**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Сумський державний університет**

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____
(підпис)

червня 2024 р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня бакалавр**

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Телеграм бот для управління фінансами з можливістю голосового керування»

здобувача групи ІН - 03 Джафарова Вагіфа-Вадима Аріфовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Вагіф-Вади
ДЖАФАРО
_____
підпис)

Керівник,
асистент кафедри комп'ютерних наук

Артем КОРОБОВ_____
(підпис)

**Суми – 2024**

**Сумський державний університет**

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

Ігор
ШЕЛЕХОВ

_____
(підпис)

# ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

**на здобуття освітнього ступеня бакалавра**

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»

здобувача групи ІН-03 Джафаров Вагіф-Вадим Аріфович

1. Тема роботи: «Телеграм бот для управління фінансами з можливістю голосового керування» затверджую наказом по СумДУ від *«22» квітня 2024 р. No 0414-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до 29 червня 2024 року*

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

*1)Аналіз проблеми та актуальності розробки чат-бота, принципів створення, інструментів створення,голосового розпізнавання, постановка й формування завдань дослідження. 2) Огляд та вибір програмних засобів. 3) Проектування та розробка телеграм боту для управління фінансами з можливістю голосового керування. 4) Аналіз результатів.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання | Завдання прийняв |

| | | | |
|---|---|---|---|
| | | вида в | |
| | | | |

Завдання прийняв до виконання Керівник

(підпис) (підпис)

7. Дата видачі завдання «____» _____ 20 ___ р.

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів кваліфікаційної роботи | Термін виконання | Примітка |
|---|---|---|---|
| 1 | Аналіз проблеми та актуальності розробки телеграм бота, принципів створення, інструментів створення,голосового розпізнавання, постановка й формування завдань дослідження. | 06.05.2024 – 08.05.2024 | |
| 2 | Огляд та вибір програмних засобів | 09.05.2024 – 10.05.2024 | |
| 3 | Проектування та розробка телеграм боту для управління фінансами з можливістю голосового керування | 11.05.2024 – 24.05.2024 | |
| 4 | Аналіз результатів. | 25.05.2024 – 27.05.2024 | |
| 5 | Оформлення пояснювальної записки до кваліфікаційної роботи. | 28.05.2024 – 31.05.2024 | |

Здобувач вищої освіти Керівни к

(підпис) (підпис)

# АНОТАЦІЯ

**Записка:** 87 сторінок, 68 рисунки, 4 таблиці, 1 додаток, 13 використаних джерел.

**Об'єктом дослідження** є Telegram-бот, який використовується для моніторингу фінансових потоків.

**Предметом дослідження** є функціональні можливості та ефективність використання Telegram-бота в процесі фінансового планування і контролю.

**Мета роботи –** створення телеграм-бота для управління фінансами з можливістю голосового керування, що забезпечить користувачам швидкий та зручний доступ до фінансової інформації, підвищить ефективність фінансового менеджменту та покращить користувацький досвід завдяки розпізнаванню мови.

**Методи дослідження –** аналіз та застосування різноманітних інструментів, які сприяють полегшенню розробки телеграм ботів, а саме Python, pyBotTelegramAPI, SpeechRecognition, PostgreSQL.

**Результати –** розроблено телеграм-бот для управління фінансами з можливістю голосового керування, який надає користувачам ефективний і зручний спосіб управління своїми фінансами. Бот забезпечує швидкий доступ до фінансової інформації, дозволяє виконувати дії за допомогою голосових команд, надає аналітику витрат та доходів, а також контролює витрати користувача за необхідністю.

ТЕЛЕГРАМ БОТ, ФІНАНСИ, ГОЛОСОВЕ КЕРУВАННЯ, PYTHON, PYBOTTELEGRAMAPI, SPEECHRECOGNITION, POSTGRESQL

**ЗМІСТ**

ВСТУП 7

1. АНАЛІТИЧНИЙ ОГЛЯД 9

1.1. Аналіз принципів створення телеграм ботів 9

1.2. Основні інструменти створення телеграм боту 10

1.3. SpeechRecognition 12

1.4. Постановка задачі 15

2. ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ 17

2.1. Вибір мови програмування 18

2.2. Вибір фреймворку 19

2.3. Вибір середовища розробки 20

2.4. Вибір бази даних 22

3. ПРОГРАМНА РЕАЛІЗАЦІЯ 28

3.1. Проектування бази даних 28

3.2. Створення телеграм-бота 30

3.3. Створення БД та таблиць 33

3.4. Розробка програмної частини 41

3.5. Тестування Telegram-боту 57

ВИСНОВКИ 71

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 73

Додаток А 75

# ВСТУП

**Актуальність.** Тема розробки телеграм-бота для управління фінансами з можливістю голосового керування є надзвичайно актуальною в сучасних умовах, коли технології відіграють ключову роль у покращенні якості життя. Зі зростанням фінансових послуг і потребою в ефективному управлінні особистими фінансами, зручність та швидкість доступу до фінансової інформації стають все більш важливими. Голосове керування додає новий рівень зручності та доступності. Крім того, у зв’язку зі зростаючої популярності месенджерів, таких як Telegram, розробка фінансового бота з голосовим інтерфейсом відповідає сучасним трендам і вимогам користувачів, забезпечуючи швидкий та легкий спосіб обліку фінансів.

**Об'єктом дослідження** є Telegram-бот, який використовується для моніторингу фінансових потоків.

**Предметом дослідження** є функціональні можливості та ефективність використання Telegram-бота в процесі фінансового планування і контролю.

**Припущення.** Впровадження телеграм-бота для управління фінансами з можливістю голосового керування покращить зручність та ефективність фінансового менеджменту користувачів. Використання голосових команд забезпечить швидкий та інтуїтивний доступ до фінансової інформації, що підвищить задоволеність користувачів. Тестування функціоналу та користувацького досвіду дозволить оптимізувати роботу бота, що сприятиме зростанню кількості користувачів та збільшенню їх лояльності.

**Унікальність.** Розроблений телеграм-бот стане інноваційним інструментом для управління фінансами завдяки інтеграції голосового керування. Це дозволить користувачам здійснювати облік фінансових операцій, отримувати аналітику та моніторити свої витрати за допомогою голосових команд, що

значно підвищить зручність використання. Використання технологій розпізнавання мови забезпечить високу точність і надійність обробки голосових запитів.

**Структура.** Дана робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу рішення поставленої задачі, опису програмного забезпечення, висновків, списку використаних джерел та додатків.

# 1. АНАЛІТИЧНИЙ ОГЛЯД

## 1.1. Аналіз принципів створення телеграм ботів

Аналіз принципів побудови телеграм-бота передбачає розгляд основних аспектів, які варто враховувати при його розробці, опишемо ключові принципи:

- Комунікація з користувачем: Телеграм-бот взаємодіє з користувачем через текстові повідомлення. Зручна та зрозуміла комунікація включає в себе вивчення та розуміння введеного користувачем тексту, відповідну обробку команд, а також відповіді на запити користувача.
- Розпізнавання команд: Важливо налаштувати бота таким чином, щоб він розпізнавав команди користувача. Це може включати в себе розпізнавання ключових слів, фраз або використання регулярних виразів для визначення запитів.
- Взаємодія з зовнішніми сервісами: Телеграм-бот може інтегруватися з різноманітними зовнішніми сервісами для розширення своєї функціональності. Зокрема, використання технології розпізнавання голосу дозволяє боту взаємодіяти з користувачами за допомогою голосових команд. Для цього, наприклад, може використовуватися бібліотека SpeechRecognition, яка дозволяє отримувати текстове представлення голосових повідомлень.
- Графічний інтерфейс: Телеграм підтримує елементи графічного інтерфейсу, такі як кнопки, стікери тощо. Вони можуть бути використані для поліпшення користувацького досвіду та зрозумілості інтерфейсу користувача.

Користуючись цим аналізом, ми зможемо розробити бота, який ефективно взаємодіє з користувачами та виконує свої функції.

### 1.2. Основні інструменти створення телеграм боту

Для розробки та реалізації телеграм-бота необхідно використовувати різноманітні інструменти, що забезпечують його функціональність та взаємодію з користувачами. Ось основні компоненти, які використовуються під час створення телеграм-ботів на Python:

BotFather

BotFather - це спеціальний обліковий запис в Telegram, який дозволяє створювати та керувати ботами. Він надає унікальні токени, які необхідні для ідентифікації бота та взаємодії з API Telegram.

Telegram API

Telegram API - це набір інструментів та інтерфейсів, які дозволяють розробникам взаємодіяти з Telegram-платформою. Використовуючи Telegram API, можна створювати ботів, обробляти повідомлення від користувачів та виконувати різноманітні дії.

Python

Python - це популярна мова програмування, яка часто використовується для розробки телеграм-ботів. Python має велику кількість бібліотек та фреймворків, що значно спрощують процес розробки та надають доступ до різноманітних функцій.

База даних

Для зберігання та обробки даних, які використовуються в телеграм-боті, часто використовують реляційні бази даних, такі як PostgreSQL, MySQL або

SQLite. Це дозволяє зберігати інформацію, наприклад про користувачів, їхні повідомлення та інші дані, необхідні для роботи бота.

Бібліотека розпізнавання голосу

Для реалізації функціоналу голосового керування можна використовувати бібліотеки розпізнавання голосу, такі як SpeechRecognition. Ці бібліотеки дозволяють отримувати аудіофайли з голосовими повідомленнями від користувачів та перетворювати їх на текст для подальшої обробки та аналізу в телеграм-боті.

Система контролю версій Git

Git — це система контролю версій, яка дозволяє слідкувати за змінами в коді, зберігаючи “знімки” на різних етапах розробки. Він допомагає розробникам співпрацювати, уникаючи конфліктів та зберігаючи історію змін. Git не має графічного інтерфейсу за замовчуванням, але є численні інструменти, які допомагають візуалізувати роботу з ним [1]. Використання Git під час розробки телеграм-бота дозволить ефективно керувати кодом проєкту, відстежувати зміни та працювати над ним на різних пристроях.

### 1.3. SpeechRecognition

Розпізнавання мови - це технологія, яка дозволяє комп'ютерам розуміти людську мову і перетворювати її на текст. Це може бути корисно для різних додатків, таких як віртуальні асистенти, сервіси транскрипції тощо. Ми будемо використовувати бібліотеку SpeechRecognition в Python для створення нашої системи розпізнавання мови [2]. Перші системи розпізнавання мови працювали з одним диктором і мали обмежений словниковий запас, що складався з десятка слів. А ось сучасні системи розпізнавання мовлення пройшли довгий шлях від своїх давніх аналогів. Вони можуть розпізнавати мову від кількох дикторів і мають великі словники на багатьох мовах.

Розпізнавання мови в Python працює з алгоритмами, які виконують лінгвістичне та акустичне моделювання. Акустичне моделювання використовується для розпізнавання фонетики в нашій мові, щоб отримати більш значущі частини мови, такі як слова та речення. Принцип роботи SpeechRecognition можна побачити на рисунку 1.1.


Рисунок 1.1 – Принцип роботи SpeechRecognition

Розпізнавання мовлення починається з отримання звукової енергії, яку видає людина, що говорить, і перетворення її в електричну енергію за допомогою мікрофона. Потім ця електрична енергія перетворюється з аналогової в цифрову і, нарешті, в текст. Він розбиває аудіодані на звуки і аналізує звуки за допомогою алгоритмів, щоб знайти найбільш ймовірне

слово, яке відповідає цьому звуку. Все це робиться за допомогою обробки природної мови та нейронних мереж [3].

А тепер порівняємо кілька різних пакетів для обробки голосу та розпізнавання мовлення. Кожен з цих інструментів має свої унікальні можливості та обмеження, які важливо врахувати при виборі найбільш підходящого рішення для вашого проєкту. Давайте розглянемо їх основні характеристики, переваги та недоліки у таблиці 1.1

| **Пакет** | **Функціональність** | **Переваги** | **Недоліки** |
|---|---|---|---|
| Apiai | Включає обробку мови для ідентифікації намірів користувача | Інтеграція з іншими сервісами Google | Обмежена функціональність без оплати |
| Google-cloud-speech | Пропонує базове перетворення мовлення на текст | Можливість інтеграції з іншими сервісами Google | Обмежена функціональність |
| Speech Recognition | Забезпечує легку обробку аудіо, перетворення голосу на текст або аналіз аудіоданих | Простота використання та підтримка різних мов | Залежність від сторонніх сервісів |

Таблиця 1.1 – порівняння пакетів для обробки голосу

Загалом, після порівняння різних пакетів для обробки голосу та розпізнавання мовлення, є один пакет, який виділяється з точки зору простоти

використання – це SpeechRecognition. Ця бібліотека виступає в ролі оболонки для декількох популярних мовних API, що надає надзвичайну гнучкість. Наприклад, одним з них є Google Web Speech API, який підтримує ключ API за замовчуванням. Це означає, що можна розпочати роботу без необхідності підписуватись на послугу. Гнучкість і простота використання пакета SpeechRecognition роблять його чудовим вибором для будь-якого проєкту на Python.

### 1.4. Постановка задачі

За допомогою вище вказаних інструментів, у пункті 1.2, та методів, потрібно створити телеграм бота, який буде використаний як система аналізу, керування та планування фінансів з можливістю голосового управління.

Перейдемо до написання переліку задач, які потребують реалізації:

Розробка:

- Створення Telegram бота.
- Створення бази даних для зберігання фінансових даних.
- Написання основного функціоналу бота, такого як додавання витрат, доходів, налаштування нагадувань та аналіз фінансів.
- Розробка логіки обробки команд, взаємодії з базою даних, обробки голосових повідомлень.
- Впровадження рішень для демонстрації фінансових даних.
- Підключення бібліотеки голосового розпізнавання для можливості голосового керування.

Нагадування:

- Реалізація системи нагадувань про фінансові обмеження(ліміти)

Голосове керування:

- Розробка механізму для обробки голосових повідомлень, включаючи визначення команд та текстової інтерпретації.
- Інтеграція обробки голосових повідомлень з бібліотекою голосового розпізнавання.

Тестування та оцінка:

- Проведення тестування.
- Оцінка роботи бота.

Ці етапи дозволять успішно реалізувати телеграм-бота для управління фінансами з можливістю голосового керування.

# 2. ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

Перш ніж приступати до реалізації будь-якого проєкту, важливо обрати найбільш підходящі методи та інструменти для досягнення поставленої задачі. Різноманітність технологій та підходів дозволяє забезпечити максимальну ефективність та зручність використання бота для користувачів. Для досягнення цього ми ретельно розглянемо популярні та доступні методи, а також визначимо найбільш оптимальні рішення для нашого проєкту.

Коли мова йде про створення телеграм-бота, існує декілька можливих підходів. Можна використовувати мову програмування Python та популярні бібліотеки для розробки бота з нуля, або ж інші мови програмування. Саме цей підхід надає повний контроль над процесами розробки та дозволяє написати бота точно під вимоги проєкту.

З іншого боку, існують конструктори ботів, які надають можливість створювати ботів без знань мов програмування. Ці конструктори можуть бути корисними для швидкої розробки прототипів або для людей без досвіду в програмуванні. Однак, ще на початку написання такого телеграм-бота, його розробник ставить себе у рамки доступного функціоналу конструктора, за які він ніяк не вийде, порівняно з розробкою на мові програмування.

Таким чином, якщо потрібен більший контроль, гнучкість та можливості для розширення, програмування - це найкращий варіант.

### 2.1. Вибір мови програмування

Вибір мови програмування для написання телеграм-бота є важливим етапом у розробці проєкту. Деякі з найпоширеніших мов програмування, які використовуються для цієї цілі, включають Python, JavaScript, Java, та навіть C#. Однак, серед цих варіантів Python обирають найчастіше.

Python (укр. Пайтон) — високорівнева мова програмування, яку називають другою за популярністю в світі. Її використовують для розробки вебзастосунків, програмного забезпечення, машинного навчання. Python застосовують для вирішення робочих завдань у компаніях Google, Instagram, Facebook, IBM, NASA, Dropbox, Netflix та інших. Розробники цінують цю мову програмування за простоту у вивченні, ефективність та мультиплатформність Для розуміння достатньо порівняти принципи написання найпростішої програми, яка виводить на екран текстове повідомлення [4]. Ось приклад як це виглядає на мовах програмування Java та Python на рисунку 2.1.

Python

main.py

```
1  print("Hello, world")
```

Java

Main.java

```
1 class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello, world");
4     }
5 }
```

Рисунок 2.1 - Порівняння коду програми Java та Python

Відразу після порівняння стає зрозуміло чому багато розробників обирає саме Python, і чому ця мова завжди займає лідируючі позиції в рейтингах використання мов програмування. Тому для розробки свого проєкту я буду використовувати Python.

### 2.2. Вибір фреймворку

Фреймворк (framework) – це набір інструментів, бібліотек та правил, який використовується для створення програмних додатків. Він зазвичай являє собою структуру, яка визначає, як компоненти програми повинні взаємодіяти між собою, які шаблони використовувати для створення інтерфейсів і які методи методи використовувати для роботи з базами даних та іншими зовнішніми ресурсами. [5]

Обираючи фреймворк для написання телеграм-бота, важливо враховувати його можливості та зручність використання. Деякі з найпопулярніших Python фреймворків для розробки телеграм-ботів включають:

- pyTelegramBotAPI (telebot) - це популярний фреймворк для створення телеграм-ботів у Python. Його перевага полягає в його простоті використання та функціональності. Цей фреймворк надає зручний та зрозумілий інтерфейс для розробки та керування ботами.
- Aiogram - це фреймворк для розробки телеграм-ботів у Python з використанням асинхронного програмування. Однією з головних особливостей Aiogram є його спрощений та зрозумілий інтерфейс, який дозволяє створювати та керувати ботами. Фреймворк надає доступ до різноманітних функцій Telegram API, включаючи роботу з повідомленнями, клавіатурами, Inline-кнопками та опитуваннями.
- python-telegram-bot - це фреймворк для роботи з API Telegram у Python, який надає простий та зрозумілий інтерфейс для створення та керування телеграм-ботами. Однією з головних переваг цього фреймворку є його легкість використання, яка дозволяє навіть початківцям швидко розпочати розробку свого бота.

- Pyrogram - асинхронний фреймворк для роботи з повним Telegram API на Python. Pyrogram також дозволяє розробляти складніші проєкти, але не спеціалізований для роботи з ботами та має вищий поріг входу для початківців.

Для розробки телеграм-бота було обрано фреймворк pyTelegramBotAPI (telebot) у поєднанні з мовою програмування Python. Це рішення обумовлене тим, що цей фреймворк надає зручний та зрозумілий інтерфейс, а також менш складний у використанні на відміну від його конкурентів.

### 2.3. Вибір середовища розробки

IDE (Integrated Development Environment) - це середовище розробки, яке надає розробникам зручні інструменти для написання, редагування, компіляції та налагодження програмного коду в одному місці. Обрати IDE можна, керуючись декількома важливими критеріями:

- Мова програмування: Вибір IDE залежить від того, яку мову програмування воно підтримує, відповідно до тої яку ви збираєтеся використовувати у розробці. Наприклад, якщо працювати з Python, підійдуть такі IDE як PyCharm, Visual Studio Code або Komodo.
- Легкість використання: Деякі IDE можуть бути більш складними для освоєння, наприклад не зрозумілий інтерфейс, тому важливо обрати той, який подобається найбільше.
- Сумісність з операційною системою: Обрана IDE повинна підтримувати потрібну операційну систему (Windows, macOS, Linux).
- Вартість: Деякі IDE є безкоштовними, а інші умовно безкоштовними, тобто потребують плати за використання.

Далі пропоную розглянути декілька середовищ розробки:

- CLion (платна розробка компанії JetBrains, призначена для платформ Windows/Linux/macOS, підтримує такі мови, як C++, C, Objective C, Kotlin, Python, JavaScript та інші);
- Komodo (умовно-безкоштовна IDE від ActveState, що працює на Windows/Linux/macOS, дозволяє працювати з такими мовами, як Python, PHP, Perl, Golang, Ruby та ін.);
- PyCharm (умовно-безкоштовне рішення від JetBrains, що випускається для платформ Windows/Linux/macOS, в основному використовується для написання коду на Python, але при оформленні підписки дозволяє писати на цілій низці інших мов програмування.
- Visual Studio (умовно безкоштовна IDE від Microsoft, призначена для платформ Windows/Linux/macOS, підтримує такі мови, як Visual Basic, Visual C#, Visual C++, Visual F#, ASP.NET та інші);
- Xcode (поширюване безкоштовне творіння компанії Apple, призначене виключно для macOS, підтримує мови C, C++, Objective-C, Swift, Java та інші).[6]

Вибір IDE є важливим етапом у процесі розробки програмного забезпечення. З урахуванням різноманітних факторів, які включають мову програмування, легкість використання, сумісність з операційною системою та вартість, було обрано середовище розробки PyCharm. Його швидкість, надійність, різноманіття плагінів та інструментів для розробки на Python, роблять його ідеальним вибором для мого проєкту.

### 2.4. Вибір бази даних

Вибір бази даних є ключовим етапом у процесі розробки телеграм бота, оскільки це визначає, як будуть зберігатися та оброблятися дані, необхідні для функціонування бота. Правильно обрана база даних забезпечує ефективну та надійну роботу бота, забезпечуючи швидкий доступ до інформації та оптимізовану роботу з даними. У цьому розділі ми розглянемо різні аспекти вибору бази даних для телеграм бота, включаючи типи баз даних, їхні особливості, переваги та недоліки, щоб зробити вдалий та оптимальний вибір для нашого проєкту.

База даних – це організоване зібрання даних, завдяки якому можна легко отримати до них доступ та керувати ними. Основною метою бази даних є працювати з великою кількістю інформації шляхом зберігання, вилучення та управління даними. [7] Одним з найпоширеніших мов запитів до баз даних є SQL, яка використовується для взаємодії з базами даних, здійснення запитів та змін в їхніх структурах та даних.

SQL — це мова програмування, яка використовується майже всіма реляційними базами даних для виконання запитів і операцій, а також для визначення даних і контролю доступу. Уперше мова SQL була розроблена в IBM у 1970-х роках за значної участі Oracle, що призвело до впровадження стандарту SQL ANSI. SQL сприяла появі багатьох розширень від таких компаній, як IBM, Oracle і Microsoft. Хоча SQL досі широко використовується, починають з’являтися нові мови програмування.[8]

Наразі існує багато баз даних, які використовуються у різних сферах та для різних завдань. Для телеграм-ботів, які зазвичай працюють з невеликим обсягом даних та не вимагають складних структур баз даних, найбільш популярними виборами є:

SQLite:

Це компактна вбудована реляційна база даних з відкритим кодом. Вона одна з найпопулярніших у світі, має нагороду Google-O'Reilly Open Source Awards і широко використовується в додатках та системах, де потрібно організувати зберігання даних. SQLite не є окремим сервером баз даних, як, наприклад, MySQL або PostgreSQL. Натомість це бібліотека, яку можна вбудувати безпосередньо в додаток. Це означає, що для роботи з SQLite не потрібно окремо встановлювати та налаштовувати сервер баз даних. [9] Приклад написання запиту до бази даних проілюстровано на рисунку 2.2, цей запит вибирає всі записи з таблиці table_name:

```
SELECT * FROM table_name;
```

Рисунок 2.2 – Запит до БД SQLite

Переваги:

- Легка в установці та використанні.
- Інтегрована безпосередньо в додаток, не потребує окремого сервера баз даних.
- Підтримує стандарт SQL та багато корисних функцій.

Недоліки:

- Менш підходить для великих обсягів даних або вимог до високої швидкодії.
- Не підтримує розподілення баз даних, що може бути проблемою для деяких випадків

MongoDB:

Це система управління базами даних, яка працює з документоорієнтованою моделлю даних. На відміну від реляційних СУБД, MongoDB не потребує таблиці, схеми або окремої мови запитів. Інформація зберігається як документ чи колекція. Розробники позиціонують продукт як проміжну ланку між класичними СУБД та NoSQL. MongoDB не використовує схеми, як це роблять реляційні бази даних, що підвищує продуктивність усієї системи.[10] Приклад написання запиту до бази даних проілюстровано на рисунку 2.3, цей запит вибирає всі документи з колекції collection:

```
db.collection.find({});
```

Рисунок 2.3 – Запит до БД MongoDB

Переваги:

- Гнучка схема даних, що підходить для структур, що змінюються.
- Відмінна продуктивність для роботи з великими обсягами неструктурованих даних.

Недоліки:

- Менш підходить для даних з високим рівнем зв'язності.
- Може потребувати більше уваги до забезпечення цілісності даних.

PostgreSQL:

Це потужна об'єктно-реляційна система баз даних з відкритим вихідним кодом яка використовує і розширює мову SQL у поєднанні з багатьма функціями, що дозволяють безпечно зберігати та масштабувати найскладніші робочі навантаження даних.[11] Приклад написання запиту до бази даних проілюстровано на рисунку 2.4, цей цей запит також вибирає всі записи з таблиці table_name.

```
SELECT * FROM table_name;
```

Рисунок 2.4 – Запит до БД PostgreSQL

Переваги:

- Висока надійність та стійкість до відмов.
- Підтримка широкого спектру функцій SQL та розширень.
- Підходить для роботи зі складними структурами даних та великими обсягами інформації.

Недоліки:

- Потребує більше ресурсів для роботи та налаштування, порівняно з деякими іншими базами даних.
- Може бути складним для освоєння новачками.

Redis:

Це сховище даних в пам'яті з відкритим вихідним кодом, яке чудово працює в якості кешу або брокера повідомлень, але його також можна використовувати як базу даних, коли вам не потрібні всі функції традиційної бази даних. Він пропонує відмінну продуктивність, з можливістю швидкого читання і запису даних в пам'ять. Крім того, Redis підтримує атомарні операції, що робить його ідеальним для сценаріїв кешування, де вам потрібен швидкий доступ. [12] Redis, як база даних ключ-значення, не має запитів як в інших БД, але приклад команди для отримання значення за ключем проілюстровано на рисунку 2.5. Ця команда отримує значення, збережене під ключем key_name:

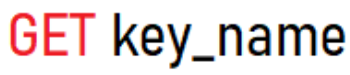



Рисунок 2.5 – Команда до сховища даних Redis

Переваги:

- Дуже швидкий доступ до даних, оскільки всі дані зберігаються в оперативній пам'яті.
- Простий у використанні та налаштуванні.

Недоліки:

- Не підходить для зберігання великих обсягів даних через обмежену ємність оперативної пам'яті.
- Може втратити дані при перезавантаженні або відмові сервера.

З урахуванням всіх розглянутих аспектів баз даних, я обрав PostgreSQL для використання у створенні телеграм-боту для управління фінансами. Ця СУБД відповідає вимогам проєкту щодо надійності, стійкості та

продуктивності. Вона підтримує широкий спектр функцій SQL та розширень, що дозволяє безпечно зберігати та масштабувати найскладніші робочі навантаження даних. Крім того, PostgreSQL відома своєю високою надійністю та стабільністю, що робить її ідеальним вибором для проєктів з обробки фінансової інформації.

# 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

## 3.1. Проектування бази даних

Проектування бази даних є ключовим етапом у створенні будь-якого програмного забезпечення, не є виключенням і телеграм-бот. Правильно спроєктована база даних дозволить зберігати, організовувати та отримувати доступ до інформації, необхідної для функціонування бота, забезпечуючи надійність та швидкодію. Розглянемо основні кроки проєктування бази даних, включаючи визначення функціональних та нефункціональних вимог, створення схеми БД та вибір типів даних.

Функціональні вимоги для телеграм боту для управління фінансами з можливістю голосового керування можуть включати:

- Додавання та видалення фінансових записів: Користувач повинен мати можливість додавати нові фінансові транзакції (витрати, доходи).

- Перегляд балансу: Бот повинен давати користувачу змогу переглядати поточний баланс фінансової інформації, такої як витрати за певний період та/або нагадувати про встановлений ліміт у разі його перевищення.

- Аналіз фінансової інформації: Бот може надавати звіти або статистику про фінансові транзакції користувача, наприклад, витрати за категоріями.

- Голосове керування: Користувач може мати можливість керувати ботом за допомогою голосових команд, таких як “Додай нову витрату” або “Покажи статистику за поточний місяць”.

Нефункціональні вимоги можуть включати:

- Надійність та безпека: Бот повинен бути захищений від несанкціонованого доступу та забезпечувати безпеку фінансової інформації користувача.

- Продуктивність: Бот повинен працювати швидко та ефективно, навіть при великому обсязі фінансових даних.

- Зручність використання: Інтерфейс бота повинен бути зрозумілим та легким у використанні для різних категорій користувачів.

- Масштабованість: Система повинна бути здатна масштабуватися для обробки зростаючого обсягу користувацьких запитів.

Описавши функціональні та нефункціональні вимоги, перейдемо до створення схеми БД, що описана у таблицях 3.1 – 3.3.

Таблиця 3.1 – схема таблиці витрат

| **Витрати** | |
|---|---|
| **Ім'я поля** | **Тип даних** |
| User_id | Integer |
| Категорія | Text |
| Сума | Double precision |
| Дата | Date |

Таблиця 3.2 – схема таблиці надходжень

| **Надходження** | |
|---|---|
| **Ім'я поля** | **Тип даних** |
| User_id | Integer |

| | |
|---|---|
| Сума | Double precision |
| Дата | Date |

Таблиця 3.3 – схема таблиці лімітів користувача

| **Ліміти** | |
|---|---|
| **Ім'я поля** | **Тип даних** |
| User_id | Integer |
| Ліміт користувача | Double precision |

Тепер, коли визначено структуру бази даних, можна розпочати розробку.

### 3.2. Створення телеграм-бота

Пропоную детально розглянути процес створення та налаштування телеграм-бота. Перш за все, початковий етап розробки полягає в реєстрації телеграм-бота в Telegram за допомогою бота BotFather , що на рисунку 3.1, а вже тільки після цього написання його функціоналу. BotFather (від англ. батько ботів) – це програма-бот яка дозволяє створювати власних ботів в телеграм. Він доступний будь-якому користувачеві, достатньо лише ввести його ім’я в рядку пошуку.[12]

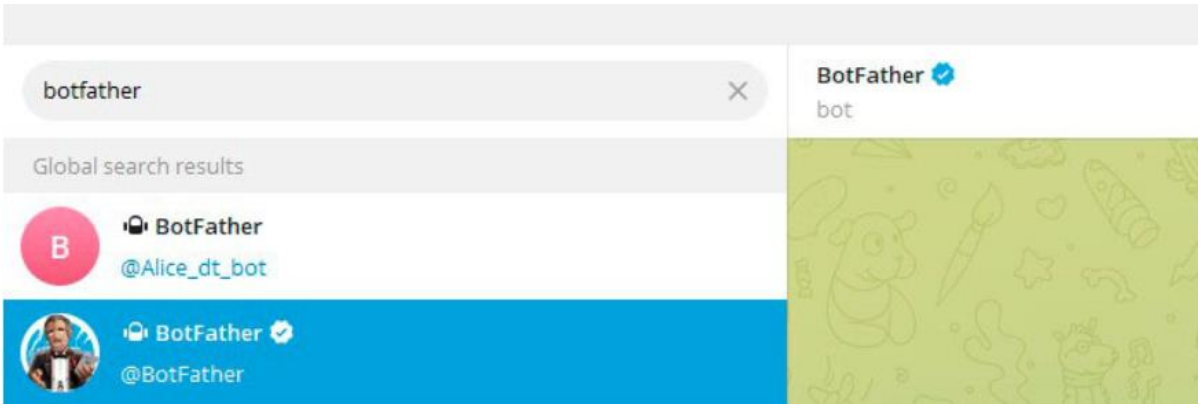


Рисунок 3.1 – Пошук BotFather

Далі, за допомогою кнопки Start, що на рисунку 3.2, починаємо “діалог” з цим чат-ботом.

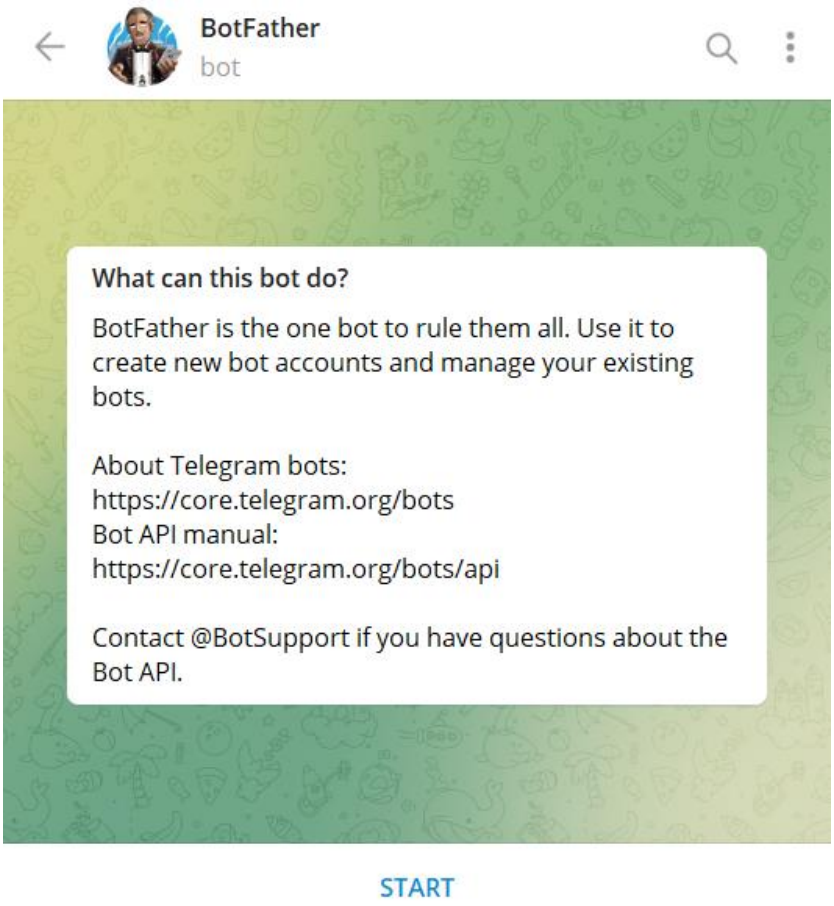



Рисунок 3.2 - BotFather

Після чого він надсилає нам меню функцій на рисунку 3.3, які доступні до виконання.

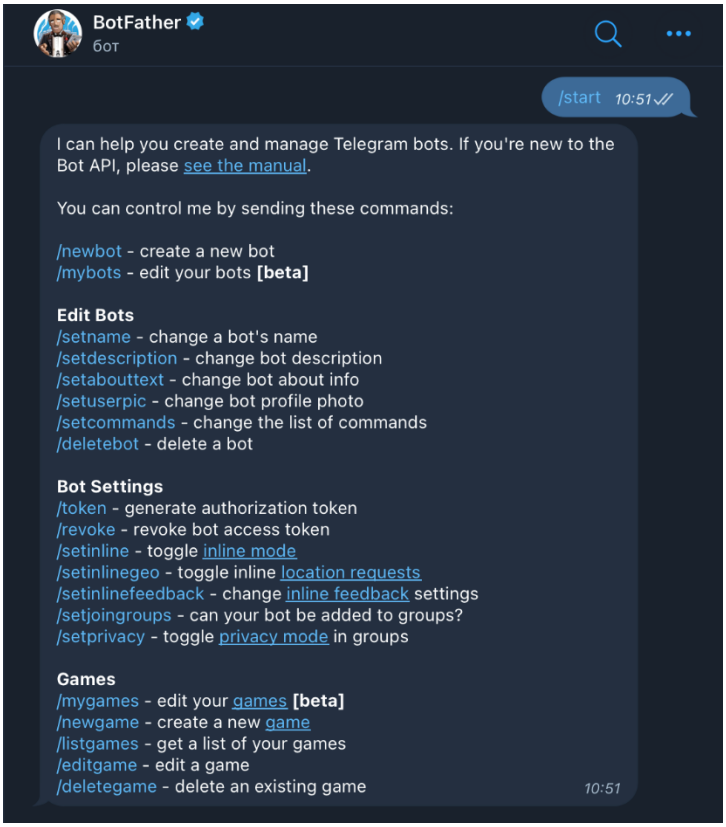



Рисунок 3.3 – Список команд BotFather

Першим кроком для створення нового бота, буде виконання команди “/newbot”, яку відправляємо BotFather, або ж натискаємо у списку. Після цього буде запропоновано обрати (ввести) ім'я для бота. Слід обрати ім'я, яке краще буде відображати функціональність бота. Після вибору імені BotFather попросить ввести користувацьке ім'я для вашого бота. Треба написати коротке ім'я, яке закінчується на “bot”, воно і буде використовуватися як ідентифікатор нашого бота в Telegram. Після введення імені BotFather повідомить про успішне створення бота, далі буде наданий токен нашого бота, а також посилання на нього в Telegram на рисунку 3.4. Токен - це ключ доступу до API вашого бота, і він буде необхідний для підключення нашого бота до месенджера.

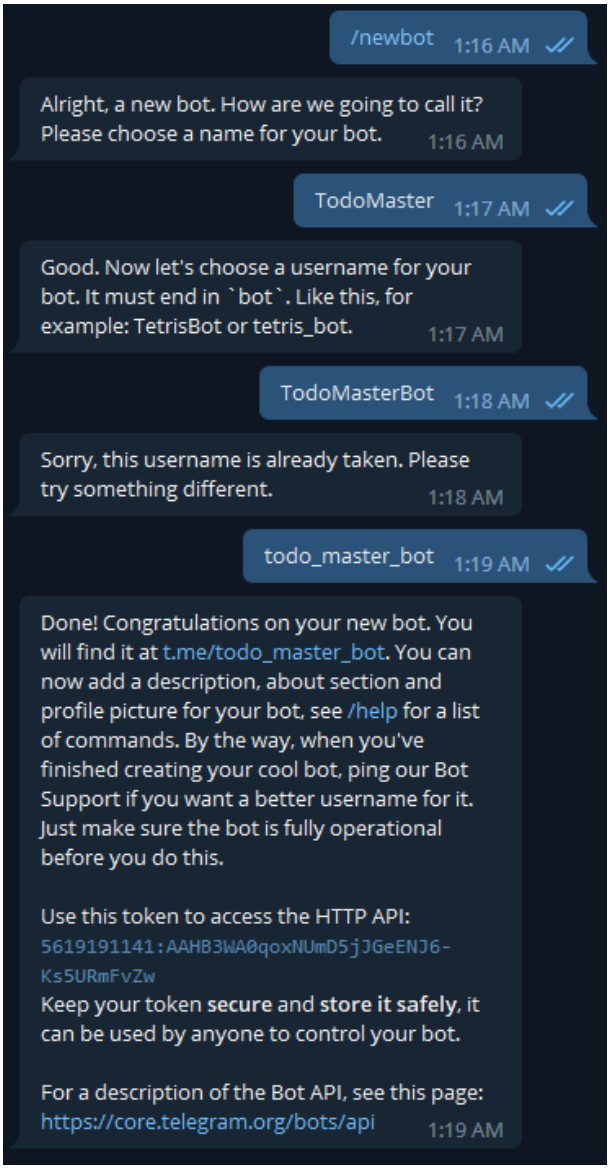



Рисунок 3.4 – Приклад створення бота

### 3.3. Створення БД та таблиць

Першим кроком у розробці бази даних для нашого проєкту є створення необхідних таблиць у системі керування базами даних (СКБД) PostgreSQL за допомогою інструменту адміністрування баз даних pgAdmin, що продемонстровано на рисунку 3.5

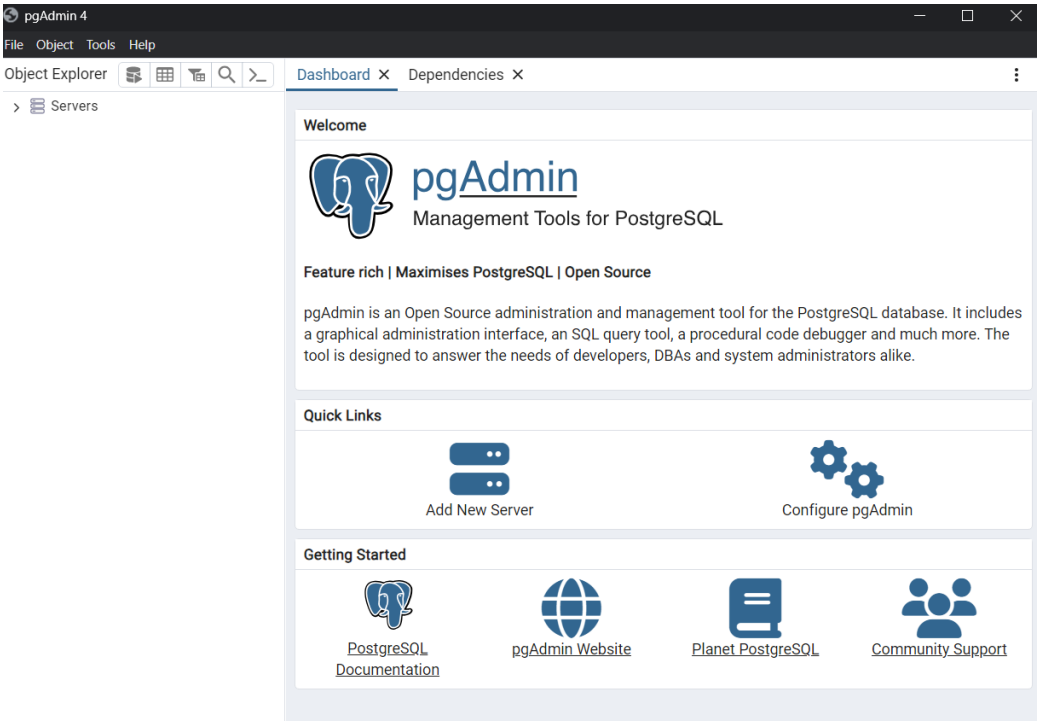



Рисунок 3.5 – Вікно pgAdmin

pgAdmin - це інструмент для керування базами даних PostgreSQL. Він надає зручний графічний інтерфейс для керування таблицями та схемами, виконання SQL-запитів, стеження за роботою сервера бази даних тощо. Цей етап розробки є одним з найважливіших для успішної реалізації проєкту.

Щоб створити базу даних за допомогою pgAdmin, перш за все, необхідно відкрити програму та підключитися до сервера баз даних PostgreSQL. Після успішного підключення можна побачити список існуючих баз даних у дереві навігації, що на рисунку 3.6.

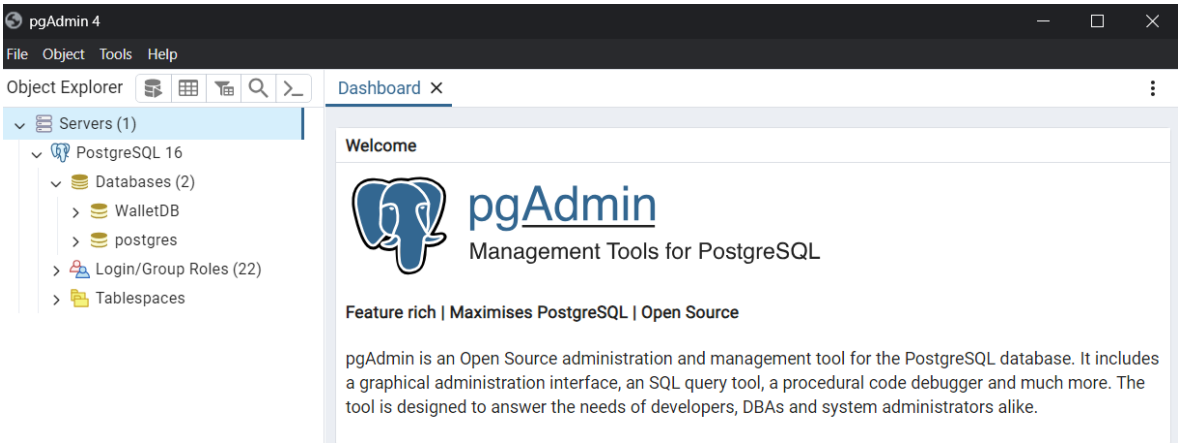



Рисунок 3.6 – Список існуючих баз даних у дереві навігації

Далі, щоб створити нову базу даних, можна скористатися контекстним меню правої кнопки миші на кнопці «Бази даних», що на рисунку 3.7.

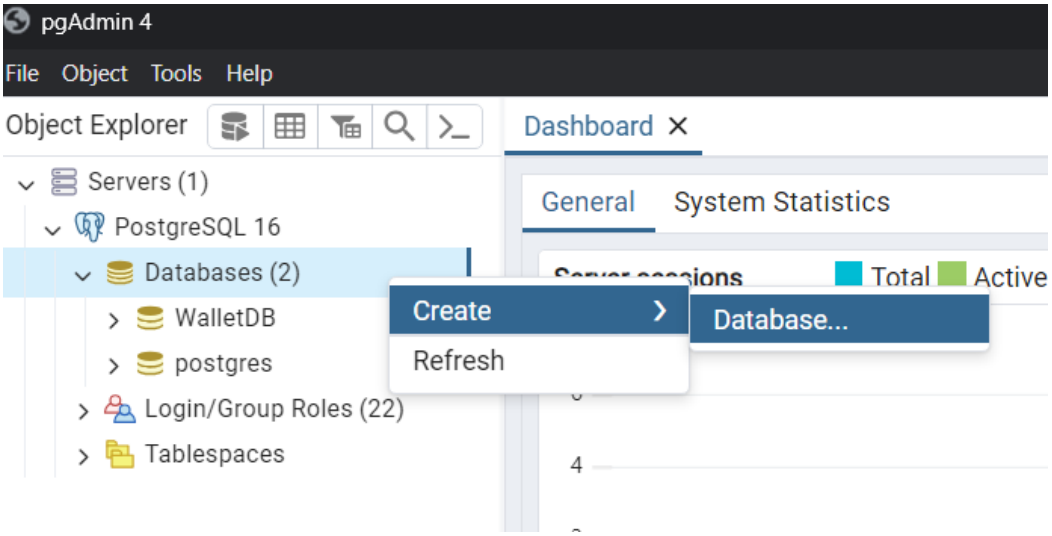



Рисунок 3.7 – Створення БД

Після того, як обрали дану опцію, відкриється вікно, в якому можна ввести назву нової бази даних та обрати параметри, такі як кодування та власника бази даних на рисунку 3.8.

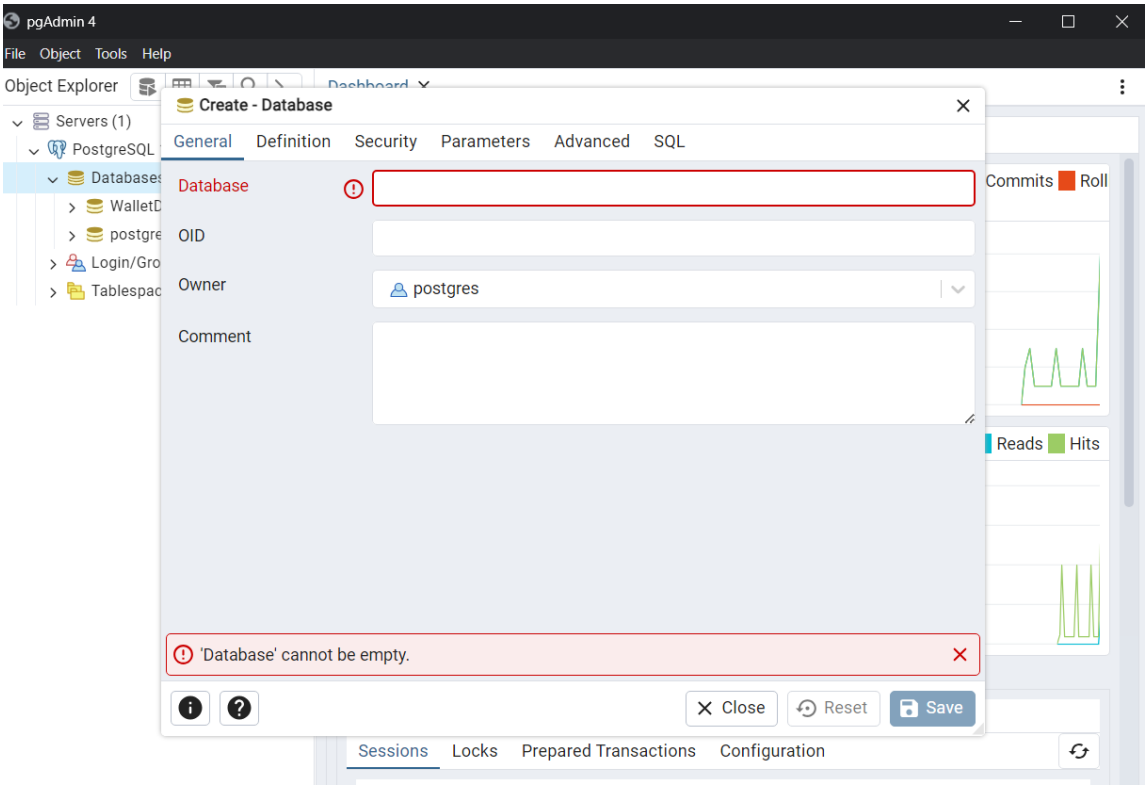



Рисунок 3.8 – Вікно налаштування нової БД

В результаті введення необхідної інформації та збереження, база даних буде створена. Ці дії я не буду повторювати, оскільки в мене вже є заздалегідь створена база даних з назвою «WalletDB».

Після створення бази даних можна створювати таблиці шляхом використання графічного інтерфейсу pgAdmin або шляхом написання SQL-запитів. Якщо мова йде про створення за допомогою pgAdmin та його інтерфейсу, то можна просто перейти до створеної БД у дереві навігації, обрати створену базу даних та виконати опцію “Створити” для таблиць. Після цього відкриється вікно, де можна вказати назву таблиці та визначити її структуру, включаючи назви стовпців, типи даних, обмеження та інші параметри продемонстровані на рисунку 3.9.

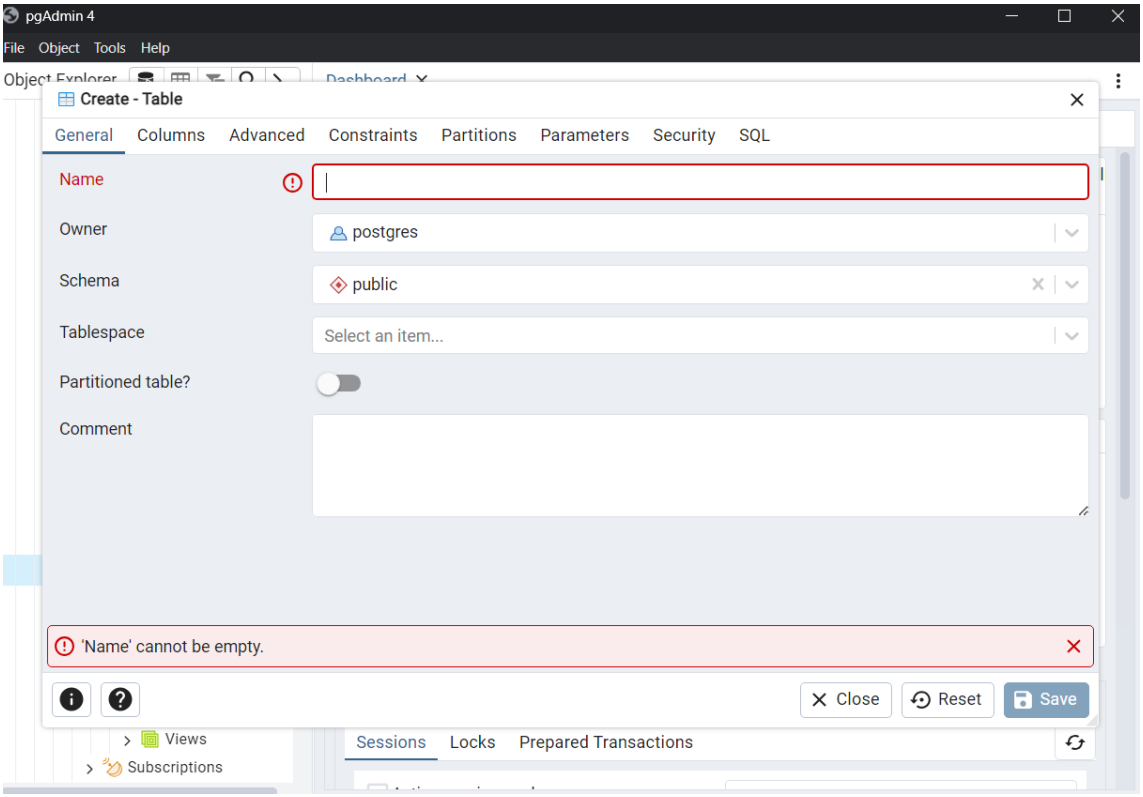



Рисунок 3.9 – Створення таблиць за допомогою графічного інтерфейсу pgAdmin

Як вже було сказано раніше, можна створювати таблиці через введення власних SQL-запитів. Для цього можна скористатися консоллю pgAdmin, перейшовши до верхнього меню виберіть “Tools” і далі “Query Tool”, після чого відкриється вікно для написання запитів, що на рисунку 3.10.

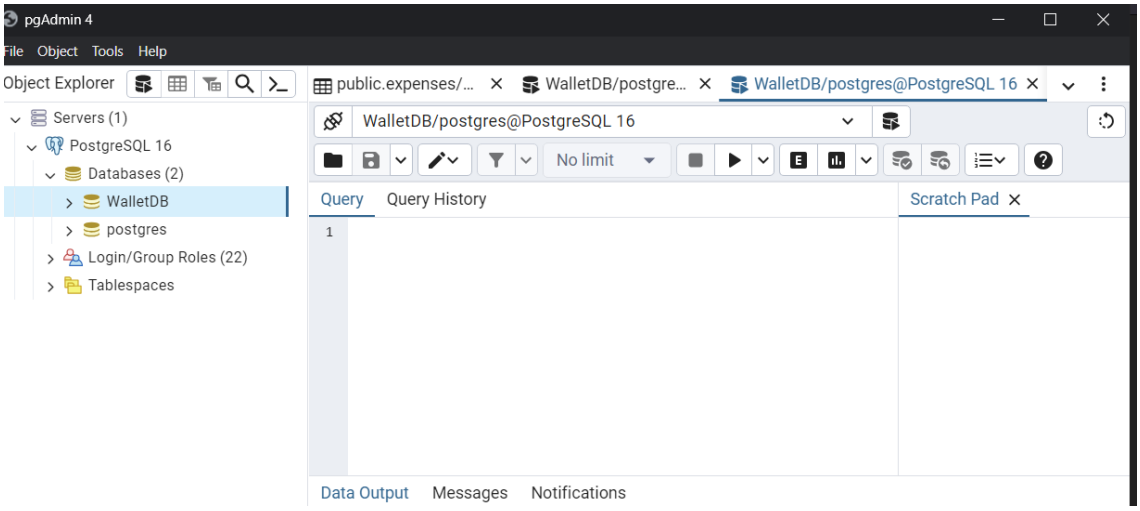



Рисунок 3.10 – Вікно для написання SQL-запитів.

Перейдемо до створення необхідних таблиць для мого проєкту за допомогою SQL-запитів, приклад наведено на рисунках 3.11 – 3.13.

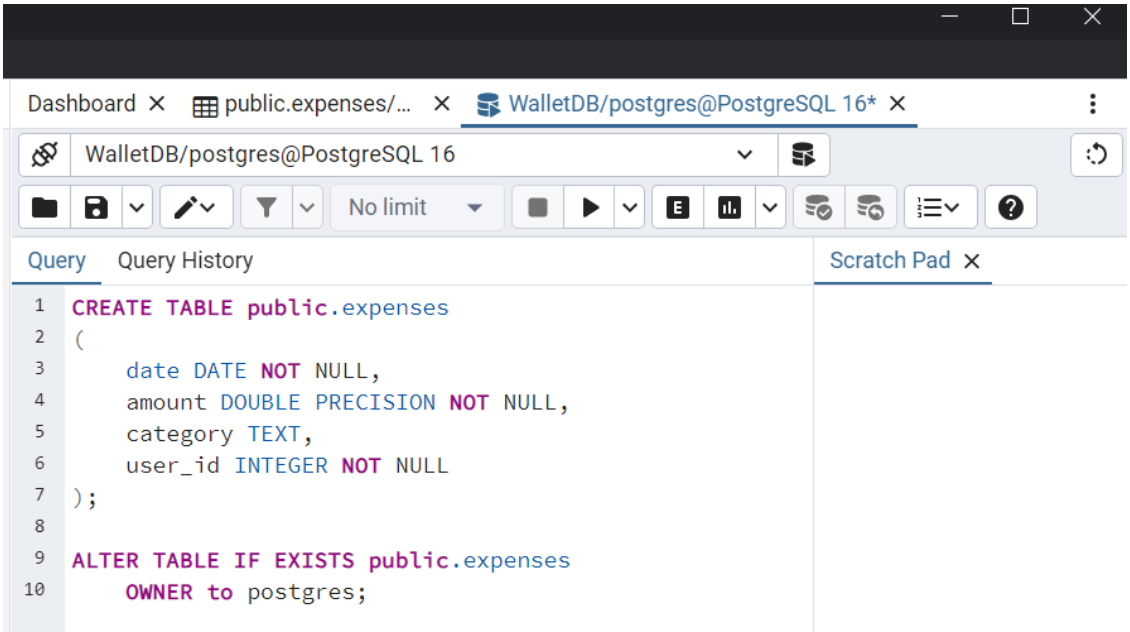



Рисунок 3.11 - Створення таблиці «expenses»


Рисунок 3.12 - Створення таблиці «incomes»

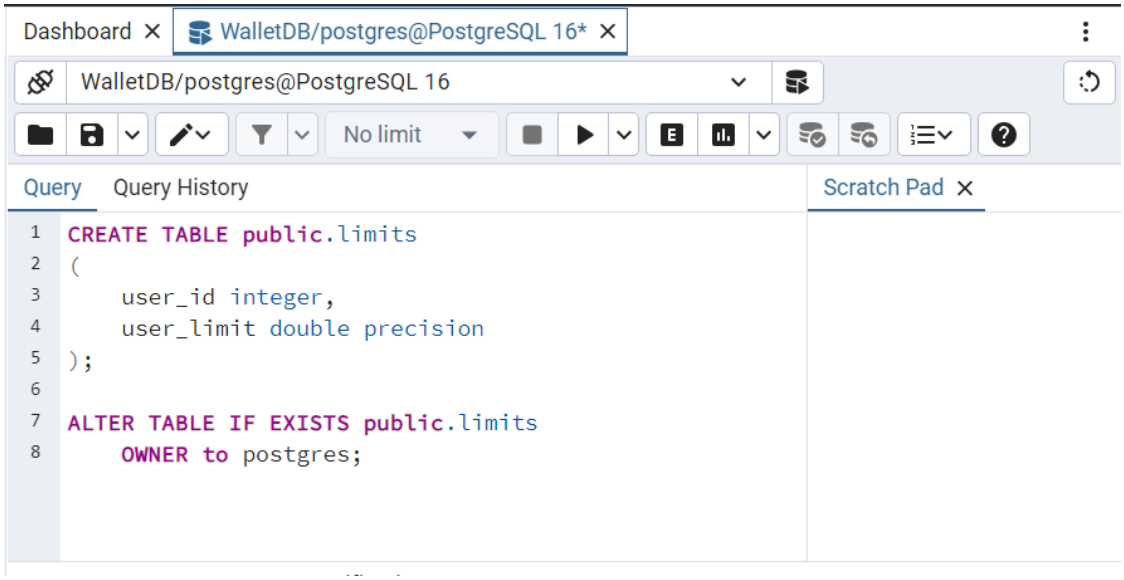



Рисунок 3.13 - Створення таблиці «limits»

Структуру бази даних можна побачити на рисунку 3.14.

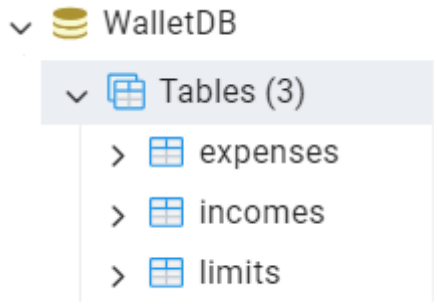



Рисунок 3.14 – Структура БД

Використовуючи SQL-запити в інтерфейсі pgAdmin, було успішно було створено базу даних та таблиці, необхідні для подальшого реалізації прокту.

### 3.4. Розробка програмної частини

Для початку програмної розробки необхідно створити проєкт у вибраній IDE, в моєму випадку це PyCharm. Я структурував свій проєкт на папки відповідно до того функціоналу який присутній в ньому, приклад наведено на рисунку 3.15.

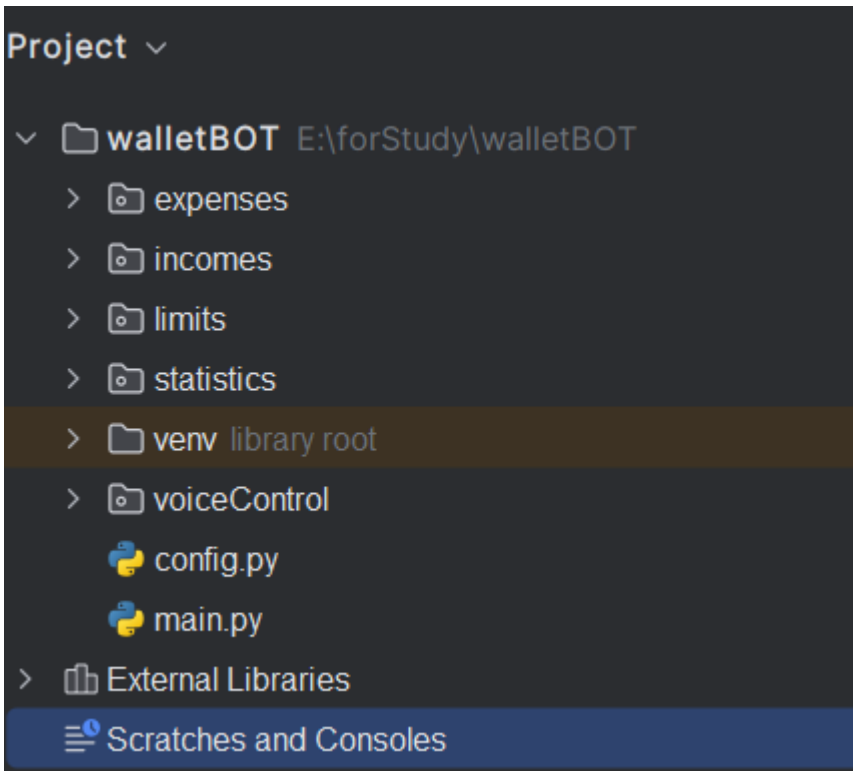



Рисунок 3.15 – Структура файлів проєкту

- Папка expenses зберігає в собі файли з кодом, що відповідають за обробку та збереження даних до таблиці про витрати, як голосовим методом так і звичайним.
- Папка incomes зберігає в собі файли з кодом, що відповідають за обробку та збереження даних до таблиці про надходження, як голосовим методом так і звичайним.

- Папка limits зберігає в собі файл з кодом, що відповідає за збереження даних про ліміти користувача.
- Папка statistics зберігає в собі файл з кодом, що відповідає за обробку даних з таблиць про витрати та надходження, та подання цих даних у вигляді статистики.
- Папка voiceControl зберігає в собі файли з кодом, що відповідають за транскрибацію голосових повідомлень, та обробку інформації з отриманого тексту.
- Файл config.py містить у собі код для підключення до БД, а також токен боту.
- Файл main.py містить у собі код для старту та функціонування проєкту.
- Решта не описаних файлів - це стандартні файли та папки, що додаються автоматично при створенні проєкту.

Файл main.py є головним файлом проєкту, з нього розпочинається програма. Почнемо саме з обробки команди “/start” та додавання кнопок, з якими буде взаємодіяти користувач, приклад написання коду зображено на рисунках 3.16. – 3.17.


Рисунок 3.16 – Обробка команди /start

```
def main_menu_markup():
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    spending = types.KeyboardButton('📉Додати витрати')
    limit = types.KeyboardButton('🚩Ліміт')
    income = types.KeyboardButton('📈Додати дохід')
    statistics = types.KeyboardButton('📊Статистика')
    other = types.KeyboardButton('🗣️Голосове керування')
    markup.add( *args: statistics, income, limit, spending, other)
    return markup
```

Рисунок 3.17 – Створення кнопок

Декоратор @bot.message_handler(commands=['start']) вказує на те, що функція main обробляє команду /start. Функція main викликає main_menu_markup(), щоб отримати кнопки меню, а також надсилає повідомлення користувачу з текстом привітання та його ім'ям. Тепер, коли необхідні кнопки створено, треба описати їх функціонал.

Додавання витрат та надходжень:

При натисканні кнопки “Додати витрати”, викликається функція add_expense(), зображена на рисунку 3.18, яка передає обробку витрат до хендлера add_expense_handler.

```
@bot.message_handler(func=lambda message: message.text == '📉Додати витрати')
def add_expense(message):
    add_expense_handler(bot, message, main)
```

Рисунок 3.18 – Функція add_expenses

В свою чергу, за допомогою функції add_expense_handler, бот починає взаємодію з користувачем надсилаючи запит на обрання дати витрати за допомогою календаря, бібліотеки telebot_calendar, перевіряючи дані на валідність, приклад коду наведено на рисунку 3.19. Було вирішено, що бот буде приймати дати тільки в поточному місяці.

```python
@bot.callback_query_handler(func=lambda call: call.data.startswith("calendar_expense"))
def calendar_handler_expense(call):
    data_parts = call.data.split(":")
    action = data_parts[1]
    year = data_parts[2]
    month = data_parts[3]
    day = data_parts[4]
    if action == "DAY":
        date = f"{year}-{month}-{day}"
        bot.send_message(call.message.chat.id, f'Ви обрали дату: {date}')
        bot.send_message(call.message.chat.id, 'Введіть суму витрати:')
        bot.register_next_step_handler(call.message, lambda msg: get_amount(msg, date))
    else:
        bot.send_message(call.message.chat.id, 'Будь ласка, оберіть дату в поточному місяці👆')
```

Рисунок 3.19 – Функція calendar_handler_expense

Після вибору дати, бот відправляє повідомлення, щоб користувач ввів суму витрати, перевіряючи її на валідність, приклад коду наведено на рисунку 3.20

```python
def get_amount(message, date):
    try:
        amount = message.text
        if amount is None:
            raise ValueError
        else:
            if ',' in amount:
                amount = amount.replace(',', '.')
            amount = float(amount)
            amount = round(amount, 2)
            bot.send_message(message.chat.id, 'Виберіть категорію витрати:', reply_markup=create_categories_keyboard())
            bot.register_next_step_handler(message, lambda msg: save_expense(msg, date, amount))
    except ValueError:
        bot.send_message(message.chat.id, 'Будь ласка, введіть коректну суму витрати (наприклад, "155.50"):')
        bot.register_next_step_handler(message, lambda msg: get_amount(msg, date))
```

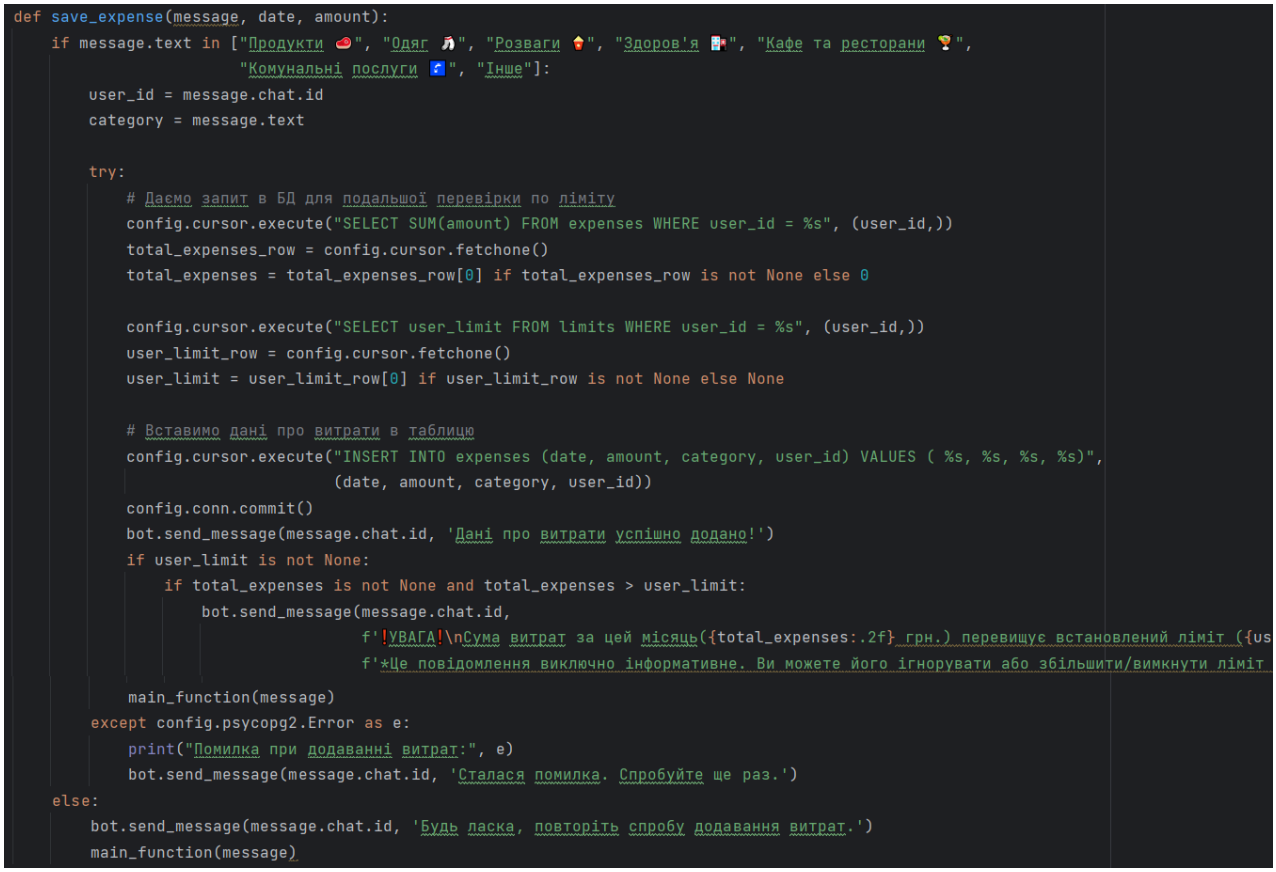
Рисунок 3.20 – Функція get_amount

Далі бот надсилає клавіатуру з категоріями витрат для вибору користувача, приклад коду наведено на рисунку 3.21.

```python
def create_categories_keyboard():
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    categories = ["Продукти 🍖", "Одяг 👔", "Розваги 🍿", "Здоров'я 🏥", "Кафе та ресторани 🍽", "Комунальні послуги 🚰", "Інше"]
    for category in categories:
        markup.add(types.KeyboardButton(category))
    return markup
```

Рисунок 3.21 – Функція create_categories_keyboard

Після вибору категорії, дані про витрату зберігаються в базу даних, приклад коду наведено на рисунку 3.22.



```
def save_expense(message, date, amount):
    if message.text in ["Продукти 🍖", "Одяг 👗", "Розваги 🍿", "Здоров'я 💊", "Кафе та ресторани 🍸",
                        "Комунальні послуги 🚰", "Інше"]:
        user_id = message.chat.id
        category = message.text

        try:
            # Даємо запит в БД для подальшої перевірки по ліміту
            config.cursor.execute("SELECT SUM(amount) FROM expenses WHERE user_id = %s", (user_id,))
            total_expenses_row = config.cursor.fetchone()
            total_expenses = total_expenses_row[0] if total_expenses_row is not None else 0

            config.cursor.execute("SELECT user_limit FROM limits WHERE user_id = %s", (user_id,))
            user_limit_row = config.cursor.fetchone()
            user_limit = user_limit_row[0] if user_limit_row is not None else None

            # Вставимо дані про витрати в таблицю
            config.cursor.execute("INSERT INTO expenses (date, amount, category, user_id) VALUES ( %s, %s, %s, %s)",
                                  (date, amount, category, user_id))
            config.conn.commit()
            bot.send_message(message.chat.id, 'Дані про витрати успішно додано!')
            if user_limit is not None:
                if total_expenses is not None and total_expenses > user_limit:
                    bot.send_message(message.chat.id,
                                     f'!УВАГА!\nСума витрат за цей місяць({total_expenses:.2f} грн.) перевищує встановлений ліміт ({us
                                     f'*Це повідомлення виключно інформативне. Ви можете його ігнорувати або збільшити/вимкнути ліміт
            main_function(message)
        except config.psycopg2.Error as e:
            print("Помилка при додаванні витрат:", e)
            bot.send_message(message.chat.id, 'Сталася помилка. Спробуйте ще раз.')
    else:
        bot.send_message(message.chat.id, 'Будь ласка, повторіть спробу додавання витрат.')
        main_function(message)
```

Рисунок 3.22 – Функція save_expense

Додатково, після збереження даних, функція перевіряє, чи не перевищує сума витрат встановлений користувачем ліміт на місяць. Якщо перевищує, бот надсилає інформаційне повідомлення про це, але не блокує процес додавання витрат. Після завершення операції, бот повертається до головного меню.

Аналогічно до витрат, при натисканні кнопки “Додати дохід” викликається функція add_income(), що на рисунку 3.23, яка передає обробку надходжень до хендлера add_income_handler.

```
@bot.message_handler(func=lambda message: message.text == '📈Додати дохід')
def add_income(message):
    add_income_handler(bot, message, main)
```

Рисунок 3.23 – Функція add_income

Функція add_income_handler майже аналогічна до функції для додавання витрат, тому повний програмний код викладено у додатку А. Користувач також обирає дату за допомогою календаря, потім вводить суму надходження. Однак, на відміну від витрат, не потрібно обирати категорію для надходжень.

Після введення суми надходження, дані зберігаються в базу даних без подальших перевірок на ліміт витрат, оскільки надходження не обмежується лімітом. Інші кроки, такі як обробка помилок та повернення до головного меню після завершення операції, залишаються такими ж, як у випадку з додаванням витрат.

Перегляд статистики:

При натисканні кнопки “Статистика” користувачу надається вибір періоду для перегляду статистики , приклад коду наведено на рисунку 3.24.

```
@bot.message_handler(func=lambda message: message.text == '📊Статистика')
def show_statistics(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    current_month_button = types.KeyboardButton('Поточний місяць')
    previous_month_button = types.KeyboardButton('Попередній місяць')
    last_3_months_button = types.KeyboardButton('Останні 3 місяці')
    back_button = types.KeyboardButton('Назад до головного меню')
    markup.add( *args: current_month_button, previous_month_button, last_3_months_button, back_button)
    bot.send_message(message.chat.id, 'Оберіть період для отримання статистики:', reply_markup=markup)
```

Рисунок 3.24 – Функція show_statistic

Ця функція створює та відправляє користувачу динамічну клавіатуру з варіантами вибору періоду для отримання статистики витрат та доходів. Користувач може обрати один з періодів, після чого інші функції оброблять цей вибір і нададуть статистику за обраний період.

Відповідно до кнопок, створено 3 функції, які обробляють їх за допомогою декораторів і викликають потрібні користувачу функції для показу статистики, що на рисунку 3.25.

```
@bot.message_handler(func=lambda message: message.text == 'Поточний місяць')
def current_month_statistics(message):
    send_statistics_current_month(message)

NotePC
@bot.message_handler(func=lambda message: message.text == 'Попередній місяць')
def previous_month_statistics(message):
    send_statistics_previous_month(message)

NotePC
@bot.message_handler(func=lambda message: message.text == 'Останні 3 місяці')
def last_3_months_statistics(message):
    send_statistics_previous_3_month(message)
```

Рисунок 3.25 – Функції для показу статистики.

Функція send_statistics_current_month відправляє користувачеві статистику за поточний місяць, яка включає загальні витрати, надходження та топ категорій витрат, приклад функції наведено на рисунку 3.26.

```
def send_statistics_current_month(message):
    user_id = message.chat.id
    total_expenses = get_total_expenses(user_id)
    total_incomes = get_total_incomes(user_id)
    top_categories = get_top_categories(user_id)

    message_text = f"📊 Статистика за поточний місяць:\n\n" \
                   f"💸 Сумарні витрати: {total_expenses:.2f} грн.\n" \
                   f"💰 Сумарні надходження: {total_incomes:.2f} грн.\n\n"
                   f"🔝 Топ категорій витрат:\n\n"

    for index, (category, amount) in enumerate(top_categories, start=1):
        message_text += f"{index}. {category}: {amount:.2f} грн.\n"

    bot.send_message(message.chat.id, message_text)
```

Рисунок 3.26 - Функція send_statistics_current_month

Сполучення функцій, яке обирає з таблиць необхідні дані відповідно до обраного періоду, та дозволяє користувачу переглядати статистику витрат та доходів зображено на рисунку 3.27. Також повний код функцій викладено у додатку А.

```
1 usage   NotePC
def get_total_expenses(user_id):...


1 usage   NotePC
def get_total_incomes(user_id):...


1 usage   NotePC
def get_top_categories(user_id):...
```

Рисунок 3.27 – Функції для отримання даних з таблиць

За аналогією написані інші 2 функції формування звітів - last_3_months_statistics та previous_month_statistics, повний програмний код викладено в додатку А.

Меню лімітів:

При натисканні кнопки “Ліміт” відкривається меню для зміни або вимкнення лімітів, що на рисунку 3.28.

```
@bot.message_handler(func=lambda message: message.text == '🚩Ліміт')
def limit_menu(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    change_limit_button = types.KeyboardButton('Змінити ліміт')
    disable_limit_button = types.KeyboardButton('Вимкнути ліміт')
    back_button = types.KeyboardButton('Назад до головного меню')
    markup.add( *args: change_limit_button, disable_limit_button, back_button)
    bot.send_message(message.chat.id, 'Меню ліміту', reply_markup=markup)
```

Рисунок 3.28 – Меню лімітів

Дві функції відповідають за обробку команд, які вводять користувачі для зміни свого ліміту витрат або його вимкнення зображено на рисунку 3.29. Вони сприймають відповідні текстові повідомлення від користувачів і запускають відповідні дії в залежності від команди, яку отримали.

```
@bot.message_handler(func=lambda message: message.text == 'Змінити ліміт')
def change_limit(message):
    bot.send_message(message.chat.id, 'Введіть новий ліміт:')
    bot.register_next_step_handler(message, limit.set_limit)

 NotePC
@bot.message_handler(func=lambda message: message.text == 'Вимкнути ліміт')
def disable_limit(message):
    limit.disable_limit(message)
```

Рисунок 3.29 – Функції для обробки команд по зміні/вимкненню ліміту

Функція change_limit обробляє кнопку “Змінити ліміт”. Після отримання такої команди бот надсилає повідомлення користувачу, яке пропонує йому ввести новий ліміт. Після цього бот очікує наступного кроку, який оброблятиме введений користувачем ліміт за допомогою функції set_limit. Ця функція призначена для встановлення нового ліміту витрат для користувача. Після введення користувачем нового ліміту, бот перевіряє правильність введення значення та наявність користувача в базі даних. Якщо користувач вже має встановлений ліміт, то він оновлюється згідно нового значення. Якщо користувача немає в базі даних (тобто, він ще не встановив ліміт), то створюється новий запис з вказаним лімітом, приклад коду наведено на рисунку 3.30.

```
def set_limit(message):
    try:
        new_limit = float(message.text)
        user_id = message.chat.id

        config.cursor.execute("SELECT * FROM limits WHERE user_id = %s", (user_id,))
        user_exists = config.cursor.fetchone()

        if user_exists:
            config.cursor.execute("UPDATE limits SET user_limit = %s WHERE user_id = %s", (new_limit, user_id))
            config.conn.commit()
            bot.send_message(message.chat.id, f"Новий ліміт встановлено: {new_limit}" + " грн.")
        else:
            config.cursor.execute("INSERT INTO limits (user_id, user_limit) VALUES (%s, %s)", (user_id, new_limit))
            config.conn.commit()
            bot.send_message(message.chat.id, f"Новий ліміт створено: {new_limit}" + " грн.")
    except ValueError:
        bot.send_message(message.chat.id, text 'Будь ласка, введіть коректну суму ліміту (наприклад, "9500"):')
        bot.register_next_step_handler(message, set_limit)
```

Рисунок 3.30 - Функція set_limit

Функція disable_limit обробляє кнопку “Вимкнути ліміт”. При отриманні такої команди вона викликає хендлер disable_limit зображений на рисунку 3.31, який вимикає ліміт витрат для користувача.

```python
def disable_limit(message):
    try:
        user_id = message.chat.id
        # Перевірка наявності користувача в базі даних
        config.cursor.execute("SELECT * FROM limits WHERE user_id = %s", (user_id,))
        user_exists = config.cursor.fetchone()

        if user_exists:
            config.cursor.execute("DELETE FROM limits WHERE user_id = %s", (user_id,))
            config.conn.commit()
            bot.send_message(message.chat.id, text: f"Ліміт вимкнено.")
        else:
            bot.send_message(message.chat.id, text: f"У вас ліміт не встановлений, тому нічого і вимикати 🙂")
    except ValueError:
        bot.send_message(message.chat.id, text: 'Щось пішло не так, спробуйте ще раз.')
        bot.register_next_step_handler(message, disable_limit)
```

Рисунок 3.31 - Функція disable_limit

Функція disable_limit перевіряє наявність користувача в базі даних. Якщо користувач вже має встановлений ліміт, то його запис видаляється, що призводить до вимкнення ліміту. Якщо ж користувача немає в базі даних (тобто, він не встановив ліміт), бот повідомляє що ліміт відсутній.

Голосове керування:

Функція для голосового керування, яка дозволяє користувачам додавати витрати або доходи за допомогою голосових повідомлень. Функція voice_control_menu відповідає за обробку голосового керування, яка зображена на рисунку 3.32. В межах цієї функції визначається внутрішня функція addVoice, яка обробляє голосові повідомлення. Коли бот отримує голосове повідомлення, він передає його на обробку до функції voice_control.

```python
@bot.message_handler(func=lambda message: message.text == '🗣Голосове керування')
def voice_control_menu(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    back_button = types.KeyboardButton('До головного меню 🔙')
    markup.add(back_button)
    bot.send_message(message.chat.id,'Ви можете записати голосове повідомлення, аби додати витрати/надходження
                     reply_markup=markup)
    NotePC
    @bot.message_handler(content_types=['voice'])
    def addVoice(message):
        voice_control(message)
```

Рисунок 3.32 – Функція voice_control_menu

Функція voice_control, яка зображена на рисунку 3.33, виконує основну обробку голосових повідомлень. Спочатку вона використовує функцію voice_recognize для перетворення голосового повідомлення в текст.

```
def voice_control(message):
    text = voice_recognize(message)
```

Рисунок 3.33 – Перетворення голосового повідомлення в текст, за допомогою voice_recognize

Функція voice_recognize зображена на рисунку 3.34.

```
def voice_recognize(message):
    file_name = f'{message.voice.file_id}.ogg'
    file_info = bot.get_file(message.voice.file_id)
    downloaded_file = bot.download_file(file_info.file_path)
    with open(file_name, 'wb') as new_file:
        new_file.write(downloaded_file)

    file_name_wav = f'{message.voice.file_id}.wav'
    subprocess.call( args: ['ffmpeg', '-i', file_name, file_name_wav], stdout=subprocess.PIPE, stderr=subprocess.PIPE)

    with sr.AudioFile(file_name_wav) as source:
        audio = r.record(source)
    try:
        text = r.recognize_google(audio, language='uk-UA')
        bot.send_message(message.chat.id, text)
        return text
    except sr.UnknownValueError:
        text = "none"
        return text
    finally:
        os.remove(file_name)
        os.remove(file_name_wav)
```

Рисунок 3.34 – Перетворення голосового повідомлення в текст

Ми використовуємо бібліотеку speech_recognition для розпізнавання мови, subprocess для виконання команд у терміналі (конвертації аудіо), os для роботи з файловою системою (створення та видалення файлів).

Функція voice_recognize починається із завантаження голосового повідомлення у форматі .ogg з сервера Telegram. Спочатку створюється ім'я

файлу на основі унікального ідентифікатора голосового повідомлення. Потім завантажується інформація про файл з сервера, і цей файл зберігається у форматі .ogg.

Після збереження аудіофайлу, його конвертуємо у формат .wav, тому що цей формат краще працює з бібліотекою speech_recognition. Для цього використовується утиліта ffmpeg. Створюється ім'я для нового файлу у форматі .wav і виконується команда ffmpeg, яка конвертує аудіофайл. Далі файл у форматі .wav відкривається і зчитується для подальшого розпізнавання. Використовуючи метод recognize_google з speech_recognition, аудіо обробляється для розпізнавання тексту, з налаштуванням на українську мову. Якщо розпізнавання успішне, результат у вигляді тексту повертається функцією. У випадку, якщо розпізнавання не вдалося (наприклад, якщо мова була нерозбірливою), повертається текст “none”. В кінці виконання, тимчасові файли видаляються з локальної системи, щоб уникнути накопичення непотрібних файлів.

Після успішного розпізнавання голосового повідомлення функція voice_control отримує текст користувача. Далі йде аналіз цього тексту використовуючи регулярні вирази для визначення суми, категорії та типу операції (витрата чи дохід), приклад коду наведено на рисунку 3.35.

```
amount_pattern = re.compile( pattern: r'\b\d+\b(?=\s*(?:грн?|гривень))', re.IGNORECASE)
decimal_amount_pattern = re.compile( pattern: r'\b\d+\b(?=\s*(?:копійки?|копійок|коп|ко))', re.IGNORECASE)
category_pattern = re.compile( pattern: r'продукти|їжа|взуття|одяг|розваги|здоров\'я|аптека|кафе|ресторани|інше', re.IGNORECASE)
expense_pattern = re.compile( pattern: r'витрати|витрату|розхід', re.IGNORECASE)
income_pattern = re.compile( pattern: r'надходження|дохід|прибуток', re.IGNORECASE)
statistic_pattern = re.compile( pattern: r'(показати|покажи)\s+(мені\s+)?статистику|стат|ста', re.IGNORECASE)

integer_amounts = [int(match.group()) for match in amount_pattern.finditer(text)]
decimal_amounts = [float(match.group()) / 100 for match in decimal_amount_pattern.finditer(text)]
```

Рисунок 3.35 – Регулярні вирази функції voice_control

amount_pattern: Шукає цілі числа перед словами “грн” або “гривень”. Використовується для виявлення основної частини суми.

decimal_amount_pattern: Шукає цілі числа перед словами “копійки”, “копійок”, “коп”, “ко”. Використовується для виявлення копійок.

category_pattern: Шукає слова, що відповідають різним категоріям витрат (наприклад, “продукти”, “їжа”, “одяг” тощо).

expense_pattern: Шукає слова, пов'язані з витратами (наприклад, “витрати”, “витрату”, “розхід”).

income_pattern: Шукає слова, пов'язані з доходами (наприклад, “надходження”, “дохід”, “прибуток”).

statistic_pattern: Шукає слова, пов'язані з запитом статистики (наприклад, “показати мені статистику”, “стат”).

integer_amounts: Використовує amount_pattern для пошуку всіх відповідних чисел у тексті, конвертуючи їх у список цілих чисел. Це основна частина суми в гривнях.

decimal_amounts: Використовує decimal_amount_pattern для пошуку всіх відповідних чисел у тексті, конвертуючи їх у список чисел з плаваючою комою, які представляють копійки.

Після аналізу тексту, якщо операція розпізнана як витрата, викликається add_voice_expense_handler, а якщо як дохід – add_voice_income_handler, і відповідно до функції передається сума та категорія, або просто сума. Якщо розпізнано запит на статистику, викликається функція send_statistics_current_month. Приклад коду наведено на рисунку 3.36.

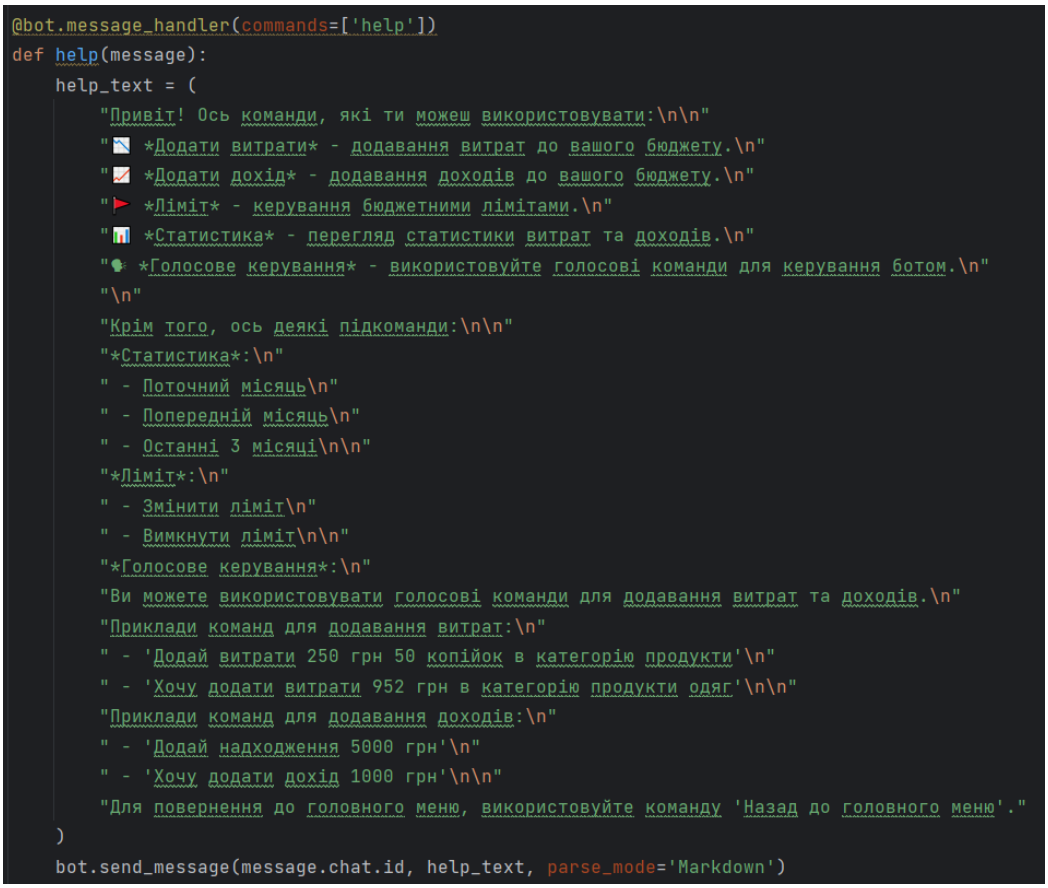
```
if category in category_mapping:
    category = category_mapping[category]
else:
    category = None
if expense_match:
    if not amount or not category:
        bot.send_message(message.chat.id, "Не вдалося розпізнати суму, категорію або тип операції.")
    else:
        add_voice_expense_handler(bot, message, amount, category)
elif income_match:
    if not amount:
        bot.send_message(message.chat.id, "Не вдалося розпізнати суму, категорію або тип операції.")
    else:
        add_voice_income_handler(bot, message, amount)
elif statistic_match:
    send_statistics_current_month(message)
else:
    bot.send_message(message.chat.id, "Не вийшло розпізнати команду 😕 Повторіть спробу ")
```

Рисунок 3.36 – Виклик функцій відповідно до розпізнаного тексту.

Функції add_voice_expense_handler та add_voice_income_handler мають багато схожостей з функціями add_expense_handler та add_income_handler, які були описані раніше. Але їх особливістю є те, що вони обробляють витрати або надходження, додані через голосові команди і не потребують ручного вводу категорій та сум витрат або надходжень, так як вони передаються в ці функції в якості параметрів, повний код можна переглянути у додатку А. Тому користувачу залишається лише зробити вибір дати (витрати чи надходження), і дані будуть записані до відповідної таблиці в базі даних. Це дозволяє автоматизувати процес введення, роблячи його більш зручним для користувача, який може просто озвучити свої витрати або надходження, а бот обробить та збереже ці дані.

**Допомога**

Функція для виведення довідкової інформації з прикладами команд наведена на рисунку 3.37. Коли користувач буде вводити команду /help, йому буде виводитись повідомлення з описом кнопок та прикладом команд для голосового керування



```python
@bot.message_handler(commands=['help'])
def help(message):
    help_text = (
        "Привіт! Ось команди, які ти можеш використовувати:\n\n"
        "📉 *Додати витрати* - додавання витрат до вашого бюджету.\n"
        "📈 *Додати дохід* - додавання доходів до вашого бюджету.\n"
        "🚩 *Ліміт* - керування бюджетними лімітами.\n"
        "📊 *Статистика* - перегляд статистики витрат та доходів.\n"
        "🗣️ *Голосове керування* - використовуйте голосові команди для керування ботом.\n"
        "\n"
        "Крім того, ось деякі підкоманди:\n\n"
        "*Статистика*:\n"
        " - Поточний місяць\n"
        " - Попередній місяць\n"
        " - Останні 3 місяці\n\n"
        "*Ліміт*:\n"
        " - Змінити ліміт\n"
        " - Вимкнути ліміт\n\n"
        "*Голосове керування*:\n"
        "Ви можете використовувати голосові команди для додавання витрат та доходів.\n"
        "Приклади команд для додавання витрат:\n"
        " - 'Додай витрати 250 грн 50 копійок в категорію продукти'\n"
        " - 'Хочу додати витрати 952 грн в категорію продукти одяг'\n\n"
        "Приклади команд для додавання доходів:\n"
        " - 'Додай надходження 5000 грн'\n"
        " - 'Хочу додати дохід 1000 грн'\n\n"
        "Для повернення до головного меню, використовуйте команду 'Назад до головного меню'."
    )
    bot.send_message(message.chat.id, help_text, parse_mode='Markdown')
```

Рисунок 3.37 – Функція help

Декоратор @bot.message_handler(commands=['help']) вказує на те, що функція help обробляє команду користувача /help.

**Обробка невідомих команд**

Функцію для обробки невідомих команд, тих що не описані під час розробки наведено на рисунку 3.38.

```
@bot.message_handler(func=lambda message: True)
def handle_text(message):
    bot.send_message(message.chat.id, 'Вибачте, я не розумію, про що йде мова.
```

Рисунок 3.38 – Функція handle_text

Останній рядок файлу main.py запускає бота в режимі постійного опитування серверу Telegram, його проілюстровано на рисунку 3.39.

```
bot.polling(none_stop=True)
```

Рисунок 3.39 – Запуск бота в режимі «без зупинки»

### 3.5. Тестування Telegram-боту

Після завершення розробки бота було проведено його тестування декількома користувачами. У результаті тестування помилок не було виявлено. Важливо зазначити, що етап тестування Telegram-боту не завершується після цієї перевірки. Постійний моніторинг і збір зворотного зв'язку від користувачів дозволять своєчасно виявляти проблеми і вдосконалювати функціонал.

Для початку роботи з Telegram ботом потрібно знайти його. Для цього слід перейти за посиланням, або у полі пошуку написати його ім’я – WalletBot, приклад написання наведено на рисунку 3.40.

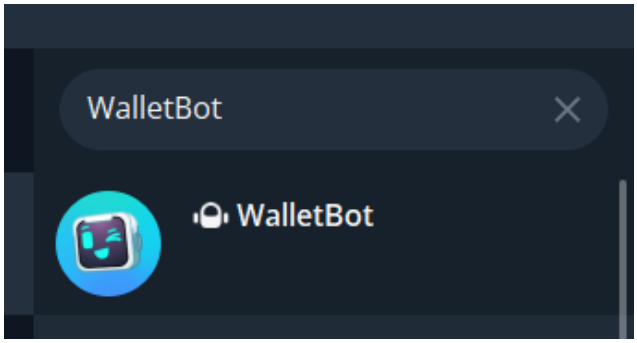



Рисунок 3.40 – Результат пошуку бота

Після відкриття бота з’являється повідомлення з коротким описом бота та кнопка Start, для початку роботи бота, що наведені на рисунку 3.41.

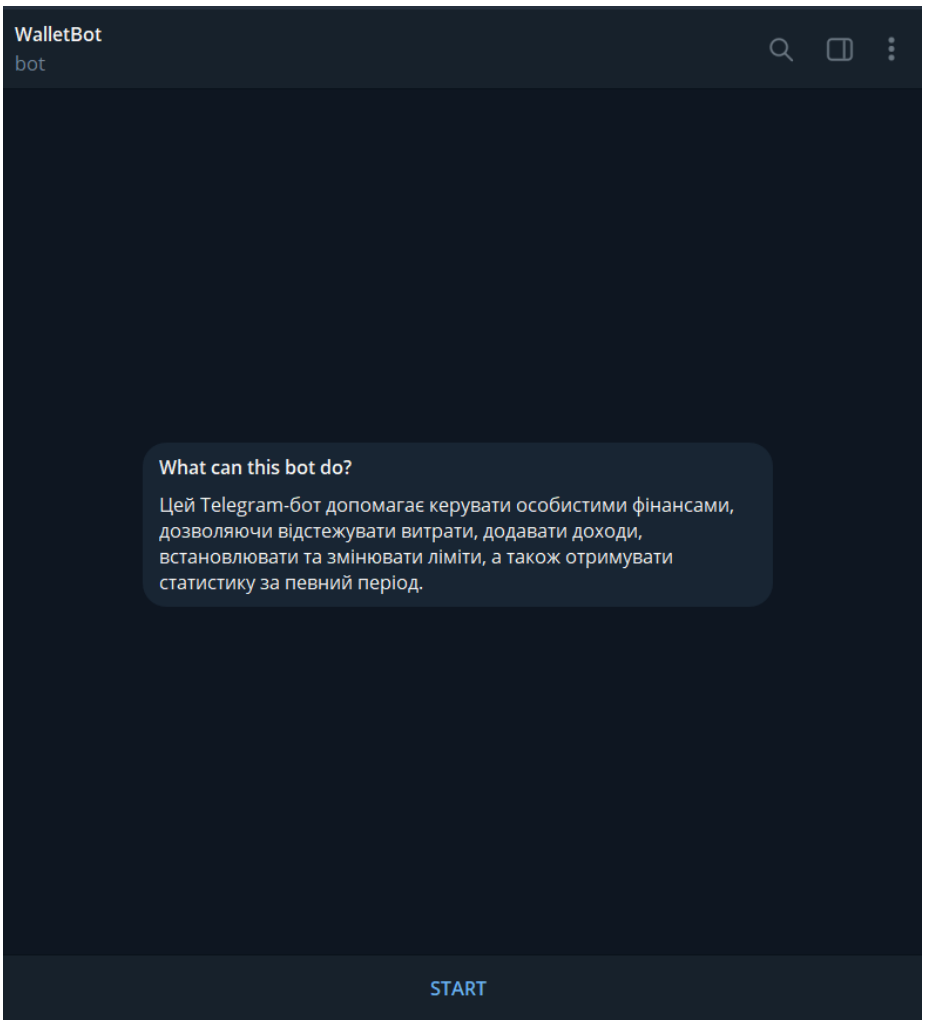



Рисунок 3.41 – Вікно бота перед початком роботи

Далі натискаємо кнопку Start і бачимо вітання та клавіатуру головного меню, наведені на рисунку 3.42.

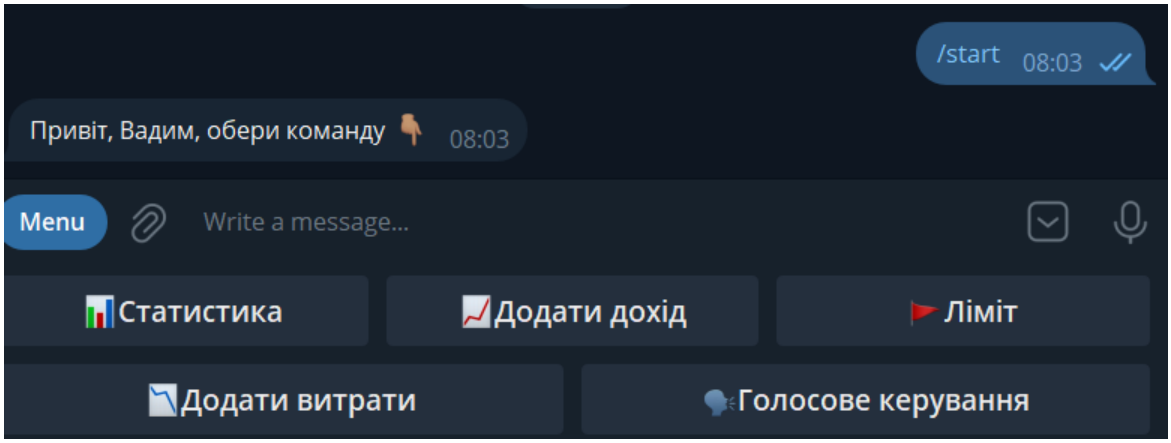



Рисунок 3.42 – Головне меню бота

Для додавання витрат необхідно натиснути кнопку “Додати витрати”. Після цього ввести необхідну інформацію яку запитує бот. Приклад дій наведено на рсиунках 3.43 – 3.45.

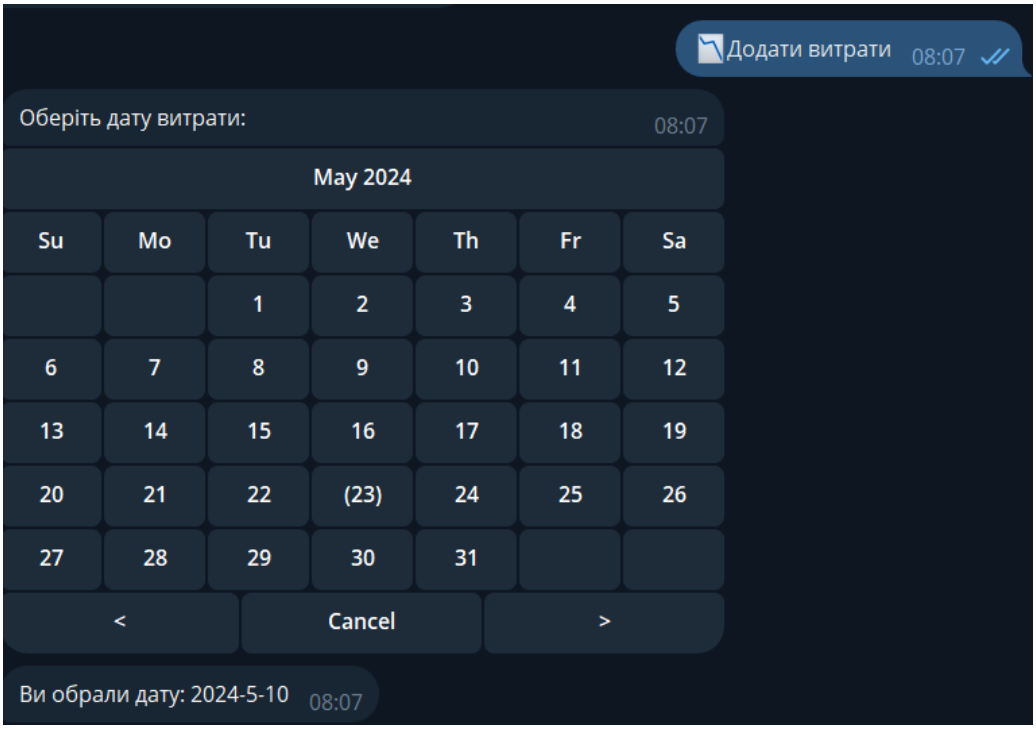



Рисунок 3.43 – Вибір дати витрати


Рисунок 3.44 – Введення суми витрати

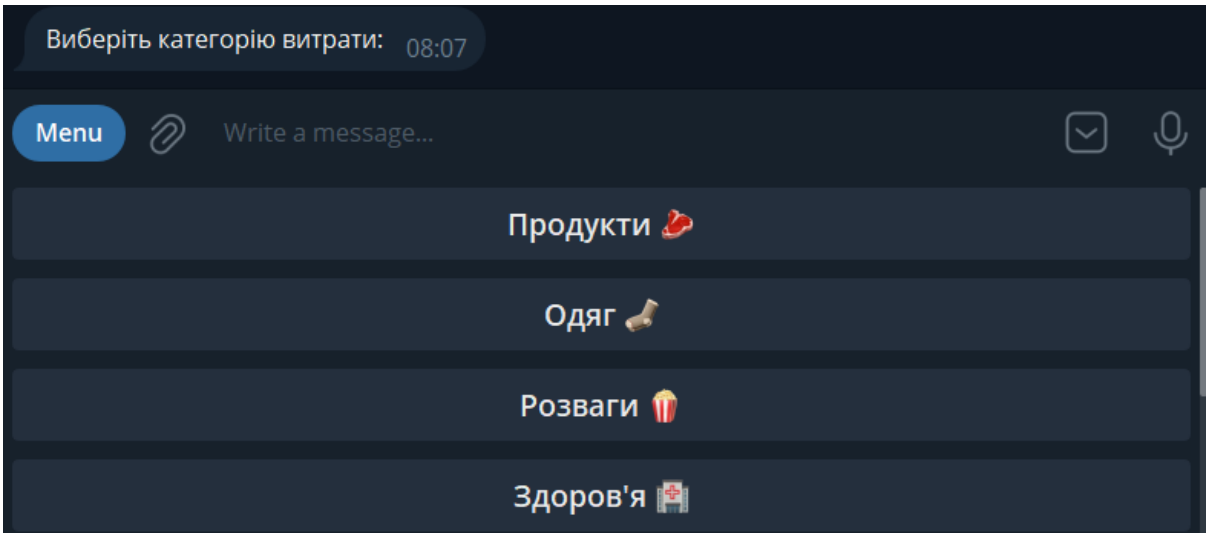



Рисунок 3.45 – Вибір категорії витрати

Після вибору та введення всієї необхідної інформації, бот надсилає повідомлення про те що дані успішно додано до бази даних, та повертає користувача до головного меню, приклад успішного додавання витрат наведено на рисунку 3.46.

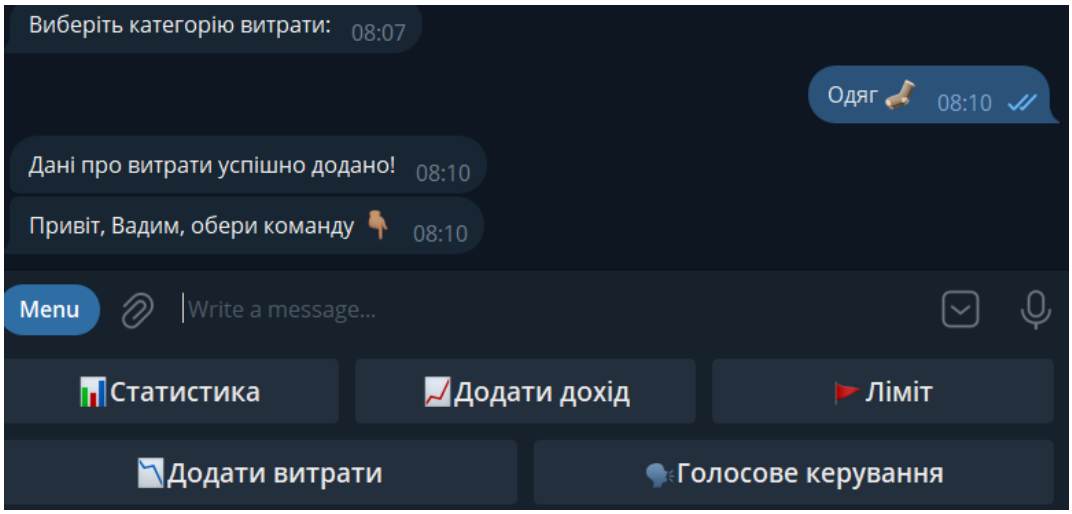


Рисунок 3.46 – Успішне додавання витрати

Крім того, було проведено тести для аналізу поведінки бота на введення невалідних даних, з яким він успішно впорався. Приклади тестів наведені на рисунках 3.47 – 3.49.


Рисунок 3.47 – Невалідний вибір дати в поточному місяці

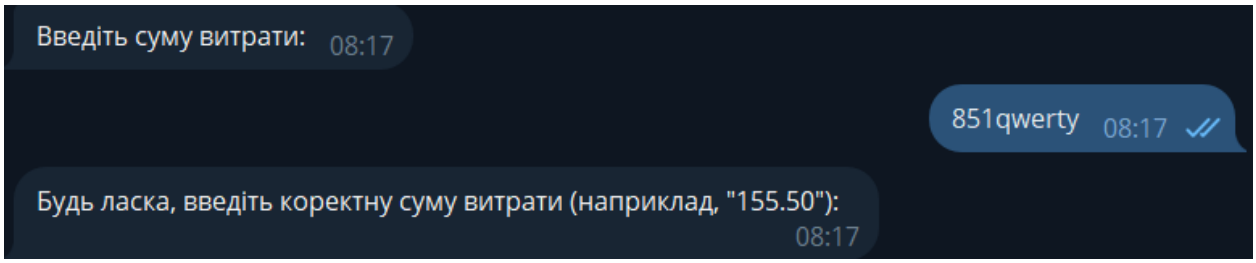



Рисунок 3.48 – Невіладне введення суми витрати

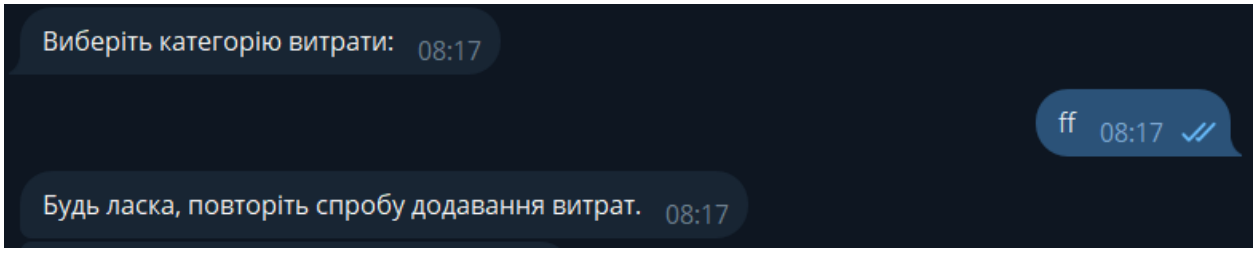



Рисунок 3.49 – Невалідне введення категорії витрат

За аналогією до витрат, було протестовано функцію додавання надходжень, оскільки вони мають між собою велику схожість у програмному коді. Телеграм бот успішно впорався з тестами для додавання надходжень.

Після того як користувач додав витрати та/або надходження, в нього є можливість переглянути цю інформацію за допомогою кнопки “Статистика”. Настикаючи на неї, бот пропонує обрати період, за який необхідно надати звіт, приклад наведено на рисунку 3.50.

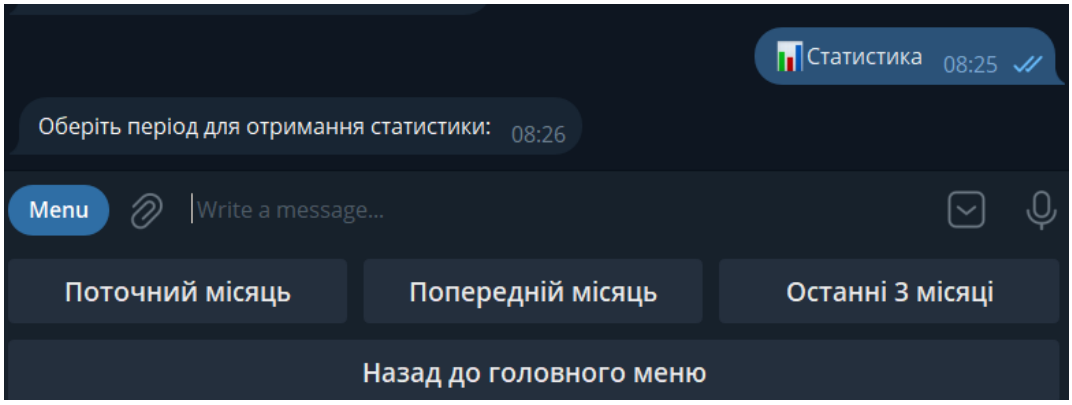



Рисунок 3.50 – Перегляд статистики

Обираємо період, наприклад поточний місяць і отримуємо звіт. Нижче, на рисунках 3.51 – 3.53, наведено такі звіти, де для прикладу вже є багато витрат за категоріями та надходжень аби продемонструвати коректність роботи модуля.

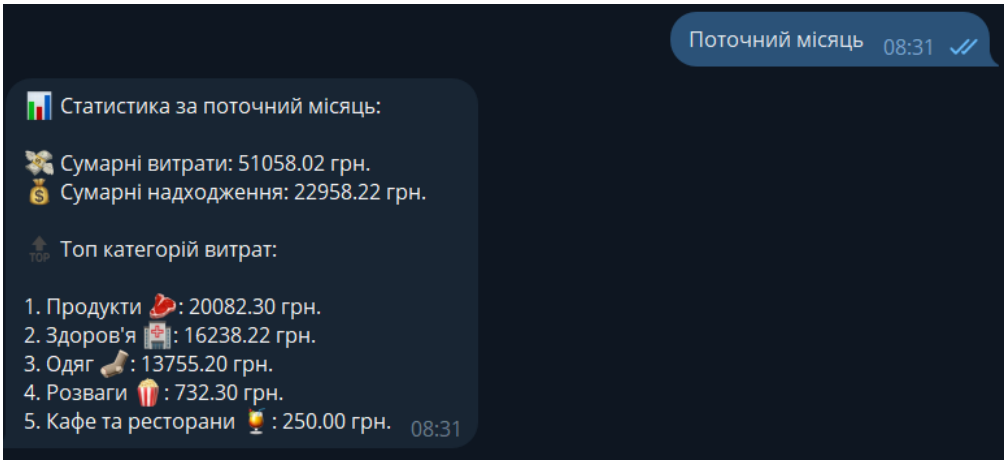



Рисунок 3.51 – Статистика користувача за поточний місяць

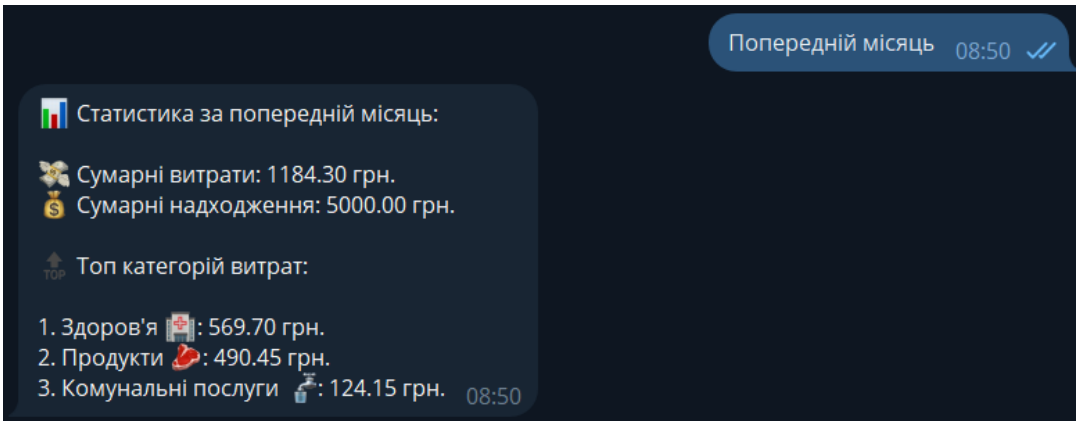



Рисунок 3.52 – Статистика користувача за попередній місяць

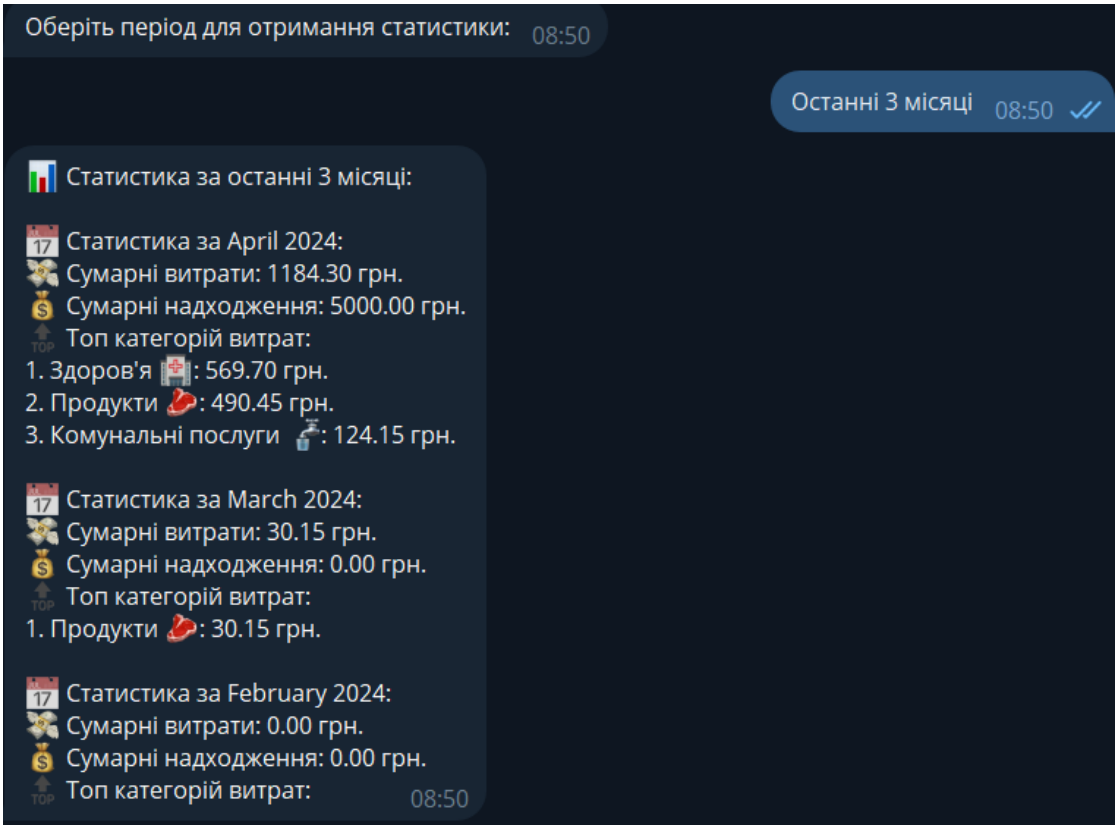



Рисунок 3.53 – Статистика користувача за останні 3 місяці

Як можемо побачити звіти формуються коректно – витрати та надходження відображаються згідно даних в БД за певний період. Крім того, є топ категорій витрат, які упорядковано за спаданням, від найбільш до найменш затратної категорії.

Результат тестування – успішне та коректне функціонування всіх модулів перегляду статистики.

Далі слід протестувати встановлення/зміни ліміту на витрати у поточному місяці. Для цього в нас є кнопка “Ліміт” у головному меню. Натискаємо на неї та потрапляємо до меню ліміту, що на рисунку 3.54

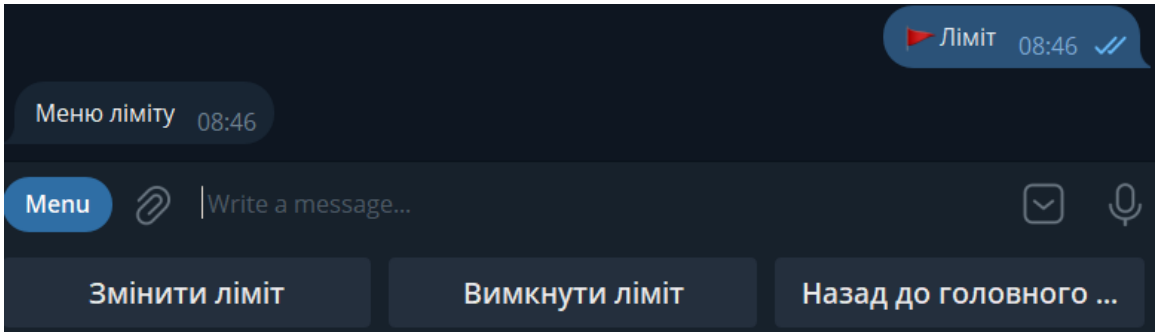



Рисунок 3.54 – Меню ліміту

Користувачу доступні 3 опції – зміна ліміту, вимикання ліміту і повернення до головного меню. Почнемо зі встановлення обмеження. Для цього натискаємо на кнопку “Змінити ліміт”, після чого бот запропонує ввести суму ліміту, яка також перевіряється на валідність, приклад на рисунку 3.55.

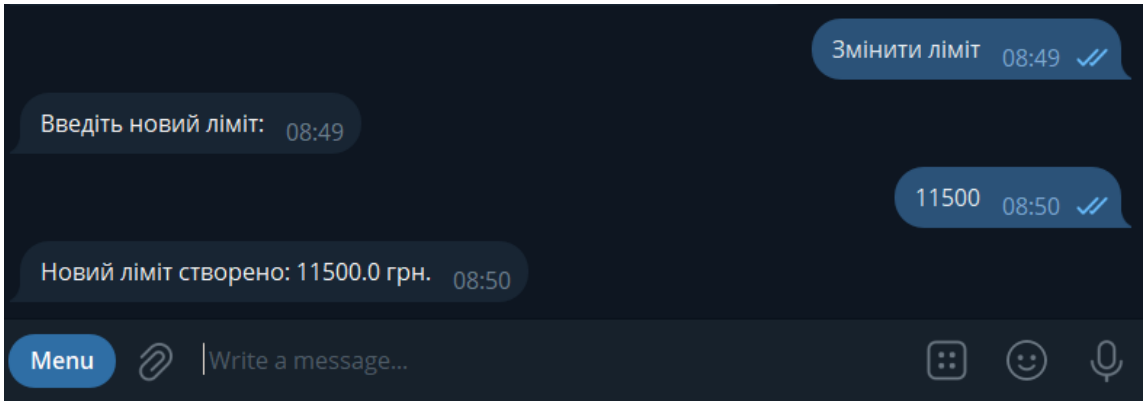



Рисунок 3.55 – Встановлення ліміту

Для того аби протестувати щойно встановлене обмеження, необхідно додати витрату. У випадку коли сумарні витрати за місяць будуть перевищувати встановлений ліміт, буде інформаційне нагадування користувачу що він вже перевищив встановлений поріг витрат на місяць, але інформація до БД і далі буде записуватись, тестування інформаційного повідомлення про ліміт наведено на рисунку 3.56.

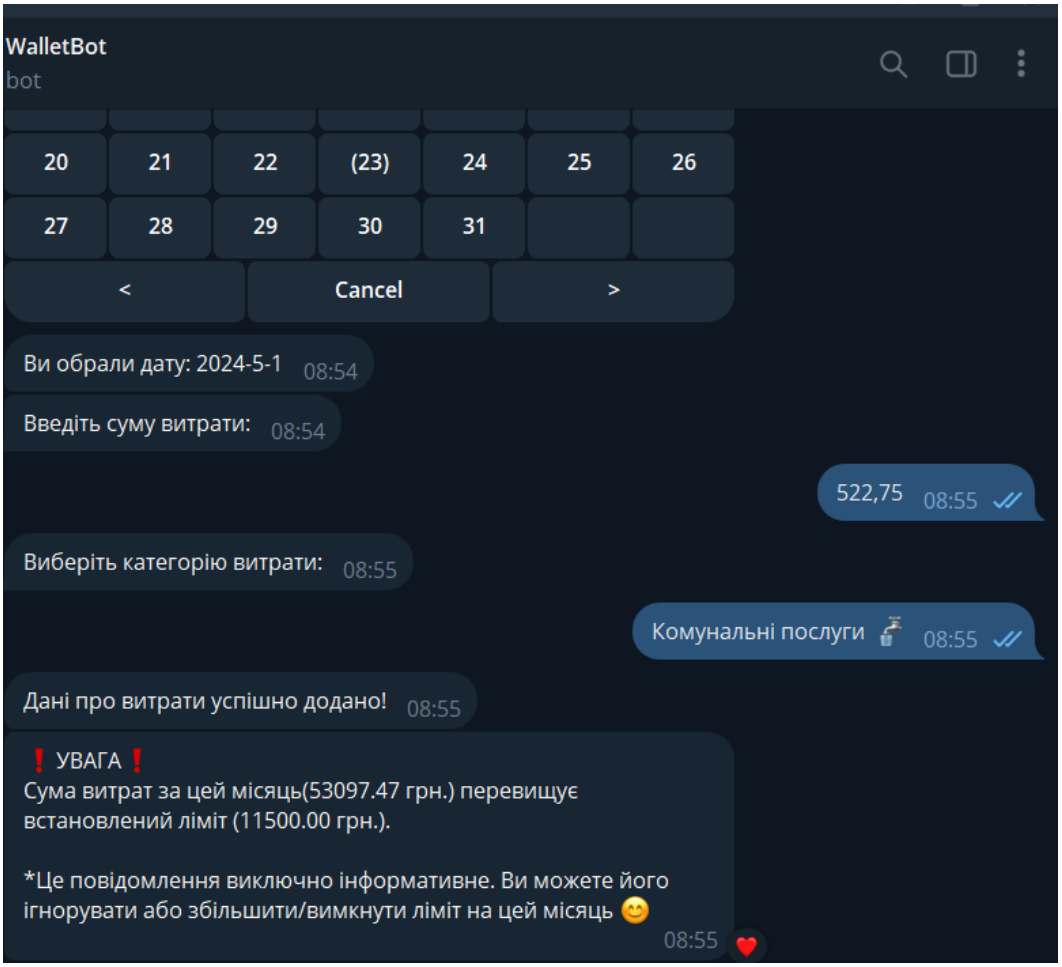



Рисунок 3.56 – Повідомлення про перевищення ліміту витрат

Коли встановлений ліміт буде не потрібний, його можна просто вимкнути відповідною кнопкою у меню ліміту. Для цього необхідно натиснути кнопку “Вимкнути ліміт”, що на рисунку 3.57, після чого встановлений ліміт буде видалено.

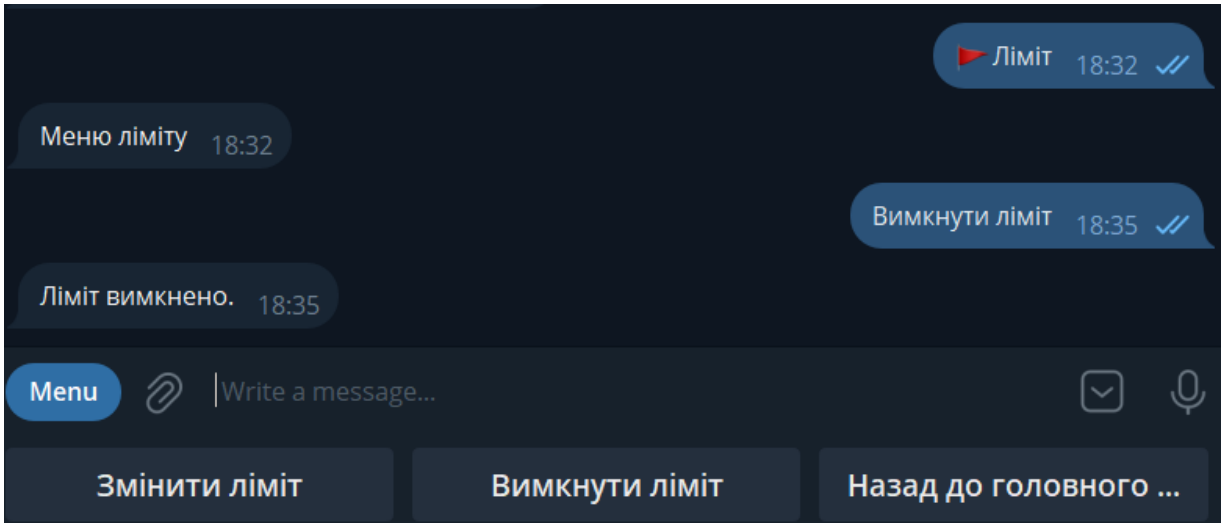



Рисунок 3.57 – Вимкнення ліміту

У випадку коли ліміт не встановлено, а користувач буде намагатись його вимкнути, бот надсилає повідомлення про те що ліміт не встановлено. Приклад повідомлення на рисунку 3.58

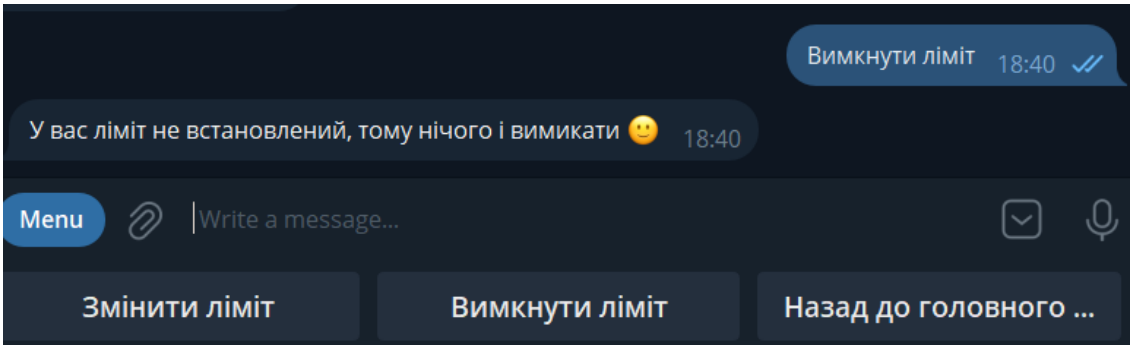



Рисунок 3.58 – Ліміт не встановлено

Після успішного тестування майже всіх модулів бота, та проходження необхідних тестів, залишається останній модуль – голосове керування. Для тестування цього модуля необхідно в головному меню бота натиснути кнопку “Голосове керування”. Після чого, користувачу буде запропоновано записати голосове повідомлення аби додати витрати або здійснити інші дії, повідомлення зображене на рисунку 3.59. Якщо користувачу необхідний

приклад для роботи з чат-ботом, він також зможе натиснути на команду “help”, щоб подивитись приклади голосових команд.

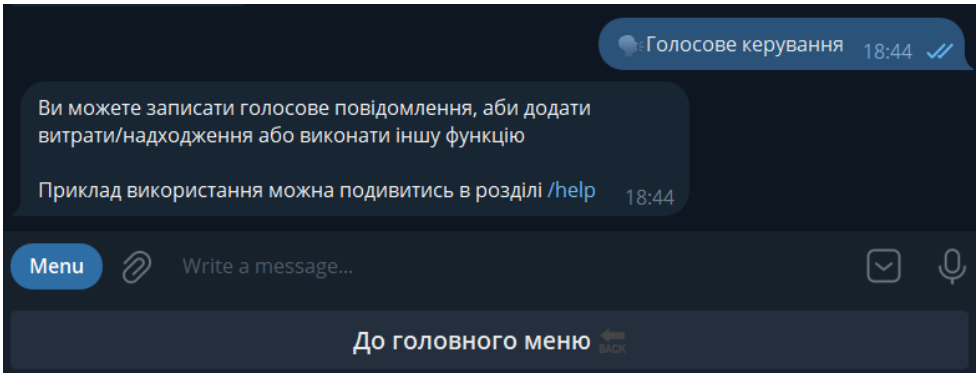



Рисунок 3.59 – меню голосового керування

Записавши голосове повідомлення, бот розпізнає тип операції(витрати чи дохід), суму та категорію (якщо це витрата). Відправляємо боту голосове повідомлення де кажемо “додай витрати 257 гривень 40 копійок в категорію їжа”. Приклад обробки аудіоповідомлення зображено на рисунку 3.60.

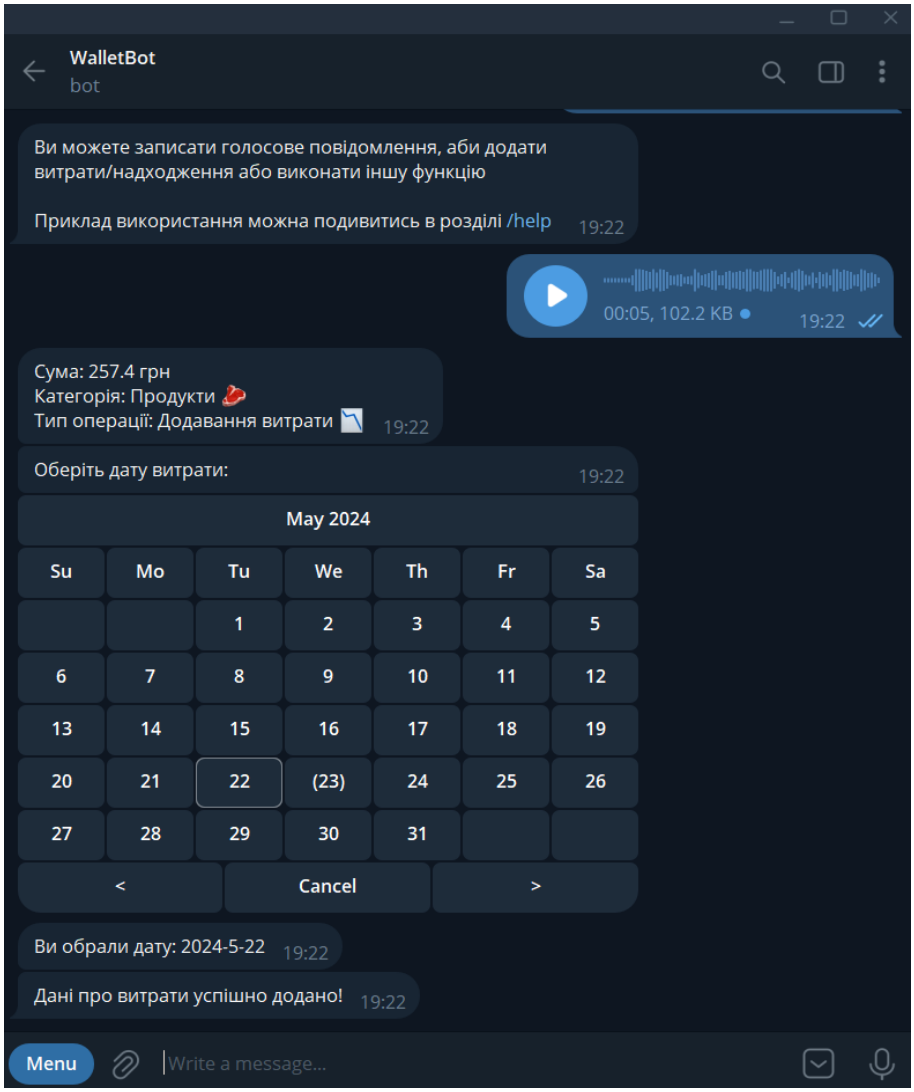



Рисунок 3.60 – Додавання витрат за допомогою аудіоповідомлення

В результаті ми маємо те, що бот підсумовує наше голосове помідомлення виведенням інформації яку йому вдалось обробити, та викликає відповідну функцію, що пропонує обрати дату витрати, після чого додає всю інформації у відповідну таблицю витрат.

За аналогією до витрат, можемо записати голосове повідомлення в якому скажемо “хочу додати надходження у розмірі 5 тисяч гривень”. Після чого знову отримуємо повідомлення від бота, де вказана необхідна сума та тип операції, який він розпізнав з аудіоповідомлення, та прохання вказати дату надходження, що на рисунку 3.61.

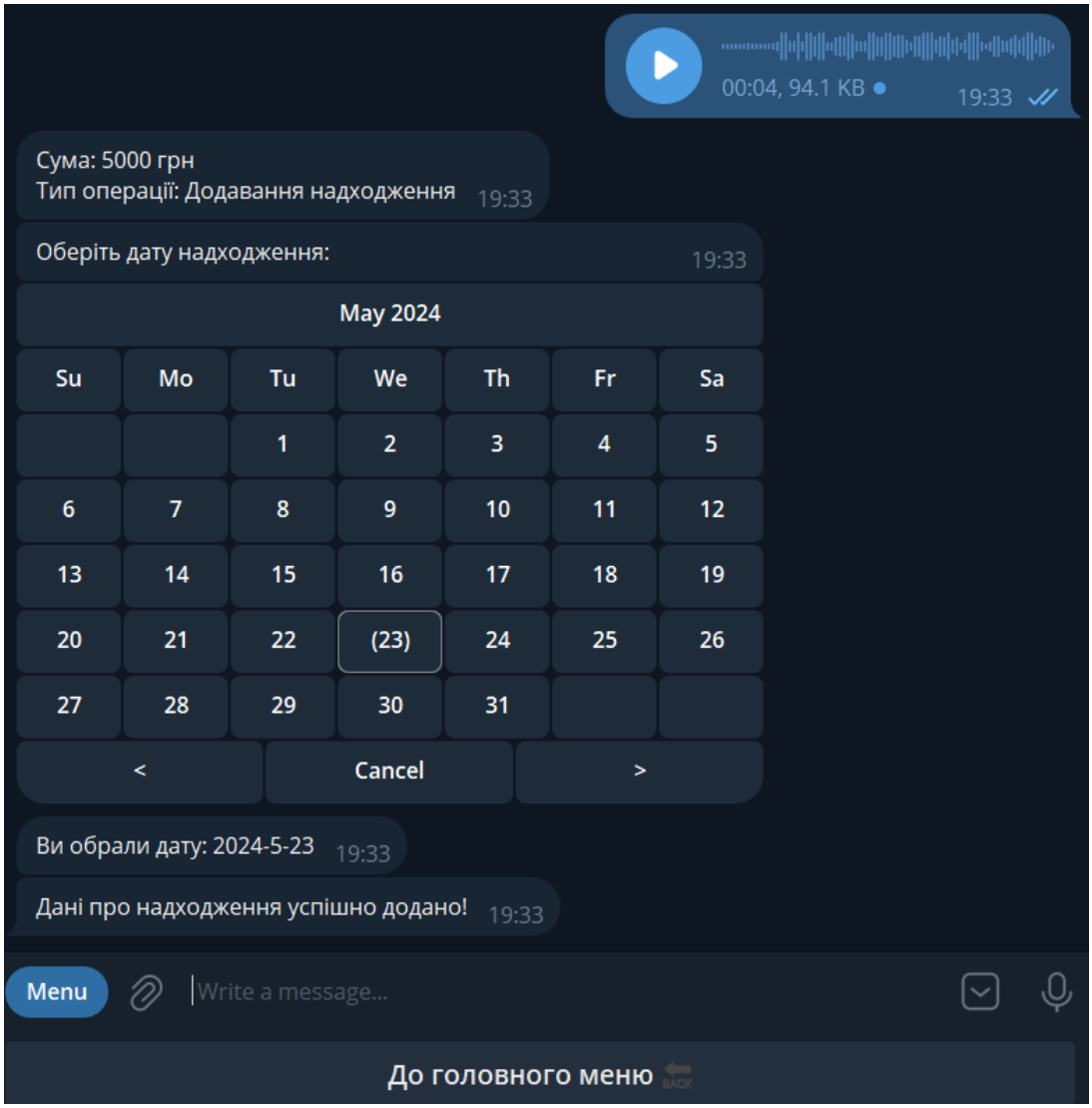



Рисунок 3.61 – Додавання надходжень за допомогою аудіоповідомлення

Слід звернути увагу на те, що фрази для додавання витрат та надходжень відрізняються між собою, отже бот працює не прямолінійно по заданому шаблону, а дійсно розпізнає команди за ключовими словами.

Залишилось перевірити виведення звіту за допомогою голосового повідомлення. Для цього надсилаємо боту аудіоповідомлення, наприклад, “показати статистику” та “покажи мені звіт”, що проілюстровано на рисунку 3.62


Рисунок 3.62 – Отриманню звіту через аудіоповідомлення

Після завершення тестування, можна з впевненістю сказати, що цей Telegram бот успішно пройшов всі тести. Було протестовано додавання витрат та надходжень як у звичний спосіб, так і за допомогою аудіоповідомлень. Крім того, перевірка отримання звітів показала, що фінансова інформація може бути отримана як шляхом натискання кнопок, так і голосовою командою. А чітке та правильне функціонування встановленних лімітів, говорить про надійність цього модулю, що допоможе уникнути перевитрат користувачів.

# ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було розроблено Telegram бот для управління фінансами з можливістю голосового керування. Проаналізувавши актуальність теми було зроблено висновок, що чат-бот для контролю фінансів є надзвичайно актуальною темою в сучасних умовах, оскільки зі зростанням фінансових обов’язків зростає і потреба у контролі бюджету. Зокрема, голосове керування додає новий рівень зручності та доступності в плані взаємодії з чат-ботом.

Для реалізації проєкту було досліджено принципи створення телеграм ботів та основні інструменти для цього. В результаті аналізу було зроблено вибір - проєкт буде написано на мові програмування Python та за допомогою фреймворку pyTelegramBotAPI, а для зберігання даних буде використана СУБД PostgreSQL.

Після вибору інструментів розробки, була розроблена база даних для зберігання фінансових даних, а також написано основний функціонал бота, який включає в себе додавання витрат, доходів, налаштування нагадувань про фінансові обмеження та аналіз фінансів. Також розроблено логіку обробки голосових повідомлень, що дозволяє користувачам керувати чат-ботом за допомогою голосу.

Після завершення розробки, Telegram бот було повністю протестовано. В ході тестування було перевірено швидкість роботи, коректність обробки інформації, точність розпізнавання голосових повідомлень, а також обробку невалідних даних. Після завершення перевірки, можна з впевненістю сказати, що чат-бот успішно пройшов тестування.

Перспективи продовження досліджень у рамках проєкту є досить широкими. По-перше, можна удосконалити логіку обробки голосових команд,

наприклад, забезпечити підтримку різних мов і діалектів. Це дозволить розширити аудиторію користувачів. По-друге, можливим оновленям чат-боту є інтеграція з фінансовими сервісами, такими як банківські додатки. Це дозволить автоматизувати процеси збору даних про фінансові операції.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ


1. Інструкція Git для новачків: що це таке, як він працює та які є основні команди | DAN IT Education. URL: https://dan-it.com.ua/uk/blog/instrukcija-git-dlja-novachkiv-shho-ce-take-jak-vin-pracjuie-ta-jaki-ie-osnovni-komandi/#_Git (дата звернення: 28.04.2024).
2. How to Create a Speech Recognition System with Python. | Reintech Ltd. URL: https://reintech.io/blog/how-to-create-a-speech-recognition-system-with-python (дата звернення: 29.04.2024).
3. A Guide to Speech Recognition in Python: Everything You Should Know | Simplilearn Solutions. URL: https://www.simplilearn.com/tutorials/python-tutorial/speech-recognition-in-python (дата звернення: 29.04.2024).
4. Що таке мова програмування Python? | FREEhost. URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-jazik-programmirovanija-python/ (дата звернення: 28.04.2024).
5. Що таке фреймворк: пояснюємо простими словами. | brainlab. URL: https://brainlab.com.ua/uk/blog-uk/shho-take-frejmvork-poyasnyuyemo-prostymy-slovamy#title_1 (дата звернення: 29.04.2024).
6. Що таке IDE та SDK? "Страшні" терміни простими словами | ApiX-Drive. URL: https://apix-drive.com/ua/blog/useful/ide-i-sdk-strashnye-terminy-prostymi-slovami#chto-takoe-ide (дата звернення: 28.04.2024).
7. Що таке бази даних, їх призначення та види? | FutureNow. Technologies & Science Blog. URL: https://futurenow.com.ua/shho-take-bazy-danyh-yih-pryznachennya-ta-vydy/ (дата звернення: 28.04.2024).

8. Що таке база даних? | Oracle. Oracle and/or its affiliates. URL: https://www.oracle.com/ua/database/what-is-database/ (дата звернення: 28.04.2024).
9. Що таке SQlite? | FREEhost. URL: https://freehost.com.ua/ukr/faq/articles/chto-takoe-sqlite/(дата звернення: 29.04.2024).
10. MongoDB | Krypton. URL: https://krypton.com.ua/tutorial/mongodb/ (дата звернення: 29.04.2024).
11. About PostgreSQL | The PostgreSQL Global Development Group. URL: https://www.postgresql.org/about/ (дата звернення: 29.04.2024).
12. Як створити телеграм бота через BotFather? | ukr-bot.com URL: https://ukr-bot.com/yak-stvoryty-telehram-bota-cherez-botfather/ (дата звернення: 29.04.2024).
13. CREATE TABLE. SQL Commands | The PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/current/sql-createtable.html (дата звернення: 29.04.2024).

# Додаток А

```
#файл main.py
from telebot import types
from config import bot
from expenses.expenses import add_expense_handler
from incomes.incomes import add_income_handler
from statistics.current_month_statistics import send_statistics_current_month
from statistics.last_3_months_statistics import
send_statistics_previous_3_month
from statistics.previous_month_statistics import
send_statistics_previous_month
from voiceControl.voiceControl import voice_control
from limits import limit


def main_menu_markup():
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    spending = types.KeyboardButton('·Додати витрати')
    limit = types.KeyboardButton('·Ліміт')
    income = types.KeyboardButton('·Додати дохід')
    statistics = types.KeyboardButton('·Статистика')
    other = types.KeyboardButton('·Голосове керування')
    markup.add(statistics, income, limit, spending, other)
    return markup

@bot.message_handler(commands=['start'])
def main(message):
    markup = main_menu_markup()
    bot.send_message(message.chat.id, f'Привіт,
{message.from_user.first_name}' + ',' + ' обери команду ⋅',
reply_markup=markup)

@bot.message_handler(func=lambda message: message.text == '·Додати витрати')
def add_expense(message):
    add_expense_handler(bot, message, main)

@bot.message_handler(func=lambda message: message.text == '·Додати дохід')
def add_income(message):
    add_income_handler(bot, message, main)

@bot.message_handler(func=lambda message: message.text == '·Статистика')
def show_statistics(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    current_month_button = types.KeyboardButton('Поточний місяць')
    previous_month_button = types.KeyboardButton('Попередній місяць')
    last_3_months_button = types.KeyboardButton('Останні 3 місяці')
    back_button = types.KeyboardButton('Назад до головного меню')
    markup.add(current_month_button, previous_month_button,
last_3_months_button, back_button)
    bot.send_message(message.chat.id, 'Оберіть період для отримання
статистики:', reply_markup=markup)

@bot.message_handler(func=lambda message: message.text == 'Поточний місяць')
def current_month_statistics(message):
    send_statistics_current_month(message)
```

```
@bot.message_handler(func=lambda message: message.text == 'Попередній
місяць')
def previous_month_statistics(message):
    send_statistics_previous_month(message)

@bot.message_handler(func=lambda message: message.text == 'Останні 3 місяці')
def last_3_months_statistics(message):
    send_statistics_previous_3_month(message)

@bot.message_handler(func=lambda message: message.text == '·Ліміт')
def limit_menu(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    change_limit_button = types.KeyboardButton('Змінити ліміт')
    disable_limit_button = types.KeyboardButton('Вимкнути ліміт')
    back_button = types.KeyboardButton('Назад до головного меню')
    markup.add(change_limit_button, disable_limit_button, back_button)
    bot.send_message(message.chat.id, 'Меню ліміту', reply_markup=markup)

@bot.message_handler(func=lambda message: message.text == 'Змінити ліміт')
def change_limit(message):
    bot.send_message(message.chat.id, 'Введіть новий ліміт:')
    bot.register_next_step_handler(message, limit.set_limit)

@bot.message_handler(func=lambda message: message.text == 'Вимкнути ліміт')
def disable_limit(message):
    limit.disable_limit(message)

@bot.message_handler(func=lambda message: message.text == 'Назад до головного
меню')
def back_to_main_menu(message):
    markup = main_menu_markup()
    bot.send_message(message.chat.id, 'Головне меню', reply_markup=markup)

@bot.message_handler(func=lambda message: message.text == '·Голосове
керування')
def voice_control_menu(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    back_button = types.KeyboardButton('До головного меню ·')
    markup.add(back_button)
    bot.send_message(message.chat.id,'Ви можете записати голосове
повідомлення, аби додати витрати/надходження або виконати іншу функцію
\n\nПриклад використання можна подивитись в розділі /help',
                     reply_markup=markup)
    @bot.message_handler(content_types=['voice'])
    def addVoice(message):
        voice_control(message)

@bot.message_handler(func=lambda message: message.text == 'До головного меню
·')
def back_to_menu(message):
    markup = main_menu_markup()
    bot.send_message(message.chat.id, 'Головне меню', reply_markup=markup)

@bot.message_handler(commands=['help'])
def help(message):
    help_text = (
        "Привіт! Ось команди, які ти можеш використовувати:\n\n"
        "· *Додати витрати* - додавання витрат до вашого бюджету.\n"
        "· *Додати дохід* - додавання доходів до вашого бюджету.\n"
```

```
        "· *Ліміт* - керування бюджетними лімітами.\n"
        "· *Статистика* - перегляд статистики витрат та доходів.\n"
        "· *Голосове керування* - використовуйте голосові команди для
керування ботом.\n"
        "\n"
        "Крім того, ось деякі підкоманди:\n\n"
        "*Статистика*:\n"
        " - Поточний місяць\n"
        " - Попередній місяць\n"
        " - Останні 3 місяці\n\n"
        "*Ліміт*:\n"
        " - Змінити ліміт\n"
        " - Вимкнути ліміт\n\n"
        "*Голосове керування*:\n"
        "Ви можете використовувати голосові команди для додавання витрат та
доходів.\n"
        "Приклади команд для додавання витрат:\n"
        " - 'Додай витрати 250 грн 50 копійок в категорію продукти'\n"
        " - 'Хочу додати витрати 952 грн в категорію продукти одяг'\n\n"
        "Приклади команд для додавання доходів:\n"
        " - 'Додай надходження 5000 грн'\n"
        " - 'Хочу додати дохід 1000 грн'\n\n"
        "Для повернення до головного меню, використовуйте команду 'Назад до
головного меню'."
    )
    bot.send_message(message.chat.id, help_text, parse_mode='Markdown')

@bot.message_handler(func=lambda message: True)
def handle_text(message):
    bot.send_message(message.chat.id, 'Вибачте, я не розумію, про що йде
мова. Будь ласка, скористайтесь командами, які я знаю.')

bot.polling(none_stop=True)
```

```
#файл expenses.py
from telebot import types
from telebot_calendar import CallbackData, Calendar
import config

calendar = Calendar()
calendar_callback = CallbackData("calendar", "action", "year", "month",
"day")

def add_expense_handler(bot, message, main_function):
    bot.send_message(message.chat.id, 'Оберіть дату витрати:',
reply_markup=calendar.create_calendar(name="calendar_expense"))

    @bot.callback_query_handler(func=lambda call:
call.data.startswith("calendar_expense"))
    def calendar_handler_expense(call):
        data_parts = call.data.split(":")
        action = data_parts[1]
        year = data_parts[2]
        month = data_parts[3]
        day = data_parts[4]
        if action == "DAY":
            date = f"{year}-{month}-{day}"
```

```
            bot.send_message(call.message.chat.id, f'Ви обрали дату: {date}')
            bot.send_message(call.message.chat.id, 'Введіть суму витрати:')
            bot.register_next_step_handler(call.message, lambda msg:
get_amount(msg, date))
        else:
            bot.send_message(call.message.chat.id, 'Будь ласка, оберіть дату
в поточному місяці.')

    def get_amount(message, date):
        try:
            amount = message.text
            if amount is None:
                raise ValueError
            else:
                if ',' in amount:
                    amount = amount.replace(',', '.')
                amount = float(amount)
                amount = round(amount, 2)
                bot.send_message(message.chat.id, 'Виберіть категорію
витрати:', reply_markup=create_categories_keyboard())
                bot.register_next_step_handler(message, lambda msg:
save_expense(msg, date, amount))
        except ValueError:
            bot.send_message(message.chat.id, 'Будь ласка, введіть коректну
суму витрати (наприклад, "155.50"):')
            bot.register_next_step_handler(message, lambda msg:
get_amount(msg, date))

    def create_categories_keyboard():
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        categories = ["Продукти ·", "Одяг ·", "Розваги ·", "Здоров'я ·", "Кафе
та ресторани ·", "Комунальні послуги ·", "Інше"]
        for category in categories:
            markup.add(types.KeyboardButton(category))
        return markup

    def save_expense(message, date, amount):
        if message.text in ["Продукти ·", "Одяг ·", "Розваги ·", "Здоров'я ·",
"Кафе та ресторани ·",
                            "Комунальні послуги ·", "Інше"]:
            user_id = message.chat.id
            category = message.text

            try:
                # Даємо запит в БД для подальшої перевірки по ліміту
                config.cursor.execute("SELECT SUM(amount) FROM expenses WHERE
user_id = %s", (user_id,))
                total_expenses_row = config.cursor.fetchone()
                total_expenses = total_expenses_row[0] if total_expenses_row
is not None else 0

                config.cursor.execute("SELECT user_limit FROM limits WHERE
user_id = %s", (user_id,))
                user_limit_row = config.cursor.fetchone()
                user limit = user limit row[0] if user limit row is not None
else None

                # Вставимо дані про витрати в таблицю
                config.cursor.execute("INSERT INTO expenses (date, amount,
```

```
category, user_id) VALUES ( %s, %s, %s, %s)",
                                        (date, amount, category, user_id))
                config.conn.commit()
                bot.send_message(message.chat.id, 'Дані про витрати успішно
додано!')
                if user_limit is not None:
                    if total_expenses is not None and total_expenses >
user_limit:
                        bot.send_message(message.chat.id,
                                         f'·УВАГА·\nСума витрат за цей
місяць({total_expenses:.2f} грн.) перевищує встановлений ліміт
({user_limit:.2f} грн.).\n\n'
                                         f'*Це повідомлення виключно
інформативне. Ви можете його ігнорувати або збільшити/вимкнути ліміт на цей
місяць ·')
                main_function(message)
            except config.psycopg2.Error as e:
                print("Помилка при додаванні витрат:", e)
                bot.send_message(message.chat.id, 'Сталася помилка. Спробуйте
ще раз.')
        else:
            bot.send_message(message.chat.id, 'Будь ласка, повторіть спробу
додавання витрат.')
            main_function(message)
```

```
#файл voiceExpenses.py
from telebot_calendar import CallbackData, Calendar
import config

calendar = Calendar()
calendar_callback = CallbackData("calendar", "action", "year", "month",
"day")


def add_voice_expense_handler(bot, message, amount_param, category_param):
    global amount, category
    amount = amount_param
    category = category_param
    bot.send_message(message.chat.id, 'Оберіть дату витрати:',

reply_markup=calendar.create_calendar(name="calendar_VoiceExpense"))

    @bot.callback_query_handler(func=lambda call:
call.data.startswith("calendar_VoiceExpense"))
    def calendar_handler_expense(call):
        global amount, category
        print("calendar " + category)
        data_parts = call.data.split(":")
        action = data_parts[1]
        year = data_parts[2]
        month = data_parts[3]
        day = data_parts[4]
        if action == "DAY":
            date = f"{year}-{month}-{day}"
            bot.send_message(call.message.chat.id, f'Ви обрали дату: {date}')
            save_expense(call.message, date, amount, category)
        else:
```

```
            bot.send_message(call.message.chat.id, 'Будь ласка, оберіть дату
в поточному місяці‥')

    def save_expense(message, date, amount, category):
        if category in ["Продукти ·", "Одяг ·", "Розваги ·", "Здоров'я ·", "Кафе
та ресторани ·",
                        "Комунальні послуги ·", "Інше"]:
            user_id = message.chat.id
            try:
                config.cursor.execute("SELECT SUM(amount) FROM expenses WHERE
user_id = %s", (user_id,))
                total_expenses_row = config.cursor.fetchone()
                total_expenses = total_expenses_row[0] if total_expenses_row
is not None else 0

                config.cursor.execute("SELECT user_limit FROM limits WHERE
user_id = %s", (user_id,))
                user_limit_row = config.cursor.fetchone()
                user_limit = user_limit_row[0] if user_limit_row is not None
else None
                config.cursor.execute("INSERT INTO expenses (date, amount,
category, user_id) VALUES ( %s, %s, %s, %s)",
                                      (date, amount, category, user_id))
                config.conn.commit()
                bot.send_message(message.chat.id, 'Дані про витрати успішно
додано!')
                if user_limit is not None:
                    if total_expenses is not None and total_expenses >
user_limit:
                        bot.send_message(message.chat.id,
                                         f'·УВАГА·\nСума витрат за цей
місяць({total_expenses:.2f} грн.) перевищує встановлений ліміт
({user_limit:.2f} грн.).\n\n'
                                         f'*Це повідомлення виключно
інформативне. Ви можете його ігнорувати або збільшити/вимкнути ліміт на цей
місяць ·')
            except config.psycopg2.Error as e:
                print("Помилка при додаванні витрат:", e)
                bot.send_message(message.chat.id, 'Сталася помилка. Спробуйте
ще раз.')
        else:
            bot.send_message(message.chat.id, 'Будь ласка, оберіть категорію
з клавіатури.')
```

```
#файл incomes.py
from telebot_calendar import CallbackData, Calendar
import config

calendar = Calendar()
calendar_callback = CallbackData("calendar", "action", "year", "month",
"day")


def add_income_handler(bot, message, main_function):
    bot.send_message(message.chat.id, 'Оберіть дату надходження:',
reply_markup=calendar.create_calendar(name="calendar_income"))
    @bot.callback_query_handler(func=lambda call:
```

```
call.data.startswith("calendar_income"))
    def calendar_handler_income(call):
        data_parts = call.data.split(":")
        action = data_parts[1]
        year = data_parts[2]
        month = data_parts[3]
        day = data_parts[4]
        if action == "DAY":
            date = f"{year}-{month}-{day}"
            bot.send_message(call.message.chat.id, f'Ви обрали дату: {date}')
            bot.send_message(call.message.chat.id, 'Введіть суму
надходження:')
            bot.register_next_step_handler(call.message, lambda msg:
get_amount(msg, date))
        else:
            bot.send_message(call.message.chat.id, 'Будь ласка, оберіть дату
в поточному місяці…')

    def get_amount(message, date):
        try:
            amount = message.text
            if amount is None:
                raise ValueError
            else:
                if ',' in amount:
                    amount = amount.replace(',', '.')
                amount = float(amount)
                amount = round(amount, 2)
                save_income(message, date, amount)
        except ValueError:
            bot.send_message(message.chat.id, 'Будь ласка, введіть коректну
суму надходження (наприклад, "1150"):')
            bot.register_next_step_handler(message, lambda msg:
get_amount(msg, date))

    def save_income(message, date, amount):
        user_id = message.chat.id
        try:
            config.cursor.execute("INSERT INTO incomes (user_id, amount,
date) VALUES (%s, %s, %s)", (user_id, amount, date))
            config.conn.commit()
            bot.send_message(message.chat.id, 'Дані про надходження успішно
додано!')
            main_function(message)
        except config.psycopg2.Error as e:
            print("Помилка при додаванні надходження:", e)
            bot.send_message(message.chat.id, 'Сталася помилка. Спробуйте ще
раз.')
            main_function(message)
```

```
#файл voiceIncomes.py
from telebot_calendar import CallbackData, Calendar
import config

calendar = Calendar()
calendar_callback = CallbackData("calendar", "action", "year", "month",
"day")
```

```
def add_voice_income_handler(bot, message, amount_param):
    global amount
    amount = amount_param
    bot.send_message(message.chat.id, 'Оберіть дату надходження:',
reply_markup=calendar.create_calendar(name="calendar_VoiceIncome"))
    @bot.callback_query_handler(func=lambda call:
call.data.startswith("calendar_VoiceIncome"))
    def calendar_handler_income(call):
        global amount
        data_parts = call.data.split(":")
        action = data_parts[1]
        year = data_parts[2]
        month = data_parts[3]
        day = data_parts[4]
        if action == "DAY":
            date = f"{year}-{month}-{day}"
            bot.send_message(call.message.chat.id, f'Ви обрали дату: {date}')
            save_income(call.message, amount, date)
        else:
            bot.send_message(call.message.chat.id, 'Будь ласка, оберіть дату
в поточному місяці.')

    def save_income(message, amount, date):
        user_id = message.chat.id
        try:
            config.cursor.execute("INSERT INTO incomes (user_id, amount,
date) VALUES (%s, %s, %s)", (user_id, amount, date))
            config.conn.commit()
            bot.send_message(message.chat.id, 'Дані про надходження успішно
додано!')
        except config.psycopg2.Error as e:
            print("Помилка при додаванні надходження:", e)
            bot.send_message(message.chat.id, 'Сталася помилка. Спробуйте ще
раз.')
```

```
#файл voice_recognize.py
import speech_recognition as sr
import subprocess
import os
from config import bot

r = sr.Recognizer()

def voice_recognize(message):
    file_name = f'{message.voice.file_id}.ogg'
    file_info = bot.get_file(message.voice.file_id)
    downloaded_file = bot.download_file(file_info.file_path)
    with open(file_name, 'wb') as new_file:
        new_file.write(downloaded_file)

    file_name_wav = f'{message.voice.file_id}.wav'
    subprocess.call(['ffmpeg', '-i', file_name, file_name_wav],
stdout=subprocess.PIPE, stderr=subprocess.PIPE)

    with sr.AudioFile(file_name_wav) as source:
```

```
        audio = r.record(source)
    try:
        text = r.recognize_google(audio, language='uk-UA')
        return text
    except sr.UnknownValueError:
        text = "none"
        return text
    finally:
        os.remove(file_name)
        os.remove(file_name_wav)
```

```
#файл voiceControl.py
from statistics.current_month_statistics import send_statistics_current_month
from expenses.voiceExpenses import add_voice_expense_handler
from incomes.voiceIncomes import add_voice_income_handler
from voiceControl.voice_recognize import voice_recognize
from config import bot
import re

def voice_control(message):
    text = voice_recognize(message)
    category_mapping = {
        "продукти": "Продукти ·",
        "їжа": "Продукти ·",
        "взуття": "Одяг ·",
        "одяг": "Одяг ·",
        "розваги": "Розваги ·",
        "здоров'я": "Здоров'я ·",
        "аптека": "Здоров'я ·",
        "кафе": "Кафе та ресторани ·",
        "ресторани": "Кафе та ресторани ·",
        "інше": "Інше"
    }

    amount_pattern = re.compile(r'\b\d+\b(?=\s*(?:грн?|гривень))',
re.IGNORECASE) #пошук цілих сум у гривнях (наприклад, "100 грн" або "100
гривень")
    decimal_amount_pattern =
re.compile(r'\b\d+\b(?=\s*(?:копійки?|копійок|коп|ко))', re.IGNORECASE)
#пошук цілих сум у копійках (наприклад, "50 коп" або "50 копійок")
    category_pattern =
re.compile(r'продукти|їжа|взуття|одяг|розваги|здоров\'я|аптека|кафе|ресторани
|інше', re.IGNORECASE) #пошук категорії витрат
    expense_pattern = re.compile(r'витрати|витрату|розхід', re.IGNORECASE)
#пошук типу операції
    income_pattern = re.compile(r'надходження|дохід|прибуток', re.IGNORECASE)
#пошук типу операції
    statistic_pattern =
re.compile(r'(показати|покажи)\s+(мені\s+)?статистику|стат|ста|звіт|з',
re.IGNORECASE) #пошук запиту статистики

    integer_amounts = [int(match.group()) for match in
amount_pattern.finditer(text)] # Створює список з знайдених цілих сум (у
гривнях) в тексті
    decimal_amounts = [float(match.group()) / 100 for match in
decimal_amount_pattern.finditer(text)] # Створює список з знайдених сум у
копійках в тексті, переведених у гривні
```

```
    amount_match = amount_pattern.search(text)
    category_match = category_pattern.search(text)
    expense_match = expense_pattern.search(text)
    income_match = income_pattern.search(text)
    statistic_match = statistic_pattern.search(text)

    amount = sum(integer_amounts + decimal_amounts) if amount_match else None
    category = category_match.group() if category_match else None

    if category in category_mapping:
        category = category_mapping[category]
    else:
        category = None
    if expense_match:
        if not amount or not category:
            bot.send message(message.chat.id, "Не вдалося розпізнати суму,
категорію або тип операції.")
        else:
            result_message = f"Сума: {amount} грн\nКатегорія: {category}\nТип
операції: Додавання витрати ·"
            bot.send_message(message.chat.id, result_message)
            add_voice_expense_handler(bot, message, amount, category)
    elif income_match:
        if not amount:
            bot.send_message(message.chat.id, "Не вдалося розпізнати суму,
категорію або тип операції.")
        else:
            result_message = f"Сума: {amount} грн\nТип операції: Додавання
надходження"
            bot.send_message(message.chat.id, result_message)
            add_voice_income_handler(bot, message, amount)
    elif statistic_match:
        send_statistics_current_month(message)
    else:
        bot.send_message(message.chat.id, "Не вийшло розпізнати команду ·
Повторіть спробу ")
```

```
#файл current_month_statistics.py
import datetime
import config

def get_month():
    today = datetime.date.today()
    first_day = today.replace(day=1)
    next_month = first_day.replace(month=first_day.month + 1)
    last_day = next_month - datetime.timedelta(days=1)
    return first_day, last_day

def get_total_expenses(user_id):
    first_day, last_day = get_month()
    config.cursor.execute("SELECT SUM(amount) FROM expenses WHERE date
BETWEEN %s AND %s AND user_id = %s", (first_day, last_day, user_id))
    total_expenses = config.cursor.fetchone()[0]
    return total_expenses if total_expenses else 0
```

```
def get_total_incomes(user_id):
    first_day, last_day = get_month()
    config.cursor.execute("SELECT SUM(amount) FROM incomes WHERE date BETWEEN
%s AND %s AND user_id = %s", (first_day, last_day, user_id))
    total_incomes = config.cursor.fetchone()[0]
    return total_incomes if total_incomes else 0


def get_top_categories(user_id):
    first_day, last_day = get_month()
    config.cursor.execute("SELECT category, SUM(amount) AS total_amount FROM
expenses WHERE date BETWEEN %s AND %s AND user_id = %s GROUP BY category
ORDER BY total_amount DESC", (first_day, last_day, user_id))
    top_categories = config.cursor.fetchall()
    return top_categories

def send statistics current month(message):
    user_id = message.chat.id
    total_expenses = get_total_expenses(user_id)
    total_incomes = get_total_incomes(user_id)
    top_categories = get_top_categories(user_id)

    message_text = f"· Статистика за поточний місяць:\n\n" \
                   f"· Сумарні витрати: {total_expenses:.2f} грн.\n" \
                   f"· Сумарні надходження: {total_incomes:.2f} грн.\n\n" \
                   f"· Топ категорій витрат:\n\n"

    for index, (category, amount) in enumerate(top_categories, start=1):
        message_text += f"{index}. {category}: {amount:.2f} грн.\n"

    config.bot.send_message(message.chat.id, message_text)
```

```
#файл previous_month_statistics.py
import datetime
import config

def get_previous_month():
    today = datetime.date.today()
    first_day = today.replace(day=1)
    first_day_of_previous_month = first_day - datetime.timedelta(days=1)
    first_day_of_previous_month = first_day_of_previous_month.replace(day=1)
    last_day_of_previous_month = first_day - datetime.timedelta(days=1)
    return first day of previous month, last day of previous month

def get_total_expenses(user_id):
    first_day, last_day = get_previous_month()
    config.cursor.execute("SELECT SUM(amount) FROM expenses WHERE date
BETWEEN %s AND %s AND user_id = %s",
                          (first_day, last_day, user_id))
    total_expenses = config.cursor.fetchone()[0]
    return total_expenses if total_expenses else 0

def get_total_incomes(user_id):
    first_day, last_day = get_previous_month()
    config.cursor.execute("SELECT SUM(amount) FROM incomes WHERE date BETWEEN
%s AND %s AND user_id = %s",
```

```
                            (first_day, last_day, user_id))
    total_incomes = config.cursor.fetchone()[0]
    return total_incomes if total_incomes else 0

def get_top_categories(user_id):
    first_day, last_day = get_previous_month()
    config.cursor.execute(
        "SELECT category, SUM(amount) AS total_amount FROM expenses WHERE
date BETWEEN %s AND %s AND user_id = %s GROUP BY category ORDER BY
total_amount DESC",
        (first_day, last_day, user_id))
    top_categories = config.cursor.fetchall()
    return top_categories

def send_statistics_previous_month(message):
    user_id = message.chat.id
    total_expenses = get_total_expenses(user_id)
    total_incomes = get_total_incomes(user_id)
    top_categories = get_top_categories(user_id)

    message_text = f"· Статистика за попередній місяць:\n\n" \
                   f"· Сумарні витрати: {total_expenses:.2f} грн.\n" \
                   f"· Сумарні надходження: {total_incomes:.2f} грн.\n\n" \
                   f"· Топ категорій витрат:\n\n"

    for index, (category, amount) in enumerate(top_categories, start=1):
        message_text += f"{index}. {category}: {amount:.2f} грн.\n"

    config.bot.send_message(message.chat.id, message_text)
```

```
#файл last_3_months_statistics.py
import datetime
import config

def get_last_3_month_periods():
    today = datetime.date.today()
    first_day_current_month = today.replace(day=1)

    periods = []
    for i in range(3):
        first_day = (first_day_current_month -
datetime.timedelta(days=1)).replace(day=1)
        last_day = first_day_current_month - datetime.timedelta(days=1)
        periods.append((first_day, last_day))
        first_day_current_month = first_day
    return periods

def get_expenses_for_period(user_id, start_date, end_date):
    config.cursor.execute("SELECT SUM(amount) FROM expenses WHERE date
BETWEEN %s AND %s AND user_id = %s",
                          (start_date, end_date, user_id))
    total_expenses = config.cursor.fetchone()[0]
    return total_expenses if total_expenses else 0

def get_incomes_for_period(user_id, start_date, end_date):
    config.cursor.execute("SELECT SUM(amount) FROM incomes WHERE date BETWEEN
```

```
%s AND %s AND user_id = %s",
                           (start_date, end_date, user_id))
    total_incomes = config.cursor.fetchone()[0]
    return total_incomes if total_incomes else 0

def get_expenses_by_category_for_period(user_id, start_date, end_date):
    config.cursor.execute(
        "SELECT category, SUM(amount) AS total_amount FROM expenses WHERE
date BETWEEN %s AND %s AND user_id = %s GROUP BY category ORDER BY
total_amount DESC",
        (start_date, end_date, user_id))
    top_categories = config.cursor.fetchall()
    return top_categories


def send_statistics_previous_3_month(message):
    user_id = message.chat.id
    periods = get_last_3_month_periods()

    message_text = "· Статистика за останні 3 місяці:\n\n"

    for start_date, end_date in periods:
        #Отримання сумарних витрат, надходжень, та ТОПу категорій витрат.
        total_expenses = get_expenses_for_period(user_id, start_date,
end_date)
        total_incomes = get_incomes_for_period(user_id, start_date, end_date)
        top_categories = get_expenses_by_category_for_period(user_id,
start_date, end_date)

        # Отримання назви місяця та року для поточного періоду
        month_name = start_date.strftime("%B %Y")
        message_text += f"· Статистика за {month_name}:\n"
        message_text += f"· Сумарні витрати: {total_expenses:.2f} грн.\n"
        message_text += f"· Сумарні надходження: {total_incomes:.2f} грн.\n"
        message_text += "· Топ категорій витрат:\n"

        for index, (category, amount) in enumerate(top_categories, start=1):
            message_text += f"{index}. {category}: {amount:.2f} грн.\n"

        message_text += "\n"

    config.bot.send_message(message.chat.id, message_text)
```

```
#файл limit.py
import config

def set_limit(message):
    try:
        new_limit = float(message.text)
        user_id = message.chat.id

        # Перевірка наявності користувача в базі даних
        config.cursor.execute("SELECT * FROM limits WHERE user_id = %s",
(user_id,))
        user_exists = config.cursor.fetchone()

        if user_exists:
```

```
            # Оновлення ліміту для існуючого користувача
            config.cursor.execute("UPDATE limits SET user_limit = %s WHERE
user_id = %s", (new_limit, user_id))
            config.conn.commit()
            config.bot.send_message(message.chat.id, f"Новий ліміт
встановлено: {new_limit}" + " грн.")
        else:
            # Вставка нового користувача з лімітом
            config.cursor.execute("INSERT INTO limits (user_id, user_limit)
VALUES (%s, %s)", (user_id, new_limit))
            config.conn.commit()
            config.bot.send_message(message.chat.id, f"Новий ліміт створено:
{new_limit}" + " грн.")
    except ValueError:
        config.bot.send_message(message.chat.id, 'Будь ласка, введіть
коректну суму ліміту (наприклад, "9500"):')
        config.bot.register_next_step_handler(message, set_limit)

def disable_limit(message):
    try:
        user_id = message.chat.id
        # Перевірка наявності користувача в базі даних
        config.cursor.execute("SELECT * FROM limits WHERE user_id = %s",
(user_id,))
        user_exists = config.cursor.fetchone()

        if user_exists:
            config.cursor.execute("DELETE FROM limits WHERE user_id = %s",
(user_id,))
            config.conn.commit()
            config.bot.send_message(message.chat.id, f"Ліміт вимкнено.")
        else:
            config.bot.send_message(message.chat.id, f"У вас ліміт не
встановлений, тому нічого і вимикати ·")
    except ValueError:
        config.bot.send_message(message.chat.id, 'Щось пішло не так,
спробуйте ще раз.')
        config.bot.register_next_step_handler(message, disable_limit)
```

```
#файл config.py
import telebot
import psycopg2

bot = telebot.TeleBot('7638558516:AoHneoACslwHsjortEj2dcdXZjJRIR_Kd-VI')
#токен боту в КРБ було змінено в цілях безпеки

try:
    conn = psycopg2.connect(
        dbname="WalletDB",
        user="postgres",
        password="admin",
        host="localhost",
        port=5432
    )
    cursor = conn.cursor()
    print("З'єднання з базою даних встановлено")
```

```
except psycopg2.Error as e:
    print("Помилка при підключенні до бази даних:", e)
```