МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

Telegram-бот «Аналітика особистих фінансів з рекомендаціями»

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Донченко Назарій

«___»_____ 2026 р.

Керівник канд. техн. наук, доцент _____ Гожий Віктор

«___»_____ 2026 р.

Миколаїв – 2026

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп'ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп'ютерні науки
Освітня програма	Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____Євген СІДЕНКО

«____» _____2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Донченка Назарія Васильовича
((прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Telegram-бот «аналітика особистих фінансів з рекомендаціями»».

Керівник роботи: Гожий Віктор Олександрович, ст. викладач кафедри ІІС, канд. техн. наук

Затверджена наказом ЧНУ ім. Петра Могили від «25» грудень 2025 р. № 353.

2. Строк представлення кваліфікаційної роботи «3» травень 2026 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: експертні оцінки підходів до обліку, аналізу та планування особистих фінансів; наявні технології та архітектурні підходи для розробки Telegram-ботів і фінансових інформаційних систем; вимоги до збереження, обробки та аналізу даних про доходи, витрати й фінансові категорії користувача; методи формування персоналізованих рекомендацій щодо оптимізації бюджету,

контролю витрат і покращення фінансової дисципліни.Очікуваний результат: Telegram-бот для аналітики особистих фінансів з рекомендаціями, який дозволяє користувачу додавати доходи та витрати, розподіляти операції за категоріями, переглядати баланс і статистику за обраний період, аналізувати структуру витрат та отримувати рекомендації щодо ефективнішого управління особистим бюджетом

4. Перелік питань, що підлягають розробці:

- аналіз наявних Telegram-ботів та застосунків для обліку й аналітики особистих фінансів;
- дослідження підходів до збереження, обробки та аналізу даних про доходи й витрати користувача.
- архітектурні підходи, патерни проєктування та технології, що використовуються для створення Telegram-бота аналітики особистих фінансів.
- розробка Telegram-бота для ведення обліку доходів і витрат, формування статистики та надання фінансових рекомендацій

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Віктор ГОЖИЙ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Назарій ДОНЧЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «25» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Telegram-бот «Аналітика особистих фінансів з рекомендаціями »

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	21.12.2025	24.12.2025	Виконано
2	Аналіз предметної області та постановка задачі	25.12.2025	30.01.2026	Виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи	31.01.2026	01.03.2026	Виконано
4	Огляд існуючих альтернатив до розроблюваної системи	02.03.2026	01.04.2026	Виконано
5	Реалізація обраних технологій з аналізом отриманих результатів	02.04.2026	24.05.2026	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	25.05.2026	25.05.2026	Виконано
7	Корегування роботи за результатами попереднього захисту	26.05.2026	04.06.2026	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	05.06.2026	05.06.2026	Виконано
9	Доробка та остаточне оформлення КР	06.06.2026	14.06.2026	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.06.2026	19.06.2026	Виконано

Керівник роботи _____ Віктор Гожий
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Назарій Васильович
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2026 р.

АНОТАЦІЯ
до кваліфікаційної роботи
здобувача групи 401з ЧНУ ім. Петра Могили

Донченка Назарія Васильовича

на тему: **«TELEGRAM-БОТ «АНАЛІТИКА ОСОБИСТИХ ФІНАНСІВ З РЕКОМЕНДАЦІЯМИ»»**

Кваліфікаційна робота присвячена розробці Telegram-бота для аналізу особистих фінансів та формування рекомендацій щодо управління бюджетом. Актуальність теми зумовлена тим, що багато користувачів потребують простого й доступного інструменту для щоденного контролю доходів і витрат без необхідності встановлення складних фінансових застосунків.

Метою роботи є створення telegram-бота, який дозволяє користувачу фіксувати свої фінансові операції, переглядати загальний стан свого бюджету, аналізувати витрати за категоріями та отримувати рекомендації щодо покращення фінансового стану.

У роботі розглянуто особливості ведення особистого бюджету. Під час проєктування було визначено основні сценарії взаємодії користувача з ботом, структуру збереження даних, логіку обробки фінансових операцій і принципи формування рекомендацій.

Розроблений Telegram-бот забезпечує можливість додавання доходів і витрат, розподілу операцій за категоріями, перегляду балансу, отримання статистики за певний період та аналізу найбільш витратних напрямів. На основі введених даних система формує рекомендації, які допомагають користувачу краще контролювати власні фінанси та приймати більш обґрунтовані рішення щодо витрат.

Практичне значення роботи полягає у створенні доступного інструменту для персонального фінансового контролю, який може бути використаний широким колом користувачів для щоденного планування бюджету.

Об’єкт роботи — процес обліку та аналізу особистих фінансів користувача за допомогою Telegram-бота.

Предмет роботи — методи, алгоритми та програмні засоби для збереження, обробки, аналізу фінансових даних і формування рекомендацій.

Мета роботи — розробка Telegram-бота для аналітики особистих фінансів з функціями обліку доходів і витрат, побудови статистики та надання рекомендацій щодо оптимізації бюджету.

Кваліфікаційна робота містить 96 сторінок, 31 рисунків, 8 таблиць, 30 використаних джерел та 1 додаток.

Ключові слова: Telegram-бот, особисті фінанси, бюджет, доходи, витрати, фінансова аналітика, рекомендації, автоматизація.

ABSTRACT

to the qualification work by the student of the group 401z of Petro Mohyla Black Sea National University

Donchenko Nazarii

on the topic: **«TELEGRAM-BOT «PERSONAL FINANCE ANALYTICS WITH RECOMMENDATIONS»»**

The qualification work is devoted to the development of a Telegram bot for analyzing personal finances and generating recommendations for budget management. The relevance of the topic is determined by the fact that many users need a simple and accessible tool for daily control of income and expenses without the need to install complex financial applications.

The purpose of the work is to create a Telegram bot that allows the user to record financial transactions, view the general state of their budget, analyze expenses by categories, and receive recommendations for improving their financial situation.

The work examines the features of personal budget management. During the design stage, the main scenarios of user interaction with the bot, the data storage structure, the logic of processing financial transactions, and the principles of generating recommendations were defined.

The developed Telegram bot provides the ability to add income and expenses, distribute transactions by categories, view the balance, receive statistics for a selected period, and analyze the most significant spending areas. Based on the entered data, the system generates recommendations that help the user better control their personal finances and make more informed spending decisions.

The practical significance of the work lies in the creation of an accessible tool for personal financial control, which can be used by a wide range of users for daily budget planning.

Object of the work — the process of recording and analyzing the user's personal finances using a Telegram bot.

Subject of the work — methods, algorithms, and software tools for storing,

processing, analyzing financial data, and generating recommendations.

Purpose of the work — to develop a Telegram bot for personal finance analytics with functions for recording income and expenses, building statistics, and providing recommendations for budget optimization.

The qualification work contains 96 pages, 31 figures, 8 tables, 30 references, and 1 appendices.

Keywords: Telegram bot, personal finance, budget, income, expenses, financial analytics, recommendations, automation.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

ОС – операційна система

ПЗ – програмне забезпечення

СУБД – система управління базами даних

API – Application Programming Interface

HTTP – HyperText Transfer Protocol

iOS – мобільна операційна система компанії Apple

SQL – Structured Query Language

SQLite – вбудована реляційна система управління базами даних

URL – Uniform Resource Locator

ВСТУП

В умовах економічної нестабільності, спричиненої воєнним станом, інфляційними процесами та змінами рівня доходів населення, питання раціонального управління особистими фінансами набуває особливого значення. Українцям стає важливіше контролювати щоденні витрати, планувати бюджет, аналізувати фінанси та своєчасно виявляти необов'язкові витрати.

Одним із зручних способів вирішення цього є використання Telegram-бота, який забезпечує швидку взаємодію з користувачем у зручному для нього середовищі. На відміну від складних фінансових застосунків, бот не потребує окремого встановлення, має простий інтерфейс і дозволяє швидко додавати доходи та витрати одразу після здійснення фінансової операції. Завдяки автоматичній обробці введених даних, користувач може отримувати статистику, аналізувати витрати і бачити рекомендації використання власного бюджету.

Основними завданнями цієї кваліфікаційної роботи бакалавра є:

– проаналізувати сучасні рішення для аналітики особистих фінансів;

– визначити вимоги до Telegram-бота для роботи з фінансовими даними користувача;

– обґрунтувати вибір технологій, методів та засобів розробки програмного продукту;

– реалізувати та протестувати Telegram-бота для обліку доходів, витрат, формування статистики й рекомендацій.

Об'єкт роботи – процес розробки Telegram-бота для аналітики особистих фінансів.

Предмет роботи – методи, алгоритми та технології обліку, збереження, обробки й аналізу фінансів користувача, а також формування рекомендацій щодо управління особистим бюджетом.

Метою даної роботи є розробка Telegram-бота для аналітики особистих фінансів.

1 ТЕОРЕТИЧНІ ОСНОВИ ОБЛІКУ ТА АНАЛІЗУ ОСОБИСТИХ ФІНАНСІВ

1.1 Характеристика предметної області обліку особистих фінансів

Telegram-бот для аналітики особистих фінансів призначений для автоматизації процесу фіксації та збереження фінансових операцій користувача. Бот дає змогу вести облік доходів і витрат, контролювати поточний баланс, аналізувати структуру витрат за категоріями та отримувати рекомендації щодо раціонального управління особистим бюджетом.

Особисті фінанси охоплюють усі грошові надходження та витрати користувача, а також процеси планування, контролю й аналізу використання коштів. Регулярний облік фінансових операцій дозволяє краще розуміти, на які потреби витрачаються гроші, визначати найбільш витратні категорії та своєчасно коригувати фінансову поведінку.

У межах даної роботи система розглядається на розробці Telegram-бота як програмного засобу для обліку та аналізу особистих фінансів. Такий формат дозволяє користувачу швидко додавати фінансові операції, переглядати статистику та отримувати рекомендації без використання складного інтерфейсу.

1.2 Система аналізу особистих фінансів користувача

Система аналізу особистих фінансів користувача призначена для обробки введених фінансових даних та формування зрозумілої інформації про стан особистого бюджету. Основою роботи такої системи є отримання даних про доходи й витрати користувача, їх збереження, категоризація, подальший аналіз та надання рекомендацій щодо більш раціонального використання коштів

У межах даної роботи систему аналізу особистих фінансів можна поділити на такі основні етапи:

– взаємодія користувача з Telegram-ботом;
– введення даних про доходи та витрати;

– збереження фінансових операцій у базі даних;
– розподіл витрат за категоріями;
– обробка та аналіз введених даних;
– відображення статистики для користувача;
– формування рекомендацій щодо управління особистим бюджетом.

Користувач взаємодіє з ботом через команди та кнопки Telegram-інтерфейсу. Після запуску бота користувач може додавати фінансові операції, вказуючи тип операції, суму, категорію та за потреби короткий опис. Такі дані зберігаються в базі даних і надалі використовуються для побудови статистики та аналізу бюджету.

Після накопичення фінансових операцій система виконує їх обробку. Доходи та витрати можуть аналізуватися за певний період, наприклад за день, тиждень або місяць. Окрему увагу приділено розподілу витрат за категоріями, оскільки саме це дозволяє користувачу зрозуміти, на які напрями витрачається найбільше коштів.

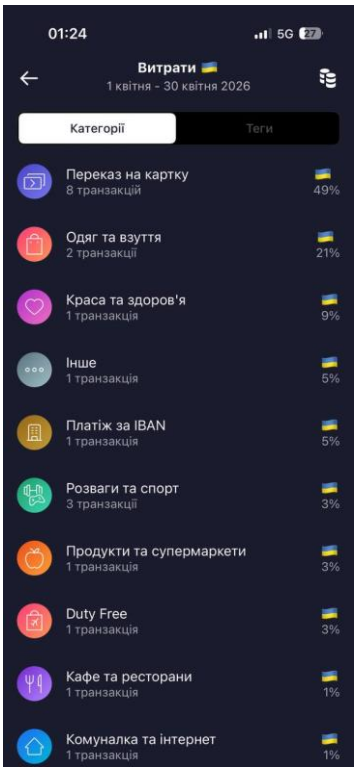
На основі оброблених даних Telegram-бот формує статистику, зокрема загальну суму доходів, суму витрат, поточний баланс та найбільш витратні категорії. Після аналізу система може надавати користувачу рекомендації, наприклад звернути увагу на надмірні витрати в певній категорії, скоротити необов’язкові покупки або більш рівномірно розподіляти бюджет протягом місяця.

1.3 Огляд сучасних цифрових рішень для контролю особистих фінансів

На сьогодні існує багато цифрових рішень, які допомагають користувачам контролювати особисті фінанси, вести облік доходів і витрат, аналізувати фінансові операції та планувати бюджет. До таких рішень належать мобільні застосунки, банківські сервіси, вебплатформи та Telegram-боти. Кожен із цих варіантів має власні переваги та обмеження, які необхідно враховувати під час розробки нового програмного продукту.

– «Monobank» – український банківський застосунок, який надає користувачу можливість керувати картками, переглядати історію фінансових

операцій, здійснювати платежі та контролювати власні витрати. У застосунку користувач може бачити інформацію про здійснені покупки, перекази та інші операції, що дозволяє швидко оцінити фінансову активність за певний період. Також у monobank передбачені інструменти для роботи з фінансовою історією (рис1.1), зокрема позначення витрат, аналітика витрат і зручне групування операцій.

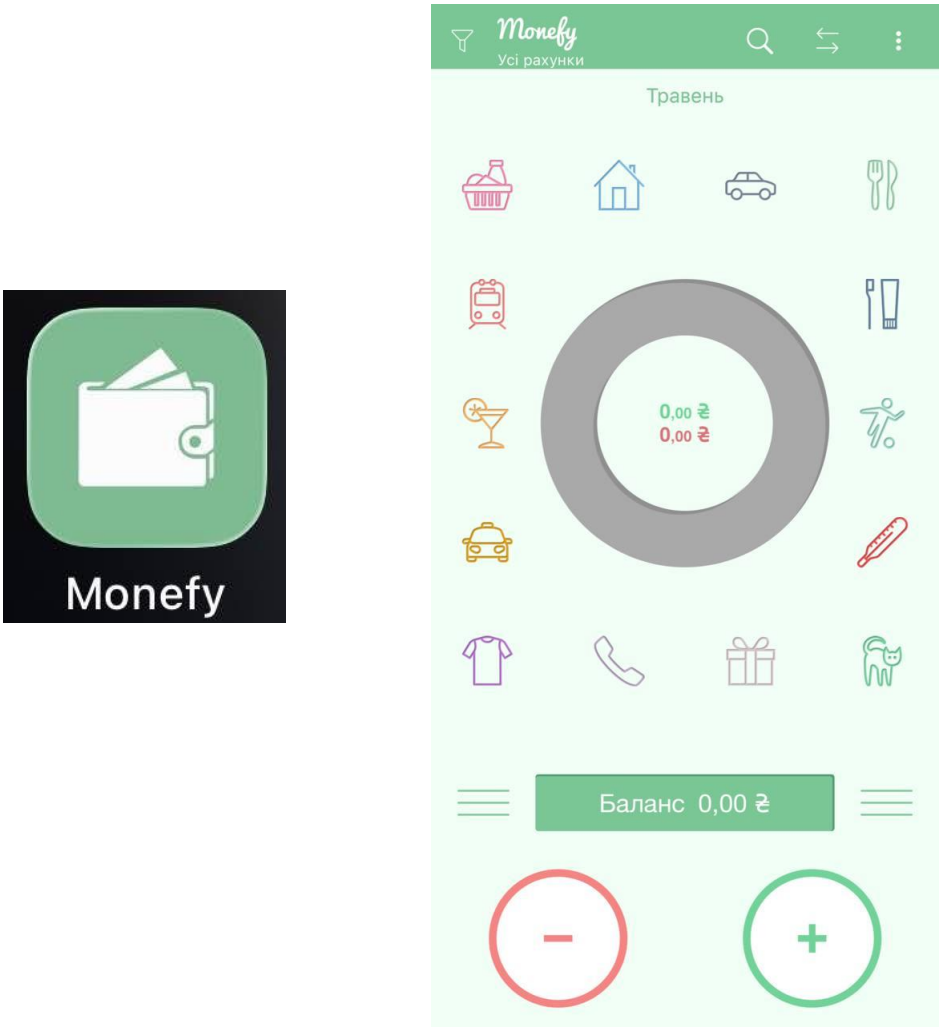


а) б)

Рисунок 1.1 – Логотип застосунку Monobank (а) та інтерфейс застосунку (б)

– «Monefy» – мобільний застосунок для обліку особистих фінансів, який орієнтований на швидке ручне введення доходів і витрат. Користувач може додавати фінансові операції, обирати відповідні категорії, переглядати поточний баланс та аналізувати структуру витрат за певний період. Інтерфейс застосунку побудований таким чином, щоб користувач міг швидко зафіксувати витрату або дохід без складних налаштувань. Однією з основних особливостей Monefy є візуальне відображення витрат за категоріями(рис. 1.2.). Застосунок показує, на які

напрями користувач витрачає найбільше коштів, що допомагає краще контролювати особистий бюджет. Перевагою такого рішення є простота використання та зрозумілий інтерфейс. Водночас недоліком можна вважати те, що більшість операцій користувач вводить вручну, а можливості формування персоналізованих рекомендацій є обмеженими.

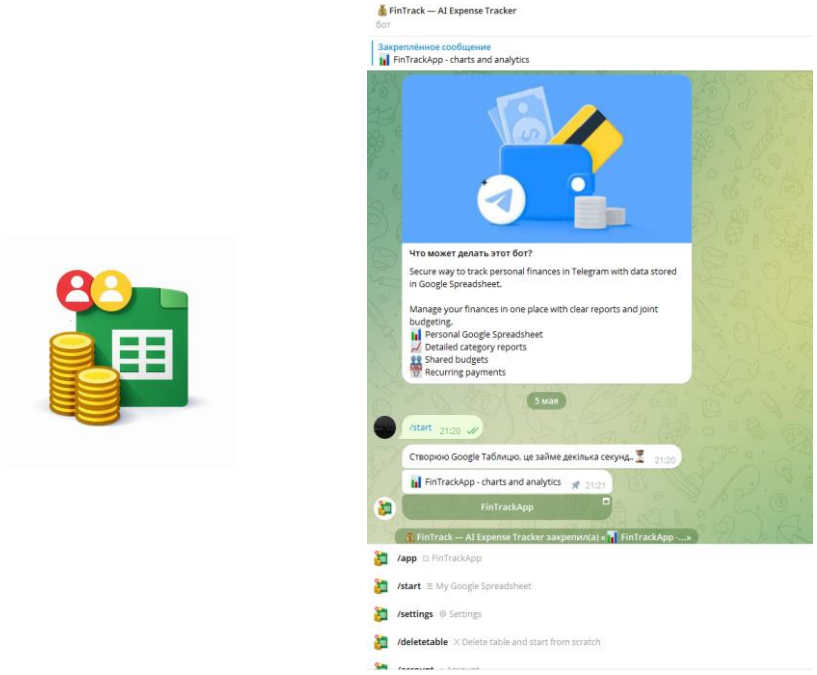


а) б)

Рисунок 1.2 – Логотип застосунку Monefy (а) та інтерфейс застосунку (б)

– «PersonalFinTrackerBot» – Український telegram-бот для ведення обліку доходів і витрат користувача. Його основна ідея полягає у швидкому записі фінансових операцій безпосередньо через Telegram з подальшим збереженням даних у Google Таблицях. Такий підхід є зручним для користувачів, які не хочуть встановлювати окремі застосунки для контролю бюджету, але потребують

простого інструменту для регулярного обліку фінансів рис(1.3).



а) б)

Рисунок 1.3 – Логотип бота PersonalFinTrackerBot (а) та інтерфейс застосунку (б)

1.4 Порівняння цифрових рішень для контролю особистих фінансів

Для більш детального аналізу цифрових рішень було проведено їх порівняння за основними характеристиками у таблиці. До уваги бралися тип програмного рішення, платформа використання, основні функціональні можливості, переваги, недоліки та можливість застосування окремих підходів під час розробки Telegram-бота для аналітики особистих фінансів.

Для об’єктивного порівняння треба використати наступні критерії:

– характеристики програмного рішення;
– платформа, на якій працює система;
– основний функціонал;
– переваги використання;
– недоліки;
– можливість застосування окремих рішень у межах розробки Telegram-бота для аналітики особистих фінансів.

Таблиця 1.1 – Порівняння цифрових рішень для контролю особистих фінансів

Характеристика /назва застосунку	**Monobank**	**Monefy**	**PersonalFinTrackerBot**
Тип рішення	Банківський застосунок	Мобільний застосунок для обліку фінансів	Telegram-бот
Платформа	IOS, Android	IOS, Android	Telegram
Функціонал	Перегляд історії операцій, контроль витрат, банківські платежі, базова аналітика	Ручне додавання доходів і витрат, категорії, баланс, діаграма витрат	Облік доходів і витрат через месенджер, збереження даних, робота з фінансовими операціями
Переваги	Автоматичне відображення банківських операцій, зручний інтерфейс, швидкий доступ до рахунків	Простий інтерфейс, швидке введення операцій, відображення витрат	Не потребує встановлення окремого застосунку, зручний формат взаємодії, доступність у Telegram
Недоліки	Аналітика обмежена операціями конкретного банку	Данні потрібно вводити вручну, обмежені рекомендації, потребує встановлення додатку	Функціональність залежить від реалізації бота, данні потрібно вводити вручну , можливості аналітики можуть бути обмеженими

1.5 Постановка задачі та визначення функціональних можливостей системи

Для розробки Telegram-бота для аналітики особистих фінансів з рекомендаціями система має забезпечувати облік доходів та витрат , збереження

фінансових операцій, формування статистики, рекомендацій та нагадувань для користувача.

Telegram-бот повинен бути простив у використанні та дозволяти швидко додавати фінансові зміни , переглядати історію операцій й аналізувати витрати за категоріями.

На схемі (рис. 1.4) зображено як працює бот.

Рисунок 1.4 – Загальна логіка роботи системи

1.6 Планування технологій для реалізації Telegram-бота

Мовою програмування для створення бота обрано Python. Такий вибір зумовлений зрозумілим і лаконічним синтаксисом мови, широкою екосистемою готових бібліотек та її поширеністю саме у задачах розробки чат-ботів, автоматизації та обробки даних. Python дає змогу швидко описувати програмну логіку, опрацьовувати повідомлення користувача й виконувати обчислення, потрібні для побудови статистики та формування порад..

Звертання до Telegram Bot API планується реалізувати засобами бібліотеки aiogram. Вона дозволяє опрацьовувати команди, текстові повідомлення та натискання кнопок, а також зручно вибудовувати діалоговий сценарій між

людиною і ботом. Застосування цієї бібліотеки знімає частину рутинної роботи й дає змогу сконцентруватися на головній функціональності системи.

Як сховище фінансових операцій передбачено використати SQLite. Перевага цього рішення в тому, що воно не вимагає окремого серверного розгортання й налаштування, але водночас дозволяє надійно тримати дані про користувачів, їхні доходи й витрати, категорії та фінансові цілі в одному файлі бази.

Окремо на етапі підготовки слід продумати схему бази даних так, щоб виключити повторюване зберігання однакової інформації та спростити подальшу обробку операцій. До ключових сутностей системи належать користувачі, фінансові операції, категорії, цілі накопичення та нагадування.

Роль робочого середовища відведено редактору Visual Studio Code. Він зручний у повсякденній роботі з кодом, дозволяє підключати потрібні розширення, працювати з віртуальним оточенням Python і тримати під контролем загальну структуру проєкту.

1.7 Вимоги до програмної реалізації

Маючи визначений набір технологій, можна окреслити ключові вимоги до програмної частини бота. Система має забезпечувати стабільну взаємодію з користувачем у Telegram, правильно опрацьовувати введені значення, зберігати операції у базі та подавати підсумки аналізу у вигляді, зрозумілому для людини.аналізу у зрозумілому для користувача вигляді.

До основного функціоналу бота належать: внесення доходів і витрат, призначення категорії операції, перегляд переліку попередніх записів, побудова підсумків за тиждень або місяць, розкладання витрат за напрямами та видача порад щодо розумнішого розпорядження бюджетом.

Слід також врахувати модуль накопичувальних цілей. Користувач має змогу задати назву цілі, потрібну суму та орієнтовний строк її досягнення, після чого система обчислює, скільки коштів варто відкладати щодня, щотижня чи щомісяця..

Передбачено й розсилання нагадувань і підбадьорливих повідомлень. Вони

допомагають користувачеві не забувати фіксувати витрати, стримувати зайві покупки та скеровувати зекономлене на пріоритетніші завдання.

Інтерфейс має лишатися максимально простим і прозорим для користувача. Основні дії варто виконувати через кнопки чи команди, щоб додавання операцій, перегляд статистики й отримання порад відбувалися без зайвих налаштувань.

Нижче в таблиці зведено основні можливості, які слід закласти в бота. Перелік охоплює і базові дії, як-от облік доходів та витрат, і допоміжні функції, пов’язані зі статистикою, накопичувальними цілями, порадами й нагадуваннями. Така зведена картина допомагає чіткіше уявити загальну логіку системи та закласти підґрунтя для її реалізації.

Завдяки цьому вже на стадії планування стає зрозуміло, які функції є опорними для роботи бота, а які виконують допоміжну роль. Це водночас полегшує подальшу розробку, бо кожна функція отримує чітке призначення й передбачуваний результат.

Основні функціональні можливості Telegram-бота реалізовано в таблиці 1.2.

Таблиця 1.2 – Основні функціональні можливості Telegram-бота

Вимога	**Опис**
Облік доходів	Користувач може додавати інформацію про отримані кошти
Облік витрат	Користувач може додавати витрати із зазначенням суми та категорії
Збереження даних	Усі фінансові операції зберігаються в базі даних
Перегляд історії	Користувач може переглядати операції за день, тиждень або місяць
Формування статистики	Бот показує суму доходів, витрат, баланс та витрати за категоріями

Вимога	Опис
Рекомендації	Система формує поради на основі аналізу фінансових операцій
Фінансові цілі	Користувач може створити ціль накопичення та отримати розрахунок необхідної суми відкладення
Нагадування	У боті користувач бачить нагадування

Сформульований перелік вимог окреслює напрями подальшої роботи над ботом. Система має органічно поєднувати облік операцій, зберігання даних, аналіз витрат, формування порад, ведення накопичувальних цілей та надсилання нагадувань користувачеві.

Висновки до розділу 1

У межах розділу проаналізовано специфіку ведення обліку та аналізу особистих фінансів. Виокремлено опорні складники фінансового контролю: фіксацію доходів і витрат, збереження операцій, поділ витрат за категоріями, побудову статистики та надання користувачеві порад.

Здійснено й огляд наявних цифрових продуктів для контролю особистих фінансів, зокрема Monobank, Monefy та PersonalFinTrackerBot. Зіставлення дало змогу побачити сильні та слабкі сторони кожного з них: Monobank вигідний для роботи з банківськими операціями, Monefy пропонує простий ручний облік витрат, а PersonalFinTrackerBot показує переваги Telegram як середовища спілкування з користувачем.Monobank є зручним для роботи з банківськими операціями, Monefy забезпечує простий ручний облік витрат, а PersonalFinTrackerBot демонструє переваги використання Telegram як середовища для взаємодії з користувачем.

У контексті обраної теми саме Telegram-бот виступає основним програмним інструментом аналітики особистих фінансів із порадами. Такий формат уможливлює швидке внесення операцій, збереження даних, аналіз витрат, побудову статистики та видачу рекомендацій.

Серед опорних можливостей майбутньої системи визначено додавання доходів і витрат, розподіл операцій за категоріями, перегляд історії за заданий проміжок, побудову статистики, роботу з накопичувальними цілями та надсилання нагадувань. Проведений аналіз дає підстави стверджувати, що запланований функціонал відповідає темі роботи й дозволяє створити бота для аналітики особистих фінансів із рекомендаціями. Сформульовані висновки стали підґрунтям для подальшого проєктування системи й окреслили коло завдань, що потребували розв’язання в наступному розділі.

2 ПРОЄКТУВАННЯ TELEGRAM-БОТА ДЛЯ АНАЛІТИКИ ОСОБИСТИХ ФІНАНСІВ

2.1 Проєктування логіки взаємодії користувача з Telegram-ботом

Етап проєктування є визначальним для всієї розробки, адже саме тут закладаються загальна логіка системи, її складники та зв’язки між ними. Для бота аналітики особистих фінансів потрібна така організація, що дала б користувачеві змогу оперативно вносити операції, дивитися статистику, отримувати поради й вести накопичувальні цілі.

Сеанс роботи стартує з команди /start, у відповідь на яку бот виводить вітання та головне меню. У меню доцільно винести головні кнопки доступу до функцій: внесення доходу, внесення витрати, перегляд статистики, історію операцій, накопичувальні цілі та поради..

Базовий сценарій занесення операції має лишатися коротким і покроковим. Спершу користувач вказує тип запису — дохід чи витрату, далі бот послідовно запитує суму, категорію та, за бажанням, короткий коментар. Після підтвердження дані потрапляють до бази й стають доступними для подальшого аналізу.

Щоб переглянути статистику, користувач задає період, що його цікавить: день, тиждень, поточний чи попередній місяць. Після цього бот опрацьовує збережені операції й виводить зведення: суму надходжень, суму витрат, баланс та категорії з найбільшими тратами..

Окремою гілкою сценарію (рис 2.1) є ведення накопичувальних цілей. Користувач створює ціль, зазначаючи потрібну суму та бажаний строк, а бот на основі цих даних обчислює орієнтовний розмір регулярного відкладення — щодня, щотижня чи щомісяця.

Отже, взаємодію з ботом варто будувати за принципом простих послідовних кроків. Такий підхід дає користувачеві змогу швидко вводити фінансові дані та одразу бачити результат аналізу, не заглиблюючись у складний

інтерфейс.інтерфейсу.

Особливе місце в структурі посідає аналітичний модуль. Він опрацьовує збережені операції й обчислює підсумкові величини: сукупні доходи, витрати, баланс, розподіл за категоріями та результати за обраний період. Спираючись на ці дані, користувач оцінює власний фінансовий стан і визначає головні напрями витрат.

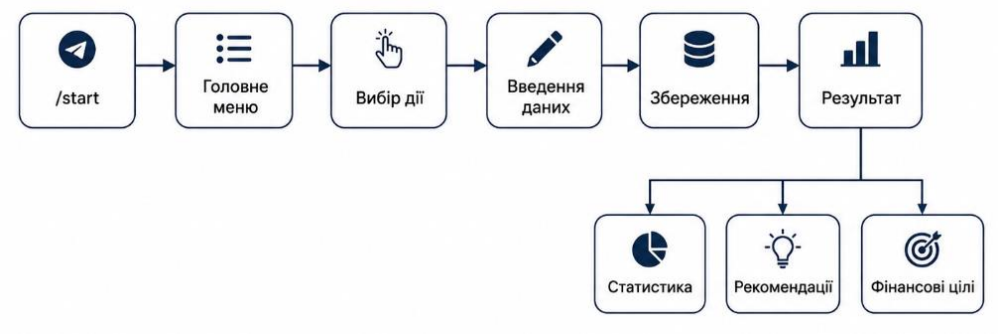


Рисунок 2.1 – Сценарій взаємодії користувача з Telegram-ботом

2.2 Проєктування функціональних модулів Telegram-бота

Систему бота аналітики особистих фінансів доцільно розбити на окремі функціональні складники. Такий поділ упорядковує логіку роботи, спрощує подальшу розробку й полегшує супровід продукту. Кожен складник відповідає за свою ділянку функціональності, проте повноцінна робота досягається завдяки їхній спільній взаємодії.

На рисунку 2.2 наведено основні функціональні модулі Telegram-бота

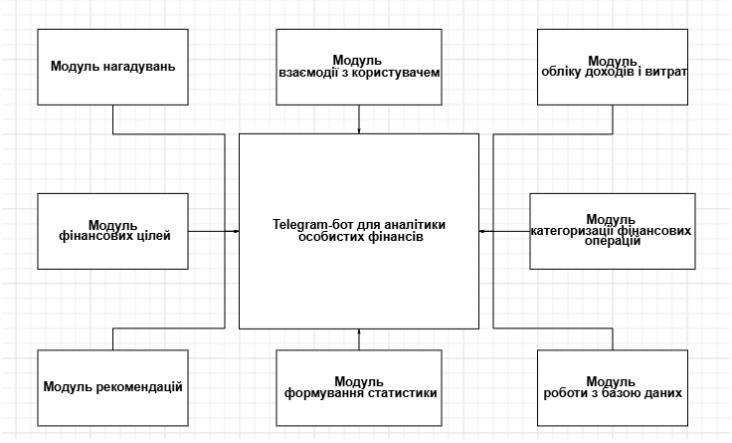


Рисунок 2.2 – Функціональні модулі Telegram-бота

Складник взаємодії з користувачем приймає команди, повідомлення й натискання кнопок у Telegram. Через нього людина запускає бота, відкриває головне меню, обирає потрібну дію та отримує відповіді системи. Головна вимога до цього складника — простота й зрозумілість, адже базові операції користувач має виконувати без зайвих ускладнень.

Складник обліку доходів і витрат відповідає за внесення операцій. Додаючи запис, користувач зазначає його тип, суму, дату, категорію та за потреби короткий опис. Цей складник є одним із центральних, бо саме на основі введених сум згодом будуються статистика, аналіз і поради.

Складник категоризації служить для розподілу витрат і надходжень за відповідними напрямами. Так, витрати можуть належати до категорій «їжа», «транспорт», «кава», «розваги», «покупки», «донати» чи «інше». Категоризація допомагає користувачеві ясніше бачити структуру трат і виявляти, куди йде найбільше коштів.

Складник роботи з базою даних опікується збереженням, пошуком, оновленням та вибіркою фінансової інформації. У базі мають зберігатися відомості про користувачів, операції, категорії, накопичувальні цілі й налаштування нагадувань. Це дозволяє переглядати історію за різні періоди та отримувати актуальну аналітику.

Складник статистики опрацьовує збережені операції. Він обчислює сукупні доходи, витрати, поточний баланс, розподіл за категоріями та результати за період. Зведення можна формувати за день, тиждень, поточний або попередній місяць, що допомагає користувачеві стежити за змінами в бюджеті.

Складник порад формує підказки на основі аналізу фінансових даних. Наприклад, коли витрати в певній категорії стають надмірними, бот звертає на це увагу й радить контролювати їх ретельніше. Якщо ж у користувача є позитивний залишок, система може запропонувати спрямувати частину коштів на

накопичувальну ціль чи інші корисні потреби.аналізу фінансових даних користувача. Наприклад, якщо витрати в певній категорії є надмірними, бот може повідомити про це користувача та запропонувати уважніше контролювати такі витрати. Якщо користувач має позитивний баланс або економію, система може порадити спрямувати частину коштів на фінансову ціль або інші корисні потреби.

Складник накопичувальних цілей дає змогу заводити цілі заощаджень. Для цього користувач указує назву, потрібну суму та бажану дату досягнення, а бот обчислює, скільки відкладати щодня, щотижня чи щомісяця. Такий складник допомагає не лише стежити за поточними витратами, а й планувати майбутні фінансові кроки.

Складник нагадувань періодично надсилає користувачеві повідомлення. Вони можуть нагадувати про потребу внести денні витрати, переглянути статистику, відкласти кошти на ціль чи стримати необов’язкові покупки. Наявність нагадувань заохочує до регулярного користування ботом і формування фінансової дисципліни.

Для кращого представлення функціональних модулів Telegram-бота їх можна узагальнити у таблиці 2.1.

Модуль	**Призначення**
Модуль взаємодії з користувачем	Обробка команд, повідомлень і кнопок Telegram
Модуль обліку доходів і витрат	Додавання та обробка фінансових операцій
Модуль категоризації	Розподіл операцій за категоріями
Модуль роботи з базою даних	Збереження та отримання фінансової інформації
Модуль статистики	Розрахунок доходів, витрат, балансу та показників за період

Модуль	Призначення
Модуль рекомендацій	Формування порад на основі аналізу витрат
Модуль фінансових цілей	Створення цілей накопичення та розрахунок суми відкладення
Модуль нагадувань	Надсилання періодичних повідомлень користувачу

2.3 Проєктування бази даних

Щоб бот працював коректно, потрібно організувати зберігання даних користувача. База акумулює відомості про операції, категорії витрат, накопичувальні цілі та нагадування, завдяки чому людина може переглядати історію доходів і витрат, отримувати статистику за період та аналізувати власну фінансову поведінку.

У межах проєкту доцільно спертися на реляційну модель бази, оскільки фінансові операції мають виразні зв’язки з користувачами, категоріями та часовими проміжками. Такий підхід зручно зберігає дані, дозволяє шукати операції за датою, категорією чи типом і будувати статистичні показники..

Опорними сутностями бази є користувачі, операції, категорії, накопичувальні цілі та нагадування. Кожна виконує власну роль: дані користувача потрібні для ідентифікації в Telegram, операції становлять основу обліку доходів і витрат, категорії групують витрати, цілі застосовуються для планування заощаджень, а нагадування — для періодичних повідомлень.

Таблиця 2.2 – Призначення основних секторів

Назва таблиці	Призначення	Основні поля
users	Збереження даних про користувачів Telegram-бота	id, telegram_id, username, created_at

Назва таблиці	Призначення	Основні поля
categories	Збереження категорій доходів і витрат	id, name, type
transactions	Збереження фінансових операцій користувача	id, user_id, category_id, type, amount, description, created_at
goals	Збереження фінансових цілей користувача	id, user_id, title, target_amount, current_amount, deadline
reminders	Збереження налаштувань нагадувань	id, user_id, text, remind_time, is_active

Таблиця users тримає основні відомості про користувачів бота. Кожен ідентифікується унікальним telegram_id, який прив’язує всі операції саме до конкретної людини. Додатково можуть зберігатися ім’я користувача в Telegram і дата першого запуску бота.

Таблиця categories відповідає за зберігання категорій операцій, що поділяються на дохідні та видаткові. Наприклад, до видаткових можуть належати «їжа», «транспорт», «кава», «розваги», «покупки», «донати» та «інше». Наявність категорій дозволяє згодом будувати статистику витрат за напрямами.

Центральною є таблиця transactions, адже саме в ній фіксуються всі доходи й витрати. Кожен запис містить ідентифікатор користувача, категорію, тип операції, суму, опис і дату створення. На її основі бот обчислює сукупні доходи, витрати, баланс та формує статистику за тиждень, місяць чи інший період.

Таблиця goals зберігає накопичувальні цілі користувача. У ній зазначаються назва, потрібна сума, уже накопичена частина та бажана дата досягнення. Спираючись на ці значення, бот обчислює, скільки коштів варто регулярно відкладати задля досягнення поставленої мети.

Таблиця reminders призначена для зберігання нагадувань. Вона може містити

текст повідомлення, час надсилання та ознаку активності. Це дає боту змогу періодично нагадувати користувачеві про потребу внести витрати, переглянути статистику чи відкласти кошти на ціль.фінансову ціль.

Зв’язки між таблицями вибудувано так, що одному користувачеві відповідає багато операцій, чимало накопичувальних цілей і декілька нагадувань. Кожну операцію прив’язано до певної категорії, завдяки чому стає можливим групувати витрати та аналізувати їх. Подібна організація бази даних робить зберігання відомостей зручним і закладає підґрунтя для подальшого укладання статистики та порад.

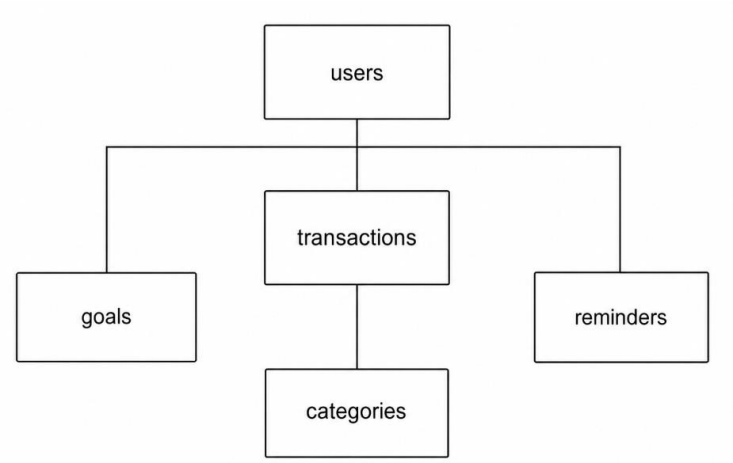


Рисунок 2.3 – Структура зв’язків між основними таблицями бази даних

2.4 Алгоритм формування статистики та рекомендацій

Спроєктувавши базу, слід окреслити, як саме бот використовуватиме збережені операції для побудови статистики та порад. Цей етап важливий для теми роботи, адже бот має не просто накопичувати доходи й витрати, а й допомагати користувачеві глибше розуміти власну фінансову поведінку.

Підґрунтям статистики слугують відомості, які людина вносить під час роботи з ботом: тип операції, сума, категорія, дата та короткий опис. Після потрапляння до бази ці записи стають матеріалом для обчислення сукупних доходів, витрат, балансу, аналізу за категоріями та підготовки порад.

У загальних рисах алгоритм працює так: користувач спершу обирає період аналізу — день, тиждень, поточний чи попередній місяць, після чого бот вибирає з бази всі операції за цей проміжок, поділяє їх на доходи й витрати та виконує потрібні обчислення.потрібний період для аналізу, наприклад день, тиждень, поточний місяць або попередній місяць. Після цього бот отримує з бази даних усі фінансові операції за вказаний проміжок часу, розподіляє їх на доходи й витрати та виконує необхідні обчислення.

Таблиця 2.3 – Основні показники, які повинна формувати система під час аналізу фінансових операцій користувача

Показник	**Опис**
Загальна сума доходів	Відображає суму всіх доходів користувача за обраний період
Загальна сума витрат	Показує суму всіх витрат користувача за обраний період
Поточний баланс	Визначається як різниця між доходами та витратами
Витрати за категоріями	Показує, скільки коштів було витрачено на кожну категорію
Найбільш витратна категорія	Дозволяє визначити напрям, на який користувач витратив найбільше коштів
Порівняння доходів і витрат	Дає змогу оцінити, чи перевищують витрати доходи
Наявність економії	Показує, чи залишилися кошти після врахування всіх витрат

Перелічені показники дають користувачеві загальне уявлення про власний фінансовий стан за обраний проміжок та допомагають побачити напрями, що потребують пильнішої уваги.

Сукупну суму доходів обчислюють додаванням усіх операцій типу «дохід»

за обраний період. Аналогічно визначають загальні витрати, але враховують лише записи типу «витрата». Далі система знаходить баланс користувача як різницю між доходами й витратами.

– У загальному вигляді баланс можна подати так:

– Баланс = Загальна сума доходів – Загальна сума витрат

– Додатний баланс означає, що користувач витратив менше, ніж отримав.

У такому разі бот показує позитивний підсумок і може запропонувати відкласти частину коштів на ціль. Від'ємний баланс свідчить, що витрати перевищили доходи, тож варто уважніше переглянути покупки та скоротити необов'язкові трати.

Важливе значення має й розкладання витрат за категоріями, бо воно показує, куди найчастіше йдуть кошти. Користувачеві може здаватися, що на каву, їжу поза домом чи розваги витрачається небагато, але після підсумовування за місяць ці суми нерідко виявляються відчутними. Саме тому категоризація не лише веде облік, а й допомагає краще бачити власні фінансові звички.

Для цього бот зводить усі витрати за категоріями й рахує суму кожної з них. Далі система визначає категорію з найбільшими тратами й використовує це для формування порад. Такий підхід підвищує користь бота, адже людина отримує не самі цифри, а підказки, на що варто звернути увагу.

Сформувавши статистику, система переходить до підготовки порад. Їхня мета — не обмежувати користувача, а звертати його увагу на певні фінансові звички. Скажімо, якщо впродовж місяця помітна частка витрат припадає на «їжу» чи «каву», бот може порадити бути уважнішим до таких покупок наступного періоду. За наявності позитивного залишку система запропонує спрямувати частину суми на заощадження або накопичувальну ціль.

Таблиця 2.5 – Приклади умов формування рекомендацій

Умова	Рекомендація бота
Витрати перевищують доходи	Рекомендується скоротити необов’язкові витрати та переглянути бюджет
Найбільші витрати у категорії «Їжа»	Варто уважніше контролювати витрати на харчування
Часті витрати на каву або розваги	Можна зменшити кількість дрібних необов’язкових покупок
Є позитивний залишок коштів	Частину коштів можна відкласти на фінансову ціль
Користувач наближається до фінансової цілі	Рекомендується продовжувати відкладати кошти в обраному темпі
До фінансової цілі залишилося мало часу	Варто збільшити суму регулярного відкладення

Визначивши базові умови, система формує поради у зручнішій для сприйняття формі. Важливо, щоб такі повідомлення не звучали як суворі заборони, а сприймалися як корисні підказки. Наприклад, помітивши надмірні витрати на каву чи розваги, бот не просто наведе суму, а делікатно зверне увагу на цю категорію.

Завдяки цьому бот стає кориснішим у щоденному вжитку. Користувач отримує не лише сухі числа, а й стислий коментар про те, на що варто зважити. Так, повідомлення про перевищення витрат над доходами допоможе вчасно переглянути бюджет, а порада відкласти частину залишку наблизить людину до накопичувальної цілі.

Варто додати, що поради можуть бути не лише застережливими, а й мотиваційними. Коли користувач дотримується бюджету або має заощадження, бот повідомляє про позитивний результат і пропонує спрямувати кошти на накопичення, навчання, благодійність чи інші корисні цілі. Це перетворює

фінансовий контроль не просто на засіб обмеження витрат, а й на інструмент планування.

Отже, поради в боті варто будувати на простих і прозорих правилах. Вони допомагають користувачеві об’єктивніше оцінювати власні звички, вчасно помічати проблемні категорії витрат і ухвалювати виваженіші рішення щодо особистого бюджету.приймати більш обдумані рішення щодо особистого бюджету.

Самостійною частиною алгоритму є робота з накопичувальними цілями. Ця функція потрібна, щоб бот не лише контролював поточні витрати, а й допомагав планувати заощадження на конкретну мету. Користувач створює ціль, задаючи її назву, потрібну суму та бажаний строк досягнення.

Скажімо, метою може стати накопичення на навчання, техніку, подорож чи іншу вагому покупку. Після введення даних бот обчислює, скільки коштів треба регулярно відкладати, щоб укластися у визначений строк. Це робить процес прозорішим, адже людина бачить не лише підсумкову суму, а й конкретний денний, тижневий або місячний орієнтир.

У загальному вигляді розрахунок регулярного відкладення можна подати так:

– Сума регулярного відкладення = Необхідна сума / Кількість періодів досягнення цілі

– Якщо користувач уже має певну накопичену суму, формула може бути уточнена:

– Сума регулярного відкладення = (Необхідна сума – Накопичена сума) / Кількість періодів.

Такий розрахунок допомагає користувачеві тверезо оцінити, наскільки досяжна поставлена ціль. Якщо отримана сума завелика, можна подовжити строк накопичення або переглянути власні витрати. Якщо ж вона прийнятна, бот час від часу нагадуватиме про потребу відкласти кошти.

Наприклад, для накопичення 30000 грн за п’ять місяців бот підрахує, що відкладати слід приблизно 6000 грн щомісяця. Якщо частину суми вже зібрано, розрахунок буде меншим і точніше відповідатиме реальній ситуації користувача.

Таблиця 2.6 – Дані для розрахунку фінансової цілі

Дані	**Призначення**
Назва цілі	Визначає, на що користувач планує накопичити кошти
Необхідна сума	Показує загальний обсяг коштів, потрібний для досягнення цілі
Накопичена сума	Відображає, скільки коштів уже відкладено
Бажана дата	Дозволяє визначити термін досягнення цілі
Регулярне відкладення	Показує, скільки потрібно відкладати за день, тиждень або місяць

До того ж накопичувальні цілі можуть переплітатися з порадами. Коли після зведення доходів та витрат у користувача лишається додатний баланс, бот здатен порадити скерувати частину цих коштів на досягнення обраної цілі. Так система не лише відображає залишок, а й спонукає розпоряджатися ним більш виважено.

Висновки до розділу 2

У другому розділі виконано проєктування бота аналітики особистих фінансів із порадами. Основну увагу приділено тому, як саме людина взаємодіятиме з ботом, з яких функціональних частин складатиметься система та яким чином фінансові дані зберігатимуться й використовуватимуться для подальшого аналізу..

Опрацьовано логіку взаємодії користувача з ботом. З’ясовано, що робота має бути простою й покроковою: людина запускає систему, відкриває головне меню, обирає дію, вводить дані та отримує результат у вигляді збереженої операції, статистики чи поради. Це робить користування ботом зручним навіть для тих, хто не має спеціальних технічних навичок.

Окреслено також ключові функціональні складники бота: взаємодію з користувачем, облік доходів і витрат, категоризацію операцій, роботу з базою даних,

побудову статистики, формування порад, ведення накопичувальних цілей та нагадування. Поділ системи на такі частини краще організовує її роботу й спрощує подальшу реалізацію.категоризації фінансових операцій, модуль роботи з базою даних, модуль формування статистики, модуль рекомендацій, модуль фінансових цілей та модуль нагадувань. Поділ системи на такі частини дає змогу краще організувати її роботу та спростити подальшу реалізацію програмного продукту.

Окремо спроєктовано структуру бази даних і визначено таблиці, потрібні для зберігання відомостей про користувачів, операції, категорії, накопичувальні цілі та нагадування. Така схема дозволяє тримати дані впорядковано й застосовувати їх для перегляду історії, побудови статистики та аналізу витрат за різні періоди.

У розділі описано й алгоритм побудови статистики та порад. Визначено основні показники, які має обчислювати бот: суми доходів і витрат, баланс, розподіл за категоріями, найвитратнішу категорію та наявність заощаджень. Спираючись на них, система формує прості й зрозумілі поради, що допомагають користувачеві краще керувати бюджетом і помічати зайві витрати.

Отже, у другому розділі закладено проєктну основу майбутнього бота. Описані сценарії взаємодії, функціональні складники, структура бази та алгоритми аналізу формують підґрунтя для наступного етапу — безпосередньої реалізації бота й перевірки його ключових функцій. Узгодженість запропонованих рішень дає змогу розглядати їх як цілісну основу, придатну до безпосереднього кодування без істотних доопрацювань.

3 РЕАЛІЗАЦІЯ TELEGRAM-БОТА ДЛЯ АНАЛІТИКИ ОСОБИСТИХ ФІНАНСІВ

3.1 Вибір середовища розробки та налаштування проєкту

Щоб практично втілити Telegram-бота, спершу слід облаштувати середовище розробки та окреслити перелік інструментів, які знадобляться під час побудови програмного продукту. На цьому кроці важить не тільки вибір мови програмування, а й продумана організація структури проєкту, інсталяція потрібних бібліотек і підготовка підґрунтя для подальшого втілення функцій бота.

Мовою програмування для бота було визначено Python. Вона добре надається до написання чат-ботів, опрацювання повідомлень, роботи з базами даних та проведення обчислень. До того ж завдяки прозорому синтаксису Python пришвидшує розробку й дає змогу оперативно втілювати окремі функціональні складники системи.

За середовище розробки може правити Visual Studio Code чи PyCharm. Обидва редактори придатні для Python-проєктів, підтримують підсвічування синтаксису, встановлення розширень, а також роботу з терміналом і віртуальним середовищем. Під час створення цього бота зручніше скористатися Visual Studio Code, адже він легкий, швидкий і дозволяє тримати всі файли проєкту в одному вікні.

Перш ніж починати розробку, варто завести окрему теку проєкту. У ній міститимуться всі файли, дотичні до роботи Telegram-бота: головний файл запуску, модулі опрацювання команд, файл взаємодії з базою даних, налаштування та додаткові файли для зберігання конфігураційних параметрів. Такий лад допомагає тримати проєкт упорядкованим і полегшує подальшу роботу з кодом.

Щоб відокремити залежності, доцільно розгорнути віртуальне середовище Python. Це дає змогу встановлювати необхідні бібліотеки саме для цього проєкту, не зачіпаючи інших програм на комп’ютері. Завдяки такому підходу проєкт легше переносити, оновлювати та супроводжувати надалі.

Ключовою бібліотекою для спілкування з Telegram слугує aiogram. Вона дає змогу опрацьовувати команди користувача, повідомлення й натискання кнопок, а також вибудовувати логіку діалогу з ботом. За її допомогою можна реалізувати запуск бота, головне меню, додавання доходів і витрат, перегляд статистики, роботу з накопичувальними цілями та розсилання повідомлень користувачеві.

Для зберігання фінансових відомостей у проєкті зручно застосувати SQLite. Ця база даних обходиться без окремого серверного налаштування, що робить її придатною для навчального та дипломного проєкту. У SQLite можна тримати дані про користувачів, фінансові операції, категорії витрат, накопичувальні цілі та нагадування. Згодом ці дані стають основою для формування статистики й порад.

Під час облаштування проєкту важливо також подбати про збереження чутливих параметрів, передусім токена Telegram-бота. Розміщувати його прямо в основному коді не варто. Натомість можна скористатися окремим файлом налаштувань чи файлом середовища, де зберігатимуться службові дані. Це підвищує захищеність проєкту й зменшує ймовірність випадкового витоку токена.

Загальна структура проєкту може мати такий вигляд як зображено на рисунку 3.1.

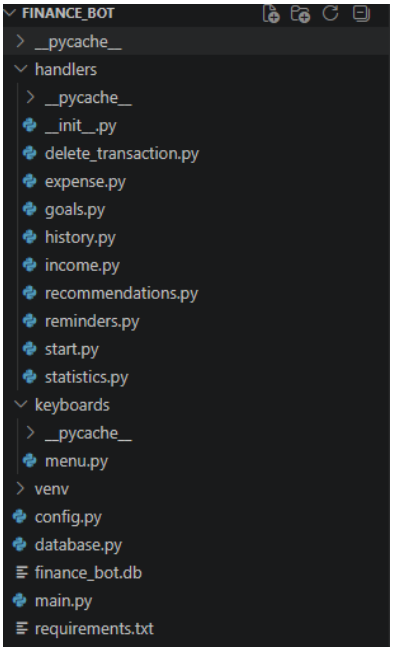


Рисунок 3.1 –Структура проєкту Telegram-бота

Файл main.py слугує для запуску Telegram-бота та під’єднання основних модулів. У config.py можуть зберігатися налаштування проєкту, зокрема токен бота. Файл database.py відповідає за з’єднання з базою даних і виконання основних операцій із нею. Тека handlers містить файли з обробниками команд та дій користувача, а тека keyboards використовується для формування кнопок меню. Файл requirements.txt потрібен, щоб фіксувати перелік бібліотек, задіяних у проєкті.

Така будова є зручною, бо дає змогу розкласти логіку програми на окремі частини. Приміром, функції додавання доходів і витрат не переплітаються з кодом статистики чи роботи з накопичувальними цілями. Це робить проєкт прозорішим і спрощує внесення правок надалі.

Отже, на етапі облаштування проєкту було підібрано основні інструменти для втілення Telegram-бота: Python як мову програмування, aiogram як бібліотеку для взаємодії з Telegram, SQLite як засіб зберігання даних та Visual Studio Code чи PyCharm як середовище розробки. Підготовлена структура проєкту закладає підґрунтя для подальшої реалізації функціональних можливостей бота.

3.2 Створення Telegram-бота через BotFather та підключення до проєкту

Аби Telegram-бот міг функціонувати в середовищі Telegram, його спершу потрібно завести через спеціальний сервіс BotFather. Це офіційний бот Telegram, який слугує для створення нових ботів, налаштування їхніх назв, опису, аватара та видачі токена доступу.

Створення бота розпочинається з відкриття BotFather у Telegram і введення команди /newbot. Далі сервіс просить указати назву бота, яку бачитимуть користувачі, а також унікальне ім’я користувача, що має завершуватися на bot. Після вдалого створення BotFather видає токен доступу, за допомогою якого програмний код може взаємодіяти з Telegram Bot API.

Токен є важливим службовим параметром, тож публікувати його відкрито або залишати у вільному доступі не можна. У проєкті токен доцільно тримати в окремому файлі налаштувань, наприклад config.py, чи у файлі середовища. Такий

підхід відмежовує службову інформацію від основного коду й підвищує захищеність проєкту.

Здобутий токен під'єднується до головного файлу запуску Telegram-бота. Відтоді програма дістає змогу надсилати повідомлення користувачеві, приймати команди, опрацьовувати натискання кнопок та виконувати інші дії через Telegram. Власне, саме токен забезпечує зв'язок між створеним у Telegram ботом і програмною реалізацією, написаною мовою Python.

Щоб поліпшити зовнішній вигляд бота, можна також налаштувати його аватар, стислий опис і вітальне повідомлення. На основну логіку системи це не впливає, проте робить бота зрозумілішим для користувача. Для бота аналітики особистих фінансів варто дібрати назву й опис, які одразу розкривають його призначення: облік доходів і витрат, перегляд статистики та отримання порад.

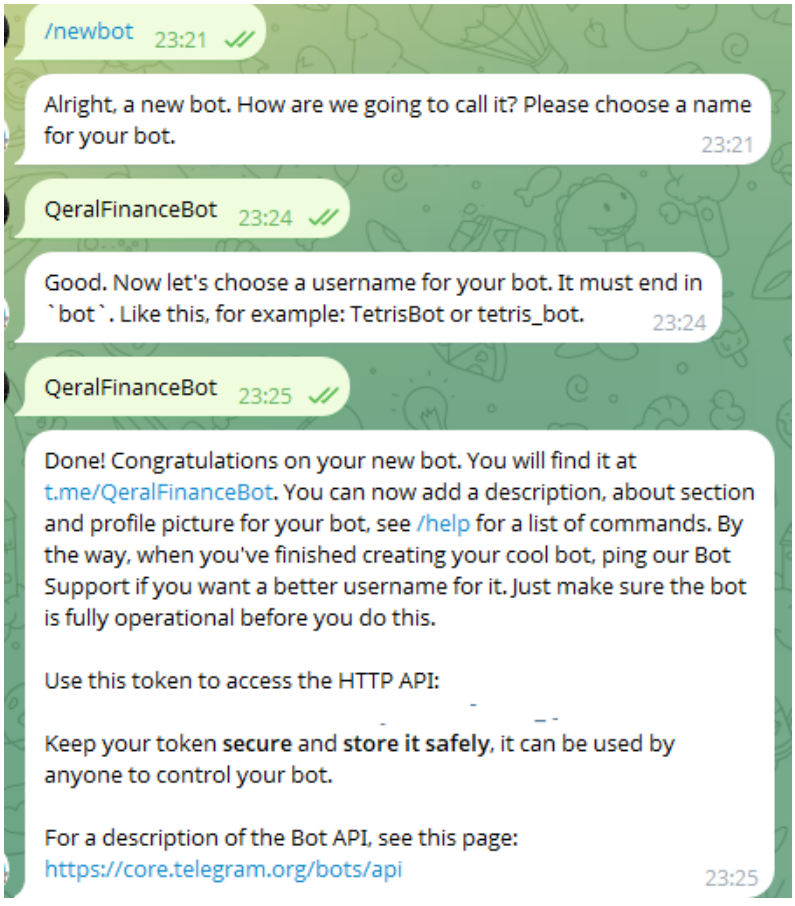


Рисунок 3.2 – Створення Telegram-бота за допомогою BotFather

Після створення Telegram-бота було проведено базове налаштування його зовнішнього вигляду. Боту додано аватар, стислий і повний опис профілю. Аватар слугує для візуального впізнавання бота в Telegram, а опис допомагає користувачеві відразу збагнути призначення системи. У стислому описі зазначено, що бот призначений для обліку доходів, витрат та аналізу особистих фінансів. У повному описі додатково перелічено можливості перегляду статистики, роботи з накопичувальними цілями та отримання порад щодо керування бюджетом.

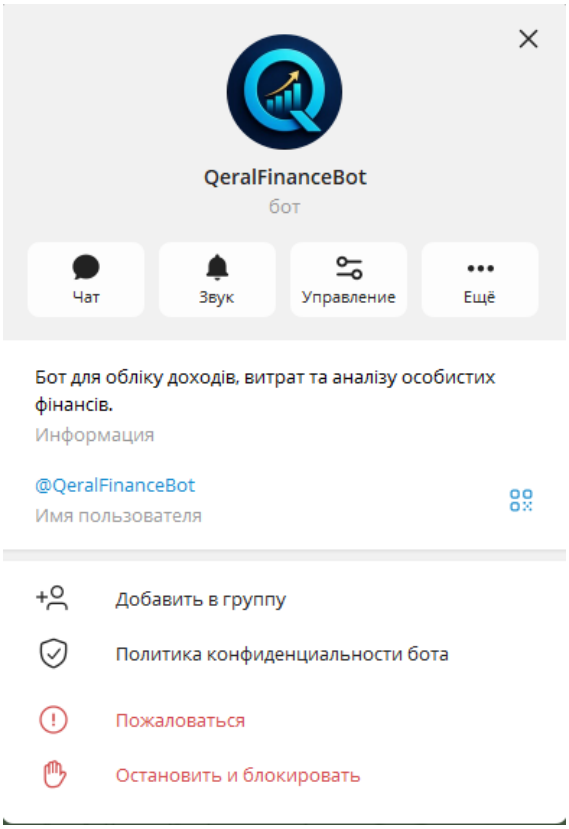


Рисунок 3.3 – Профіль Telegram-бота QeralFinanceBot

3.3 Реалізація головного меню та навігації Telegram-бота

Щойно Telegram-бота створено й під’єднано до проєкту, наступним кроком стає реалізація головного меню. Саме з нього бере початок основна взаємодія користувача з ботом, тому меню має бути простим, зрозумілим і відкривати доступ до всіх ключових функцій системи.

Під час першого запуску користувач надсилає команду /start. У відповідь бот

надсилає вітальне повідомлення та пропонує перейти до головного меню. У цьому повідомленні стисло пояснюється призначення бота: облік доходів і витрат, перегляд статистики, отримання порад і робота з накопичувальними цілями.

Головне меню збудовано на кнопках Telegram. Такий підхід зручний, адже користувачеві не доводиться тримати в пам’яті безліч команд. Достатньо натиснути потрібну кнопку, після чого бот переходить до відповідного сценарію роботи.

До головного меню додано основні кнопки (рис. 3.4).

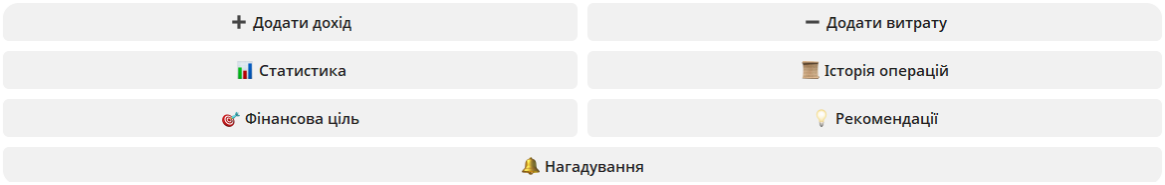


Рисунок 3.4 – Створення кнопок головного меню

Кнопка додавання доходу спрямовує користувача до сценарію введення отриманих коштів. Кнопка додавання витрати слугує для фіксації витрачених коштів із зазначенням категорії. Розділ статистики дає змогу отримати підсумки за певний проміжок, приміром за тиждень чи місяць. Історія операцій потрібна для перегляду попередніх записів, а розділ накопичувальних цілей дозволяє створювати цілі та стежити за поступом.

Окрему увагу приділено розділу порад. Тут користувач може отримати стислі рекомендації на основі своїх витрат. Скажімо, якщо чималу частку бюджету з’їдають кава, їжа або розваги, бот запропонує пильніше контролювати ці категорії. Розділ нагадувань слугує для повідомлень, які допомагають користувачеві не забувати вносити витрати та дотримуватися фінансової дисципліни.

Навігацію в боті варто облаштувати так, щоб користувач завжди мав змогу повернутися до головного меню. Це суттєво, адже під час роботи з ботом він може переходити між різними сценаріями: додаванням витрат, переглядом статистики чи створенням накопичувальної цілі. Кнопка повернення до меню робить взаємодію зручнішою та знижує ймовірність помилок.

Таким чином, головне меню постає осердям взаємодії користувача з Telegram-ботом (рис. 3.5). Воно відкриває швидкий доступ до ключових функцій системи й робить роботу з ботом зрозумілою навіть тим, хто раніше не послуговувався подібними інструментами для контролю особистих фінансів.

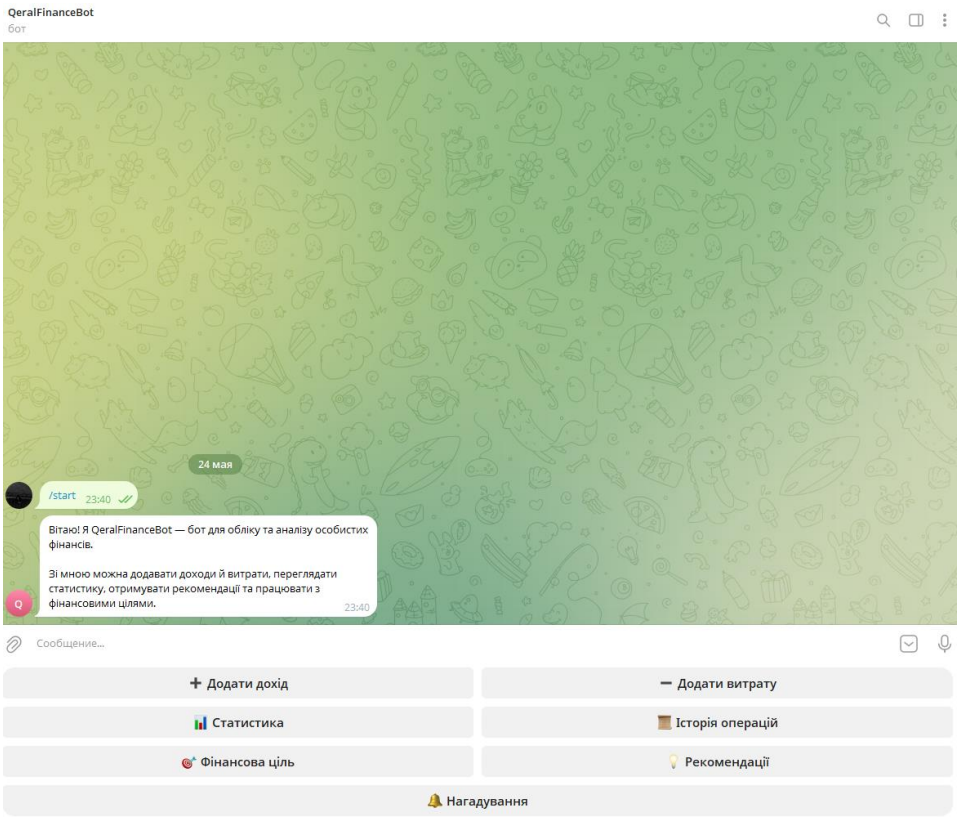


Рисунок 3.5 – Головне меню Telegram-бота QeralFinanceBot

3.4 Реалізація додавання доходів і витрат

Однією з провідних функцій Telegram-бота QeralFinanceBot є додавання фінансових операцій користувача. Саме на підставі внесених доходів і витрат згодом вибудовується статистика, історія операцій, поради та аналіз особистого бюджету. Тому реалізація цієї частини є вагомим етапом створення бота.

У головному меню користувачеві доступні дві окремі кнопки: «Додати дохід» і «Додати витрату». Такий поділ зроблено, щоб користувач одразу обирав потрібний тип операції й не марнував час на додаткові команди. Якщо кошти надійшли, він обирає додавання доходу. Якщо ж кошти витрачено, він переходить

до сценарію додавання витрати.

Бот пропонує користувачеві ввести суму отриманих коштів (рис.3.6). Після введення система перевіряє, чи є воно коректним числом і чи перевищує нуль. Коли значення хибне, бот просить повторити введення. Коли сума вказана правильно, дохід записується до бази даних, а користувач отримує сповіщення про успішне додавання операції.

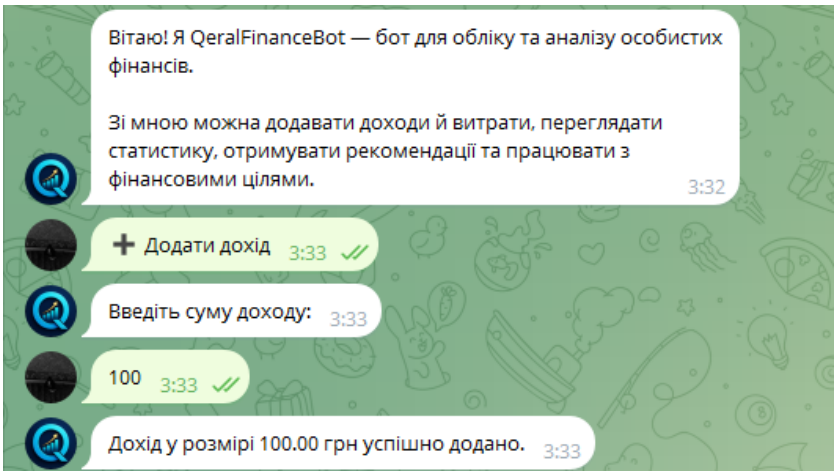


Рисунок 3.6 – Додавання доходу в Telegram-боті QeralFinanceBot

Додавання витрати влаштоване дещо ширше, бо тут важливо не лише зафіксувати суму, а й визначити категорію. Це потрібно для подальшого розбору структури витрат користувача. Після натискання кнопки «Додати витрату» бот пропонує обрати одну з наявних категорій, як-от їжа, транспорт, кава, розваги, покупки, здоров’я, навчання, донати чи інше (рис. 3.7).

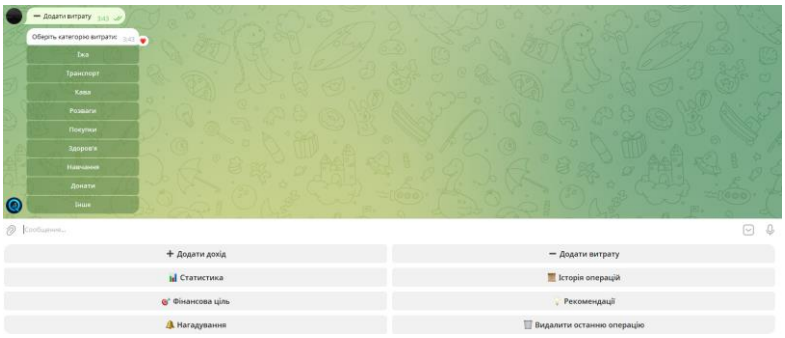


Рисунок 3.7 – Вибір категорії витрати в Telegram-боті QeralFinanceBot

Обравши категорію, користувач вводить суму витрати. Як і з доходами, система перевіряє правильність наданих даних. Якщо сума коректна, бот зберігає витрату в базі даних разом із зазначеною категорією (рис 3.8). Надалі ця інформація стає в пригоді для формування статистики та порад.

Рисунок 3.8 – Додавання витрати в Telegram-боті QeralFinanceBot

Під час втілення цієї функції було також ураховано проблему повторного додавання категорій. Оскільки категорії постають під час запуску бота, важливо було убезпечитися від їх дублювання в базі даних. Задля цього в таблиці категорій застосовано обмеження унікальності для назви й типу категорії. Це дає змогу зберігати кожну категорію лише раз і уникати повторення однакових кнопок у меню вибору витрат.

Отже, реалізація додавання доходів і витрат забезпечує опорну частину роботи Telegram-бота. Користувач може оперативно вносити фінансові операції, а система впорядковано зберігає їх для подальшого аналізу, формування статистики та порад.

3.5 Реалізація перегляду статистики та історії операцій

Услід за додаванням доходів і витрат наступним кроком стала розробка

функцій перегляду збереженої фінансової інформації. Користувачеві важливо не лише вносити операції, а й мати змогу будь-якої миті побачити загальний стан бюджету, суму доходів, суму витрат, баланс та історію попередніх записів.

У Telegram-боті QeralFinanceBot задля цього передбачено два окремі розділи: «Статистика» та «Історія операцій». Розділ статистики слугує для швидкого огляду фінансових підсумків, а історія операцій дозволяє користувачеві переглянути останні внесені доходи й витрати.

Натиснувши кнопку «Статистика», бот дістає з бази даних усі фінансові операції конкретного користувача й опрацьовує їх. Система окремо підсумовує загальну суму доходів, загальну суму витрат і визначає поточний баланс. Баланс обчислюється як різниця між доходами та витратами, тож користувач одразу бачить, скільки коштів лишилося після здійснених витрат.

Окрім підсумкових сум, бот також показує витрати за категоріями (рис 3.9). Це робить статистику кориснішою, адже користувач бачить не лише загальну суму витрат, а й розуміє, на які саме напрями пішли кошти. Приміром, якщо найбільша частка витрат припадає на їжу, каву чи розваги, користувач може звернути на це увагу та скоригувати витрати надалі.

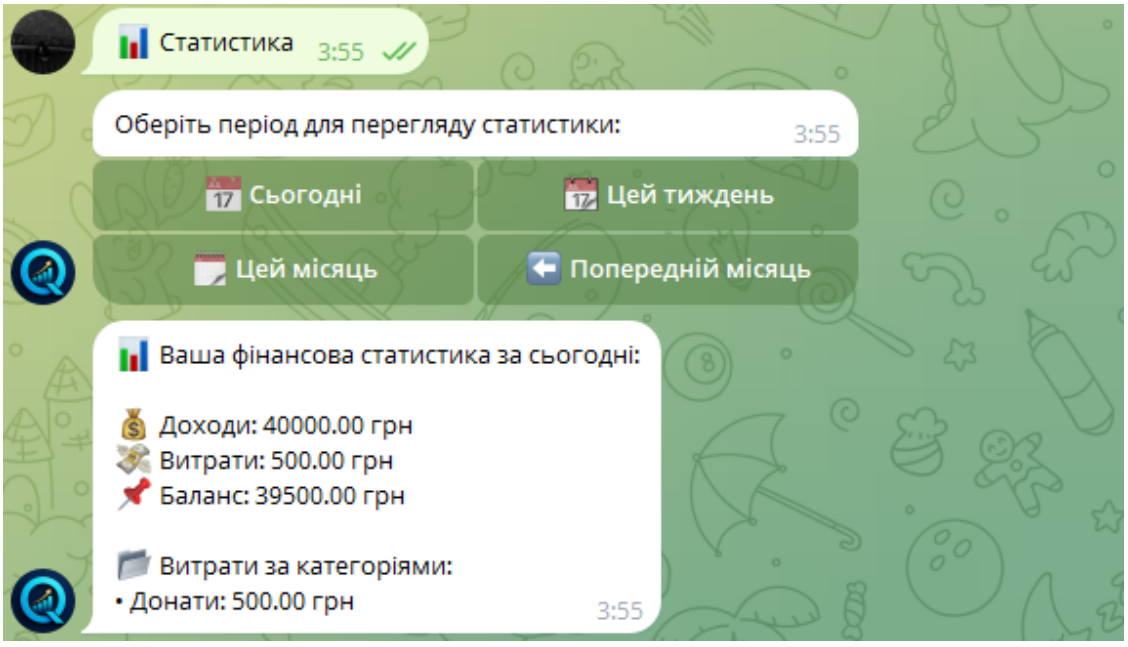


Рисунок 3.9 – Перегляд статистики в Telegram-боті QeralFinanceBot

Окремо реалізовано перегляд історії операцій. Ця функція потрібна, щоб користувач міг перевірити останні внесені записи й переконатися, що дохід чи витрату додано правильно. У переліку історії бот показує тип операції, суму, категорію та дату створення запису.

Історія операцій надто корисна тоді, коли користувач хоче швидко пригадати, які фінансові дії здійснено останніми (рис. 3.10). Скажімо, якщо він випадково ввів хибну суму, то може переглянути історію й за потреби скористатися видаленням останньої операції.

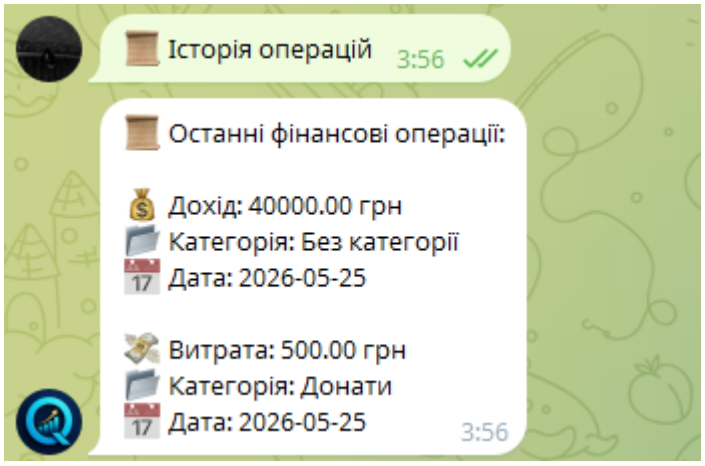


Рисунок 3.10 – Перегляд історії фінансових операцій

Реалізація статистики та історії операцій робить Telegram-бота кориснішим для щоденного вжитку. Користувач дістає не просто засіб для запису доходів і витрат, а змогу аналізувати власну фінансову поведінку. Завдяки цьому бот допомагає швидше помічати зрушення в бюджеті, тримати витрати під контролем і ухвалювати виваженіші фінансові рішення.

3.6 Реалізація фінансових цілей

Поряд з обліком доходів і витрат у Telegram-боті QeralFinanceBot втілено можливість роботи з накопичувальними цілями. Ця функція потрібна, щоб користувач міг не лише бачити поточні витрати, а й планувати накопичення коштів на певну мету.

Після натискання кнопки «Фінансова ціль» бот показує користувачеві перелік уже створених цілей або сповіщає, що цілей поки що немає. Далі користувач може завести нову ціль чи поповнити наявну. Такий підхід робить функцію зручнішою, адже поступ можна відстежувати крок за кроком.

Створюючи нову ціль, бот по черзі запитує її назву, потрібну суму та бажану дату досягнення(рис. 3.11). Здобувши ці дані, система прикидає, скільки коштів варто приблизно відкладати щодня, щотижня чи щомісяця. Завдяки цьому користувач одразу оцінює, наскільки реально досягти мети в обраний строк.

Рисунок 3.11 – Створення фінансової цілі в Telegram-боті QeralFinanceBot

Окремо втілено можливість поповнення вже створеної цілі. Якщо в користувача кілька цілей, бот пропонує обрати конкретну зі списку (рис.3.12). Після вибору користувач вводить суму, яку хоче докласти, а система оновлює

накопичену суму та показує залишок до досягнення мети

Рисунок 3.12 – Додавання коштів до фінансової цілі

Реалізація накопичувальних цілей робить Telegram-бота кориснішим для планування бюджету. Користувач здобуває не лише відомості про минулі витрати, а й інструмент поступового накопичення коштів. Це допомагає краще розподіляти бюджет і свідоміше ставитися до власних фінансових рішень.

3.7 Реалізація рекомендацій, нагадувань та видалення останньої операції

У Telegram-боті QeralFinanceBot, окрім базового обліку доходів і витрат, реалізовано додаткові функції, які роблять роботу з ботом кориснішою. До них належать поради, нагадування та можливість видалити останню фінансову

операцію.

Функція порад потрібна, щоб бот не лише показував користувачеві цифри, а й допомагав глибше розуміти власну фінансову поведінку. Після натискання кнопки «Рекомендації» система розбирає збережені доходи, витрати, баланс і найвитратніші категорії. Спираючись на ці дані, бот укладає стислі поради, що можуть спонукати користувача уважніше ставитися до бюджету.

Коли витрати користувача перевищують доходи, бот наголошує на потребі переглянути необов'язкові покупки (рис. 3.13). Якщо найбільше коштів пішло на певну категорію, приміром їжу, каву чи розваги, система звертає на неї увагу користувача. Якщо ж у користувача є додатний залишок, бот може порадити відкласти частину суми на накопичувальну ціль або скерувати її на важливі потреби.

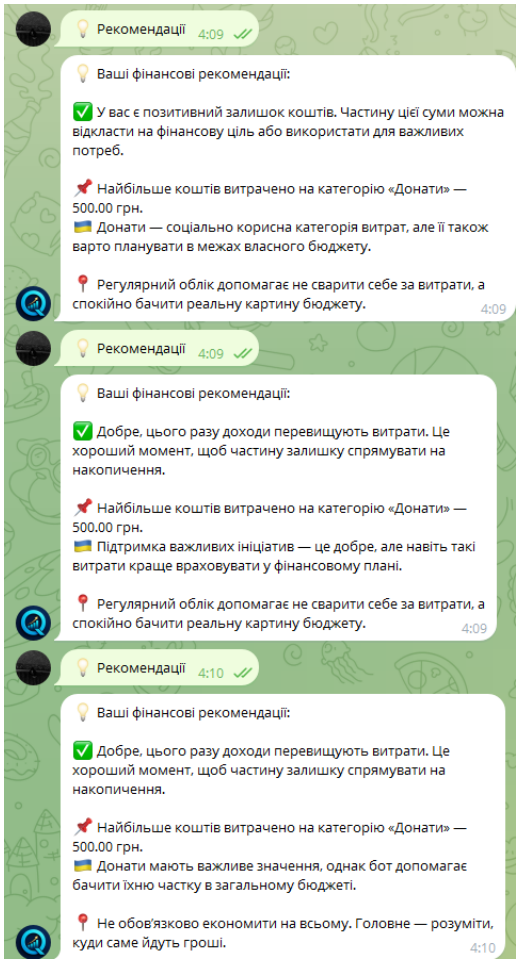


Рисунок 3.13 – Формування рекомендацій у Telegram-боті QeralFinanceBot

Окремо реалізовано функцію нагадувань. Вона потрібна, щоб користувач не забував регулярно вносити фінансові операції та стежити за щоденними витратами. У втіленій версії бот надсилає користувачеві мотиваційні повідомлення, які нагадують записувати витрати, переглядати бюджет чи відкладати кошти на накопичувальну ціль (рис 3.14).

Такі повідомлення не є жорсткими обмеженнями для користувача. Їхня мета — лагідно нагадати про фінансову дисципліну й допомогти виробити звичку регулярного обліку. Скажімо, бот може порадити утриматися від необов'язкової покупки або відкласти невелику суму на ціль. Завдяки цьому спілкування з ботом виглядає живішим і кориснішим.

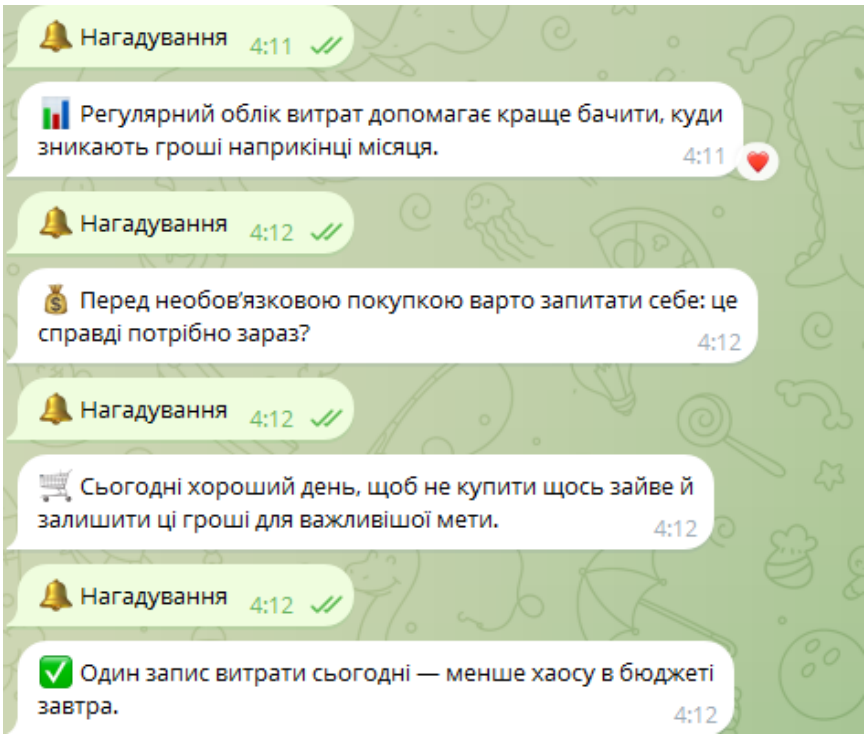


Рисунок 3.14 – Приклад нагадування в Telegram-боті QeralFinanceBot

Також у боті втілено можливість видалити останню операцію. Ця функція стає в пригоді, коли користувач випадково ввів хибну суму або додав операцію помилково. Після натискання кнопки «Видалити останню операцію» бот відшукує останній запис користувача в базі даних, прибирає його та повідомляє, яку саме операцію видалено (рис.3.15).

Наявність такої функції робить роботу з ботом зручнішою. Користувач не лишається з помилково внесеними даними й може швидко виправити ситуацію, не редагуючи базу даних вручну та не очищаючи всю історію операцій.

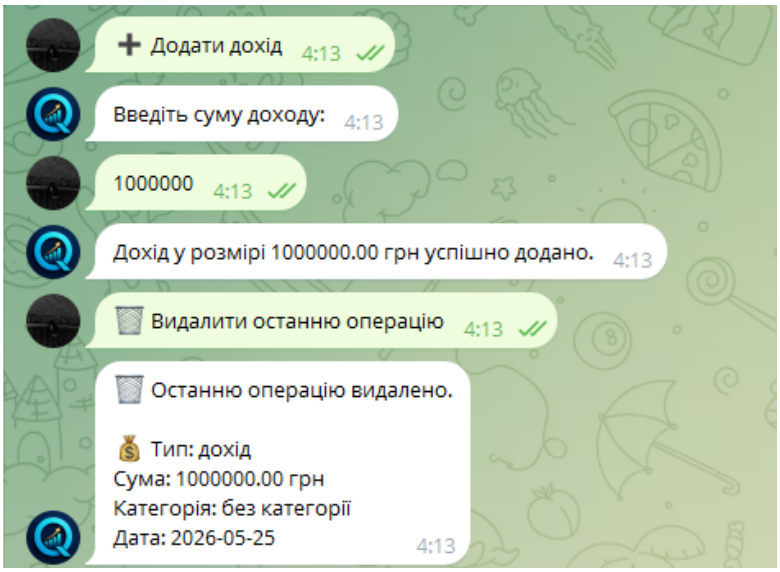


Рисунок 3.15 – Видалення останньої фінансової операції

Отже, реалізація порад, нагадувань і видалення останньої операції розширює спроможності Telegram-бота. Користувач здобуває не лише засіб для збереження доходів і витрат, а й додаткову підтримку у вигляді порад, мотиваційних повідомлень та змоги швидко виправляти огріхи під час введення фінансових даних.

3.8 Тестування основних функцій Telegram-бота

Після втілення основних функцій Telegram-бота було проведено тестування його роботи. Мета тестування — пересвідчитися в правильності виконання ключових сценаріїв взаємодії користувача з ботом, збереження фінансових операцій, формування статистики, роботи з накопичувальними цілями, порадами та нагадуваннями.

Тестування провадилося шляхом послідовного натискання основних кнопок головного меню. Для кожної функції перевіряли, чи правильно бот відгукується на

дії користувача, чи зберігаються внесені дані та чи відповідає результат очікуваній поведінці системи.

Таблиця 3.1 – Результати тестування основних функцій Telegram-бота

Функція	**Дія користувача**	**Очікуваний результат**	**Результат**
Запуск бота	Введення команди /start	Бот надсилає привітальне повідомлення та головне меню	Виконано
Додавання доходу	Натискання кнопки «Додати дохід» і введення суми	Дохід зберігається в базі даних	Виконано
Додавання витрати	Вибір категорії та введення суми витрати	Витрата зберігається з відповідною категорією	Виконано
Перегляд статистики	Натискання кнопки «Статистика»	Бот показує доходи, витрати, баланс і витрати за категоріями	Виконано
Перегляд історії	Натискання кнопки «Історія операцій»	Бот показує останні фінансові операції користувача	Виконано

Функція	**Дія користувача**	**Очікуваний результат**	**Результат**
Створення фінансової цілі	Введення назви, суми та дати цілі	Бот створює ціль і розраховує суму регулярного відкладення	Виконано
Додавання коштів до цілі	Вибір цілі та введення суми	Бот оновлює накопичену суму та показує залишок	Виконано
Формування рекомендацій	Натискання кнопки «Рекомендації»	Бот формує поради на основі доходів, витрат і категорій	Виконано
Нагадування	Натискання кнопки «Нагадування»	Бот надсилає мотиваційне повідомлення	Виконано
Видалення останньої операції	Натискання кнопки «Видалити останню операцію»	Бот видаляє останній фінансовий запис	Виконано

За підсумками тестування з’ясовано, що основні функції Telegram-бота працюють. Бот правильно відгукується на команди користувача, зберігає доходи й витрати, відображає статистику, показує історію операцій і дозволяє працювати з накопичувальними цілями.

Окремо перевірено випадки з некоректним введенням даних. Приміром, якщо користувач вводить не число або суму, меншу за нуль, бот не зберігає таку операцію, а просить указати коректне значення. Це допомагає зменшити кількість помилок під час роботи з фінансовими даними.

Щоб підтвердити працездатність Telegram-бота, було також проведено візуальну перевірку основних сценаріїв роботи. Зокрема, перевірено додавання доходів, формування статистики, роботу з накопичувальними цілями та виведення порад. Приклади результатів тестування показано на рисунках 3.15–3.19.

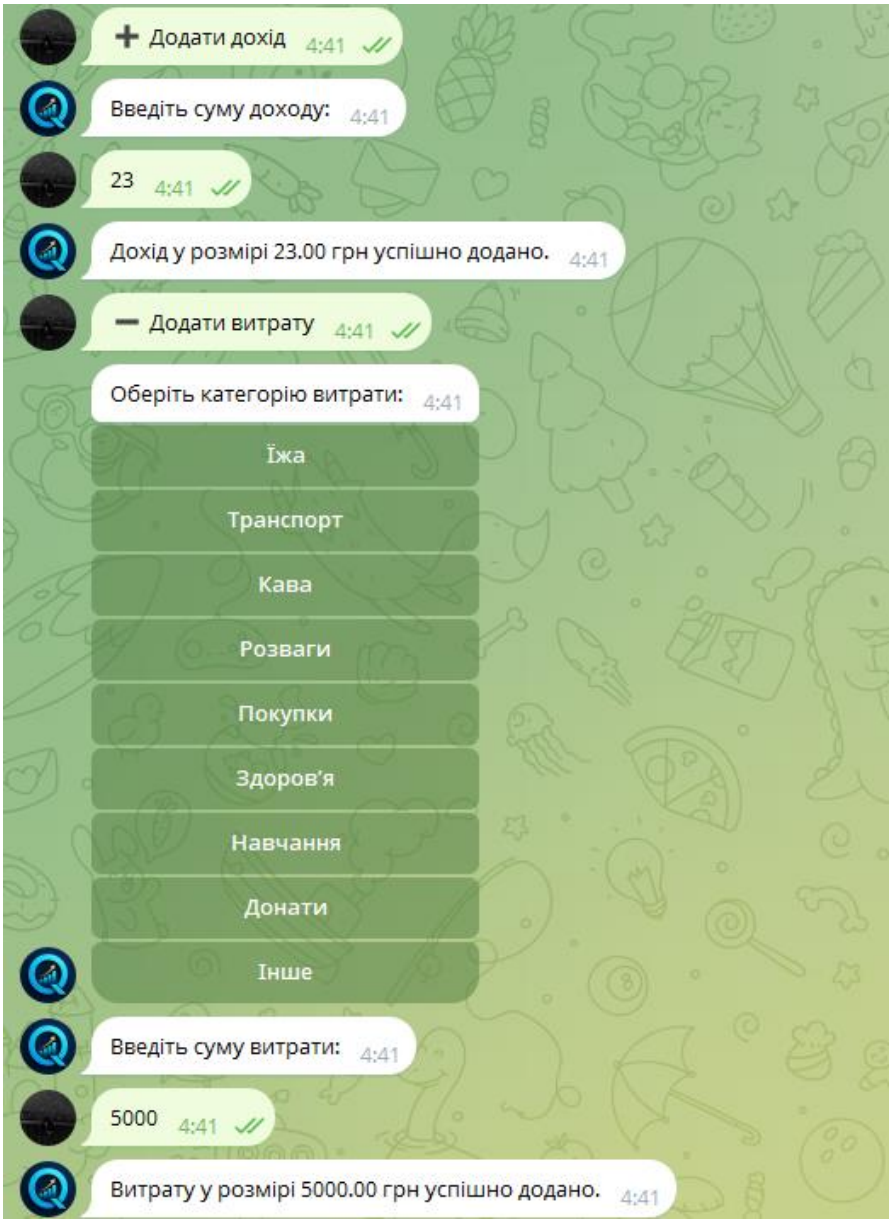


Рисунок 3.15 – Перевірка функції додавання доходут та витрат

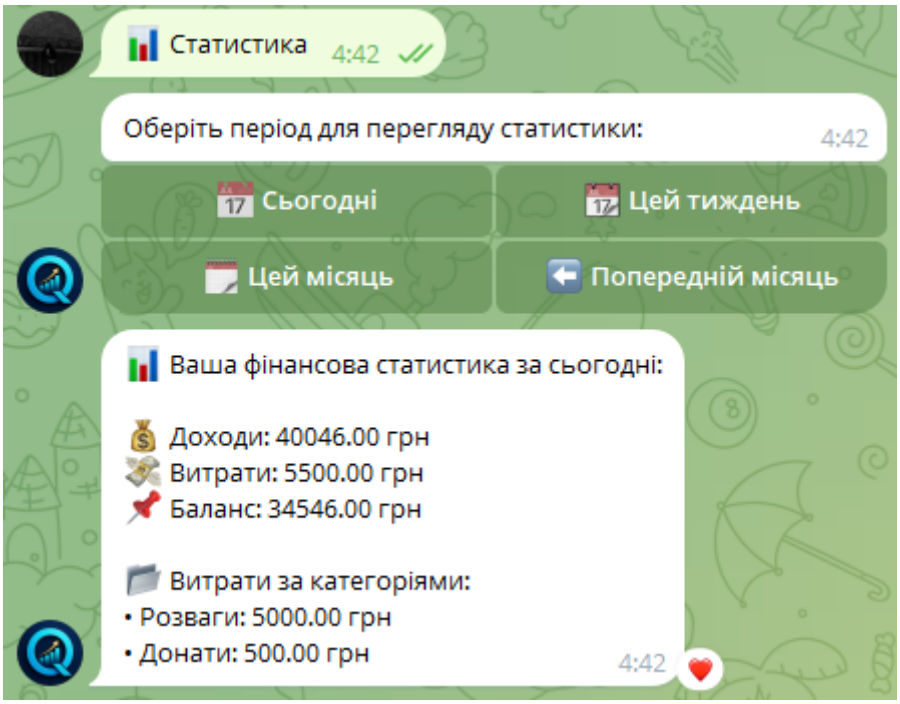


Рисунок 3.16 – Перевірка формування статистики

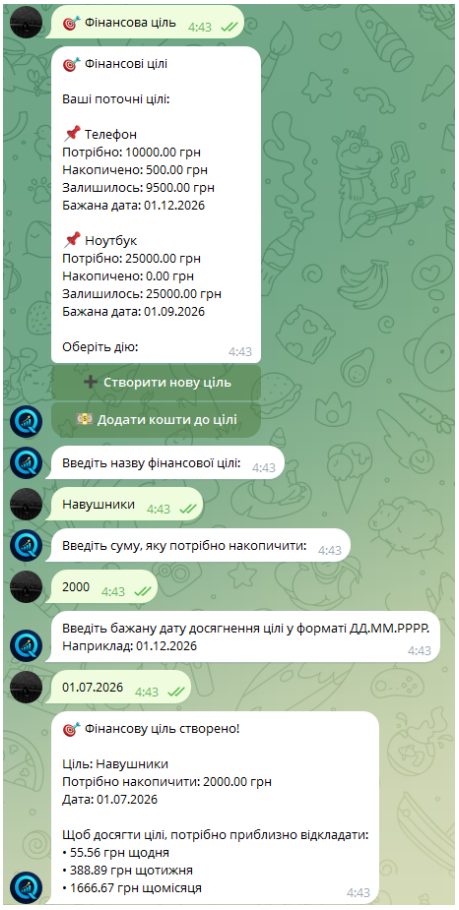


Рисунок 3.17 – Перевірка створення фінансової цілі

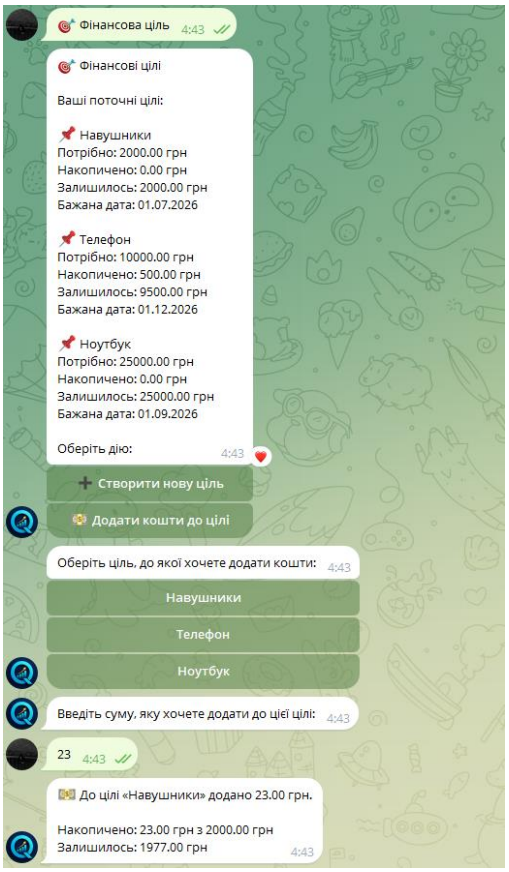


Рисунок 3.18 – Перевірка на працездатність вибору цілі

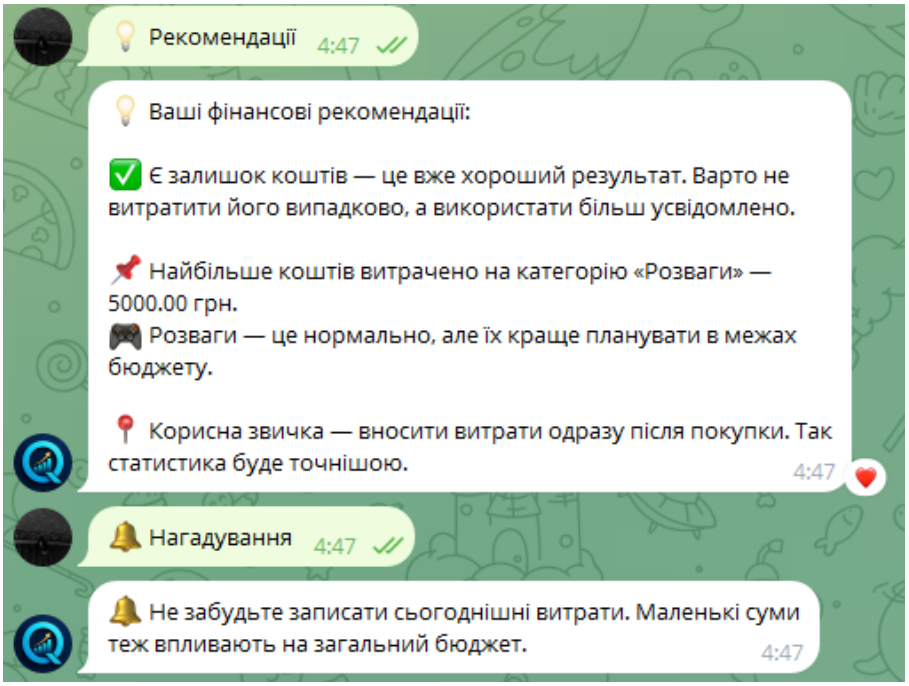


Рисунок 3.19 – Перевірка формування рекомендацій або нагадування

Висновки до розділу 3

У третьому розділі здійснено практичне втілення Telegram-бота QeralFinanceBot для аналітики особистих фінансів із порадами. На цьому етапі облаштовано середовище розробки, сформовано структуру проєкту, заведено Telegram-бота через BotFather та під’єднано його до програмного коду.

Під час реалізації сформовано головне меню бота, через яке користувач дістає доступ до основних функцій системи. Завдяки кнопкам меню взаємодія з ботом проста й зрозуміла: користувачеві не доводиться тримати в пам’яті окремі команди, достатньо обрати потрібну дію.

У боті втілено додавання доходів і витрат. Для витрат передбачено вибір категорії, що згодом дозволяє аналізувати структуру витрат користувача. Усі фінансові операції зберігаються в базі даних, тож користувач може переглядати історію операцій, статистику та послуговуватися цими даними для отримання порад.

Також реалізовано перегляд статистики та історії фінансових операцій. Статистика дає змогу побачити загальну суму доходів, витрат, баланс і розподіл витрат за категоріями. Історія операцій допомагає користувачеві перевіряти останні внесені записи й контролювати правильність наданих даних.

Окрему увагу приділено накопичувальним цілям. Користувач може завести ціль, указати потрібну суму та дату досягнення, а також поступово докладати кошти до вже створеної цілі. Бот обчислює орієнтовну суму регулярного відкладення й показує поступ накопичення.

Крім основних функцій, реалізовано поради, нагадування та видалення останньої операції. Поради вибудовуються на підставі фінансових даних користувача, нагадування підтримують звичку регулярного обліку витрат, а видалення останньої операції дозволяє швидко виправити помилково внесені дані.

Проведене тестування засвідчило, що основні функції Telegram-бота працюють. Бот приймає дані від користувача, зберігає фінансові операції, формує

статистику, працює з накопичувальними цілями та надає поради. Таким чином, у третьому розділі підтверджено працездатність розробленого програмного продукту та його відповідність поставленому завданню. Отримані під час перевірки результати підтвердили стабільність закладеної логіки, а поодинокі зауваження було усунено ще на етапі тестування.

4 РЕЗУЛЬТАТИ РЕАЛІЗАЦІЇ TELEGRAM-БОТА ТА ПЕРСПЕКТИВИ ВДОСКОНАЛЕННЯ

4.1 Аналіз отриманих результатів розробки

Унаслідок виконання роботи було створено Telegram-бота QeralFinanceBot, призначеного для обліку та аналізу особистих фінансів користувача. Закладена в нього ідея полягає в тому, щоб дати користувачеві простий засіб щоденного контролю доходів, витрат, накопичувальних цілей та отримання порад щодо керування бюджетом.

Створений Telegram-бот діє через інтерфейс Telegram, тож користувачеві не доводиться встановлювати окремий застосунок. Це робить систему зручною для оперативного вжитку, адже більшість дій виконується через кнопки головного меню. Користувач може вносити фінансові операції, переглядати статистику, розбирати історію витрат, працювати з накопичувальними цілями та отримувати поради.

Під час реалізації було сформовано головне меню, яке зводить докупи всі основні функції бота. Через нього користувач переходить до додавання доходу, додавання витрати, перегляду статистики, історії операцій, накопичувальних цілей, порад, нагадувань і видалення останньої операції. Завдяки такому підходу робота з ботом стає зрозумілою навіть тому, хто раніше не послуговувався подібними інструментами.

Щоб наочно показати здобутий результат, на рис. 4.1 наведено опис Telegram-бота QeralFinanceBot у середовищі Telegram, а на рис. 4.2 — головне меню бота після запуску.

Рисунок 4.1 – Опис Telegram-бота QeralFinanceBot у Telegram

Рисунок 4.2 – Головне меню розробленого Telegram-бота QeralFinanceBot

Одним із ключових результатів розробки є втілення обліку доходів і витрат. Користувач може додати дохід, указавши суму отриманих коштів, або внести витрату, попередньо обравши відповідну категорію. Розподіл витрат за категоріями становить важливу частину системи, адже саме він уможливлює подальший аналіз того, на які напрями користувач витрачає найбільше грошей.

Також було втілено збереження фінансових операцій у базі даних. Завдяки цьому бот не лише відгукується на поточні дії користувача, а й накопичує відомості для подальшого аналізу. Збережені дані стають у пригоді для формування статистики, історії операцій, порад та обчислення накопичувальних цілей.

Окремим результатом є реалізація статистики. Бот показує користувачеві сукупну суму доходів, витрат, поточний баланс і розподіл витрат за категоріями. Завдяки цьому користувач здатен швидко оцінити свій фінансовий стан і збагнути, які категорії найдужче впливають на бюджет.

Поряд зі статистикою в боті втілено історію операцій. Ця функція дає змогу переглядати останні внесені доходи та витрати. Вона корисна для перевірки правильності наданих даних і допомагає користувачеві краще стежити за послідовністю своїх фінансових дій.

Вагомою частиною створеного бота є робота з накопичувальними цілями. Користувач може завести ціль, указати потрібну суму та дату досягнення. Опісля бот прикидає, скільки приблизно слід відкладати щодня, щотижня чи щомісяця. Передбачено також змогу докладати кошти до вже створеної цілі та стежити за поступом накопичення.

У системі також реалізовано поради. Вони вибудовуються на основі

фінансових даних користувача: доходів, витрат, балансу та найвитратніших категорій. Поради подаються в зрозумілій формі й допомагають звернути увагу на необов'язкові витрати, додатний залишок коштів або потребу уважнішого планування бюджету.

Додатково втілено функцію нагадувань. Вона містить набір мотиваційних повідомлень, що допомагають користувачеві не забувати про фінансовий облік, витрати та накопичувальні цілі. Такі повідомлення роблять взаємодію з ботом живішою та менш формальною.

Ще одним практичним результатом стала функція видалення останньої операції. Вона потрібна тоді, коли користувач випадково ввів хибну суму або додав операцію помилково. Завдяки їй можна швидко виправити огріх, не змінюючи дані в базі вручну.

Таким чином, унаслідок розробки постав Telegram-бот, який виконує основні завдання, окреслені в роботі. QeralFinanceBot дає змогу вести облік доходів і витрат, зберігати фінансові операції, переглядати статистику, працювати з накопичувальними цілями, отримувати поради та користуватися нагадуваннями. Це підтверджує, що створений програмний продукт придатний як зручний засіб базового контролю особистих фінансів.

4.2 Практичне використання розробленого Telegram-бота

Створений Telegram-бот QeralFinanceBot може слугувати для щоденного контролю особистих фінансів. Користувач має змогу оперативно вносити доходи й витрати, переглядати статистику, розбирати витрати за категоріями та працювати з накопичувальними цілями.

Практична користь бота в тому, що користувач може заносити фінансові операції одразу після їх здійснення. Завдяки цьому дані не губляться, а поступово складаються в зрозумілу картину особистого бюджету. Особливо це стає в пригоді для контролю дрібних щоденних витрат, які поодинці видаються незначними, проте за місяць можуть сягати помітної суми.

Приклад практичного застосування Telegram-бота показано на рисунку 4.1.

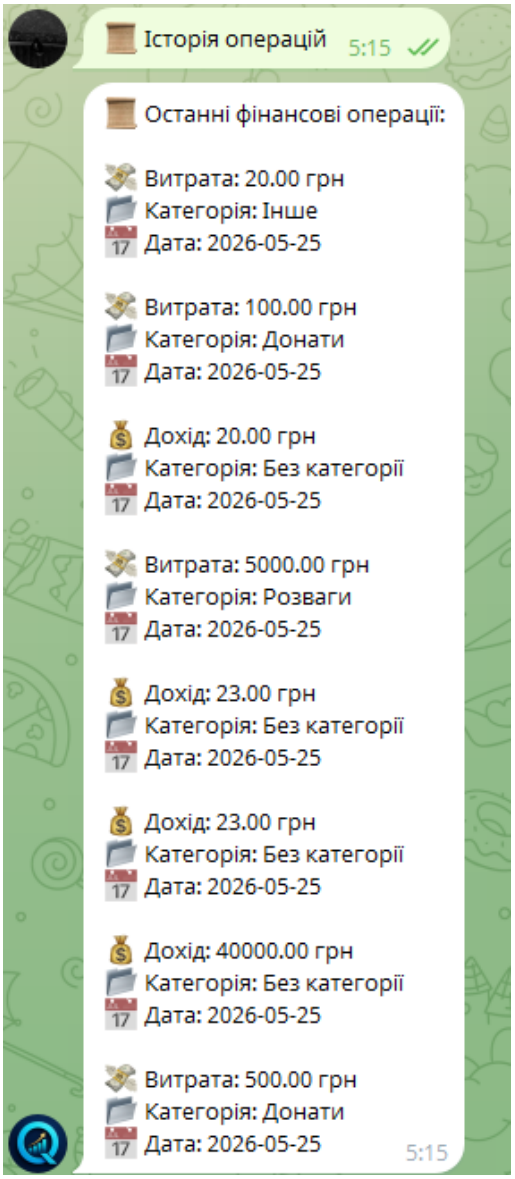


Рисунок 4.3 – Приклад практичного використання Telegram-бота QeralFinanceBot

Також QeralFinanceBot може прислужитися для планування накопичень. Користувач заводить ціль, зазначає потрібну суму та дату, після чого бот прикидає орієнтовну суму регулярного відкладення. Це допомагає краще планувати майбутні витрати й поступово наближатися до поставленої мети.

Отже, практичне застосування QeralFinanceBot зводиться до простого щоденного обліку фінансових операцій, контролю витрат, перегляду статистики та планування накопичувальних цілей через звичний інтерфейс Telegram.

4.3 Обмеження системи та перспективи подальшого вдосконалення

Хоча Telegram-бот QeralFinanceBot виконує основні функції обліку та аналізу особистих фінансів, створена система має певні обмеження. Передусім у нинішній версії всі фінансові операції користувач уносить вручну. Це робить систему простою та зрозумілою, проте вимагає від користувача регулярності й уважності під час внесення даних.

Ще одне обмеження полягає в тому, що бот не під’єднано до банківських сервісів. Через це система не здатна самостійно діставати відомості про платежі з банківських карток чи рахунків. З одного боку, ручне введення дає змогу врахувати будь-які витрати, зокрема готівкові. З іншого боку, надалі автоматична синхронізація з банківськими сервісами зробила б облік фінансів іще зручнішим.

Варто також зауважити, що поради в нинішній версії укладаються за простими правилами. Бот розбирає доходи, витрати, баланс і найвитратніші категорії, після чого дає користувачеві відповідні рекомендації. Для базової версії системи цього достатньо, проте надалі поради можна зробити гнучкішими та персоналізованішими.

Окремим напрямом подальшого поступу є розгортання Telegram-бота на сервері. На етапі розробки бот стартує локально на комп’ютері, тож його робота залежить від того, чи запущено програму. Для постійного вжитку доцільно розмістити бота на сервері або хмарній платформі, щоб він працював безперервно.

Крім того, у нинішній версії система обмежено візуалізує дані. Бот показує користувачеві статистику в текстовому вигляді, що зручно для швидкого перегляду, однак для глибшого фінансового аналізу цього може бракувати. Приміром, користувачеві було б зручніше бачити структуру витрат не лише переліком категорій, а й у вигляді кругової діаграми чи графіка. Така візуалізація допомогла б швидше збагнути, які категорії посідають найбільшу частку бюджету.

Також надалі можна розширити функцію статистики. У втіленій версії користувач переглядає основні фінансові показники, проте систему можна

доповнити докладнішими періодами аналізу. Скажімо, варто додати перегляд статистики за довільний проміжок, порівняння витрат між різними місяцями або обчислення середніх щоденних витрат. Це зробило б фінансовий аналіз точнішим і кориснішим для користувача.

У перспективі бот міг би готувати стислий місячний звіт у форматі таблиці чи документа. Це знадобилося б користувачам тримати фінансову історію окремо, опрацьовувати її на комп’ютері або застосовувати для власного планування

Висновки до розділу 4

У четвертому розділі розглянуто результати розробки Telegram-бота QeralFinanceBot для аналітики особистих фінансів із порадами. З’ясовано, що створений бот виконує основні функції, потрібні для обліку доходів і витрат, перегляду статистики, роботи з накопичувальними цілями, отримання порад та нагадувань.

Також описано практичне застосування створеного бота. QeralFinanceBot може стати в пригоді для щоденного внесення фінансових операцій, контролю витрат за категоріями, перегляду загального стану бюджету та планування накопичень. Завдяки роботі через Telegram користувач здатен швидко виконувати основні дії без встановлення окремого застосунку.

У розділі також окреслено можливі напрями подальшого вдосконалення системи. Надалі Telegram-бота можна доповнити автоматичними нагадуваннями за розкладом, лімітами витрат за категоріями, побудовою діаграм, експортом фінансових звітів, гнучкішою системою порад та розгортанням на сервері для безперервної роботи.

Таким чином, четвертий розділ засвідчив, що створений Telegram-бот є працездатним засобом базового контролю особистих фінансів і має простір для подальшого розвитку. Перелічені напрями вдосконалення не впливають на поточну працездатність системи, проте окреслюють природний шлях її подальшого розвитку.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було створено Telegram-бота QeralFinanceBot для аналітики особистих фінансів із порадами. Головна мета роботи полягала в розробці зручного програмного засобу, що дає користувачеві змогу вести облік доходів і витрат, переглядати статистику, розбирати фінансові операції та отримувати поради щодо ощадливішого керування особистим бюджетом.

У першому розділі розглянуто особливості обліку та аналізу особистих фінансів. З'ясовано, що регулярна фіксація доходів і витрат допомагає користувачеві краще розуміти будову власного бюджету, тримати під контролем необов'язкові витрати та планувати майбутні фінансові рішення. Також проаналізовано сучасні цифрові рішення для контролю особистих фінансів, зокрема Monobank, Monefy та PersonalFinTrackerBot. На підставі цього аналізу окреслено основні функціональні можливості майбутнього Telegram-бота.

У другому розділі виконано проєктування Telegram-бота. Описано логіку взаємодії користувача з ботом, визначено основні функціональні складники системи, спроєктовано базу даних та розглянуто алгоритм формування статистики й порад. Особливу увагу приділено збереженню фінансових операцій, розподілу витрат за категоріями, роботі з накопичувальними цілями та укладанню порад на основі внесених користувачем даних.

У третьому розділі втілено основні функції Telegram-бота QeralFinanceBot. Сформовано головне меню, додавання доходів і витрат, вибір категорій, перегляд статистики, історію операцій, накопичувальні цілі, поради, нагадування та видалення останньої операції. Усі фінансові дані зберігаються в базі даних, що дає змогу послуговуватися ними для подальшого аналізу. Проведене тестування підтвердило коректну роботу основних функцій бота.

У четвертому розділі розглянуто результати розробки та практичне застосування Telegram-бота. Показано, що QeralFinanceBot може слугувати для

щоденного контролю фінансів, аналізу витрат за категоріями, планування накопичень та отримання порад. Також окреслено обмеження нинішньої версії системи й можливі напрями її подальшого вдосконалення.

Отже, унаслідок виконання кваліфікаційної роботи постав працездатний Telegram-бот, який відповідає обраній темі та виконує основні завдання обліку й аналізу особистих фінансів. Створена система проста у використанні, доступна через Telegram і може прислужитися користувачам, які прагнуть краще контролювати власні доходи, витрати та накопичувальні цілі. Надалі бота можна вдосконалити, додавши автоматичні нагадування за розкладом, графіки витрат, експорт звітів, ліміти за категоріями та розгортання на сервері для безперервної роботи. Загалом досягнуті результати підтверджують, що поставлену мету кваліфікаційної роботи виконано в повному обсязі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Фінанси у смартфоні – мобільні застосунки для планування бюджету. Гаразд. URL:https://harazd.bank.gov.ua/article/finansove-planuvanna/finansovi-cili/finansi-u-smartfoni-mobilni-zastosunki-dla-planuvanna-budzetu (дата звернення: 20.04.2026).

2. Персональні фінанси: для чого вести їхній облік? MyKRP. URL: https://mykrp.com.ua/personal-financing-reasons/ (дата звернення: 21.04.2026).

3. П’ять українських додатків для контролю витрат. Рубрика. URL: https://rubryka.com/article/dlya-kontrolyu-vytrat/ (дата звернення: 21.04.2026).

4. Топ-9 застосунків для контролю особистих фінансів. WoMo. URL: https://womo.ua/top-10-dodatkiv-dlya-kontrolyu-osobistih-finansiv/ (дата звернення: 22.04.2026).

5. Five apps for budget planning. PostFinance. URL: https://www.postfinance.ch/en/blog/money-in-simple-terms/five-apps-for-budget-planning.html (дата звернення: 22.04.2026).

6. Budget Planner. Financial Consumer Agency of Canada. URL: https://itools-ioutils.fcac-acfc.gc.ca/BP-PB/budget-planner (дата звернення: 23.04.2026).

7. Best Budgeting Apps. Forbes Advisor. URL: https://www.forbes.com/advisor/banking/best-budgeting-apps/ (дата звернення: 23.04.2026).

8. Looking for mobile banking? A modern bank in your phone. Monobank. URL: https://monobank.ua/en/ (дата звернення: 24.04.2026).

9. monobank – modern digital bank. Google Play. URL: https://play.google.com/store/apps/details?id=com.ftband.mono (дата звернення: 24.04.2026).

10. monobank — digital mobile bank. App Store. URL: https://apps.apple.com/us/app/monobank-digital-mobile-bank/id1287005205 (дата

звернення: 25.04.2026).

11. Monefy | Budget & Track Your Money. Monefy. URL: https://monefy.com/ (дата звернення: 25.04.2026).

12. Monefy: Money Tracker. App Store. URL: https://apps.apple.com/ch/app/monefy-money-tracker/id1212024409 (дата звернення: 26.04.2026).

13. Monefy - Budget & Expenses app. Google Play. URL: https://play.google.com/store/apps/details?id=com.monefy.app.lite (дата звернення: 26.04.2026).

14. Money Manager Expense & Budget. Google Play. URL: https://play.google.com/store/apps/details?id=com.realbyteapps.moneymanagerfree (дата звернення: 27.04.2026).

15. Українець запустив бот в Telegram для моніторингу власних фінансів. Kosht.Media. URL: https://kosht.media/ukrainets-zapustyv-bot-v-telegram-dlia-monitorynhu-vlasnykh-finansiv/ (дата звернення: 27.04.2026).

16. Telegram-бот для ведення бюджету в Google таблиці. DOU. URL: https://dou.ua/forums/topic/36215/ (дата звернення: 28.04.2026).

17. FinTrack — Expense Tracker. Telegram. URL: https://t.me/PersonalFinTrackerBot (дата звернення: 28.04.2026).

18. Telegram Finance Bot. GitHub. URL: https://github.com/nenertiy/telegram-finance-bot (дата звернення: 29.04.2026).

19. Creating the Ultimate Finance Telegram bot using Google Sheets. Medium. URL: https://medium.com/@apes.finance/creating-the-ultimate-finance-telegram-bot-using-google-sheets-424e19e0a87e (дата звернення: 29.04.2026).

20. Telegram APIs. Telegram. URL: https://core.telegram.org/ (дата звернення: 30.04.2026).

21. Telegram Bot API. Telegram. URL: https://core.telegram.org/bots/api (дата звернення: 30.04.2026).

22. BotFather. Telegram. URL: https://t.me/BotFather (дата звернення:

01.05.2026).

23. aiogram Documentation. aiogram. URL: https://docs.aiogram.dev/ (дата звернення: 01.05.2026).

24. aiogram. PyPI. URL: https://pypi.org/project/aiogram/ (дата звернення: 02.05.2026).

25. Python 3 Documentation. Python. URL: https://docs.python.org/3/ (дата звернення: 02.05.2026).

26. venv — Creation of virtual environments. Python. URL: https://docs.python.org/3/library/venv.html (дата звернення: 03.05.2026).

27. sqlite3 — DB-API 2.0 interface for SQLite databases. Python. URL: https://docs.python.org/3/library/sqlite3.html (дата звернення: 03.05.2026).

28. SQLite Documentation. SQLite. URL: https://www.sqlite.org/docs.html (дата звернення: 04.05.2026).

29. Visual Studio Code Documentation. Microsoft. URL: https://code.visualstudio.com/docs (дата звернення: 04.05.2026).

30. Matplotlib Documentation. Matplotlib. URL: https://matplotlib.org/stable/ (дата звернення: 05.05.2026).

ДОДАТОК А ЛСТИНГИ ПРОГРАМНОГО КОДУ TEéEGRAM-БОТА

Лістинг коду main.py

```
import asyncio

from aiogram import Bot, Dispatcher



from config import BOT_TOKEN

from database import init_db, seed_categories

from handlers import start, income, expense, statistics, history, goals,
recommendations, reminders, delete_transaction

async def main():

   init_db()

   seed_categories()

   bot = Bot(token=BOT_TOKEN)

   dp = Dispatcher()

   dp.include_router(start.router)

   dp.include_router(income.router)

   dp.include_router(expense.router)

   dp.include_router(statistics.router)

   dp.include_router(history.router)

   dp.include_router(goals.router)

   dp.include_router(recommendations.router)

   dp.include_router(reminders.router)

   dp.include_router(delete_transaction.router)

   print("Бот запущено...")

   await dp.start_polling(bot)

if __name__ == "__main__":

   asyncio.run(main())
```

Лістинг коду start.py

```
from aiogram import Router
from aiogram.filters import CommandStart
from aiogram.types import Message

from database import add_user
from keyboards.menu import main_menu

router = Router()

@router.message(CommandStart())
async def start_command(message: Message):
    add_user(
        telegram_id=message.from_user.id,
        username=message.from_user.username
    )

    await message.answer(
        "Вітаю! Я QeralFinanceBot — бот для обліку та аналізу особистих фінансів.\n\n"
        "Зі мною можна додавати доходи й витрати, переглядати статистику, "
        "отримувати рекомендації та працювати з фінансовими цілями.",
        reply_markup=main_menu
    )
```

Лістинг коду menu.py

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton

main_menu = ReplyKeyboardMarkup(
    keyboard=[
        [
            KeyboardButton(text="➕ Додати дохід"),
            KeyboardButton(text="➖ Додати витрату"),
        ],
        [
            KeyboardButton(text="📊 Статистика"),
            KeyboardButton(text="📜 Історія операцій"),
        ],
        [
            KeyboardButton(text="🎯 Фінансова ціль"),
            KeyboardButton(text="💡 Рекомендації"),
        ],
        [
```

```
            KeyboardButton(text="🔔 Нагадування"),
            KeyboardButton(text="🗑 Видалити останню операцію"),
        ],
    ],
    resize_keyboard=True
)

from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton

def expense_categories_keyboard(categories):
    keyboard = []

    for category_id, name in categories:
        keyboard.append([
            InlineKeyboardButton(
                text=name,
                callback_data=f"expense_category:{category_id}"
            )
        ])

    return InlineKeyboardMarkup(inline_keyboard=keyboard)

from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton

goals_action_keyboard = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(
                text="➕ Створити нову ціль",
                callback_data="goal_create"
            )
        ],
        [
            InlineKeyboardButton(
                text="💰 Додати кошти до цілі",
                callback_data="goal_add_money"
            )
        ],
    ]
)

def goals_list_keyboard(goals):
    keyboard = []

    for goal_id, title, target_amount, current_amount, deadline in goals:
        keyboard.append([
            InlineKeyboardButton(
                text=title,
                callback_data=f"goal_select:{goal_id}"
            )
        ])

    return InlineKeyboardMarkup(inline_keyboard=keyboard)

statistics_period_keyboard = InlineKeyboardMarkup(
    inline_keyboard=[
        [
```

```
            InlineKeyboardButton(text="📆 Сьогодні", callback_data="stats:today"),
            InlineKeyboardButton(text="🗓 Цей тиждень", callback_data="stats:week"),
        ],
        [
            InlineKeyboardButton(text="🗓 Цей місяць", callback_data="stats:month"),
            InlineKeyboardButton(text="⬅ Попередній місяць",
callback_data="stats:last_month"),
        ],
    ]
)
```

Лістинг коду income.py

```
from aiogram import Router, F
from aiogram.types import Message
from aiogram.fsm.state import State, StatesGroup
from aiogram.fsm.context import FSMContext

from database import add_transaction
from keyboards.menu import main_menu

router = Router()

class IncomeState(StatesGroup):
    waiting_for_amount = State()

@router.message(F.text == "➕ Додати дохід")
async def add_income_start(message: Message, state: FSMContext):
    await message.answer("Введіть суму доходу:")
    await state.set_state(IncomeState.waiting_for_amount)

@router.message(IncomeState.waiting_for_amount)
async def add_income_amount(message: Message, state: FSMContext):
    try:
        amount = float(message.text.replace(",", "."))
    except ValueError:
        await message.answer("Будь ласка, введіть коректну суму числом.")
        return

    if amount <= 0:
        await message.answer("Сума доходу повинна бути більшою за нуль.")
        return

    success = add_transaction(
        telegram_id=message.from_user.id,
        transaction_type="income",
        amount=amount,
        description="Дохід користувача"
    )

    if success:
        await message.answer(
            f"Дохід у розмірі {amount:.2f} грн успішно додано.",
            reply_markup=main_menu
        )
    else:
```

```
        await message.answer(
            "Не вдалося зберегти дохід. Спробуйте ще раз.",
            reply_markup=main_menu
        )

    await state.clear()
```

Лістинг коду expense.py

```
from aiogram import Router, F
from aiogram.types import Message, CallbackQuery
from aiogram.fsm.state import State, StatesGroup
from aiogram.fsm.context import FSMContext
from database import add_transaction, get_categories
from keyboards.menu import main_menu, expense_categories_keyboard
router = Router()
class ExpenseState(StatesGroup):
    waiting_for_category = State()
    waiting_for_amount = State()
@router.message(F.text == "➖ Додати витрату")
async def add_expense_start(message: Message, state: FSMContext):
    categories = get_categories("expense")
    await message.answer(
        "Оберіть категорію витрати:",
        reply_markup=expense_categories_keyboard(categories)
    )
    await state.set_state(ExpenseState.waiting_for_category)
@router.callback_query(F.data.startswith("expense_category:"))
async def choose_expense_category(callback: CallbackQuery, state: FSMContext):
    category_id = int(callback.data.split(":")[1])
    await state.update_data(category_id=category_id)
    await callback.message.answer("Введіть суму витрати:")
    await callback.answer()
    await state.set_state(ExpenseState.waiting_for_amount)
@router.message(ExpenseState.waiting_for_amount)
async def add_expense_amount(message: Message, state: FSMContext):
    try:
        amount = float(message.text.replace(",", "."))
    except ValueError:
        await message.answer("Будь ласка, введіть коректну суму числом.")
        return
    if amount <= 0:
        await message.answer("Сума витрати повинна бути більшою за нуль.")
        return
    data = await state.get_data()
    category_id = data.get("category_id")

    success = add_transaction(
        telegram_id=message.from_user.id,
        transaction_type="expense",
        amount=amount,
        category_id=category_id,
        description="Витрата користувача"
    )
    if success:
        await message.answer(
```

```
            f"Витрату у розмірі {amount:.2f} грн успішно додано.",
            reply_markup=main_menu
        )
    else:
        await message.answer(
            "Не вдалося зберегти витрату. Спробуйте ще раз.",
            reply_markup=main_menu
        )
    await state.clear()
```

Лістинг коду statistics.py

```
from datetime import datetime, timedelta

from aiogram import Router, F
from aiogram.types import Message, CallbackQuery

from database import get_summary_by_period, get_expenses_by_category_by_period
from keyboards.menu import main_menu, statistics_period_keyboard

router = Router()

def get_period_dates(period: str):
    now = datetime.now()

    if period == "today":
        start = now.replace(hour=0, minute=0, second=0, microsecond=0)
        end = now

        title = "за сьогодні"

    elif period == "week":
        start = now - timedelta(days=now.weekday())
        start = start.replace(hour=0, minute=0, second=0, microsecond=0)
        end = now

        title = "за поточний тиждень"

    elif period == "month":
        start = now.replace(day=1, hour=0, minute=0, second=0, microsecond=0)
        end = now

        title = "за поточний місяць"

    elif period == "last_month":
        first_day_current_month = now.replace(day=1, hour=0, minute=0, second=0,
microsecond=0)
        last_day_previous_month = first_day_current_month - timedelta(seconds=1)
        start = last_day_previous_month.replace(day=1, hour=0, minute=0, second=0,
microsecond=0)
        end = last_day_previous_month

        title = "за попередній місяць"

    else:
        start = now.replace(day=1, hour=0, minute=0, second=0, microsecond=0)
        end = now
```

```
        title = "за поточний місяць"

    return start.isoformat(), end.isoformat(), title

@router.message(F.text == "📊 Статистика")
async def statistics_menu(message: Message):
    await message.answer(
        "Оберіть період для перегляду статистики:",
        reply_markup=statistics_period_keyboard
    )

@router.callback_query(F.data.startswith("stats:"))
async def show_statistics_by_period(callback: CallbackQuery):
    period = callback.data.split(":")[1]

    start_date, end_date, title = get_period_dates(period)

    summary = get_summary_by_period(
        telegram_id=callback.from_user.id,
        start_date=start_date,
        end_date=end_date
    )

    categories = get_expenses_by_category_by_period(
        telegram_id=callback.from_user.id,
        start_date=start_date,
        end_date=end_date
    )

    if summary is None:
        await callback.message.answer(
            "Спочатку запустіть бота командою /start.",
            reply_markup=main_menu
        )
        await callback.answer()
        return

    text = (
        f"📊 Ваша фінансова статистика {title}:\n\n"
        f"💰 Доходи: {summary['income']:.2f} грн\n"
        f"💸 Витрати: {summary['expense']:.2f} грн\n"
        f"📌 Баланс: {summary['balance']:.2f} грн\n"
    )

    if categories:
        text += "\n📂 Витрати за категоріями:\n"
        for category_name, amount in categories:
            if category_name is None:
                category_name = "Без категорії"
            text += f"• {category_name}: {amount:.2f} грн\n"
    else:
        text += "\nЗа цей період витрат за категоріями немає."

    await callback.message.answer(text, reply_markup=main_menu)
    await callback.answer()
```

Лістинг коду history.py

```
from aiogram import Router, F
from aiogram.types import Message

from database import get_last_transactions
from keyboards.menu import main_menu

router = Router()

@router.message(F.text == "📜 Історія операцій")
async def show_history(message: Message):
   transactions = get_last_transactions(message.from_user.id)

   if not transactions:
      await message.answer(
         "Історія операцій поки що порожня.",
         reply_markup=main_menu
      )
      return

   text = "📜 Останні фінансові операції:\n\n"

   for transaction_type, amount, description, created_at, category_name in
transactions:
      if transaction_type == "income":
         icon = "💰"
         operation_name = "Дохід"
      else:
         icon = "💸"
         operation_name = "Витрата"

      category_text = category_name if category_name else "Без категорії"
      date_text = created_at[:10]

      text += (
         f"{icon} {operation_name}: {amount:.2f} грн\n"
         f"📂 Категорія: {category_text}\n"
         f"📅 Дата: {date_text}\n\n"
      )

   await message.answer(text, reply_markup=main_menu)
```

Лістинг коду recommendations.py

```
import random

from aiogram import Router, F
from aiogram.types import Message

from database import get_financial_data_for_recommendations
from keyboards.menu import main_menu

router = Router()

POSITIVE_BALANCE_MESSAGES = [
    "✅ У вас є позитивний залишок коштів. Частину цієї суми можна відкласти на фінансову ціль або використати для важливих потреб.",
    "✅ Добре, цього разу доходи перевищують витрати. Це хороший момент, щоб частину залишку спрямувати на накопичення.",
    "✅ Бюджет виглядає стабільно: після витрат у вас залишилися кошти. Можна подумати про фінансову ціль або резерв.",
    "✅ Є залишок коштів — це вже хороший результат. Варто не витратити його випадково, а використати більш усвідомлено."
]

NEGATIVE_BALANCE_MESSAGES = [
    "⚠️ Ваші витрати перевищують доходи. Варто переглянути необов’язкові покупки та тимчасово зменшити зайві витрати.",
    "⚠️ За поточними даними витрати більші за доходи. Це сигнал, що бюджет потребує більш уважного контролю.",
    "⚠️ Баланс вийшов від’ємним. Спробуйте подивитися, які витрати можна скоротити найближчим часом.",
    "⚠️ Ви витратили більше, ніж отримали. Краще зупинитися й переглянути покупки, які не були обов’язковими."
]

BALANCED_MESSAGES = [
    "ℹ️ Доходи та витрати майже збалансовані. Рекомендується уважно стежити за щоденними витратами.",
    "ℹ️ Бюджет зараз приблизно на одному рівні. Варто не розслаблятися й продовжувати записувати операції.",
    "ℹ️ Ситуація виглядає нейтральною: великих перевищень немає, але й запас коштів поки невеликий.",
    "ℹ️ Доходи й витрати близькі між собою. Краще контролювати дрібні покупки, щоб не вийти за межі бюджету."
]

FOOD_MESSAGES = [
    "🍽️ Витрати на їжу вийшли помітними. Можливо, варто частіше планувати покупки заздалегідь.",
    "🍽️ Категорія їжі займає значну частину витрат. Невеликий контроль у цій сфері може дати хорошу економію.",
    "🍽️ Схоже, на їжу пішло багато коштів. Варто перевірити, чи не було зайвих покупок або частих перекусів."
]

COFFEE_MESSAGES = [
    "☕ Кава — маленька витрата тільки на перший погляд. За місяць вона може перетворитися на
```

```
непогану суму.",
    "🍩 Витрати на каву помітні. Можливо, іноді кава вдома допоможе швидше наблизитися до фінансової цілі.",
    "🍩 Якщо трохи скоротити кавові витрати, можна непомітно зекономити частину бюджету."
]

ENTERTAINMENT_MESSAGES = [
    "🎮 Розваги — це нормально, але їх краще планувати в межах бюджету.",
    "🎮 На розваги витрачено помітну суму. Варто залишити місце для відпочинку, але без шкоди для основних цілей.",
    "🎮 Якщо хочеться зекономити, категорія розваг може бути одним із перших місць для невеликого скорочення."
]

SHOPPING_MESSAGES = [
    "🛒 Покупки займають значну частину витрат. Перед новою покупкою варто подумати, чи справді вона потрібна зараз.",
    "🛒 У категорії покупок є простір для економії. Іноді відкладена покупка — це вже фінансове рішення.",
    "🛒 Схоже, покупки впливають на бюджет. Можна спробувати планувати їх за списком."
]

DONATION_MESSAGES = [
    "UA Донати — соціально корисна категорія витрат, але її також варто планувати в межах власного бюджету.",
    "UA Підтримка важливих ініціатив — це добре, але навіть такі витрати краще враховувати у фінансовому плані.",
    "UA Донати мають важливе значення, однак бот допомагає бачити їхню частку в загальному бюджеті."
]

GENERAL_CATEGORY_MESSAGES = [
    "📌 Ця категорія займає найбільшу частину витрат. Варто періодично переглядати її, щоб краще контролювати бюджет.",
    "📌 Саме ця категорія найбільше вплинула на витрати. Зверніть на неї увагу під час планування наступного періоду.",
    "📌 Тут зосереджена значна частина витрат. Можливо, саме з цієї категорії варто почати аналіз бюджету."
]

FINAL_TIPS = [
    "📍 Загальна порада: записуйте витрати регулярно. Чим більше даних буде в боті, тим точніше можна буде оцінити фінансову ситуацію.",
    "📍 Корисна звичка — вносити витрати одразу після покупки. Так статистика буде точнішою.",
    "📍 Не обов’язково економити на всьому. Головне — розуміти, куди саме йдуть гроші.",
    "📍 Регулярний облік допомагає не сварити себе за витрати, а спокійно бачити реальну картину бюджету.",
    "📍 Маленькі витрати не страшні, якщо вони під контролем. Саме для цього й потрібна статистика."
]

def get_category_advice(category_name: str):
   if category_name == "Їжа":
      return random.choice(FOOD_MESSAGES)
```

```
    if category_name == "Кава":
        return random.choice(COFFEE_MESSAGES)

    if category_name == "Розваги":
        return random.choice(ENTERTAINMENT_MESSAGES)

    if category_name == "Покупки":
        return random.choice(SHOPPING_MESSAGES)

    if category_name == "Донати":
        return random.choice(DONATION_MESSAGES)

    return random.choice(GENERAL_CATEGORY_MESSAGES)

@router.message(F.text == "💡 Рекомендації")
async def show_recommendations(message: Message):
    data = get_financial_data_for_recommendations(message.from_user.id)

    summary = data["summary"]
    categories = data["categories"]

    if summary is None:
        await message.answer(
            "Спочатку запустіть бота командою /start.",
            reply_markup=main_menu
        )
        return

    income = summary["income"]
    expense = summary["expense"]
    balance = summary["balance"]

    text = "💡 Ваші фінансові рекомендації:\n\n"

    if income == 0 and expense == 0:
        text += (
            "Поки що недостатньо даних для аналізу.\n"
            "Додайте кілька доходів і витрат, щоб я міг сформувати корисні поради."
        )
        await message.answer(text, reply_markup=main_menu)
        return

    if expense > income:
        text += random.choice(NEGATIVE_BALANCE_MESSAGES) + "\n\n"
    elif balance > 0:
        text += random.choice(POSITIVE_BALANCE_MESSAGES) + "\n\n"
    else:
        text += random.choice(BALANCED_MESSAGES) + "\n\n"

    if categories:
        top_category, top_amount = categories[0]

        if top_category is None:
            top_category = "Без категорії"

        text += (
            f"📌 Найбільше коштів витрачено на категорію «{top_category}» "
```

```
            f"— {top_amount:.2f} грн.\n"
        )

        text += get_category_advice(top_category) + "\n\n"

    text += random.choice(FINAL_TIPS)

    await message.answer(text, reply_markup=main_menu)
```

Лістинг коду reminders.py

```
import random

from aiogram import Router, F
from aiogram.types import Message

from keyboards.menu import main_menu

router = Router()

REMINDER_MESSAGES = [
    "🔔 Не забудьте записати сьогоднішні витрати. Маленькі суми теж впливають на загальний бюджет.",
    "☕ Можливо, сьогодні кава вдома — теж непоганий варіант. А зекономлені кошти можна відкласти на фінансову ціль.",
    "💰 Перед необов'язковою покупкою варто запитати себе: це справді потрібно зараз?",
    "📊 Регулярний облік витрат допомагає краще бачити, куди зникають гроші наприкінці місяця.",
    "🎯 Якщо маєте фінансову ціль, спробуйте сьогодні відкласти хоча б невелику суму.",
    "🛒 Сьогодні хороший день, щоб не купити щось зайве й залишити ці гроші для важливішої мети.",
    "UA Зекономлені кошти можна спрямувати на щось корисне: фінансову ціль, навчання, здоров'я або благодійність.",
    "✅ Один запис витрати сьогодні — менше хаосу в бюджеті завтра.",
    "🍔 Якщо витрати на їжу ростуть занадто швидко, варто трохи уважніше подивитися на покупки цього тижня.",
    "📌 Бюджет не любить сюрпризів. Краще записати витрату одразу, ніж згадувати її через тиждень.",
    "💡 Невеликі щоденні витрати здаються непомітними, але за місяць можуть перетворитися на серйозну суму.",
    "📉 Якщо сьогодні вдалося не витратити зайве — це вже маленький крок до фінансової цілі.",
]

@router.message(F.text == "🔔 Нагадування")
async def show_reminder(message: Message):
    reminder = random.choice(REMINDER_MESSAGES)

    await message.answer(
        reminder,
        reply_markup=main_menu
    )
```

Лістинг коду delete_transaction.py

```
from aiogram import Router, F
from aiogram.types import Message

from database import delete_last_transaction
from keyboards.menu import main_menu

router = Router()

@router.message(F.text == "🗑 Видалити останню операцію")
async def delete_last_operation(message: Message):
   deleted = delete_last_transaction(message.from_user.id)

   if deleted is None:
      await message.answer(
         "Поки що немає операцій, які можна видалити.",
         reply_markup=main_menu
      )
      return

   if deleted["type"] == "income":
      operation_name = "дохід"
      icon = "💰"
   else:
      operation_name = "витрату"
      icon = "💸"

   category = deleted["category"] if deleted["category"] else "без категорії"

   await message.answer(
      f"🗑 Останню операцію видалено.\n\n"
      f"{icon} Тип: {operation_name}\n"
      f"Сума: {deleted['amount']:.2f} грн\n"
      f"Категорія: {category}\n"
      f"Дата: {deleted['created_at'][:10]}",
      reply_markup=main_menu
   )
```

Лістинг коду server.js

```
import sqlite3
from datetime import datetime

DB_NAME = "finance_bot.db"

def get_connection():
   return sqlite3.connect(DB_NAME)

def init_db():
   conn = get_connection()
   cursor = conn.cursor()

   cursor.execute("""
      CREATE TABLE IF NOT EXISTS users (
         id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
            telegram_id INTEGER UNIQUE NOT NULL,
            username TEXT,
            created_at TEXT NOT NULL
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS categories (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL,
            type TEXT NOT NULL,
            UNIQUE(name, type)
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS transactions (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            category_id INTEGER,
            type TEXT NOT NULL,
            amount REAL NOT NULL,
            description TEXT,
            created_at TEXT NOT NULL,
            FOREIGN KEY (user_id) REFERENCES users (id),
            FOREIGN KEY (category_id) REFERENCES categories (id)
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS goals (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            title TEXT NOT NULL,
            target_amount REAL NOT NULL,
            current_amount REAL DEFAULT 0,
            deadline TEXT,
            created_at TEXT NOT NULL,
            FOREIGN KEY (user_id) REFERENCES users (id)
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS reminders (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            text TEXT NOT NULL,
            remind_time TEXT,
            is_active INTEGER DEFAULT 1,
            FOREIGN KEY (user_id) REFERENCES users (id)
        )
    """)

    conn.commit()
    conn.close()

def add_user(telegram_id: int, username: str | None):
```

```
    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        INSERT OR IGNORE INTO users (telegram_id, username, created_at)
        VALUES (?, ?, ?)
    """, (telegram_id, username, datetime.now().isoformat()))

    conn.commit()
    conn.close()

def get_user_id(telegram_id: int):
    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        SELECT id FROM users WHERE telegram_id = ?
    """, (telegram_id,))

    result = cursor.fetchone()
    conn.close()

    if result:
        return result[0]

    return None

def add_transaction(
    telegram_id: int,
    transaction_type: str,
    amount: float,
    category_id=None,
    description: str | None = None
):
    user_id = get_user_id(telegram_id)

    if user_id is None:
        return False

    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        INSERT INTO transactions (
            user_id, category_id, type, amount, description, created_at
        )
        VALUES (?, ?, ?, ?, ?, ?)
    """, (
        user_id,
        category_id,
        transaction_type,
        amount,
        description,
        datetime.now().isoformat()
    ))

    conn.commit()
```

```
    conn.close()

    return True

def seed_categories():
    conn = get_connection()
    cursor = conn.cursor()

    categories = [
        ("Їжа", "expense"),
        ("Транспорт", "expense"),
        ("Кава", "expense"),
        ("Розваги", "expense"),
        ("Покупки", "expense"),
        ("Здоров'я", "expense"),
        ("Навчання", "expense"),
        ("Донати", "expense"),
        ("Інше", "expense"),
    ]

    for name, category_type in categories:
        cursor.execute("""
            INSERT OR IGNORE INTO categories (name, type)
            VALUES (?, ?)
        """, (name, category_type))

    conn.commit()
    conn.close()

def get_categories(category_type: str = "expense"):
    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        SELECT MIN(id), name
        FROM categories
        WHERE type = ?
        GROUP BY name
        ORDER BY MIN(id)
    """, (category_type,))

    categories = cursor.fetchall()
    conn.close()

    return categories

def get_summary(telegram_id: int):
    user_id = get_user_id(telegram_id)

    if user_id is None:
        return None

    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        SELECT
```

```
            COALESCE(SUM(CASE WHEN type = 'income' THEN amount ELSE 0 END), 0),
            COALESCE(SUM(CASE WHEN type = 'expense' THEN amount ELSE 0 END), 0)
         FROM transactions
         WHERE user_id = ?
      """, (user_id,))

      income, expense = cursor.fetchone()
      conn.close()

      return {
         "income": income,
         "expense": expense,
         "balance": income - expense
      }

   def get_expenses_by_category(telegram_id: int):
      user_id = get_user_id(telegram_id)

      if user_id is None:
         return []

      conn = get_connection()
      cursor = conn.cursor()

      cursor.execute("""
         SELECT categories.name, SUM(transactions.amount)
         FROM transactions
         LEFT JOIN categories ON transactions.category_id = categories.id
         WHERE transactions.user_id = ? AND transactions.type = 'expense'
         GROUP BY categories.name
         ORDER BY SUM(transactions.amount) DESC
      """, (user_id,))

      result = cursor.fetchall()
      conn.close()

      return result

   def get_last_transactions(telegram_id: int, limit: int = 10):
      user_id = get_user_id(telegram_id)

      if user_id is None:
         return []

      conn = get_connection()
      cursor = conn.cursor()

      cursor.execute("""
         SELECT
            transactions.type,
            transactions.amount,
            transactions.description,
            transactions.created_at,
            categories.name
         FROM transactions
         LEFT JOIN categories ON transactions.category_id = categories.id
         WHERE transactions.user_id = ?
```

```
      ORDER BY transactions.created_at DESC
      LIMIT ?
   """, (user_id, limit))

   result = cursor.fetchall()
   conn.close()

   return result

def add_goal(
   telegram_id: int,
   title: str,
   target_amount: float,
   deadline: str | None = None
):
   user_id = get_user_id(telegram_id)

   if user_id is None:
      return False

   conn = get_connection()
   cursor = conn.cursor()

   cursor.execute("""
      INSERT INTO goals (
         user_id, title, target_amount, current_amount, deadline, created_at
      )
      VALUES (?, ?, ?, ?, ?, ?)
   """, (
      user_id,
      title,
      target_amount,
      0,
      deadline,
      datetime.now().isoformat()
   ))

   conn.commit()
   conn.close()

   return True

def get_goals(telegram_id: int):
   user_id = get_user_id(telegram_id)

   if user_id is None:
      return []

   conn = get_connection()
   cursor = conn.cursor()

   cursor.execute("""
      SELECT title, target_amount, current_amount, deadline
      FROM goals
      WHERE user_id = ?
      ORDER BY created_at DESC
   """, (user_id,))
```

```
    result = cursor.fetchall()
    conn.close()

    return result

def get_financial_data_for_recommendations(telegram_id: int):
    summary = get_summary(telegram_id)
    categories = get_expenses_by_category(telegram_id)

    return {
        "summary": summary,
        "categories": categories
    }

def get_goals_with_id(telegram_id: int):
    user_id = get_user_id(telegram_id)

    if user_id is None:
        return []

    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        SELECT id, title, target_amount, current_amount, deadline
        FROM goals
        WHERE user_id = ?
        ORDER BY created_at DESC
    """, (user_id,))

    result = cursor.fetchall()
    conn.close()

    return result

def add_money_to_goal(telegram_id: int, goal_id: int, amount: float):
    user_id = get_user_id(telegram_id)

    if user_id is None:
        return None

    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        SELECT title, target_amount, current_amount
        FROM goals
        WHERE id = ? AND user_id = ?
    """, (goal_id, user_id))

    goal = cursor.fetchone()

    if goal is None:
        conn.close()
        return None

    title, target_amount, current_amount = goal
```

```
    new_current_amount = current_amount + amount

    cursor.execute("""
        UPDATE goals
        SET current_amount = ?
        WHERE id = ? AND user_id = ?
    """, (new_current_amount, goal_id, user_id))

    conn.commit()
    conn.close()

    return {
        "title": title,
        "target_amount": target_amount,
        "current_amount": new_current_amount,
        "remaining": max(target_amount - new_current_amount, 0)
    }

from datetime import datetime, timedelta

def get_summary_by_period(telegram_id: int, start_date: str, end_date:
str):
    user_id = get_user_id(telegram_id)

    if user_id is None:
        return None

    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        SELECT
            COALESCE(SUM(CASE WHEN type = 'income' THEN amount ELSE 0 END), 0),
            COALESCE(SUM(CASE WHEN type = 'expense' THEN amount ELSE 0 END), 0)
        FROM transactions
        WHERE user_id = ?
        AND created_at >= ?
        AND created_at <= ?
    """, (user_id, start_date, end_date))

    income, expense = cursor.fetchone()
    conn.close()

    return {
        "income": income,
        "expense": expense,
        "balance": income - expense
    }

def get_expenses_by_category_by_period(telegram_id: int, start_date: str,
end_date: str):
    user_id = get_user_id(telegram_id)

    if user_id is None:
        return []

    conn = get_connection()
```

```
    cursor = conn.cursor()

    cursor.execute("""
        SELECT categories.name, SUM(transactions.amount)
        FROM transactions
        LEFT JOIN categories ON transactions.category_id = categories.id
        WHERE transactions.user_id = ?
        AND transactions.type = 'expense'
        AND transactions.created_at >= ?
        AND transactions.created_at <= ?
        GROUP BY categories.name
        ORDER BY SUM(transactions.amount) DESC
    """, (user_id, start_date, end_date))

    result = cursor.fetchall()
    conn.close()

    return result

def delete_last_transaction(telegram_id: int):
    user_id = get_user_id(telegram_id)

    if user_id is None:
        return None

    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        SELECT
            transactions.id,
            transactions.type,
            transactions.amount,
            transactions.created_at,
            categories.name
        FROM transactions
        LEFT JOIN categories ON transactions.category_id = categories.id
        WHERE transactions.user_id = ?
        ORDER BY transactions.created_at DESC
        LIMIT 1
    """, (user_id,))

    transaction = cursor.fetchone()

    if transaction is None:
        conn.close()
        return None

    transaction_id, transaction_type, amount, created_at, category_name =
transaction

    cursor.execute("""
        DELETE FROM transactions
        WHERE id = ? AND user_id = ?
    """, (transaction_id, user_id))

    conn.commit()
```

```
        conn.close()

        return {
            "type": transaction_type,
            "amount": amount,
            "created_at": created_at,
            "category": category_name
        }
```

Лістинг коду Speedometer.jsx

```
from datetime import datetime
from aiogram import Router, F
from aiogram.types import Message, CallbackQuery
from aiogram.fsm.state import State, StatesGroup
from aiogram.fsm.context import FSMContext

from database import add_goal, get_goals_with_id, add_money_to_goal
from keyboards.menu import main_menu, goals_action_keyboard, goals_list_keyboard

router = Router()

class GoalState(StatesGroup):
    waiting_for_title = State()
    waiting_for_amount = State()
    waiting_for_deadline = State()
    waiting_for_add_amount = State()

@router.message(F.text == "🎯 Фінансова ціль")
async def goals_menu(message: Message, state: FSMContext):
    await state.clear()

    goals = get_goals_with_id(message.from_user.id)

    text = "🎯 Фінансові цілі\n\n"

    if goals:
        text += "Ваші поточні цілі:\n\n"

        for goal_id, title, target_amount, current_amount, deadline in goals:
            remaining = max(target_amount - current_amount, 0)

            text += (
                f"📌 {title}\n"
                f"Потрібно: {target_amount:.2f} грн\n"
                f"Накопичено: {current_amount:.2f} грн\n"
                f"Залишилось: {remaining:.2f} грн\n"
            )

            if deadline:
```

```
            text += f"Бажана дата: {deadline}\n"

        if remaining == 0:
            text += "✅ Ціль уже досягнута!\n"

        text += "\n"
    else:
        text += "Поки що у вас немає створених фінансових цілей.\n\n"

    text += "Оберіть дію:"

    await message.answer(text, reply_markup=goals_action_keyboard)

@router.callback_query(F.data == "goal_create")
async def create_goal_start(callback: CallbackQuery, state: FSMContext):
    await callback.message.answer("Введіть назву фінансової цілі:")
    await callback.answer()
    await state.set_state(GoalState.waiting_for_title)

@router.message(GoalState.waiting_for_title)
async def goal_title(message: Message, state: FSMContext):
    title = message.text.strip()

    if len(title) < 2:
        await message.answer("Назва цілі занадто коротка. Введіть назву ще раз.")
        return

    await state.update_data(title=title)
    await message.answer("Введіть суму, яку потрібно накопичити:")
    await state.set_state(GoalState.waiting_for_amount)

@router.message(GoalState.waiting_for_amount)
async def goal_amount(message: Message, state: FSMContext):
    try:
        amount = float(message.text.replace(",", "."))
    except ValueError:
        await message.answer("Будь ласка, введіть коректну суму числом.")
        return

    if amount <= 0:
        await message.answer("Сума повинна бути більшою за нуль.")
        return

    await state.update_data(target_amount=amount)
    await message.answer(
        "Введіть бажану дату досягнення цілі у форматі ДД.ММ.РРРР.\n"
        "Наприклад: 01.12.2026"
    )
    await state.set_state(GoalState.waiting_for_deadline)
```

```
@router.message(GoalState.waiting_for_deadline)
async def goal_deadline(message: Message, state: FSMContext):
    deadline_text = message.text.strip()

    try:
        deadline_date = datetime.strptime(deadline_text, "%d.%m.%Y")
    except ValueError:
        await message.answer("Невірний формат дати. Введіть дату у форматі ДД.ММ.РРРР.")
        return

    today = datetime.now()
    days_left = (deadline_date - today).days

    if days_left <= 0:
        await message.answer("Дата має бути в майбутньому. Введіть іншу дату.")
        return

    data = await state.get_data()

    title = data["title"]
    target_amount = data["target_amount"]

    success = add_goal(
        telegram_id=message.from_user.id,
        title=title,
        target_amount=target_amount,
        deadline=deadline_text
    )

    if not success:
        await message.answer(
            "Не вдалося зберегти фінансову ціль. Спробуйте ще раз.",
            reply_markup=main_menu
        )
        await state.clear()
        return

    monthly_amount = target_amount / max(days_left / 30, 1)
    weekly_amount = target_amount / max(days_left / 7, 1)
    daily_amount = target_amount / days_left

    await message.answer(
        f"🎯 Фінансову ціль створено!\n\n"
        f"Ціль: {title}\n"
        f"Потрібно накопичити: {target_amount:.2f} грн\n"
        f"Дата: {deadline_text}\n\n"
        f"Щоб досягти цілі, потрібно приблизно відкладати:\n"
        f"• {daily_amount:.2f} грн щодня\n"
```

```
        f"• {weekly_amount:.2f} грн щотижня\n"
        f"• {monthly_amount:.2f} грн щомісяця",
        reply_markup=main_menu
    )

    await state.clear()

@router.callback_query(F.data == "goal_add_money")
async def add_money_start(callback: CallbackQuery, state: FSMContext):
    goals = get_goals_with_id(callback.from_user.id)

    if not goals:
        await callback.message.answer(
            "У вас ще немає фінансових цілей. Спочатку створіть ціль.",
            reply_markup=goals_action_keyboard
        )
        await callback.answer()
        return

    await callback.message.answer(
        "Оберіть ціль, до якої хочете додати кошти:",
        reply_markup=goals_list_keyboard(goals)
    )

    await callback.answer()

@router.callback_query(F.data.startswith("goal_select:"))
async def select_goal_for_money(callback: CallbackQuery, state: FSMContext):
    goal_id = int(callback.data.split(":")[1])

    await state.update_data(goal_id=goal_id)

    await callback.message.answer("Введіть суму, яку хочете додати до цієї цілі:")
    await callback.answer()

    await state.set_state(GoalState.waiting_for_add_amount)

@router.message(GoalState.waiting_for_add_amount)
async def add_money_amount(message: Message, state: FSMContext):
    try:
        amount = float(message.text.replace(",", "."))
    except ValueError:
        await message.answer("Будь ласка, введіть коректну суму числом.")
        return

    if amount <= 0:
        await message.answer("Сума повинна бути більшою за нуль.")
        return

    data = await state.get_data()
```

```
goal_id = data.get("goal_id")

result = add_money_to_goal(
    telegram_id=message.from_user.id,
    goal_id=goal_id,
    amount=amount
)

if result is None:
    await message.answer(
        "Не вдалося знайти ціль. Спробуйте ще раз.",
        reply_markup=main_menu
    )
    await state.clear()
    return

text = (
    f"🎯 До цілі «{result['title']}» додано {amount:.2f} грн.\n\n"
    f"Накопичено: {result['current_amount']:.2f} грн "
    f"з {result['target_amount']:.2f} грн\n"
    f"Залишилось: {result['remaining']:.2f} грн"
)

if result["remaining"] == 0:
    text += "\n\n✅ Вітаю! Ціль досягнута."

await message.answer(text, reply_markup=main_menu)
await state.clear()
```