“Київський фаховий коледж зв’язку”

Циклова комісія комп’ютерної та програмної інженерії

**ЗВІТ ПО ВИКОНАННЮ**

**КОМПЛЕКСНОГО ТВОРЧОГО ЗАВДАННЯ**

з дисципліни: «Хмарні технології та сервіси»

**Тема: «BlinchikBot на базі BotFather на платформі AWS»**

Виконували

студенти)

групи РПЗ-24б

Мірошніченко А.Ю.

Танєєва А.І.

Туровський В.В.

Перевірила викладач

Сушанова В.С.

Київ 2025

## **Зміст**

[**Зміст** 1](#_Toc209783599)

[**Вступ** 1](#_Toc209783600)

[**2. Етап 1. Формування ідеї та вибір технологій** 2](#_Toc209783601)

[**3. Етап 2. Розгортання хмарного сервера** 3](#_Toc209783602)

[**4. Етап 3. Розгортання прикладного застосунку** 7](#_Toc209783603)

[**5.** **Висновки** 13](#_Toc209783604)

[**6.** **Список використаних джерел** 14](#_Toc209783605)

[**7. Додатки** 15](#_Toc209783606)

# **Вступ**

У сучасній цифровій епосі хмарні технології відіграють ключову роль у забезпеченні стабільної та масштабованої роботи застосунків. Вони дозволяють розробникам та компаніям ефективно управляти ресурсами, швидко розгортати сервіси та гарантувати їх високу доступність, що є особливо важливим для інтерактивних сервісів, таких як Telegram-боти. Актуальність використання хмарних рішень зумовлена необхідністю гнучкої інфраструктури, яка може адаптуватися до зростаючого навантаження, забезпечуючи при цьому надійність і безперебійність сервісів.

Мета цього завдання — розробити та розгорнути Telegram-бота, призначеного для обробки інтернет-замовлень, у хмарному середовищі Amazon Web Services (AWS).

Для досягнення поставленої мети необхідно виконати наступні завдання:

1. Підготувати Python-застосунок (Telegram-бота) до розгортання.
2. Налаштувати необхідну інфраструктуру в AWS.
3. Запустити та забезпечити безперебійну роботу бота в хмарному середовищі AWS.

# **2. Етап 1. Формування ідеї та вибір технологій**

**Назва проєкту:** Blinchik;

**Опис ідеї:** Розробити телеграм бота для обробки інтернет замовлень. Забезпечити зручніть і легкість використання;

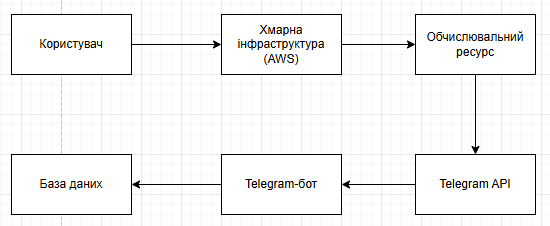
**Цільова аудиторія:** Клієнти будь-якого віку, які зацікавлені в створені замовлення;

**Вибраний стек технологій:**

* Backend: Python, JDS(Json Data Base);
* Frontend: Python, BotFather;
* База даних: NoSQL;
* Хмарна платформа: AWS.

**Обґрунтування вибору:** Було обрано мову програмування «Python», через великий стек зручних бібліотек, автоматизовану роботу з пам’яттю та об’єктно орієнтованістю. NoSQL є зручною СУБД зі зручною та гибкою структурою, яка ідеально підходить для роботи з об’єктно орієнтованими мовами програмування. BotFather – найвідоміший і зручніший інтрумент для розробки телеграм ботів, він є безкоштовним і загальнодоступним. AWS(Amazon Web Services) обрано через її масштабованість, гнучкість, економічну ефективність (оплата за використання), найширший набір хмарних сервісів та найбільшу глобальну інфраструктуру з високим рівнем надійності та безпеки.

**Схема архітектури (рисунок):**

**

# **3. Етап 2. Розгортання хмарного сервера**

**Обрана платформа:** Amazon Web Services

**Опис створення сервера:**

Повна інструкція по розгортанню сервера зберігається на GitHub, з назвою «AWS\_Instruction».

Створюємо Lambda проєкт. Додаємо слой, в який загружаємо бота архівованого в .zip архів та налаштовуємо конфігурацію платформи див. рис. 1.1.

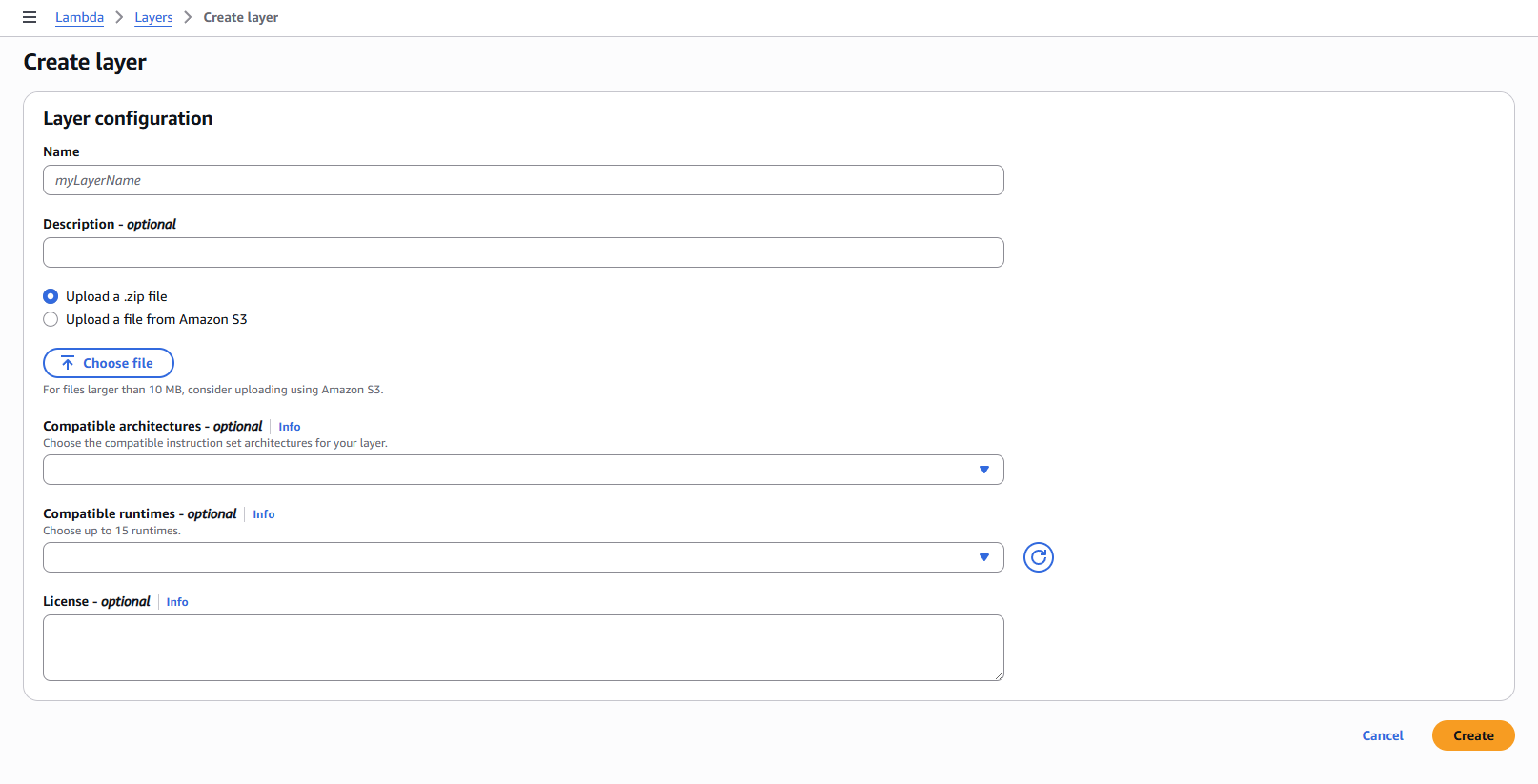


Рис. 1.1 «Конфігурація платформи»

Після цього потрібно створити нову функцію. Також налаштовуюємо конфігурацію. Важливо щоб вона не відрізнялась від налаштувань проєкту (крім назви і опису), оскільки у останній версії Python 3.13.0 при роботі із API може виникати конфлікти (при використанні бібліотеки «aiofiles»). Після цього відкриється вікно з діаграмою проєкта, потрібно підв’язати слой до функції (обираємо потрібний слой і підтверджуємо).

Далі необхідно створити API для цього бота, щоб він міг працювати з AWS. Заповнююмо форму зображену на рис. 1.2 і завантажуємо архів з ботом.

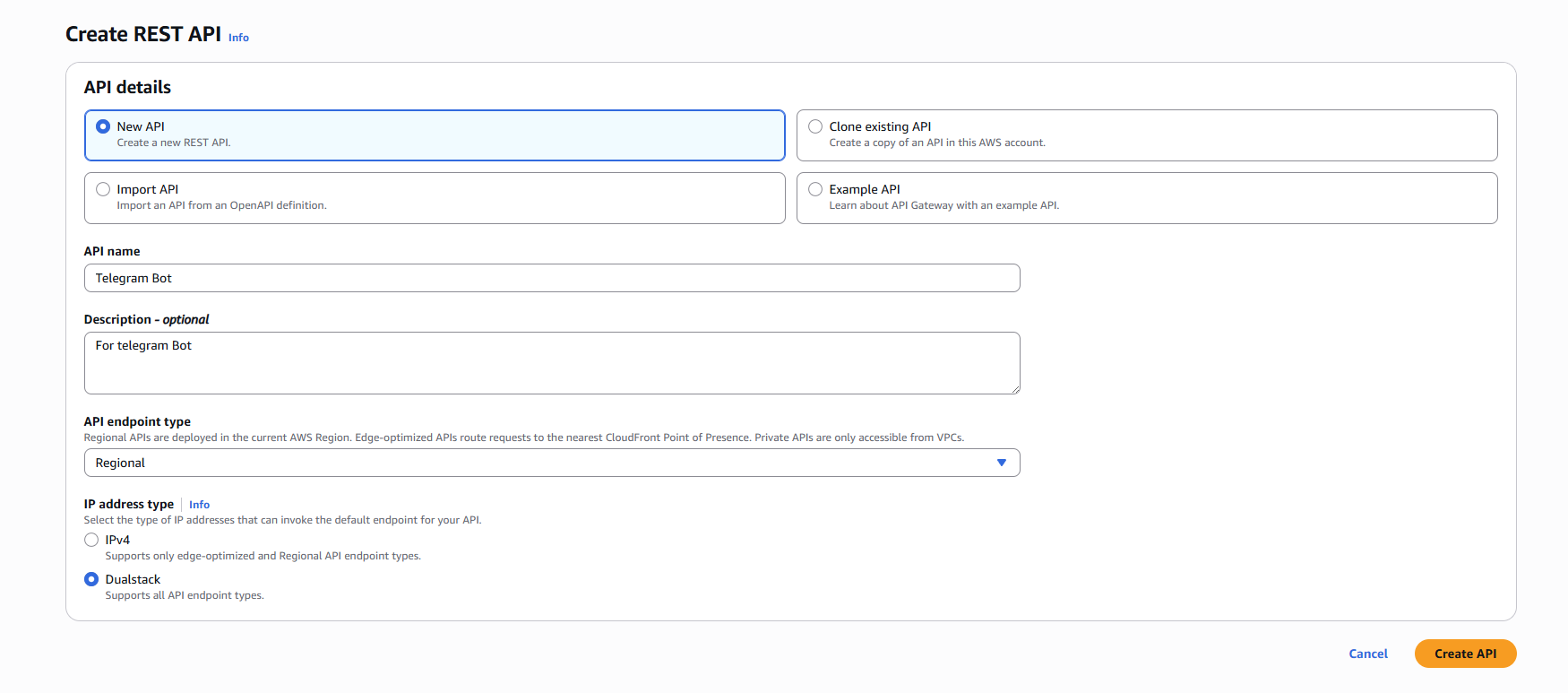


Рис. 1.2 «Форма створення API»

І остання дія, треба заповнити стартер боту. На кожній мові він може відрізнятись але зазвичай це виглядає наступним чином:

“Змінна що зберігає ключ бота” = “Ключ бота”

В цьому боті ще є айді телеграм аккаунту адміна (поянення в етапі 3), тому стартер зображено на рис. 1.3.

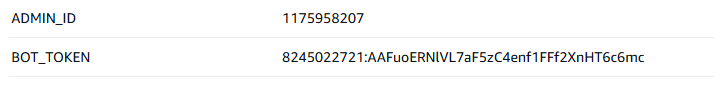


Рис. 1.3 «Стартер бота»

**Кроки налаштування середовища (команди, скрипти):**

Ключ підключення бота: 8245022721:AAFuoERNlVL7aF5zC4enf1FFf2XnHT6c6mc

Invoke URL: https://m4krbscki4.execute-api.eu-north-1.amazonaws.com/V0

Ключ ARN: arn:aws:lambda:eu-north-1:269825205566:layer:BlinchikChatBot:1

Запуск бота: python bot.py

Файл із підключенням необхідних бібліотек знаходиться на GitHub, з назвою «requirements.txt».

Оскільки телеграм зберігає зображення на власному сервері, потрібно дізнатись їхній айді, щоб вони відображались для кожного користувача. Для цього потрібен наступний код:

@router.message(F.photo)

async def get\_photo\_id(message: types.Message):

photo\_id = message.photo[-1].file\_id

await message.answer(f"ID цього фото: `{photo\_id}`")

**Скріншоти:**

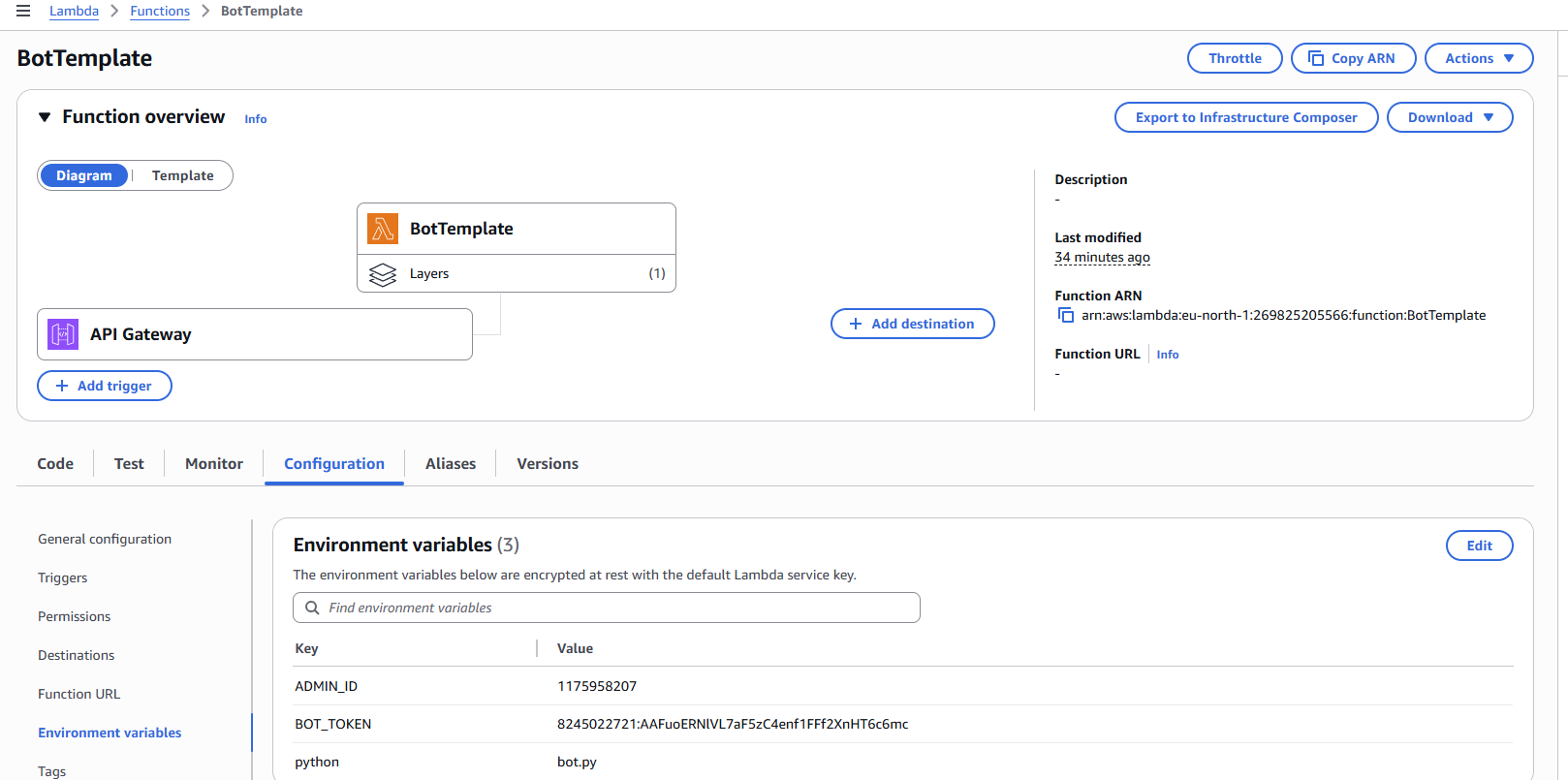


Рис. 2.1 «Створений сервер, діаграма сервера»

Скріншот 2.2 створити неможливо, оскільки в AWS Lambda немає постійної IP-адреси чи порту для SSH-підключення. Код завантажується безпосередньо через веб-інтерфейс або через S3/CLI.

  
Рис.2.3 «Середовище та конфігурація»

AWS Lambda не маєте звичайного робочого столу чи довготривалого сервера, код виконується в ізольованому (serverless) контейнері. В цей контейнер завантажено середовище Python 3.13.0.

# **4. Етап 3. Розгортання прикладного застосунку**

**Короткий опис застосунку:**

Це повнофункціональний Telegram-бот, розроблений на aiogram v3, який створює міні-магазин комп'ютерних комплектуючих. Бот дозволяє користувачам переглядати каталог товарів з фото та описом, додавати вибрані позиції до кошика, змінювати кількість товарів у ньому, а потім переходити до покрокового оформлення замовлення із симуляцією оплати. Уся логіка побудована на вбудованих клавіатурах, що забезпечує інтуїтивно зрозумілу навігацію. Крім того, бот містить адміністративну панель, яка доступна за командою /orders і дозволяє керувати новими замовленнями, переглядати деталі та позначати їх як відправлені, а також автоматично сповіщає адміністратора про нове оплачене замовлення.

**Функціональність реалізована:**

Функціонал для користувача (Магазин)

1. Навігація та меню: Користувач розпочинає роботу з командою /start, яка виводить головне меню. Уся подальша взаємодія відбувається через вбудовані клавіатури (Inline Keyboards).
2. Каталог товарів: Доступний повний список товарів з описом, цінами та інформацією про наявність.
3. Перегляд товару: При натисканні на товар відображається його детальна картка, включно з фотографією (використовуючи file\_id Telegram) та кнопкою "Додати в кошик".
4. Кошик (Cart): Дозволяє додавати, видаляти та змінювати кількість товарів. Користувач може переглянути загальну суму замовлення та очистити кошик.
5. Оформлення замовлення (Checkout): Реалізований покроковий процес (FSMState) для збору інформації:

* Ім'я.
* Телефон.
* Адреса доставки.
* Фінальне підтвердження з деталізацією замовлення та загальною сумою.

1. Симуляція оплати: Після підтвердження виконується симуляція успішної оплати, замовленню присвоюється унікальний ID, і воно зберігається у системі.
2. "Мої замовлення": Користувач може переглянути список своїх попередніх замовлень та їхні статуси.

Функціонал для адміністратора (Управління)

1. Доступ з обмеженням: Адміністративний функціонал доступний лише користувачеві з попередньо встановленим ADMIN\_ID.
2. Система замовлень: Усі оплачені замовлення зберігаються у файлі orders.json (у Lambda використовується тимчасове сховище /tmp/orders.json).
3. Команда /orders: Дозволяє адміністратору:

* Переглянути список усіх замовлень із загальною сумою та статусом.
* Побачити деталі будь-якого замовлення.
* Змінити статус замовлення (наприклад, "Позначити як відправлене").

1. Сповіщення: Бот автоматично надсилає адміністратору повідомлення прокожне нове оформлене та оплачене замовлення.

**Підготовка застосунку до деплою (налаштування):**

1. Код бота архівовано в .zip;
2. Створено Бот-стартер;
3. Перенесено усі необхідні бібліотеки в окремий файл;
4. Увесь код структуровано для зручності роботи.

**Кроки деплою:**

Код повністю підготовлений для передачі чи загрузки у хмару. Для зручності його завантажено на GitHub, Самого бота розбито на декілька документів для уникнення конфліктів під час завантаження, при виникненні помилки буде одразу зрозуміло що пійшло не так. Оскільки код архівовано, хмара дозволяє коректно його завантажити.

**Перевірка роботи застосунку (скріншоти):**

Інтерфейс роботи з ботом див. рис. 3.1 – 3.6.

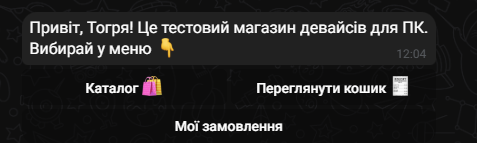
****

Рис. 3.1 «Початок роботи з ботом»



Рис. 3.2 «Каталог техніки»

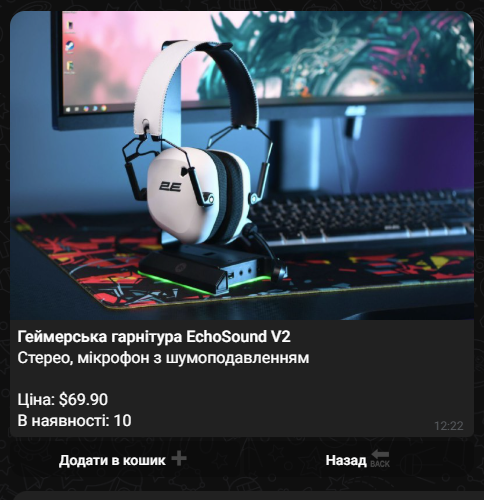


Рис. 3.3 «Обрана техніка»

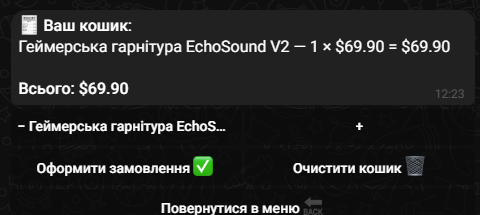


Рис 3.4 «Кошик з доданою технікою»

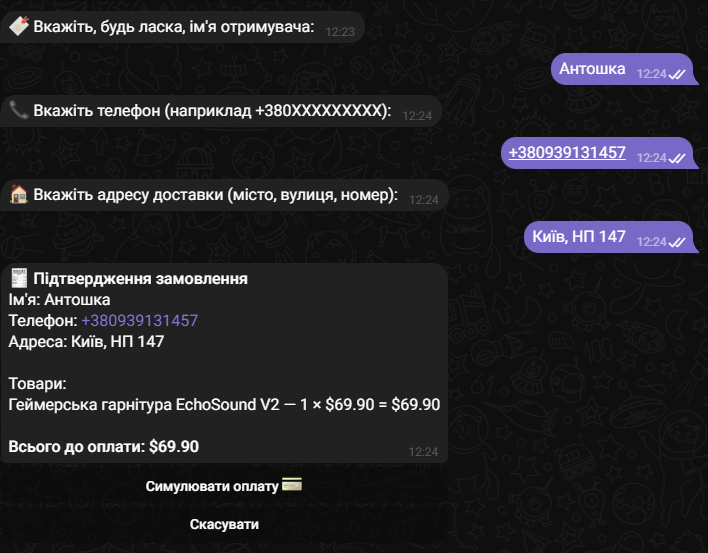


Рис. 3.5 «Інтерфейс оформлення замовлення»

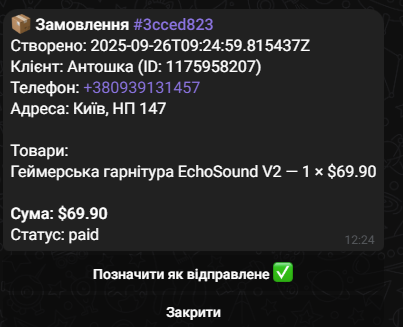


Рис. 3.6 «Адмінський інтерфейс отримання замовлення»

В цьому проєкті використувається база даних NoSQL. Замовлення зберігаються в документі orders.json див. Рис. 3. 7.

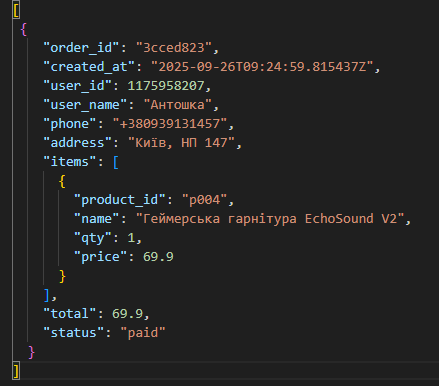
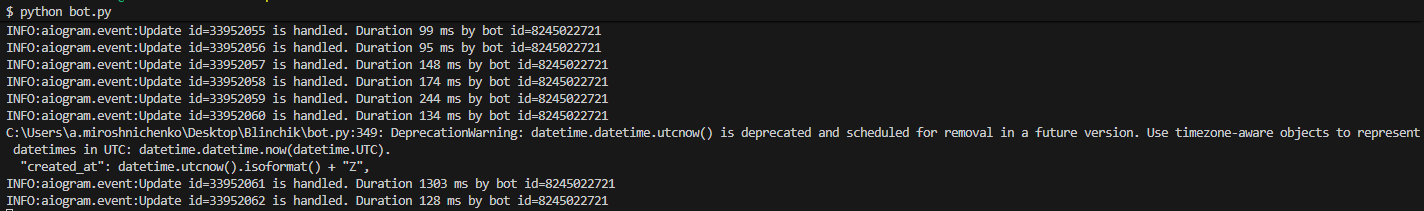


Рис. 3.7 «Збереження замовлення в NoSQL БД»

Роботу програми зображено на рис. 3.1 – 3.6. На рис. 3.8 показано відображення результату взаємодії користувача з ботом в логах сервера

  
Рис. 3.8 «Логи сервера»

# **Висновки**

Під час виконання роботи було отримано навички роботи з NoSQL базою даних, з розгортанням серверу черз веб сервіси і подальшого встановленням свого продукту на хмару. Отримано навички розробки чат ботів, планування і тестування цього продукту.

Серед труднощів було запустити бота на сервері. В перших версіях програми він запускався напряму через команду. Але для роботи з AWS було перероблено цю механіку на стартер.

Серед переваг використання хмарних сервісів хочеться зазначити зручність у використанні. При відсутності можливості створити власний сервіс це може економити як гроші, так і час з зусиллям. Із недоліків, потрібен час на засвоєння нових технологій, самої платформи. Наразі усі платформи є або частково безкоштовними, або повністю платними.

# **Список використаних джерел**

1. Деплой Телеграм Бота на AWS. *YouTube* : [відео]. 2017. URL: <https://www.youtube.com/watch?v=BPtiabGGOts>

2. Amazon Web Services. *Amazon Web Services* : вебсайт. URL: <https://aws.amazon.com/>

3. Технологія Push to Deploy. *Freehost* : вебсторінка. URL: <https://freehost.com.ua/ukr/faq/articles/tehnologija-push-to-deploy/>

4. yarl. *PyPI* : вебсторінка. URL: <https://pypi.org/project/yarl/>

5. aiofiles. *PyPI* : вебсторінка. URL: <https://pypi.org/project/aiofiles/>

# **7. Додатки**

Bot.py

#!/usr/bin/env python3

"""

Telegram Shop Bot (aiogram v3)

Features:

- Catalog of PC devices

- Cart (add/remove/view)

- Checkout flow with simulated payment

- Orders saved to orders.json

- Admin /orders command + order details + mark shipped

Requirements:

pip install aiogram python-dotenv

"""

import os

import json

import asyncio

import time

import logging

from uuid import uuid4

from functools import wraps

from datetime import datetime

from typing import Dict, Any

from aiogram import Bot, Dispatcher, types, F, Router

from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton, Message

from aiogram.filters.command import Command

from aiogram.filters.callback\_data import CallbackData

from aiogram.fsm.storage.memory import MemoryStorage

from aiogram.fsm.context import FSMContext

from aiogram.fsm.state import StatesGroup, State

from aiogram.enums import ParseMode

# --- CONFIG ---

import os

BOT\_TOKEN = os.getenv("BOT\_TOKEN")

ADMIN\_ID = os.getenv("ADMIN\_ID")

# Перевірка, чи змінні оточення були встановлені

if not BOT\_TOKEN:

    print("Error: BOT\_TOKEN environment variable not set.")

    exit(1)

if not ADMIN\_ID:

    print("Error: ADMIN\_ID environment variable not set.")

    exit(1)

# Конвертуємо ADMIN\_ID у ціле число, оскільки getenv повертає рядок

ADMIN\_ID = int(ADMIN\_ID)

# --- logging ---

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

# --- Initialization ---

dp = Dispatcher(storage=MemoryStorage())

bot = Bot(token=BOT\_TOKEN)

router = Router()

ORDERS\_FILE = "orders.json"

# --- Sample products (invented) ---

PRODUCTS = {

    "p001": {"id": "p001", "name": "Механічна клавіатура RedStorm K8", "price": 79.99, "stock": 12, "desc": "RGB, Cherry-like switches, USB-C", "photo\_id": "AgACAgIAAxkBAAM3aNUi5BWdnV3EMgHXSsr1beLptmYAAk\_\_MRswAbFKCoYCDM8-gywBAAMCAAN5AAM2BA"},

    "p002": {"id": "p002", "name": "Оптична миша SilentPro M3", "price": 29.50, "stock": 25, "desc": "2400 DPI, безшумні кнопки", "photo\_id": "AgACAgIAAxkBAAM5aNUi96Rc3QABHAs1qfI1SOSSzz5jAAJU\_zEbMAGxSop9z6ZYCe3LAQADAgADeAADNgQ"},

    "p003": {"id": "p003", "name": "Ігрова миша HyperX Nova", "price": 54.00, "stock": 8, "desc": "6400 DPI, програмовані кнопки", "photo\_id": "AgACAgIAAxkBAAM1aNUiyqVCLSuURhuPEcE8ZyVM7OUAAk7\_MRswAbFK5U8-QTEUnZEBAAMCAAN4AAM2BA"},

    "p004": {"id": "p004", "name": "Геймерська гарнітура EchoSound V2", "price": 69.90, "stock": 10, "desc": "Стерео, мікрофон з шумоподавленням", "photo\_id": "AgACAgIAAxkBAAMxaNUinvsLY4jiFqq9nUMjkQ29T1cAAkv\_MRswAbFKuNvpraYM1jEBAAMCAAN5AAM2BA"},

    "p005": {"id": "p005", "name": "Підставка для ноутбука CoolLift", "price": 19.99, "stock": 30, "desc": "Алюмінієва, регульована", "photo\_id": "AgACAgIAAxkBAAMvaNUihPxJB2M\_muFjG\_Mbt5HvUfQAAkn\_MRswAbFKvg-AORBBfB8BAAMCAAN4AAM2BA"},

    "p006": {"id": "p006", "name": "Килимок для миші XXL Glide", "price": 14.99, "stock": 50, "desc": "Великий, гумове підставлення", "photo\_id": "AgACAgIAAxkBAAMzaNUitzrnLlkAAY49olxnb1NPusvDAAJN\_zEbMAGxSi5ZTqvjIlVSAQADAgADeAADNgQ"},

    "p007": {"id": "p007", "name": "Вебкамера ClearView 1080p", "price": 39.99, "stock": 15, "desc": "Full HD, вбудований мікрофон", "photo\_id": "AgACAgIAAxkBAAMtaNUia5rWbFDfAAFhX3WYOeAnd2MVAAJH\_zEbMAGxSuVHQfn-C5oMAQADAgADeAADNgQ"},

    "p008": {"id": "p008", "name": "Підставка-підсилювач USB Hub 4 порти", "price": 24.90, "stock": 20, "desc": "USB 3.0, LED індикатор", "photo\_id": "AgACAgIAAxkBAAM7aNUjB0rBX6eAtEYy6gcddCWdfaoAAmD\_MRswAbFKWXWagaH8ry0BAAMCAAN4AAM2BA"}

}

# --- FSM States ---

class Checkout(StatesGroup):

    ask\_name = State()

    ask\_phone = State()

    ask\_address = State()

    confirm = State()

# --- Callback data ---

class CatalogCallback(CallbackData, prefix="cat"):

    action: str

    product\_id: str

class CartCallback(CallbackData, prefix="cart"):

    action: str

    product\_id: str

class OrderCallback(CallbackData, prefix="order"):

    action: str

    order\_id: str

# --- Utilities ---

def load\_orders():

    if not os.path.exists(ORDERS\_FILE):

        with open(ORDERS\_FILE, "w", encoding="utf-8") as f:

            json.dump([], f, indent=2, ensure\_ascii=False)

        return []

    with open(ORDERS\_FILE, "r", encoding="utf-8") as f:

        try:

            return json.load(f)

        except json.JSONDecodeError:

            return []

def save\_orders(orders):

    with open(ORDERS\_FILE, "w", encoding="utf-8") as f:

        json.dump(orders, f, indent=2, ensure\_ascii=False)

def require\_admin(func):

    @wraps(func)

    async def wrapper(message: Message, \*args, \*\*kwargs):

        user\_id = message.from\_user.id

        if ADMIN\_ID is None:

            await message.answer("Адмін ID не налаштований.")

            return

        if user\_id != ADMIN\_ID:

            await message.answer("Доступ заборонено. Ця команда тільки для адміністратора.")

            return

        return await func(message, \*args, \*\*kwargs)

    return wrapper

def format\_price(p):

    return f"${p:.2f}"

# --- Keyboards ---

def main\_menu\_kb():

    kb = InlineKeyboardMarkup(row\_width=2, inline\_keyboard=[

        [InlineKeyboardButton(text="Каталог 🛍️", callback\_data="catalog"),

         InlineKeyboardButton(text="Переглянути кошик 🧾", callback\_data="view\_cart")],

        [InlineKeyboardButton(text="Мої замовлення", callback\_data="my\_orders")]

    ])

    return kb

def product\_kb(product\_id):

    kb = InlineKeyboardMarkup(row\_width=2, inline\_keyboard=[

        [InlineKeyboardButton(text="Додати в кошик ➕", callback\_data=CatalogCallback(action="add", product\_id=product\_id).pack()),

         InlineKeyboardButton(text="Назад 🔙", callback\_data="catalog")]

    ])

    return kb

async def cart\_kb(user\_id: int, state: FSMContext):

    data = await state.get\_data()

    cart = data.get("cart", {})

    kb = InlineKeyboardMarkup(row\_width=2, inline\_keyboard=[])

    for pid, qty in cart.items():

        kb.inline\_keyboard.append([

            InlineKeyboardButton(text=f"− {PRODUCTS[pid]['name']} ({qty})", callback\_data=CartCallback(action="decrease", product\_id=pid).pack()),

            InlineKeyboardButton(text="+", callback\_data=CartCallback(action="increase", product\_id=pid).pack())

        ])

    if cart:

        kb.inline\_keyboard.append([

            InlineKeyboardButton(text="Оформити замовлення ✅", callback\_data="checkout"),

            InlineKeyboardButton(text="Очистити кошик 🗑️", callback\_data="clear\_cart")

        ])

    kb.inline\_keyboard.append([InlineKeyboardButton(text="Повернутися в меню 🔙", callback\_data="menu")])

    return kb

# --- Handlers ---

@router.message(Command("start"))

async def cmd\_start(message: Message, state: FSMContext):

    user = message.from\_user

    await state.set\_data({"cart": {}})

    text = f"Привіт, {user.first\_name}! Це тестовий магазин девайсів для ПК.\nВибирай у меню 👇"

    await message.answer(text, reply\_markup=main\_menu\_kb())

@router.callback\_query(F.data == "menu")

async def cb\_menu(query: types.CallbackQuery):

    await query.message.edit\_text("Головне меню:", reply\_markup=main\_menu\_kb())

    await query.answer()

@router.callback\_query(F.data == "catalog")

async def cb\_catalog(query: types.CallbackQuery):

    text\_lines = ["🛍️ <b>Каталог товарів</b>:"]

    kb = InlineKeyboardMarkup(row\_width=1, inline\_keyboard=[])

    for pid, p in PRODUCTS.items():

        text\_lines.append(f"\n<b>{p['name']}</b>\n{p['desc']}\nЦіна: {format\_price(p['price'])} | В наявності: {p['stock']}")

        kb.inline\_keyboard.append([InlineKeyboardButton(text=f"Дивитись: {p['name']}", callback\_data=CatalogCallback(action="view", product\_id=pid).pack())])

    kb.inline\_keyboard.append([InlineKeyboardButton(text="Повернутися в меню 🔙", callback\_data="menu")])

    # Видаляємо старе повідомлення і надсилаємо нове, щоб уникнути помилок при редагуванні

    await bot.delete\_message(chat\_id=query.message.chat.id, message\_id=query.message.message\_id)

    await bot.send\_message(chat\_id=query.message.chat.id, text="\n".join(text\_lines), parse\_mode=ParseMode.HTML, reply\_markup=kb)

    await query.answer()

@router.callback\_query(CatalogCallback.filter(F.action == "view"))

async def cb\_catalog\_view(query: types.CallbackQuery, callback\_data: CatalogCallback):

    pid = callback\_data.product\_id

    p = PRODUCTS.get(pid)

    if not p or 'photo\_id' not in p:

        await query.answer("Товар не знайдено або фото відсутнє", show\_alert=True)

        return

    text = f"<b>{p['name']}</b>\n{p['desc']}\n\nЦіна: {format\_price(p['price'])}\nВ наявності: {p['stock']}"

    # Видаляємо старе повідомлення

    await bot.delete\_message(chat\_id=query.message.chat.id, message\_id=query.message.message\_id)

    # Відправляємо нове повідомлення з фото

    try:

        await bot.send\_photo(

            chat\_id=query.message.chat.id,

            photo=p['photo\_id'],

            caption=text,

            parse\_mode=ParseMode.HTML,

            reply\_markup=product\_kb(pid)

        )

        await query.answer()

    except Exception as e:

        await query.answer(f"Помилка при відправці фото: {e}", show\_alert=True)

        print(f"Помилка: {e}")

@router.callback\_query(CatalogCallback.filter(F.action == "add"))

async def cb\_catalog\_add(query: types.CallbackQuery, callback\_data: CatalogCallback, state: FSMContext):

    pid = callback\_data.product\_id

    p = PRODUCTS.get(pid)

    if not p:

        await query.answer("Товар не знайдено", show\_alert=True)

        return

    data = await state.get\_data()

    cart = data.get("cart", {})

    cart[pid] = cart.get(pid, 0) + 1

    await state.update\_data(cart=cart)

    await query.answer(f"Додано {p['name']} до кошика ✅")

@router.callback\_query(F.data == "view\_cart")

async def cb\_view\_cart(query: types.CallbackQuery, state: FSMContext):

    data = await state.get\_data()

    cart = data.get("cart", {})

    if not cart:

        await query.answer()

        await query.message.edit\_text("Кошик порожній. Додайте товари з каталогу.", reply\_markup=main\_menu\_kb())

        return

    lines = ["🧾 <b>Ваш кошик</b>:"]

    total = 0.0

    for pid, qty in cart.items():

        p = PRODUCTS[pid]

        subtotal = p["price"] \* qty

        total += subtotal

        lines.append(f"{p['name']} — {qty} × {format\_price(p['price'])} = {format\_price(subtotal)}")

    lines.append(f"\n<b>Всього: {format\_price(total)}</b>")

    await query.message.edit\_text("\n".join(lines), parse\_mode=ParseMode.HTML, reply\_markup=await cart\_kb(query.from\_user.id, state))

    await query.answer()

@router.callback\_query(CartCallback.filter())

async def cb\_cart\_actions(query: types.CallbackQuery, callback\_data: CartCallback, state: FSMContext):

    action = callback\_data.action

    pid = callback\_data.product\_id

    data = await state.get\_data()

    cart = data.get("cart", {})

    if pid not in cart:

        await query.answer("Товар не в кошику", show\_alert=True)

        return

    if action == "increase":

        cart[pid] += 1

    elif action == "decrease":

        cart[pid] -= 1

        if cart[pid] <= 0:

            del cart[pid]

    await state.update\_data(cart=cart)

    await query.answer("Оновлено кошик")

    await cb\_view\_cart(query, state)

@router.callback\_query(F.data == "clear\_cart")

async def cb\_clear\_cart(query: types.CallbackQuery, state: FSMContext):

    await state.update\_data(cart={})

    await query.answer("Кошик очищено")

    await query.message.edit\_text("Кошик очищено.", reply\_markup=main\_menu\_kb())

# --- Checkout flow ---

@router.callback\_query(F.data == "checkout")

async def cb\_checkout(query: types.CallbackQuery, state: FSMContext):

    data = await state.get\_data()

    cart = data.get("cart", {})

    if not cart:

        await query.answer("Кошик порожній", show\_alert=True)

        return

    await state.set\_state(Checkout.ask\_name)

    await query.message.answer("🔖 Вкажіть, будь ласка, ім'я отримувача:")

    await query.answer()

@router.message(Checkout.ask\_name)

async def process\_name(message: Message, state: FSMContext):

    await state.update\_data(name=message.text.strip())

    await state.set\_state(Checkout.ask\_phone)

    await message.answer("📞 Вкажіть телефон (наприклад +380XXXXXXXXX):")

@router.message(Checkout.ask\_phone)

async def process\_phone(message: Message, state: FSMContext):

    await state.update\_data(phone=message.text.strip())

    await state.set\_state(Checkout.ask\_address)

    await message.answer("🏠 Вкажіть адресу доставки (місто, вулиця, номер):")

@router.message(Checkout.ask\_address)

async def process\_address(message: Message, state: FSMContext):

    await state.update\_data(address=message.text.strip())

    data = await state.get\_data()

    cart = data.get("cart", {})

    lines = [f"🧾 <b>Підтвердження замовлення</b>\nІм'я: {data['name']}\nТелефон: {data['phone']}\nАдреса: {data['address']}\n\nТовари:"]

    total = 0.0

    for pid, qty in cart.items():

        p = PRODUCTS[pid]

        subtotal = p["price"] \* qty

        total += subtotal

        lines.append(f"{p['name']} — {qty} × {format\_price(p['price'])} = {format\_price(subtotal)}")

    lines.append(f"\n<b>Всього до оплати: {format\_price(total)}</b>")

    kb = InlineKeyboardMarkup(inline\_keyboard=[

        [InlineKeyboardButton(text="Симулювати оплату 💳", callback\_data="simulate\_pay")],

        [InlineKeyboardButton(text="Скасувати", callback\_data="cancel\_checkout")]

    ])

    await state.set\_state(Checkout.confirm)

    await message.answer("\n".join(lines), parse\_mode=ParseMode.HTML, reply\_markup=kb)

@router.callback\_query(F.data == "cancel\_checkout", Checkout.confirm)

async def cb\_cancel\_checkout(query: types.CallbackQuery, state: FSMContext):

    await state.clear()

    await query.message.edit\_text("Оформлення замовлення скасовано.", reply\_markup=main\_menu\_kb())

    await query.answer()

@router.callback\_query(F.data == "simulate\_pay", Checkout.confirm)

async def cb\_simulate\_payment(query: types.CallbackQuery, state: FSMContext):

    uid = query.from\_user.id

    data = await state.get\_data()

    cart = data.get("cart", {})

    if not data or not cart:

        await query.answer("Немає даних для оплати", show\_alert=True)

        return

    await query.answer("Проведення оплати...", show\_alert=False)

    await asyncio.sleep(1)

    orders = load\_orders()

    order\_id = str(uuid4())[:8]

    total = sum(PRODUCTS[pid]["price"] \* qty for pid, qty in cart.items())

    order = {

        "order\_id": order\_id,

        "created\_at": datetime.utcnow().isoformat() + "Z",

        "user\_id": uid,

        "user\_name": data["name"],

        "phone": data["phone"],

        "address": data["address"],

        "items": [{"product\_id": pid, "name": PRODUCTS[pid]["name"], "qty": qty, "price": PRODUCTS[pid]["price"]} for pid, qty in cart.items()],

        "total": total,

        "status": "paid"

    }

    orders.append(order)

    save\_orders(orders)

    await state.clear()

    text = f"✅ Оплата пройшла успішно!\nНомер вашого замовлення: <b>{order\_id}</b>\nМи зв'яжемося з вами для підтвердження."

    await query.message.edit\_text(text, parse\_mode=ParseMode.HTML, reply\_markup=main\_menu\_kb())

    await query.answer("Оплата успішна ✅")

    if ADMIN\_ID:

        try:

            order\_text = f"📬 Нове замовлення #{order\_id}\nКлієнт: {order['user\_name']} ({order['phone']})\nАдреса: {order['address']}\nСума: {format\_price(order['total'])}"

            kb = InlineKeyboardMarkup(inline\_keyboard=[[InlineKeyboardButton(text="Деталі", callback\_data=OrderCallback(action="details", order\_id=order\_id).pack())]])

            await bot.send\_message(ADMIN\_ID, order\_text, reply\_markup=kb)

        except Exception as e:

            logger.exception("Не вдалось надіслати адміну повідомлення: %s", e)

# --- Admin: list orders ---

@router.message(Command("orders"))

@require\_admin

async def cmd\_orders(message: Message):

    orders = load\_orders()

    if not orders:

        await message.answer("Поки що немає замовлень.")

        return

    kb = InlineKeyboardMarkup(row\_width=1, inline\_keyboard=[])

    for o in orders[::-1]:

        short = f"#{o['order\_id']} — {format\_price(o['total'])} — {o['status']}"

        kb.inline\_keyboard.append([InlineKeyboardButton(text=short, callback\_data=OrderCallback(action="details", order\_id=o["order\_id"]).pack())])

    await message.answer("Список замовлень:", reply\_markup=kb)

@router.callback\_query(OrderCallback.filter())

async def cb\_order\_actions(query: types.CallbackQuery, callback\_data: OrderCallback):

    action = callback\_data.action

    oid = callback\_data.order\_id

    orders = load\_orders()

    order = next((o for o in orders if o["order\_id"] == oid), None)

    if not order:

        await query.answer("Замовлення не знайдено", show\_alert=True)

        return

    if action == "details":

        lines = [

            f"📦 <b>Замовлення #{order['order\_id']}</b>",

            f"Створено: {order['created\_at']}",

            f"Клієнт: {order['user\_name']} (ID: {order['user\_id']})",

            f"Телефон: {order['phone']}",

            f"Адреса: {order['address']}",

            "\nТовари:"

        ]

        for it in order["items"]:

            lines.append(f"{it['name']} — {it['qty']} × {format\_price(it['price'])}")

        lines.append(f"\n<b>Сума: {format\_price(order['total'])}</b>")

        lines.append(f"Статус: {order['status']}")

        kb = InlineKeyboardMarkup(inline\_keyboard=[])

        if order["status"] != "shipped":

            kb.inline\_keyboard.append([InlineKeyboardButton(text="Позначити як відправлене ✅", callback\_data=OrderCallback(action="mark\_shipped", order\_id=oid).pack())])

        kb.inline\_keyboard.append([InlineKeyboardButton(text="Закрити", callback\_data="menu")])

        await query.message.edit\_text("\n".join(lines), parse\_mode=ParseMode.HTML, reply\_markup=kb)

        await query.answer()

    elif action == "mark\_shipped":

        for o in orders:

            if o["order\_id"] == oid:

                o["status"] = "shipped"

                break

        save\_orders(orders)

        await query.answer("Позначено як відправлене")

        await cb\_order\_actions(query, callback\_data=OrderCallback(action="details", order\_id=oid))

# --- User: view own orders ---

@router.callback\_query(F.data == "my\_orders")

async def cb\_my\_orders(query: types.CallbackQuery):

    user\_id = query.from\_user.id

    orders = load\_orders()

    user\_orders = [o for o in orders if o["user\_id"] == user\_id]

    if not user\_orders:

        await query.answer()

        await query.message.edit\_text("У вас поки немає замовлень.", reply\_markup=main\_menu\_kb())

        return

    lines = ["Ваші замовлення:"]

    kb = InlineKeyboardMarkup(row\_width=1, inline\_keyboard=[])

    for o in user\_orders[::-1]:

        lines.append(f"#{o['order\_id']} — {format\_price(o['total'])} — {o['status']}")

        kb.inline\_keyboard.append([InlineKeyboardButton(text=f"Деталі #{o['order\_id']}", callback\_data=OrderCallback(action="user\_details", order\_id=o["order\_id"]).pack())])

    kb.inline\_keyboard.append([InlineKeyboardButton(text="Повернутися в меню 🔙", callback\_data="menu")])

    await query.message.edit\_text("\n".join(lines), reply\_markup=kb)

    await query.answer()

@router.callback\_query(OrderCallback.filter(F.action == "user\_details"))

async def cb\_user\_order\_details(query: types.CallbackQuery, callback\_data: OrderCallback):

    oid = callback\_data.order\_id

    orders = load\_orders()

    order = next((o for o in orders if o["order\_id"] == oid and o["user\_id"] == query.from\_user.id), None)

    if not order:

        await query.answer("Замовлення не знайдено", show\_alert=True)

        return

    lines = [

        f"📦 <b>Замовлення #{order['order\_id']}</b>",

        f"Статус: {order['status']}",

        f"Сума: {format\_price(order['total'])}",

        "\nТовари:"

    ]

    for it in order["items"]:

        lines.append(f"{it['name']} — {it['qty']} × {format\_price(it['price'])}")

    kb = InlineKeyboardMarkup(inline\_keyboard=[[InlineKeyboardButton(text="Повернутися", callback\_data="my\_orders")]])

    await query.message.edit\_text("\n".join(lines), parse\_mode=ParseMode.HTML, reply\_markup=kb)

    await query.answer()

# --- Fallback text handler for help ---

@router.message(Command("help"))

async def cmd\_help(message: Message):

    await message.answer("/start — запустити бота\nКаталог, Кошик, Оформлення замовлення через меню.\nАдмін: /orders")

# --- Temporary photo ID handler ---

@router.message(F.photo)

async def get\_photo\_id(message: types.Message):

    photo\_id = message.photo[-1].file\_id

    await message.answer(f"ID цього фото: `{photo\_id}`")

async def main():

    dp.include\_router(router)

    print("Bot starting...")

    await dp.start\_polling(bot, skip\_updates=True)

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

requirement.txt

aiofiles==24.1.0

aiogram==3.22.0

aiohappyeyeballs==2.6.1

aiohttp==3.12.15

aiosignal==1.4.0

annotated-types==0.7.0

attrs==25.3.0

certifi==2025.8.3

frozenlist==1.7.0

idna==3.10

magic-filter==1.0.12

multidict==6.6.4

propcache==0.3.2

pydantic==2.11.9

pydantic\_core==2.33.2

python-dotenv==1.1.1

typing-inspection==0.4.1

typing\_extensions==4.15.0

yarl==1.20.1

startBot.txt

export BOT\_TOKEN="8245022721:AAFuoERNlVL7aF5zC4enf1FFf2XnHT6c6mc"

export ADMIN\_ID="1175958207"

python bot.py

Github: <https://github.com/TigroviyRyletik/Gurren_Lagann>

Bot Name in Telegram: @Blinchik322\_Bot