МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра програмного забезпечення автоматизованих систем
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма «Інженерія програмного забезпечення»

«Допущений до захисту»
Завідувач кафедри
___ ___ проф. Приходько С.Б.
«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти «бакалавр»

на тему: Розробка телеграм-бота "DueMaster" для відстеження дедлайнів з інтеграцією штучного інтелекту

Виконав: студент групи 4151

_____Іванченко Р.В.

(підпис, ПІБ)

Керівник роботи:

д.т.н., професор

(посада, науковий ступень вчене звання)

_____Приходько С.Б.

(підпис, ПІБ)

Миколаїв – 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра програмного забезпечення автоматизованих систем
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма «Інженерія програмного забезпечення»

«ЗАТВЕРДЖУЮ»
Гарант освітньої програми
_____доц. Макарова Л.М.
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття ступеня вищої освіти «бакалавр»

Студенту Іванченко Руслан Володимирович
(Прізвище, ім'я, по батькові)

1. Тема роботи: Розробка телеграм-бота "DueMaster" для відстеження дедлайнів з інтеграцією штучного інтелекту
Керівник роботи Приходько Сергій Борисович
Затверджені наказом ректора №256-уч від «11» травня 2025 р.
2. Термін подання роботи: 10.06.2025 р.
3. Вихідні дані по роботі:

4. Перелік питань, що належать до розробки (найменування розділів)
Титульний аркуш; Завдання на кваліфікаційну роботу; Анотація (українською та англійською мовами); Зміст; Перелік умовних позначень, символів, одиниць та термінів (при необхідності); Вступ; Опис предметної галузі та постановка задачі; Проєкт програмного забезпечення (ескізний, технічний та робочий); Результати розробки програмного забезпечення; Розділ з охорони праці; Висновки; Список використаних джерел; Додатки (технічне завдання, текст програми, опис програми, інструкція користувача, програма та методика випробувань програмного забезпечення).

5. Перелік презентаційних матеріалів Титульний слайд, Мета кваліфікаційної роботи, Актуальність розробки, Аналіз існуючих аналогів, Функціональне призначення програмного забезпечення телеграм-боту "DueMaster", Діаграма варіантів використання, Діаграма діяльності для варіанту використання "Додавання дедлайну", Статична модель програмного забезпечення, Інструменти розробки та технології, Результати розробки, Заключний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
4	Гурець Н.В., ст. викладач		

7. Дата видачі завдання 11.05.2025ц р.

К А Л Е Н Д А Р Н И Й П Л А Н

Номер	Назва етапів роботи	Терміни виконання	Примітка
1.	Підготовка розділу Вступ	24.03.2025	ПП*
2.	Опис та аналіз предметної галузі	28.03.2025	ПП*
3.	Постановка задачі	31.03.2025	ПП*
4.	Ескізний проєкт програмного забезпечення	27.04.2025	
5.	Технічний проєкт програмного забезпечення	06.05.2025	
6.	Робочий проєкт програмного забезпечення	13.05.2025	
7.	Підготовка розділу Результати розробки програмного забезпечення	17.05.2025	
8.	Підготовка розділу з охорони праці	18.05.2025	ПП*
9.	Підготовка розділу Висновки	19.05.2025	
10.	Оформлення списку використаних джерел та додатків	20.05.2025	
11.	Подання на кафедру ПЗАС тексту остаточного варіанту роботи, підписаного її керівником, у роздрукованому та електронному форматі разом із заявами щодо самостійності виконання роботи та ідентичності друкованої та електронної версій роботи (Додатки 1 і 2 «Порядку здійснення заходів з перевірки робіт на наявність текстових збігів/ідентичності/схожості із використанням програмно-технічних засобів», який введений в дію наказом ректора НУК за №20 від 20.01.2020 р.)	10.06.2025	не пізніше, ніж за 14 діб до захисту (згідно п.4.1 зазначеного Порядку)
12.	Підготовка презентації та доповіді	12.06.2025	
13.	Попередній захист роботи на засіданні кафедри ПЗАС	14.06.2025	
14.	Подання на кафедру ПЗАС електронних версій наступних документів у форматі pdf: кваліфікаційної роботи; файлу-опису кваліфікаційної роботи (згідно Додатку до наказу ректора НУК за №287-уч від 19.05.2020 р.); презентації доповіді	24.06.2025	

* - за результатами переддипломної практики (ПП), яка була з 21.03.2022 до 08.05.2022 р.

Студент (підпис) Іванченко Р.В. (ПІБ)

Керівник роботи (підпис) Приходько С. Б. (ПІБ)

АНОТАЦІЯ

Кваліфікаційна робота присвячена розв’язанню практичної задачі інженерії програмного забезпечення, що вирізняється складністю та невизначеністю умов – розробці телеграм-бота «DueMaster» для відстеження дедлайнів з інтеграцією штучного інтелекту.

У роботі проведено аналіз практичної задачі інженерії програмного забезпечення, досліджено існуючі рішення та сформульовано постановку задачі на розробку програмного забезпечення. Розроблено ескізний, технічний та робочий проекти телеграм-бота, представлено результати розробки, а також включено розділ з охорони праці.

Кваліфікаційна робота викладена на 93 сторінках друкованого тексту та містить: 31 рисунок, 3 таблиці, 5 додатків, список використаних джерел з 20 найменувань.

ABSTRACT

The qualification work is dedicated to solving practical problems in software engineering, which is characterized by complexity and uncertainty of conditions - the development of a telegram bot "DueMaster" for tracking deadlines with the integration of artificial intelligence.

The work analyzes a practical problem in software engineering, investigates existing solutions and formulates a problem statement for software development. A draft, technical and working project of the telegram bot is developed, the results of the development are presented, and a section on labor protection is also included.

The qualification work is presented on 93 pages of printed text and contains: 31 figures, 3 tables, 5 appendices, a list of used sources of 20 names.

ЗМІСТ

ВСТУП

Складно уявити життя у двадцять першому столітті, де у студентів та різноманітних фахівців щоденний графік вільний від численних зустрічей та завдань, що необхідно виконати вчасно. Розробка програмного забезпечення для автоматизації відстеження дедлайнів є актуальною практичною задачею інженерії програмного забезпечення, що характеризується комплексністю та невизначеністю умов. Ця галузь стосується проблематики обробки живої мови, планування нагадувань та інтеграції штучного інтелекту задля збільшення продуктивності користувачів.

Джерелом, що досліджує тему автоматизації управління часом, є [1], де розглядаються поточні варіанти створення інтелектуальних систем, зокрема чат-ботів, їхні технологічні складові та факт того, як вони відображаються на ефективності користувачів.

Ринок пропонує чимало готових продуктів для самоорганізації (наприклад, Todoist чи Google Calendar), та вони часто не враховують особисті потреби користувачів або можуть містити зайвий функціонал, що лише спантеличує користувачів [2]. Розробка телеграм-бота "DueMaster" дозволяє сфокусувати увагу на таких функціях як автоматичне розпізнавання дедлайнів із вводу та генерація підбадьорливих порад, що робить його унікальним і практичним рішенням.

Метою даної кваліфікаційної роботи є розробка телеграм-бота "DueMaster" для відстеження дедлайнів з інтеграцією штучного інтелекту. Головна причина актуальності – підвищення попиту на автоматизовані інструменти для керування часом. Форм-фактор телеграм-ботів виділяється на фоні інших варіантів своєю зручністю, доступністю та швидкістю взаємодії на базі популярного месенджеру Telegram, що вже стали невід’ємною частиною нашого повсякдення. Для ефективної роботи "DueMaster" потрібне надійне

програмне забезпечення, яке автоматизує розпізнавання дедлайнів, планування нагадувань і генерацію персоналізованих повідомлень.

Сутність і стан практичної задачі, що вирішується в роботі, характеризується складністю та невизначеністю умов. Створення функціонального телеграм-бота вимагає врахування таких аспектів: обробка масивів неструктурованих даних у текстовому форматі, поєднання з планувальником і забезпечення стабільної роботи в реальному часі. Задля якісної та ефективної розробки програмного забезпечення у цій роботі будуть використані новітні зразки інформаційних технологій, зокрема численні модулі мови програмування Python [3].

Для досягнення мети кваліфікаційної роботи необхідно вирішити наступні завдання:

– Аналіз практичної задачі інженерії програмного забезпечення з теми розробки.

– Аналіз та вибір необхідних технологій та інструментів для розробки телеграм-бота "DueMaster".

– Розробка функціоналу бота, який дозволяє користувачам додавати дедлайни у форматі чату та отримувати нагадування.

– Розробка системи автоматичного розпізнавання дедлайнів із тексту.

– Інтеграція штучного інтелекту для генерації мотиваційних порад.

– Забезпечення стабільної роботи бота в реальному часі через хостинг.

Об’єктом кваліфікаційної роботи є процес розробки телеграм-бота "DueMaster" для відстеження дедлайнів з інтеграцією штучного інтелекту.

.

1 АНАЛІЗ ПРАКТИЧНОЇ ЗАДАЧІ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ НА РОЗРОБКУ ТЕЛЕГРАМ-БОТУ "DUEMASTER" ДЛЯ ВІДСТЕЖЕННЯ ДЕДЛАЙНІВ З ІНТЕГРАЦІЄЮ ШТУЧНОГО ІНТЕЛЕКТУ

1.1 Аналіз практичної задачі інженерії програмного забезпечення

Розробка програмного забезпечення є головною частиною поточних новітніх технологій та надає нам можливість автоматизувати рутину бути більш продуктивними як у професійній сфері, так і побуті. Телеграм-бот "DueMaster" – це корисний інструмент, який допомагає користувачам, зокрема студентам, різноманітним фахівцям та професіоналам, ефективніше організовувати робочі процеси та повсякденне життя, отримуючи нагадування та заохочення через мотиваційні поради.

Основна мета функціонування "DueMaster" базується на наданні користувачу зручного способу організації дедлайнів, використовуючи для цього текстові повідомлень, автоматизацію планування нагадувань та максимізацію рівня мотивації через персоналізовані поради, які будуть генеруватися завдяки штучному інтелекту. Це програмне забезпечення розробляється, аби зробити процес управління часом максимально простим та необтяжливим. Як приклад, студент може надіслати повідомлення "Здати лабораторну роботу до 10 червня 14:00", в свою чергу, телеграм-бот автоматично, розпізнавши дату, зберігає завдання у базі даних та встановлює нагадування за день до кінцевого терміну виконання. Фахівець може написати "Здати звіт до п’ятниці", та телеграм-бот автоматичним чином врахує поточну дату та вирахує найближчу п’ятницю, та окрім цього згенерує для користувача мотиваційне гасло, на кшталт "Розпочни зараз, аби потім не робити все в останню ніч!". Серед клієнтів бота можуть бути як й індивідуальні користувачі, так і невеликі колективи, що мають груповий чат в Telegram та потребують синхронізації робочого процесу. Подібний підхід

робить бота гнучким для використання у різних варіаціях: від особистих потреб до роботи у команді.

Під час роботи телеграм-боту виникають такі задачі:

– Забезпечення точного і гнучкого розпізнавання дедлайнів із масивів неструктурованих даних у текстовому форматі (наприклад, "завтра", "через 3 дні" чи "15 листопада 14:00").

– Забезпечення ефективного планування та своєчасного надсилання нагадувань у реальному часі.

– Генерація корисних мотиваційних порад за допомогою штучного інтелекту.

– Забезпечення зручного та інтуїтивно зрозумілого інтерфейсу на базі платформи Telegram.

– Забезпечення надійного та безпечного зберігання користувацьких даних у базі

Задля розв’язання поставлених задач у цій роботі будуть використані новітні зразки інформаційних технологій, зокрема численні модулі мови программування Python [3], методи обробки текстових даних (NLP) [4] та ефективні способи планування. Безпека даних забезпечується локальним зберіганням в базі SQLite [5] із можливістю шифрування в майбутньому.

Такі заходи, як захищені протоколи трансферу даних, можливість шифрування даних та двофакторна автенифікація можуть гарантувати безпеку користувацьких даних. Окрім цього, не менш важливою є наявність політики конфіденційності та захисту персональних даних користувачів.

1.2 Аналіз існуючих інструментів для розробки телеграм-ботів

Репозитарій Python пропонує декілька бібліотек та методологій створення ботів на базі Telegram.

python-telegram-bot – це розповсюджена серед розробників бібліотека для створення ботів на базі Telegram, використовуючи мову Python. Ця бібліотека використовує Telegram Bot API, пропонує простий інтерфейс для обробки повідомлень, команд і кнопок. Підтримуються базові функції, такі як: відповідь на текстові повідомлення, відправлення вкладень і інтеграцію inline-клавіатур. Наприклад, кілька рядків коду дозволяють нам реалізувати команду “/start”, що робить цей варіант оптимальним за потреби зекономити час при розробці. Документація доступна за [6] і описує приклади для типових сценаріїв: обробка тексту, надсилання повідомлень, інтеграція inline-клавіатур тощо.

Основні переваги python-telegram-bot

– Бібліотека надає простий і зрозумілий API, що не потребує багато часу на опанування для розробка базових ботів навіть розробникам, що не володіють великим досвідом. Це робить бібліотеку зручною у використанні та швидкою в опануванні. Наприклад, реалізація відповіді на користувацьку команд не займе більше кількох хвилин.

– Поширеність бібліотеки зумовлює чимале різноманіття форумів, групових чатів та матеріалів на таких порталах, як Github, завдяки чому стає можливим отримати вже готове рішення або відповідь на своє питання.

– Активна спільнота та детально прописана документація надають чимало реалізацій необхідного функціоналу, що, в свою чергу, допомагає швидше та більш якість освоїти інструмент та адаптувати його під певні задачі.

– Відмінно взаємодіє з іншими бібліотеками (sqlite для роботи з базами даних, requests для інтеграції API).

– Завдяки проактивній спільноті, на GitHub своєчасно оновлюється бібліотека, де невпинно зростає кількість прикладів та готових рішень, що спрощує та пришвидшує розробку.

– Розробники забезпечують своєчасні оновлення бібліотеки згідно з оновленням Telegram API.

Основні недоліки python-telegram-bot:

– За замовчуванням, бібліотека функціонує в синхронному режимі і не підтримує асинхроність. Це може викликати затримку під час обробки великої кількості користувацьких запитів одночасно. Це є критичним для боту, основною задачею якого є система нагадувань.

– Синхронний підхід обмежує швидкість опрацювання запитів, знижуючи продуктивність при навантаження, що не є оптимальним для ПЗ, що потребує високої продуктивності під час реакції на події.

– Для повноцінної реалізації функцій, що включає в себе бот з системою нагадувань, необхідне підключення додаткових модулів. Подібні рішення можуть ускладнити структуру проєкта та знизити його надійність. Одним з таких модулів є schedule.

– Необхідні додаткові вдосконалення значного характеру для підтрмки непростих сценаріїв: таких, як стани користувача. Це може провокувати проблеми для ботів інтерактивного характеру.

– Сторонні бібліотеки, що надають додатковий функціонал, можуть зумовлювати конфлікти пов’язані з несумісністю.

aiogram – це асинхронна бібліотека для роботи з Telegram API на Python, зосереджена на asyncio. Ця бібліотека, зазвичай, використовується для ботів з високим навантаженням у вигляді одночасної обробки великої кількості запитів. Підтримуються асинхронні функції завдяки механізмам async та await, що вирішує проблему зниження продуктивності під час надсилань нагадувань великій кількості користувачів, уникаючи затримки. Для керування станами користувачів, що використовується під час складних діалогів, одним з яких є уточнення дедлайну після користувацького вводу, бібліотека пропонує FSM (Finite State Machine). Документація за [7] містить приклади асинхронної обробки та інтеграції з базами даних.

Основні переваги aiogram:

– Використовуючи асинхронність, бібліотека спроможна ефективно опрацювувати тисячі користувацьких запитів паралельно. Таким чином, ця

бібліотека є етанолонним рішенням для ботів, що містять системи, які працюють у реальному часі, наприклад, нагадування.

– Миттєва реакція на такі події, як оперативне надсилання нагадування про прийдешні дедлайни, реалізується завдяки асинхронному програмуванню.

– FSM (Finite State Machine) дозволяє без труднощів налаштувати складні сценарії інтеракції: створити та надіслати запит уточнення вводу у користувача. Наприклад: “Ви мали на увазі 17 листопада поточного року?”.

– Ідеально інтегрується з різноманітними бібліотеками: NLP (spacy) [8], AI (transformers) та apscheduler [9] для планування.

– Завдяки проактивному товариству розробників та користувачів, GitHub щоденно поповнюється новими прикладами, оновленнями бібліотеки та вирішеннями проблем.

– Є бездоганним рішенням, якщо проєкт має перспективу масштабування: розширення функціоналу від простого боту до складно багаторівневої системи, що нараховує десятки або навіть сотні функцій.

Основні недоліки aiogram:

– Програмування асинхронних ботів базований на концепціях asyncio, тому для опанування та розуміння подібних рішень необхідно більше зусиль та часу. Більший поріг входження у порівнянні з простішими бібліотеками часто є критичним фактором для початків.

– Асинхронний характер бібліотеки має наслідком більш масивні конструкції коду для реалізації базових функцій (обробка команд, надсилання запитів).

– Для повноцінної реалізації функцій, що включає в себе бот з системою нагадувань, вбудованого інструментарію може не бути достатньо, внаслідок чого необхідне підключення додаткових модулів. Подібні рішення можуть ускладнити структуру проєкта та знизити його надійність.

– async та await потребують певної обережності у використанні, позаяк, у разі некоректного застосування можуть виникати критичні збої, наприклад, зависання задач.

– Вищезгадана бібліотека під назвою python-telegram-bot налічує більше прикладів, зокрема документація, яка є корисною для новачків.

Telebot – це мінімалістична бібліотека для швидкої реалізації ботів на базі Telegram. Ця бібліотека є спрощеним варіацією python-telegram-bot і використовується для нескладних проєктів. Простота цієї бібліотеки надає розробнику можливість за короткий проміжок часу та мінімальних зусиль швидко реалізувати у боті відповіді на запити чи повідомлення від користувача. Наприклад, відправити “Привіт” у відповідь на “/start”. Головним обмеженням для використання цієї бібліотеки для ботів, задачею яких є підтримка функцій, що використовуються у реальному часі, у нашому випадку система нагадувань, є відсутність підтримки асинхронності. Це може бути проблемою під час паралельної обробки великої кількості запитів. Telebot є загальноприйнятою, як бібліотека для новачків, тому що потребує мінімального часу для опанування, втім її функціоналу недостатньо, якщо у проєкта з’являється перспектива масштабування. Документація доступна за [10] і описує приклади для типових сценаріїв: обробка тексту, надсилання повідомлень, інтеграція inline-клавіатур тощо.

Основні переваги Telebot:

– Завдяки простому та зрозумілому без додаткових пояснень синтаксису Telebot потребує лише декілька хвилин для розгортання базових ботів. Це є найбільш зручним варіантом для створення макетів та навчальних проєктів.

– Лише близько 10 рядків коду достатньо, аби налаштувати конфігурацію бота однією командою. Це гарантує швидкий старт під час першої фази розробки.

– Новачки без зайвих зусиль можуть знайти на різноманітних форумах та порталах безліч матеріалів та прикладів для навчання та освоєння бібліотеки.

– Внаслідок відсутності додаткового багатокрокового конфігурування чи налаштування додаткових залежностей для реалізації базових механізмів, бібліотека висуває мінімальну кількість вимог для початку роботи.

– Такі модулі як time, random та інші, з легкістю інтегруються у боти, що написані на Telebot.

Основні недоліки Telebot:

– Високопродуктивна обробка великої кількості запитів, що надіслані одночасно, є неможливою через відсутність асинхронності.

– Стани користувача, що є прикладом складного сценарію, та інтеграція зі штучним інтелектом є неможливою через обмежений вбудований функціонал.

– Ефективність обробки запитів значно знижується при зростанні користувацької бази або при збільшенні функціоналу боту внаслідок відсутності оптимізації. Це має певні ризики, оскільки не дозволяє масштабувати проєкт.

– Для реалізації системи нагадувань, з огляду на те, що бібліотека не має вбудованого планування, необхідно буде використовувати сторонні бібліотеки. Це знижує ясність коду та його стабільність.

– Такі бібліотеки, як aiogram та python-telegram-bot, мають ширші спільноти, ніж Telebot. Це робить пошук відповідей на питання, що виникають, не таким простим.

Було прийнято рішення для розробки телеграм-боту “DueMaster” обрати бібліотеку aiogram, позаяк саме вона забезпечує усі необхідні аспекти для повноцінної реалізаії поставленої задачі: високу ефективність при обробці великої кількості запитів паралельно, підтримку асинхронного програмування, що є дуже значимим для ПЗ, що працює з функціями реального часу, зокрема відправки нагадувань. Для розпізнавання та обробки дат було обрано бібліотеку Pendulum [11], яка забезпечує зручну роботу з часовими зонами та форматами часу. Для типізації даних використовується модуль typing [12], що дозволяє підвищити надійність коду та полегшити масштабування проєкту. Для генерації мотиваційних повідомлень на основі штучного інтелекту бот інтегрується з платформою OpenRouter [13], через яку здійснюється доступ до безкоштовної великої мовної моделі DeepSeek, що дозволяє створювати адаптивний, емоційно підтримуючий контент.

1.3 Постановка задачі на розробку телеграм-бота “DueMaster” для відстеження дедлайнів з інтеграцією штучного інтелекту

Згідно з проведеним аналізом існуючих підходів до розробки програмного забезпечення для створення телеграм-ботів, сформулюємо постанову задачі: потрібно розробити телеграм-бот "DueMaster", що забезпечить зручне додавання дедлайнів, автоматичне надсилання нагадувань і генерацію мотиваційних порад за допомогою штучного інтелекту.

Експлуатаційним призначенням програмного забезпечення, що розробляється, є створення функціонального та ефективного телеграм-бота, щоб поліпшити продуктивність та швидкість роботи користувачів, надійність і точність обробки дедлайнів, надання зручного та інтуїтивного інтерфейсу в межах месенджера Telegram.

Функціональним призначенням програмного забезпечення, що розробляється, є автоматизація управління дедлайнами, включаючи розпізнавання завдань із тексту, планування нагадувань і генерацію персоналізованих порад.

Були сформульовані наступні функціональні вимоги до програмного забезпечення:

Для користувача:

– Реєстрація користувача не потрібна, але бот має ідентифікувати його за Telegram ID для збереження персональних дедлайнів.

– Додавання дедлайнів через текстові повідомлення (наприклад, "Завершити проєкт до 20 травня 15:00").

– Перегляд списку дедлайнів за командою "/list" із сортуванням за датою.

– Отримання нагадувань за заданий час до дедлайну

– Отримання мотиваційних порад, згенерованих штучним інтелектом, після додавання дедлайну чи за командою "/motivate".

Для розробника:

– Збереження дедлайнів у базі даних із можливістю їх редагування чи видалення.

– Налаштування частоти перевірки дедлайнів.

– Інтеграція моделі штучного інтелекту для генерації тексту з можливістю оновлення моделі в майбутньому.

– Логування дій бота для відлагодження та аналізу помилок.

– Можливість масштабування функцій, наприклад, додавання групового режиму для командної роботи.

Вимоги до організаційних вхідних та вихідних даних:

Вхідні дані:

– Текстові повідомлення з дедлайнами: наприклад, "Здати курсову до п’ятниці" або "Зустріч 10 червня о 14:00".

– Команди: "/list" (перегляд дедлайнів), "/add" (додавання дедлайну), "/delete" (видалення дедлайну), "/motivate" (отримання поради).

– Налаштування користувача: вибір часу нагадувань.

Вихідні дані:

– Підтвердження додавання дедлайну: "Дедлайн ‘Здати курсову’ збережено на п’ятницю, 15 травня 2025, 23:59".

– Список дедлайнів: "1. Здати курсову – 15.05.2025 23:59; 2. Зустріч – 10.06.2025 14:00".

– Нагадування: "Нагадування: завтра о 14:00 дедлайн для ‘Зустріч’!".

– Мотиваційні поради: "Ти на правильному шляху, головне –почати вчасно!".

Мінімальна комплектація смартфону:

– Процесор: AMD A10-5750m 2.5 GHZ 4 ядра.

– Оперативна пам’ять: 4 гб.

– Жорсткий диск: 2 гб.

Мінімальна комплектація смартфону:

– Версія Android 8.0 / iOS 12.4.9.

– ОЗУ 2 ГБ.
– 500 МБ вільної пам’яті.

Вимоги до інформаційної та програмної сумісності:

Запуск та роботу програмного забезпечення необхідно виконувати на операційній системі Windows 7/8/10 (32 або 64 біти), с допомогою програми XAMPP, що дозволяє створювати і розгортати веб-сайти та веб-додатки на локальному комп’ютері.

Дотримуючись цих кроків постановки задачі, потрібно розробити програмне забезпечення телеграм-бота "DueMaster", спрямоване на створення зручної та ефективної платформи для управління дедлайнами, щоб задовольнити потреби користувачів у швидкому доступі до функцій планування та мотивації через платформу Telegram.

2 ПРОЕКТ ТЕЛЕГРАМ-БОТУ “DUEMASTER” ДЛЯ ВІДСТЕЖЕННЯ ДЕДЛАЙНІВ З ІНТЕГРАЦІЄЮ ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Аналіз вимог до програмного забезпечення телеграм-боту “DueMaster” для відстеження дедлайнів з інтеграцією штучного інтелекту

2.1.1 Побудова моделі варіантів використання

Діаграма варіантів використання визначає доступний функціонал програмної системи для різних груп користувачів, забезпечуючи чітке уявлення про взаємодію з телеграм-ботом "DueMaster" [15]. Вона відображає основні дії, які користувачі можуть виконувати, та допомагає структурувати вимоги до системи.

На основі аналізу функціональних вимог до телеграм-бота "DueMaster" було виділено такі основні варіанти використання:

– Додавання дедлайну через текстове повідомлення (наприклад, "Завершити завдання до 20 травня 15:00").

– Перегляд списку дедлайнів за допомогою команди /list

– Отримання нагадувань про дедлайни за заданий час до їх настання.

– Отримання мотиваційних порад, згенерованих штучним інтелектом, після додавання дедлайну або за командою /motivate.

– Видалення дедлайнів за допомогою команди /delete.

– Редагування дедлайну за командою /edit.

– Налаштування часу нагадувань за допомогою команди /reminder.

Рисунок 2.1 ілюструє діаграму варіантів використання телеграм-боту “DueMaster”.

Рисунок 2.1 – Діаграма варіантів використання телеграм-боту “DueMaster”

Отже, було створено діаграму використання для подальшого проектування телеграм-бота "DueMaster".

2.1.2 Розробка специфікації варіантів використання

На основі аналізу діаграми варіантів використання для телеграм-бота "DueMaster" було здійснено детальну специфікацію кожного варіанту використання. Нижче представлено реєстр варіантів використання, приведений у таблиці 2.1, та їх детальний опис.

Таблиця 2.1 Реєстр варіантів використання

Код	Основна дійова особа (Актор)	Назва варіанту використання	Стислий опис
1	2	3	4
A1	Користувач	Додати дедлайн	Дозволяє користувачу додати дедлайн через текстове повідомлення.
A2	Користувач	Переглянути список дедлайнів	Надає можливість переглянути список усіх збережених дедлайнів із сортуванням за датою.

Продовження таблиці 2.1

1	2	3	4
A3	Користувач	Отримати нагадування	Забезпечує автоматичне надсилання нагадувань про наближення дедлайнів.
A4	Користувач	Отримати мотиваційну пораду	Дозволяє отримати персоналізовану мотиваційну пораду від штучного інтелекту.
A5	Користувач	Редагувати створений дедлайн	Дозволяє змінити створений дедлайн.
A6	Користувач	Видалити дедлайн	Дозволяє видалити існуючий дедлайн.
A7	Користувач	Налаштувати час нагадувань	Дозволяє користувачу встановити бажаний час для нагадувань.

A1. Додати дедлайн

Короткий опис: Дозволяє користувачу додати новий дедлайн через текстове повідомлення (наприклад, "/add Завершити звіт 15 червня 14:00").

Основна дійова особа: Користувач.

Інші учасники: Відсутні.

Передумови: Користувач активував бота та готовий ввести дедлайн.

Основний потік подій:

1. Користувач запускає бота.

2. Бот надсилає вітальне повідомлення з пропозицією ввести дедлайн, показуючи основні команди.

3. Користувач надсилає текстове повідомлення з дедлайном.

Альтернативний потік подій: При некоректному форматі дати бот просить уточнити введені дані.

Постумови: Бот підтверджує збереження дедлайну та показує його деталі.

A2. Переглянути список дедлайнів

Короткий опис: Надає можливість переглянути список усіх збережених

дедлайнів.

Основна дійова особа: Користувач.

Інші учасники: Відсутні.

Передумови: Користувач бажає переглянути усі свої активні дедлайни.

Основний потік подій:

1. Користувач запускає бота.

2. Бот надсилає вітальне повідомлення з пропозицією ввести дедлайн, показуючи основні команди.

3. Користувач вводить команду /list.

Альтернативний потік подій: У разі якщо дедлайнів немає, бот повідомляє про їх відсутність.

Постумови: Бот виводить список усіх дедлайнів із датами та часом.

А3. Отримати нагадування

Короткий опис: Забезпечує автоматичне надсилання нагадувань про наближення дедлайнів за заданим часом.

Основна дійова особа: Користувач.

Інші учасники: Відсутні.

Передумови: Є збережені дедлайни з налаштованими нагадуваннями.

Основний потік подій:

1. Бот перевіряє базу даних дедлайнів.

2. За заданий час до дедлайну бот надсилає нагадування.

Альтернативний потік подій: Відсутній.

Постумови: Користувач отримує повідомлення з нагадуванням про дедлайн.

А4. Отримати мотиваційне повідомлення.

Короткий опис: Дозволяє отримати персоналізовану мотиваційну пораду від штучного інтелекту.

Основна дійова особа: Користувач.

Інші учасники: Відсутні.

Передумови: Користувач бажає отримати мотивацію.

Основний потік подій:

1. Користувач запускає бота.

2. Бот надсилає вітальне повідомлення з пропозицією ввести дедлайн, показуючи основні команди.

3. Користувач вводить команду /motivate або додає новий дедлайн.

Альтернативний потік подій: Відсутній.

Постумови: Бот надсилає мотиваційну пораду згенеровану штучним інтелектом.

А5. Редагувати створений дедлайн.

Короткий опис: Дозволяє відредагувати створений дедлайн.

Основна дійова особа: Користувач.

Інші учасники: Відсутні.

Передумови: Користувач має доданий дедлайн, який потребує редагування.

Основний потік подій:

1. Користувач запускає бота.

2. Бот надсилає вітальне повідомлення з пропозицією ввести дедлайн, показуючи основні команди.

3. Користувач вводить команду /edit і обирає дедлайн для редагування.

Альтернативний потік подій: Якщо дедлайн не знайдено, бот повідомляє про помилку.

Постумови: Бот оновлює дані дедлайну та підтверджує зміни.

А6. Видалити дедлайн.

Короткий опис: Дозволяє видалити раніше введений дедлайн.

Основна дійова особа: Користувач.

Інші учасники: Відсутні.

Передумови: Користувач бажає видалити певний дедлайн.

Основний потік подій:

1. Користувач запускає бота.

2. Бот надсилає вітальне повідомлення з пропозицією ввести дедлайн, показуючи основні команди.

3. Користувач вводить команду /delete і обирає дедлайн для видалення.

Альтернативний потік подій: Якщо дедлайн не існує, бот сповіщає про це.

Постумови: Бот підтверджує видалення дедлайну.

A7. Налаштувати час нагадувань

Короткий опис: Дозволяє користувачу встановити бажаний час для отримання нагадувань.

Основна дійова особа: Користувач.

Інші учасники: Відсутні.

Передумови: Користувач хоче змінити час нагадувань.

Основний потік подій:

1. Користувач запускає бота.

2. Бот надсилає вітальне повідомлення з пропозицією ввести дедлайн, показуючи основні команди.

3. Користувач вводить команду /reminder і вказує новий час нагадувань.

Альтернативний потік подій: При некоректному форматі часу бот просить зробити коректне введення.

Постумови: Бот зберігає нові налаштування часу нагадувань

Ця специфікація забезпечує чітке розуміння функціонування телеграм-бота “DueMaster” та слугує основою для подальшої розробки та впровадження програмного забезпечення.

2.2 Архітектура програмного забезпечення телеграм-боту “DueMaster” для відстеження дедлайнів з інтеграцією штучного інтелекту

Розробка архітектури телеграм-боту “DueMaster” відіграє центральну роль у його створенні, визначаючи організацію компонентів, їхню взаємодію та принципи функціонування. Щоб забезпечити надійне відстеження дедлайнів і впровадження штучного інтелекту, система потребує ретельного планування, яке гарантує як її ефективність, так і можливість розширення. Одним із ключових засобів для зображення цієї архітектури є діаграма компонентів – вид діаграм класів, що зосереджується на структурних частинах системи. Такі діаграми активно використовуються для ілюстрації статичної реалізації програмного забезпечення, показуючи зв’язки між модулями та їхнє фізичне розміщення [19].

Діаграма компонентів телеграм-боту “DueMaster”, представлена на рисунку 2.2, показує основні елементи системи та їхні взаємозв’язки. Вона відображає модульну структуру бота, де кожен компонент має свою окрему роль. Зокрема, є частина, яка обробляє команди від користувачів і інтерпретує їх, а також елемент, що забезпечує зв’язок із API Telegram для інтеграції з платформою месенджера.

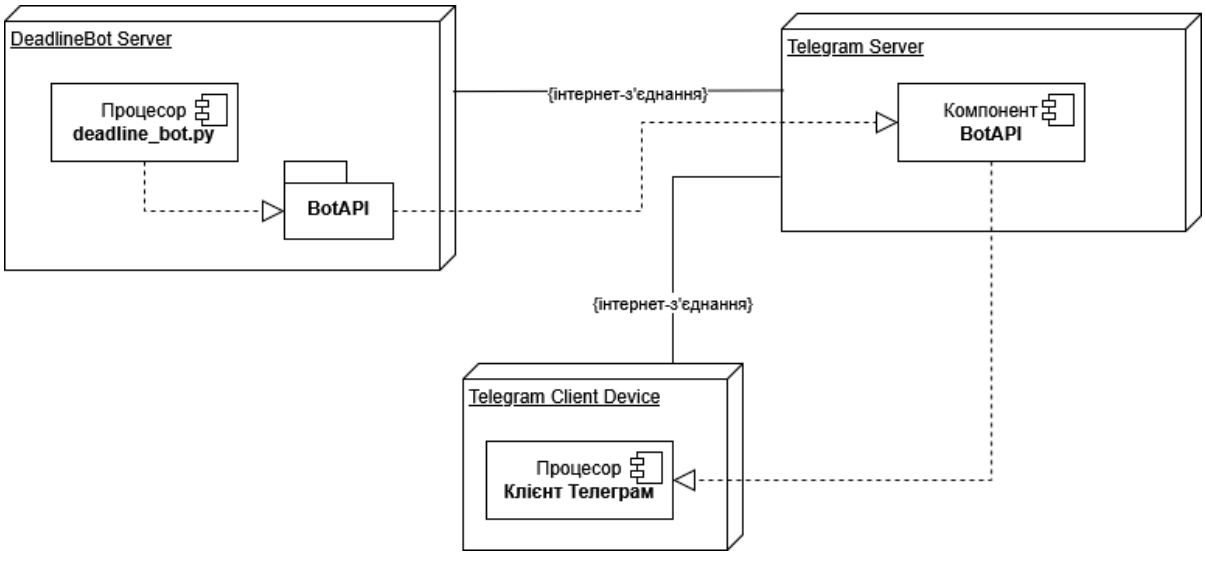

Рисунок 2.2 – Діаграма компонентів телеграм-боту “DueMaster”

Таким чином, була створена діаграма компонентів, яка ілюструє взаємодію між окремими елементами системи, розкриває особливості їх фізичної реалізації та допомагає визначити архітектуру програмного забезпечення. Цей інструмент став важливим етапом у процесі проектування, оскільки дозволяє чітко зрозуміти, як усі частини системи працюють разом і як вони впливають одна на одну. Діаграма допомагає візуалізувати складні взаємозв'язки між компонентами, що полегшує їх структурування та аналіз. Це дозволяє не тільки оцінити поточний стан архітектури, але й передбачити, як система може розвиватися в майбутньому, додаючи нові функції або вдосконалюючи існуючі. Крім того, вона відображає фізичне розташування елементів, наприклад, розподіл між сервером і клієнтським пристроєм, що є ключем до забезпечення стабільної роботи та ефективного обміну даними. Таким чином, діаграма служить надійною основою для подальшої розробки, тестування та оптимізації програмного забезпечення, сприяючи його адаптивності та зручності використання.

2.3 Проєкт програмного забезпечення телеграм-боту “DueMaster” для відстеження дедлайнів з інтеграцією штучного інтелекту

2.3.1 Ескізний проект програмного забезпечення

Сценарії варіантів використання

На цьому етапі розробки програмного забезпечення створюється діаграма діяльності телеграм-бота "DueMaster".

Діаграма діяльності демонструє динамічні аспекти поведінки системи та надає детальну блок-схему, яка чітко показує послідовність переходів потоку керування між окремими діями, такими як обробка команд користувача, взаємодія з базою даних та виконання фонових завдань.

Цей інструмент дозволяє спостерігати за логікою роботи бота, включаючи обробку таких запитів, як /add, /list, /edit, /delete, /motivate та /remind, а також реалізацію циклічної перевірки нагадувань. Розробка такої діаграми необхідна для аналізу базового та основного варіанту використання, а саме: надання користувачеві доступу до відповідної інформації про дедлайни та своєчасне отримання нагадувань, що детально представлено на рисунку 2.3.

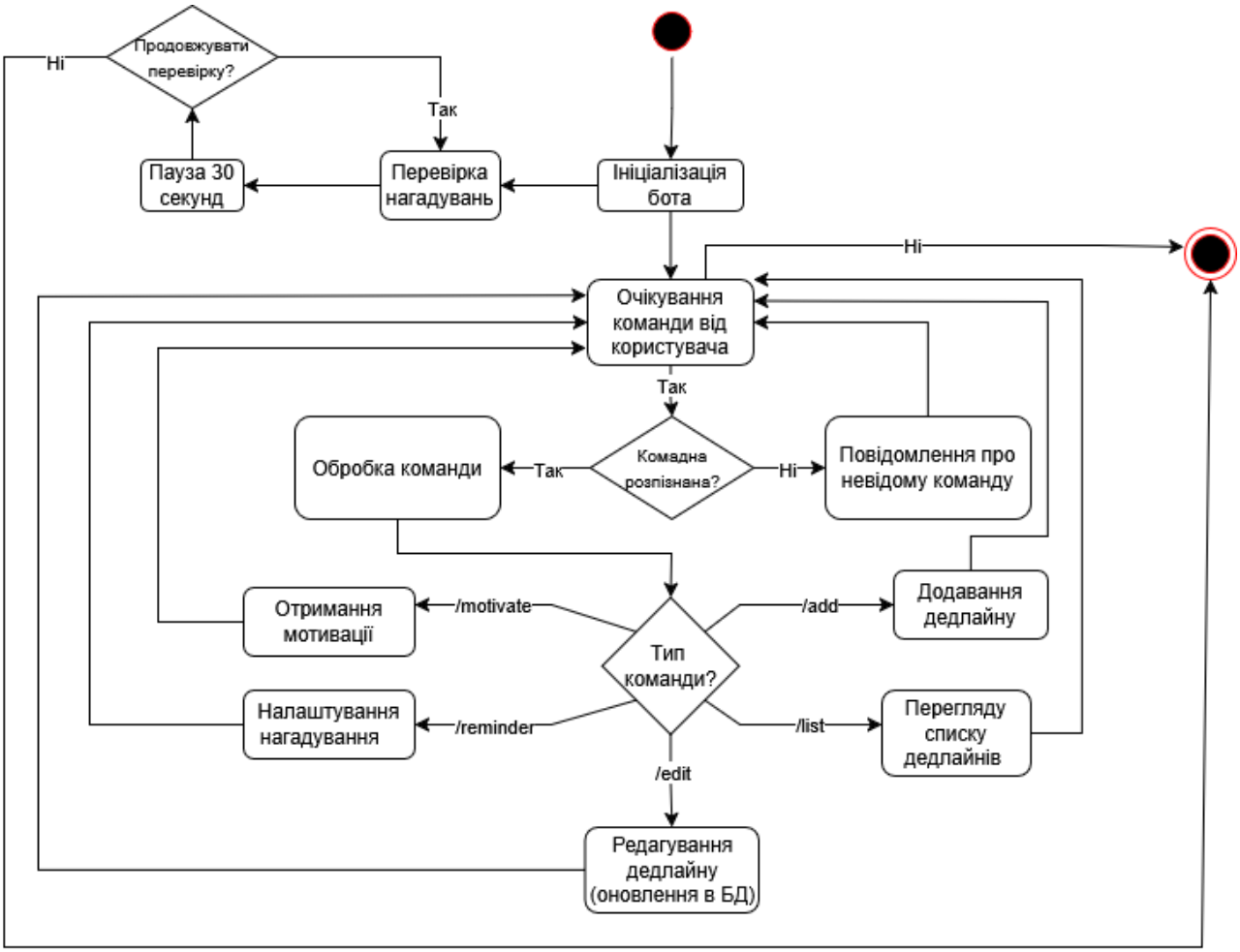

Рисунок 2.3 – Діаграма діяльності телеграм-боту “DueMaster”

Цей етап відіграє ключову роль у подальшому проектуванні, оскільки допомагає виявити можливі недоліки в логіці системи та формує основу для її оптимізації. Крім того, схема робіт слугує важливим елементом у підготовці до фаз перевірки, тестування та впровадження архітектури бота, а також

забезпечує належну інтеграцію всіх програмних компонентів в єдину функціональну систему.

Інформаційна модель даних

Діаграма зв’язків сутностей (ERD) є графічним зображенням різних об’єктів у системі та їхніх взаємодій між собою [16].

Інформаційна модель телеграм-боту “DueMaster” ілюструється за допомогою ER-діаграми, представленої на рисунку 2.4.

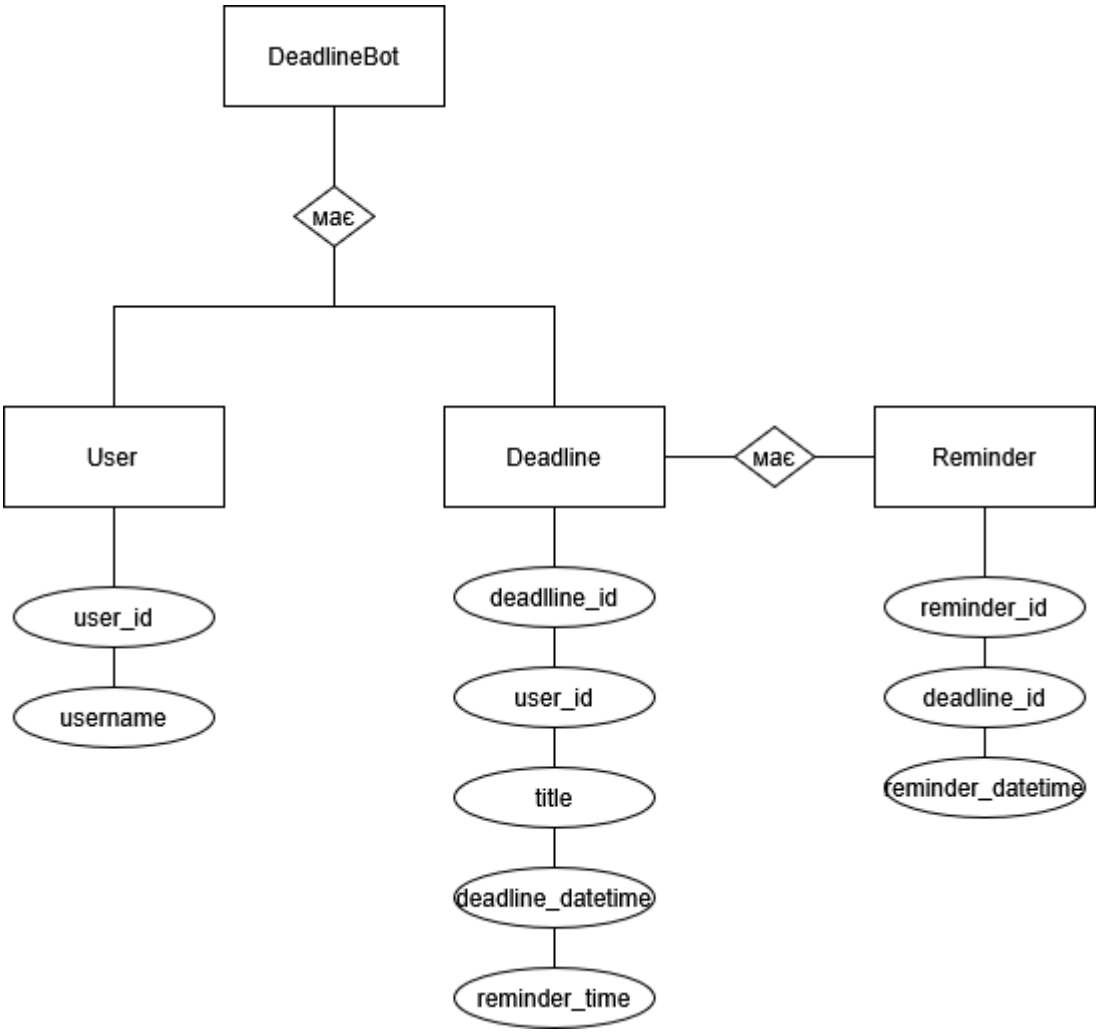

Рисунок 2.4 – ER діаграма телеграм-боту “DueMaster”

Таким чином, для створюваного програмного забезпечення була розроблена діаграма сутностей-зв'язків (ERD), яка дозволяє візуально представити структуру даних та взаємодії між основними об'єктами системи.

Розробка інтерфейсу телеграм-боту “DueMaster”

Інтерфейс – це спеціальний механізм або інструмент, що складається з набору об’єктів, таких як піктограми, кнопки та інші графічні елементи, що слугують метафорами або символами для різних дій та завдань, які користувач може виконувати на комп’ютері, смартфоні або інших пристроях з доступом до програмного забезпечення .

Ці елементи відіграють ключову роль у створенні інтуїтивно зрозумілого середовища, полегшуючи взаємодію між людьми та технологіями. У цьому розділі кваліфікаційної роботи представлені детальні схеми, що демонструють процеси взаємодії користувача з ботом DeadlineBot, призначеним для управління дедлайнами та нагадуваннями.

На початку чату з ботом користувач отримує вітальне повідомлення, яке є першим кроком для початку взаємодії між користувачем і ботом. Воно включає в себе короткі інструкції, як взаємодіяти з ботом, а саме основні команди, такі як додавання нового дедлайну та виведення довідки з описом усіх функцій. Такий підхід забезпечує зручний початок роботи з ботом та допомагає новачкам швидко зорієнтуватися в його функціональності. Вищезгадане вітальне повідомлення, яке відображає початковий етап взаємодії, проілюстровано на рисунку 2.5.

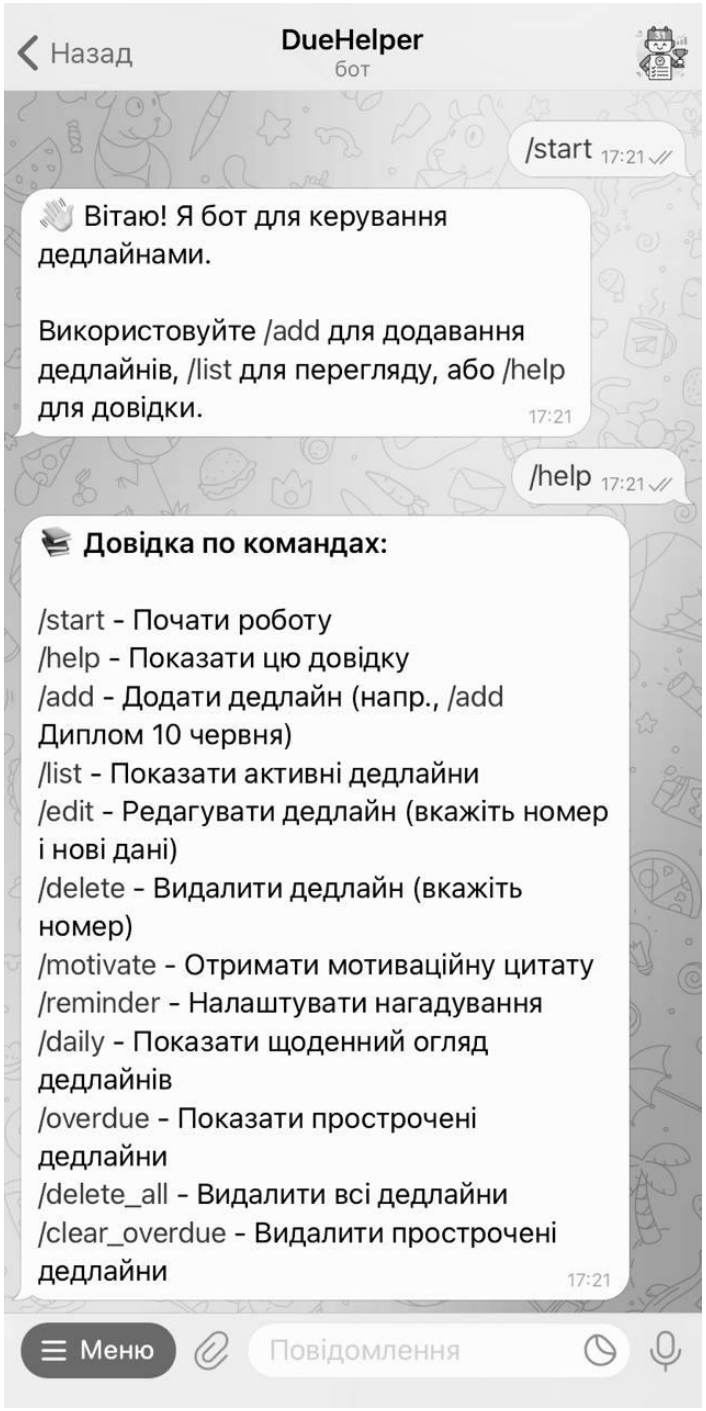

Рисунок 2.5 – Відображення вітального повідомлення з короткою інструкцією

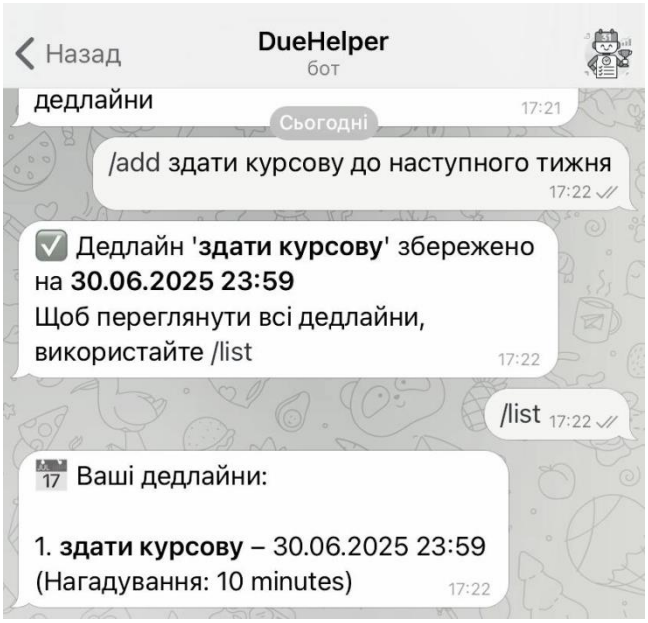

Рисунок 2.6 – Додавання дедлайну командою /add та перегляд активних дедлайнів командою /list

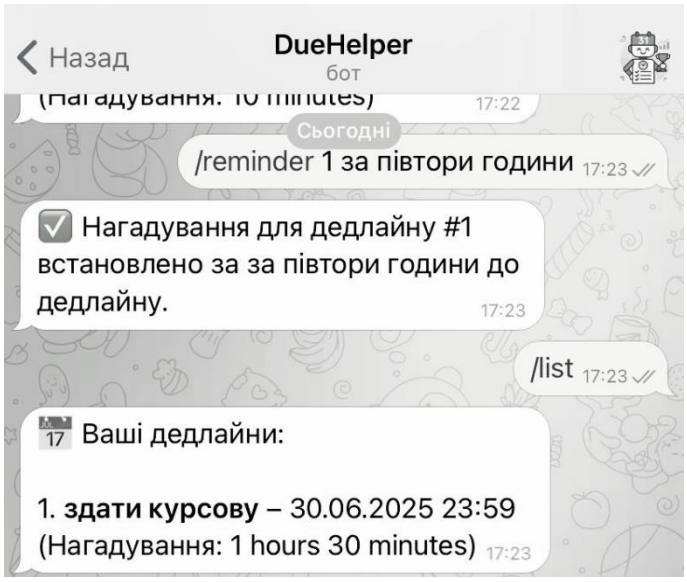

Рисунок 2.7 – Встановлення часу нагадування для дедлайну командою /reminder

Розроблено інтуїтивно зрозумілий інтерфейс для бота DeadlineBot, який через вітальне повідомлення, піктограми, кнопки та команди забезпечує зручну взаємодію користувача з функціями управління дедлайнами та нагадуваннями, полегшуючи швидке освоєння функціональності.

2.3.2 Технічний проект програмного забезпечення

Після детального вивчення ескізного проєкту було створено технічний проєкт програмного забезпечення завдяки розробці статичної моделі, відображеної через діаграму класів, а також динамічної моделі, представленої у вигляді діаграми послідовності.

Діаграма класів

Діаграма класів являє собою сукупність статичних та описових елементів моделі. Інформація, що міститься в такій діаграмі, безпосередньо лягає в основу вихідного коду програми [17]. Тобто, діаграма класів є завершальним етапом етапу проєктування й водночас слугує відправною точкою для подальшої розробки.

Статичну структуру системи ілюструє діаграма класів телеграм-боту “DueMaster”, наведена на рисунку 2.8.

Специфікації класів

Клас “DeadlineBot”

Відповідальність: головний клас, який координує роботу бота, взаємодіючи з користувачами через Telegram API та іншими модулями.

Атрибути:

- bot – об’єкт бота.
- dp – диспетчер для обробки подій.
- logger – об’єкт для логування.
- motivator – об’єкт для мотиваційних повідомлень.
- parser – об’єкт для парсингу дедлайнів.
- reminder_parser – об’єкт для парсингу нагадувань.
- pluralizer – об’єкт для множення слів.

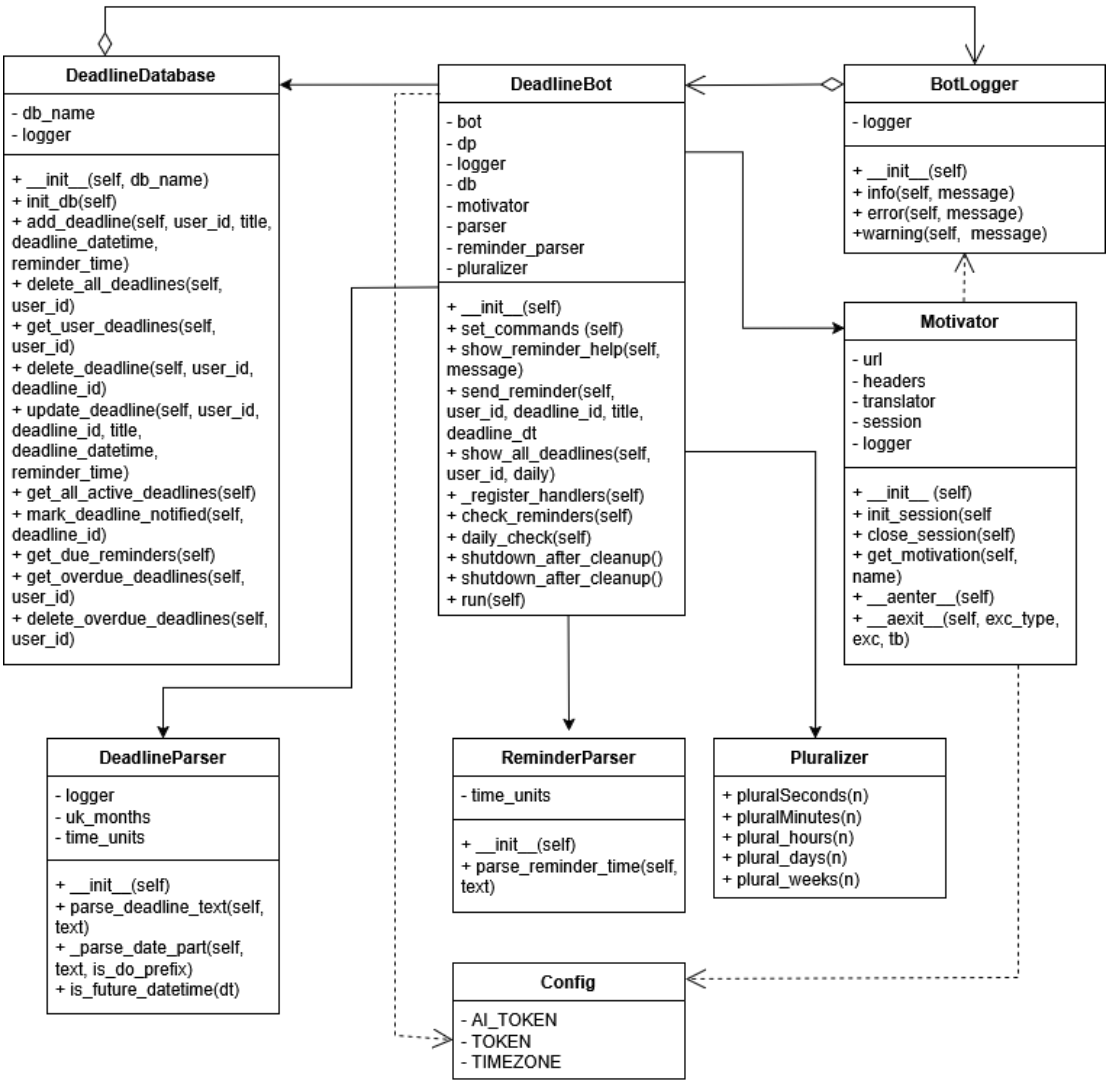

Рисунок 2.8 – Діаграма класів телеграм-боту “DueMaster”

Методи:

- init(self) – ініціалізація бота.
- set_commands(self) – налаштування доступних команд.
- show_reminder_help(self, message) – показ допомоги щодо нагадувань.
- send_reminder(self, user_id, deadline_id, title, deadline_datetime) – відправка нагадування.
- show_all_deadlines(self, user_id, daily) – показ усіх дедлайнів (щоденний або повний список).
- register_handlers(self) – реєстрація обробників подій.
- check_reminders(self) – перевірка нагадувань.
- daily_check(self) – щоденна перевірка статусу.

- shutdown_after_cleanup() – завершення роботи з очищенням.
- run(self) – запуск бота.

Клас “DeadlineDatabase”

Відповідальність: клас, призначений для управління базою даних дедлайнів, включаючи додавання, оновлення та видалення записів.

Атрибути:

- db_name – назва бази даних.
- logger – об’єкт для логування.

Методи:

- init(self, db_name) – ініціалізація бази даних із заданою назвою.
- init_db(self) – ініціалізація структури бази даних.
- add_deadline(self, user_id, title, deadline_datetime, reminder_time) – додавання нового дедлайну.
- update_deadline(self, user_id, title, deadline_datetime, reminder_time) – оновлення існуючого дедлайну.
- delete_deadline(self, user_id, deadline_id) – видалення дедлайну за ідентифікатором.
- get_all_deadlines(self) – отримання всіх дедлайнів.
- get_user_deadlines(self, user_id) – отримання дедлайнів конкретного користувача.
- get_deadline(self, user_id, deadline_id) – отримання конкретного дедлайну.
- get_active_deadlines(self) – отримання активних дедлайнів.
- get_overdue_deadlines(self) – отримання прострочених дедлайнів.
- mark_deadline_notified(self, deadline_id) – позначення дедлайну як повідомленого.
- get_due_reminders(self) – отримання нагадувань, які потрібно відправити.
- delete_overdue_deadlines(self, user_id) – видалення прострочених

дедлайнів для користувача.

Клас “BotLogger”

Відповідальність: клас для ведення логів дій бота.

Атрибути:

- logger – об’єкт логера.

Методи:

- init(self) – ініціалізація логера.
- info(self, message) – запис інформаційного повідомлення.
- error(self, message) – запис повідомлення про помилку.
- warning(self, message) – запис попередження.

Клас “Motivator”

Відповідальність: клас для генерації мотиваційних повідомлень.

Атрибути:

- url – URL для отримання даних.
- headers – заголовки для запиту.
- translator – об’єкт для перекладу.
- session – сесія для HTTP-запитів.
- logger – об’єкт логера.

Методи:

- init(self) – ініціалізація модуля мотивації.
- init_session(self) – ініціалізація HTTP-сесії.
- close_session(self) – закриття сесії.
- get_motivation(self, name) – отримання мотиваційного повідомлення.
- enter(self) – вхід у режим мотивації.
- aexit(self, exc_type, exc, tb) – вихід із режиму з обробкою винятків.

Клас “DeadlineParser”

Відповідальність: клас для аналізу тексту з інформацією про дедлайни.

Атрибути:

- logger – об’єкт логера.
- uk_months – список місяців українською.
- time_units – одиниці часу.

Методи:

- init(self) – ініціалізація парсера.
- parse_deadline_text(self, text) – парсинг тексту для витягнення дедлайну.
- parse_date_part(self, text, is_do_prefix) – парсинг частини дати.
- is_future_datetime(dt) – перевірка, чи дата в майбутньому.

Клас “ReminderParser”

Відповідальність: клас для обробки тексту з інформацією про нагадування.

Атрибути:

- time_units – одиниці часу.

Методи:

- init(self) – ініціалізація парсера нагадувань.
- parse_reminder_time(self, text) – парсинг часу нагадування.

Клас “Pluralizer”

Відповідальність: клас для коректного множення слів залежно від чисел.

Атрибути:

- time_units – одиниці часу.

Методи:

- pluralSeconds(n) – множення для секунд.
- pluralMinutes(n) – множення для хвилин.
- pluralHours(n) – множення для годин.
- pluralDays(n) – множення для днів.
- pluralWeeks(n) – множення для тижнів.

Клас “ReminderParser”

Відповідальність: клас для обробки тексту з інформацією про нагадування.

Атрибути:

- AI_TOKEN – токен для AI.
- TIMEZONE – часовий пояс.

Діаграми послідовностей

Діаграми послідовностей застосовуються для точнішого розкриття логіки сценаріїв використання. У таких діаграмах, як правило, представлені об’єкти, що беруть участь у взаємодії в межах певного сценарію, повідомлення, які вони надсилають одне одному, а також відповідні результати, пов’язані з цими повідомленнями [18].

Виходячи зі створеної статичної моделі та діаграми варіантів використання, було розроблено динамічну модель програмного забезпечення. Діаграма послідовності, що ілюструє сценарій «Додавання дедлайну», подана на рисунку 2.9.

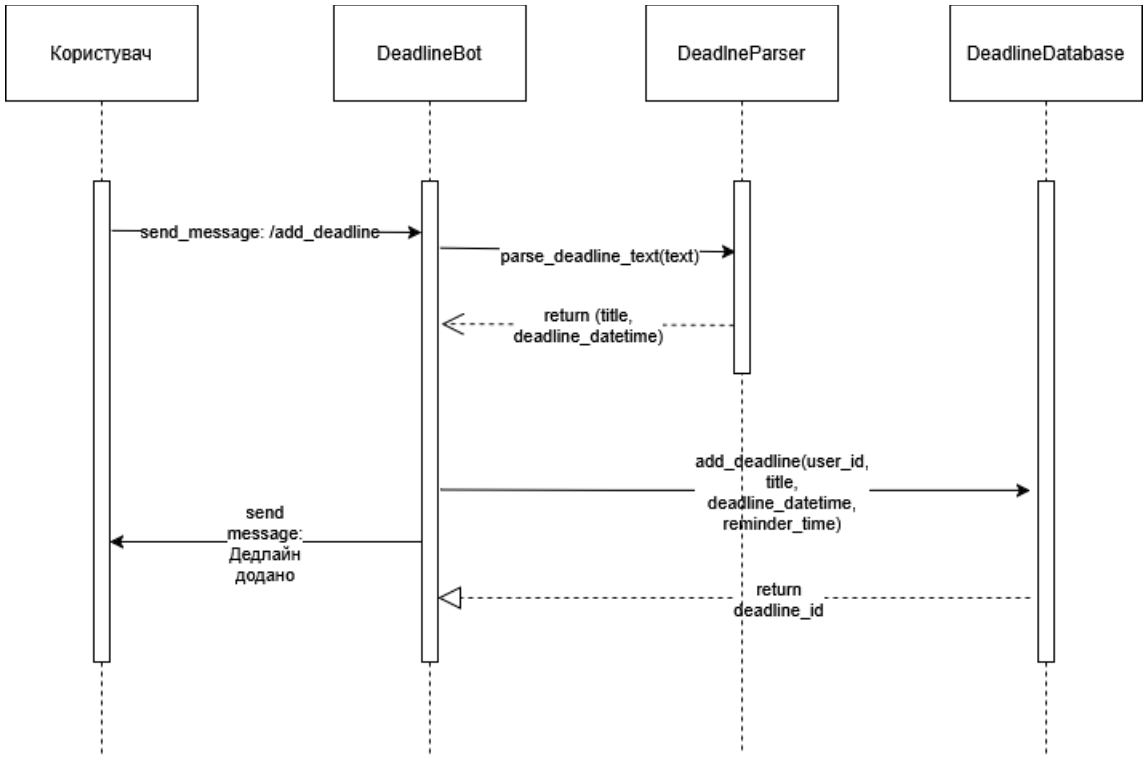

Рисунок 2.9 – Діаграма послідовності для варіанту використання «Додавання дедлайну»

Отже, для демонстрації динамічної поведінки системи було створено відповідну діаграму послідовності.

2.3.3 Робочий проект програмного забезпечення

Вибір мови програмування

У процесі розробки програмного забезпечення було прийнято рішення застосовувати мову програмування Python як базову технологію. Python є високорівневою інтерпретованою мовою програмування, призначеною для створення широкого спектра програмних продуктів, включаючи веб-застосунки, десктопні системи, ігрові додатки, а також рішення для роботи з базами даних. Зокрема, Python здобув значну популярність у галузях машинного навчання та штучного інтелекту [14].

Основними характеристиками мови Python є:

– виконання програмного коду у вигляді скриптів без необхідності попередньої компіляції;

– підтримка багатьох парадигм програмування, серед яких об’єктно-орієнтована, процедурна та функціональна;

– незалежність від платформи, що забезпечує безперешкодне виконання розроблених програм на різних операційних системах, таких як Windows, macOS та Linux, за умови наявності відповідного інтерпретатора;

– автоматизоване управління пам’яттю, що спрощує процес розробки та експлуатації програм;

– динамічна типізація, яка підвищує гнучкість коду та прискорює розробку.

Мова Python вирізняється порівняно простим та інтуїтивно зрозумілим синтаксисом, що сприяє її широкому впровадженню у навчальний процес, а також робить її доступною для початківців. Водночас, завдяки широкому набору потужних бібліотек та активній спільноті розробників, Python знайшов

широке застосування і у комерційній розробці, забезпечуючи розробникам інструменти для реалізації складних і масштабних проектів. Значна кількість доступних ресурсів та документації дає змогу оперативно знаходити приклади, рекомендації та отримувати підтримку від фахівців.

Реалізація та тестування програмного забезпечення

Програмне забезпечення було розроблено за допомогою мови програмування Python із застосуванням технологій Telegram Bot API для реалізації інтерфесу.

Для тестування обрано основну функцію - варіант використання «додавання дедлайну» (/add). Поведінкове тестування, відоме також як тестування «чорного ящика», є методом оцінки функціональної поведінки програми з погляду зовнішнього користувача, за якого внутрішня структура коду залишається недоступною для тестувальників. Ця стратегія передбачає систематичний підбір і створення тестових сценаріїв для формування тестового набору.

Тестування варіанту використання «Додавання дедлайну»

Користувач вводить команду /add із параметрами (текст і дата), щоб додати новий дедлайн. Вхідні дані повинні містити коректний формат: текст завдання і дату в стилі DD.MM.YYYY. Створено 6 тестів для перевірки обробки неправильного введення:

Тест 1

Вхідні дані:

– /add текст 01.13.2025 (неправильний місяць)

Очікуваний результат:

– бот видає повідомлення про помилку (наприклад, "❌Неправильний формат дати").

Результат тестування представлений на рисунку 2.10.

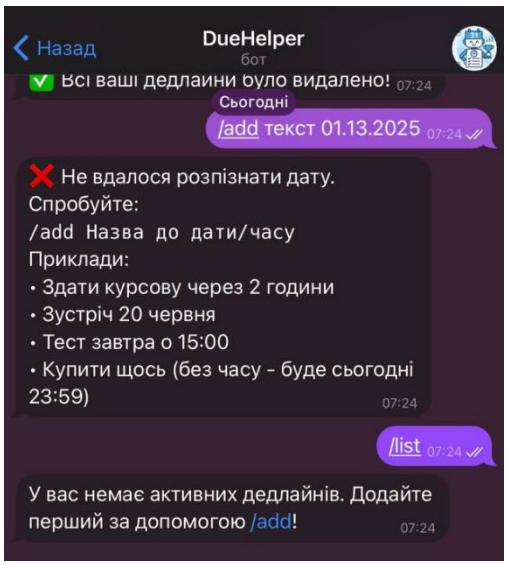

Рисунок 2.10 – Результати тестування варіанту використання «Додавання дедлайну» (тест 1)

Тест 2

Вхідні дані:

– /add текст без дати

Очікуваний результат:

– бот помічає, що при додаванні дедлайну не було введення дати, тому він автоматично ставить датою дедлайну сьогоднішній день та час 23:59.

Результат тестування представлений на рисунку 2.11.

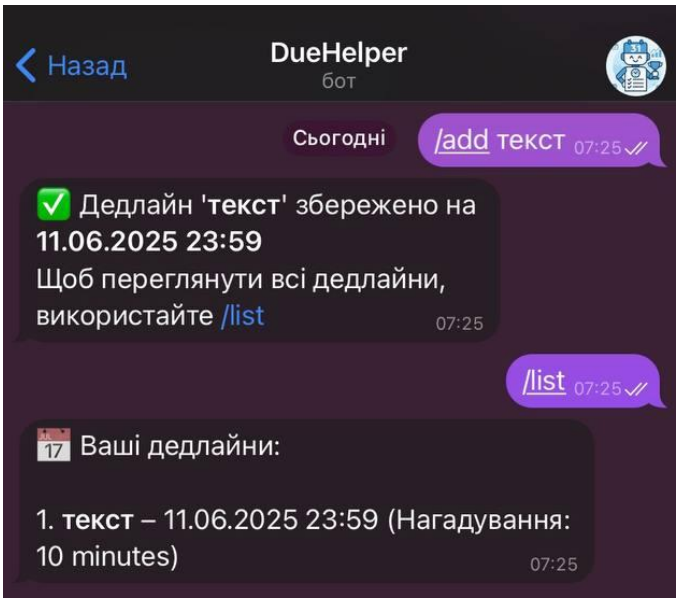

Рисунок 2.11 – Результати тестування варіанту використання «Додавання дедлайну» (тест 2)

Тест 3

Вхідні дані:

– /add текст 01.01.2024 (минула дата)

Очікуваний результат:

– Бот видає повідомлення про помилку (наприклад, "❌Неправильний формат дати").

Результат тестування представлений на рисунку 2.12.

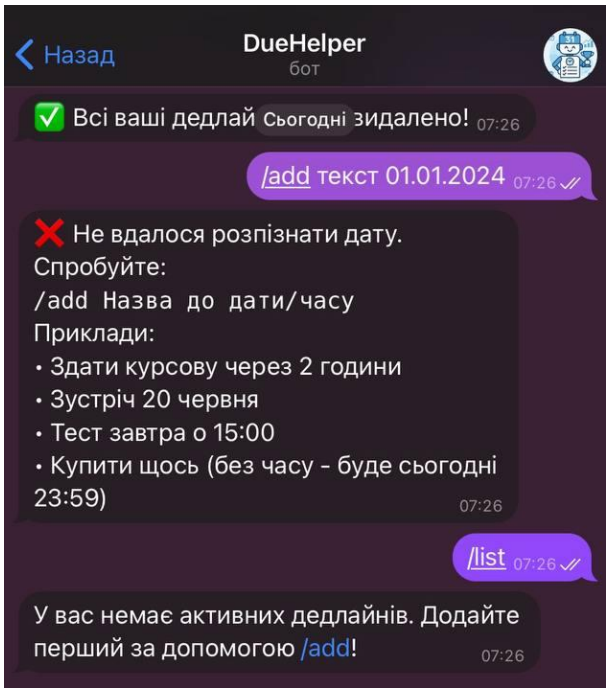

Рисунок 2.12 – Результати тестування варіанту використання «Додавання дедлайну» (тест 3)

Тест 4

Вхідні дані:

– /add (без параметрів)

Очікуваний результат:

– Бот видає повідомлення про помилку (наприклад, "❌Вкажіть текст і дату").

Результат тестування представлений на рисунку 2.13.

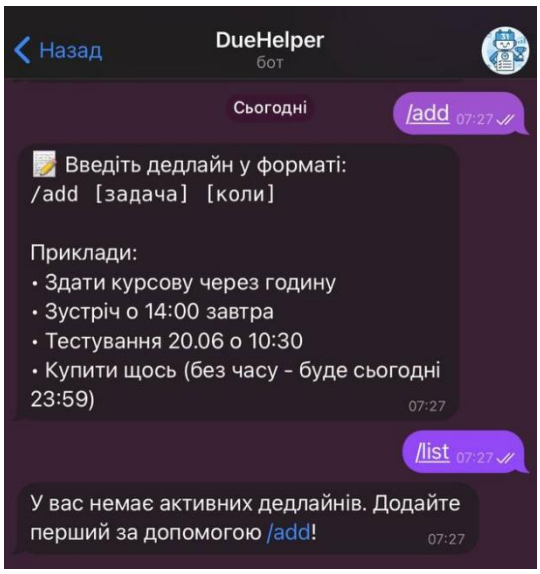

Рисунок 2.13 – Результати тестування варіанту використання «Додавання дедлайну» (тест 4)

Тест 5

Вхідні дані:

– /add текст 32.01.2025 (неіснуюча дата)

Очікуваний результат:

– Бот видає повідомлення про помилку (наприклад, "❌Неправильна дата").

Результат тестування представлений на рисунку 2.14.

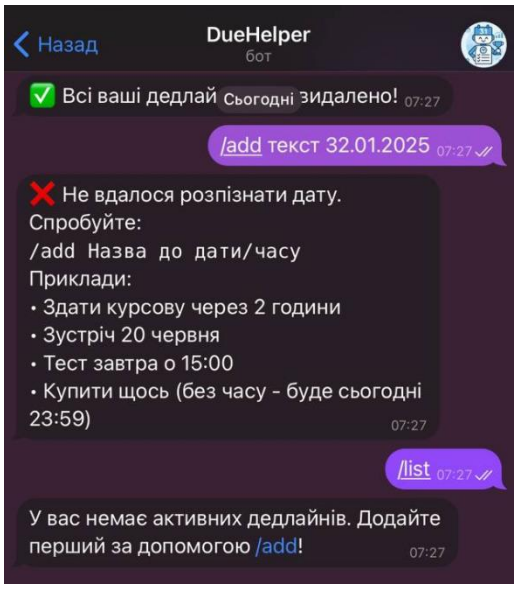

Рисунок 2.14 – Результати тестування варіанту використання «Додавання дедлайну» (тест 5)

Тест 6

Вхідні дані:

– /add 13.05.2025 (без назви)

Очікуваний результат:

– Бот помічає, що в повідомленні відсутня назва, тому автоматично встановлює стандарту назву “Дедлайн”.

Результат тестування представлений на рисунку 2.15.

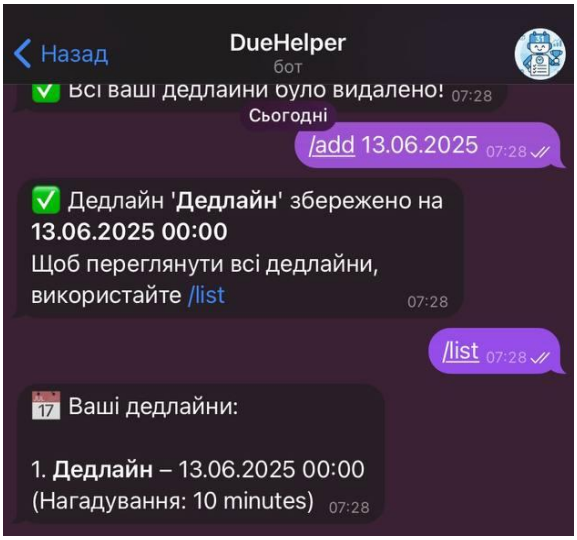

Рисунок 2.15 – Результати тестування варіанту використання «Додавання дедлайну» (тест 6)

Випробування програмного забезпечення

У Додатку В міститься детальний опис процедури тестування розробленого програмного забезпечення, включаючи методику перевірки його функціоналу. Відповідно до вказаних у додатку методичних рекомендацій було проведено комплексне тестування роботи бота.

Після активації програми користувач отримує інформаційне повідомлення-привітання, яке супроводжується стислими вказівками щодо використання функціоналу бота. Візуальне представлення інтерфейсу проілюстровано на рисунках 2.16 - 2.24.

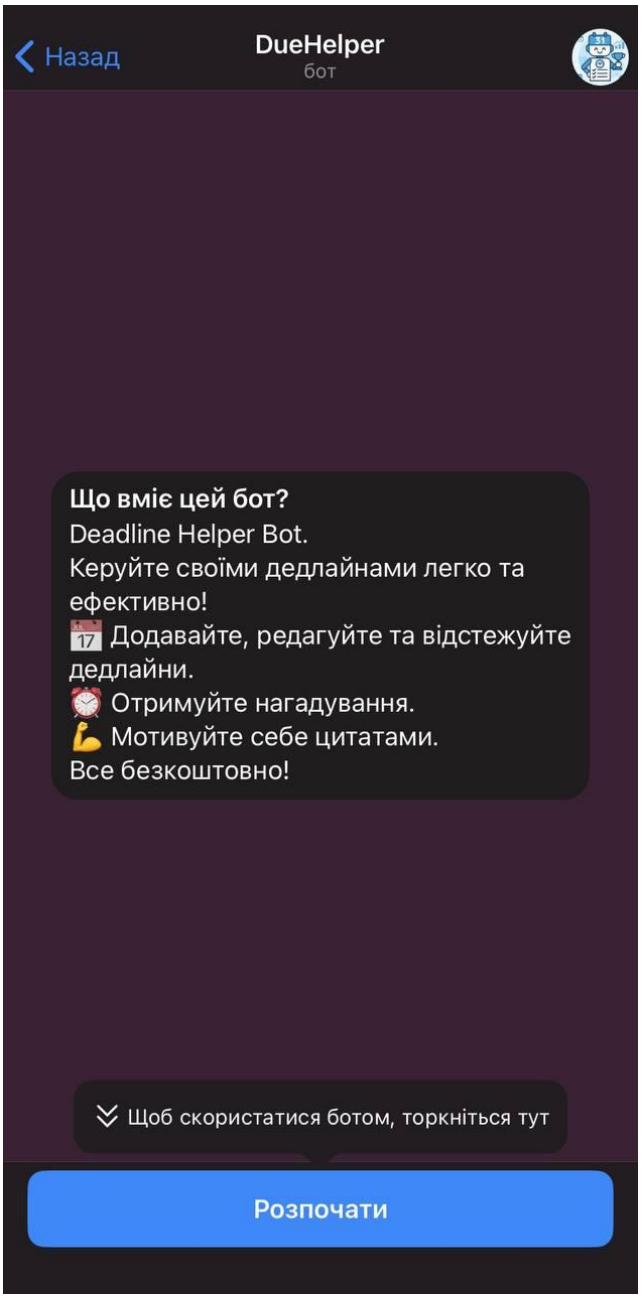

Рисунок 2.16 – Короткий інформативний блок перед початком роботи

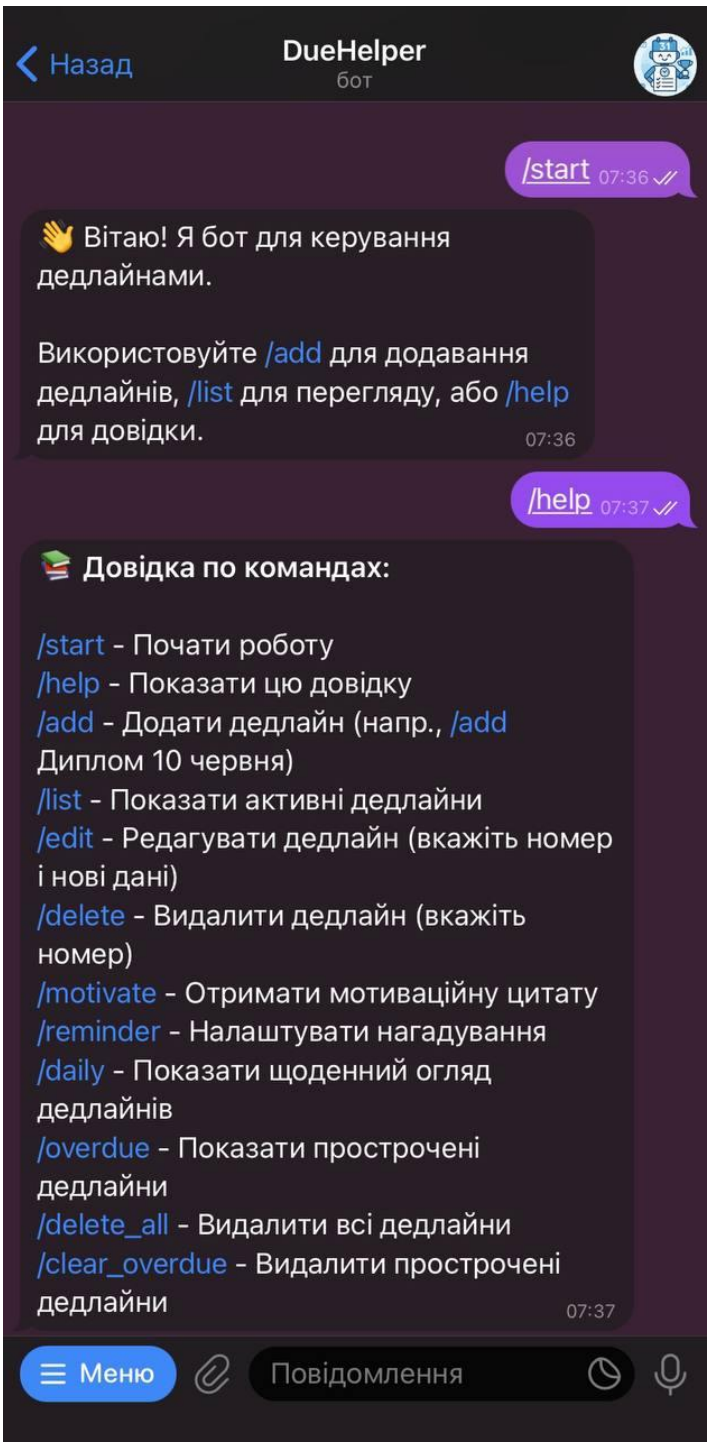

Рисунок 2.17 – Привітання на початку роботи і довідка викликана командою /help

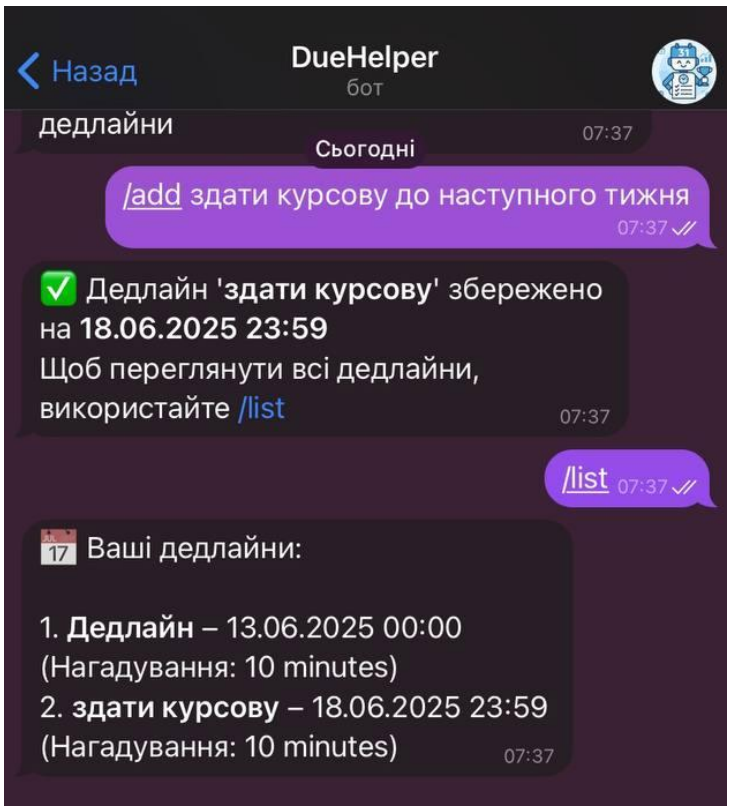

Рисунок 2.18 – Додавання дедлайну командою /add та перегляд створених дедлайнів командою /list

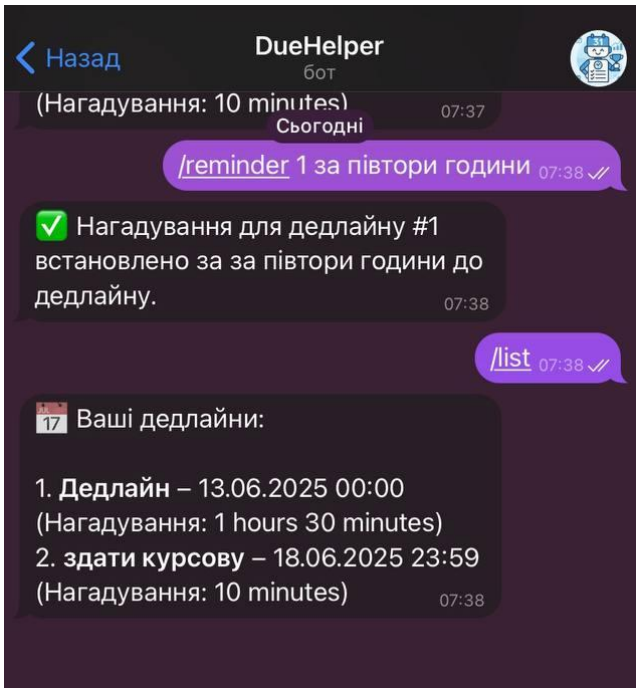

Рисунок 2.19 – Встановлення нагадування для дедлайну командою /reminder та перегляд змін командою /list

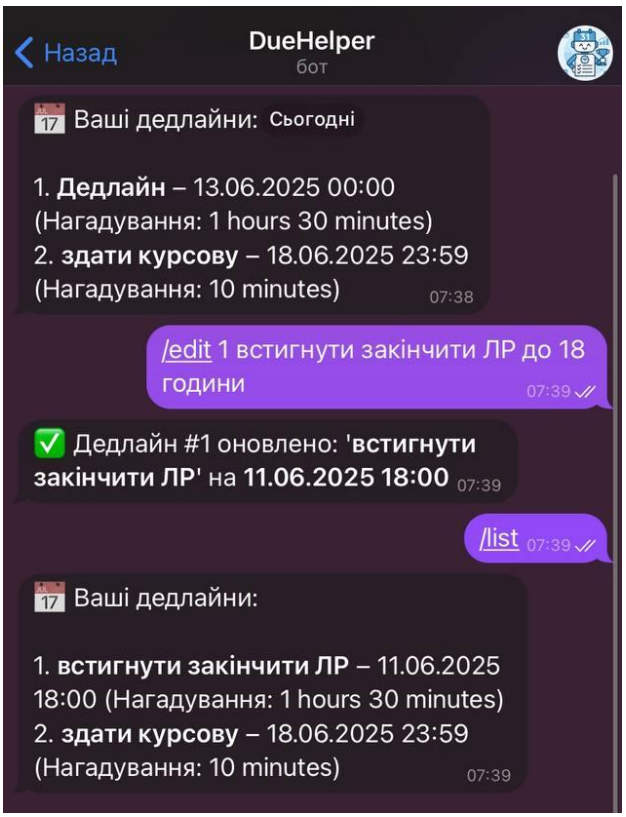

Рисунок 2.20 – Редагування вже існуючого дедлайна командою /edit та перегляд змін командою /list

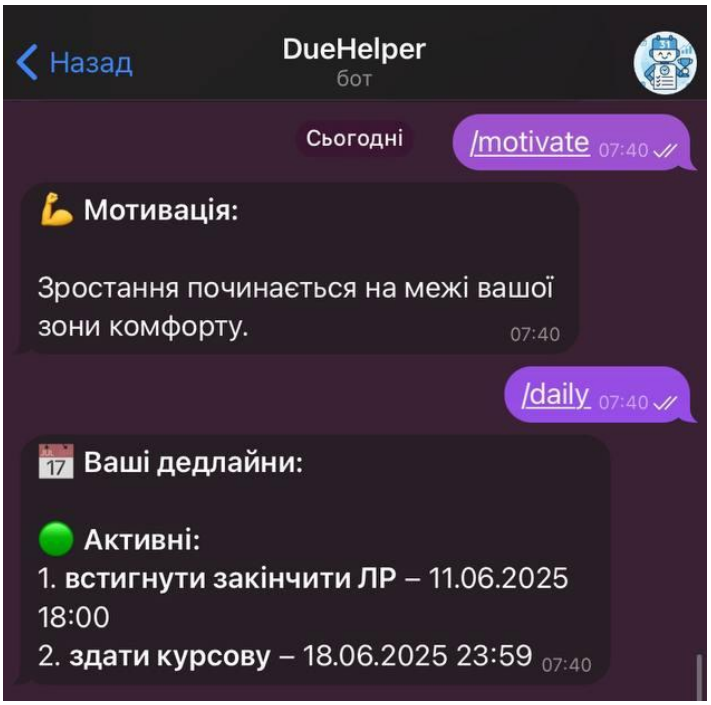

Рисунок 2.21 – Генерація мотиваційного повідомлення командою /motivate та перегляд дедлайнів на сьогодні командою /daily

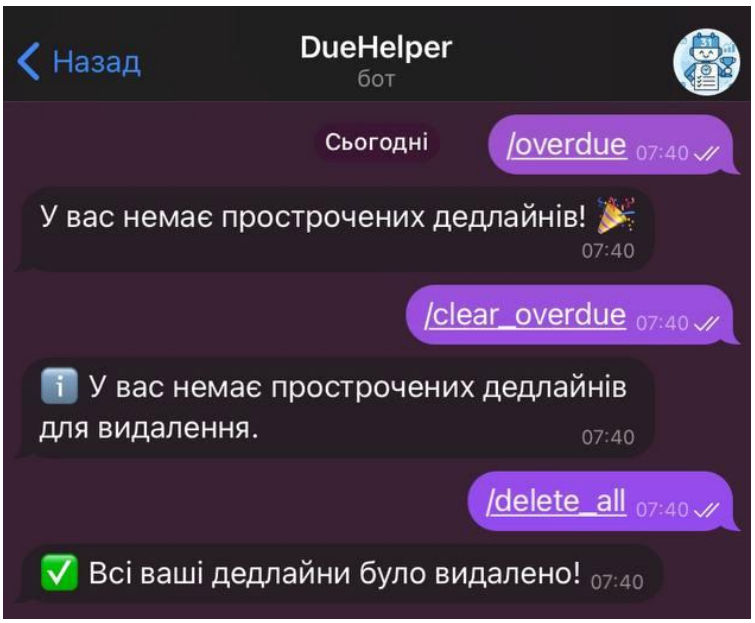

Рисунок 2.22 – Перегляд протермінованих дедлайнів командою /overdue та видалення усіх дедлайнів командою /delete_all

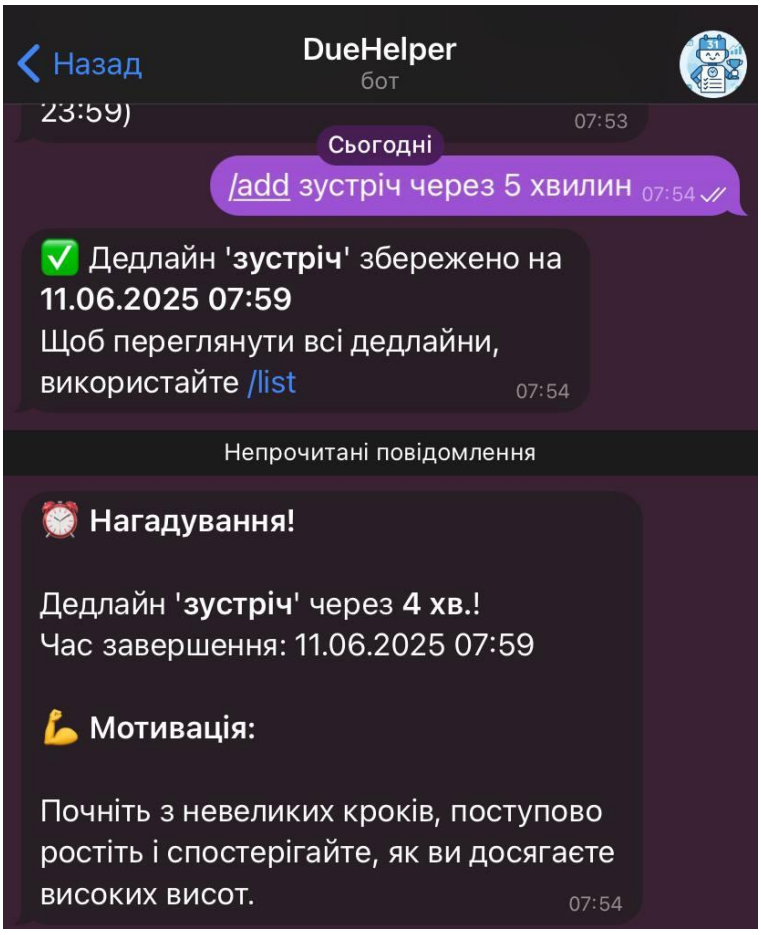

Рисунок 2.23 – Отримання повідомлення з нагадуванням та мотиваційною порадою

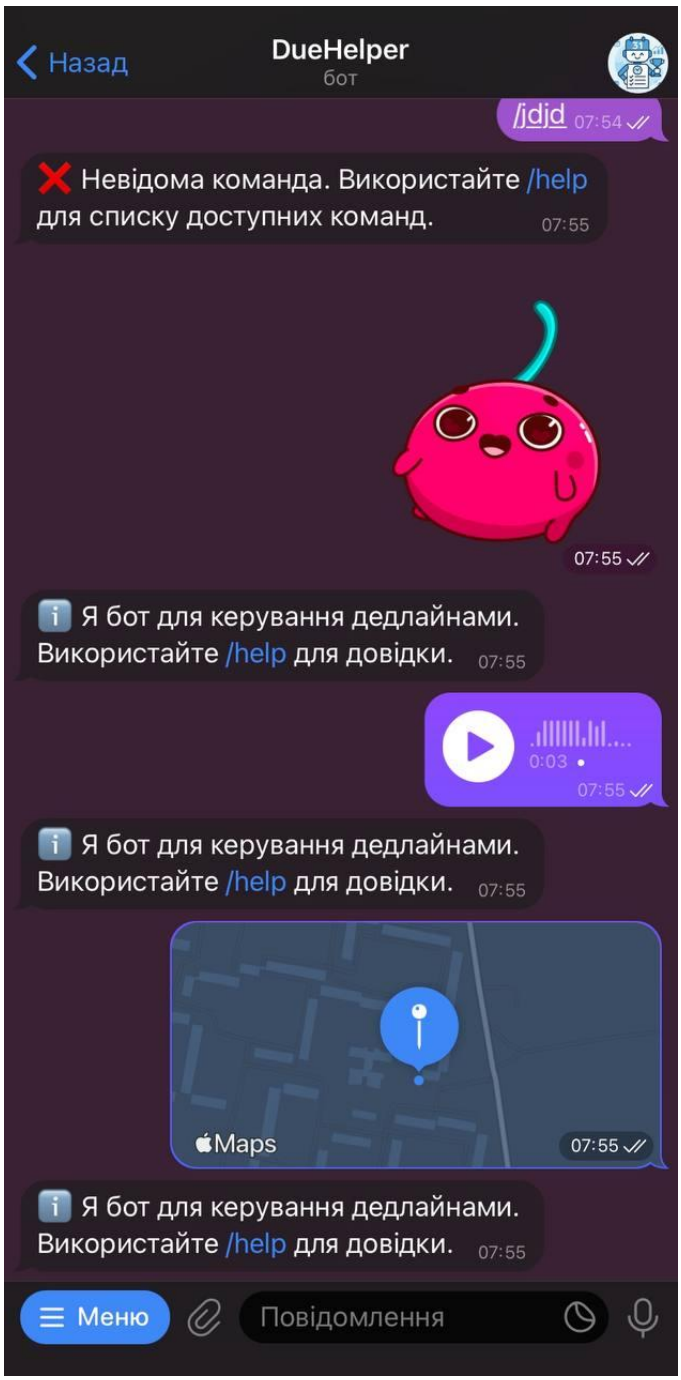

Рисунок 2.24 – Реакція бота на некоректний тип повідомлення

Отже, проведене тестування розробленого програмного забезпечення телеграм-бота «DueMaster» підтвердило його повну працездатність. Це сприяє подальшому вдосконаленню та розширенню функціональних можливостей бота.

3 РЕЗУЛЬТАТИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТЕЛЕГРАМ- БОТУ "DUEMASTER" ДЛЯ ВІДСТЕЖЕННЯ ДЕДЛАЙНІВ З ІНТЕГРАЦІЄЮ ШТУЧНОГО ІНТЕЛЕКТУ

У процесі виконання кваліфікаційної роботи було створено програмне забезпечення телеграм-бота "DueMaster" для відстеження дедлайнів з інтеграцією штучного інтелекту. Розроблений бот відповідає вимогам, визначеним у технічному завданні, а саме: обробка текстових повідомлень для додавання дедлайнів, зберігання даних, надсилання нагадувань та генерація мотиваційних порад. Рисунок 3.1 демонструє повідомлення, що бот надсилає користувачу при першому запуску.

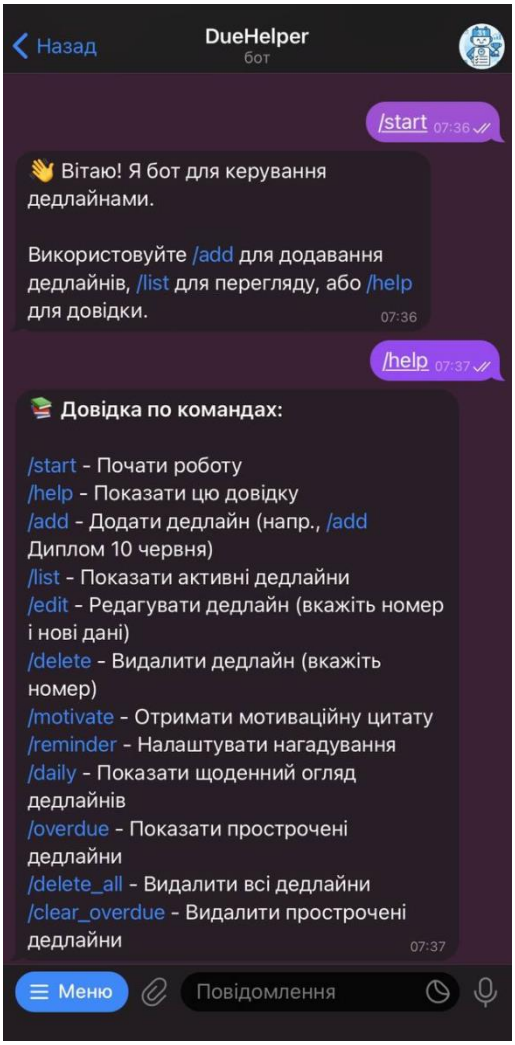

Рисунок 3.1 – Початок роботи з телеграм-ботом "DueMaster"

У рамках кваліфікаційної роботи проведено огляд інтернет-джерел для аналізу роботи систем управління часом, месенджерів, телеграм-ботів та методів розробки програмного забезпечення для реалізації телеграм-бота "DueMaster".

Здійснено пошук і аналіз існуючих програмних рішень для автоматизації відстеження дедлайнів та підвищення продуктивності користувачів.

Виконано постановку задачі та розроблено технічне завдання для створення телеграм-бота "DueMaster", яке наведено в Додатку А – Технічне завдання.

Виконано аналіз вимог до програмного забезпечення, розроблено архітектуру програмного забезпечення. У процесі роботи реалізовано ескізний, технічний та робочий проекти:

У рамках ескізного проекту створено інформаційну модель та ескіз інтерфейсу користувача.

У рамках технічного проекту розроблено діаграму класів для статичного представлення системи та діаграму послідовності для динамічного відображення роботи бота.

У рамках робочого проекту виконано кодування, проведено тестування та випробування програмного забезпечення, які підтвердили його повну працездатність і відповідність вимогам технічного завдання для телеграм-бота "DueMaster".

Для розробки використано мову програмування Python. Текст програми містить 1114 рядків та займає 47,8 МБ дискового простору. Код програми наведено в Додатку Б – Текст програми.

Після тестування функціональних характеристик підтверджено відповідність програмного забезпечення технічному завданню та готовність до впровадження. Випробування проведено відповідно до програми та методики, наведених у Додатку В – Програма та методика випробування ПЗ.

Опис телеграм-бота "DueMaster" представлено в Додатку Г – Опис програми.

Для роботи програмного забезпечення на сервері необхідно встановити інтерпретатор мови Python. Користувачу достатньо мати встановлений додаток Telegram на своєму пристрої або використовувати веб-версію для роботи з ПК.

Інструкцію для користувача телеграм-бота "DueMaster" наведено в Додатку Д – Інструкція користувача.

4 ОХОРОНА ПРАЦІ

4.1 Вступ

Згідно положенням закону “Про охорону праці” [20], основним принципом державної політики є захист життя та здоров’я працівників, який має перевагу над будь-якими виробничими інтересами.

Забезпечення належних умов праці та безпеки робочого середовища є однією з ключових задач соціально політики сучасної держави, що розвивається. Перехід до ринково-економічних відносин та демократичні тенденції у суспільстві вимагають комплексного покращення умов праці, створення безпечного робочого простору, а також ефективного захисту населення від наслідків техногенних аварій.

Це зумовлює необхідність реорганізації підходів до охорони праці на кожному з рівнів: від звичайного працівника до керівних посад. Необхідно підвищувати рівень обізнаності та фахової підготовки спеціалістів стосовно безпеки на виробництві.

Сьогоденна науково-технічна революція спричинила масштабне застосування техніки у виробничі та офісні процеси. Не дивлячись на те, що це значно підвищило продуктивність праці, робота з ПК потребує чималих фізичних, емоційних та розумових ресурсів.

Що, насамперед, висуває нові вимоги до організації робочого процесу та відпочинку. Робота з комп’ютером пов’язана з низкою шкідливих та небезпечних факторів: перенапруження зору, незручне положення тіла, електромагнітне випромінювання, стресові навантаження тощо.

Дотримання належного режиму праці, правильне облаштування робочого місця та виконання фізичних вправ – головні складові запобігання захворювань, що пов’язані з тривалою роботою за комп’ютером.

4.2 Оцінка негативного та небезпечного впливу, що виникає при використанні персонального комп’ютера

Задля ефективної та комфортної роботи за комп’ютером слід ретельно організувати власний робочий простір.

Площа, відведена одному працівникові, повинна становити щонайменше 6 м², а відстань між робочими місцями – не менше 2 метрів. Конструкція столу має забезпечувати зручне розміщення усіх необхідних пристроїв. Оптимальна висота столу – 725 мм, ширина – від 800 до 1400 мм, глибина – 800–1000 мм. Простір для ніг повинен становити не менше: по висоті – 600 мм, по ширині – 500 мм, у глибину на рівні колін – 450 мм, а на рівні витягнутих ніг – не менше 650 мм.

Клавіатура розміщується на відстані 100–300 мм від краю столу, а монітор встановлюється на відстані 600–700 мм від очей користувача (мінімум – 500 мм).

У приміщенні має проводитися регулярне вологе прибирання та обов’язкове провітрювання не рідше ніж щогодини. Обладнання, що створює шум (принтери, сканери), може залишатися на столі лише за умови, що не заважає роботі та не закриває екран. При перевищенні допустимого шумового рівня такі пристрої повинні переміщуватись за межі робочої зони.

Перед початком роботи працівник має переконатися у:

– коректному розміщенні монітора, клавіатури та іншої периферії;

– Зручному положенні тіла –правильна висота стільця і столу;

– Достатньому освітленні робочого місця, щоб уникати надмірного напруження очей;

– Справності та цілісності кабелів, роз’ємів та пристроїв

Заборонено приступати до роботи, якщо виявлено пошкодження або несправності будь-якого обладнання, шнурів живлення чи з’єднувальних елементів.

4.3 Заходи пожежної безпеки на робочому місці за персональним комп’ютером

Забезпечення пожежної безпеки в офісному середовищі, де використовуються персональні комп’ютери, є одним з ключових елементів охорони праці. Пожежна безпека охоплює як профілактику виникнення займання, так і оперативне реагування у разі надзвичайної ситуації. З метою зменшення ризиків необхідно впровадити низку організаційних та технічних заходів:

– Наявність і технічне обслуговування первинних засобів пожежогасіння. Вогнегасники повинні бути встановлені на легкодоступних, добре помітних ділянках робочої зони. Працівники мають бути ознайомлені з їх розташуванням і порядком використання. Періодична перевірка та регламентне обслуговування є обов’язковими для забезпечення їх ефективного функціонування.

Для гасіння пожежі застосовуватиметься установка об’ємного пожежогасіння. Розміри приміщення: АхВхН = 4х5х2,5 м;

Кількість відкритих прорізів – 2 шт.

Площа відкритих прорізів по відношенню до площі огороджувальних конструкцій 4% та 5%. Оскільки сумарна площа відкритих прорізів не перевищує 15%, то можна застосувати об’ємне пожежогасіння. Обрано вогнегасний порошок загального призначення П-2АП (ТУ У6-057663662-001ТУ).

Загальний об’єм приміщення:

$$V = A * B * H = 4 * 5 * 2,5 = 50 \text{ куб.м.}$$

Для обраного порошку норма подачі для об’ємного гасіння 0,6 кг/куб.м.

Основна маса порошку становитиме:

$$M_1 = q_{v0} * V = 0,6 * 50 = 30 \text{ кг.}$$

Оскільки площа отворів менше ніж 5%, то для визначення додаткової маси вогнегасного порошку застосовуємо формулу:

$$M_2 = 2{,}5 * (S_1 + S_2) = 2{,}5 * (2 + 2{,}5) = 11{,}25 \text{ кг.}$$

Таким чином, загальна мінімальна кількість вогнегасного порошку, необхідного для захисту приміщення, буде складати:

$$M_{min} = M_1 + M_2 = 30 + 11{,}25 = 41{,}25 \text{ кг.}$$

Визначимо мінімальну витрату порошку:

$$\mathrm{G} = \frac{\mathrm{M_{min}}}{30} = \frac{41.25}{30} = \frac{1.375 \text{ кг}}{30 \text{ с.}}$$

Визначимо мінімальну тривалість випуску вогнегасної речовини за формулою:

$$t_{min} = 0{,}67 q_{v0} * I^{-1} = 0{,}67 * 0{,}6 * 0{,}02^{-1} = 20{,}1\ c.$$

За отриманими значеннями M_{min}, G_{min}, t_{min} обираємо установку порошкового пожежогасіння.

– Розробка та впровадження плану евакуації.

План евакуації – важливий аспект пожежної безпеки в офісі. Включає чітке позначення усіх евакуаційних виходів (основних та запасних), вказання маршрутів виходу, розташування місць збору персоналу після евакуації. Призначаються відповідальні особи, які координують евакуацію, контролюють переміщення працівників та здійснюють облік персоналу після виходу.

Ось певні основні аспекти плану евакуації:

1. Визначення виходів: у плані евакуації повинна бути зазначена інформація про всі основні та запасні шляхи виходу з приміщення. Усі виходи мають бути чітко промарковані, не загромаджені сторонніми предметами. Працівники повинні знати місцезнаходження найближчих виходів і вміти швидко зорієнтуватися у разі необхідності.

2. Складальні пункти: план повинен включати конкретно визначені місця збору працівників поза межами будівлі, які забезпечують можливість оперативного контролю чисельності персоналу та виключення залишення когось у небезпечній зоні.

3. Ролі та відповідальності: в межах плану евакуації мають бути передбачені ролі відповідальних осіб: координаторів евакуації, працівників, які слідкують за дотриманням маршруту, та осіб, що здійснюють перевірку

присутності персоналу в точках збору. Це сприяє злагодженому та безпечному проведенню евакуаційних заходів.

4. Актуалізація та інформування: План евакуації повинен проходити регулярну перевірку на актуальність, особливо після змін у плануванні офісу або у кадровому складі. Новоприйняті працівники обов’язково мають бути ознайомлені з процедурою евакуації в межах вступного інструктажу.

5. Навчання та практичні тренінги: усі працівники проходять інструктаж із пожежної безпеки, а також беруть участь у періодичних навчаннях з евакуації, що дозволяє знизити паніку та підвищити організованість у разі загрози.

– Монтаж систем раннього виявлення пожежі. У ключових зонах офісу встановлюються пожежні та димові датчики, які мають проходити регулярне обслуговування і тестування на справність. Їхнє функціонування забезпечує оперативне реагування до моменту поширення вогню.

4.4 Вимоги безпеки, яких потрібно дотримуватися під час виконання робіт

Під час тривалої роботи за комп’ютером важливо дотримуватись встановленого режиму праці з регулярними перервами для зменшення зорового та фізичного навантаження. Це сприяє підтриманню продуктивності та збереженню здоров’я працівника.

Робота за монітором має чергуватись з короткочасними відпочинками або з виконанням інших видів діяльності, що не пов’язані з використанням екрана. Такий підхід допомагає знизити втому та навантаження на опорно-руховий апарат.

Тривалість та частота регламентованих перерв визначаються залежно від тривалості зміни та категорії виконуваних робіт відповідно до нормативних рекомендацій.

При 8-годинній робочій зміні і роботі з ПК регламентовані перерви встановлюються:

– Для II категорії робіт через 2 години від початку робочої зміни і через 1,5-2 години після обідньої перерви тривалістю 15 хвилин кожен або тривалістю 10 хвилин через кожну годину роботи;

– Для III категорії робіт через 1,5-2 години від початку робочої зміни і через 1,5-2 години після обідньої перерви тривалістю 20 хвилин кожен або тривалістю 15 хвилин через кожну годину роботи.

При 12-годинній робочій зміні і роботі з ПК регламентовані перерви встановлюються в перші 8 годин роботи аналогічно перерв при 8-годинній робочій зміні, а протягом останніх 4 годин роботи, незалежно від категорії і виду робіт, щогодини тривалістю 15 хвилин.

При роботі з ПК в нічну зміну (з 22.00 до 6.00) незалежно від категорії і виду трудової діяльності сумарна тривалість регламентованих перерв збільшується на 60 хвилин.

Тривалість безперервної роботи з ПК без регламентованого перерви не повинна перевищувати 2 годин.

У періоди відпочинку рекомендується виконувати спеціальні комплекси вправ для розвантаження очей, опорно-рухового апарату, а також для зниження психоемоційної напруги.

Щоб знизити негативний вплив монотонної праці, доцільно чергувати види завдань, якщо це можливо.

Не слід залишати обладнання включеним без нагляду. При необхідності припинення на деякий час роботи коректно закриваються всі активні завдання та обладнання вимикається.

Для запобігання нещасним випадкам і несправностям обладнання необхідно дотримуватись наступних правил::

– не залишати ПК та периферійні пристрої ввімкненими без нагляду;

– перед перервою або завершенням роботи коректно завершувати всі активні процеси та вимикати обладнання;

– не торкатися кабелів живлення, портів і з’єднань під час роботи обладнання;

– не захаращувати поверхні робочої зони сторонніми предметами;

– не відключати живлення під час активної роботи програмного забезпечення;

– уникати потрапляння рідини на корпус чи елементи ПК;

– не вмикати техніку, яка тільки-но була занесена з вулиці в холодну пору року;

– не здійснювати самостійного розкриття корпусів чи ремонтів апаратури;

– не очищати пристрої від пилу, поки вони підключені до мережі живлення;

– не допускати присутності сторонніх осіб у зоні роботи з обладнанням.

ВИСНОВКИ

У рамках кваліфікаційної роботи розв’язано практичну задачу інженерії програмного забезпечення, що характеризується комплексністю та невизначеністю умов – розроблено телеграм-бот «DueMaster» для відстеження дедлайнів з інтеграцією штучного інтелекту, що забезпечує ефективне управління часом.

Всі задачі, які потрібно було вирішити для досягнення мети кваліфікаційної роботи виповнені в повному обсязі, а саме:

– Проаналізовано практичну задачу інженерії програмного забезпечення для розробки -телеграм-бота "DueMaster".

– Проаналізовано та порівняно існуючі інструменти для розробки подібних рішень.

– Здійснено постановку задачі на створення телеграм-бота «DueMaster».

– Проведено аналіз вимог до програмного забезпечення для реалізації функціоналу відстеження дедлайнів з інтеграцією штучного інтелекту.

– Розроблено архітектуру телеграм-бота «DueMaster».

– Створено технічне завдання на розробку телеграм-бота «DueMaster» для відстеження дедлайнів.

– Здійснено проектування та попереднє тестування основних компонентів бота, що підтвердило їхню працездатність.

– Розроблено документацію до програмного забезпечення телеграм-бота «DueMaster».

– Розглянуто актуальні питання охорони праці при роботі за персональним комп’ютером.

Мета кваліфікаційної роботи досягнута, усі завдання виконано в повному обсязі. Робота дозволила не лише теоретично осмислити задачу, але й практично реалізувати основні етапи розробки програмного забезпечення, що відповідає актуальним вимогам до автоматизації управління часом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Laudon K.C., Traver C.G. E-commerce: Business, Technology, Society. London, 2020. 912 p.

2. Trello Blog: Time Management Tools. Overview of tools for deadline tracking. URL: https://blog.trello.com/time-management-tools (дата звернення: 17.02.2025)

3. Документація до мови програмування Python. URL: https://docs.python.org/3/ (дата звернення: 20.02.2025)

4. Обробка природної мови (NLP). URL: https://uk.shaip.com/blog/what-is-nlp-how-it-works-benefits-challenges-examples/ (дата звернення 25.03.2025).

5. Документація до системи керування базами даних SQLite. URL: https://www.sqlite.org/docs.html (дата звернення: 05.03.2025)

6. Документація до бібліотеки python-telegram-bot для розробки телеграм-ботів. URL: https://python-telegram-bot.readthedocs.io/ (дата звернення: 02.03.2025).

7. Документація до бібліотеки Aiogram для розробки асинхроних ботів . URL: https://docs.aiogram.dev/ (дата звернення: 02.03.2025).

8. Документація до бібліотеки Spacy для обробки природної мови на Python. URL: https://spacy.io/api/doc (дата звернення: 17.03.2025).

9. Документація до бібліотеки APScheduler для планування завдань. URL: https://apscheduler.readthedocs.io/ (дата звернення: 23.03.2025).

10. Документаця до бібліотеки Telebot для розробки телеграм-ботів. URL: https://pytba.readthedocs.io/en/latest/ (дата звернення: 28.04.2025).

11. Документація до бібліотеки Pendulum для роботи з датами. URL: https://pendulum.eustace.io/docs/ (дата звернення: 02.03.2025).

12. Документація до бібліотеки typing для статичної типізації коду. URL: https://docs.python.org/3/library/typing.html (дата звернення: 04.03.2025)

13. Документація до OpenRouter API. URL: https://openrouter.ai/docs (дата звернення: 26.03.2025)

14. Уніфікована мова моделювання UML. URL: https://uk.wikipedia.org/wiki/Unified_Modeling_Language (дата звернення: 20.04.2025)

15. Діаграма варіантів використання. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 20.04.2025)

16. Модель «сутностей-зв'язків». URL: https://virt.ldubgd.edu.ua/pluginfile.php/38821/mod_resource/content/1/ (дата звернення: 20.04.2025)

17. Діаграма класів. URL: https://flexberry.github.io/ru/fd_class-diagram.html (дата звернення: 01.05.2022).

18. Діаграма послідовності. URL: https://flexberry.github.io/ru/fd_sequence-diagram.html (дата звернення: 02.05.2025).

19. Фізична модель даних. URL: https://stud.com.ua/93804/informatika/stvorennya_fizichnoyi_modeli_bazi_danih (дата звернення: 03.05.2025).

20. Про охорону праці: Закон України від 21.11.2002 р. № 229-IV. URL: https://zakon.rada.gov.ua/laws/main/2694-12#Text (дата звернення: 17.04.2025).

Додаток А – Технічне завдання на розробку програмного забезпечення телеграм-боту для відстеження дедлайнів з інтеграцією штучного інтелекту "DueMaster"

Вступ

Назва розробки – телеграм-бот "DueMaster" для відстеження дедлайнів з інтеграцією штучного інтелекту.

Підстави для розробки даного програмного забезпечення є наказ на затвердження тем кваліфікаційних робіт бакалаврів зі спеціальності 121 Інженерія програмного забезпечення в Національному університеті кораблебудування ім. адмірала Макарова.

1 Призначення розробки:

Програмне забезпечення телеграм-боту “DueMaster” призначене для ефективного та надійного функціонування для відстеження дедлайнів.

1.1 Експлуатаційне призначення

Експлуатаційним призначенням програмного забезпечення, що розробляється, є створення функціонального та ефективного телеграм-бота, щоб підвищити продуктивність та швидкість роботи користувачів, надійність і точність обробки дедлайнів, надання зручного та інтуїтивного інтерфейсу в межах месенджера Telegram.

1.2 Функціональне призначення

Функціональним призначенням програмного забезпечення, що розробляється, є автоматизація управління дедлайнами, включаючи розпізнавання завдань із тексту, планування нагадувань і генерацію персоналізованих порад.

Були сформульовані наступні функціональні вимоги до програмного забезпечення:

Для користувача:

– Реєстрація користувача не потрібна, але бот має ідентифікувати його за Telegram ID для збереження персональних дедлайнів.

– Додавання дедлайнів через текстові повідомлення (наприклад, "Завершити проєкт до 20 травня 15:00").

– Перегляд списку дедлайнів за командою "/list" із сортуванням за датою.

– Отримання нагадувань за заданий час до дедлайну

– Отримання мотиваційних порад, згенерованих штучним інтелектом, після додавання дедлайну чи за командою "/motivate".

Для розробника:

– Збереження дедлайнів у базі даних із можливістю їх редагування чи видалення.

– Налаштування частоти перевірки дедлайнів.

– Інтеграція моделі штучного інтелекту для генерації тексту з можливістю оновлення моделі в майбутньому

– Логування дій бота для відлагодження та аналізу помилок.

– Можливість масштабування функцій, наприклад, додавання групового режиму для командної роботи.

2 Вимоги до програмного забезпечення

2.1 Вимоги до функціональних характеристик

2.1.1 Вимоги до складу виконуваних функцій

Для користувача:

– Реєстрація користувача не потрібна, але бот має ідентифікувати його за Telegram ID для збереження персональних дедлайнів.

– Додавання дедлайнів через текстові повідомлення (наприклад, "Завершити проєкт до 20 травня 15:00").

– Перегляд списку дедлайнів за командою "/list" із сортуванням за датою.

– Отримання нагадувань за заданий час до дедлайну

– Отримання мотиваційних порад, згенерованих штучним інтелектом, після додавання дедлайну чи за командою "/motivate".

Для розробника:

– Збереження дедлайнів у базі даних із можливістю їх редагування чи видалення.

– Налаштування частоти перевірки дедлайнів.

– Інтеграція моделі штучного інтелекту для генерації тексту з можливістю оновлення моделі в майбутньому.

– Логування дій бота для відлагодження та аналізу помилок.

– Можливість масштабування функцій, наприклад, додавання групового режиму для командної роботи.

2.1.2 Вимоги до організаційних вхідних та вихідних даних:

Вхідні дані:

– Текстові повідомлення з дедлайнами: наприклад, "Здати курсову до п’ятниці" або "Зустріч 10 червня о 14:00".

– Команди: "/list" (перегляд дедлайнів), "/add" (додавання дедлайну), "/delete" (видалення дедлайну), "/motivate" (отримання поради)

– Налаштування користувача: вибір часу нагадувань.

– Вихідні дані:

– Підтвердження додавання дедлайну: "Дедлайн ‘Здати курсову’ збережено на п’ятницю, 15 травня 2025, 23:59".

– Список дедлайнів: "1. Здати курсову – 15.05.2025 23:59; 2. Зустріч – 10.06.2025 14:00".

– Нагадування: "Нагадування: завтра о 14:00 дедлайн для ‘Зустріч’!".

– Мотиваційні поради: "Ти на правильному шляху, головне – почати вчасно!".

2.2 Вимоги до надійності

Програмне забезпечення повинно функціонувати стабільно і без збоїв, щоб забезпечити безперебійну роботу телеграм-боту. При виникненні збоїв в роботі, відновлення нормальної роботи повинне проводитися після перезавантаження програми.

Телеграм-бот повинен продовжувати коректне функціонувати при втраті частини інформації. У випадку неможливого продовження коректної роботи

має бути повідомлення про це. Телеграм-бот повинен на регулярній основі створювати резервні копії баз даних.

У разі введення користувачем некоректної інформації телеграм-бот повинен повідомити про помилку і надати можливість виправити її.

2.3 Вимоги до умов експлуатації

Необхідний рівень підготовки користувачів: мінімальні навички в користуванні комп’ютером. Комп’ютер призначений для роботи в закритому опалювальному приміщені.

2.4 Вимоги до складу та параметрів технічних засобів

Мінімальні комплектація комп’ютеру:

– Процесор: AMD A10-5750m 2.5 GHZ 4 ядра.
– Оперативна пам’ять: 4 гб.
– Жорсткий диск: 2 гб.

Мінімальна комплектація смартфону:

– Версія Android 8.0 / iOS 12.4.9.
– ОЗУ 2 ГБ.
– 500 МБ вільної пам’яті.

2.5 Вимоги до інформаційної та програмної сумісності:

Запуск і робота програмного забезпечення здійснюється на операційній системі Windows 7/8/10 (32 або 64 біти) з використанням середовища Python та бібліотеки aiogram, що дозволяє створювати та розгортати телеграм-ботів локально на комп’ютері.

3 Вимоги до програмної документації

Програмна документація повинна містити:

– технічне завдання
– текст програми
– опис програми
– інструкцію користувача
– програму і методику випробувань ПЗ.

4 Стадії та етапи розробки

Стадії та етапи розробки представлені в таблиці А.1.

Таблиця А.1 – Стадії та етапи розробки програмного забезпечення

Стадії розробки	Етапи робіт	Дата завершення
Технічне завдання	Обумовлення необхідності розробки	07.04.2025
	Розробка та затвердження технічного завдання	14.04.2025
Ескізний проект	Розробка ескізного проекту	21.04.2025
	Затвердження ескізного проекту	05.04.2025
Технічний проект	Розробка технічного проекту	05.05.2025
	Затвердження технічного проекту	07.05.2025
Робочий проект	Розробка ПЗ	05.05.2025
	Розробка програмної документації	15.05.2025
	Випробування ПЗ	17.05.2025

5 Порядок контролю і приймання

Контроль за аналізом та проектуванням кожної окремої частини програмного забезпечення на кожному етапі з урахуванням вимог, визначених у технічному завдані. Кожна стадія розробки повинна бути представлена в зазначені строки та узгоджена із замовником.

Приймання проводяться відповідно до програми і методики випробувань і документують за допомогою протоколу проведення випробувань. У разі знаходження помилок під час прийому програмного виробу складається акт про знайдені помилки, який підписуються представниками замовника і розробника і затверджуються керівником організації – замовника та організації – розробника. Розробник повинен протягом не більше ніж двох тижнів виправити зазначені помилки і оповістити замовника про повторне проведення перевірки.

Додаток Б – Код програми

main.py

```
from src.bot.deadline_bot import DeadlineBot
import asyncio

if name == "main":
    bot = DeadlineBot()
asyncio.run(bot.run())
```

deadline_bot.py

```
import asyncio
import sys
import signal
import pendulum
from aiogram import Bot, Dispatcher, F
from aiogram.filters import CommandStart, Command
from aiogram.types import Message, BotCommand
from aiogram.client.default import DefaultBotProperties
from aiogram.enums import ParseMode
import sqlite3
from src.config import Config
from src.database import DeadlineDatabase
from src.services import BotLogger, Motivator
from src.parsers import DeadlineParser, ReminderParser,
Pluralizer

class DeadlineBot:
    def __init__(self):
        self.bot = Bot(token=Config.TOKEN,
default=DefaultBotProperties(parse_mode=ParseMode.HTML))
        self.dp = Dispatcher()
        self.logger = BotLogger()
        self.db = DeadlineDatabase()
        self.motivator = Motivator()
        self.parser = DeadlineParser()
        self.reminder_parser = ReminderParser()
        self.pluralizer = Pluralizer()
        self._register_handlers()
```

```
    async def set_commands(self):
        commands = [
            BotCommand(command="/start",
description="Початок роботи"),
            BotCommand(command="/help",
description="Довідка"),
            BotCommand(command="/add", description="Додати
дедлайн"),
            BotCommand(command="/list", description="Список
дедлайнів"),
            BotCommand(command="/edit",
description="Редагувати дедлайн"),
            BotCommand(command="/delete",
description="Видалити дедлайн"),
            BotCommand(command="/motivate",
description="Мотиваційна цитата"),
            BotCommand(command="/reminder",
description="Налаштувати нагадування"),
            BotCommand(command="/daily",
description="Щоденний огляд"),
            BotCommand(command="/overdue",
description="Прострочені дедлайни"),
            BotCommand(command="/delete_all",
description="Видалити всі дедлайни"),
            BotCommand(command="/clear_overdue",
description="Видалити прострочені"),
        ]
        await self.bot.set_my_commands(commands)
        description = (
            "Deadline Helper Bot.\n"
            "Керуйте своїми дедлайнами легко та
ефективно!\n"
            "□ Додавайте, редагуйте та відстежуйте
дедлайни.\n"
            "□ Отримуйте нагадування.\n"
            "□ Мотивуйте себе цитатами.\n"
            "Все безкоштовно!"
        )
        await self.bot.set_my_description(description)
```

```
    async def show_reminder_help(self, message: Message):
        await message.answer(
            "  Вкажіть номер дедлайну та час нагадування:\n"
            "<code>/reminder [номер] [час]</code>\n"
            "Приклад: /reminder 1 за 30 хвилин"
        )

    async def send_reminder(self, user_id: int, deadline_id:
int, title: str, deadline_dt: pendulum.DateTime):
        try:
            now = pendulum.now(Config.TIMEZONE)
            time_left = deadline_dt - now
            hours_left = int(time_left.total_hours())
            minutes_left = int(time_left.total_minutes() %
60)
            time_str = ""
            if hours_left > 0:
                time_str += f"{hours_left} год. "
            time_str += f"{minutes_left} хв."
            motivation = await
self.motivator.get_motivation("")
            message = (
                f"  <b>Нагадування!</b>\n\n"
                f"Дедлайн '<b>{title}</b>' через
<b>{time_str}</b>!\n"
                f"Час завершення:
{deadline_dt.strftime('%d.%m.%Y %H:%M')}\n\n"
                f"  <b>Мотивація:</b>\n{motivation}"
            )
            await self.bot.send_message(user_id, message)
            self.db.mark_deadline_notified(deadline_id)
        except Exception as e:
            self.logger.error(f"Помилка відправки
нагадування: {e}")

    async def show_all_deadlines(self, user_id: int, daily:
bool = False):
        active = self.db.get_user_deadlines(user_id)
        overdue = self.db.get_overdue_deadlines(user_id)
```

```
        if not active and not overdue:
            message = "У вас немає дедлайнів. Додайте перший
за допомогою /add!"
            if daily:
                return
            await self.bot.send_message(user_id, message)
            return
        response = "  <b>Ваші дедлайни:</b>\n\n"
        if active:
            response += "  <b>Активні:</b>\n"
            for idx, deadline in enumerate(active, 1):
                _, title, deadline_datetime, reminder_time =
deadline
                try:
                    dt = pendulum.parse(deadline_datetime,
tz=Config.TIMEZONE)
                    formatted_date = dt.strftime("%d.%m.%Y
%H:%M")
                    response += f"{idx}. <b>{title}</b> -
{formatted_date}\n"
                except ValueError as e:
                    self.logger.error(f"Помилка парсингу
дати {deadline_datetime}: {e}")
                    continue
        if overdue:
            response += "\n  <b>Прострочені (щоб видалити -
<code>/clear_overdue</code>):</b>\n"
            for idx, deadline in enumerate(overdue, 1):
                _, title, deadline_datetime, _ = deadline
                try:
                    dt = pendulum.parse(deadline_datetime,
tz=Config.TIMEZONE)
                    formatted_date = dt.strftime("%d.%m.%Y
%H:%M")
                    response += f"{idx}. <b>{title}</b> -
{formatted_date} (прострочено)\n"
                except ValueError as e:
                    self.logger.error(f"Помилка парсингу
дати {deadline_datetime}: {e}")
                    continue
```

```
        if not daily:
            response += "\nЩоб видалити всі прострочені
дедлайни, використайте /clear_overdue"
        await self.bot.send_message(user_id, response)

    def _register_handlers(self):
        @self.dp.message(CommandStart())
        async def start_handler(message: Message):
            await message.answer("  Вітаю! Я бот для
керування дедлайнами.\n\nВикористовуйте /add для додавання
дедлайнів, /list для перегляду, або /help для довідки.")

        @self.dp.message(Command("help"))
        async def help_handler(message: Message):
            response = (
                "  <b>Довідка по командах:</b>\n\n"
                "/start - Почати роботу\n"
                "/help - Показати цю довідку\n"
                "/add - Додати дедлайн (напр., /add Купити
пиво через годину)\n"
                "/list - Показати активні дедлайни\n"
                "/edit - Редагувати дедлайн (вкажіть номер і
нові дані)\n"
                "/delete - Видалити дедлайн (вкажіть
номер)\n"
                "/motivate - Отримати мотиваційну цитату\n"
                "/reminder - Налаштувати нагадування\n"
                "/daily - Показати щоденний огляд
дедлайнів\n"
                "/overdue - Показати прострочені дедлайни\n"
                "/delete_all - Видалити всі дедлайни\n"
                "/clear_overdue - Видалити прострочені
дедлайни"
            )
            await message.answer(response)

        @self.dp.message(Command("add"))
        async def add_deadline_handler(message: Message):
            text = message.text.replace('/add', '').strip()
            if not text:
```

```
                await message.answer(
                    "  Введіть дедлайн у
форматі:\n<code>/add [задача] [коли]</code>\n\n"
                    "Приклади:\n• Купити пиво через
годину\n• Зустріч о 14:00 завтра\n"
                    "• Тестування 20.06 о 10:30\n• Купити
пиво (без часу - буде сьогодні 23:59)"
                )
                return
            title, deadline =
self.parser.parse_deadline_text(text)
            if not deadline:
                await message.answer(
                    "❌ Не вдалося розпізнати дату.
Спробуйте:\n<code>/add Назва до дати/часу</code>\n"
                    "Приклади:\n• Купити пиво через 2
години\n• Зустріч 20 червня\n"
                    "• Тест завтра о 15:00\n• Купити пиво
(без часу - буде сьогодні 23:59)"
                )
                return
            if not self.parser.is_future_datetime(deadline):
                await message.answer("❌ Не можна
встановлювати дедлайн у минулому часі. Будь ласка, вкажіть
майбутню дату/час.")
                return
            if self.db.add_deadline(message.from_user.id,
title, deadline):
                await message.answer(
                    f"✅ Дедлайн '<b>{title}</b>' збережено
на <b>{deadline.strftime('%d.%m.%Y %H:%M')}</b>\n"
                    "Щоб переглянути всі дедлайни,
використайте /list"
                )
            else:
                await message.answer("❌ Помилка при
додаванні дедлайну")

        @self.dp.message(Command("list"))
        async def list_deadlines_handler(message: Message):
```

```
            deadlines =
self.db.get_user_deadlines(message.from_user.id)
            if not deadlines:
                await message.answer("У вас немає активних
дедлайнів. Додайте перший за допомогою /add!")
                return
            response = "📋 Ваші дедлайни:\n\n"
            for idx, deadline in enumerate(deadlines, 1):
                _, title, deadline_datetime, reminder_time =
deadline
                try:
                    dt = pendulum.parse(deadline_datetime,
tz=Config.TIMEZONE)
                    formatted_date = dt.strftime("%d.%m.%Y
%H:%M")
                    response += f"{idx}. <b>{title}</b> -
{formatted_date} (Нагадування: {reminder_time or 'не
встановлено'})\n"
                except ValueError as e:
                    self.logger.error(f"Помилка парсингу
дати {deadline_datetime}: {e}")
                    continue
            await message.answer(response)

        @self.dp.message(Command("edit"))
        async def edit_handler(message: Message):
            text = message.text.replace('/edit', '').strip()
            if not text:
                await message.answer(
                    "📝 Вкажіть номер дедлайну та нові
дані:\n<code>/edit [номер] [нова назва] [новий
час]</code>\n"
                    "Приклад: /edit 1 Купити пиво завтра о
15:00"
                )
                return
            try:
                parts = text.split(maxsplit=1)
                deadline_pos = int(parts[0]) - 1
                if len(parts) < 2:
```

```
                    await message.answer("❌ Вкажіть нову
назву та/або час.")
                    return
                deadlines =
self.db.get_user_deadlines(message.from_user.id)
                if not deadlines or deadline_pos < 0 or
deadline_pos >= len(deadlines):
                    await message.answer(f"❌ Дедлайн з
номером {deadline_pos+1} не знайдено. Використайте /list для
перегляду.")
                    return
                deadline_id = deadlines[deadline_pos][0]
                new_text = parts[1].strip()
                title, deadline =
self.parser.parse_deadline_text(new_text)
                if not deadline:
                    await message.answer("❌ Не вдалося
розпізнати дату. Спробуйте:\n<code>/edit [номер] Назва до
дати/часу</code>")
                    return
                if not
self.parser.is_future_datetime(deadline):
                    await message.answer("❌ Не можна
встановлювати дедлайн у минулому часі.")
                    return
                if
self.db.update_deadline(message.from_user.id, deadline_id,
title, deadline):
                    await message.answer(
                        f"✅ Дедлайн #{deadline_pos+1}
оновлено: '<b>{title}</b>' на
<b>{deadline.strftime('%d.%m.%Y %H:%M')}</b>"
                    )
                else:
                    await message.answer("❌ Помилка при
оновленні дедлайну. Спробуйте ще раз.")
            except ValueError:
                await message.answer("❌ Номер дедлайну має
бути числом.")
```

```
        @self.dp.message(Command("delete"))
        async def delete_handler(message: Message):
            text = message.text.replace('/delete',
'').strip()
            if not text:
                await message.answer("❌ Вкажіть номер
дедлайну для видалення:\n<code>/delete
[номер]</code>\nПриклад: /delete 1")
                return
            try:
                deadline_pos = int(text) - 1
                deadlines =
self.db.get_user_deadlines(message.from_user.id)
                if not deadlines or deadline_pos < 0 or
deadline_pos >= len(deadlines):
                    await message.answer(f"❌ Дедлайн з
номером {deadline_pos+1} не знайдено.")
                    return
                deadline_id = deadlines[deadline_pos][0]
                if
self.db.delete_deadline(message.from_user.id, deadline_id):
                    await message.answer(f"✅ Дедлайн
#{deadline_pos+1} успішно видалено!")
                else:
                    await message.answer("❌ Помилка при
видаленні дедлайну.")
            except ValueError:
                await message.answer("❌ Номер дедлайну має
бути числом.")

        @self.dp.message(Command("motivate"))
        async def motivate_handler(message: Message):
            motivation = await
self.motivator.get_motivation("")
            await message.answer(f"
<b>Мотивація:</b>\n{motivation}")

        @self.dp.message(Command("reminder"))
        async def reminder_handler(message: Message):
            text = message.text.replace('/reminder',
```

```
').strip()
            if not text:
                await self.show_reminder_help(message)
                return
            try:
                parts = text.split(maxsplit=1)
                if len(parts) < 2:
                    await message.answer("❌ Вкажіть номер
дедлайну та час нагадування.")
                    return
                deadline_pos = int(parts[0]) - 1
                reminder_text = parts[1].strip()
                deadlines =
self.db.get_user_deadlines(message.from_user.id)
                if not deadlines or deadline_pos < 0 or
deadline_pos >= len(deadlines):
                    await message.answer(f"❌ Дедлайн з
номером {deadline_pos+1} не знайдено. Використайте /list для
перегляду.")
                    return
                deadline_id = deadlines[deadline_pos][0]
                reminder_time =
self.reminder_parser.parse_reminder_time(reminder_text)
                if reminder_time is None:
                    await message.answer("❌ Не вдалося
розпізнати час нагадування. Спробуйте: 30 хвилин, 1 година,
2 дні.")
                    return
                if
self.db.update_deadline(message.from_user.id, deadline_id,
reminder_time=reminder_time):
                    await message.answer(f"✅ Нагадування для
дедлайну #{deadline_pos+1} встановлено за {reminder_text} до
дедлайну.")
                else:
                    await message.answer("❌ Не вдалося
встановити нагадування. Перевірте номер дедлайну.")
            except ValueError:
                await message.answer("❌ Номер дедлайну має
бути числом.")
```

```
        @self.dp.message(Command("daily"))
        async def daily_handler(message: Message):
            await
self.show_all_deadlines(message.from_user.id, daily=True)

        @self.dp.message(Command("overdue"))
        async def list_overdue_handler(message: Message):
            deadlines =
self.db.get_overdue_deadlines(message.from_user.id)
            if not deadlines:
                await message.answer("У вас немає
прострочених дедлайнів! ")
                return
            response = " <b>Ваші прострочені
дедлайни:</b>\n\n"
            for idx, deadline in enumerate(deadlines, 1):
                _, title, deadline_datetime, _ = deadline
                try:
                    dt = pendulum.parse(deadline_datetime,
tz=Config.TIMEZONE)
                    formatted_date = dt.strftime("%d.%m.%Y
%H:%M")
                    response += f"{idx}. <b>{title}</b> -
{formatted_date} (прострочено)\n"
                except ValueError as e:
                    self.logger.error(f"Помилка парсингу
дати {deadline_datetime}: {e}")
                    continue
            response += "\nЩоб видалити всі прострочені
дедлайни, використайте /clear_overdue"
            await message.answer(response)

        @self.dp.message(Command("delete_all"))
        async def delete_all_handler(message: Message):
            if
self.db.delete_all_deadlines(message.from_user.id):
                await message.answer("✅ Всі ваші дедлайни
було видалено!")
            else:
```

```
                await message.answer("❌ Сталася помилка при
видаленні дедлайнів")

        @self.dp.message(Command("clear_overdue"))
        async def clear_overdue_handler(message: Message):
            deleted_count =
self.db.delete_overdue_deadlines(message.from_user.id)
            if deleted_count > 0:
                await message.answer(f"✅ Видалено
{deleted_count} прострочених дедлайнів!")
            else:
                await message.answer("ℹ️ У вас немає
прострочених дедлайнів для видалення.")

        @self.dp.message()
        async def handle_unknown_commands(message: Message):
            if message.text and
message.text.startswith('/'):
                await message.answer("❌ Невідома команда.
Використайте /help для списку доступних команд.")
            else:
                await message.answer("🤖 Я бот для керування
дедлайнами. Використайте /help для довідки.")

        @self.dp.message(F.sticker)
        async def handle_sticker(message: Message):
            await message.answer("🎭 Ви надіслали стікер. Я
бот для керування дедлайнами. Використайте /add для
додавання дедлайнів або /help для довідки.")

        @self.dp.message(F.location)
        async def handle_location(message: Message):
            await message.answer("📍 Ви надіслали геолокацію.
Я бот для керування дедлайнами. Використайте /add для
додавання дедлайнів або /help для довідки.")

        @self.dp.message(F.voice)
        async def handle_voice(message: Message):
            await message.answer("🎤 Ви надіслали голосове
повідомлення. Я бот для керування дедлайнами. Використайте
```

```
/add для додавання дедлайнів або /help для довідки.")

        @self.dp.message(F.audio)
        async def handle_audio(message: Message):
            await message.answer("🎵 Ви надіслали аудіофайл.
Я бот для керування дедлайнами. Використайте /add для
додавання дедлайнів або /help для довідки.")

        @self.dp.message(F.video)
        async def handle_video(message: Message):
            await message.answer("🎥 Ви надіслали відео. Я
бот для керування дедлайнами. Використайте /add для
додавання дедлайнів або /help для довідки.")

        @self.dp.message(F.document)
        async def handle_document(message: Message):
            await message.answer("📄 Ви надіслали документ. Я
бот для керування дедлайнами. Використайте /add для
додавання дедлайнів або /help для довідки.")

        @self.dp.message(F.photo)
        async def handle_photo(message: Message):
            await message.answer("🖼 Ви надіслали фотографію.
Я бот для керування дедлайнами. Використайте /add для
додавання дедлайнів або /help для довідки.")

    async def check_reminders(self):
        while True:
            try:
                reminders = self.db.get_due_reminders()
                sent_reminders = set()
                for reminder in reminders:
                    deadline_id, user_id, title,
deadline_datetime = reminder
                    if deadline_id in sent_reminders:
                        continue
                    try:
                        deadline_dt =
pendulum.parse(deadline_datetime, tz=Config.TIMEZONE)
                        await self.send_reminder(user_id,
```

```
deadline_id, title, deadline_dt)
                        sent_reminders.add(deadline_id)
                    except ValueError as e:
                        self.logger.error(f"Помилка парсингу
дати {deadline_datetime}: {e}")
                        continue
                await asyncio.sleep(30)
            except Exception as e:
                self.logger.error(f"Помилка в
check_reminders: {e}")
                await asyncio.sleep(60)

    async def daily_check(self):
        while True:
            try:
                now = pendulum.now(Config.TIMEZONE)
                if now.hour == 9 and now.minute == 0:
                    conn = sqlite3.connect(self.db.db_name)
                    cursor = conn.cursor()
                    cursor.execute("SELECT DISTINCT user_id
FROM deadlines")
                    users = cursor.fetchall()
                    for user_id in users:
                        try:
                            await
self.show_all_deadlines(user_id[0], daily=True)
                        except Exception as e:
                            self.logger.error(f"Помилка
відправки щоденного огляду для {user_id[0]}: {e}")
                    conn.close()
                    await asyncio.sleep(3600)
                else:
                    await asyncio.sleep(60)
            except Exception as e:
                self.logger.error(f"Помилка в daily_check:
{e}")
                await asyncio.sleep(60)

    async def shutdown_after_cleanup(self, tasks):
        try:
```

```
            await asyncio.gather(*tasks)
            print("✅ Робота завершена")
            sys.exit(0)
        except Exception as e:
            self.logger.error(f"Помилка під час завершення:
{e}")
            sys.exit(1)

    def graceful_shutdown(self, signum, frame):
        print("\n🛑 Завершення роботи...")
        loop = asyncio.get_event_loop()
        tasks = [
            loop.create_task(self.dp.stop_polling()),

loop.create_task(self.motivator.close_session()),
            loop.create_task(self.bot.session.close())
        ]
        loop.create_task(self.shutdown_after_cleanup(tasks))

    async def run(self):
        signal.signal(signal.SIGINT, self.graceful_shutdown)
        signal.signal(signal.SIGTERM,
self.graceful_shutdown)
        await self.set_commands()
        await self.motivator.init_session()
        self.db.init_db()
        asyncio.create_task(self.check_reminders())
        asyncio.create_task(self.daily_check())
        print("🤖 Бот запущений. Натисніть Ctrl+C для
зупинки")
        try:
            await self.dp.start_polling(self.bot)
        finally:
            await self.motivator.close_session()
            await self.bot.session.close()
            print("✅ Робота завершена")
```

Додаток В – Програма та методика випробовування ПЗ

1 Об’єкт випробовувань

Об’єктом випробувань є програмне забезпечення телеграм-боту “DueMaster” для відстеження дедлайнів з інтеграцією штучного інтелекту.

Область застосування ПЗ – надання користувачам інформації про дедлайни та нагадування, інтегровані з можливостями штучного інтелекту.

2 Мета випробувань

Перевірка відповідності характеристик та вимог розробленої програми, викладених у технічному завданні, перевірка працездатності даного програмного забезпечення, приведення прикладу роботи та отримання відповідних навичок.

3 Вимоги до застосунку

При проведенні випробувань, можливості програми підлягають перевірці на відповідність вимогам, викладених у пункті «Вимоги до функціональних характеристик» технічного завдання.

4 Вимоги до програмної документації

Програмна документація повинна містити:

- технічне завдання
- текст програми
- програму і методику випробувань ПЗ.
- опис програми

– інструкцію користувача

5 Склад та порядок випробувань

Вимоги до складу та параметрів технічних і програмних засобів вказані в Додатку А.

Процес виконання випробувань включає такі етапи:

– проводяться контрольні перевірки в довільній послідовності;

– здійснюється аналіз отриманих даних із визначенням відповідності програмного продукту встановленим вимогам і технічній документації.

У разі виявлення збоїв у роботі системи формується список проблем, після чого узгоджується з розробником строк їх усунення. Надалі замовник проводить повторне тестування в повному обсязі, допускаючи при цьому використання оновлених або додаткових тестових сценаріїв.

6 Методи випробувань

Методом оцінки функціональності створеного програмного забезпечення слугує тестування типу "чорна скринька", яке передбачає запуск програми для виявлення можливих недоліків. Послідовність дій визначається за допомогою спеціально розроблених тестових кейсів.

У таблиці В.1, що наведена нижче, вказано перелік випробувань.

Таблиця В.1 – Методика випробування

№	Дія	Очікуваний результат	Результат перевірки	Зауваження
1	2	3	4	5
1	Перевірка коректності обробки запиту при додаванні дедлайну з неправильним номером місяця.	Програма має провести валідацію запиту на відповідність формату		

Продовження таблиці В.1

1	2	3	4	5
2	Перевірка коректності обробки запиту при додаванні дедлайну без специфікації дати	Програма має провести валідацію запиту на відповідність формату		
3	Перевірка коректності обробки запиту при додаванні дедлайну з минулою датою	Програма має провести валідацію запиту на відповідність формату		
4	Перевірка коректності обробки запиту при додаванні дедлайну без специфікації будь-яких даних	Програма має провести валідацію запиту на відповідність формату		
5	Перевірка коректності обробки запиту при додаванні дедлайну з неіснуючою датою	Програма має провести валідацію запиту на відповідність формату		

Додаток Г – Опис програми

1 Загальні відомості

Основною метою програми є автоматизація процесу планування та контролю дедлайнів за допомогою телеграм-бота "DueMaster", що сприяє зручному та ефективному управлінню часом.

Розробка призначена для збереження, перегляду, редагування та видалення даних про дедлайни та нагадування в базі даних, а також для інтеграції мотиваційних порад, створених штучним інтелектом, у межах платформи Telegram.

2 Функціональне призначення

Програмне забезпечення має забезпечувати виконання таких функцій:

– реєстрація користувача: ідентифікація користувача за Telegram ID без необхідності окремої реєстрації для зберігання особистих даних;

– авторизація користувача: верифікація через унікальний ідентифікатор Telegram;

– робота з даними:

1) Дедлайни:

– додавання нового дедлайну (наприклад, "Завершити задачу до 20 червня 15:00");

– пошук даних за критеріями (наприклад, дата чи назва дедлайну);

– редагування інформації про дедлайн (зміна дати чи опису);

– видалення дедлайну з бази.

2) Нагадування:

– налаштування нагадувань (наприклад, за день до дедлайну);

– перегляд списку нагадувань;

– редагування часу нагадувань;

– видалення непотрібних нагадувань.

3 Опис логічної структури

3.1 Структура програми

Програмне забезпечення телеграм-бота "DueMaster" складається з таких основних компонентів:

– сервер, на якому виконується скрипт бота.

– платформа Telegram, що забезпечує зв’язок із ботом через Telegram Bot API.

– пристрій користувача з встановленим додатком Telegram та активним підключенням до інтернету.

3.2 Алгоритм програми

Програмне забезпечення "DueMaster" функціонує за моделлю «клієнт-сервер». Тут клієнтом є додаток Телеграм, а сервером виступає комп’ютер із запущеним скриптом бота. Процес роботи системи включає такі етапи:

1) Користувач взаємодіє з ботом через інтерфейс Телеграм.

2) Телеграм передає запити до сервера через інтернет.

3) Сервер аналізує отриманий запит.

4) Скрипт обробляє дані та формує відповідь.

5) Телеграм відображає результат у вигляді повідомлень від бота.

3.3 Використовувані методи

Розробка телеграм-бота "DueMaster" для відстеження дедлайнів виконана мовою програмування Python 3. Як інструмент для створення коду рекомендовано використовувати середовище розробки PyCharm.

4 Технічні та програмні засоби

Мінімальна комплектація комп’ютеру:

– Процесор: AMD A10-5750m 2.5 GHZ 4 ядра.

– Оперативна пам’ять: 4 гб.

– Жорсткий диск: 2 гб.

Мінімальна комплектація смартфону:

– Версія Android 8.0 / iOS 12.4.9.

– ОЗУ 2 ГБ.

– 500 МБ вільної пам’яті.

Запуск і робота програмного забезпечення здійснюється на операційній системі Windows 7/8/10 (32 або 64 біти) з використанням середовища Python та бібліотеки aiogram, що дозволяє створювати та розгортати телеграм-ботів локально на комп’ютері.

Додаток Д – Інструкція користувача

Як тільки користувач здійснює надсилання команди /start, бот у відповідь надсилає вітальне повідомлення, в якому зазначені підказки, як працювати з ботом, що зображено на рисунку Д.1.

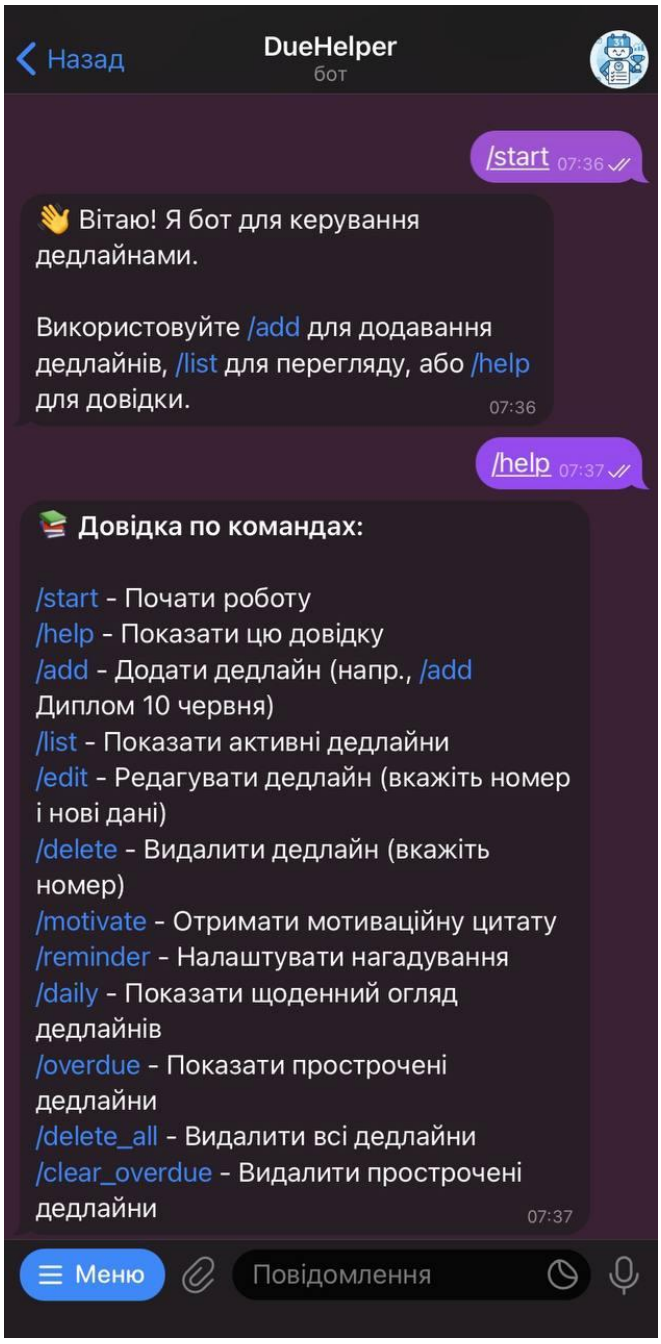

Рисунок Д.1 – Початок роботи

Додати дедлайн можливо надіславши команду /add [текст задачі] [дата виконання]. Після чого бот надішле повідомлення: “Дедлайн “текст задачі” збережено на [дата]”. Щоб переглянути усі дедлайни, використайте /list”. Щоб переглянути список усіх створених дедлайнів, дата виконання яких ще не настала, користувач надсилає команду /list. Після чого бот надсилає повідомлення з усіми активними дедлайнами для цього користувач. Зображено на рисунку Д.2.

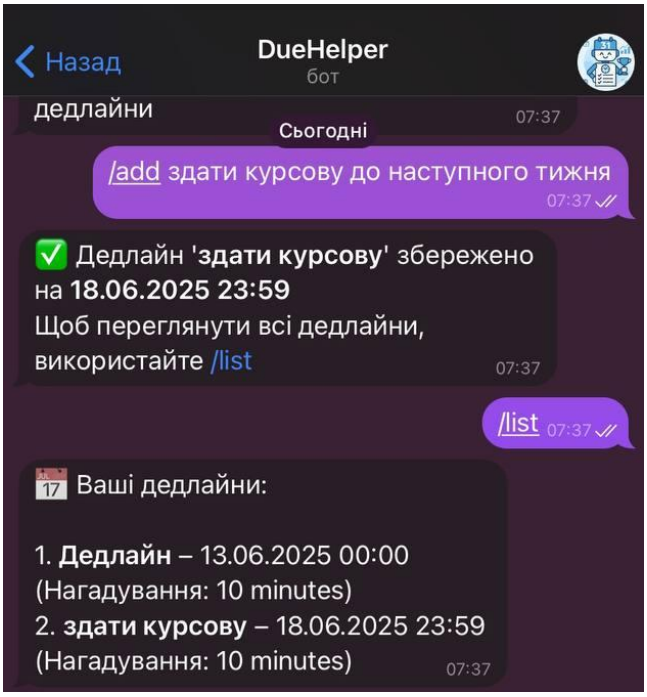

Рисунок Д.2 – Додавання дедлайну командою /add та перегляд активних дедлайнів командою /list

Щоб змінити час нагадування для певного дедлайну, користувач має надіслати команду /reminer [номер дедлайну] [проміжок часу]. Після чого бот надсилає повідомлення: “Нагадування для дедлайну [номер дедлайну] встановлена за [проміжок часу] до дедлайну.”. Зображено на рисунку Д.3.

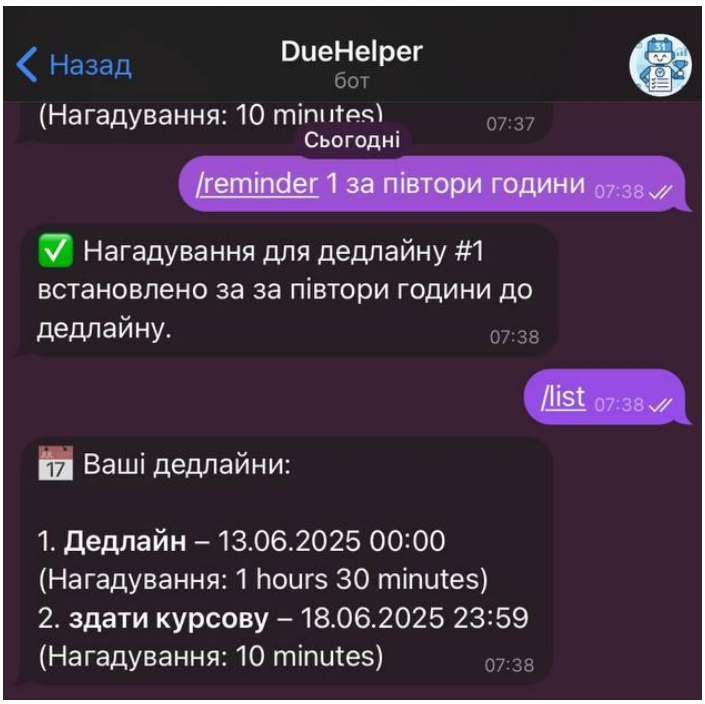

Рисунок Д.3 – Встановлення часу нагадування для дедлайну командою /reminder

Користувач може відредагувати будь-який активний дедлайн, змінивши як його текст, так і його назву, командою /edit [номер дедлайну] [нова назва] [нова дата]. У відповідь на цей запит бот надішле підтвердження: “Дедлайн [номер] оновлено: ‘[нова назва] на [нова дата]. Зображено на рисунку Д.4.

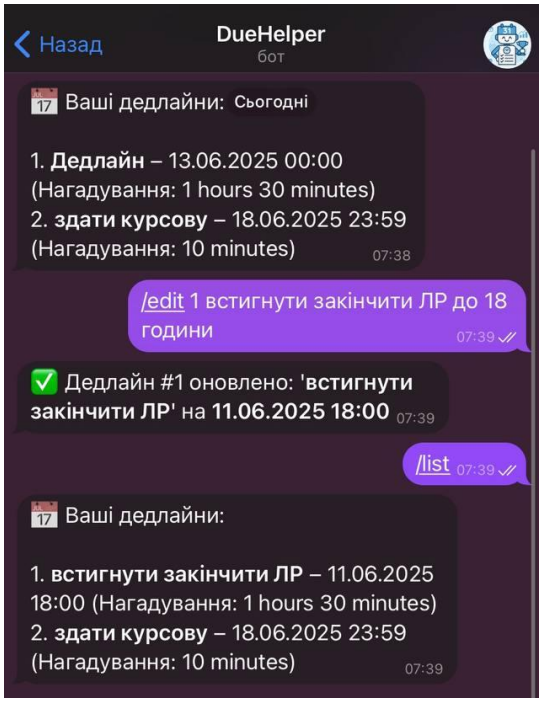

Рисунок Д.4 – Редагування дедлайну командою /edit.

Користувач може згенерувати та отримати мотиваційну пораду від штучного інтелекту, надіславши боту команду ‘/motivate’. Після чого бот надішле повідомлення, що містить мотиваційну пораду, яка була згенеровано штучним інтелектом. Також командою /daily користувач може переглянути усі активні дедлайни, які необхідно закінчити сьогодні. Зображено на рисунку Д.5.

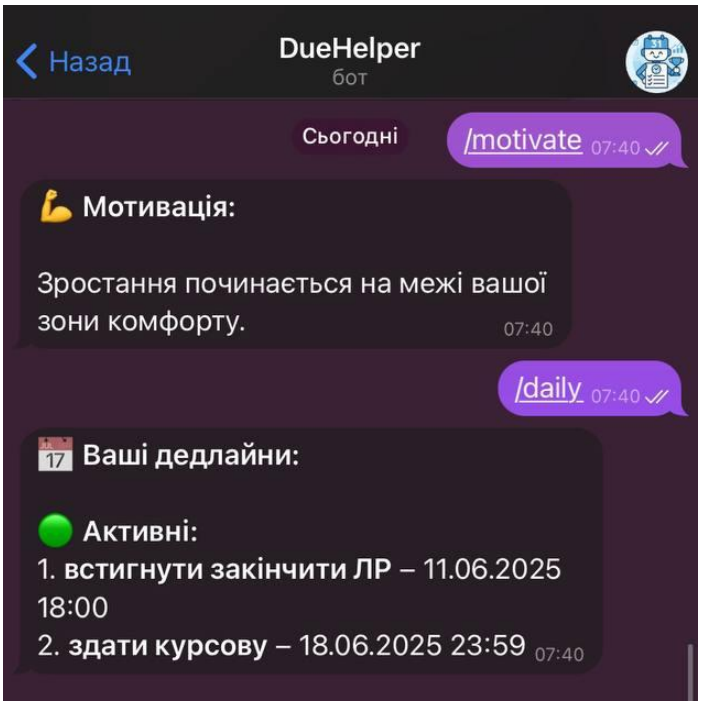

Рисунок Д.5 – Отримання мотиваційної поради командою /motivate та перегляд активних дедлайнів на сьогодні /daily

Користувач має можливість переглянути та видалити усі прострочені дедлайни командами /overdue та /clear_overdue. У разі відсутності прострочених дедлайнів бот сповістить про це користувача. Також командою /delete_all користувач може видалити усі активні дедлайни. Зображено на рисунку Д.6.

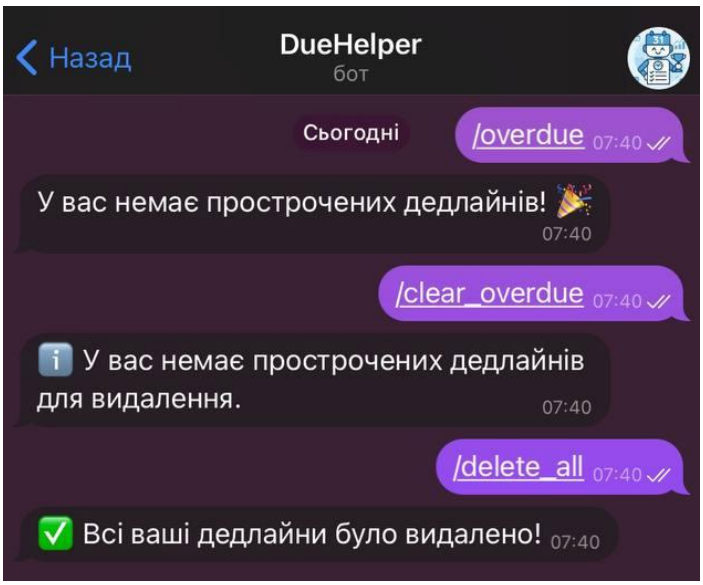

Рисунок Д.6 – Перегляд усіх прострочених дедлайнів командою /overdue та їх подальше видалення командою /clear_overdue, видалення усіх дедлайнів командою /delete_all