Вот текст, извлечённый из изображения:

1. Настроить Django-приложение так, чтобы оно отправляло POST-запросы на адрес бота, например, http://127.0.0.1:8001/webhook. Бот должен быть запущен на другом порту (например, 8001), чтобы не конфликтовать с Django.
2. Запустить бот как приложение FastAPI или Flask, чтобы обрабатывать вебхуки, или использовать ту же библиотеку aiogram с поддержкой вебхуков.
3. **Настройка локального взаимодействия**
4. **1. Django: Отправка данных в вебхук**
5. Создайте функцию во views.py в Django-приложении для отправки данных в бот:

import requests

from django.http import JsonResponse

def update\_order\_status(request, order\_id):

# Пример данных для отправки

order\_key = "example\_key"

status = "Обновлен"

webhook\_url = "http://127.0.0.1:8001/webhook"

try:

response = requests.post(webhook\_url, json={"order\_id": order\_id, "status": status})

if response.status\_code == 200:

return JsonResponse({"success": "Notification sent to bot"})

else:

return JsonResponse({"error": "Failed to send notification"}, status=500)

except Exception as e:

return JsonResponse({"error": str(e)}, status=500)

куда вставить этот код, в какую вьюшку. Заменить такую же процедуру в orders/views?

path("update\_order/<int:order\_id>/", update\_order\_status, name="update\_order")

Теперь вы можете отправлять запросы через браузер или Postman: <http://127.0.0.1:8000/update_order/123/>

2. Aiogram бот для обработки вебхуков

Ваш бот будет принимать данные от Django-приложения на локалхосте. Внесём изменения в обработчик:

from aiogram import Bot, Dispatcher, types

from aiogram.types import WebhookRequest

from aiogram.utils.executor import start\_webhook

from database import get\_user\_and\_order\_status import logging

BOT\_TOKEN = "YOUR\_TELEGRAM\_BOT\_TOKEN"

WEBHOOK\_PATH = "/webhook"

BOT\_HOST = "127.0.0.1" BOT\_PORT = 8001

bot = Bot(token=BOT\_TOKEN)

dp = Dispatcher(bot)

2-1-6

**Зарегистрируйте маршрут в urls.py:**

python

Копировать код

from django.urls import path

from .views import update\_order\_status

urlpatterns = [

path("update\_order/<int:order\_id>/", update\_order\_status, name="update\_order\_status"),

]

Теперь вы можете отправлять запросы через браузер или Postman:

bash

Копировать код

http://127.0.0.1:8000/update\_order/123/

2-1-9

**Запуск бота и Django**

1. Запустите Django-приложение на порту 8000:

bash

Копировать код

python manage.py runserver 127.0.0.1:8000

1. Запустите бота на порту 8001:

bash

Копировать код

python bot.py

**Протестируйте**

1. Вызовите URL в Django-приложении:

http://127.0.0.1:8000/update\_order/123/

1. Django отправит POST-запрос на ваш бот.

http://127.0.0.1:8001/webhook

1. Бот обработает запрос и отправит уведомление пользователю через Telegram.

Теперь Django и бот взаимодействуют на локалхосте через вебхуки! (2-1-10)

**3-1**

**А как в твоем примере запущен сервер для вебхука?**

В моем примере сервер для обработки вебхуков запускается через встроенный веб-сервер библиотеки aiogram. Он стартует с использованием функции start\_webhook.

**Как именно работает запуск:**

1. Сервер слушает указанный порт (например, 8001) и ожидает запросы по пути /webhook.
2. Когда приходит запрос, обработчик handle\_webhook принимает его и выполняет соответствующую логику.

**Вот ключевые моменты:**

1. **Функция start\_webhook**: Она создает HTTP-сервер для обработки запросов. Вы указываете:
   * dispatcher — ваш диспетчер команд aiogram.
   * webhook\_path — маршрут, на который бот будет принимать запросы.
   * host и port — адрес и порт, на которых запускается сервер.

**Пример в коде:**

python

Копировать код

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

start\_webhook(

dispatcher=dp, # Диспетчер aiogram

webhook\_path=WEBHOOK\_PATH, # Путь для вебхуков

on\_startup=on\_startup, # Действия при запуске

on\_shutdown=on\_shutdown, # Действия при завершении

skip\_updates=True,

host=BOT\_HOST,

port=BOT\_PORT,

)

**3-3 54 строка main.py bot**

async def order\_status\_handler(message: Message) -> None:

"Не удалось найти заказ с таким ключом. Проверьте ключ и попробуйте снова."

async def on\_startup(bot: Bot):

# Устанавливаем URL вебхука

await bot.set\_webhook(WEBHOOK\_URL)

logging.info(f"Webhook установлен: {WEBHOOK\_URL}")

async def main() -> None:

# Инициализация бота

bot = Bot(token=TOKEN, default=DefaultBotProperties(parse\_mode=ParseMode.HTML))

await dp.start\_polling(bot)

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

logging.basicConfig(level=logging.INFO)

asyncio.run(main())

3-4 дополняет некоторые строки 3-3

# Устанавливаем URL вебхука

await bot.set\_webhook(WEBHOOK\_URL)

logging.info(f"Webhook установлен: {WEBHOOK\_URL}")

async def on\_shutdown(bot: Bot):

# Удаляем вебхук при остановке

await bot.delete\_webhook()

logging.info("Webhook удалён")

async def main() -> None:

# Инициализация бота

bot = Bot(token=TOKEN, default=DefaultBotProperties(parse\_mode=ParseMode.HTML))

app = web.Application()

await dp.start\_polling(bot)

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

logging.basicConfig(level=logging.INFO)

asyncio.run(main())

Добавлены строки:

async def main() -> None:  
 # Инициализация бота  
 bot = Bot(token=TOKEN, default=DefaultBotProperties(parse\_mode=ParseMode.HTML))  
  
 app = web.Application()  
  
 webhook\_handler = SimpleRequestHandler(dispatcher=dp, bot=bot)  
 webhook\_handler.register(app, path=WEBHOOK\_PATH)  
  
 setup\_application(app, dp, bot=bot)  
 await web.run\_app(app, host="127.0.0.1", port=WEBHOOK\_PATH)  
  
 await dp.start\_polling(bot)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 logging.basicConfig(level=logging.INFO)  
 asyncio.run(main())

**4 дополняем процедуру в db\_conttoller**

import sqlite3

import logging

DB\_PATH = "ваш\_путь\_к\_базе\_данных"

def set\_user\_telegram\_id(order\_key, telegram\_id):

conn = sqlite3.connect(DB\_PATH)

cursor = conn.cursor()

try:

# Выполняем запрос для обновления Telegram ID в таблице main\_user

cursor.execute("""

UPDATE main\_user

SET telegram = ?

WHERE id = (

SELECT user\_id

FROM main\_order

WHERE order\_key = ?

)

""", (telegram\_id, order\_key))

conn.commit()

except sqlite3.Error as e:

logging.error(f"Database error: {e}")

finally:

conn.close()

6.

Добавляем процедуру в db\_controller

def get\_user\_and\_order\_status(order\_id): conn = sqlite3.connect(DB\_PATH) cursor = conn.cursor() try: # Выполняем запрос для получения Telegram ID пользователя и статуса заказа cursor.execute(""" SELECT u.telegram, o.order\_key, o.status FROM main\_user u JOIN main\_order o ON u.id = o.user\_id WHERE o.id = ? """, (order\_id,)) result = cursor.fetchone() return result # Вернём результат запроса except sqlite3.Error as e: logging.error(f"Database error: {e}") return None finally: conn.close()

StatusCode : 200

StatusDescription : OK

Content : {"ok":true,"result":{"url":"https://f6ab-141-227-138-58.ngrok-f

ree.app/webhook","has\_custom\_certificate":false,"pending\_update

\_count":6,"last\_error\_date":1735772533,"last\_error\_message":"Wr

ong respons...

RawContent : HTTP/1.1 200 OK

Connection: keep-alive

Strict-Transport-Security: max-age=31536000; includeSubDomains;

preload

Access-Control-Allow-Origin: \*

Access-Control-Allow-Methods: GET, POST, OPTIONS

Acce...

Forms : {}

Headers : {[Connection, keep-alive], [Strict-Transport-Security, max-age=

31536000; includeSubDomains; preload], [Access-Control-Allow-Or

igin, \*], [Access-Control-Allow-Methods, GET, POST, OPTIONS]...

}

Images : {}

InputFields : {}

Links : {}

ParsedHtml : mshtml.HTMLDocumentClass

RawContentLength : 286