Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – бакалавр
за освітньо-професійною програмою
«Комп'ютерні науки»
зі спеціальності
122 – Комп'ютерні науки

тема роботи:

«Розробка Telegram-боту для інтернет-магазину з використанням інтегрованої платіжної системи Telegram-Stars»

Виконав студент гр. КН-21 _____ Ласісі Ідріс Оладіпупо

Керівник _____ Тронь В. В.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2025

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *29* » *січня* 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи КН-21 *Ласісі Ідріс Оладіпупо*

1. Тема кваліфікаційної роботи: *«Розробка Telegram-боту для інтернет-магазину з використанням інтегрованої платіжної системи Telegram-Stars»*

затверджено наказом по університету № *254с* від *13.05.2025 р.*

2. Термін здачі кваліфікаційної роботи: *06.06.2025 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 78с., додатки, презентація у Microsoft PowerPoint (15 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 доц. Тронь В. В.

_____

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*01.04.25*
2	*Розділ 1*	*05.04.25*
3	*Розділ 2*	*01.05.25*
4	*Висновки*	*25.05.25*
5	*Оформлення кваліфікаційної роботи*	*28.05.25*
6	*Підготовка презентації та графічного матеріалу*	*20.05.25*
7	*Підготовка доповіді до захисту*	*05.06.25*

6. Дата видачі завдання: *28.01.2025р.*

Керівник _____ / Тронь В. В./

7. Запевнення: *Я, Ласісі Ідріс Оладіпупо, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ / Ласісі Ідріс Оладіпупо /

АНОТАЦІЯ

Ласісі Ідріс Оладіпупо. «Розробка Telegram-боту для інтернет-магазину з використанням інтегрованої платіжної системи Telegram-Stars» : кваліфікаційна робота бакалавра : 122 – Комп’ютерні науки. Кривий Ріг. Криворізький національний університет, 2025. 62 с.

Розробка Telegram-бота для продажу цифрових товарів є актуальною, оскільки дозволяє швидко запускати інтернет-магазини з мінімальними витратами та забезпечує високу мобільність бізнесу.

Метою роботи є розробка повнофункціонального Telegram-бота для інтернет-магазину цифрових товарів з інтегрованою платіжною системою Telegram-Stars, який забезпечує зручну взаємодію з користувачами, обробку замовлень, адміністрування контенту та безпечне зберігання даних.

У першому розділі було розглянуто призначення, функціональні можливості та особливості використання Telegram-ботів як ефективного інструменту взаємодії між бізнесом і кінцевим користувачем. На основі цього аналізу було обґрунтовано доцільність вибору саме Telegram як платформи для реалізації програмного продукту завдяки її відкритості, широкому функціоналу.

У другому розділі було здійснено повний цикл розробки Telegram-боту для інтернет-магазину цифрових товарів з інтеграцією платіжної системи Telegram Stars. Виконано реєстрацію Telegram-бота за допомогою сервісу BotFather, налаштовано обробку webhook-запитів та інтегровано платіжну систему Stars для здійснення транзакцій безпосередньо в межах месенджера. Таким чином, у результаті розробки було створено повноцінний Telegram-бот, який забезпечує повний цикл взаємодії користувача з інтернет-магазином цифрових товарів - від перегляду до покупки й отримання продукту.

ІНТЕГРОВАНА ПЛАТІЖНА СИСТЕМА, ІНТЕРНЕТ МАГАЗИН ЦИФРОВИХ ТОВАРІВ, TELEGRAM-БОТ, AIOGRAM 3 , PYTHON, SQLALCHEMY, TELEGRAM-STARS

ЗМІСТ

ВСТУП

Чат-боти є сучасним технологічним рішенням, яке стрімко набуває популярності у сфері бізнесу. Їх впровадження сприяє налагодженню стабільного зв’язку з клієнтами та забезпеченню швидкого й зручного обслуговування.

Однією з ключових переваг чат-ботів є їхня постійна доступність - вони функціонують цілодобово, без вихідних. Це дозволяє користувачам звертатися за допомогою у будь-який зручний для них момент, що значно підвищує зручність взаємодії, оптимізує процес оформлення замовлень та полегшує доступ до інформації про товари.

У сучасному цифровому середовищі Telegram став не лише популярним месенджером, але й ефективною платформою для реалізації бізнес-процесів, зокрема у сфері електронної комерції. Зростаюча популярність Telegram-ботів зумовлена їхньою доступністю, багатофункціональністю та простотою використання для кінцевого споживача. Боти дозволяють автоматизувати обробку замовлень, прийом платежів, комунікацію з клієнтами та керування цифровими товарами.

Метою роботи є розробка повнофункціонального Telegram-бота для інтернет-магазину цифрових товарів з інтегрованою платіжною системою Telegram-Stars, який забезпечує зручну взаємодію з користувачами, обробку замовлень, адміністрування контенту та безпечне зберігання даних.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

– Проаналізувати можливості Telegram Bot API та платіжної системи Telegram-Stars.

– Створити архітектуру Telegram-бота на основі фреймворку Aiogram 3, дотримуючись принципів асинхронного програмування.

– Реалізувати систему управління даними через SQLAlchemy 2 з використанням бази даних SQLite та драйвера Aiosqlite.

– Імплементувати валідацію та конфігурацію даних за допомогою Pydantic 2.

– Налаштувати міграції бази даних з використанням Alembic для зручного управління змінами в структурі.

– Розробити адміністративну панель для керування товарами, користувачами та замовленнями.

– Створити користувацький інтерфейс бота з можливістю перегляду товарів, додавання до кошика та оформлення замовлення.

– Інтегрувати платіжну систему Telegram-Stars для прийому платежів.

– Забезпечити логування, обробку помилок та безпеку даних.

– Протестувати бота на різних сценаріях взаємодії та задокументувати процес розробки.

З появою в Telegram вбудованої платіжної системи Telegram-Stars, відкрилися нові можливості для створення повністю інтегрованих сервісів без потреби у зовнішніх платіжних шлюзах. Це дозволяє суттєво спростити процес покупки, зменшити технічні витрати на інтеграцію та підвищити зручність для користувачів.

Розробка Telegram-бота для продажу цифрових товарів є актуальною, оскільки дозволяє швидко запускати інтернет-магазини з мінімальними витратами та забезпечує високу мобільність бізнесу. Використання сучасних асинхронних технологій Python (Aiogram 3, SQLAlchemy 2, Aiosqlite, Pydantic 2, Alembic) забезпечує високу продуктивність, масштабованість і гнучкість рішення.

Таким чином, дана розробка відповідає актуальним потребам малого та середнього бізнесу, а також демонструє ефективне застосування сучасних технологій у реальному проєкті.

РОЗДІЛ 1
ПРИЗНАЧЕННЯ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ СТВОРЕННЯ ТЕЛЕГРАМ-БОТУ

1.2 Призначення та можливості Telegram-ботів

Telegram-бот — це спеціальна програма, яка забезпечує обмін інформацією з користувачем через платформу Telegram, дотримуючись заданого алгоритму або використовуючи елементи штучного інтелекту. Популярність таких ботів стрімко зростає завдяки їхній здатності замінювати традиційні мобільні додатки, надаючи доступ до послуг без необхідності встановлення окремих програм. Це особливо зручно для користувачів, які можуть взаємодіяти з сервісами безпосередньо у звичному месенджері.

Telegram, поряд із такими компаніями, як Facebook, Microsoft і Viber, активно розвиває власну платформу для створення ботів. Telegram-боти вже стали важливою частиною цифрової екосистеми багатьох українських та міжнародних організацій. Зокрема, популярними прикладами є RailwayBot від Укрзалізниці та NovaPoshtaBot від Нової Пошти.

Створення Telegram-ботів вимагає знань у галузі серверного програмування, розуміння принципів роботи API Telegram Bot, вміння працювати з веб-технологіями, такими як HTTP(S), а також навичок у розробці асинхронних застосунків. Це забезпечує ефективну взаємодію між клієнтом і ботом у реальному часі та відкриває широкі можливості для автоматизації бізнес-процесів.

У сучасному цифровому середовищі Telegram став не лише популярним месенджером, але й ефективною платформою для реалізації бізнес-процесів, зокрема у сфері електронної комерції. Зростаюча популярність Telegram-ботів зумовлена їхньою доступністю, багатофункціональністю та простотою використання для кінцевого споживача. Боти дозволяють автоматизувати

обробку замовлень, прийом платежів, комунікацію з клієнтами та керування цифровими товарами.

З появою в Telegram вбудованої платіжної системи Telegram-Stars, відкрилися нові можливості для створення повністю інтегрованих сервісів без потреби у зовнішніх платіжних шлюзах. Це дозволяє суттєво спростити процес покупки, зменшити технічні витрати на інтеграцію та підвищити зручність для користувачів.

Розробка Telegram-бота для продажу цифрових товарів є актуальною, оскільки дозволяє швидко запускати інтернет-магазини з мінімальними витратами та забезпечує високу мобільність бізнесу. Використання сучасних асинхронних технологій Python (Aiogram 3, SQLAlchemy 2, Aiosqlite, Pydantic 2, Alembic) забезпечує високу продуктивність, масштабованість і гнучкість рішення.

1.2 Структура та порівняльний аналіз месенджерів

Існують різні формати взаємодії користувача з Telegram-ботом: за допомогою кнопок, текстових повідомлень або голосових команд. Найпоширенішим є кнопковий інтерфейс, у якому діалог побудовано на виборі з запропонованих варіантів - категорій, дій або відповідей. Користувач просто натискає відповідну кнопку, що значно спрощує процес навігації та зменшує ймовірність помилок у спілкуванні.

На рисунку 1.1 наведено модель взаємодії ботом та платформою месенджера.

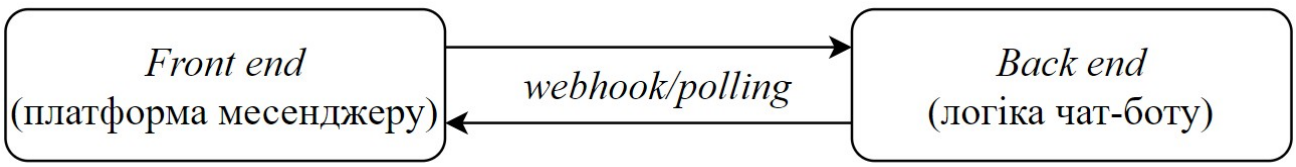

Рисунок 1.1 – Взаємодія боту з месенджером

Back end - це серверна частина додатку, яка відповідає за обробку подій, що надходять від клієнтської сторони (front end), зокрема дій та команд користувача.

Front end у контексті боту - це клієнтська платформа, через яку користувачі взаємодіють із ботом. Вона може бути реалізована в різних месенджерах, таких як Telegram, Facebook Messenger, Skype, Viber тощо.

Webhook виступає каналом зв'язку між клієнтською та серверною частинами (рис. 1.2). Коли на стороні front end відбувається якась подія (наприклад, користувач надсилає повідомлення), платформа месенджера автоматично надсилає HTTP POST-запит до серверної частини (back end). Найчастіше дані передаються у форматі JSON. Для безпечної взаємодії між сторонами зазвичай використовується автентифікація, яка може здійснюватися різними методами - наприклад, через токени або цифрові підписи.

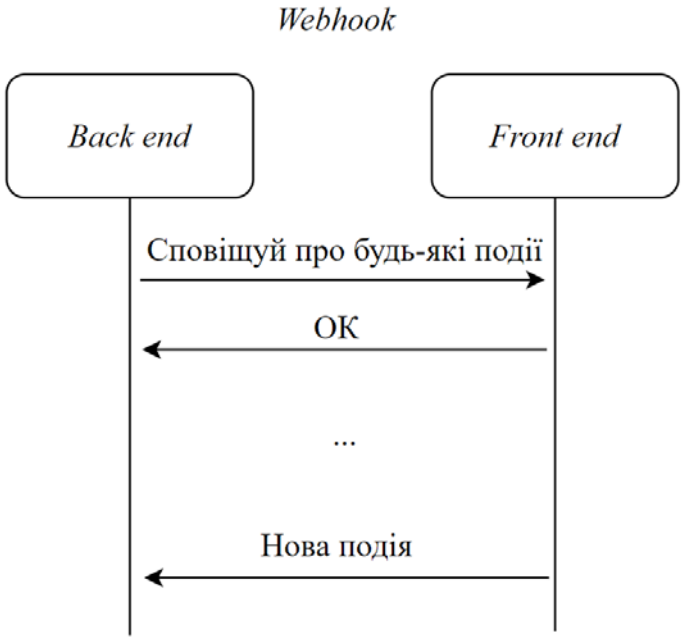

Рисунок 1.2 – Структура *Webhook*

Polling - це альтернативний механізм взаємодії між серверною (back end) та клієнтською (front end) частинами, за якого саме сервер ініціює запит до платформи месенджера. У цьому випадку back end періодично надсилає HTTP POST або GET-запити до front end, очікуючи на появу нових подій (рис. 1.3).

З’єднання залишається активним до моменту, поки не настане певна подія, після чого платформа відповідає, а цикл запитів повторюється.

Обмін даними зазвичай здійснюється у форматі JSON або у вигляді URL query string. Для забезпечення безпеки використовується автентифікація через унікальний токен Telegram-бота, який передається разом із кожним запитом у визначеній структурі.

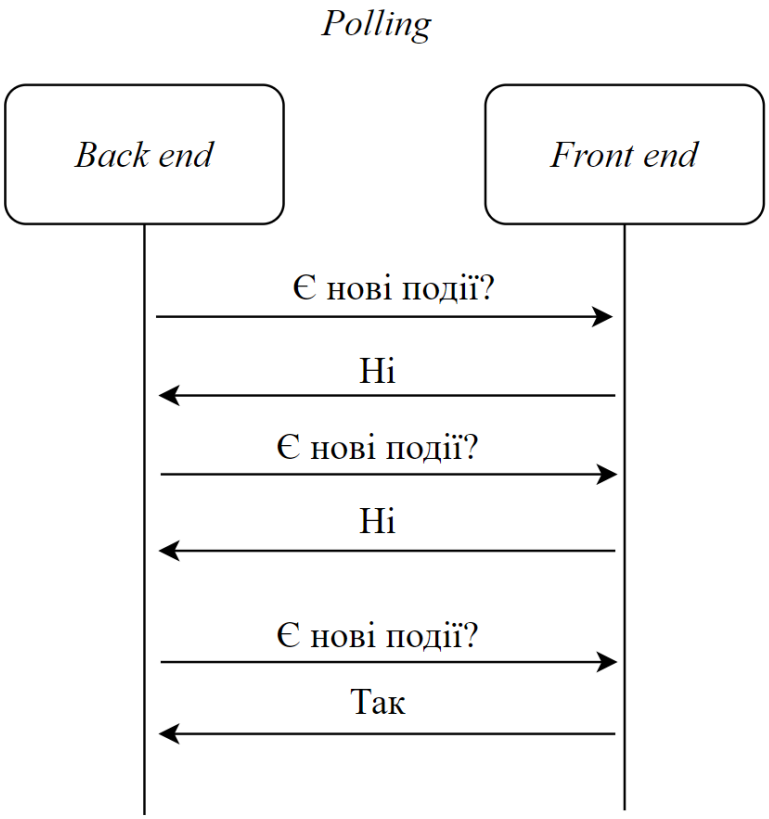

Рисунок 1.3 – Структура Polling

Для аналізу було розглянуто такі ключові параметри: спосіб взаємодії з ботом, рівень доступності, тип інтерфейсу, механізми зв’язку, протоколи та формати передачі даних, а також базові функціональні можливості, зокрема обмін повідомленнями, проведення опитувань, передача файлів, підтримка покупок та інтеграція ігрових елементів. У таблиці 1.1 представлено порівняльну характеристику найпопулярніших месенджерів, які підтримують інтеграцію з бот-платформами.

Таблиця 1.1 – Порівняльна характеристика платформ для чат-ботів

	Facebook	***Telegram***	***Skype***	***Viber***
Форма використання	Умовно безоплатна	Безоплатна	Умовно безоплатна	Умовно безоплатна
Доступи	Доданий до групи, за підпискою, вбудований в діалог	Доданий до групи, за підпискою, вбудований в діалог	Доданий до групи, за підпискою	За підпискою, вбудований в діалог
Інтерфейси користувача	Текстовий, кнопковий, голосовий	Текстовий, кнопковий, голосовий	Текстовий, кнопковий, голосовий	Текстовий, кнопковий, голосовий
Механізми зв'язку	*webhook*	*webhook, polling*	*webhook*	*webhook*
Протоколи передачі даних	*https*			
Формати передачі даних	*JSON, multipart/form-data*	*JSON, URL query, multipart/form-data*	*JSON, multipart/form-data*	*JSON, multipart/form-data*
Функціональні можливості	Листування, опитування, передача файлів, магазин покупок, ігри	Листування, опитування, передача файлів, магазин покупок, ігри	Листування, передача файлів, магазин покупок, ігри	Листування, передача файлів, магазин покупок, ігри

На основі проведеного аналізу було визначено, що саме Telegram є найбільш оптимальною платформою для реалізації поставленої мети кваліфікаційної роботи. Такий вибір зумовлений низкою переваг, серед яких:

- Повністю безкоштовне використання, що робить Telegram доступним як для розробників, так і для кінцевих користувачів;
- Підтримка механізму Polling, який значно спрощує процес розробки та тестування на початкових етапах створення системи;
- Широкий набір функціональних можливостей, що дозволяє не лише задовольнити поточні вимоги до системи, але й забезпечує гнучкість для подальшого вдосконалення та масштабування.

Таким чином, Telegram повністю відповідає технічним і функціональним критеріям, необхідним для ефективної реалізації Telegram-бота для інтернет-магазину.

1.3 Аналіз й вибір інструментів реалізації

1.3.1 Вибір мови програмування

Вибір Python зумовлений його простотою, сучасністю, наявністю асинхронних інструментів та великою екосистемою бібліотек, що дозволяє створювати ефективні, масштабовані та підтримувані Telegram-боти.

Python має дуже чистий, читабельний синтаксис, що значно спрощує процес розробки, особливо для складних проєктів. Це дозволяє швидко писати код і легко його підтримувати.

Python має широкий вибір готових бібліотек і фреймворків, що спрощують розробку ботів:

- Aiogram - потужний асинхронний фреймворк для Telegram-ботів.
- SQLAlchemy - популярний ORM для роботи з базами даних.
- Pydantic - сучасний інструмент для валідації даних.
- Та багато інших для тестування, обробки даних, роботи з API.

З виходом Python 3.7+ мова отримала нативну підтримку асинхронності (async/await), що ідеально підходить для створення високопродуктивних мережевих застосунків, зокрема Telegram-ботів, які повинні швидко реагувати на велику кількість одночасних запитів.

Python працює на різних операційних системах (Windows, Linux, macOS), що робить розробку і розгортання більш гнучкими.

Завдяки простоті і швидкості розробки на Python, можна швидко створити робочий прототип бота (MVP), а потім поступово розширювати функціонал і масштабувати систему.

Python легко інтегрується з різними базами даних (SQLite, PostgreSQL, MySQL тощо) і зовнішніми API, що необхідно для створення повнофункціонального бота.

1.3.2 Вибір фреймворку для розробки Telegram-бота

Aiogram 3 - сучасний Python-фреймворк, побудований на основі асинхронної бібліотеки asyncio. Він розроблений спеціально для ефективної роботи з Telegram Bot API.

Переваги:

- Повністю асинхронний, що дозволяє обробляти велику кількість одночасних запитів без блокувань.
- Легко інтегрується з іншими асинхронними бібліотеками Python.
- Активна спільнота та регулярне оновлення.
- Вбудовані засоби для роботи з inline-кнопками, командами, фільтрами, FSM (машина станів).

Недоліки:

- Для новачків може здатися складним через асинхронність.
- Вимагає розуміння async/await.

Таблиця 1.2 – Порівняльний аналіз

Технологія	Опис	Переваги	Недоліки
Aiogram 3	Асинхронний фреймворк на Python для Telegram-ботів	Висока продуктивність, підтримка async/await, активна спільнота, детальна документація	Може мати криву навчання для початківців
python-telegram-bot	Синхронний фреймворк з деякою підтримкою async	Простий у використанні, популярний	Менш ефективний в асинхронних задачах
Telethon	Бібліотека для Telegram API, орієнтована на user-ботів	Гнучкість, підтримка MTProto	Складніший для початківців, більше орієнтований на user-ботів

Aiogram є одним із найбільш оптимізованих і популярних фреймворків для Telegram-ботів на Python. Він дозволяє будувати масштабовані, швидкі боти з мінімальними накладними витратами, що ідеально підходить для інтернет-магазину з інтегрованою оплатою.

1.3.3 Вибір ORM для роботи з базою даних

SQLAlchemy 2 - це одна з найпопулярніших ORM (Object-Relational Mapper) для Python, що дає можливість працювати з базами даних через об’єктно-орієнтований інтерфейс. У другій версії з’явилась підтримка асинхронності.

Переваги:

- Підтримка асинхронних операцій, що підвищує продуктивність при великій кількості паралельних запитів.
- Гнучкий та потужний інструмент для складних SQL-запитів.
- Сумісність з різними СУБД: SQLite, PostgreSQL, MySQL тощо.
- Величезна спільнота та документація.

Недоліки:

- Має більш складний синтаксис у порівнянні з легковагими ORM.
- Вимагає часу на вивчення.

Таблиця 1.3 - Порівняльний аналіз

Технологія	Опис	Переваги	Недоліки
SQLAlchemy2	Потужний асин-хронний ORM з підтримкою багатьох СУБД	Гнучкість, підтримка асинхронності, широкі можливості для складних запитів	Більш складний, ніж прості бібліотеки
Tortoise ORM	Легкий асинхронний ORM	Простота у використанні, асинхронність	Менша гнуч-кість, менша документація
Peewee	Легкий ORM для Python	Простота, легка вага	Відсутність асинхронної підтримки

Для проекту важлива асинхронність та можливість масштабування з простою заміною СУБД у майбутньому, тому SQLAlchemy 2 є оптимальним вибором.

1.3.4 Вибір драйверу для роботи з SQLite

Aiosqlite - це асинхронний Python-драйвер для роботи з базою даних SQLite, який ідеально підходить для інтеграції з асинхронними фреймворками.

Переваги:

- Підтримка асинхронних викликів, що дозволяє не блокувати виконання програми при роботі з базою.
- Простота використання і сумісність зі стандартним SQL-запитами.
- Легка інтеграція з SQLAlchemy.

Недоліки:

- SQLite - не найкраще рішення для великих проектів з високим навантаженням.
- Обмежені можливості багатокористувацького доступу.

Таблиця 1.4 - Порівняльний аналіз

Технологія	Опис	Переваги	Недоліки
Aiosqlite	Асинхронний драйвер SQLite	Підтримка async, проста інтеграція з SQLAlchemy	Обмежена підтримка складних запитів
sqlite3 (стандартний модуль)	Синхронний драйвер SQLite	Вбудований у Python	Не підтримує асинхронність

Для розробки MVP або невеликого проекту SQLite є швидким і простим варіантом, а Aiosqlite забезпечує асинхронність, необхідну для ефективної роботи з Aiogram.

1.3.5 Валідація даних і налаштувань

Pydantic2 - бібліотека для валідації та серіалізації даних, що використовує типізацію Python.

Переваги:

- Автоматична валідація даних на основі типів.
- Зручна робота з конфігураціями та даними, що приходять з зовнішніх джерел (наприклад, API або користувача).
- Чітке визначення структур даних.

Недоліки:

- Потребує розуміння типів Python і їх особливостей.
- Може ускладнити код, якщо дані дуже різнорідні.

Таблиця 1.5 - Порівняльний аналіз

Технологія	Опис	Переваги	Недоліки
Pydantic 2	Модель валідації та налаштувань з підтримкою типізації	Швидка, проста у використанні, автогенерація моделей	Вимагає певних знань Python типів
Marshmallow	Схема валідації для Python	Гнучкість, сумісність	Складніша у налаштуванні

В проекті важливо надійно перевіряти всі вхідні дані — як від користувачів, так і з налаштувань, тому Pydantic є ідеальним інструментом для цього.

1.3.6 Вибір інструменту для управління міграціями бази даних

Alembic - інструмент для керування версіями схеми бази даних, який працює з SQLAlchemy.

Переваги:

- Автоматизація створення, застосування та відкату змін у структурі бази.
- Інтеграція зі SQLAlchemy.
- Можливість контролювати всі зміни в базі через скрипти.

Недоліки:

- Потребує додаткового налаштування.
- Крива навчання для новачків.

Таблиця 1.6 - Порівняльний аналіз

Технологія	Опис	Переваги	Недоліки
Alembic	Автоматизація міграцій у SQLAlchemy	Інтеграція з SQLAlchemy, зручність	Потрібен час на налаштування
Flyway	Загальний інструмент міграцій	Підтримка різних СУБД	Не інтегрований з Python

Для масштабованих проектів міграції - необхідність, і Alembic - стандартний та перевірений інструмент у Python-екосистемі.

Отже обрані інструменти - Aiogram 3, SQLAlchemy 2, Aiosqlite, Pydantic 2 та Alembic - є сучасними, ефективними та найкраще підходять для розробки асинхронного Telegram-бота з базою даних SQLite та інтегрованою платіжною системою. Вони забезпечують баланс між продуктивністю, гнучкістю та простотою підтримки, що є критично важливим для успішної реалізації проекту.

1.4 Чат-бот у контексті Telegram

Telegram-бот - це сторонній додаток, який функціонує в екосистемі Telegram. Користувачі взаємодіють із ботом, надсилаючи йому повідомлення, команди або inline-запити. Для керування ботами використовується Telegram API [22], що вимагає застосування HTTPS-протоколу.

Функціональні можливості Telegram-ботів включають:

- Надсилання персоналізованих сповіщень і новин.
- Інтеграцію зі сторонніми сервісами, такими як Gmail, Wikipedia, YouTube, GitHub та інші.
- Прийом платежів від користувачів.
- Створення корисних утиліт, наприклад, перекладачів, інструментів для форматування тексту або нагадувань про події.

Існує два основні способи взаємодії з ботом у Telegram: через відкриття чату з ботом або додавання його до групи, де користувачі можуть надсилати

текстові повідомлення чи команди; а також через використання inline-ботів, коли у будь-якому чаті вводиться "@ім'я_бота" у полі введення повідомлення.

Варто зазначити, що ініціатором спілкування завжди є користувач, бот не може починати діалог. Ім’я ботів традиційно закінчується на "bot".

Через обмежене хмарне сховище Telegram для повідомлень, застарілі або великі обсяги даних рекомендується зберігати у локальній базі даних.

Крім того, всі методи Telegram Bot API нечутливі до регістру символів, а передані дані мають бути кодувані у форматі UTF-8.

При аналізі роботи ботів важливо розрізняти процес отримання оновлень та виклик методів API. Оновлення надходять у вигляді об'єкта Update, який містить інформацію про дії користувача щодо бота. Для виконання дій бот надсилає відповідні дані користувачу через виклики API.

Детальний опис функціональних можливостей, методів API та їх обмежень надається у табл. 1.7.

Таблиця 1.7 – Основні поля об’єкту *Update*

Поле	Тип	Опис
update_id	Цілочисельний	Унікальний ідентифікатор оновлення.
message	*Message*	Необов’язкове. Нове повідомлення – текст, фото, відео або голосове повідомлення тощо.
edited_message	*Message*	Необов’язкове. Відредаговане повідомлення.
callback_query	*CallbackQuery*	Необов’язкове. Зворотній запит (результат натискання на кнопку клавіатури).

Одна з корисних можливостей Telegram — це прикріплення клавіатури з кастомними кнопками до будь-якого повідомлення. Це значно покращує користувацький досвід і робить взаємодію з ботом більш зручною, дозволяючи створювати меню, відкривати посилання на зовнішні ресурси або проводити опитування. Використання клавіатури замість відправки додаткових повідомлень є ефективною практикою, що допомагає зменшити кількість повідомлень у чаті. Для цього бот додає клавіатуру до повідомлення через поле "reply_markup".

Таблиця 1.8 – Основні поля об’єкту *InlineKeyboardMarkup*

Поле	**Тип**	**Опис**
inline_keyboard	Масив з масиву *InlineKeyboardButton*	Масив рядів кнопок.
InlineKeyboardButton		
text	Рядок	Текст, що відображається на кнопці.
url	Рядок	Необов’язкове. Посилання.
callback_data	Рядок	Дані, що будуть надіслані разом з зворотнім запитом (*CallbackQuery*).
CallbackQuery		
id	Рядок	Унікальний ідентифікатор запиту.
from	*User*	Чат, з якого прийшов цей запит.

Крім того, Telegram Bot API дозволяє форматувати текст повідомлень. Тип форматування задається через параметр parse_mode при виклику методів надсилання повідомлень. Основні методи надсилання повідомлень представлені в табл. 1.9.

Таблиця 1.9 – Основні методи надсилання повідомлень

Поле	**Тип**	**Опис**
sendMessage		
chat_id	Цілочисельний	Унікальний ідентифікатор чату.
text	Рядок	Текст повідомлення. 0-4096 символів.
parse_mode	Рядок	Необов’язкове. Тип форматування.
reply_markup	Масив *InlineKeyboardMarkup*	Необов’язкове. *Inline*-клавіатура.
sendPhoto, sendDocument		
chat_id	Цілочисельний	Унікальний ідентифікатор чату.
photo, document	*InputFile*	Фото або файл.
caption	Рядок	Необов’язкове. Підпис фото. 0-1024 символів.

Клієнти Telegram підтримують форматування тексту (parse_mode) у форматах HTML (див. рис. 1.4) та Markdown (див. рис. 1.5). Зазначені формати підтримують тільки обмежені набори тегів.

```
<b>bold</b>, <strong>bold</strong>
<i>italic</i>, <em>italic</em>
<u>underline</u>, <ins>underline</ins>
<s>strikethrough</s>, <strike>strikethrough</strike>, <del>strikethrough</del>
<b>bold <i>italic bold <s>italic bold strikethrough</s> <u>underline italic bold</u></i> bold</b>
<a href="http://www.example.com/">inline URL</a>
<a href="tg://user?id=123456789">inline mention of a user</a>
<code>inline fixed-width code</code>
<pre>pre-formatted fixed-width code block</pre>
```

Рисунок 1.4 – Підтримувані *HTML*-теги в *Telegram*

```
*bold \*text*
_italic \*text_
__underline__
~strikethrough~
*bold _italic bold ~italic bold strikethrough~ __underline italic bold___ bold*
[inline URL](http://www.example.com/)
[inline mention of a user](tg://user?id=123456789)
`inline fixed-width code`
```
pre-formatted fixed-width code block
```
```python
pre-formatted fixed-width code block written in the Python programming language
```
```

Рисунок 1.5 – Підтримуваний *Markdown*-синтаксис в *Telegram*

Описаний функціонал та *Telegram Bot API* [12] буде використано для розробки системи керування чат-ботом *Telegram*.

1.5 Постановка задачі дослідження

У сучасному цифровому середовищі Telegram став не лише популярним месенджером, але й ефективною платформою для реалізації бізнес-процесів, зокрема у сфері електронної комерції. Зростаюча популярність Telegram-ботів

зумовлена їхньою доступністю, багатофункціональністю та простотою використання для кінцевого споживача. Боти дозволяють автоматизувати обробку замовлень, прийом платежів, комунікацію з клієнтами та керування цифровими товарами.

Метою роботи є розробка повнофункціонального Telegram-бота для інтернет-магазину цифрових товарів з інтегрованою платіжною системою Telegram-Stars, який забезпечує зручну взаємодію з користувачами, обробку замовлень, адміністрування контенту та безпечне зберігання даних.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

– Проаналізувати можливості Telegram Bot API та платіжної системи Telegram-Stars.

– Створити архітектуру Telegram-бота на основі фреймворку Aiogram 3, дотримуючись принципів асинхронного програмування.

– Реалізувати систему управління даними через SQLAlchemy 2 з використанням бази даних SQLite та драйвера Aiosqlite.

– Імплементувати валідацію та конфігурацію даних за допомогою Pydantic 2.

– Налаштувати міграції бази даних з використанням Alembic для зручного управління змінами в структурі.

– Розробити адміністративну панель для керування товарами, користувачами та замовленнями.

– Створити користувацький інтерфейс бота з можливістю перегляду товарів, додавання до кошика та оформлення замовлення.

– Інтегрувати платіжну систему Telegram-Stars для прийому платежів.

– Забезпечити логування, обробку помилок та безпеку даних.

– Протестувати бота на різних сценаріях взаємодії та задокументувати процес розробки.

З появою в Telegram вбудованої платіжної системи Telegram-Stars, відкрилися нові можливості для створення повністю інтегрованих сервісів без потреби у зовнішніх платіжних шлюзах. Це дозволяє суттєво спростити процес покупки, зменшити технічні витрати на інтеграцію та підвищити зручність для користувачів.

Розробка Telegram-бота для продажу цифрових товарів є актуальною, оскільки дозволяє швидко запускати інтернет-магазини з мінімальними витратами та забезпечує високу мобільність бізнесу. Використання сучасних асинхронних технологій Python (Aiogram 3, SQLAlchemy 2, Aiosqlite, Pydantic 2, Alembic) забезпечує високу продуктивність, масштабованість і гнучкість рішення.

Таким чином, дана розробка відповідає актуальним потребам малого та середнього бізнесу, а також демонструє ефективне застосування сучасних технологій у реальному проєкті.

Висновки до розділу:

У першому розділі було розглянуто призначення, функціональні можливості та особливості використання Telegram-ботів як ефективного інструменту взаємодії між бізнесом і кінцевим користувачем. Проведено порівняльний аналіз популярних месенджерів, що підтримують бот-платформи, за ключовими критеріями, такими як механізми зв’язку, рівень доступу, інтерфейс, протоколи передачі даних і функціональні можливості. На основі цього аналізу було обґрунтовано доцільність вибору саме Telegram як платформи для реалізації програмного продукту завдяки її відкритості, широкому функціоналу, підтримці Polling та Webhook, а також безкоштовному використанню.

Були розглянуті альтернативи та виконано обґрунтований вибір інструментів для реалізації Telegram-бота:

- Мова програмування Python була обрана за простоту синтаксису, асинхронні можливості, багатий набір бібліотек і широку підтримку спільноти.
- Aiogram 3 - як сучасний асинхронний фреймворк для Telegram-ботів.
- SQLAlchemy 2 - як потужний ORM з підтримкою асинхронності.
- Aiosqlite - для взаємодії з SQLite у асинхронному режимі.
- Pydantic 2 - для ефективної валідації даних.
- Alembic - як надійний інструмент для управління міграціями структури бази даних.

Також детально проаналізовано архітектурні особливості ботів, такі як Webhook і Polling, їх переваги й недоліки, а також механізми автентифікації. В результаті було чітко визначено задачі, які необхідно реалізувати у межах проєкту, сформовано технічну базу для практичної реалізації Telegram-бота для інтернет-магазину цифрових товарів.

РОЗДІЛ 2
РОЗРОБКА ТА РЕАЛІЗАЦІЯ ТЕЛЕГРАМ-БОТУ ІНТЕРНЕТ МАГАЗИНУ

2.1 Основні поняття та складові для розробки телеграм-боту

Цей бот використовує технологію вебхука, яка забезпечує високу продуктивність. Хуки реалізуються через aiohttp і використовуються як для підняття бота, так і для обробки платежів.

Поняття вебхуків і полінгу присутнє не лише в контексті Telegram-ботів, а й у світі програмування загалом.

Суть підходу полінгу полягає в тому, що ми самостійно й з певними інтервалами опитуємо сервіси щодо появи нових оновлень. Ось два приклади:

Телеграм-бот через полінг: бот постійно опитує сервери Telegram про нові події. Якщо оновлення є (наприклад, клієнт натиснув кнопку «Мої покупки»), сервер повідомляє боту: «Є подія, користувач з ID 123456 натиснув кнопку». Після цього бот виконує потрібну дію. Якщо оновлень немає, запит повторюється через певний інтервал часу. Це відбувається навіть у періоди бездіяльності користувачів.

Платіжна система через полінг: після виставлення рахунку клієнту ви починаєте регулярно надсилати запити на сервер платіжної системи, щоб дізнатися статус платежу. Як тільки система повертає статус «оплачено», процес завершується.

Принципово інший підхід - це використання вебхуків. Вебхук - це URL-адреса, куди сервіс автоматично надсилає повідомлення про подію.

Телеграм-бот через вебхуки: щойно сервер Telegram фіксує подію (наприклад, користувач натиснув кнопку), він надсилає повідомлення на вказаний вебхук. Бот миттєво обробляє подію без додаткових запитів.

Платіжна система: щойно статус платежу змінився, система сама повідомляє ваш сервер через вебхук. Вам залишається лише обробити дані.

Переваги вебхуків

- Економія ресурсів: мінімізація кількості запитів.
- Миттєва обробка подій: дані надходять одразу після змін.
- Підходить для проєктів з високим навантаженням: зниження навантаження на сервер.

Бот включает в себя три вида оплаты:

- Telegram-Stars - интеграция для осуществления платежей.
- Redsys - интеграция через BotFather.
- Portmone - интеграция напрямую в обход BotFather, обслуживаемая через веб-хуки.

У проекті використовуються такі технології:

– *aiogram* – це асинхронний фреймворк для розробки ботів у Telegram.

– *aiosqlite* - асинхронний драйвер для роботи з SQLite.

– *Loguru 2* - це бібліотека для розширеного ведення журналів.

– *pydantic-settings* - Керування налаштуваннями за допомогою Pydantic.

– *SQLAlchemy* - це бібліотека SQL та ORM для Python.

– *pydantic* – це бібліотека для перевірки даних та керування налаштуваннями.

– *Alembic* – інструмент для управління міграціями баз даних.

– *AIOHTTP* – це веб-сервер для обслуговування веб-хуків у боті.

Структура проекту

```
project
│
├── bot/                    # Код ядра бота
│   ├── admin/              # Логіка адмін-панелі
│   ├── user/               # Користувальницька логіка
│   ├── dao/                # Робота з базою даних
│   ├── config.py           # Налаштування проекту
│   └── main.py             # Основний файл програми
├── data/                   # Зберігання баз даних
├── .env                    # Змінні середовища (токени, налаштування)
└── requirements.txt        # Залежності проекту
```

Рисунок 2.1 – Структура проекту

Опис структури папок бота

У папці бота ми сконцентруємо весь основний код бота. У нього входять:

– admin/ - каталог для коду, пов'язаного з адмін-панеллю бота.

– user/ - каталог, де ми розмістимо логіку користувальницької частини бота, включаючи команди, меню та інші взаємодії.

– DAO/ – модуль для роботи з базою даних. Саме тут будуть зберігатися моделі та методи взаємодії з даними.

– config.py - файл налаштувань проекту, де ми вкажемо основні параметри і шляхи.

– main.py – основний файл, з якого запускається програма.

Додаткові файли та каталоги

– data/ - папка для зберігання бази даних.

– .env – це файл для змінних середовища, таких як токен бота, токен платіжної системи та інші конфіденційні дані.

– requirements.txt – список всіх залежностей проекту для швидкої установки.

Представити взаємозв’язки модулів можна у вигляді діаграми архітектури наступним чином (рис. 2.2)

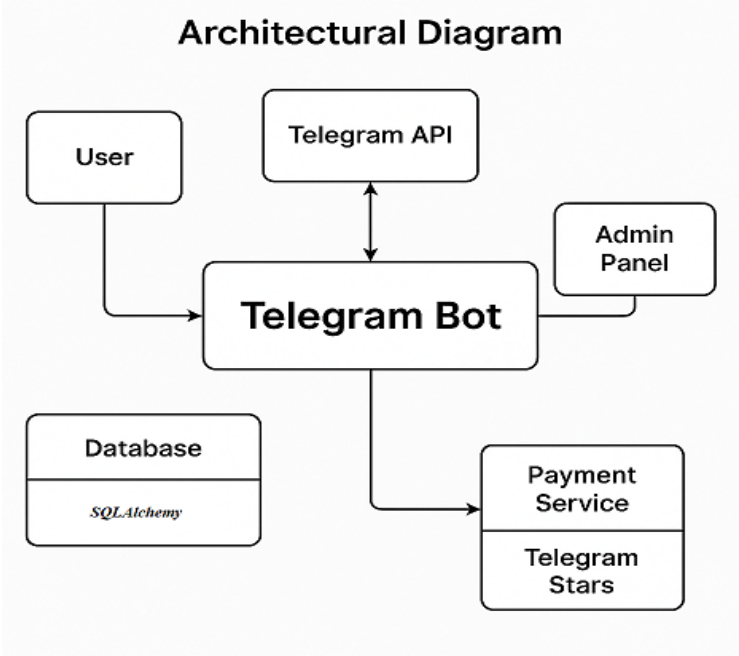

Рисунок 2.2 – Діаграма архітектури

На рис. 2.3 наведено діаграму сценарієв Sequence diagram:

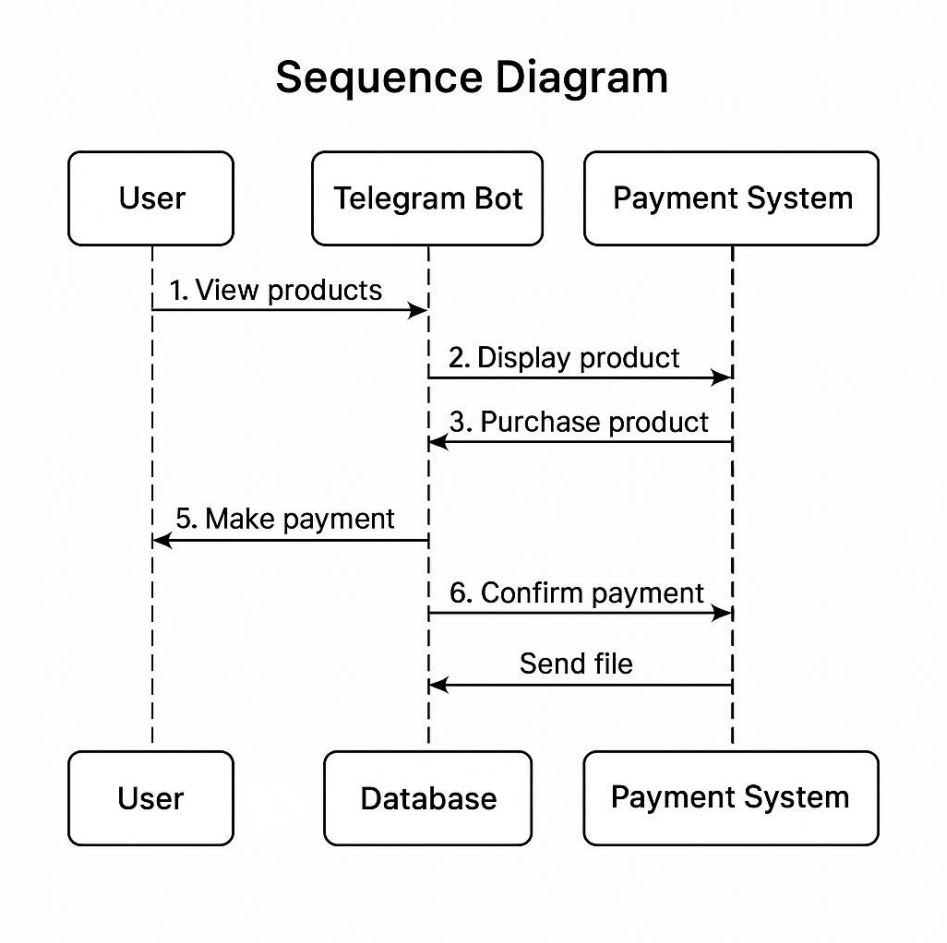

Рисунок 2.3 – Діаграма сценарієв - Sequence diagram

2.2 Реєстрація боту та інтеграція платіжних систем

Значення змінної TOKEN отримуємо під час реєстрації бота. Для цього перейти в Telegram і знайти канал @BotFather, через який відбувається реєстрація ботів (рис. 24). Якщо ваш токен став комусь відомий, вам потрібно ввести команду /revoke для створення нового токена. Оскільки ми будемо працювати з файловою *СУБД SQLite*, Необхідно вказати ім'я файлу БД. Також можна задати версію та ім'я автора проекту.

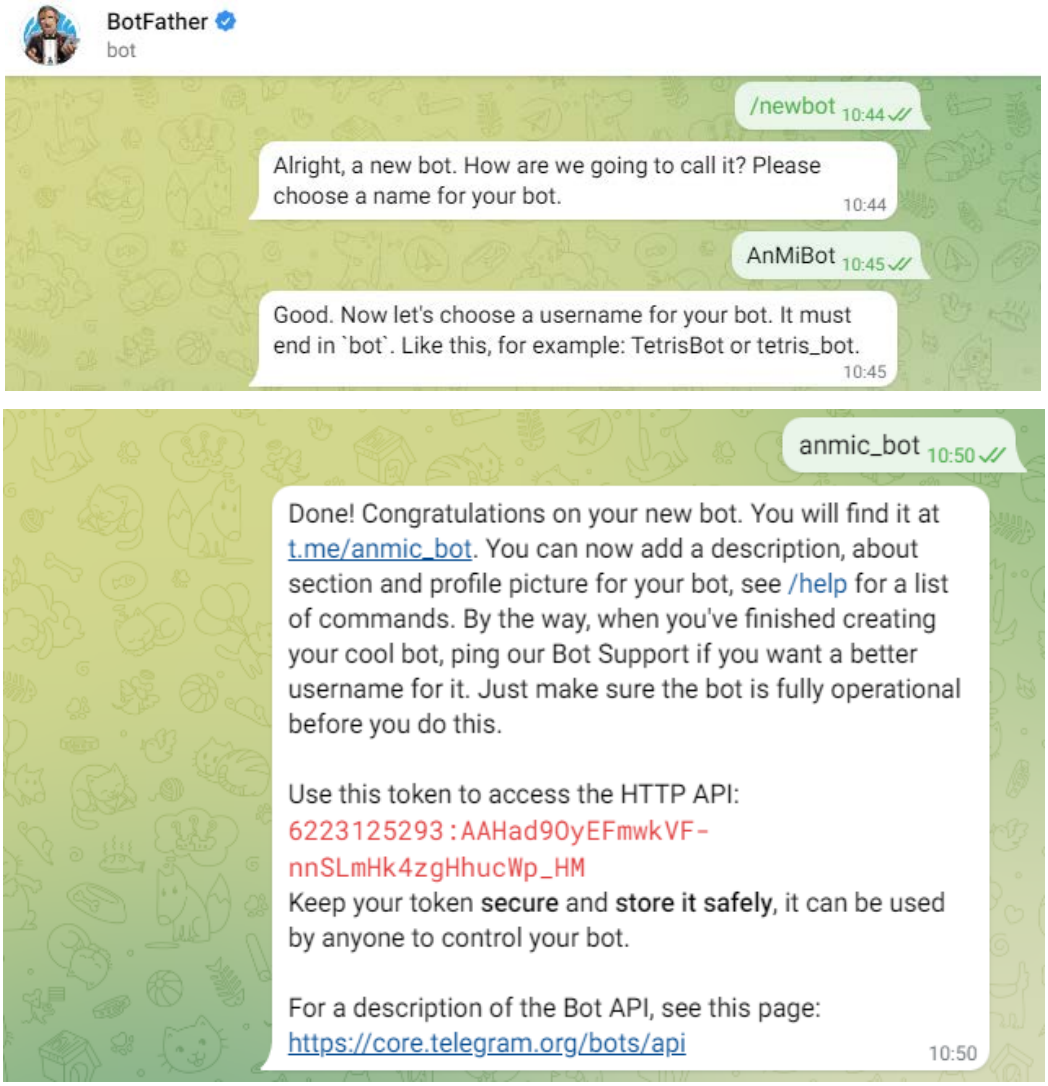

Рисунок 2.4 – Реєстрація бота та отримання Токена

Тепер підключимо платіжні системи за допомогою Payments (рис. 2.5):

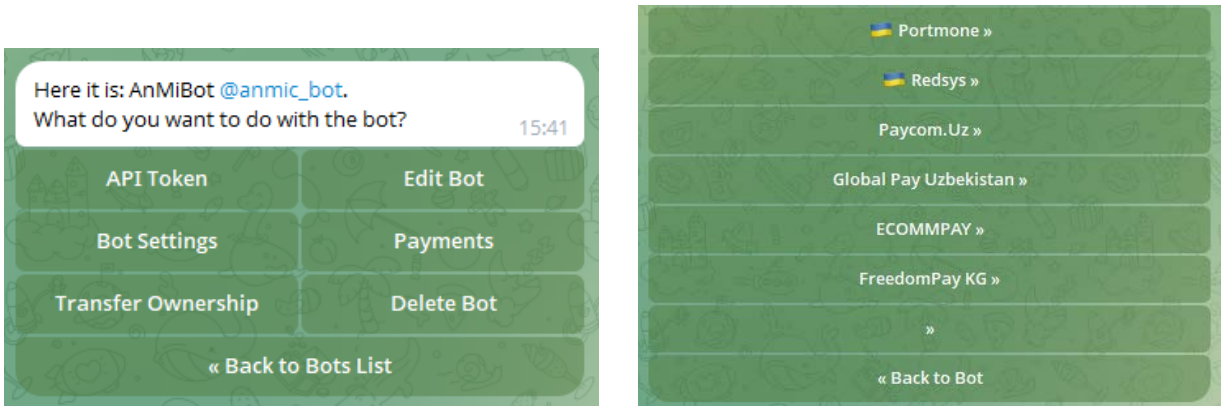

Рисунок 2.5 – Підключення та вибір платіжних систем

З початку обираємо Portmone (рис.2.6):

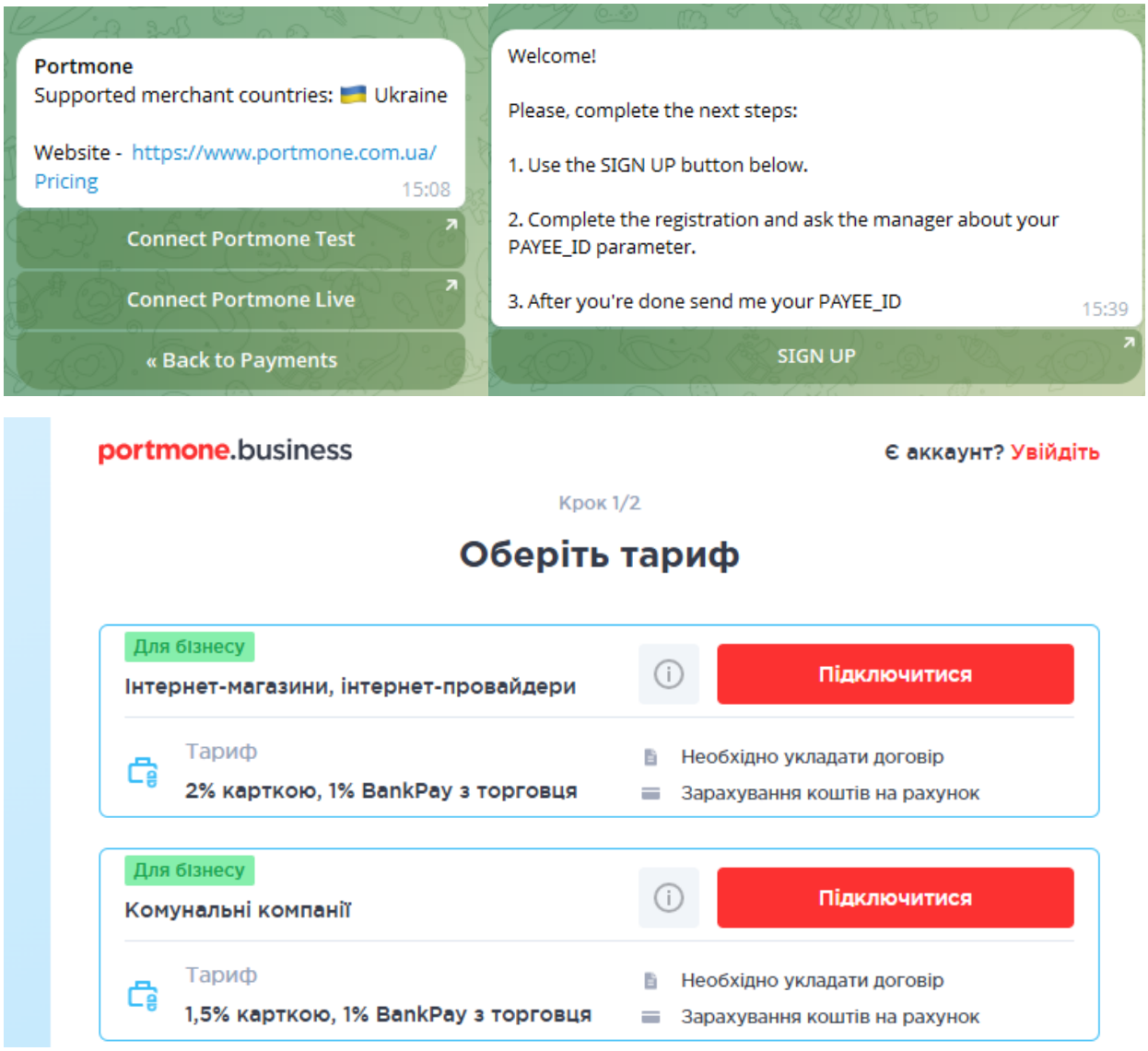

Рисунок 2.6 – Підключення платіжної системи Portmone

А потім Redsys (рис. 2.7):

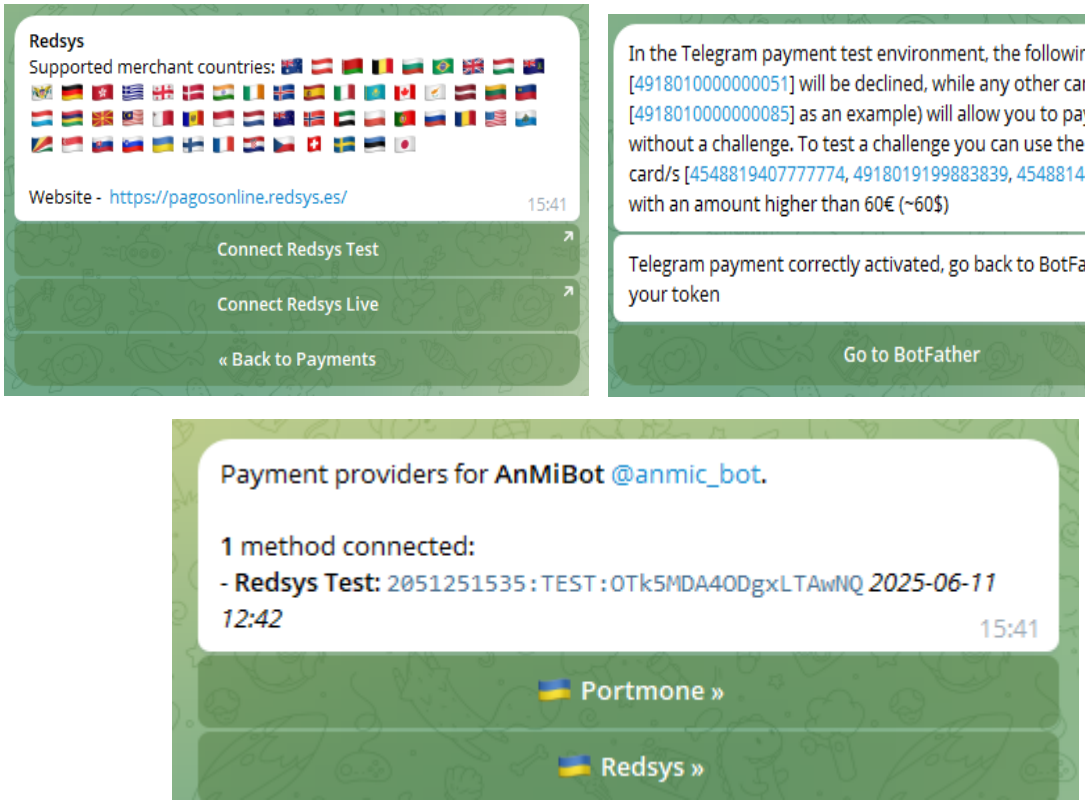

Рисунок 2.7 – Підключення платіжної системи Redsys

2.3 Встановлення та налаштування вебсервера

Так як ми будемо працювати з вебхуками, що потребує запуску вебсервера й забезпечення доступу до нього через доменне ім’я з HTTPS-протоколом. Вебсервер ми будемо піднімати за допомогою бібліотеки Aiohttp. Ця зручна бібліотека широко використовується для обробки вебхуків у Telegram-ботах. У нашому проєкті Aiohttp виконуватиме роль міні-сайту, що містить хуки, які ініціюють різні події в системі.

Для коректної роботи вебхуків необхідне доменне ім’я з HTTPS-протоколом. На етапі розробки ми будемо використовувати NGROK (рис. 2.8).

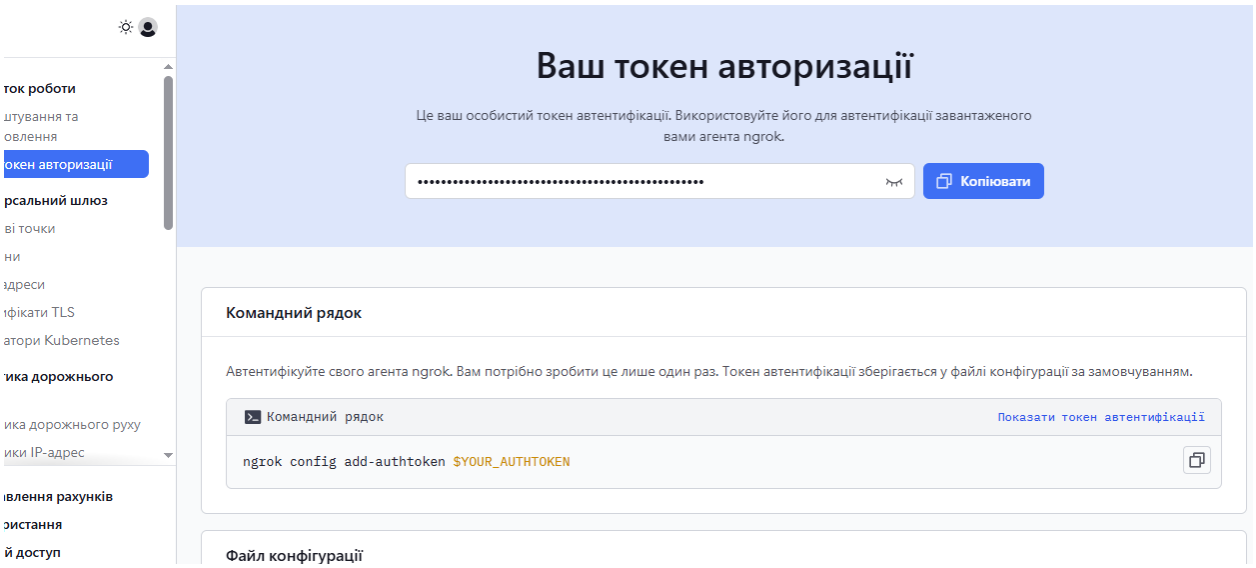

Рисунок 2.8 – Реєстрація та отримання токену NGROK

Запускаємо вебсервер на локальному порту:

```
ngrok http 8000
```

Підключаємо тунель до цього порту та отримуємо посилання з HTTPS-протоколом, яке використовується як тимчасовий домен. Таким чином, наш локальний вебсервер стає доступним для зовнішніх запитів.

Встановлюємо залежності (рис.2.9):

```
pip install -r requirements.txt
```

```
aiogram==3.15.0
aiosqlite==0.20.0
loguru==0.7.2
pydantic-settings==2.7.0
SQLAlchemy==2.0.35
pydantic&gt;=2.4.1,&lt;2.10
alembic==1.14.0
```

Рисунок 2.9 – Файл requirements.txt

Створюємо файл .env в корені проекту й заповніть його такими даними:

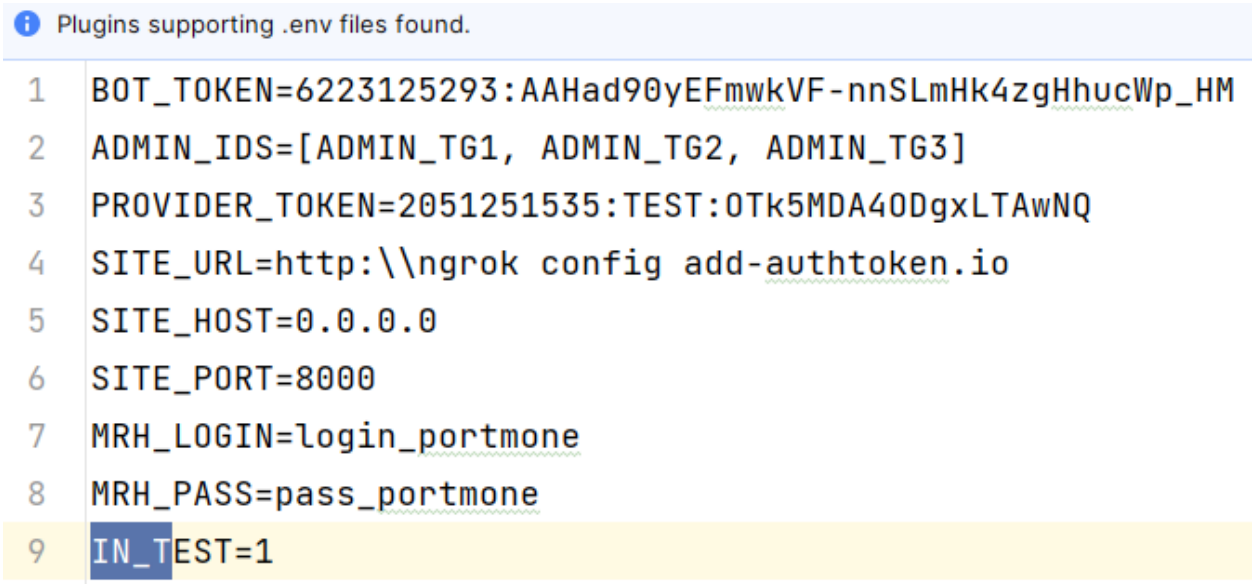

Рисунок 2.10 – Зміст файлу .env

- *BOT_TOKEN:* Токен Telegram-бота. Забезпечує авторизацію бота на платформі Telegram.
- *ADMIN_IDS*: Список ID адміністраторів, які будуть керувати ботом. Вказується у форматі списку: [ID1, ID2, ID3].
- *PROVIDER_TOKEN*: Токен платіжної системи redsys, прив'язаний до бота через @BotFather. Використовується для обробки платежів.
- *SITE_URL:* Повний URL веб-сайту, який буде приймати вебхуки (посилання від NGROK)
- *SITE_HOST:* Хост для вашого локального сервера.
- *SITE_PORT:* Порт, на якому запускається веб-сервер.
- *MRH_LOGIN:* Логін від portmone, що використовується для ідентифікації вашого магазину у системі.
- *MRH_PASS*: Пароль для взаємодії з portmone. Використовується для формування хешу при валідації запитів.
- *IN_TEST:* Флаг тестового режиму. (1 для тестових платежів і в 0 - для робочого режиму).

Тестові данні для оплати:

- Карта: 1111 1111 1111 1026
- Термін дії: 12/26
- CVC-код: 000

2.4 Налаштування конфігурації

Відкриваємо файл bot/config.py і додайте нові параметри до класу Settings:

```python
class Settings(BaseSettings):
    BOT_TOKEN: str
    ADMIN_IDS: List[int]
    PROVIDER_TOKEN: str
    FORMAT_LOG: str = "{time:YYYY-MM-DD at HH:mm:ss} | {level} | {message}"
    LOG_ROTATION: str = "10 MB"
    DB_URL: str = 'sqlite+aiosqlite:///data/db.sqlite3'
    SITE_URL: str
    SITE_HOST: str
    SITE_PORT: int
    MRH_LOGIN: str
    MRH_PASS_1: str
    MRH_PASS_2: str
    IN_TEST: int
    model_config = SettingsConfigDict(
        env_file=os.path.join(os.path.dirname(os.path.abspath(__file__)), "..", ".env")
    )

    @property
    def get_webhook_url(self) -> str:
        return f"{self.SITE_URL}/{self.BOT_TOKEN}"
```

Рисунок 2.11 – Підключення веб-хуків

Властивість get_webhook_url автоматично формує URL для вебхука, що спрощує управління налаштуваннями.

У папці bot створюємо пакет app з файлом init.py. Усередині цієї папки буде така структура:

```
app/
    __init__.py
    app.py     # Обробчик для Aiohttp
    utils.py   # Утіліти для роботи веб-сервера
```

Рисунок 2.12 – Створення структури вебзастосунку

У файлі app.py реалізуємо основний обробник вебхуків. Почнемо з імпортів:

```
from aiohttp import web
from aiogram.types import Update
from loguru import logger
from bot.config import bot, dp, settings
```

Рисунок 2.13 – Реалізація обробника вебхуків

Додаємо обробник:

```
async def handle_webhook(request: web.Request):
    try:
        update = Update(**await request.json())
        await dp.feed_update(bot, update)
        return web.Response(status=200)
    except Exception as e:
        logger.error(f" помилка обробки вебхука {e}")
        return web.Response(status=500)
```

Рисунок 2.14 – Додавання обробника

Запит від Telegram надходить на сервер і перетворюється в об'єкт Update. Диспетчер (dp) обробляє оновлення через метод feed_update. У разі успіху повертається статус 200 (успішно), а при помилці - статус 500.

Метод dp.feed_update у контексті Telegram-ботів виконує ключову задачу - приймає об'єкт оновлення (Update) і передає його зареєстрованим у диспетчері обробникам для виконання відповідних дій.

```
from aiogram.webhook.aiohttp_server import setup_application
from aiohttp import web
from aiogram.types import BotCommand, BotCommandScopeDefault
from loguru import logger
from bot.app.app import handle_webhook, robokassa_result, robokassa_fail, home_page
from bot.config import bot, admins, dp, settings
from bot.dao.database_middleware import DatabaseMiddlewareWithoutCommit, DatabaseMiddlewareWi
from bot.admin.admin import admin_router
from bot.user.user_router import user_router
from bot.user.catalog_router import catalog_router
```

Рисунок 2.15 – Налаштування головного файлу bot/main.py

`from aiogram.webhook.aiohttp_server import setup_application`:

Цей імпорт надає функцію setup_application, яка інтегрує диспетчер (Dispatcher) та бота (Bot) з вебзастосунком Aiohttp. Вона зв’язує сервер Aiohttp з Telegram, дозволяючи обробникам, зареєстрованим у диспетчері, реагувати на вхідні вебхуки. Після виклику цієї функції сервер Aiohttp починає приймати запити, що надсилає Telegram, і передає їх у диспетчер для обробки.

`from aiohttp import web`:

Цей імпорт дає інструменти для роботи з Aiohttp:

- web.Application - створення вебзастосунку.
- web.Request - об'єкт запиту.
- web.Response - об'єкт відповіді.

Aiohttp тут виступає повноцінним вебсервером, який може обробляти як вебхуки від Telegram, так і будь-які інші HTTP-запити. У сукупності ці імпорти пов’язують Telegram-сервер, сервер Aiohttp і бота, реалізованого через Aiogram.

2.5 Основні функції боту

Рядок *await bot.set_webhook(settings.get_webhook_url* реєструє вебхук у Telegram, вказуючи, куди Telegram має надсилати оновлення (наприклад, повідомлення чи події).

```python
# Функція для встановлення команд за замовчуванням для бота
async def set_default_commands():
    """
    Встановлює команди за замовчуванням для бота.
    """
    commands = [BotCommand(command='start', description='Запустити бота')]
    await bot.set_my_commands(commands, BotCommandScopeDefault())
```

```python
# Функції для запуску та зупинки бота
async def on_startup(app):
    """
    Виконується під час запуску застосунку.
    """
    await set_default_commands()
    await bot.set_webhook(settings.get_webhook_url)
    for admin_id in admins:
        try:
            await bot.send_message(admin_id, 'Бота запущено 🥳.')
        except Exception as e:
            logger.error(f"Не вдалося надіслати повідомлення адміну {admin_id}: {e}")
    logger.info("Бота успішно запущено.")
```

Рисунок 2.16 – Встановлення команд за замовчуванням та обробка запуску бота

Після цього Telegram самостійно надсилає POST-запити на сервер - нам не потрібно запускати polling.

```python
async def on_shutdown(app):
    """
    Виконується під час зупинки застосунку.
    """
    for admin_id in admins:
        try:
            await bot.send_message(admin_id, 'Бота зупинено. Чому? 😔')
        except Exception as e:
            logger.error(f"Не вдалося надіслати повідомлення адміну {admin_id}: {e}")
    await bot.delete_webhook(drop_pending_updates=True)
    await bot.session.close()
    logger.error("Бота зупинено!")
```

Рисунок 2.17 - Функція, що виконується при зупинці застосунку

```python
# Реєстрація мідлварів і роутерів
def register_middlewares():
    """
    Реєструє мідлвари для диспетчера.
    """
    dp.update.middleware.register(DatabaseMiddlewareWithoutCommit())
    dp.update.middleware.register(DatabaseMiddlewareWithCommit())


def register_routers():
    """
    Реєструє маршрути для диспетчера.
    """
    dp.include_router(catalog_router)
    dp.include_router(user_router)
    dp.include_router(admin_router)
```

Рисунок 2.18 - Реєстрація мідлварів і роутерів

Реєстрацію мідлварів і роутерів (маршрутів) бота винесено в окремі функції - просто задля зручності підтримки коду.

```python
# Функція для створення застосунку aiohttp
def create_app():
    """
    Створює та налаштовує застосунок aiohttp.
    """
    # Створюємо застосунок
    app = web.Application()

    # Реєстрація обробників маршрутів
    app.router.add_post(f"/{settings.BOT_TOKEN}", handle_webhook)
    app.router.add_post("/robokassa/result/", robokassa_result)
    app.router.add_get("/robokassa/fail/", robokassa_fail)
    app.router.add_get("/", home_page)

    # Налаштування застосунку з диспетчером і ботом
    setup_application(app, dp, bot=bot)

    # Реєстрація функцій запуску та зупинки
    app.on_startup.append(on_startup)
    app.on_shutdown.append(on_shutdown)

    return app
```

Рисунок 2.19 - Функція створення застосунку Aiohttp (вебсервера)

create_app:

- Створює екземпляр застосунку Aiohttp (web.Application).
- Реєструє маршрути, які обробляють запити від Telegram і головну сторінку.
- Налаштовує інтеграцію диспетчера та бота через setup_application.
- Додає функції, що виконуються при запуску (on_startup) і зупинці (on_shutdown).

Тепер залишається лише об'єднати всі функції в єдину структуру та запустити вебсервер. Саме він забезпечить роботу нашого бота. Для цього створюємо головну функцію:

```
# Головна функція
def main():
    """
    Головна функція для запуску застосунку.
    """
    # Реєстрація мідлварів і роутерів
    register_middlewares()
    register_routers()

    # Створюємо застосунок і запускаємо його
    app = create_app()
    web.run_app(app, host=settings.SITE_HOST, port=settings.SITE_PORT)
```

Рисунок 2.20 - Функція створення застосунку Aiohttp (вебсервера)

Завдяки цій оптимізації та додаванню кількох рядків коду бот тепер функціонує через вебхуки, і більше не потребує використання методу polling.

Запуск бота здійснюється звичною командою з кореня проєкту:

```
python -m bot.main
```

Переконаємось у працездатності боту.

2.6 Тестування розробленого боту

Для того, щоб перевірити нашого бота заходимо у Telegram і задаємо пошук: @anmic_bot. У результати знаходимо нашого бота та обираємо його запуск (start) (рис.2.21).

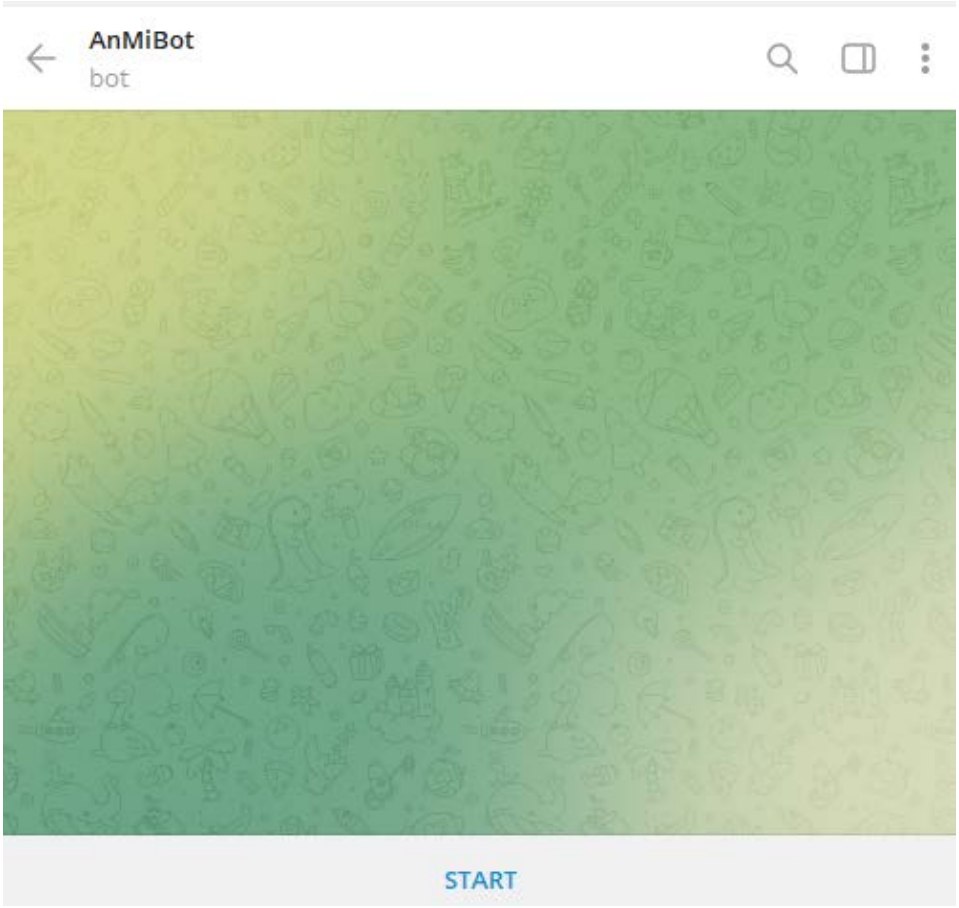

Рисунок 2.21 – Запуск бота

У результаті запуску отримує головне вікно з привітанням та головними модулями (клавіатурами, рис. 2.22):

Рисунок 2.22 – Головне меню бота

Так як ми зайшли перший раз, то покупок у нас відповідно не може бути, після натискання кнопки Мої покупки отримаємо(рис. 2.23):

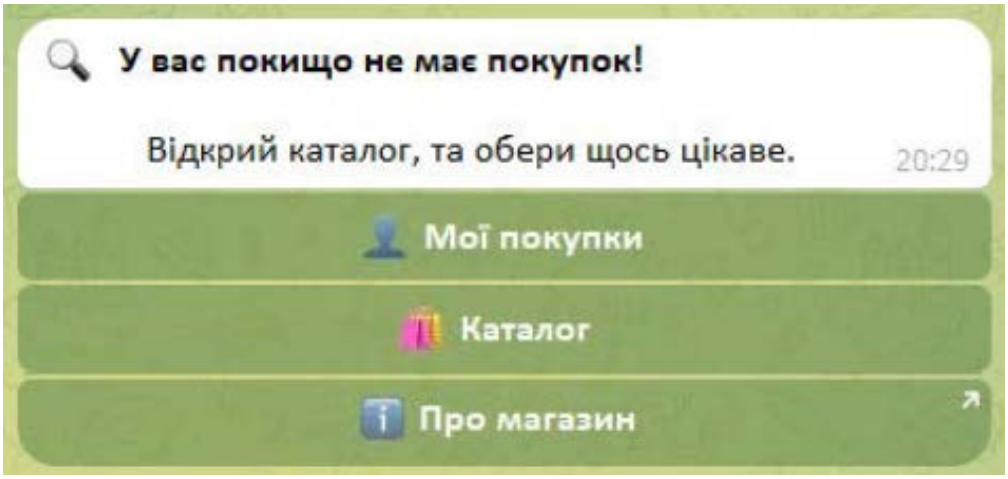

Рисунок 2.23 – Перевірка покупок

У разі натискання кнопки Про магазин, у нас виводиться наступна реклама або ознайомча інформація (Рис. 2.24):

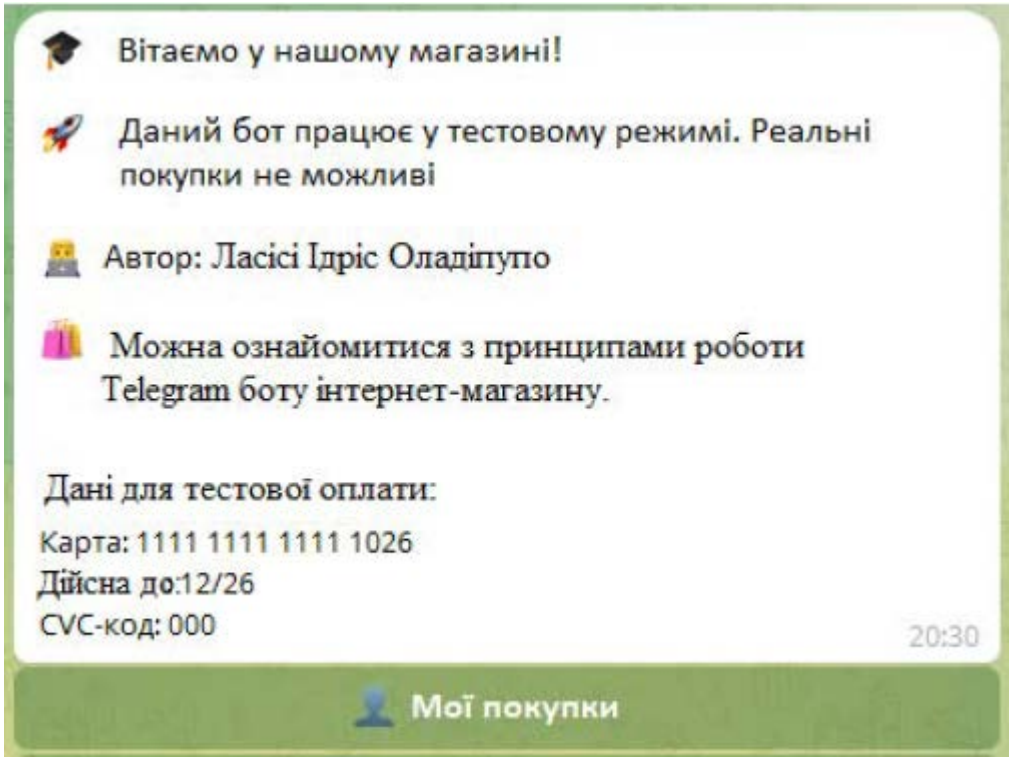

Рисунок 2.23 – Інформація про магазин

У результаті натискання кнопки каталог ми побачимо наступні блоки, а саме перелік наших товарів (рис. 2.24):

Рисунок 2.24 – Вигляд меню Каталог

Обираємо пункт меню Методички, на даний час там у нас один товар, у результаті його вибору ми отримаємо наступне вікно (Рис. 2.25):

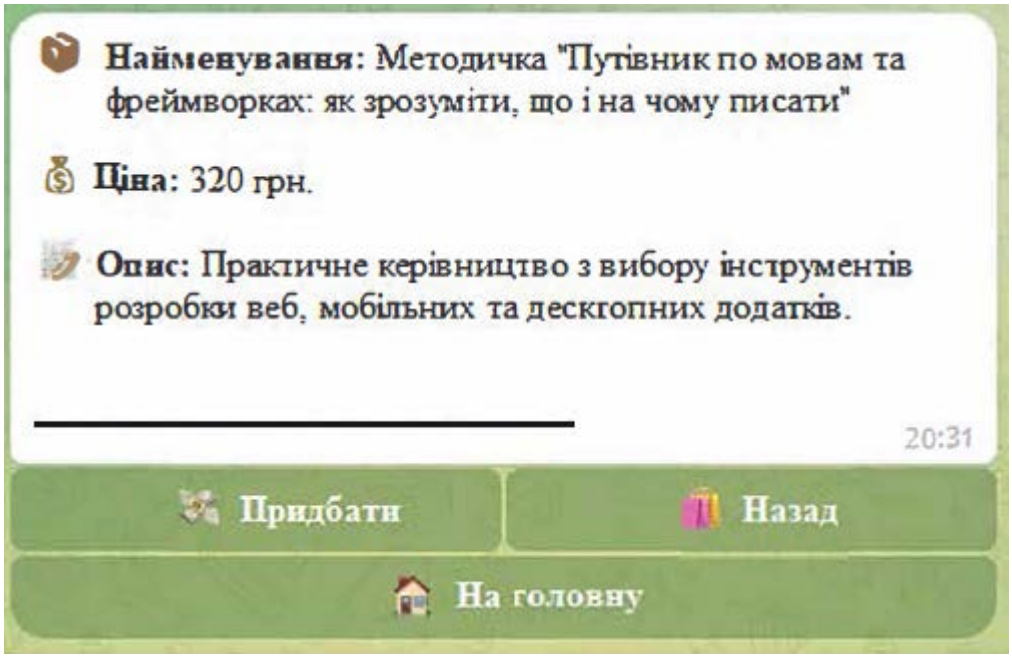

Рисунок 2.25 – Вибір товару

При виборі товару ми побачимо у новому вікні: найменування товару, його ціну та опис. При натискання кнопки Придбати побачимо наступне вікно (рис.2.26):

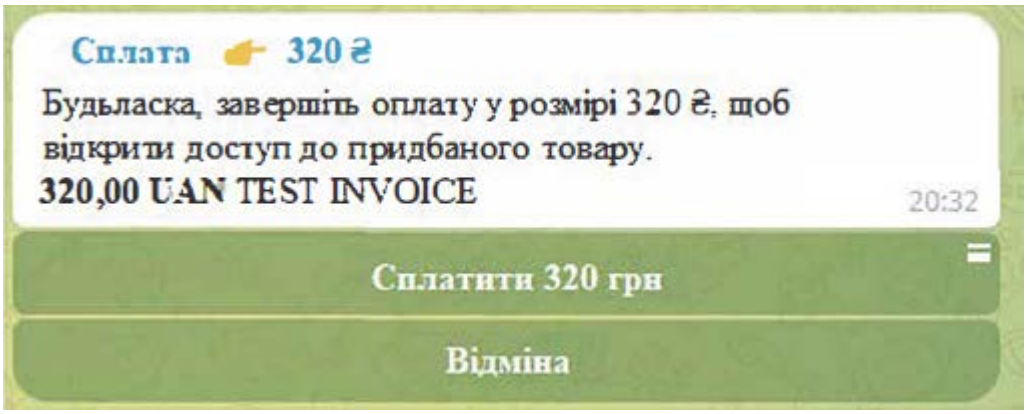

Рисунок 2.26 – Оформлення покупки

Після цього натиснувши Сплатити, нас переведе на сервіс оплати і ми побачимо сформоване повідомлення оплати (рис.2.27)

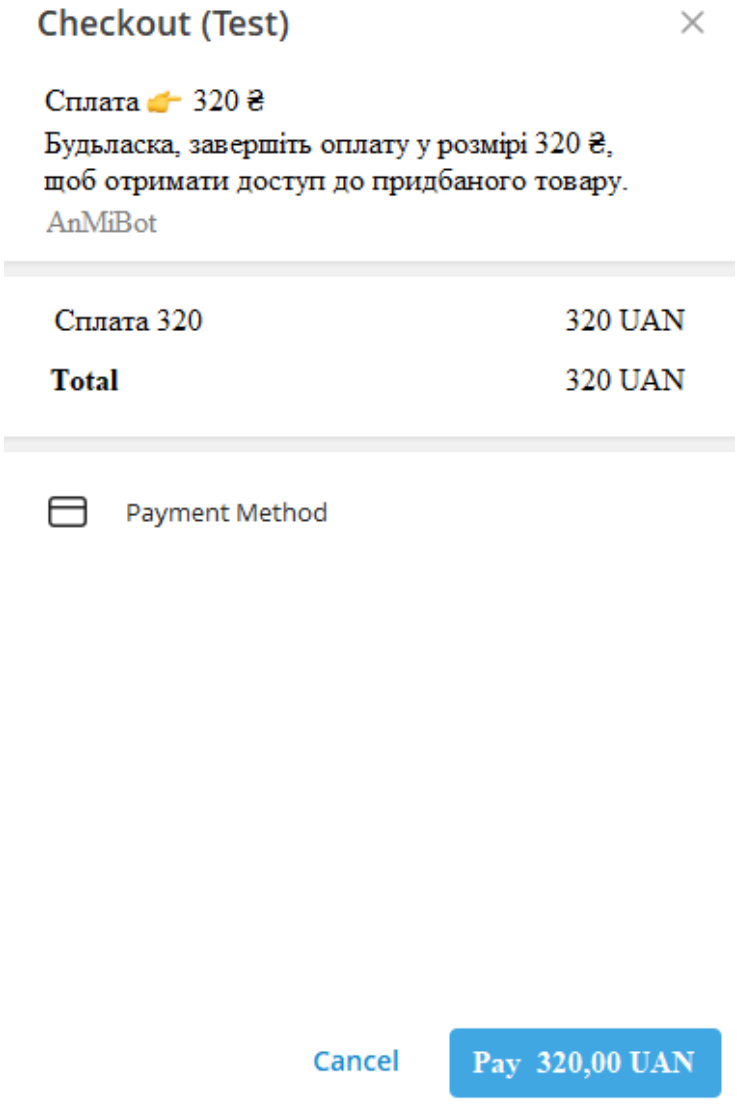

Рисунок 2.27 – Онлайн-рахунок

Натиснувши кнопку Pay 320,00 UAN нас повинно перевести на шлю оплати, але починаючи з 1 червня 2025року для доступу тестових платежів та отримання платіжних даних необхідна повна реєстрація ФОП та наявність розрахункового рахунку, тому продемонструвати оплату в рамках роботи не можливо (отримаємо повідомлення з помилкою оплати)

2.7 Підключаємо оплату через Telegram Stars (зірки)

Наш поточний функціонал вебхука повністю покриває потреби бота для коректної обробки внутрішніх платежів. Це включає інтеграцію з Redsys (реалізованою через BotFather) та Telegram Stars - платіжною системою, вбудованою в Telegram за замовчуванням.

Але починаючи з 1 червня 2025року для доступу тестових платежів та отримання платіжних даних необхідна повна реєстрація ФОП та наявність розрахункового рахунку, тому продемонструвати оплату в рамках роботи не можливо.

Щоб спростити інтеграцію, ми почнемо з виділення логіки обробки успішної оплати в окрему функцію. Нагадаю, що зараз ця логіка безпосередньо прив’язана до обробника успішного платежу в Telegram:

```
@catalog_router.message(F.content_type == ContentType.SUCCESSFUL_PAYMENT)
```

Для цього створимо у папці bot/user файл utils.py. Код даного модуля наведено у Додатку А.

Рядок

```
if payment_type == 'stars':
        await bot.refund_star_payment(user_id=user_tg_id, telegram_payment_charge_id=payment_id)
```

нам потрібен для того, щоб після оплати зірками, якщо був обраний такий формат оплати, зірки поверталися на рахунок користувача. Для тестових платежів це дуже зручно.

Сам метод досить простий. У нього потрібно передати телеграм-айді користувача і telegram_payment_charge_id.

Що стосується системи зірок, я встановив для всіх товарів ціну у 10 зірок при виборі оплати в цій системі. Це зроблено для економії часу, оскільки проєкт у вигляді демо.

Оскільки у нас з'являються нові варіанти оплати — потрібно адаптувати клавіатури.

По-перше, оновимо клавіатуру, яка з'являється при відкритті товару в нашому каталозі. Тепер вона виглядатиме так:

```python
def product_kb(product_id, price, stars_price) -> InlineKeyboardMarkup:
    kb = InlineKeyboardBuilder()
    kb.button(text="💳 Сплатити через ЮKassa", callback_data=f"buy_yukassa_{product_id}_{price}")
    kb.button(text="💳 Сплатити через Robokassa", callback_data=f"buy_robocassa_{product_id}_{price}")
    kb.button(text="⭐ Сплатити зірками", callback_data=f"buy_stars_{product_id}_{stars_price}")
    kb.button(text="◀️ Назад", callback_data="catalog")
    kb.button(text="🏠 На головну", callback_data="home")
    kb.adjust(2)
    return kb.as_markup()
```

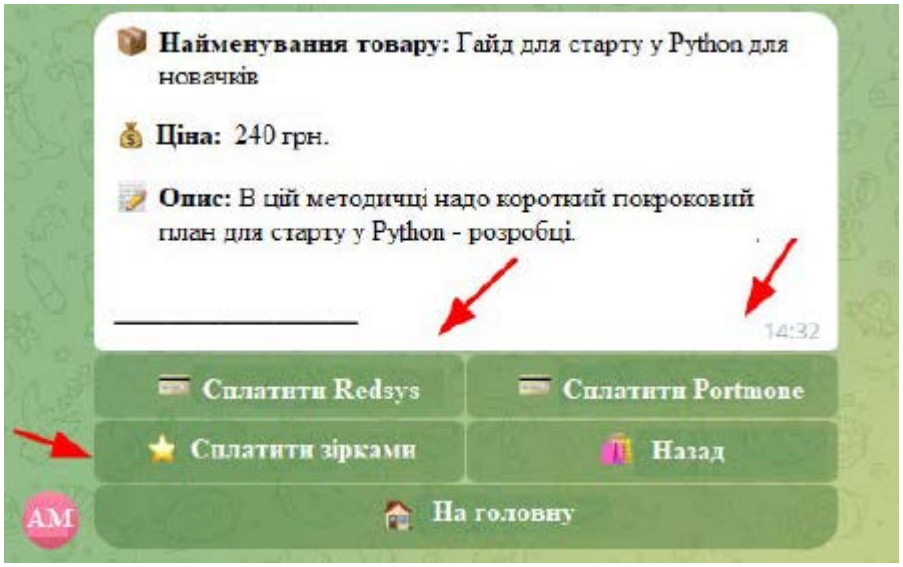

Рисунок 2.28 – Клавіатури оплати

Потім, щоб уникнути плутанини і полегшити читання коду, я створив окремі клавіатури для кожного типу оплати на умовній сторінці оплати. Ось що ми отримали:

```python
def get_product_buy_youkassa(price) -> InlineKeyboardMarkup:
    return InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(text=f'Сплатити {price}₽', pay=True)],
        [InlineKeyboardButton(text='Скасувати', callback_data='home')]
    ])


def get_product_buy_robocassa(price: int, payment_link: str) -> InlineKeyboardMarkup:
    return InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(
            text=f'Сплатити {price}₽',
            web_app=WebAppInfo(url=payment_link)
        )],
        [InlineKeyboardButton(text='Скасувати', callback_data='home')]
    ])


def get_product_buy_stars(price) -> InlineKeyboardMarkup:
    return InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(text=f"Сплатити {price} ⭐", pay=True)],
        [InlineKeyboardButton(text='Скасувати', callback_data='home')]
    ])
```

Рисунок 2.29 – Сторінка оплати

Клавіатура для оплати Telegram Stars не відрізняється по формату від клавіатури для оплати через Redsys. Це не випадковість.

Всі внутрішні платежі Telegram, включаючи Telegram Stars і методи, підключені через BotFather, обробляються однаковою логікою. Такий уніфікований підхід значно спрощує інтеграцію оплати зірками, роблячи процес більш зручним і зрозумілим.

Тепер залишилось внести правки у файл з каталогом (продажами).

Перші зміни починаються в обробнику callback-даних, які починаються на buy_. У новій версії обробник виглядає так:

```python
@catalog_router.callback_query(F.data.startswith('buy_'))
async def process_about(call: CallbackQuery, session_without_commit: AsyncSession):
    user_info = await UserDAO.find_one_or_none(
        session=session_without_commit,
        filters=TelegramIDModel(telegram_id=call.from_user.id)
    )
    _, payment_type, product_id, price = call.data.split('_')

    if payment_type == 'yukassa':
        await send_yukassa_invoice(call, user_info, product_id, price)
    elif payment_type == 'stars':
        await send_stars_invoice(call, user_info, product_id, 10)
    elif payment_type == 'robocassa':
        await send_robocassa_invoice(call, user_info, product_id, price, session_without_commit)

    await call.message.delete()
```

Рисунок 2.30 - Файл bot/user/catalog_router.py.

Процес виставлення рахунку зі зірками майже повністю повторює процес оплати через Redsys.

```
async def send_stars_invoice(call, user_info, product_id, stars_price):
    await bot.send_invoice(
        chat_id=call.from_user.id,
        title=f'Оплата 👉 {stars_price} ⭐',
        description=(
            f'Будь ласка, завершіть оплату у розмірі {stars_price} зірок, '
            f'щоб отримати доступ до вибраного товару.'
        ),
        payload=f"stars_{user_info.id}_{product_id}",
        provider_token="",
        currency='XTR',
        prices=[LabeledPrice(
            label=f'Оплата {stars_price} ⭐',
            amount=int(stars_price)
        )],
        reply_markup=get_product_buy_stars(stars_price)
    )
```

Рисунок 2.31 – Процес оплати зірками

Таким чином, тепер в залежності від обраного користувачем способу оплати ми застосовуємо різний алгоритм для виставлення рахунку (інвойса).

Однією з головних відмінностей є те, що для роботи системи оплати зірками достатньо передати у параметрі provider_token порожній рядок. Це означає, що нам не потрібно отримувати окремий токен для оплати цим способом.

І найголовніше - це зміна валюти, у якій виставляються рахунки. Тепер вона позначається як «XTR», на відміну від звичного «UA».

Оплата через Redsys, так і оплата зірками відносяться до внутрішніх платежів Telegram. Це означає, що для їх обробки достатньо скористатися стандартними методами Aiogram 3. Такий підхід дозволяє спочатку перевірити можливість проведення платежу, а потім успішно обробити його.

Завдяки цьому ми можемо написати універсальний код, який однаково ефективно працює як з оплатою через Redsys у, так і з оплатою зірками.

```python
@catalog_router.pre_checkout_query(lambda query: True)
async def pre_checkout_query(pre_checkout_q: PreCheckoutQuery):
    await bot.answer_pre_checkout_query(pre_checkout_q.id, ok=True)


@catalog_router.message(F.content_type == ContentType.SUCCESSFUL_PAYMENT)
async def successful_payment(message: Message, session_with_commit: AsyncSession):
    payment_info = message.successful_payment
    payment_type, user_id, product_id = payment_info.invoice_payload.split('_')

    if payment_type == 'stars':
        price = payment_info.total_amount
        currency = '⭐'
    else:
        price = payment_info.total_amount / 100
        currency = '₽'

    payment_data = {
        'user_id': int(user_id),
        'payment_id': payment_info.telegram_payment_charge_id,
        'price': price,
        'product_id': int(product_id),
        'payment_type': payment_type
    }

    await successful_payment_logic(session=session_with_commit,
                                   payment_data=payment_data, currency=currency,
                                   user_tg_id=message.from_user.id, bot=bot)
```

Рисунок 2.32 – Об’єднання інтегрованих платежів

У блоці обробки успішного платежу я виконав невелику адаптацію під оновлений обробник successful_payment_logic.

Перезапускаємо бота і перевіряємо оплату зірками.

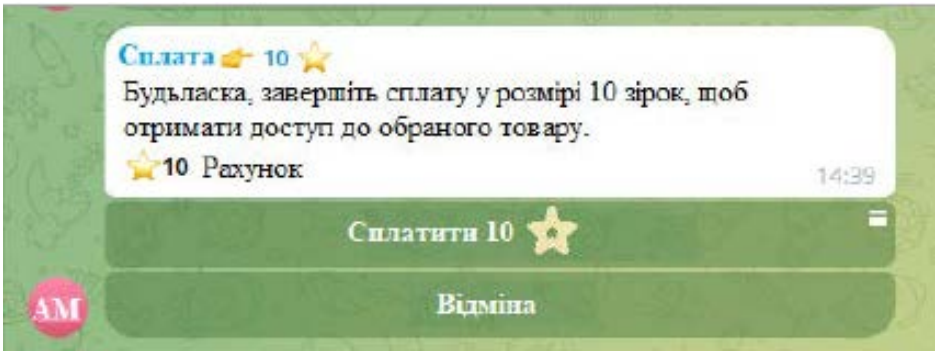

Рисунок 2.33 - Виставляємо рахунок

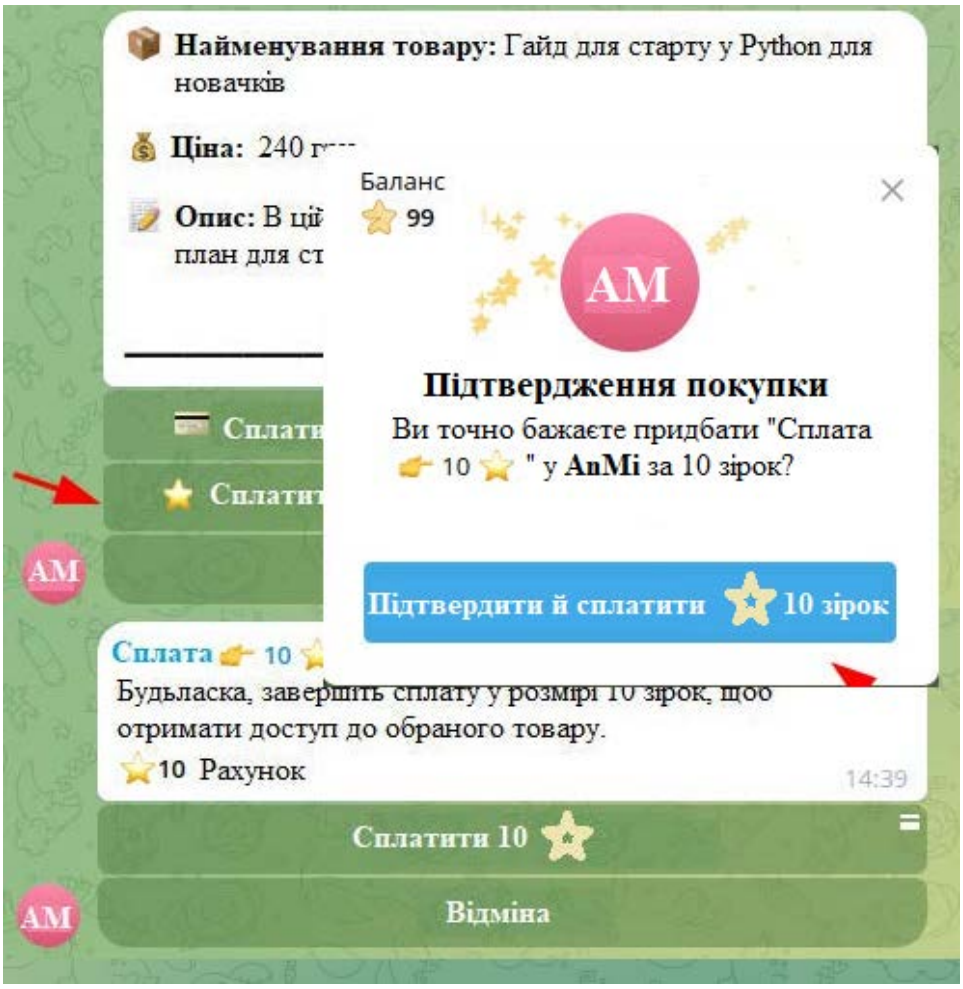

Рисунок 2.34 - Відкривається вікно оплати

Після оплати бот повідомляє, що суму у розмірі 10-ти зірок повернуто:

Рисунок 2.35 – Повернення тестового платежу

Висновки до розділу:

У другому розділі було здійснено повний цикл розробки Telegram-боту для інтернет-магазину цифрових товарів з інтеграцією платіжної системи Telegram Stars. В рамках розробки проведено аналіз основних понять та технологій, необхідних для створення подібного рішення, зокрема

асинхронного фреймворку Aiogram 3, ORM-бібліотеки SQLAlchemy 2, системи валідації Pydantic 2, а також допоміжних інструментів для конфігурації, логування та міграцій бази даних.

Також виконано реєстрацію Telegram-бота за допомогою сервісу BotFather, налаштовано обробку webhook-запитів та інтегровано платіжну систему Telegram Stars для здійснення транзакцій безпосередньо в межах месенджера.

Реалізовано асинхронний вебсервер на основі бібліотеки Aiohttp, що забезпечує стабільну взаємодію з Telegram API через webhook-механізм. Було також налаштовано маршрути та обробники запитів до боту. Для безпечного зберігання конфіденційної інформації використано .env-файл, а самі налаштування реалізовано через модуль Pydantic Settings, що забезпечує централізоване керування параметрами проєкту.

Реалізовано основні функції бота: перегляд каталогу цифрових товарів, оформлення замовлення, інтеграцію з платіжною системою та автоматичну видачу файлів після оплати. Також передбачено адміністративний функціонал для керування асортиментом товарів (додавання, редагування, видалення).

Виконано тестування функціональних можливостей Telegram-бота. Проведено перевірку коректності обробки запитів, дій користувачів, статусів замовлень, а також логіки оплати й видачі товарів.

Реалізовано безпосереднє підключення Telegram Stars як платіжної системи, налаштовано обробку callback-повідомлень після здійснення оплати, а також реалізовано автоматичне надсилання цифрового файлу після підтвердження успішної транзакції.

Таким чином, у результаті розробки було створено повноцінний Telegram-бот, який забезпечує повний цикл взаємодії користувача з інтернет-магазином цифрових товарів - від перегляду до покупки й отримання продукту.

ВИСНОВКИ

У першому розділі було розглянуто призначення, функціональні можливості та особливості використання Telegram-ботів як ефективного інструменту взаємодії між бізнесом і кінцевим користувачем. Проведено порівняльний аналіз популярних месенджерів, що підтримують бот-платформи, за ключовими критеріями, такими як механізми зв'язку, рівень доступу, інтерфейс, протоколи передачі даних і функціональні можливості. На основі цього аналізу було обґрунтовано доцільність вибору саме Telegram як платформи для реалізації програмного продукту завдяки її відкритості, широкому функціоналу, підтримці Polling та Webhook, а також безкоштовному використанню.

Були розглянуті альтернативи та виконано обґрунтований вибір інструментів для реалізації Telegram-бота:

– Мова програмування Python була обрана за простоту синтаксису, асинхронні можливості, багатий набір бібліотек і широку підтримку спільноти.

– Aiogram 3 - як сучасний асинхронний фреймворк для Telegram-ботів.

– SQLAlchemy 2 - як потужний ORM з підтримкою асинхронності.

– Aiosqlite - для взаємодії з SQLite у асинхронному режимі.

– Pydantic 2 - для ефективної валідації даних.

– Alembic - як надійний інструмент для управління міграціями структури бази даних.

Також детально проаналізовано архітектурні особливості ботів, такі як Webhook і Polling, їх переваги й недоліки, а також механізми автентифікації. В результаті було чітко визначено задачі, які необхідно реалізувати у межах проєкту, сформовано технічну базу для практичної реалізації Telegram-бота для інтернет-магазину цифрових товарів.

Розробка Telegram-бота для продажу цифрових товарів є актуальною, оскільки дозволяє швидко запускати інтернет-магазини з мінімальними

витратами та забезпечує високу мобільність бізнесу. Використання сучасних асинхронних технологій Python (Aiogram 3, SQLAlchemy 2, Aiosqlite, Pydantic 2, Alembic) забезпечує високу продуктивність, масштабованість і гнучкість рішення.

Таким чином, дана розробка відповідає актуальним потребам малого та середнього бізнесу, а також демонструє ефективне застосування сучасних технологій у реальному проєкті.

У другому розділі було здійснено повний цикл розробки Telegram-боту для інтернет-магазину цифрових товарів з інтеграцією платіжної системи Telegram Stars. В рамках розробки проведено аналіз основних понять та технологій, необхідних для створення подібного рішення, зокрема асинхронного фреймворку Aiogram 3, ORM-бібліотеки SQLAlchemy 2, системи валідації Pydantic 2, а також допоміжних інструментів для конфігурації, логування та міграцій бази даних.

Також виконано реєстрацію Telegram-бота за допомогою сервісу BotFather, налаштовано обробку webhook-запитів та інтегровано платіжну систему Telegram Stars для здійснення транзакцій безпосередньо в межах месенджера.

Реалізовано асинхронний вебсервер на основі бібліотеки Aiohttp, що забезпечує стабільну взаємодію з Telegram API через webhook-механізм. Було також налаштовано маршрути та обробники запитів до боту. Для безпечного зберігання конфіденційної інформації використано .env-файл, а самі налаштування реалізовано через модуль Pydantic Settings, що забезпечує централізоване керування параметрами проєкту.

Реалізовано основні функції бота: перегляд каталогу цифрових товарів, оформлення замовлення, інтеграцію з платіжною системою та автоматичну видачу файлів після оплати. Також передбачено адміністративний функціонал для керування асортиментом товарів (додавання, редагування, видалення).

Виконано тестування функціональних можливостей Telegram-бота. Проведено перевірку коректності обробки запитів, дій користувачів, статусів замовлень, а також логіки оплати й видачі товарів.

Реалізовано безпосереднє підключення Telegram Stars як платіжної системи, налаштовано обробку callback-повідомлень після здійснення оплати, а також реалізовано автоматичне надсилання цифрового файлу після підтвердження успішної транзакції.

Таким чином, у результаті розробки було створено повноцінний Telegram-бот, який забезпечує повний цикл взаємодії користувача з інтернет-магазином цифрових товарів - від перегляду до покупки й отримання продукту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Васильєв О. М. Програмування в PYTHON. Теорія і практика. Навчальний посібник. Ліра-К, 2023.

2. Васильєв О. М. Програмування мовою Python. Навчальна книга - Богдан, 2019.

3. Eric Matthes. Python Crash Course: A Hands-On, Project-Based Introduction to Programming, 1st Edition. 2015.

4. Mark Lutz, David Ascher. Head First Python: A Brain-Friendly Guide. Second edition. O'Reilly Media, 2004.

5. Al Sweigart. Automate the Boring Stuff with Python: Practical Programming for Total Beginners, 1st edition. No Starch Press, 2015.

6. Michael Dawson. Python Programming for the Absolute Beginner, Second Edition. O'Reilly Media, 2015.

7. Zed Shaw's. Learn Python 3 the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code. Addison-Wesley Professional, 2017.

8. Dan Bader. Python Tricks: A Buffet of Awesome Python Features, 1st edition. Dan Bader, 2017.

9. David Beazley, Brian K. Jones. Python Cookbook: Recipes for Mastering Python 3, 3rd edition. O'Reilly Media, 2013.

10. Bill Lubanovic. Introducing Python: Modern Computing in Simple Packages, 2nd edition. O'Reilly Media, 2019.

11. Luciano Ramalho. Fluent Python. O'Reilly Media, Inc, 2015.

12. Пол Беррі. Head First. Python, 2021.

13. Ерік Маттес. Пришвидшений курс Python. Практичний, проєктно-орієнтований вступ до програмування. Видавництво Старого Лева, 2021.

14. Керол Вордерман. Computer Coding. Python Projects for Kids. A Step-by-Step Visual Guide. DK (Dorling Kindersley), 2017.

15. Повне керівництво по SQLAlchemy. url: https://pythonru.com/biblioteki/ vvedenie-v-sqlalchemy (дата звернення - 20.05.23).

16. Документація telegram bot api. url: https://core.telegram.org/bots/api (дата зверення - 15.05.23).

17. Повна документація з Python 3.11. url: https://docs.python.org/uk/3/ (дата звернення - 20.04.23).

18. Моркун Н. В., Завсегдашня І. В. Методичні вказівки до виконання кваліфікаційної роботи бакалавру для студентів спеціальності 122 “Комп’ютерні науки”. Кривий Ріг : Видавничий центр КНУ, 2021. 50 с.

19. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

20. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

21. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація).

ДОДАТОК А

Виділення логіки обробки успішної оплати в окрему функцію utils.py.

```
from aiogram import Bot
from loguru import logger
from sqlalchemy.ext.asyncio import AsyncSession
from bot.config import settings
from bot.dao.dao import PurchaseDao, ProductDao
from bot.user.kbs import main_user_kb
from bot.user.schemas import PaymentData
async def successful_payment_logic(session: AsyncSession, payment_data, currency, user_tg_id, bot: Bot):
    product_id = int(payment_data.get("product_id"))
    price = payment_data.get("price")
    payment_type = payment_data.get("payment_type")
    payment_id = payment_data.get("payment_id")
    user_id = payment_data.get("user_id")
    await PurchaseDao.add(session=session, values=PaymentData(**payment_data))
    product_data = await ProductDao.find_one_or_none_by_id(session=session, data_id=product_id)
    # Надсилання сповіщень адміністраторам
    for admin_id in settings.ADMIN_IDS:
      try:
        await bot.send_message(
          chat_id=admin_id,
          text=(
            f"💲 Користувач з ID {user_id} придбав товар <b>{product_data.name}</b> (ID: {product_id}) "
            f"за <b>{price} {currency}</b>."
          )
        )
      except Exception as e:
        logger.error(f"Помилка під час надсилання сповіщення адміністраторам: {e}")
    # Надсилання інформації користувачу
    file_text = "📦 <b>Товар містить файл:</b>" if product_data.file_id else "📄 <b>Товар не містить файлів:</b>"
    product_text = (
      f"🎉 <b>Дякуємо за покупку!</b>\n\n"
      f"🛒 <b>Інформація про ваш товар:</b>\n"
      f"━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\n"
      f"🔷 <b>Назва:</b> <b>{product_data.name}</b>\n"
      f"🔷 <b>Опис:</b>\n<i>{product_data.description}</i>\n"
      f"🔷 <b>Ціна:</b> <b>{price} {currency}</b>\n"
```

```
            f"🔷 <b>Закритий опис:</b>\n<i>{product_data.hidden_content}</i>\n"
            f"━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\n"
            f"{file_text}\n\n"
            f"ℹ️ <b>Інформацію про всі ваші покупки ви можете знайти в особистому профілі.</b>"
        )
        if product_data.file_id:
            await bot.send_document(
                document=product_data.file_id,
                chat_id=user_tg_id,
                caption=product_text,
                reply_markup=main_user_kb(user_tg_id)
            )
        else:
            await bot.send_message(
                chat_id=user_tg_id,
                text=product_text,
                reply_markup=main_user_kb(user_tg_id)
            )
        # Автоматичне повернення зірок за покупку
        if payment_type == 'stars':
            await bot.refund_star_payment(user_id=user_tg_id, telegram_payment_charge_id=payment_id)
```

ДОДАТОК Б

Основні моделі бази даних:користувачі, категорії, товари й покупки

```
from typing import List
from sqlalchemy.orm import Mapped, mapped_column, relationship
from sqlalchemy import BigInteger, Text, ForeignKey
from bot.dao.database import Base
class User(Base):
    telegram_id: Mapped[int] = mapped_column(BigInteger, unique=True,
nullable=False)
    username: Mapped[str | None]
    first_name: Mapped[str | None]
    last_name: Mapped[str | None]
    purchases: Mapped[List['Purchase']] = relationship(
        "Purchase",
        back_populates="user",
        cascade="all, delete-orphan"
    )
    def __repr__(self):
        return f"<User(id={self.id}, telegram_id={self.telegram_id},
username='{self.username}')>"
class Category(Base):
    __tablename__ = 'categories'
    category_name: Mapped[str] = mapped_column(Text, nullable=False)
    products: Mapped[List["Product"]] = relationship(
        "Product",
        back_populates="category",
        cascade="all, delete-orphan"
    )
    def __repr__(self):
        return f"<Category(id={self.id}, name='{self.category_name}')>"
class Product(Base):
    name: Mapped[str] = mapped_column(Text)
    description: Mapped[str] = mapped_column(Text)
    price: Mapped[int]
    file_id: Mapped[str | None] = mapped_column(Text)
    category_id: Mapped[int] = mapped_column(ForeignKey('categories.id'))
    hidden_content: Mapped[str] = mapped_column(Text)
    category: Mapped["Category"] = relationship("Category",
back_populates="products")
    purchases: Mapped[List['Purchase']] = relationship(
        "Purchase",
        back_populates="product",
        cascade="all, delete-orphan"
    )
    def __repr__(self):
```

```
        return f"<Product(id={self.id}, name='{self.name}', price={self.price})>"
class Purchase(Base):
    user_id: Mapped[int] = mapped_column(ForeignKey('users.id'))
    product_id: Mapped[int] = mapped_column(ForeignKey('products.id'))
    price: Mapped[int]
    payment_id: Mapped[str] = mapped_column(unique=True)
    user: Mapped["User"] = relationship("User", back_populates="purchases")
    product: Mapped["Product"] = relationship("Product",
back_populates="purchases")
    def __repr__(self):
        return f"<Purchase(id={self.id}, user_id={self.user_id},
product_id={self.product_id}, date={self.created_at})>"
```

ДОДАТОК В

Файл с клавіатурами

```
from typing import List
from aiogram.types import InlineKeyboardMarkup
from aiogram.utils.keyboard import InlineKeyboardBuilder
from bot.dao.models import Category
def catalog_admin_kb(catalog_data: List[Category]) ->
InlineKeyboardMarkup:
    kb = InlineKeyboardBuilder()    for category in catalog_data:
        kb.button(text=category.category_name,
callback_data=f"add_category_{category.id}")
    kb.button(text="Відміна", callback_data="admin_panel")
    kb.adjust(2)
    return kb.as_markup()
def admin_send_file_kb() -> InlineKeyboardMarkup:
    kb = InlineKeyboardBuilder()
    kb.button(text="Без файлу", callback_data="without_file")
    kb.button(text="Відміна", callback_data="admin_panel")
    kb.adjust(2)
  return kb.as_markup()
def admin_kb() -> InlineKeyboardMarkup:
    kb = InlineKeyboardBuilder()
    kb.button(text="📊 Статистика", callback_data="statistic")
    kb.button(text="🏭 Керувати товарами",
callback_data="process_products")
    kb.button(text="🏠 На головну", callback_data="home")
    kb.adjust(2)
    return kb.as_markup()
def admin_kb_back() -> InlineKeyboardMarkup:
    kb = InlineKeyboardBuilder()
    kb.button(text="⚙ Адмін панель", callback_data="admin_panel")
    kb.button(text="🏠 На головну", callback_data="home")
    kb.adjust(1)
    return kb.as_markup()
def dell_product_kb(product_id: int) -> InlineKeyboardMarkup:
    kb = InlineKeyboardBuilder()
    kb.button(text="🗑 Видалити", callback_data=f"dell_{product_id}")
    kb.button(text="⚙ Адмін панель", callback_data="admin_panel")
    kb.button(text="🏠 На головну", callback_data="home")
    kb.adjust(2, 2, 1)
    return kb.as_markup()
def product_management_kb() -> InlineKeyboardMarkup:
    kb = InlineKeyboardBuilder()
    kb.button(text="➕ Додати товар", callback_data="add_product")
```

```
        kb.button(text="🗑 Видалити товар",
callback_data="delete_product")
        kb.button(text="⚙ Адмін панель", callback_data="admin_panel")
        return kb.as_markup()
```