

**Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення**

**КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»**

**РОЗРОБКА МОДУЛЬНОЇ ІНТЕГРАЦІЙНОЇ СИСТЕМИ НА ОСНОВІ
TELEGRAM-БОТА З AI ДЛЯ ВЗАЄМОДІЇ З КОРПОРАТИВНИМИ
СЕРВІСАМИ ТА АВТОМАТИЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ
ВИКОРИСТОВУЮЧИ PYTHON**

**DEVELOPMENT OF A MODULAR INTEGRATION SYSTEM BASED ON
TELEGRAM BOT WITH AI FOR INTERACTION WITH CORPORATE
SERVICES AND AUTOMATION OF BUSINESS PROCESSES USING
PYTHON**

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
**Свиржевський Максим
Анатолійович**

Керівник:
Ящук Андрій Анатолійович

Кваліфікаційну роботу
допущено до захисту
«__» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти: бакалавр

Галузь знань: 12 Інформаційні технології

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

«__» _____ 20__ р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Свиржеському Максиму Анатолійовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: «Розробка модульної інтеграційної системи на основі Telegram-бота з AI для взаємодії з корпоративними сервісами та автоматизації бізнес-процесів використовуючи Python».

Керівник роботи: Ящук Андрій Анатолійович

затверджені наказом закладу вищої освіти від «31» грудня 2025 р. № 541/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до роботи: функціональні вимоги до програмного продукту, технічні обмеження, вимоги до програмного середовища та апаратного забезпечення.

4. Зміст розрахунково-пояснювальної записки: аналіз предметної області, постановка завдання, формування вимог до Telegram-бота, проєктування архітектури та діаграм, обґрунтування вибору технологій, розробка структури бази даних, реалізація функціоналу бота, забезпечення захисту даних, тестування та налагодження системи, підготовка документації.

5. Перелік графічного матеріалу: презентація, 16 рисунків, 7 лістингів, 2 таблиці

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
<i>Аналіз предметної області</i>	<i>Ящук А. А.</i>		
<i>Специфікація вимог до розроблювальної системи</i>	<i>Ящук А. А.</i>		
<i>Розробка об'єкта проектування</i>	<i>Ящук А. А.</i>		
<i>Нормоконтроль</i>	<i>Суринович О. М.</i>		
<i>Гарант ОП</i>	<i>Ліщина Н. М.</i>		
<i>Показник запозичень тексту</i>		%	
<i>Академічна доброчесність</i>	<i>Ящук А. А.</i>		

7. Дата видачі завдання «3» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/ п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	<i>Огляд літературних джерел по темі кваліфікаційної роботи бакалавра</i>	<i>до 14.02.2026 р.</i>	
2	<i>Аналіз проблеми розробки та впровадження об'єкту проектування</i>	<i>до 03.03.2026 р.</i>	
3	<i>Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання</i>	<i>до 14.03.2026 р.</i>	
4	<i>Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних</i>	<i>до 14.04.2026 р.</i>	
5	<i>Практична реалізація об'єкта проектування та розробка бази даних</i>	<i>до 20.04.2026 р.</i>	
6	<i>Тестування та налагодження об'єкта проектування</i>	<i>до 04.05.2026 р.</i>	
7	<i>Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі</i>	<i>до 29.05.2026 р.</i>	

Здобувач вищої освіти

Максим СВІРЖЕВСЬКИЙ

Керівник роботи

Андрій ЯЩУК

АНОТАЦІЯ

Свиржевський М. А. Розробка модульної інтеграційної системи на основі Telegram-бота з AI для взаємодії з корпоративними сервісами та автоматизації бізнес-процесів використовуючи Python. Рукопис.

Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків.

У першому розділі розглянуто предметну область автоматизації робочих процесів, чат-боти та AI-асистентів. У другому розділі сформульовано вимоги й описано архітектуру системи. У третьому розділі наведено практичну реалізацію Telegram-бота, workflow-рушія, Telegram-джерела та інтеграцій.

У висновках було підбито підсумки кожного етапу проведеної роботи над кваліфікаційною роботою бакалавра.

Ключові слова: Telegram-бот, штучний інтелект, workflow, API, Gemini, Redis, MySQL, aiogram, aiohttp, GooseCRM, Gmail, Google Calendar, Docker.

ABSTRACT

Svirzhevskyi M. A. Development of a modular integration system based on a Telegram Bot with AI for interaction with corporate services and automation of business processes using Python.

Bachelor's Qualification Work in the study program «Software Engineering». Lutsk National Technical University. Lutsk, 2026.

The work presents the design and implementation of AutomationBot, an intelligent Telegram bot that automates work processes by combining webhook-based message processing, workflow execution, AI agents, Redis state management, MySQL logging, and integrations with external APIs such as GooseCRM, Gmail, Google Calendar, Jira, and Worksection.

The conclusions summarize the results of each stage of work on this bachelor's qualification project.

Keywords: Telegram bot, artificial intelligence, workflow, API, Gemini, Redis, MySQL, aiogram, aiohttp, GooseCRM, Gmail, Google Calendar, Docker.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ РОЗРОБКИ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ	9
1.1 Аналіз сучасного стану проблеми.....	9
1.2 Постановка завдання на кваліфікаційну роботу бакалавра.....	12
Висновки до розділу 1	13
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ....	15
2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення	15
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання	22
Висновки до розділу 2.....	24
РОЗДІЛ 3 РОЗРОБКА МОДУЛЬНОЇ ІНТЕГРАЦІЙНОЇ СИСТЕМИ НА ОСНОВІ TELEGRAM-БОТА З AI ДЛЯ ВЗАЄМОДІЇ З КОРПОРАТИВНИМИ СЕРВІСАМИ ТА АВТОМАТИЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ ВИКОРИСТОВУЮЧИ PYTHON.....	26
3.1 Практична реалізація об'єкта проектування.....	26
3.2 Тестування та налагодження інформаційно-комп'ютерної системи.....	38
3.3 Проектування структури бази даних	40
3.4 Захист інформаційно-комп'ютерної системи	43
3.5 Розробка документації для програмного продукту.....	45
Висновки до розділу 3.....	47
ВИСНОВКИ	48
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	50
ДОДАТКИ.....	52

ВСТУП

Актуальність теми. Сучасні організації використовують значну кількість цифрових сервісів для керування задачами, комунікації, планування робочого часу, ведення клієнтських карток та контролю виконання доручень. На практиці це означає, що користувач змушений постійно перемикатися між месенджером, CRM, поштовим клієнтом, календарем, таск-трекером і внутрішніми вебсервісами. Така фрагментація робочого середовища знижує швидкість прийняття рішень і збільшує ризик втрати важливої інформації.

Особливо помітною ця проблема є в невеликих командах, де одна людина може виконувати ролі розробника, менеджера, аналітика або технічного спеціаліста підтримки. Частина робочих дій є повторюваною: створити задачу за повідомленням, перевірити власні задачі, знайти листи за дату, переглянути зустрічі, уточнити статус заявки. Якщо такі дії виконуються вручну, вони займають час і створюють додаткове навантаження.

Одним із ефективних підходів до автоматизації є використання чат-ботів. Бот у месенджері не потребує окремого встановлення для користувача, працює в знайомому інтерфейсі й може виконувати роль інтеграційного шару між людиною та зовнішніми сервісами. Telegram є зручною платформою для такого рішення, оскільки надає відкритий Bot API, підтримує webhook-механізм, текстові повідомлення.

Додаткові можливості відкриває використання генеративного штучного інтелекту. У класичному боті користувач має знати точну команду, наприклад `/create_task` або `/list_tasks`. Інтелектуальний бот може приймати повідомлення природною мовою, визначати намір користувача, виділяти параметри й запускати потрібний сценарій без жорсткої прив'язки до синтаксису команд.

Метою кваліфікаційної роботи є розробка інтелектуального Telegram-бота для автоматизації робочих процесів із використанням зовнішніх API, workflow-

рушія, AI-агентів, збереження стану діалогу та підтримки текстових повідомлень.

Об'єктом дослідження є процес автоматизації робочих комунікацій і виконання типових дій у зовнішніх інформаційних системах через месенджер.

Предметом дослідження є методи, засоби та програмні компоненти створення Telegram-бота з підтримкою AI-розпізнавання намірів, workflow-сценаріїв, інтеграцій із зовнішніми API, збереження контексту діалогу й контролю доступу користувачів.

Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати предметну область, існуючі аналоги та сучасні підходи до побудови систем відповідного класу;
- сформулювати функціональні та нефункціональні вимоги до розроблюваної системи і спроектувати її архітектуру;
- реалізувати програмний продукт (Telegram-бот) із необхідним функціоналом, інтеграцією зовнішніх сервісів та підтримкою текстових повідомлень, приватних і групових чатів;
- провести тестування системи, перевірку її безпеки та стійкості до навантажень;
- підготувати комплексну технічну документацію до розробленого програмного забезпечення.

Практична цінність розробки полягає в можливості використання бота як внутрішнього асистента команди. Система дозволяє створювати задачі, переглядати списки задач, отримувати листи й події календаря без переходу в окремі інтерфейси. Це скорочує кількість ручних операцій і підвищує оперативність роботи.

Апробацію результатів кваліфікаційної роботи здійснено шляхом участі в X Міжнародній науково-практичній конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2026)», м. Луцьк, 23-24 травня 2026 р. [1] (додаток А).

РОЗДІЛ 1

АНАЛІЗ МЕТОДІВ РОЗРОБКИ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Автоматизація робочих процесів є одним із ключових напрямів розвитку сучасних інформаційних систем. Її основна мета полягає у зменшенні кількості ручних операцій, стандартизації виконання типових дій і підвищенні прозорості роботи. За даними актуальних досліджень McKinsey Global Institute (2023), стрімкий розвиток генеративного штучного інтелекту значно підвищив потенціал автоматизації: на сьогодні близько 60-70 % робочого часу працівників може бути вивільнено завдяки впровадженню існуючих технологій. Зокрема, прогнозується, що автоматизація окремих бізнес-процесів дозволить підвищити продуктивність світової економіки на трильйони доларів щорічно, що підтверджує доцільність розробки інтелектуальних систем інтеграції, подібних до AutomationBot [1]. Для команд, які працюють із клієнтськими заявками, задачами, електронною поштою та календарями, автоматизація дає змогу швидше реагувати на події й уникати дублювання інформації.

Традиційні системи автоматизації зазвичай реалізуються у вигляді вебплатформ або корпоративних порталів. Вони надають широкі можливості, але потребують переходу в окремий інтерфейс. У багатьох випадках користувачу потрібно виконати лише одну коротку дію, наприклад створити задачу за повідомленням або отримати список подій на сьогодні. Для таких сценаріїв повноцінний вебінтерфейс є надлишковим. Дослідження Harvard Business Review (2022) демонструють феномен «перемикання між контекстами» (context switching): середньостатистичний працівник перемикається між різними застосунками та вебсайтами до 1200 разів на день. Це призводить до того, що фахівці витрачають до 9 % свого річного робочого часу лише на відновлення концентрації після таких переходів [2].

Месенджери стали природним середовищем робочої комунікації. Працівники обговорюють задачі, отримують уточнення та діляться статусами саме в чатах. Станом на 2024 рік, кількість активних користувачів Telegram перевищила 900 мільйонів на місяць [3], що робить його однією з найбільш затребуваних платформ для інтеграції бізнес-інструментів. Тому інтеграція автоматизації в месенджер дозволяє виконувати робочі дії там, де виникає потреба, без перемикання контексту.

Telegram-боти підтримують як приватні діалоги, так і групові чати. Це дає змогу використовувати одного бота для персональної роботи користувача та для командної взаємодії. Наприклад, у груповому чаті користувач може згадати бота й попросити створити задачу за поточним обговоренням, а в приватному діалозі переглянути власні задачі або події календаря.

Поява великих мовних моделей змінила підхід до взаємодії з ботами. Якщо раніше основним способом керування були кнопки або команди, то тепер бот може аналізувати повідомлення природною мовою. Це особливо важливо для робочих процесів, де користувачі формулюють запити неформально й не хочуть пам'ятати точний формат команди.

Разом із перевагами виникають і технічні виклики. Система повинна правильно визначати намір користувача, передавати до workflow лише валідні параметри, контролювати доступ, обробляти помилки зовнішніх API та зберігати контекст попередніх повідомлень. Без цього інтелектуальний бот може створювати некоректні задачі або повертати неповні результати.

Аналогічні рішення можна поділити на кілька груп: корпоративні чат-боти, інтеграційні no-code платформи, вбудовані асистенти окремих сервісів і самостійно розроблені боти. Кожен підхід має власну сферу застосування, переваги та обмеження.

Корпоративні чат-боти часто інтегруються з одним сервісом: CRM, сервіс-деском або таск-трекером. Вони добре виконують заздалегідь визначені сценарії, але менш придатні для розширення під специфічні процеси команди. Такі боти нерідко працюють за принципом меню або набору команд.

No-code та low-code платформи дозволяють швидко зв'язувати різні сервіси, приклад no-code платформи зображено на рисунку 1.1.

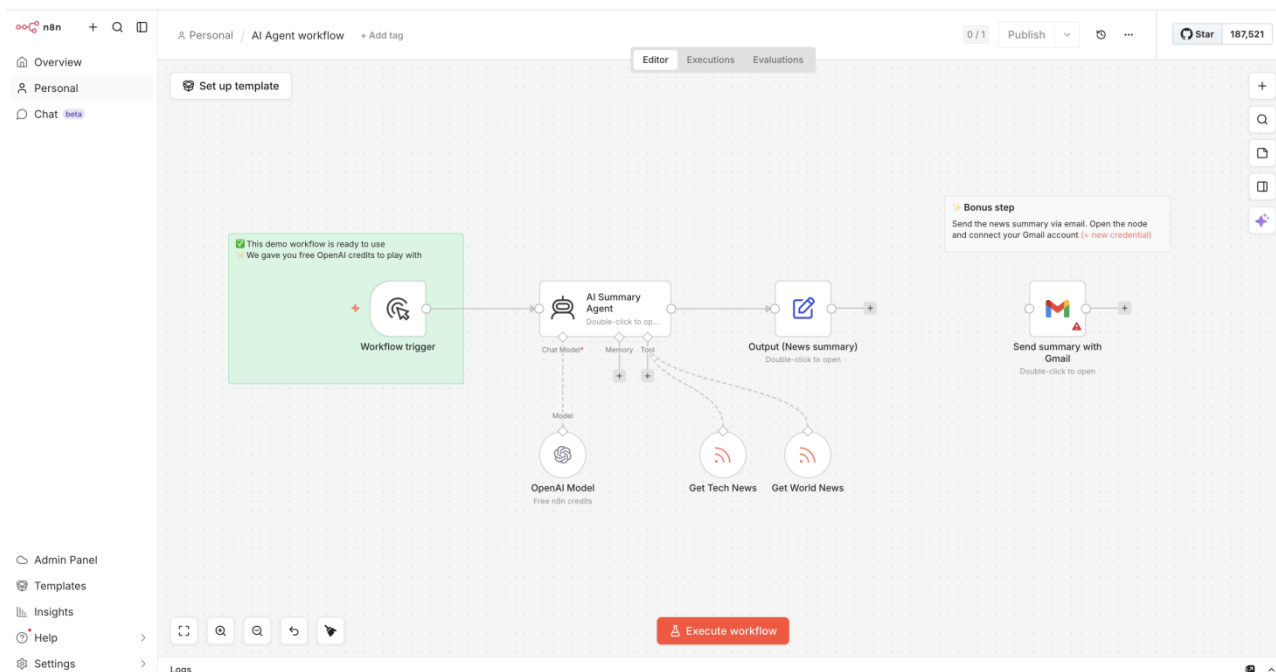


Рисунок 1.1 – Приклад інтерфейсу No-code платформи для автоматизацій [4]

Їхньою перевагою є швидкий старт без глибокого програмування. Однак складна логіка, збереження контексту, AI-обробка повідомлень і точна валідація параметрів можуть бути обмеженими або залежати від тарифного плану.

Вбудовані асистенти CRM або таск-трекерів мають добру інтеграцію з власною системою, але зазвичай не є універсальною точкою доступу до Gmail, Calendar, Jira, Worksection і внутрішніх API одночасно. Вони вирішують локальну задачу, а не проблему фрагментації робочого середовища.

Розробка власного рішення потребує більше часу, але дає повний контроль над архітектурою, логікою доступу, набором workflow, форматом відповідей і механізмом інтеграції. Саме цей підхід обрано для моєї розробки, оскільки система має працювати з кількома джерелами повідомлень і різними зовнішніми сервісами.

Порівняльний аналіз показує, що найбільш доцільним є створення модульного Telegram-бота з AI-розпізнаванням намірів (табл. 1.1). Така система

може починатися з невеликого набору workflow, а надалі розширюватися новими джерелами, агентами та інтеграціями без повної перебудови архітектури. Це дозволить мінімізувати стартові витрати на розробку та високу адаптивність.

Таблиця 1.1 – Порівняльний аналіз рішень для автоматизації

Рішення	Переваги	Недоліки
Бот з фіксованими командами	Простота реалізації, передбачувана логіка	Користувач має знати точний формат команд
No-code інтеграції	Швидке налаштування простих сценаріїв	Обмежена гнучкість і залежність від платформи
Асистент у CRM	Глибока інтеграція з однією системою	Не покриває кілька сервісів одночасно
AutomationBot	AI-намір, Telegram, API, workflow, кілька інтеграцій	Потребує розгортання й налаштування ключів

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

За мету цієї кваліфікаційної роботи взято розробку інтелектуального Telegram-бота для автоматизації робочих процесів із використанням зовнішніх API, workflow-сценаріїв та модуля штучного інтелекту. Розроблюваний програмний продукт має забезпечити користувачам швидкий і зручний доступ до робочих сервісів без необхідності постійного перемикання між CRM, поштою, календарем, таск-трекерами та іншими системами. Основна ідея проєкту полягає у створенні єдиного діалогового інтерфейсу в Telegram, через який користувач може формулювати запити природною мовою, а система автоматично визначає потрібну дію, запускає відповідний workflow і повертає результат.

Розроблюваний бот повинен підтримувати створення задач у GooseCRM, отримання списку задач, перегляд листів Gmail за вказану дату, отримання подій Google Calendar, обробку текстових повідомлень, а також запуск окремих сценаріїв через HTTP API. Інтеграція зі штучним інтелектом необхідна для розпізнавання намірів користувача, виділення параметрів із природномовного

повідомлення та формування більш зручної відповіді. Такий підхід дозволить зменшити кількість ручних операцій, пришвидшити виконання типових дій і зробити взаємодію з робочими сервісами більш централізованою.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- проаналізувати предметну область, існуючі аналоги та сучасні підходи до побудови систем відповідного класу;
- сформулювати функціональні та нефункціональні вимоги до розроблюваної системи і спроектувати її архітектуру;
- реалізувати програмний продукт (Telegram-бот) із необхідним функціоналом, інтеграцією зовнішніх сервісів та підтримкою текстових повідомлень, приватних і групових чатів;
- провести тестування системи, перевірку її безпеки та стійкості до навантажень;
- підготувати комплексну технічну документацію до розробленого програмного забезпечення.

Висновки до розділу 1

У першому розділі було проаналізовано предметну область автоматизації робочих процесів за допомогою чат-ботів та зовнішніх API. Встановлено, що в сучасному цифровому середовищі користувачі часто змушені працювати одночасно з кількома сервісами: CRM, електронною поштою, календарем, таск-трекерами та месенджерами. Така фрагментація ускладнює виконання типових робочих дій, збільшує кількість ручних операцій і підвищує ризик втрати важливої інформації.

Було визначено, що Telegram є зручною платформою для реалізації бота, оскільки має відкритий Bot API, підтримує webhook-механізм, текстові повідомлення, роботу в приватних, а також доступний на більшості популярних платформ. Це дозволяє використовувати Telegram як єдиний інтерфейс для взаємодії з різними робочими сервісами.

Важливим аспектом проєктування такої системи є побудова наскрізних сценаріїв (workflows), які пов'язують ізольовані API сторонніх платформ у єдині логічні ланцюжки. Автоматизація на основі подій або користувацьких тригерів дозволяє мінімізувати рутинне перемикання між вкладками браузера чи різними застосунками. Наприклад, створення завдання в таск-трекері на основі повідомлення з чату, автоматичне надсилання сповіщення в CRM та фіксація дедлайну в календарі мають відбуватися в межах одного безшовного процесу. Це вимагає розробки гнучкої архітектури, здатної ефективно обробляти черги запитів.

У ході аналізу аналогічних рішень встановлено, що існуючі інструменти автоматизації часто або прив'язані до одного конкретного сервісу, або потребують складного налаштування, або не забезпечують достатньої гнучкості для природномовної взаємодії з користувачем. Саме тому доцільним є створення власного інтелектуального Telegram-бота, який поєднує workflow-підхід, AI-розпізнавання намірів і інтеграцію із зовнішніми API. Також було обґрунтовано необхідність використання штучного інтелекту для обробки повідомлень природною мовою. Завдяки AI-модулю користувач не повинен запам'ятовувати точні команди, а може формулювати запити у звичній формі.

Отже, у першому розділі було визначено актуальність теми, проаналізовано наявні підходи до автоматизації, обґрунтовано вибір Telegram як основної платформи взаємодії та сформульовано завдання кваліфікаційної роботи. Отримані результати стали основою для подальшого визначення вимог, вибору технологій і практичної реалізації інтелектуального Telegram-бота AutomationBot.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Метою розроблюваної системи є створення інтелектуального Telegram-бота, який забезпечує автоматизацію типових робочих процесів користувача через єдиний зручний діалоговий інтерфейс. Система повинна приймати повідомлення користувачів у Telegram, аналізувати їхній зміст за допомогою AI-модуля, визначати потрібний сценарій обробки, взаємодіяти із зовнішніми сервісами та повертати користувачу зрозумілу відповідь.

Розроблюваний програмний продукт орієнтований на автоматизацію дій, які зазвичай виконуються вручну: створення задач у CRM, перегляд списку задач, отримання листів із Gmail, перегляд подій Google Calendar, а також запуск workflow через HTTP API. Такий підхід дозволяє скоротити кількість переходів між різними сервісами та зробити роботу користувача більш швидкою й централізованою.

Особливістю системи є використання Telegram не лише як каналу комунікації, а як повноцінного інтерфейсу доступу до робочих сервісів. Користувач може сформулювати запит природною мовою, наприклад: «створи задачу на перевірку авторизації», «покажи мої відкриті задачі», «які події в календарі завтра» або «покажи листи за сьогодні». Після цього система повинна визначити намір користувача, виділити необхідні параметри та виконати відповідний сценарій.

Для забезпечення повноцінної роботи інтелектуального Telegram-бота необхідно реалізувати набір функціональних вимог, які охоплюють основні сценарії взаємодії користувача із системою. До них належать такі функціональні вимоги:

- ідентифікація та авторизація користувача (система повинна автоматично розпізнавати унікальний ідентифікатор користувача (`telegram_id`),

перевіряти наявність облікового запису в базі даних та рівень його доступу для запобігання несанкціонованому використанню бізнес-сценаріїв);

- обробка повідомлень у приватних та групових чатах (забезпечення коректного приймання, фільтрації та обробки текстових запитів природною мовою як у приватних діалогах, так і в межах спільних робочих груп (при зверненні до асистента через згадку). Користувацькі відповіді повинні бути короткими, зрозумілими й придатними для зручного читання в месенджері);

- автоматичне визначення наміру користувача (семантичний аналіз тексту повідомлення для точного розпізнавання наміру користувача, виділення сутностей і параметрів запиту та подальшого запуску відповідного інтеграційного сценарію);

- збереження історії взаємодії та відповідей (система повинна підтримувати тимчасове збереження контексту діалогу для обробки пов'язаних реплік, відповідей на попередні повідомлення та послідовного уточнення параметрів у межах одного бізнес-процесу. Окрім основної функціональності, важливо забезпечити стабільність, безпеку, масштабованість і зручність супроводу системи).

Нефункціональні вимоги визначають якісні характеристики програмного продукту, які не описують конкретну дію користувача, але впливають на працездатність і придатність системи до реального використання. Для розроблюваного Telegram-бота було сформульовано такі нефункціональні вимоги:

- асинхронність та швидкодія обробки запитів (архітектура програмного забезпечення має базуватися на асинхронній моделі обробки подій, що забезпечує високу швидкість реакції системи та запобігає блокуванню потоків під час очікування відповідей від зовнішніх API);

- надійність та стійкість до відмов (будь-які виняткові ситуації, збої або помилки під час інтеграційної взаємодії із зовнішніми сервісами повинні ізолюватися перехоплюватися і логуватися системою, не призводячи до повного чи часткового зупинення роботи всього застосунку);

- логування та діагностика системи (у програмному комплексі має бути передбачене обов'язкове централізоване логування вхідних і вихідних повідомлень, а також виникнення системних помилок для забезпечення можливості подальшої діагностики й моніторингу стану застосунку);
- безпека та оптимізація збереження даних (секретні ключі, токени та паролі повинні зберігатися у змінних середовища або в окремому захищеному каталозі secrets, взаємодія з Telegram webhook і зовнішніми API має виконуватися через захищене HTTPS-з'єднання, а база даних повинна містити лише необхідну інформацію без дублювання даних із зовнішніх сервісів);
- масштабованість, контейнеризація та розгортання (архітектура повинна мати слабопов'язану структуру для додавання нових джерел повідомлень без переписування основної логіки й розширення workflow-сценаріїв шляхом створення та реєстрації окремих класів, а також підтримувати контейнерне розгортання за допомогою Docker і Docker Compose).

Для візуального опису структури системи було використано діаграми, які демонструють основні компоненти, потоки даних і логічну модель збереження даних. Використання діаграм під час проєктування дає змогу краще зрозуміти взаємодію між складовими частинами програмного продукту, виявити можливі логічні помилки ще до етапу реалізації та спростити подальше документування.

Функціональна архітектура AutomationBot зображена на рисунку 2.1. Вона демонструє основні підсистеми програмного продукту: джерела повідомлень, workflow-рушій, AI-агентів, зовнішні API, Redis, MySQL та Docker-середовище.

Таким чином, поєднання чітко визначених нефункціональних критеріїв та модульного підходу до проєктування дозволило сформуванню стійкої архітектурної концепції майбутнього застосунку. Наведена схема детально відображає розподіл відповідальності між модулями, де кожен компонент виконує ізольовану функцію, але працює в синергії з іншими для забезпечення наскрізної автоматизації робочих процесів. Детальний аналіз взаємодії представлених підсистем, специфіка обміну даними між ними та логіка інтеграції AI-інструментів розглядаються далі.

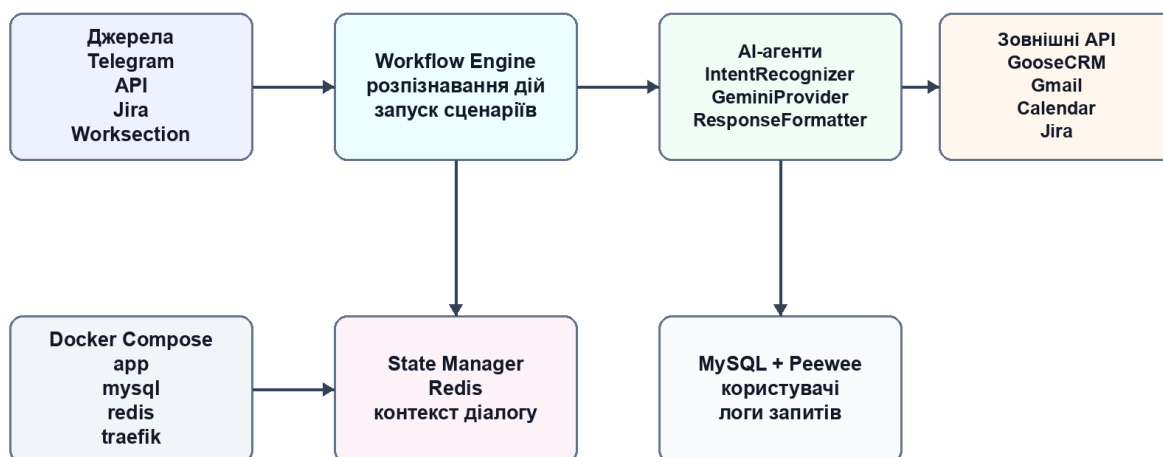


Рисунок 2.1 – Функціональна архітектура AutomationBot

У цій архітектурі джерела повідомлень відповідають за приймання запитів із різних каналів. Основним джерелом є Telegram, однак система також підтримує API-запити, Jira, Worksection та інші джерела, які можуть бути додані в майбутньому. Кожне джерело має власну webhook-адресу та обробник, але після попередньої обробки повідомлення всі вони передають запит до спільного workflow-рушія. Workflow Engine є центральним компонентом системи. Він отримує текст повідомлення, передає його до AI-агента для визначення наміру, отримує список дій і запускає відповідні workflow. Такий підхід дозволяє відокремити логіку приймання повідомлень від бізнес-логіки виконання конкретних дій. AI-агенти відповідають за обробку природномовних повідомлень. Вони формують prompt, звертаються до Gemini API [5] та перетворюють відповідь моделі у структурований формат, придатний для подальшого виконання. Це дозволяє користувачу не запам'ятовувати конкретні команди, а формулювати запит природною мовою. Зовнішні API забезпечують виконання практичних робочих дій. Наприклад, GooseCRM використовується для створення та отримання задач, Gmail API – для перегляду листів, Google Calendar API – для отримання подій календаря. У майбутньому ця частина може бути розширена іншими сервісами. Окрему роль відіграє Redis [6], який використовується для тимчасового збереження стану діалогу. Це важливо для

сценаріїв, у яких користувач відповідає на попереднє повідомлення бота або уточнює параметри вже розпочатого workflow. MySQL використовується для збереження користувачів і логів запитів, тобто тих даних, які мають довготривале значення.

Процес обробки інформаційних потоків у системі AutomationBot побудований як багатоетапний цикл послідовної трансформації вхідного неструктурованого запиту у формалізовану дію, що базується на розробленій діаграмі потоків даних (рис. 2.2).

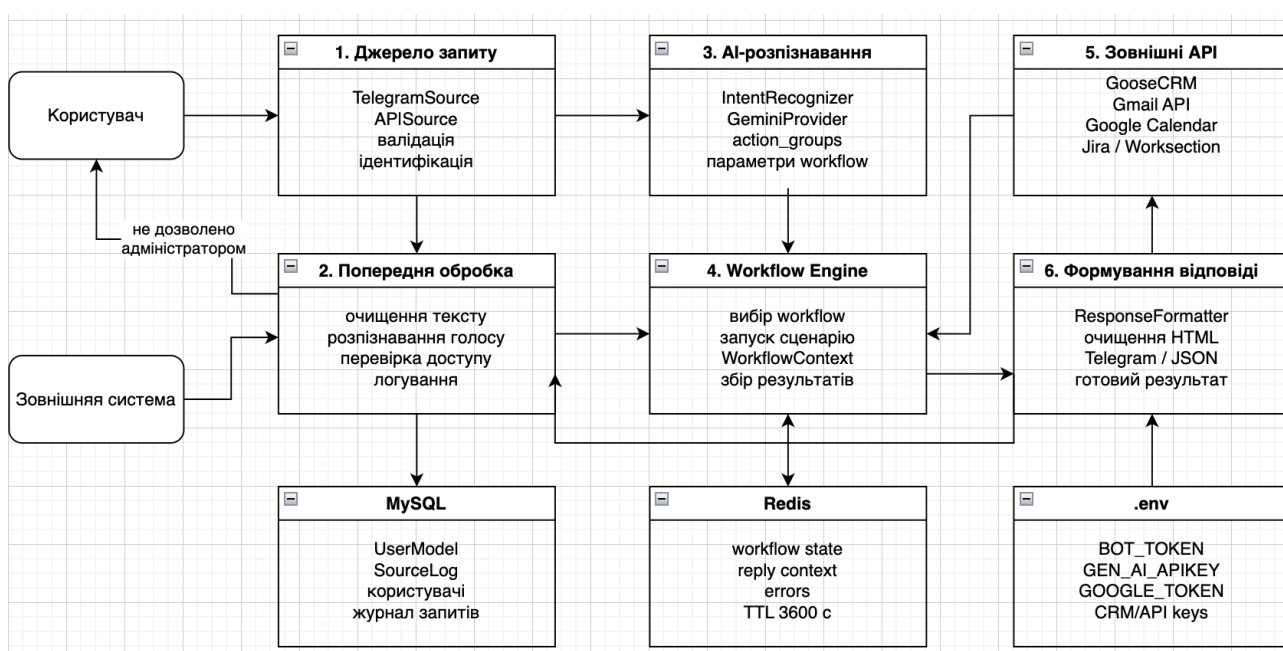


Рисунок 2.2 – Діаграма потоків даних у системі

На початковій стадії функціонування системи здійснюється визначення та приймання повідомлення від зовнішнього ініціатора, при цьому залежно від каналу надходження даних активуються відповідні компоненти рівня джерел (Source Layer). Зокрема, при ініціації запиту через месенджер обробка виконується модулем TelegramSource, тоді як вхідні запити через програмний інтерфейс приймаються компонентом APISource. На цьому етапі реалізовано механізм вхідної фільтрації та валідації, що включає перевірку базових критеріїв, таких як наявність текстового контенту, відповідність типу переданих даних,

верифікація прав доступу користувача через поле `is_approved` або аутентифікація за допомогою API-ключа для зовнішніх систем.

Після успішного проходження первинної перевірки система переходить до етапу семантичного аналізу та ідентифікації намірів (Intent Recognition), де здійснюється інтелектуальна інтерпретація повідомлення модулем `IntentRecognizer`. Даний компонент агрегує перелік доступних системних сценаріїв (workflow) та формує структурований запит до AI-моделі на базі Gemini API, результатом чого є отримання формалізованої відповіді із назвою конкретної дії, виділеними параметрами та сервісним повідомленням для подальшої передачі у відповідний операційний контекст.

Наступним кроком є безпосередня активація бізнес-логіки системи через Workflow Engine, який ініціює запуск спеціалізованого сценарію залежно від результатів попереднього аналізу: при необхідності реєстрації інциденту в системі управління задачами запускається `CreateGooseIssueWorkflow`, для агрегації електронної кореспонденції за вказаний період задіюється `GmailsListWorkflow`, а взаємодія з календарними сервісами реалізується через модуль `GoogleCalendarListWorkflow`.

Завершальна стадія обробки полягає у синтезі та агрегації фінальної відповіді, де отримані результати виконання сценарію передаються до компонента `ResponseFormatter` для структурування та форматування тексту згідно з ергономічними вимогами інтерфейсу месенджера. Готова відповідь ретранслюється кінцевому клієнту через початковий шлюз (`TelegramSource` або `APISource`), при цьому паралельно із процесом формування відповіді здійснюється логування транзакції у сутності `SourceLog` та оновлення стану діалогу в оперативному сховищі `Redis`, що забезпечує цілісність, простежуваність та підтримку контексту в усіх внутрішніх процесах системи. Логічна модель даних і стану що підтримує ці процеси, деталізована на рисунку 2.3.

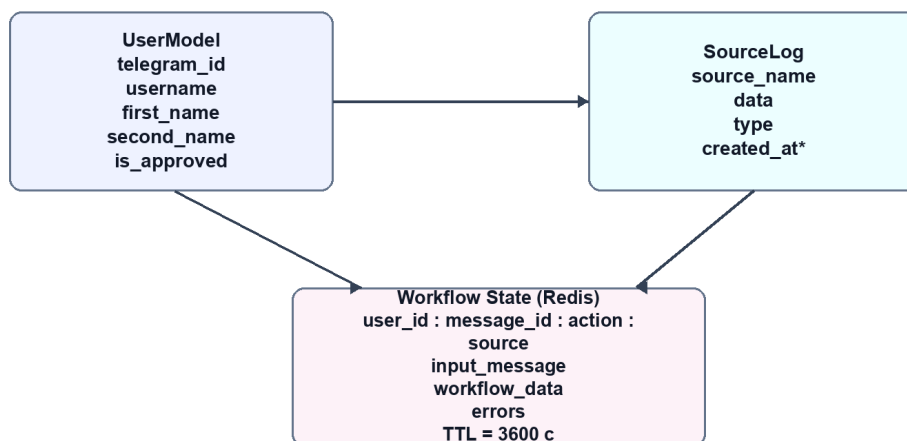


Рисунок 2.3 – Логічна модель даних і стану

Логічна модель даних проектованої системи характеризується компактністю та відсутністю надлишковості, оскільки архітектура AutomationBot виключає дублювання бізнес-інформації, що належить зовнішнім сервісам. Зокрема, об'єкти задач, електронна кореспонденція та календарні події залишаються в межах компетенції GooseCRM, Jira, Gmail та Google Calendar відповідно, тоді як внутрішня база даних бота оперує лише сутностями, необхідними для забезпечення його працездатності. Центральною ланкою моделі є сутність UserModel, яка акумулює ідентифікаційні дані користувачів Telegram, включаючи telegram_id для унікальної адресації, username, first_name та second_name для формування контекстуального профілю, а також атрибут is_approved для адміністрування прав доступу до функціоналу системи. Для цілей моніторингу та діагностики використовується сутність SourceLog, що фіксує параметри вхідних запитів, вихідних відповідей та типи операцій, дозволяючи проводити ретроспективний аналіз поведінки системи та ідентифікувати потенційні помилки взаємодії. Тимчасове зберігання стану активних сценаріїв реалізовано на базі Redis, де за унікальним ключем, що поєднує ідентифікатор користувача та параметри дії, зберігається вхідний контент, помилки виконання та контекстуальні дані конкретного workflow. Механізм автоматичного очищення застарілого контексту забезпечується через встановлення обмеженого часу життя записів (TTL), що оптимізує використання

оперативної пам'яті. Таким чином, формалізація вимог до структури даних та ідентифікація ключових компонентів дозволили чітко розмежувати зони відповідальності в межах AutomationBot, забезпечуючи перехід до етапу реалізації програмного продукту на базі верифікованих архітектурних рішень.

Діаграма компонентів (рис. 2.4) відображає архітектурну структуру програмного продукту та взаємодію між Telegram Bot API, aiohttp-сервером, workflow-рушієм, AI-модулем, Redis, MySQL і зовнішніми API. Вона дозволяє виявити, які частини системи є автономними, а які взаємодіють через чітко визначені інтерфейси.

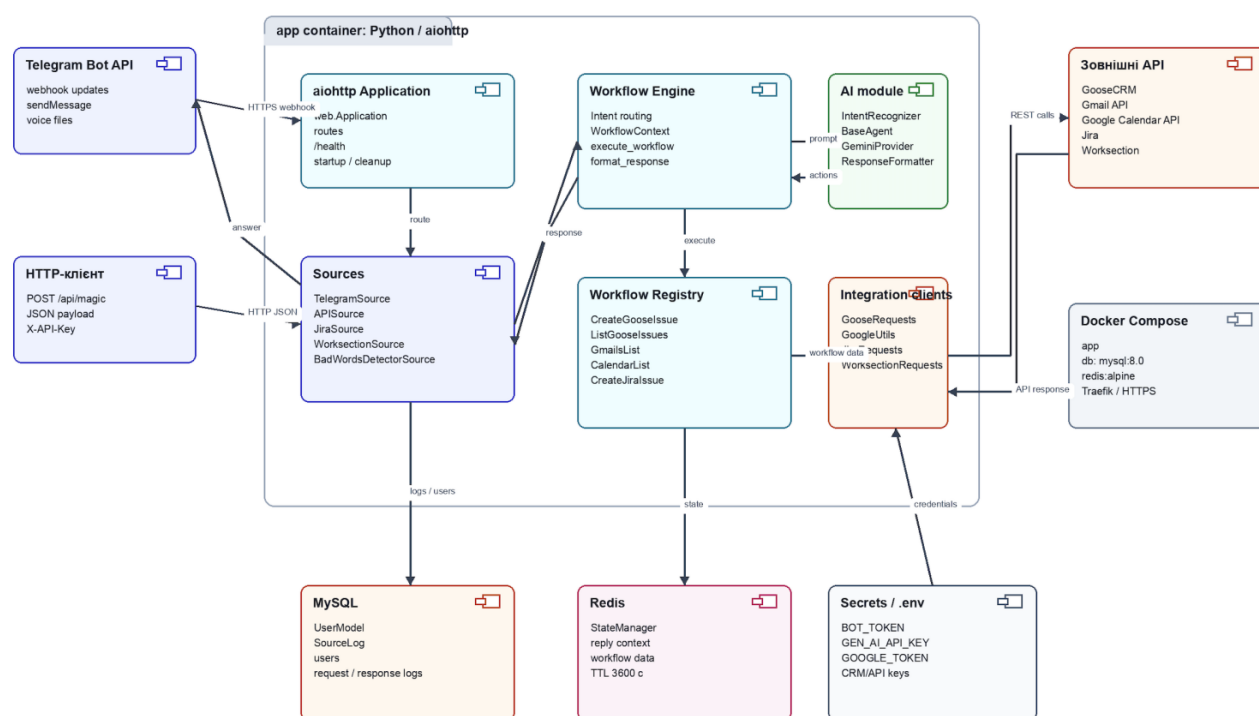


Рисунок 2.4 – Діаграма компонентів AutomationBot

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Процес вибору технологій для реалізації інтеграційної системи «AutomationBot» базувався на необхідності забезпечення високої швидкості обробки асинхронних запитів та гнучкості при роботі з різномірними

корпоративними API. Платформою для розгортання клієнтської частини було обрано месенджер Telegram, оскільки він пропонує найбільш розвинений інструментарій Bot API порівняно з аналогами на кшталт WhatsApp чи Viber. Це рішення дозволяє використовувати сучасні методи взаємодії, включаючи вебхуки та інлайн-меню, забезпечуючи при цьому кросплатформеність та легкість доступу для кінцевого користувача без потреби у додатковому програмному забезпеченні.

Центральним елементом розробки стала мова програмування Python [7], вибір якої зумовлений її домінуючим становищем у сфері розробки штучного інтелекту та наявністю зрілих бібліотек для асинхронного мережевого програмування. На відміну від мов Java або C#, Python дозволяє значно швидше прототипувати модульні структури та інтегрувати складні NLP-моделі. Для написання коду використовувалось інтегроване середовище VSCode, яке завдяки вбудованим інструментам аналізу та підтримці віртуальних середовищ мінімізує ризики виникнення синтаксичних та логічних помилок на етапі розробки.

Реалізація логіки взаємодії з Telegram Bot API була здійснена за допомогою бібліотеки aiogram версії 3.x [8]. Цей фреймворк було обрано серед альтернатив на кшталт Telethon через його повну відповідність принципам асинхронності asyncio та зручну систему маршрутизації подій [9]. Це дозволяє системі обробляти велику кількість паралельних запитів без затримок. HTTP-серверна частина, що відповідає за приймання вхідних вебхуків від зовнішніх систем та самого месенджера, базується на фреймворку aiohttp [10], який ідеально підходить для високонавантажених сервісів з великою кількістю вхідних і вихідних мережевих з'єднань.

Ключову роль у системі відіграє модуль штучного інтелекту на базі Google Gemini API. Вибір даної моделі був продиктований необхідністю глибокої інтеграції з сервісами Google Workspace та високою точністю генерації структурованих відповідей у форматі JSON. Порівняно з моделями OpenAI, Gemini забезпечує вищу швидкість обробки контексту при роботі з поштою та календарями. Розроблена архітектура дозволяє інкапсулювати звернення до AI-

провайдера, що забезпечує можливість швидкої заміни моделі у разі появи ефективніших рішень без необхідності рефакторингу основної бізнес-логіки.

Архітектура збереження даних побудована на поєднанні реляційної бази MySQL та нереляційного сховища Redis [11]. MySQL у зв'язці з ORM Peewee [12] забезпечує надійне довготривале зберігання інформації про користувачів та журналів повідомлень, гарантуючи цілісність даних при високих навантаженнях. У той же час Redis використовується як високошвидкісне сховище для керування станами діалогів та тимчасовим контекстом workflow. Це дозволяє системі миттєво відновлювати стан взаємодії з користувачем, не навантажуючи основну базу даних частими запитами на читання та запис короткоживучих об'єктів.

Технологічний стек системи доповнено інструментами контейнеризації Docker та Docker Compose [13], що забезпечують повну ізоляцію компонентів програми, баз даних та систем кешування у незалежних контейнерах. Використання даної технології гарантує ідентичність конфігурацій робочого середовища на всіх стадіях життєвого циклу – від локальної розробки до фінального розгортання на сервері. Таке архітектурне рішення спрощує горизонтальне масштабування системи [14] та підвищує її загальну відмовостійкість завдяки автономному управлінню окремими компонентами. Для систематизації процесу розробки, контролю версій та забезпечення цілісності вихідного коду застосовано платформу GitHub, що дозволило автоматизувати процеси збірки та відстежувати історію змін протягом усього періоду проектування [15].

Висновки до розділу 2

У процесі виконання другого розділу було проведено детальний аналіз предметної області та сформовано комплексну специфікацію вимог до розроблюваного програмного забезпечення модульної інтеграційної системи AutomationBot. Усі технічні вимоги було класифіковано на чотири ключові

функціональні блоки (ідентифікація користувачів, обробка чатів, семантичний аналіз намірів та збереження контексту) і п'ять нефункціональних категорій (швидкодія, надійність, логування, безпека й масштабованість).

Обґрунтовано вибір технологічного стеку для практичної реалізації системи. Основним інструментом розробки обрано мову програмування Python завдяки її широкій екосистемі для роботи з асинхронним кодом та штучним інтелектом. Для реалізації гнучкої взаємодії з інтерфейсом месенджера обрано асинхронний фреймворк `aiogram`, а як інтелектуальний аналізатор неструктурованого тексту користувачів – генеративну модель штучного інтелекту Gemini API, що дозволяє динамічно визначати наміри (intents) без жорсткого програмування правил.

Спроектовано архітектурну структуру інтеграційного комплексу, що базується на слабопов'язаних модулях (workflow-сценаріях). Побудовано діаграми прецедентів (Use Case), послідовності (Sequence) та класів, які візуалізують логіку обробки вхідних подій та механізми взаємодії між ядром бота, базою даних і зовнішніми корпоративними шлюзами. Визначено, що використання технології контейнеризації Docker забезпечить високу портативність застосунку, а шифрування HTTPS та ізоляція токенів – надійний захист даних. Створений теоретичний та архітектурний базис дозволив повністю перейти до етапу практичної реалізації системи.

Отже, аналітичні та проектувальні результати, отримані на цьому етапі, заклали надійне підґрунтя для подальшої технічної розробки. Чітко визначена специфікація вимог у поєднанні з побудованими UML-моделями дозволяє оптимізувати процес написання коду та гарантує відповідність архітектури критеріям стійкості й безпеки. Сформований інженерний каркас та обраний інструментарій створюють умови для переходу до наступного розділу, який присвячено безпосередньому програмному втіленню модулів AutomationBot, налаштуванню інфраструктури та проведенню комплексного тестування системи.

РОЗДІЛ 3

РОЗРОБКА МОДУЛЬНОЇ ІНТЕГРАЦІЙНОЇ СИСТЕМИ НА ОСНОВІ TELEGRAM-БОТА З AI ДЛЯ ВЗАЄМОДІЇ З КОРПОРАТИВНИМИ СЕРВІСАМИ ТА АВТОМАТИЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ ВИКОРИСТОВУЮЧИ PYTHON

3.1 Практична реалізація об'єкта проєктування

Практична реалізація AutomationBot розпочалася з розробки структури проєкту, яка б дозволяла одночасно керувати багатьма процесами: отриманням повідомлень через Telegram, роботою зі штучним інтелектом та взаємодією із зовнішніми сервісами. Оскільки основним інтерфейсом системи є чат месенджера, серверна частина проєкту спроектована так, щоб максимально швидко приймати вхідні події (webhook) і обробляти їх без затримок. Для цього код було розділено на окремі модулі, що дозволяє уникнути змішування технічних налаштувань Telegram із логікою роботи нейронної мережі чи бази даних. Логічна організація всіх компонентів системи та їхнє розташування в ієрархії файлів представлені на рисунку 3.1.

Весь програмний код розподілений між двома основними каталогами, кожен з яких виконує свою роль. Директорія core містить базові механізми системи, які не залежать від конкретних задач автоматизації. Тут зосереджено рушій для керування сценаріями діалогів, класи для роботи зі станом користувача та інструменти взаємодії з базами даних MySQL і Redis. Це ядро забезпечує стабільну роботу бота та відповідає за те, щоб кожне повідомлення було вчасно доставлене до потрібного обробника. Таке архітектурне розділення дозволяє ізолювати системні процеси від прикладного рівня, спрощуючи підтримку коду та мінімізуючи ризик виникнення критичних помилок при модифікації сервісів. Завдяки такому підходу основна технічна логіка залишається незмінною, навіть якщо в майбутньому знадобиться додати нові функції.

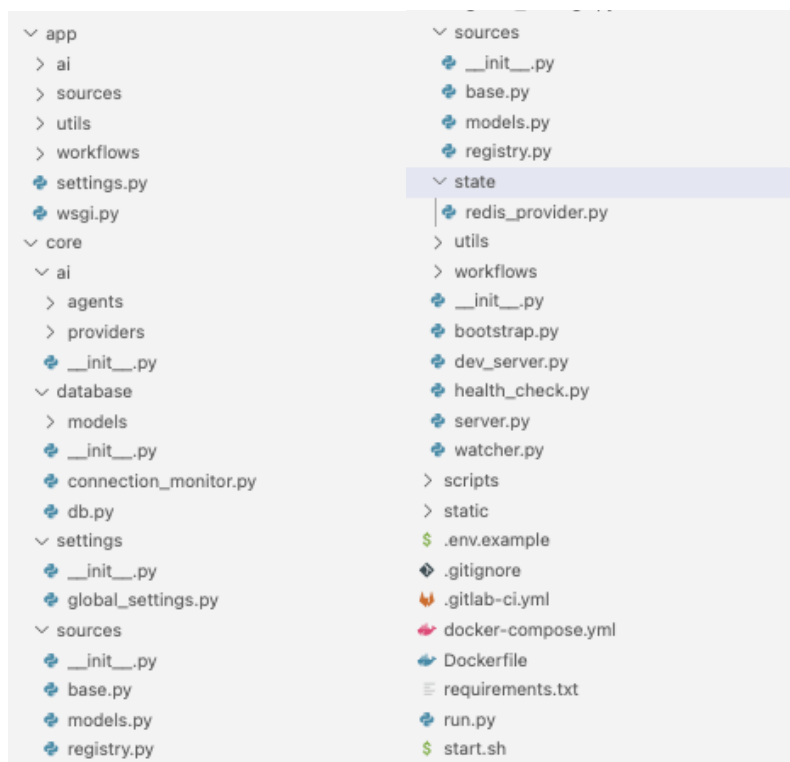


Рисунок 3.1 – Файлова структура AutomationBot

Прикладна частина проєкту розташована в каталозі `app`, який відповідає за безпосереднє виконання робочих сценаріїв. У цьому модулі зберігаються налаштування інтеграцій із зовнішніми сервісами, такими як Jira чи Worksection, а також програмні компоненти для звернення до моделі Gemini. Саме тут відбувається основна робота з аналізу тексту повідомлень і формування відповідей. Такий розподіл на два блоки дозволяє розробнику легко масштабувати систему: наприклад, для додавання нового AI-провайдера або інтеграції з іншим таск-трекером достатньо внести зміни лише в пакет `app`, не втручаючись у роботу базових механізмів ядра.

У процесі розробки ключовими залежностями стали `aiogram`, `aiohhttp`, `google-generativeai`, `google-api-python-client`, `redis`, `peewee`, `PyMySQL` і `python-decouple`. `Aiogram` використовується для роботи з Telegram Bot API, `aiohhttp` – для створення `webhook`-сервера, `google-generativeai` – для звернення до Gemini, `Redis` – для тимчасового стану діалогу, а `MySQL` разом із `Peewee` – для збереження користувачів і журналу запитів (ліст. 3.1).

Лістинг 3.1 – Фрагмент основних залежностей проєкту

```

aiofiles==24.1.0
aiogram==3.15.0
aiohappyeyeballs==2.4.4
aiohttp==3.10.11
aiosignal==1.3.1
blinker==1.8.2
chardet==3.0.4
charset-normalizer==3.4.0
click==8.1.7
colorama==0.4.6
cryptography==44.0.0
Flask==3.0.3
frozenlist==1.5.0
google-ai-generativelanguage==0.6.10
google-api-core==2.21.0
google-api-python-client==2.149.0
google-auth==2.35.0
google-auth-httpplib2==0.2.0
google-auth-oauthlib==1.2.1
google-generativeai==0.8.3
googleapis-common-protos==1.65.0
googletrans==3.0.0
pyscraper==2.22
pydantic==2.9.2
pydantic_core==2.23.4
PyMySQL==1.1.1
Telethon==1.37.0
tqdm==4.66.5
typing_extensions==4.12.2
uritemplate==4.1.1
urllib3==2.2.3
uvicorn==0.34.0
uvloop==0.21.0
google-api-python-client>=2.100.0
google-auth-httpplib2>=0.1.1
google-auth-oauthlib>=1.1.0

```

кінець лістингу 3.1

Налаштування системи винесені у файл `app/settings.py` (ліст. 3.2). У ньому зчитуються токени Telegram, ключ Gemini, параметри MySQL, Redis, GooseCRM, Worksection, Google OAuth та інші значення. Такий підхід важливий з точки зору безпеки: секретні значення не зберігаються безпосередньо в коді, а передаються через `.env` та каталог `secrets`.

Лістинг 3.2 – Налаштування бази даних, Redis і AI-провайдера

```

from app.ai.providers.gemini_provider import GeminiProvider
from decouple import config
from core.state.redis_provider import RedisProvider

BOT_TOKEN = config("BOT_TOKEN")
GEN_AI_API_KEY = config("GEN_AI_API_KEY")
AI_PROVIDER = GeminiProvider(
    api_key=GEN_AI_API_KEY,

```

```

        model="gemini-2.5-flash"
    )

    REDIS_CONFIG = {
        "host": config("REDIS_HOST", default="localhost"),
        "port": config("REDIS_PORT", default="6379"),
        "db": config("REDIS_DB", default="/0"),
    }
    redis_url =
    f"redis://{REDIS_CONFIG['host']}:{REDIS_CONFIG['port']}:{REDIS_CONFIG['db']}"
    REDIS_PROVIDER = RedisProvider(redis_url)

    DATABASE = {
        "database": config("DB_NAME"),
        "user": config("DB_USER"),
        "host": config("DB_HOST"),
        "port": int(config("DB_PORT")),
        "password": config("DB_PASSWORD"),
    }
}

```

кінець лістингу 3.2

Запуск у production-режимі виконується через Gunicorn з aiohttp worker. Це дозволяє використовувати AutomationBot як повноцінний серверний застосунок, який приймає webhook-запити з Telegram і HTTP-запити від зовнішніх систем. Для локальної розробки використовується run.py, а для серверного запуску – start.sh.

Ініціалізація застосунку відбувається в core/bootstrap.py (ліст. 3.3). Саме тут підключається база даних, створюються таблиці, завантажуються workflow та джерела повідомлень із налаштувань, реєструються endpoint-и й додається health-check. Завдяки цьому структура системи залишається гнучкою: нове джерело або workflow можна додати через settings.py без переписування основного коду запуску.

Лістинг 3.3 – Ініціалізація застосунку та реєстрація джерел

```

def initialize_application():
    database = database_instance.initialize(settings.DATABASE)
    initialize_tables()

    load_workflows_from_settings(settings.WORKFLOWS)
    load_sources_from_settings(settings.SOURCES)

    app = web.Application(client_max_size=1024**3)
    app.router.add_get("/health", health_check)

    for source in SOURCE_REGISTRY.values():
        source_instance = source()
        app.router.add_post(

```

```

        source_instance.webhook_url,
        source_instance.handler
    )
    app.router.add_get(
        source_instance.webhook_url,
        get_handler
    )

    if hasattr(source_instance, "on_startup"):
        app.on_startup.append(source_instance.on_startup)
    if hasattr(source_instance, "on_shutdown"):
        app.on_shutdown.append(source_instance.on_shutdown)

    return app

```

кінець лістингу 3.3

Для розгортання було підготовлено Dockerfile і docker-compose.yml (ліст. 3.4). У Dockerfile описується створення контейнера застосунку на основі Python 3.11, встановлення системних залежностей для MySQL-клієнта, встановлення бібліотек із requirements.txt і запуск start.sh. У docker-compose описано три основні сервіси: app, db і redis.

Лістинг 3.4 – Фрагмент Docker Compose для запуску сервісів

```

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "6667:8000"
    restart: always
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.goose.rule=Host(`goose.unweb.solutions`)"
      - "traefik.http.routers.goose.entrypoints=websecure"
      - "traefik.http.services.goose.loadbalancer.server.port=8000"
    networks:
      - app_network
    depends_on:
      db:
        condition: service_healthy
      redis:
        condition: service_started
    volumes:
      - ./secrets:/app/secrets
  db:
    image: mysql:8.0
    restart: unless-stopped
    networks:
      - app_network
    ports:
      - "3313:3306"
    environment:
      MYSQL_DATABASE: ${DB_NAME}
      MYSQL_USER: ${DB_USER}

```

```

    MYSQL_PASSWORD: ${DB_PASSWORD}
    MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWORD}
  volumes:
    - mysql_data:/var/lib/mysql
  redis:
    image: redis:alpine
    restart: unless-stopped
  networks:
    - app_network
  ports:
    - "6388:6379"
  volumes:
    - redis_data:/data
networks:
  app_network:
    driver: bridge
volumes:
  mysql_data:
  redis_data:

```

кінець лістингу 3.4

У середовищі запускаються три головні сервіси: app, db і redis. Контейнер app містить Python-застосунок, db працює на MySQL 8.0, а redis використовується для збереження тимчасового стану діалогу. Така структура спрощує розгортання на VPS, тому що всі залежності запускаються однією командою (рис. 3.2).

```

app-1 | 2026-05-07 11:08:42.492 | DEBUG | core.database.connection_monitor:start_monitoring:42 - Database connection is healthy
app-1 | [2026-05-07 11:09:15 +0000] [7] [INFO] Starting gunicorn 23.0.0
app-1 | [2026-05-07 11:09:15 +0000] [7] [INFO] Listening at: http://0.0.0.0:8000 (7)
app-1 | [2026-05-07 11:09:15 +0000] [7] [INFO] Using worker: aiohttp.GunicornWebWorker
app-1 | [2026-05-07 11:09:15 +0000] [8] [INFO] Booting worker with pid: 8
app-1 | [11:09:15] 🚀 Initializing Goose application
app-1 | 2026-05-07 11:09:16.574 | INFO | app.settings:<module>:54 - Redis URL: redis://redis:6379/0
app-1 | 2026-05-07 11:09:16.581 | INFO | core.database.db:initialize:62 - Successfully connected to the database
app-1 | 2026-05-07 11:09:16.589 | INFO | core.sources.models:initialize_tables:16 - SourceLog table created or already exists
app-1 | [11:09:16] 🔄 Loading workflows and sources
app-1 | 2026-05-07 11:09:21.519 | INFO | app.sources.badwords_detector:initialize_tables:12 - BadWord table created or already exists
app-1 | [11:09:21] 🔄 Checking database connection
app-1 | [11:09:21] ✅ Database connection successful
app-1 | [11:09:21] 🔄 Registered health check endpoint at https://goose.unweb.solutions/health
app-1 | [11:09:21] ✅ Registering source: Telegram on https://goose.unweb.solutions/telegram_webhook
app-1 | [11:09:21] ✅ Registering source: Worksection on https://goose.unweb.solutions/defence/worksection/webhook
app-1 | [11:09:21] ✅ Registering source: Jira on https://goose.unweb.solutions/defence/jira/webhook
app-1 | [11:09:21] ✅ Registering source: API on https://goose.unweb.solutions/api/magic
app-1 | [11:09:21] ✅ Registering source: BadWordsDetector on https://goose.unweb.solutions/badwords_detector
app-1 | [11:09:21] ✅ Registering analytics endpoint: https://goose.unweb.solutions/badwords_analytics
app-1 | [11:09:21] 🌟 Application initialized!
app-1 | 2026-05-07 11:09:21.683 | INFO | app.sources.telegram_source.source:on_startup:45 - Setting webhook: https://goose.unweb.solutions/telegram_webhook
app-1 | [11:09:21] ✅ Database connection monitor started
app-1 | 2026-05-07 11:09:21.749 | INFO | core.database.connection_monitor:start_monitoring:18 - Starting database connection monitor (interval: 60s)
app-1 | 2026-05-07 11:09:21.752 | DEBUG | core.database.connection_monitor:start_monitoring:42 - Database connection is healthy

```

Рисунок 3.2 – Запуск застосунку на сервері

На основі розробленого конфігураційного файлу docker-compose.yml здійснено контрольне розгортання системи на VPS-сервері. Після проведення етапу налагодження було підтверджено коректність функціонування всіх

сервісів. Результати перевірки працездатності системи та стан активності сервісів наведено на рисунку 3.3.

```

root@Ubuntu-2494-noble-amd64-base /projects/chyzik-bot # docker compose ps
NAME                IMAGE              COMMAND              SERVICE    CREATED    STATUS      PORTS
chyzik-bot-app-1    chyzik-bot-app    "sh /start.sh"      app        8 days ago Up 17 minutes 0.0.0.0:6667->8000/tcp, [::]:6667->8000/tcp
chyzik-bot-db-1     mysql:8.0         "docker-entrypoint.s db        9 days ago Up 9 days (healthy) 33060/tcp, 0.0.0.0:3313->3306/tcp, [::]:3313->3306/tcp
chyzik-bot-redis-1  redis:alpine      "docker-entrypoint.s redis    9 days ago Up 9 days      0.0.0.0:6388->6379/tcp, [::]:6388->6379/tcp
root@Ubuntu-2494-noble-amd64-base /projects/chyzik-bot #

```

Рисунок 3.3 – Стан сервісів на сервері

Після підготовки середовища було реалізовано TelegramSource (ліст. 3.5). Цей компонент відповідає за створення об'єкта бота, реєстрацію Dispatcher, встановлення webhook і приймання оновлень від Telegram. Під час запуску застосунок видаляє попередній webhook і встановлює новий за адресою домену, вказаного в налаштуваннях. Це потрібно, тому що Telegram надсилає повідомлення лише на одну активну webhook-адресу.

Лістинг 3.5 – Ініціалізація TelegramSource та встановлення webhook

```

class TelegramSource(BaseSource):
    @property
    def webhook_url(self):
        return "/telegram_webhook"

    def __init__(self):
        self.bot = Bot(
            token=settings.BOT_TOKEN,
            default=DefaultBotProperties(parse_mode=ParseMode.HTML),
        )
        self.dp = Dispatcher()
        self.register_handlers()

    async def on_startup(self, app):
        await self.bot.delete_webhook()
        await self.bot.set_webhook(settings.DOMAIN + self.webhook_url)
        self.me = await self.bot.get_me()

```

кінець лістингу 3.5

Основний webhook-обробник приймає JSON від Telegram, перетворює його на Update і передає в Dispatcher. Якщо під час обробки виникає виняток, система все одно повертає Telegram відповідь зі статусом 200. Це зроблено для того, щоб Telegram не повторював одне й те саме повідомлення багато разів у разі внутрішньої помилки.

Першим користувацьким сценарієм стала команда `/start`. Вона використовується не лише як привітання, а і як перевірка правильності webhook: якщо команда доходить до бота й користувач отримує відповідь, значить Telegram Bot API, домен, сервер і обробники працюють коректно (рис. 3.4).

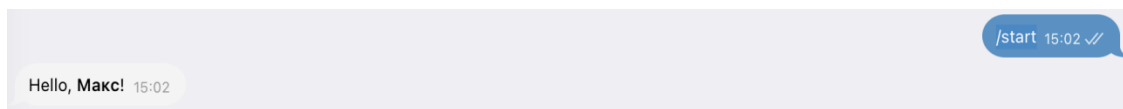


Рисунок 3.4 – Перше повідомлення після команди `/start`

Окрім стандартних сценаріїв, було передбачено та реалізовано алгоритм обробки вхідних повідомлень у випадках, коли користувач ініціює діалог без використання команди `/start`. Приклад коректної реакції системи на таке звернення наведено на рисунку 3.5.

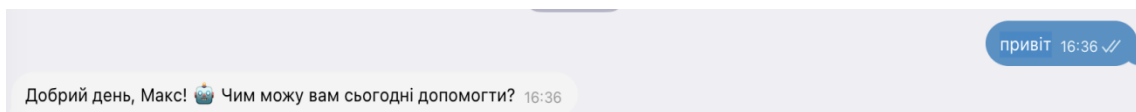


Рисунок 3.5 – Обробка простого першого повідомлення

Основний обробник повідомлень виконує кілька практичних дій. Спочатку він записує запит у журнал, потім створює або знаходить користувача, перевіряє, чи було повідомлення відповіддю на попередню відповідь бота, і лише після цього запускає workflow. Якщо повідомлення надійшло у груповому чаті, бот реагує тільки тоді, коли його згадали через username (ліст. 3.6). Це запобігає випадковій обробці всіх повідомлень у групі.

Лістинг 3.6 – Маршрутизація Telegram-повідомлення

```

async def message_handler(self, message: Message) -> None:
    await self.source.log_request(request_object=message)
    user = await self.source.process_user(message)

    is_mentioned = await self.source.check_if_mentioned(message)
    is_reply_to_bot = (
        message.reply_to_message
        and message.reply_to_message.text
        and message.reply_to_message.from_user.id == self.source.me.id
    )

```

```

)

if is_reply_to_bot:
    await self._handle_reply_to_bot(message, user)
elif message.chat.id == message.from_user.id or is_mentioned:
    await self._handle_direct_message(message, user)

```

кінець лістингу 3.6

У системі також реалізовано контроль доступу. Якщо користувач не схвалений у базі даних, бот не запускає workflow і повертає повідомлення про відсутність дозволу. Це важливо, оскільки бот має доступ до робочих сервісів, а отже не повинен виконувати запити будь-якого користувача, який випадково знайшов бота.

Для підтвердження роботи цього механізму в розділ доцільно вставити скріншот діалогу, де користувач без доступу надсилає повідомлення боту, а система повертає відповідь про відсутність дозволу. Такий скріншот демонструє не лише інтерфейс, а й практичну роботу обмеження доступу (рис. 3.6).

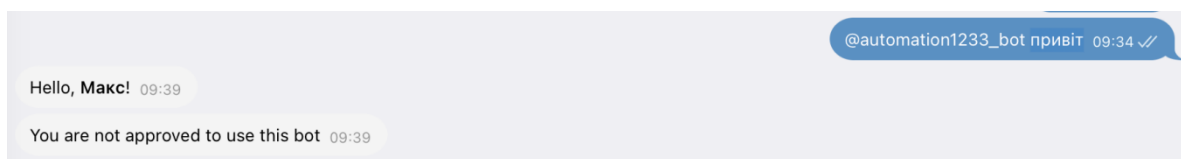


Рисунок 3.6 – Відповідь бота для користувача без доступу

Головна відмінність AutomationBot від простого Telegram-бота з командами полягає у workflow-архітектурі. TelegramSource не створює задачі самостійно і не знає деталей Gmail або Google Calendar. Його завдання – прийняти повідомлення і передати підготовлений текст у WorkflowEngine. Саме WorkflowEngine визначає, який сценарій потрібно виконати.

WorkflowEngine готує повідомлення, додає контекст попередньої відповіді, якщо користувач відповідає на повідомлення бота, передає текст в IntentRecognizer і отримує список дій (ліст. 3.7). Якщо AI не зміг визначити дію, повертається fallback-відповідь. Якщо дію визначено, рушій створює екземпляр потрібного workflow, передає йому користувача та виконує сценарій.

Лістинг 3.7 – Основний алгоритм WorkflowEngine

```

async def execute(self, message: str, user: Any) -> OutputMessage:
    processed_message = self._prepare_message(message)
    actions = await self.get_actions(processed_message)

    if self._is_fallback_action(actions):
        return OutputMessage(actions[0].data["fallback"], {})

    self.context = WorkflowContext(
        user_id=user.id,
        message_id=self.message_id,
        source_name=self.from_source.name,
        thread_id=self.thread_id,
    )

    responses = await self._execute_all_workflows(
        actions, processed_message, user
    )
    return await self.format_response(processed_message, responses, user)

```

кінець лістингу 3.7

Для ідентифікації намірів користувача в системі застосовано інтелектуальний модуль IntentRecognizer на базі моделі Gemini. На відміну від традиційних систем з жорстко визначеним набором команд, дане рішення дозволяє опрацьовувати запити у довільній формі, такі як: «створи задачу перевірити webhook», «які листи прийшли сьогодні» або «що в календарі завтра». Модуль аналізує вхідне повідомлення, враховуючи поточну дату та перелік доступних бізнес-процесів (workflows), після чого генерує структуровану дію у форматі JSON.

Така архітектура поєднує гнучкість взаємодії (UX) із високим рівнем безпеки: система не виконує довільний код, а лише запускає попередньо зареєстровані workflow, перевіряючи валідність отриманого action та параметрів.

Ключовим функціональним сценарієм є автоматизоване створення задач у GooseCRM. Процес ініціюється текстовим запитом користувача природною мовою. Агент GooseTaskCreator на основі аналізу тексту формує метадані задачі (назву, опис, приналежність до проєкту та виконавця), а модуль GooseRequests забезпечує передачу даних через API до CRM-системи. У випадку неможливості однозначного визначення параметрів (наприклад, назви проєкту), бот запитує уточнювальну інформацію.

Процес успішної взаємодії користувача з ботом для постановки завдання відображено на рисунку 3.7, де продемонстровано роботу AI-модуля та workflow-рушія в реальному часі.

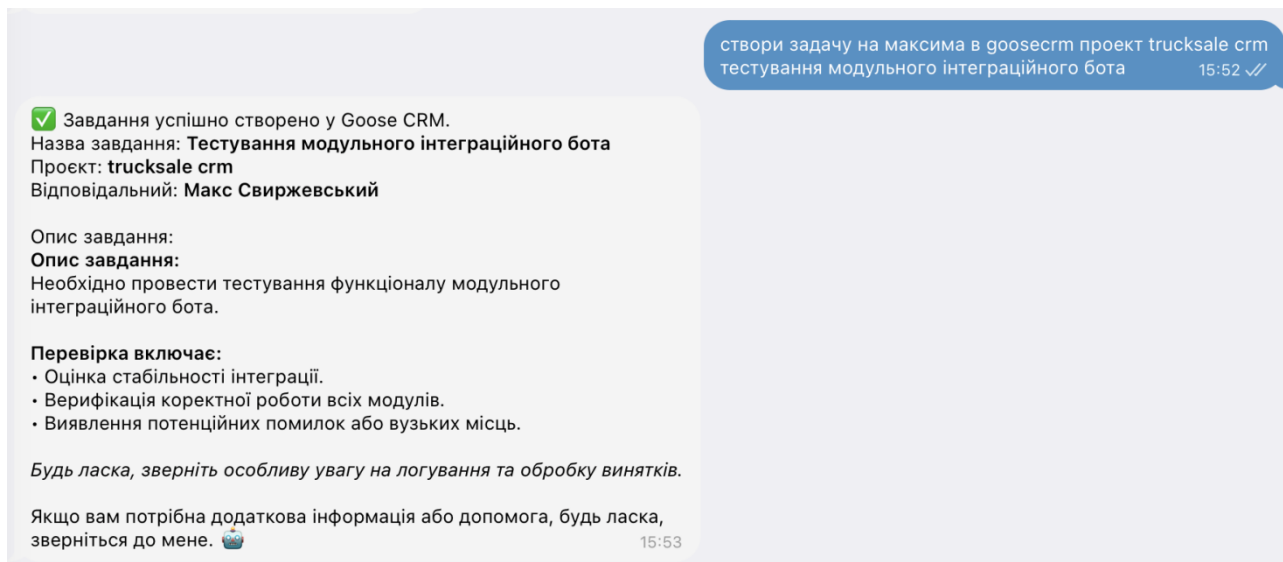


Рисунок 3.7 – Створення задачі через Telegram

Результат автоматичної генерації відповідного запису в інтерфейсі системи керування проєктами наведено на рисунку 3.8.

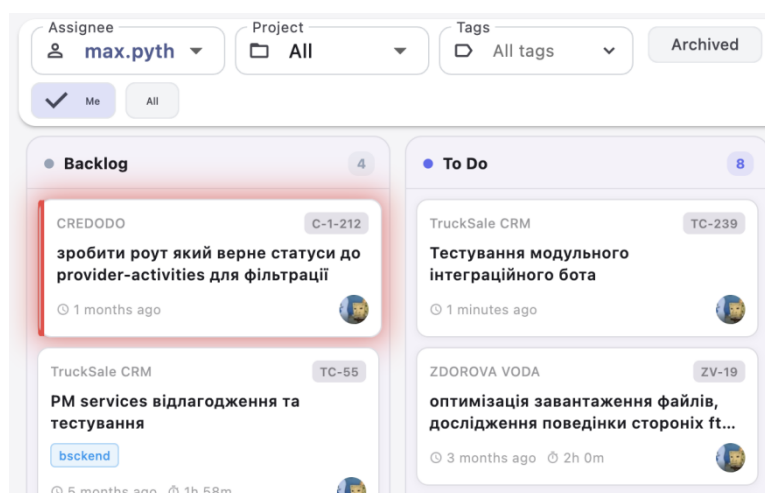


Рисунок 3.8 – Створена задача в GooseCRM

Іншим критично важливим сценарієм є отримання переліку поточних задач через ListGooseIssuesWorkflow. Система агрегує дані з GooseCRM, групує їх за проєктами та трансформує сирий JSON-відгук у читабельний для

месенджера формат, що забезпечує оперативний контроль над робочим процесом безпосередньо з Telegram. Наочне представлення обробленого списку в Telegram-боті відображено на рисунку 3.9.

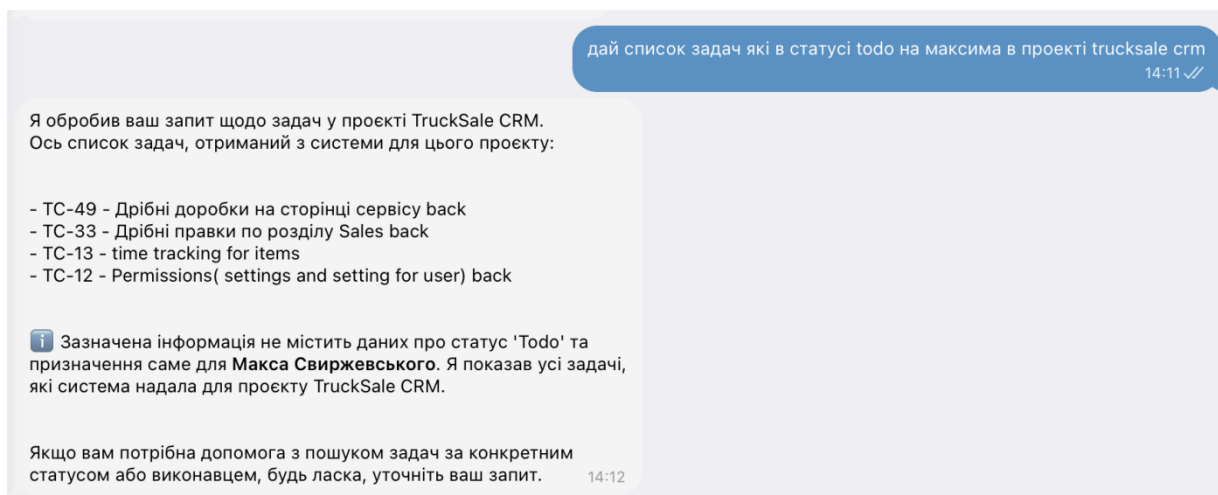


Рисунок 3.9 – Відповідь бота зі списком задач користувача

Окремим функціональним блоком системи є модулі взаємодії з Gmail та Google Calendar. Безпека доступу до конфіденційних даних користувача забезпечується протоколом Google OAuth 2.0, а його практична реалізація в сервісах Google базується на офіційних рекомендаціях Google Identity Platform [16]. При первинній авторизації система генерує файл доступу token.json, який надійно зберігається у захищеному каталозі secrets та використовується для подальших сесій без повторного підтвердження прав.

Реалізовані сценарії дозволяють користувачу оперативно отримувати робочу зведену інформацію безпосередньо у месенджері:

– сценарій Gmail: здійснює пошук та фільтрацію вхідної пошти за заданими часовими параметрами. Робота цього модуля, а саме процес формування та виводу переліку тем листів та їх відправників, відображена на рисунку 3.10;

– сценарій Google Calendar: базується на окремому workflow, який звертається до відповідного API для отримання розкладу подій на обраний день, виконання якого продемонстровано на рисунку 3.11.

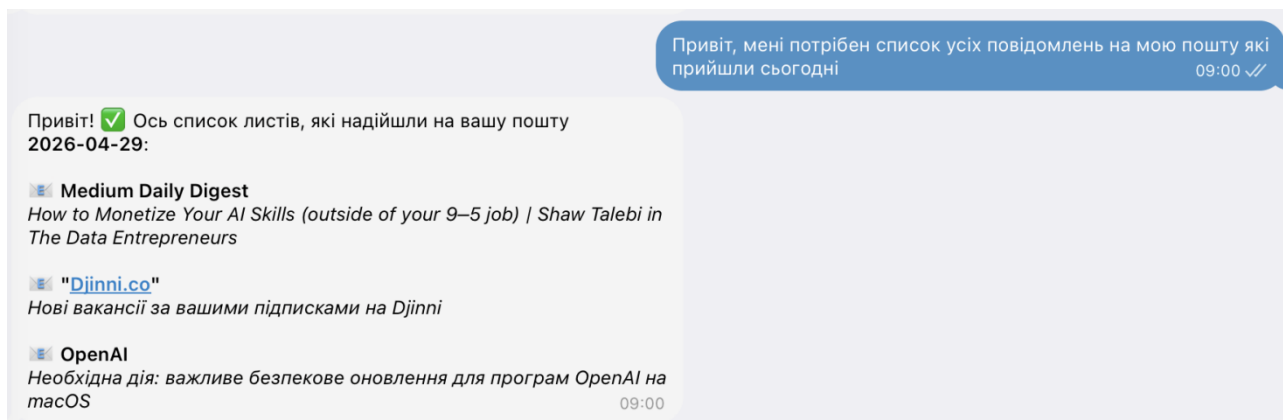


Рисунок 3.10 – Відповідь бота зі списком листів Gmail

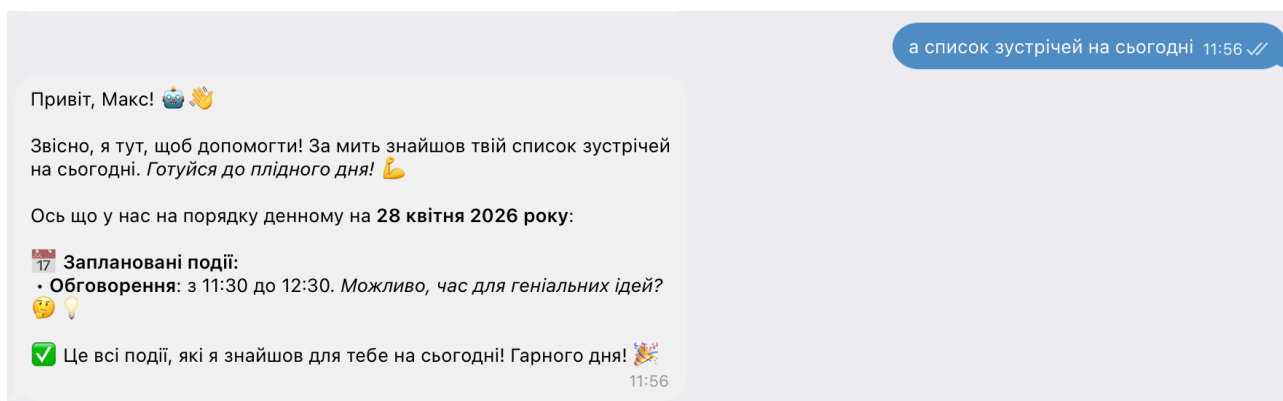


Рисунок 3.11 – Відповідь бота зі списком подій Google Calendar

Завдяки такій інтеграції реалізовано концепцію «єдиного вікна», де архітектурне розмежування різних джерел даних (пошта та календар) залишається непомітним для кінцевого користувача, забезпечуючи цілісний інтерфейс взаємодії.

3.2 Тестування та налагодження інформаційно-комп'ютерної системи

Після завершення розробки основних компонентів системи AutomationBot було проведено комплексне тестування, спрямоване на верифікацію коректності обробки повідомлень, стабільності роботи через Telegram Webhook та точності класифікації намірів користувача. Оскільки архітектура продукту є модульною, процес перевірки охоплював не лише фінальний інтерфейс взаємодії, а й

автономну роботу workflow-рушія, AI-агентів, систем збереження стану в Redis та MySQL, а також інтеграційних шлюзів із GooseCRM і сервісами Google.

На етапі перевірки серверної частини основна увага приділялася стабільності ініціалізації застосунку та надійності з'єднань із базами даних. Було впроваджено механізм контролю, який блокує запуск системи у разі недоступності критичних вузлів, таких як Redis або MySQL, що запобігає частковому функціонуванню сценаріїв. Для моніторингу працездатності в реальному часі використовувався ендпоінт /health та аналіз логів контейнерів, що дозволило підтвердити коректність розгортання всіх сервісів у середовищі Docker.

Під час налагодження взаємодії з Telegram було виявлено необхідність специфічної обробки повідомлень у групових чатах. Зокрема, реалізовано алгоритм фільтрації тексту від згадувань юзернейму бота, що забезпечило подальшу коректну роботу AI-модуля. Паралельно з цим тестувалася система розмежування прав доступу, яка гарантує, що конфіденційні робочі сценарії та доступ до API зовнішніх сервісів мають лише авторизовані користувачі, занесені до внутрішньої бази даних системи.

Значна частина тестування була присвячена валідації модульної архітектури. Чіткий поділ функцій між класами TelegramSource, WorkflowEngine та прикладними workflows дозволив ізолювати процеси обробки даних від транспортного рівня. Така декомпозиція спростила пошук та усунення помилок, оскільки будь-які розбіжності у форматі даних із зовнішніми API, наприклад, при створенні задач у GooseCRM, локалізувалися у відповідних модулях інтеграції, не впливаючи на загальну стабільність бота.

Окремим і найскладнішим етапом стало налагодження AI-модуля на базі моделі Gemini. У процесі тестування було оптимізовано структуру системного промпту для забезпечення стабільного повернення даних у форматі JSON. Важливим покращенням стало впровадження передачі часового контексту, що навчило модель правильно інтерпретувати запити типу «сьогодні» або «на завтра». Вибір моделі Gemini обґрунтований її здатністю точно витримувати

задану структуру відповідей, що є критичним для автоматичного запуску програмних сценаріїв без участі людини.

Тестування функціональних сценаріїв створення та перегляду задач підтвердило ефективність обробки запитів природною мовою. Було встановлено, що система успішно ідентифікує наміри користувача навіть при використанні синонімів або технічних формулювань. При реалізації виводу списку задач було додано етап форматування даних, завдяки якому сирий JSON-відгук від CRM-системи трансформується у читабельний вигляд із групуванням за проектами, що значно покращує досвід користувача в інтерфейсі месенджера.

На фінальних етапах було протестовано обробку мультимедійного контенту та складні інтеграції з Gmail і Google Calendar. Перевірка інтеграції з сервісами Google підтвердила надійність механізму OAuth 2.0 та коректність використання токенів доступу. У підсумку, проведене тестування підтвердило повну працездатність основних сценаріїв автоматизації та готовність системи до практичного використання.

3.3 Проектування структури бази даних

Під час розроблення програмного застосунку AutomationBot важливим етапом стало проектування структури бази даних, оскільки саме база даних забезпечує збереження основної службової інформації, необхідної для коректної роботи системи. Незважаючи на те, що застосунок не є класичною інформаційною системою з великою кількістю сутностей, база даних у ньому виконує важливу роль. Вона використовується для збереження даних про користувачів, журналювання запитів, фіксації результатів обробки повідомлень та підтримки зв'язку між окремими інтеграціями.

Оскільки AutomationBot орієнтований на взаємодію з користувачами через Telegram, а також на роботу із зовнішніми сервісами, структура бази даних була спроектована з урахуванням простоти, надійності та можливості подальшого розширення. Основною метою було не створення надмірно складної схеми, а

формування такої структури, яка покриває реальні потреби застосунку та не ускладнює його підтримку. Такий підхід є доцільним для ботів та сервісних застосунків, де значна частина інформації надходить із зовнішніх API, а локальна база використовується переважно для збереження службових даних.

У проєкті використовується реляційна база даних MySQL. Вибір реляційної моделі пояснюється тим, що дані застосунку мають чітку структуру, а окремі сутності можуть бути представлені у вигляді таблиць із визначеними полями. Для взаємодії з базою даних застосовується ORM-бібліотека Peewee, яка дозволяє описувати таблиці у вигляді моделей мовою Python. Завдяки цьому робота з базою даних стає більш зручною, оскільки розробник може оперувати класами та об'єктами, а не формувати SQL-запити вручну для кожної операції.

Усі моделі бази даних наслідуються від базової моделі BaseModel. Вона використовується як спільна основа для інших сутностей і містить посилання на підключення до бази даних. Такий підхід дозволяє централізовано керувати параметрами з'єднання та не дублювати однакову логіку в кожній моделі. Базова модель є важливою частиною архітектури, оскільки забезпечує єдиний принцип взаємодії всіх таблиць із базою даних.

Однією з головних сутностей системи є модель користувача. Вона призначена для збереження інформації про користувачів Telegram, які взаємодіють із ботом. Для кожного користувача зберігається його Telegram ID, ім'я користувача, ім'я, прізвище та ознака підтвердження доступу. Поле `telegram_id` дозволяє ідентифікувати користувача у межах Telegram, а поле `username` використовується для зручного відображення або пошуку користувача. Поле `is_approved` має логічний тип і використовується для контролю доступу до функцій бота. Це дає змогу обмежити використання системи лише дозволеним користувачам.

Окрему роль у структурі бази даних виконує таблиця журналювання джерел даних. Вона представлена моделлю SourceLog і використовується для збереження інформації про події, що надходять із різних джерел. У цій таблиці передбачено поля для назви джерела, типу події та самих даних. Оскільки дані

можуть мати різний формат і обсяг, для їх збереження використовується текстове поле. Такий підхід дозволяє фіксувати службову інформацію без необхідності створювати окрему таблицю для кожного типу події. Це особливо зручно для системи, яка взаємодіє з різними зовнішніми сервісами та може отримувати неоднорідні дані.

Для обробки запитів через API передбачена модель `APIRequestLog`. Вона використовується для фіксації вхідних запитів, параметрів, тексту повідомлення, ідентифікатора користувача та часу створення запису. Наявність такої таблиці є важливою з погляду контролю роботи системи, оскільки вона дозволяє відстежувати, які запити надходили до застосунку, які дії виконувалися та коли саме це відбулося. У майбутньому такі журнали можуть використовуватися для аналізу помилок, перевірки активності користувачів або вдосконалення логіки обробки запитів.

Проектування структури бази даних виконувалося з урахуванням принципу мінімальної достатності. Це означає, що до схеми були включені лише ті таблиці та поля, які справді потрібні для роботи застосунку. У системі відсутні зайві сутності, які не використовуються в бізнес-логіці, оскільки їх наявність ускладнила б підтримку проєкту та збільшила б ризик помилок. Водночас структура залишає можливість для розширення. У разі потреби до неї можна додати нові таблиці для збереження історії діалогів, ролей користувачів, налаштувань інтеграцій або статистики виконання команд.

Важливим аспектом є також організація підключення до бази даних. У проєкті реалізовано окремий механізм ініціалізації з'єднання з MySQL. Він передбачає повторні спроби підключення у випадку тимчасових помилок або втрати з'єднання. Це підвищує стабільність роботи застосунку, особливо в умовах розгортання через Docker, де база даних і основний застосунок запускаються як окремі сервіси. Такий підхід дозволяє уникнути збоїв під час старту системи, коли база даних може бути ще не повністю готовою до приймання запитів.

Логічна організація бази даних AutomationBot базується на розподілі інформації за кількома функціональними напрямками. До першої групи належать дані користувачів, що використовуються для ідентифікації та розмежування прав доступу. Друга група включає журнали подій і запитів, необхідні для моніторингу працездатності системи. Третя група охоплює службову інформацію інтеграцій, яка забезпечує взаємодію із зовнішніми сервісами. Четверта група містить результати аналізу повідомлень, зокрема відомості про виявлені небажані терміни. Таке структурування спрощує розуміння ролі кожної таблиці та полегшує подальшу модернізацію системи.

Структура бази даних AutomationBot є лаконічною, проте її функціональних можливостей достатньо для повноцінної роботи застосунку. Вона гарантує надійне збереження профілів користувачів, детальне журналювання процесів, стабільну підтримку інтеграційних модулів та фіксацію результатів обробки вхідних даних. Застосування СУБД MySQL у поєднанні з ORM Peewee дозволяє ефективно поєднати переваги реляційної моделі зі зручністю програмного керування об'єктами. Розроблена архітектура бази даних повністю відповідає поточним технічним вимогам та дозволяє масштабувати систему в майбутньому без необхідності докорінної зміни її структури.

3.4 Захист інформаційно-комп'ютерної системи

Забезпечення безпеки інформаційно-комп'ютерної системи «AutomationBot» є критично важливим аспектом реалізації, оскільки програмний продукт оперує конфіденційними даними з корпоративних сервісів, таких як Gmail, Google Calendar та CRM-системи. Захист реалізовано як багаторівневу структуру, що охоплює інфраструктурний, програмний та авторизаційний рівні. На інфраструктурному рівні безпека досягається шляхом використання технології Docker-контейнеризації, яка дозволяє ізолювати основний додаток, базу даних MySQL та кеш-сховище Redis у внутрішній віртуальній мережі. Це мінімізує зовнішній вплив на компоненти системи, а взаємодія з серверами

Telegram через захищений протокол HTTPS із шифруванням трафіку повністю нівелює загрозу перехоплення даних за допомогою атак типу «Man-in-the-Middle».

Програмний рівень захисту конфіденційних параметрів реалізовано через механізм використання змінних середовища у файлах формату `.env`, які винесені за межі основного репозиторію коду. Специфічні облікові дані, зокрема файли авторизації Google OAuth, розміщуються у відокремленому каталозі `secrets` із налаштованими обмеженнями прав доступу на рівні операційної системи. Для запобігання несанкціонованому доступу до активних сесій у сховищі Redis впроваджено політику автоматичного видалення тимчасового контексту користувача через механізм TTL. Це гарантує, що після години неактивності будь-які сліди діалогу або тимчасові дані будуть видалені з оперативної пам'яті сервера.

Контроль доступу до функціональних модулів системи базується на дворівневій ідентифікації запитів. Для взаємодії через месенджер застосовується спеціалізований фільтр `is_approved`, який звіряє унікальний ідентифікатор Telegram-користувача зі списком дозволених осіб у базі даних перед кожним запуском сценарію. Для вхідних HTTP-запитів через програмний інтерфейс реалізовано обов'язкову перевірку авторизаційного заголовка `X-API-Key`. У межах розробки було проведено детальний аналіз потенційних загроз, результати якого, разом із впровадженими технічними заходами захисту від витоку токенів, несанкціонованого доступу та помилок AI-розпізнавання, наведено в таблиці 3.1. Такий комплексний підхід до автентифікації та мінімізації ризиків дозволяє забезпечити цілісність інформаційного периметра системи на всіх рівнях її функціонування. Окреслені превентивні механізми гарантують високу стійкість архітектури до зовнішніх загроз та ізоляцію критичних модулів від несанкціонованих дій. Зрештою, впровадження цих засобів контролю доступу у поєднанні з валідацією вхідних пакетів даних дозволяє мінімізувати вектор потенційних атак і закладає надійний базис для безпечного масштабування інтеграційних шлюзів платформи.

Таблиця 3.1 – Ергономічні характеристики AutomationBot

Критерій	Реалізація в AutomationBot
Зручність введення	Підтримка природномовних текстових запитів через AI-обробку.
Читабельність	Лаконічні відповіді, автоматична агрегація та групування даних за проектами.
Контекстність	Можливість продовження діалогу через функцію «Reply» на повідомлення бота.
Обробка помилок	Зрозумілі повідомлення («немає доступу», «проект не знайдено») без технічних дамів.
Безпечна робота	Автоматичне обмеження функціоналу для неавторизованих користувачів.
Простота доступу	Використання Telegram-чату без розгортання окремого вебінтерфейсу.

3.5 Розробка документації для програмного продукту

Розробка документації до програмного продукту є невід’ємним етапом проектування інформаційної системи, що забезпечує умови для її коректного встановлення, конфігурування, експлуатації та подальшої технічної підтримки. Документація до системи AutomationBot сформована з урахуванням її архітектурних особливостей як серверного застосунку з інтерфейсом у месенджері Telegram та сформована в корені проєкту у файлі DEPLOYMENT_AND_USAGE.md. Основна увага приділена опису системних вимог, процедурі розгортання в ізолюваному середовищі, налаштуванню взаємодії з базами даних та специфікації інтеграційних інтерфейсів.

Функціонування AutomationBot у серверному середовищі висуває певні вимоги до апаратної частини. Мінімальні системні вимоги, необхідні для розгортання базової конфігурації, включають сервер під управлінням ОС Linux

із двоядерним процесором частотою від 1.2 ГГц, обсягом оперативної пам'яті не менше 2 ГБ та вільним дисковим простором від 10 ГБ. Стабільна робота інтеграційних сценаріїв та обробка медіафайлів потребує наявності встановленого програмного забезпечення Docker та Docker Compose, а також захищеного мережевого з'єднання з підтримкою протоколу HTTPS для забезпечення роботи Webhook-запитів від Telegram API.

Процедура встановлення та розгортання програмного продукту базується на використанні технології контейнеризації. На підготовчому етапі адміністратор здійснює налаштування змінних середовища у конфігураційних файлах, де визначаються токени доступу до Telegram API, параметри підключення до реляційної бази даних MySQL та нереляційного сховища Redis. Такий підхід дозволяє відокремити конфіденційні дані від програмного коду, підвищуючи загальний рівень безпеки системи та спрощуючи її міграцію між різними середовищами виконання.

На етапі ініціалізації бази даних MySQL здійснюється автоматичне створення структури таблиць, що необхідні для збереження ідентифікаторів користувачів, журналів подій та параметрів зовнішніх інтеграцій. Використання Redis як кешувального шару дозволяє підтримувати стан діалогів та зберігати контекст взаємодії, що є критично важливим для коректної роботи алгоритмів штучного інтелекту при уточненні запитів. Безпосередній запуск сервісів виконується через Docker Compose, що забезпечує синхронний підйом усіх компонентів системи та їх автоматичне мережеве поєднання.

Окремий аспект документації присвячено налаштуванню взаємодії з месенджером Telegram. Встановлення Webhook виконується автоматично під час старту основного модуля, що забезпечує отримання оновлень у режимі реального часу. Для кінцевих користувачів передбачено вбудовану систему навігації, яка активується командою старту та надає інструкції щодо подальшої роботи. Довідкові матеріали описують можливості використання як текстового введення, при цьому вказується на механізми обмеження доступу для

неавторизованих осіб, що реалізовано з метою захисту корпоративної інформації.

Технічна документація також містить опис прикладного програмного інтерфейсу (API), який дозволяє виконувати інтеграцію AutomationBot із сторонніми інформаційними системами. Специфікація API визначає структуру HTTP-запитів, обов'язковість передачі ключа авторизації у заголовках та формат JSON-об'єктів для обміну даними. Описана логіка обробки помилок та журналювання операцій дозволяє адміністратору здійснювати ефективний моніторинг стану системи та оперативно реагувати на збої у роботі зовнішніх сервісів.

Завершальна частина документації містить опис структури даних та рекомендації щодо регламентного обслуговування системи, включаючи перевірку цілісності бази даних та оновлення API-ключів. Хоча довідкові матеріали не інтегровані у вигляді окремого графічного вікна, їх подання у форматі технічного опису є оптимальним для цільової аудиторії – системних адміністраторів та розробників. У перспективі передбачено можливість розширення довідки через створення спеціалізованої команди допомоги безпосередньо у функціоналі бота.

Висновки до розділу 3

У третьому розділі представлено практичну реалізацію розробленої архітектури інтелектуальної системи на мові Python. Реалізовано Telegram-джерело з використанням aiogram для обробки повідомлень та Workflow Engine для координації бізнес-сценаріїв. Ключовим досягненням стала інтеграція AI-модуля на базі Gemini API, що забезпечує розпізнавання намірів користувача без використання жорстко заданих команд. Впроваджено асинхронні інтеграції із зовнішніми сервісами GooseCRM, Gmail та Google Calendar. Проведене тестування підтвердило стабільність роботи системи, коректність обробки помилок зовнішніх API та ефективність збереження контексту в Redis.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було повністю реалізовано поставлені завдання та отримано обґрунтовані науково-практичні результати. На першому етапі було проведено комплексний аналіз предметної області автоматизації сучасних робочих процесів, досліджено чинні аналогічні рішення на ринку та систематизовано сучасні підходи до побудови модульних інтеграційних систем відповідного класу. На основі проведеного аналізу встановлено, що критична фрагментація корпоративних сервісів, таких як CRM, таск-трекери та поштові клієнти, суттєво знижує загальну продуктивність праці через постійне перемикавання контексту користувача. Це дозволило обґрунтувати, що використання кросплатформеного месенджера Telegram як єдиної уніфікованої точки доступу у поєднанні з передовими технологіями штучного інтелекту є найбільш доцільним, гнучким та ефективним інструментом розв’язання цієї проблеми в сучасному цифровому середовищі.

На основі проведеного дослідження було сформульовано вичерпні функціональні та нефункціональні вимоги до модульної інтеграційної системи AutomationBot, які стали детермінованою базою для проєктування її системної архітектури. За допомогою інструментарію уніфікованої мови моделювання UML побудовано комплекс діаграм прецедентів, послідовності та класів. Ці моделі дозволили детально візуалізувати внутрішню логіку обробки вхідних подій, черговість виконання операцій, а також структуру взаємодії між подієво-орієнтованим ядром бота, високопродуктивними базами даних MySQL і Redis та зовнішніми шлюзами сторонніх сервісів, мінімізуючи ризики архітектурних помилок.

У межах практичної реалізації було розроблено програмний продукт – інтелектуальний Telegram-бот AutomationBot. Його архітектуру побудовано на базі мови програмування Python та асинхронного фреймворку aiogram, що забезпечує високу швидкість обробки великої кількості конкурентних запитів. У систему успішно інтегровано генеративну модель штучного інтелекту Gemini

API, яка виконує інтелектуальний семантичний аналіз неструктурованого тексту користувачів для динамічного визначення намірів без жорсткого кодування правил. При цьому було забезпечено повноцінну роботу застосунку як у приватних діалогах, так і в межах групових чатів, а також створено ізольований рушій координування сценаріїв для гнучкого підключення нових корпоративних API.

Для верифікації розробленого програмного комплексу було проведено комплексне тестування, у ході якого перевірено відповідність системи затвердженим функціональним критеріям. Шляхом симуляції інтенсивного потоку запитів підтверджено високу стійкість, швидкодію та відмовостійкість архітектури в асинхронному режимі функціонування. Крім того, виконаний аудит безпеки підтвердив повну ізоляцію критично важливих даних, таких як API-ключі та токени авторизації, від несанкціонованого доступу завдяки впровадженню превентивних засобів контролю, шифрування з'єднань HTTPS та контейнеризації середовища за допомогою Docker і Docker Compose.

Фінальним етапом роботи підготовлено комплексну та структуровану технічну документацію до розробленого програмного забезпечення. Вона містить вичерпний опис ієрархічної структури каталогів проєкту, специфікації розроблених інтеграційних інтерфейсів взаємодії, детальні інструкції щодо налаштування змінних середовища, а також готові конфігураційні файли для швидкого розгортання та автоматизації міграцій бази даних, що забезпечує легкість подальшого супроводу й масштабування системи.

Отже, мету кваліфікаційної роботи повністю досягнуто, оскільки розроблена система AutomationBot виконує всі задекларовані функції, демонструючи ефективність поєднання AI-технологій та месенджер-інтерфейсів для автоматизації бізнес-задач. Подальший розвиток проєкту вбачається у впровадженні розширеної аналітичної панелі для моніторингу транзакцій, поглибленні інтеграції з корпоративними ERP-системами та вдосконаленні моделей розпізнавання контексту тривалих багатокрокових діалогів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

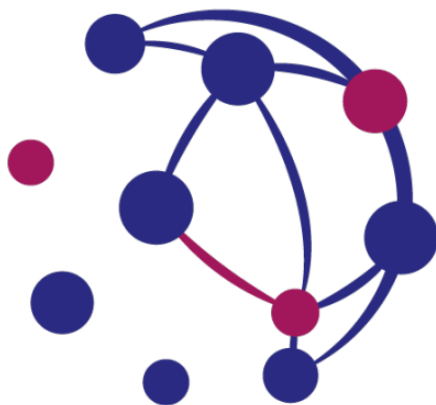
1. The economic potential of generative AI: The next productivity frontier / McKinsey Global Institute. URL: <https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier> (дата звернення: 03.03.2026).
2. Murty R. N., Dadlani S., Das R. B. How Much Time and Energy Do We Waste Toggling Between Applications? Harvard Business Review. URL: <https://hbr.org/2022/08/how-much-time-and-energy-do-we-waste-toggling-between-applications> (дата звернення: 30.05.2026).
3. Telegram reached 900 million monthly active users / Statista Data. URL: <https://www.statista.com/statistics/234038/telegram-messenger-mau-users/> (дата звернення: 19.03.2026).
4. n8n Documentation. URL: <https://docs.n8n.io/> (дата звернення: 19.03.2026).
5. Google AI for Developers. Gemini API Documentation. URL: <https://ai.google.dev/docs> (дата звернення: 25.03.2026).
6. Redis Documentation. URL: <https://redis.io/docs/> (дата звернення: 02.04.2026).
7. Python Language Reference, version 3.11. URL: <https://docs.python.org/3/> (дата звернення: 08.04.2026).
8. Aiogram Documentation. Welcome to aiogram documentation. URL: <https://docs.aiogram.dev/> (дата звернення: 14.04.2026).
9. Aiohttp Documentation. Async HTTP client/server for asyncio and Python. URL: <https://docs.aiohttp.org/> (дата звернення: 21.04.2026).
10. Gao P., Chen Q., Xie X., Wang C. Research on Performance Optimization of MySQL Database. 2023 IEEE 3rd International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA). IEEE, 2023. Vol. 3. P. 869–872. DOI: 10.1109/ICIBA56860.2023.10165291 (дата звернення: 24.04.2026).

11. Peewee ORM Documentation. URL: <http://docs.peewee-orm.com/> (дата звернення: 28.04.2026).
12. Docker Documentation. Docker Compose overview. URL: <https://docs.docker.com/compose/> (дата звернення: 04.05.2026).
13. Pan Zhang. Adaptive load balancing and fault-tolerant microservices architecture for high-availability web systems using docker and spring cloud. URL: <https://link.springer.com/article/10.1007/s42452-025-07320-7> (дата звернення: 07.05.2026).
14. Mairieli Wessel. The github development workflow automation ecosystems. URL: https://link.springer.com/chapter/10.1007/978-3-031-36060-2_8 (дата звернення: 10.05.2026).
15. Philippaerts P. et al. OAuch: Exploring Security Compliance in the OAuth 2.0 Ecosystem. Computers & Security. 2022. URL: https://www.researchgate.net/publication/364754298_OAuch_Exploring_Security_Compliance_in_the_OAuth_20_Ecosystem (дата звернення: 12.05.2026).
16. Google Identity. OAuth 2.0 for Web Server Applications. URL: <https://developers.google.com/identity/protocols/oauth2/web-server> (дата звернення: 15.05.2026).

ДОДАТКИ

Додаток А

Підтвердження апробації результатів кваліфікаційної роботи



MINTECH 2026

1st International Scientific and Practical Conference

**MODERN INFORMATION TECHNOLOGIES:
FROM THEORY TO PRACTICE**

BOOK OF ABSTRACTS

22–23 May, 2026

Lutsk National
Technical University



УДК 004

С 91

Міністерство освіти і науки України	Ministry of Education and Science of Ukraine
Волинська обласна рада	Volyn Regional Council
Луцька міська рада	Lutsk City Council
Луцький національний технічний університет	Lutsk National Technical University (Ukraine)
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»	National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" (Ukraine)
Національний університет «Львівська політехніка»	Lviv Polytechnic National University (Ukraine)
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут (м. Київ)	Kruty Heroes Military Institute of Telecommunications and Information Technology (Ukraine)
Рівненський державний гуманітарний університет	Rivne State University of the Humanities (Ukraine)
Державний торговельно-економічний університет (м. Київ)	State University of Trade and Economics (Ukraine)
Чернівецький національний університет імені Юрія Федьковича	Yuriy Fedkovych Chernivtsi National University (Ukraine)
Люблінська політехніка (Польща)	Lublin University of Technology (Poland)
Гронінгенський університет (Нідерланди)	University of Groningen (the Netherlands)
Кавказький університет (Грузія)	Caucasus University (Georgia)
Політехнічний університет Браганси (Португалія)	Polytechnic Institute of Bragança (Portugal)
Університет Сямень (Малайзія)	Xiamen University (Malaysia)
Мінюський університет (Португалія)	University of Minho (Portugal)

Рекомендовано до опублікування науково-технічною радою
Луцького національного технічного університету
(протокол № 10 від 27.05.2026 р.)

Редакційна колегія:

Кондіус Інна Степанівна, к.е.н., доцент (декан факультету комп'ютерних та інформаційних технологій)
Ліщина Наталія Миколаївна, к.т.н., доцент
Ліщина Валерій Олександрович, к.т.н., доцент
Андрущак Ігор Євгенович, д.т.н., професор
Тулашвілі Юрій Йосипович, д.п.н., професор
Козубцов Ігор Миколайович д.п.н., с.н.с.
Ящук Андрій Анатолійович, к.т.н., доцент (головний редактор)
Повстяна Юлія Славомірівна, к.т.н., доцент (відповідальний секретар)
Суринович Олена Миколаївна, к.т.н., доцент
Газдюк Катерина Петрівна, PhD, Чернівецький національний університет імені Юрія Федьковича
Дунець Роман Богданович, д.т.н., професор, Львівська політехніка
Komada Pawel, PhD, Politechnika Lubelska (Poland)
Shulga Artem, PhD, University of Groningen (the Netherlands)

С 91 Тези доповідей I Міжнародної науково-практичної конференції «Сучасні інформаційні технології: від теорії до практики (СінТех-2026)» (22-23 травня 2026 року). Луцьк: ЛНТУ, 2026. 1086 с.

Book of abstracts of the 1st International Scientific and Practical Conference "Modern Information Technologies: From Theory to Practice (MInTech-2026)" (22-23 May 2026). Lutsk: Lutsk National Technical University, 2026. 1086 p.

Матеріали I Міжнародної науково-практичної конференції «Сучасні інформаційні технології: від теорії до практики». Видання містить аналіз та результати досліджень, що стосуються актуальних питань інформаційних технологій в галузях освіти, науки та промисловості. Тези доповідей надано в авторській редакції. За фактичний матеріал і його інтерпретацію відповідають автори.

Proceedings of the 1st International Scientific and Practical Conference "Modern Information Technologies: From Theory to Practice". The publication contains analyses and research results addressing current issues in information technologies in the fields of education, science, and industry. The abstracts are published as submitted by the authors. The authors are responsible for the factual content and interpretation of the submitted materials.

Відповідальний за випуск: к.т.н., доцент Ліщина Н.М.

© Колектив авторів

ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ ВЕБЗАСТОСУНКІВ ІЗ ВИКОРИСТАННЯМ БАГАТОРІВНЕВОГО КЕШУВАННЯ.....	271
<i>Надашкевич Анастасія Геннадіївна</i>	
<i>Ліщина Наталія Миколаївна</i>	
ВИВЧЕННЯ ВПЛИВУ СТРУКТУРИ ВХІДНИХ ДАНИХ НА ДЕГРАДАЦІЮ АЛГОРИТМУ QUICK SORT.....	274
<i>Нестерук Ірина Юріївна</i>	
<i>Кіндєєва Анастасія Олександрівна</i>	
<i>Лучшева Оксана Вадимівна</i>	
АНАЛІЗ ЕФЕКТИВНОСТІ LLM-ІНСТРУМЕНТІВ У ПРОЦЕСАХ ГЕНЕРАЦІЇ ТА РЕФАКТОРИНГУ КОДУ	280
<i>Носик Євген Володимирович</i>	
МЕТОДИКА ПОРІВНЯННЯ ПЛАТФОРМ СПОСТЕРЕЖУВАНOSTІ ТА AIOPS У DEVOPS	284
<i>Пилипенко Даніїл Тарасович</i>	
ВИЗНАЧЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СЕРЕДНЬОЇ ЯКОСТІ ЗА ДОПОМОГОЮ ДОВІРЧОГО ЕЛІПСОЇДУ ДЛЯ НОРМАЛІЗОВАНИХ МЕТРИК RFC, SVO ТА WMC	287
<i>Приходько С.Б.</i>	
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПІДТРИМКИ ПРОКАТУ ГАБАРИТНИХ ПОБУТОВИХ ПРИЛАДІВ	291
<i>Піддубчак Данило Володимирович</i>	
<i>Льовкін Валерій Миколайович</i>	
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПІДТРИМКИ СПІЛЬНОЇ РИБОЛОВЛІ	293
<i>Пічко Герман Геннадійович</i>	
<i>Льовкін Валерій Миколайович</i>	
МОБІЛЬНІ ТЕХНОЛОГІЇ ЯК ІНСТРУМЕНТ УПРАВЛІННЯ ФІТНЕС-ОРІЄНТОВАНОЮ ДІЯЛЬНІСТЮ	295
<i>Рижок Соломія Юріївна</i>	
<i>Повстяна Юлія Славомірівна</i>	
ОГЛЯД ТА ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ІНСТРУМЕНТІВ АВТОМАТИЗАЦІЇ УПРАВЛІННЯ РИЗИКАМИ.....	299
<i>Романенко Микола Юрійович</i>	
<i>Ящук Андрій Анатолійович</i>	
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПІДТРИМКИ ОРЕНДИ М'ЯКИХ МЕБЛІВ.....	303
<i>Романік Данило Андрійович</i>	
<i>Льовкін Валерій Миколайович</i>	
ВЕБЗАСТОСУНКИ ЯК ЗАСІБ ЗАБЕЗПЕЧЕННЯ ЦИФРОВОЇ КОМУНІКАЦІЇ В РЕАЛЬНОМУ ЧАСІ	305
<i>Савлук Максим Олександрович</i>	
<i>Повстяна Юлія Славомірівна</i>	
РОЗРОБКА МОДУЛЬНОЇ ІНТЕГРАЦІЙНОЇ СИСТЕМИ НА ОСНОВІ TELEGRAM-БОТА З AI ДЛЯ ВЗАЄМОДІЇ З КОРПОРАТИВНИМИ СЕРВІСАМИ ТА АВТОМАТИЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ ВИКОРИСТОВУЮЧИ RUTRON.....	309
<i>Свиржевський Максим Анатолійович</i>	
<i>Ящук Андрій Анатолійович</i>	
АНАЛІЗ МОДЕЛЕЙ ПРОГНОЗУВАННЯ ПОТРЕБ У ФАРМАЦЕВТИЧНОМУ ВИРОБНИЦТВІ АПТЕК.....	313
<i>Тарновецька Ольга</i>	
<i>Дмитращук Кристина</i>	
ПЕРАНЖУВАННЯ РЕЗУЛЬТАТІВ ВІЗУАЛЬНОГО ПОШУКУ ТОВАРІВ НА ОСНОВІ КОМБІНОВАНОЇ ОЦІНКИ ВІЗУАЛЬНОЇ ПОДІБНОСТІ ТА КІЛЬКОСТІ СПІВПАДІНЬ.....	316
<i>Тиский Християн</i>	

РОЗРОБКА МОДУЛЬНОЇ ІНТЕГРАЦІЙНОЇ СИСТЕМИ НА ОСНОВІ TELEGRAM-БОТА З АІ ДЛЯ ВЗАЄМОДІЇ З КОРПОРАТИВНИМИ СЕРВІСАМИ ТА АВТОМАТИЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ ВИКОРИСТОВУЮЧИ PYTHON

Свиржевський Максим Анатолійович

Луцький національний технічний університет, здобувач вищої
освіти бакалаврського рівня спеціальності інженерія
програмного забезпечення, svyrzhevskiy.m2412@lntu.edu.ua

Яшук Андрій Анатолійович

Луцький національний технічний університет, доцент кафедри
інженерії програмного забезпечення, a.yashchuk@lutsk-
ntu.com.ua

DEVELOPING A PYTHON-BASED MODULAR INTEGRATION AI TELEGRAM BOT FOR CORPORATE SERVICES AND BUSINESS PROCESS AUTOMATION

Svyrzhevskiy Maksym Anatoliiovych

Lutsk National Technical University, bachelor's degree student in
Software Engineering, svyrzhevskiy.m2412@lntu.edu.ua

Yashchuk Andrii Anatoliiovych

Lutsk National Technical University, Associate Professor at the
Department of Software Engineering, a.yashchuk@lutsk-ntu.com.ua

The development of an intelligent Telegram bot for automating work processes is presented. The system combines webhook-based message processing, intent recognition using a large language model, workflow execution, voice message transcription, and integrations with external services such as CRM, Gmail, Google Calendar, Jira, and Worksection. The proposed modular architecture enables extension through new sources and workflows, while Redis-based state storage supports contextual interaction.

Keywords: Telegram bot, artificial intelligence, workflow automation, API integration, CRM, large language model.

У сучасних умовах цифрової трансформації організації дедалі частіше потребують інструментів, які дають змогу швидко

обробляти повідомлення, створювати задачі, отримувати інформацію з робочих сервісів і підтримувати комунікацію без переходу між багатьма окремими системами. Особливо актуальними є рішення на базі месенджерів, оскільки вони мають низький поріг входу для користувача та забезпечують оперативний доступ до функцій автоматизації.

Telegram є зручною платформою для реалізації таких рішень завдяки відкритому Bot API, підтримці webhook-механізму, роботі з текстовими й голосовими повідомленнями та можливості інтеграції з зовнішніми сервісами [1]. У межах розробленого проєкту створено Telegram-бота, який приймає запити користувачів, визначає їхній намір, запускає відповідні сценарії обробки та повертає результат у звичному діалоговому форматі.

Програмна реалізація побудована мовою Python із використанням асинхронного підходу. Для взаємодії з Telegram застосовано бібліотеку aiogram, що підтримує сучасну модель диспетчеризації повідомлень і зручну роботу з обробниками подій [2]. HTTP-рівень системи реалізовано на базі aiohttp, що дає змогу реєструвати webhook-адреси для різних джерел даних і обробляти запити без блокування виконання [3].

Ключовою особливістю системи є модульна архітектура, у якій джерела повідомлень та бізнес-сценарії відокремлені одне від одного. У проєкті передбачено джерела для Telegram, API-запитів, Jira, Worksection. Кожне джерело реєструє власний endpoint, а подальша обробка делегується workflow-рушію, який визначає потрібний сценарій відповідно до змісту повідомлення.

Для розпізнавання намірів користувача та підготовки структурованих даних використано AI-агентів на базі Gemini. Gemini API надає доступ до генеративних моделей і може застосовуватись для задач аналізу тексту, формування відповідей і виділення параметрів із природномовних запитів [4]. У розробленій системі це дає змогу перетворити повідомлення користувача на конкретну дію: створення задачі, перегляд списку задач, отримання листів або подій календаря.

Окрему увагу приділено підтримці контекстної взаємодії.

Якщо користувач відповідає на повідомлення бота, система відновлює попередній стан діалогу та може продовжити виконання відповідного сценарію. Для цього використовується Redis, який доцільний для швидкого зберігання тимчасових структур даних, станів і кешованої інформації [5]. Такий підхід підвищує природність діалогу й дозволяє реалізувати багатокрокові сценарії без втрати контексту.

Функціональність бота охоплює створення задач у GooseCRM, отримання списку задач за користувачем і статусом, перегляд листів Gmail за вказану дату, отримання подій Google Calendar, а також обробку голосових повідомлень. Інтеграція з Gmail API та Google Calendar API забезпечує доступ до робочих повідомлень і подій календаря через офіційні механізми авторизації OAuth та відповідні REST-методи [6, 7].

На рис. 1 подано узагальнену архітектуру розробленої системи. Вона демонструє рух запиту від джерела повідомлення до workflow-рушія, взаємодію з AI-агентами, зовнішніми сервісами та підсистемою зберігання стану.

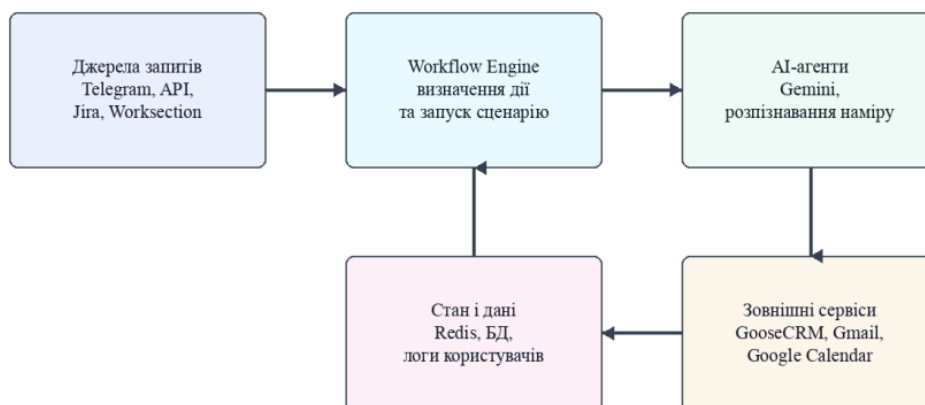


Рисунок 1. Узагальнена архітектура Telegram-бота

Важливою перевагою запропонованого рішення є можливість розширення. Додавання нового джерела або workflow не потребує зміни всієї системи: достатньо реалізувати окремий клас, зареєструвати його в налаштуваннях і описати очікуваний формат даних. Це спрощує супровід програмного продукту та зменшує ризик помилок під час подальшого розвитку.

Запропонована система також враховує питання контролю

доступу та надійності. У Telegram-джерелі передбачено перевірку схвалення користувача, логування запитів, обробку помилок без повторного надсилання webhook-подій і перевірку стану підключення до бази даних. Це робить застосунок придатним не лише для демонстраційного використання, а й для практичного впровадження у робочих командах.

Отже, розробка інтелектуального Telegram-бота для автоматизації робочих процесів є актуальним і практично значущим завданням. Реалізований проєкт поєднує можливості месенджера, генеративного штучного інтелекту, workflow-підходу та зовнішніх API. Така архітектура скорочує час виконання типових дій, централізує доступ до робочих сервісів і створює основу для подальшого масштабування системи.

Список використаних джерел

1. Telegram Bot API. Telegram APIs. URL: <https://core.telegram.org/bots/api> (дата звернення: 03.05.2026).
2. Welcome to aiogram's documentation. aiogram documentation. URL: <https://docs.aiogram.dev/en/v3.15.0/> (date of access: 03.05.2026).
3. Server. aiohttp documentation. URL: <https://docs.aiohttp.org/en/stable/web.html> (date of access: 03.05.2026).
4. Gemini API reference. Google AI for Developers. URL: <https://ai.google.dev/gemini-api/docs/api-overview> (date of access: 03.05.2026).
5. Redis Open Source. Redis Documentation. URL: <https://redis.io/docs/latest/get-started/> (date of access: 03.05.2026).
6. Method: users.messages.list. Gmail API. Google for Developers. URL: <https://developers.google.com/workspace/gmail/api/reference/rest/v1/users.messages/list> (date of access: 03.05.2026).
7. Events. Google Calendar API. Google for Developers. URL: <https://developers.google.com/workspace/calendar/api/v3/reference/events> (date of access: 03.05.2026).