

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інтерактивний сервіс управління особистими фінансовими ресурсами»

Студента 2 курсу, 3м групи,
спеціальності 121 «Інженерія
програмного забезпечення»
освітньої програми «Інженерія
програмного забезпечення»

підпис студента

Залеського Нікіти
Олександровича

Науковий керівник
кандидат фізико-
математичних наук,
доцент кафедри інженерії
програмного забезпечення та
кібербезпеки

підпис керівника

Горохова Олена
Миколаївна

Гарант освітньої програми
кандидат педагогічних наук,
доцент кафедри інженерії
програмного забезпечення та
кібербезпеки

підпис гаранта

Котенко Наталія
Олексіївна

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення та кібербезпеки

Освітній ступінь магістр

Освітня програма «Інженерія програмного забезпечення»

Затверджую

Зав. кафедри інженерії програмного
забезпечення та кібербезпеки

Криворучко О. В.

«13» грудня 2023 р.

Завдання на кваліфікаційну роботу студентів

Залеському Нікіті Олександровичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Інтерактивний сервіс управління особистими фінансовими ресурсами»

Затверджена наказом ректора від «27» листопада 2023 р. № 4193

2. Строк здачі студентом закінченої роботи 15 листопада 2024

3. Цільова установка та вихідні дані до роботи

Мета роботи – дослідження і реалізація нового продукту у темі управління особистими фінансами у інтернеті.

Об'єкт дослідження – Telegram-бот для допомоги у управлінні особистими фінансами

Предмет дослідження – розробка та реалізація телеграм-боту для управління особистими фінансами

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)
ВСТУП

РОЗДІЛ 1 АНАЛІЗ ГАЛУЗІ ТЕЛЕГРАМ БОТІВ ДЛЯ УПРАВЛІННЯ
ОСОБИСТИМИ ФІНАНСОВИМИ РЕСУРСАМИ ТА ПРЕДМЕТНОЇ
ОБЛАСТІ

1.1. Аналіз галузі ботів для управління фінансами.

1.2. Аналіз сучасних проектних рішень

1.3. Постановка завдання та вимог до проектування

1.4. Висновки до розділу 1

РОЗДІЛ 2 АНАЛІЗ ВИМОГ ТА МОДЕЛЮВАННЯ

2.1. Аналіз основних вимог

2.2. Функціональні вимоги

2.3. Алгоритм роботи Telegram-боту, проектування.

2.4. Розробка UML-діаграм

2.5. Структура програмного комплексу

2.6. Архітектура проекту

2.7. Технології розробки

2.8. Висновки до розділу 2

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ТА ПРАКТИЧНЕ
ВПРОВАДЖЕННЯ

3.1. Реалізація Telegram-боту

3.2. Керівництво користувача

3.4. Висновок до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання роботи

№ пор.	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	<i>Вибір теми випускної кваліфікаційної роботи</i>	07.11.2023	07.11.2023
2.	<i>Розробка та затвердження завдання на роботу магістра (стац/заоч)</i>	13.12.2023	13.12.2023
3.	<i>Вступ та перелік літературних джерел</i>	22.02.2024	22.02.2024
4.	<i>Розробка технічного завдання</i>	14.03.2024	14.03.2024
5.	<i>Розділ 1. АНАЛІЗ ГАЛУЗІ ТЕЛЕГРАМ БОТІВ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСОВИМИ РЕСУРСАМИ ТА ПРЕДМЕТНОЇ ОБЛАСТІ</i>	10.04.2024	10.04.2024
6.	<i>Розділ 2. АНАЛІЗ ВИМОГ ТА МОДЕЛЮВАННЯ</i>	23.05.2024	23.05.2024
7.	<i>Розділ 3. РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ТА ПРАКТИЧНЕ ВПРОВАДЖЕННЯ</i>	05.09.2024	05.09.2024
8.	<i>Розробка програми та методики тестування</i>	27.09.2024	27.09.2024
9.	<i>Написання наукової статті</i>	16.04.2024	16.04.2024
10.	<i>Керівництво користувача</i>	11.10.2024	11.10.2024
11.	<i>Висновки та пропозиції</i>	16.10.2024	16.10.2024
12.	<i>Здача випускної кваліфікаційної роботи на кафедрі (перша перевірка)</i>	18.10.2024	18.10.2024
13.	<i>Підготовка реферату та презентації доповіді</i>	28.10.2024	28.10.2024
14.	<i>Попередній захист випускної кваліфікаційної роботи</i>	29.10.2024 – 31.10.2024	29.10.2024 – 31.10.2024
15.	<i>Здача зброшурованої випускної кваліфікаційної роботи</i>	15.11.2024	15.11.2024
16.	<i>Зовнішнє рецензування випускної кваліфікаційної роботи</i>	28.10.2024	28.10.2024
17.	<i>Підготовка до публічного захисту випускної кваліфікаційної роботи</i>	02.12.2024- 03.12.2024	02.12.2024- 03.12.2024

7. Дата видачі завдання «13» грудня 2023 р.

8. Науковий керівник кваліфікаційної роботи Горохова О.М.

(прізвище, ініціали, підпис)

9. Гарант освітньої програми Котенко Н.О.

(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Залеський Н.О.

(прізвище, ініціали, підпис)

АНОТАЦІЯ

Відповідно до мети дослідження робота присвячена дослідженню і реалізації нового продукту у темі управління особистими фінансами у інтернеті. Випускна кваліфікаційна робота на тему «Інтерактивний сервіс управління особистими фінансовими ресурсами» містить 52 сторінки, 27 рисунків і 1 таблицю. Перелік використаних джерел налічує 26 найменувань.

Були досліджені існуючі інструменти для управління особистими фінансами, а також їхні недоліки в контексті зручності використання та функціональних можливостей.

В результаті було створено функціональний та інтуїтивно зрозумілий телеграм-бот, який дозволить користувачам легко управляти своїми фінансами, включаючи доходи, витрати, криптовалюти, а також інші аспекти фінансового планування.

Ключові слова: дизайн, інтерфейс, проектування, розробка, фінанси, аналіз, бот, курс валют, користувач, криптовалюти.

ABSTRACT

According to the purpose of the study, this work is devoted to the research and implementation of a new product in the area of managing personal finances on the Internet. The graduation qualification work, titled "Interactive Service for Managing Personal Financial Resources," consists of 52 pages, 27 figures, and 1 table. The list of sources includes 26 references.

Existing personal finance management tools were analyzed, along with their shortcomings in terms of usability and functionality.

As a result, a functional and intuitive Telegram bot was developed, enabling users to efficiently manage their finances, including income, expenses, cryptocurrencies, and other aspects of financial planning.

Keywords: design, interface, development, finance, analysis, bot, exchange rate, user, cryptocurrencies.

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ,
СКОРОЧЕНЬ І ТЕРМІНІВ**

ШІ – штучний інтелект;

API – application programming interface;

СТА – call to action;

UML – unified modeling language;

БД – база даних;

HTTP – HyperText Transfer Protocol;

PEP – Python Enhanced Proposal.

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1 АНАЛІЗ ГАЛУЗІ ТЕЛЕГРАМ БОТІВ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСОВИМИ РЕСУРСАМИ ТА ПРЕДМЕТНОЇ ОБЛАСТІ.....	8
1.1. Аналіз галузі ботів для управління фінансами.	8
1.2. Аналіз сучасних проектних рішень.....	11
1.3. Постановка завдання та вимог до проектування	15
1.4. Висновки до розділу 1	16
РОЗДІЛ 2 АНАЛІЗ ВИМОГ ТА МОДЕЛЮВАННЯ	17
2.1. Аналіз основних вимог.....	17
2.2. Функціональні вимоги.	18
2.3. Алгоритм роботи Telegram-боту, проектування.	20
2.4. Розробка UML-діаграм.....	24
2.5. Структура програмного комплексу.	28
2.6. Архітектура проекту.....	31
2.7. Технології розробки.	33
2.8. Висновки до розділу 2.....	35
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ТА ПРАКТИЧНЕ ВПРОВАДЖЕННЯ.....	36
3.1. Реалізація Telegram-боту.....	36
3.2. Керівництво користувача.	43
3.4. Висновок до розділу 3	48
ВИСНОВКИ ТА ПРОПОЗИЦІЇ	50
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	51
ДОДАТКИ	

ВСТУП

У сучасному світі управління особистими фінансами стає важливою складовою повсякденного життя, і тому попит на зручні та ефективні інструменти для обліку і контролю фінансів зростає. Одним із перспективних рішень у цій сфері є створення інтерактивного сервісу у вигляді телеграм-бота. Такий інструмент дозволяє користувачам вести облік фінансових операцій, планувати витрати та аналізувати фінансові дані, не залишаючи межі звичного телеграм-додатку.

Мета даної кваліфікаційної роботи полягає в розробці та впровадженні телеграм-бота для управління фінансами, який допоможе користувачам контролювати свої фінанси, оптимізувати витрати та більш усвідомлено планувати бюджет. Бот буде підключений до бази актуальних курсів валют та криптовалют і матиме можливість автоматичного оновлення цих даних в режимі реального часу. Це дозволить користувачам управляти своїми фінансовими ресурсами максимально ефективно та зручно.

У рамках розробки телеграм-боту, передбачається вирішення таких завдань:

1. Створення бази даних для зберігання всієї інформації про фінансові операції користувачів.
2. Розробка функцій для введення, обліку та управління доходами і витратами з можливістю їх категоризації.
3. Реалізація можливості автоматичного оновлення курсів валют та відслідковування змін на ринку криптовалют.
4. Забезпечення телеграм-бота простим, інтуїтивно зрозумілим інтерфейсом для зручного використання.
5. Інтеграція інтерактивних елементів для виконання різних фінансових операцій, таких як кнопки для швидкого введення даних.

Виконання даної кваліфікаційної роботи сприятиме підвищенню ефективності управління особистими фінансами, надаючи користувачам

зручний інструмент для повноцінного контролю та планування своїх фінансових ресурсів. Телеграм-бот стане незамінним помічником як для тих, хто прагне мати повний контроль над своїми витратами та доходами, так і для тих, хто цікавиться криптовалютами й бажає зручно управляти своїми інвестиційними портфелями.

Об'єктом дослідження даної кваліфікаційної роботи є існуючі інструменти для управління особистими фінансами, а також їхні недоліки в контексті зручності використання та функціональних можливостей.

Предметом дослідження є процес розробки і впровадження телеграм-бота, що забезпечує інтерактивне управління особистими фінансовими ресурсами. Проект зосереджується на розробці та інтеграції фінансових інструментів, а також на побудові логіки обробки фінансових запитів користувачів.

Основними методами дослідження в рамках цієї роботи є аналіз наукової літератури, огляд існуючих публікацій і статей, пов'язаних із тематикою управління фінансами, а також дослідження актуальних рішень у сфері особистих фінансів.

Результатами роботи є функціональний та інтуїтивно зрозумілий телеграм-бот, який дозволить користувачам легко управляти своїми фінансами, включаючи доходи, витрати, криптовалюти, а також інші аспекти фінансового планування.

Опубліковано статтю "ІНТЕРАКТИВНИЙ СЕРВІС УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСОВИМИ РЕСУРСАМИ" у збірнику "Програмування та захист інформації", в якій розглянуто основні засади побудови та функціонування інтерактивного сервісу управління особистими фінансовими ресурсами, а також зазначено переваги застосування програмних продуктів в управлінні особистими фінансами [3].

РОЗДІЛ 1

АНАЛІЗ ГАЛУЗІ ТЕЛЕГРАМ БОТІВ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСОВИМИ РЕСУРСАМИ ТА ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз галузі ботів для управління фінансами.

"Боти — це невеликі програми, які повністю працюють у програмі Telegram. Користувачі взаємодіють із ботами через гнучкі інтерфейси, які можуть підтримувати будь-які завдання чи послуги" [7]. Завдяки своїм простим і зрозумілим інтерфейсам, боти можуть відповідати на запити, пропонувати користувачам послуги чи навіть розважати. Вони працюють за принципом звичайних додатків, але без необхідності їхнього встановлення на пристрій.

Попри те, що Telegram був заснований російськими розробниками, цей месенджер не заборонений в Україні. Його популярність у країні залишається високою, і він є важливим інструментом для комунікації, обміну інформацією та організації заходів. Українська влада на даний момент не вводила обмежень на використання Telegram, оскільки платформа забезпечує конфіденційність даних і широко використовується як для особистого спілкування, так і для офіційних каналів новин. "Посадовцям забороняється залишати свої робочі комп'ютери без нагляду та використовувати Telegram на робочих пристроях. При цьому для особистих цілей месенджер залишається доступним" [11].

Боти можуть виконувати практично будь-які завдання, які можуть виконати користувачі в онлайн-сервісах, такі як навчання, розваги, пошук інформації, трансляція, нагадування та багато іншого. Вони є зручним інтерфейсом для роботи з веб-службами.

Головною перевагою ботів є значне спрощення взаємодії з цифровими послугами. Замість того, щоб встановлювати окремі додатки та витратити час на їхнє налаштування, користувачі можуть виконувати ті ж самі дії

просто через чат, надсилаючи команди. Telegram став одним із перших месенджерів, який запровадив підтримку ботів, що зробило їх дуже популярними і призвело до їх поширення на інші платформи.

Для початку роботи з ботом необхідно мати акаунт у Telegram. Взаємодія з ботами відбувається в форматі чатів, але замість людини на іншому кінці працює програма, яка може використовувати алгоритми штучного інтелекту. На відміну від звичайних користувачів, боти не мають статусу присутності в мережі і позначаються як "роботи". Щоб розпочати взаємодію, необхідно або додати бота до групи, або почати спілкування з ним напямую. Назви ботів завжди закінчуються на "bot", наприклад, @FinanceBot.

Запущений у 2013 році, Telegram швидко зарекомендував себе як один з найбільш популярних месенджерів, особливо завдяки своїй мультиплатформенній підтримці. Однією з найцікавіших функцій месенджера є інтеграція ботів, які можна використовувати як у приватних чатах, так і в групах.

Telegram постійно розширює свої можливості, додаючи нові функції, завдяки чому утримує провідні позиції на ринку месенджерів. Однією з найкорисніших функцій стала підтримка ботів, яка значно спростила доступ до різноманітних онлайн-сервісів і дозволила користувачам вирішувати завдання безпосередньо в чаті.

Таблиця 1.1.

Класифікація функцій ботів

Класифікація функції	Варіація функції
Доступ	Особистий чат з ботом
	Бот, що доданий до чату користувачів
	Додаткові функції за підпискою
Інтерфейс	Командний і кнопочний

	Командний і текстовий
Принцип роботи	Шаблонний, має певний алгоритм
	З можливістю машинного навчання
Призначення	Зворотного зв'язку
	Функціональний

Джерело: побудовано автором

Telegram-боти відкривають перед користувачами широкі можливості для інтерактивного спілкування та обміну інформацією. Завдяки тому, що розробники можуть створювати спеціалізовані клавіатури для полегшення введення команд, користування ботами стає більш зручним і гнучким.

"Боти можуть інтерпретувати вільний текст, який вводять користувачі, але пропонувати конкретні пропозиції часто більш інтуїтивно зрозуміло — саме тут клавіатури, що налаштовуються, можуть бути надзвичайно корисними" [17].

У сучасному світі, де цифрові технології проникають в усі сфери життя, управління фінансовими ресурсами стає все більш популярною послугою. З розвитком мобільних додатків та онлайн-платформ з'явилася можливість керувати власними фінансами у будь-який момент та з будь-якої точки світу. Це надає користувачам змогу контролювати витрати і доходи, планувати бюджет, а також займатися інвестуванням.

Одним із ефективних підходів до управління фінансами стало використання чат-ботів. Вони забезпечують легкий у користуванні інтерфейс, що спрощує роботу з фінансовими інструментами. Завдяки різноманітним алгоритмам, боти здатні пропонувати користувачам індивідуальні рекомендації, аналізувати їх фінансову поведінку та пропонувати оптимальні рішення.

Дослідження сучасних сервісів для управління фінансами охоплює аналіз технологій, що використовуються в інтерактивних сервісах, зокрема чат-ботах, а також вивчення ключових вимог до управління особистими

фінансовими ресурсами. Це включає оцінку наявних на ринку рішень, їх переваг і недоліків, а також аналіз впливу інтерактивних сервісів на фінансову грамотність користувачів.

Інтеграція чат-ботів із зовнішніми фінансовими сервісами, такими як банківські додатки, платіжні системи та інвестиційні платформи, є важливим кроком для створення єдиного середовища для керування фінансовими операціями. Це значно підвищує ефективність та зручність використання сервісів, дозволяючи користувачам контролювати свої фінанси більш комплексно.

Отже, ринок сервісів для управління фінансами продовжує активно розвиватися, пропонуючи швидкі, зручні та безпечні інструменти для керування особистими фінансовими ресурсами. Правильний вибір таких сервісів та обґрунтований підхід до управління своїми коштами можуть сприяти заощадженню і раціональному використанню ресурсів у довготривалій перспективі.

1.2. Аналіз сучасних проектних рішень.

Сучасні сервіси активно впроваджують технології штучного інтелекту для автоматизації процесів та покращення користувацького досвіду. Чат-боти, які використовують ШІ, можуть відповідати на запити користувачів, надавати рекомендації щодо оптимізації витрат або інвестицій, а також нагадувати про регулярні платежі. Машинне навчання дозволяє аналізувати фінансову поведінку користувачів і пропонувати індивідуальні поради, що підвищує ефективність управління фінансами.

"Чат-боти в основному використовуються для покращення взаємодії з клієнтами, пропонуючи цілодобову підтримку клієнтів, але в економічно ефективний спосіб. Компанії також почали використовувати чат-боти для обслуговування внутрішніх клієнтів, обмінюючись знаннями та виконуючи рутинні завдання." [21]

Мобільні додатки є одним із найпопулярніших рішень для управління фінансами, оскільки дозволяють користувачам контролювати свої фінанси в будь-який час і з будь-якого місця. Наприклад, такі додатки, як Mint, YNAB (You Need A Budget) і Spendee, надають можливість інтеграції з банківськими рахунками, автоматичну категоризацію витрат, створення бюджетів і надання детальних звітів про фінансову активність.

Інтеграція з банківськими системами полегшує процес управління фінансами, дозволяючи автоматично отримувати дані про транзакції. Сервіси, такі як Plaid або TrueLayer, пропонують API, що дозволяє підключати банківські рахунки до фінансових додатків і автоматизувати процеси обліку доходів та витрат.

Персоналізація також стає важливим аспектом у фінансових додатках, оскільки сервіси використовують дані про фінансову поведінку користувачів для створення індивідуальних рекомендацій. Наприклад, деякі сервіси пропонують функції прогнозування витрат і доходів, що допомагає користувачам краще планувати свої фінансові показники та досягати поставлених цілей.

Безпека залишається ключовим аспектом інтерактивних сервісів, які працюють із чутливими фінансовими даними. Для забезпечення безпеки використовуються технології двофакторної аутентифікації, шифрування даних та інші заходи захисту. Наприклад, додатки на зразок Revolut дозволяють користувачам встановлювати ліміти на транзакції, блокувати карти або отримувати сповіщення про підозрілу активність.

У світлі глобалізації та зростання популярності криптовалют, сучасні сервіси підтримують роботу з різними валютами, включаючи криптовалюти. Додатки, такі як Coinbase і BlockFi, надають можливість керувати криптовалютними активами, дозволяючи користувачам відслідковувати та контролювати свої активи з одного інтерфейсу.

Приклади сучасних фінансових рішень включають Mint, що дозволяє інтегрувати банківські рахунки, кредитні карти та інші фінансові інструменти в одному місці, автоматично категоризуючи транзакції і надаючи персоналізовані поради. YNAB (You Need A Budget) допомагає користувачам ставити фінансові цілі та планувати витрати за допомогою системи управління бюджетом, пропонуючи також курси та вебінари для підвищення фінансової грамотності.

Також гарним прикладом можна назвати "FinTrackBot" , сервіс що включає в себе інструменти для обліку доходів та витрат, а також ведення статистики вашого фінансового стану. Користувач може легко контролювати свої розходи, планувати бюджет і визначати прибутковість різних інвестицій.

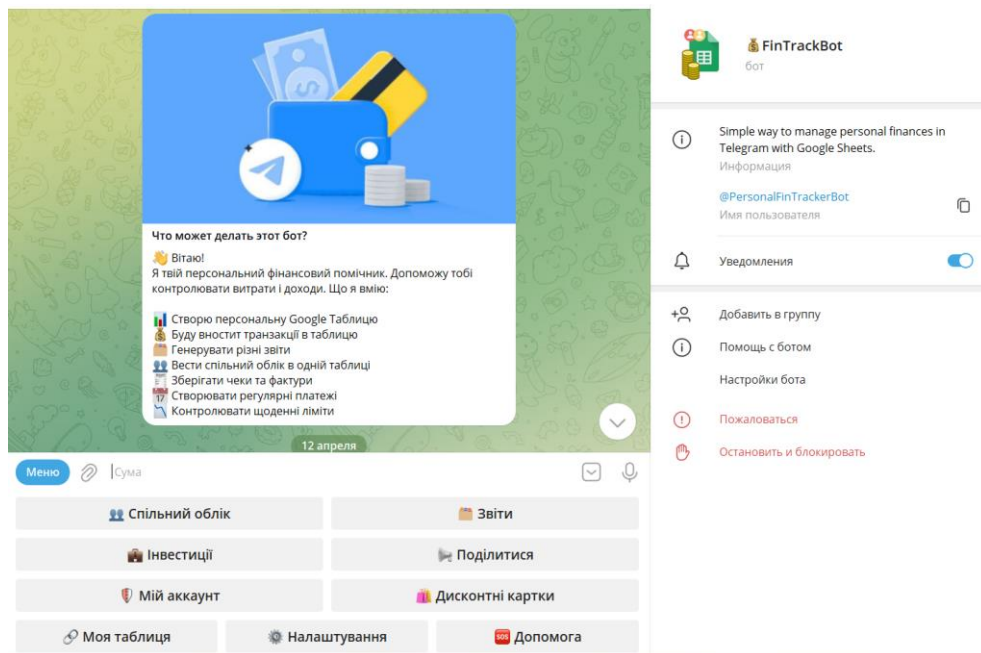


Рис. 1.1. Інтерфейс « FinTrackBot »
Джерело: знімок з екрану

Користувацький інтерфейс чат-бота для управління фінансами простий і інтуїтивно зрозумілий для звичайних користувачів. Після запуску бота користувач обирає мову спілкування, що дозволяє налаштувати зручну взаємодію з ботом. Далі автоматично створюється нова таблиця за

визначеним шаблоном, доступ до якої мають лише сам користувач і ті, кому він відкриє доступ. Бот також надає коротку інструкцію, яка містить огляд доступних команд і можливостей.

Одразу після запуску рекомендується встановити основну валюту для обліку фінансів, що забезпечить автоматичну конвертацію інших валют у задану. Наприклад, якщо валюта за замовчуванням – гривня, суми в доларах або інших валютах будуть автоматично перераховані в гривні. Такий підхід є дуже зручним для різних ситуацій, як-от оплата підписок на інтернет-сервіси або під час подорожей, коли користувачі стикаються з різними валютами.

Важливо зазначити, що таблиця, яка створюється при першому запуску бота, зберігається на серверах розробника. Це означає, що розробник може мати доступ до фінансових даних користувача, таких як доходи та витрати. Навіть якщо ви не надасте свій електронний лист, розробник все одно може побачити ваше ім'я в Telegram. Хоча поточна версія бота не оперує особливо чутливими даними, такими як детальна інформація про платежі, варто пам'ятати про можливі ризики, пов'язані з безпекою даних. Це особливо актуально, якщо функціонал бота буде розширюватися в майбутньому, і доведеться вживати додаткових заходів для захисту конфіденційної інформації користувачів.

"У міру того, як споживачі відходять від традиційних форм спілкування, багато експертів очікують, що методи спілкування на основі чатів зростатимуть. Організації все частіше використовують віртуальних помічників на основі чат-ботів для виконання простих завдань, дозволяючи агентам-людям зосередитися на інших обов'язках." [23].

Ці сервіси демонструють різні підходи до управління особистими фінансами та відповідають основним трендам у цій сфері — інтеграції з банківськими системами, підтримці криптовалют, персоналізації та конвертації валют.

Сучасні рішення у сфері інтерактивного управління фінансами спрямовані на максимальну зручність для користувачів. Вони забезпечують високу безпеку і пропонують персоналізовані функції завдяки технологіям штучного інтелекту та інтеграції з різними фінансовими системами.

1.3. Постановка завдання та вимог до проектування

У сучасному світі важко уявити життя без програм для управління фінансовими ресурсами. Вони стали невід'ємною частиною нашої повсякденності, допомагаючи ефективно контролювати витрати, доходи та планувати бюджет. Щоб відповідати сучасним реаліям і спростити життя користувачів, необхідно впроваджувати ці інструменти найбільш зручним і доступним способом.

Програми для управління фінансами повинні бути інтуїтивно зрозумілими, легкими у використанні та доступними на різних платформах. Лише таким чином можна підвищити якість життя користувачів, зробивши управління фінансами не лише ефективним, але й простим. Мета роботи – створення телеграм боту для сервісу управління особистими фінансовими ресурсами.

Відповідно до мети в проекті поставлені наступні завдання:

1. Побудувати зручного у використанні та просто в навігації бота для управління особистими фінансовими ресурсами.

2. Телеграм бот для конвертації валют має бути можливо використовувати на будь-якому пристрої, при умови підключення до інтернету та встановленому месенджеру.

3. Перетворити бот на інструмент для особистого використання, а не для комерційних фінансових операцій. Основна функція бота – допомога у веденні особистих фінансів, а не в комерційних транзакціях.

4. Лише декілька натискань клавіш, або команд і виконуються наступні дії:

– додавання та видалення витрат;

- перегляду статистики витрат за день та за місяць;
- роботи з категоріями витрат;
- отримання курсів валют та криптовалют;
- конвертація необхідної кількості криптовалют;
- розрахунок прибутку та збитку в категорії криптовалют;
- управління портфелем;
- деякі інші дії.

5. У головному вікні має відображатися головне меню бота і з можливістю переходу до інших меню. Телеграм бот має можливість одразу перейти до роботи боту при відкритті, а вже за кнопок у меню управління ботом вивести інструкції що можуть бути корисними користувачу.

1.4. Висновки до розділу 1

У першому розділі кваліфікаційної роботи було проведено детальний аналіз існуючих інструментів для управління фінансами, виявлено проблеми, пов'язані з їх використанням, а також досліджено можливості для вдосконалення. Аналіз засвідчив, що сучасні рішення мають значний потенціал для розвитку, зокрема в аспекті інтеграції нових технологій і спрощення користувацького досвіду.

Запропоноване рішення – створення телеграм-бота для управління особистими фінансовими ресурсами – спрямоване на задоволення потреб користувачів, які прагнуть мати доступ до простого й ефективного інструмента для контролю своїх фінансів. Бот надасть можливість легко вести облік витрат і доходів, а також здійснювати фінансові операції з традиційними валютами й криптовалютами. Це забезпечить користувачам зручний доступ до необхідних функцій у будь-який час і з будь-якого пристрою.

РОЗДІЛ 2

АНАЛІЗ ВИМОГ ТА МОДЕЛЮВАННЯ

2.1. Аналіз основних вимог.

Основною метою цього дослідження є розробка інтерактивного чат-бота для управління особистими фінансовими ресурсами на платформі Telegram. Цей бот надаватиме користувачам актуальну інформацію про фінансові показники, конвертацію валют та криптовалют, а також можливість управляти особистими фінансами.

У порівнянні з наявними джерелами, розроблений чат-бот повинен поєднувати в собі кілька важливих якостей: інформативність, актуальність, зручність, лаконічність, доступність, широке охоплення питань і зрозумілість.

Основна інформаційна функція чат-бота полягає в наданні точних даних про особисті фінанси, що дозволить користувачам управляти фінансами з високою точністю. Для цього бот має постійно оновлювати свою базу даних, оскільки фінансові показники, такі як курси валют і криптовалют, можуть швидко змінюватися.

Чат-бот повинен відповідати наступним вимогам:

- Актуальність: Постійне оновлення фінансових даних, щоб забезпечити користувачів найсвіжішою інформацією.
- Зручність: Інтерфейс має бути інтуїтивно зрозумілим, що забезпечить комфортне використання сервісу.
- Лаконічність: Інформація має бути чіткою та стислою, щоб користувачі могли швидко отримувати потрібні дані.
- Доступність: Бот повинен працювати на різних платформах, бути доступним за будь-якого рівня Інтернет-з'єднання і забезпечувати стабільну роботу.
- Широке охоплення: Чат-бот повинен підтримувати різноманітні валюти та криптовалюти для здійснення конвертації і аналізу.

- Зрозумілість: Бот має надавати інформацію зрозумілими термінами для широкого кола користувачів.

Ці критерії спрямовані на те, щоб зробити чат-бот ефективним інструментом для користувачів, які бажають легко контролювати свої фінансові ресурси.

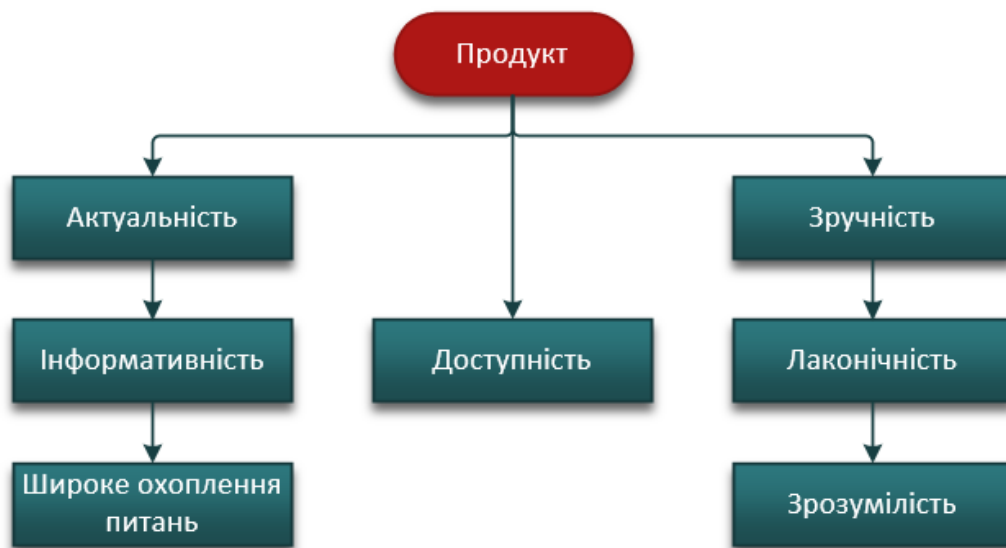


Рис.2.1. Основні вимоги до ПЗ в ієрархічній залежності

Джерело: побудовано автором

Окрім формалізації основних вимог до чат-бота для управління особистими фінансовими ресурсами, важливо враховувати взаємозв'язки між різними параметрами, які впливають на його функціональність. Ці параметри взаємодіють як у напрямку зверху вниз, так і знизу вверх. Наприклад, інформативність залежить як від актуальності даних, так і від широкого охоплення питань.

2.2. Функціональні вимоги.

Функціональні вимоги для чат-бота з управління фінансами це - облік витрат, доходів, конвертація валют, додавання або видалення криптовалют з портфеля. Вимоги можуть бути подані у вигляді правил або сценаріїв використання. Наприклад, система може мати такі вимоги, як надання

можливості користувачам створювати звіти про витрати за певний період або конвертувати задану суму у вибрану валюту на основі актуальних курсів.

Кожен сценарій або варіант використання детально описує процес взаємодії користувача з системою. Наприклад, сценарій, у якому користувач додає нову витрату, передбачає опис дій, які повинен виконати користувач, і які дані будуть введені. На основі функціональних вимог також формуються початкові прототипи системи, які служать керівництвом для того, як повинні виглядати та працювати основні функції на першому етапі.

Для чат-бота з управління фінансами прикладом функціональних вимог є така умова: бот повинен надавати користувачам можливість додавати витрати з вказівкою категорій та сум. Бот має автоматично оновлювати курси валют і криптовалют, надаючи актуальну інформацію для конвертації. Також користувачі повинні мати можливість створювати звіти витрат на заданий період часу. Крім того, бот повинен підтримувати інтеграцію з зовнішніми фінансовими API для отримання актуальних даних, а користувачі — мати можливість керувати своїм криптовалютним портфелем, додаючи та видаляючи активи.

Таким чином, функціональні вимоги є фундаментом для розробки програмного забезпечення і слугують інструментом для досягнення поставлених цілей проєкту.

До програмної системи поставлені наступні функціональні вимоги:

1. Реалізувати можливість додавання та видалення витрат.
2. Реалізувати можливість перегляду статистики витрат за день та за місяць.
3. Реалізувати можливість роботи з категоріями витрат.
4. Реалізувати можливість отримання курсів валют та криптовалют.
5. Реалізувати можливість конвертація необхідної кількості криптовалюти.

6. Реалізувати можливість розрахунків прибутку та збитку в категорії криптовалют.

7. Реалізувати можливість управління портфелем.

Серед технічних вимог потрібно виділити:

- надійність;
- технічну естетику та ергономіку;
- захист інформації;
- показ інформації при доступі до Інтернету;
- швидка відповідь на запити;
- зберігання інформації.

2.3. Алгоритм роботи Telegram-боту, проектування.

При проектуванні чат-бота важливо розглянути загальну структуру, яка є типовою для багатьох шаблонних чат-ботів, хоча функціонал може варіюватись. Для того, щоб проектувати будь-який чат-бот треба розглянути їх загальну структуру. Структура звичайних шаблонних ботів з меню – зазвичай проста, але при цьому досить розгалужена:

На рисунку 2.2 представлена загальна структура розробленого Telegram-боту. Після запуску бота зазвичай потім з'являється головне меню, а також очікуються дії користувача. Через кнопки в меню можна перейти до різних підменю з варіантами функцій.

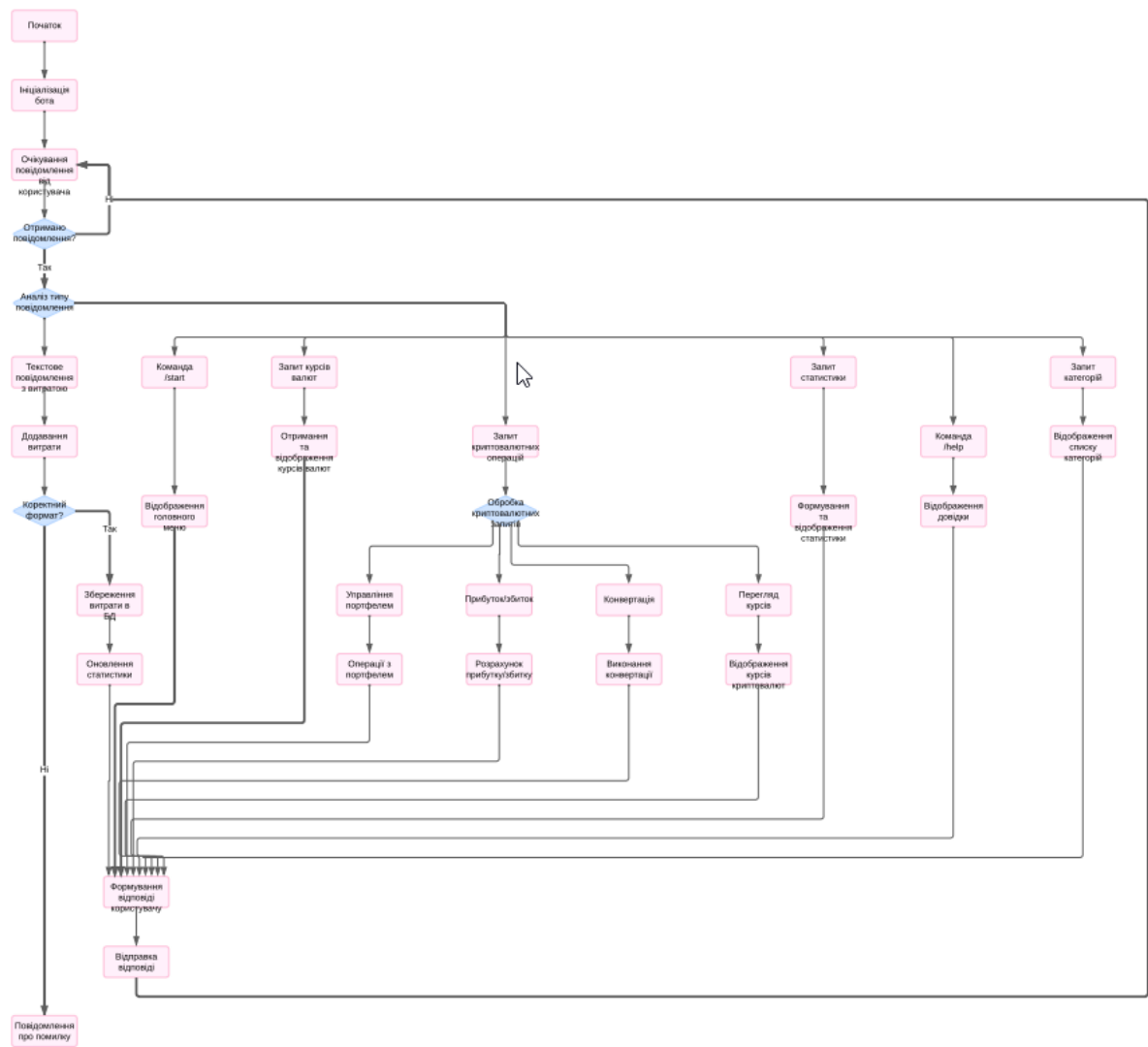


Рис.2.2. Алгоритм блок-схема роботи чат-бота

Джерело: побудовано автором

Після кожного блоку функцій користувач може знову скористатися функцією або повернутися до головного меню за бажанням. Якщо користувач не бажає отримувати додаткову інформацію, він може завершити роботу бота у меню програми. Важливо зауважити, що ця схема алгоритму є схематичною і надає лише уявлення про приблизний порядок роботи.

При створенні програми, а також для приблизного схематичного уявлення роботи блоків коду було розроблено алгоритмічний опис функції додавання витрат, а також блок-схему до неї на рисунку 2.3.

Алгоритмічний опис:

1. Отримати повідомлення від користувача
2. Перевірити формат повідомлення
3. Якщо формат правильний, розпарсити суму та категорію
4. Визначити відповідну категорію витрат
5. Додати витрату до бази даних
6. Повернути підтвердження користувачу

Блок-схема:

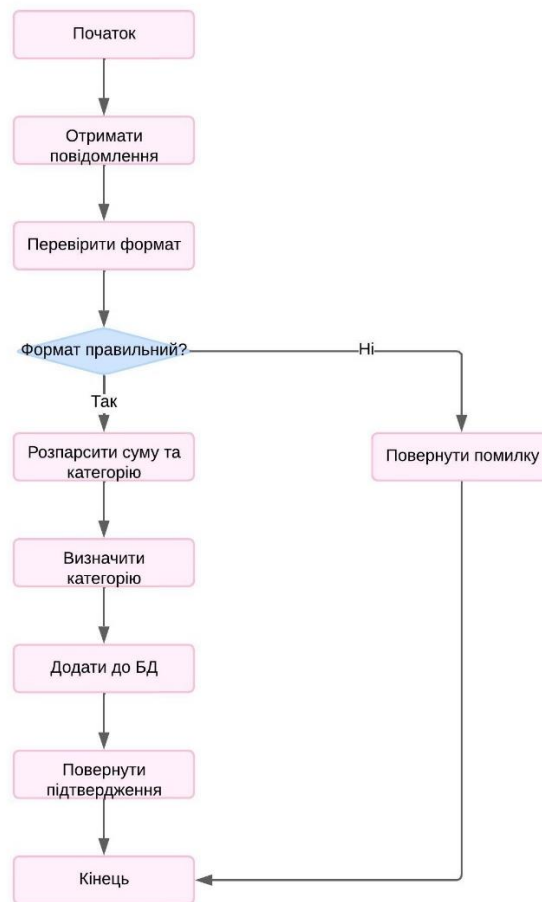


Рис.2.3. Алгоритм блок-схема роботи функції додавання витрат

Джерело: побудовано автором

Кодовий зразок створений на мові програмування Python:

```

def add_expense(raw_message: str) -> Expense:
    try:
        parsed_message =
        _parse_message(raw_message)
  
```

```

        category = Categories().get_category(parsed_message.category_text)
        inserted_row_id = db.insert("expense", {
            "amount": parsed_message.amount,
            "created": _get_now_formatted(),
            "category_codename": category.codename,
            "raw_text": raw_message
        })
        return Expense(id=None,
            amount=parsed_message.amount,
            category_name=category.name)
    except exceptions.NotCorrectMessage as e:
        raise e

```

Меню ботів зазвичай виглядає таким чином, що користувач обирає Call To Action (CTA). Більшість чат-ботів мають чітку мету і працюють за шаблоном, тому використання CTA є надзвичайно важливим під час розробки бота. Наприклад, у випадку припинення роботи бота можна запропонувати кілька варіантів виходу з ситуації, щоб користувач не відмовлявся від подальшого використання бота.

Був створений текстовий опис роботи чат-боту:


1. Бот ініціалізується, підключається до Telegram API і встановлює обробники повідомлень та команд. Також відбувається перевірка та ініціалізація бази даних SQLite.


2. При отриманні команди `"/start"`, бот відправляє привітальне повідомлення з інформацією про доступні функції та показує головне меню з кнопками.


3. При отриманні команди `"? Допомога"`, бот відправляє повідомлення з інструкцією щодо використання всіх функцій.


4. При виборі `"👁️ Курси валют"`, бот показує поточні курси USD та


EUR, отримані через API ПриватБанку.

5. При виборі " Криптовалюти", бот показує меню криптовалют з опціями: курс криптовалют, конвертація, прибуток/збиток, портфель.



6. При виборі " Курс криптовалют", бот запитує дані про курси Bitcoin, Ethereum та Litecoin через API CoinCap і показує їх користувачу.


7. При виборі " Конвертація", бот запитує користувача ввести суму та назву криптовалюти для конвертації у USD.

8. При виборі " Прибуток/Збиток", бот запитує користувача ввести дані про криптовалюту, кількість, ціну купівлі та поточну ціну для розрахунку прибутку чи збитку.

9. При виборі " Портфель", бот показує меню управління портфелем з опціями додавання, видалення та перегляду звіту.

10. При додаванні витрати (введення суми та категорії), бот зберігає інформацію в базу даних та оновлює статистику.

11. При виборі " Сьогодні" або " Місяць", бот показує відповідну статистику витрат.

12. При виборі " Витрати", бот показує останні витрати з можливістю їх видалення.

13. Бот обробляє всі введення користувача, перевіряє їх коректність та надає відповідні повідомлення про помилки у разі потреби.

14. Бот використовує middleware для перевірки доступу користувача за ID перед виконанням будь-яких операцій.

15. Бот працює асинхронно, постійно очікуючи нові повідомлення та команди від користувача, і повторює цикл обробки до завершення роботи.

2.4. Розробка UML-діаграм.

Для моделювання системи використовуються різні діаграми UML, такі як діаграма послідовності, діаграма класів та діаграма станів. UML, або

Уніфікована Мова Моделювання, є стандартом в об'єктно-орієнтованому програмуванні, що дозволяє створювати абстрактні моделі систем з використанням графічних символів. Основна мета UML полягає у визначенні, візуалізації, проектуванні та документуванні програмних систем, допомагаючи розробникам та інженерам чітко розуміти й моделювати архітектуру та поведінку системи. Хоча UML не є мовою програмування, він дозволяє генерувати код на основі створених моделей, тим самим спрощуючи процес розробки.

Одним із найважливіших інструментів UML є діаграма послідовності, яка використовується для моделювання послідовності взаємодії між об'єктами в системі. Діаграма послідовності наочно показує, як об'єкти обмінюються повідомленнями та в якому порядку ці повідомлення надсилаються і отримуються. На ній об'єкти зображуються вертикальними лініями, а повідомлення між ними відображаються стрілками, що показують напрямок і послідовність взаємодій. Час на діаграмі відображається горизонтально, що дозволяє відстежити порядок виконання дій.

Головна мета такої діаграми полягає в тому, щоб продемонструвати взаємодію між об'єктами системи в рамках конкретної послідовності дій. Кожен об'єкт має свою власну вертикальну лінію, і повідомлення передаються від одного об'єкта до іншого за допомогою стрілок, які перетинаються з лініями об'єктів, вказуючи на їхню взаємодію.

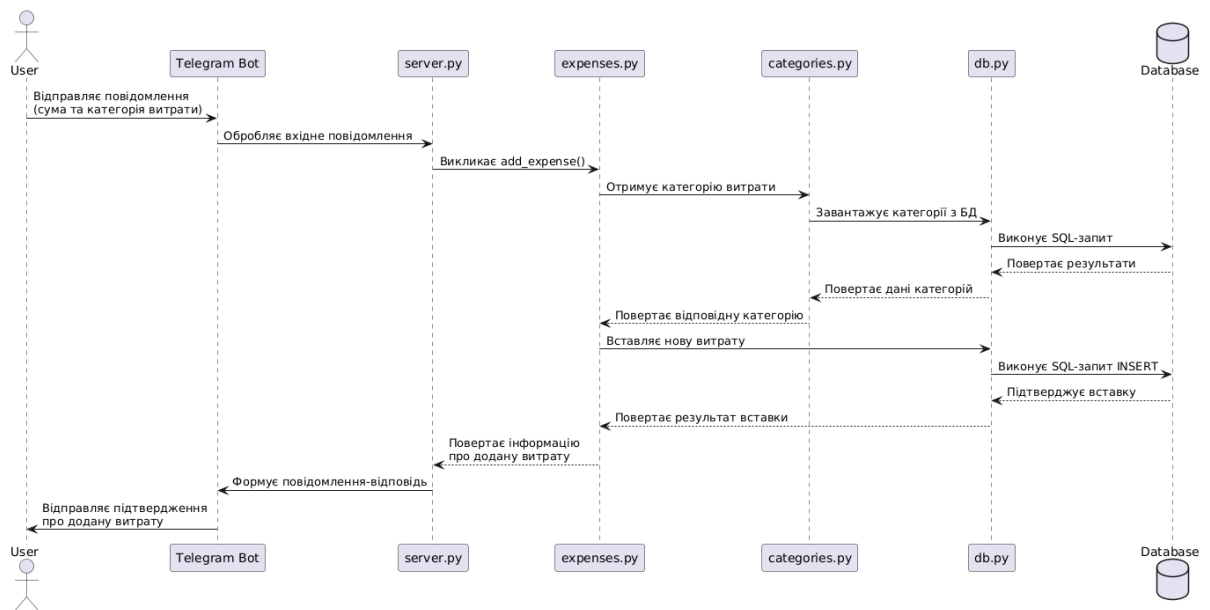


Рис. 2.4. Діаграма послідовності
 Джерело: побудовано автором

Діаграма показує послідовність взаємодій між користувачем, Telegram ботом, різними модулями застосунку (server, expenses, categories, db) та базою даних при додаванні нової витрати.

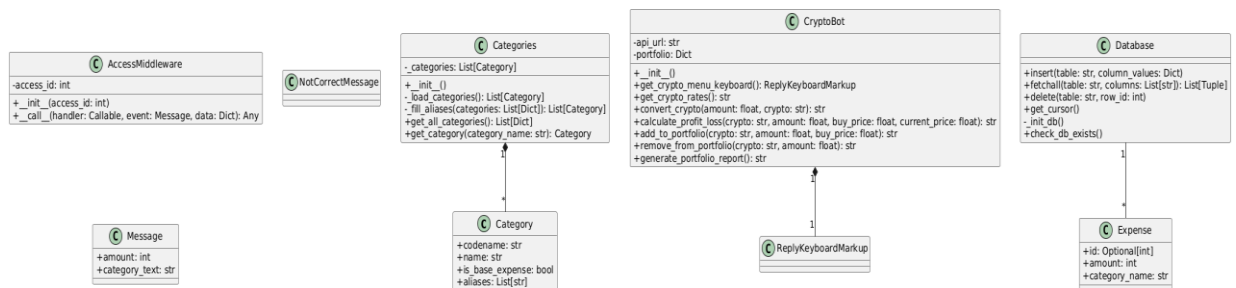


Рис. 2.5. Діаграма класів
 Джерело: побудовано автором

Ця діаграма класів відображає основні класи та їх взаємозв'язки у проєкті:

1. Category і Categories: Представляють категорії витрат та їх управління.
2. CryptoBot: Відповідає за функціональність, пов'язану з криптовалютами.

3. AccessMiddleware: Middleware для контролю доступу.
4. NotCorrectMessage: Користувацьке виключення.
5. Message і Expense: Структури даних для повідомлень та витрат.
6. Database: Представляє взаємодію з базою даних.

Діаграма класів (class diagram) є одним із основних інструментів для моделювання статичної структури системи. Вона використовується як під час проектування нових систем, так і для зворотного проектування існуючих. Оскільки діаграма класів показує класи, їхні атрибути, методи та зв'язки між ними, вона дає детальну картину компонентів системи та їхніх взаємозв'язків. Ці елементи можуть бути автоматично перетворені у вихідний код програми завдяки кодогенерації, яку надають інструменти UML-моделювання, що робить процес проектування більш ефективним. Це особливо корисно для таких мов програмування, як Java або C++, де моделі можна використовувати як основу для коду.

На діаграмі класів зображені класи системи, а також відображені статичні зв'язки між ними, що показують, які класи взаємодіють один з одним або які класи є частинами інших. Оскільки ця діаграма статична, вона не відображає виклики методів між класами, а лише структуру системи на рівні класів, їхніх атрибутів та методів. Таким чином, діаграма класів дозволяє розуміти загальну архітектуру системи і відслідковувати її основні компоненти.

Діаграма станів (state diagram) є інструментом для моделювання динамічної поведінки системи або об'єкта. Вона демонструє всі можливі стани об'єкта та переходи між ними в залежності від подій або умов. Це дає змогу візуально зрозуміти, як система реагує на різні ситуації та як змінюється її поведінка. Діаграма станів є особливо корисною для опису складних об'єктів, чия поведінка може змінюватися в залежності від певних умов, що дозволяє краще відслідковувати логіку переходів між станами.

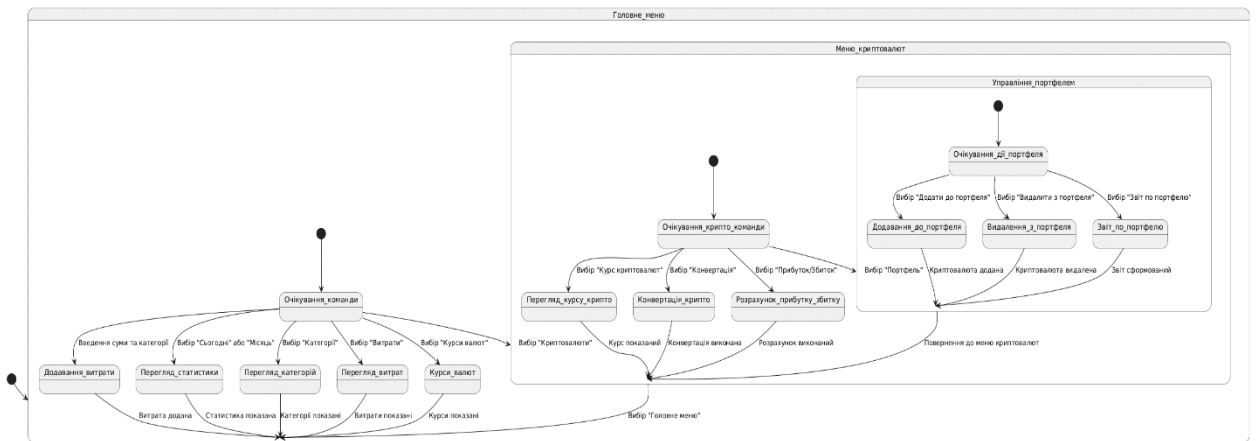


Рис. 2.6. Діаграма станів

Джерело: побудовано автором

Ця діаграма станів відображає:

1. Головне меню як початковий стан, з якого користувач може перейти до різних функцій бота.
2. Меню криптовалют як окремий стан з власними підстанами.
3. Управління портфелем криптовалют як підстан меню криптовалют.
4. Переходи між різними станами на основі дій користувача.

Діаграма показує, як користувач може переміщатися між різними функціями бота, включаючи додавання витрат, перегляд статистики, роботу з криптовалютами тощо.

Діаграма станів допомагає зрозуміти послідовність переходів між станами та логіку поведінки об'єкта. Вона є корисним інструментом для аналізу та проектування систем, особливо тих, де важлива взаємодія об'єктів та контроль їхнього стану.

2.5. Структура програмного комплексу.

Під час розробки програми була створена структура програмного комплексу для інтерактивного сервісу управління особистими фінансовими ресурсами.

1. Структура програмного комплексу:
 - а) Основні компоненти:

- Telegram Bot API Interface
- Core Bot Logic
- Database Management
- Crypto API Interface
- User State Management

b) Допоміжні модулі:

- Expense Tracker
- Category Manager
- Crypto Portfolio Manager
- Statistics Generator

2. Опис взаємозв'язків між компонентами:

a) Telegram Bot API Interface:

- Взаємодіє з Telegram Bot API для отримання повідомлень від користувачів та відправки відповідей.

- Передає отримані повідомлення до Core Bot Logic.

b) Core Bot Logic:

- Обробляє вхідні повідомлення та визначає необхідні дії.

- Взаємодіє з User State Management для відстеження стану користувача.

- Викликає відповідні модулі (Expense Tracker, Category Manager, тощо) залежно від запиту користувача.

- Формує відповіді для користувача та передає їх назад до Telegram Bot API Interface.

c) Database Management:

- Забезпечує взаємодію з базою даних SQLite.

- Виконує операції читання та запису для всіх інших компонентів.

d) Crypto API Interface:

- Взаємодіє з зовнішнім API для отримання актуальних даних про криптовалюту.

- Передає отримані дані до Crypto Portfolio Manager та Statistics Generator.

e) User State Management:

- Зберігає та управляє поточним станом кожного користувача.
- Дозволяє боту правильно інтерпретувати послідовні повідомлення від користувача.

3. Діаграма взаємозв'язків компонентів:

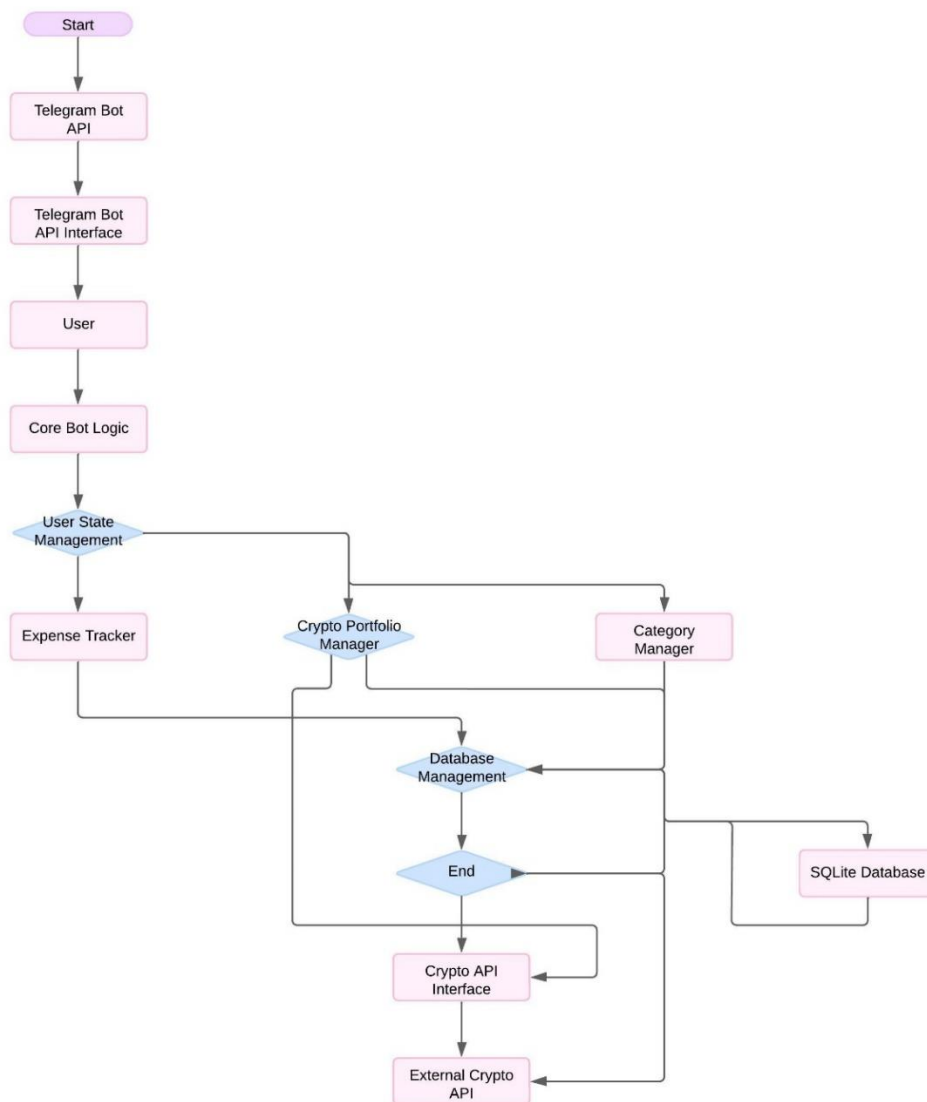


Рис. 2.7. Діаграма взаємозв'язків компонентів

Джерело: побудовано автором

Ця структура забезпечує модульність, масштабованість та ефективність роботи бота. Вибір Python як основної мови програмування дозволяє швидко розробляти та легко підтримувати код, а використання асинхронного програмування забезпечує високу продуктивність при обробці багатьох запитів одночасно. SQLite як база даних ідеально підходить для проєктів середнього розміру, забезпечуючи простоту налаштування та достатню швидкодію.

2.6. Архітектура проєкту.

При реалізації цього проєкту була застосована модульна з елементами багат шарової архітектура. Ось опис архітектури:

1. Основний модуль (server):

- Виступає як точка входу та основний контролер програми
- Містить логіку обробки повідомлень і команд від користувача
- Взаємодіє з Telegram API через бібліотеку aiogram
- Координує роботу інших модулів

2. Модуль бази даних (db):

- Забезпечує абстракцію для роботи з базою даних SQLite
- Містить функції для вставки, вибірки та видалення даних
- Ініціалізує базу даних при першому запуску

3. Модуль категорій (categories):

- Відповідає за логіку роботи з категоріями витрат
- Завантажує категорії з бази даних та керує ними

4. Модуль витрат (expenses):

- Містить логіку для додавання, видалення та аналізу витрат
- Взаємодіє з модулем бази даних для збереження та отримання інформації про витрати

5. Модуль криптовалют (crypto):

- Реалізує функціонал для роботи з криптовалютами

- Взаємодіє з зовнішнім API (CoinCap) для отримання даних про криптовалюту
 - Керує криптопортфелем користувача
6. Модуль middleware (middlewares):
- Забезпечує перевірку доступу користувача перед виконанням операцій
7. Модуль винятків (exceptions):
- Визначає користувацькі винятки для обробки помилок

Архітектурні особливості:

1. Розділення відповідальності: Кожен модуль відповідає за конкретну функціональність, що полегшує розробку та підтримку.
2. Абстракція даних: Модуль бази даних абстрагує роботу з SQLite, дозволяючи легко змінювати або розширювати функціонал без впливу на інші частини програми.
3. Асинхронність: Використання асинхронного програмування (через `aiogram` та `aiohhttp`) для ефективної обробки запитів.
4. Модульність: Кожен компонент може бути розроблений, тестований та масштабований окремо.
5. Слабка зв'язаність: Модулі взаємодіють через чітко визначені інтерфейси, що зменшує залежності між ними.
6. Багатошаровість: Можна виділити шари презентації (взаємодія з користувачем), бізнес-логіки (обробка даних) та доступу до даних (робота з БД та API).

Розробка програми саме з такою архітектурою допомогло легко розширювати функціональність бота, додавати нові можливості та підтримувати код у довгостроковій перспективі.

2.7. Технології розробки.

Технологія розробки цього проекту базується на сучасних практиках розробки Python-додатків, зокрема для створення Telegram-ботів. Було описано основні технологічні аспекти:

1. Мова програмування:
 - Python: Використовується як основна мова розробки, що забезпечує читабельність коду та швидкість розробки.
2. Фреймворк для роботи з Telegram API:
 - aiogram: Асинхронний фреймворк для створення Telegram-ботів, що дозволяє ефективно обробляти велику кількість запитів.
3. Асинхронне програмування:
 - asyncio: Використовується для асинхронного виконання операцій, що покращує продуктивність бота.
4. Робота з базою даних:
 - SQLite: Легка вбудована реляційна база даних для зберігання даних про витрати та категорії.
 - sqlite3: Стандартна бібліотека Python для роботи з SQLite.
5. HTTP-клієнт:
 - aiohttp: Асинхронний HTTP-клієнт для взаємодії з зовнішніми API (наприклад, для отримання курсів криптовалют).
6. Обробка помилок:
 - Власні винятки: Використовуються для обробки специфічних помилок додатку.
7. Типізація:
 - Анотації типів: Використовуються для покращення читабельності коду та запобігання помилкам.
8. Структурування проекту:
 - Модульна архітектура: Код розділений на окремі модулі за функціональністю.

9. Конфігурація:

- Змінні середовища: Використовуються для зберігання конфіденційних даних (наприклад, токена бота).

10.Версійний контроль:

- Git: Ймовірно використовується для контролю версій та спільної розробки.

11.Форматування коду:

- PEP 8: Дотримання стандартів оформлення коду Python.

12.Робота з датами та часом:

- datetime: Стандартна бібліотека Python для роботи з датами і часом.
- pytz: Бібліотека для роботи з часовими поясами.

13.Обробка введення користувача:

- Регулярні вирази (re): Використовуються для парсингу введених користувачем даних.

14.Безпека:

- Middleware для перевірки доступу: Забезпечує базовий рівень безпеки, обмежуючи доступ до бота.

15.Робота з файловою системою:

- os: Стандартна бібліотека Python для роботи з файловою системою.

Ця технологія розробки дозволяє створити ефективний, масштабований та легкий у підтримці Telegram-бот для управління фінансами та роботи з криптовалютами. Використання асинхронного програмування та модульної архітектури забезпечує високу продуктивність та гнучкість системи.

2.8. Висновки до розділу 2

У другому розділі кваліфікаційної роботи було детально проаналізовано та обґрунтовано вибір методів вирішення задачі й засобів для розробки. Проведено аналіз загальних, функціональних та нефункціональних вимог, а також їх формалізацію. У цьому розділі були розглянуті загальні поняття вимог, визначені ключові вимоги для Telegram-бота університету, а також створені UML-діаграми модулів програми з відповідним текстовим описом функціонування чат-бота.

Додатково було розроблено попередній алгоритм роботи Telegram-бота, досліджені основні команди й можливості його функціоналу, а також проаналізована архітектура й структура Telegram-ботів у цілому.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ТА ПРАКТИЧНЕ ВПРОВАДЖЕННЯ

3.1. Реалізація Telegram-боту.

Для того, щоб почати роботу з створення Telegram-боту необхідно його створити та отримати токен – унікальний ідентифікатор для налаштування та підключення API. Створювати ботів Telegram надзвичайно просто, але Вам знадобляться хоча б деякі навички комп'ютерного програмування. На жаль, не існує готових способів створити робочого бота, якщо Ви не розробник. У чат-ботів майже необмежені можливості у месенджері, у них лише є одне обмеження – вони не можуть бачити повідомлення від інших ботів, це зроблено для того, щоб не створювати конфузи з нескінченним спілкуванням ботів між собою.

Сама процедура створення чат-боту на базі Telegram дуже проста і займає всього декілька кліків:

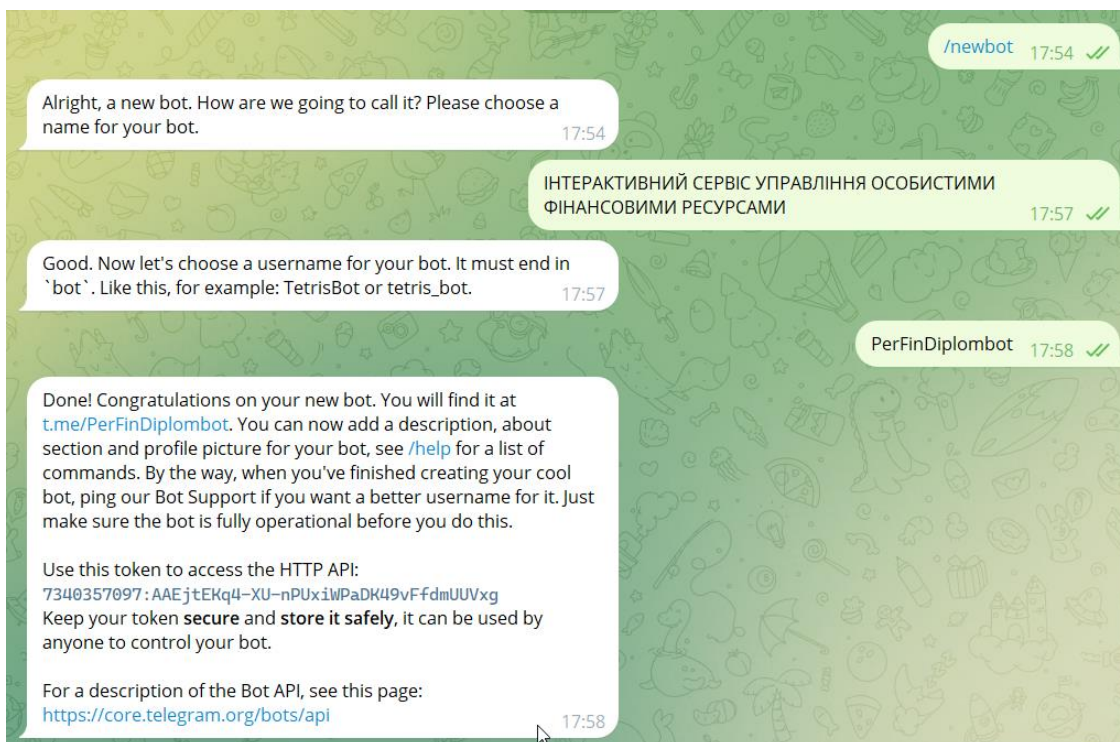


Рис.3.1. Знімок екрану вікна створення ботів

Джерело: знімок з екрану

На знімку екрану видно, що для того, щоб створити Telegram-бота достатньо відкрити діалог із BotFather – бот від Telegram для створення ботів, написати команду /newbot, дати назву боту та унікальне посилання-ідентифікатор з закінченням на «bot». Далі, якщо унікальний ідентифікатор не зайнятий буде надана відповідь про успішне створення чат-бота та надано унікальний токен для API та подальшого його налаштування.

Необхідно додати опис і текст про роботу (команди /setdescription і /setabouttext.

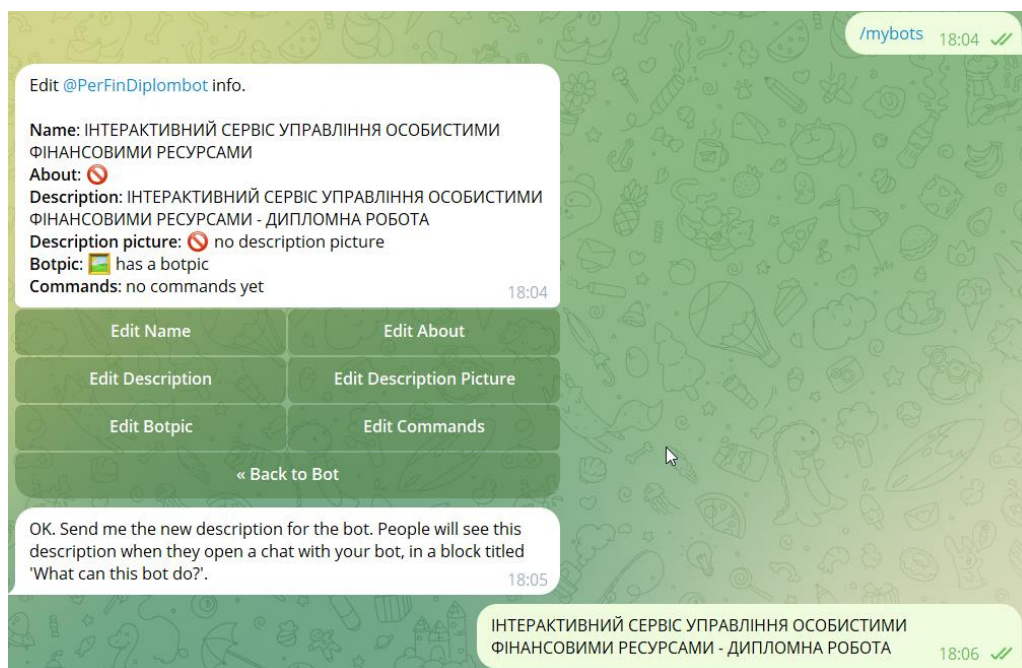


Рис.3.2. Знімок екрану вікна з командою /setdescription

Джерело: знімок з екрану

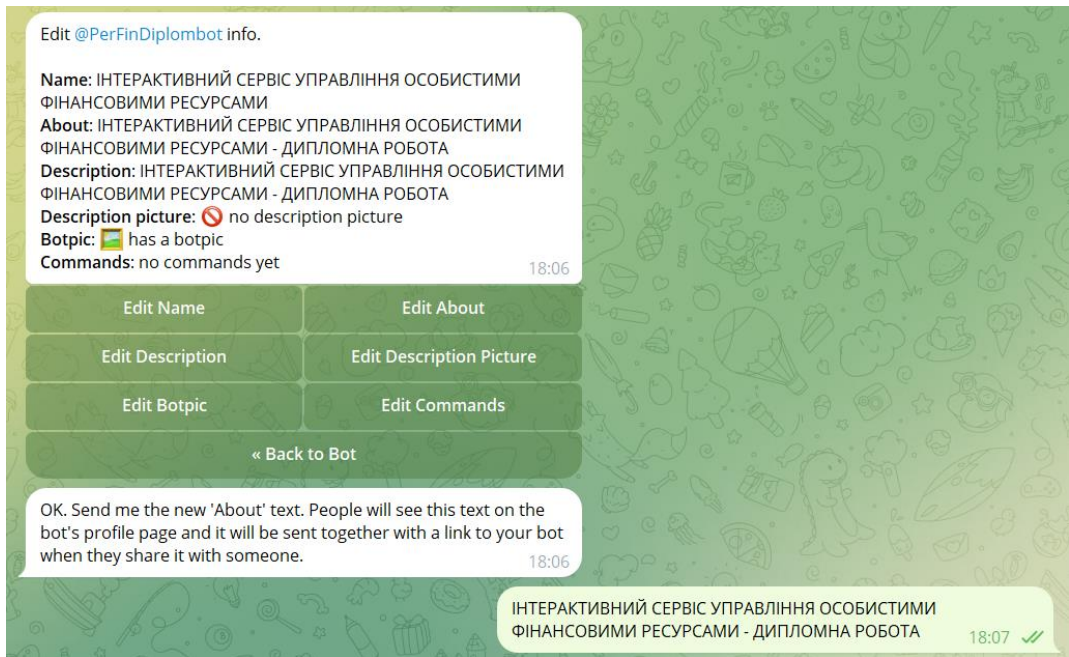


Рис.3.3. Знімок екрану вікна з командою /setabouttext.

Джерело: знімок з екрану

Залишилось додати фото профілю бота (/setuserpic).

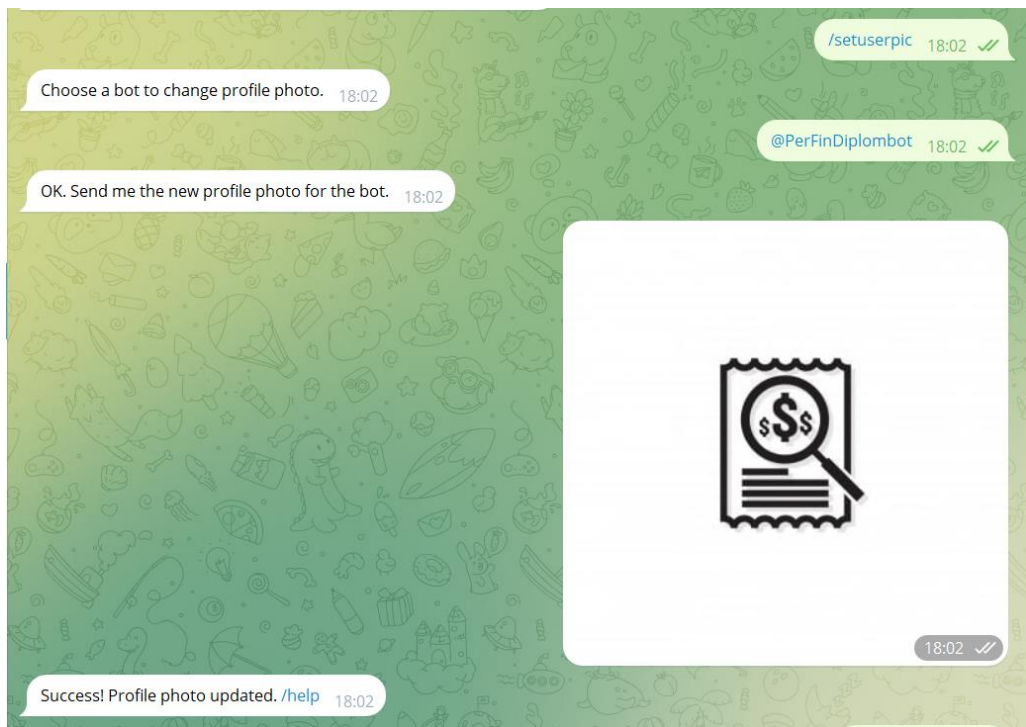


Рис.3.4. Знімок екрану вікна з командою /setuserpic

Джерело: знімок з екрану

Боти зазвичай поєднують в собі різноманітні функціональні можливості, а деякі навіть можуть мати штучний інтелект, який дозволяє їм відповідати на повідомлення, працювати з файлами, змінювати їх або створювати нові. У них є величезний потенціал розвитку, і є всі підстави вважати, що в майбутньому цей напрямок буде набувати ще більшого розмаху.

Для налаштування Telegram-ботів використовується Telegram Bot API - інтерфейс на основі HTTP, який був спеціально створений для розробників, що прагнуть створювати ботів для платформи Telegram. Цей інтерфейс надає широкі можливості для програмування та керування ботами, дозволяючи їм взаємодіяти з користувачами та виконувати різноманітні завдання.

Усі Telegram-боти незалежно від налаштувань отримають:

- усі службові повідомлення;
- всі повідомлення з особистих чатів від користувачів;
- усі типи повідомлень з каналів та чатів, учасником яких вони є;
- за умови вимкненого режиму конфіденційності адміністратори ботів та самі боти можуть отримати абсолютно всі типи повідомлень, окрім повідомлень, що надсилають інші боти.

Було записано основні моменти створення коду для програми, з поясненням коду.

Налаштування середовища та імпорт бібліотек:

```
pythonCopyimport os
import logging
from aiogram import Bot, Dispatcher, types
import aiohttp
import asyncio
```

Ініціалізація бота та диспетчера:

```
API_TOKEN = os.getenv("TELEGRAM_API_TOKEN")
ACCESS_ID = os.getenv("TELEGRAM_ACCESS_ID")
```

```
bot = Bot(token=API_TOKEN)
```

```
dp = Dispatcher()
```

Налаштування логування:

```
pythonCopylogging.basicConfig(level=logging.INFO)
```

Створення основних обробників команд:

```
pythonCopy@dp.message_handler(commands=['start'])
```

```
async def on_start(message: types.Message):
```

```
    await message.reply("Привіт! Я бот для обліку  
фінансів.", reply_markup=get_main_keyboard())
```

Реалізація функцій для роботи з витратами:

```
pythonCopyasync def add_expense(message:  
types.Message):
```

```
    expense = expenses.add_expense(message.text)  
    await message.answer(f"Додано витрату  
{expense.amount} грн на {expense.category_name}.")
```

Реалізація функцій для роботи з криптовалютами:

```
pythonCopyasync def on_crypto_rates(message:  
types.Message):
```

```
    rates = await crypto_bot.get_crypto_rates()  
    await message.answer(rates)
```

Створення клавіатур для інтерфейсу:

```
pythonCopydef get_main_keyboard():
```

```
    keyboard = [  
        [KeyboardButton(text="📊 Сьогодні"),  
KeyboardButton(text="📅 Місяць")],  
        [KeyboardButton(text="📁 Категорії"),  
KeyboardButton(text="💰 Витрати")],
```

```

        [KeyboardButton(text="📄 Курси валют"),
KeyboardButton(text="📄 Криптовалюти")]
    ]
    return ReplyKeyboardMarkup(keyboard=keyboard,
resize_keyboard=True)

```

Реалізація функцій для роботи з базою даних:

```

pythonCopydef insert(table: str, column_values:
Dict):
    columns = ', '.join(column_values.keys())
    values = [tuple(column_values.values())]
    placeholders = ', '.join("{}" *
len(column_values.keys()))
    cursor.executemany(f"INSERT INTO {table}
({columns}) VALUES ({placeholders})", values)
    conn.commit()

```

Створення класу для роботи з категоріями:

```

pythonCopyclass Categories:
    def __init__(self):
        self._categories = self._load_categories()
    def _load_categories(self) -> List[Category]:
        categories = db.fetchall("category",
"codename name is_base_expense aliases".split())
        return self._fill_aliases(categories)

```

Реалізація асинхронних функцій для роботи з API:

```

pythonCopyasync def get_currency_rates():
    url =
"https://api.privatbank.ua/p24api/pubinfo?exchange&cour
sid=5"
    async with aiohttp.ClientSession() as session:

```

```
    async with session.get(url) as response:
        if response.status == 200:
            data = await response.json()
```

Створення middleware для перевірки доступу:

```
pythonCopyclass AccessMiddleware:
    def __init__(self, access_id: int):
        self.access_id = access_id
    async def __call__(self, handler, event, data):
        if event.from_user.id != self.access_id:
            return await event.answer("Access
Denied")
        return await handler(event, data)
```

Обробка помилок:

```
pythonCopytry:
    # Код, який може викликати помилку
except exceptions.NotCorrectMessage as e:
    await message.answer(str(e))
```

Запуск бота:

```
pythonCopyasync def main():
    logging.info("Запуск бота")
    try:
        await dp.start_polling(bot)
    except Exception as e:
        logging.error(f"Виникла помилка: {e}")
if __name__ == '__main__':
    asyncio.run(main())
```

Таким чином, було реалізовано Telegram-бот для управління особистими фінансовими ресурсами. Ці фрагменти коду демонструють ключові аспекти розробки програми, включаючи налаштування бота,

створення обробників команд, роботу з базою даних, використання асинхронного програмування та обробку помилок. Розробка включає в себе створення модульної структури, де кожен компонент (витрати, категорії, криптовалюти) має свій окремий модуль, що полегшує підтримку та розширення функціоналу бота.

3.2. Керівництво користувача.

Щоб розпочати роботу бота, при умови вже встановленого Telegram, необхідно додати його в свій список контактів у Telegram та запустити бот @PerFinDiplombot.

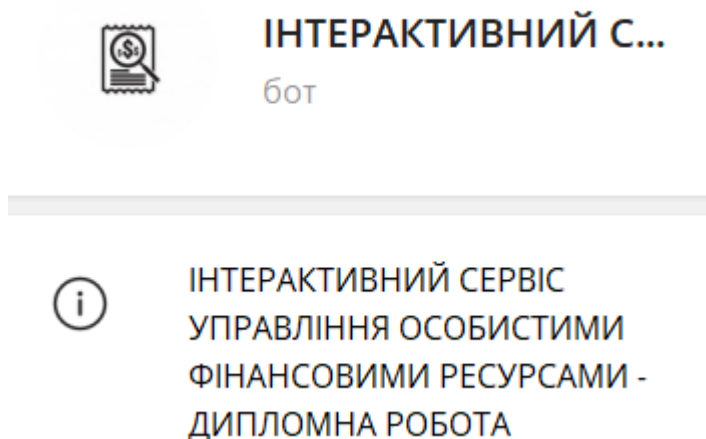


Рис.3.5. Знімок екрану профілю бота

Джерело: знімок з екрану

Для початку роботи запустіть бота. Бот відправить вам привітання та короткий опис його функціоналу(рисунок 3.1).

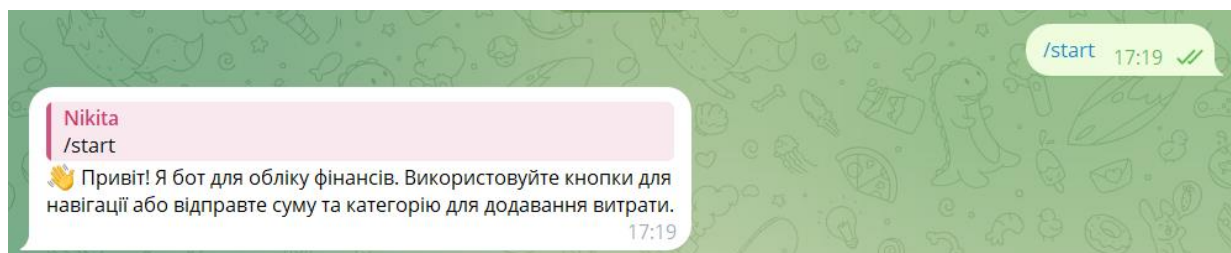


Рис.3.6. Знімок екрану старту бота

Джерело: знімок з екрану

У бота є головне меню, а також підменю для роботи з криптовалютами. У головному меню є кнопки: Допомога, Курси валют, Криптовалюти, Витрати, Сьогодні, Місяць, Категорії(рисунок 3.1).

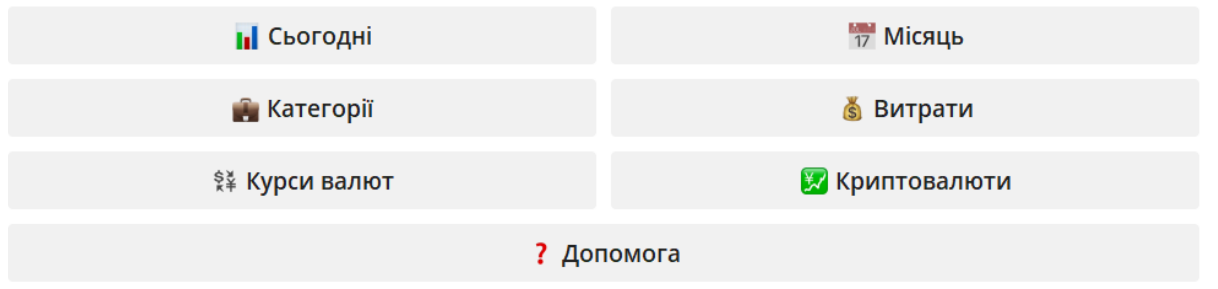


Рис.3.7. Знімок екрану головного меню

Джерело: знімок з екрану

У підменю "Криптовалюти" є такі кнопки: Курс криптовалют, Конвертація, Прибуток/Збиток, Портфель(рисунок 3.1).

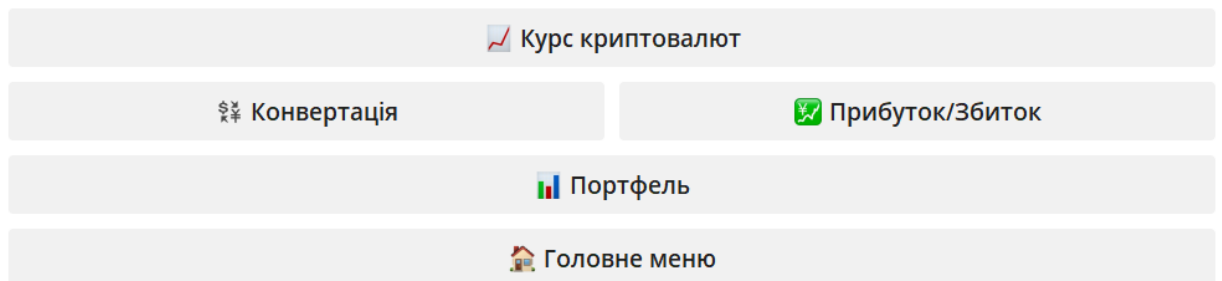


Рис.3.8. Знімок екрану меню криптовалют

Джерело: знімок з екрану

Щоб отримати допомогу, натисніть кнопку Допомога. Бот надішле вам пояснення щодо використання функцій(рисунок 3.1).

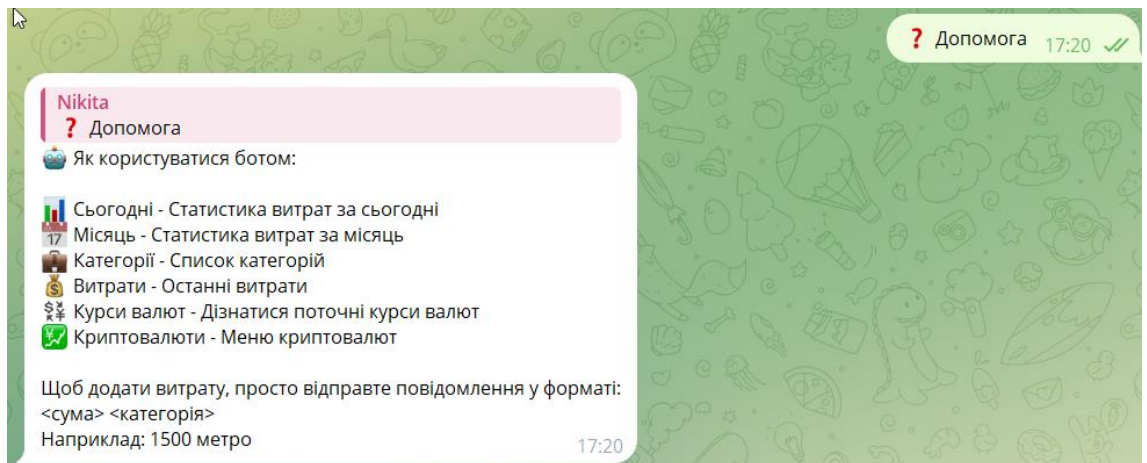


Рис.3.9. Знімок екрану виклику довідки

Джерело: знімок з екрану

Щоб дізнатися курси валют, натисніть команду Курси валют. Бот надішле вам поточні курси USD та EUR.

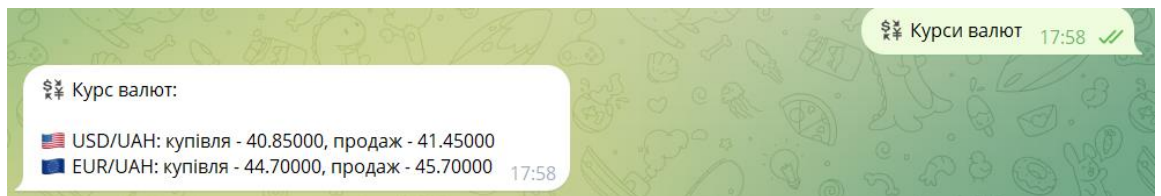


Рис.3.10. Знімок екрану з курсами валют

Джерело: знімок з екрану

Щоб переглянути поточні курси криптовалют, натисніть команду Курс криптовалют. Бот надішле вам інформацію про курси Bitcoin, Ethereum та Litecoin.

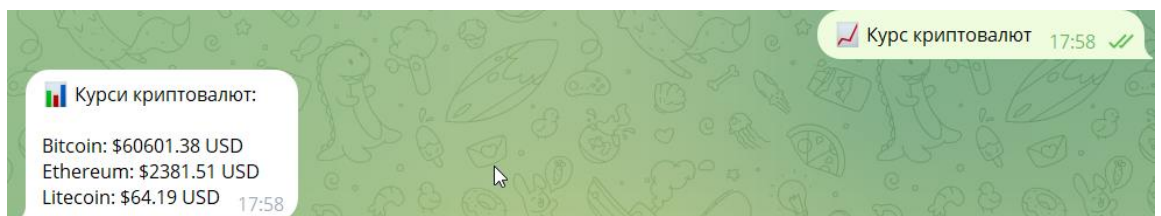


Рис.3.11. Знімок екрану курсів криптовалют

Джерело: знімок з екрану

Для конвертації криптовалют натисніть команду Конвертація. Виберіть криптовалюту та введіть суму для конвертації. Бот надішле результат.



Рис.3.12. Знімок екрану введення суми для конвертації

Джерело: знімок з екрану

Для розрахунку прибутку або збитку від криптовалюти натисніть команду Прибуток/Збиток. Введіть дані про купівлю та поточну ціну криптовалюти.



Рис.3.13. Знімок екрану розрахунку прибутку/збитку

Джерело: знімок з екрану

Для управління криптопортфелем натисніть команду Портфель. Бот надішле меню з можливостями додавання, видалення або перегляду звіту.

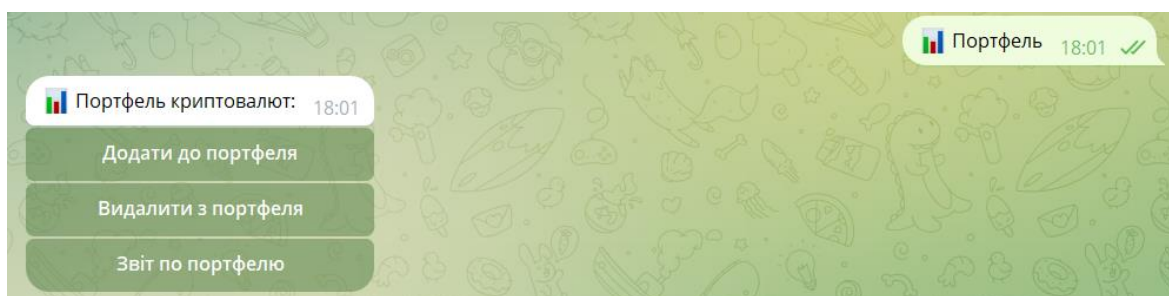


Рис.3.14. Знімок екрану меню управління портфелем

Джерело: знімок з екрану

Щоб додати витрати, введіть суму та категорію витрат. Бот збереже інформацію та оновить статистику.

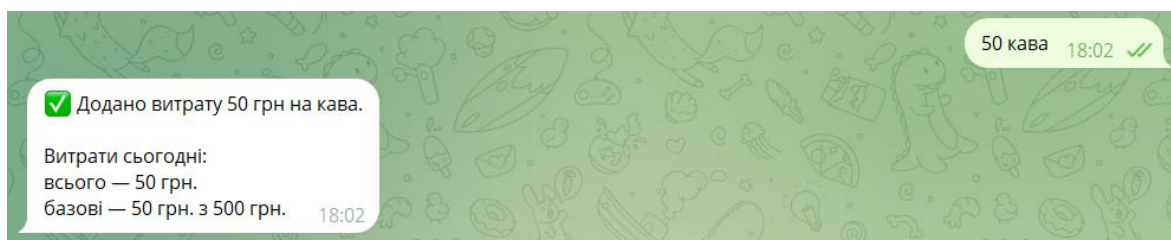


Рис.3.15. Знімок екрану введення витрат

Джерело: знімок з екрану

Для перегляду витрат за сьогодні натисніть команду Сьогодні. Бот покаже статистику за поточний день.



Рис.3.16. Знімок екрану статистики витрат за день

Джерело: знімок з екрану

Для перегляду витрат за місяць натисніть команду Місяць. Бот покаже статистику за місяць.



Рис.3.17. Знімок екрану статистики витрат за місяць

Джерело: знімок з екрану

Щоб переглянути останні витрати або видалити їх, натисніть команду Витрати. Бот покаже список останніх витрат з можливістю видалення.

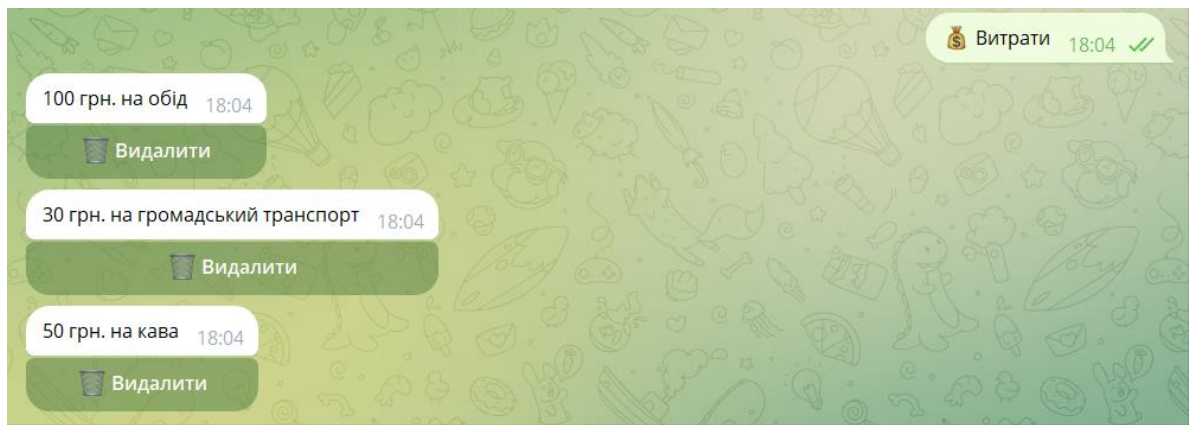


Рис.3.18. Знімок екрану останніх витрат

Джерело: знімок з екрану

Для перегляду можливих категорій витрат натисніть команду Категорії. Бот покаже список категорії витрат.

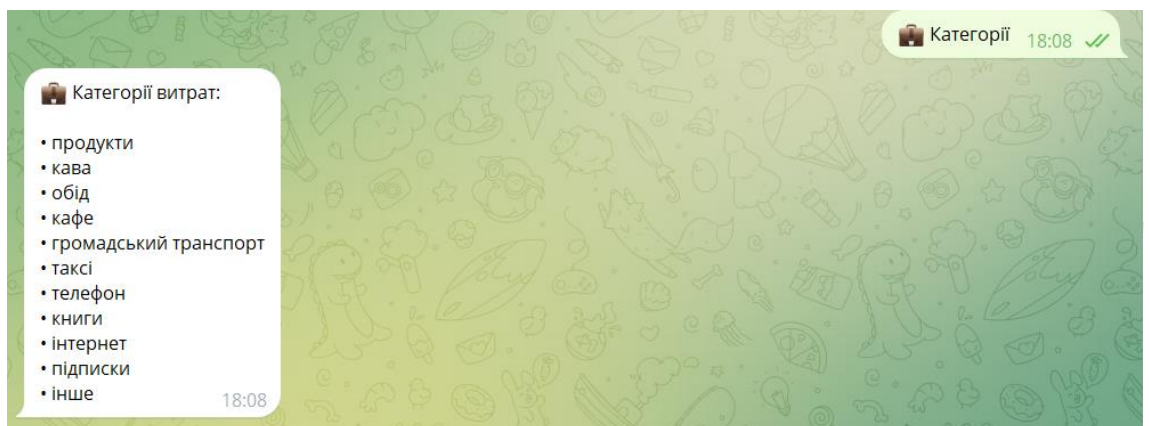


Рис.3.19. Знімок екрану категорій витрат

Джерело: знімок з екрану

3.4. Висновок до розділу 3

У даному розділі кваліфікаційної роботи у процесі створення Telegram-бота для управління особистими фінансами було реалізовано всі необхідні компоненти для його ефективної та зручної роботи. Було розглянуто основні кроки від реєстрації бота до налаштування та інтеграції з Telegram API, що дозволяє взаємодіяти з користувачами через команди та обробники повідомлень.

Розробка включала налаштування середовища, підключення бібліотек, створення обробників команд і клавіатур для користувацького інтерфейсу, а також роботу з базою даних та API для отримання актуальних даних про курси валют і криптовалют. Крім того, важливим аспектом була реалізація обробки помилок і використання асинхронних функцій для підвищення продуктивності.

Telegram-бот успішно поєднує функціональні можливості для обліку витрат і управління криптовалютичним портфелем, дозволяючи користувачам отримувати детальні звіти, конвертувати валюту, а також контролювати свої фінанси в реальному часі. Всі ці функції можуть бути зручно використані через інтуїтивно зрозумілий інтерфейс із меню та підменю.

Таким чином, було створено інструмент, що забезпечує користувачам просте й ефективне управління особистими фінансами через Telegram, з можливістю подальшого розширення функціоналу відповідно до потреб користувачів.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В ході виконання даної кваліфікаційної роботи були вивчені технології для реалізації Telegram-ботів на мові програмування Python. Також, було досліджено технології, що необхідні для створення таких чат-ботів. Крім того, у роботі було проаналізовано проблеми користувачів програм для управління особистими фінансами, а також проведено власне дослідження з приводу якості цієї галузі.

Також, у роботі було досліджено предметну область Telegram-ботів, їх специфіку, алгоритми роботи та можливості. В цілому, основна мета проекту, тобто створення інтерактивного сервісу управління особистими фінансовими ресурсами виконана успішно.

У новоствореному засобі реалізовано максимально зручний та інтуїтивний інтерфейс, проаналізовано та систематизовано інформацію, яка необхідна в першу чергу. Цей інструмент може спокійно працювати на будь-якому пристрої за умови відкриття месенджеру Telegram. Також, варто звернути увагу, що навіть при відсутності Інтернету, старі запити зберігаються в історії переписки та їх можна подивитись навіть не маючи зв'язку з Інтернетом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тестування програмного забезпечення: навч. посіб. / А.С. Авраменко, В.С. Авраменко, Г.В. Косенюк. – Черкаси: ЧНУ імені Богдана Хмельницького, 2017. – 284 с.
2. Інформаційні технології та моделювання бізнес-процесів: навч. посіб. / О. Томашевський, Г. Цегелик, М. Ветер, В. Дубук. – К.: Київ, 2012. – 304 с.
3. Залеський Н.О. Інтерактивний сервіс управління особистими фінансовими ресурсами / Програмування та захист інформації [Електронний ресурс] : зб. наук. ст. студ. / відп. ред. Т. О. Жирова. – Київ : Держ. торг.-екон. ун-т, 2024. – Ч. 1. – с.110-116.
4. Global Finance Magazine. Best Financial Innovations 2023. URL: <https://gfmag.com/award/best-financial-innovations-2023/> (дата звернення: 25.09.2024).
5. Global Finance Magazine. Best Financial Innovations 2024. URL: <https://gfmag.com/banking/best-financial-innovations-2024/> (дата звернення: 25.09.2024).
6. BotFather. URL: <http://t.me/BotFather> (дата звернення: 25.09.2024).
7. Telegram. Bots: An introduction for developers. URL: <https://core.telegram.org/bots> (дата звернення: 18.09.2024).
8. Exchange Rates API. URL: <https://exchangeratesapi.io> (дата звернення: 02.09.2024).
9. Straits Research. List of key players in Personal Finance Software Market. URL: <https://straitsresearch.com/report/personal-finance-software-market> (дата звернення: 25.09.2024).
10. Вікіпедія. Python. URL: <https://uk.wikipedia.org/wiki/Python> (дата звернення: 03.09.2024).
11. Заборонили месенджер Telegram в держорганах України. URL: <https://uablocklist.com/news/zaboronyly-mesendzher-telegram-v-derzhorhanakh-ukrainy> (дата звернення: 23.10.2024).
12. Python Software Foundation. Python. URL: <https://www.python.org/about/> (дата звернення: 03.09.2024).
13. pyTelegramBotAPI. URL: <https://pypi.org/project/pyTelegramBotAPI/0.3.0/> (дата звернення: 30.09.2024).
14. pyTelegramBotAPI. URL: <https://github.com/eternnoir/pyTelegramBotAPI> (дата звернення: 30.09.2024).
15. NerdWallet. Smart Money Podcast: Millennials' Financial Challenges and What They're Doing About Them. URL: <https://www.nerdwallet.com/article/finance/smart-money-podcast-millennials-financial-challenges-and-what-theyre-doing-about-them> (дата звернення: 25.09.2024).

- 16.Spiral Model. URL: <https://www.duo.uio.no/bitstream/handle/10852/75462/Master-Thesis---Sheikh-Muhammad-Ali.pdf?sequence=7&isAllowed=y> (дата звернення: 02.09.2024).
- 17.Telegram. URL: <https://telegram.org/> (дата звернення: 30.09.2024).
- 18.Telegram. Telegram Bot Features. URL: <https://core.telegram.org/bots/features> (дата звернення: 18.09.2024).
- 19.Telegram. TelegramBots. URL: <https://telegram.org/faq?setln=uk#q-how-do-i-create-a-bot> (дата звернення: 30.09.2024).
- 20.Rolling Out. The role of technology in managing personal finance. URL: <https://rollingout.com/2024/03/21/the-role-of-technology-personal-finance/> (дата звернення: 25.09.2024).
- 21.FIS Global. What is fintech? Defining financial technology. URL: <https://www.fisglobal.com/insights/what-is-fintech-defining-financial-technology> (дата звернення: 25.09.2024).
- 22.Forbes. What Is A Chatbot? Everything You Need To Know. URL: <https://www.forbes.com/advisor/business/software/what-is-a-chatbot/> (дата звернення: 19.09.2024).
- 23.IBM. What is a chatbot? URL: <https://www.ibm.com/topics/chatbots> (дата звернення: 19.09.2024).
- 24.TechTarget. What is a Chatbot and Why is it Important? URL: <https://www.techtarget.com/searchcustomerexperience/definition/chatbot> (дата звернення: 19.09.2024).
- 25.Propelrr. 11 Must-have Features for Fintech Apps. URL: <https://propelrr.com/blog/must-have-features-fintech-apps> (дата звернення: 25.09.2024).
- 26.Matellio. Finance Management App Development: Benefits, Features, and Use Cases. URL: <https://www.matellio.com/blog/finance-management-app-development/> (дата звернення: 25.09.2024).

ДОДАТКИ

Додаток А

Реалізація на мові програмування Python.

Лістинг програмного коду

```
server.py
import logging
import os
import aiohttp
import asyncio
from aiogram import Bot, Dispatcher, types
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton, InlineKeyboardMarkup,
InlineKeyboardButton
from crypto import CryptoBot
import exceptions
import expenses
from categories import Categories
from middlewares import AccessMiddleware
# Налаштування логування
logging.basicConfig(level=logging.INFO)
API_TOKEN = os.getenv("TELEGRAM_API_TOKEN")
ACCESS_ID = os.getenv("TELEGRAM_ACCESS_ID")
bot = Bot(token=API_TOKEN)
dp = Dispatcher()
dp.message.middleware(AccessMiddleware(ACCESS_ID))
crypto_bot = CryptoBot()
# Глобальна змінна для відстеження поточного стану користувача
user_state = {}
def get_main_keyboard():
    keyboard = [
        [KeyboardButton(text="📅 Сьогодні"), KeyboardButton(text="📅 Місяць")],
        [KeyboardButton(text="📁 Категорії"), KeyboardButton(text="💰 Витрати")],
        [KeyboardButton(text="🌐 Курси валют"), KeyboardButton(text="📈 Криптовалюти")],
        [KeyboardButton(text="❓ Допомога")]
    ]
    return ReplyKeyboardMarkup(keyboard=keyboard, resize_keyboard=True)
def get_expense_delete_keyboard(expense_id: int) -> InlineKeyboardMarkup:
    button = InlineKeyboardButton(text="🗑️ Видалити", callback_data=f"del_{expense_id}")
    keyboard = InlineKeyboardMarkup(inline_keyboard=[[button]])
    return keyboard
```

```
def get_crypto_menu_keyboard() -> ReplyKeyboardMarkup:
```

```
    keyboard = [  
        [KeyboardButton(text="📊 Курс криптовалют")],  
        [KeyboardButton(text="🔄 Конвертація"), KeyboardButton(text="📈 Прибуток/Збиток")],  
        [KeyboardButton(text="📁 Портфель")],  
        [KeyboardButton(text="🏠 Головне меню")]  
    ]  
    return ReplyKeyboardMarkup(keyboard=keyboard, resize_keyboard=True)
```

```
def get_portfolio_keyboard() -> InlineKeyboardMarkup:
```

```
    keyboard = [  
        [InlineKeyboardButton(text="Додати до портфеля", callback_data="add_to_portfolio")],  
        [InlineKeyboardButton(text="Видалити з портфеля", callback_data="remove_from_portfolio")],  
        [InlineKeyboardButton(text="Звіт по портфелю", callback_data="portfolio_report")]  
    ]  
    return InlineKeyboardMarkup(inline_keyboard=keyboard)
```

```
async def get_currency_rates():
```

```
    url = "https://api.privatbank.ua/p24api/pubinfo?exchange&coursid=5"
```

```
    async with aiohttp.ClientSession() as session:
```

```
        async with session.get(url) as response:
```

```
            if response.status == 200:
```

```
                data = await response.json()
```

```
                rates = {item['ccy']: item for item in data}
```

```
                usd_uah_buy = rates['USD']['buy']
```

```
                usd_uah_sale = rates['USD']['sale']
```

```
                eur_uah_buy = rates['EUR']['buy']
```

```
                eur_uah_sale = rates['EUR']['sale']
```

```
                return (f"📊 Курс валют:\n\n"
```

```
                        f"us USD/UAH: купівля - {usd_uah_buy}, продаж - {usd_uah_sale}\n"
```

```
                        f"eu EUR/UAH: купівля - {eur_uah_buy}, продаж - {eur_uah_sale}")
```

```
            else:
```

```
                return "❌ Не вдалося отримати курси валют"
```

```
async def on_start(message: types.Message):
```

```
    user_id = message.from_user.id
```

```
    user_state[user_id] = 'main_menu'
```

```
    await message.reply(  
        "👋 Привіт! Я бот для обліку фінансів. Використовуйте кнопки для навігації або відправте  
суму та категорію для додавання витрати.",  
        reply_markup=get_main_keyboard())
```

```

async def on_help(message: types.Message):
    help_text = (
        "👤 Як користуватися ботом:\n\n"
        "📊 Сьогодні - Статистика витрат за сьогодні\n"
        "📅 Місяць - Статистика витрат за місяць\n"
        "📁 Категорії - Список категорій\n"
        "💰 Витрати - Останні витрати\n"
        "🌐 Курси валют - Дізнатися поточні курси валют\n"
        "📁 Криптовалюти - Меню криптовалют\n\n"
        "Щоб додати витрату, просто відправте повідомлення у форматі:\n"
        "<сума> <категорія>\n"
        "Наприклад: 1500 метро"
    )
    await message.reply(help_text, reply_markup=get_main_keyboard())
async def on_categories(message: types.Message):
    categories = Categories().get_all_categories()
    answer_message = "📁 Категорії витрат:\n\n" + "\n".join([f"• {c.name}" for c in categories])
    await message.answer(answer_message, reply_markup=get_main_keyboard())
async def on_today(message: types.Message):
    answer_message = expenses.get_today_statistics()
    await message.answer(answer_message, reply_markup=get_main_keyboard())
async def on_month(message: types.Message):
    answer_message = expenses.get_month_statistics()
    await message.answer(answer_message, reply_markup=get_main_keyboard())
async def on_expenses(message: types.Message):
    last_expenses = expenses.last()
    if not last_expenses:
        await message.answer("Витрати ще не введені", reply_markup=get_main_keyboard())
        return
    for expense in last_expenses:
        answer_message = f"{expense.amount} грн. на {expense.category_name}"
        await message.answer(answer_message, reply_markup=get_expense_delete_keyboard(expense.id))
async def on_add_expense(message: types.Message):
    try:
        expense = expenses.add_expense(message.text)
    except exceptions.NotCorrectMessage as e:
        await message.answer(str(e), reply_markup=get_main_keyboard())
        return
    answer_message = (
        f"✅ Додано витрату {expense.amount} грн на {expense.category_name}.\n\n"

```

```

    f"{expenses.get_today_statistics()}")
    await message.answer(answer_message, reply_markup=get_main_keyboard())
async def on_delete_expense(call: types.CallbackQuery):
    expense_id = int(call.data.split("_")[1])
    expenses.delete_expense(expense_id)
    await call.answer("✅ Витрату видалено")
    await call.message.edit_text(f"{call.message.text}\n\n❌ Видалено")
async def on_currency_rates(message: types.Message):
    rates = await get_currency_rates()
    await message.answer(rates, reply_markup=get_main_keyboard())
async def on_crypto_menu(message: types.Message):
    user_id = message.from_user.id
    user_state[user_id] = 'crypto_menu'
    await message.answer("📑 Меню криптовалют:", reply_markup=get_crypto_menu_keyboard())
async def on_view_crypto_rates(message: types.Message):
    rates = await crypto_bot.get_crypto_rates()
    await message.answer(rates, reply_markup=get_crypto_menu_keyboard())
async def on_convert_crypto(message: types.Message):
    user_id = message.from_user.id
    user_state[user_id] = 'crypto_conversion'
    await message.answer(
        "Введіть суму та криптовалюту для конвертації у форматі:\n<сума> <криптовалюта>\nПриклад:
1 bitcoin",
        reply_markup=get_crypto_menu_keyboard())
async def on_profit_loss_calculator(message: types.Message):
    user_id = message.from_user.id
    user_state[user_id] = 'profit_loss'
    await message.answer(
        "Введіть дані для розрахунку прибутку/збитку у форматі:\n<криптовалюта> <кількість> <ціна
купівлі> <поточна ціна>\nПриклад: bitcoin 0.5 30000 35000",
        reply_markup=get_crypto_menu_keyboard())
async def on_portfolio(message: types.Message):
    await message.answer("📊 Портфель криптовалют:", reply_markup=get_portfolio_keyboard())
async def handle_crypto_conversion(message: types.Message):
    user_id = message.from_user.id
    parts = message.text.split()
    if len(parts) == 2:
        try:
            amount = float(parts[0])
            crypto = parts[1]
            response = await crypto_bot.convert_crypto(amount, crypto)

```

```

        await message.answer(response, reply_markup=get_crypto_menu_keyboard())
    except ValueError:
        await message.answer(" ✘ Помилка: Невірний формат суми. Переконайтеся, що сума вказана
коректно.", reply_markup=get_crypto_menu_keyboard())
    else:
        await message.answer(" ✘ Невірний формат. Використовуйте формат: <сума>
<криптовалюта>", reply_markup=get_crypto_menu_keyboard())
        user_state[user_id] = 'crypto_menu' # Скидаємо стан назад до меню криптовалюот
async def handle_profit_loss(message: types.Message):
    user_id = message.from_user.id
    parts = message.text.split()
    if len(parts) == 4:
        try:
            crypto = parts[0]
            amount = float(parts[1])
            buy_price = float(parts[2])
            current_price = float(parts[3])
            result = await crypto_bot.calculate_profit_loss(crypto, amount, buy_price, current_price)
            await message.answer(result, reply_markup=get_crypto_menu_keyboard())
        except ValueError:
            await message.answer(" ✘ Помилка при розрахунку. Переконайтеся, що всі числа вказані
правильно.", reply_markup=get_crypto_menu_keyboard())
    else:
        await message.answer(" ✘ Невірний формат. Використовуйте формат: <криптовалюта>
<кількість> <ціна купівлі> <поточна ціна>", reply_markup=get_crypto_menu_keyboard())
        user_state[user_id] = 'crypto_menu' # Скидаємо стан назад до меню криптовалюот
async def handle_portfolio_callback(call: types.CallbackQuery):
    if call.data == "add_to_portfolio":
        await call.message.answer(
            "Введіть дані для додавання до портфеля у форматі:\n<криптовалюта> <кількість> <ціна
купівлі>\nПриклад: bitcoin 0.5 30000")
    elif call.data == "remove_from_portfolio":
        await call.message.answer(
            "Введіть дані для видалення з портфеля у форматі:\n<криптовалюта> <кількість>\nПриклад:
bitcoin 0.5")
    elif call.data == "portfolio_report":
        report = await crypto_bot.generate_portfolio_report()
        await call.message.answer(report)
    await call.answer()

```

```

async def handle_add_to_portfolio(message: types.Message):
    user_id = message.from_user.id
    parts = message.text.split()
    if len(parts) == 3:
        try:
            crypto = parts[0]
            amount = float(parts[1])
            buy_price = float(parts[2])
            result = crypto_bot.add_to_portfolio(crypto, amount, buy_price)
            await message.answer(result, reply_markup=get_crypto_menu_keyboard())
        except ValueError:
            await message.answer(" ❌ Помилка: Невірний формат. Переконайтеся, що всі числа вказані коректно.", reply_markup=get_crypto_menu_keyboard())
        else:
            await message.answer(" ❌ Невірний формат. Використовуйте формат: <криптовалюта> <кількість> <ціна купівлі>", reply_markup=get_crypto_menu_keyboard())
            user_state[user_id] = 'crypto_menu' # Скидаємо стан назад до меню криптовалюти
    async def handle_remove_from_portfolio(message: types.Message):
        parts = message.text.split()
        if len(parts) == 2:
            try:
                crypto = parts[0]
                amount = float(parts[1])
                result = crypto_bot.remove_from_portfolio(crypto, amount)
                await message.answer(result, reply_markup=get_crypto_menu_keyboard())
            except ValueError:
                await message.answer(" ❌ Помилка: Невірний формат. Переконайтеся, що кількість вказана коректно.",
                    reply_markup=get_crypto_menu_keyboard())
            else:
                await message.answer(" ❌ Невірний формат. Використовуйте формат: <криптовалюта> <кількість>",
                    reply_markup=get_crypto_menu_keyboard())
    async def handle_crypto_message(message: types.Message):
        user_id = message.from_user.id
        parts = message.text.split()
        if len(parts) == 3:
            # Обробка додавання до портфеля
            await handle_add_to_portfolio(message)
        elif len(parts) == 2:
            # Обробка видалення з портфеля

```

```

        await handle_remove_from_portfolio(message)
    else:
        await message.answer(" ❌ Невірний формат. Будь ласка, перевірте інструкції та спробуйте
знову.",

                                reply_markup=get_crypto_menu_keyboard())
        user_state[user_id] = 'crypto_menu' # Скидаємо стан назад до меню криптовалюти
async def handle_message(message: types.Message):
    user_id = message.from_user.id
    text = message.text
    if text == '/start' or text == '🏠 Головне меню':
        user_state[user_id] = 'main_menu'
        await on_start(message)
    elif text == '🔗 Допомога':
        await on_help(message)
    elif text == '📁 Категорії':
        await on_categories(message)
    elif text == '📊 Сьогодні':
        await on_today(message)
    elif text == '📅 Місяць':
        await on_month(message)
    elif text == '💰 Витрати':
        await on_expenses(message)
    elif text == '📈 Курси валют':
        await on_currency_rates(message)
    elif text == '📊 Криптовалюти':
        user_state[user_id] = 'crypto_menu'
        await on_crypto_menu(message)
    elif text == '📊 Портфель':
        await on_portfolio(message)
    elif text == '📊 Курс криптовалюти':
        await on_view_crypto_rates(message)
    elif text == '🔄 Конвертація':
        user_state[user_id] = 'crypto_conversion'
        await on_convert_crypto(message)
    elif text == '📊 Прибуток/Збиток':
        user_state[user_id] = 'profit_loss'
        await on_profit_loss_calculator(message)
    else:
        # Перевіряємо поточний стан користувача
        current_state = user_state.get(user_id, 'main_menu')

```

```

if current_state == 'crypto_menu':
    await handle_crypto_message(message)
elif current_state == 'crypto_conversion':
    await handle_crypto_conversion(message)
elif current_state == 'profit_loss':
    await handle_profit_loss(message)
else:
    # Обробляємо як додавання витрати
    await on_add_expense(message)
# Реєстрація обробників для повідомлень та колбеків
dp.message.register(handle_message)
dp.callback_query.register(on_delete_expense, lambda c: c.data.startswith('del_'))
dp.callback_query.register(handle_portfolio_callback, lambda c: c.data in ["add_to_portfolio",
"remove_from_portfolio", "portfolio_report"])

```

```

async def main():
    logging.info("Запуск бота")
    try:
        await dp.start_polling(bot)
    except Exception as e:
        logging.error(f"Виникла помилка: {e}")
if __name__ == '__main__':
    asyncio.run(main())

```

```

db.py
import os
from typing import Dict, List, Tuple
import sqlite3
conn = sqlite3.connect(os.path.join("db", "finance.db"))
cursor = conn.cursor()
def insert(table: str, column_values: Dict):
    columns = ', '.join( column_values.keys() )
    values = [tuple(column_values.values())]
    placeholders = ", ".join( "?" * len(column_values.keys()) )
    cursor.executemany(
        f"INSERT INTO {table} "
        f"({columns}) "
        f"VALUES ({placeholders})",
        values)
    conn.commit()

```

```

def fetchall(table: str, columns: List[str]) -> List[Tuple]:
    columns_joined = ", ".join(columns)
    cursor.execute(f"SELECT {columns_joined} FROM {table}")
    rows = cursor.fetchall()
    result = []
    for row in rows:
        dict_row = {}
        for index, column in enumerate(columns):
            dict_row[column] = row[index]
        result.append(dict_row)
    return result

def delete(table: str, row_id: int) -> None:
    row_id = int(row_id)
    cursor.execute(f"delete from {table} where id={row_id}")
    conn.commit()

def get_cursor():
    return cursor

def _init_db():
    """Ініціалізує БД"""
    with open("createdb.sql", "r", encoding='utf-8') as f:
        sql = f.read()
    cursor.executescript(sql)
    conn.commit()

def check_db_exists():
    """Перевіряє, чи ініціалізована БД, якщо ні — ініціалізує"""
    cursor.execute("SELECT name FROM sqlite_master "
                   "WHERE type='table' AND name='expense'")
    table_exists = cursor.fetchall()
    if table_exists:
        return
    _init_db()

check_db_exists()

categories.py
"""Робота з категоріями витрат"""
from typing import Dict, List, NamedTuple
import db

class Category(NamedTuple):
    """Структура категорії"""
    codename: str
    name: str

```

```

is_base_expense: bool
aliases: List[str]
class Categories:
    def __init__(self):
        self._categories = self._load_categories()
    def _load_categories(self) -> List[Category]:
        """Повертає довідник категорій витрат з БД"""
        categories = db.fetchall(
            "category", "codename name is_base_expense aliases".split()
        )
        categories = self._fill_aliases(categories)
        return categories
    def _fill_aliases(self, categories: List[Dict]) -> List[Category]:
        """Заповнює по кожній категорії aliases, тобто можливі
        назви цієї категорії, які можемо писати в тексті повідомлення.
        Наприклад, категорія «кафе» може бути написана як cafe,
        ресторан тощо."""
        categories_result = []
        for index, category in enumerate(categories):
            aliases = category["aliases"].split(",")
            aliases = list(filter(None, map(str.strip, aliases)))
            aliases.append(category["codename"])
            aliases.append(category["name"])
            categories_result.append(Category(
                codename=category['codename'],
                name=category['name'],
                is_base_expense=category['is_base_expense'],
                aliases=aliases
            ))
        return categories_result
    def get_all_categories(self) -> List[Dict]:
        """Повертає довідник категорій."""
        return self._categories
    def get_category(self, category_name: str) -> Category:
        """Повертає категорію за одним з її аліасів."""
        findex = None
        other_category = None
        for category in self._categories:
            if category.codename == "other":
                other_category = category
            for alias in category.aliases:
                if category_name in alias:

```

```
        finded = category
    if not finded:
        finded = other_category
    return finded
```

crypto.py

```
import aiohttp
```

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton
```

```
class CryptoBot:
```

```
    def __init__(self):
```

```
        self.api_url = 'https://api.coincap.io/v2/assets'
```

```
        self.portfolio = {} # Словник для зберігання портфеля
```

```
    def get_crypto_menu_keyboard(self):
```

```
        keyboard = [
```

```
            [KeyboardButton(text="📈 Курс криптовалют")],
```

```
            [KeyboardButton(text="🔄 Конвертація"), KeyboardButton(text="📊 Прибуток/Збиток")],
```

```
            [KeyboardButton(text="📁 Портфель"), KeyboardButton(text="🏠 Головне меню")]
```

```
        ]
```

```
        return ReplyKeyboardMarkup(keyboard=keyboard, resize_keyboard=True)
```

```
    async def get_crypto_rates(self):
```

```
        try:
```

```
            async with aiohttp.ClientSession() as session:
```

```
                async with session.get(f"{self.api_url}?ids=bitcoin,ethereum,litecoin") as response:
```

```
                    data = await response.json()
```

```
                    rates = [f"{asset['name']}: ${float(asset['priceUsd']):.2f} USD" for asset in data['data']]
```

```
                    return "📈 Курси криптовалют:\n\n" + "\n".join(rates)
```

```
        except Exception as e:
```

```
            return f"❌ Не вдалося отримати курс криптовалюти: {e}"
```

```
    async def convert_crypto(self, amount, crypto):
```

```
        try:
```

```
            async with aiohttp.ClientSession() as session:
```

```
                async with session.get(self.api_url) as response:
```

```
                    data = await response.json()
```

```
                    asset = next((item for item in data['data'] if item['id'] == crypto.lower()), None)
```

```
                    if asset is None:
```

```
                        return f"❌ Криптовалюту {crypto} не знайдено."
```

```
                    price = float(asset['priceUsd'])
```

```
                    converted = amount * price
```

```
                    return f"🔄 {amount} {crypto} = ${converted:,.2f} USD"
```

```
        except Exception as e:
```

```
            return f"❌ Помилка при конвертації: {e}"
```

```

async def calculate_profit_loss(self, crypto, amount, buy_price, current_price):
    try:
        initial_investment = amount * buy_price
        current_value = amount * current_price
        profit_loss = current_value - initial_investment
        percentage = (profit_loss / initial_investment) * 100
        result = (
            f"🧮 Розрахунок прибутку/збитку для {crypto}:\n\n"
            f"Кількість: {amount}\n"
            f"Ціна купівлі: ${buy_price:,.2f}\n"
            f"Поточна ціна: ${current_price:,.2f}\n\n"
            f"Початкові інвестиції: ${initial_investment:,.2f}\n"
            f"Поточна вартість: ${current_value:,.2f}\n"
            f"{'Прибуток' if profit_loss >= 0 else 'Збиток'}: ${abs(profit_loss):,.2f} ({percentage:,.2f}%)"
        )
        return result
    except Exception as e:
        return f"❌ Помилка при розрахунку: {e}"

def add_to_portfolio(self, crypto, amount, buy_price):
    if crypto in self.portfolio:
        self.portfolio[crypto]['amount'] += amount
        self.portfolio[crypto]['buy_price'] = buy_price # Оновлюємо ціну купівлі
    else:
        self.portfolio[crypto] = {'amount': amount, 'buy_price': buy_price}
    return f"✅ {amount} {crypto} додано до портфеля за ціною ${buy_price:,.2f}."

def remove_from_portfolio(self, crypto, amount):
    if crypto in self.portfolio:
        if self.portfolio[crypto]['amount'] >= amount:
            self.portfolio[crypto]['amount'] -= amount
            if self.portfolio[crypto]['amount'] == 0:
                del self.portfolio[crypto]
            return f"✅ {amount} {crypto} видалено з портфеля."
        else:
            return "❌ Недостатньо криптовалюти для видалення."
    else:
        return "❌ Криптовалюту не знайдено в портфелі."

async def generate_portfolio_report(self):
    try:
        report = "📊 Звіт по портфелю:\n\n"
        if not self.portfolio:

```

```

return " ✘ Портфель порожній."

total_value = 0
total_investment = 0

async with aiohttp.ClientSession() as session:
    for crypto, data in self.portfolio.items():
        async with session.get(f"{self.api_url}/{crypto}") as response:
            if response.status == 200:
                coin_data = await response.json()
                current_price = float(coin_data['data']['priceUsd'])
                amount = data['amount']
                buy_price = data['buy_price']

                current_value = amount * current_price
                initial_investment = amount * buy_price
                profit_loss = current_value - initial_investment
                percentage = (profit_loss / initial_investment) * 100

                total_value += current_value
                total_investment += initial_investment

                report += (f"{crypto.title()}\n"
                    f"Кількість: {amount}\n"
                    f"Ціна купівлі: ${buy_price:,.2f}\n"
                    f"Поточна ціна: ${current_price:,.2f}\n"
                    f"Поточна вартість: ${current_value:,.2f}\n"
                    f"{'Прибуток' if profit_loss >= 0 else 'Збиток'}: ${abs(profit_loss):,.2f}
                    ({percentage:,.2f}%) \n\n")
            else:
                report += f" ✘ Не вдалося отримати дані для {crypto} \n\n"

total_profit_loss = total_value - total_investment
total_percentage = (total_profit_loss / total_investment) * 100
report += (f"Підсумок:\n"
    f"Загальна вартість портфеля: ${total_value:,.2f}\n"
    f"Загальні інвестиції: ${total_investment:,.2f}\n"
    f"{'Загальний прибуток' if total_profit_loss >= 0 else 'Загальний збиток'}:
    ${abs(total_profit_loss):,.2f} ({total_percentage:,.2f}%)")

return report

```

```

    except Exception as e:
        return f"✘ Помилка при створенні звіту: {e}"
middlewares.py
from typing import Callable, Dict, Any, Awaitable
from aiogram.types import Message
class AccessMiddleware:
    def __init__(self, access_id: int):
        self.access_id = int(access_id) # Перетворюємо у ціле число
    async def __call__(
        self,
        handler: Callable[[Message, Dict[str, Any]], Awaitable[Any]],
        event: Message,
        data: Dict[str, Any]
    ) -> Any:
        if event.from_user and int(event.from_user.id) != self.access_id:
            await event.reply("Access Denied") # Використовуємо reply замість answer для повідомлень
            return
        return await handler(event, data)
exceptions.py
class NotCorrectMessage(Exception):
    pass
expenses.py
""" Робота з витратами — їх додавання, видалення, статистика """
import datetime
import re
from typing import List, NamedTuple, Optional
import pytz
import db
import exceptions
from categories import Categories
class Message(NamedTuple):
    """Структура розпаршеного повідомлення про нову витрату"""
    amount: int
    category_text: str
class Expense(NamedTuple):
    """Структура доданої в БД нової витрати"""
    id: Optional[int]
    amount: int
    category_name: str
def add_expense(raw_message: str) -> Expense:
    """Додає нове повідомлення.
    Приймає на вхід текст повідомлення, що надійшло в бот."""

```

```

parsed_message = _parse_message(raw_message)
category = Categories().get_category(
    parsed_message.category_text)
inserted_row_id = db.insert("expense", {
    "amount": parsed_message.amount,
    "created": _get_now_formatted(),
    "category_codename": category.codename,
    "raw_text": raw_message
})
return Expense(id=None,
               amount=parsed_message.amount,
               category_name=category.name)
def get_today_statistics() -> str:
    """Повертає рядком статистику витрат за сьогодні"""
    cursor = db.get_cursor()
    cursor.execute("select sum(amount)
                   "from expense where date(created)=date('now', 'localtime')")
    result = cursor.fetchone()
    if not result[0]:
        return "Сьогодні ще немає витрат"
    all_today_expenses = result[0]
    cursor.execute("select sum(amount) "
                  "from expense where date(created)=date('now', 'localtime') "
                  "and category_codename in (select codename "
                  "from category where is_base_expense=true)")
    result = cursor.fetchone()
    base_today_expenses = result[0] if result[0] else 0
    return (f"Витрати сьогодні:\n"
           f"всього — {all_today_expenses} грн.\n"
           f"базові — {base_today_expenses} грн. з {_get_budget_limit()} грн.\n\n")
def get_month_statistics() -> str:
    """Повертає рядком статистику витрат за поточний місяць"""
    now = _get_now_datetime()
    first_day_of_month = f'{now.year:04d}-{now.month:02d}-01'
    cursor = db.get_cursor()
    cursor.execute(f"select sum(amount) "
                  f"from expense where date(created) >= '{first_day_of_month}'")
    result = cursor.fetchone()
    if not result[0]:
        return "У цьому місяці ще немає витрат"
    all_today_expenses = result[0]
    cursor.execute(f"select sum(amount) ")

```

```

        f"from expense where date(created) >= '{first_day_of_month}' "
        f"and category_codename in (select codename "
        f"from category where is_base_expense=true)")
result = cursor.fetchone()
base_today_expenses = result[0] if result[0] else 0
return (f"Витрати в поточному місяці:\n"
        f"всього — {all_today_expenses} грн.\n"
        f"базові — {base_today_expenses} грн. з "
        f"{now.day * _get_budget_limit()} грн.")
def last() -> List[Expense]:
    """Повертає останні кілька витрат"""
    cursor = db.get_cursor()
    cursor.execute(
        "select e.id, e.amount, c.name "
        "from expense e left join category c "
        "on c.codename=e.category_codename "
        "order by created desc limit 10")
    rows = cursor.fetchall()
    last_expenses = [Expense(id=row[0], amount=row[1], category_name=row[2]) for row in rows]
    return last_expenses
def delete_expense(row_id: int) -> None:
    """Видаляє повідомлення за його ідентифікатором"""
    db.delete("expense", row_id)
def _parse_message(raw_message: str) -> Message:
    """Парсить текст отриманого повідомлення про нову витрату."""
    regex_result = re.match(r"([d ]+) (.*)", raw_message)
    if not regex_result or not regex_result.group(0) \
        or not regex_result.group(1) or not regex_result.group(2):
        raise exceptions.NotCorrectMessage(
            "Не можу зрозуміти повідомлення. Напишіть повідомлення у форматі, "
            "наприклад:\n1500 метро")
    amount = regex_result.group(1).replace(" ", "")
    category_text = regex_result.group(2).strip().lower()
    return Message(amount=amount, category_text=category_text)
def _get_now_formatted() -> str:
    """Повертає сьогоднішню дату рядком"""
    return _get_now_datetime().strftime("%Y-%m-%d %H:%M:%S")
def _get_now_datetime() -> datetime.datetime:
    """Повертає сьогоднішній datetime з урахуванням часового поясу Києву."""
    tz = pytz.timezone("Europe/Kiev")
    now = datetime.datetime.now(tz)
    return now

```

```

def _get_budget_limit() -> int:
    """Повертає денний ліміт витрат для основних базових витрат"""
    return db.fetchall("budget", ["daily_limit"])[0]["daily_limit"]

createddb.sql

create table budget(
    codename varchar(255) primary key,
    daily_limit integer
);

create table category(
    codename varchar(255) primary key,
    name varchar(255),
    is_base_expense boolean,
    aliases text
);

create table expense(
    id integer primary key,
    amount integer,
    created datetime,
    category_codename integer,
    raw_text text,
    FOREIGN KEY(category_codename) REFERENCES category(codename)
);

insert into category (codename, name, is_base_expense, aliases)
values
    ("products", "продукти", true, "їжа"),
    ("coffee", "кава", true, ""),
    ("dinner", "обід", true, "їдальня, ланч, бізнес-ланч, бізнес ланч"),
    ("cafe", "кафе", true, "ресторан, рест, мак, макдональдс, макдак, kfc, ilpatio, il patio"),
    ("transport", "громадський транспорт", false, "метро, автобус, metro"),
    ("taxi", "таксі", false, "таксі, taxi"),
    ("phone", "телефон", false, "водофон, зв'язок"),
    ("books", "книги", false, "література, літра, літ-ра"),
    ("internet", "інтернет", false, "інет, inet"),
    ("subscriptions", "підписки", false, "підписка"),
    ("other", "інше", true, "");

insert into budget(codename, daily_limit) values ('base', 500);

Dockerfile
FROM python:3.8
WORKDIR /home
ENV TELEGRAM_API_TOKEN="7340357097:AAEjtEKq4-XU-nPUxiWPaDK49vFfdmUUVxg"
ENV TELEGRAM_ACCESS_ID="317340176"

```

```
ENV TZ=Europe/Kiev
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
RUN pip install -U pip aiogram pytz && apt-get update && apt-get install sqlite3
COPY *.py ./
COPY createdb.sql ./
ENTRYPOINT ["python", "server.py"]
```