МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «**РОЗРОБКА ПЕРСОНАЛЬНОГО ФІНАНСОВОГО ДОДАТКА З МОЖЛИВІСТЮ АКУМУЛЯЦІЇ Й ОПТИМІЗАЦІЇ ЗАОЩАДЖЕНЬ**»

Виконала: студентка 2 курсу, групи 8.1212-іпз-1
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітньої програми Інженерія програмного забезпечення
(назва освітньої програми)

А.О. Зінченко
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 Інженерія програмного забезпечення
(шифр і назва)

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Зінченко Алісі Олександрівні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка персонального фінансового додатка з можливістю акумуляції й оптимізації заощаджень

керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 27.11.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Збір даних.
3. Реалізація.
4. Перевірка працездатності програмного продукту.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 03.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.05.2023	
2.	Збір вихідних даних.	15.05.2023	
3.	Обробка методичних та теоретичних	05.06.2023	
	джерел.		
4.	Розробка першого та другого розділу.	28.08.2023	
5.	Розробка третього розділу.	13.11.2023	
6.	Оформлення та нормоконтроль	20.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	14.12.2023	

Студент _____ А.О. Зінченко
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка персонального фінансового додатка з можливістю акумуляції й оптимізації заощаджень»: 58 с., 23 рис., 4 табл., 15 джерел, 2 додатки.

БЮДЖЕТУВАННЯ, ЗАОЩАДЖЕННЯ, ОПТИМІЗАЦІЯ, ФІНАНСОВА ОСВІТА, PYTHON, TELEGRAM.

Об’єктом дослідження є персональний фінансовий додаток з можливістю акумуляції та оптимізації заощаджень.

Мета роботи – розробка персонального фінансового бота для платформи Telegram.

Метод дослідження – аналітичний.

Кваліфікаційна робота нараховує 3 розділи. У першому розділі викладено методологічні аспекти дослідження, зокрема обґрунтування актуальності теми, огляд сучасних підходів та інструментів управління фінансами, теоретичні основи фінансового управління з фокусом на доходах, витратах та плануванні значних фінансових подій. Докладно розглянуто план розробки моделі з урахуванням її властивостей, а також роль статистики у процесі дослідження. У другому розділі розглянуто реалізацію програмного забезпечення, включаючи опис інструментів розробки, таких як мова програмування Python, програмне середовище Replit, сервіс моніторингу UptimeRobot та переваги використання месенджера Telegram. Також подано архітектуру додатка. У третьому розділі розглянуто експериментальне вивчення фінансового бота, включаючи апробацію додатка та рекомендації щодо подальших досліджень.

SUMMARY

Master’s qualifying paper «Development of a Personal Financial Application with the Possibility of Accumulation and Optimization of Savings»: 58 pages, 23 figures, 4 tables, 15 references, 2 supplements.

BUDGETING, SAVINGS, OPTIMIZATION, FINANCIAL EDUCATION, PYTHON, TELEGRAM.

Object of the study is a personal financial application designed for the accumulation and optimization of savings.

Aim of the study is to develop a personal financial bot for the Telegram platform.

Research method is analytical.

The qualifying work consists of 3 sections. The first section delves into the methodological aspects of the research, including the justification of the topic's relevance, a review of contemporary approaches and tools in financial management, and theoretical foundations of financial management with a focus on incomes, expenditures, and planning significant financial events. The development plan of the model, considering its properties, is thoroughly examined, along with the role of statistics in the research process.

In the second section, the implementation of the software is discussed, encompassing a description of development tools such as the Python programming language, the Replit programming environment, the UptimeRobot monitoring service, and the advantages of using the Telegram messenger. The architecture of the application is also presented. The third section explores the experimental study of the financial bot, including the application's testing and recommendations for further research.

ЗМІСТ

ВСТУП

Людство переживає стрімкий розвиток інформаційних технологій, що крок за кроком вносять глибокі зміни до підходів фінансового менеджменту. Ефективне бюджетування є ключовим для забезпечення економічної гармонії серед всіх шарів суспільства. В умовах цифрової трансформації і глобальної економічної нестабільності, необхідність вдосконалення інструментів та технологій для фінансового управління стає більш актуальною, ніж будь-коли раніше. Кваліфікаційну роботу спрямовано на розробку ефективного додатка, який відповідатиме сучасним потребам користувачів у галузі організації фінансів.

Завданням дослідження визначено розробку та впровадження інноваційного додатка, який відповідатиме сучасним потребам користувачів у сфері бюджетування. Розроблений інструмент буде спрямовано на полегшення процесу ведення грошового обліку, визначення і досягнення фінансових цілей, а також готовий ресурс для розвитку фінансової освіти людей будь-якого віку.

В процесі дослідження буде розглянуто актуальні проблеми економічних стратегій, основні вимоги до функціонування застосунків, а також різнобічно досліджено підходи до їх створення. Результатом стане інноваційний застосунок, спрямований на полегшення ведення грошового обліку, визначення та досягнення поставлених цілей, а також розроблений ресурс для поліпшення фінансової освіти.

Додаток буде реалізовано в якості бота за допомогою мови програмування Python та сервісу онлайн моніторингу UptimeRobot на хостингу Replit. Застосунок надаватиме як можливість виконання стандартних математичних розрахунків для аналізу витрат, так і матиме розділ для підвищення самосвідомості користувачів. Зокрема, користувачі зможуть додавати нові категорії витрат, контролювати їх, а також отримувати доступ до навчального ресурсу з фінансової грамотності. В боті буде забезпечено можливість збереження даних користувача до файлу зручного

формату.

Особливої уваги заслуговує розділ про фінансову освіту, що надаватиме доступ до корисних матеріалів та статей з фінансової грамотності, котрі сприятимуть саморозвитку та кращому орієнтуванню в логіці розподілення бюджету.

При формуванні базового функціоналу додатка виокремлено наступні завдання:

– розробка інтерактивного інтерфейсу для зручного керування витратами;
– розробка алгоритмів для обробки даних та їх візуалізація у доступній користувачу формі;
– забезпечення безпеки та конфіденційності даних користувача;
– підвищення фінансової грамотності користувачів через самостійно розроблений навчальний ресурс.

В першому розділі кваліфікаційної роботи обґрунтовано актуальність теми та проведено огляд сучасних підходів та інструментів управління фінансами. Також розглянуті теоретичні основи фінансового управління, зокрема питання доходів, витрат та планування значних фінансових подій [1]. Представлено план розробки моделі з урахуванням її властивостей та визначено роль статистики у дослідженні.

У другому розділі роботи описана реалізація програмного забезпечення. Надано детальний огляд інструментів розробки, таких як мова програмування Python, програмне середовище Replit, сервіс моніторингу UptimeRobot та переваги використання месенджера Telegram. Також розглянута архітектура додатка.

В третьому розділі подано результати експериментального вивчення фінансового бота. Описана апробація додатка та надані рекомендації щодо подальших досліджень у цьому напрямку.

Заключні розділи роботи включають висновки, перелік посилань та додатки,

які доповнюють та підтверджують проведене дослідження.

Підсумовуючи, реалізація додатка сприятиме систематичному розвитку фінансової самодисципліни користувачів. Інтуїтивний інтерфейс зробить його ефективним інструментом для контролю та управління особистими фінансами, а доступність забезпечить зручність та легкість використання.

1 МЕТОДОЛОГІЧНІ АСПЕКТИ ДОСЛІДЖЕННЯ

В огляді обґрунтовується актуальність теми дослідження, виконується аналіз сучасних підходів та інструментів управління фінансами. Також висвітлюються теоретичні основи фінансового управління, з фокусом на вивченні доходів, витрат, плануванні фінансових подій та ролі статистики. Наприкінці розділу спроєктовано план розробки моделі.

1.1 Обґрунтування актуальності теми

В умовах глобалізації і цифрової трансформації, зростає проблема комплексності фінансових питань, що стосуються кожного з нас. Структурування отриманих коштів, раціональне планування витрат, накопичення та оптимізація заощаджень назавжди залишаться складовою буденності. Навіть невеликі фінансові помилки можуть мати серйозні наслідки, тому необхідно розвивати інструменти, які допоможуть кожному користувачеві ефективно розподіляти свої фінанси [5].

Однак, не зважаючи на важливість фінансового планування, більша частина суспільства залишається пасивною в бюджетуванні не лише через недолік зручних інструментів для моніторингу грошових витрат, а й за нестачі відповідних знань, чи взагалі не підозрюють про необхідність наявності розуміння хоча б базису в цьому напрямі. Ситуація надмірно загострилась в результаті інфляції, кіберзлочинів та несподіваних труднощів, що можуть спіткати будь-якого громадянина. Саме тому, тема розробки фінансового додатка є дуже актуальною.

Детальніше, важливість досліду підкреслюється наступними факторами:

– людям необхідно неперервно поліпшувати навички прогнозування потреб

заради уникнення непередбачуваних витрат, банкрутства, а згодом й заборгованостей;

– навіть невеликі фінансові помилки можуть призвести до серйозних наслідків, тому «кишенькові помічники» стають незамінними для досягнення фінансового благополуччя;
– інвестиції, податки й страхування стають дедалі складнішими за структурою, тому люди повинні стати більш освіченими щодо цих та інших питань;
– планування наперед забезпечує фінансову безпеку в майбутньому;
– зростаюча нестабільність світових фінансових ринків, зміни відсоткових ставок та курсів валют.

У світлі цих факторів, розробка фінансового додатка є відповіддю на реальні потреби сучасного суспільства у сфері фінансового управління. Споживачі шукають зручні і доступні інструменти, які допоможуть їм краще розуміти та контролювати свої фінанси. Мобільні додатки вплелись до буденності і цим надали мільйони можливостей покращити якість життя [3].

Швидкозростаюча популярність мобільних додатків обумовлена багатьма чинниками, серед яких:

– зручність і доступність;
– багатий функціонал;
– заощадження часу.

Мобільні додатки дозволяють легко та швидко відстежувати стан балансу без необхідності відвідувати банк або використовувати складні програми за допомогою комп'ютера. Ця доступність й збільшує кількість користувачів. До того ж, функціонал застосунків постійно підтримується і розвивається. Додатки дозволяють автоматизувати безліч процесів, що значно заощаджує час [3].

Реалізація достатньо зручного портативного помічника вимагає наявності конкурентоспроможних функцій для задоволення потреб найбільш вимогливих

користувачів. З роками люди стають все більш свідомими. Іншими словами, розуміють важливість фінансової грамотності та її вплив на якість життя. Глобальні фінансові кризи підштовхують людей вдосконалювати свої навички. Самоосвіта стає провідним трендом. Неможливо не відзначити ріст інтересу до інвестицій та ринків акцій. Це перетворює багатьох на активних інвесторів, котрі мають бажання оптимізувати прибутковість свого захоплення. Кожна людина має унікальні фінансові цілі, потреби та обставини. Наразі, готові фінансові додатки можуть не задовольняти індивідуальні вимоги кожного користувача. Тому розробка фінансового додатка, який здатен адаптуватися до унікальних потреб будь-якого користувача, наділить змогою створювати персоналізовані фінансові плани та рекомендації, що відповідають конкретним фінансовим цілям і обставинам.

Отже, перераховані тенденції свідчать про важливість розробки фінансових інструментів, які допоможуть задовольнити потреби як вимогливих користувачів, так і новачків.

До огляду слід включити відомості про цифрову безпеку та конфіденційність даних. Високий рівень кіберзлочинності спровокував зацікавленість безпекою в інтернеті повсюдно, оскільки з розвитком технологій дані стали використовуватись для шахрайства та злочинів. Для успішності запуску фінансового додатка дуже важливо заробити довіру користувачів, гарантуючи їм надійний захист їх особистих даних.

Наостанок, завдяки прогресу в галузі штучного інтелекту та аналітики даних, відкриваються нові можливості для розробки інноваційних фінансових додатків. Використання алгоритмів машинного навчання та штучного інтелекту дозволить персональному фінансовому додатка навчатися на основі поведінки користувача, аналізувати дані та надавати індивідуалізовані поради щодо оптимізації заощаджень.

У підсумку, актуальність розробки фінансового бота полягає в наданні

зручного та доступного інтерфейсу й функціоналу для контролю за витратами, який допоможе уникнути можливих фінансових труднощів.

1.2 Огляд сучасних підходів та інструментів управління фінансами

Тут можна виокремити декілька ключових напрямків, які існують на ринку.

На сьогоднішній день, електронні гаманці стали невід'ємною частиною повсякденності більшості громадян. Такі гаманці дозволяють зручно керувати грошовими потоками та виконувати безконтактні платежі. Зазвичай ці додатки пропонують користувачам інтерактивні інтерфейси для аналізу фінансів та генерації звітів. Вони стали надзвичайно популярними завдяки легкості використання та доступності. Завдяки ним контролювати витрати можна в будь-який час та в будь-якому місці. Користувачі мають змогу переглядати баланс рахунків та відстежувати транзакції. Наступний список надає інформацію про найбільш найбільш поширені гаманці, такі як [2]:

– PayPal – один з найбільш поширених, за допомогою якого можна виконувати електронні платежі, отримувати гроші та здійснювати перекази в різних валютах;
– Apple Pay – гаманець, що дозволяє здійснювати безконтактні платежі за допомогою пристроїв: iPhone та Apple Watch;
– Google Pay – також дозволяє користувачам виконувати безконтактні платежі, здійснювати перекази, а також додавати пластикові картки для зручності оплати;
– Samsung Pay – працює лише на смартфонах Samsung і дозволяє здійснювати безконтактні платежі через NFC та MST технології;
– Venmo – використовується для особистих платежів та розділення витрат між друзями, де можна коментувати транзакції;

– Cash App – дозволяє виконувати платежі та отримувати гроші, а також інвестувати в акції та криптовалюти.

Слід виділити фінансові порадники, блоги та ресурси для фінансової освіти. Вони надають користувачам можливість отримувати поради щодо інвестування, планування пенсії, керування боргами та багато іншого. Більшість ресурсів надають безкоштовний доступ до корисних матеріалів і статей, які сприяють підвищенню рівня фінансової грамотності. Ці ресурси допомагають користувачам розуміти принципи інвестування, розподілення ризиків, планування бюджету та інші аспекти фінансового управління. Проте зазвичай користувачі вимушені переривати безліч вебсторінок в пошуках цінної та актуальної інформації, що дуже заторможує процес та не завжди гарантує якість результату.

Банки надають своїм клієнтам можливість використовувати цифровий банкінг для зручного керування коштами. Такі сервіси забезпечують високий рівень безпеки і конфіденційності. За допомогою цифрового банкінгу можна контролювати свої рахунки, здійснювати оплату, переказувати гроші, інвестувати та вести облік транзакцій у реальному часі. Банки надають клієнтам можливість виконувати ці операції з будь-якого пристрою, що в свою чергу перетворює процес у просту для розуміння послідовність дій.

Нарешті, можна перелічити найбільш популярні додатки для ведення обліку прибутків та витрат [3]:

– Monefy – дозволяє швидко вводити та класифікувати витрати, використовуючи категорії та теги;
– Spendee – надає можливість відстежувати доходи та витрати, а також створювати бюджети на категорії витрат, аналізує фінансову активність за допомогою звітів та графіків;
– Money Manager Expense & Budget – те ж, що і Spendee;
– Wallet – дозволяє створювати групи витрат, керувати банківськими рахунками та відслідковувати статистику за допомогою звітів;

– Family Budget – дозволяє кільком користувачам вести облік витрат, створювати спільні бюджети та координувати фінансові рішення в рамках сім'ї. Додаток надає можливість ведення окремих рахунків для кожного члена сім'ї та відслідковування загального бюджету.

Усі ці інструменти вже існують, і їх функціонал поступово розширюються. Розробка універсального фінансового додатка, який поєднуватиме зручний інтерфейс, алгоритми обробки даних, безпеку та доступ до освітніх ресурсів, відповідає сучасним вимогам може стати цінним інструментом для користувачів.

1.3 Теоретичні основи фінансового управління

У рамках дослідження теми фінансового освіти та її впливу на якість життя, необхідно розглянути теоретичні основи, організаційні аспекти та фактори, які визначають ефективність фінансового управління.

1.3.1 Доходи та витрати

Доходи визначають економічну самостійність та соціальний добробут особи. Необхідність доходів визначається багатогранною функцією, що включає забезпечення базових потреб, економічну самостійність та забезпечення стабільного майбутнього. Аналізуючи дану інформацію, слід звертати увагу на джерела прибутків та оцінювати ризики. Величина доходів може змінюватися під впливом різних факторів, таких як зміни на ринку праці, інфляція, ріст економіки, або надходження додаткових джерел витрат [1].

Для зручності наведено класифікацію прибутків у формі таблиці (див. табл. 1.1).

Таблиця 1.1 – Класифікація прибутків

Вид	За джерелами	За стабільністю	За ризиками
Критерії	Заробітна плата	Стабільні: регульовані угодами та контрактами	Низькі: стабільність робочого місця та системи виплат
	Пасивний дохід, інвестиції	Нестабільні: залежать від ринкових коливань	Високі: економічні фактори, можливі збитки

Прогнозування грошових надходжень є стратегічним елементом ефективного управління фінансами. Одинаки швидко адаптуються до змін, в той час як особи із сімейним статусом можуть мати фінансовий тиск через сімейні обов'язки. Тому слід розрізняти такі поняття як «короткі» та «довгі» гроші. Перші, «короткі» гроші, витрачаються на поточні потреби, тоді як «довгі» – використовуються для інвестицій та формування запасів на майбутнє. Такий розподіл допомагає забезпечити баланс між миттєвими потребами та стратегічними фінансовими цілями, забезпечуючи ефективне управління ресурсами та стійке економічне положення в довгостроковій перспективі [1].

Витрати – це грошові ресурси, які витрачаються для задоволення певних потреб. Процес виникнення витрат обумовлений різноманітними причинами, такими як основні потреби, соціокультурні впливи, технологічні зміни та особисті уподобання [1].

Наступна таблиця відображає основні критерії класифікації витрат, приклади для кожного критерію та важливість цих класифікацій у контексті управління бюджетом (див. табл. 1.2).

Таблиця 1.2 – Класифікація витрат

Вид	За призначенням	За часовим інтервалом	За характером
Критерії	Основні: для забезпечення базового рівня життя	Постійні: регулярні виплати	Обов’язкові: для забезпечення базового рівня життя
	Додаткові: розваги, подарунки	Тимчасові: святкові витрати	Вибіркові: адаптовані

1.3.2 Планування фінансових подій та роль статистики

Персональний бюджет – це систематизований план витрат та доходів особи на певний період.

Основним його призначенням є забезпечення фінансової стабільності та досягнення поставлених фінансових цілей. Визначні події можуть значно погіршити стан бюджету, і саме тому виникає необхідність розробки універсальної стратегії для розподілення фінансових ресурсів [1].

Керування бюджетом – це процес максимізації чистого прибутку особи шляхом оптимізації витрат та інших факторів, що впливають на фінансові показники. Необхідно зберігати баланс між збільшенням прибутку та забезпеченням стійкості фінансового стану заради уникнення фінансових ризиків.

Наразі існує декілька перевірених стратегій розподілення витрат, котрі будуть подані нижче в таблицях [4] (див. табл. 1.3, 1.4).

Правило 50/30/20 – це фінансова стратегія, де 50% доходів виділяються на основні потреби, 30% на особисті потреби та розваги, і 20% на збереження та погашення боргів.

Таблиця 1.3 – Правило 50/30/20

Категорія витрат	Відсоток від бюджету
Базові потреби	50%
Другорядні	30%
Заощадження	20%

Таблиця 1.4 – Метод 6 глечиків

Категорія витрат	Відсоток від бюджету
Базові потреби	55%
Другорядні	10%
Заощадження	10%
Освіта	10%
Резерв	10%
Подарунки і благодійність	5%

Метод 6 глечиків – це система розподілу доходів, де кожна частина представляється глечиком, який представляє одну з фінансових категорій. Шість глечиків включають: поточні витрати, розваги, заощадження, інвестиції, погашення боргів та непередбачені витрати. Цей метод спрямований на більш ефективне управління фінансами та раціональне розподілення коштів.

За рівнем збалансованості виділяються 2 типи бюджетів: рівноважний та нерівноважний. Рівноважний бюджет передбачає, що доходи дорівнюють витратам, що сприяє униканню боргів. Нерівноважний – навпаки [1].

Такий інструмент як фінансовий додаток має бути спрямованим на врахування сучасних тенденцій, забезпечуючи доступність та безпеку для кінцевого користувача.

Використання елементів статистики при розробці фінансового бота

дозволить здійснювати докладний аналіз користувацьких витрат, прогнозування майбутніх фінансових потреб, а також виявлення тенденцій та важливих патернів. На основі цих даних бот зможе надавати правдиві персоналізовані рекомендації щодо оптимізації бюджетування та досягнення поставлених фінансових цілей користувача [5].

Статистика допомагатиме відстежувати динаміку фінансових показників, оцінювати ефективність різних стратегій управління грошовими коштами та аналізувати вплив різних факторів на фінансовий стан користувача. Це сприяє більш точному прогнозуванню та плануванню фінансів, забезпечуючи кращий контроль над бюджетуванням.

Окрім того, статистичний аналіз може служити основою для вдосконалення алгоритмів бота, роблячи його більш адаптованим до індивідуальних потреб користувачів та забезпечуючи найбільш ефективний та персоналізований досвід управління фінансами.

Отже, використання статистики є ключовим елементом для забезпечення точності, надійності та раціональності фінансового управління.

1.4 План розробки моделі з урахуванням її властивостей

Застосування Telegram, Replit та UptimeRobot у поєднанні для створення телеграм-бота забезпечить надзвичайно зручну платформу для широкого кола користувачів. Наведений далі план розробки дозволить створити продукт, який відповідатиме сучасним стандартам та вимогам:

а) основи фінансового трекінгу:
   - реалізація додавання та видалення категорій;
   - введення та видалення витрат користувачем;

б) управління обмеженнями:

- додавання можливості встановлення обмежень для будь-яких категорій на деякий проміжок часу;
- можливість корегування обмежень за потреби;

в) система мрій:

- введення системи для відстеження та досягнення фінансових цілей користувача;
- динамічне відображення заощаджень для поточної цілі;

г) статистика та звіти:

- розробка функціоналу для збереження та виведення статистики витрат користувача;
- повідомлення про перевищення встановлених обмежень;

д) допоміжний функціонал:

- команда для запуску та інструкція щодо використання бота;
- інформаційний ресурс з порадами для користувачів;

е) розміщення готового застосунку на хостингу Replit;

ж) налаштування безперебійного доступу до бота з UptimeRobot.

Таким чином, поданий план розробки моделі для фінансового бота, спрямованого на використання Telegram, Replit та UptimeRobot, пропонує впровадження широкого функціоналу, що відповідає сучасним стандартам та вимогам користувачів. Реалізація базових фінансових трекінгових опцій, управління обмеженнями, системи мрій та детальної статистики становитиме основу для ефективного фінансового управління. Додатково, інструкції та інформаційний ресурс з порадами для користувачів допоможуть максимізувати користь від використання бота. Розміщення застосунку на Replit та його безперебійний доступ забезпечать зручну та надійну платформу для користувачів.

2 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Опис інструментів розробки

Підбір доречних інструментів грає важливу роль у долі проєкту, оскільки вони сприяють оптимізації робочих процесів, підвищенню продуктивності та забезпеченню ефективного виконання завдань. Точний підбір інструментарію впливає на якість і результативність проєкту.

2.1.1 Мова програмування Python

Вибір мови програмування грає важливу роль у процесі реалізації додатків будь-якого роду. Мова програмування Python була обрана з кількох поважних причин.

По-перше, Python є високорівневою та динамічною мовою програмування, що дозволяє зручно реалізовувати алгоритми та забезпечує швидку розробку програм. Синтаксис Python є простим і лаконічним, і це полегшує читання та обслуговування коду [6].

По-друге, ключовою перевагою Python є його широка підтримка в галузі статистичних обчислень та машинного навчання. Наявність потужних бібліотек, таких як telebot для роботи з Telegram API, робить Python ідеальним інструментом для розробки ботів.

За роки існування, ця мова програмування оволоділа великою спільнотою розробників, що робить її ідеальним кандидатом для наукових досліджень. Наявність багатьох онлайн-ресурсів, форумів та документації значно полегшує вирішення задач та виникаючих питань під час розробки.

Крім того, Python є крос-платформеним інструментом. Завдяки цій привілеї можна легко переносити розроблені моделі між різними операційними системами.

Таким чином, обираючи Python для розробки фінансового бота, можна розраховувати на універсальність готового продукту та швидку його оптимізацію. Начало формы

2.1.2 Огляд програмного середовища Replit та можливі альтернативи

Обране програмне середовище має значний вплив на зручність розробки та впровадження проєкту, тому для реалізації та тестування фінансового бота було обрано середовище Replit (див. рис. 2.1).

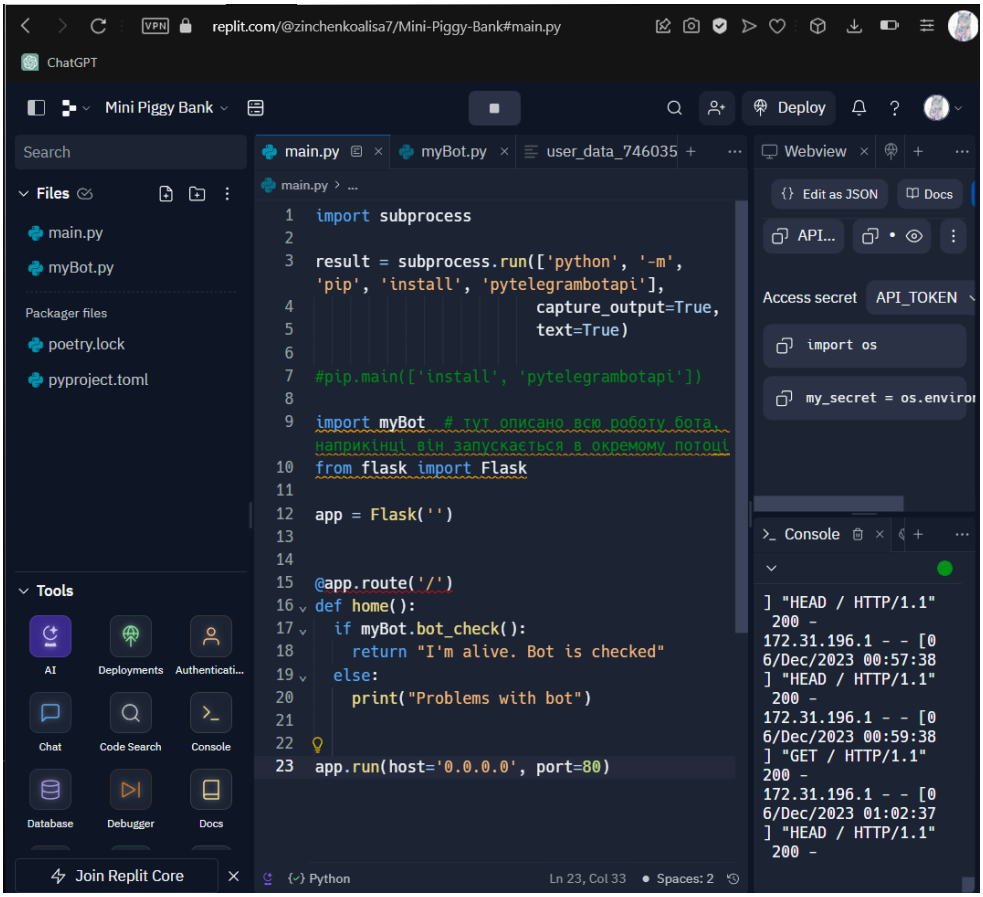

Рисунок 2.1 – Демонстрація середовища Replit при розробці бота

Replit – це онлайн-інтерфейс для розробки, який надає зручний доступ до середовища Python без необхідності завантаження додаткового середовища на локальний комп'ютер. Однією з основних переваг Replit є можливість збереження та обміну проєктами онлайн, що спрощує колективну роботу та уникнення проблем зі сумісністю версій [10].

Крім того, Replit дозволяє запускати бота безпосередньо в хмарі, що робить процес тестування більш зручним та ефективним. Це особливо актуально для розробки ботів, які взаємодіють з зовнішніми сервісами, наприклад, Telegram API. Хоча будь-який проєкт на безкоштовній основі тут публічний, API можна зашифрувати у вигляді секрету та не переживати за його збереженність.

Можливі альтернативи Replit включають онлайн середовища, такі як Glitch, Visual Studio Online. Обидва ці сервіси можуть використовуватися як платформи для хостингу проєктів. Replit і Glitch добре підходять для швидкої розробки та експериментів, Visual Studio Online може бути вибором для великих проєктів з широким функціоналом та інтегрованими інструментами Visual Studio. Проте Replit є найбільш доступним для запланованого.

2.1.3 Сервіс моніторингу UptimeRobot

UptimeRobot – це сервіс моніторингу доступності вебресурсів та служб, який дозволяє розробникам програмного забезпечення стежити за роботою вебсайтів, серверів і.т.д. Сервіс автоматично перевіряє доступність ресурсу з різних місць у світі, надсилаючи сповіщення про екстрене завершення роботи через електронну пошту, SMS, Slack тощо в разі виявлення недоступності [8].

UptimeRobot пропонує інтуїтивно зрозумілий інтерфейс для налаштування моніторів та отримання звітів. Сервіс підтримує різноманітні протоколи, такі як HTTP, HTTPS, Ping, TCP, та інші, і надає статистику щодо часу відповіді сервера,

кодів відповіді та інших параметрів.

Основний безкоштовний тарифний план UptimeRobot відповідає потребам невеликих проєктів. Користувачі можуть налаштовувати інтервали часу для моніторингу враховуючи специфіку свого проєкту. Так як потреба в моніторингу доступності стає критичною для забезпечення безперебійної роботи бота. У поєднанні з Replit, UptimeRobot стає частиною комплексного рішення для розробки та моніторингу. Разом вони забезпечують стабільну «життєдіяльність» додатка та ефективне виявлення можливих похибок.

Процес створення монітору для бота демонструє рисунок (див. рис. 2.2).

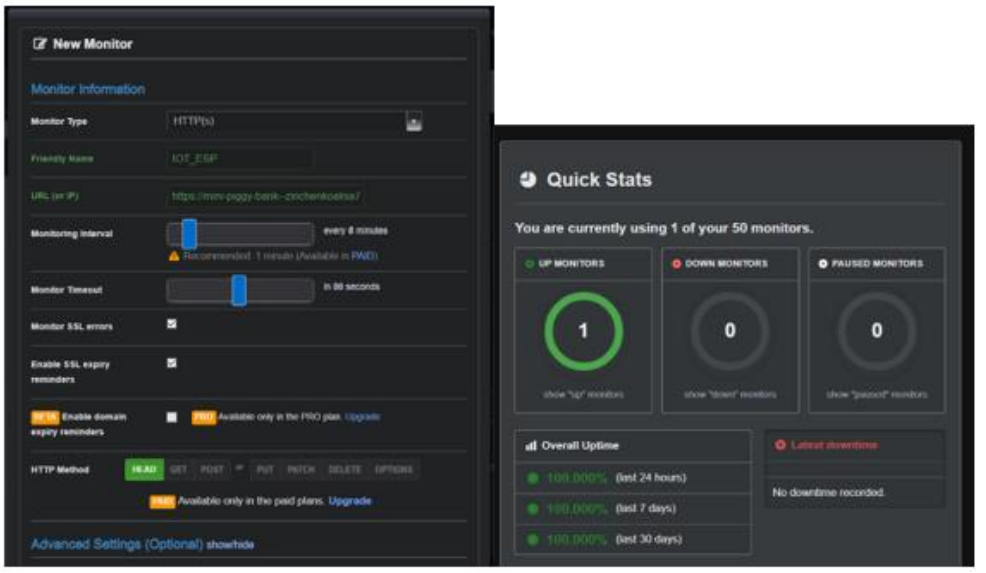

Рисунок. 2.2 – Візуалізація готового монітору

Таким чином UptimeRobot взаємодіє з фінансовим ботом.

2.1.4 Переваги використання месенджера Telegram

Telegram надає розширені можливості для створення ботів, гарантуючи їх функціональність та ефективність. Інтеграція з inline-командами, обробка

різноманітних запитань та команд дозволяють створювати різноманіття функцій для комфортного та інтуїтивно зрозумілого управління фінансовим ботом. Telegram використовує потужні засоби шифрування, забезпечуючи безпеку та конфіденційність обміну фінансовою інформацією через бота. Це стає важливим аспектом при створенні фінансового інструменту для користувачів. Сервіс надає великі можливості для вдосконалення інтерактивності та забезпечення більш гнучкого функціоналу фінансового бота [9].

Далі декілька прикладів найбільш відомих ботів:

– BotFather – офіційний бот Telegram, який дозволяє користувачам створювати власні боти та отримувати токени для керування ними;
– @TriviaBot – для проведення вікторин та тестів на різні теми;
– @gif – дозволяє користувачам знаходити та надсилати анімовані GIF-зображення за ключовими словами;
– @weatherman_bot – для отримання актупльної інформації про погоду в будь-яких регіонах;
– @GithubBot – для відстеження оновлень та інших подій у репозиторіях GitHub.

Ці боти представляють лише невелику частину широкого спектру функціональності, яку можуть надавати боти в Telegram. Залежно від потреб користувача, можна знайти багато різноманітних ботів для різних завдань та інтересів [7].

Альтернатив до задуманого функціоналу бота знайдено не було. Серед схожих за наповненням можна виділити бот під назвою @smart_kopilka_bot, за допомогою котрого користувач має змогу фіксувати щоденні накопичення на певну мету та отримувати поради щодо накопичення. Окрім цього, існує безліч фінансових порадників з вузьким напрямком відомостей.

Фінансовий бот вигідно виділяється порівняно з іншими застосунками, такими як Monefy, Spendee, Money Manager Expense & Budget, Wallet та Family

Budget, завдяки кільком ключовим перевагам:

- завжди «під рукою», не потрібно завантажувати додаткові застосунки чи файли для вільного доступу;
- пропонує інтуїтивний та зручний інтерфейс для введення та класифікації витрат;
- можливість встановлення обмежень на категорії окремо;
- додаткова можливість вести облік та відстежувати прогрес у реалізації фінансових мрій;
- сприяє покращенню фінансової грамотності користувачів, надаючи поради та рекомендації.

2.2 Архітектура додатка

При описі процесу створення фінансового бота, архітектуру можна умовно розділити на 2 блоки [11].

Перший блок описує запуск бота в окремому потоці, в той час як другий відповідає за весь функціонал додатка. Створення такої архітектури значно полегшить процес оптимізації в майбутньому.

Завдяки використанню Replit як програмного середовища, можна легко відстежити коли, де і як було запущено додаток, адже обране програмне середовище міть безліч вбудованних функцій, що полегшують процес розробки.

Для запуску бота в окремому потоці через Replit використовується наступний код (див. рис. 2.3).

Для ініціалізації та підключення бота використовується бібліотека Telegram для роботи з API бота і токен Telegram Bot API відповідно (див. рис. 2.4). В свою чергу, токен зберігається у вигляді секрета API_TOKEN, що забезпечує конфіденціальність ключа та не дасть зловмисникам викрасти доступ до додатка.

Дані користувачів зберігаються у вигляді словників окремими групами, то ж доступ до власних ланих залищиться лише у користувачів.

```
import subprocess

result = subprocess.run(['python', '-m', 'pip', 'install', 'pytelegrambotapi'],
                        capture_output=True,
                        text=True)

#pip.main(['install', 'pytelegrambotapi'])

import myBot
from flask import Flask

app = Flask('')

@app.route('/')
def home():
  if myBot.bot_check():
    return "I'm alive. Bot is checked"
  else:
    print("Problems with bot")

app.run(host='0.0.0.0', port=80)
```

Рисунок 2.3 – Код для запуску в окремому потоці

```
import telebot as tg
from telebot import types
import threading
import os
import time
import datetime
import re
from collections import defaultdict
import openpyxl

bot = tg.TeleBot(os.getenv("API_TOKEN"))

def bot_check():
  return bot.get_me()

def bot_runner():
  bot.infinity_polling(none_stop=True)

t = threading.Thread(target=bot_runner)
t.start()
```

Рисунок 2.4 – Код ініціалізації та підключення бота

Команди можна умовно поділити на загальні, взаємодію з витратами, взаємодію з мріями та інші функції. Всі вони забезпечують широкий спектр можливостей для роботи з ботом, включаючи корекцію витрат, керування обмеженнями, отримання статистики витрат, роботу з мріями користувача та загальні адміністративні функції.

Далі детальний перелік команд та функцій до них. Назви функцій інтуїтивно зрозумілі, тому нема необхідності описувати кожну окремо.

а) команди для взаємодії з категоріями:

- додати категорію: process_add_category(message);
- видалити категорію: process_remove_category(message);

б) команди для взаємодії з витратами:

1) виправити витрати за категорією:
   - process_correct_expenses_selection(message);
   - process_correct_data(message, selected_category);
   - process_correct_expense_value(message,selected_category, expense_number);
2) виправити обмеження для категорії:
   - process_correct_limit(message);
   - process_choose_category_for_limit_correction(message);
   - process_correct_limit_value(message, selected_category);
3) статистика витрат: handle_statistics(message);

в) команди для взаємодії з мріями:

1) накопичити на мрію:
   - handle_dream_button(message);
   - process_dream_name(message);
   - process_add_savings(message, dream_name);
2) скорегувати мрію:
   - handle_change_dream(message);

– process_change_dream(message);

г) загальні команди:

1) зберегти дані: handle_save_data(message);

2) очистити історію:

– handle_reset_data(message);

– process_reset_confirmation(message);

3) відображення команд: show_commands(user_id);

4) збереження даних: save_data_to_excel(user_id);

5) запуск бота у режимі постійного очікування повідомлень: bot.infinity_polling(none_stop=True).

Таким чином, для функціонування бота використовується код, який визначає необхідні бібліотеки та створює окремий потік через сервіс Replit. Ініціалізація та підключення бота реалізовані за допомогою бібліотеки Telegram, використовуючи токен Telegram Bot API. Надано детальний опис команд та функцій бота, розподілених за категоріями, що включають взаємодію з витратами, мріями та загальні адміністративні опції. Подано також перелік команд для корекції витрат, роботи з обмеженнями, отримання статистики витрат, а також для взаємодії з мріями користувача та адміністративними функціями.

3 ЕКСПЕРИМЕНТАЛЬНЕ ВИВЧЕННЯ ФІНАНСОВОГО БОТА

3.1 Апробація додатка

Для демонстрації роботи програми нижче наведено приклади для кожної команди та обробка всіх можливих помилок при введені даних користувачем.

Щоб знайти бот в пошуку Telegram, необхідно знайти його через пошук [12]. Отримали такий результат (див. рис. 3.1).

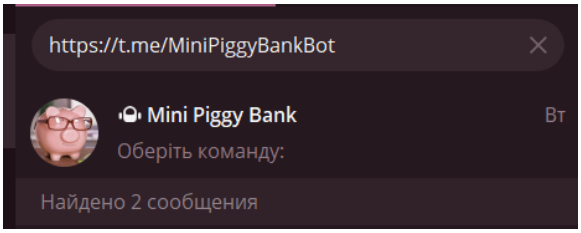

Рисунок 3.1 – Демонстрація бота у рядку пошуку Telegram

Запуск фінансового бота Mini Piggy Bank та виклик команди «допомога» здійснюється шляхом введення /start і /help відповідно (див. рис. 3.2).

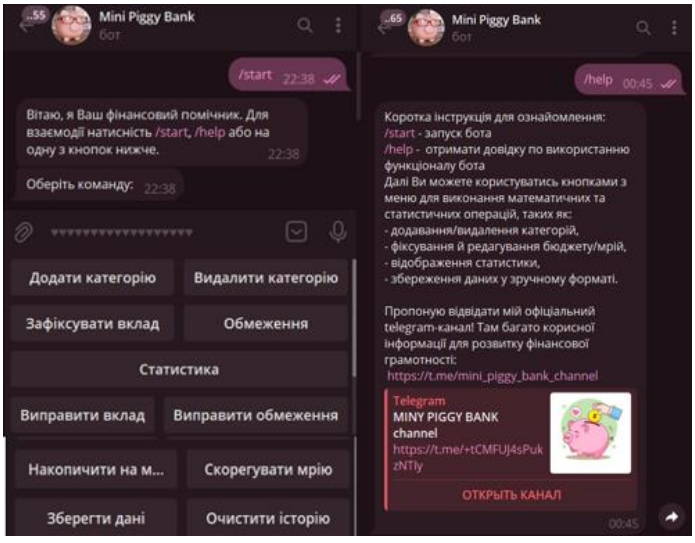

Рисунок 3.2 – Демонстрація початку взаємодії з ботом

Виклик команди "Додати категорію" у фінансовому боті спрямований на можливість користувача створювати нові категорії для визначення та класифікації своїх витрат. Ця функція сприяє персоналізації та адаптації додатка до індивідуальних фінансових потреб користувачів (див. рис. 3.3).

Рисунок 3.3 – Демонстрація виклику команди «Додати категорію»

Команда «Видалити категорію» призначена для видалення існуючої категорії витрат користувача, а можливі помилки передбачають взаємодію бота з користувачем для забезпечення правильності операцій та уникнення непорозумінь. Демонстрація на рисунку (див. рис. 3.4).

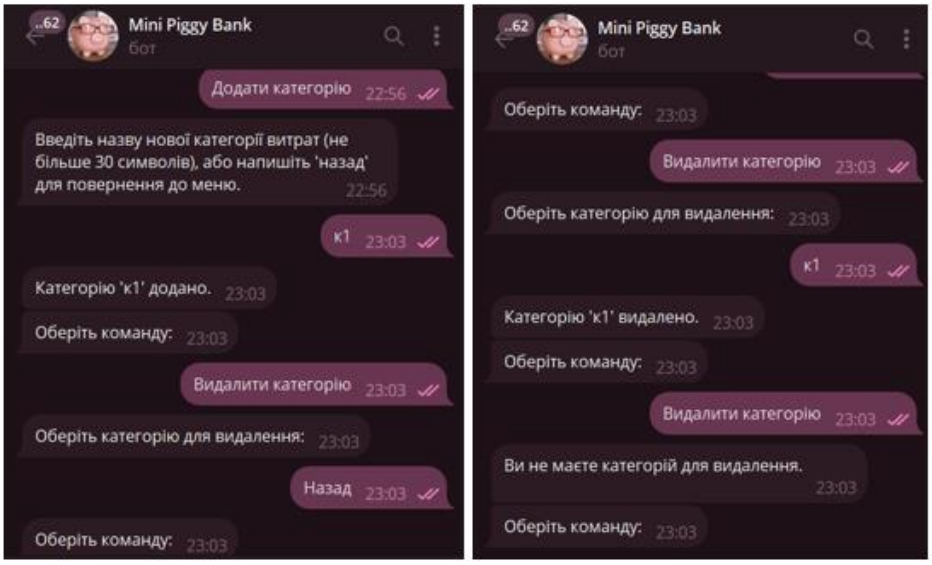

Рисунок 3.4 – Демонстрація виклику команди «Видалити категорію»

Виклик команди «Зафіксувати вклад» використовується для реєстрації внесення грошового внеску користувача у фінансовому боті (див. рис. 3.5).

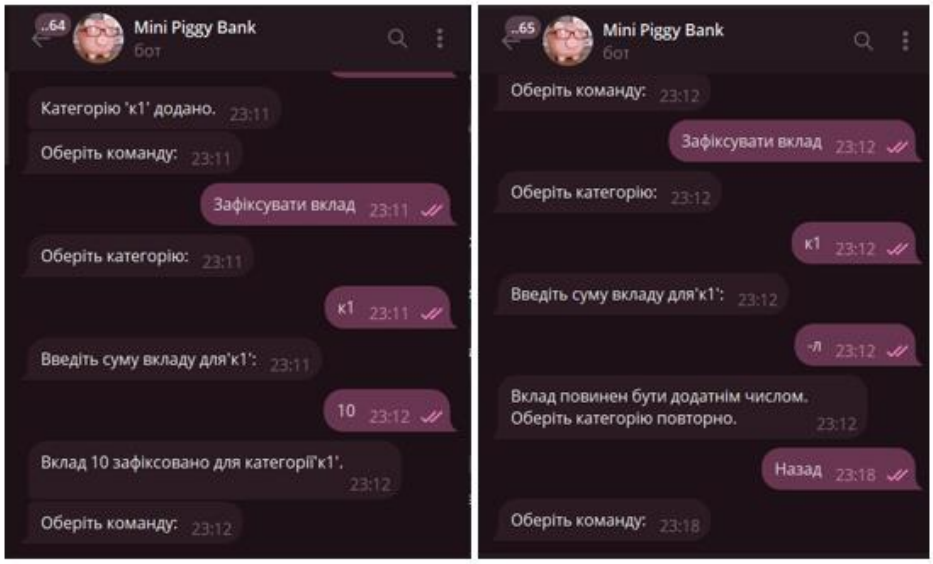

Рисунок 3.5 – Демонстрація виклику команди «Зафіксувати вклад»

Виклик команди «Обмеження» використовується для встановлення обмежень на витрати в діючих категоріях, також Система обробляє можливі

помилки, такі як введення нечислових значень або від’ємних чисел, та повідомляє користувача про необхідні правки. Демонстрація роботи команди на рисунку (див. рис. 3.6).

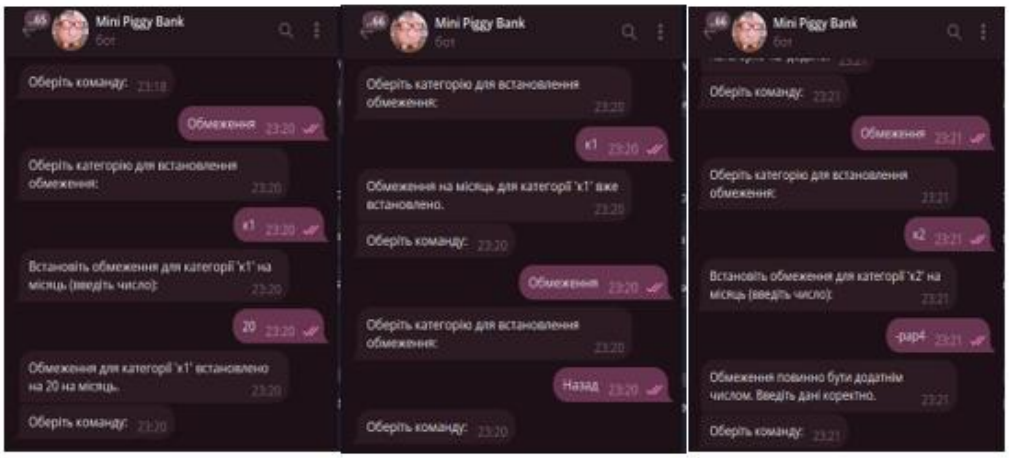

Рисунок 3.6 – Демонстрація виклику команди «Обмеження»

Команда «Статистика» відображає зведену інформацію про фінансовий стан користувача та можливі перевищення обмежень. Демонстрація роботи команди нижче. В першій частині візуалізуються стандартне відображення, при нестачі даних (див. рис. 3.7).

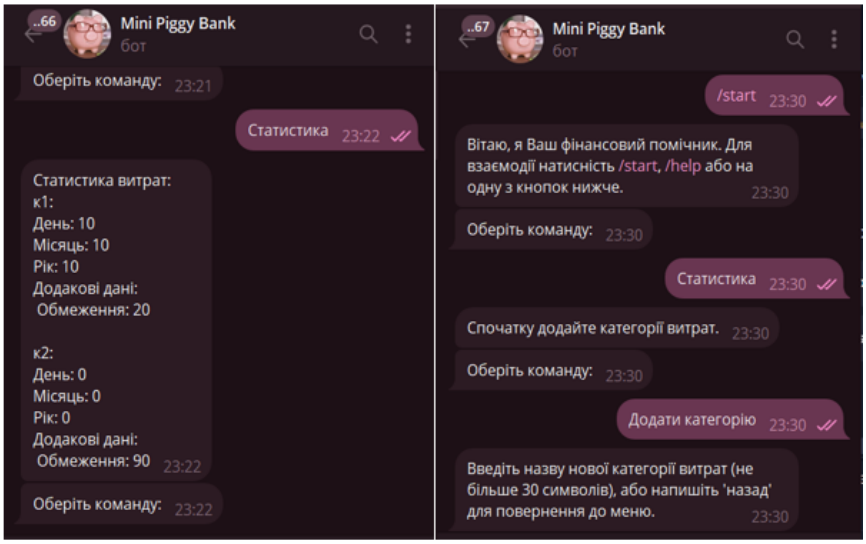

Рисунок 3.7 – Демонстрація виклику команди «Статистика» та обробка помилок

В другій частині візуалізуються пусті категорії, перевищення обмежень (див. рис. 3.8).

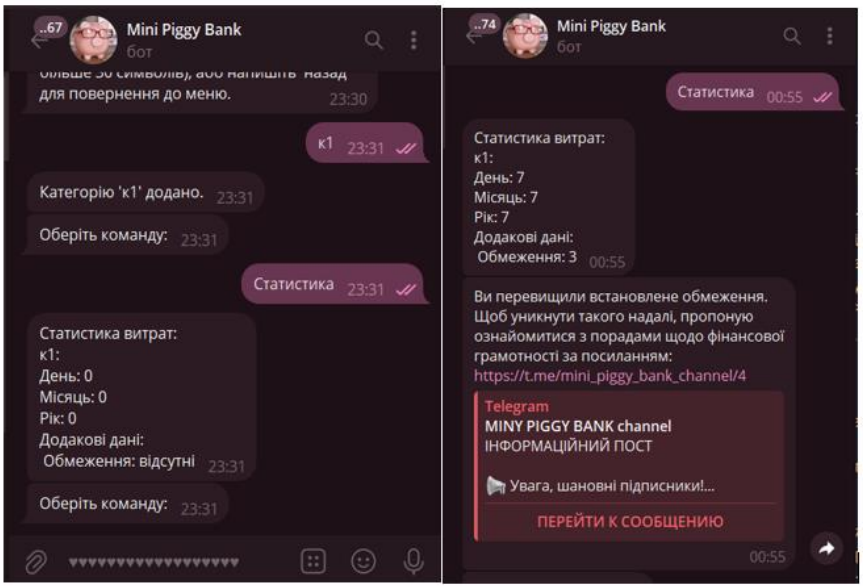

Рисунок 3.8 – Демонстрація виклику команди «Статистика» та обробка помилок

Виклик команди «Виправити вклад» дозволяє змінити інформацію про раніше зафіксований вклад у дійсних категоріях. В першій частині візуалізуються стандартне виправлення та пусті категорії (див. рис. 3.9).

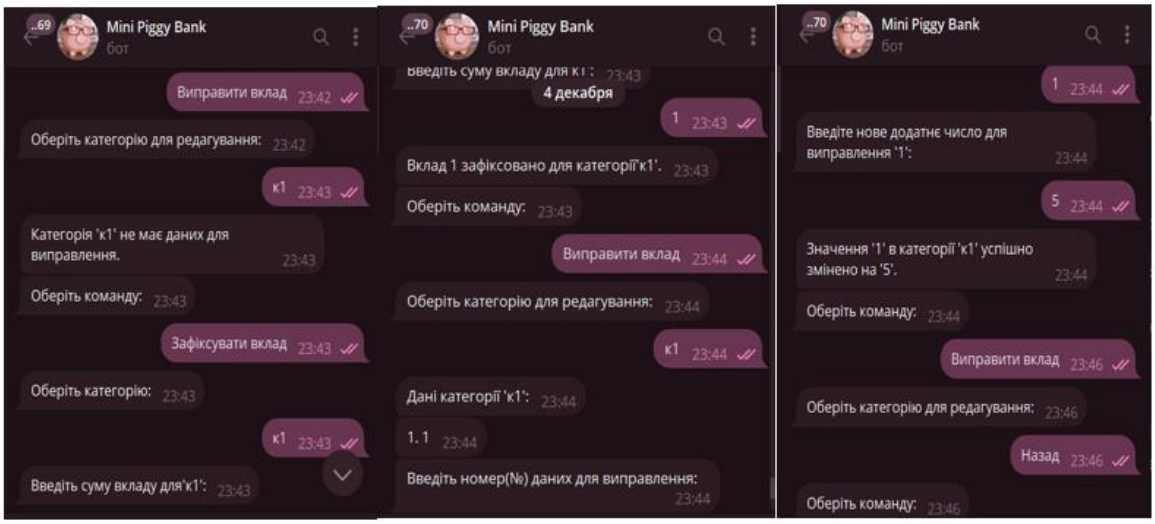

Рисунок 3.9 – Демонстрація виклику команди «Виправити вклад»

В другій частині візуалізуються вибір хибного номеру, похибка при введенні числа та виправлення з декількома значеннями (див. рис. 3.10).

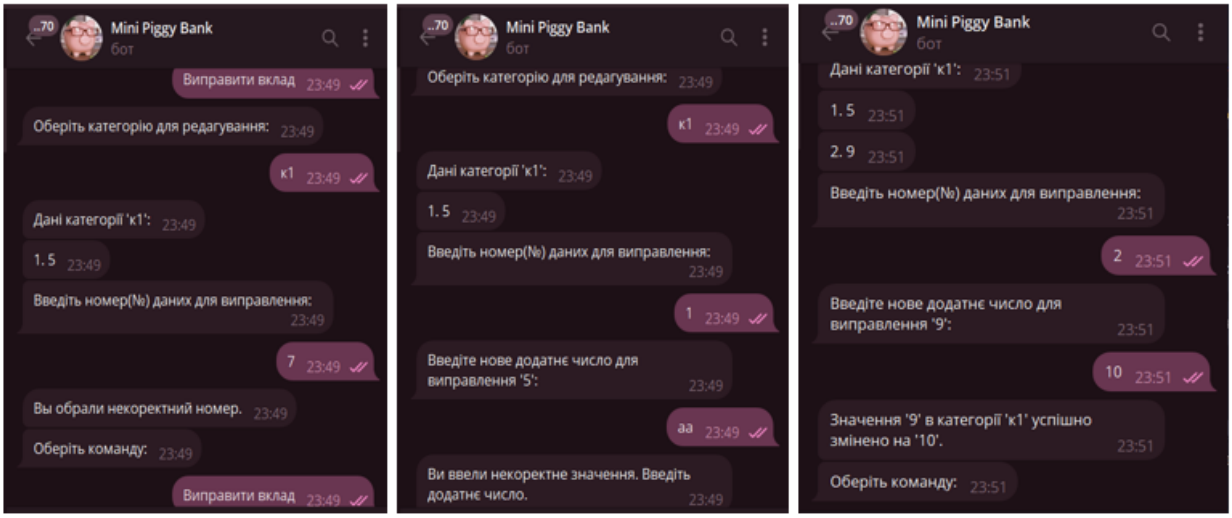

Рисунок 3.10 – Демонстрація виклику команди «Виправити вклад»

Виклик команди «Виправити обмеження» дозволяє користувачам змінювати встановлені обмеження на витрати для наявних категорій. Користувач обирає категорію, згодом бот показує поточне обмеження для обраної категорії та запитує нове значення обмеження (див. рис. 3.11).

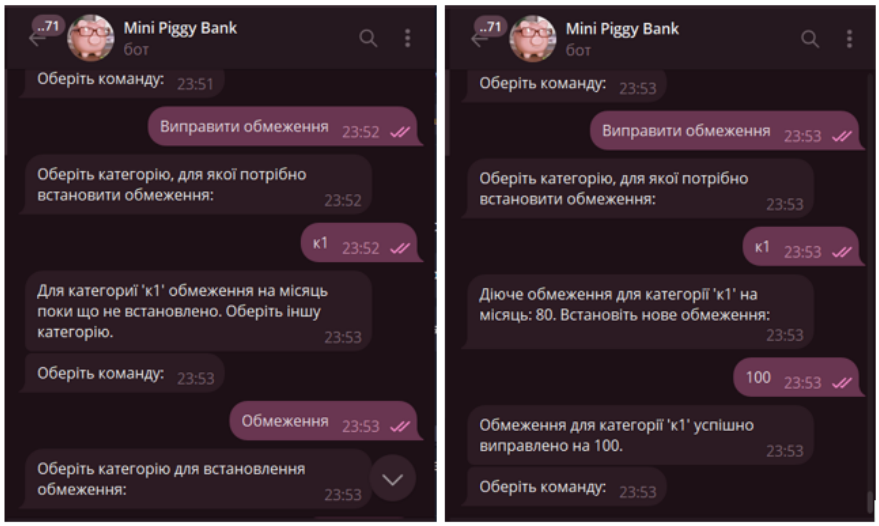

Рисунок 3.11 – Демонстрація виклику команди «Виправити обмеження»

Для обробки помилок також наведено приклади (див. рис. 3.12).

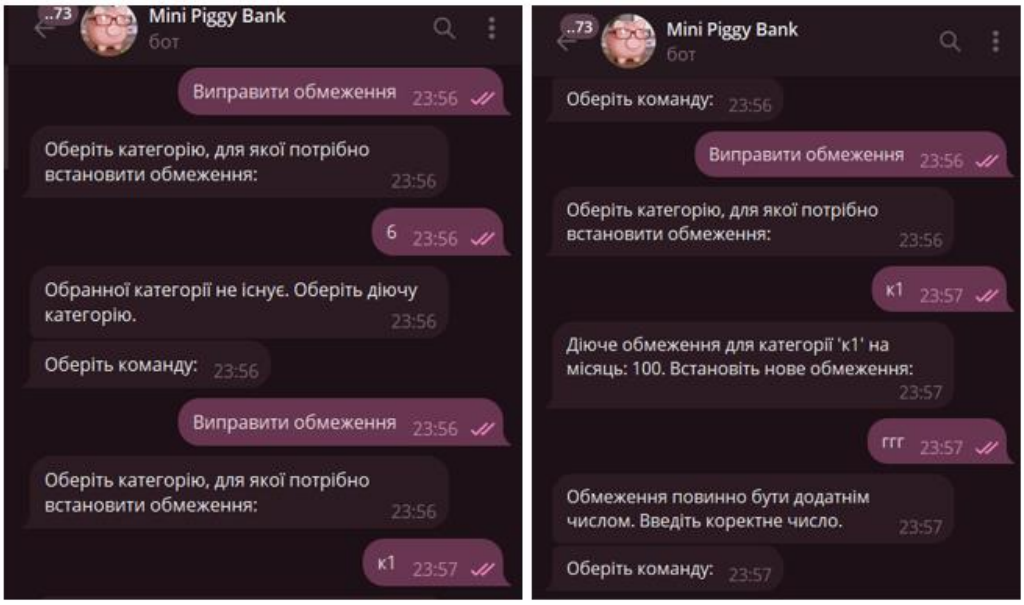

Рисунок 3.12 – Демонстрація виклику команди «Виправити обмеження»

Команда «Накопичити на мрію» дозволяє користувачам встановлювати та відстежувати фінансові мрії та цілі, на які вони хочуть зберігати кошти. Демонстрацію роботи команди наведено на рисунку (див. рис. 3.13).

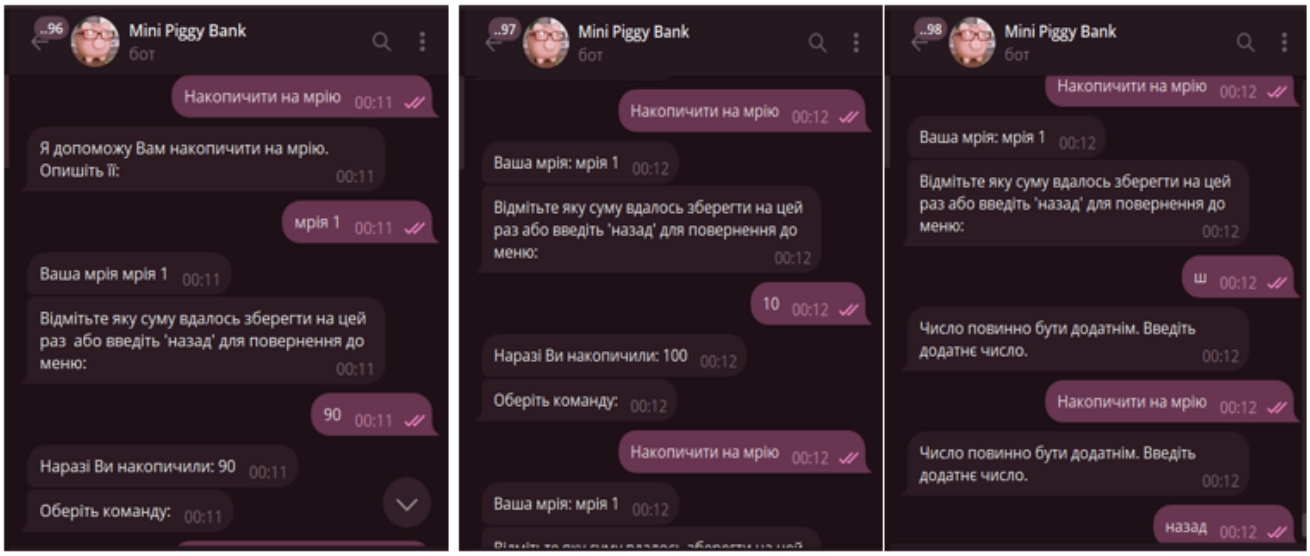

Рисунок 3.13 – Демонстрація виклику команди «Накопичити на мрію»

Команда "Скорегувати мрію" дозволяє користувачам змінювати опис фінансових мрій та обнуляє лічильник заощаджень. Демонстрацію роботи команди наведено на рисунку (див. рис. 3.14).

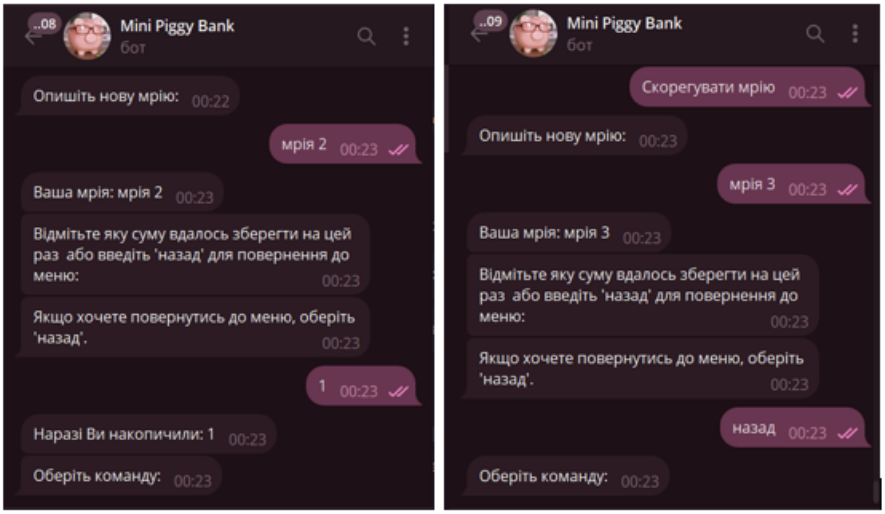

Рисунок 3.14 – Демонстрація виклику команди «Скорегувати мрію»

Команда "Зберегти дані" використовується для збереження даних користувача до файлу в форматі Excel, зокрема назв категорій витрат та перелік доданих чисел по стовпцях (див. рис. 3.15).

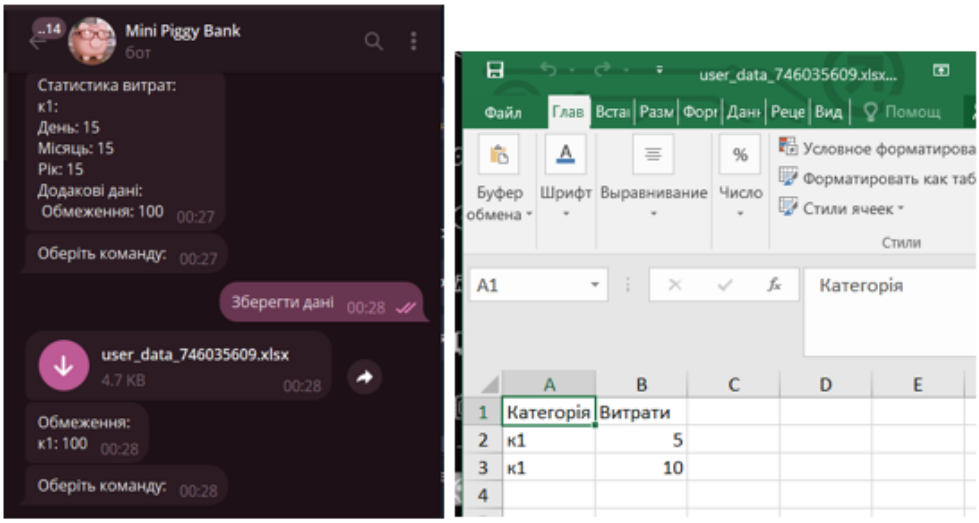

Рисунок 3.15 – Демонстрація виклику команди «Зберегти дані»

Команда "Очистити історію" призначена для повного видалення історії фінансових даних користувача. Бот виводить запит на підтвердження видалення історії.

Після отримання підтвердження, історія фінансових даних користувача повністю видаляється. Демонстрація на рисунку (див. рис. 3.16).

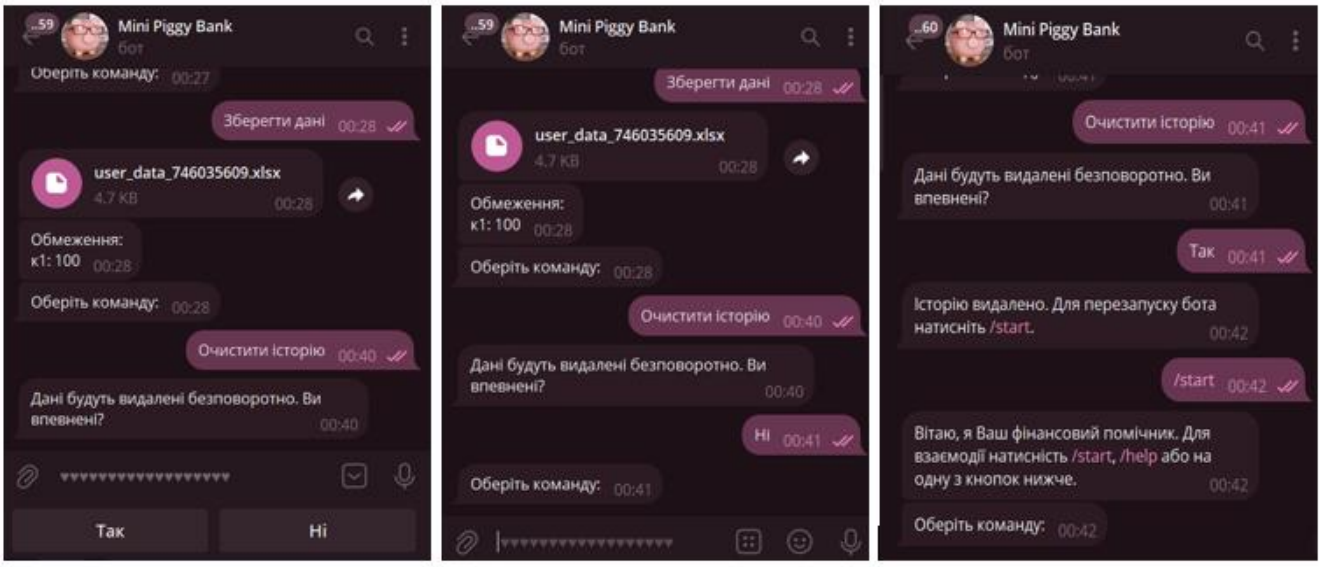

Рисунок 3.16 – Демонстрація виклику команди «Очистити історію»

Це базовий функціонал фінансового бота, за допомогою котрого користувач може здійснювати контроль за витратами та зберігати дані у зручному форматі.

Доступ до ресурсу з інформацією про фінансову освіту, користувач може отримати наступним чином:

– в описі бота;
– через команду /help;
– якщо перевищить одне із встановлених ним самим обмежень.

Ресурс представляє собою Telegram-канал, що напряму пов’язаний з фінансовим ботом. За посиланням в боті користувач побачить повідомлення-привітання з переліком доступних посилань (див. рис. 3.17).

Рисунок 3.17 – Демонстрація привітального повідомлення ресурсу

Статті під посиланням мають вигляд стислих інформативних довідок з теорією та практичними порадами з бюджетування і.т.д. (див. рис. 3.18) [13–15].

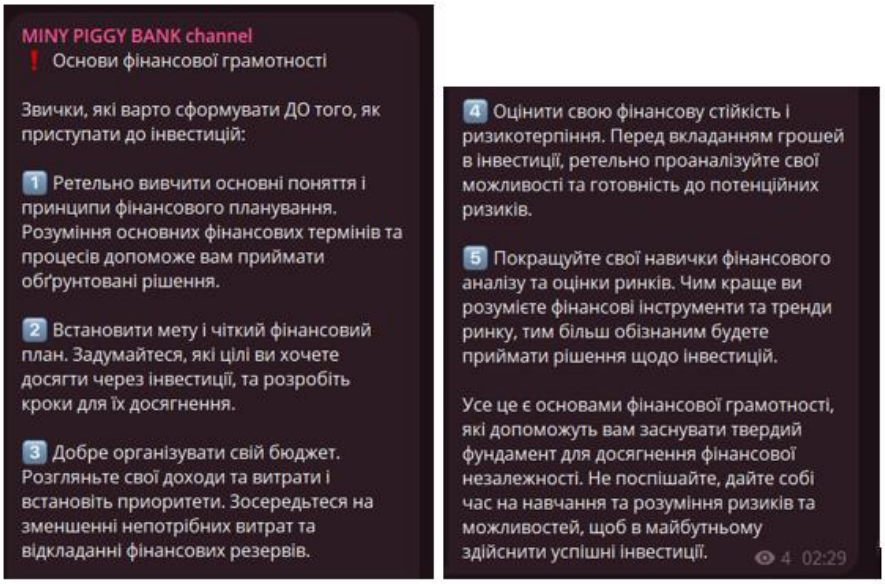

Рисунок 3.18 – Демонстрація прикладу статті з інформаційного ресурсу

Посилання для довідок додано через засіб редагування тексту Telgram. Завдяки цьому, в будь-який момент посилання на статті чи самі статті можна буде легко оформити в іншому форматі (див. рис. 3.19).

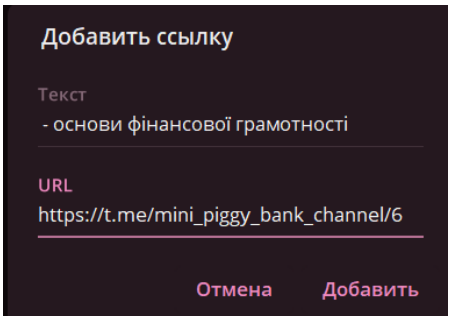

Рисунок 3.19 – Демонстрація додавання посилання для довідки

Таким чином, під час демонстрації фінансового додатку виявлено математичні принципи та елементи інтелектуальної автоматизації, що трансформують його в неординарну синергію фінансового управління. У результаті показу всіх його функцій можна визначити, що даний додаток екстраполює в себе сучасні концепції фінансової науки, створюючи зручний інструмент для систематичного контролю та оптимізації фінансового потоку.

3.2 Рекомендації щодо подальших досліджень

В результаті проведеного дослідження реалізовано фінансовий додаток, призначений для вдосконалення системи користувацького бюджетування. Демонстрація різноманіття його функціоналу вказала на значущий потенціал для подальших удосконалень [12].

Пріоритетним аспектом подальших досліджень є розгляд можливостей розширення функціоналу фінансового бота шляхом інтеграції мовного вибору, сприяючи тим самим розширенню аудиторії користувачів, зацікавлених у

глибокому розвитку фінансових аспектів. Додатково, пропонується реалізація автоматичного аналізу річних витрат на основі накопиченої інформації, що сприятиме візуалізації бюджету та вдосконаленню фінансової грамотності користувача.

Перспективою є інтеграція з банківськими системами для автоматичного отримання та аналізу транзакцій, спрямована на оптимізацію процесу введення інформації. Значущим вектором також є розгляд можливості співпраці з досвідченими фінансовими консультантами для надання незамінних персоналізованих порад користувачам.

Таким чином, подальші дослідження мають фокусуватись на збільшенні інтелектуальних можливостей фінансового бота для максимізації задоволеності та зацікавленості додатком користувачів.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи досягнута мета, цілі та завдання, що були поставлені на початку. Кінцевим продуктом став фінансовий додаток, розроблений на мові програмування Python, призначений для вдосконалення системи користувацького бюджетування. Функціонал бота дозволяє здійснювати контроль та аналіз фінансових витрат, а також встановлювати обмеження для ефективного керування бюджетом.

Важливою характеристикою розробленого бота є можливість користувачів встановлювати та редагувати категорії витрат, що робить його інтуїтивно зрозумілим та гнучким інструментом для фінансового планування. Також надається можливість відстеження прогресу у досягненні фінансових мрій, що сприяє мотивації та контролю над фінансами.

У подальших дослідженнях рекомендується розглядати можливості розширення функціоналу бота. Зокрема, це може включати автоматичний аналіз витрат та інтеграцію з банківськими системами для автоматичного отримання та аналізу транзакцій. Також рекомендується дослідити можливості інтеграції з фінансовими консультантами для отримання персоналізованих порад щодо фінансового планування.

В цілому, розроблений фінансовий бот має значний потенціал для подальших удосконалень та розширення функціоналу, що робить його ефективним інструментом для фінансового управління та досягнення фінансових цілей користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Грищенко Т. С., Вітка Ю. В., Дягілєв А. Б. Фінансова грамотність. Фінанси. Що? Чому? Як? Навчальний посібник. Проект USAID «Трансформація фінансового сектору», 2019. URL: https://drive.google.com/file/d/1sX0_yXLchhQ_pj_QLyakgbpGqo5Sdf0K/view (дата звернення: 05.09.2023).

2. Електронні гаманці: для чого і як ними користуватися? URL: https://netpeak.net/uk/blog/yelektronni-gamantsi-dlya-chogo-i-yak-nimi-koristuvatisya/ (дата звернення: 07.09.2023).

3. Фінанси у смартфоні – мобільні застосунки для планування бюджету. URL: https://harazd.bank.gov.ua/article/finansove-planuvanna/finansovi-cili/finansi-u-smartfoni-mobilni-zastosunki-dla-planuvanna-budzetu (дата звернення: 07.09.2023).

4. Витрачайте розумно – ефективні методи планування витрат. URL: https://harazd.bank.gov.ua/article/finansove-planuvanna/finansovi-cili/vitracajte-rozumno-efektivni-metodi-planuvanna-vitrat (дата звернення: 08.09.2023).

5. Як оцінити своє поточне фінансове становище? URL: https://harazd.bank.gov.ua/article/finansove-planuvanna/finansovi-cili/ak-ociniti-svoe-potocne-finansove-stanovise (дата звернення: 15.09.2023).

6. Python – документація. Офіційний сайт: https://docs.python.org/3/ (дата звернення: 20.09.2023).

7. Як створити Telegram бота на Python. URL: https://foxminded.ua/telehram-bot-na-python/ (дата звернення: 20.09.2023).

8. UptimeRobot – офіційний сайт. URL: https://uptimerobot.com (дата звернення: 10.10.2023).

9. Telegram – офіційний сайт. URL: https://telegram.org (дата звернення: 20.09.2023).

10. Replit – офіційний сайт. URL: https://docs.replit.com (дата звернення: 20.09.2023).

11. Miny Piggy Bank. URL: https://replit.com/@zinchenkoalisa7/Mini-Piggy-Bank?v=1 (дата звернення: 30.09.2023).

12. Miny Piggy Bank – посилання на Telegram. URL: https://t.me/MiniPiggyBankBot (дата звернення: 30.09.2023).

13. Фінансова “подушка безпеки” – як створити та якою вона має бути? URL: https://harazd.bank.gov.ua/article/finansove-planuvanna/finansovi-cili/finansova-poduska-bezpeki-ak-stvoriti-ta-akou-vona-mae-buti (дата звернення: 30.09.2023).

14. Платіжне шахрайство як глобальний тренд – способи протидії. URL: https://harazd.bank.gov.ua/article/sahrajstvo/platizne-sahrajstvo/platizne-sahrajstvo-ak-globalnij-trend-sposobi-protidii (дата звернення: 30.09.2023).

15. 7 ефективних способів захистити свій фінансовий номер телефону від шахраїв. URL: https://harazd.bank.gov.ua/article/theme/sahrajstvo/life-hack/7-efektivnih-sposobiv-zahistiti-svij-finansovij-nomer-telefonu-vid-sahraiv (дата звернення: 30.09.2023).

ДОДАТОК А

Main.py – розгортання бота на хостингу

```
import subprocess

result = subprocess.run(['python', '-m', 'pip', 'install', 'pytelegrambotapi'],
                        capture_output=True,
                        text=True)

#pip.main(['install', 'pytelegrambotapi'])

import myBot
from flask import Flask

app = Flask('')


@app.route('/')
def home():
  if myBot.bot_check():
    return "I'm alive. Bot is checked"
  else:
    print("Problems with bot")


app.run(host='0.0.0.0')
```

ДОДАТОК Б

MyBot.py – реалізація функціоналу бота

```
import telebot as tg
from telebot import types
import threading
import os
import time
import datetime
import re
from collections import defaultdict
import openpyxl

bot = tg.TeleBot(os.getenv("API_TOKEN"))

def bot_check():
  return bot.get_me()

def bot_runner():
  bot.infinity_polling(none_stop=True)

t = threading.Thread(target=bot_runner)
t.start()

# Словник для зберігання даних користувача
user_data = {}

# Функція для перевірки коректності назви категорії
def is_valid_category_name(category_name):
  return bool(category_name)

# Функція для отримання/створення словника для визначеного користувача і категорії
def get_or_create_user_category_expenses(user_id, category):
  if user_id not in user_data:
    user_data[user_id] = {'categories': {}, 'expenses': {}}
  if category not in user_data[user_id]['expenses']:
    user_data[user_id]['expenses'][category] = []
  return user_data[user_id]['expenses'][category]

# Функція для зберігання даних в Excel
def save_data_to_excel(user_id):
  if user_id in user_data:
    workbook = openpyxl.Workbook()
    sheet = workbook.active

    # Заголовок для таблиці
    sheet.append(["Категорія", "Витрати"])

    # Додавання даних
    for category, expenses in user_data[user_id]['expenses'].items():
      for expense in expenses:
        sheet.append([category, expense])
```

```
    # Збереження до файлу
    filename = f"user_data_{user_id}.xlsx"
    workbook.save(filename)

    bot.send_document(user_id, open(filename, 'rb'))
    os.remove(filename)
  else:
    bot.send_message(user_id, "Немає даних для збереження.")

# Команда /старт
@bot.message_handler(commands=['start'])
def handle_start(message):
  user_id = message.from_user.id
  bot.send_message(
      user_id,
      "Вітаю, я Ваш фінансовий помічник. Для взаємодії натисність /start, /help або на одну з кнопок
нижче."
  )
  show_commands(user_id)
# Команда /допомога
@bot.message_handler(commands=['help'])
def handle_help(message):
  user_id = message.from_user.id
  help_text = (
      "Коротка інструкція для ознайомлення:\n"
      "/start - запуск бота\n"
      "/help -  отримати довідку по використанню функціоналу бота\n"
      "Далі Ви можете користуватись кнопками з меню для виконання математичних та статистичних
операцій, таких як:\n"
      "- додавання/видалення категорій,\n"
      "- фіксування й редагування бюджету/мрій,\n"
      "- відображення статистики,\n"
      "- збереження даних у зручному форматі. \n\n"
      "Пропоную відвідати мій офіціальний telegram-канал! Там багато корисної інформації для розвитку
фінансової грамотності:\n https://t.me/mini_piggy_bank_channel"
  )
  bot.send_message(user_id, help_text)
# Функція для показу команд
def show_commands(user_id):
  markup = tg.types.ReplyKeyboardMarkup(resize_keyboard=True)
  markup.row("Додати категорію", "Видалити категорію")
  markup.row("Зафіксувати вклад", "Обмеження")
  markup.row("Статистика")
  markup.row("Виправити вклад", "Виправити обмеження")
  markup.add("Накопичити на мрію", "Скорегувати мрію")
  markup.row("Зберегти дані", "Очистити історію")
  bot.send_message(user_id, "Оберіть команду:", reply_markup=markup)

  # Обробник команди "Додати категорію"
@bot.message_handler(func=lambda message: message.text == "Додати категорію")
def handle_add_category(message):
  user_id = message.from_user.id

  if user_id not in user_data:
    user_data[user_id] = {'categories': {}, 'expenses': {}}
```

```
  if 'categories' not in user_data[user_id]:
    user_data[user_id]['categories'] = {}

  bot.send_message(
      user_id,
      "Введіть назву нової категорії витрат (не більше 30 символів), або напишіть 'назад' для повернення
до меню."
  )
  bot.register_next_step_handler(message, process_add_category)

def process_add_category(message):
  user_id = message.from_user.id
  category_name = message.text

  if category_name.lower() == 'назад':
    show_commands(user_id)
    return

  if len(category_name) > 30:
    bot.send_message(
        user_id,
        "Назва категорії занадто довга. Введіть назву не більше ніж 30 символів."
    )
    show_commands(user_id)
    return

  if category_name in user_data[user_id]['categories']:
    bot.send_message(user_id, "У Вас вже є така категорія.")
  else:
    user_data[user_id]['categories'][category_name] = []
    bot.send_message(user_id, f"Категорію '{category_name}' додано.")
  show_commands(user_id)

# Обробник команди "Зафіксувати вклад"
@bot.message_handler(func=lambda message: message.text == "Зафіксувати вклад")
def handle_add_expense(message):
  user_id = message.from_user.id
  user_categories = user_data.get(user_id, {}).get('categories', {})

  if not user_categories:
    bot.send_message(user_id, "Спочатку додайте категорії витрат.")
    show_commands(user_id)
    return

  markup = tg.types.ReplyKeyboardMarkup(resize_keyboard=True)
  for category in user_categories:
    markup.add(category)

  markup.add("Назад")
  bot.send_message(user_id, "Оберіть категорію:", reply_markup=markup)
  bot.register_next_step_handler(message, process_choose_category)

def process_choose_category(message):
  user_id = message.from_user.id
  user_categories = user_data.get(user_id, {}).get('categories', {})
```

```
  user_expenses = user_data.get(user_id, {}).get('expenses', {})
  selected_category = message.text

  if selected_category == "Назад":
    show_commands(user_id)
    return

  if selected_category not in user_categories:
    bot.send_message(
        user_id,
        "Обраної категорії не існує. Оберіть існуючу категорію або додайте нову."
    )
    show_commands(user_id)
  else:
    if selected_category not in user_expenses:
      user_expenses[selected_category] = []
    bot.send_message(user_id, f"Введіть суму вкладу для'{selected_category}':")
    bot.register_next_step_handler(message, process_add_expense,
                                   selected_category)

def process_add_expense(message, selected_category):
  user_id = message.from_user.id
  user_expenses = user_data.get(user_id, {}).get('expenses', {})
  amount_text = message.text

  if not amount_text.isdigit():
    bot.send_message(
        user_id,
        "Вклад повинен бути додатнім числом. Оберіть категорію повторно.")
    bot.register_next_step_handler(message, process_choose_category)
    return

  amount = int(amount_text)
  user_expenses[selected_category].append(amount)
  bot.send_message(
      user_id,
      f"Вклад {amount} зафіксовано для категорії'{selected_category}'.")
  show_commands(user_id)

# Обробник команди "Видалити категорію"
@bot.message_handler(func=lambda message: message.text == "Видалити категорію")
def handle_delete_category(message):
  user_id = message.from_user.id
  user_categories = user_data.get(user_id, {}).get('categories', {})

  if not user_categories:
    bot.send_message(user_id, "Ви не маєте категорій для видалення.")
    show_commands(user_id)
    return

  markup = tg.types.ReplyKeyboardMarkup(resize_keyboard=True)
  for category in user_categories:
    markup.add(category)

  markup.add("Назад")
  bot.send_message(user_id,
```

```
            "Оберіть категорію для видалення:",
            reply_markup=markup)
  bot.register_next_step_handler(message, process_delete_category)

def process_delete_category(message):
  user_id = message.from_user.id
  user_categories = user_data[user_id]['categories']
  selected_category = message.text

  if selected_category == "Назад":
    show_commands(user_id)
    return

  if selected_category not in user_categories:
    bot.send_message(
       user_id, "Обранної категорії не існує. Оберіть існуючу категорію.")
    show_commands(user_id)
    return

  if selected_category in user_data[user_id]['expenses']:
    del user_data[user_id]['expenses'][selected_category]

  del user_data[user_id]['categories'][selected_category]
  bot.send_message(user_id, f"Категорію '{selected_category}' видалено.")
  show_commands(user_id)

# Кнопка "Обмеження"
@bot.message_handler(func=lambda message: message.text == "Обмеження")
def handle_budget_limit(message):
  user_id = message.from_user.id
  user_categories = user_data.get(user_id, {}).get('categories', {})

  if not user_categories:
    bot.send_message(user_id, "Спочатку додайте категорії витрат.")
    show_commands(user_id)
    return

  markup = tg.types.ReplyKeyboardMarkup(resize_keyboard=True)
  for category in user_categories:
    markup.add(category)

  markup.add("Назад")
  bot.send_message(user_id,
            "Оберіть категорію для встановлення обмеження:",
            reply_markup=markup)
  bot.register_next_step_handler(message, process_set_budget_limit)

def process_set_budget_limit(message):
  user_id = message.from_user.id
  user_categories = user_data.get(user_id, {}).get('categories', {})
  selected_category = message.text

  if selected_category.lower() == 'назад':
    show_commands(user_id)
    return
```

```
  if selected_category not in user_categories:
    bot.send_message(user_id,
                     "Обранної категорії не існує. Оберіть діючу категорію.")
    show_commands(user_id)
  else:
    if user_data[user_id]['categories'][selected_category]:
      bot.send_message(
          user_id,
          f"Обмеження на місяць для категорії '{selected_category}' вже встановлено."
      )
      show_commands(user_id)
    else:
      bot.send_message(
          user_id,
          f"Встановіть обмеження для категорії '{selected_category}' на місяць (введіть число):"
      )
      bot.register_next_step_handler(message, process_set_limit_value,
                                     selected_category)

def process_set_limit_value(message, selected_category):
  user_id = message.from_user.id
  user_categories = user_data.get(user_id, {}).get('categories', {})
  limit_text = message.text

  if limit_text.lower() == 'назад':
    show_commands(user_id)
    return

  if not limit_text.isdigit():
    bot.send_message(
        user_id,
        "Обмеження повинно бути додатнім числом. Введіть дані коректно.")
    show_commands(user_id)
    return

  limit = int(limit_text)
  user_data[user_id]['categories'][selected_category].append(limit)
  bot.send_message(
      user_id,
      f"Обмеження для категорії '{selected_category}' встановлено на {limit} на місяць."
  )
  show_commands(user_id)

# Обробник команди "Виправити вклад"
@bot.message_handler(func=lambda message: message.text == "Виправити вклад")
def process_correct_expenses(message):
  user_id = message.from_user.id
  user_categories = user_data.get(user_id, {}).get('categories', {})
  user_expenses = user_data.get(user_id, {}).get('expenses', {})

  if not user_categories and not any(user_expenses.values()):
    bot.send_message(user_id, "Спочатку додайте категорії витрат.")
    show_commands(user_id)
    return

  markup = tg.types.ReplyKeyboardMarkup(resize_keyboard=True)
```

```
  for category in user_categories:
    markup.add(category)

  markup.add("Назад")
  bot.send_message(user_id,
                   "Оберіть категорію для редагування:",
                   reply_markup=markup)
  bot.register_next_step_handler(message, process_correct_expenses_selection)

def process_correct_expenses_selection(message):
  user_id = message.from_user.id
  user_categories = user_data.get(user_id, {}).get('categories', {})
  user_expenses = user_data.get(user_id, {}).get('expenses', {})

  selected_category = message.text
  if selected_category.lower() == 'назад':
    show_commands(user_id)
    return

  if selected_category not in user_expenses:
    bot.send_message(
        user_id,
        f"Категорія '{selected_category}' не має даних для виправлення.")
    show_commands(user_id)
    return

  user_expenses = user_data[user_id]['expenses'][selected_category]

  bot.send_message(user_id, f"Дані категорії '{selected_category}':")
  for idx, expense in enumerate(user_expenses, start=1):
    bot.send_message(user_id, f"{idx}. {expense}")

  bot.send_message(user_id, "Введіть номер(№) даних для виправлення:")
  bot.register_next_step_handler(message, process_correct_data,
                                 selected_category)

def process_correct_data(message, selected_category):
  user_id = message.from_user.id
  user_expenses = user_data.get(user_id, {}).get('expenses', {})

  try:
    expense_number = int(message.text)
  except ValueError:
    bot.send_message(user_id, "Ви обрали некоректний номер(№).")
    show_commands(user_id)
    return

  if expense_number < 1 or expense_number > len(
      user_expenses[selected_category]):
    bot.send_message(user_id, "Вы обрали некоректний номер.")
    show_commands(user_id)
    return

  # Виправлення даних
  old_expense = user_expenses[selected_category][expense_number - 1]
  bot.send_message(
```

```
    user_id, f"Введіте нове додатнє число для виправлення '{old_expense}':")
  bot.register_next_step_handler(message, process_correct_expense_value,
                                 selected_category, expense_number)

def process_correct_expense_value(message, selected_category, expense_number):
  user_id = message.from_user.id
  user_expenses = user_data.get(user_id, {}).get('expenses', {})
  expense_text = message.text

  if expense_text.lower() == 'назад':
    show_commands(user_id)
    return

  try:
    new_expense = int(expense_text)
  except ValueError:
    bot.send_message(user_id,
                     "Ви ввели некоректне значення. Введіть додатнє число.")
    show_commands(user_id)
    return

  if new_expense <= 0:
    bot.send_message(
      user_id,
      "Значення повинно бути додатнім числом. Введіть додатнє число.")
    show_commands(user_id)
    return

  old_expense = user_expenses[selected_category][expense_number - 1]
  user_expenses[selected_category][expense_number - 1] = new_expense
  bot.send_message(
    user_id,
    f"Значення '{old_expense}' в категорії '{selected_category}' успішно змінено на '{new_expense}'."
  )
  show_commands(user_id)
# Обробник команди "Виправити обмеження"
@bot.message_handler(func=lambda message: message.text == "Виправити обмеження"
                     )
def process_correct_limit(message):
  user_id = message.from_user.id
  user_categories = user_data.get(user_id, {}).get('categories', {})

  if not user_categories:
    bot.send_message(user_id, "Спочатку додайте категорії витрат.")
    show_commands(user_id)
    return

  markup = tg.types.ReplyKeyboardMarkup(resize_keyboard=True)
  for category in user_categories:
    markup.add(category)

  markup.add("Назад")
  bot.send_message(
    user_id,
    "Оберіть категорію, для якої потрібно встановити обмеження:",
    reply_markup=markup)
```

```
  # Стан
  bot.register_next_step_handler(message,
                           process_choose_category_for_limit_correction)

def process_choose_category_for_limit_correction(message):
  user_id = message.from_user.id
  user_categories = user_data.get(user_id, {}).get('categories', {})
  selected_category = message.text

  print(f"Debug: selected_category = {selected_category}")

  if selected_category.lower() == 'отмена':
    show_commands(user_id)
    return

  if selected_category not in user_categories:
    bot.send_message(user_id,
                "Обранної категорії не існує. Оберіть діючу категорію.")
    show_commands(user_id)
    return

  if not user_data[user_id]['categories'][selected_category]:
    bot.send_message(
       user_id,
       f"Для категорії '{selected_category}' обмеження на місяць поки що не встановлено. Оберіть іншу
категорію."
    )
    show_commands(user_id)
    return

  current_limit = user_data[user_id]['categories'][selected_category][0]
  bot.send_message(
     user_id,
     f"Діюче обмеження для категорії '{selected_category}' на місяць: {current_limit}. Встановіть нове
обмеження:"
  )

  # Стан
  bot.register_next_step_handler(message, process_correct_limit_value,
                           selected_category)

def process_correct_limit_value(message, selected_category):
  user_id = message.from_user.id
  user_categories = user_data.get(user_id, {}).get('categories', {})
  limit_text = message.text

  print(f"Debug: limit_text = {limit_text}")

  if limit_text.lower() == 'назад':
    show_commands(user_id)
    return

  if not limit_text.isdigit():
    bot.send_message(
       user_id,
```

```
        "Обмеження повинно бути додатнім числом. Введіть коректне число.")
    show_commands(user_id)
    return

  limit = int(limit_text)
  user_data[user_id]['categories'][selected_category][0] = limit
  bot.send_message(
      user_id,
      f"Обмеження для категорії '{selected_category}' успішно виправлено на {limit}."
  )
  show_commands(user_id)

# Обработчик команды "Статистика"
@bot.message_handler(func=lambda message: message.text == "Статистика")
def handle_statistics(message):
  user_id = message.from_user.id
  user_categories = user_data.get(user_id, {}).get('categories', {})
  user_expenses = user_data.get(user_id, {}).get('expenses', {})

  if not user_categories:
    bot.send_message(user_id, "Спочатку додайте категорії витрат.")
    show_commands(user_id)
    return

  today = datetime.date.today()
  current_month = today.month
  current_year = today.year

  daily_total = defaultdict(int)
  monthly_total = defaultdict(int)
  yearly_total = defaultdict(int)
  exceeded_limit_categories = []

  for category, expenses in user_expenses.items():
    if not expenses:
      daily_total[category] = monthly_total[category] = yearly_total[
          category] = 0
    else:
      for expense in expenses:
        daily_total[
            category] += expense if category in user_expenses and expense in user_expenses[
                category] else 0
        monthly_total[
            category] += expense if category in user_expenses and expense in user_expenses[
                category] else 0
        yearly_total[
            category] += expense if category in user_expenses and expense in user_expenses[
                category] else 0

      # Перевірка перевищення встановленного обмеження
      limit_list = user_data[user_id]['categories'].get(category, [])
      limit = limit_list[0] if limit_list else None

      if limit is not None and monthly_total[category] > limit:
        exceeded_limit_categories.append(category)
```

```
  # Показ результату користувачу
  result_message = "Статистика витрат:\n"
  for category in user_categories:
    result_message += f"{category}:\n"
    result_message += f"День: {daily_total[category]}\n"
    result_message += f"Місяць: {monthly_total[category]}\n"
    result_message += f"Рік: {yearly_total[category]}\n"
    # Показ обмежень
    limit_list = user_data[user_id]['categories'].get(category, [])
    limit = limit_list[0] if limit_list else None
    result_message += f"Додакові дані:\n Обмеження: {limit if limit is not None else 'відсутні'}\n\n"

  bot.send_message(user_id, result_message.strip())

  # Перевірка перевищення обмежень
  if exceeded_limit_categories:
    bot.send_message(
        user_id,
        "Ви перевищили встановлене обмеження. Щоб уникнути такого надалі, пропоную ознайомитися з
порадами щодо фінансової грамотності за посиланням: https://t.me/mini_piggy_bank_channel/4"
    )

  show_commands(user_id)

# Обробник команди "Зберегти дані"
@bot.message_handler(func=lambda message: message.text == "Зберегти дані")
def handle_save_data(message):
  user_id = message.from_user.id
  save_data_to_excel(user_id)

  # Додавання збреження обмежень
  user_categories = user_data.get(user_id, {}).get('categories', {})
  result_message = "Обмеження:\n"
  for category, limit in user_categories.items():
    result_message += f"{category}: {limit[0] if limit else 'Не встановлено'}\n"

  bot.send_message(user_id, result_message.strip())
  show_commands(user_id)

# Обробник команди "Очистити історію"
@bot.message_handler(func=lambda message: message.text == "Очистити історію")
def handle_reset_data(message):
  user_id = message.from_user.id
  confirmation_markup = tg.types.ReplyKeyboardMarkup(resize_keyboard=True)
  confirmation_markup.row("Так", "Ні")
  bot.send_message(user_id,
                   "Дані будуть видалені безповоротно. Ви впевнені?",
                   reply_markup=confirmation_markup)
  bot.register_next_step_handler(message, process_reset_confirmation)

def process_reset_confirmation(message):
  user_id = message.from_user.id
  if message.text == "Так":
    if user_id in user_data:
      del user_data[user_id]
    bot.send_message(
```

```
        user_id, "Історію видалено. Для перезапуску бота натисніть /start.")
    else:
        show_commands(user_id)

# Обробник команди "Накопичити на мрію"
@bot.message_handler(func=lambda message: message.text == "Накопичити на мрію")
def handle_dream_button(message):
    user_id = message.from_user.id
    user_dreams = user_data.get(user_id, {}).get('dreams', {})

    if not user_dreams:
        bot.send_message(user_id, "Я допоможу Вам накопичити на мрію. Опишіть її:")
        bot.register_next_step_handler(message, process_dream_name)

        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        item_cancel = types.KeyboardButton("назад")
        markup.add(item_cancel)
    else:
        dream_name = user_dreams.get('name', '')
        bot.send_message(user_id, f"Ваша мрія: {dream_name}")
        bot.send_message(
            user_id,
            "Відмітьте яку суму вдалось зберегти на цей раз або введіть 'назад' для повернення до меню:"
        )
        bot.register_next_step_handler(message, process_add_savings, dream_name)

# Функція для обробки вводу назви мрії
def process_dream_name(message):
    user_id = message.from_user.id
    dream_name = message.text

    if user_id not in user_data:
        user_data[user_id] = {}
    user_data[user_id]['dreams'] = {'name': dream_name, 'savings': []}
    bot.send_message(user_id, f"Ваша мрія {dream_name}")
    bot.send_message(
        user_id,
        "Відмітьте яку суму вдалось зберегти на цей раз  або введіть 'назад' для повернення до меню:"
    )
    bot.register_next_step_handler(message, process_add_savings, dream_name)

# Функція для обробки вводу числа для мрії
def process_add_savings(message, dream_name):
    user_id = message.from_user.id
    user_savings = user_data.get(user_id, {}).get('dreams',
                                                  {}).get('savings', [])

    savings_text = message.text

    if savings_text.lower() == 'назад':
        show_commands(user_id)
        return

    if not savings_text.isdigit():
        bot.send_message(user_id,
                         "Число повинно бути додатнім. Введіть додатнє число.")
```

```
    bot.register_next_step_handler(message, process_add_savings, dream_name)
    return

  savings = int(savings_text)
  if savings <= 0:
    bot.send_message(user_id,
                     "Число повинно бути додатнім. Введіть додатнє число.")
    bot.register_next_step_handler(message, process_add_savings, dream_name)
    return

  user_savings.append(savings)
  total_savings = sum(user_savings)

  bot.send_message(user_id, f"Наразі Ви накопичили: {total_savings}")
  show_commands(user_id)

# Обработчик команды "Скорегувати мрію"
@bot.message_handler(func=lambda message: message.text == "Скорегувати мрію")
def handle_change_dream(message):
user_id = message.from_user.id
  bot.send_message(user_id, "Опишіть нову мрію:")
  bot.register_next_step_handler(message, process_change_dream)

# Функція для обробки вводу нового значення для мрії
def process_change_dream(message):
  user_id = message.from_user.id
  new_dream_name = message.text

  if user_id not in user_data:
    user_data[user_id] = {}

  if 'dreams' not in user_data[user_id]:
    user_data[user_id]['dreams'] = {}

  user_data[user_id]['dreams']['savings'] = []
  user_data[user_id]['dreams']['name'] = new_dream_name

  bot.send_message(user_id, f"Ваша мрія: {new_dream_name}")
  bot.send_message(
      user_id,
      "Відмітьте яку суму вдалось зберегти на цей раз  або введіть 'назад' для повернення до меню:"
  )
  bot.register_next_step_handler(message,
                                 process_add_savings,
                                 dream_name=new_dream_name)

  markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
  item_cancel = types.KeyboardButton("назад")
  markup.add(item_cancel)
  bot.send_message(user_id,
                   "Якщо хочете повернутись до меню, оберіть 'назад'.",
                   reply_markup=markup)

if __name__ == "__main__":
  bot.infinity_polling(none_stop=True)
```