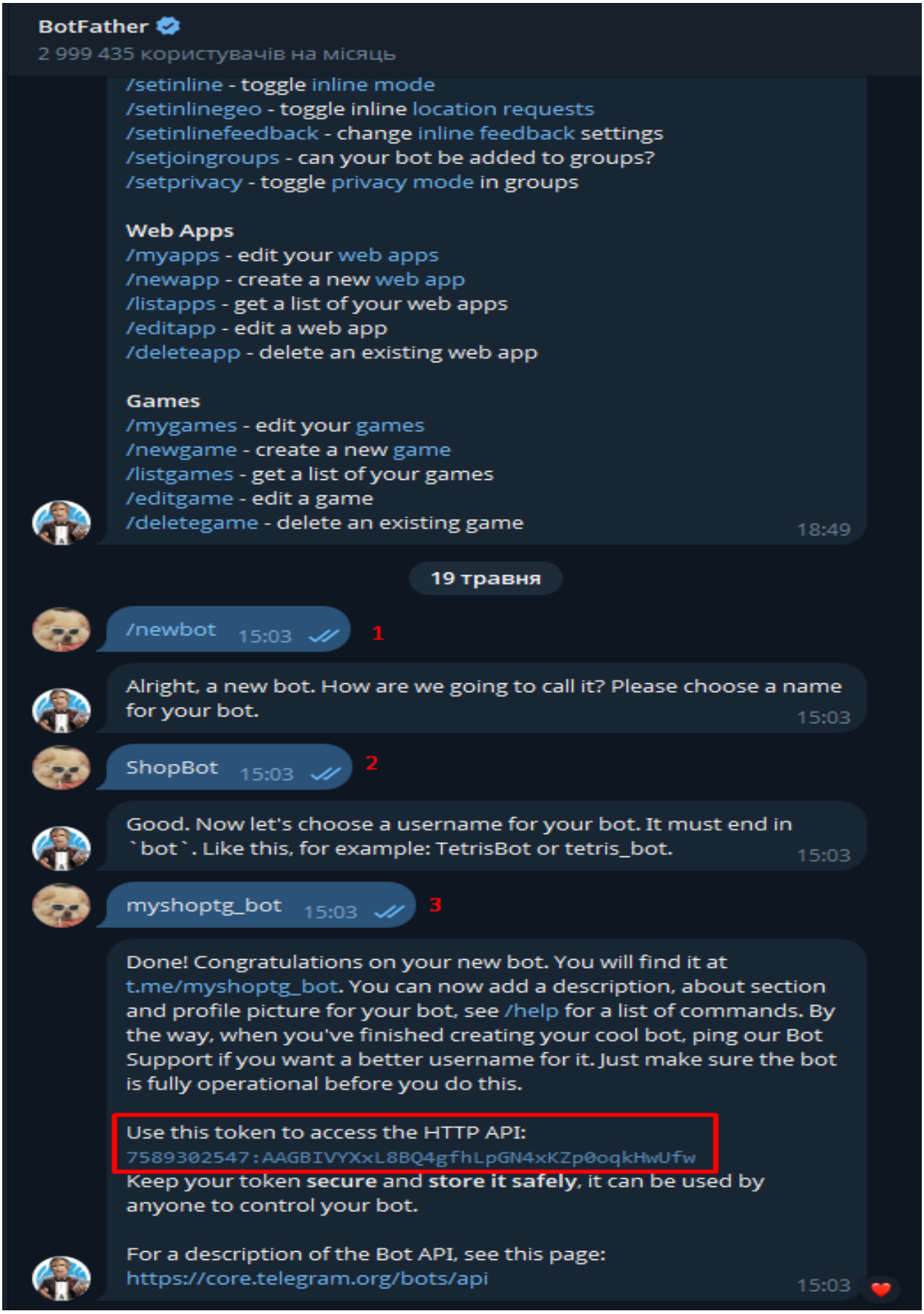

Рисунок 6.1 – Отримання API токена

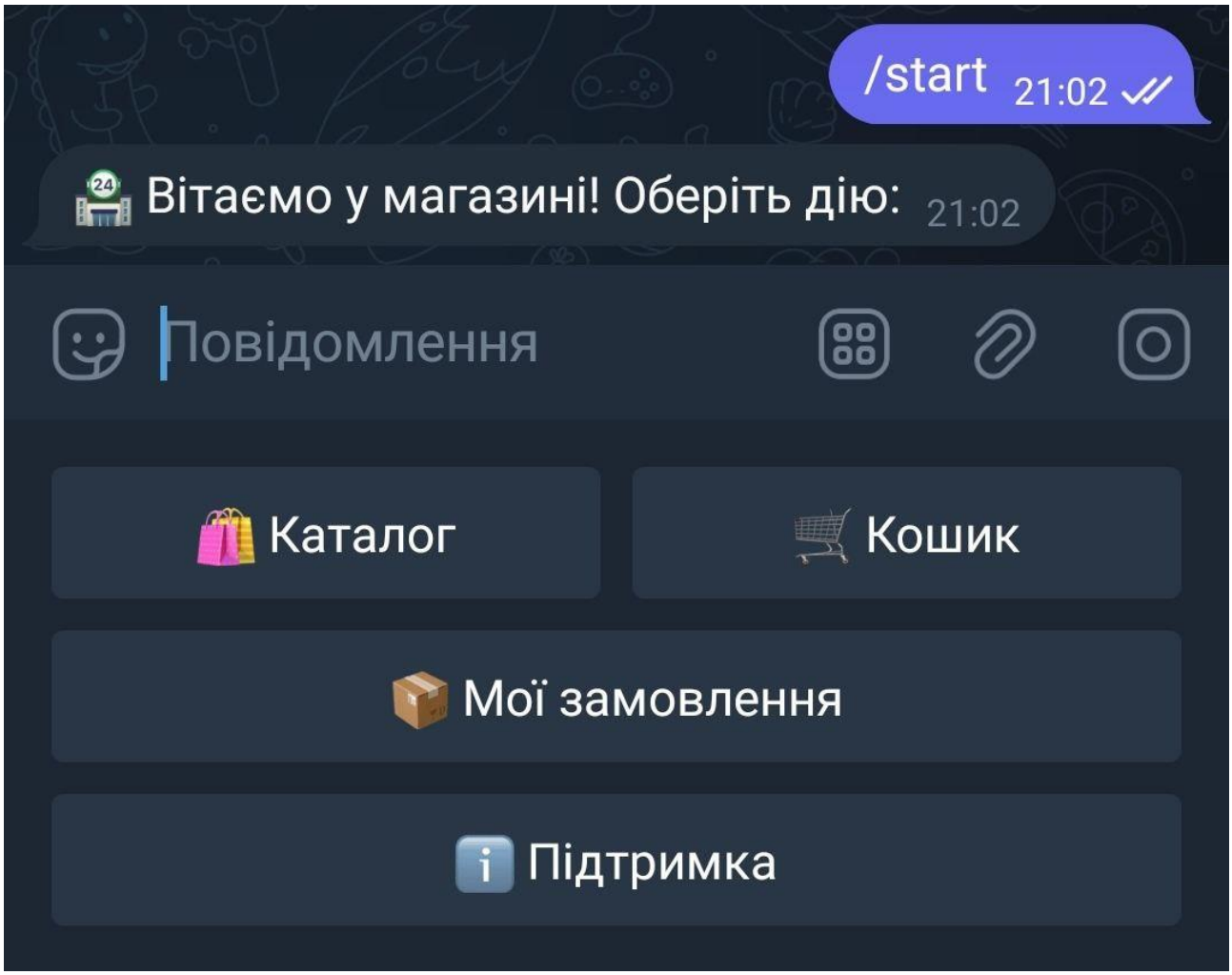

Рисунок 6.2 – Головне меню для звичайного користувача

2. При натисненні на кнопку «Каталог» користувачу відображаються всі доступні товари для замовлення в цьому магазині та можливість додати їх до кошику з подальшим оформлення замовлення

Рисунок 6.3 – кнопка «Каталог»

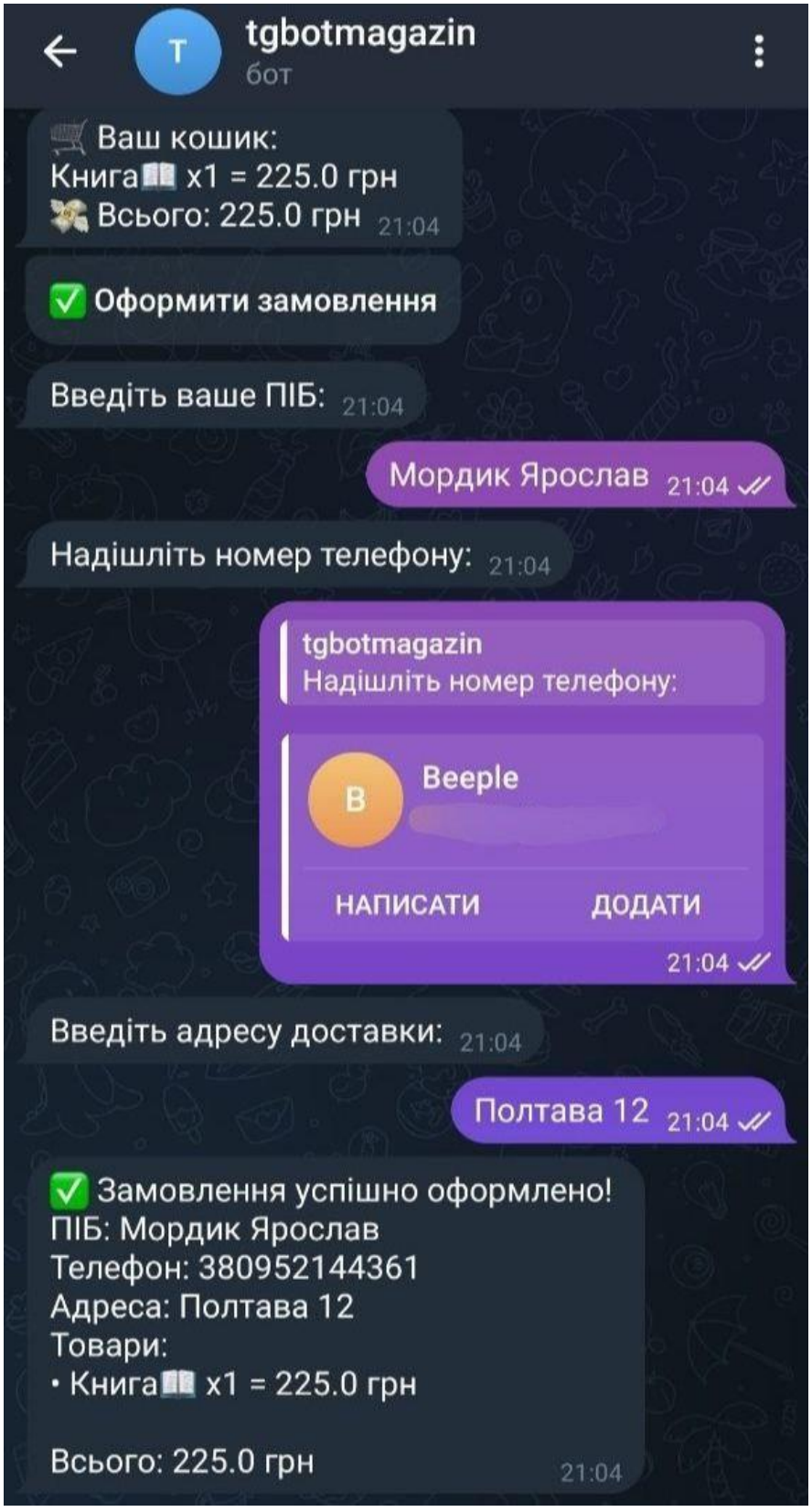

Рисунок 6.4 – Оформлення замовлення

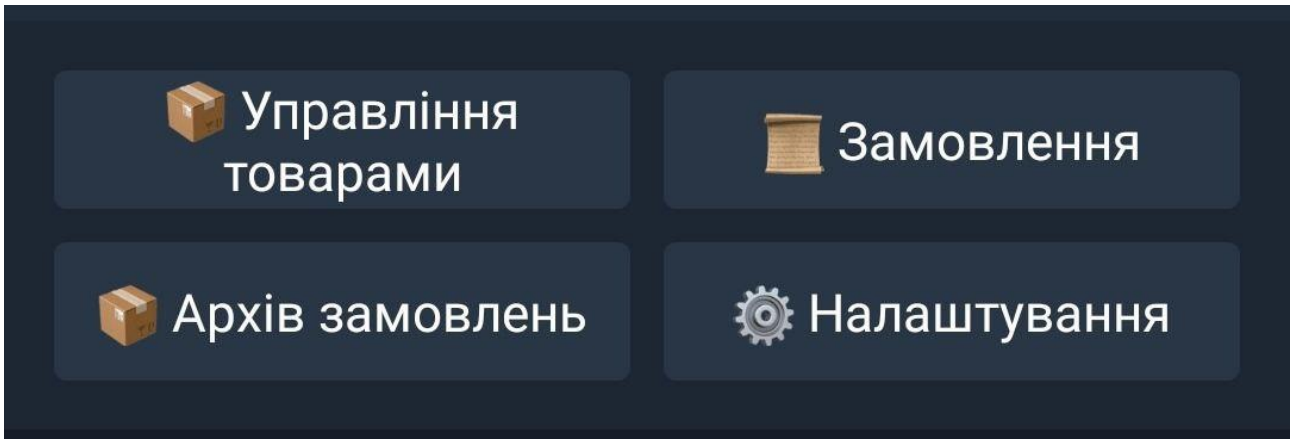

Рисунок 6.5 – Адмін-панель

Рисунок 6.6 – Додавання товару адміністратором

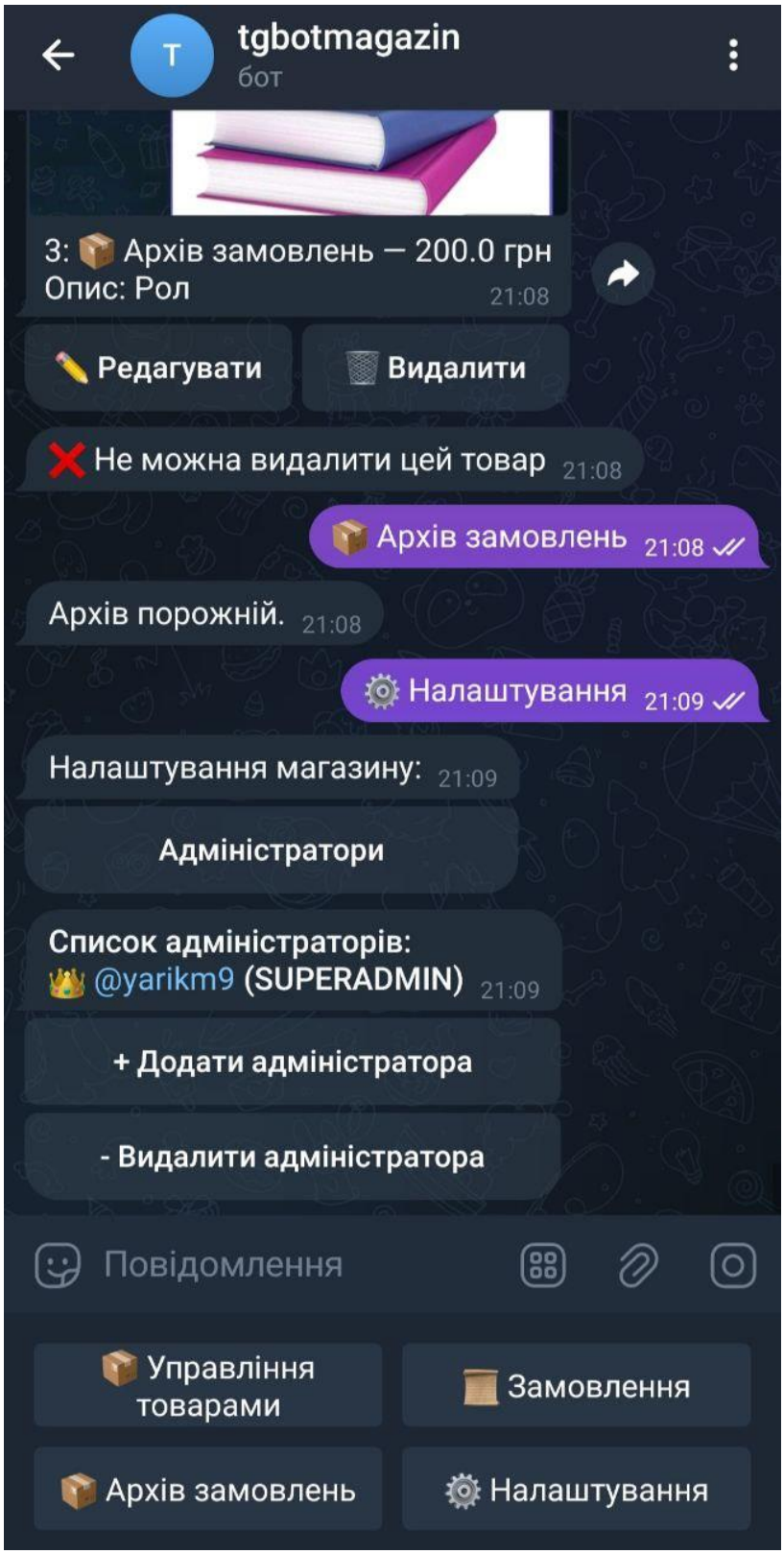

Рисунок 6.7 – Кнопка «Налаштування» для адміністраторів

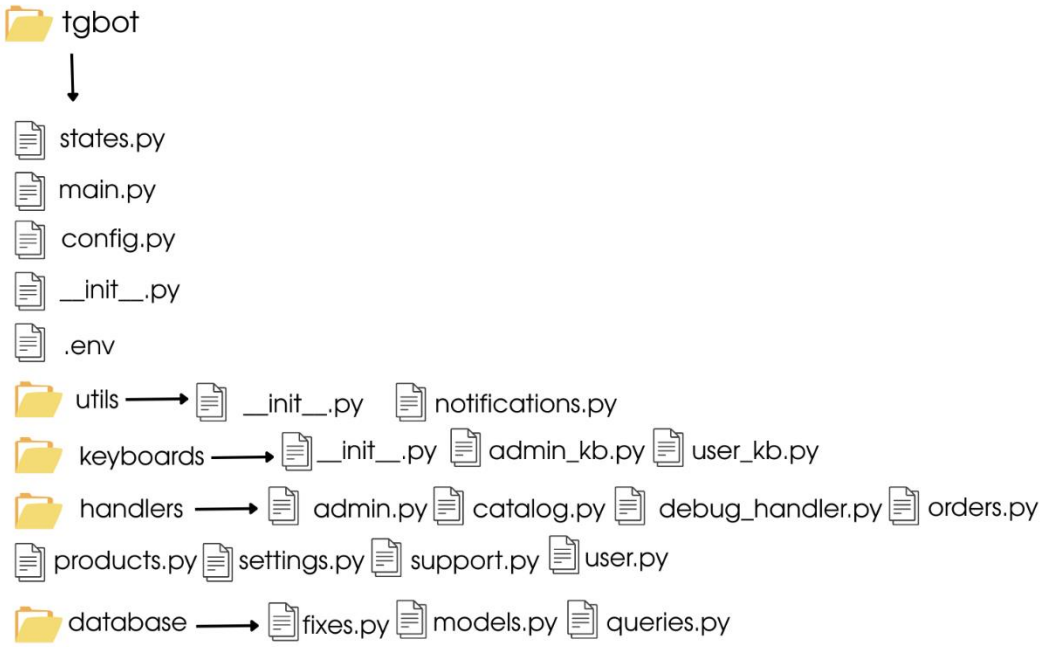

Рисунок 6.8 – Повна структура готового інтернет-магазину

4.6 Тестування та безпека роботи системи

Перед запуском Telegram-магазину було проведено комплексне тестування всіх основних функцій, щоб переконатися, що система працює стабільно та відповідає вимогам. Спочатку перевірили окремі частини програми — як бот взаємодіє з базою даних, чи правильно працюють основні бізнес-функції (наприклад, додавання товару, оформлення замовлення), а також як система реагує на помилки чи некоректні дані. Для цього використовували фреймворк pytest. Це допомогло вчасно знайти та виправити баги на ранньому етапі розробки.

Потім протестували повну роботу бота у реальному Telegram-чаті з тестовим користувачем і адміністратором. Перевірили основні сценарії: перегляд каталогу, додавання товару до кошика, оформлення та підтвердження замовлення, а також роботу адміністратора з цими замовленнями. Також оцінили зручність інтерфейсу для звичайних користувачів — навіть ті, хто раніше не користувався ботами, змогли зробити покупку без проблем. Усі виявлені дрібні недоліки, як-то орфографічні помилки чи неточності у валідації, були швидко виправлені, а повторне тестування показало, що всі функції працюють коректно.

Рисунок 7.1 - Use Case-діаграма основних сценаріїв Telegram-магазину

Особливу увагу приділили безпеці роботи системи, оскільки бот обробляє особисті дані користувачів і може працювати з оплатами. Токен доступу до Telegram-бота зберігається у захищеному файлі, а не у відкритому коді, і не потрапляє у репозиторій. Всі з’єднання з Telegram-серверами відбуваються через зашифровані канали (HTTPS), а сервер бота додатково захищений паролями і двофакторною автентифікацією.

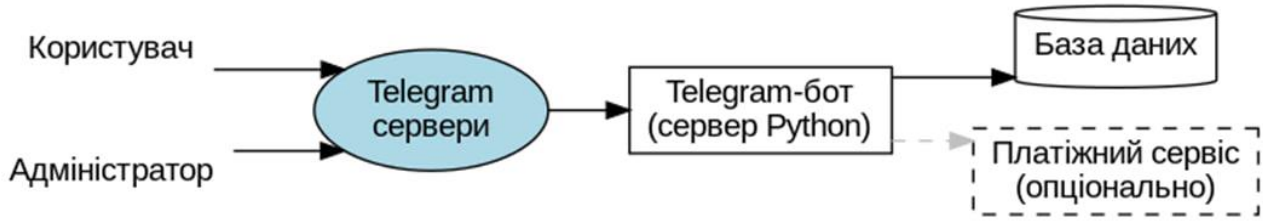

Рисунок 7.2 – Схема Telegram магазину

Усі SQL-запити виконуються безпечним способом, що виключає ризик SQL-ін’єкцій, а дані користувача перевіряються на коректність. Адміністративний функціонал доступний лише перевіреним користувачам (ідентифікатори адміністраторів зберігаються окремо), а всі важливі дії адміністраторів журналюються. База даних регулярно резервується, а політика зберігання персональних даних відповідає сучасним стандартам захисту (наприклад, GDPR). Також сервер має налаштований брандмауер і обмеження на кількість запитів, що захищає систему від спаму чи перевантаження.

Усі ці заходи забезпечують надійність та безпеку Telegram-магазину. Система готова до використання, а при необхідності її захист можна посилювати і надалі, оновлюючи засоби безпеки відповідно до нових викликів.

ВИСНОВОК

У процесі виконання дипломної роботи на тему «Проектування та впровадження інтернет-магазину на платформі Telegram» було проведено аналіз сучасного стану ринку електронної комерції, розглянуто особливості впровадження інтернет-магазинів у месенджерах, а також здійснено практичну розробку програмного рішення із використанням мови Python.

На основі інформаційного огляду визначено основні тенденції розвитку електронної торгівлі, обгрунтовано актуальність використання платформи Telegram для створення сучасних інтернет-магазинів. Теоретична частина роботи дозволила сформувати вимоги до функціоналу системи, спроектувати архітектуру проєкту та вибрати оптимальні підходи для реалізації ключових функцій.

У практичній частині дипломної роботи розроблено інтернет-магазин у вигляді Telegram-бота. Описано основні компоненти проєкту: структуру коду, принципи роботи з базою даних, реалізацію функцій для користувачів і адміністратора. Проведене тестування підтвердило працездатність запропонованого рішення та зручність використання бот-системи для здійснення онлайн-покупок.

Результати роботи свідчать про те, що Telegram-бот може стати ефективним інструментом для автоматизації процесів онлайн-торгівлі, забезпечуючи швидкий старт і гнучке налаштування під потреби бізнесу. Розроблений проєкт є універсальною основою, яку можна розширювати відповідно до зростання вимог і масштабування бізнесу: додавати інтеграції з платіжними системами, службами доставки, особисті кабінети, аналітичні модулі тощо.

Подальші дослідження можуть бути спрямовані на удосконалення алгоритмів взаємодії з користувачем, впровадження елементів штучного інтелекту для персоналізації пропозицій, а також підвищення безпеки обробки персональних даних і фінансових транзакцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Telegram Bot API Documentation. https://core.telegram.org/bots/api
2. aiogram Documentation. https://docs.aiogram.dev/
3. Eric Matthes. *Python Crash Course: A Hands-On, Project-Based Introduction to Programming*. No Starch Press, 2019. 544 p.
4. Al Sweigart. *Automate the Boring Stuff with Python: Practical Programming for Total Beginners*. No Starch Press, 2015. 504 p.
5. Mark Lutz. *Learning Python*. O'Reilly Media, 2013. 1648 p.
6. Guido van Rossum. *Python Programming for the Absolute Beginner*. No Starch Press, 2010. 464 p.
7. Allen Downey. *Think Python: How to Think Like a Computer Scientist*. O'Reilly Media, 2012. 292 p.
8. Sweigart, Al. *Beyond the Basic Stuff with Python: Best Practices for Writing Clean Code*. No Starch Press, 2020. 328 p.
9. SQLite Tutorial. https://www.sqlitetutorial.net/
10. Open-source projects and tutorials on GitHub. https://github.com/aiogram/aiogram https://github.com/python-telegram-bot/python-telegram-bot
11. Academic articles on e-commerce automation and chatbot development (IEEE, ACM Digital Library)
12. University methodology guidelines (for thesis preparation and formatting)
13. Charles Severance. *Python for Everybody: Exploring Data in Python 3*. CreateSpace Independent Publishing Platform, 2016. 244 p.
14. Al Sweigart. *Invent Your Own Computer Games with Python*. CreateSpace Independent Publishing Platform, 2010. 366 p.
15. Telegram Payments Documentation. https://core.telegram.org/bots/payments

16. Real Python. https://realpython.com
17. Jose Portilla. *Complete Python Bootcamp: Go from zero to hero in Python 3*. Udemy, 2020. Online Course.
18. Luciano Ramalho. *Fluent Python: Clear, Concise, and Effective Programming*. O'Reilly Media, 2015. 792 p.
19. GitHub Repository: Telegram Bots Examples (Python). https://github.com/python-telegram-bot/python-telegram-bot
20. OWASP Foundation. *OWASP Secure Coding Practices - Quick Reference Guide*. https://owasp.org/www-project-secure-coding-practices/

ДОДАТКИ

А. Лістинг коду.

main.py:

```
import os
    import sys
    import logging
    import asyncio

    from aiogram import Bot, Dispatcher
    from aiogram.fsm.storage.memory import MemoryStorage
    from database.models import init_db

    from config import BOT_TOKEN

    # Project root to sys.path
    project_root = os.path.dirname(os.path.abspath(__file__))
    sys.path.insert(0, project_root)

    from handlers import (
        user,
        admin,
        support,
        catalog,
        products,
        orders,
        settings,
```

```
    debug_handler,
)

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[logging.FileHandler("bot.log"), logging.StreamHandler()]
)

storage = MemoryStorage()
bot = Bot(token=BOT_TOKEN)
dp = Dispatcher(storage=storage)

async def main():
    init_db()
    dp.include_router(admin.router)
    dp.include_router(user.router)
    dp.include_router(support.router)
    dp.include_router(catalog.router)
    dp.include_router(products.router)
    dp.include_router(orders.router)
    dp.include_router(settings.router)
    dp.include_router(debug_handler.router)

    await bot.delete_webhook(drop_pending_updates=True)
    await dp.start_polling(bot)
    await bot.session.close()
```

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

states.py:

```
from aiogram.fsm.state import State, StatesGroup

class AddProduct(StatesGroup):
    NAME = State()
    DESCRIPTION = State()
    PRICE = State()
    PHOTO = State()

class EditProduct(StatesGroup):
    PRODUCT_ID = State()
    SELECT = State()
    NAME = State()
    DESCRIPTION = State()
    PRICE = State()
    PHOTO = State()
    CONFIRM = State()
    FIELD = State()
    VALUE = State()

class OrderProcess(StatesGroup):
    NAME = State()
    PHONE = State()
    ADDRESS = State()
```

```
    class SupportProcess(StatesGroup):
        MESSAGE = State()
        REPLY = State()


    class AdminNoteProcess(StatesGroup):
        NOTE = State()
        ORDER_ID = State()
```

config.py:

```
import os
    from dotenv import load_dotenv


    load_dotenv()


    BOT_TOKEN = os.getenv("BOT_TOKEN")
    ADMIN_ID = int(os.getenv("ADMIN_ID"))
    SUPERADMIN_ID = int(os.getenv("SUPERADMIN_ID"))
```

__init__.py:

```
import sys
    import os
    sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))


    from aiogram import Bot, Dispatcher
    from config import BOT_TOKEN
    from handlers import admin, user, orders, support
    from database.models import init_db
    from states import (
```

```
    AddProduct,
    EditProduct,
    OrderProcess,
    SupportProcess,
    AdminNoteProcess
)

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

async def setup():
    init_db()
    dp.include_router(admin.router)
    dp.include_router(user.router)
    dp.include_router(orders.router)
    dp.include_router(support.router)
    return dp, bot
```

notifications.py:

```
from aiogram import Bot
    from config import BOT_TOKEN

    bot = Bot(token=BOT_TOKEN)

    async def notify_user(user_id, message_text):
        try:
            await bot.send_message(user_id, message_text)
        except Exception as e:
```

```
            print(f"Помилка надсилання повідомлення: {e}")
```

admin_kb.py:

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton


    def create_admin_keyboard():
        return ReplyKeyboardMarkup(
            keyboard=[
                [KeyboardButton(text="📦 Управління товарами"),
KeyboardButton(text="📜 Замовлення")],
                [KeyboardButton(text="📦 Архів замовлень"),
KeyboardButton(text="⚙️ Налаштування")]
            ],
            resize_keyboard=True
        )
```

user_kb.py:

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton
    from database.queries import is_admin
    from database.queries import get_user_orders


    def create_user_keyboard(user_id):
        buttons = [
            [KeyboardButton(text="🛍️ Каталог"), KeyboardButton(text="🛒
Кошик")],
            [KeyboardButton(text="📦 Мої замовлення")],
            [KeyboardButton(text="ℹ️ Підтримка")]
        ]
        # Додаємо окремо для адміна:
```

```
    if is_admin(user_id):
        buttons.append([KeyboardButton(text="🛠 Адмін-панель")])
    return ReplyKeyboardMarkup(
        keyboard=buttons,
        resize_keyboard=True
    )
```

admin.py:

```
from aiogram import Router, F
    from aiogram.types import Message, CallbackQuery, ReplyKeyboardMarkup, KeyboardButton, InlineKeyboardMarkup, InlineKeyboardButton
    from aiogram.fsm.context import FSMContext
    from aiogram.filters import StateFilter
    from aiogram.fsm.state import State, StatesGroup
    from database.queries import get_products, update_product, delete_product, can_delete_product, is_admin
    from database.queries import get_pending_orders, get_archive_orders, set_order_status
    from keyboards.admin_kb import create_admin_keyboard
    from states import EditProduct, AddProduct
    from config import ADMIN_ID, SUPERADMIN_ID
    from database.queries import (
        get_order_details,
        get_order_details_by_id,
        get_order_items,
        update_order_status_admin
    )
    import json
```

```
router = Router(name="admin_router")


async def send_admin_menu(message: Message):
    kb = create_admin_keyboard()
    await message.answer("👋 Вітаю, адміністратор! Оберіть опцію:", reply_markup=kb)


@router.message(F.text == "📦 Управління товарами")
async def manage_products(message: Message):
    if not is_admin(message.from_user.id):
        return
    kb = InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(text="👁 Переглянути товари", callback_data="view_products")],
        [InlineKeyboardButton(text="➕ Додати товар", callback_data="add_product")]
    ])
    await message.answer("📦 Управління товарами:", reply_markup=kb)


@router.callback_query(F.data == "view_products")
async def view_products_cb(query: CallbackQuery):
    if not is_admin(query.from_user.id):
        await query.answer("Доступно лише адміністратору", show_alert=True)
        return
    products = get_products()
    if not products:
        await query.message.answer("📝 Немає товарів.")
```

```
        else:
            for p in products:
                msg = f"{p['id']}: {p['name']} — {p['price']} грн"
                if p.get("description"):
                    msg += f"\nОпис: {p['description']}"
                if p.get("category"):
                    msg += f"\nКатегорія: {p['category']}"
                kb = InlineKeyboardMarkup(
                    inline_keyboard=[[
                        InlineKeyboardButton(text="✏ Редагувати",
callback_data=f"editprod_{p['id']}"),
                        InlineKeyboardButton(text="🗑 Видалити",
callback_data=f"delprod_{p['id']}")
                    ]]
                )
                if p.get("photo_id"):
                    await query.message.answer_photo(
                        photo=p["photo_id"], caption=msg, reply_markup=kb
                    )
                else:
                    await query.message.answer(msg, reply_markup=kb)
            await query.answer()


        @router.message(F.text == "Додати товар")
        async def add_product_start(message: Message, state: FSMContext):
            await message.answer("Введіть назву товару:")
            await state.set_state(AddProduct.NAME)
```

```
@router.message(StateFilter(AddProduct.NAME))
async def add_product_name(message: Message, state: FSMContext):
    await state.update_data(name=message.text)
    await message.answer("Введіть опис товару:")
    await state.set_state(AddProduct.DESCRIPTION)

@router.message(StateFilter(AddProduct.DESCRIPTION))
async def add_product_desc(message: Message, state: FSMContext):
    await state.update_data(description=message.text)
    await message.answer("Введіть ціну товару (числом):")
    await state.set_state(AddProduct.PRICE)

@router.message(StateFilter(AddProduct.PRICE))
async def add_product_price(message: Message, state: FSMContext):
    try:
        price = float(message.text)
    except Exception:
        await message.answer("Введіть коректну ціну (тільки число):")
        return
    await state.update_data(price=price)
    await message.answer("Надішліть фото товару:")
    await state.set_state(AddProduct.PHOTO)

@router.message(StateFilter(AddProduct.PHOTO))
async def add_product_photo(message: Message, state: FSMContext):
    if not message.photo:
        await message.answer("Будь ласка, надішліть саме ФОТО!")
        return
```

```
        photo_id = message.photo[-1].file_id
        data = await state.get_data()
        # Тут твоя функція додавання товару
        from database.queries import add_product
        add_product(
            name=data['name'],
            description=data['description'],
            price=data['price'],
            photo_id=photo_id
        )
        await message.answer("✅ Товар додано!")
        await state.clear()


    @router.callback_query(F.data.startswith("delprod_"))
    async def delete_product_cb(query: CallbackQuery):
        if not is_admin(query.from_user.id):
            await query.answer("Доступно лише адміністратору",
show_alert=True)
            return
        try:
            product_id = int(query.data.split("_")[1])
            if can_delete_product(product_id):
                delete_product(product_id)
                await query.message.answer(f"✅ Товар №{product_id} видалено!")
                await query.answer("Товар видалено")
            else:
                await query.message.answer("❌ Не можна видалити цей товар")
                await query.answer("Неможливо видалити")
```

```
    except Exception as e:
        print(f"Error deleting product: {e}")
        await query.answer("Помилка при видаленні товару")
        await query.message.answer("❌ Помилка при видаленні товару")


@router.message(F.text.in_(["🛒 Замовлення", "📜 Замовлення"]))
async def show_orders(message: Message):
    if not is_admin(message.from_user.id):
        return
    orders = get_order_details(status="pending")
    if not orders:
        await message.answer("Замовлень немає.")
        return
    kb = InlineKeyboardMarkup(
        inline_keyboard=[
            [InlineKeyboardButton(
                text=f"Замовлення #{order['id']} від {order['created_at'][:16]}",
                callback_data=f"order_{order['id']}"
            )]
            for order in orders
        ]
    )
    await message.answer("Оберіть замовлення:", reply_markup=kb)

@router.callback_query(F.data.startswith("order_"))
async def order_details_cb(query: CallbackQuery):
    if not is_admin(query.from_user.id):
```

```
        await query.answer("Доступно лише адміністратору",
show_alert=True)
        return
    data = query.data
    if "_" not in data:
        await query.answer("Некоректний callback!", show_alert=True)
        return
    order_id_str = data.split("_", 1)[1]
    if not order_id_str.isdigit():
        await query.answer("Некоректний id!", show_alert=True)
        return
    order_id = int(order_id_str)
    try:
        order = get_order_details_by_id(order_id)
        items = get_order_items(order_id)
        if not order:
            await query.message.answer("❌ Замовлення не знайдено.")
            await query.answer()
            return
        txt = (
            f"🛒 <b>Замовлення #{order['id']}</b>\n"
            f"👤 ПІБ: {order.get('name', '-')}\n"
            f"📱 Тел: {order.get('phone', '-')}\n"
            f"🏠 Адреса: {order.get('address', '-')}\n"
            f"Username: @{order.get('username', '-')}\n"
            f"Статус: {order.get('status', '-')}\n"
            f"Дата: {order.get('created_at', '-')}\n\n"
            f"📦 <b>Товари:</b>\n"
```

```
            )
            total = 0
            for item in items:
                txt += f"• {item['name']} x{item['quantity']} = {item['price'] *
item['quantity']} грн\n"
                total += item['price'] * item['quantity']
            txt += f"\n💰 <b>Загалом:</b> {total} грн"
            kb = InlineKeyboardMarkup(
                inline_keyboard=[
                    [
                        InlineKeyboardButton(text="✅ Виконано",
callback_data=f"done_{order_id}"),
                        InlineKeyboardButton(text="❌ Відхилити",
callback_data=f"cancel_{order_id}")
                    ]
                ]
            )
            await query.message.answer(txt, parse_mode="HTML",
reply_markup=kb)
            await query.answer()
        except Exception as e:
            await query.answer("❌ Сталася помилка!", show_alert=True)
            print("Admin order_details_cb error:", e)


    @router.callback_query(F.data.startswith("done_"))
    async def order_done_cb(query: CallbackQuery):
        if not is_admin(query.from_user.id):
            await query.answer("Доступно лише адміністратору",
show_alert=True)
```

```
            return
        data = query.data
        if "_" not in data:
            await query.answer("Некоректний callback!", show_alert=True)
            return
        order_id_str = data.split("_", 1)[1]
        if not order_id_str.isdigit():
            await query.answer("Некоректний id!", show_alert=True)
            return
        order_id = int(order_id_str)
        update_order_status_admin(order_id, status="done")
        await query.message.answer("✅ Замовлення відмічено як виконане!")
        await query.answer()


    @router.callback_query(F.data.startswith("cancel_"))
    async def order_cancel_cb(query: CallbackQuery):
        if not is_admin(query.from_user.id):
            await query.answer("Доступно лише адміністратору",
show_alert=True)
            return
        data = query.data
        if "_" not in data:
            await query.answer("Некоректний callback!", show_alert=True)
            return
        order_id_str = data.split("_", 1)[1]
        if not order_id_str.isdigit():
            await query.answer("Некоректний id!", show_alert=True)
            return
```

```
        order_id = int(order_id_str)
        update_order_status_admin(order_id, status="canceled")
        await query.message.answer("❌ Замовлення відмінено.")
        await query.answer()


    @router.callback_query(F.data.startswith("editprod_"))
    async def edit_product_cb(query: CallbackQuery, state: FSMContext):
        if not is_admin(query.from_user.id):
            await query.answer("Доступно лише адміністратору",
show_alert=True)
            return
        data = query.data
        if "_" not in data:
            await query.answer("Некоректний callback!", show_alert=True)
            return
        product_id_str = data.split("_", 1)[1]
        if not product_id_str.isdigit():
            await query.answer("Некоректний id!", show_alert=True)
            return
        product_id = int(product_id_str)
        await state.update_data(product_id=product_id)
        await query.message.answer("Що редагувати?\nОбери:",
reply_markup=InlineKeyboardMarkup(
            inline_keyboard=[
                [InlineKeyboardButton(text="✏ Назву",
callback_data="edit_name")],
                [InlineKeyboardButton(text="📝 Опис", callback_data="edit_desc")],
                [InlineKeyboardButton(text="💸 Ціну", callback_data="edit_price")],
```

```
            [InlineKeyboardButton(text="🖼 Фото", callback_data="edit_photo")]
        ]
    ))
    await state.set_state(EditProduct.SELECT)
    await query.answer()

@router.callback_query(EditProduct.SELECT, F.data == "edit_name")
async def edit_name_cb(query: CallbackQuery, state: FSMContext):
    await query.message.answer("Введи нову назву товару:")
    await state.set_state(EditProduct.NAME)
    await query.answer()

@router.message(EditProduct.NAME)
async def save_new_name(message: Message, state: FSMContext):
    data = await state.get_data()
    update_product(data['product_id'], name=message.text)
    await message.answer("✅ Назву оновлено!")
    await state.clear()

@router.callback_query(EditProduct.SELECT, F.data == "edit_desc")
async def edit_desc_cb(query: CallbackQuery, state: FSMContext):
    await query.message.answer("Введи новий опис товару:")
    await state.set_state(EditProduct.DESCRIPTION)
    await query.answer()

@router.message(EditProduct.DESCRIPTION)
async def save_new_desc(message: Message, state: FSMContext):
    data = await state.get_data()
```

```
    update_product(data['product_id'], description=message.text)
    await message.answer("✅ Опис оновлено!")
    await state.clear()

@router.callback_query(EditProduct.SELECT, F.data == "edit_price")
async def edit_price_cb(query: CallbackQuery, state: FSMContext):
    await query.message.answer("Введи нову ціну товару (тільки число):")
    await state.set_state(EditProduct.PRICE)
    await query.answer()

@router.message(EditProduct.PRICE)
async def save_new_price(message: Message, state: FSMContext):
    data = await state.get_data()
    try:
        price = float(message.text)
        update_product(data['product_id'], price=price)
        await message.answer("✅ Ціну оновлено!")
    except ValueError:
        await message.answer("❌ Введи правильне число!")
        return
    await state.clear()

@router.callback_query(EditProduct.SELECT, F.data == "edit_photo")
async def edit_photo_cb(query: CallbackQuery, state: FSMContext):
    await query.message.answer("Надішли нове фото товару:")
    await state.set_state(EditProduct.PHOTO)
    await query.answer()
```

```
@router.message(EditProduct.PHOTO)
async def save_new_photo(message: Message, state: FSMContext):
    data = await state.get_data()
    if not message.photo:
        await message.answer("❌ Надішли саме фото!")
        return
    file_id = message.photo[-1].file_id
    update_product(data['product_id'], photo_id=file_id)
    await message.answer("✅ Фото оновлено!")
    await state.clear()


@router.message(F.text == "📦 Архів замовлень")
async def show_archive_orders(message: Message):
    orders = get_archive_orders()
    if not orders:
        await message.answer("Архів порожній.")
        return
    for order in orders:
        await send_order_brief(message, order)


async def send_order_brief(message, order):
    text = (
        f"<b>ID замовлення:</b> {order['id']}\n"
        f"<b>Ім'я:</b> {order.get('name', '-')}\n"
        f"<b>Телефон:</b> {order.get('phone', '-')}\n"
        f"<b>Адреса замовлення:</b> {order.get('address', '-')}\n"
        f"<b>Дата створення:</b> {order.get('created', order.get('created_at', '-
'))}\n"
```

```
        f"<b>Статус:</b> {order.get('status', '-')}\n"
        f"<b>Товари:</b>\n"
    )
    items = get_order_items(order['id'])
    for item in items:
        text += f"- {item['name']} x{item['quantity']} = {item['price']} грн\n"

    buttons = []
    if order.get('status') == 'pending':
        buttons = [
            [InlineKeyboardButton(text="✅ Виконано",
callback_data=f"order_done_{order['id']}"),
             InlineKeyboardButton(text="❌ Відхилити",
callback_data=f"order_reject_{order['id']}")]
        ]
    await message.answer(text, parse_mode="HTML",

reply_markup=InlineKeyboardMarkup(inline_keyboard=buttons) if buttons else
None)



@router.callback_query(F.data.startswith("order_done_"))
async def mark_order_done(query: CallbackQuery):
    order_id = int(query.data.split("_")[-1])
    set_order_status(order_id, "completed")
    await query.answer("Позначено як виконане!")
    await query.message.edit_reply_markup(reply_markup=None)
    await query.message.answer("✅ Замовлення виконано!")
```

```
@router.callback_query(F.data.startswith("order_reject_"))
async def mark_order_reject(query: CallbackQuery):
    order_id = int(query.data.split("_")[-1])
    set_order_status(order_id, "rejected")
    await query.answer("Позначено як відхилене.")
    await query.message.edit_reply_markup(reply_markup=None)
    await query.message.answer("❌ Замовлення відхилено.")


@router.message(F.text == "📦 Архів замовлень")
async def admin_archive_orders(message: Message):
    orders = get_archive_orders()
    if not orders:
        await message.answer("Архів порожній.")
        return
    for order in orders:
        status = order.get('status', '')
        text = (
            f"🛒 <b>Замовлення #{order['id']}</b>\n"
            f"Статус: <b>{status}</b>\n"
            f"Дата: {order.get('created', '')}\n"
        )
        if 'items_json' in order:
            import json
            items = json.loads(order['items_json'])
            for item in items:
                text += f"- {item['name']} x{item['quantity']} = {item['price']} грн\n"
        await message.answer(text, parse_mode="HTML")
```

catalog.py:

```
from aiogram import Router, F
    from aiogram.types import Message, CallbackQuery, InlineKeyboardMarkup, InlineKeyboardButton
    from aiogram.utils.keyboard import InlineKeyboardBuilder
    from database.models import connect_db
    from database.queries import add_to_cart

    router = Router()

    @router.message(F.text == "🛍️ Каталог")
    async def show_catalog(message: Message):
        try:
            with connect_db() as conn:
                categories = conn.execute(
                    "SELECT DISTINCT category FROM products WHERE category IS NOT NULL"
                ).fetchall()

                # Якщо категорії не налаштовані, показуємо всі товари
                if not categories:
                    await show_all_products(message)
                    return

                # Інакше показуємо список категорій
                builder = InlineKeyboardBuilder()
```

```
            # Додаємо кнопки категорій
            for category in categories:
                builder.button(
                    text=category['category'],
                    callback_data=f"category_{category['category']}"
                )

            # Додаємо кнопку "Всі товари"
            builder.button(
                text="Всі товари",
                callback_data="all_products"
            )

            builder.adjust(1)  # По одній кнопці в рядку

            await message.answer("Виберіть категорію:",
reply_markup=builder.as_markup())
        except Exception as e:
            await message.answer(f"❌ Помилка: {str(e)}")

    @router.callback_query(F.data == "all_products")
    async def show_all_products_callback(query: CallbackQuery):
        await show_all_products(query.message)
        await query.answer()

    async def show_all_products(message):
        try:
            with connect_db() as conn:
```

```
                products = conn.execute(
                    "SELECT id, name, price, description FROM products ORDER BY
id"
                ).fetchall()

                if not products:
                    await message.answer("📦 Каталог товарів порожній")
                    return

                # Розбиваємо товари на сторінки по 5 товарів
                products_per_page = 5
                total_pages = (len(products) + products_per_page - 1) //
products_per_page

                # Створюємо клавіатуру для першої сторінки
                await show_products_page(message, products, 1, total_pages)
            except Exception as e:
                await message.answer(f"❌ Помилка: {str(e)}")

        @router.callback_query(F.data.startswith("category_"))
        async def show_category(query: CallbackQuery):
            category = query.data.split("_", 1)[1]

            try:
                with connect_db() as conn:
                    products = conn.execute(
                        "SELECT id, name, price, description FROM products WHERE
category = ? ORDER BY id",
                        (category,)
```

```
            ).fetchall()

            if not products:
                await query.message.answer(f"В категорії '{category}' немає товарів")
                return await query.answer()

            # Розбиваємо товари на сторінки по 5 товарів
            products_per_page = 5
            total_pages = (len(products) + products_per_page - 1) // products_per_page

            # Створюємо клавіатуру для першої сторінки категорії
            await show_products_page(query.message, products, 1, total_pages, category)
        except Exception as e:
            await query.message.answer(f"❌ Помилка: {str(e)}")

        await query.answer()

    @router.callback_query(F.data.startswith("page_"))
    async def handle_page_navigation(query: CallbackQuery):
        parts = query.data.split("_")
        page = int(parts[1])
        total_pages = int(parts[2])
        category = parts[3] if len(parts) > 3 else None

        try:
            with connect_db() as conn:
```

```
            if category and category != "None":
                products = conn.execute(
                    "SELECT id, name, price, description FROM products WHERE
category = ? ORDER BY id",
                    (category,)
                ).fetchall()
            else:
                products = conn.execute(
                    "SELECT id, name, price, description FROM products ORDER
BY id"
                ).fetchall()

            if not products:
                await query.message.answer("📦 Каталог товарів порожній")
                return await query.answer()

            await show_products_page(query.message, products, page, total_pages,
category)
        except Exception as e:
            await query.message.answer(f"❌ Помилка: {str(e)}")

        await query.answer()

    async def show_products_page(message, products, page, total_pages,
category=None):
        products_per_page = 5
        start_idx = (page - 1) * products_per_page
        end_idx = min(start_idx + products_per_page, len(products))
        page_products = products[start_idx:end_idx]
```

```
        for product in page_products:
            # Форматуємо ціну
            price = product['price']
            if price == int(price):
                price = int(price)
            # Оформлення для юзера
            text = (
                f"<b>{product['name']}</b>\n"
                f"{product.get('description','')}\n\n"
                f"<b>Ціна:</b> {price} грн"
            )

            kb = InlineKeyboardMarkup(inline_keyboard=[
                [InlineKeyboardButton(text="🛒 В кошик",
callback_data=f"add_to_cart_{product['id']}")]
            ])

            try:
                # Отримуємо фото товару
                with connect_db() as conn:
                    photo_id = conn.execute(
                        "SELECT photo_id FROM products WHERE id = ?",
                        (product['id'],)
                    ).fetchone()
                if photo_id and photo_id['photo_id']:
                    await message.answer_photo(
                        photo=photo_id['photo_id'],
```

```
                caption=text,
                reply_markup=kb,
                parse_mode="HTML"
            )
        else:
            await message.answer(text, reply_markup=kb,
parse_mode="HTML")
    except Exception as e:
        await message.answer(f"❌ Помилка з фото товару: {str(e)}")
        await message.answer(text, reply_markup=kb, parse_mode="HTML")


# Клавіші сторінок
builder = InlineKeyboardBuilder()
if page > 1:
    builder.button(text="⬅ Назад", callback_data=f"page_{page-
1}_{total_pages}_{category}")
builder.button(text=f"{page}/{total_pages}", callback_data="current_page")
if page < total_pages:
    builder.button(text="➡ Вперед",
callback_data=f"page_{page+1}_{total_pages}_{category}")
builder.button(text="🔙 До категорій",
callback_data="back_to_categories")
builder.adjust(3, 1)
await message.answer("📄 Сторінка:", reply_markup=builder.as_markup())


@router.callback_query(F.data == "current_page")
async def handle_current_page(query: CallbackQuery):
    await query.answer("Поточна сторінка")
```

```
@router.callback_query(F.data == "back_to_categories")
async def back_to_categories(query: CallbackQuery):
    await show_catalog(query.message)
    await query.answer()

@router.callback_query(F.data.startswith("add_to_cart_"))
async def add_product_to_cart(query: CallbackQuery):
    product_id = int(query.data.split("_")[3])
    user_id = query.from_user.id

    try:
        result = add_to_cart(user_id, product_id)
        await query.answer("✅ Товар додано в кошик!")
    except Exception as e:
        await query.message.answer(f"❌ Помилка: {str(e)}")
        await query.answer()
```