

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему: РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТЕЛЕГРАМ-БОТА
МАГАЗИНУ
SOFTWARE DEVELOPMENT FOR A STORE TELEGRAM BOT

Виконав: студент(ка) 4 курсу, групи КНТз-219
Спеціальності 122 Комп'ютерні науки
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ТИМОШЕНКО В. О.

(прізвище та ініціали)

Керівник СЕРДЮК С. М.

(прізвище та ініціали)

Рецензент ПОЛЯКОВ М. С.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
 Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Тимошенка Владислава Олександровича
 (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програмного забезпечення Телеграм-бота магазину. Software Development for a Store Telegram Bot
 керівник проєкту (роботи) СЕРДЮК Сергій Микитович, к.т.н., доцент
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом закладу вищої освіти від “4” квітня 2023 року № 87
2. Строк подання студентом проєкту (роботи) 07 червня 2023 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та постановка завдання. 2. Розробка архітектури програми. 3. Розробка програмного забезпечення. 4. Керівництво програміста. 5. Керівництво оператора. Додатки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	СЕРДЮК С.М., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання “28” лютого 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програмного забезпечення	4– 5 тижні	Розділ 3
5	Керівництво програміста	6 тиждень	Розділ 4
6	Керівництво оператора	7 тиждень	Розділ 5
8	Оформлення пояснювальної записки. Нормоконтроль та рецензування	8 тиждень	Додатки
9	Захист роботи	9 тиждень	

Студент(ка)

_____ Владислав ТИМОШЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Сергій СЕРДЮК
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавру: 112 с., 3 табл., 42 рис., 2 дод., 13 джерел.

ТЕЛЕГРАМ БОТ МАГАЗИН, ОНЛАЙН МАГАЗИН, МАГАЗИН, PYTHON.

Об'єкт дослідження – процес продажу товару онлайн.

Предмет дослідження – онлайн телеграм боти для продажу товару через телеграм.

Мета роботи – підвищити кількість продажів шляхом більш ефективної роботи користувача з онлайн-засобами.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, мови програмування Python, персональний комп'ютер з процесором AMD Ryzen 7 2700 під управлінням операційної системи Microsoft Windows 10.

Результати. Створено телеграм бота, який керується за допомогою адміністратора та сервера, який за допомогою взаємодії з користувачем дає можливість робити покупки товарів.

Висновки. Розроблено телеграм бот для магазину одягу, який автоматизує та підвищує кількість продаж.

Галузь використання – онлайн торгівля.

ABSTRACT

Explanatory note to the bachelor's thesis: 112 pages, 3 tables, 42 figures, 2 appendices, 13 sources.

TELEGRAM BOT SHOP, ONLINE SHOP, SHOP, PYTHON.

The object of research is the process of selling goods online.

The subject of the research is online Telegram bots for selling goods via Telegram.

The goal of the work is to increase the number of sales through more effective user work with online tools.

Materials, methods and technical means: structural and object-oriented programming, Python programming languages, a personal computer with an AMD Ryzen 7 2700 processor running the Microsoft Windows 10 operating system.

The results. A Telegram bot has been created, which is controlled by the administrator and the server, which, through interaction with the user, makes it possible to make purchases of goods.

Conclusions. A telegram bot for a clothing store was developed, which automates and increases the number of sales.

The field of use is online trade.

ЗМІСТ

Перелік скорочень та умовних познач	9
Вступ.....	10
1 Аналіз предметної області.....	11
1.1 Огляд інформаційних потреб у сфері роздрібної торгівлі.....	11
1.2 Аналіз існуючих аналогів	13
1.3 Технічне завдання	15
1.3.1 Підстава для розробки	15
1.3.2 Мета розробки та її призначення.....	15
1.3.3 Основні вимоги до програмного продукту.....	15
1.3.4 Вимоги до технічних засобів, операційної системи і мови програмування	17
1.3.6 Вимоги до програмної документації.....	17
1.3.7 Порядок приймання і тестування	18
1.4 Висновки за розділом	18
2 Розробка архітектури програми.....	20
2.1 Моделювання програмного забезпечення	20
2.2 Розробка архітектури програмного забезпечення	22
2.2.1 Опис модулів системи	23
2.3 Обґрунтування вибору інструментів розробки.....	25
2.3.1 Обґрунтування вибору мови програмування	25
2.3.2 Обґрунтування вибору фреймворків та бібліотек	25
2.3.3 Обґрунтування вибору бази даних	27
2.3.4 Обґрунтування вибору IDE	28

	7
2.4 Висновки за розділом 2	28
3 Розробка програмного забезпечення.....	30
3.1 Розробка компонентів системи.....	30
3.1.1 Розробка компонентів запуску бота.....	31
3.1.2 Розробка компонентів моделі авторизації користувача.....	37
3.2 Розробка компонентів моделі інтерфейсу	40
3.2.1 Розробка компонентів користувацького інтерфейсу.....	40
3.2.2 Розробка компонентів адміністраторського інтерфейсу	54
3.2.3 Розробка компонентів функціоналу зв'язку з підтримкою	66
3.2.4 Розробка компонентів функціоналу роботи з балансом	68
3.3 Тестування застосунку	70
3.4 Висновки за розділом 3	72
4 Керівництво програміста.....	74
4.1 Призначення та умови застосування.....	74
4.1.1 Призначення програми	74
4.1.2 Умови застосування програми	74
4.2 Характеристики програми.....	75
4.3 Звертання до програми	75
4.4 Початкові та вихідні дані	75
4.5 Повідомлення програмісту.....	75
5 Керівництво оператора	77
5.1 Призначення програми	77
5.2 Умови виконання програми	77
5.3 Виконання програми.....	77
5.4 Повідомлення	78

	8
Висновки	80
Перелік джерел посилання	81
Додаток А Код програми.....	83
Додаток Б Слайди презентації	105

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- API – програмний інтерфейс програми;
- DOM – документна об’єктна модель сторінки браузера;
- E-MAIL – адреса онлайн поштової скриньки;
- HTML – мова гіпертекстової розмітки сторінки браузера;
- HTTP – протокол передачі даних у інтернеті;
- ID – унікальний ідентифікатор;
- IDE – інтегрована середовище розробки;
- JSON – JavaScript об’єктна нотація документа;
- JSX – розширення мови JavaScript;
- MVC – модель, представлення, контролер;
- REST - передача репрезентативного стану;
- SEO – пошукова оптимізація;
- SR – принцип єдиної відповідальності;
- UI – користувацький інтерфейс;
- UML – уніфікована мова моделювання;
- URL – уніфікований локатор ресурсів, адресна строка у браузері.

ВСТУП

У сучасному світі сфера електронної комерції стає все більш популярною та затребуваною. Серед різних інструментів, що використовуються для продажу товарів та послуг в інтернеті, телеграми займають особливе місце. Телеграм – це месенджер, який на сьогоднішній день є одним із найпопулярніших у світі. Більше того, існує можливість створення телеграм ботів, які дозволяють автоматизувати продаж та полегшити процес взаємодії з клієнтами.

У цій роботі будуть розглянуті методи та засоби розробки телеграм бота для онлайн-магазину, а також буде представлений сам код бота та його функціональні можливості. Дипломна робота складатиметься з вступу, основної частини, в якій будуть описані всі етапи розробки телеграм бота, та висновки, де будуть підбиті підсумки роботи та зроблені висновки про її результативність та практичну значимість.

Мета роботи – підвищити кількість продажів шляхом більш ефективної роботи користувача з онлайн-засобами та полегшити процес покупок, покращити комунікацію з користувачами та забезпечити задоволення їх потреб.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд інформаційних потреб у сфері роздрібної торгівлі

Предметною областю дослідження є розробка телеграм-бота для магазину, що дозволить здійснювати покупки через месенджер Telegram. У зв'язку зі стрімким розвитком електронної комерції, магазини постійно стикаються з необхідністю покращувати процеси продажу та обслуговування клієнтів, що вимагає від них додаткової роботи з обробки даних та взаємодії з клієнтами.

Аналіз інформаційних потреб магазинів дозволив визначити, що вони потребують зручного та ефективного інструменту для збільшення продажів, а також для полегшення процесу замовлення та оплати товарів. Водночас, з урахуванням тенденції зростання популярності месенджерів серед користувачів, створення телеграм-бота є доцільним та актуальним рішенням для покращення комунікації з клієнтами та збільшення продажів в онлайн-режимі.

Для успішної реалізації проекту, необхідно провести детальний аналіз проблемної області, а також вивчити досвід використання телеграм-ботів у магазинній діяльності, що дозволить врахувати особливості ринку та користувацьких потреб, та врахувати їх при розробці функціоналу програми.

При аналізі предметної області було виявлено, що для успішної діяльності магазину важливо мати зручний і швидкий спосіб замовлення товарів. Одним з ефективних рішень для цього є використання телеграм-боту, який дозволяє клієнтам здійснювати замовлення в режимі онлайн, а також отримувати інформацію про наявність товарів та їх характеристики. Таким чином, розробка телеграм-боту для магазину є актуальним та важливим завданням, що може значно полегшити процес замовлення товарів для клієнтів та покращити роботу магазину в цілому.

Також у рамках аналізу було визначено, що телеграм бот є перспективним інструментом для забезпечення ефективної взаємодії з

клієнтами. Він дозволяє в автоматичному режимі надавати інформацію про продукти, приймати замовлення та взаємодіяти з покупцями. Крім того, використання телеграм боту може зменшити витрати на підтримку клієнтів, оскільки він дозволяє автоматизувати багато процесів, пов'язаних із обслуговуванням клієнтів. Таким чином, використання телеграм робота в магазинній діяльності може принести значну користь і допомогти підвищити ефективність бізнесу.

В рамках аналізу інформаційних потреб магазинної діяльності також були розглянуті основні функції, які повинен виконувати телеграм бот магазину, а саме надання інформації про товари в магазині, обробка замовлень від користувачів, оформлення покупки та оплата замовлення, зворотній зв'язок з користувачами для вирішення питань та проблем та збір та аналіз даних про покупки та переваги клієнтів для покращення якості обслуговування.

Було виявлено, що в даний час в умовах пандемії COVID-19 онлайн-торгівля отримала новий імпульс і стала більш затребуваною, що наголошує на актуальності створення телеграм бот магазину.

Були виявлені основні типи інформаційних систем, що використовуються в магазинах, такі як системи управління складом, системи обліку продажів, системи управління персоналом та ін. Було проведено дослідження основних функцій та можливостей кожної системи, а також їх переваг та недоліків. Це допомогло сформулювати уявлення про те, які функції повинна виконувати розроблювана інформаційна система, щоб найкращим чином відповідати потребам і завданням магазину.

В процесі аналізу інформаційних потреб магазину були виявлені не тільки основні завдання та потреби, але й проблеми, з якими стикаються співробітники під час роботи з поточними системами та програмним забезпеченням. Це дозволило більш глибоко зрозуміти, які саме функціональні вимоги мають бути враховані у розробці нової програми для оптимізації роботи магазину.

Розробка програмного продукту для автоматизації магазинної діяльності, дозволить покращити якість обслуговування покупців, оптимізувати управління товаром та покращити аналіз продажів для прийняття більш ефективних управлінських рішень. Пропонований програмний продукт буде включати такі функції: управління товаром (його ціною, описом та наявністю), управління продажами і клієнтською базою, генерацію звітності, аналіз даних та багато іншого.

У телеграм боті, який буде розроблений у ході дипломної роботи, буде присутні два типи користувачів, вони будуть чітко розділяти функціонал телеграм – боту. Одні будуть виконувати роль покупців, а інші виконувати роль адміністратора.

За додатковою інформацією про можливості боту та різних типів користувачів, детальніше буде описано в технічному завданні, яке міститиме повний перелік функцій та вимог до розробки програмного продукту.

1.2 Аналіз існуючих аналогів

Під час вибору аналогів було проаналізовано два онлайн-магазини, що діють у вигляді телеграм-ботів, перший це свій, а другий магазин іншого бренду. Для кожного з них було проведено детальний опис функцій, а також виділено основні переваги та недоліки використання даного застосунку.

Для порівняння бота, Shop Lolipop Bot, з іншим схожим ботом, візьмемо, наприклад, ShoppingExpress Bot. Критерії порівняння, що подані в табл. 1.1:

- можливість робити замовлення – даний критерій дає змогу робити замовлення в самому додатку не виходячи з нього або за іншими посиланнями;
- оплата онлайн – цей критерій дає можливість сплачувати замовлення відразу у додатку;
- комунікація з користувачами – цей критерій дає змогу спілкування користувача з будь – яких питань з адміністратором в самому боті;

- розширені можливості – цей критерій відображає наявність додаткових функцій та можливостей у боті;
- безкоштовність – доступ до додатку та його функціоналу безкоштовна;
- актуальність інформації – цей критерій відповідає за оновлення та перевірку нової інформації в застосунку;
- стабільність роботи – цей критерій відповідає за безперебійну роботу додатку.

Таблиця 1.1 – Порівняльна таблиця критеріїв

Критерії	Результати порівняння	
	Shop Lolipop Bot	ShoppingExpress Bot
Можливість робити замовлення	+	+
Оплата онлайн	+	-
Комунікація з користувачем	+	+
Розширені можливості	+	-
Безкоштовність	+	+
Актуальність інформації	+	-
Стабільність роботи	+	-

Враховуючи вищезазначені відмінності, можна зробити висновок, що обидва боти мають свої унікальні особливості і можуть задовольнити потреби різних типів користувачів. Shop Lolipop Bot зосереджується на стабільності роботи та можливостях відстеження статусу замовлення та своєму вбудованому каталозі, також власному кабінеті користувача з балансом, тоді як ShoppingExpress Bot пропонує більш мінімалістичний дизайн та обмежений функціонал та рідше оновленою інформацією.

1.3 Технічне завдання

1.3.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу бакалавра на тему Розробка застосунку «телеграм-бота магазину», затверджене наказом № 102 від 27 квітня 2022 р. по НУ «Запорізька політехніка».

1.3.2 Мета розробки та її призначення

Метою розробки є створення програмного забезпечення для магазину, яке дозволить автоматизувати процеси, пов'язані з обліком товарів та продажами через меседжер телеграм. Головним призначенням програми є полегшення роботи персоналу магазину, зменшення часу, витраченого на ведення обліку та підвищення ефективності управління магазином. Крім того, програма дозволить покращити обслуговування клієнтів, забезпечити швидкий та точний облік товарів, максимальну взаємодію між різними відділами магазину та підвищити рівень контролю над діяльністю магазину.

1.3.3 Основні вимоги до програмного продукту

Телеграм – бот має мати два типи користувачів, а саме: звичайний користувач (покупець) користувач та адміністратор. У різних типів користувачів, будуть відрізнятись функціональні можливості боту. Адміністраторські можливості будь мати також спроможності покупця.

Вимоги до програмного продукту повинні враховувати всі аспекти його функціональності та якості. Для досягнення цього мети, основні вимоги до програмного продукту наведені в наступному переліку [2], [3], [5]:

а) ффункціональність: програмний продукт повинен мати всі необхідні функції для забезпечення ефективної роботи користувачів;

б) надійність: програмний продукт повинен бути стабільним та надійним в роботі, забезпечуючи відсутність помилок та аварій;

в) безпека: програмний продукт повинен бути захищеним від можливих загроз безпеці, таких як злами, віруси тощо;

г) простота використання: програмний продукт повинен бути зрозумілим та простим у використанні для користувачів різного рівня кваліфікації;

г) масштабованість: програмний продукт повинен бути здатним забезпечувати ефективну роботу при збільшенні обсягу даних та користувачів;

д) сумісність: програмний продукт повинен бути сумісним з іншими програмними продуктами та операційними системами;

е) підтримка та розвиток: програмний продукт повинен мати можливість підтримки та подальшого розвитку для забезпечення його актуальності та відповідності вимогам користувачів та ринку.

Бот, що розроблюється має надавати користувачеві наступні можливості:

а) для адміністратора:

- 1) редагувати товар (добавляти та видаляти його, змінювати його параметри);
- 2) виконувати обов'язки технічної підтримки;
- 3) приймати оплату;
- 4) відстежувати замовлення;
- 5) виконувати також роль покупця;
- 6) перевірка оплати;
- 7) змога робити повідомлення для кожного (розсилка), отримання контактної інформації користувача.

б) для зареєстрованих користувачів:

- 1) змоги вибору товару та оформлення замовлення;
- 2) оплата товару;
- 3) зв'язок з технічною підтримкою;

4) Можливість отримання інформації про своє замовлення.

1.3.4 Вимоги до технічних засобів, операційної системи і мови програмування

Програма, що розробляється у вигляді телеграм - боту, має бути написана на мові Python у середовищі розробки PyCharm. Програма кросплатформена, та підтримує будь-яку операційну систему та пристрій, який підтримує telegram. Вимоги до технічних засобів, мають складати вимоги до операційної системи на якій запускається telegram-web, telegram (у вигляді застосунку) або у смартфоні, також вимогою є наявність стабільного інтернет зв'язку [1], [2].

1.3.5 Вимоги до інформаційної та програмної сумісності

Вимоги до інформаційної та програмної сумісності пов'язані з тим, щоб програмний продукт міг працювати в різних середовищах, з різними версіями операційної системи та з різним програмним забезпеченням. Для забезпечення інформаційної та програмної сумісності, програмний продукт повинен бути розроблений з використанням стандартів та протоколів, які підтримуються різними операційними системами та програмним забезпеченням[1].

1.3.6 Вимоги до програмної документації

Вимоги до програмної документації включають наступні пункти [1]:

- документація повинна бути зрозумілою та зручною для використання;
- вся документація повинна бути відформатована та відповідати встановленим стандартам;

- документація повинна бути актуальною та відображати останні зміни в програмному продукті;
- документація повинна бути детальною та містити достатньо інформації для розуміння функціональності продукту та його використання;
- документація повинна включати опис всіх модулів, функцій та процедур, що використовуються в програмному продукті;
- документація повинна містити інструкції з установки, настройки та використання програмного продукту;
- документація повинна містити опис можливих помилок та способів їх виправлення;
- документація повинна бути доступною для перегляда та завантаження з офіційного сайту програмного продукту або вбудованої довідки в програмному продукті.

1.3.7 Порядок приймання і тестування

Для проведення тестування розробленого програмного забезпечення необхідно виконати усі можливі сценарії роботи оператора, з метою перевірки відповідності вимогам, визначеним у технічному завданні. У разі успішного проходження етапу тестування та відповідності програми усім вимогам, вона вважається прийнятною для використання.

1.4 Висновки за розділом

У даному розділі були визначені вимоги до програмного продукту з урахуванням його функціональності тобто купувати товар через застосунок телеграм, програмної сумісності на яких пристроях можливе користування застосунком, а також програмної документації.

Були визначені основні функції та можливості програмного продукту, такі як обробка замовлень, управління інвентарем, автоматизована комунікація з користувачами через месенджер, інтеграція з платіжними системами тощо. Було встановлено, що програмний продукт повинен мати зручний інтерфейс для двох видів користувачів, перший це адміністратор, а другий звичайна людина. В результаті проведеного аналізу предметної області та вимог до програмного продукту були визначені основні функціональні вимоги до програми, а також вимоги до технічних засобів, було визначено що люди зможуть купувати твоар через смартфон на настільний ПК. Мовою програмування було обрано Python. Були також встановлені вимоги до інформаційної та програмної сумісності, а також до програмної документації. Для успішного виконання проекту було розроблено порядок приймання та тестування програмного забезпечення.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Моделювання програмного забезпечення

У розроблюваній системі користувачі можуть мати різний рівень доступу до функціоналу залежно від їхнього статусу - адміністратор та звичайний користувач. Це означає, що адміністратори мають доступ до більш розширеного функціоналу, ніж звичайні користувачі. Адміністратором вважається той, хто має право на редагування товару та загалом управління ботом. Зазвичай це власники магазину чи наймані працівники які дають доступу людині права адміністратора, а звичайний користувач, це той користувач який після запуску бота може його використовувати лише для придбання товару чи його перегляду. Ця можливість різного рівня доступу забезпечує зручну та безпечну роботу з системою, дозволяючи користувачам використовувати тільки той функціонал, який їм необхідний.

Функціонал звичайного користувача:

- користувач може переглядати будь – який товар та купляти його;
- може спілкуватися з технічною підтримкою (адміністратором).

Функціонал адміністратора:

- редагування та додавання товару;
- функція відповіді на повідомлення в тех - підтримці;
- робити розсилку, перегляд замовлень;
- отримання всіх користувачів бота.

Процес реєстрації користувача відбувається за рахунок проходження капчі. Якщо людина не пройде її то функціонал та доступ до бота не буде наданий. Капча з'являється після натискання на кнопку «start», яка наведена на рис. 2.1. Капча складається з того, що потрібно вибрати вірний фрукт. Після успішного проходження її користувачу надається доступ до функціоналу бота. Вона зроблена для того щоб можливо було перевірити людина чи бот користується ботом.

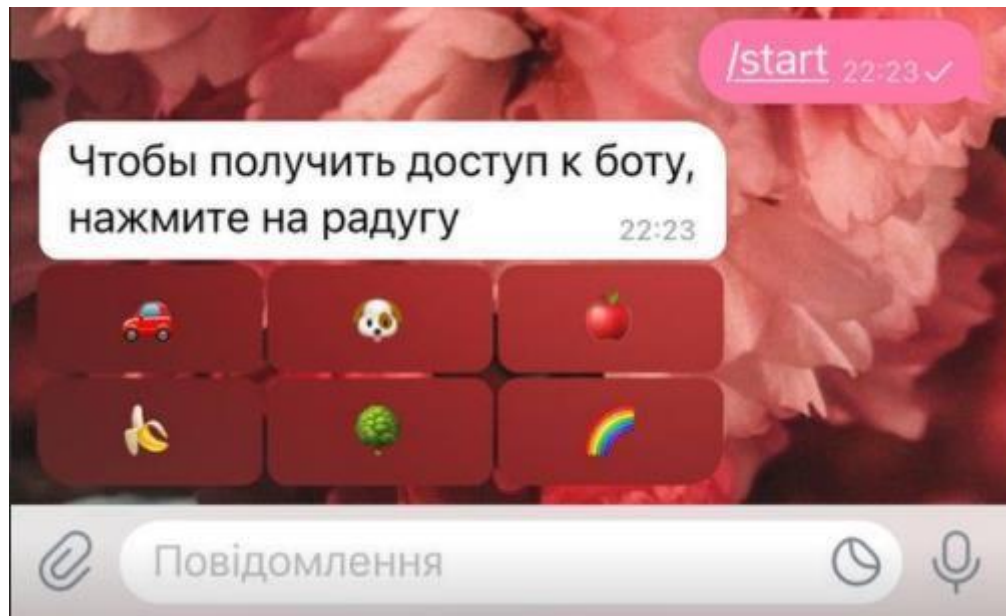


Рисунок 2.1 - Приклад капчі

Після проходження реєстрації користувач переходить до «Меню користувача». У якому йому доступні такі функції як:

- магазин – товар;
- меню користувача – функціонал для користувача;
- «F.A.Q» – правила;
- підтримка – зв'язок з адміністратором.

У розділі «Магазин» користувачу відображаються категорії. Після вибору категорії відображається підкатегорії даної категорії. Після чого людина може переглянути товар який є в магазині, прочитати його опис, подивитися які є розміри та ціну. Якщо товар сподобався то користувач може додати його в «Кошик». По завершенню всього вибору людині треба перейти до «Меню користувача» та обрати там «Кошик». Після чого там можливо оформити та сплатити замовлення.

У розділі «Меню користувача» є такі функції:

- баланс – для перегляду вашого балансу якщо ви його поповнювали;
- мої покупки – перегляд ваших замовлень;
- кошик – функція збору ваших обраних товарів для оплати та замовлення їх;

– реферальна система – якщо в вас є промо – код, то ви можете його активувати та він надає знижку.

Для більш чіткого розуміння процесів та сценаріїв програми, було розроблено діаграму прецедентів (рис 2.2), яка показує зв'язаність окремих процесів.

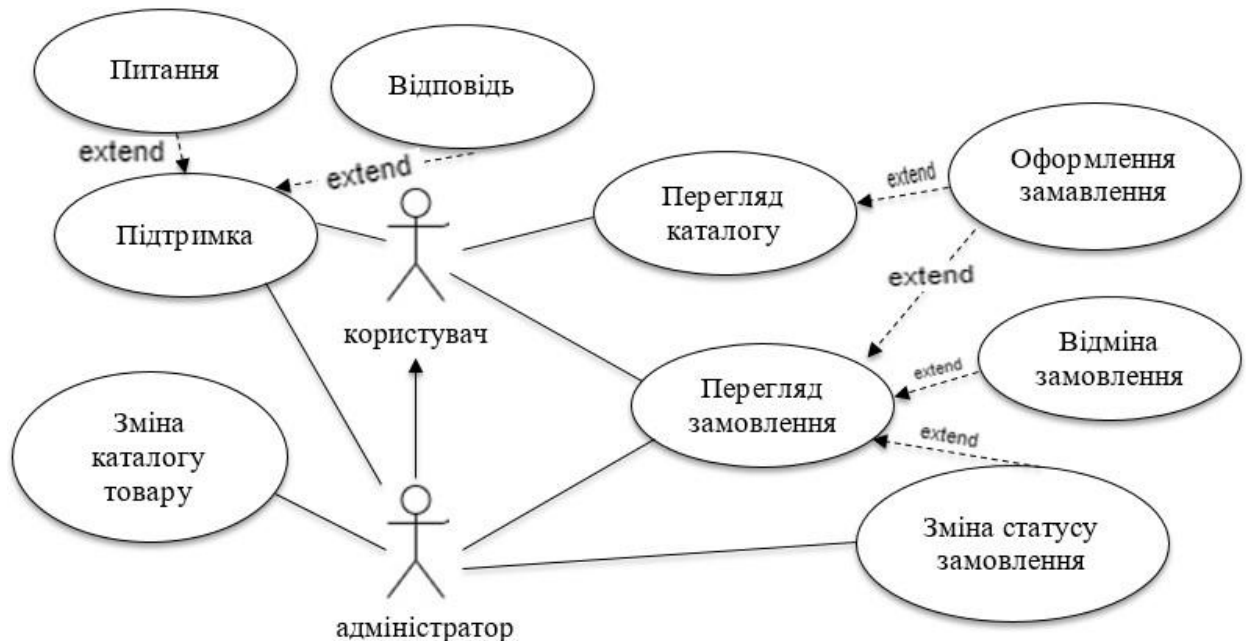


Рисунок 2.2 - Діаграма прецедентів

2.2 Розробка архітектури програмного забезпечення

Розробка архітектури програмного забезпечення включає в себе визначення структури та організації компонентів програмного продукту, що дозволяє досягти максимальної ефективності та забезпечити його стабільну роботу. Для даного проекту, архітектура програмного забезпечення буде побудована з урахуванням функціональних вимог та вимог до ефективності та безпеки. Для цього буде використана модульна архітектура з можливістю додавання та видалення компонентів залежно від потреб користувачів. Для забезпечення безпеки даних буде використана модель з розділенням прав доступу до даних для різних користувачів.

Телеграм бот який розроблюється, буде складатися з чотирьох основних етапів:

- розробка адмін-панелі та її функціоналу;
- розробка панелі користувача;
- розробка взаємодії бота з базою даних;
- підключення оплати товару та поповнення балансу.

Крім того, важливим етапом розробки архітектури програмного забезпечення є вибір інструментів для реалізації проекту. Необхідно визначитись з мовою програмування, фреймворками, бібліотеками та іншими технологіями, які будуть використовуватись під час розробки. Мовою програмування було обрано Python. Для розробки телеграм боту було використано бібліотеку `aiogram`, для роботи с базою даних `psycopg2`.

2.2.1 Опис модулів системи

Опис модулів системи відображає структуру програмного забезпечення та взаємозв'язки між його компонентами. Модуль - це функціонально завершена частина програмного забезпечення, яка виконує певні функції та може бути розглянута як окремий блок коду.

В системі для зареєстрованих та незареєстрованих користувачів можна виділити такі модулі:

- модуль авторизації - відповідає за процес входу користувача в систему з використанням капчі;
- модуль профілю користувача - зберігає особисту інформацію користувача, таку як ім'я, прізвище, номер телефона та інші деталі в базу даних;
- модуль каталогу товарів - містить перелік усіх товарів, які доступні для перегляду та придбання в системі;
- модуль кошика - зберігає товари, які додані до кошика користувачем, а також надає можливість оформити замовлення та оплатити його;

- модуль оплати - відповідає за здійснення оплати замовлення, яке зробив користувач;
- модуль зв'язку з користувачем - надає можливість користувачам зв'язатися з підтримкою сервісу, надіслати запитання чи відгук;
- модуль бази даних – цей модуль відповідає за зберігання даних о користувачах, даних товару, замовлень та балансу. Цей модуль являє собою частину програмного інтерфейсу, який відповідає за збереження даних о системі, та працює лише з модулем серверної частини, наведено на рис. 2.3.

Кожен з цих модулів має свої вхідні та вихідні параметри, а також взаємозв'язки з іншими модулями системи. Взаємодія модулів системи забезпечується за допомогою спеціальних протоколів та інтерфейсів, що визначають стандарти обміну даними.

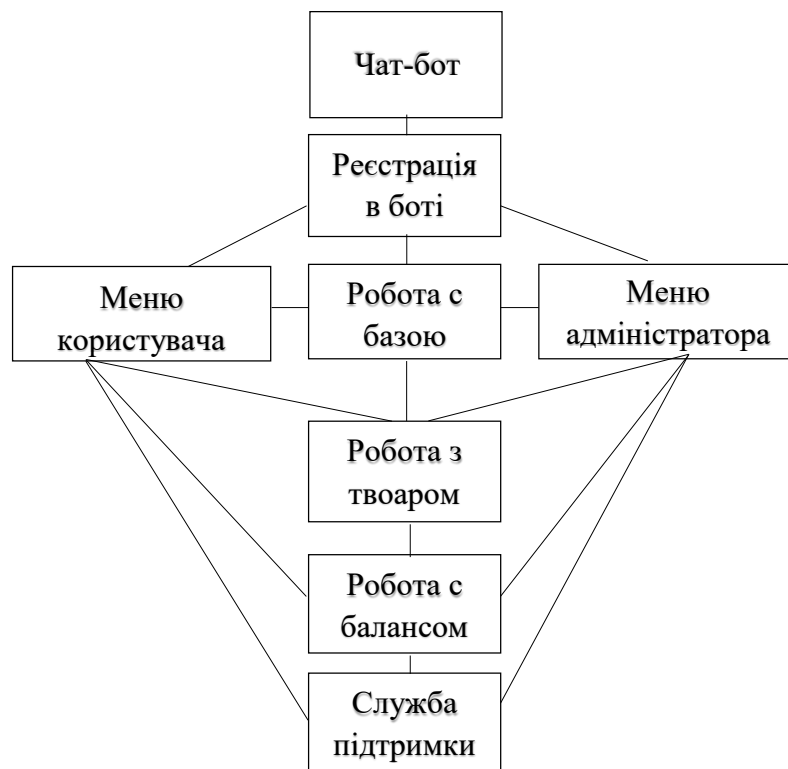


Рисунок 2.3 – Структура системи

2.3 Обґрунтування вибору інструментів розробки

2.3.1 Обґрунтування вибору мови програмування

Вибір мови програмування є важливим етапом розробки програмного забезпечення, оскільки він визначає можливості та обмеження розробки. При виборі мови програмування були враховані такі критерії:

- функціональні можливості: мова програмування повинна мати достатній набір функцій, щоб забезпечити необхідний функціонал системи;
- швидкодія: система повинна працювати достатньо швидко для забезпечення комфортної роботи користувачів;
- підтримка: мова програмування повинна мати активну спільноту, яка надає допомогу та підтримку вирішення проблем;
- безпека: мова програмування повинна мати можливість забезпечення безпеки системи та її даних.

В результаті аналізу цих критеріїв, мова програмування Python була обрана для розробки системи. Python - це високорівнева мова програмування, що дозволяє швидко та ефективно розробляти складні програмні продукти. Python має багатий набір бібліотек та фреймворків, що спрощують розробку, а також активну спільноту, яка надає підтримку та допомогу. Python також має вбудовані функції забезпечення безпеки, такі як обробка винятків та перевірка типів даних.

2.3.2 Обґрунтування вибору фреймворків та бібліотек

Python є досить популярною мовою програмування для створення ботів Telegram. Для розробки чат-ботів на основі API Telegram Bot, можна використовувати фреймворк aiogram. Основними перевагами aiogram є його простота використання та повна асинхронність, що дозволяє оптимізувати роботу з ботом. Крім того, aiogram побудований на стандартній бібліотеці

Python `asyncio` та асинхронному веб-фреймворку `aiohttp`, що дозволяє легко розширювати функціональність бота.

Для роботи з базою даних було обрано PostgreSQL, що є однією з найбільш потужних та надійних реляційних баз даних. Вона має вбудовані засоби забезпечення безпеки даних та підтримки транзакцій, що є важливим для роботи з конфіденційною інформацією.

Окрім фреймворку, необхідні ще додаткові бібліотеки які оптимізують роботу з задачами при розробці, а саме:

- `psycopg2` - вона надає зручний інтерфейс для роботи з PostgreSQL, включаючи підтримку транзакцій, курсорів та зовнішніх ключів. Крім того, `psycopg2` досить швидка та ефективна, що особливо важливо для систем, які отримують велику кількість запитів до бази даних;
- API Telegram Bot - це інтерфейс з HTTP протоколом для забезпечення взаємодії з ботами у Telegram. Кожен бот представляє собою окремий обліковий запис, створений з метою автоматизувати обробку та надсилання повідомлень;
- для використання платіжної системи використовується її API документація;
- `pickle` – бібліотека `pickle` дозволяє зберігати об'єкти Python у файли, які можуть бути прочитані та використані пізніше;
- `json` - це набір функцій для роботи з даними у форматі JSON (JavaScript Object Notation). Ця бібліотека забезпечує можливість кодування та декодування даних у форматі JSON;
- `aiohttp` - бібліотека `aiohttp` є асинхронною бібліотекою для створення мережових програм на Python, заснованої на бібліотеці `asyncio`. Вона надає можливість створювати клієнтів та сервери HTTP та WebSocket;
- бібліотека `asyncio` - це бібліотека для асинхронного програмування у Python. Вона дозволяє створювати асинхронний код з використанням корутин (`async/await`) та інших інструментів, які дозволяють уникнути блокування потоку виконання;

- бібліотека `os` - це модуль мови програмування Python, який надає функції для роботи з операційною системою. Вона дозволяє взаємодіяти з файловою системою, отримувати інформацію про поточний процес, отримувати доступ до середовища виконання тощо;
- також використовуються стандартні бібліотеки Python (`random`, `time` та інші).

2.3.3 Обґрунтування вибору бази даних

PostgreSQL є потужною та надійною базою даних з відкритим вихідним кодом, яка має багато переваг в порівнянні з іншими базами даних, такими як MySQL чи SQLite.

Основні переваги PostgreSQL включають:

- надійність та стабільність: PostgreSQL має високий рівень стійкості до збоїв та забезпечує збереження даних навіть у разі аварійного завершення роботи;
- розширюваність: PostgreSQL забезпечує можливість розширення та підключення різних модулів для роботи з базою даних;
- підтримка SQL-стандартів: PostgreSQL підтримує більшість SQL-стандартів та забезпечує сумісність з іншими системами управління базами даних;
- масштабованість: PostgreSQL може обробляти великі обсяги даних та запитів, що робить його ідеальним вибором для багатокористувацьких додатків;
- відкритість вихідного коду: PostgreSQL є вільно поширюваним програмним забезпеченням з відкритим вихідним кодом, що забезпечує його безпеку та незалежність від комерційних вендорів.

Отже, вибір PostgreSQL в якості бази даних для розробки програмного забезпечення є обґрунтованим та забезпечить надійність та стабільність роботи системи.

2.3.4 Обґрунтування вибору IDE

PyCharm є однією з найбільш популярних інтегрованих середовищ розробки (IDE) для Python, розроблених компанією JetBrains. Він має широкий набір функцій, що роблять процес розробки більш продуктивним та ефективним.

Основні переваги PyCharm:

- підтримка різних мов програмування: PyCharm підтримує не тільки Python, але й інші мови програмування, такі як HTML, CSS, JavaScript, SQL, і багато інших;
- функціональність для розробки веб-додатків: PyCharm має вбудований сервер, що дозволяє виконувати та налагоджувати веб-додатки безпосередньо з IDE;
- налагоджувальник: PyCharm має потужний налагоджувальник, який допомагає знайти та виправити помилки в коді;
- підтримка систем контролю версій: PyCharm підтримує різні системи контролю версій, такі як Git, SVN та Mercurial;
- автодоповнення коду та вбудований редактор: PyCharm має вбудований редактор, який дозволяє зручно редагувати код, а також автоматично доповнювати код, що зменшує кількість помилок у процесі розробки.

Таким чином, обґрунтування вибору PyCharm полягає у тому, що він є потужним та зручним інструментом для розробки програмного забезпечення на мові програмування Python з підтримкою великої кількості функцій, що полегшують процес розробки та знижують кількість помилок.

2.4 Висновки за розділом 2

В цьому розділі було розглянуто функціональні можливості програмного забезпечення, а також сценарії, за якими вони повинні

виконуватись. Були визначені ролі користувачів та їх функціональні можливості в системі. Були визначені задачі кожного компонента та визначена їх взаємодія між собою.

В процесі вибору інструментів для розробки програмного забезпечення було прийнято рішення використовувати мову програмування Python, а також були обрані фреймворки та бібліотеки для покращення продуктивності розробки. Для зберігання даних було вибрано базу даних PostgreSQL, що забезпечує надійність та швидкість доступу до інформації. Для зручного та продуктивного програмування було обрано IDE PyCharm, що має багатий функціонал та підтримку відразу декількох мов програмування.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка компонентів системи

Згідно з обраним функціоналом який було обрано в даному проекті, розробка поділяється на дві частини:

- адміністраторська частина;
- панель користувача.

До частини панелі адміна належить такий функціонал:

- редагування товару (опису, ціни, назви та іншого);
- перегляд замовлень та користувачів боту;
- розробка розсилки користувачам у боті;
- зв'язок з користувачем через бота.

До панелі користувача належать функції перегляд товару, проходження реєстрації (проходження капчі), можливість зробити замовлення та зв'язок з адміністратором.

Модулі поділені на компоненти системи, які містять у собі реалізацію певних функціональних можливостей програмного застосунку.

Опис модулів та їх компонентів наведено у табл. 3.1.

Таблиця 3.1 – Модулі та їх компоненти

Назва модулю	Назва компоненту	Опис компонента
1	2	3
Модуль користувацького інтерфейсу	usermenu	Даний компонент системи містить у собі функції обробки усього інтерфейсу користувача від замовлення до зв'язку з адміністратором.
	box	Даний компонент відповідає за роботи з кошиком. Тобто оформлення замовлення, оплата його та інше.
	catalog	Даний компонент відповідає за роботу з товаром. Щоб користувач міг переглядати та обирати товар який хоче купувати.
	payments	Даний компонент містить у собі функціонал оплати або поповнення балансу.

Продовження таблиці 3.1

1	2	3
Модуль авторизації користувача	captcha	Даний компонент містить у собі функціонал проходження капчі, суть якої полягає у виборі правильної картинки. Після успішного проходження користувач додається до бази даних та надається доступ до бота.
Модуль адміністраторської частини	admin	Даний компонент містить у собі функціонал створення функціоналу для роботи адміністратора. Це редагування категорій, правил, можливість робити розсилку користувачам, отримання списку користувачів та змога їх блокувати.
	userfaq	Даний компонент містить у собі функціонал створення та редагування правил.
	support	Даний компонент містить у собі функціонал спілкування користувача з адміністратором.
Модуль допоміжного функціоналу для бота.	markups	Даний компонент містить у собі функціонал створення клавіатур для бота.
	functions	Даний компонент відповідає за роботу редагування товару.
	states	Даний компонент відповідає за роботу машини стану яка запам'ятовує дії користувача.
	config	Даний компонент відповідає за данні бота (Токен, id адміністратора, підключення до бази даних та інше).
Модуль запуску	main	Даний компонент відповідає за підключення всіх модулів до бота.
	start	Даний компонент відповідає за запуск бота.
Модуль бази даних	createtables	Даний компонент використовується для створення таблиць у базі даних.
	database	Даний компонент описує модель користувача у базі даних також здійснення його замовлень, вносить зміни у базі даних при редагуванні або якщо добавляти товар в боті. Вносить інформацію про замовлення.

3.1.1 Розробка компонентів запуску бота.

До компонентів запуску відносяться файли Main та Start. Для того щоб працювати потрібен допоміжний файл config. В якому знаходиться TOKEN бота та архітектура бота для його запуску та роботи.

Main.py є кореневим файлом завдяки якому працюють загалом весь функціонал.

Код компоненту main.py:

```

from config import dp
from aiogram.utils import executor
import start
import support
import admin
import userfaq
import catalog
import box
import usermenu

if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)

```

Містить у собі виклик компонентів Main, Start, config, states, admin та інших. У цьому файлі відбувається обробка всіх компонентів бота та його функціоналу (Рисунок 3.1).

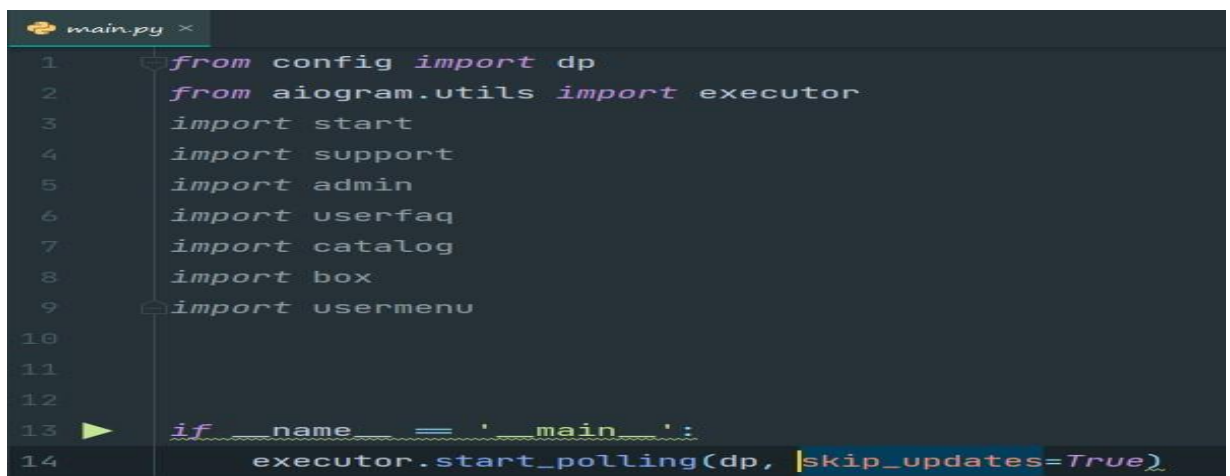


Рисунок 3.1 – Виклик всіх компонентів у Main.py

У файлі config.py при розробці телеграм бота міститься конфігураційна інформація, яка потрібна для роботи бота [10]. Це:

- TOKEN - токен бота, який отримується під час реєстрації бота у Telegram API. Цей токен використовується для авторизації бота при взаємодії з Telegram API;
- DATABASE_URL - URL до бази даних, яка використовується для зберігання інформації про користувачів та їхні запити до бота;

- BOT_URL - URL до телеграм боту;
- ADMIN_CHAT_ID - ідентифікатор чату адміністратора бота, до якого будуть надсилатись повідомлення про помилки та іншу важливу інформацію;
- інші налаштування, такі як TIMEZONE, LOGGING_LEVEL тощо, які допомагають забезпечити коректну роботу бота та зручне відлагодження коду.

Реалізація компонента config.py:

```
from aiogram import Bot
from aiogram.dispatcher import Dispatcher
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from database import DB

admins = [531852145]

url = 'https://t.me/shop_lolipop_bot'

bot = Bot(token='5618528182:AAGr9w9xqAaG0aTzb8dgm2T0kU1jxkjGk3k',
parse_mode='html')
storage = MemoryStorage()
dp = Dispatcher(bot, storage=storage)

db = DB('shop', 'postgres', '1111', 'localhost')
```

Файл start.py відповідає за надсилання привітального повідомлення після проходження успішної реєстрації. Та перевірку чи заблокований користувач. Після успішної проходження реєстрації користувач буде доданий до бази даних, якщо користувач заблокований то йому прийде повідомлення по те, що він заблокований.

Код компоненты start.py:

```

from config import dp, db
from aiogram import types
from captcha import Captcha
from markups import rules_mkp, menu_mkp

@dp.message_handler(commands='start')
async def startcmd(message: types.Message):
    stat = db.check_userstat(message.from_user.id)
    if len(message.text.split()) == 2:
        db.add_user(message.from_user.id,
message.from_user.username, int(message.text.split()[1]))
    else:
        db.add_user(message.from_user.id,
message.from_user.username, 0)
        if stat == 'rules':
            await message.answer('Тут правила',
reply_markup=rules_mkp())
        elif stat == 'ban':
            await message.answer('Вы заблокированы')
        elif stat == 'ok':
            await message.answer('Вы в меню',
reply_markup=menu_mkp())
        else:
            captcha = Captcha()
            captcha.register_handlers(dp)

            await message.answer(
                captcha.get_caption(),
                reply_markup=captcha.get_captcha_keyboard()
            )

```

Функція `startcmd` відповідає за обробку команди `/start`. Спочатку вона перевіряє статус користувача, якщо статус є `"rules"`, то відбувається виведення правил за допомогою `message.answer` зі створенням `inline` клавіатури за допомогою `reply_markup=rules_mkr()`. Якщо статус є `"ban"`, то користувач буде повідомлений про своє блокування. Якщо статус є `"ok"`, то відбувається виведення меню за допомогою `message.answer` зі створенням `inline` клавіатури за допомогою `reply_markup=menu_mkr()`. Якщо статус не підходить ні до одного з вищезазначених, то створюється об'єкт `Captcha`, він реєструється з `dp` та відбувається відправка повідомлення з підписом та клавіатурою за допомогою `message.answer` та методів класу `Captcha`.

Функція `idcmd` відповідає за обробку команди `/id`. Вона відправляє повідомлення з інформацією про ID користувача.

Функції `rulesokcall` та `rulesnocall` відповідають за обробку зворотніх викликів. Якщо користувач погоджується з правилами, то його статус змінюється на `"ok"` за допомогою `db.change_status`, і відбувається виведення меню за допомогою `message.answer` зі створенням `inline` клавіатури за допомогою `reply_markup = menu_mkr()`. Якщо користувач не погоджується з правилами, то його статус змінюється на `"ban"` за допомогою `db.change_status`, і відбувається повідомлення про його блокування за допомогою `message.answer`.

Робота відбувається за рахунок файлу `main.py`. В якому відбувається підключення всіх модулів та запуск самого бота, через створену його архітектуру.

Нижче наведений приклад файлової структури телеграм бота:

```

├── images
├── start.py
├── config.py
├── box.py
├── captcha.py
└── admin.py

```

- |— catalog.py
- |— createtables.py
- |— database.py
- |— functions.py
- |— main.py
- |— murkups.py
- |— payments.py
- |— states.py
- |— support.py
- |— userfaq.py
- |— usermenu.py

Наведемо опис структури файлів:

- images: Директорія, що містить зображення або медіафайли, які використовуються в проекті;
- start.py: Файл, який містить код для привітання та перевірки користувача;
- config.py: Файл з конфігураційними налаштуваннями, такими як ключі API, налаштування бази даних тощо;
- box.py: Файл, що містить класи або функції, пов'язані з операціями або функціоналом коробки або контейнера;
- captcha.py: Файл, що містить класи або функції для генерації та перевірки капчі;
- admin.py: Файл, який містить класи або функції, пов'язані з функціоналом адміністратора або керуванням системою;
- catalog.py: Файл, що містить класи або функції для роботи з каталогом або списком товарів;
- createtables.py: Файл, що містить код для створення таблиць або схеми бази даних;

- database.py: Файл, що містить класи або функції для взаємодії з базою даних;
- functions.py: Файл, що містить різні корисні функції, які використовуються в проекті;
- main.py: Файл, який містить код для початкової ініціалізації і запуску програми або бота;
- markups.py: Файл, що містить класи або функції для створення розмітки або клавiатур для інтерфейсу користувача;
- payments.py: Файл, що містить класи або функції, пов'язані з оплатою або платіжними системами;
- states.py: Файл, що містить класи або функції для визначення та керування станами або режимами програми або бота;
- support.py: Файл, що містить класи або функції для обробки запитів або звернень користувачів до служби підтримки або адміністраторів;
- userfaq.py: Файл, що містить класи або функції для обробки запитів або надання інформації користувачам з питань FAQ (часто задаваних питань);
- usermenu.py: Файл, що містить класи або функції для реалізації меню користувача та взаємодії з різними опціями або функціями в системі.

Кожен файл відповідає за певну функціональну частину системи та допомагає досягти мети розробки.

3.1.2 Розробка компонентів моделі авторизації користувача

В даному прикладі використовується як спосіб реєстрації проходження капчі, нижче буде приведено приклад роботи (рис. 2.1) та код з описом його роботи

Код реалізації капчі [8]:

```

from typing import Dict
from aiogram import types, Dispatcher
from aiogram.types import InlineKeyboardButton,
InlineKeyboardMarkup
from aiogram.utils.exceptions import MessageNotModified
import random
from markups import rules_mkp

class Captcha:
    captcha_id = 0
    passed_captcha_users = set()

    def __init__(self, choices: Dict[str, str] = None) -> None:
        if choices and isinstance(choices, dict):
            self.choices = choices
        else:
            self.choices = {
                "яблоко": "🍏",
                "автомобіль": "🚗",
                "собаку": "🐶",
                "дерево": "🌳",
                "райдугу": "🌈",
                "банан": "🍌",
            }
            self.correct_choice =
random.choice(list(self.choices.keys()))

```

Наведений код визначає клас `Captcha`, який використовується для створення капчі (захисного завдання) в телеграм боті. Давайте розглянемо роботу коду поетапно:

- клас `Captcha` має декілька статичних змінних: `captcha_id` і `passed_captcha_users`. `captcha_id` використовується для створення унікальних ідентифікаторів капчі, а `passed_captcha_users` містить набір ідентифікаторів користувачів, які успішно пройшли капчу;
- у конструкторі класу `Captcha` виконується ініціалізація об'єкту. Якщо передано словник `choices`, то він використовується як набір варіантів вибору капчі, інакше використовується вбудований набір. Також обирається правильний варіант капчі з допомогою `random.choice`;
- метод `get_captcha_keyboard` створює і повертає клавіатуру з варіантами вибору капчі. Кнопки містять емодзі, які представляють різні варіанти капчі;
- метод `get_caption` повертає текст, який показується користувачеві разом з капчею. В цьому тексті вказується правильний варіант вибору;
- метод `captcha_choice_handler` викликається при виборі користувачем одного з варіантів капчі. Якщо вибраний варіант правильний, встановлюється прапор `captcha_passed` у значення `True` і ідентифікатор користувача додається до `passed_captcha_users`. Потім повідомлення редагується для показу результату капчі і викликається метод `answer` для надання користувачеві додаткової інформації;
- метод `register_handlers` реєструє обробники подій для вибору капчі. Використовується метод `dp.register_callback_query_handler`, який визиває `captcha_choice_handler` при співпадінні перших символів `callback_data`, що починаються з `self.callback_name`;
- у коді також імпортуються необхідні залежності, такі як `types`, `Dispatcher`, `InlineKeyboardButton`, `InlineKeyboardMarkup` та `MessageNotModified` з бібліотеки `aiogram`. Крім цього, імпортується `random` для вибору випадкового варіанту капчі, а також імпортуються інші модулі з проекту, такі як `markups`;
- цей код використовується для створення капчі, її відображення користувача та обробки вибору користувачем правильного варіанту. Капча

використовується для перевірки, чи є користувач реальним людиною, а не ботом, і дає доступ до додаткових функцій боту для користувачів, які успішно пройшли капчу.

3.2 Розробка компонентів моделі інтерфейсу

3.2.1 Розробка компонентів користувацького інтерфейсу

Розробка компонентів моделі користувацького інтерфейсу включає створення та налаштування різних елементів, які взаємодіють з користувачем у боті. Основні компоненти моделі користувацького інтерфейсу включають: клавіатури, меню, повідомлення, інлайн – клавіатури та робота з товаром.

Після проходження реєстрації користувач попадає до головного меню бота (рис.3.2).

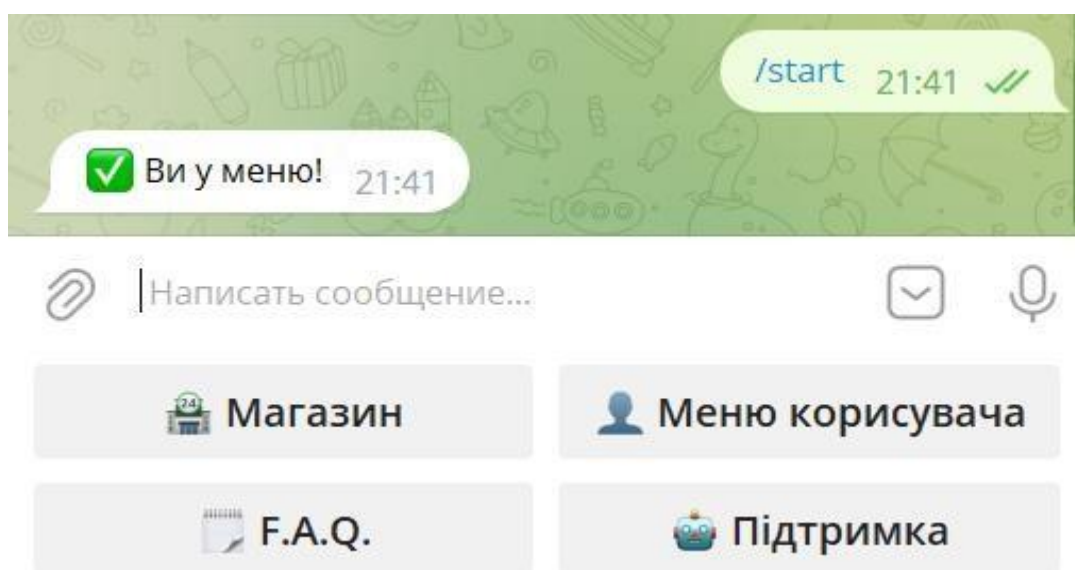


Рисунок 3.2 – Інтерфейс користувача головного меню

Меню користувача (Рисунок 3.3) виглядає з функцій для перегляду товару «Магазин», з правил які потрібно дотримуватись, а саме їх можливо переглянути у розділі «F.A.Q.». Розділ «Підтримка» служить для зв'язку з адміністратором магазину, тобто людина може задати питання людині через бота та отримати у ньому відповідь. Розділ «Меню користувача» складається з декількох додаткових розділів та функціоналу (рис. 3.3).

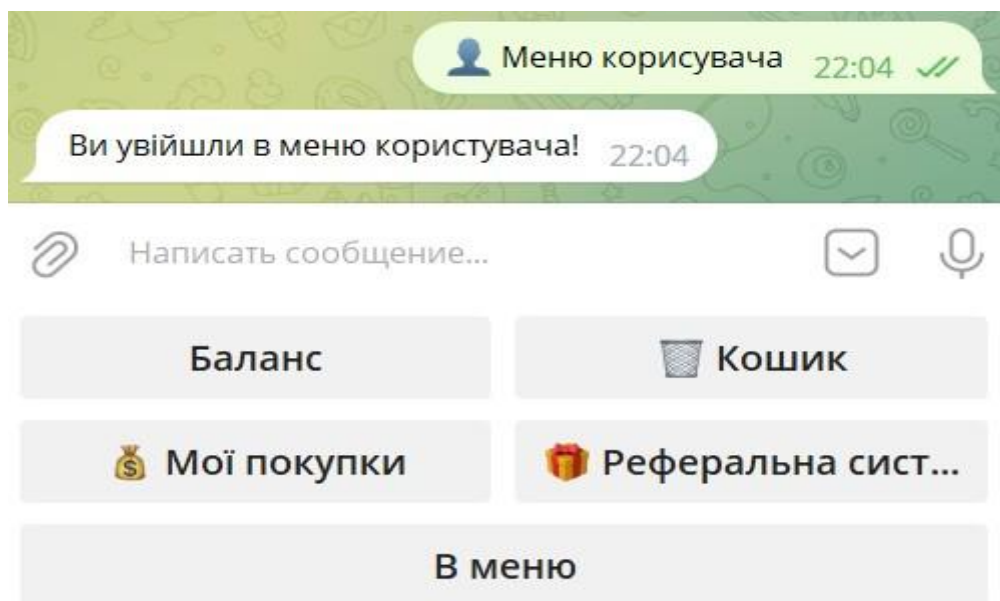


Рисунок 3.3 – Меню користувача

Меню користувача складається з п'яти елементів (Рисунок 3.4). Кнопка «Баланс» відповідає за відображення балансу та можливість його поповнення. У розділі «Кошик» користувач після вибору товару продовжує оформлення замовлення та оплату його. Після чого замовлення приходить адміністратору.

Функція «Мої покупки» відображає замовлення користувача, які вже були виконанні та які ще не виконанні.

Реферальна система відповідає за знижку для покупки. Реферальний код видається адміністратором або в розіграші. Кнопка «В меню» відповідає за повернення в головне меню.

Код файлу з клавіатурами для головного меню та меню користувача.

```

14 def menu_mkp():
15     mkr = types.ReplyKeyboardMarkup(resize_keyboard=True)
16
17     btn1 = types.KeyboardButton('🏠 Магазин')
18     btn2 = types.KeyboardButton('👤 Меню користувача')
19     btn3 = types.KeyboardButton('❏ F.A.Q.')
20     btn4 = types.KeyboardButton('🛎 Підтримка')
21     mkr.add(btn1, btn2).add(btn3, btn4)
22     return mkr
23
24 def usermenu_mkp():
25     mkr = types.ReplyKeyboardMarkup(resize_keyboard=True)
26
27     btn1 = types.KeyboardButton('Баланс')
28     btn2 = types.KeyboardButton('🛒 Кошик')
29     btn3 = types.KeyboardButton('💰 Мої покупки')
30     btn4 = types.KeyboardButton('🎁 Реферальна система')
31     btn5 = types.KeyboardButton('🔙 У меню')
32     mkr.add(btn1, btn2).add(btn3, btn4).add(btn5)
33     return mkr

```

Рисунок 3.4 – Створення клавіатури для користувача.

Файл markups.py:

```

def menu_mkp():
    mkr = types.ReplyKeyboardMarkup(resize_keyboard=True)

    btn1 = types.KeyboardButton('❏ Магазин')
    btn2 = types.KeyboardButton('❏ Меню користувача')
    btn3 = types.KeyboardButton('❏ F.A.Q.')
    btn4 = types.KeyboardButton('🛎 Підтримка')
    mkr.add(btn1, btn2).add(btn3, btn4)
    return mkr

```

```
def usermenu_mkp():
    mkr = types.ReplyKeyboardMarkup(resize_keyboard=True)

    btn1 = types.KeyboardButton('Баланс')
    btn2 = types.KeyboardButton('🛒 Кошик')
    btn3 = types.KeyboardButton('🛒 Мої покупки')
    btn4 = types.KeyboardButton('📄 Реферальна система')
    btn5 = types.KeyboardButton('У меню')
    mkr.add(btn1, btn2).add(btn3, btn4).add(btn5)
    return mkr
```

Функціонал спілкування користувача з службою підтримки. Код інтерфейсу служби підтримки support.py [13]:

```
@dp.message_handler(text='📄 Підтримка')
async def suppmg(message: types.Message):
    if db.check_ban(message.from_user.id):
        await message.answer(Введіть своє питання та ми як
        омога скоріше вам відповімо:', reply_markup=cancel_mkr())
        await SuppUser.UserId.set()

@dp.callback_query_handler(text='cancel', state=SuppUser.UserId)
async def cancelstatesuppuser(call: types.CallbackQuery, state:
FSMContext):
    await call.message.delete_reply_markup()
    await call.message.answer('Отменено. Ви були повернуті до
меню!', reply_markup=menu_mkr())
```

Зараз буде наведений приклад реалізації кнопки «F.A.Q.» та її функціонал.

Файл userfaq.py:

```
@dp.message_handler(text='📄 F.A.Q.')
```

```

async def faqmsg(message: types.Message):
    if db.check_ban(message.from_user.id):
        await message.answer('Оберіть розділ для ознайомлення:',
reply_markup=get_faq_user())

@dp.callback_query_handler(text='tomenu')
async def tomenucall(call: types.CallbackQuery):
    if db.check_ban(call.from_user.id):
        await call.message.delete()
        await call.message.answer('Ви були повернені до меню',
reply_markup=menu_mkp())

@dp.message_handler(text='В меню')
async def tomenumsg(message: types.Message):
    if db.check_ban(message.from_user.id):
        await message.answer('Ви були повернені до головного
меню!', reply_markup=menu_mkp())

```

Даний код реалізує функціонал, пов'язаний з відображенням Часто Задаваних Питань (FAQ) в телеграм-боті. Основні функції коду такі:

- функція `faqmsg`: Ця функція обробляє повідомлення з текстом "❓ F.A.Q.". Якщо користувач не заблокований (перевірка здійснюється за допомогою функції `check_ban`), то бот відправляє повідомлення з запитом вибрати розділ FAQ, використовуючи клавіатуру, яка формується за допомогою функції `get_faq_user`;
- функція `tomenucall`: Ця функція обробляє `callback`-запит з текстом "tomenu". Якщо користувач не заблокований, то бот видаляє поточне повідомлення і відправляє повідомлення з повідомленням "Ви були повернені до меню", використовуючи клавіатуру меню `menu_mkp`;
- функція `tomenumsg`: Ця функція обробляє повідомлення з текстом "В меню". Якщо користувач не заблокований, то бот відправляє повідомлення

з текстом "Ви були повернені до головного меню!", використовуючи клавіатуру меню `menu_mkr`;

- функція `getfaqcall`: Ця функція обробляє `callback`-запити з текстом, що містить `"getfaq_"`. Якщо користувач не заблокований, то вона отримує ідентифікатор питання з `callback`-даних, отримує відповідну інформацію з бази даних (функція `get_faq`) і редагує початкове повідомлення, відображаючи текст питання і його відповіді. При цьому також відправляється клавіатура з розділами FAQ.

Загалом, цей код дозволяє користувачеві переглядати розділи і запитання FAQ у телеграм-боті. Якщо користувач не заблокований, то він може відкривати розділи FAQ, переглядати конкретні запитання та отримувати відповіді на них. При цьому використовуються клавіатури для зручності навігації користувача. Якщо користувач заблокований, то йому надсилаються повідомлення про блокування і неможливість отримання доступу до функціоналу FAQ.

Зазначений функціонал допомагає користувачам знайти відповіді на питання та ознайомитись з інформацією, що стосується бота чи певних послуг, що надаються. Крім того, використання клавіатур та `callback`-запитів дозволяє забезпечити зручну та інтерактивну взаємодію з користувачами.

Весь цей функціонал реалізований у контексті телеграм-бота за допомогою бібліотеки `aiogram`, яка надає зручні інструменти для створення ботів у Telegram.

Компонент `catalog.py` являє собою функціонал для роботи з товаром для звичайного користувача.

Код компоненту `catalog.py`:

```
@dp.message_handler(text='☐ Магазин')
async def shopmsg(message: types.Message):
    if db.check_ban(message.from_user.id):
```

```

        await message.answer('Выберите категорию:',
reply_markup=get_categories_user())

```

```

@dp.callback_query_handler(text='toshop')
async def toshopcall(call: types.CallbackQuery):
    if db.check_ban(call.from_user.id):
        await call.message.delete()
        await call.message.answer('Виберіть підкатегорію:',
reply_markup=get_categories_user())

```

```

@dp.callback_query_handler(text_contains='usercat_')
async def usercatcall(call: types.CallbackQuery):
    if db.check_ban(call.from_user.id):
        catid = call.data.split('_')[1]
        await call.message.delete()
        await call.message.answer('Виберіть підкатегорію:',
reply_markup=get_subcategories_user(int(catid)))

```

Даний код містить функціонал, пов'язаний з взаємодією користувачів з магазином в телеграм-боті. Основні функції цього коду включають:

- `shopmsg`: Ця функція відповідає на повідомлення з текстом "👑 Магазин". Якщо користувач не заблокований (`db.check_ban`), то відправляється повідомлення з запитом вибору категорії магазину, використовуючи клавіатуру `get_categories_user()`;
- `toshopcall`: Ця функція обробляє натискання на кнопку з текстом "toshop". Якщо користувач не заблокований (`db.check_ban`), то попереднє повідомлення видаляється, а потім надсилається повідомлення з запитом вибору підкатегорії магазину, використовуючи клавіатуру `get_categories_user()`;
- `usercatcall`: Ця функція обробляє натискання на кнопки з префіксом "usercat_". Якщо користувач не заблокований (`db.check_ban`), то витягується ідентифікатор категорії, попереднє повідомлення видаляється, а

потім надсилається повідомлення з запитом вибору підкатегорії магазину, використовуючи клавіатуру `get_subcategories_user()`;

- `usersubcatcall`: Ця функція обробляє натискання на кнопки з префіксом `"usersubcat_"`. Якщо користувач не заблокований (`db.check_ban`), то витягується ідентифікатор підкатегорії. Якщо у цій підкатегорії відсутні товари, то надсилається повідомлення з повідомленням про відсутність товарів. В іншому випадку попереднє повідомлення видаляється, а потім викликається функція `send_good()` для надсилання товарів користувачеві;

- `catbackcall`: Ця функція обробляє натискання на кнопки з префіксом `"catback_"`. Якщо користувач не заблокований (`db.check_ban`), то витягуються ідентифікатор підкатегорії та кроку. Попереднє повідомлення видаляється, а потім викликається функція `send_good()` зі зменшеним значенням кроку, щоб відобразити попередні товари в межах підкатегорії;

- `catnextcall`: Ця функція обробляє натискання на кнопки з префіксом `"catnext_"`. Якщо користувач не заблокований (`db.check_ban`), то витягуються ідентифікатор підкатегорії та кроку. Попереднє повідомлення видаляється, а потім викликається функція `send_good()` зі збільшеним значенням кроку, щоб відобразити наступні товари в межах підкатегорії;

- `toboxcall`: Ця функція обробляє натискання на кнопки з префіксом `"tobox_"`. Якщо користувач не заблокований (`db.check_ban`), то витягується ідентифікатор товару. Запис про доданий товар до кошика додається у базу даних (`db.add_box`). Потім надсилається повідомлення з підтвердженням додавання товару до кошика.

Загалом, цей код відповідає за обробку повідомлень та натискань кнопок, пов'язаних з функціоналом магазину. Він використовує базу даних, клавіатури та функції для отримання категорій, підкатегорій та відправки товарів користувачам.

Наступний функціонал був виконаний на натискання кнопки « Меню користувача», ця клавіша відповідає за перехід до меню у якому користувач

може переглянути різноманітну всю інформацію що йому необхідна для використання бота.

Робота кнопки "☐ Меню користувача" полягає в відображення підменю, навігація між різними опціями, виконання дій на основі вибору користувача. Отже, кнопка дозволяє користувачам отримати доступ до функціоналу, пов'язаного з їх меню користувача.

Після використання кнопки «☐ Меню користувача» користувач попадає до свого основного меню, реалізація якого зображена на Рисунку 3.6. Далі задачею було реалізувати це меню, зображення цього меню було наведено на Рисунку 3.4.

Реалізація створення самого меню було наведено на Рисунку 3.5. Нижче буде наведено реалізацію функціоналу клавіші «Баланс».

```

10 @dp.message_handler(text='👤 Меню користувача')
11 async def usermenumsg(message: types.Message):
12     if db.check_ban(message.from_user.id):
13         await message.answer('Ви увійшли в меню користувача!', reply_markup=usermenu_mkp())
14 
```

Рисунок 3.5 – Реалізація дії кнопки «☐ Меню користувача»

Код компоненту usermenu.py:

```

17 @dp.message_handler(text='Баланс')
18 async def balancemsg(message: types.Message):
19     if db.check_ban(message.from_user.id):
20         balance = db.get_balance(message.from_user.id)
21         ref_balance = db.get_refbalance(message.from_user.id)
22         mkp = types.InlineKeyboardMarkup()
23         btn1 = types.InlineKeyboardButton('Поповнити', callback_data='deposit')
24         btn2 = types.InlineKeyboardButton('Вивести', callback_data='withdraw')
25         mkp.add(btn1, btn2)
26         await message.answer(f'Ваш баланс</b>: <code>{float(balance)}</code> $<n>Заработано з рефералів</b>: <code>{float(ref_balance)}</code>', reply_markup=mkp)

```

Рисунок 3.6 – Реалізація функціоналу клавіші «☐ Кошик»

Опис роботи коду:

– бот перевіряє, чи користувач не заблокований (db.check_ban(message.from_user.id)). Якщо користувач заблокований, подальша обробка повідомлення не відбувається;

- З бази даних отримуються значення балансу користувача (balance) і сума, зароблена з рефералів (ref_balance);
- Створюється клавіатура типу InlineKeyboardMarkup з двома кнопками: "Поповнити" і "Вивести". Кнопки мають відповідні callback_data для подальшої обробки;
- бот надсилає відповідь користувачу, в якій відображається інформація про баланс користувача і суму, зароблену з рефералів. Ця інформація відображається у вигляді HTML-розмітки, оформленої тегом . Значення балансу і суми відображаються у вигляді <code>;
- клавіатура mkr додається до відповіді, щоб користувач мав можливість взаємодіяти з кнопками "Поповнити" і "Вивести".

Далі буде показано реалізацію клавіши « Кошик».

Код компоненту box.py:

```
@dp.message_handler(text='🛒 Кошик')
async def boxmsg(message: types.Message):
    box = db.get_box(message.from_user.id)
    if len(box) == 0:
        await message.answer('Кошик пустий. Перейдіть до магазину
щоб обрати товар.')
    else:
        step = 1
        price = 0
        text = ''
        for i in box:
            try:
                good_info = db.get_good_info(i[0])
                text = f'{text}{step}.      <b>Товар</b>:
<code>{good_info[0]}</code> | <b>Ціна</b>: {good_info[1]} $\\n '
                price = price+float(good_info[1])
                step=step+1
            except:
```

```

        db.del_box_good(int(i[0]))
        text=f'Ваша                корзина:\n{text}\n\n<b>Итого</b>:
<code>{price}</code> <b>$</b>'
        mkp = types.InlineKeyboardMarkup()
        btn1 = types.InlineKeyboardButton('Оформити замовлення',
callback_data='goorder')
        btn2  =  types.InlineKeyboardButton('Очистити кошик',
callback_data='boxclear')
        mkp.add(btn1).add(btn2)
        await message.answer(text, reply_markup=mkp)

```

Цей код є обробником повідомлень з текстом "🛒 Кошик". Коли користувач надсилає повідомлення з таким текстом, бот викликає цю функцію-обробник.

Опис роботи коду:

- бот отримує вміст кошика для користувача з бази даних (db.get_box(message.from_user.id)) ;
- якщо кошик порожній (не містить жодного товару), бот надсилає повідомлення з текстом "Кошик пустий. Перейдіть до магазину щоб обрати товар." ;
- у протилежному випадку, бот формує текст повідомлення зі списком товарів у кошику. Для кожного товару отримується інформація про назву (good_info[0]) та ціну (good_info[1]), і ця інформація додається до тексту повідомлення у вигляді HTML-розмітки. Також обчислюється загальна вартість товарів у кошику;
- формується остаточний текст повідомлення, в якому відображаються всі товари у кошику разом з їх цінами і загальна сума. Ця інформація відображається у вигляді HTML-розмітки, оформленої тегами і <code>;

- створюється клавіатура типу `InlineKeyboardMarkup` з двома кнопками: "Оформити замовлення" і "Очистити кошик". Кнопки мають відповідні `callback_data` для подальшої обробки;
- бот надсилає повідомлення зі сформованим текстом кошика та клавіатурою для взаємодії з кнопками.

Далі було розроблено функціонал клавiши «`☐` Мої покупки», її основною задачею було те щоб при її використанні користувачу показувало його виконанні замовлення чи якщо є активні замовлення. Нижче наведено код реалізації.

Код компоненту `box.py`:

```
@dp.message_handler(text='☐ Мої покупки')
async def myordersmsg(message: types.Message):
    if db.check_ban(message.from_user.id):
        active_order = db.get_actorders(message.from_user.id)
        end_orders = db.get_endorders(message.from_user.id)
        if len(active_order) == 0:
            await message.answer('Активних замовлень немає!')
        else:
            text = ''
            for i in active_order:
                text=f'{text}-----\n'
                step = 1
                for a in i:
                    try:
                        text=f'{text}Загалом: {float(a)}\n $'
                    except:
                        text=f'{text}{step}.{i[0]}\n'
                        step=step+1
            await message.answer(f'Активні замовлення:\n{text}')

    if len(end_orders) == 0:
```

```

        await message.answer('Закінчених замовлень не
знайдено!')
    else:
        text = ''
        for i in end_orders:
            text=f'{text}-----\n'
            step = 1
            for a in i:
                try:
                    text=f'{text}Сумма: {float(a)}\n'
                except:
                    text=f'{text}{step}.{i[0]}\n'
                step=step+1
        await message.answer(f'Остані 2 завершених
замовлення:\n{text}')

```

Цей код є обробником повідомлень з текстом "□ Мої покупки". Коли користувач надсилає повідомлення з таким текстом, бот викликає цю функцію-обробник.

Опис роботи коду:

- бот перевіряє, чи є користувач заблокованим (db.check_ban(message.from_user.id));
- отримується список активних замовлень для користувача (db.get_actorders(message.from_user.id));
- якщо список активних замовлень порожній, бот надсилає повідомлення з текстом "Активних замовлень немає!";
- у протилежному випадку, формується текст повідомлення, в якому відображаються активні замовлення. Для кожного замовлення виводиться номер (step) та його елементи. Якщо елемент є числом, він трактується як сума замовлення, інакше вважається, що це назва товару. Ця інформація додається до тексту повідомлення;
- бот надсилає повідомлення зі списком активних замовлень;

- отримується список останніх 2 завершених замовлень для користувача (`db.get_endorders(message.from_user.id)`);
- якщо список закінчених замовлень порожній, бот надсилає повідомлення з текстом "Закінчених замовлень не знайдено!";
- у протилежному випадку, формується текст повідомлення, в якому відображаються останні 2 завершених замовлення. Для кожного замовлення виводиться номер (step) та його елементи. Якщо елемент є числом, він трактується як сума замовлення, інакше вважається, що це назва товару. Ця інформація додається до тексту повідомлення;
- бот надсилає повідомлення зі списком останніх 2 завершених замовлень.

Та остання кнопка у цьому меню відповідає за реферальну систему (рис. 3.7). Її задачею було отримати посилання на бота від вашого імені. Якщо людина перейшла у бота по вашому посиланню та здійснила покупку тоді вам надається знижка.

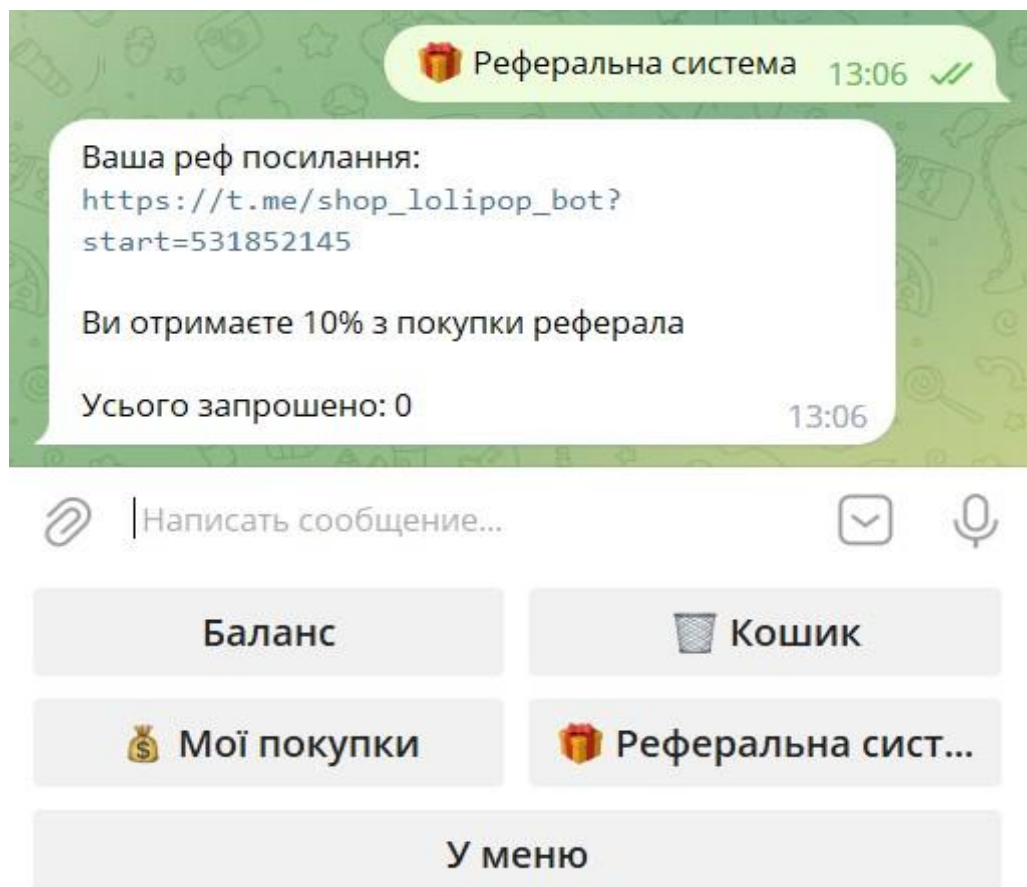


Рисунок 3.7 – Отримання реферального посилання

Код компоненту usermenu.py:

```
@dp.message_handler(text='□ Реферальна система')
async def refsystemmsg(message: types.Message):
    if db.check_ban(message.from_user.id):
        ref = db.get_refproc()
        count_ref = db.get_all_refs(message.from_user.id)
        await message.answer(f'Ваша реф посилання:
<code>{url}?start={message.from_user.id}</code>\n\nВи отримуєте {ref}%
з покупки реферала\n\nУсього запрошено: {len(count_ref)}')
```

Уся інформація про запрошених рефералів зберігається у базі даних. Тому при використанні цієї функції інформація береться з бази.

3.2.2 Розробка компонентів адміністраторського інтерфейсу

Головною задачею при розробці адміністраторського інтерфейсу була розробка функцій додавання та редагування товару. Окрім цього адміністраторська панель має різноманітний функціонал, такий як:

- замовлення – розділ відповідає за перегляд активних замовлень;
- користувачі – за допомогою цього розділу можна передивитися всіх користувачів та також заблокувати чи розблокувати їх;
- налаштування – в цьому розділі можливе редагування розділів F.A.Q;
- зміна реф – цей розділ відповідає за зміну проценту за приведенного користувача;
- режим покупця – цей функціонал для того якщо адміністратор хоче подивитися зміни від лиця користувача чи хоче щось замовити.

Для створення адміністраторської панелі було використано ід телеграм користувача який виконує обов'язки адміністратора (рис. 3.8). Для того щоб увійти в адмін – панель потрібно прописати команду /admin.

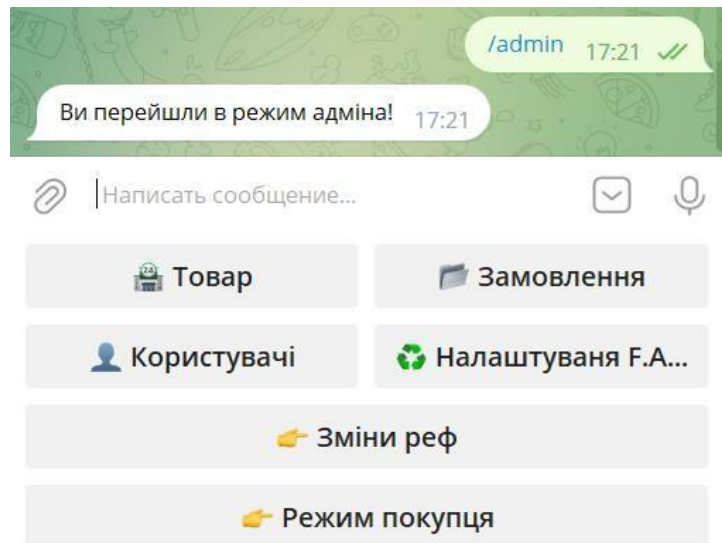


Рисунок 3.8 – Вигляд адмін – панелі

Після реалізації переходу до адмін – панелі було перейдено до розробки функціоналу клавіатури та її функціоналу яка була наведена на малюнку 3.9. Загалом спочатку було старено спочатку клавіатуру для адмін - панелі.

Код компоненту admin.py :

```

15 @dp.message_handler(commands='admin')
16 async def admincmd(message: types.Message):
17     if message.from_user.id in admins:
18         await message.answer('Ви перейшли в режим адміна!', reply_markup=admin_mkp())

```

Рисунок 3.9 – Реалізація переходу до адмін – панелі

Далі було реалізовано функціонал клавiши « Товар», створення якої зображено на малюнку 3.10. Вона відповідає за додавання категорій, підкатегорій, товару та редагування всіх цих перерахованих пунктів. Нижче будуть зображені підменю кнопки « Товар» (рис 3.11).

```

43 def admin_mkp():
44     mkp = types.ReplyKeyboardMarkup(resize_keyboard=True)
45
46     btn1 = types.KeyboardButton('👑 Товар')
47     btn2 = types.KeyboardButton('📦 Замовлення')
48     btn3 = types.KeyboardButton('👤 Користувачі')
49     btn4 = types.KeyboardButton('🌱 Налаштування F.A.Q')
50     btn5 = types.KeyboardButton('🔧 Зміни реф')
51     btn6 = types.KeyboardButton('🔧 Режим покупця')
52     mkp.add(btn1, btn2).add(btn3, btn4).add(btn5).add(btn6)
53     return mkp

```

Рисунок 3.10 – Створення клавіатури адмін – панелі

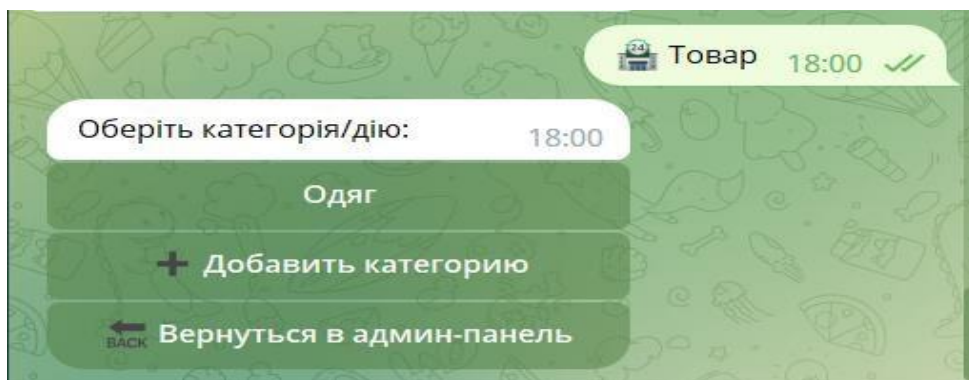


Рисунок 3.11 – Перше підменю

Код компоненту functions.py:

Створення клавіатури за допомогою функції:

```

def get_categories_admin():
    cat_list = db.get_all_cat()
    mkp = types.InlineKeyboardMarkup()
    for i in cat_list:

```

```

        mkr.add(types.InlineKeyboardButton(i[1],
callback_data=f'admincat_{i[0]}'))
        mkr.add(types.InlineKeyboardButton('✚ Додати катеогрію
категорию', callback_data='addcat'))
        mkr.add(types.InlineKeyboardButton('◻ Повернутися в адмін-
панель', callback_data='admin'))
    return mkr

```

Реалізація її функціоналу з додаванням катеогрії:

```

@dp.message_handler(text='◻ Товар')
async def ordersmsg(message: types.Message):
    await message.answer('Оберіть категорія/дію:',
reply_markup=get_categories_admin())

@dp.callback_query_handler(text='addcat')
async def addcatcall(call: types.CallbackQuery):
    await call.message.delete()
    await call.message.answer('Введіть назву катеогрії:',
reply_markup=cancel_mkr())
    await AddCat.CatName.set()

@dp.message_handler(state=AddCat.CatName)
async def addcatcatnamemsg(message: types.Message, state:
FSMContext):
    db.add_cat(message.text)
    await message.answer('Катеогрія створена! Вы были повернені в
адмін-панель', reply_markup=admin_mkr())
    await state.finish()

```

Далі було реалізовано появу клавіатури під-категорій та їх редагування чи додавання. При виборі категорії це повідомлення видаляється та з'являється наступне меню (рис. 12) . Якщо обрати дію повернутися в меню також видаляється те меню в якому було обрана дія. Це відбувається за для того щоб не засмічувати не потрібними повідомлення вікно чату з ботом.

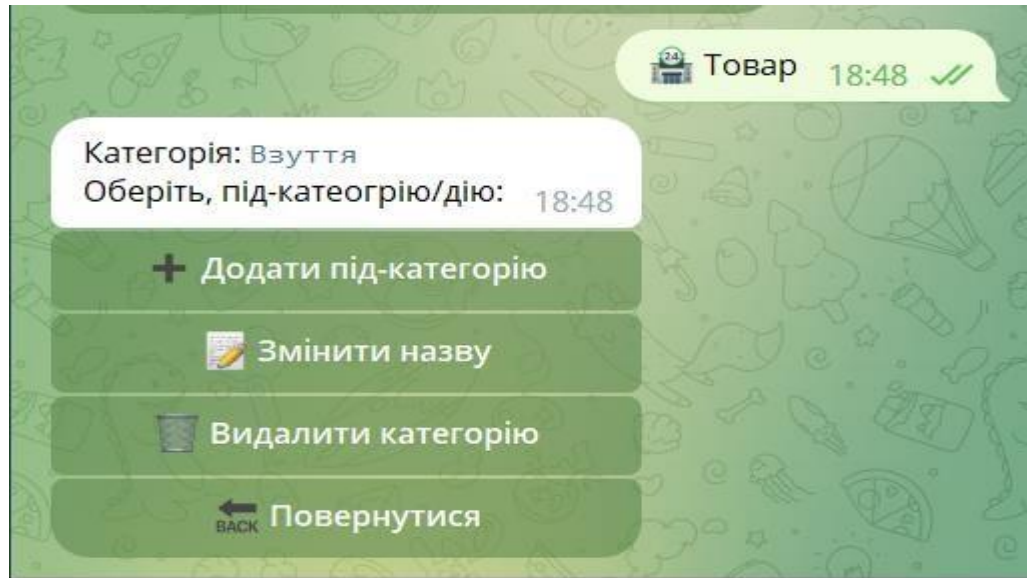


Рисунок 3.12 – Меню під-категорій

Код компоненту functions.py (реалізація меню на рис. 3. 13):

Створення клавіатури за допомогою функції:

```
def get_subcategories_admin(cat_id):
    subcat_list = db.get_subcat(cat_id)
    mkp = types.InlineKeyboardMarkup()
    for i in subcat_list:
        mkp.add(types.InlineKeyboardButton(i[1],
        callback_data=f'adminsubcat_{i[0]}'))
    mkp.add(types.InlineKeyboardButton('✚ Додати під-категорію',
    callback_data=f'addsubcat_{cat_id}'))
    mkp.add(types.InlineKeyboardButton('✎ Змінити назву',
    callback_data=f'changenamecat_{cat_id}'))
    mkp.add(types.InlineKeyboardButton('🗑 Видалити категорію',
    callback_data=f'delcat_{cat_id}'))
    mkp.add(types.InlineKeyboardButton('⬅ Повернутися',
    callback_data='products'))
```

```
return mkr
```

Код компоненту admin.py:

Реалізація її функціоналу з додаванням під - категорій:

```
@dp.callback_query_handler(text_contains='addsubcat_')
async def addsubcatcall(call: types.CallbackQuery, state:
FSMContext):
    cat_id = call.data.split('_')[1]
    await call.message.delete()
    await call.message.answer('Введіть назву підкатегорії',
reply_markup=cancel_mkr())
    await AddSubcat.SubcatName.set()
    async with state.proxy() as data:
        data['SubcatName'] = cat_id
```

Після додавання категорій було перейдено до розробки можливості додавання товару та його редагування. Додавання товару відбувається поступово. З першу відбувається додавання назви, потім опис, фото та ціна, наведено на рис. 3.13, рис 3.14, рис. 3.15, рис. 3.16.



Рисунок 3.13 – Додавання назви



Рисунок 3.14 – Додавання опису товару

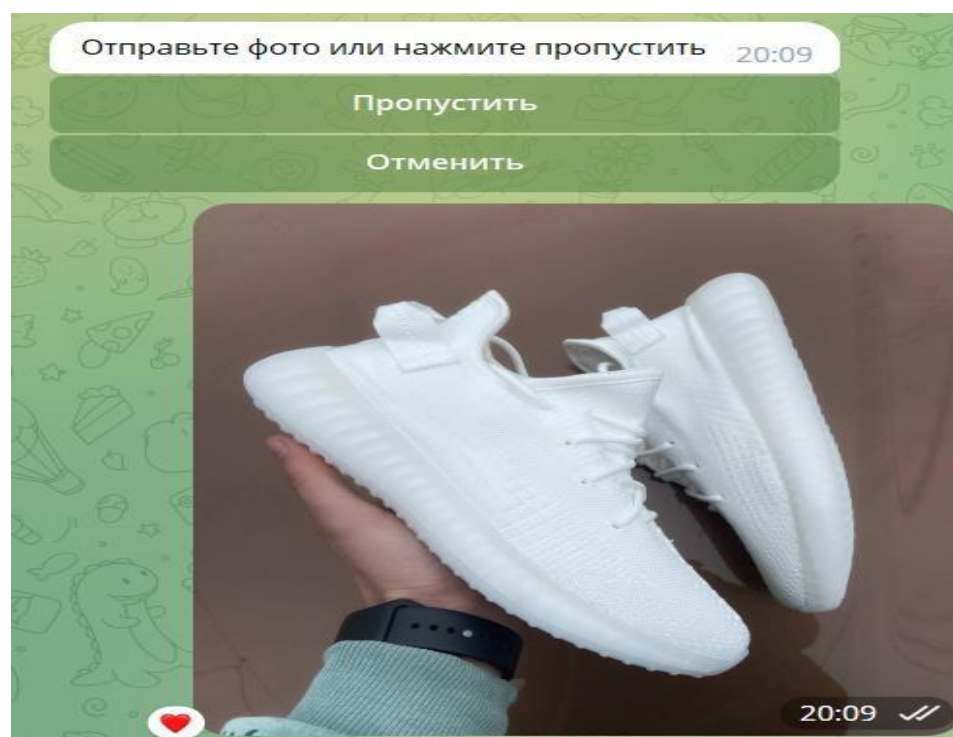


Рисунок 3.15 – Додавання фото товару

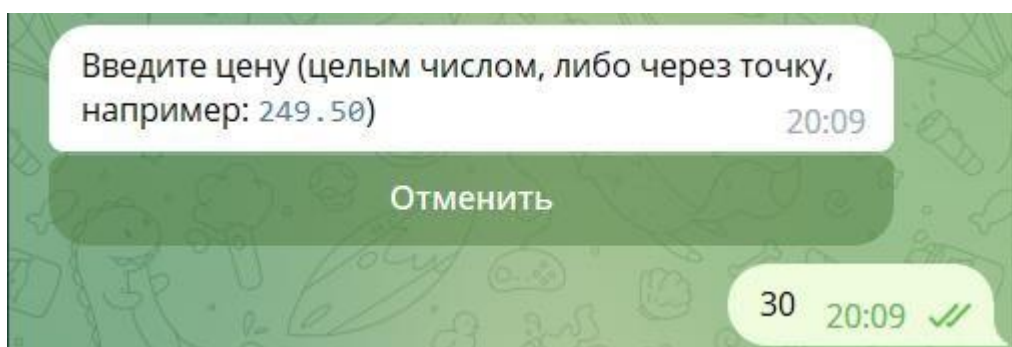


Рисунок 3.16 – Додавання ціни товару

Все це відбувається послідовно. Якщо ціна буде вказана не вірна то станеться помилка і бот попросить ще раз її ввести вірно. Після додавання товару він додається до своєї під – категорії, картка товару зображена на рис. 3.17.



Рисунок 3.17 – Картка товару

Реалізація додавання товару:

Додавання назви, опису товару, фото товару, ціни для товару буде вказано в додатку А (admin.py)

Результат готової картки та завершення її додавання

Код компоненту admin.py:

```
if data['Photo'] == 'None':
```

```

        await message.answer(f'Назва товару:
<code>{name}</code>\nОпис: <code>{description}</code>\nЦіна:
<code>{price}</code>', reply_markup=mkp)
    else:
        photo = data['Photo']
        await
message.answer_photo(open(f'images/{photo}', 'rb'),
caption=f'Назва товару: <code>{name}</code>\nОпис:
<code>{description}</code>\nЦіна: <code>{price}</code>',
reply_markup=mkp)

```

У результаті розробки функціоналу додавання товару також було розроблено додаткові inline кнопки для керування. Наприклад у меню між категоріями вони використовуються для переходу далі чи навпаки назад. При перегляді товару вони служать для перелистування товару, додавання його до кошику, видалення звідти або повернення до головного меню магазину. У дипломній роботі я показав основні функції для реалізації додавання товару. Далі було перейдено до розробки функціоналу та її оживлення кнопки «☐ Замовлення».

Код компоненту admin.py:

```

@dp.message_handler(text='<input type="checkbox"/> Замовлення')
async def ordersmsg(message: types.Message):
    orders = db.get_all_activeorders()
    if len(orders) == 0:
        await message.answer('Активних замовлень не виявлено')
    else:
        mkp = types.InlineKeyboardMarkup()
        for i in orders:
            mkp.add(types.InlineKeyboardButton(f'Активне
замовлення №{i[0]}', callback_data=f'orderadmin_{i[0]}'))

```

```
await message.answer('Список активных заказов:',
reply_markup=mkp)
```

Вище зазначеним кодом було реалізований функціонал клавiши «□ Замовлення». Її основна функція це передати список замовлень для їх підтверджень та зберігання виконаних, для того щоб можливо було відстежувати які були виконані, а які ще ні.

Після чого було реалізовано дії клавiши «👤 Користувачі» (рис.3.18). Вона дає змогу переглянути весь список користувачів та зробити розсилку. При перегляді списку можливо блокувати користувача.

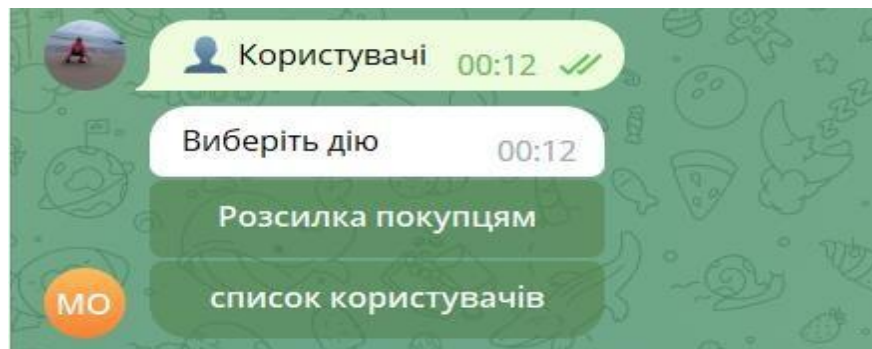


Рисунок 3.18 – Приклад активної кнопки «□ Користувачі»

В дному кодi буде наведено приклад реалізації реакції клавiши на її натискання, а також виконання розсилки та перегляду списку з його здатністю блокувати користувачів [10], [13].

Код компоненту admin.py:

```
@dp.callback_query_handler(text='rassilka')
async def rassilkacall(call: types.CallbackQuery):
    await call.message.delete()
    await call.message.answer('Введіть повідомлення для
розсилки:', reply_markup=cancel_mkp())
    await Rassilka.Text.set()
```

Після перейшов до розробки F.A.Q (рис3.19). Функціонал та завдання розділу F.A.Q. (Часто задавані питання) у телеграм-боті магазину наведено нижче.

Функціонал:

- користувач може відкрити розділ F.A.Q., натиснувши на кнопку "☐ F.A.Q." в меню;
- бот відображає список часто задаваних питань та відповідей;
- користувач може прокручувати список питань і відповідей, щоб знайти потрібну інформацію;
- при необхідності, користувач може відкрити додаткові розділи або підрозділи F.A.Q. для отримання більш детальної інформації.

Завдання:

- забезпечити доступ користувачів до інформації, яка часто запитується, і надати їм відповіді на популярні питання;
- зменшити кількість повторних запитів користувачів, шляхом надання чіткої та доступної інформації в розділі F.A.Q;
- забезпечити структуровану організацію питань та відповідей, щоб користувачам було легко знайти потрібну інформацію;
- забезпечити можливість розширення та додавання нових питань та відповідей в майбутньому, якщо з'являться нові запити користувачів.

Задачі розділу F.A.Q. в боті:

- підготування списку популярних питань та відповідей, які часто задаються користувачами;
- розробка логіки та функціоналу для відображення цих питань та відповідей у вигляді структурованого списку;
- надання можливості користувачам прокручувати список питань та відповідей та знаходити потрібну інформацію.

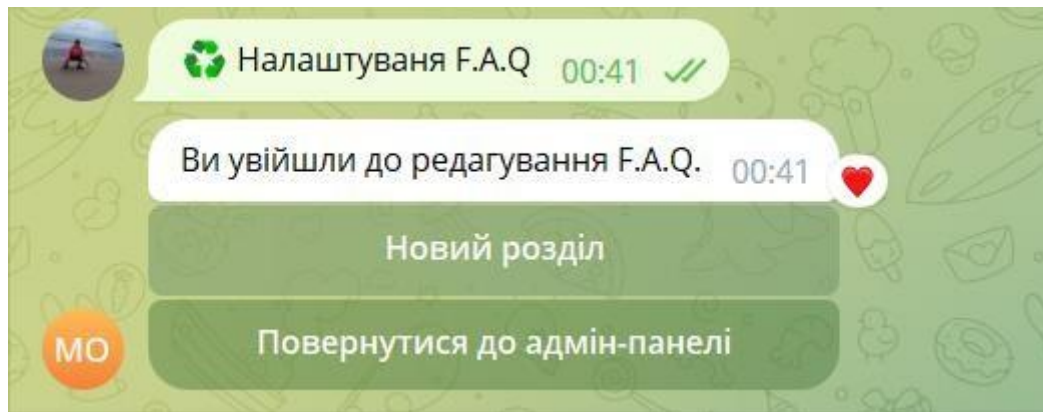


Рисунок 3.19 - F.A.Q

Код компоненту admin.py:

```
@dp.message_handler(text='□ Налаштування F.A.Q')
async def fqsetmsg(message: types.Message):
    await message.answer('Ви увійшли до редагування F.A.Q.',
reply_markup=get_faq_admin())

@dp.callback_query_handler(text='newfaq')
async def newfaqcall(call: types.CallbackQuery):
    await call.message.delete_reply_markup()
    await call.message.answer('Введіть назву розділу',
reply_markup=cancel_mkp())
    await NewFaq.Name.set()

@dp.message_handler(state=NewFaq.Name)
async def newfaqnamemsg(message: types.Message, state:
FSMContext):
    await message.answer(f'Добре, назва буде:
<code>{message.text}</code>')
```

У коді вказаному вище показано реалізацію F.A.Q. Також функції редагування цього розділу. Далі було реалізовано зміну реферального проценту (рис3.20).

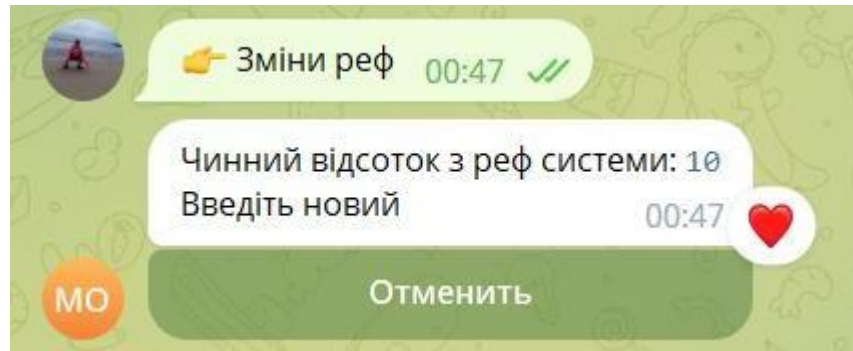


Рисунок 3.20 – Зміна реферального проценту

Код компоненту admin.py:

```
@dp.message_handler(state=ChangeRef.Ref)
async def changerefrefmsg(message: types.Message, state:
FSMContext):
    if message.text.isdigit():
        db.change_ref(message.text)
        await message.answer('Відсоток з реф системи успішно
змінено! Ви були повернуті до адмін-панелі', reply_markup=admin_mkp())
        await state.finish()
```

Даний фрагмент коду дає можливість редагувати відсоток від реферала. Тобто адміністратор може редагувати який відсоток користувач буде отримувати від його покупки відсоток знижки.

3.2.3 Розробка компонентів функціоналу зв'язку з підтримкою

Розділ "Підтримка" в телеграм-боті магазину відіграє важливу роль у забезпеченні зручності та задоволення користувачів. Задача цього розділу полягає у наданні швидкої та ефективної підтримки користувачам, вирішенні їх запитань, наданні необхідної інформації і допомозі у вирішенні проблем, пов'язаних з магазином.

Функціонал розділу "Підтримка" (рис 3.21, рис. 3.22, рис. 3.23) може включати наступні можливості:

- обробка запитів користувачів: Бот може приймати повідомлення, які надіслали користувачі з питаннями, проблемами або запитами щодо магазину. Це може включати запитання про товари, оплату, доставку, повернення товару, гарантійні зобов'язання тощо;
- відповіді на запити: за допомогою оператора підтримки відповідати на запити користувачів. Це може бути надання необхідної інформації, посилання на сторінки з детальними відомостями, інструкції щодо вирішення проблем, надання контактів для прямого зв'язку з оператором підтримки і т. д;
- обробка скарг і проблем: Користувачі можуть надсилати скарги або повідомлення про проблеми, які виникли під час взаємодії з магазином.

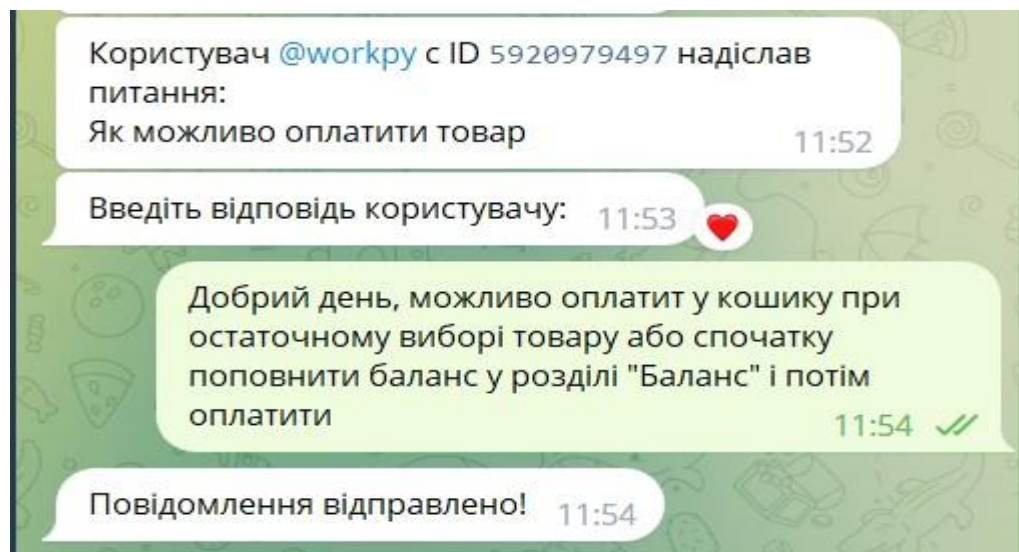


Рисунок 3.21 – Звернення до підтримки користувача

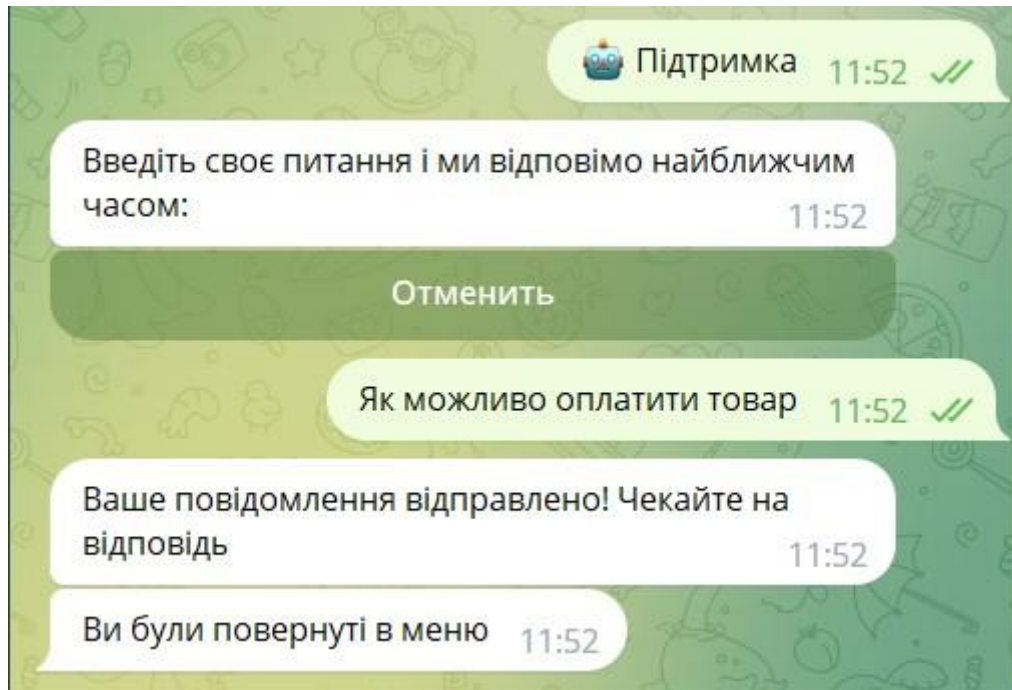


Рисунок 3.22 – Відповідь адміністратора користувачу

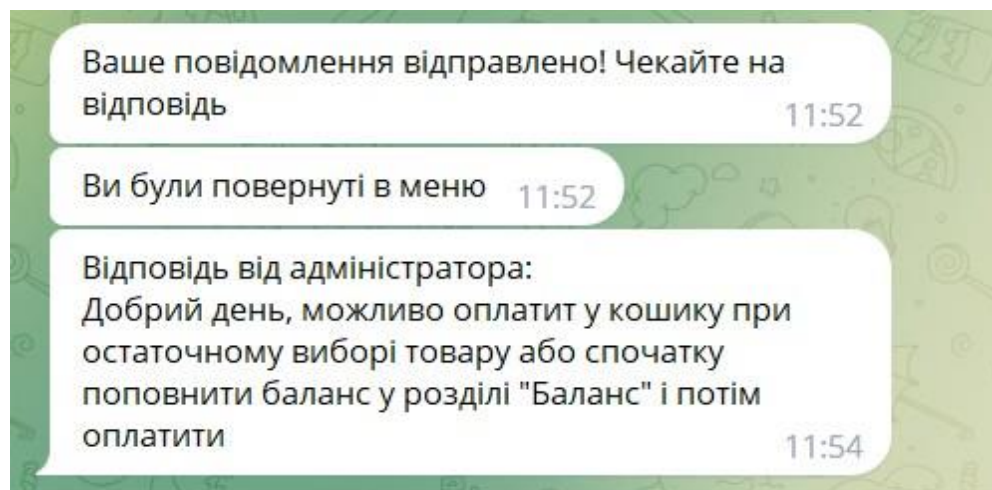


Рисунок 3.23 – Відповідь у чаті користувача

Код компоненту supports.py [13] вказано у додатку А

3.2.4 Розробка компонентів функціоналу роботи з балансом

Робота поповнення балансу в магазинному телеграм-боті полягає у забезпеченні можливості користувачам поповнювати свій баланс для подальшого використання при покупках (рис. 3.24).



Рисунок 3.24 – Поповнення балансу криптовалютою

Основний функціонал поповнення балансу може включати наступні кроки:

- відображення поточного балансу: Користувач може отримати інформацію про свій поточний баланс перед поповненням. Це може бути відображено в текстовому форматі або за допомогою графічних елементів на сторінці поповнення балансу;
- вибір способу поповнення: Користувач може мати можливість вибрати бажаний спосіб поповнення балансу. Це може включати оплату з банківської картки, електронного гаманця, платіжної системи або іншого доступного способу;
- обробка платежу: Після вибору способу поповнення, бот проводить обробку платежу. Це може включати взаємодію з платіжною системою або іншим сервісом для здійснення фінансової транзакції;
- підтвердження поповнення: Після успішного завершення платежу, користувач отримує підтвердження про поповнення балансу. Це може бути

відображено у вигляді повідомлення або спеціального поп-ап вікна з відповідним повідомленням;

— оновлення балансу: Після поповнення балансу, система оновлює інформацію про баланс користувача, зберігаючи відповідні зміни у базі даних або іншому сховищі. Таким чином, користувач може бачити актуальну інформацію про свій баланс при подальших взаємодіях з магазином.

Робота поповнення балансу завершується після оновлення інформації про баланс користувача. Після цього користувач може використовувати свій поповнений баланс для здійснення покупок в магазині. Розділ поповнення балансу забезпечує зручність та легкість використання для користувачів, а також допомагає підтримувати фінансову стабільність магазину та забезпечувати безперебійну роботу платіжних процесів. Приклади реалізації вказані в Додатку А.

У зв'язку з обмеженням на обсяг пояснювальної записки текст програми буде наданий не весь. Тільки основні фрагменти коду такі як користувацька панель, оплата та ін.

3.3 Тестування застосунку

Тестування застосунку є важливою частиною розробки будь-якого програмного продукту, включаючи телеграм-бота магазину. Цей процес допомагає виявити та виправити помилки, переконатися в коректному функціонуванні функцій та забезпечити задоволення користувачів. Тестування відбувалось за допомогою веб – застосунку телеграм. Звіт про тестування наведено в таблиці 3.1.

Таблиця 3.2 - Тестування функціональних вимог

Номер тесту	Опис тесту	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
1	2	3	4	5	6
1	Функціональне тестування придбання товару	Обрано товар, натиснуто кнопку "Придбати"	Покупка успішно завершена	Покупка успішно завершена	Пройде но
2	Функціональне тестування додавання товару в кошик	Обрано товар, натиснуто кнопку "Додати в кошик"	Товар додано в кошик	Товар додано в кошик	Пройде но
3	Функціональне тестування оформлення замовлення	В кошику є товари, натиснуто кнопку "Оформити"	Замовлення успішно оформлено	Замовлення успішно оформлено	Пройде но
4	Тестування взаємодії з базою даних	Виконано запит до бази даних	Повернені коректні дані	Повернені коректні дані	Пройде но
5	Додавання категорій та під-категорій	Обрати функцію додавання категорій та додати її	Створення нових категорій та під-категорій	Створення нових категорій та під-категорій	Пройде но
6	Тестування додавання товару	Введення всіх даних товару та додання його до категорії	Успішне створення картки товару	Успішне створення картки товару	Пройде но
7	Тестування зв'язку з підтримкою	Отримання повідомлення від користувача та отримання зворотнього зв'язку	Успішне отримання повідомлення та відповідь на нього	Успішне отримання повідомлення та відповідь на нього	Пройде но
8	Тестування розділу F.A.Q.	Обрано розіл, та безпроблемний перегляд його	Успішний перегляд або додання новиз розділів	Успішний перегляд або додання новиз розділів	Пройде но
9	Тестування перегляду розділу «Мої покупки»	Перегляд свої покупок або замовлень	Успішний перегляд свої замовлень	Успішний перегляд свої замовлень	Пройде но
10	Поповнення балансу	Поповнення балансу до моменту його оновлення в боті	Успішне зарахування грошей на баланс	Успішне зарахування грошей на баланс	Пройде но
11	Тестування негативного сценарію	Введено неправильні дані або натиснуто неправильну кнопку	Виведено повідомлення про помилку або ігнорування помилки	Виведено повідомлення про помилку або ігнорування помилки	Пройде но

Ця таблиця надає огляд тестів, їх опис, вхідні дані, очікувані та фактичні результати, а також статус кожного тесту. Завершений тест, який пройшов успішно, має статус "Пройдено". У разі виявлення проблем або помилок під час виконання тестів, відповідний статус слід встановити як "Не пройдено" та описати проблему в колонці "Фактичний результат".

Тестування є невід'ємною частиною розробки телеграм-бота магазину, яке допомагає впевнитися в його якості та надійності перед релізом. Важливо докладати зусиль для проведення якісного тестування, виявлення та виправлення помилок, а також забезпечення відповідності функціональних вимог та очікувань користувачів.

3.4 Висновки за розділом 3

У розділі 3 "Розробка телеграм-бота магазину на Python" було розглянуто процес розробки та реалізації основних компонентів телеграм-бота. Було описано файлову структуру проекту, функціонал класу Captcha для роботи з капчею, а також розроблено компоненти моделі користувацького інтерфейсу.

Під час розробки було використано бібліотеку aiogram для взаємодії з Telegram API та обробки повідомлень. Також були використані різні типи клавіатур та кнопок для зручності користувачів.

Код для роботи з моделлю користувацького інтерфейсу містить обробники повідомлень та відповідних кнопок. Наприклад, кнопка "Магазин" викликає обробник, який перевіряє наявність блокування користувача та відображає категорії товарів. Аналогічно, інші кнопки активують відповідні функції бота, такі як виведення списку товарів, робота з кошиком, перегляд покупок тощо.

Користувачів магазину телеграм-бота і надає їм зручний інтерфейс для здійснення покупок та взаємодії з магазином. Було реалізовано можливість

переглядати категорії товарів і підкатегорії, вибирати конкретні товари та додавати їх до кошика.

Також було зроблено тестування програмног забезпечення та по всіх показникам, функціонал працює та не видає ніяких помилок.

Розділ 3 розробки телеграм-бота магазина на Python відповідає за реалізацію основного функціоналу, який забезпечує зручну та ефективну взаємодію користувачів з ботом. В результаті розробки цих компонентів магазин може надати користувачам широкий спектр функцій і можливостей для здійснення покупок та керування своїми покупками.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення та умови застосування

4.1.1 Призначення програми

Призначення програми полягає у розробці та функціонуванні телеграм-бота магазину на платформі Python. Ця програма забезпечує зручну та ефективну взаємодію користувачів з магазином, дозволяючи їм переглядати товари, вибирати та додавати їх до кошика, переглядати баланс, здійснювати операції з поповнення та виведення коштів, а також переглядати свої активні та завершені замовлення.

До функцій програми відносяться:

- перегляд та купівля товару;
- зв'язок з підтримкою;
- онлайн оплата;
- відстежування замовлень;
- перегляд правил та частих питань;
- додавання товару (адміністраторська частина);
- редагування розділів та товару (адміністраторська частина).

4.1.2 Умови застосування програми

Наведене програмне забезпечення сумісне з будь-якою операційною системою та пристроєм, що підтримує телеграм. Воно має адаптивний інтерфейс, який забезпечує зручне використання на екранах з різними роздільними здатностями. Для успішної роботи програми необхідно мати пристрій, який може запустити телеграм, а також стабільне з'єднання з Інтернетом.

4.2 Характеристики програми

Програмне забезпечення реагує на помилки, що виникають у системі, і можуть бути викликані як неправильним введенням користувачем даних, наприклад, невірно введеною ціною, так і системними помилками, такими як відсутність з'єднання з Інтернетом але всі помилики пропускаються та бот продовжує роботу.

Для належної функціонування програми необхідно мати постійно працюючий сервер, який забезпечує надійне зберігання та доступ до програмного забезпечення.

4.3 Звертання до програми

Звертання до програми відбувається шляхом інтеракції користувача з інтерфейсом програми. Користувач може взаємодіяти з програмою шляхом введення даних, вибору опцій, натискання кнопок та взаємодії з елементами інтерфейсу.

4.4 Початкові та вихідні дані

Початковими даними у застосунку є інтерфейс бота, інформація про товар.

Вихідними даними є детальна інформація про той чи інший товар та його замовлення.

4.5 Повідомлення програмісту

Помилки які можуть виникнути при запиті на сервер:

- 400 badrequest– виникає у разі надіслання невірних даних;

— 403: Доступ заборонено - виникає, коли користувач спробує виконати дію або отримати доступ до ресурсів, на які він не має права.

5 КЕРІВНИЦТВО ОПЕРАТОРА

5.1 Призначення програми

Призначення програми наведено у розділі 4.1.1.

5.2 Умови виконання програми

Умови виконання програми наведено у розділі 4.1.2.

5.3 Виконання програми

Для початку роботи з застосунком, необхідно перейти по URL-адресі телеграм бота , за якою буде знаходитись застосунок.

Після успішного переходу по адресі користувачу відкривається сам бот, далі для доступу йому потрібно пройти реєстрацію, тобто капчу. Після чого йому надається повний функціонал, а саме:

- перегляд та купівля товару;
- зв'язок з підтримкою;
- доступ до панелі користувача;
- правила та часто задавані питання;
- робота с балансом.

На зображеннях 5.1 та 5.2 відображено те, що користувач отримує в результаті отримання доступу до бота. Далі буде показано зображення католгу товару та картки його.

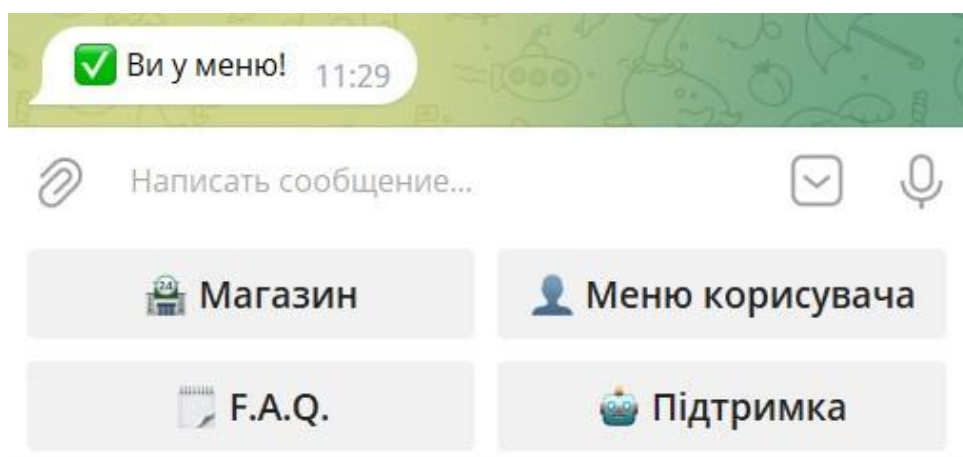


Рисунок 5.1 – Головне меню

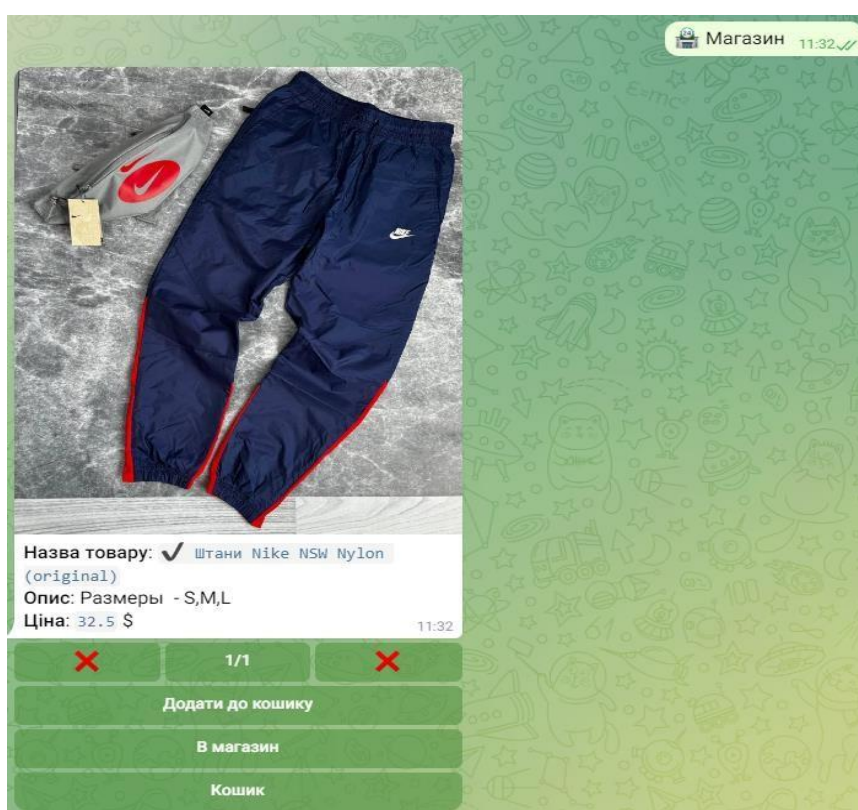


Рисунок 5.2 – Перегляд товару

5.4 Повідомлення

Повідомлення, які система може видати користувачу:

- ви ввели не число! Введіть ціле число або число з точкою, наприклад: 134.50 – ця помилка з'являється коли користувач вводить не вірно число для поповнення балансу;

— проводяться технічні роботи — це повідомлення може з'явитися коли користувач намагається використати якийсь функціонал в боті, але в нього не виходить тому, що відбуваються технічні роботи в боті (наприклад робиться оновлення в застосунку).

ВИСНОВКИ

У даній кваліфікаційній роботі бакалавра була розроблена та реалізована система телеграм-бота для онлайн-магазину. В процесі розробки було використано мову програмування Python та фреймворк Aiogram для побудови функціоналу бота.

Проведено дослідження та аналіз сучасних технологій для розробки телеграм-ботів, що дозволило вибрати оптимальний набір інструментів для реалізації поставлених завдань.

Були визначені та розроблені ключові компоненти системи, зокрема модель користувацького інтерфейсу, функціонал магазину, управління кошиком покупок, перегляд покупок користувача та інші.

Програма успішно пройшла процес тестування, в ході якого було перевірено коректність роботи функцій, взаємодію з базою даних та відповідність вимогам функціональності.

Отже, розроблена програма дозволяє користувачам здійснювати покупки через телеграм-бота, переглядати та керувати своїми замовленнями, отримувати повідомлення про акції та новини магазину. Вона має потенціал для подальшого розширення та вдосконалення з урахуванням потреб та вимог користувачів.

Застосунок поділений на окремі модулі, що у майбутньому допоможе більш ефективно та швидко розширювати його функціонал.

Розробка даної системи телеграм-бота для онлайн-магазину відповідає актуальним тенденціям розвитку електронної комерції та надає можливість покупцям зручно та швидко здійснювати покупки та отримувати необхідну інформацію про товари.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Stack Overflow - Онлайн-спільнота програмістів, де можна знайти відповіді на питання щодо програмування та рішення технічних проблем [Електронний ресурс]. – Режим доступу: <https://stackoverflow.com/>.
2. Асинхронний Telegram бот мовою Python 3 з використанням бібліотеки aiogram [Електронний ресурс]. – Режим доступу: <https://opensourcelibs.com/lib/aiogram-lessons>.
3. Блог Real Python - Цей блог надав корисні статті, уроки та приклади коду, пов'язані з розробкою програм на Python. Він містить широкий спектр тем, включаючи розробку телеграм-ботів, веб-фреймворків та інші аспекти програмування на Python [Електронний ресурс]. – Режим доступу: <https://realpython.com/>.
4. Використання мови Python для розробки науково-технічного програмного забезпечення [Електронний ресурс]. – Режим доступу: <https://dou.ua/lenta/articles/python-for-science>.
5. Відеоуроки на YouTube [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=ObwoMskHDoA&ab_channel=%D0%93%D0%BE%D1%88%D0%B0%D0%94%D1%83%D0%B4%D0%B0%D1%80%D1%8C.
6. Все, про що повинен знати розробник Телеграм-ботів [Електронний ресурс]. – Режим доступу: <https://habr.com/ru/post/543676/>
7. Документація Python: Офіційна документація мови програмування Python [Електронний ресурс]. – Режим доступу: <https://docs.python.org/>.
8. Документація по платіжній системі: [Електронний ресурс]. – Режим доступу: <https://documenter.getpostman.com/view/7907941/2s93JusNJt#standard-e-commerce-flow-for-nowpayments-api>.
9. Документація по створенню Telegram-ботів [Електронний ресурс]. – Режим доступу: <https://core.telegram.org/bots>.

10. Ласкаво просимо до документації aiogram! [Електронний ресурс]. – Режим доступу: <https://docs.aiogram.dev/en/latest/>.
11. Наводимо порядки та додаємо фільтри [Електронний ресурс]. – Режим доступу: <https://geekstand.top/development/aiogram-urok-2-navodim-porjadki-i-dobavljaem-filtry/>.
12. Офіційна документація PostgreSQL [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/docs/>.
13. Посібник з мови програмування Python. [Електронний ресурс]. – Режим доступу: https://www.udemy.com/course/bestpython/learn/lecture/14102616?components=add_to_cart%2Cavailable_coupons%2Cbase_purchase_section%2Cbuy_button%2Cbuy_for_team%2Ccacheable_buy_button%2Ccacheable_deal_badge%2Ccacheable_discount_expiration%2Ccacheable_price_text%2Ccacheable_purchase_text%2Ccurated_for_ufb_notice_context%2Ccurriculum_context%2Cdeal_badge%2Cdiscount_expiration%2Cgift_this_course%2Cincentives%2Cinstructor_links%2Clifetime_access_context%2Cmoney_back_guarantee%2Cprice_text%2Cpurchase_tabs_context%2Cpurchase%2Crecommendation%2Credeem_coupon%2Csidebar_container%2Cpurchase_body_container#overview.

ДОДАТОК А**Код програми**

A.1 Start.py

```

from config import dp, db
from aiogram import types
from captcha import Captcha
from markups import rules_mkp, menu_mkp

@dp.message_handler(commands='start')
async def startcmd(message: types.Message):
    stat = db.check_userstat(message.from_user.id)
    if len(message.text.split()) == 2:
        db.add_user(message.from_user.id, message.from_user.username,
int(message.text.split()[1]))
    else:
        db.add_user(message.from_user.id, message.from_user.username, 0)
    if stat == 'rules':
        await message.answer('Тут правила', reply_markup=rules_mkp())
    elif stat == 'ban':
        await message.answer('❌ Ви заблоковані!')
    elif stat == 'ok':
        await message.answer('✅ Ви у меню!', reply_markup=menu_mkp())
    else:
        captcha = Captcha()
        captcha.register_handlers(dp)

        await message.answer(
            captcha.get_caption(),
            reply_markup=captcha.get_captcha_keyboard()
        )

@dp.message_handler(commands='id')
async def idcmd(message: types.Message):
    await message.answer(f'Ваш ID: <code>{message.from_user.id}</code>')

@dp.callback_query_handler(text='rulesok')
async def rulesokcall(call: types.CallbackQuery):
    db.change_status(call.from_user.id, 'ok')
    await call.message.delete()

```

```

        await call.message.answer('Вы в меню', reply_markup=menu_mkp())

@dp.callback_query_handler(text='rulesno')
async def rulesnocall(call: types.CallbackQuery):
    db.change_status(call.from_user.id, 'ban')
    await call.message.delete()
    await call.message.answer('Ви заблоковані!')

```

A.2 Main.py

```

from config import dp
from aiogram.utils import executor
import start
import support
import admin
import userfaq
import catalog
import box
import usermenu

if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)

```

A.3 Markups.py

```

from aiogram import types
from config import db

def rules_mkp():
    mkp = types.InlineKeyboardMarkup()

    btn1 = types.InlineKeyboardButton('Прийняти', callback_data='rulesok')
    btn2 = types.InlineKeyboardButton('Відхилити', callback_data='rulesno')
    mkp.add(btn1, btn2)
    return mkp

def menu_mkp():
    mkp = types.ReplyKeyboardMarkup(resize_keyboard=True)

    btn1 = types.KeyboardButton('☐ Магазин')

```

```

btn2 = types.KeyboardButton('☐ Меню корисувача')
btn3 = types.KeyboardButton('☐ F.A.Q.')
btn4 = types.KeyboardButton('🔗 Підтримка')
mkp.add(btn1, btn2).add(btn3, btn4)
return mkp

```

```

def usermenu_mkp():
    mkp = types.ReplyKeyboardMarkup(resize_keyboard=True)

```

```

    btn1 = types.KeyboardButton('Баланс')
    btn2 = types.KeyboardButton('☐ Кошик')
    btn3 = types.KeyboardButton('☐ Мої покупки')
    btn4 = types.KeyboardButton('☐ Реферальна система')
    btn5 = types.KeyboardButton('У меню')
    mkp.add(btn1, btn2).add(btn3, btn4).add(btn5)
    return mkp

```

```

def cancel_mkp():
    mkp = types.InlineKeyboardMarkup()

    btn1 = types.InlineKeyboardButton('Отменить', callback_data='cancel')
    mkp.add(btn1)
    return mkp

```

```

def admin_mkp():
    mkp = types.ReplyKeyboardMarkup(resize_keyboard=True)

    btn1 = types.KeyboardButton('☐ Товар')
    btn2 = types.KeyboardButton('☐ Замовлення')
    btn3 = types.KeyboardButton('☐ Користувачі')
    btn4 = types.KeyboardButton('☐ Налаштування F.A.Q')
    btn5 = types.KeyboardButton('☐ Зміни реф')
    btn6 = types.KeyboardButton('☐ Режим покупця')
    mkp.add(btn1, btn2).add(btn3, btn4).add(btn5).add(btn6)
    return mkp

```

```

def all_users_mkp(page):
    users_list = db.get_all_users()
    mkp = types.InlineKeyboardMarkup(row_width=2)

    if page == 1:

```

```

        if len(users_list) < 11:
            for i in users_list:
                try:
                    mkp.add(types.InlineKeyboardButton(f'Пользователь
{i[1]}', callback_data=f'getuser_{i[1]}'))
                except Exception as ex:
                    print(ex)
            else:
                try:
                    for i in range(page-1, page*10):
                        mkp.add(types.InlineKeyboardButton(f'Пользователь
{users_list[i][1]}', callback_data=f'getuser_{users_list[i][1]}'))
                        mkp.add(types.InlineKeyboardButton('Далее',
callback_data=f'usersnext_{page+1}'))
                    except:
                        pass
                else:
                    try:
                        for i in range((page-1)*10, page*10):
                            mkp.add(types.InlineKeyboardButton(f'Пользователь
{users_list[i][1]}', callback_data=f'getuser_{users_list[i][1]}'))
                            mkp.add(types.InlineKeyboardButton('Назад',
callback_data=f'usersback_{page-1}'),
                                types.InlineKeyboardButton('Далее',
callback_data=f'usersnext_{page+1}'))
                        except:
                            mkp.add(types.InlineKeyboardButton('Назад',
callback_data=f'usersback_{page-1}'))
                            mkp.add(types.InlineKeyboardButton('Отменить', callback_data='admin'))
                    return mkp

```

A.4 functions.py

```

from config import db, bot
from aiogram import types

def get_faq_admin():
    faq_list = db.get_all_faq()
    mkp = types.InlineKeyboardMarkup()
    for i in faq_list:

```

```

        mkp.add(types.InlineKeyboardButton(i[1],
callback_data=f'changefaq_{i[0]}'))

        mkp.add(types.InlineKeyboardButton('Новий розділ',
callback_data='newfaq'))

        mkp.add(types.InlineKeyboardButton('Повернутися до адмін-панелі',
callback_data='admin'))

        return mkp

    def get_faq_user():
        faq_list = db.get_all_faq()
        mkp = types.InlineKeyboardMarkup()
        for i in faq_list:
            mkp.add(types.InlineKeyboardButton(i[1],
callback_data=f'getfaq_{i[0]}'))

            mkp.add(types.InlineKeyboardButton('Вернутися в меню',
callback_data='tomenu'))

        return mkp

    def get_categories_admin():
        cat_list = db.get_all_cat()
        mkp = types.InlineKeyboardMarkup()
        for i in cat_list:
            mkp.add(types.InlineKeyboardButton(i[1],
callback_data=f'admindcat_{i[0]}'))

            mkp.add(types.InlineKeyboardButton('✚ Додати катеогрію ',
callback_data='addcat'))

            mkp.add(types.InlineKeyboardButton('◻ Повернутися в адмін-панель',
callback_data='admin'))

        return mkp

    def get_categories_user():
        cat_list = db.get_all_cat()
        mkp = types.InlineKeyboardMarkup()
        for i in cat_list:
            mkp.add(types.InlineKeyboardButton(i[1],
callback_data=f'usercat_{i[0]}'))

            mkp.add(types.InlineKeyboardButton('◻ Повернутися в меню',
callback_data='tomenu'))

        return mkp

    def get_subcategories_admin(cat_id):

```

```

        subcat_list = db.get_subcat(cat_id)
        mkp = types.InlineKeyboardMarkup()
        for i in subcat_list:
            mkp.add(types.InlineKeyboardButton(i[1],
callback_data=f'adminsubcat_{i[0]}'))
            mkp.add(types.InlineKeyboardButton('✚          Додати          під-категорію',
callback_data=f'addsubcat_{cat_id}'))
            mkp.add(types.InlineKeyboardButton('✎          Змінити          назву',
callback_data=f'changenamecat_{cat_id}'))
            mkp.add(types.InlineKeyboardButton('✖          Видалити          категорію',
callback_data=f'delcat_{cat_id}'))
            mkp.add(types.InlineKeyboardButton('⬅          Повернутися',
callback_data='products'))
        return mkp

    def get_subcategories_user(cat_id):
        subcat_list = db.get_subcat(cat_id)
        mkp = types.InlineKeyboardMarkup()
        for i in subcat_list:
            mkp.add(types.InlineKeyboardButton(i[1],
callback_data=f'usersubcat_{i[0]}'))
            mkp.add(types.InlineKeyboardButton('⬅          Повернутися',
callback_data='toshop'))
        return mkp

    def get_goods_admin(subcat_id):
        goods_list = db.get_goods(subcat_id)
        mkp = types.InlineKeyboardMarkup()
        for i in goods_list:
            mkp.add(types.InlineKeyboardButton(i[1],
callback_data=f'admingood_{i[0]}'))
            mkp.add(types.InlineKeyboardButton('✚          Додати          товар',
callback_data=f'addgood_{subcat_id}'))
            mkp.add(types.InlineKeyboardButton('✎          Змінити          назву',
callback_data=f'changenamesubcat_{subcat_id}'))
            mkp.add(types.InlineKeyboardButton('✖          Видалити          під-категорію',
callback_data=f'delsubcat_{subcat_id}'))
            mkp.add(types.InlineKeyboardButton('⬅          Повернутися',
callback_data='products'))
        return mkp

    async def send_admin_good(goodid, user_id):

```

```

        good_info = db.get_goodinfo(int(goodid))
        mkp = types.InlineKeyboardMarkup()
        btn1 = types.InlineKeyboardButton('Назва',
callback_data=f'changegoodname_{goodid}')
        btn2 = types.InlineKeyboardButton('Опис',
callback_data=f'changegooddesc_{goodid}')
        btn3 = types.InlineKeyboardButton('Ціну',
callback_data=f'changegoodprice_{goodid}')
        btn4 = types.InlineKeyboardButton('❏ Видалити',
callback_data=f'delgood_{goodid}')
        btn5 = types.InlineKeyboardButton('Відмінити', callback_data='admin')
        mkp.add(btn1).add(btn2).add(btn3).add(btn4).add(btn5)
        if good_info[3] == 'None':
            await bot.send_message(user_id, f'Назва товару:
<code>{good_info[0]}</code>\nОпис товару: <code>{good_info[1]}</code>\nЦіна:
<code>{good_info[2]}</code>${\n\nВиберіть, що потрібно змінити', reply_markup=mkp)
        else:
            await bot.send_photo(user_id, open(f'images/{good_info[3]}', 'rb'),
caption=f'Назва товару: <code>{good_info[0]}</code>\nОпис товару:
<code>{good_info[1]}</code>\nЦіна: <code>{good_info[2]}</code> ${\n\nВиберіть, що
потрібно змінити', reply_markup=mkp)

    async def send_good(step, subcatid, user_id):
        goods = db.get_goods_user(subcatid)

        name = goods[step][1]
        description = goods[step][2]
        price = goods[step][3]
        photo = goods[step][4]
        goodid = goods[step][0]

        mkp = types.InlineKeyboardMarkup()
        if step == 0:
            btn1 = types.InlineKeyboardButton('✕', callback_data='none')
        else:
            btn1 = types.InlineKeyboardButton('←',
callback_data=f'catback_{subcatid}_{step-1}')
            btn2 = types.InlineKeyboardButton(f'{step+1}/{len(goods)}',
callback_data='none')
            if step+1 == len(goods):
                btn3 = types.InlineKeyboardButton('✕', callback_data='none')

```

```

        else:
            btn3 = types.InlineKeyboardButton('→',
callback_data=f'catnext_{subcatid}_{step+1}')
            btn4 = types.InlineKeyboardButton('Додати до кошику',
callback_data=f'tobox_{goodid}')
            btn5 = types.InlineKeyboardButton('В магазин', callback_data='toshop')
            btn6 = types.InlineKeyboardButton('Кошик', callback_data='korzina')
            if len(db.get_box(user_id)) == 0:
                mkp.add(btn1, btn2, btn3).add(btn4).add(btn5)
            else:
                mkp.add(btn1, btn2, btn3).add(btn4).add(btn5).add(btn6)

        if photo == 'None':
            await bot.send_message(user_id, f'<b>Назва товару</b>:
<code>{name}</code>\n<b>Опис</b>: {description}\n<b>Ціна</b>: <code>{price}</code>',
reply_markup=mkp)
        else:
            await bot.send_photo(user_id, open(f'images/{photo}', 'rb'),
caption=f'<b>Назва товару</b>: <code>{name}</code>\n<b>Опис</b>:
{description}\n<b>Ціна</b>: <code>{price}</code> $', reply_markup=mkp)

```

A.5 States.py

```
from aiogram.dispatcher.filters.state import State, StatesGroup
```

```
class SuppUser(StatesGroup):
    UserId = State()
```

```
class SuppAdmin(StatesGroup):
    UserId = State()
```

```
class NewFaq(StatesGroup):
    Name = State()
    Text = State()
```

```
class FaqName(StatesGroup):
    FaqId = State()
```

```
class FaqText(StatesGroup):
    FaqId = State()
```

```

class AddCat(StatesGroup):
    CatName = State()

class ChangeNameecat(StatesGroup):
    CatId = State()

class AddSubcat(StatesGroup):
    SubcatName = State()

class ChangeNamesubcat(StatesGroup):
    SubcatId = State()

class AddGood(StatesGroup):
    SubcatId = State()
    Name = State()
    Description = State()
    Photo = State()
    Price = State()

class ChangeNameGood(StatesGroup):
    GoodId = State()

class ChangeDescGood(StatesGroup):
    GoodId = State()

class ChangePriceGood(StatesGroup):
    GoodId = State()

class ChangeRef(StatesGroup):
    Ref = State()

class NewOrder(StatesGroup):
    Adress = State()
    Comment = State()

class OrderEnd(StatesGroup):
    OrderId = State()

class Rassilka(StatesGroup):
    Text = State()

```

```
class Withdraw(StatesGroup):
    Amount = State()
    Req = State()
```

```
class Deposit(StatesGroup):
    Amount = State()
```

A.6 payments.py

```
from config import dp, bot, db, url, admins
from aiogram import types
from aiogram.dispatcher import FSMContext
from markups import cancel_mkp, menu_mkp, usermenu_mkp
from states import Withdraw, Deposit
from payments import getCoins, createPayment, check_pay

@dp.message_handler(text='☐ Меню користувача')
async def usermenumsg(message: types.Message):
    if db.check_ban(message.from_user.id):
        await message.answer('Ви увійшли в меню користувача!',
reply_markup=usermenu_mkp())

@dp.message_handler(text='Баланс')
async def balancemsg(message: types.Message):
    if db.check_ban(message.from_user.id):
        balance = db.get_balance(message.from_user.id)
        ref_balance = db.get_refbalance(message.from_user.id)
        mkp = types.InlineKeyboardMarkup()
        btn1 = types.InlineKeyboardButton('Поповнити',
callback_data='deposit')
        btn2 = types.InlineKeyboardButton('Вивести', callback_data='withdraw')
        mkp.add(btn1, btn2)
        await message.answer(f'<b>Ваш баланс</b>:
<code>{float(balance)}</code> $\n<b>Заработано з рефералів</b>:
<code>{float(ref_balance)}</code>', reply_markup=mkp)

@dp.callback_query_handler(text='deposit')
async def depositcall(call: types.CallbackQuery):
    await call.message.delete_reply_markup()
```

```

        await call.message.answer('Введіть суму депозиту:',
reply_markup=cancel_mkp())
        await Deposit.Amount.set()

@dp.message_handler(state=Deposit.Amount)
async def depositamountmsg(message: types.Message, state: FSMContext):
    try:
        amount = float(message.text)
        async with state.proxy() as data:
            data['Amount'] = amount
        mkp = types.InlineKeyboardMarkup()
        btn1 = types.InlineKeyboardButton('Криптовалюта',
callback_data='decrypto')
        btn2 = types.InlineKeyboardButton('Картка', callback_data='card')
        btn3 = types.InlineKeyboardButton('Скасувати', callback_data='cancel')
        mkp.add(btn1, btn2).add(btn3)
        await message.answer('Виберіть спосіб оплати:', reply_markup=mkp)
    except:
        await message.answer('Ви ввели не число! Введіть ціле число або число
з точкою, наприклад: <code>134.50</code>', reply_markup=cancel_mkp())

@dp.callback_query_handler(text='decrypto', state=Deposit.Amount)
async def decryptodepamountcall(call: types.CallbackQuery, state:
FSMContext):
    async with state.proxy() as data:
        pass
    amount = data['Amount']
    await call.message.delete_reply_markup()
    await call.answer('Завантажую список криптовалют')
    coins = await getCoins()
    mkp = types.InlineKeyboardMarkup(row_width=3)
    for coin in coins:
        mkp.insert(types.InlineKeyboardButton(coin,
callback_data=f'cryptodep_{coin}_n_{amount}'))
    await call.message.answer('Виберіть криптовалюту для оплати',
reply_markup=mkp)
    await state.finish()

@dp.callback_query_handler(text_contains='cryptodep_')

```

```

async def cryptodepcall(call: types.CallbackQuery):
    crypto = call.data.split('_')[1]
    amount = call.data.split('_')[3]
    try:
        paym = await createPayment(amount, crypto)
        pay_amount = paym['pay_amount']
        pay_address = paym['pay_address']
        pay_currency = paym['pay_currency']
        pay_id = paym['payment_id']
        mkp = types.InlineKeyboardMarkup()
        btn = types.InlineKeyboardButton('Перевірити оплату',
callback_data=f'cryptocheck_{pay_id}_n_{amount}')
        mkp.add(btn)
        await call.message.answer(f'Перекладіть: <code>{pay_amount}</code>
{pay_currency}\nНа гаманець: <code>{pay_address}</code>\n\nПісля оплати не забудьте
натиснути на "Перевірити оплату". Транзакція може проходити до 2-х годин',
reply_markup=mkp)
    except:
        await call.message.answer('Сума оплати нижча за мінімальну. Спробуйте
виконати депозит на велику суму', reply_markup=menu_mkp())

@dp.callback_query_handler(state=Deposit.Amount, text='cancel')
async def canceldepositamountcall(call: types.CallbackQuery, state:
FSMContext):
    await call.message.delete()
    await call.message.answer('Скасовано. Ви були повернуті в меню',
reply_markup=menu_mkp())
    await state.finish()

@dp.callback_query_handler(text='withdraw')
async def withdrawcall(call: types.CallbackQuery):
    balance = db.get_balance(call.from_user.id)
    if float(balance) > 0:
        await call.message.delete_reply_markup()
        await call.message.answer('Введіть суму для виведення:',
reply_markup=cancel_mkp())
        await Withdraw.Amount.set()
    else:
        await call.answer('На вашому балансі 0 $', show_alert=True)

@dp.message_handler(state=Withdraw.Amount)

```

```

async def withdrawamountmsg(message: types.Message, state: FSMContext):
    try:
        msg = float(message.text)
        bal = float(db.get_balance(message.from_user.id))
        if msg <= bal:
            async with state.proxy() as data:
                data['Amount'] = msg
            await message.answer('Введіть реквізити, наприклад: <code>Номер  
картки: XXXX XXXX XXXX XXXX</code>', reply_markup=cancel_mkp())
            await Withdraw.next()
        else:
            await message.answer('Ви ввели більше, ніж у вас є! Спробуйте ще  
раз', reply_markup=cancel_mkp())
    except:
        await message.answer('Введіть суму для виведення цілим числом або через  
точку, наприклад: <code>151.21</code>', reply_markup=cancel_mkp())

@dp.message_handler(state=Withdraw.Req)
async def withdrawreqmsg(message: types.Message, state: FSMContext):
    async with state.proxy() as data:
        data['Req'] = message.text
    amount = data['Amount']
    req = data['Req']
    mkp = types.InlineKeyboardMarkup()
    btn1 = types.InlineKeyboardButton('Так', callback_data='go')
    btn2 = types.InlineKeyboardButton('Скасувати', callback_data='cancel')
    mkp.add(btn1).add(btn2)
    await message.answer(f'Ви хочете подати заявку на висновок  
<code>{amount}</code> $\\nНа реквізити: {req}', reply_markup=mkp)

@dp.callback_query_handler(text='go', state=Withdraw.Req)
async def gowithdrawreqcall(call: types.CallbackQuery, state: FSMContext):
    async with state.proxy() as data:
        pass
    amount = data['Amount']
    req = data['Req']
    await call.message.delete()
    await call.message.answer('Заявка на висновок успішно подана. Ви були  
повернуті в меню', reply_markup=menu_mkp())
    db.pay_balance(call.from_user.id, float(amount))
    await state.finish()

```



```

        step=step+1
        await message.answer(f'Активні замовлення:\n{text}')

    if len(end_orders) == 0:
        await message.answer('Закінчених замовлень не знайдено!')
    else:
        text = ''
        for i in end_orders:
            text=f'{text}-----\n'
            step = 1
            for a in i:
                try:
                    text=f'{text}Сумма: {float(a)}\n'
                except:
                    text=f'{text}{step}.{i[0]}\n'
            step=step+1
        await message.answer(f'Остані 2 завершених замовлення:\n{text}')
```

A.7 support.py

```

from config import admins, dp, bot, db
from aiogram import types
from aiogram.dispatcher import FSMContext
from states import SuppUser, SuppAdmin
from markups import cancel_mkp, menu_mkp

@dp.message_handler(text='🛠 Підтримка')
async def suppmg(message: types.Message):
    if db.check_ban(message.from_user.id):
        await message.answer('Введіть своє питання і ми відповімо найближчим часом:', reply_markup=cancel_mkp())
        await SuppUser.UserId.set()

@dp.callback_query_handler(text='cancel', state=SuppUser.UserId)
async def cancelstatesuppuser(call: types.CallbackQuery, state: FSMContext):
    await call.message.delete_reply_markup()
    await call.message.answer('Скасовано. Ви були повернуті в меню',
reply_markup=menu_mkp())

@dp.message_handler(state=SuppUser.UserId)
```

```

async def suppuseruseridmsg(message: types.Message, state: FSMContext):
    mkp = types.InlineKeyboardMarkup()
    btn = types.InlineKeyboardButton('Відповісти',
callback_data=f'answer_{message.from_user.id}')
    mkp.add(btn)
    for admin in admins:
        await bot.send_message(admin, f'Користувач
@{message.from_user.username} с ID {message.from_user.id} надіслав
питання:\n{message.text}', reply_markup=mkp)
        await message.answer('Ваше повідомлення відправлено! Чекайте на
відповідь')
        await message.answer('Ви були повернуті в меню', reply_markup=menu_mkp())
        await state.finish()

@dp.callback_query_handler(text_contains='answer_')
async def answercall(call: types.CallbackQuery, state: FSMContext):
    await call.message.delete_reply_markup()
    await call.message.answer('Введіть відповідь користувачу:')
    await SuppAdmin.UserId.set()
    user_id = call.data.split('_')[1]
    async with state.proxy() as data:
        data['UserId'] = user_id

@dp.message_handler(state=SuppAdmin.UserId)
async def suppadminuseridmsg(message: types.Message, state: FSMContext):
    await message.answer('Повідомлення відправлено!')
    async with state.proxy() as data:
        pass
    user_id = data['UserId']
    await state.finish()
    await bot.send_message(int(user_id), f'Відповідь від адміністратора:
\n{message.text}')

```

A.8 config.py

```

from aiogram import Bot
from aiogram.dispatcher import Dispatcher
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from database import DB

```

```

admins = [531852145]

url = 'https://t.me/shop_lolipop_bot'

bot = Bot(token='5618528182:AAGr9w9xqAaG0aTzb8dgm2T0kUljxkjGk3k',
parse_mode='html')
storage = MemoryStorage()
dp = Dispatcher(bot, storage=storage)

db = DB('shop', 'postgres', '1111', 'localhost')

```

A.9 userfaq.py

```

from config import dp, db
from aiogram import types
from functions import get_faq_user
from markups import menu_mkp

@dp.message_handler(text='❏ F.A.Q.')
async def faqmsg(message: types.Message):
    if db.check_ban(message.from_user.id):
        await message.answer('Оберіть розділ для ознайомлення:',
reply_markup=get_faq_user())

@dp.callback_query_handler(text='tomenu')
async def tomenucall(call: types.CallbackQuery):
    if db.check_ban(call.from_user.id):
        await call.message.delete()
        await call.message.answer('Ви були повернені до меню',
reply_markup=menu_mkp())

@dp.message_handler(text='В меню')
async def tomenumsg(message: types.Message):
    if db.check_ban(message.from_user.id):
        await message.answer('Ви були повернені до головного меню!',
reply_markup=menu_mkp())

@dp.callback_query_handler(text_contains='getfaq_')
async def getfaqcall(call: types.CallbackQuery):
    if db.check_ban(call.from_user.id):

```

```

        faqid = call.data.split('_')[1]
        faq_info = db.get_faq(int(faqid))
        await
    call.message.edit_text(f'<b>{faq_info[0]}</b>\n\n{faq_info[1]}',
        reply_markup=get_faq_user())

```

A.10 captcha.py

```

from typing import Dict
from aiogram import types, Dispatcher
from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup
from aiogram.utils.exceptions import MessageNotModified
import random
from markups import rules_mkp

class Captcha:
    captcha_id = 0
    passed_captcha_users = set()

    def __init__(self, choices: Dict[str, str] = None) -> None:
        if choices and isinstance(choices, dict):
            self.choices = choices
        else:
            self.choices = {
                "яблоко": "□",
                "автомобіль": "□",
                "собаку": "□",
                "дерево": "□",
                "райдугу": "□",
                "банан": "□",
            }
        self.correct_choice = random.choice(list(self.choices.keys()))

    # ID for captcha needs for creating unique callback_data for keyboard
    Captcha.captcha_id += 1
    self.captcha_id = Captcha.captcha_id
    self.callback_name = f"_Captcha{self.captcha_id}"

    self.captcha_passed = False

```

```

def get_captcha_keyboard(self) -> InlineKeyboardMarkup:
    captcha_keyboard = InlineKeyboardMarkup()

    for choice in random.sample(list(self.choices.keys()),
len(self.choices)):
        captcha_keyboard.insert(
            InlineKeyboardButton(
                self.choices[choice],
                callback_data=f"{self.callback_name}_{choice}_"
                + ("1" if choice == self.correct_choice else "0")
                # 1 at the end of callback_data means, that this button is
correct one
            )
        )

    return captcha_keyboard

def get_caption(self) -> str:
    return f"Щоб отримати доступ до боту, натисніть на
{self.correct_choice}"

async def captcha_choice_handler(
    self,
    callback_query: types.CallbackQuery,
) -> None:
    if callback_query.data.split("_")[-1] == "0":
        self.correct_choice = random.choice(list(self.choices.keys()))
        try:
            await callback_query.message.edit_text(
                "Невірно. Спробуйте ще раз\n" + self.get_caption(),
                reply_markup=self.get_captcha_keyboard(),
            )
        except MessageNotModified:
            ...
        return

    self.captcha_passed = True
    Captcha.passed_captcha_users.add(callback_query.from_user.id)

    await callback_query.message.edit_text(
        "Капча пройдена. Вам було надано доступ до боту", reply_markup=None
    )

```

```

    )
    await callback_query.message.answer('Тут правила',
reply_markup=rules_mkp())

```

```

def register_handlers(self, dp: Dispatcher):
    dp.register_callback_query_handler(
        self.captcha_choice_handler,
        lambda c: c.data.startswith(f"{self.callback_name}_choice_"),
    )

```

A.11 Payments.py

```

from config import db
import aiohttp
import asyncio
import json

api_crypto = 'TQC29Q0-F6N44HD-JB2RDDC-XM0A7VW'

async def getCoins():
    async with aiohttp.ClientSession() as session:
        async with session.get('https://api.nowpayments.io/v1/merchant/coins',
                                headers={'x-api-key':
api_crypto}) as resp:
            response = await resp.json()
            return response['selectedCurrencies']

async def createPayment(amount, paycurrency):
    headers = {
        'x-api-key': api_crypto
    }
    payload = {
        # "price_amount": 3,
        "price_amount": float(amount),
        "price_currency": "usd",
        "pay_currency": paycurrency,
    }
    async with aiohttp.ClientSession() as session:
        async with session.post('https://api.nowpayments.io/v1/payment',
                                headers=headers, data=payload) as resp:

```

```
        response = await resp.json()
        return response

    async def check_pay(payment_id):
        headers = {
            'x-api-key': api_crypto
        }
        async with aiohttp.ClientSession() as session:
            async with session.get(f'https://api.nowpayments.io/v1/payment/{payment_id}',
                                   headers=headers) as resp:
                response = await resp.json()
                return response['payment_status']
```

ДОДАТОК Б
Слайди презентації

ДИПЛОМНА РОБОТА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТЕЛЕГРАМ-БОТА МАГАЗИНУ

Розробив: ст. групи КНТз-219
Керівник: доцент, к.т.н.

Владислас ТИМОШЕНКО
Сергій СЕРДЮК

Рисунок Б.1 – Слайд 1

Мета роботи

Підвищити кількість продажів шляхом більш ефективної роботи користувача з онлайн-засобами.

Об'єкт дослідження – процес продажу товару онлайн.

Предмет дослідження – онлайн телеграм боти для продажу товару через телеграм.



НУ "ЗП", Кафедра програмних засобів

2

Рисунок Б.2 – Слайд 2

Задачі, що вирішуються

- провести аналіз предметної області;
- провести огляд програмних аналогів;
- розробка технічного завдання;
- розробка структури програмного забезпечення;
- розробка структури програмного забезпечення;
- програмна реалізація застосунку;
- розробка необхідної документації.



НУ "ЗП", Кафедра програмних засобів

3

Рисунок Б.3 – Слайд 3

Аналіз програм-аналогів

Критерії	Результати порівняння	
	Shop Lolipop Bot	ShoppingExpress Bot
Можливість робити замовлення	+	+
Оплата онлайн	+	-
Комунікація з користувачем	+	+
Розширені можливості	+	-
Безкоштовність	+	+



НУ "ЗП", Кафедра програмних засобів

4

Рисунок Б.4 – Слайд 4

Структурна схема програми

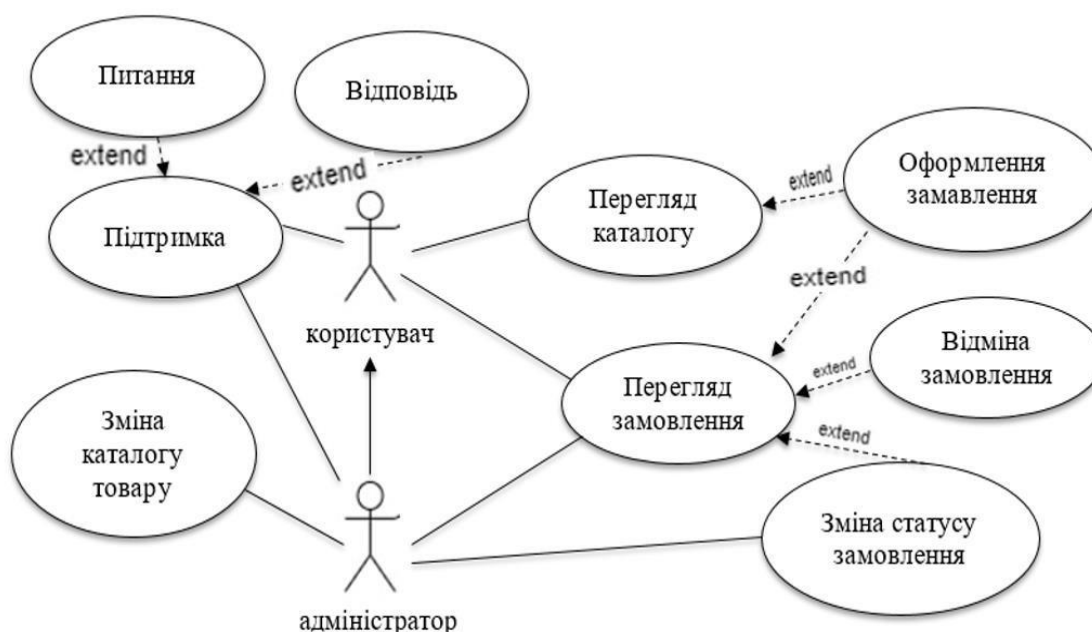


НУ "ЗП", Кафедра програмних засобів

5

Рисунок Б.5 – Слайд 5

Діаграма прецедентів застосунку



НУ "ЗП", Кафедра програмних засобів

6

Рисунок Б.6 – Слайд 6

Файлова структура проекту

```

|— images
|— start.py
|— config.py
|— box.py
|— captcha.py
|— admin.py
|— catalog.py
|— createtables.py
|— database.py
|— functions.py
|— main.py
|— murkups.py
|— payments.py
|— states.py
|— support.py
|— userfaq.py
|— usermenu.py

```



НУ "ЗП", Кафедра програмних засобів

7

Рисунок Б.7 – Слайд 7

Основні бібліотеки, що використані у програмі

Назва	Опис	Використання
aiogram	робудований на стандартній бібліотеці Python asyncio та асинхронному веб-фреймворку aiohttp, що дозволяє легко розширювати функціональність бота.	Дозволяє використовувати базові елементи.
psycopg2	надає зручний інтерфейс для роботи з PostgreSQL, включаючи підтримку транзакцій, курсорів та зовнішніх ключів	Дозволяє взаємодіяти з базою даних.
API Telegram Bot	це інтерфейс з HTTP протоколом для забезпечення взаємодії з ботами у Telegram	Для взаємодії з телеграм ботом
pickle	бібліотека pickle дозволяє зберігати об'єкти Python у файли, які можуть бути прочитані та використані пізніше.	Для роботи з файлами
asyncio	для асинхронного програмування у Python.	дозволяє створювати асинхронний код

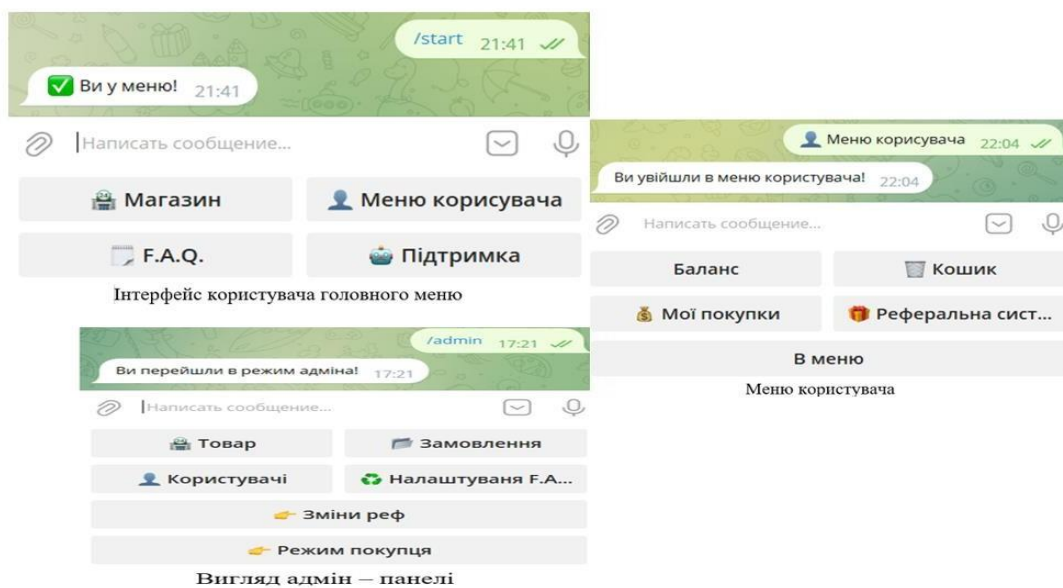


НУ "ЗП", Кафедра програмних засобів

8

Рисунок Б.8 – Слайд 8

Головне вікно застосунку

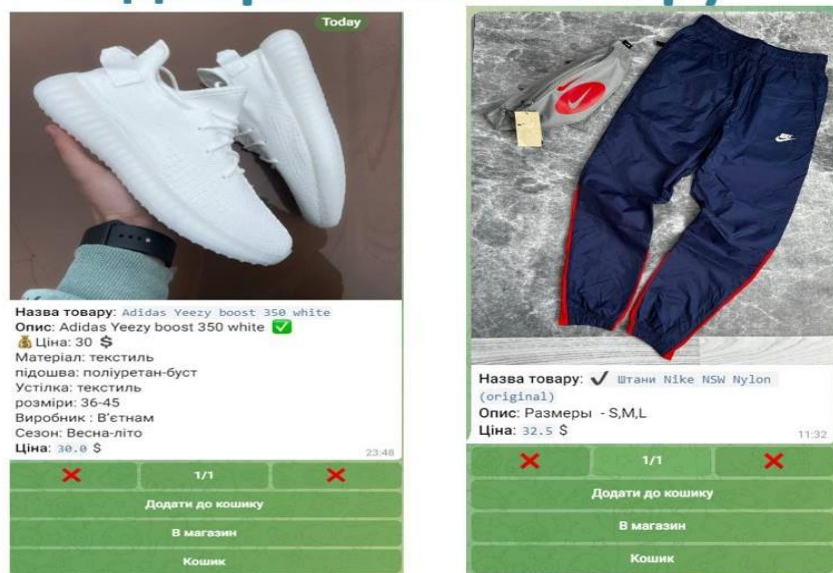


НУ "ЗП", Кафедра програмних засобів

9

Рисунок Б.9 – Слайд 9

Відображення товару



НУ "ЗП", Кафедра програмних засобів

10

Рисунок Б.10 – Слайд 10

Робота з балансом



Поповнення балансу



НУ "ЗП", Кафедра програмних засобів

11

Рисунок Б.11 – Слайд 11

Висновки

- проведено аналіз предметної області;
- проведено огляд програмних аналогів;
- розроблено технічне завдання;
- розроблено структуру програмного забезпечення;
- розроблено структуру програмного забезпечення;
- програмно реалізовано застосунок;
- розроблено необхідну документацію.



НУ "ЗП", Кафедра програмних засобів

12

Рисунок Б.12 – Слайд 12

**Доклад закінчено,
дякую за увагу!**



НУ “ЗП”, Кафедра програмних засобів

13

Рисунок Б.13 – Слайд 13