МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія планування та контролю бюджету»

Завідувач
випускаючої кафедри **А. С. Довбиш**

Керівник роботи **О. А. Шовкопляс**

Студентка групи ІК.м – 01 **А. О. Чернякова**

СУМИ 2021

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:
зав.кафедри _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТОВІ

Черняковій Анні Олександрівні
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) *Інформаційна технологія планування та контролю бюджету*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проєкту (роботи)_____

3. Вхідні данні до проєкту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Огляд технологій розробки та існуючих рішень, що застосовуються для контролю фінансів. 2) Формування мети та постановки задачі, визначення етапів реалізації проєкту. 3) Вибір технологій та інструментів, що необхідні для розробки, огляд та аналіз баз даних. 4) Проєктування бази даних. 5) Моделювання системи. 6) Розробка телеграм-бота. 7) Тестування готового програмного продукту та аналіз результатів.

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

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Термін виконання проєкту (роботи)	Примітка
1.	Огляд технологій розробки та існуючих рішень		
2.	Формування мети та постановка задачі		
3.	Вибір технологій та інструментів для розробки		
4.	Проєктування бази даних та моделювання системи		
5.	Розробка телеграм-бота		
6.	Тестування готового програмного продукту		
7.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____

(підпис)

Керівник проєкту _____

(підпис)

РЕФЕРАТ

Записка: 59 стор., 24 рис., 2 таблиці, 1 додаток, 18 літературних джерел.

Об’єкт дослідження — методи проєктування та технології створення ботів.

Мета роботи — розробка та програмна реалізація інформаційної технології планування та контролю бюджету у формі чат-боту.

Результати — проведений аналіз технологій розробки і огляд відомих рішень, на основі чого було сформовано постановку задачі. Було обрано мову програмування, платформу та середовище для розробки. Також, проведено огляд та аналіз баз даних. Після чого – спроєктовано базу даних та змодельовано систему телеграм-бот. У результаті, розроблено інформаційну технологію планування та контролю бюджету у формі чат-бота за допомогою мови програмування Java Script та середовища Node.js. Програмний продукт було успішно протестовано на мобільному пристрої та на комп’ютері.

СИСТЕМА КОНТРОЛЮ ФІНАНСІВ ТЕЛЕГРАМ-БОТ BUDGET PLANNING JAVA SCRIPT NODE.JS MYSQL TELEGRAM

ЗМІСТ

ВСТУП

Питання контролю фінансів – це те, що завжди актуально, як у глобальному масштабі, наприклад, у бізнесі, так і для особистих цілей. Прибуток є основною базою витрат та показником того, наскільки розумно та вміло людина розпоряджається грошовими надходженнями. Дослідження показують, що незначний відсоток людей слідкує за своїм бюджетом та грамотно його розподіляє, багато хто обирає спосіб тримати все в голові. Також, з'ясувалося, що жінки контролюють витрати рідше ніж чоловіки. Таким чином, більшість людей не знають, на що вони витрачають гроші, що ускладнює процес накопичення конкретної суми, заважає плануванню бюджету для найближчих і майбутніх цілей, збільшує ризик того, що одного дня рівень витрат перевищить рівень доходів [1].

Люди прагнуть економити час настільки, наскільки це можливо і, навіть, більше. Звісно, зберігати та аналізувати свої витрати без допоміжної оптимізованої системи складно. Повна або часткова втрата записаних даних, довгий пошук інформації, неточний аналіз прибутку та витрат – усе це проблеми, з якими стикається людина, яка веде облік фінансів, і як наслідок, втрачається бажання та мотивація продовжувати цей контроль.

Завдяки постійному розвиту технологій життя людини стає більш комфортним, відбувається оптимізація багатьох процесів. Вже давно мобільний телефон використовується не тільки для зв'язку, адже цей гаджет – значно більше, корисний інструмент, який допомагає вирішувати безліч задач. Саме тому, використання смартфону для відслідковування фінансів може зробити цей процес досить зручним та значно зекономить час. Варто зазначити, що кількість потреб людини постійно зростає , саме тому кожного дня створюються нові програми, які пропонують оптимальні рішення. Серед них є і ті, що призначені для контролю фінансів. Але велика кількість таких програм не задовольняє потреби користувача, пропонуючи незначний набір дійсно корисних функцій. Крім того, форма розроблюваної системи грає

важливу роль, визначає доступність та зручність використання. Саме тому, дана дипломна робота присвячена розробці такої інформаційної системи, яка зробить процес контролю фінансів комфортним та швидким. При цьому надані функції будуть корисними, а програмний продукт дружнім до користувача та зрозумілим на інтуїтивному рівні.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД ТА АНАЛІЗ

1.1 Аналіз існуючих рішень

Існує декілька рішень для реалізації системи контролю фінансів. Найбільш розповсюдженими та популярними серед них є мобільні додатки та чат-боти. Мобільні додатки вже давно активно використовуються юзерами, а саме після відкриття онлайн-магазину App Store від Apple у 2008 році. З того моменту додатки багато разів оновлювалися та змінювалися, стали підтримувати роботу на різних пристроях, наприклад, стали доступними на комп'ютерах, телевізорах, годинниках тощо. В той час, як розповсюдження чат-ботів почало активно набирати оберту десь у 2016 році. Вони стали важливими помічниками людей та компаній, виконуючи свою роботу ефективно та швидко. На додаток, завдяки застосуванню машинного навчання, боти можуть постійно вчитися та розвиватися, стаючи ще більш багатофункціональними та розумними асистентами.

Від вибору рішення залежить кінцевий продукт, саме тому необхідно приділити особливу увагу цьому етапу. Отже, проведемо огляд та проаналізуємо вже існуючі рішення, визначимо переваги та недоліки кожного з них. На основі отриманої інформації можна буде зробити висновки та обрати найбільш підходящий варіант для розробки програмного продукту.

Отже, розглянемо такі рішення:

1. Мобільний додаток.

2. Чат-бот.

Мобільний додаток – це програма, яка призначена для роботи на конкретному обладнанні, а саме на мобільних переносних обчислювальних пристроях, таких як планшети та смартфони. Мобільні програми надають клієнтам можливості адміністрування, які можна порівняти з тими, що доступні на ПК. Програми, як правило, є невеликими окремими програмними блоками з обмеженими функціями. Вони запускаються при натисканні іконки

на мобільному пристрої. Доступно безліч таких додатків, і вони поділяються на три основні категорії: нативні, веб-додатки та гібридні додатки.

Нативний мобільний додаток працює тільки на тому обладнанні, для якого він був розроблений, тобто користування обмежено платформою, наприклад, якщо це Android, то встановлення та подальша робота на iOS є неможливим.

Веб-додаток – мобільний додаток, який використовує можливість підключення до Інтернету для забезпечення деяких або всіх своїх функцій. Це не окрема програма, робота відбувається на веб-сервері.

Гібрид – такий тип додатків поєднує у собі технології нативних програм та веб-додатків. Вони є кросплатформенними, тобто підтримують різні операційні системи.

Щодо розробки та розповсюдження, мобільні пристрої накладають значні обмеження на розробників з боку пам'яті, сховища та обчислювальної потужності. Що стосується середовищ розробки, вони нагадують ті, які є доступними для розробки настільних додатків. Наприклад, Visual Studio – одне з основних середовищ розробки Windows, на додаток до традиційних робочих столів, воно надає можливість розробляти програмне забезпечення та програми ще й для смартфонів і планшетів. Поширення мобільних додатків відбувається через онлайн-магазин, такі як Google (Google Play) та Apple (App Store) [2].

Переваги:

— висока продуктивність;

— надійний захист даних;

— зручність користувацького інтерфейсу;

— доступ до повного набору функцій пристрою;

— можливість працювати без інтернету.

Недоліки:

— індивідуальний підхід для кожної операційної системи;

— обмеження аудиторії користувачів;

— складність у підтримці та при оновленні;

— час та складність розробки [3].

Чат-бот – це комп'ютерна програма, яка імітує та обробляє людську розмову (письмово чи усно), дозволяючи людям взаємодіяти з цифровими пристроями, начебто вони спілкуються з реальною людиною. Чат-боти можуть бути такими ж простими, як елементарні програми, які відповідають на простий запит однорядковою відповіддю, або настільки ж складними, як цифрові помічники, які навчаються та розвиваються для забезпечення більш високого рівня персоналізації для збору та обробки інформації [4, 5].

Є два основні типи чат-ботів:

Орієнтовані на завдання (декларативні) боти – це одноцільові програми, що призначені для виконання тільки однієї функції. Використовуючи певні правила, NLP (Natural Language Processing) та дуже небагато машинного навчання, вони генерують автоматизовані, але діалогові відповіді на запити користувачів. Взаємодія з цими чат-ботами досить специфічна, структурована і найбільш застосовна до функцій підтримки та обслуговування. Також, чат-боти, що націлені на завдання, можуть опрацьовувати загальні питання, такі як запити про розклад годин роботи, надання інформації про сервіси, запис користувача на прийом та консультацію або проведення простих транзакції, які не пов'язані з різними змінними. Хоч вони й використовують обробку природної мови для того, щоб кінцеві користувачі могли використовувати їх у розмові, можливості, які вони надають, досить прості. В даний час це найбільш популярні та найчастіше використовувані чат-боти.

Керовані даними та прогнозуючі (розмовні) системи часто називають віртуальними помічниками або цифровими помічниками, і вони набагато складніші, більш персоналізовані та інтерактивні, ніж ті чат-боти, що орієнтовані на виконання конкретних завдань. Ці програми обізнані про контекст і використовують обробку природної мови (NLP), розуміння природної мови (NLU) та машинне навчання (ML) для навчання на ходу.

Вони застосовують прогнозний інтелект та аналітику для забезпечення персоналізації на основі профілів користувачів та поведінки користувачів у минулому. Цифрові помічники можуть з часом дізнаватися про переваги користувача, надавати рекомендації і, також, передбачувати потреби. Окрім моніторингу даних та намірів, вони можуть ініціювати бесіди. Siri від Apple та Alexa від Amazon – це приклади орієнтованих на споживача, керованих даними та прогнозуючих чат-ботів [6].

Керовані штучним інтелектом, автоматизованими правилами, обробкою природної мови (NLP) та машинним навчанням (ML), чат-боти виконують два різні завдання:

1. Аналіз запитів користувачів.
2. Повернення відповіді.

Отже, аналіз запитів користувачів – це перше завдання, яке виконує чат-бот. Його метою є проаналізувати запит користувача, щоб визначити намір юзера та отримати відповідні об'єкти.

Повернення відповіді. Після того, як намір користувача був ідентифікований, чат-бот підбирає найбільш підходящу відповідь на запит юзера. Відповідь може бути такою:

— загальний та зумовлений текст;
— текст, витягнутий із бази знань, що містить різні відповіді;
— контекстуалізована частина інформації, що базується на даних, наданих користувачем;
— дані, що зберігаються у корпоративних системах;
— результат дії, яку чат-бот виконав, при взаємодії з одним або декількома серверними програмами;
— додаткове уточнювальне питання, що допомагає боту правильно зрозуміти запит користувача.

Отже, розглянемо переваги та недоліки чат-ботів.

Плюси:

— не потребують завантаження на пристрій;

— не прив’язані до конкретної операційної системи та платформи;

— не займають місце на пристрої;

— дружні до користувача;

— швидкість та легкість розробки;

— не потребують постійного оновлення.

Мінуси:

— складність у реалізації певного функціоналу;

— одноманітний інтерфейс;

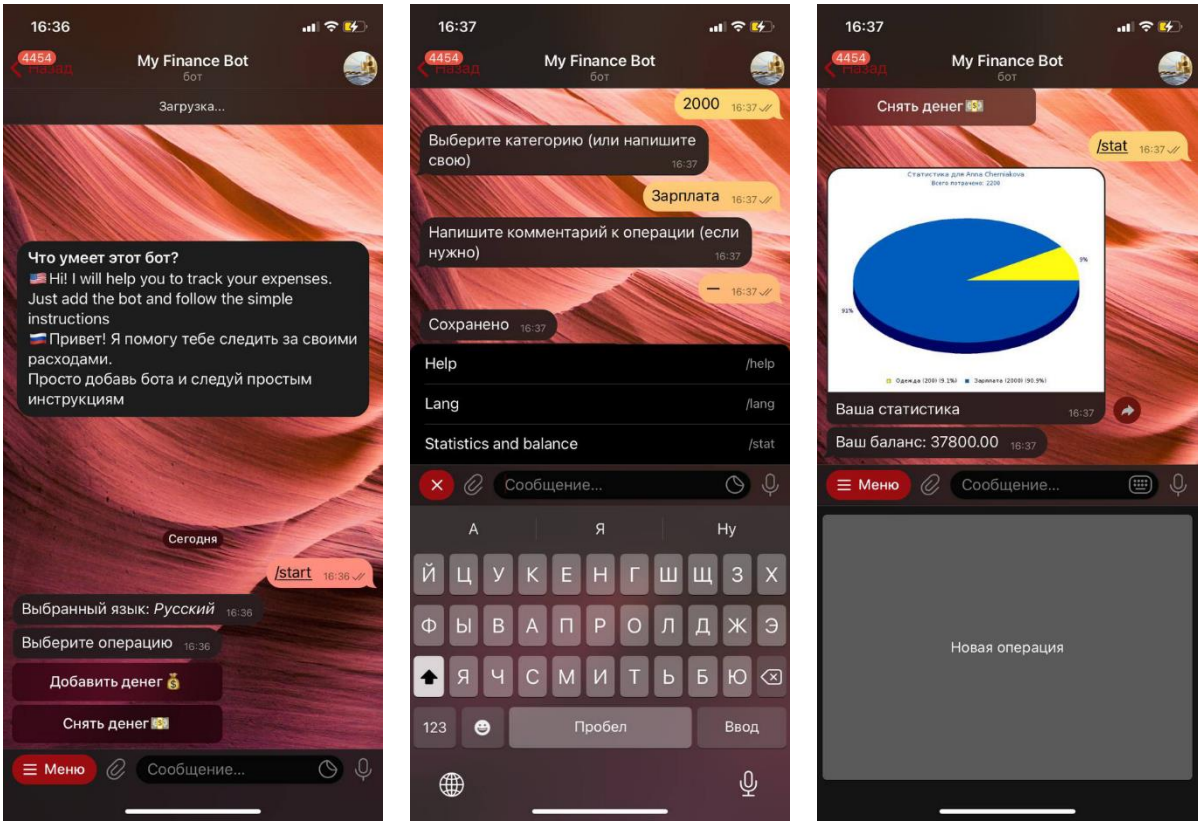
— можуть працювати з перебоями [7, 8].

Дивлячись на всі переваги чат-ботів та додатків, стає ясно, що боти виявилися значним і цінним розвитком технологій, але бувають ситуації, коли вони не вирішують належним чином усі проблеми, які вирішує додаток. Але в той же час, чат-боти кращі за мобільні додатки тим, що не потребують завантаження, не займають місце на пристрої, не прив’язані до конкретної платформи та багато інших сильних сторін. Із урахуванням сучасного розвитку та прогресу в галузі штучного інтелекту, чат-боти поступово стануть частиною життя користувача, адже їх сміливо можна назвати однією з найкорисніших речей, які будь-коли відбувалися всередині технологічної промисловості.

1.2 Огляд відомих рішень для контролю фінансів

Етап огляду відомих рішень є досить важливим, аналіз вже створених продуктів допоможе встановити вимоги до власного проєкту таким чином, щоб це було максимально корисно та зручно для кінцевого користувача. Переглянувши декілька сторінок в інтернеті з рекомендаціями чат-ботів та мобільних додатків для контролю фінансів, було обрано найбільш популярні з них для проведення огляду та аналізу, визначення слабких та сильних сторін кожного. Отже розглянемо таких ботів: «My Finance Bot», «Финансовый бот-ассистент» та мобільні додатки: «Тяжеловато», «Money Lover» [9, 10].

«My Finance Bot» – телеграм-бот для контролю фінансів. Досить простий та зрозумілий у користуванні. Перед початком роботи з ботом, користувач бачить інформацію щодо того, що вміє цей бот, що є корисним. У головному меню міститься три команди для взаємодії з програмою, а саме /help - надає інформацію про можливості бота, /lang – використовується для зміни мови(російська або англійська), /stat – показує статистику витрат у вигляді діаграми. Дві основні операції – це додати гроші та відняти гроші, користувач сам визначає категорії для запису витрат та надходжень. За бажанням, кожного разу при внесенні нової суми, можна додати коментар (рис. 1.1).

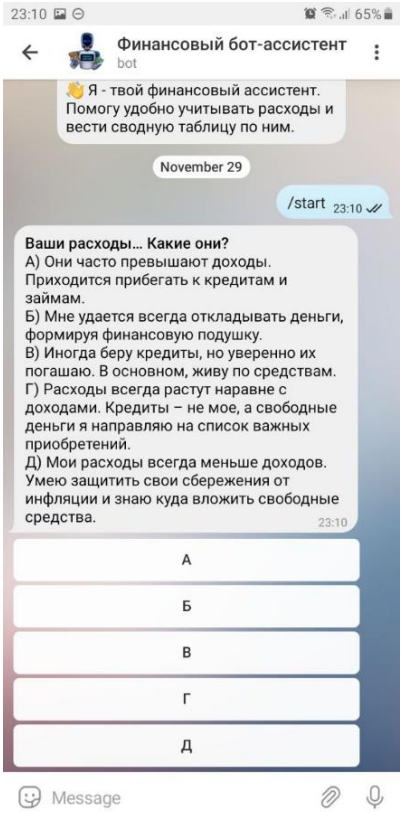


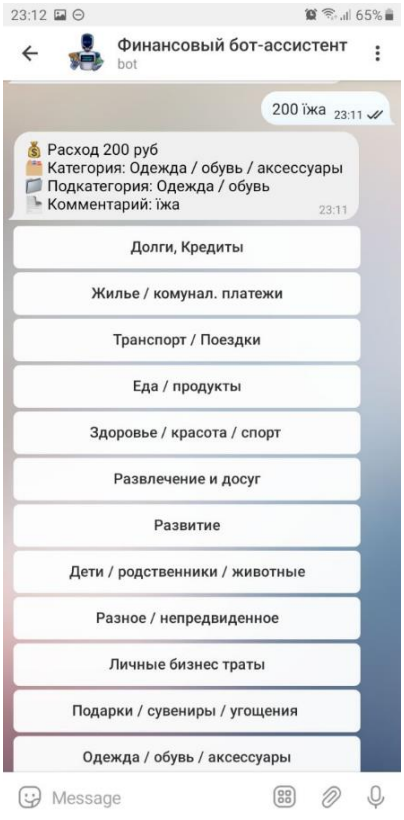
а б в

Рисунок 1.1 – Функції «My Finance Bot»: початок роботи з ботом, основні операції (а), перелік команд меню (б), статистика витрат у вигляді діаграми (в)

Із недоліків, можна виділити те, що команда /help працює не зовсім коректно, окрім інформації щодо можливостей бота, вона має показувати усі доступні команди для взаємодії з програмою, але це недоступно. Крім того, незрозуміла логіка розподілення коштів по категоріях, бот пропонує вибрати категорію або вигадати свою, але ніяких готових варіантів для вибору немає. Категорії, які користувач додав сам, також не зберігаються і подальший їх вибір неможливий. Телеграм-бот надає можливість додати коментар при внесенні витрати, але потім він ніде не відображається, тому наявність даної функції невідома. Мінусом є те, що витрати не фіксуються щомісячно, існує єдина діаграма на весь час. Такий функціонал не можна назвати зручним та корисним, тому що користувач не має змоги відслідковувати свою статистику витрат та надходжень, щоб коригувати її.

«Финансовый бот-ассистент» допомагає контролювати та аналізувати бюджет. Перед початком роботи даний бот пропонує пройти невеликий тест для визначення до якого типу людей по відношенню до грошей належить користувач, і у результаті отримати опис ключових особливостей визначеного типу, поради на майбутнє стосовно фінансів та чим бот-асистент буде корисним для юзера. Великим плюсом є те, що програма містить посилання на відео-урок по користуванню ботом. До основних функцій, які надає програма, відноситься можливість записувати витрати по категоріях, що пропонуються ботом. Також, можна запросити інформацію по витратам. Вона вивантажується у файл з розширенням .xlsx. Файл містить дві вкладинки: «Общая_выгрузка» та «Список_транзакций». У першій знаходиться інформація про витрати по категоріям, що розділено по місяцям, також є підсумок на місяць, щомісячні витрати та всього за рік. Додатково, ці ж дані зображено у вигляді діаграми. На другій вкладинці витрати розписані більш детально: дата та час, сума, категорія. Розробник та його команда відкриті для спілкування, можна задати питання, яке цікавить та, навіть, запропонувати ідею щодо покращення роботи бота (рис. 1.2).



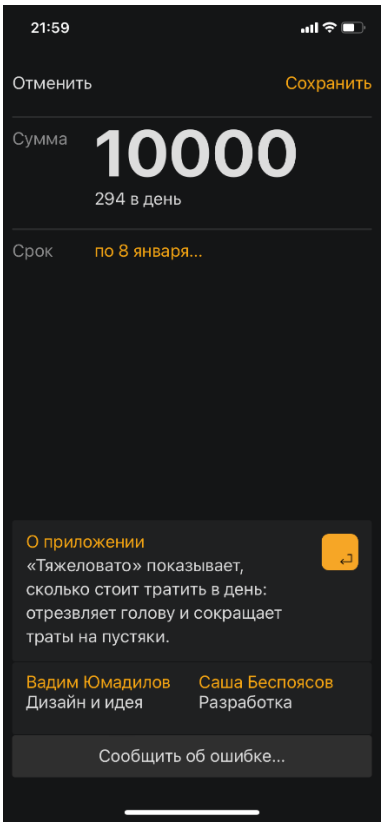


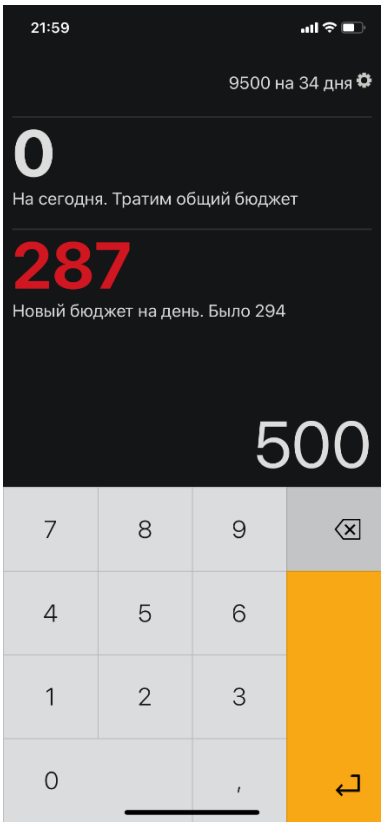
а б в

Рисунок 1.2 – Функції «Финансовый бот-ассистент»: старт роботи з ботом, тест на визначення відношення до грошей (а), приклад внесення витрат і вибір відповідної категорії (б), таблиця витрат (в)

До недоліків, в першу чергу необхідно віднести те, що бот-асистент має обмежений терміни дії – тиждень безкоштовного користування, далі треба оформити підписку на рік, щоб продовжити контролювати свої витрати. Програма не пропонує особливих функцій, за які, дійсно, є сенс платити гроші. Бот не має функції накопичення грошей, що не дозволяє в повному обсязі відстежувати та аналізувати, скільки бюджету фактично було внесено та скільки було витрачено. Можливість додавати коментар до запису лише витрачає час користувача, тому що далі ця інформація недоступна, її немає у файлі з витратами, і, узагалі, ніде немає. Бот кожного дня присилає нагадування, що необхідно внести витрати, але налаштування цієї функції відсутнє. Тобто, користувач не має змоги контролювати процес надсилання сповіщень.

«Тяжеловато» – це мобільний додаток, створений для контролю витрат. Він надає можливість встановлювати бюджет, іншими словами, ліміт на певний проміжок часу, і на основі цього рахує оптимальну суму, яку можна використати протягом дня. Якщо користувач витрачає увесь дозволений бюджет на день, тоді програма попереджає його про це та пропонує почати використовувати ту суму, яка розрахована для наступного дня. Усі витрати записуються в історію, у будь-який час можна продивитися записи. Також, якщо юзер не витратив увесь ліміт на день, тоді додаток пропонує або додати збережені кошти до загальної дозволеної суми на наступний день, або збільшити ліміт бюджету на увесь період, наприклад на місяць. Узагалі, додаток має простий та зручний інтерфейс, усі функції безкоштовні


21:59
Отменить
Сохранить
Сумма
10000
294 в день
Срок
по 8 января...
О приложении
«Тяжеловато» показывает, сколько стоит тратить в день: отрезвляет голову и сокращает траты на пустяки.
Вадим Юмадилов
Дизайн и идея
Саша Беспоясов
Разработка
Сообщить об ошибке...


21:59
9500 на 34 дня
0
На сегодня. Тратим общий бюджет
287
Новый бюджет на день. Было 294
500
7
8
9
4
5
6
1
2
3
0
,

(рис. 1.3).

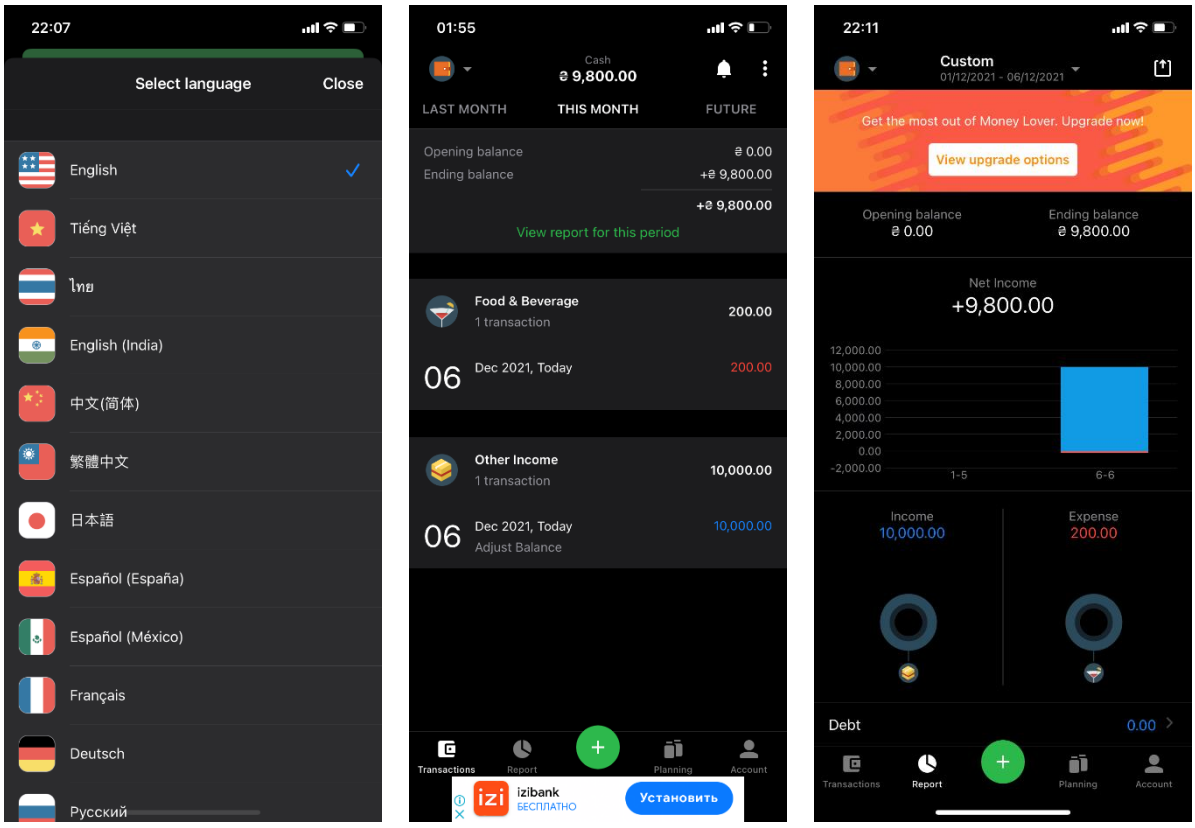

История
9578 на 34 дня
Сегодня, 22:00
63
Сегодня, 22:00
52
Сегодня, 22:00
100
Сегодня, 22:00
36
Сегодня, 21:59
69
Сегодня, 21:59
50
Сегодня, 21:59

а б в

Рисунок 1.3 – Функції «Тяжеловато»: екран для встановлення ліміту (а), приклад внесення витрат, що виходять за дозволений бюджет на день (б), історія витрат за день по годинам (в)

До мінусів можна віднести дуже обмежений функціонал додатка, немає розподілу витрат по категоріям. Програма не надає статистики або файлу з витратами за певний період часу, таким чином користувачу важко буде аналізувати свої витрати. Історія записів зберігається тільки один день, тому користь цієї функції залишається під питанням. Інтерфейс мобільного додатка занадто простий, непривабливий, місцями нагадує стандартну програму калькулятор на телефоні.

«Money Lover» – даний мобільний додаток призначений для контролю фінансів. Він має безліч цікавих функцій, але головними з них є можливість додавати певний бюджет та записувати витрати. Програма підтримує різні мови інтерфейсу, тобто вона не обмежується однією аудиторією. Є окремий режим – подорожі. Коли він ввімкнений, усі витрати позначаються певною міткою для швидкого пошуку. Якщо повернутися до звичайних витрат, то всі вони розподіляються по категоріям, система вже пропонує безліч готових варіантів, але користувач, також, має змогу додати свою власну категорію. До корисних функцій відноситься наявність статистики, період юзер може налаштувати сам. Також, можна подивитися історію витрат. Додаток надає можливість підв’язати банківську карту, щоб автоматично зчитувати та записувати витрати, що були проведені через картку (рис. 1.4).



а б в

Рисунок 1.4 – Функції «Money Lover»: екран для вибору мови інтерфейсу (а), екран з загальною сумою та витратами (б), статистика прибутку та витрат (в)

Серед мінусів можна виділити те, що додаток не має окремої функції накопичення бюджету. Відсутня можливість налаштувати нагадування про необхідність внести кошти. Додаток дуже багатофункціональний, але більшість функцій недоступна, щоб користуватися ними, потрібно оформити підписку. Наприклад, користувачу надається можливість розподіляти кошти по категоріях, але з 20 або, навіть, більше доступних категорій, додати витрати можна тільки у дві. До того ж, створити власні категорії також можна тільки за гроші. До такого небезкоштовного функціоналу відноситься ще й статистика, лише один варіант діаграми доступний для користувача. Експорт звіту про витрати без підписки не працює взагалі. Таких прикладів обмеженого функціоналу ще досить багато, тому використання цього додатку безкоштовно узагалі не має сенсу та не несе ніякої користі юзеру. До того ж, постійна реклама заважає роботі у мобільному додатку.

1.3 Постановка задачі

Провівши аналіз доступних рішень, розглянувши та розібравши приклади вже готових проєктів, було поставлено мету даної роботи – розробити та програмно реалізувати інформаційну технологію планування та контролю бюджету у формі чат-боту.

Чат-бот повинен виконувати такі задачі:

- наявність двох бюджетів: перший – для накопичування грошей, другий – для внесення витрат;
- розподіл витрат по категоріям;
- видалення балансу для кожного бюджету окремо;
- можливість налаштувати дату для сповіщення – нагадування про необхідність відкласти кошти;
- збір інформації про щомісячні витрати та вивантаження даних у файл формату PDF.

Для досягнення поставленої мети необхідно виконати такі завдання:

1) Обрати мову програмування, платформу та середовище для розробки.
2) Підібрати та спроєктувати базу даних.
3) Виконати моделювання системи.
4) Програмно реалізувати телеграм-бота.
5) Виконати тестування готового продукту та варіантів його використання.

2 ВИБІР МЕТОДУ ВИРІШЕННЯ ЗАДАЧІ

У попередньому розділі було розглянуто існуючі форми розроблюваної системи, проведено аналіз відомих рішень та, на основі визначених переваг та недоліків, було вирішено реалізовувати систему у вигляді бота. Далі необхідно визначитися з платформою для розміщення бота.

Telegram, наприклад, є одним з найкращих програм серед сервісів обміну повідомленнями і вже зібрав досить багато користувачів по всьому світу, завдяки безлічі цікавих функцій, простоті та швидкості використання. До речі, месенджер надає не лише можливість відправляти текстові повідомлення, аудіо та відео-дзвінки, але також надає такі функції, як профіль користувача, групи, канали та телеграм-боти на відміну від будь-якої іншої платформи. Telegram орієнтований на конфіденційність, шифрування та API з відкритим вихідним кодом. Останнє означає, що Telegram дозволяє будь-кому використовувати свій API та код, що прискорює розробку та робить її комфортною. На додаток, він зберігає всі повідомлення та дані на захищеному сервері. Це означає, що можна отримати доступ до інформації з будь-якого підключеного пристрою, що робить Telegram багатоплатформенним. Він охоплює всі основні платформи, такі як телефони Android, iOS, Windows з настільними програмами для Mac, Linux та Windows. Крім того, він також має веб-версію [11].

2.1 Вибір мови програмування та середовища

У якості мови програмування було обрано Java Script. Java Script – це текстова мова програмування, яка використовується як на стороні клієнта, так і на стороні сервера, та дозволяє робити веб-сторінки інтерактивними. Згідно з опитуванням розробників 2020 року, JavaScript вважається однією з найпопулярніших технологій. Вона має найбільший та найактивніший репозиторій бібліотечного коду у світі.

В основному, JavaScript використовується для веб-додатків та веб-браузерів. Але крім того, він також використовується за межами Інтернету у програмному забезпеченні, серверах та вбудованих апаратних засобах управління. Ось деякі основні речі, для яких застосовується JavaScript:

1. Додавання інтерактивної поведінки до веб-сторінок.
2. Створення веб-додатків та мобільних додатків.
3. Створення веб-серверів та розробка серверних програм.
4. Розробка ігор [12].

Node.js – це кросплатформенне середовище виконання з відкритим вихідним кодом для розробки серверних та мережевих програм. Програми Node.js написані на JavaScript і можуть запускатися в середовищі виконання Node.js у OS X, Microsoft Windows та Linux.

Він дозволяє розробникам використовувати JavaScript для запуску скриптів на стороні сервера, що означає створення повноцінного вмісту сторінки до того, як цю сторінку буде передано до браузера.

Node.js має архітектуру, засновану на подіях, яка підтримує процес асинхронного введення-виведення (Input/Output). Такий вибір дизайну спрямований на підвищення продуктивності та масштабованості у додатках з кількома операціями введення-виводу, а також додатків реального часу.

Провідні компанії, що використовують Node.js – це Yahoo, LinkedIn, Netflix, Github та багато інших.

Розглянемо особливості Node.js:

1. Швидкий розвиток.

Він побудований на движку Google V8 JavaScript Engine і надає бібліотеку Node.js, яка швидко виконує код.

2. Без буферизації.

Програми, що створені за допомогою Node.js, не буферизують дані. Вони просто масово виконують дані розробників.

3. На основі подій та асинхронно.

Кожен інтерфейс прикладного програмування (API) бібліотеки Node.js є асинхронним. Це означає, що він не блокує інші з’єднання. Сервер Node.js не чекає, доки будь-яка API регресує дані. Він переходить на майбутні API-інтерфейси та систему повідомлень подій, щоб допомогти серверу отримати відповідь від попереднього виклику інтерфейсу прикладного програмування.

4. Ліцензія:

Node.js видається під ліцензією MIT. Він доступний для всіх користувачів.

5. Однопоточний і масштабований:

Порівняно з традиційними методами веб-обслуговування, де кожне з'єднання (запит) породжує новий потік, займаючи системну оперативну пам'ять і, зрештою, досягаючи максимального об'єму доступної оперативної пам'яті, Node.js працює в однопотоковому режимі, використовуючи введення / виведення, що не блокує. Таким чином, Node.js здатний обробляти безліч одночасних підключень з високою пропускною здатністю, що прирівнюється до високої масштабованості.

Основні елементи архітектури Node.js:

1. Запити.

Вхідний запит може бути описаний як неблокуючий (простий) або блокуючий (складний) залежно від задачі, яку користувач вимагає виконати у додатку.

2. Сервер.

Це сервер, який є структурою, задача якої збирати запити від користувача, обробляти їх і повертати відповідь наступним користувачам.

3. Черга подій:

Накопичує вхідний запит і передає такі запити окремо в Event-Loop (цикл подій).

4. Пул потоків.

Він містить усі потоки, доступні для виконання деяких завдань, які можуть знадобитися для виконання запиту клієнта.

5. Цикл подій.

На невизначений термін підтримує процеси та запити, а потім повертає ці відповіді подальшим клієнтам.

6. Зовнішні ресурси.

Ці ресурси мають займатися блокуванням клієнтських запитів. Їх можна використовувати для зберігання даних та обчислень [13].

2.2 Використання бібліотек

Потужність Node.js підтримується фреймворками з відкритим вихідним кодом. За допомогою таких пакетів можна значно заощадити час і розробляти легкі, масштабовані та високошвидкісні програми з величезною продуктивністю.

Пакет (бібліотека) – це код, написаний кимось іншим. Він розроблений, щоб допомогти спільноті легко та швидко вирішувати повсякденні проблеми без переписування коду, особливо для великої програми.

Розглянемо декілька найбільш популярних API фреймфорків для Node.js:

- Telegraf.js – сучасний фреймворк API бота Telegram для Node.js;
- Node-telegram-bot-api – модуль Node.js для взаємодії з офіційним API бота Telegram;
- Telegram-node-bot – потужний модуль вузла для створення ботів Telegram [14].

Розглянемо та проаналізуємо переваги та недоліки кожного з них (табл.2.1).

Таблиця 2.1 – Порівняння фреймворків

	Telegraf	Node-telegram-bot-api	Telegram-node-bot
+	1. Є підтримка Telegram bot API 5.3. 2. Використання TypeScript. 3. Реалізовано функціонал роботи з AWS, FireBase, Glitch, Heroku. 4. Описані вебхуки. 5. Легко розширити.	1. Реалізовані контролери для повідомленя, колбеків та інлайнів. 2. Кластеризація; 3. Є веб панель адміністрування.	1. Є докладні туторіали. 2. Експериментальні функції. 3. Внутрішній API.
-	1. Відсутня асинхронність для реалізації функцій робота.	1. Мало документації. 2. Неможливо розширити. 3. Немає готового функціоналу для роботи зі сторонніми веб-сервісами.	1. Відсутнє логування. 2. Обробка помилок працює не завжди коректно.

Дивлячись на порівняльну таблицю, можна підсумувати кількість слабких та сильних сторін кожної бібліотеки та побачити, що Telegraf є лідером за кількістю переваг, саме тому я обрала цей фреймворк для свого проєкту.

2.3 Аналіз та вибір СУБД

Після визначення мови програмування, платформи для розробки чат-бота та допоміжної бібліотеки, необхідно перейти до вибору СУБД.

Однією з найважливіших складових будь-якого успішного програмного продукту є розумно організований спосіб управління даними. Система має вміти швидко отримувати та обробляти інформацію, надійно зберігати отримані дані та забезпечувати дотримання політик конфіденційності та безпеки даних. Цих вимог можна легко досягти за допомогою правильно обраної системи управління базами даних.

База даних – це систематичний збір даних. Вона підтримує електронне зберігання та обробку даних. Бази даних полегшують процес управління даними.

Системи управління базами даних (СУБД) – це інструменти управління базами даних, єдина технологія, яка допомагає підприємствам оптимізовувати, управляти, зберігати та витягувати дані з основних баз даних. Інтерфейс користувача СУБД – це системний підхід, який може обробляти великі обсяги робочих навантажень з даними.

Базова структура СУБД ґрунтується на трьох життєво важливих елементах: самі дані, схема бази даних, ядро бази даних, де схема відповідає за логічну структуру, а двигун забезпечує доступ, блокування та зміну даних користувачами.

До появи СУБД використовувалися пласкі файлові системи. У плоских файлових системах дані зберігаються в окремих списках або таблицях, які можуть містити лише обмежені обсяги керованих даних. До цих таблиць можна отримати доступ лише з одним файлом за раз і, зазвичай, не використовувати разом із іншими. Занадто великі таблиці у плоских файлових системах стало дуже незручно обробляти та підтримувати. Саме

тому користувачі часто створювали багато таблиць, кожна для певної мети, незалежно від того, чи містять ці таблиці схожі дані.

У цілому нині, розробка СУБД справила революцію обсягом керованого сховища даних. Це також різко змінило безпеку даних, спосіб зберігання та вилучення даних, можливість спільного використання даних та кількість одночасних користувачів.

У залежності від цілей розроблюваного проєкту, особливостей зберігання та обробки даних використовують різні системи управління базами даних. До основних типів СУБД на основі моделі даних відносять:

— реляційна;

— об'єктно-орієнтована;

— ієрархічна;

— мережева.

База даних відносин

Система управління реляційними базами даних (СУБД) – це система, в якій дані організовані у двовимірні таблиці з використанням рядків та стовпців.

Це одна з найпопулярніших моделей даних, яка використовується у багатьох різних галузях. Вона базується на SQL. Кожна таблиця у базі даних має ключове поле, яке однозначно ідентифікує кожен запис. Програмне забезпечення системи управління реляційними базами даних доступне для персональних комп'ютерів, робочих станцій та великих мейнфреймів.
Ось приклади СУБД такого типу: Oracle Database, MySQL, Microsoft SQL Server і т.д.

Об'єктно-орієнтована база даних

Це система, в якій інформація або дані представлені у формі об'єктів, що використовується в об'єктно-орієнтованому програмуванні. Іншими словами, це поєднання концепцій реляційних баз даних та об'єктно-орієнтованих принципів. Перше надає можливість управління паралелізмом, транзакції, а принципи ООП – це інкапсуляція даних, успадкування та

поліморфізм. Варто зазначити, що така СУБД вимагає менше коду та проста в обслуговуванні.

Прикладом такої системи управління базою даних є Object DB.

Ієрархічна база даних

Це система, в якій елементи даних мають відношення "один до багатьох" (1: N). Тут дані організовані у вигляді дерева, що схоже на структуру папок у звичайній комп'ютерній системі. Ієрархія починається з кореневого вузла, поєднуючи всі дочірні вузли з батьківським вузлом. До того ж, одному дочірньому вузлу відповідає тільки один кореневий вузол.

Такий тип СУБД частіше використовується у промисловості на платформах мейнфреймів.

Наприклад, IMS (IBM).

Мережева база даних

Система управління мережевою базою даних – це система, в якій елементи даних підтримують стосунки один до одного (1: 1) або багато відношення до багатьох (N: N). Вона також має ієрархічну структуру, але дані організовані як граф, і для одного дочірнього запису можна мати більше одного батька.

Серед таких СУБД найбільш популярними є, наприклад, СООБЗ Cerebrum, dbVista [15].

Серед розглянутих існуючих типів СУБД, по характеристикам та опису, для розроблюваного продукту найбільше підходить реляційна.

Розглянемо декілька найбільш популярних реляційних СУБД:

Postgre SQL – це розширена реляційна база даних корпоративного класу з відкритим вихідним кодом, яка підтримує запити як SQL (реляційні), так і JSON (нереляційні). Postgre включає такі функції, як успадкування таблиць та навантаження функцій, які можуть бути важливими для певних додатків. Також, вона суворо дотримується стандартів SQL. Модуль поділу та успадкування дозволяє розбивати великі таблиці на дрібніші розділи, підвищуючи загальну продуктивність запитів. Крім того, можна виконувати

операції резервного копіювання, відновлення та цілісності даних з використанням декількох методологій, таких як логічна реплікація, синхронна реплікація, відновлення на певний момент часу та гаряче резервування.

Плюси:

— зберігання та управління даними у великих обсягах;
— досить безпечна обробка даних;
— простий процес встановлення в операційних системах (ОС) Linux та Windows;
— доступність корисних матеріалів, таких як навчальний посібник з вивчення інструменту;
— підходить для великого обсягу даних.

Мінуси:

— власний інтерфейс обмежує маніпуляцію з даними;
— розширений характер інструменту уповільнює додавання невеликих баз даних;
— займає багато пам’яті;
— не дуже висока продуктивність;
— складне встановлення та налаштування програмного забезпечення [16].

My SQL – це інструмент для високошвидкісної обробки та підвищення продуктивності даних з широким набором функцій, яка дозволяє безпечно записувати, оновлювати та керувати даними відповідно до GDPR, PCI, HIPPA та різних нормативних стандартів. Можливості включають маскування даних, аудит, деідентифікацію, моніторинг у реальному часі, ініціалізацію, встановлення виправлень та аутентифікацію. Інструмент розроблений для підвищення безпеки та масштабованості баз даних. Пропонує технічну підтримку та протидіє потенційним ризикам.

MySQL – одна з найпопулярніших СУБД, що реалізує модель клієнт-сервер. Вона може створювати базу даних для зберігання та обробки даних, визначаючи взаємозв'язок кожної таблиці. Потім клієнти мають змогу

зробити запит, набравши певні оператори SQL MySQL. Нарешті, серверна програма відповість на запитану інформацію, і вона з'явиться на стороні клієнта.

Плюси:

— безпека даних;

— масштабованість за запитом;

— цілодобова безвідмовна робота;

— комплексна транзакційна підтримка;

— повний контроль робочого процесу;

— гнучкість відкритого вихідного коду;

— широко прийнятий та простий у використанні;

— швидкий, портативний та безпечний.

Мінуси:

— обмежено складною бізнес-логікою;

— можуть бути проблеми зі стабільністю;

— транзакції обробляються не дуже ефективно;

— функціональність, як правило, дуже залежить від доповнень;

— існує надмірна залежність від сторонніх надбудов [17].

Oracle – продукт корпорації Oracle, який надає систему управління реляційними базами даних. Підтримує будь-які моделі даних та має різні випуски продуктів, такі як Standard Edition, Enterprise Edition, Express Edition та Personal Edition, серед яких користувач може вибрати систему баз даних залежно від своїх потреб. Oracle Database Cloud – це рішення «база даних як послуга», яке допомагає підприємствам керувати даними за допомогою автоматичного виправлення, резервного копіювання, відновлення, зберігання та багато іншого. Він використовує машинне навчання для автоматизації завдань, що повторюються, і дозволяє користувачам керувати адміністративною звітністю через сервісну консоль на основі браузера.

Узагалі, така багатомодельна система управління реляційними базами даних, призначена для корпоративних обчислень і сховищ даних. Це один із перших варіантів для підприємств, що пропонують рентабельні рішення для своїх додатків та управління даними. Він підтримує SQL як мову запитів взаємодії з базою даних.

У даний час база даних представлена у п'яти різних редакціях залежно від доступних функцій:

1. Standard edition one: підходить для односерверних або розгалужених бізнес-програм з обмеженими функціями.
2. Standard edition: надає всі можливості standard edition one. Крім того, забезпечує підтримку більших комп'ютерів та службу кластеризації oracle real application.
3. Enterprise edition: ця версія містить такі функції, як безпека, продуктивність, масштабованість та доступність, що необхідні для критично важливих програм, в яких задіяна онлайн-обробка транзакцій;

   Експрес-випуск є випуском початкового рівня, який можна безкоштовно завантажити, встановити, керувати, розробляти та розгортати;
4. Personal edition: він має ті ж функції, що й enterprise edition, крім oracle real application clustering.

   Плюси:

— сильна технічна підтримка та документація;

— функціональність;

— швидке відновлення;

— швидкість та продуктивність;

— підтримка декількох баз даних;

— надійність;

— безперебійний транзакційний процес та безпека даних;

Мінуси:

— несумісність і складність;

— труднощі в управлінні структурою;

— клієнтські програми часто підключаються автоматично;

— безкоштовні випуски обмежені з погляду функціональності;

— ресурсомістка технологія;

— складна у вивченні [18].

Розглянувши та проаналізувавши існуючі типи СУБД за моделлю даних, розібравши переваги та недоліки розповсюджених систем управління базами даними, для даної роботи було обрано реляційну СУБД MySQL. Вона має багато переваг над іншими варіантами, гарантує безпеку даних, швидка та високопродуктивна. До того ж, MySQL є найбільш популярною системою управління базами даних, вона дає зручний доступ для управління БД та підтримує велику кількість таблиць різних типів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проєктування бази даних

Розглянемо спроєктовану базу даних. Вона містить дві основні таблиці: «users» та «waste», між якими існує інформаційний зв’язок (рис. 3.1).

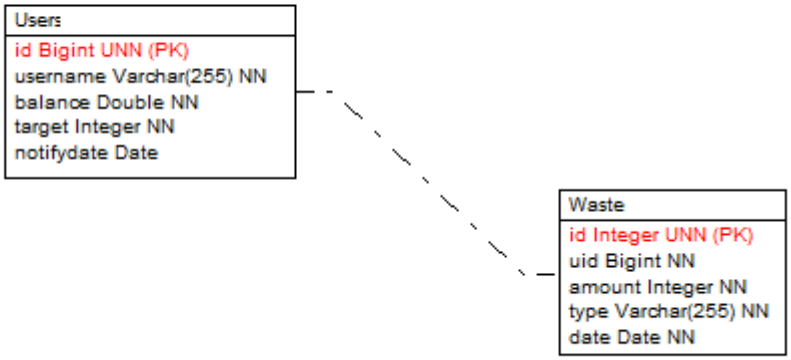


Рисунок 3.1 – ER-діаграма бази даних

Таблиця «users» зберігає дані про усіх користувачів бота та записи їх бюджету накопичень, витрат і дату встановлення сповіщення. У таблиці «waste» міститься інформація про усі витрати юзерів, яка далі використовуються для експорту у файл із записами витрат (табл.3.1).

Таблиця 3.1 – Структура бази даних

Таблиця	Поле	Зміст	Тип	Ключі
users	id	Ідентифікатор юзера	bigint	PK
	username	Ім’я юзера	varchar(255)	
	balance	Баланс накопичення	double	
	target	Баланс витрат	integer	
	notifydate	Дата нотіфікації	date	
waste	id	Ідентифікатор порядкового номеру запису витрати	integer	PK
	uid	Ідентифікатор юзера	bigint	
	amount	Сума витрати	integer	
	type	Категорія витрати	varchar(255)	
	date	Дата внесення витрати	date	

3.2 Моделювання системи

Етап моделювання майбутнього проєкту є невід’ємною частиною розробки у великих та успішних ІТ-компаніях. Це дозволяє чітко визначити вимоги до системи, продумати структуру, зрозуміти логіку та порядок роботи, побудувати план розробки. Потік ідей у голові може бути важко реалізувати так, як хотілося б та не забути про якісь важливі деталі, саме тому ідею та загальне уявлення майбутнього проєкту необхідно візуалізувати. Бачення повної картини задуманого допоможе визначити слабкі сторони, недопрацювання, місця з втраченою логікою. На основі чого, можна буде змінювати проєкт до тих пір, доки він повністю не задовольнить виконавця.

Для більш чіткого розуміння програми та пришвидшення розробки, було спроєктовано діаграму, яка демонструє основні функції, які повинен надавати бот, щоб покрити усі вимоги до проєкту. Отже, згідно діаграми, чат-бот повинен мати 1 головне меню та 3 допоміжних. Блоки поєднані між собою стрілочками, щоб побачити їх взаємодію між собою. У кожному блоці описано функції, які надає кожне меню відповідно (рис. 3.2).

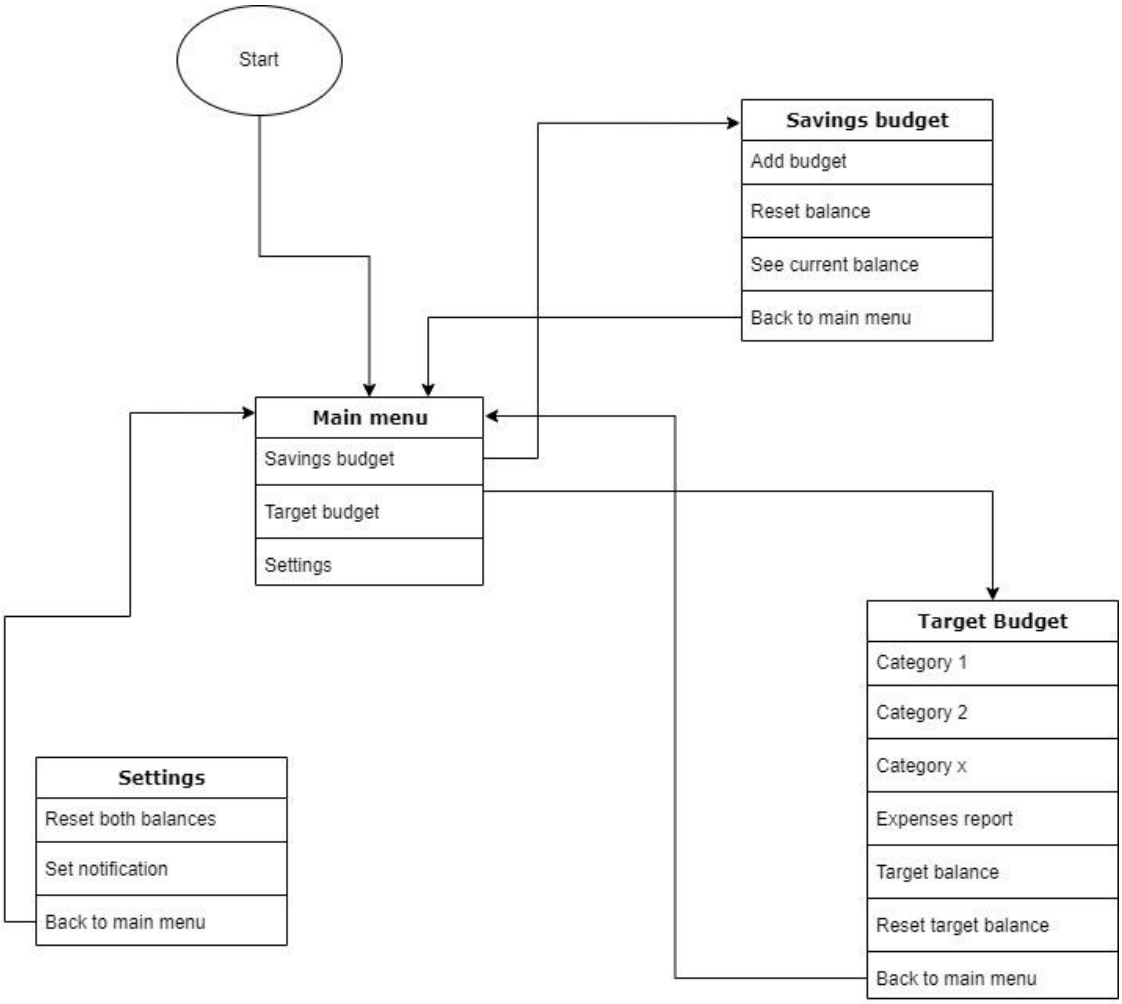
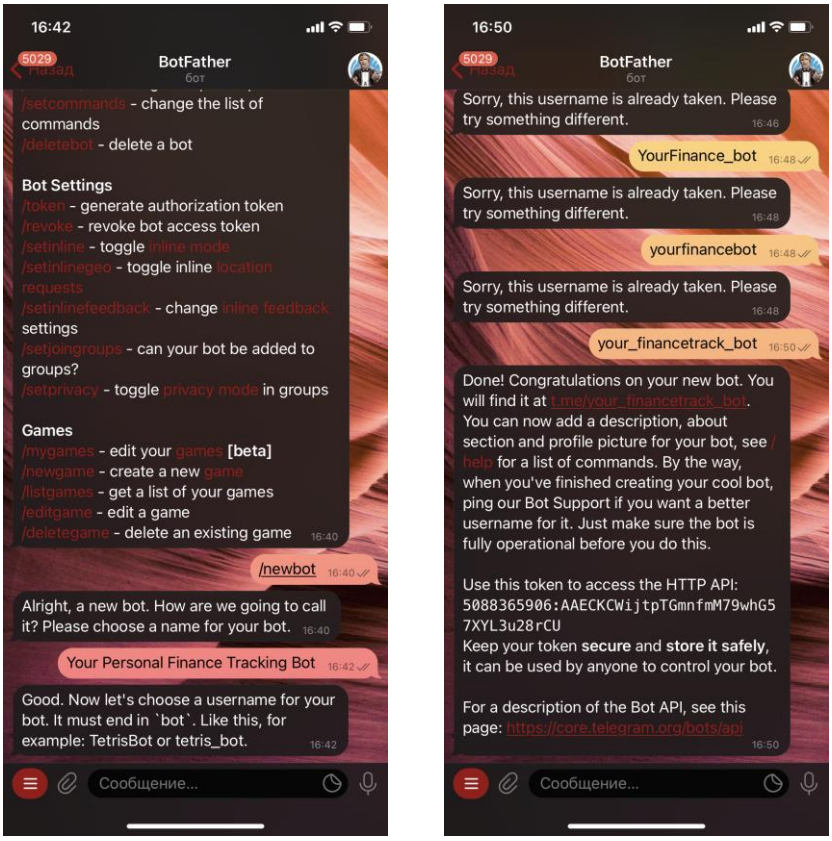


Рисунок 3.2 – Діаграма меню та основних функцій бота

3.3 Розробка та програмна реалізація

Перед початком розробки необхідно авторизувати нового бота в системі Telegram та отримати API токен. Це унікальний набір символів, який надається кожному боту при створенні. Він необхідний для того, щоб керувати чат-ботом за допомогою HTTPS-запитів із Telegram Bot API. Його можна отримати за допомогою іншого бота BotFather, цю інформацію надає

офіційне керівництво месенджера Telegram. Отже, щоб почати взаємодіяти з ботом, необхідно відкрити месенджер Telegram та ввести у пошук BotFather й натиснути на нього. Після того, як чат із ботом відкриється, необхідно натиснути кнопку /start. У результаті стає доступним список усіх можливих команд, які надає бот, обираємо команду /newbot. Далі необхідно ввести назву майбутнього бота, після чого BotFather попросить вигадати нікнейм, важливо, щоб ця назва закінчувалася на bot. Телеграм бот отримав назву «Your Personal Finance Tracking Bot». Токен успішно створено, тепер можна переходити до розробки (рис. 3.3).



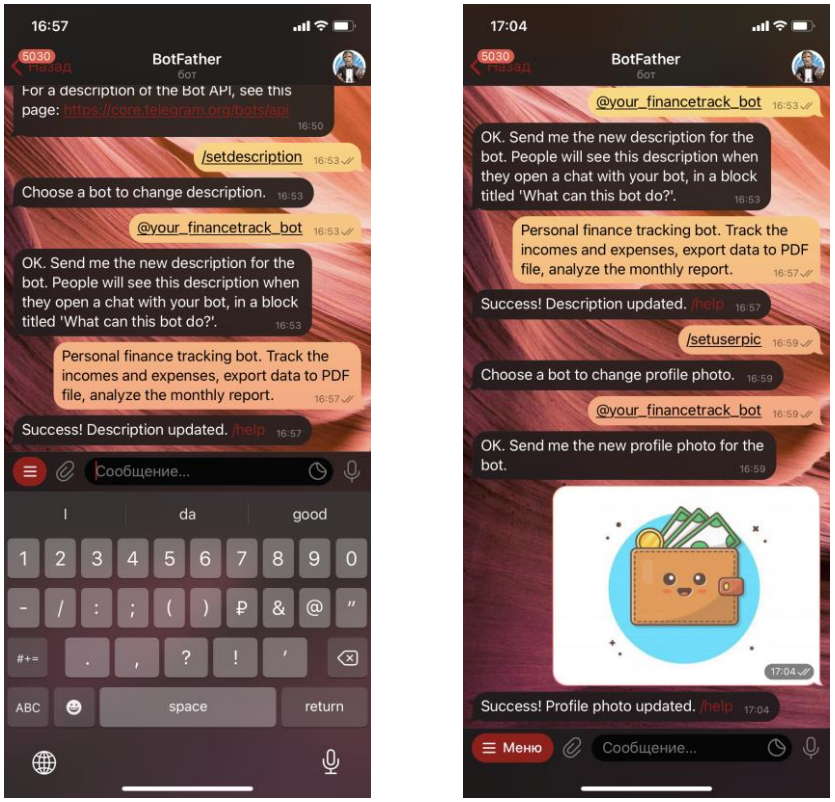
а б

Рисунок 3.3 – Запит на створення бота та вибір назви (а), вибір нікнейму та отримання API токена (б).

Але перед цим ще можна скористатись іншими командами від BotFather, щоб зробити бота більш інформативним та дружнім до користувача. Наприклад, за допомогою команди /setdescription додамо інформацію про те, що вміє «Your Personal Finance Tracking Bot».

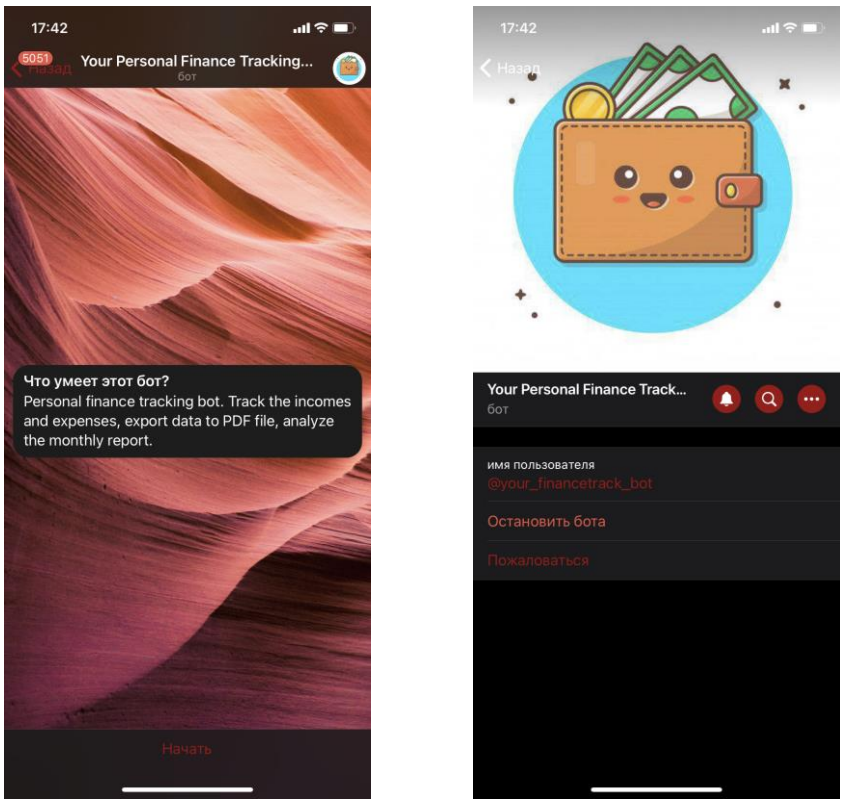
Скориставшись командою /setuserpic встановим зображення профілю бота (рис. 3.4).

У результаті отримано основу для подальшої розробки. Бот показує привітальне інформативне повідомлення та має фото профілю (рис. 3.5).



а б

Рисунок 3.4 – додавання опису бота (а), встановлення фото профілю (б)



а б

Рисунок 3.5 – Інформативне повідомлення (а), іконка телеграм-бота (б)

Перед початком розробки було встановлено платформу Node.js та Visual Studio Code – це спрощений, але потужний редактор коду, що підтримує такі операції розробки, як налагодження, виконання завдань та контроль версій. Далі було створено папку для майбутнього проєкту з назвою «bot». У Visual Studio Code пишемо команду `npm init -y`, щоб ініціалізувати новий проєкт. Npm – це пакетний менеджер, за допомогою якого можна встановлювати будь-які доступні модулі. Створюємо у ньому новий файл – app.js, у якому буде зберігатися основний код. Встановимо бібліотеку Telegraf, яка надасть додаткові можливості та прискорить розробку, ввівши команду `npm i telegraf`. Після успішного встановлення, додався новий каталог під назвою node_modules, у якому зберігаються сам Telegraf та всі залежності, які необхідні для його роботи. Також, було встановлено інші додаткові модулі для подальшої розробки.

Відкриємо файл package.json, у ньому знаходиться базова інформація про проєкт, така як назва програми, версія, ім’я автора, залежності, підключені бібліотеки (рис. 3.6).

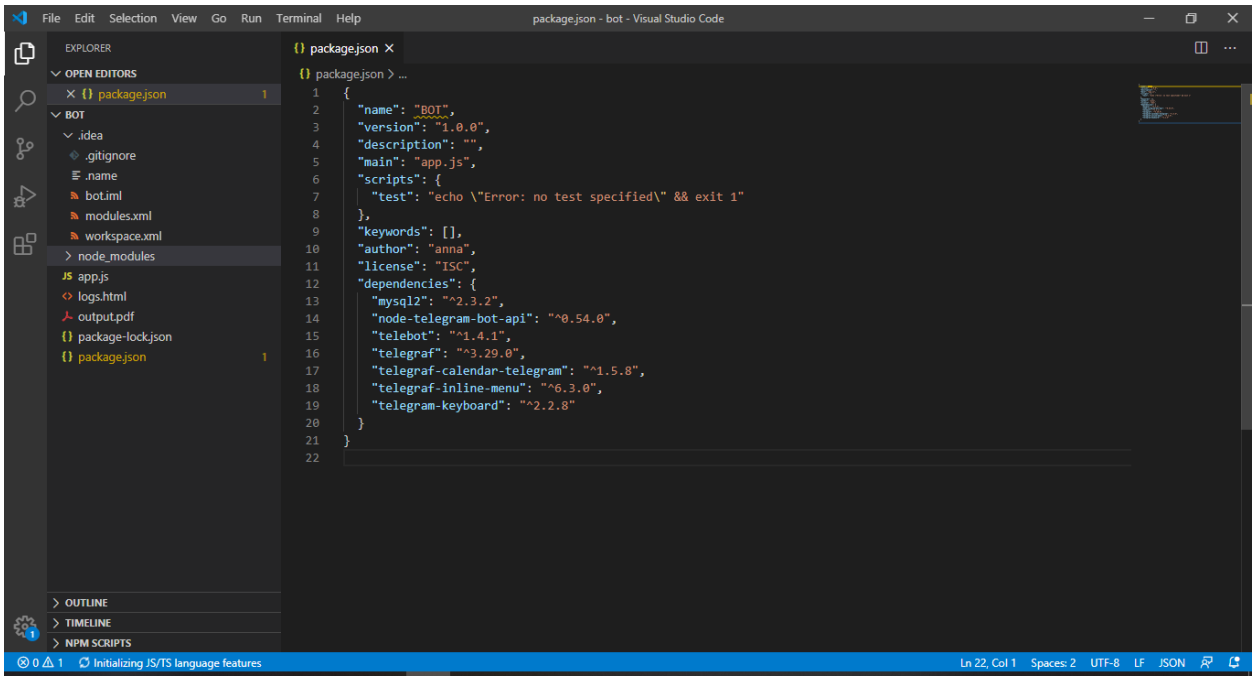


Рисунок 3.6 – Код із основною інформацією про проєкт

Завершальним етапом для початку розробки є підключення унікального API токену нашого бота, який раніше було отримано від BotFather. Це відбувається у головному файлі app.js. Отриманий токен необхідно вставити у const token = “…” (рис. 3.7).

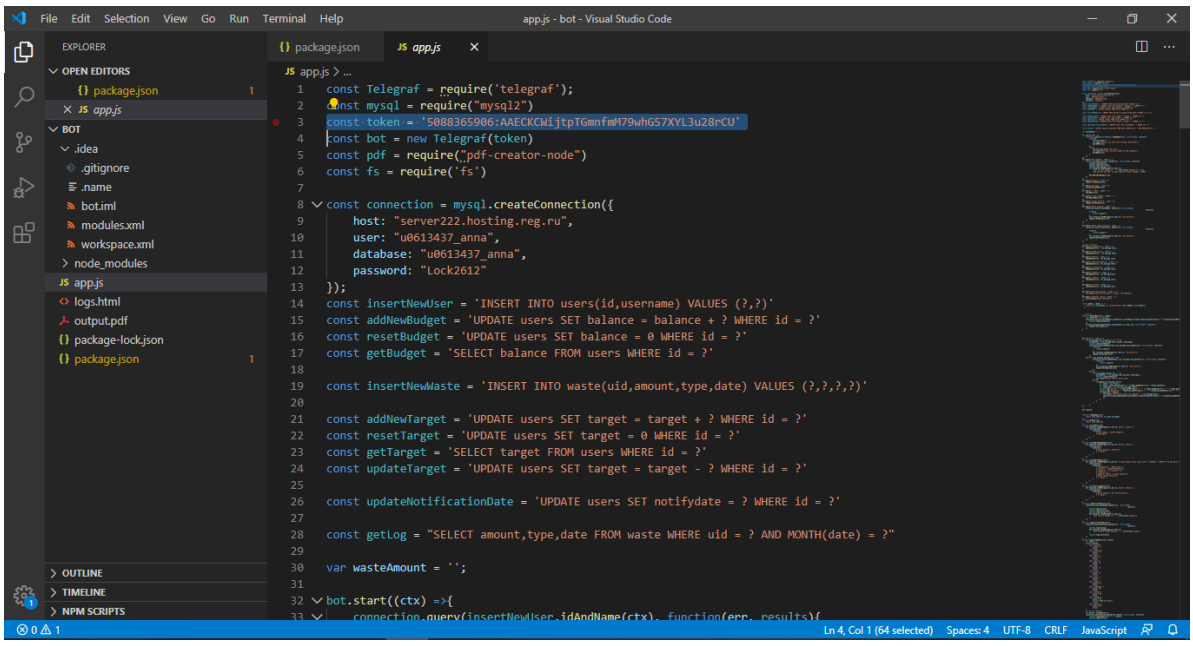


Рисунок 3.7 – Місце знаходження Telegram API токена в коді

Розглянемо інші частини основного коду (рис. 3.8–3.10).

```
14  const insertNewUser = 'INSERT INTO users(id,username) VALUES (?,?)'
15  const addNewBudget = 'UPDATE users SET balance = balance + ? WHERE id = ?'
16  const resetBudget = 'UPDATE users SET balance = 0 WHERE id = ?'
17  const getBudget = 'SELECT balance FROM users WHERE id = ?'
18
19  const insertNewWaste = 'INSERT INTO waste(uid,amount,type,date) VALUES (?,?,?,?)'
20
21  const addNewTarget = 'UPDATE users SET target = target + ? WHERE id = ?'
22  const resetTarget = 'UPDATE users SET target = 0 WHERE id = ?'
23  const getTarget = 'SELECT target FROM users WHERE id = ?'
24  const updateTarget = 'UPDATE users SET target = target - ? WHERE id = ?'
25
26  const updateNotificationDate = 'UPDATE users SET notifydate = ? WHERE id = ?'
27
28  const getLog = "SELECT amount,type,date FROM waste WHERE uid = ? AND MONTH(date) = ?"
```

Рисунок 3.8 – Основні запити у базу даних

```
32  bot.start((ctx) =>{
33      connection.query(insertNewUser,idAndName(ctx), function(err, results){
34          if(err){
35              console.log(err)
36              ctx.reply("Sorry, you have been already registered")
37              buildMenu(ctx)
38          }
39          else{
40              console.log("Added new user")
41              ctx.reply("Welcome, you were added to the database")
42              buildMenu(ctx)
43          }
44      })
45  });
```

Рисунок 3.9 – Варіанти привітального повідомлення бота в залежності від того, чи зареєстровано користувача у системі

```
function createLog(ctx,json){
    var html = fs.readFileSync('logs.html', 'utf8')
    var options = { format: "A3", orientation: "portrait", border: "10mm" };
    var document = {
        html: html,
        data: {
            wastes: json
        },
        path: "./output.pdf"
    };
    pdf.create(document, options)
        .then(res => {
            console.log(res)
        })
        .catch(error => {
            console.error(error)
        });
    ctx.telegram.sendDocument(ctx.from.id, {
        source: './output.pdf',
        filename: './output.pdf'
    }).catch(function(error){ console.log(error); })
}
```

Рисунок 3.10 – Формування та вивантаження історії витрат користувача

3.4 Тестування готового програмного продукту та варіантів його використання

Для того щоб перевірити працездатність бота, його було успішно запущено та протестовано на мобільному телефоні та на комп'ютері. Телеграм-бот працює коректно, швидко та без помилок. Результати тестування на різних пристроях наведено нижче (рис. 3.6–3.15).

Коли бот вперше запущено, на екрані з'являється коротка інформація про те, які функції він надає. Такий опис корисний для користувача, який встановив бота вперше. Після натискання на кнопку /start бот може відповісти по-різному в залежності від того, новий це користувач чи ні. Якщо новий, то бот відповість привітальним повідомлення. Якщо користувач вже існує в базі даних, то бот йому про це повідомить. Одночасно з одним із двох варіантів вітального повідомлення, відкривається меню із трьома кнопками: «Savings», «Target Budget», та «Settings». Користувач може обрати опцію, яка його цікавить. «Savings» – це збереження користувача. Юзер може

накопичувати гроші для якоїсь мети, здійснення мрії тощо. «Target Budget» – це окремий бюджет, виділений на витрати. «Settings» – це налаштування (рис. 3.11–3.12).

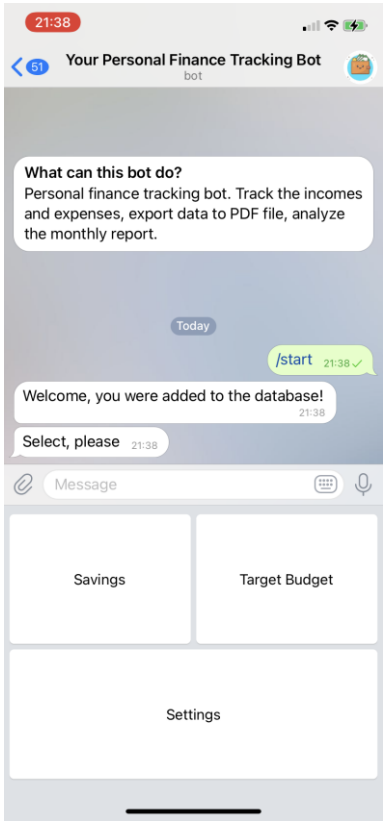


Рисунок 3.11 – Повідомлення-привітання від бота для нового користувача

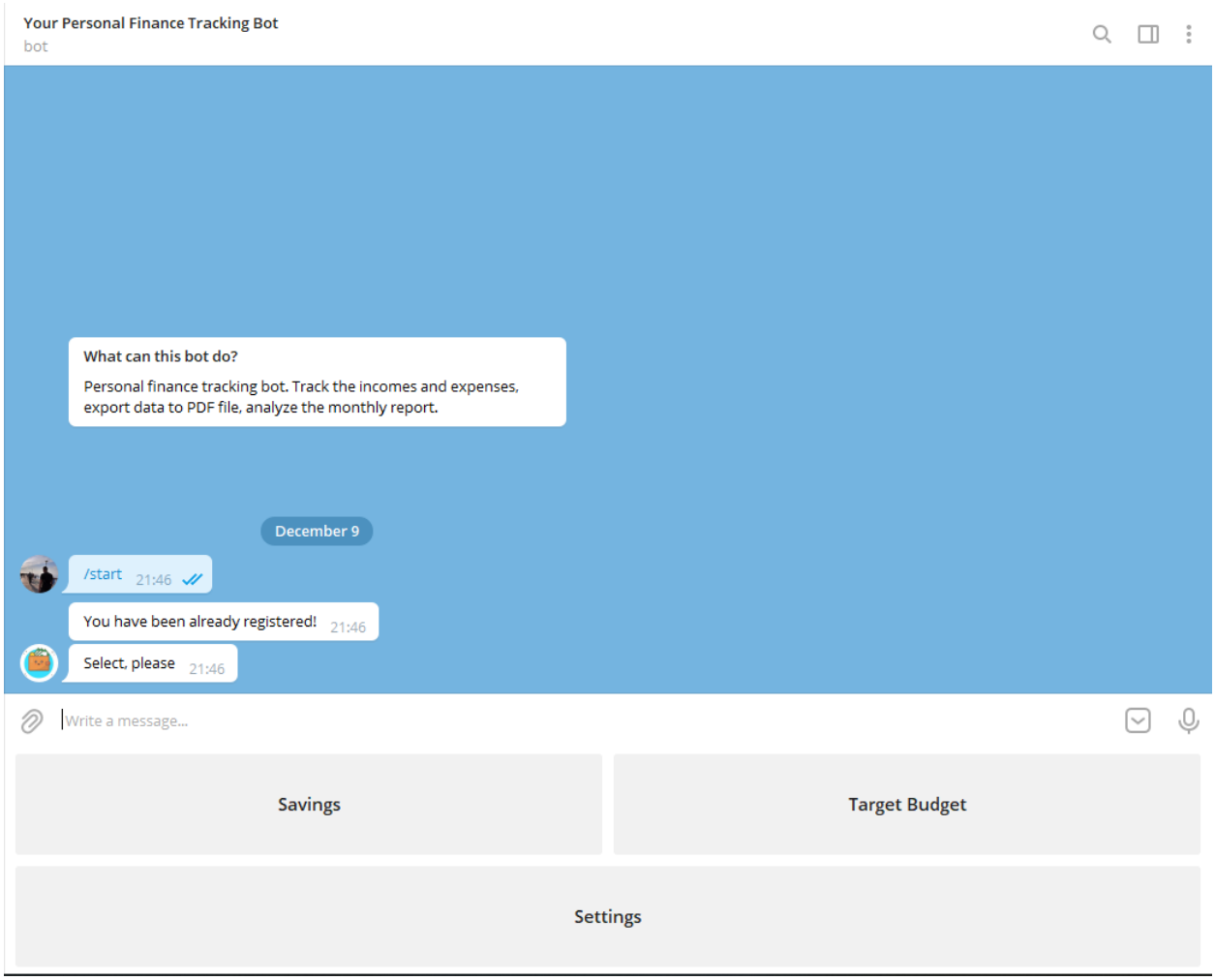


Рисунок 3.12 – Привітальне повідомлення від бота для вже зареєстрованого юзера

При натисканні кнопки «Збереження», відкривається інше меню з кнопками: «Reset Balance», «Balance», «Back». Одночасно, бот присилає повідомлення, де вказується актуальний бюджет користувача та коротка інструкція, як додати гроші. Для цього необхідно ввести бажану суму через +, наприклад +3000. Кожен раз після чергового оновлення бюджету, бот показує поточний баланс. Також, його можна побачити, натиснувши кнопку «Balance». «Reset Balance» призначено для видалення усього бюджету. Кнопка «Back» повертає користувача у головне меню (рис. 3.13–3.14).

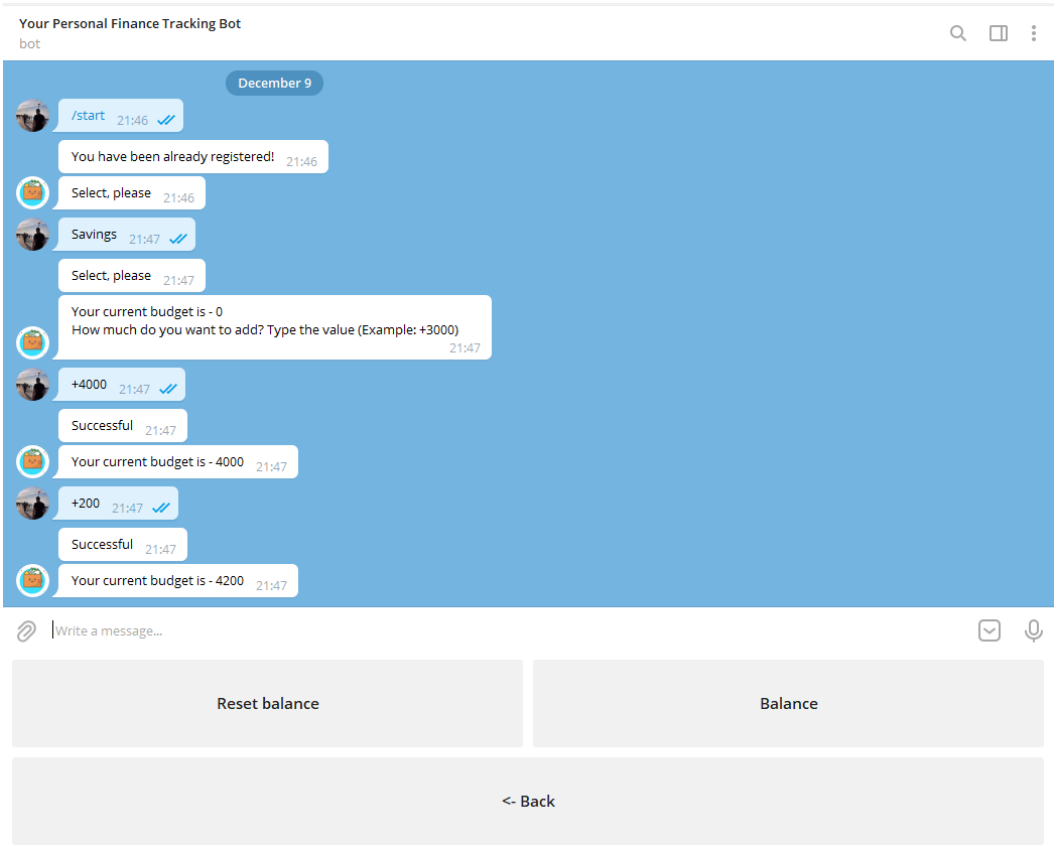


Рисунок 3.13 – Процес поповнення бюджету накопичень

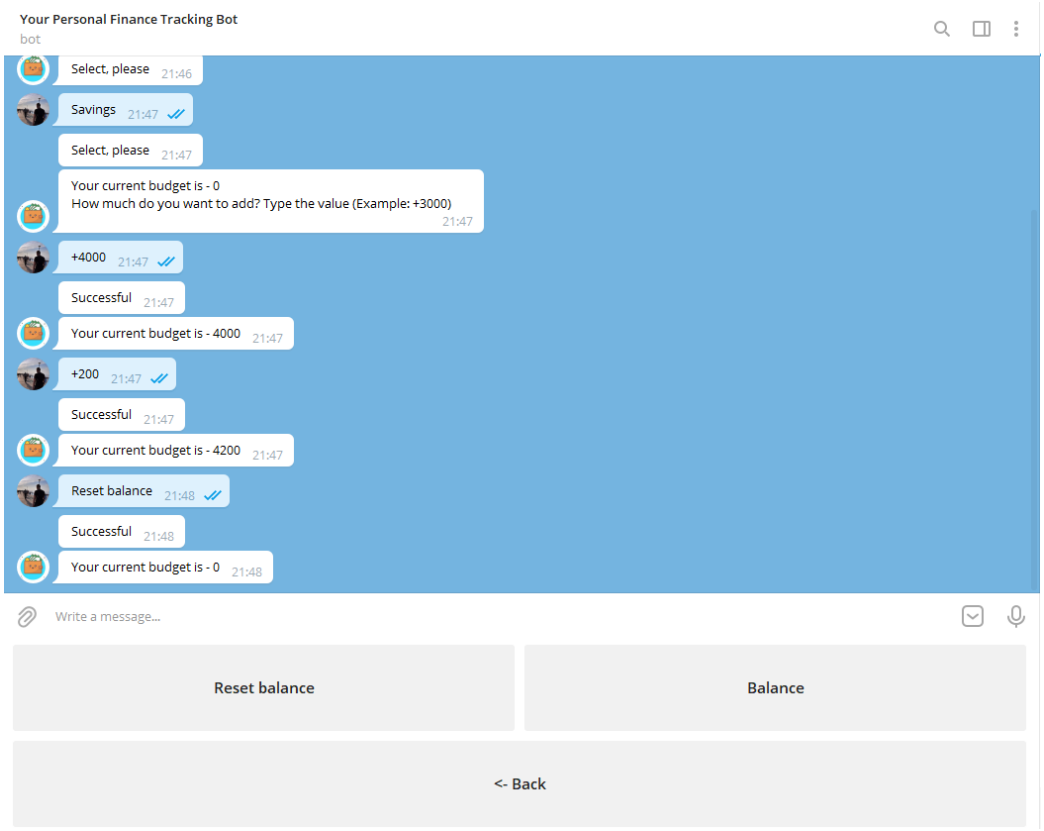


Рисунок 3.14 – Процес видалення усіх збережень

При натисканні на кнопку «Settings» в основному меню, відкриваються наступні можливі опції: «Reset Balance», «Set reminder to top up your savings» та «Back». Перше видаляє весь бюджет, тобто той, що призначений для збережень і другий – для витрат. «Set reminder date to update your savings» надає можливість встановити сповіщення-нагадування, щодо необхідності відкласти гроші у збереження. Користувач має змогу обрати дату такого повідомлення самостійно. Після того, як дату обрано, бот присилає повідомлення, про успішне встановлення нагадування. Можна повернутися у головне меню, натиснувши «Back». Після того, як настане дата, на яку було встановлено сповіщення, бот відправить повідомлення з нагадуванням внести кошти (рис. 3.15–3.17).

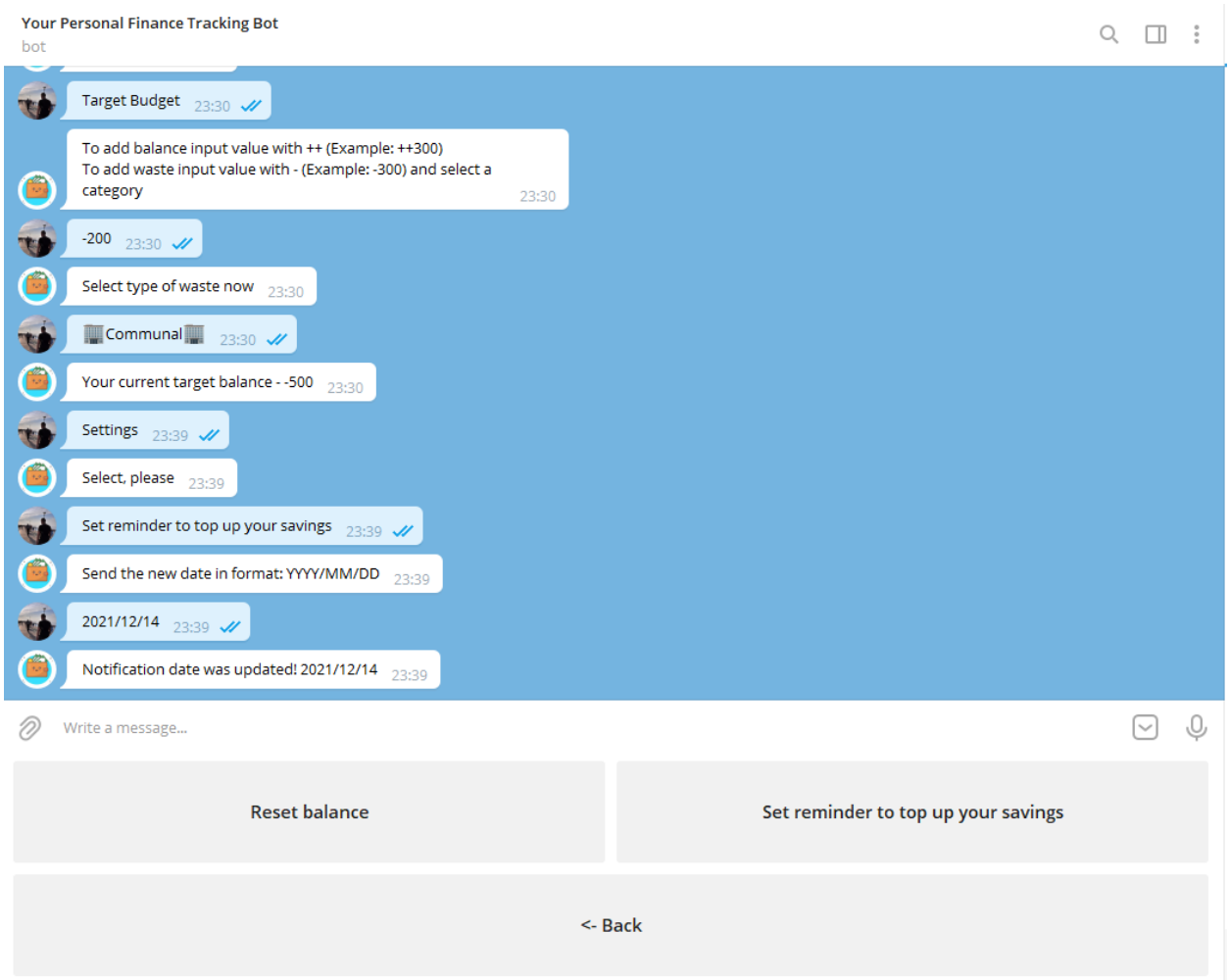


Рисунок 3.15 – Успішне повернення користувача назад у головне меню. Перехід до налаштувань та встановлення дати сповіщення-нагадування про необхідність внести кошти у бюджет накопичень

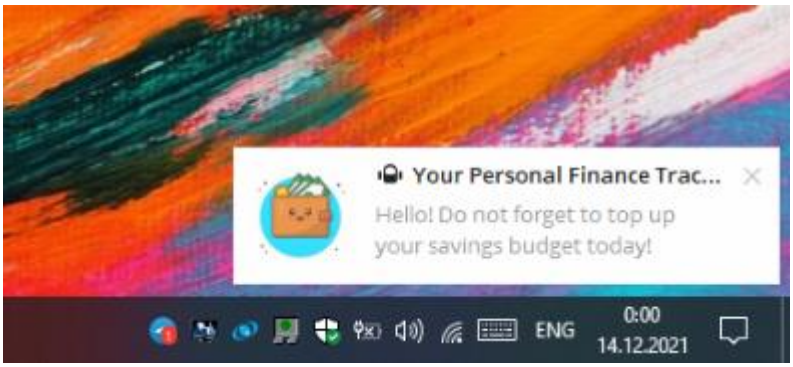


Рисунок 3.16 – Спливаюче сповіщення від бота на робочому столі з нагадуванням поповнити бюджет накопичень

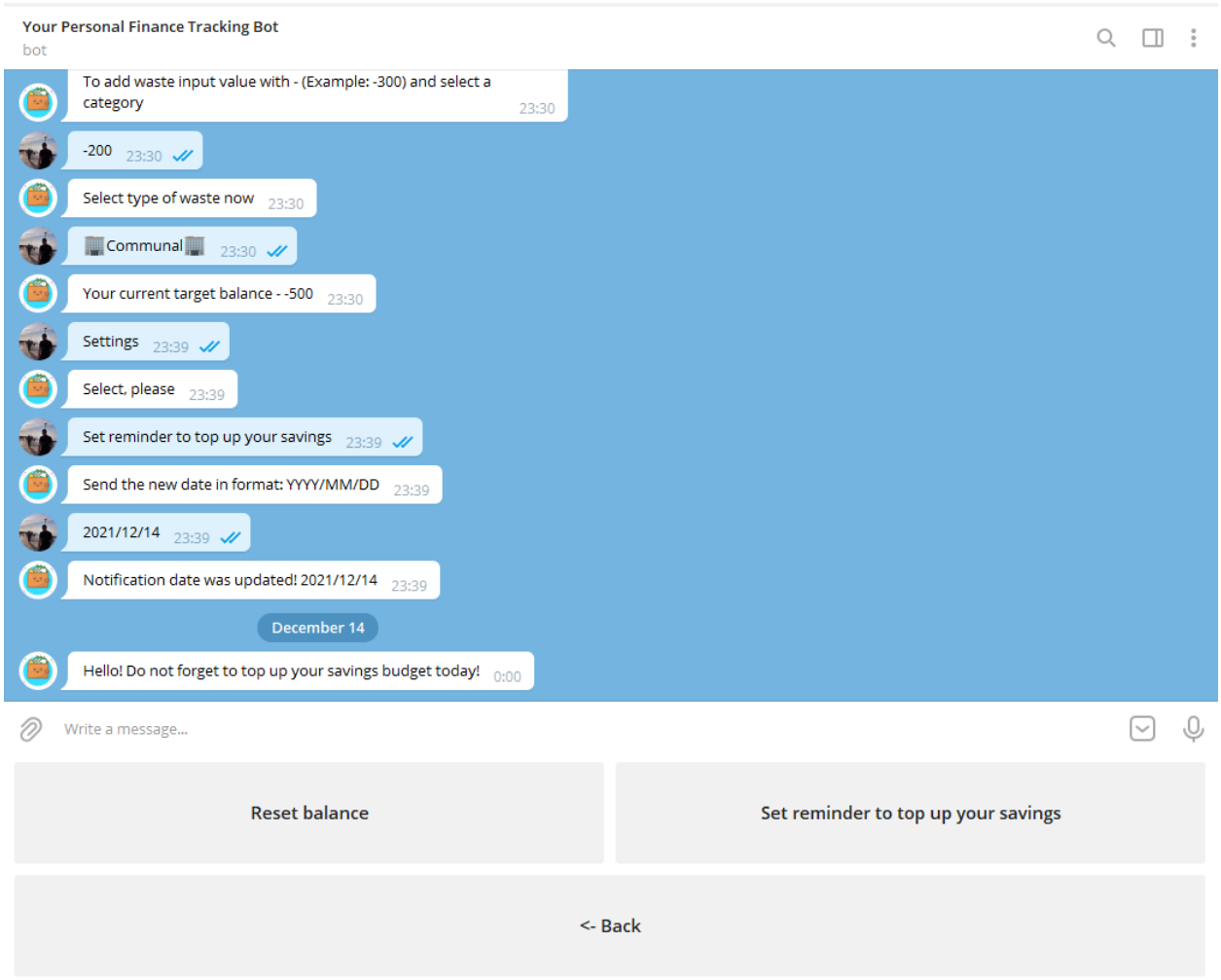
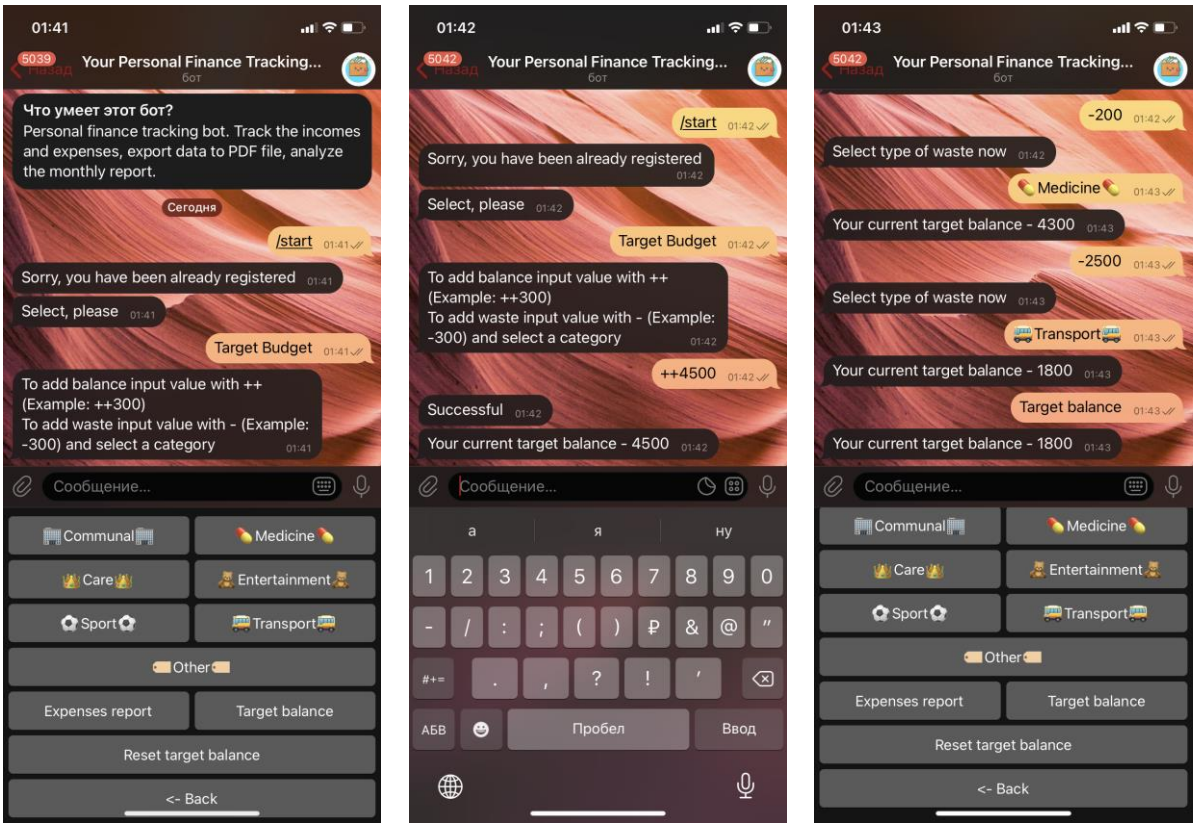


Рисунок 3.17 – Повідомлення-нагадування від бота про необхідність відкласти гроші у бюджет для накопичення

Натиснувши «Target budget», користувач переходить до меню витрат. Бот одразу присилає інформативне повідомлення про порядок дій, які

необхідно виконати, щоб додати або відняти кошти у бюджеті. Одночасно відкривається меню, де перелічено усі можливі категорії для розподілу грошей, а також інші додаткові функції, такі як «Expenses report», «Target balance», «Reset target balance», «Back». Спочатку необхідно задати стартову суму для витрат, далі можна віднімати кошти і обирати категорію, в яку їх необхідно списати. Після цього, бот повідомляє про залишок на балансі. Також, актуальну суму усього бюджету для витрат можна побачити, скориставшись кнопкою «Target balance». Щоб очистити усі записи коштів та видалити бюджет витрат, необхідно натиснути «Reset target balance» (рис. 3.18).



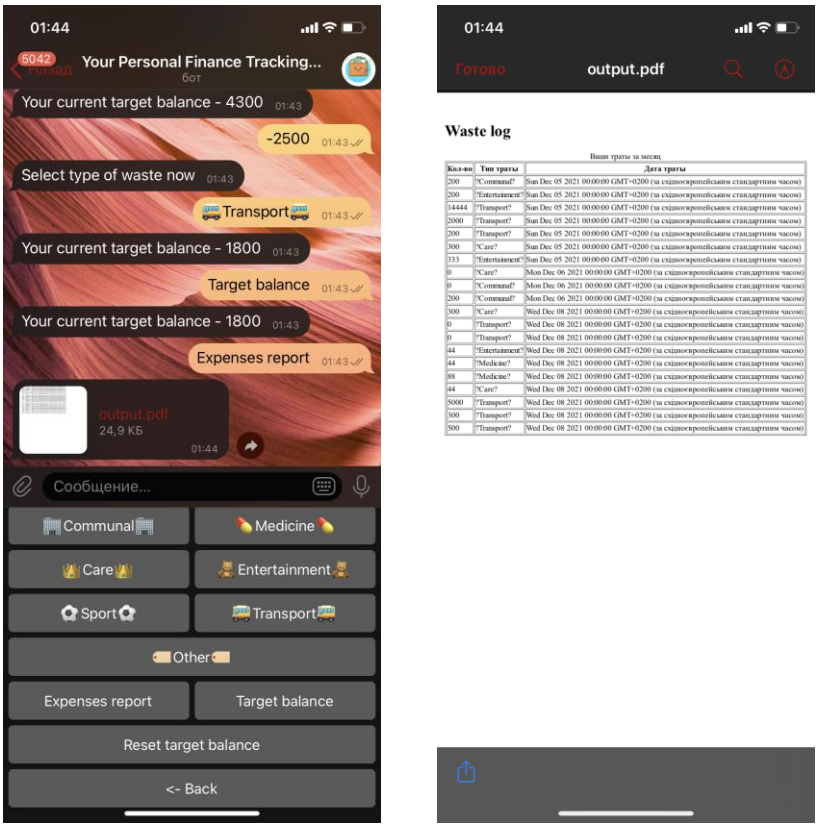
а б в

Рисунок 3.18 – Перехід до меню витрат, відображення доступних категорій (а), поповнення бюджету витрат (б), внесення витрат та розподіл їх по категоріям (в)

Користувач має можливість продивитися звіт по витратам, натиснувши «Expenses report». Усі дані вивантажуються системою автоматично у файл

формату PDF. Записи містяться у таблиці, що показує суму, категорію, дату та час кожної витрати. Користувач може завантажити даний файл або поділитися ним (рис. 3.19).

Також розглянемо результат взаємодії користувача та бота на прикладі запиту звіту по витратам на рівні CLI (інтерфейсу командного рядка) (рис. 3.20).



а б

Рисунок 3.19 – Вивантаження звіту по витратам у файл формату PDF (а), вигляд файлу із записами про усі витрати за місяць (б)

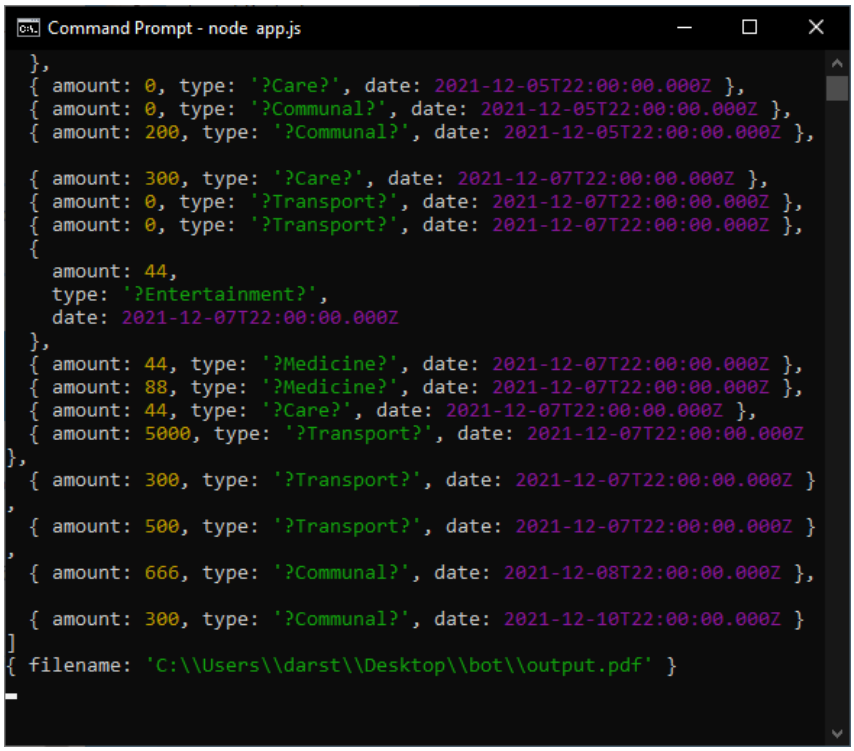


Рисунок 3.20 – Формування звіту по витратам у файл

ВИСНОВКИ

Перший розділ дипломної роботи відведено для аналізу та вибору рішення, а точніше форми розроблюваної системи. Також, було розглянуто існуючі приклади програмних продуктів і визначено переваги та недоліки кожного з них. На основі отриманої інформації було сформовано чітку мету та постановку задачі.

У другому розділі було обрано платформу для розміщення бота, мову програмування, платформу та фреймворки для розробки. Далі було розглянуто різні СУБД та їх структуру й особливості застосування, проаналізовано слабкі та сильні сторони кожної.

Третій розділ присвячений програмній реалізації. Спочатку, було спроєктовано базу даних та змодельовано діаграму, що демонструє структуру розроблюваного програмного продукту. Далі, було коротко описано підготовку необхідних інструментів, налаштування середовища, початок розробки та логіку роботи системи. Крім того, було представлено деякі частини коду, які покривають основний функціонал проєкту. На останньому етапі було проведено тестування готового програмного продукту на мобільному телефоні та на ПК і наведено результати роботи функціонуючого бота.

Результатом даної роботи є розроблена інформаційна технологія планування та контролю бюджету та програмно реалізована у формі телеграм-боту. Бот швидко реагує на команди та миттєво відповідає користувачу, зрозумілий на інтуїтивному рівні, має простий інтерфейс та зручний у використанні. Система надає наступні функції:

- наявність двох бюджетів: перший – для накопичування грошей, другий – для внесення витрат;
- розподіл витрат по категоріям;

— видалення балансу для кожного бюджету окремо;
— можливість налаштувати дату для сповіщення – нагадування про необхідність відкласти кошти;
— збір інформації про щомісячні витрати та вивантаження даних у файл формату PDF.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Richard Klein, “Importance of Keeping Track of Your Expenses”, 2014. [Online]. Available: https://richardkleincpa.com/importance-of-keeping-track-of-your-expenses/
2. Agnieszka Mroczkowska, “What Is a Mobile App? | App Development Basics for Businesses”, 2021. [Online]. Available: https://www.thedroidsonroids.com/blog/what-is-a-mobile-app-app-development-basics-for-businesses
3. Anton Diduh, “Mobile App vs. Mobile Website: What is the Best Option for Your Business?”, 2021. [Online]. Available: https://www.cleveroad.com/blog/mobile-app-vs-mobile-website
4. “What is a Chatbot” – https://www.oracle.com/chatbots/what-is-a-chatbot/
5. М. Акулич, *Чат-боты и маркетинг*. Издательские решения, 2018.
6. Eda Kavlakoglu, “NLP vs. NLU vs. NLG: the differences between three natural language processing concepts”, 2020. [Online]. Available: https://www.ibm.com/blogs/watson/2020/11/nlp-vs-nlu-vs-nlg-the-differences-between-three-natural-language-processing-concepts/
7. Subodh Dharmwan, “Bots vs Apps for business needs – know what to expect”, 2021. [Online]. Available: https://cynoteck.com/blog-post/bots-vs-apps-for-usiness-needs/#Mobile_Apps
8. Prateek Saxena, “A Quick Guide to Pros & Cons of Chatbot Development”, 2020. [Online]. Available: https://appinventiv.com/blog/a-quick-guide-to-pros-and-cons-of-chatbot-development/
9. Валентина Фомина «8 полезных ботов для финансов», 2016. [Электронный ресурс]. Доступно: https://www.sravni.ru/text/8-poleznykh-botov-dlja-finansov

10. Екатерина Надежкина «9 лучших приложений для учета финансов», 2021. [Электронный ресурс]. Доступно: https://daily.afisha.ru/entry/amp/20691/
11. Scott Adam Gordon, Nick Fernandez, “What is Telegram and why should I use it?”, 2021. [Online]. Available: https://www.androidauthority.com/what-is-telegram-messenger-979357/
12. Nicholas C. Zakas, “Introduction” in *The Principles of Object-Oriented JavaScript*, W. Pollock and S. Yang, Eds. San Francisco, CA, USA: No Starch Press, Inc., 2014, pp. 18-19.
13. Jim R. Wilson, “Getting Up to Speed on Node.js 8” in *Node.js 8 the Right Way*, A. Hunt, Eds. Raleigh, North Carolina, USA: The Pragmatic Programmers, LLC, 2018, pp. 3-10.
14. “10 Best Node.js Telegram API Libraries” – https://openbase.com/categories/js/best-nodejs-telegram-api-libraries
15. C. J. Date, *An Introduction to Database Systems*. USA: Pearson Education, Inc., 2004.
16. Б. Шварц, П. Зайцев, В. Ткаченко. «История и архитектура MySQL», в *MySQL по максимуму*, Н. Гринчик и Н. Рощина, Ред. Санкт-Петербург, Россия: Питер, 2018, с.28-65.
17. N. Matthew and R. Stones, “Relational Database Principles” in *Beginning Databases with PostgreSQL*, J. Gilmore, K. Stence Eds. New York, USA: Springer-Verlag New York, Inc., 2005, pp. 17-42.
18. С. Мишра, А. Бьюли. «Введение в SQL», в *Секреты Oracle SQL*, А. Галунов и А. Королев, Ред. Санкт-Петербург, Россия: Символ-Плюс, 2003, с.19-24.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
const Telegraf = require('telegraf');
const mysql = require("mysql2")
const token = '5088365906:AAECKCWijtpTGmnfmM79whG57XYL3u28rCU'
const bot = new Telegraf(token)
const pdf = require("pdf-creator-node")
const fs = require('fs');
const { time } = require('console');

const connection = mysql.createConnection({
    host: "server222.hosting.reg.ru",
    user: "u0613437_anna",
    database: "u0613437_anna",
    password: "Lock2612"
});
const insertNewUser = 'INSERT INTO users(id,username) VALUES (?,?)'
const addNewBudget = 'UPDATE users SET balance = balance + ? WHERE id = ?'
const resetBudget = 'UPDATE users SET balance = 0 WHERE id = ?'
const getBudget = 'SELECT balance FROM users WHERE id = ?'
const getNotification = 'SELECT notifydate From users WHERE id = ?'

const insertNewWaste = 'INSERT INTO waste(uid,amount,type,date) VALUES (?,?,?,?)'

const addNewTarget = 'UPDATE users SET target = target + ? WHERE id = ?'
const resetTarget = 'UPDATE users SET target = 0 WHERE id = ?'
const getTarget = 'SELECT target FROM users WHERE id = ?'
const updateTarget = 'UPDATE users SET target = target - ? WHERE id = ?'

const updateNotificationDate = 'UPDATE users SET notifydate = ? WHERE id = ?'

const getLog = "SELECT amount,type,date FROM waste WHERE uid = ? AND MONTH(date) = ?"

var wasteAmount = '';

bot.start((ctx) =>{
    connection.query(insertNewUser,idAndName(ctx), function(err, results){
        if(err){
            console.log(err)
            ctx.reply("You have been already registered!")
            buildMenu(ctx)
        }
        else{
            console.log("Added new user")
            ctx.reply("Welcome, you were added to the database!")
            buildMenu(ctx)
        }
    })
});
```

```
bot.hears('Savings', (ctx) => {
    connection.query(getBudget,getId(ctx), function(err, results){
        console.log(results)
        console.log(results[0])
        console.log(results['0'])
        bot.telegram.sendMessage(ctx.chat.id,
            "Your current budget is - " + results[0]['balance'] + "\n"+
            "How much do you want to add? Type the value (Example: +3000)"
        )
        buildMainMenuBudget(ctx);
    })
})
bot.hears('Balance', (ctx) =>{
    typeCurrentBalance(ctx)
})
bot.hears('Settings', (ctx) =>{
    buildSettingsMenu(ctx)
})
bot.hears('<- Back', (ctx) => {
    buildMenu(ctx)
})
bot.hears('Target Budget', (ctx) => {
    buildWasteMenu(ctx);
})
bot.hears('Target balance', (ctx) =>{
    typeCurrentTarget(ctx)
})
bot.hears('Reset balance', (ctx) => {
    connection.query(resetBudget, getId(ctx), function(err,
                                                       results){
        if(err){
            console.log(err)
        }
        bot.telegram.sendMessage(ctx.chat.id, "Successful");
        typeCurrentBalance(ctx);
    })
})

bot.hears('Reset target balance', (ctx) => {
    connection.query(resetTarget, getId(ctx), function(err,
                                                       results){
        if(err){
            console.log(err)
        }
        bot.telegram.sendMessage(ctx.chat.id, "Successful");
        typeCurrentTarget(ctx);
    })
})
```

```
bot.hears('🏢Communal🏢', (ctx) => {
    addnewWaste(ctx,  ctx.message.text);
})
bot.hears('💊Medicine💊', (ctx) => {
    addnewWaste(ctx,  ctx.message.text);
})
bot.hears('⚽Sport⚽', (ctx) => {
    addnewWaste(ctx, ctx.message.text);
})
bot.hears('🧸Entertainment🧸', (ctx) => {
    addnewWaste(ctx, ctx.message.text);
})
bot.hears('🚌Transport🚌', (ctx) => {
    addnewWaste(ctx, ctx.message.text);
})
bot.hears('Other', (ctx) => {
    addnewWaste(ctx, ctx.message.text);
})
bot.hears('👑Care👑', (ctx) => {
    addnewWaste(ctx, ctx.message.text);
})

bot.hears('👑Care👑', (ctx) => {
    addnewWaste(ctx, ctx.message.text);
})

bot.hears('Set reminder to top up your savings', (ctx) => {
    ctx.reply("Send the new date in format: YYYY/MM/DD");
})
```

```
bot.hears('Expenses report', (ctx) => {
    generateReport(ctx,'November')
})

const isDate = (date) => {
    return (new Date(date) !== "Invalid Date") && !isNaN(new Date(date));
  }



function addnewWaste(ctx, typeW){
    var requested = new Date();
    connection.query(insertNewWaste,[getId(ctx),wasteAmount,typeW,requested.getFullYear()+
        "/"+(requested.getMonth()+1)+"/"+requested.getDate()], function(err, results) {
        console.log(wasteAmount)
    })
    connection.query(updateTarget,[wasteAmount,ctx.chat.id], function(err, results) {
        typeCurrentTarget(ctx)
    })
}
```

```
bot.on("text", (ctx) => {
    if(ctx.message.text.slice(0,2) === "++"){
        wasteAmount = ctx.message.text.slice(2).toString();
        console.log(wasteAmount)
        connection.query(addNewTarget,[ctx.message.text,getId(ctx)], function(err, results){
            if(err){
                console.log(err)
            }
            bot.telegram.sendMessage(ctx.chat.id, "Successful");
            typeCurrentTarget(ctx);
        })
    }else if(ctx.message.text[0] === "+"){
            connection.query(addNewBudget,[ctx.message.text,getId(ctx)], function(err, results){
                if(err){
                    console.log(err)
                }
                bot.telegram.sendMessage(ctx.chat.id, "Successful");
                typeCurrentBalance(ctx);
            })
        }else{
            if(ctx.message.text[0] ===  '-'){
                wasteAmount = ctx.message.text.slice(1).toString();
                console.log(wasteAmount)
                ctx.reply("Select type of waste now")
            }else{
                if(isDate(ctx.message.text)){
                    var today = new Date();
                    var date = today.getFullYear()+'-'+(today.getMonth()+1)+'-'+today.getDate();
                    var requested = new Date(ctx.message.text);
                    console.log("Current " + today.getFullYear() + " " + (today.getMonth()+1) +
                     " " + today.getDate());
                    console.log("Requested " + requested.getFullYear() +
                     " " + (requested.getMonth()+1) + " " + requested.getDate());
                    if(requested > today){
                        ctx.reply("Notification date was updated! " + ctx.message.text);
                        connection.query(updateNotificationDate,[requested.getFullYear()+
                            "/"+(requested.getMonth()+1)+"/"+requested.getDate(),getId(ctx)],
                             function(err, results){
                        })
                    }
                    checkTimeAndNotify(ctx)
                }
            }
        }
})

bot.launch()


function idAndName(ctx){
    return [ctx.chat.id, ctx.chat.username]
}
function getId(ctx){
    return [ctx.chat.id]
}
```

```
}
function buildMenu(ctx){
    bot.telegram.sendMessage(ctx.chat.id,'Select, please',{
        reply_markup:{
            keyboard:[
                ['Savings','Target Budget'],
                ['Settings']
            ]
        }
    })
}
function buildMainMenuBudget(ctx){
    bot.telegram.sendMessage(ctx.chat.id,'Select, please',{
        reply_markup:{
            keyboard:[
                ['Reset balance','Balance'],
                ['<- Back']
            ]
        }
    })
}
function buildWasteMenu(ctx){
    bot.telegram.sendMessage(ctx.chat.id,'To add balance input value with ++ (Example: ++300)\n'+
    'To add waste input value with - (Example: -300) and select a category\n',{
        reply_markup:{
            keyboard:[
                ['🏢Communal🏢','💊Medicine💊'],
                ['🏰Care🏰','🧸Entertainment🧸'],
                ['⚽Sport⚽','🚌Transport🚌'],
                ['📄Other📄'],

                    ['Expenses report','Target balance'],
                    ['Reset target balance'],
                    ['<- Back']
                ]
            }
        })
}
function buildSettingsMenu(ctx){
    bot.telegram.sendMessage(ctx.chat.id,'Select, please',{
        reply_markup:{
            keyboard:[
                ['Reset balance','Set reminder to top up your savings'],
                ['<- Back']
            ]
        }
    })
}
```

```
function typeCurrentBalance(ctx){
    connection.query(getBudget,getId(ctx), function(err,
                                                    results){
        console.log(results)
        console.log(results[0])
        console.log(results['0'])
        bot.telegram.sendMessage(ctx.chat.id,
            "Your current budget is - " + results[0]['balance']
        )
    })
}
function typeCurrentTarget(ctx){
    connection.query(getTarget,getId(ctx), function(err,
                                                    results){
        console.log(results)
        bot.telegram.sendMessage(ctx.chat.id,
            "Your current target balance - " + results[0]['target']
        )
        console.log(results[0])
    })
}
function generateReport(ctx, month){
    var value = 1
    switch (month){
        case 'January':
            value = 1
            break;
        case 'February':
            value = 2
            break;
         case 'March':
             value = 3
             break;
         case 'April':
             value = 4
             break;
         case 'May':
             value = 5
             break;
         case 'June':
             value = 6
             break;
         case 'July':
             value = 7
             break;
```

```
        case 'August':
            value = 8
            break;
        case 'September':
            value = 9
            break;
        case 'October':
            value = 10
            break;
        case 'November':
            value = 11
            console.log("Accessed")
            break;
        case 'December':
            value = 12
            break;
    }
    var d = new Date();
    var month = d.getMonth()+1;
    connection.query(getLog,[getId(ctx),month], function(err, results){
        console.log(getId(ctx) + " " + month);
        console.log(results);
        createLog(ctx,results)
    })

}


function createLog(ctx,json){
    var html = fs.readFileSync('logs.html', 'utf8')
    var options = { format: "A3", orientation: "portrait", border: "10mm" };
    var document = {
        html: html,
        data: {
            wastes: json
        },
        path: "./output.pdf"
    };
    pdf.create(document, options)
        .then(res => {
            console.log(res)
        })
        .catch(error => {
            console.error(error)
        });
    ctx.telegram.sendDocument(ctx.from.id, {
        source: './output.pdf',
        filename: './output.pdf'
    }).catch(function(error){ console.log(error); })
}
```

```
async function checkTimeAndNotify(ctx){

    var scheduled = new Date();
    connection.query(getNotification,getId(ctx), function(err, results){
        console.log(results[0]['notifydate'])
        scheduled = results[0]['notifydate']
    })

    notif = setInterval(() => {
        currentDateTime = new Date();
        if(
            scheduled.getDate() === currentDateTime.getDate() &&
            (scheduled.getMonth()+1) === (currentDateTime.getMonth()+1) &&
            scheduled.getFullYear() === currentDateTime.getFullYear()){
                ctx.reply('Hello! Do not forget to top up your savings budget today!')
                stopNotification(notif)
        }else{
            console.log("Time = " + scheduled.getDate() + " " + (scheduled.getMonth()+1)
             + " " + scheduled.getFullYear());
            console.log("Current = " + currentDateTime.getDate()
            + " " + (currentDateTime.getMonth()+1) + " " + currentDateTime.getFullYear());
        }
    }, 1800) //43200000 day, 1800 - 30 mins
}

async function stopNotification(notif){
    clearInterval(notif)
    console.log('stopped')
}
```