МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

МАТВІЙЧУК Олег Олександрович

Програмний модуль обробки текстових запитів для рекомендацій фільмів у Telegram-боті / Software Module for Processing Text Queries for Movie Recommendations in a Telegram Bot

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма - Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-41
О.О. Матвійчук
_____

Науковий керівник
к.т.н., доцент П. Є. Биковий
_____

Кваліфікаційну роботу допущено до захисту
«___»_____ 20___ р.

Завідувач кафедри
_____ Н.В. Дзюбановська

ТЕРНОПІЛЬ – 2026

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність: 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.В. Дзюбановська
«____» _____20__р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Матвійчуку Олегу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Програмний модуль обробки текстових запитів для рекомендацій фільмів у Telegram-боті / Software Module for Processing Text Queries for Movie Recommendations in a Telegram Bot

керівник роботи к.т.н., доцент, П. Є. Биковий
затверджений наказом по університету від 01 грудня 2025 року № 894
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література, офіційна документація до бібліотек spaCy, python-telegram-bot та The Movie Database (TMDB) API.
4. Основні питання, які потрібно розробити:
- Проаналізувати методи обробки природної мови (NLP) та підходи до побудови сучасних рекомендаційних діалогових систем;
- Спроєктувати асинхронну архітектуру програмного модуля Telegram-бота та даталогічну модель локальної бази даних SQLite;
- Розробити гібридний NLP-алгоритм для точної екстракції темпоральних меж, рейтингів, розпізнавання інтенту та семантичного мапування жанрів;
- Програмно реалізувати систему, виконати її інтеграцію із зовнішнім API TMDB та провести комплексне тестування розроблених модулів.

5. Перелік графічного матеріалу у роботі:
- Загальна структурна схема інформаційних потоків програмного модуля;
- Схема алгоритму обробки природної мови;
- UML-діаграма класів програмної системи;
- UML-діаграма станів графічного інтерфейсу користувача.

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 01 грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Огляд літературних джерел відповідно до предметної області та складання плану роботи	до 01.01.2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02.2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 15.03.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05.2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06.2026 р.	

Студент _____ О.О. Матвійчук
підпис

Керівник роботи _____ к.т.н., доцент П.Є. Биковий
підпис

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний модуль обробки текстових запитів для рекомендацій фільмів у Telegram-боті» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом у 57 сторінок і містить 8 ілюстрацій, 4 таблиці, 3 додатки та 28 використаних джерел.

Метою кваліфікаційної роботи є розробка та програмна реалізація інтелектуального модуля обробки неструктурованих текстових запитів для персоналізованої рекомендації багатоформатного кіноконтенту у месенджері Telegram.

Методами дослідження є методи системного аналізу, обробки природної мови, теорії реляційних баз даних та об'єктно-орієнтованого проєктування.

Внаслідок виконання роботи розроблено асинхронну архітектуру програмного модуля з використанням мови Python та фреймворку python-telegram-bot. Створено гібридний NLP-конвеєр на базі векторної лінгвістичної моделі spaCy та системи регулярних виразів, що дозволяє виконувати токенізацію, семантичне мапування жанрів за допомогою методу семантичних якорів, розпізнавання логічних операторів (AND/OR) та екстракцію числових параметрів з вільного тексту. Локальна база даних SQLite забезпечує персоналізацію та фільтрацію історії переглядів користувачів під час інтеграції з API The Movie Database.

Розроблений програмний продукт є готовою до використання інформаційною системою, яка може бути розгорнута як самостійний сервіс або інтегрована в існуючі онлайн-кінотеатри для забезпечення швидкого інтелектуального текстового пошуку українською мовою.

Ключові слова: TELEGRAM-БОТ, ОБРОБКА ПРИРОДНОЇ МОВИ, NLP, РЕКОМЕНДАЦІЙНА СИСТЕМА, SPACY, THE MOVIE DATABASE, СЕМАНТИЧНИЙ АНАЛІЗ.

ANNOTATION

Qualification work on the topic «Software Module for Processing Text Queries for Movie Recommendations in a Telegram Bot» for obtaining a bachelor's degree in specialty 122 «Computer Science» of the educational program «Computer Science» is written on 57 pages and contains 8 illustrations, 4 tables, 3 attachments, and 28 references.

The aim of the qualification work is to develop and programmatically implement an intelligent module for processing unstructured text queries for personalized multi-format movie content recommendations in the Telegram messenger.

Research methods include system analysis, natural language processing (NLP), relational database theory, and object-oriented design.

As a result of the work, an asynchronous software module architecture was developed using Python and the python-telegram-bot framework. A hybrid NLP pipeline was created based on the spaCy vector linguistic model and regular expressions, enabling tokenization, semantic genre mapping via semantic anchors, logical operators recognition (AND/OR), and numerical parameters extraction from free text. A local SQLite database provides personalization and user viewing history filtering during integration with The Movie Database API.

The developed software product is a ready-to-use information system that can be deployed as a standalone service or integrated into existing online cinemas to provide fast intelligent text search in the Ukrainian language.

Keywords: TELEGRAM BOT, NATURAL LANGUAGE PROCESSING, NLP, RECOMMENDER SYSTEM, SPACY, THE MOVIE DATABASE, SEMANTIC ANALYSIS.

ЗМІСТ

ВСТУП

Швидкий розвиток цифрових технологій та стрімінгових сервісів призвів до безпрецедентного накопичення мультимедійного контенту, що створює для користувачів проблему інформаційного перевантаження. Паралельно відбувається зміна парадигми користувацьких інтерфейсів: на зміну складним графічним меню приходять діалогові системи у популярних месенджерах, зокрема в Telegram [1]. Однак більшість існуючих рекомендаційних ботів покладаються на жорстко задані команди, кнопкові меню або примітивний пошук за ключовими словами, які не здатні розуміти «живу» мову з її сленгом та неформальною лексикою. Відповідно, виникає гостра потреба у розробці оптимізованих інтелектуальних NLP-модулів, здатних ефективно перетворювати вільні текстові запити українською мовою у структуровані параметризовані виклики до зовнішніх баз даних.

Мета дослідження – розробка та програмна реалізація інтелектуального модуля обробки неструктурованих текстових запитів для персоналізованої рекомендації багатоформатного кіноконтенту у месенджері Telegram.

Завдання дослідження:

1) проаналізувати предметну область, методи побудови рекомендаційних систем та алгоритми розуміння природної мови;

2) спроєктувати асинхронну архітектуру програмного модуля, інфологічну модель локальної бази даних та загальні інформаційні потоки;

3) розробити гібридний NLP-алгоритм для точної екстракції темпоральних меж, рейтингу, розпізнавання жанрів за допомогою векторної семантики та класифікації інтенту користувача;

4) програмно реалізувати інформаційну систему з використанням об'єктно-орієнтованого підходу, мови Python, асинхронного фреймворку python-telegram-bot, лінгвістичної моделі spaCy та СУБД SQLite;

5) провести комплексне модульне та інтеграційне тестування розробленого програмного забезпечення на предмет точності розпізнавання намірів та відмовостійкості.

Об’єкт дослідження – процес інтелектуальної обробки природномовних запитів для пошуку та рекомендації мультимедійного контенту в системах з діалоговим інтерфейсом.

Предмет дослідження – алгоритмічне та програмно-технологічне забезпечення для розпізнавання інтенту, семантичного мапінгу тексту та асинхронної взаємодії із зовнішніми інформаційними системами (TMDB API).

Розроблений програмний модуль є самостійним, готовим до використання додатком, який значно спрощує процес пошуку релевантного кіноконтенту. Запропоновані архітектурні рішення та побудований гібридний NLP-конвеєр можуть бути легко масштабовані або інтегровані в існуючі стрімінгові платформи, онлайн-кінотеатри та інформаційні сервіси для суттєвого покращення користувацького досвіду шляхом впровадження швидкого інтелектуального текстового пошуку українською мовою.

Результати роботи були апробовані на І Міжнародній науково-практичній конференції студентів та молодих учених «Intelligent Computing Systems and Project Management (ICSPM 2026)», м. Тернопіль, Україна. Текст публікації наведено у Додатку Б [2].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕОРЕТИКО-МЕТОДОЛОГІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ

1.1 Аналіз предметної області

Розвиток цифрових технологій та популяризація стрімінгових сервісів призвели до безпрецедентного накопичення мультимедійного контенту. Сучасні користувачі щодня стикаються з феноменом інформаційного перевантаження – станом, за якого обсяг доступних даних значно перевищує когнітивні здібності людини до їх ефективного опрацювання. Наявність десятків тисяч фільмів у базах даних (наприклад, The Movie Database – TMDB) унеможливлює ручний пошук релевантного контенту без застосування спеціалізованих алгоритмів фільтрації. Понад 80% взаємодій користувачів із контентом на сучасних платформах відбувається завдяки алгоритмічним рекомендаціям, що мінімізує час на пошук та максимізує задоволеність клієнтів.

Паралельно з розвитком рекомендаційних алгоритмів спостерігається зміна парадигми користувацьких інтерфейсів: від класичних графічних до діалогових систем. Графічні системи вимагають навігації складною ієрархією веб-форм, натомість діалоговий інтерфейс дозволяє стиснути багатовимірні вимоги в одне речення природною мовою. Платформа Telegram на сьогодні є одним із найбільш розповсюджених середовищ для розгортання таких систем, гарантуючи високу швидкість, відсутність потреби в додаткових додатках та кросплатформність.

Однак інтеграція інтелектуальних систем у месенджер породжує специфічні науково-технічні виклики. Діалоговий інтерфейс спонукає до вільного формулювання думок: користувачі використовують сленг, скорочення, ігнорують пунктуацію та припускаються помилок. Відповідно, об'єктом дослідження стає «зашумлений» потік неструктурованих даних. Крім того, очікування миттєвої відповіді виключає використання надто повільних архітектур.

Виходячи з цього, актуальність роботи полягає у необхідності створення оптимізованого інтелектуального прошарку на базі методів обробки природної мови (NLP). Цей модуль має виступати сполучною ланкою між вільно

сформульованим запитом українською мовою та жорстко параметризованими інтерфейсами зовнішніх кінобаз.

Для обґрунтування вибору архітектурних рішень необхідно здійснити теоретичний аналіз фундаментальних принципів побудови рекомендаційних систем [3]. Глобально вони класифікуються на контентні, колаборативні та гібридні, кожна з яких має специфічні математичні моделі та сферу застосування [4].

Контентна фільтрація базується на схожості внутрішніх характеристик об'єктів (жанр, режисер, опис). Математично кожен фільм представляється як багатовимірний вектор у просторі ознак. Для текстових описів традиційно застосовується метод частотно-зворотної частотності документа (TF-IDF) [5]:

$$TF - IDF(t,d) \ = \ TF(t,d) \ * \ log\left(\frac{N}{DF(t)}\right), \tag{1.1}$$

де $TF(t,d)$ – частота появи терміна t у документі d,

N – загальна кількість фільмів,

$DF(t)$ – кількість описів фільмів, у яких зустрічається даний термін

Ступінь схожості між вектором фільму та профілем користувача обчислюється за метрикою косинусної близькості [3]:

$$S_C(A,B) \ = \ \frac{A * B}{||A||*||B||} \ = \ \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{1.2}$$

де $S_C()$ – ступеня косинусної схожості,

A – вектор фільму,

B – вектор профілю користувача,

де A_i – значення і-ї ознаки у багатовимірному векторі фільму A,

B_i – значення і-ї ознаки у векторі профілю користувача B,

n – загальна розмірність векторного простору ознак

Перевагою методу є вирішення проблеми «холодного старту» для нових фільмів, але він страждає від надмірної спеціалізації, замикаючи користувача в «інформаційній бульбашці».

Колаборативна фільтрація діє за принципом соціальної схожості, аналізуючи виключно матрицю взаємодій «користувач-об'єкт». Використовуючи алгоритми машинного навчання, зокрема сингулярний розклад матриць, система знаходить приховані фактори взаємодії.

Незважаючи на здатність знаходити неочевидні рекомендації, цей метод є абсолютно непридатним для ізольованих чат-ботів через проблему «холодного старту» для користувачів: алгоритм не здатен нічого порекомендувати новій людині без достатньої історії її оцінок.

Сучасний стан розвитку передбачає використання гібридних систем та методів глибинного навчання [6, 7]. Наприклад, нейронна колаборативна фільтрація замінює внутрішній добуток векторів на багатошарові перцептрони.

Окремим напрямком є парадигма LLM4Rec – використання великих мовних моделей архітектури Transformer. Попри безпрецедентну якість, використання хмарних LLM у Telegram-ботах спричиняє мережеві затримки, фінансові витрати та ризик фактологічних помилок («галюцинацій»). Тому для розробки доцільно використовувати локальні легковагові NLP-моделі та гібридні алгоритми. Порівняльну характеристику базових алгоритмів наведено в таблиці 1.1.

Перехід до діалогової моделі вимагає впровадження модуля розуміння природної мови для трансформації вводу у структурований JSON-запит до API. У сучасній лінгвістиці існує еволюційний спектр методів обробки:

1) детерміновані системи: базуються на регулярних виразах (Regex). Забезпечують абсолютну точність при пошуку дат, але нестійкі до варіативності людської мови;

2) векторні простори: дозволяють репрезентувати слово як вектор і знаходити семантичну близькість синонімів (Word2Vec, GloVe) [8];

3) моделі архітектури Transformer: моделі типу BERT обчислюють динамічне векторне подання, але потребують значних ресурсів для інференсу.

Таблиця 1.1 – Порівняльна характеристика базових алгоритмів рекомендаційних систем

Тип алгоритму	Вхідні дані моделі	Ключовий математичний апарат	Переваги	Основні недоліки
Контентна фільтрація	Метадані фільмів, описи	Векторний простір, TF-IDF, косинусна близькість	Відсутність проблеми холодного старту;	Надмірна спеціалізація
Колаборативна фільтрація	Історія оцінок	Кореляція Пірсона	Здатність знаходити неочевидні рекомендації	Проблема холодного старту; розрідженість даних
Нейронні мережі	Нелінійні патерни	Багатошарові перцептрони	Виявлення прихованих часових трендів	Висока обчислювальна складність
LLM4Rec	Текстові промпти, контекст	Механізм само-уваги, генерація	Найкраще розуміння контексту	Висока вартість повільна генерація

У межах розробки доцільним є впровадження гібридного NLP-конвеєра, який поєднує швидкість регулярних виразів та адаптивність векторних просторів:

– екстракція дат та точних рейтингів: regex використовується для виявлення темпоральних меж (років випуску) та точних числових значень рейтингу;

– екстракція жанрів: обчислення косинусної близькості між токеном та еталонним вектором жанру;

– сентимент-аналіз: якісні характеристики («шедевр», «погано») мапляться на числові фільтри рейтинг.

Стратегії мапінгу природно-мовних висловлювань на параметри бази даних представлено в таблиці 1.2.

Таблиця 1.2 – Стратегії мапінгу природно-мовних висловлювань на параметри бази даних

Тип вхідної сутності	Приклад у тексті запиту користувача	Застосований NLP інструмент	Згенерований параметр для API
Темпоральна межа	«…зняті після 2018 року…»	Регулярні вирази (Regex)	primary_release_date.gte=2018-01-01
Емоційне забарвлення	«…хочу посміятися…»	Словник семантичного розширення	with_genres=35 (Comedy)
Оціночне судження	«…порадь якийсь шедевр…»	Словник сентименту	vote_average.gte=8.5
Семантичний жанр	«фантастичний бойовик…»	Векторне представлення слів	with_genres=878, 28

Реалізація такого конвеєра значно ускладнюється флективною природою української мови. Вільний порядок слів та багата словозміна роблять класичний алгоритм стемінгу (відсікання закінчень) деструктивним. Вирішення проблеми полягає у застосуванні глибокої лематизації – зведення словоформи до канонічної основи за допомогою нейронних мереж.

У промисловій розробці стандартом для таких задач є бібліотека spaCy [9]. Для української мови розроблено модель uk_core_news_lg, натреновану на великих відкритих корпусах. Вона містить векторні представлення, що дозволяє обчислювати семантичну схожість. Огляд відкритих україномовних текстових корпусів для NLP-моделювання наведено в таблиці 1.3.

Для компенсації неформального сленгу та морфологічних особливостей у розроблюваному модулі додатково застосовується гарантована евристика кореня: якщо лема токена має спільний корінь із семантичним якорем жанру, алгоритм ігнорує похибку векторної відстані та жорстко фіксує максимальну схожість.

Таблиця 1.3 – Огляд відкритих україномовних текстових корпусів для NLP-моделювання

Назва корпусу	Обсяг даних / Токенів	Застосування у NLP
Kobza	1.3 TB (60 млрд токенів)	Навчання великих мовних моделей (LLMs), Embeddings
UberText 2.0	> 5 GB	Тренування моделей класифікації, токенізації та розпізнавання іменованих сутностей
Brown-UK	1 млн токенів	Оцінка якості лематизаторів та тегерів частин мови

1.2 Огляд та аналіз існуючих програмних рішень

Дослідження ринку Telegram-ботів підтверджує поділ на піратські файлообмінники та інформаційні сервіси. Аналіз легальних інформаційних рішень виявив технологічну прогалину:

1) MovieFinderBot: базується на примітивному пошуку за підрядком. Бот не розуміє контексту, не здатний обробляти критерії (жанри, роки) і видає помилки при хибному введінні;

2) CineDor Bot V3: інтегрує базу TMDB, але взаємодія базується на класичних Inline-клавіатурах. Користувач змушений натискати серію кнопок, що руйнує парадигму вільного діалогу;

3) IAI MovieBot: сучасна академічна розробка з повноцінною діалоговою архітектурою на базі трансформерів. Проте вона занадто перевантажена для масового Telegram-бота та має обмежену підтримку української мови.

Функціональний аналіз існуючих рекомендаційних Telegram-ботів зведено у таблиці 1.4.

Таблиця 1.4 – Функціональний аналіз існуючих рекомендаційних Telegram-ботів

Назва програмного рішення	Механізм введення	Розуміння інтенту (NLP)	Збереження історії
MovieFinderBot	Статичні команди	Відсутнє	Ні
CineDor Bot V3	Кнопкове меню	Відсутнє (Static filtering)	Частково
IAI MovieBot	Вільний текст	Високий рівень (Transformer)	Так (User Model)
Запропоноване рішення	Вільний текст	Середній (spaCy + Regex)	Так (SQLite)

Аналіз підтверджує наявність технологічної прогалини в обраній предметній області. На ринку домінують або примітивні системи з кнопковим керуванням, або важкі академічні проєкти. Запропоноване у дипломній роботі рішення дозволяє заповнити цю нішу шляхом розробки швидкого, легковагового модуля, який завдяки гібридному застосуванню регулярних виразів та локальної NLP-моделі (spaCy) забезпечить високу гнучкість обробки української мови без потреби у залученні високовартісних хмарних LLM.

1.3 Архітектурні вимоги та постановка задачі дослідження

Метою проєкту є розробка інтелектуального модуля обробки неструктурованих запитів для рекомендацій багатоформатного кіноконтенту у месенджері Telegram.

Виходячи з виявлених недоліків існуючих рішень та сформульованої мети, постановка задачі дослідження полягає у виконанні такого комплексу взаємопов'язаних етапів:

– проведення системного аналізу предметної області, методів обробки природної мови (NLP) та алгоритмів побудови сучасних рекомендаційних діалогових систем;

– проєктування багаторівневої асинхронної архітектури програмного модуля Telegram-бота, загальних інформаційних потоків та інфологічної моделі локальної реляційної бази даних;

– розроблення алгоритмічного забезпечення, зокрема гібридного NLP-алгоритму для точної екстракції темпоральних меж, розпізнавання жанрів за допомогою векторної семантики та класифікації інтенту користувача;

– програмна реалізація спроєктованої системи з використанням об'єктно-орієнтованого підходу, мови Python, фреймворку python-telegram-bot, лінгвістичної моделі spaCy та СУБД SQLite, а також її інтеграція із зовнішнім API The Movie Database;

– проведення комплексного модульного та інтеграційного тестування розробленого програмного забезпечення для підтвердження його надійності, відмовостійкості та точності розпізнавання неструктурованих запитів.

Для забезпечення миттєвого обслуговування багатьох користувачів архітектура повинна будуватися на принципах асинхронного програмування (python-telegram-bot, httpx) [10, 11]. Управління станом є критичним для персоналізації [12], оскільки Telegram-боти є Stateless-додатками. Інтеграція СУБД SQLite дозволить локально зберігати історію переглядів (усуваючи проблему рекомендації вже переглянутого контенту), списки обраних медіафайлів, а також індивідуальні профілі налаштувань.

Програмний модуль повинен виконувати наступні етапи NLP-конвеєра:

1) токенізація та нормалізація: вилучення пунктуації та лематизація (spaCy);

2) фільтрація шуму: видалення галузевих стоп-слів;

3) екстракція числових параметрів: застосування Regex для пошуку дат та точних значень користувацького рейтингу;

4) семантичне мапування: використання векторних уявлень для знаходження жанрів та сентимент-аналіз для рейтингу;

5) визначення намірів: розділення запитів на пошук за назвою або за критеріями. Також необхідна реалізація механізму м'якої деградації: у разі збою NLP-моделі бот повинен переходити в базовий режим замість аварійного завершення роботи.

2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ МОДУЛЯ

2.1 Загальна структура проєктованої системи

Метою даного підрозділу є формування концептуальних засад вирішення поставленої задачі. Згідно з методологією проєктування складних інформаційних систем [13], для реалізації програмного модуля обробки текстових запитів було використано структурно-орієнтований та об'єктно-орієнтований підходи [14, 15, 16].

Структурно-орієнтований підхід дозволив здійснити декомпозицію процесу взаємодії користувача із системою. Архітектура розробленого модуля будується на принципах асинхронної взаємодії, керованої подіями, і містить п'ять ключових логічних компонентів (модулів):

1) клієнтський рівень: забезпечує графічний та текстовий інтерфейс користувача. У межах Telegram API цей рівень відповідає за відображення результатів пошуку (картки фільмів з постерами) та обробку взаємодій із ReplyKeyboardMarkup (головне меню) та InlineKeyboardMarkup (кнопки «Додати в обране», «Деталі»);

2) асинхронний контролер: реалізований на базі бібліотеки python-telegram-bot (з використанням циклу подій asyncio). Він приймає вхідні повідомлення через метод тривалого опитування та перенаправляє їх до відповідних обробників, не блокуючи основний потік виконання сервера;

3) модуль розуміння природної мови: головне інтелектуальне логічне ядро системи. Інкапсульоване у класі QueryProcessor, воно відповідає за токенізацію, лематизацію українського тексту, застосування словників сентименту, вилучення іменованих сутностей та класифікацію інтенту користувача;

4) рівень взаємодії із зовнішніми даними: забезпечує асинхронне виконання HTTP-запитів до бази даних The Movie Database (за допомогою сучасної бібліотеки httpx) [17]. Функції-обгортки (наприклад, get_items_by_genre, search_item) формують структуровані запити з врахуванням параметрів пагінації, мови та встановлених фільтрів;

5) рівень персистентності даних: модуль для взаємодії з локальною реляційною базою даних SQLite, що відповідає за збереження історії взаємодій, вподобань та глобальних налаштувань профілів користувачів.

Взаємодію та інформаційні потоки між описаними компонентами наочно зображено на рисунку 2.1.

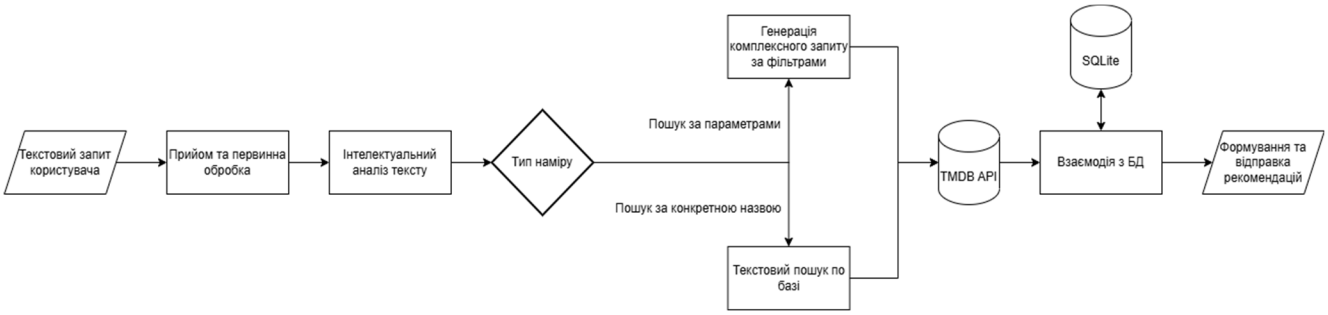


Рисунок 2.1 – Загальна структурна схема інформаційних потоків програмного модуля

Ієрархічний характер системи вибудовується з використанням ієрархії класів (об'єктно-орієнтований підхід). Клас контролера бота ініціалізує екземпляри класу процесора бази даних та NLP-обробника. При цьому застосовано патерн проєктування Singleton для NLP-модуля. Масивна векторна модель spaCy (uk_core_news_lg), яка займає понад 500 МБ оперативної пам'яті, завантажується лише один раз під час ініціалізації сервера, після чого цей єдиний екземпляр обслуговує всі паралельні асинхронні запити, що критично знижує затримку системи. Крім того, на рівні контролера реалізовано механізм м'якої деградації: якщо модуль QueryProcessor не може бути ініціалізований, система перехоплює виключення і бот продовжує роботу у базовому режимі точних команд без аварійного завершення.

2.2 Алгоритмічне забезпечення модуля

Алгоритмічне забезпечення описує детальну логіку обробки вхідних неструктурованих повідомлень та спосіб формування результатів. Головним алгоритмом розв'язання задачі є гібридний NLP-конвеєр, реалізований у класі

QueryProcessor, який поєднує детерміновані правила із нейромережевими векторними моделями [18, 5].

Процес трансформації текстового повідомлення *T* у кортеж параметрів бази даних проходить через п'ять послідовних кроків.

Крок 1. Екстракція детермінованих числових параметрів.

Перш ніж текст буде розбито на токени, до нього застосовуються скомпільовані регулярні вирази для виявлення темпоральних меж. Алгоритм послідовно перевіряє чотири патерни:

1) діапазон років: регулярний вираз шукає конструкції типу з [рік] по [рік]. У разі збігу встановлюються параметри min_date та max_date;

2) нижня межа: патерни з тригерами «після», «від», «новіші за» встановлюють параметр min_date (наприклад, «після 2015» → primary_release_date.gte=2015-01-01);

3) верхня межа: патерни з тригерами «до», «старіші за» встановлюють параметр max_dat;

4) точний рік: якщо діапазонів не знайдено, система виконує пошук 4-значних чисел формату 19XX або 20XX, інтерпретуючи їх як конкретний рік випуску.

Знайдені числові значення зберігаються в об'єкт стану поточного запиту, а відповідні текстові фрагменти (підрядки) вилучаються із загального тексту повідомлення *T*, щоб уникнути їх хибної інтерпретації на наступних кроках.

Окрім виявлення дат, алгоритм застосовує окремий регулярний вираз для екстракції точного користувацького рейтингу. Якщо запит містить конструкції на кшталт «рейтинг вище 7,5», система вилучає числове значення, конвертує його у тип float та автоматично встановлює параметр min_rating.

Крок 2. Токенізація, нормалізація та лематизація.

Залишковий текст передається до конвеєра spaCy. Текст розбивається на базові лексичні одиниці (токени). Алгоритм відкидає токени, які є знаками пунктуації або мають довжину менше 3 символів. Кожен токен приводиться до

нижнього регістру та проходить процес лематизації – зведення до словникової форми (наприклад, слово «смішне» зводиться до «смішний»).

Далі застосовується фільтр галузевих стоп-слів. Алгоритм містить словник специфічних для кінематографу слів (наприклад, «кіно», «фільм», «подивитися», «порадь», «серіал», «хочу»). Якщо лема токена збігається зі словом із цього словника, вона відкидається як шумовий текст, що не несе критеріального навантаження.

Крок 3. Сентимент-аналіз та оцінка якості.

На цьому етапі алгоритм перевіряє наявність лем токенів у словнику якісних характеристик. Суб'єктивні оціночні судження конвертуються в об'єктивні числові фільтри:

- слова «шедевр», «легенда» встановлюють параметр мінімального рейтингу min_rating = 8,5;
- слова «топ», «найкращий», «бомбезний» встановлюють min_rating = 8,0;
- слова «класний», «чудовий» встановлюють min_rating = 7,5;
- слова «хороший», «добрий» встановлюють min_rating = 7,0;
- негативні слова або тригери (наприклад, «треш», «крінж») ігноруються алгоритмом рекомендації або встановлюють min_rating = 0,0.

Крок 4. Семантичне мапування жанрів.

Це ключовий етап інтелектуальної обробки, який поєднує пряме мапування та аналіз векторної близькості. Спочатку застосовується механізм прямого семантичного розширення: система перевіряє, чи не є токен дієсловом або описовим прикметником, який прямо вказує на жанр. Наприклад, лема «посміятися» жорстко мапиться на жанр «комедія», а лема «магія» – на «фентезі».

Для решти токенів обчислюється метрика косинусної близькості між вектором токена та попередньо розрахованими векторами назв еталонних жанрів з бази TMDB [8]:

$$S_C(t_i, G_j) = \frac{t_i * G_j}{||t_i|| * ||G_j||}, \quad (2.1)$$

де $S_C(t_i, G_j)$ – метрика косинусної близькості,

t_i – векторне представлення і-го токена із запиту користувача,

G_j – векторне представлення j-го еталонного жанру з бази даних TMDB.

Для визначення жанрової приналежності використовується метод семантичних якорів. Замість порівняння лише з однією назвою жанру, кожен еталонний жанр описується кластером базових понять (наприклад, для комедії це векторний простір слів: «комедія», «гумор», «сміх»).

Алгоритм обчислює метрику косинусної близькості між вектором введеного токена та векторами кожного слова-якоря. Якщо максимальне значення схожості перевищує жорсткий поріг відсіювання шуму 0,75, система фіксує семантичний зв'язок.

Для адаптації моделі до української морфології (де закінчення можуть значно впливати на вектор) введено гарантовану евристику кореня: якщо лема введеного слова має спільний корінь із семантичним якорем (наприклад, «жахастик» і «жах» або «гумористичний» і «гумор»), алгоритм ігнорує косинусну відстань і примусово встановлює максимальний рівень схожості $S_C = 1,0$. Виявлені жанри додаються до списку genre_ids.

Крок 5. Визначення наміру.

На фінальному етапі алгоритм формує так званий clean_query – рядок, що містить залишок токенів після вилучення дат, жанрів, оцінок та стоп-слів (наприклад, у запиті «хороший бойовик про термінатора», слово «термінатора» залишиться у clean_query).

Послідовність виконання описаних кроків та узагальнену логіку роботи цього алгоритму обробки природної мови наочно представлено у вигляді блок-схеми на рисунку 2.2

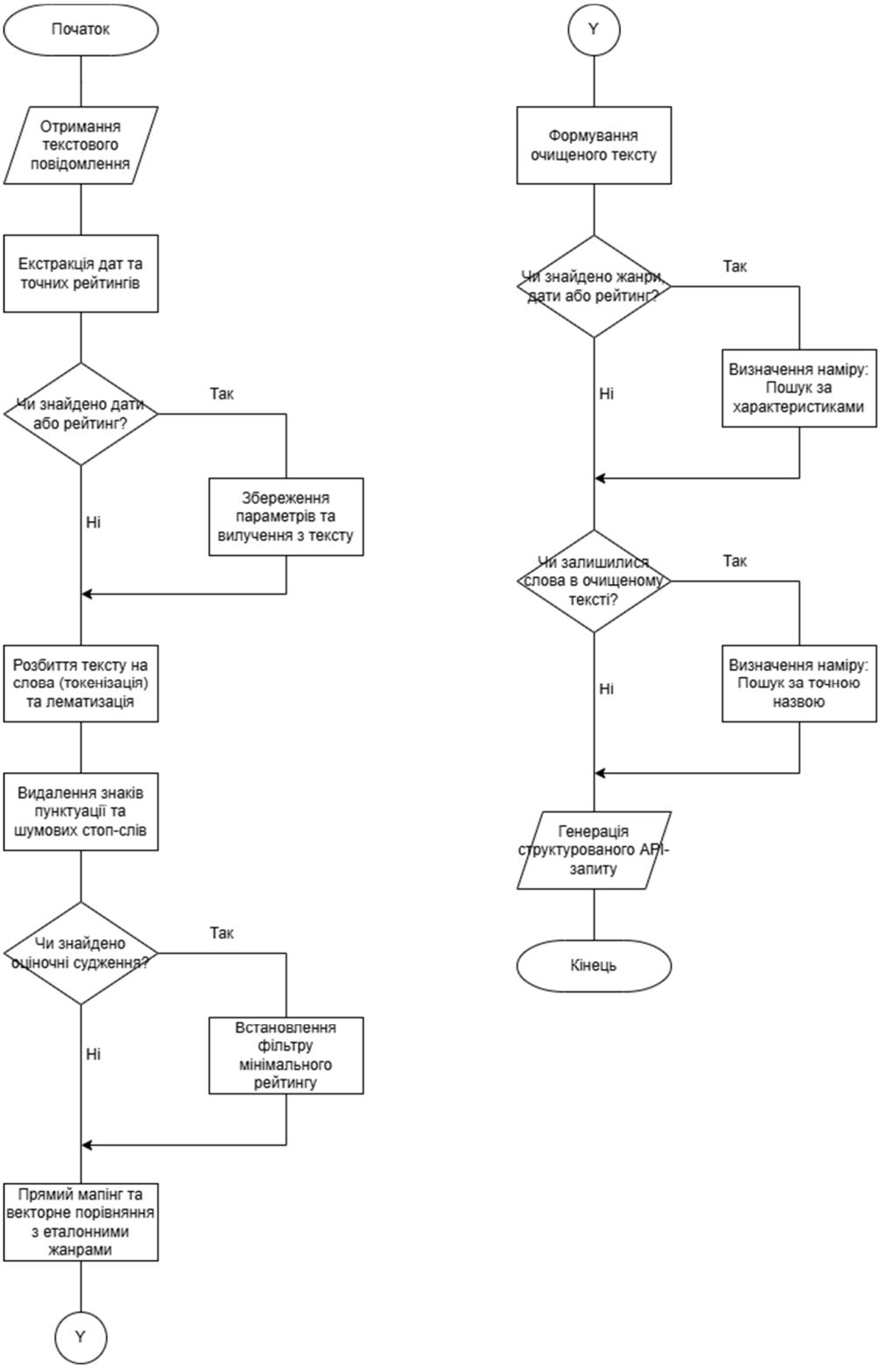


Рисунок 2.2 – Схема алгоритму обробки природної мови

Під час формування clean_query алгоритм додатково вилучає цифрові значення рейтингів за допомогою регулярного виразу, щоб запобігти їх потраплянню у текстовий пошук. Далі система визначає намір *I* за наступною логікою:

– якщо списки genre_ids, параметри date_filter не є порожніми, або у тексті знайдено вимогу до рейтингу (min_rating), системі призначається намір I = search_criteria (пошук за фільтрами);

– якщо всі критерії відсутні, але залишковий clean_query не є порожнім, системі призначається намір I = search_title (прямий пошук за точною назвою у базі);

– якщо виявлено спеціальні тригери рандомізації (наприклад, «здивуй», «будь-що»), алгоритм активує функцію random для вибору випадкових параметрів із довідника.

Крім того, на цьому етапі реалізовано алгоритм розпізнавання логічних операторів для комбінування жанрів. Система аналізує наявність розділових сполучників («або», «чи», «хоча б») у тексті запиту за допомогою регулярних виразів:

– якщо такі маркери виявлені, алгоритм встановлює логіку об'єднання OR (в API TMDB передається розділювач |), що дозволяє шукати фільми, які належать до будь-якого із зазначених жанрів.

– за замовчуванням застосовується логіка AND (розділювач ,), що забезпечує жорсткий пошук контенту на перетині жанрів (наприклад, одночасно і бойовик, і комедія)

2.3 Інформаційне забезпечення модуля

Інформаційне забезпечення охоплює організацію даних, їх структурне представлення у системі та методи управління ними в межах життєвого циклу програмного модуля.

Вхідна інформація системи складається з двох принципових потоків:

1) неструктуровані текстові повідомлення та Callback-запити. Отримуються від клієнта через Telegram API. Вони містять метадані повідомлення (ідентифікатор чату chat_id, ідентифікатор користувача user_id, текстовий рядок, мовні налаштування пристрою);

2) структуровані дані від TMDB API [19]. Надходять у форматі JSON у відповідь на HTTP GET-запити. Ці дані мають складну ієрархічну структуру: масиви об'єктів results, які містять id (унікальний код), title (назва), overview (опис сюжету), poster_path (посилання на зображення), genre_ids (масив жанрів) та vote_average (рейтинг).

Окремої уваги заслуговує структура відповіді від TMDB API. Для забезпечення максимальної швидкодії бот не завантажує весь масив доступних метаданих (акторський склад, бюджет, відгуки), а виконує вибіркову десеріалізацію лише критично необхідних полів під час відображення результатів пошуку:

- id: унікальний числовий ідентифікатор контенту, який використовується системою для генерації кнопок CallbackQuery (наприклад, функцій «Детальніше» та «Додати в обране»);
- title (або name для телесеріалів): локалізована назва медіапродукту;
- genre_ids: масив числових ідентифікаторів, який динамічно зіставляється з локальним словником жанрів (id_to_name) для формування текстового опису;
- poster_path: відносний шлях до графічного файла обкладинки, який на етапі форматування повідомлення конкатенується з базовою URL-адресою графічного сервера TMDB (IMAGE_BASE_URL);
- vote_average: показник рейтингу у форматі числа з плаваючою комою, який використовується для застосування користувацьких фільтрів якості;
- release_date (або first_air_date): дата релізу у форматі YYYY-MM-DD, з якої система екстрагує лише рік для компактного відображення у короткій картці результатів.

Такий архітектурний підхід до вибіркового парсингу JSON гарантує мінімальне споживання оперативної пам'яті під час генерації видачі та швидке формування InlineKeyboardMarkup.

Вихідною інформацією є відформатовані повідомлення, які бот надсилає користувачу. Вона формується шляхом злиття даних із локальними файлами

локалізації. До вихідної інформації також входять структуровані об'єкти InlineKeyboardMarkup, які генерують кнопки під повідомленнями.

Концептуальна інфологічна модель визначає основні сутності предметної області, абстрагуючись від конкретної СУБД. Модель предметної області системи базується на трьох основних сутностях:

– сутність «Користувач» (User): відображає персону в межах екосистеми Telegram. Характеризується унікальним ідентифікатором та індивідуальними налаштуваннями взаємодії з ботом;

– сутність «Історія» (History): транзакційна сутність, яка фіксує факт видачі рекомендації або перегляду деталей контенту конкретним користувачем. Використовується для уникнення дублювання результатів;

– сутність «Обране» (Favorites): асоціативна сутність, що зв'язує користувача та медіа-об'єкт, який було свідомо збережено для майбутнього перегляду.

Глобальна даталогічна модель описує структуру даних у термінах реляційної алгебри. Для забезпечення роботи модуля була розроблена схема, що складається з трьох нормалізованих таблиць: таблиці налаштувань (таблиця 2.1), таблиці історії переглядів (таблиця 2.2) та таблиці вподобань (таблиця 2.3).

Для забезпечення цілісності даних на рівні схеми бази даних для таблиць history та favorites встановлено композитне обмеження унікальності на комбінацію полів (user_id, item_id, item_type). Це гарантує неможливість повторного запису одного і того ж фільму в обране або історію.

Для реалізації даталогічної моделі на рівні фізичного носія обрано систему управління реляційними базами даних SQLite 3 [20].

Вибір SQLite як ядра персистентності обумовлений архітектурними вимогами легковагового Telegram-бота. На відміну від громіздких клієнт-серверних СУБД (PostgreSQL або MySQL), SQLite зберігає всю структуру таблиць, індекси та дані у єдиному локальному файлі. Це усуває накладні витрати на мережеву взаємодію між додатком та сервером бази даних, забезпечуючи мінімальну затримку при читанні профілів.

Таблиця 2.1 – Структура таблиці налаштувань (user_settings)

Назва поля	Тип даних	Обмеження	Опис призначення
user_id	INTEGER	PRIMARY KEY	Унікальний ідентифікатор користувача у Telegram
language	TEXT	DEFAULT 'uk'	Обрана мова інтерфейсу бота
min_rating	REAL	DEFAULT 0.0	Персональний поріг мінімального рейтингу

Таблиця 2.2 – Структура таблиці історії переглядів (history)

Назва поля	Тип даних	Обмеження	Опис призначення
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Суррогатний ключ запису
user_id	INTEGER	FOREIGN KEY, NOT NULL	Ідентифікатор користувача
item_id	INTEGER	NOT NULL	Ідентифікатор фільму/серіалу в базі TMDB
item_type	TEXT	NOT NULL	Тип контенту ('movie' або 'tv')
timestamp	DATETIME	DEFAULT CURRENT_TIMESTAMP	Часова мітка додавання запису
title	TEXT	Відсутні	Збережена назва контенту для швидкого виведення

Таблиця 2.3 – Структура таблиці вподобань (favorites)

Назва поля	Тип даних	Обмеження	Опис призначення
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Сурогатний ключ запису
user_id	INTEGER	FOREIGN KEY, NOT NULL	Ідентифікатор користувача
item_id	INTEGER	NOT NULL	Ідентифікатор фільму/серіалу в базі TMDB
item_type	TEXT	NOT NULL	Тип контенту ('movie' або 'tv')
title	TEXT	NOT NULL	Збережена назва (для швидкого виведення списку)

Оскільки Telegram-бот функціонує в асинхронному середовищі, використання стандартного синхронного модуля sqlite3 призвело б до блокування основного потоку виконання під час кожної транзакції до бази даних. Для вирішення цієї архітектурної проблеми у програмному модулі реалізовано асинхронний рівень доступу до даних за допомогою бібліотеки aiosqlite [21]. Це дозволяє виконувати SQL-запити у неблокуючому режимі, значно підвищуючи пропускну здатність системи при одночасному обслуговуванні багатьох користувачів.

Для оптимізації зберігання часових міток в історії переглядів використовується поле timestamp, яке автоматично фіксує точний час додавання запису на рівні рушія СУБД завдяки обмеженню DEFAULT CURRENT_TIMESTAMP. Усі транзакції обгорнуті в асинхронні контекстні менеджери, що гарантує безпечне закриття з'єднань та вивільнення курсорів навіть у випадку виникнення мережевих виключень чи помилок обробки. Запис у таблиці history та favorites супроводжується механізмом INSERT OR REPLACE та INSERT OR IGNORE, що делегує обробку дублікатів безпосередньо системі управління базами даних, зменшуючи навантаження на логічне ядро застосунку.

3 ПРОГРАМНО-ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ МОДУЛЯ

3.1 Реалізація програмного забезпечення

Для програмної реалізації інформаційної системи розроблено архітектуру, що спирається на модульний принцип та концепції об'єктно-орієнтованого програмування (ООП). Архітектурний шаблон системи тяжіє до моделі «Модель-Представлення-Контролер», адаптованої під специфіку месенджера [22].

Зв'язки між програмними компонентами, їхнє призначення та логічну взаємодію проілюстровано на загальній структурній схемі програмної системи показаній на рисунку 3.1.

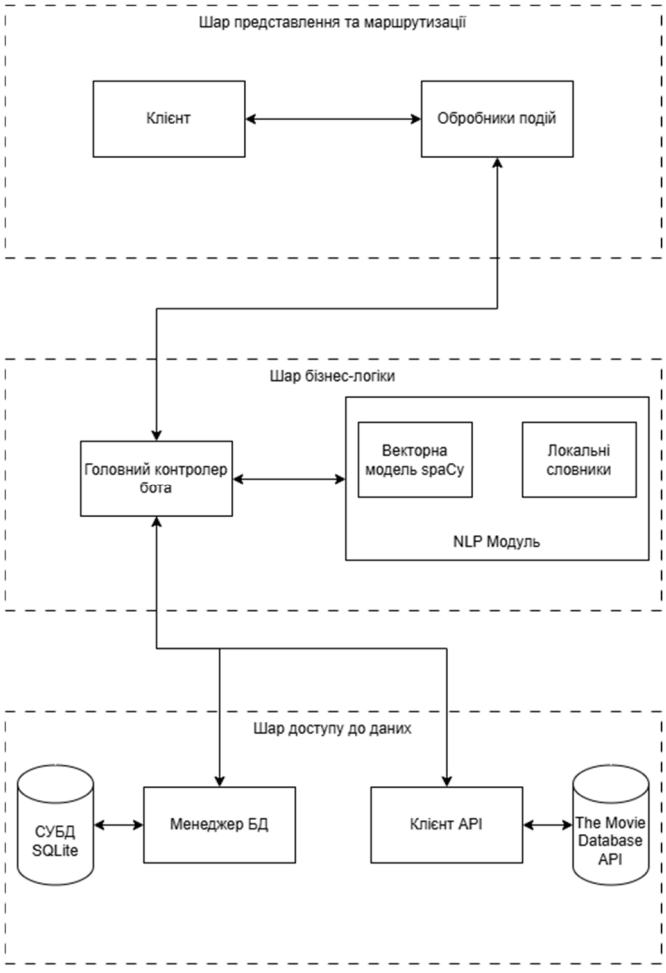


Рисунок 3.1 – Структурна схема програмної системи

Згідно зі структурною схемою, програмна система логічно поділена на три ізольовані шари:

1) шар представлення та маршрутизації: реалізований через Telegram Bot API. Забезпечує прийом вхідних оновлень, їх маршрутизацію та форматування вихідних повідомлень.

2) шар бізнес-логіки: інкапсулює алгоритми розуміння природної мови. Відповідає за лематизацію, семантичний мапінг та дизамбігуацію намірів користувача.

3) шар доступу до даних: відповідає за постійне зберігання інформації у локальній СУБД SQLite та комунікацію з хмарним API The Movie Database.

Структуру розробленого програмного забезпечення на рівні об'єктів детально відображено на рисунку 3.2 – UML-діаграмі класів.

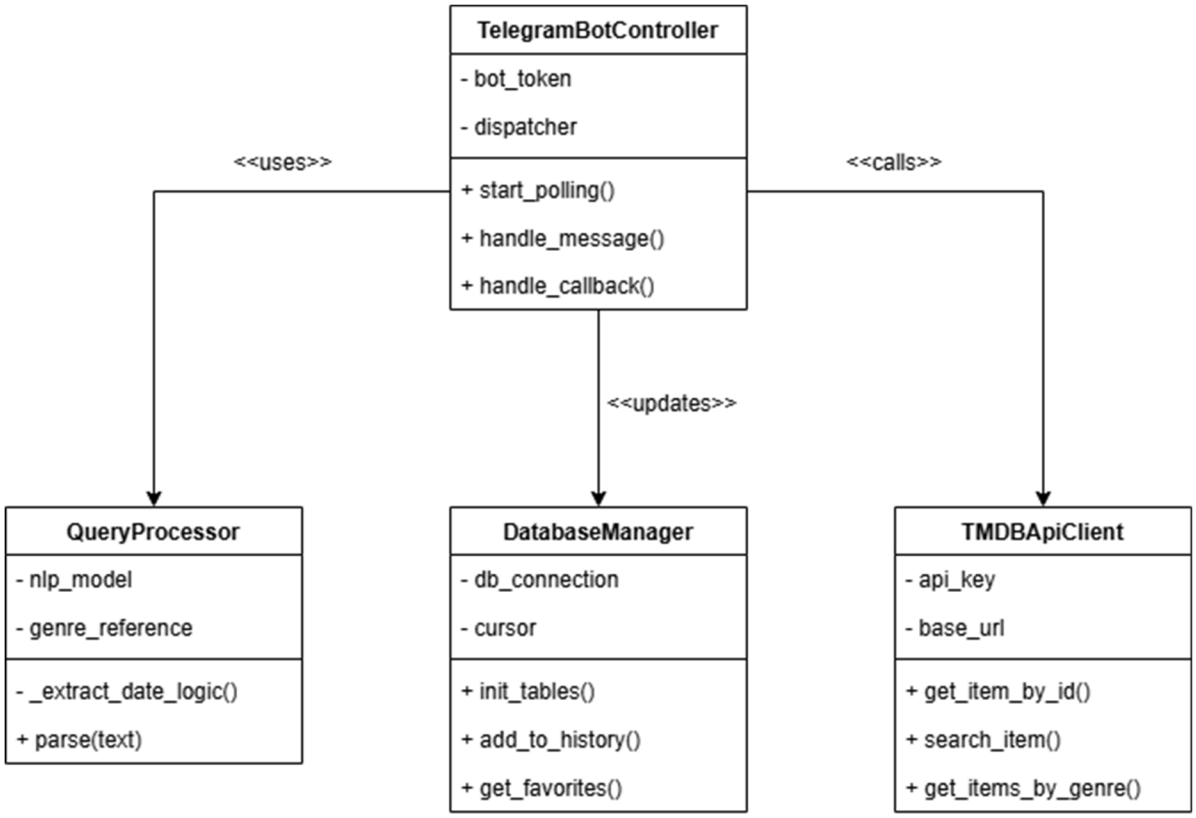


Рисунок 3.2 – UML-діаграма класів програмної системи

На діаграмі класів представлено такі ключові класи та їхні методи:

а) клас TelegramBotController:

1) атрибути: bot_token, dispatcher;

2) методи: start_polling() – ініціалізація асинхронного циклу подій; handle_message() – перехоплення текстових повідомлень; handle_callback() – обробка натискань на Inline-кнопки.

б) клас QueryProcessor (Реалізований за патерном Singleton):

1) атрибути: nlp_model (завантажена модель spaCy uk_core_news_lg), genre_reference (словник еталонних жанрів);

2) методи: _extract_date_logic() – застосування регулярних виразів; parse(text) – основний метод конвеєра, що здійснює токенізацію та обчислює косинусну близькість;

в) клас TMDBApiClient (Реалізує патерн Façade):

1) атрибути: api_key, base_url;

2) методи: get_item_by_id(id) – отримання детальної інформації про фільм; search_item(query) – пошук за точною назвою; get_items_by_genre(genre_ids, dates) – генерація складного параметризованого GET-запиту. Всі методи є асинхронними (async def) [23, 24].

г) Клас DatabaseManager:

1) атрибути: db_connection, cursor;

2) методи: init_tables() – створення DDL-структур при першому запуску; add_to_history(user_id, item_id) – виконання SQL-запиту INSERT з обробкою виключень на дублікати (IntegrityError); get_favorites(user_id) – повернення збережених списків.

Згідно з наведеною діаграмою, взаємодія між екземплярами класів будується за принципом централізованого управління (патерн Controller), де TelegramBotController виступає головним керуючим компонентом інформаційних потоків.

Життєвий цикл обробки запиту виглядає наступним чином:

1) контролер через метод handle_message() приймає неструктурований текст від користувача та делегує його обробку до єдиного екземпляра QueryProcessor;

2) NLP-модуль повертає контролеру структурований словник даних (JSON-подібний об'єкт), що містить розпізнаний інтент, логіку об'єднання, ідентифікатори жанрів та часові межі;

3) отримавши ці параметри, TelegramBotController викликає відповідні методи TMDBApiClient (наприклад, get_items_by_genre()), який асинхронно звертається до зовнішнього REST API та повертає масив знайденого медіаконтенту;

4) на фінальному етапі, або під час обробки натискань на Inline-кнопки (через метод handle_callback()), контролер ініціює взаємодію з DatabaseManager. Це дозволяє виконати транзакції збереження історії переглядів (add_to_history()) або оновити список обраного, забезпечуючи персистентність користувацької сесії.

Для забезпечення необхідного рівня швидкодії та надійності системи обґрунтовано вибір таких програмних засобів та технологій :

- мова програмування Python (версія 3.10+) [25, 26]: обрана як індустріальний стандарт для задач обробки даних [27]. Наявність вбудованої бібліотеки asyncio дозволяє ефективно обходити обмеження глобального блокування інтерпретатора під час виконання великої кількості мережевих запитів;
- бібліотека python-telegram-bot [28]: сучасний асинхронний фреймворк для розробки Telegram-ботів. На відміну від синхронних бібліотек, він забезпечує неблокуючу обробку сотень одночасних підключень, що є критично важливим для чат-ботів з великою аудиторією;
- бібліотека spaCy: обрана замість застарілих аналогів завдяки промисловій оптимізації. Ядро spaCy написано на мові Cython, що забезпечує виконання ресурсомістких лінгвістичних операцій зі швидкістю компільованих мов (C/C++);
- СУБД SQLite: ідеально підходить для архітектури легковагових мікросервісів, оскільки повністю відповідає вимогам ACID (Атомарність, Узгодженість, Ізольованість, Довговічність) і не потребує виділеного сервера.

Фрагмент вихідного коду, що демонструє реалізацію методу розпізнавання інтенту, наведено на рисунку 3.3.

```python
# --- РОЗУМНЕ ВИЗНАЧЕННЯ НАМІРУ ---
if not result['genre_ids'] and not result['date_filter']:
    if len(clean_query) > 0 or not result['min_rating']:
        result['intent'] = 'search_title'
    else:
        result['intent'] = 'search_criteria'
else:
    result['intent'] = 'search_criteria'
```

Рисунок 3.3 – Фрагмент вихідного коду: реалізація методу розпізнавання інтенту

Лістинг програми, що містить повний вихідний код основних класів бізнес-логіки, наведено у Додатку А.

Динаміку переходу системи між різними етапами роботи під час взаємодії з користувачем відображено на UML-діаграмі станів (рисунок 3.4). Вона демонструє життєвий цикл сесії взаємодії: від стартового стану очікування вводу (Awaiting_Input) через стан обробки NLP (Processing_Query_State), перехід до стану асинхронного очікування відповіді від зовнішньої БД (Fetching_API_State) і, нарешті, до стану відображення динамічної клавіатури з результатами (Displaying_Results) та обробки Callback-запитів (Viewing_Details_State).

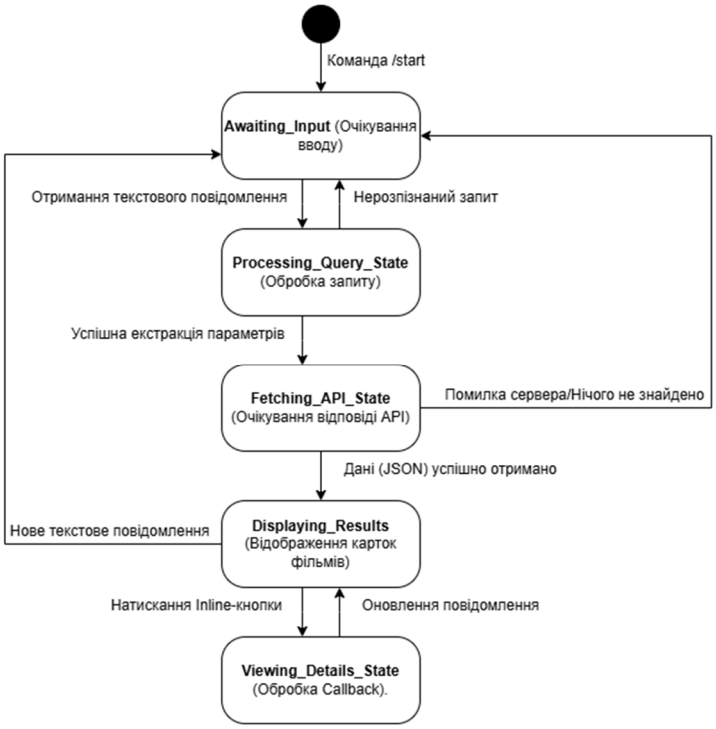


Рисунок 3.4 – UML-діаграма станів графічного інтерфейсу користувача

3.2 Інтерфейс користувача

Інтерфейс користувача Telegram-бота спроєктований з урахуванням сучасних принципів ергономіки діалогових систем. Головне завдання розробленого інтерфейсу – допомогти користувачеві вирішити завдання пошуку релевантного контенту з мінімальними когнітивними витратами.

Взаємодія з програмою реалізується через оптимізовану комбінацію двох типів графічних елементів керування :

1) текстовий ввід та Reply-клавіатури (Головне меню): використовується для загальної навігації. Після введення команди /start користувачеві доступні постійні кнопки швидкого доступу внизу екрана: «Пошук за критеріями», «Моє Обране», «Історія переглядів», «Налаштування»;

2) inline-клавіатури (Контекстне меню): прикріплюються безпосередньо до повідомлення з результатом (карткою фільму, яка містить постер, назву, рік та короткий опис). Ці кнопки генерують події і містять такі опції: «Додати в обране», «Детальніше».

Приклад інтерфейсу користувача при видачі рекомендації кіноконтенту наведено на рисунку 3.5.

Рисунок 3.5 – Інтерфейс користувача при видачі рекомендації кіноконтенту

Важливим принципом розробки якісного інтерфейсу є забезпечення комфорту користувача при виникненні непередбачуваних та виключних ситуацій (Error Handling & User Comfort). У системі реалізовано багаторівневу обробку некоректних дій:

- фільтрація нетекстового вводу: якщо користувач надсилає зображення, відео, стікер або голосове повідомлення, система не ігнорує його і не видає системну помилку. Натомість спрацьовує спеціальний обробник (Handler), який виводить інформаційне повідомлення: «Будь ласка, скористайтесь меню або напишіть запит чіткіше. »;
- механізм відкату: у випадку, коли NLP-модуль після лематизації не може розпізнати жодного жанру, дати чи критерію з тексту, система виводить інформаційне повідомлення;
- відсутність результатів: якщо за сформованими вузькими критеріями в базі TMDB не знайдено жодного фільму, система коректно повідомляє що за запитом нічого не знайдено і пропонує спробувати інші слова;
- відмовостійкість API: якщо сервери TMDB тимчасово недоступні (відповідь 503 Service Unavailable або Timeout), бот перехоплює мережеве виключення та інформує користувача: «На жаль, база даних фільмів зараз недоступна. Будь ласка, спробуйте пізніше», не дозволяючи програмі аварійно завершити роботу.

Детальний приклад взаємодії користувача з системою наведено у Додатку В. На рисунках проілюстровано повний сценарій: від введення неструктурованого тексту до отримання релевантної рекомендації кіноконтенту.

3.3 Тестування розробленої програми

Для підтвердження надійності, коректності роботи алгоритмів та відповідності програмного продукту поставленим вимогам було проведено комплексне тестування розробленої системи. Процедура тестування охоплювала три рівні: модульне, інтеграційне та навантажувальне тестування.

Основну увагу на етапі тестування було приділено перевірці логіки класу QueryProcessor, який відповідає за розпізнавання інтенту. Було сформовано репрезентативну вибірку тестових запитів (Test Cases) українською мовою з різним рівнем семантичної та синтаксичної зашумленості.

Результати модульного тестування гібридного конвеєра наведено у таблиці 3.1.

Таблиця 3.1 – Результати модульного тестування NLP-конвеєра

Вхідний текстовий запит	Очікуваний результат (Інтент + Параметри)	Фактичний результат обробки	Статус тесту
«порадь стару комедію зняту до 2005 року»	search_criteria ; max_date : 2005; genres : [35]	search_criteria ; max_date : 2005; genres : [35]	PASS
«знайди фільм володар перснів»	search_title; query: "володар перснів"	search_title; query: "володар перснів"	PASS
«хочу подивитись якийсь шедевр про космос»	search_criteria; min_rating: 8.5; genres: 878	search_criteria; min_rating: 8.5; genres: 878	PASS
«трилер з 2010 по 2020 рік»	search_criteria; min_date: 2010; max_date: 2020; genres:	search_criteria; min_date: 2010; max_date: 2020; genres:	PASS
«фдвлофі фівдо» (беззмістовний набір літер)	search_title (f□llb□□k); query: "фдвлофі фівдо"	search_title; query: "фдвлофі фівдо"	PASS

Як свідчать дані з таблиці 3.1, завдяки комбінації регулярних виразів та вектора uk_core_news_lg, модуль успішно відсіює стоп-слова («порадь», «знайди», «фільм») і точно екстрагує числові параметри та ідентифікатори жанрів. Усі розроблені тестові сценарії завершилися зі статусом PASS, що підтверджує

здатність алгоритму коректно обробляти як стандартні, так і зашумлені текстові запити.

На цьому етапі перевірялася коректність взаємодії між логічним ядром бота, базою даних SQLite та зовнішнім REST API.

– тестування БД: було зімітовано спробу користувача додати один і той самий фільм до списку «Обране» двічі поспіль. Тест підтвердив, що завдяки використанню SQL-конструкції INSERT OR IGNORE, система коректно ігнорує дублікати на рівні бази даних без виклику критичних виключень, а інтерфейс бота динамічно оновлює Inline-клавіатуру (змінюючи кнопку «Додати в обране» на «Видалити з обраного»);

– тестування API: за допомогою бібліотеки pytest-asyncio та створення мок-об'єктів було зімітовано затримку відповіді від TMDB API понад 10 секунд. Тест підтвердив, що асинхронний HTTP-клієнт httpx коректно розриває з'єднання за тайм-аутом і виводить користувачу повідомлення про помилку, при цьому інші користувачі продовжують отримувати обслуговування без затримок.

Оскільки завантаження масивної векторної моделі spaCy (розміром понад 500 МБ) у пам'ять потребує значних ресурсів, було перевірено вплив реалізації паттерну Singleton на продуктивність.

Тестування показало, що «холодний старт» системи (початкова ініціалізація моделі при запуску сервера) триває близько 5–8 секунд. Однак, завдяки збереженню моделі в оперативній пам'яті, обробка кожного наступного текстового запиту займає в середньому від 40 до 70 мілісекунд. Це повністю задовольняє вимоги до систем реального часу у середовищі месенджерів. Журнал логування результатів модульного тестування зображено на рисунку 3.6.

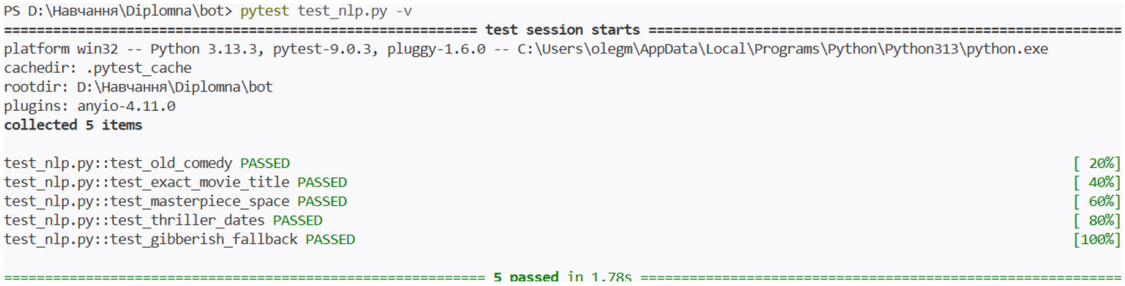


Рисунок 3.6 – Журнал логування результатів модульного тестування (Unit Tests)

Результати комплексного тестування підтверджують, що розроблений програмний модуль функціонує стабільно, гібридні алгоритми обробки природної мови коректно інтерпретують інтенції користувача, а асинхронна архітектура та правильне проєктування інтерфейсу забезпечують високу швидкість та ергономіку взаємодії.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було успішно вирішено актуальне науково-прикладне завдання – спроєктовано та розроблено програмний модуль обробки неструктурованих текстових запитів для рекомендацій багатоформатного кіноконтенту (фільмів та телесеріалів) у месенджері Telegram.

Виконаний теоретичний та практичний комплекс досліджень дозволяє зробити такі висновки:

1. Досліджено предметну область та обґрунтовано вибір технологій. Аналіз проблеми інформаційного перевантаження підтвердив актуальність переходу до діалогових інтерфейсів. Доведено, що для обробки «живої» флективної української мови в умовах вимог до високої швидкодії оптимальним є впровадження гібридного NLP-конвеєра. Такий підхід дозволяє уникнути проблеми «холодного старту» класичної колаборативної фільтрації та виключає затримки і фактологічні помилки, притаманні хмарним великим мовним моделям.

2. Спроєктовано та реалізовано відмовостійку асинхронну архітектуру. Програмний модуль побудовано на принципах подієво-орієнтованої архітектури з використанням фреймворку python-telegram-bot та сучасного HTTP-клієнта httpx. Для управління stateless-сесіями Telegram інтегровано локальну СУБД SQLite. Це дозволило реалізувати механізми збереження історії переглядів (для автоматичної фільтрації дублікатів при видачі результатів), списків обраного та індивідуальних параметрів локалізації користувачів.

3. Розроблено інтелектуальний алгоритм розуміння природної мови. Створено гібридний NLP-конвеєр на базі нейромережевої моделі spaCy (uk_core_news_lg) та гнучкої системи регулярних виразів. Алгоритм успішно виконує токенізацію, глибоку лематизацію, відсіювання галузевих стоп-слів, а також екстракцію числових параметрів (часових меж та точного рейтингу). Застосування векторного порівняння та мапінгу сентименту дозволяє системі з високою точністю класифікувати наміри користувача (пошук за критеріями або точною назвою).

4. Розроблено ергономічний інтерфейс користувача. Взаємодія з модулем оптимізована через поєднання текстового вводу, Reply-меню та динамічних Inline-клавіатур. Реалізовано механізм м'якої деградації та багаторівневу обробку виключень: система коректно реагує на нетекстовий ввід, незрозумілі запити, відсутність контенту в базі The Movie Database (TMDB) або мережеві тайм-аути, не допускаючи аварійного завершення роботи.

5. Проведено комплексне тестування системи. Результати модульного тестування підтвердили коректність математичної логіки визначення намірів: усі заплановані тестові сценарії, включаючи запити зі складними темпоральними межами та зашумленим текстом, були успішно пройдені. Інтеграційні тести довели цілісність бази даних при спробах запису дублікатів та стабільність обробки асинхронних мережевих з'єднань під час імітації затримок API.

Таким чином, розроблений програмний продукт повністю відповідає поставленим вимогам. Він забезпечує швидкий, інтуїтивно зрозумілий та глибоко персоналізований пошук медіаконтенту, демонструючи високу ефективність застосування методів обробки природної мови у сучасних комунікаційних платформа

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Telegram Bot API. Official Documentation [Електронний ресурс]. URL: https://core.telegram.org/bots/api (дата звернення: 04.02.2026).

2. Матвійчук О. О., Биковий П. Є. Програмний модуль обробки текстових запитів для рекомендацій фільмів у Telegram-боті // Intelligent Computing Systems and Project Management (ICSPM 2026) : збірник тез доповідей І Міжнародної науково-практичної конференції студентів і молодих вчених, м. Тернопіль, 21–22 травня 2026 р. Тернопіль : ЗУНУ, 2026. С. 30–33.

3. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook. 3rd ed. Cham : Springer, 2022. 1068 p.

4. Шаховська Н. Б., Пасічник В. В. Системи штучного інтелекту : навч. посіб. Львів : Видавництво Львівської політехніки, 2022. 392 с.

5. Jurafsky D., Martin J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition [Електронний ресурс]. 3rd ed. draft. 2024. URL: https://web.stanford.edu/~jurafsky/slp3/ (дата звернення: 15.02.2026).

6. Ґудфеллов І., Бенджо Й., Курвіль А. Глибоке навчання : пер. з англ. Київ : Професіонал, 2020. 848 с.

7. Goldberg Y. Neural Network Methods for Natural Language Processing. San Rafael : Morgan & Claypool Publishers, 2017. 310 p.

8. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space [Електронний ресурс] // arXiv preprint arXiv:1301.3781. 2013. URL: https://arxiv.org/abs/1301.3781 (дата звернення: 04.02.2026).

9. Honnibal M., Montani I. spaCy: Industrial-Strength Natural Language Processing in Python [Електронний ресурс]. 2023. URL: https://spacy.io/ (дата звернення: 04.02.2026).

10. Asynchronous I/O in Python: A Complete Walkthrough [Електронний ресурс] // Real Python. URL: https://realpython.com/async-io-python/ (дата звернення: 04.02.2026).

11. PEP 492 – Coroutines with async and await syntax [Електронний ресурс] / Python Software Foundation. URL: https://peps.python.org/pep-0492/ (дата звернення: 04.02.2026).

12. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol : O'Reilly Media, 2017. 616 p.

13. Комар М. П., Саченко А. О., Васильків Н. М., Гладій Г. М., Коваль В. С., Ліп'яніна-Гончаренко Х. В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль : ЗУНУ, 2024. 52 с.

14. Любченко В. В. Проектування архітектури програмного забезпечення : навч. посіб. Одеса : ОНПУ, 2021. 210 с.

15. Пелещишин А. М., Жежнич П. І., Пасічник В. В. Розроблення складних інформаційних систем : навч. посіб. Львів : Видавництво Львівської політехніки, 2021. 248 с.

16. Мельник А. О. Архітектура сучасних розподілених інформаційних систем та баз даних // Інформаційні технології та комп'ютерна інженерія. 2023. № 2. С. 45–52.

17. HTTPX: A Next-Generation HTTP Client for Python [Електронний ресурс]. URL: https://www.python-httpx.org/ (дата звернення: 15.05.2026).

18. Bird S., Klein E., Loper E. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. Sebastopol : O'Reilly Media, 2019. 504 p.

19. The Movie Database (TMDB) API Documentation [Електронний ресурс]. URL: https://developer.themoviedb.org/docs (дата звернення: 04.02.2026).

20. SQLite Official Documentation [Електронний ресурс]. URL: https://www.sqlite.org/docs.html (дата звернення: 04.02.2026).

21. aiosqlite: asyncio wrapper for sqlite3 [Електронний ресурс]. URL: https://aiosqlite.omnilib.dev/en/latest/ (дата звернення: 04.02.2026).

22. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley Professional, 2021. 560 p.

23. Percival H., Gregory B. Architecture Patterns with Python: Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Microservices. Sebastopol : O'Reilly Media, 2020. 306 p.

24. Richardson L., Ruby S. RESTful Web Services. Sebastopol : O'Reilly Media, 2018. 454 p.

25. Python 3.10 Documentation [Електронний ресурс] / Python Software Foundation. URL: https://docs.python.org/3/ (дата звернення: 04.02.2026).

26. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming. 2nd ed. Sebastopol : O'Reilly Media, 2022. 1014 p.

27. Трофименко О. Г., Прокоп Ю. В., Швайко І. Г. Python: алгоритми та програмування : навч. посіб. Одеса : ОНАЗ, 2022. 344 с.

28. python-telegram-bot Documentation [Електронний ресурс]. URL: https://docs.python-telegram-bot.org/ (дата звернення: 04.02.2026).

Додаток А

Лістинг вихідного коду програмного модуля

```
import spacy
import re
import random
from typing import Dict, Any, List

class QueryProcessor:
    def __init__(self):
        print("⏳ Завантаження NLP моделі (spaCy)...")
        try:
            self.nlp = spacy.load("uk_core_news_lg")
            print("✅ NLP модель завантажена.")
        except OSError:
            print("❌ ПОМИЛКА: Модель не знайдена. Виконайте: python -m spacy download uk_core_news_lg")
            raise

        # Базовий словник: Назва еталонного жанру -> ID в базі TMDB
        self.genre_reference = {
            "бойовик": "28", "пригоди": "12", "анімація": "16", "комедія": "35",
            "кримінал": "80", "документальний": "99", "драма": "18", "сімейний": "10751",
            "фентезі": "14", "історія": "36", "жах": "27", "музика": "10402",
            "детектив": "9648", "романтика": "10749", "фантастика": "878",
            "трилер": "53", "військовий": "10752", "вестерн": "37"
        }

    # Словник семантичних якорів для векторного порівняння
    self.genre_anchors = {
        "бойовик": ["бойовик", "екшн", "бійка", "зброя", "стрілянина"],
        "пригоди": ["пригоди", "подорож", "квест", "мандрівка"],
        "анімація": ["анімація", "мультфільм", "аніме", "мальований", "мультик"],
        "комедія": ["комедія", "гумор", "сміх", "веселий", "жарт", "кумедний"],
        "кримінал": ["кримінал", "злочин", "мафія", "бандит", "детектив"],
        "документальний": ["документальний", "біографія", "реальний", "факти", "пізнавальний"],
        "драма": ["драма", "трагедія", "емоції", "життєвий", "сумний"],
        "сімейний": ["сімейний", "діти", "родина", "добрий", "затишний"],
        "фентезі": ["фентезі", "магія", "казка", "ельфи", "чари"],
        "історія": ["історія", "історичний", "минуле", "епоха"],
        "жах": ["жах", "страх", "моторошно", "монстр", "страшний", "хоррор"],
        "музика": ["музика", "мюзикл", "спів", "пісня"],
        "детектив": ["детектив", "розслідування", "таємниця", "загадка"],
        "романтика": ["романтика", "любов", "кохання", "стосунки"],
        "фантастика": ["фантастика", "космос", "майбутнє", "науковий", "прибульці"],
        "трилер": ["трилер", "напруга", "саспенс", "небезпека", "маніяк"],
        "військовий": ["військовий", "війна", "армія", "солдат", "фронт"],
        "вестерн": ["вестерн", "ковбой", "дикий захід"]
    }

    # Кешування векторів для швидкодії
    print("⚙️ Підготовка векторних якорів...")
    self.anchor_docs = {}
    for genre, anchors in self.genre_anchors.items():
        self.anchor_docs[genre] = [self.nlp(anchor)[0] for anchor in anchors]

    self.random_triggers = {"вибір", "здивуй", "рандом", "будь-що", "якесь"}
```

```
        self.quality_keywords = {
            "шедевр": 8.5, "топ": 8.0, "бомба": 8.0, "легенда": 8.5, "хіт": 8.0,
            "шедевральний": 8.5, "шедевральне": 8.5, "геніальне": 8.5,
            "топовий": 8.0, "топове": 8.0, "круте": 8.0, "крутий": 8.0,
            "найкращий": 8.0, "найкраще": 8.0, "бомбезний": 8.0, "бомбезне": 8.0, "потужне": 8.0,
            "класний": 7.5, "класне": 7.5, "чудовий": 7.5,
            "хороший": 7.0, "хороше": 7.0, "добрий": 7.0, "варте": 7.0,
            "нормальний": 6.0, "нормальне": 6.0, "цікаве": 6.5,
            "треш": 0.0, "погане": 0.0, "кринж": 0.0
        }

        self.domain_stop_words = {
            "фільм", "кіно", "стрічка", "серіал", "відео", "щось",
            "подивитися", "бачити", "знайти", "хочу", "хотів", "би", "якийсь",
            "порадь", "треба", "можна", "був", "була", "було", "року", "рік",
            "глянути", "або", "та", "і", "на", "10", "топ-10", "список",
            "добірка", "найкращі", "про", "рейтинг", "рейтингом", "вище", "нижче", "від", "більше"
        }

        #Ручний мапінг тільки для жорсткого сленгу (який вектори ніколи не зрозуміють)
        self.manual_boosts = {
            "ржака": "комедія", "угар": "комедія", "кек": "комедія",
            "кріпове": "жах", "стрьомне": "жах", "м'ясо": "жах",
            "мордобій": "бойовик", "рубилово": "бойовик", "махач": "бойовик",
            "соплі": "романтика", "скло": "драма", "сай-фай": "фантастика"
        }

        # --- ПАТЕРНИ ДАТ ---
        self.range_pattern = re.compile(r'\b(?:з|від)?\s*(\d{4})\s*(?:по|до|-)\s*(\d{4})\b', re.IGNORECASE)
        self.after_pattern = re.compile(r'\b(?:після|від|з|новіш\w*(?:\s*за)?|не\s*старіш\w*(?:\s*за)?)\s*(\d{4})\b', re.IGNORECASE)
        self.before_pattern = re.compile(r'\b(?:до|перед|старіш\w*(?:\s*за)?|не\s*новіш\w*(?:\s*за)?)\s*(\d{4})\b', re.IGNORECASE)
        self.exact_year_pattern = re.compile(r'\b(?:19|20)\d{2}\b')

        # --- ПАТЕРН ДЛЯ ТОЧНОГО РЕЙТИНГУ ---
        self.rating_pattern = re.compile(r'рейтинг\w*\s*(?:вище|від|більше|>|не\s*менше)?\s*(\d(?:[.,]\d)?)', re.IGNORECASE)

    def _extract_date_logic(self, text: str) -> Dict[str, str]:
        result = {}
        range_match = self.range_pattern.search(text)
        if range_match:
            y1, y2 = range_match.groups()
            result['min_date'] = f"{y1}-01-01"
            result['max_date'] = f"{y2}-12-31"
            return result

        after_match = self.after_pattern.search(text)
        if after_match:
            y = int(after_match.group(1))
            result['min_date'] = f"{y}-01-01"

        before_match = self.before_pattern.search(text)
        if before_match:
            y = int(before_match.group(1))
            if "старіш" in before_match.group(0).lower():
                y -= 1
            result['max_date'] = f"{y}-12-31"

        if not result:
            exact_match = self.exact_year_pattern.search(text)
            if exact_match:
                y = exact_match.group(0)
                result['min_date'] = f"{y}-01-01"
                result['max_date'] = f"{y}-12-31"

        return result
```

```python
def parse(self, text: str) -> Dict[str, Any]:
    result = {
        'date_filter': self._extract_date_logic(text),
        'genre_ids': [],
        'genre_names': [],
        'min_rating': None,
        'intent': 'search_criteria'
    }

    # РОЗПІЗНАВАННЯ ЛОГІКИ (І / АБО)
    # Якщо в тексті є слова "або", "чи", "хоча б" - перемикання на логіку OR
    if re.search(r'\b(або|чи|хоча б)\b', text.lower()):
        result['genre_logic'] = 'OR'
        print("   🔀 LOGIC DETECTED: OR (АБО)")
    else:
        print("   🔗 LOGIC DETECTED: AND (ТА)")

    doc = self.nlp(text.lower())
    found_genres = set()

    print(f"\n--- 🧠 ANALYZING QUERY: '{text}' ---")

    # --- ПОШУК ТОЧНОГО ЧИСЛОВОГО РЕЙТИНГУ ---
    rating_match = self.rating_pattern.search(text)
    if rating_match:
        try:
            val = float(rating_match.group(1).replace(',', '.'))
            if val <= 10.0:
                result['min_rating'] = val
                print(f"   🌟 EXACT NUMERIC RATING DETECTED: >= {val}")
        except ValueError:
            pass
```

```
            # Перевірка на тригери рандому
            for token in doc:
                if token.lemma_ in self.random_triggers:
                    print("🎲 RANDOM TRIGGER DETECTED!")
                    random_keys = random.sample(list(self.genre_reference.keys()), 2)
                    for k in random_keys: found_genres.add(k)
                    break

            # Аналіз слів (NLP конвеєр)
            if not found_genres:
                for token in doc:
                    lemma = token.lemma_
                    word = token.text

                    if token.is_stop or token.is_punct or len(word) < 3: continue
                    if lemma in self.domain_stop_words: continue

                    # РЕЙТИНГ СЛОВАМИ (якщо немає числового)
                    if lemma in self.quality_keywords and not result['min_rating']:
                        rating = self.quality_keywords[lemma]
                        result['min_rating'] = rating
                        print(f"    🌟 QUALITY WORD DETECTED: '{lemma}' -> Rating >= {rating}")
                        continue

                    # ТОЧНИЙ ЗБІГ ЖАНРУ
                    if lemma in self.genre_reference or word in self.genre_reference:
                        exact_genre = lemma if lemma in self.genre_reference else word
                        print(f"    🎯 EXACT GENRE MATCH: '{exact_genre}'")
                        found_genres.add(exact_genre)
                        continue

# СЛЕНГ (Manual Boosts)
boost_key = None
if lemma in self.manual_boosts: boost_key = lemma
elif word in self.manual_boosts: boost_key = word

if boost_key:
    genre = self.manual_boosts[boost_key]
    print(f"    🚀 BOOST (SLANG): '{boost_key}' -> {genre}")
    found_genres.add(genre)
    continue

# РОЗУМНІ ВЕКТОРИ
for genre_name, anchor_tokens in self.anchor_docs.items():
    max_sim = 0.0

    for anchor_token in anchor_tokens:
        if token.has_vector and anchor_token.has_vector:
            sim = token.similarity(anchor_token)
            if sim > max_sim:
                max_sim = sim

    # ФІНАЛЬНА ЕВРИСТИКА КОРЕНЯ:
    # ГА (variable) anchor_token: Any
    for anchor_token in anchor_tokens:
        anchor_root = anchor_token.lemma_[:4] if len(anchor_token.lemma_) >= 4 else anchor_token.lemma_

        if len(anchor_root) >= 3 and lemma.startswith(anchor_root):
            max_sim = 1.0
            break
```

```
            # ДУЖЕ ЖОРСТКИЙ ПОРІГ ВІДСІЮВАННЯ ШУМУ (0.75):
            if max_sim > 0.75:
                print(f"    🔹 MATCH (Anchor): '{word}' vs '{genre_name}' = {max_sim:.3f}")
                found_genres.add(genre_name)

# Конвертуємо знайдені назви жанрів у ID для TMDB API
if found_genres:
    result['genre_names'] = list(found_genres)
    result['genre_ids'] = [self.genre_reference[g] for g in found_genres]

# --- ОЧИЩЕННЯ ЗАПИТУ (CLEAN QUERY) ---
leftover_tokens = []
for token in doc:
    lemma = token.lemma_
    word = token.text.lower()

    if token.is_stop or token.is_punct: continue
    if lemma in self.domain_stop_words or word in self.domain_stop_words: continue
    if lemma in self.quality_keywords: continue
    if lemma in self.genre_reference or word in self.genre_reference: continue
    if lemma in self.manual_boosts or word in self.manual_boosts: continue

    # Також видаляємо слова-якорі з чистого запиту
    is_anchor = any(lemma in anchors for anchors in self.genre_anchors.values())
    if is_anchor: continue

    leftover_tokens.append(token.text)

clean_query = " ".join(leftover_tokens).strip()
clean_query = re.sub(r'\b\d(?:[.,]\d)?\b', '', clean_query).strip()
result['clean_query'] = clean_query
```

```
# --- РОЗУМНЕ ВИЗНАЧЕННЯ НАМІРУ ---
if not result['genre_ids'] and not result['date_filter']:
    if len(clean_query) > 0 or not result['min_rating']:
        result['intent'] = 'search_title'
    else:
        result['intent'] = 'search_criteria'
else:
    result['intent'] = 'search_criteria'

print(f"    🧹 Clean Query: '{clean_query}'")
print(f"    🎯 Intent: {result['intent']}")
print("-----------------------------------\n")
return result
```

Додаток Б

Копія публікації

Міністерство освіти і науки України
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

Матеріали

І Міжнародної науково-практичної конференції студентів і молодих вчених

«ICSPM 2026: Intelligent Computing Systems and Project Management /

Інтелектуальні обчислювальні системи та управління проєктами»

21–22 травня 2026 р.

м. Тернопіль - 2026

УДК 004.8:005.8](063)

Присвячено 60-річчю Західноукраїнського національного університету

ICSPM 2026: Інтелектуальні обчислювальні системи та управління проєктами : збірник тез доповідей І Міжнародної науково-практичної конференції студентів і молодих вчених (Тернопіль, 21–22 травня 2026 року). – Тернопіль : ЗУНУ, 2026. – 190 с.

До збірника увійшли тези доповідей учасників І Міжнародної науково-практичної конференції студентів і молодих вчених «ICSPM 2026: Intelligent Computing Systems and Project Management / Інтелектуальні обчислювальні системи та управління проєктами», що відбулася на базі кафедри інформаційно-обчислювальних систем і управління факультету комп'ютерних інформаційних технологій Західноукраїнського національного університету.

Матеріали відображають результати досліджень за такими напрямами:

1. Інтелектуальні обчислення та програмні системи.
2. Проєктно-орієнтоване управління та процеси прийняття рішень.
3. Штучний інтелект, аналітика даних і кібернетика.
4. Прикладні технології та інформаційно-комунікаційні системи.
5. Сучасні інформаційні технології в економіці, праві та освіті.

Рекомендовано до друку на засіданні кафедри інформаційно-обчислювальних систем і управління Західноукраїнського національного університету (протокол №12 від 26.05.2026 р.).

За зміст наукових праць, точність наведених цитувань, перекладу та достовірність фактологічних матеріалів відповідальність несуть автори публікацій. У збірнику збережено авторську стилістику, пунктуацію та орфографію.

ЗМІСТ

УДК 004.89:004.5:004.77

ПРОГРАМНИЙ МОДУЛЬ ОБРОБКИ ТЕКСТОВИХ ЗАПИТІВ ДЛЯ РЕКОМЕНДАЦІЙ ФІЛЬМІВ У TELEGRAM-БОТІ

Матвійчук Олег Олександрович, студент,
matvijchykoleg@gmail.com

Науковий керівник: Биковий Павло Євгенович,
к.т.н., доцент, доцент кафедри інформаційно-обчислювальних систем і управління,
Західноукраїнський національний університет,
pb@wunu.edu.ua
ORCID: 0000-0002-5705-5702

Зростання обсягів медіаконтенту на сучасних стрімінгових платформах призводить до проблеми інформаційного перевантаження користувачів [1]. Традиційні інтерфейси веб-сайтів часто вимагають значного часу на навігацію, тому все більшої популярності набувають чат-боти в месенджерах, зокрема в Telegram [2]. Telegram забезпечує кросплатформеність, високу швидкість взаємодії та можливість асинхронної обробки запитів.

Метою даної роботи є розробка та практична реалізація програмного модуля, який забезпечує автоматизовану обробку текстових запитів користувача для надання релевантних рекомендацій фільмів, базуючись на інтеграції з зовнішніми API та використанні локальних баз даних для персоналізації.

Архітектура розробленої системи базується на модульному підході, що забезпечує гнучкість та легкість розширення функціоналу. Основними компонентами системи є:

1) Інтерфейсний рівень: реалізований за допомогою бібліотеки python-telegram-bot, що підтримує асинхронну обробку повідомлень [3, 9].
2) Рівень обробки запитів: спеціалізований модуль, що відповідає за розпізнавання намірів користувача та зіставлення текстових запитів з параметрами пошуку.
3) Рівень даних: динамічне отримання актуальної інформації про фільми та серіали через зовнішній API-інтерфейс [4].
4) Рівень збереження: локальне сховище для підтримки історії переглядів, списків «Обраного» та налаштувань користувача.

Для наочного відображення процесів маршрутизації та обробки запитів було розроблено структурну схему системи (рисунок 1). Як видно зі схеми, після первинного прийому повідомлення та аналізу текстового запиту, модуль визначає стратегію обробки: пошук за комплексними параметрами або за конкретною назвою.

Важливою архітектурною особливістю є те, що незалежно від обраної гілки пошуку, система спочатку звертається до зовнішнього сервісу для отримання

актуального списку медіаконтенту. Отримані дані обов'язково проходять через локальну базу даних для оновлення історії взаємодій користувача, і лише після цього формується фінальний список персоналізованих рекомендацій для відправки у чат.

Рисунок 1 – Структурна схема обробки текстового запиту в програмному модулі

Використання асинхронності дозволяє боту обробляти декілька запитів одночасно без затримок у чаті, що є критичним для користувацького досвіду [8]. Для захисту конфіденційних даних (API-ключів) використано систему оточення .env, що відповідає стандартам безпечної розробки програмного забезпечення.

Основне технічне завдання полягає у перетворенні неструктурованого або напівструктурованого текстового вводу користувача у формалізований запит до бази даних фільмів [5]. У розробленому модулі реалізовано три основні стратегії обробки запитів:

1) Змістове зіставлення емоційного стану (Mood-based Search).

Замість класичного пошуку за назвою, модуль дозволяє здійснювати пошук за «настроєм». Користувач вводить опис свого стану, а модуль обробки запитів здійснює зіставлення цих описів із відповідними ідентифікаторами жанрів у системі TMDB.

2) Багатокритеріальна фільтрація за метаданими

Підтримка запитів, що містять конкретні фільтри: жанри, часові межі (рік випуску) та рейтинговий поріг (наприклад, 6.0+), що дозволяє автоматично відсіювати контент низької якості.

3) Алгоритм персоналізації на основі історії та уподобань

У базі даних SQLite реалізовано облік попередніх взаємодій, що дозволяє зберігати вибір користувача та уникати повторних рекомендацій тих самих фільмів [6, 7].

Програмна реалізація бота підтримує двомовний інтерфейс (українська та англійська), що забезпечується окремим модулем локалізації. Взаємодія з Telegram Bot API [3] реалізована через систему асинхронних хендлерів, що дозволяє інтегрувати складні меню для управління результатами пошуку. Основні технологічні компоненти та їх функціональне призначення наведено в таблиці 1.

Таблиця 1 – Характеристики компонентів програмного модуля

Компонент	**Технологія**	**Функція**
Core Language	Python 3.10+	Основна логіка та обробка даних
Bot Framework	python-telegram-bot	Взаємодія з API месенджера
External Data	TMDB API	Отримання метаданих фільмів
Local Storage	SQLite3	Персоналізація та налаштування
Deployment	Cloud VDS / Docker	Забезпечення доступності 24/7

Результати тестування модуля показали, що середній час відгуку системи на текстовий запит становить менше 500 мс. Система стабільно обробляє велику кількість одночасних сесій завдяки використанню asyncio та асинхронних хендлерів Telegram Bot API.

Головною технічною особливістю розробленого рішення є оптимізований процес зіставлення за змістом суб'єктивних текстових описів настрою користувача з об'єктивними категоріями метаданих фільмів у середовищі месенджерів. На відміну від громіздких нейромережевих моделей, запропонований підхід забезпечує високу швидкість обробки на обмежених обчислювальних потужностях при збереженні прийнятної релевантності.

Практичне значення результатів дослідження полягає у створенні готового програмного модуля, який може бути використаний як персональний асистент для вибору медіаконтенту. Модульна архітектура дозволяє легко інтегрувати інші джерела даних (наприклад, IMDb або Rotten Tomatoes) без зміни основної логіки бота.

У результаті виконання роботи було розроблено програмний модуль для Telegram-бота, який забезпечує обробку текстових запитів для рекомендацій фільмів. Використання асинхронної архітектури у поєднанні з динамічним отриманням даних через TMDB API та локальним збереженням профілів користувачів дозволило створити ефективний інструмент персоналізації. Запропонований метод зіставлення настроїв із жанрами забезпечує інтуїтивно зрозумілу взаємодію з користувачем, мінімізуючи зусилля на пошук контенту. Подальші вдосконалення системи будуть спрямовані на впровадження елементів машинного навчання для глибшого аналізу текстових відгуків користувачів.

Список використаних джерел

1. State of the Art Movie Recommendation Systems: Optimized Hybrid Filtering and RAG Pipeline. *International Journal of Engineering Development and Research*. 2025. Vol. 13, Issue 4.

2. Волівач А. П., Рудий М. С., Аршад С. М. Алгоритм створення чат-бота для оптимізації організаційної роботи. *Мехатронні системи: інновації та інжиніринг*. 2023. С. 268–269.

3. Telegram Bot API Documentation. URL: https://core.telegram.org/bots/api (дата звернення: 25.04.2025).

4. TMDB API Documentation. URL: https://developer.themoviedb.org/docs (дата звернення: 25.04.2025).

5. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. Pearson, 2020. 1166 p.

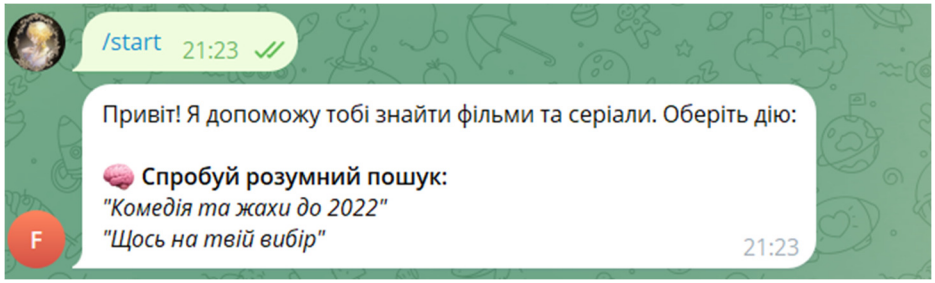
6. Хох Р. В. Підвищення якості пропозицій рекомендаційних систем. *Збірник наукових праць ПНТУ*. 2018. Вип. 1(47). С. 131–136.

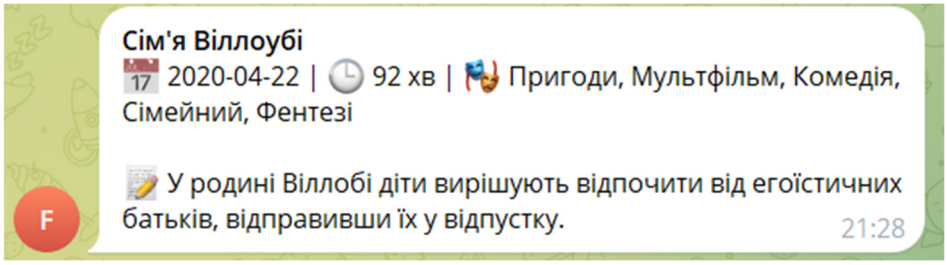
7. Ricci F., Rokach L., Shapira B. *Recommender Systems Handbook*. 3rd ed. Springer, 2022. DOI: https://doi.org/10.1007/978-1-0716-2197-4.

8. Python Software Foundation. asyncio — Asynchronous I/O. URL: https://docs.python.org/3/library/asyncio.html (дата звернення: 25.04.2025).
python-telegram-bot Documentation. URL: https://docs.python-telegram-bot.org/

Додаток В

Приклади взаємодії користувача з розробленим програмним модулем




Сім'я Віллоубі
2020-04-22 | 92 хв | Пригоди, Мультфільм, Комедія, Сімейний, Фентезі
У родині Віллобі діти вирішують відпочити від егоїстичних батьків, відправивши їх у відпустку.
21:28
F