Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА БОТА В TELEGRAM ДЛЯ АНАЛІЗУ НОВИН НА БАЗІ CHATGPT ВІД OPENAI
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-1
Галета В. Ю.
(прізвище, ініціали)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Інформатика
(повна назва освітньої програми)
Керівник доц. Тітова О. В.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Галеті Владиславу Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка бота в Telegram для аналізу новин на базі ChatGpt від OpenAI

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 25 травня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, платформа .NET 8, мова C#; бібліотеки Telegram.Bot і WTelegramClient для роботи з Telegram API; OpenAI SDK із доступом до GPT-4o mini; ORM-бібліотека Entity Framework Core; СКБД SQL Server; середовище розробки Visual Studio 2022; система журналювання Serilog.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз існуючих сервісів персоналізації новин і методів фільтрації контенту.
2. Дослідження можливостей Telegram API та GPT-4o mini для семантичного аналізу повідомлень.
3. Проєктування архітектури застосунку та структури бази даних.
4. Реалізація й тестування Telegram-бота на платформі .NET 8 із забезпеченням доставки релевантних новин у реальному часі.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми перевантаження новинами, постановка задачі, лістинги реалізації застосунку, тестові зображення.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-10.04.25	
3	Аналіз літератури з досліджуваної проблеми	11.04.25-14.04.25	
4	Аналіз технічних засобів	15.04.25-20.04.25	
5	Розробка методів	21.04.25-27.04.25	
6	Програмна реалізація	28.04.25-11.05.25	
7	Оформлення пояснювальної записки	12.05.25-20.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Тітова О. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 66 с., 6 табл., 24 рис., 4 дод., 30 джерел.

ТЕЛЕГРАМ-БОТ, ПЕРСОНАЛІЗАЦІЯ НОВИН, GPT-4O MINI, ФІЛЬТРАЦІЯ КОНТЕНТУ, ІНФОРМАЦІЙНЕ ПЕРЕВАНТАЖЕННЯ, .NET 8, ПАТЕРНИ ПРОЄКТУВАННЯ.

Об’єктом роботи є інформаційні потоки публічних каналів Telegram, у яких сьогодні формуються головні стрічки оперативних новин для української аудиторії.

Метою роботи є розробка та експериментальна перевірка сервісу-бота, що у режимі реального часу відбирає й доставляє користувачеві лише ті повідомлення, які відповідають його індивідуальним інтересам, застосовуючи сучасні методи обробки природної мови на базі GPT-4o mini та забезпечуючи прозоре пояснення причин відбору.

Використано методи семантичного аналізу тексту.

У результаті роботи створено крос-платформний Telegram-бот, який забезпечує персоналізовану новинну стрічку, знижує інформаційне навантаження користувача та скорочує середній час отримання релевантної новини.

TELEGRAM BOT, NEWS PERSONALIZATION, GPT-4O MINI, CONTENT FILTERING, INFORMATION OVERLOAD, .NET 8, DESIGN PATTERNS.

The object of this work is the information streams of public Telegram channels, which currently serve as the main sources of real-time news for the Ukrainian audience.

The aim of the work is to develop and experimentally validate a bot service that, in real time, selects and delivers to the user only those messages that match their individual interests, using modern natural language processing methods based on GPT-4o mini, while providing transparent explanations for each selection.

Semantic text analysis methods were applied.

As a result, a cross-platform Telegram bot was developed that provides a personalized news feed, reduces the user’s information load, and decreases the average time to receive relevant news.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (програмний інтерфейс застосунків)

MTProto – Mobile Telegram Protocol (мобільний протокол Telegram)

GPT-4o mini – Generative Pre-trained Transformer 4o mini (генеративна попередньо навчена трансформер-модель 4o mini)

LLM – Large Language Model (велика мовна модель)

MMLU – Measuring Massive Multitask Language Understanding (бенчмарк багатозадачного мовного розуміння)

TTFT – Time To First Token (час до появи першого токена)

MAU – Monthly Active Users (щомісячні активні користувачі)

RSS – Really Simple Syndication (спрощений формат розсилки новин)

NLP – Natural Language Processing (обробка природної мови)

EF Core – Entity Framework Core (фреймворк Entity Framework Core)

ORM – Object-Relational Mapping (об’єктно-реляційне відображення)

DI – Dependency Injection (ін’єкція залежностей)

IoC – Inversion of Control (інверсія керування)

ACID – Atomicity Consistency Isolation Durability (атомарність, цілісність, ізоляція, довговічність)

LTS – Long-Term Support (тривала підтримка версії)

SDK – Software Development Kit (набір засобів розробника)

JSON – JavaScript Object Notation (текстовий формат обміну даними)

Callback-запит – Callback Request (зворотний HTTP-виклик, який надходить від сервера до клієнта)

Handshake – процедура «рукостискання» для встановлення та узгодження параметрів з’єднання

Key-exchange – криптографічний обмін ключами між сторонами

Middleware-конвеєр – Middleware Pipeline (послідовність проміжного ПЗ, крізь яку проходять запити й відповіді)

Mock – Mock Object (імітаційний об’єкт для ізольованого тестування)

Stub – Test Stub (заглушка, що підміняє компонент у тестах)

Unit-тест – Unit Test (модульний тест окремої одиниці коду)

SOLID – Single-responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion (п’ять принципів ООП-проєктування)

CLI – Command-Line Interface (інтерфейс командного рядка)

Singleton –патерн «одиночка», що гарантує єдиний екземпляр класу

Code-first підхід – Code-First Approach (підхід «спочатку код», коли модель БД генерується з класів)

Парсинг – Parsing (синтаксичний розбір і перетворення даних)

Інкапсуляція – Encapsulation (приховування внутрішньої реалізації та відкриття лише публічних інтерфейсів)

Рантайм – Runtime (середовище виконання програми)

ВСТУП

Інформаційні технології, що працюють із текстовими даними, традиційно поділяють на три ключові напрямки: інформаційний пошук, фільтрацію контенту та рекомендаційні системи.

Інформаційний пошук (search) – це алгоритмічний процес знаходження документів, що відповідають формально сформульованому запиту користувача. Класичні приклади – вебпошуковики, корпоративні пошукові системи, а також спеціалізовані каталоги наукових публікацій.

Фільтрація контенту (filtering) передбачає відсікання інформації, яка наперед вважається нерелевантною певному профілю. На практиці це реалізується у системах модерації, «розумних» стрічках соціальних мереж і клієнтських фільтрах електронної пошти.

Нарешті, рекомендаційні системи (recommendation) об’єднують результати пошуку та фільтрації, адаптуючи їх під індивідуальні вподобання користувача. Вони активно застосовуються на стримінгових платформах, у маркетплейсах і медіа-агрегаторах. У певному сенсі рекомендація є зворотним завданням до класичного пошуку: не користувач шукає контент, а система «шукає» і пропонує релевантний контент користувачеві.

Актуальність роботи зумовлена вибуховим зростанням обсягу новин у цифровому просторі та зміщенням каналів споживання з традиційних ЗМІ до месенджерів. Зокрема, Telegram перетворився з простого засобу комунікації на універсальну платформу розповсюдження інформації, де щодня публікуються тисячі новинних повідомлень. У результаті користувач стикається з інформаційним перевантаженням, що ускладнює отримання саме тієї інформації, яка відповідає його інтересам та оперативним потребам. Розробка сервісу, здатного автоматично аналізувати вміст Telegram-каналів, зіставляти його з профілем користувача й своєчасно надсилати релевантні повідомлення, є важливим кроком до підвищення ефективності споживання інформації та зниження когнітивного навантаження.

1 АНАЛІЗ МЕТОДІВ ПЕРСОНАЛІЗАЦІЇ НОВИННОГО КОНТЕНТУ

1.1 Проблема інформаційного перевантаження та потреба персонального фільтру новин

Термін «інформаційне перевантаження» визначає ситуацію, коли обсяг, швидкість та різноманіття вхідних повідомлень перевищують здатність людини обробити їх ефективно. До класичних чинників належать зростання обсягу даних, обмеженість когнітивних ресурсів і часу, нестача довіри до джерел, що змушує користувача перевіряти інформацію вручну.

На цю тему вже проводилися дослідження:

Уперше на брак людської уваги як на «дефіцитний ресурс» звернув увагу Г. Саймон [1], зауваживши, що багатство інформації породжує злиденність уваги.

Системний огляд Eppler & Mengis [2] узагальнив понад 120 робіт і визначив інформаційне перевантаження як стан, коли обсяг, швидкість чи складність інформації перевищують когнітивні можливості суб’єкта. З переходом до Web 2.0 у 2004 році, а потім до мобільних «always-on» (мобільні застосунки, які постійно активні у фоновому режимі) платформ, фокус зсунувся з обсягу контенту на швидкість надходження та функціональні тригери (push-нотифікації, рекомендальні стрічки).

Наведемо статистику по навантаженню на день у таблиці 1.1 [3-5]

Таблиця 1.1 – Статистика по навантаженню на день

Показник	**Значення**
1	**2**
Середня кількість push-нотифікацій на смартфон	46-85 на добу
Новинні матеріали, опубліковані онлайн щодня	≈ 0,85–1 млн

Продовження таблиці 1.1

1	2
Час, витрачений на «довільне» гортання стрічок	139 м хв/добу
Частка користувачів, які усвідомлено уникають новин	39 % (середнє по 47 країнах)

Наслідки для користувачів:

– когнітивне виснаження. Зростає час на перемикання уваги. Дослідження Microsoft показало: середній інтервал безперервної концентрації на смартфоні зменшився з 12 с (2019) до 8 с (2024);

– емоційна втома й новинна втома. Reuters DNR-2024 [6] фіксує стрибок з 29 % (2017 р.) до 39 % (2024 р.) тих, хто «часто або іноді» уникає новин, пояснюючи це «негативністю» та «перевантаженням» контентом;

– соціальні: поширення дезінформації та поляризація, оскільки люди діляться контентом, який не встигли уважно перевірити.

Чому потрібен персональний фільтр?

Найлогічніше місце споживання контенту – застосунки, якими люди користуються щодня. Для України таким є Telegram. Завдяки відкритим каналам там уже існує безліч новинних каналів, але сам месенджер не пропонує тонкої персоналізації. Користувачам важливо бачити лише релевантні теми, скорочуючи шум, отримувати миттєві сповіщення без переходу на сторонні сайти, мати прозорий механізм налаштування інтересів.

1.2 Telegram як платформа для поширення та споживання новин: огляд можливостей API

Починаючи з 2022 року Telegram перетворився з «чергового месенджера» на глобальну інформаційну екосистему: у березні 2025 р.

компанія повідомила про подолання позначки 1 млрд щомісячних активних користувачів (MAU), що на 50 млн більше, ніж у липні 2024 р. [7] Відносно розвинених ринків Telegram посідає восьме місце серед соціальних платформ, але його динаміка випереджає конкурентів, зокрема через «ефект каналів» – унікальний формат односторонньої трансляції повідомлень на необмежену аудиторію.

В Україні месенджер став основним джерелом новин для 73 % населення: за даними щорічного дослідження USAID / Internews (листопад 2024 р.) [8] частка українців, що отримують новини саме тут, зросла ще на 1 % порівняно з 2023 р. Швидкість публікації, відсутність алгоритмічної фільтрації та можливість отримувати push-сповіщення навіть за слабкого мобільного сигналу зробили платформу головним «хабом» оперативної інформації – від зведень Повітряних сил до розслідувань незалежних видань. Telegram випереджає всі традиційні сайти ЗМІ.

На відміну від традиційних соціальних мереж, де стрічка формується алгоритмом, Telegram використовує простішу модель підписки: користувач бачить усі повідомлення каналів, на які підписаний, у хронологічному порядку. Для медіа це означає повний контроль за охопленням – жодних тіньових блокувань чи різниці між підписниками та реальним охопленням.

Ризики: відсутність офіційної модерації. Служба безпеки України зафіксувала > 800 інформаційних вкидів у топ-каналах за пів року; частка дезінформації або малоперевіреного контенту сягає 14 % [9]. Це посилює потребу фільтра, здатного оцінювати ще й достовірність джерела. Однак, в рамках цієї роботи достовірність джерела лежить на відповідальності користувача.

Також впливають функціональні обмеження самого Telegram. Немає гнучкого групування каналів за темами, це лежить на плечах користувачів. Глобальний пошук працює по фразі, але не дозволяє фільтрувати «тільки англомовні IT-новини» або «новини певної країни».

Для інтеграції з платформою розробник має три головні інструменти:

– Bot API – HTTP-інтерфейс для взаємодії від імені бот-акаунта. Це REST-сервіс із JSON-відповідями, який покриває все, що стосується діалогу з користувачем: прийом команд, надсилання повідомлень, реагування на натискання кнопок, прийом платежів тощо;

– Core API (клієнт MTProto) – низькорівневий бінарний протокол, яким користуються офіційні клієнти. Дає повний доступ до вмісту каналів і чатів, у тому числі історичних повідомлень та закритих спільнот (за наявності запрошення).

У роботі використано гібридний підхід: клієнт MTProto безперервно слухає обрані новинні канали і передає пости у чергу, тоді як Bot API обслуговує користувацький інтерфейс – налаштування тем, push-доставку та зворотний зв’язок. Така схема дозволяє оминати обмеження Bot API на читання історії каналу й водночас не зберігати приватні дані користувачів на сервері.

1.3 Поточні підходи до контент-фільтрації й рекомендацій (ключові слова, векторні моделі, LLM)

Сервіс у Telegram отримує повідомлення з каналу, перед ним постає завдання – з’ясувати, чи є цей текст релевантним для конкретного користувача. Протягом останніх двадцяти років у науковій та прикладній літературі сформувалося три «покоління» методів:

– ключові слова й ручні правила;

– статистичні та нейронні векторні моделі;

– великі мовні моделі (LLM), здатні працювати у zero-shot (модель виконує завдання без прикладів, лише за інструкцією) та few-shot (модель отримує кілька прикладів перед виконанням завдання) режимах.

Моделі GPT-3.5 / 4, Claude 3 або Llama 3 здатні вирішувати задачу категоризації в zero-shot режимі: ми передаємо коротку інструкцію на кшталт

«Поверни true, якщо текст стосується кібербезпеки або cloud-інфраструктури». У нещодавньому порівняльному дослідженні [24] GPT-4 перевищив результати fine-tuned BERT (це модель, яку додатково навчено на конкретному завданні).

В роботі використовується GPT-4o mini (індекс «o» – omni) презентовано у липні 2024 р. як малий варіант GPT-4o, оптимізований під швидкість і собівартість. За даними OpenAI [2], модель перевершує GPT-3.5 Turbo на більшості академічних тестів продуктивності, зокрема демонструє MMLU (Measuring Massive Multitask Language Understanding – є засобом оцінки можливостей великих мовних моделей) 0,65 проти 0,48 у 3.5 Turbo, та утримує можливості обробляти різні типи інформації старшої 4-ої серії. У 2025 р. компанія позиціонує її як робоче рішення для real-time застосунків (це програми, які працюють у режимі реального часу) із жорсткими бюджетними вимогами й вимогами до максимально допустимої затримки.

Ця модель була обрана на основі порівняння за наступними критеріями:

– ціна input / output (USD / 1 M токенів) – Це вартість обробки одного мільйона токенів (словесних одиниць) у запиті (input) та відповіді (output);

– TTFT (Time To First Token) – час від моменту надсилання запиту до появи першого токена у відповіді моделі. Чим менше значення, тим швидше користувач отримає результат. У контексті реального часу це критичний параметр: якщо TTFT > 1 сек, сервіс може здаватися «повільним» або непридатним для термінових новин;

– макс. контекст, ток. – це максимальна довжина запиту + відповіді, яку модель може обробити. Вимірюється в токенах;

– zero-shot пояснення – цей показник оцінює здатність моделі не просто класифікувати текст без попереднього навчання (zero-shot), а й обґрунтувати свій вибір.

Порівняння з альтернативами розглянуто у таблиці 1.2.

Таблиця 1.2 – Порівняння мовних моделей

Показник	GPT-3.5 Turbo	GPT-4o mini	GPT-4o (full)
Ціна input / output (USD / 1 M ток.)	0,50 / 1,50	0,15 / 0,60	2,00 / 10,00
TTFT, с	0,65	0,44	0,62
Макс. контекст, ток.	16 k	128 k	128 k
Zero-shot пояснення	+	++	++

1.4 Аналіз існуючих рішень

Перед розробкою Telegram-сервісу доцільно з’ясувати, які вже є існуючі конкурентні продукти й де вони не покривають потреб користувача.

З прямих аналогів існує Digest Technology Bot. Це один із найновіших українських ботів-дайджестів: у квітні 2025 р. [10] розробники презентували функцію добового й вечірнього «шот-листу» новин з до п’яти тем, що задаються ключовими словами. Алгоритм працює на регулярних виразах: бот відсіює пости, де є точний збіг із білим списком, і відправляє зведення двічі на добу. Рекламний банер у TAdviser підкреслює, що «читачі отримують лише півтора-два десятки заголовків, аби уникнути спаму», – але саме це означає втрату оперативності, бо живі push-повідомлення замінені статичним дайджестом. Зовнішній вигляд застосунку показано на рисунку 1.1.

З непрямих аналогів існують RSS-агрегатори. Наприклад Feedly. Це класичний RSS-агрегатор із модулем Leo – AI-фільтра, який може прибирати дублі, підсвічувати «важливі» статті. На тарифі Pro+ (8,25$ / міс) користувачеві доступно до 2 500 джерел і базовий набір «AI-skills». Система чудово працює для англомовних корпоративних блогів, але українські RSS-стрічки Feedly опитує раз на 5-15 хв. залежно від навантаження. Додатково сам

Leo поки не має української мови, тож зупиняється на keyword-match (це підхід, при якому система шукає точні збіги ключових слів у тексті) або помилково класифікує такі пости як «Other». Головний недолік для нашого випадку – відсутність інтеграції з Telegram-каналами: користувач мусить шукати RSS-версію або взагалі обходити улюблені джерела.

Рисунок 1.1 - Digest Technology: Дайджест Telegram-бот

Також існують мобільні новинні застосунки з власними алгоритмами. Наприклад Google News. Мобільний застосунок використовує складну комбінацію машинного навчання й «editorial signals» (сигнали або критерії, які використовуються редакторами, щоб оцінити якість, важливість чи достовірність новинного матеріалу) для вкладки «For You». Зовнішній вигляд застосунку показано на рисунку 1.2.

На глобальних англомовних URL індексація відбувається швидко, але локальний контент часто з’являється із затримкою в десятки хвилин. Статус сайту самого Google підтверджує залежність швидкості від стану центрального індексу. Для українських регіональних ЗМІ дослідження Media Development Foundation [5] також фіксує лаг 15-45 хв між публікацією і появою у Google News. Додаймо до цього, що сервіс не має доступу до

Telegram-джерел, а пояснення алгоритму залишається «чорним ящиком» – і стає зрозуміло, чому користувачі паралельно тримають відкритим месенджер.

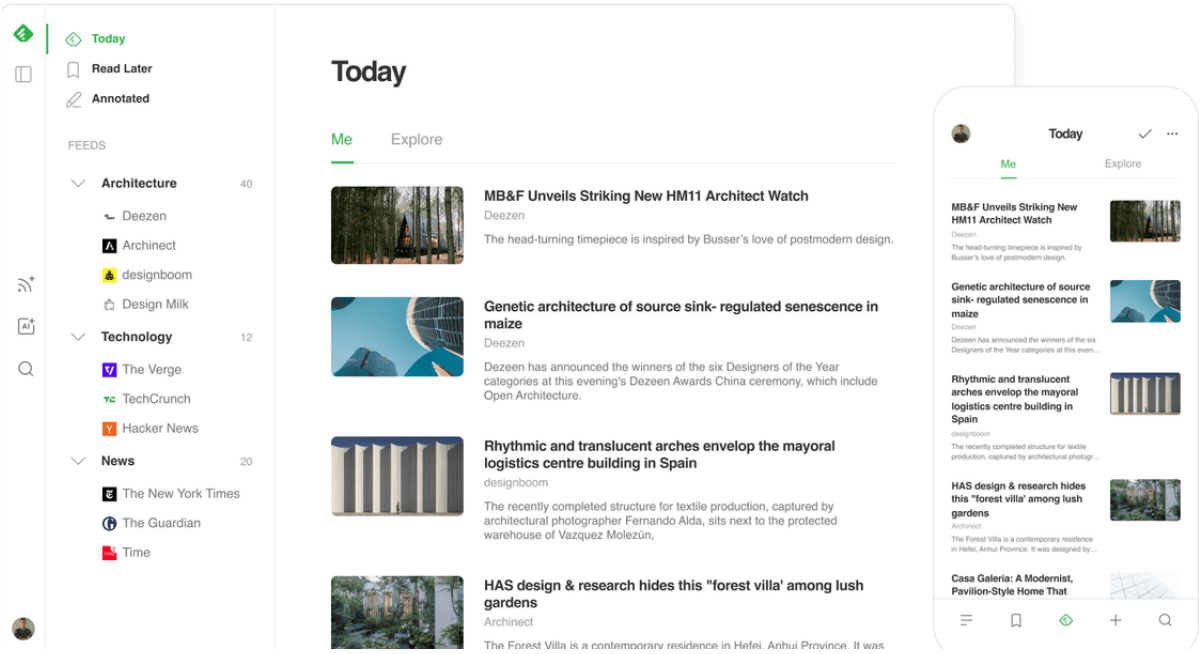

Рисунок 1.2 – Feedly News Reader

На рисунку 1.3 можна побачити як виглядає Google News.

Що ж відрізняє розроблюваний проєкт від вище наведених:

– миттєва доставка (< 1 с). На відміну від Digest Technology Bot, який збирає дайджест раз на 12 годин, бот слухає всі нові пости через MTProto й шле push після семантичної перевірки – середня затримка ≈ 1-2 с, що важливо під час повітряних тривог чи «breaking news»;

– покриття Telegram-джерел + мультимовність. Feedly й Google News ігнорують закриті або напівофіційні Telegram-канали, що давно стали первинним джерелом інформації в Україні. Розроблений сервіс працює саме з ними й водночас обробляє українську, англійську та змішані пости без морфологічних втрат;

– економіка та доступність. Собівартість GPT-4o mini (≈ 0,00075 $ на пост) дає змогу дуже дешево підтримувати проєкт, якщо він стане комерційним;

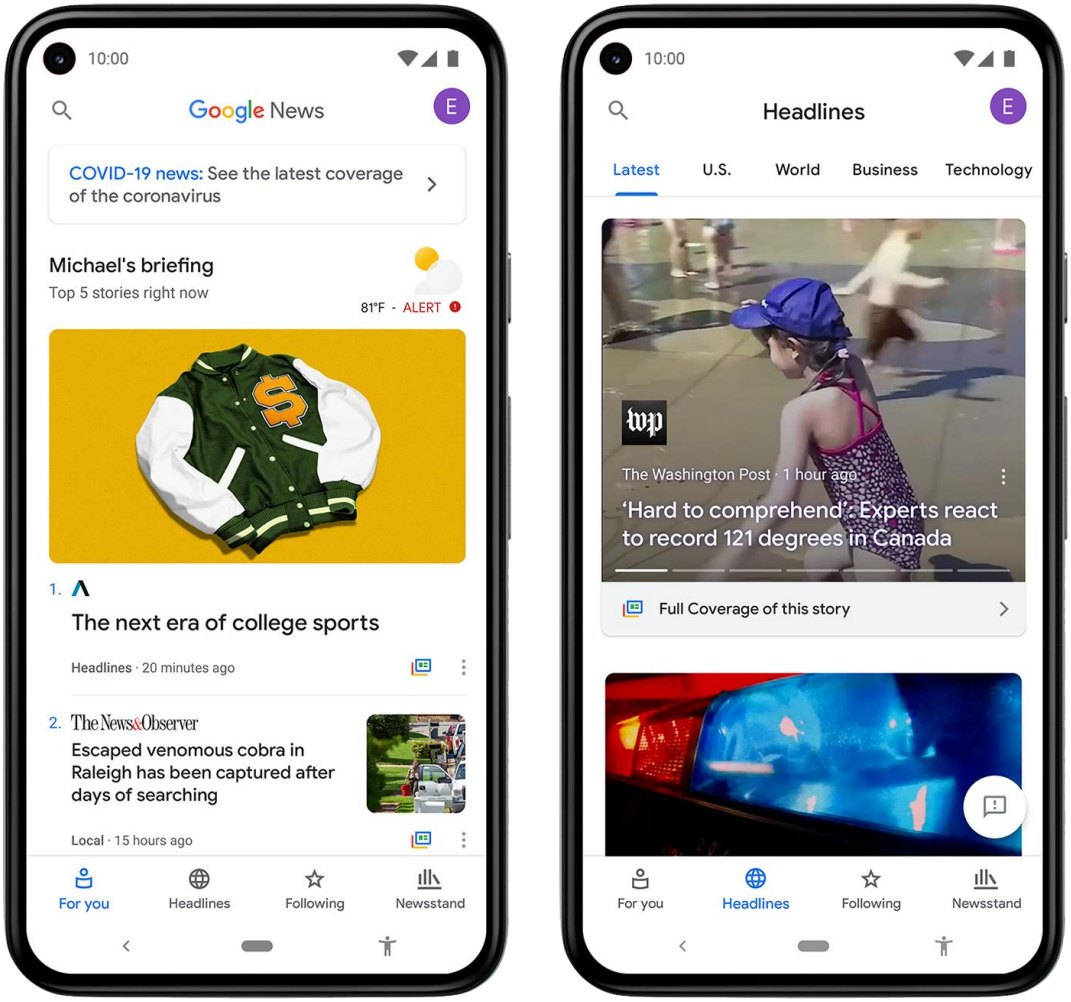

Рисунок 1.3 – Google News

– гнучке налаштування через один інтерфейс. Користувач не шукає RSS-посилання, не ставить окремих застосунків. Усі команди виконуються у тому самому застосунку, де він уже щодня проводить час.

У сукупності це робить сервіс єдиним рішенням, що поєднує: реальний час Telegram-потоку, глибинний семантичний аналіз GPT-4-рівня й низьку вартість. Тим самим він перекриває слабкі місця одразу трьох класів конкурентів і пропонує користувачу простий та зручний досвід споживання новин із мінімальним шумом.

1.5 Постановка задачі

Об’єктом роботи є інформаційні потоки публічних каналів Telegram, у яких сьогодні формуються головні стрічки оперативних новин для української аудиторії.

Метою роботи є розробка та експериментальна перевірка сервісу-бота, що у режимі реального часу відбирає й доставляє користувачеві лише ті повідомлення, які відповідають його індивідуальним інтересам, застосовуючи сучасні методи обробки природної мови на базі GPT-4o mini та забезпечуючи прозоре пояснення причин відбору.

Для досягнення мети необхідно вирішити такі завдання:

– зібрати корпус Telegram-повідомлень;

– спроєктувати схему бази даних уподобань користувачів;

– реалізувати MTProto-лісенер для безперервного отримання контенту;

– реалізувати звернення до GPT-4o mini для семантичної оцінки релевантності;

– створити дружній інтерфейс бота з підтримкою команд.

2 ПРОЄКТУВАННЯ СИСТЕМИ, КОМПОНЕНТІВ, БАЗИ ДАНИХ

2.1 Вибір технологій та обґрунтування

У межах роботи визначальною стала потреба побудувати сервіс, який у режимі реального часу спостерігатиме за публікаціями Telegram-каналів, та оцінюватиме їхню релевантність індивідуальним уподобанням користувача за допомогою великої мовної моделі та доставлятиме відібраний контент через чат-бот. Щоб забезпечити таку функціональність, було сформовано критерії вибору технологій: сумісність із Telegram-екосистемою, легкість інтеграції з OpenAI-API, можливість розгортання на сервері та відсутність ліцензійних обмежень для проєкту.

Базовою платформою обрано .NET 8 (LTS), оскільки вона поєднує високу продуктивність з крос-платформеними офіційними образами для Docker, що суттєво спрощує подальше розгортання.

Бібліотека Telegram.Bot є офіційним .NET-SDK для Bot API. Бібліотека підтримує усі методи Telegram Bot API 6.9 і містить високорівневі обгортки для роботи з вебхуками, отримання та надсилання повідомлень, клавіатурою й callback-запитами. Готові типи дозволяють швидко розгорнути довірений канал між ботом і користувачем та обробляти тисячі оновлень паралельно завдяки асинхронному API. Наявність LTS-версії та активного супроводу спільнотою мінімізує ризики при подальших оновленнях Telegram-протоколу.

WTelegramClient доповнює попередню бібліотеку, оскільки реалізує низькорівневий протокол MTProto. На відміну від Bot API, MTProto відкриває доступ до публічних каналів у реальному часі, зокрема дозволяє читати повідомлення без затримок на веб-хуки та обходить обмеження на кількість запитів. Клас Client автоматично опрацьовує handshake, key-exchange та надсилання авторизаційних даних (API ID, Hash, OTP), а також утримує постійне TCP-з’єднання з дата-центром Telegram. Таким чином сервіс отримує надійний «стрім» новин із необхідною пропускною здатністю.

Entity Framework Core обрано як ORM-шар для реляційного сховища. EF Core надає виразний LINQ-API, засоби міграцій та змін блоб-сторінок без простою. Функції Change Tracking та Concurrency Tokens особливо корисні при зберіганні взаємопов’язаних таблиць, де потрібна ACID-цілісність. Крім того, у разі переходу на PostgreSQL чи Azure SQL знадобиться лише заміна провайдера без зміни доменних моделей.

Serilog разом із розширенням Serilog.AspNetCore забезпечує структуроване логування. Завдяки конфігурації JSON можна спрямовувати записи одночасно у консоль, файл та віддалені агрегатори без зміни прикладного коду. Крім того, Serilog інтегрується з middleware-конвеєром ASP.NET Core через єдиний виклик builder.Host.UseSerilog(), не вимагаючи кастомного провайдера.

Microsoft.Extensions.DependencyInjection слугує стандартним контейнером інверсії керування. Вибір саме цього пакета, а не сторонніх IoC-фреймворків (Autofac, LightInject), пояснюється його вбудованістю у .NET 8, відсутністю зовнішніх залежностей. Він підтримує життєві цикли Singleton, Scoped і Transient, що дозволяє коректно реєструвати довгоживучі об’єкти (наприклад Client із WTelegramClient) та короткочасні сервісні класи. Крім того, Serilog.Extensions.Hosting уже постачається з адаптером до цього контейнера, тож не виникає дублювання об’єктів у різних DI-системах

Сумарно зазначений набір бібліотек утворює цілісний, взаємодоповнюваний стек: Telegram.Bot та WTelegramClient забезпечують повний спектр інтеграції з Telegram, EF Core – зручний та переносний доступ до даних, Serilog – спостережуваність, а вбудований Dependency Injection – гнучкість та тестованість коду.

2.2 Модульне проєктування

На етапі концептуального проєктування систему розділено на слабо зв’язані модулі, що локалізує функціональність і мінімізує ризик каскадних змін: оновлення одного сервісу майже ніколи не потребує перекомпіляції чи повторного розгортання інших. Кожен модуль працює окремо, тому вузькі місця можна масштабувати горизонтально, не збільшуючи ресурси всього застосунку. Усі компоненти збираються під єдину цільову рамку .NET 8, поділяють спільний набір NuGet-пакетів для логування, конфігурації та DI і дотримуються єдиних код-конвенцій, що спрощує супровід.

Також були використані деякі правила проєктування системи:

– Domain-Driven Design: бізнес-поняття (канал, користувач, інтерес, пост) виокремлюються на рівні доменних моделей і живуть незалежно від технічних деталей транспорту чи сховища;

– подіє-орієнтована взаємодія: між модулями передаються лише події з мінімальним набором полів, тому будь-який сервіс можна замінити або масштабувати без зміни контрактів.

Логічне ядро складається з чотирьох сервісів, між якими організовано обмін подіями. TgListenerService виступає асинхронним вхідним шлюзом: він автентифікується в Telegram, підтримує стійке з’єднання, керує списком каналів і для кожного нового повідомлення формує стандартну подію NewPostEvent, публікуючи її до внутрішньої черги. ChatGptService, що виконує семантичну оцінку, спершу готує промпт, додаючи ключові слова інтересів користувача й системні інструкції, далі викликає GPT-4o mini з детермінованими параметрами, а потім парсить відповідь у модель PostEvaluationResult, де містяться прапорець релевантності та короткий конспект.

TgListenerService – сервіс, що підключається до публічних Telegram-каналів і в режимі реального часу оброблює поступальний потік повідомлень та розсилає повідомлення користувачам. У модулі реалізовано три базові

функції. Перша – автентифікація й підтримання стійкого з’єднання з Telegram-інфраструктурою. Друга – менеджмент каналів, які треба слухати. Він надає реалізацію інтерфейсу через який можна підписатися на канал за публічним посиланням. Третя – розсилка підписаним користувачам сповіщення про новий пост, який їх може зацікавити. Завдяки такому поділові решта компонентів працює з абстракцією у вигляді інтерфейсів і не переймається мережевими деталями.

Сервіс ChatGpt одержує стандартизовані події від TgListenerService для подальшої оцінки. У процесі обробки присутні 3 етапи:

– формування запиту, де створюється контекст для мовної моделі, додаючи ключові слова користувача, інформацію про його інтереси та системні інструкції;

– етап виклику моделі з задаванням параметрів щодо формату виводу;

– приймання відповіді і перетворення її на очікувану модель.

Сервіс Database зберігає довготривалі об’єкти – профілі користувачів, канали, підписки та історію взаємодій. Доступ відбувається лише через репозиторії EF Core 9, що приховують SQL-деталі й дозволяють у перспективі мігрувати на інший тип сховища. З погляду архітектури між Database і рештою модулів укладено контракт репозиторію: жоден модуль не формує SQL-запитів напряму, натомість викликає методи репозиторіїв, а це спрощує майбутню міграцію на інше сховище.

Сервіс TgUiService забезпечує двосторонній зв’язок з користувачем через Telegram-бота. В нього є дві основні задачі. Перша – прийом команд і повідомлень від користувача та перетворення їх на доменні дії (зміна тем, запит дайджесту, відмова від підписки). Друга – ведення актуальної інформації про користувачів через виклик репозиторіїв бази даних. Цей модуль ізолює бізнес-логіку від Telegram-специфіки; якщо в майбутньому з’явиться мобільний застосунок або вебкабінет, достатньо буде створити ще один front-end-модуль, не торкаючись ядра.

2.3 Патерни проєктування системи

Паттерн Dependency Injection (DI), або ін’єкція залежностей, є фундаментальним принципом об’єктно-орієнтованого програмування та широко використовується в сучасних програмних архітектурах для досягнення слабкого зв’язування між компонентами системи. Його суть полягає у передачі зовнішніх залежностей об’єкту не шляхом створення їх усередині класу, а через конструктор, методи або властивості. Це дозволяє досягти гнучкості, легкості у тестуванні та зменшення жорсткої зв’язності між модулями. Схему патерну можна побачити на рисунку 2.1.

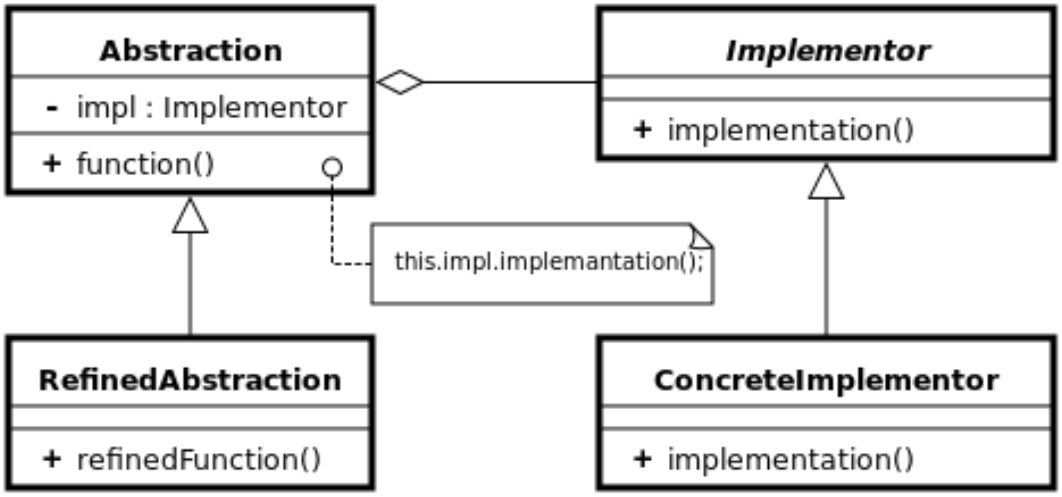

Рисунок 2.1 – Патерн Dependency Injection

Одна з головних переваг DI полягає у спрощенні процесу модульного тестування. Завдяки можливості передавати залежності ззовні, розробник може легко підставити фіктивні або замінні реалізації інтерфейсів (наприклад, mock або stub об’єкти), що суттєво полегшує розробку unit-тестів. Крім того, зміна конкретної реалізації залежності не вимагає модифікації самого класу, що покращує підтримку та розвиток системи.

У рамках роботи використання паттерна Dependency Injection дозволяє відокремити логіку роботи сервісів від логіки їх створення. Вся конфігурація

залежностей виконується централізовано на етапі запуску додатку. Це дозволяє гнучко керувати життєвим циклом сервісів, а також забезпечує інтеграцію сторонніх бібліотек – наприклад, систем логування, доступу до бази даних, або API-клієнтів.

Таким чином, DI не лише покращує структуру коду, роблячи її чистішою та підтримуваною, але й є ключовим механізмом для побудови масштабованої та розширюваної архітектури. У проєкті паттерн DI використовується для ін'єкції основних сервісів – зокрема, сервісів логування, взаємодії з Telegram API, GPT-клієнту, а також для організації взаємодії з базою даних через ApplicationDbContext. Це дозволяє гнучко замінювати частини системи без впливу на інші компоненти, дотримуючись принципів SOLID та забезпечуючи розділення відповідальностей між модулями.

Паттерн Repository (або репозиторій) є структурним шаблоном, що слугує посередником між логікою бізнес-рівня та рівнем доступу до даних. Його основна мета полягає у створенні абстракції над джерелами даних – базою даних, зовнішніми сервісами або навіть файловою системою. Це дозволяє приховати технічні деталі збереження, пошуку та оновлення об'єктів, представляючи доступ до даних через зручний інтерфейс. Схему патерну можна побачити на рисунку 2.2.

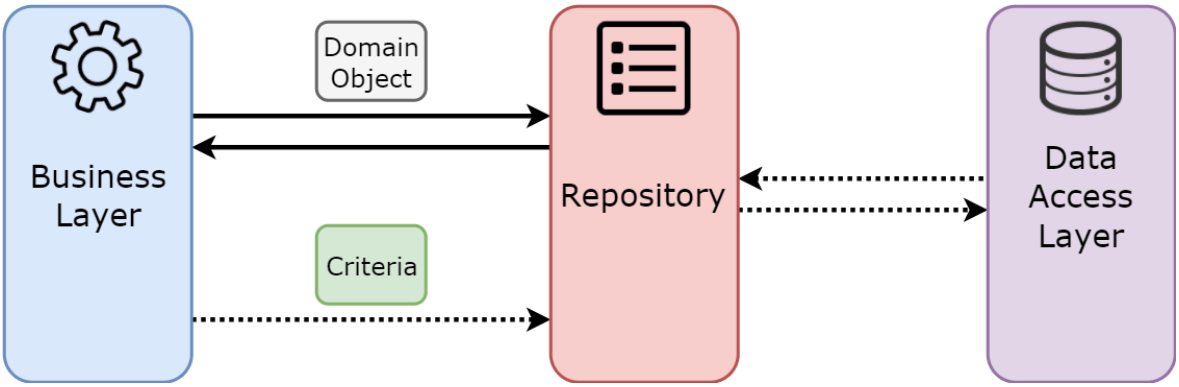

Рисунок 2.2 – Патерн Repository

Використання паттерна Repository значно покращує читабельність і підтримуваність коду. Він дозволяє уникнути дублювання SQL-запитів або

викликів Entity Framework API по всій системі, централізуючи логіку доступу до певного типу даних в одному класі. Також він сприяє дотриманню принципу Single Responsibility з SOLID, оскільки обробка запитів до даних ізолюється від бізнес-логіки, а також забезпечує можливість легко замінити джерело даних (наприклад, з реальної бази на фіктивну в тестах).

У даній роботі реалізація паттерна Repository дозволяє уніфікувати взаємодію з базою даних за допомогою Entity Framework Core. Наприклад, для кожного доменного типу створюється окремий репозиторій, що відповідає за всі CRUD-операції. Надалі ці репозиторії впроваджуються у відповідні сервіси через механізм Dependency Injection, завдяки чому вся логіка маніпуляцій із даними зосереджена в одному місці.

Крім того, при потребі в складніших операціях, таких як фільтрація, сортування або проекція, репозиторій може містити спеціалізовані методи, які реалізують необхідні запити з використанням LINQ. Це сприяє збереженню єдиної точки входу до даних та полегшує відстеження і налагодження помилок, пов’язаних із збереженням чи отриманням інформації.

Таким чином, паттерн Repository у проєкті виконує роль структурного мосту між даними та бізнес-рівнем, дозволяючи розробляти систему згідно з принципами чистої архітектури, спрощувати супровід коду та забезпечувати високий рівень масштабованості. Це також полегшує впровадження unit-тестів, оскільки залежності можна легко замінити на moq-реалізації.

Паттерн Factory Method (або фабричний метод) належить до групи породжувальних патернів і дозволяє делегувати створення об'єктів підкласам, не вказуючи конкретного класу створюваного об’єкта. Іншими словами, він інкапсулює логіку створення об’єктів у спеціальні «фабричні» методи або класи, що дозволяє розширювати або змінювати логіку ініціалізації без зміни коду, який користується цими об’єктами. Це особливо корисно у випадках, коли процес створення об’єкта є складним або залежить від зовнішніх параметрів (наприклад, конфігурації, середовища виконання чи контексту запиту). Схему патерну можна побачити на рисунку 2.3.

У роботі фабричний метод застосовується для ініціалізації ApplicationDbContext для виклику міграції бази даних через CLI.

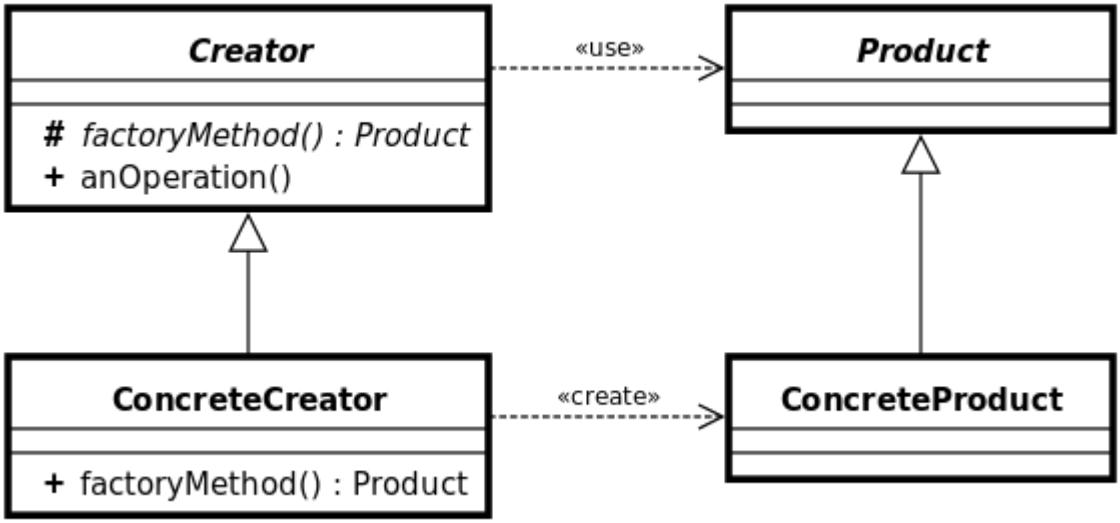

Рисунок 2.3 – Патерн Factory Method

Паттерн Singleton (Одиночка) – це один із найпоширеніших породжувальних шаблонів, що гарантує існування лише одного екземпляра певного класу протягом усього життєвого циклу програми. Його основна ідея полягає в тому, щоб забезпечити глобальну точку доступу до цього єдиного об’єкта, уникаючи повторного створення об’єкта та забезпечуючи контроль над його станом. Це особливо актуально в тих випадках, коли одночасна присутність кількох екземплярів може призвести до помилок, конфліктів або дублювання роботи.

Реалізація Singleton передбачає створення статичного методу або властивості, через яку клієнт отримує доступ до єдиного об’єкта. Це дозволяє централізовано управляти життєвим циклом екземпляра та забезпечити його створення лише при першому зверненні (лінива ініціалізація) або одразу при запуску програми (рання ініціалізація).

У контексті .NET-розробки, реалізація паттерна Singleton тісно пов’язана з системою впровадження залежностей (Dependency Injection). У

таких випадках клас реєструється як singleton-сервіс через метод services.AddSingleton<T>(), що гарантує створення лише одного об’єкта протягом усього життєвого циклу застосунку. Це дозволяє уникнути надлишкового використання ресурсів, забезпечити узгоджений стан сервісу та зберігати глобальний контекст, якщо це необхідно. Схему патерну можна побачити на рисунку 2.4.

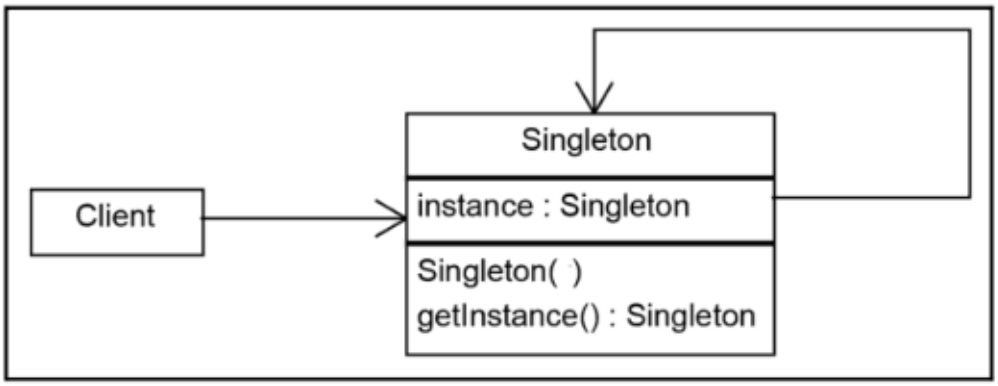

Рисунок 2.4 – Патерн Singleton

У проєкті паттерн Singleton використовується для реєстрації глобальних об’єктів, таких як TelegramBotClient (Telegram.Bot), Client (WTelegram) та OpenAIClient (OpenAI). Ці об’єкти не повинні дублюватися, оскільки кожен їхній екземпляр несе в собі загальний стан, який має бути однаковим для всіх компонентів системи. Зокрема GPT-клієнт – спільно використовувати єдиний ключ і параметри взаємодії з API. Таким чином, використання паттерна Singleton забезпечує контроль над ресурсами, покращує продуктивність та спрощує архітектуру застосунку, уникаючи надмірної зв’язності.

2.4 Проєктування бази даних

У центрі сервісу персоналізованих новин лежить реляційна база даних, покликана синхронізувати три незалежні простори інформації: простір

користувачів Telegram-бота, простір джерел (каналів) і простір подій взаємодії між ними. Від коректності схеми залежить не тільки швидкодія рекомендаційного алгоритму, а й можливість масштабувати систему без дорогих міграцій даних. Після порівняльного аналізу було обрано тандем SQL Server Express (LocalDB) + Entity Framework Core 9.

У контексті проєкту вибір реляційної СУБД та ORM Entity Framework Core продиктований насамперед природою самих даних. Зміст системи – це суворо структуровані сутності «користувач», «канал» і «підписка», зв’язані типовими відношеннями «один-до-багатьох» та «багато-до-багатьох». Реляційна модель, закладена в SQL Server, автоматично встановлює зв’язки між таблицями через ключі і сам стежить за коректністю даних, без потреби писати додаткові перевірки в коді. Гарантії ACID-транзакцій, що їх надає SQL Server, важливі для сценаріїв, де підписка або відмова від каналу повинні зберігатися атомарно: неконсистентність навіть одного запису може призвести до втрати налаштувань користувача.

Використання Entity Framework Core додає до цього функціонального фундаменту шар продуктивності розробки. По-перше, «code-first»-підхід дозволяє описати модель у вигляді C#-класів, а всі зміни схеми відстежуються міграціями – це звільняє від ручного написання SQL-скриптів і знижує ризик помилок при еволюції бази. По-друге, LINQ-вирази, що автоматично транслюються EF Core у T-SQL, забезпечують інтуїтивний спосіб формулювання запитів, залишаючись читабельними. Отже валідація відбувається ще на етапі компіляції. По-третє, EF Core має вбудовані механізми кешування змінених сутностей і відкладеного виконання, що мінімізує кількість звернень до БД без потреби ручного мікроменеджменту пулу з’єднань.

Окремого згадування вартує екосистема навколо SQL Server і EF Core. Наявність зрілих інструментів – від Visual Studio Data Tools до Azure Monitor і Query Store – спрощує масштабування системи, що особливо цінно для проєкту з обмеженими ресурсами. Крім того, SQL Server Express доступний

безкоштовно і не накладає ліцензійних обмежень у навчальному середовищі, а перехід на Azure SQL Database у виробничому оточенні відбувається без зміни коду: EF Core потребує лише іншого провайдера.

Нарешті, з погляду довгострокової підтримки реляційна СУБД дає голові рішення до аналітики. Під час запуску проєкту дані підписок і їхню динаміки можна безпосередньо використовувати для подальшого аналізу. Таким чином, поєднання SQL Server з EF Core забезпечує баланс між формальною надійністю, швидкістю розробки й подальшою розширюваністю, що робить цей вибір найбільш обґрунтованим для сервісу персоналізованого пошуку новин.

Схему можна уявити як чотири сутності у таблицях 2.1-2.4

Таблиця 2.1 – TgUser – ядро профілю користувача

Назва поля	**Тип даних**	**Пояснення**
1	**2**	**3**
ChatId	Int64	Унікальний ідентифікатор користувача в Telegram. Це ключове поле, яке використовується для ідентифікації користувача в системі. Завдяки тому, що значення `ChatId` є незмінним і унікальним для кожного чату, воно дозволяє точно визначати користувача без потреби в додатковій автентифікації.
UserTag	String	Ім’я користувача в Telegram (нікнейм), який зазвичай починається зі знаку @. Це поле використовується для відображення користувача в інтерфейсі адміністратора, а також може застосовуватись для персоналізованого спілкування з користувачем.

Продовження таблиці 2.1

1	2	3
FirstName	String	Ім’я користувача, яке витягується з Telegram-акаунта. Використовується в повідомленнях для персоналізації звернення, а також може допомогти у відлагодженні або аналітиці, наприклад, у випадку масових кампаній або опитувань.
LastName	String	Прізвище користувача з Telegram. Як і FirstName, може використовуватись для персоналізації або аналітики, однак є необов’язковим, оскільки деякі користувачі Telegram не вказують прізвище.
Preferences	String	Впродобання, які ввів користувач. Ці дані використовуються для налаштування персоналізованої видачі новин.
State	Int	Числовий код, що відображає поточний статус користувача в рамках сценарію взаємодії з Telegram-ботом. Це поле дозволяє реалізувати просту модель станів для діалогового інтерфейсу
Language	Int	Числовий код, що відповідає обраній мові користувача.

Таблиця 2.2 – Subscribe – таблиця для зв’язку багато до багатьох між користувачами і каналами

Назва поля	Тип даних	Пояснення
ChatId	Int64	Унікальний ідентифікатор користувача в Telegram.
ChannelId	Int64	Унікальний ідентифікатор Telegram-каналу.

Таблиця 2.3 – Channel – каталог джерел, які система відстежує

Назва поля	**Тип даних**	**Пояснення**
ChannelId	Int64	Унікальний ідентифікатор Telegram-каналу. Значення автоматично отримується під час взаємодії з Telegram API або бібліотекою WTelegram, і є незмінним у межах Telegram.
Name	String	Повна назва каналу, яка відображається в інтерфейсі Telegram. Це поле використовується для зручності під час виведення інформації про джерело новини або в адміністративному інтерфейсі для керування списком каналів.
Username	String	Унікальне публічне ім’я каналу в Telegram, що використовується для прямого посилання на нього

Таблиця Session використовується для ведення журналу часу роботи системи. Використовується при старті системи для обробки повідомлень з каналів, які були опубліковані підчас простою системи.

Таблиця 2.4 – Таблиця Session

Назва поля	**Тип даних**	**Пояснення**
SessionId	Guid	Автоматично створений ідентифікатор сесії
StartDate	DateTime	Дата і час запуску системи
EndDate	DateTime	Дата і час вимкнення системи

За замовчанням кластерні індекси збігаються з первинними ключами (PK_Users, PK_Channels, PK_Sessions, PK_Subscribe). Така конфігурація

покриває 90 % OLTP-операцій, адже всі внутрішні JOIN-и виконуються через прямі ключі.

Концептуальна й фізична схеми, індексація під критичні запити, а також продумана політика резервного копіювання формують цілісну, завершену картину. У межах проєкту це означає, що наступні етапи розробки можуть спиратися на стабільну платформу зберігання даних, не побоюючись дорогих переробок чи втрати узгодженості інформації. У ширшій перспективі саме ця база даних стане точкою, де збиратимуться та перехрещуватимуться потоки даних від Telegram-каналів, користувачів та інтелектуального ядра, перетворюючи сервіс на повноцінну інфраструктуру персоналізованої інформаційної підтримки.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВІСУ ДЛЯ ПОШУКУ НОВИН

3.1 Функціональні та нефункціональні вимоги

У роботі передбачається реалізувати повний цикл оброблення інформаційного потоку Telegram-каналів – від первинного формування профілю користувача до доставлення фільтрованих повідомлень.

Далі розглянемо функціональні вимоги.

Має виконуватися керування профілем та уподобаннями. Система має автоматично ідентифікувати користувача за унікальним Telegram Chat ID, створюючи відповідний запис у таблиці Users під час першого звернення до бота. Передбачається підтримка інтерфейсу, що забезпечує додавання, редагування та перегляд тематичних ключових слів. Останні зберігатимуться у таблиці Users. Очікується можливість мультимовного введення тем.

Має виконуватися управління джерелами новин. Команди через інтерфейс користувача мають забезпечити інтерактивне додавання та вилучення публічних Telegram-каналів за HTTP-посиланнями. Після успішної підписки у таблиці Channels необхідно зберігати метадані джерела.

Має виконуватися оброблення потоку повідомлень у реальному часі. Компонент TgListenerService повинен підтримувати стійке з’єднання MTProto та безперервно транслювати нові повідомлення.

Має виконуватися семантична фільтрація на основі GPT-4o mini. Сервіс ChatGptService має формувати промпти, що включають системну інструкцію, мовну підказку та повний текст поста. Виклик до моделей GPT-4o mini слід здійснювати через офіційний SDK OpenAI. На виході очікується JSON-об’єкт, який міститиме «так/ні» результат Interesting та коротке резюме поста Summary. Для кожного релевантного повідомлення необхідно формувати подію для надсилання сповіщення.

Має виконуватися доставлення повідомлень про надходження події. Модуль TgUiService повинен генерувати картку новини, що включатиме назву

каналу, стисле резюме та інтерактивну кнопку «Переглянути повністю». Її натискання має направляти користувача на оригінальний пост.

Має виконуватися логування. Для спостережуваності та відлагодження необхідно інтегрувати логування із виведенням релевантних даних про усі події що відбуваються у застосунку: від звернень до мовної моделі, до помилок що виникли при виконанні. Логування має записуватися у консоль та файли.

Має бути доступна можливість зміни мови, з Англійської на Українську і навпаки.

Далі розглянемо нефункціональні вимоги.

Система має бути продуктивною. Система має забезпечувати наскрізний час реакції – від моменту появи повідомлення у відстежуваному Telegram-каналі до доставлення push-сповіщення користувачеві – не більше ніж 2 с, за умови паралельного моніторингу 500 каналів. Середній час генерації відповіді мовної моделі GPT-4o mini не повинен перевищувати 1 с при довжині вхідного тексту до 4096 токенів.

Система має бути масштабованою. Архітектура має бути написана у вигляді мікросервісів для виконання горизонтального масштабування кожного мікросервісу окремо (TgListenerService, ChatGptService, TgUiService). Щоб при необхідності додавання можливості розгортання більшої кількості сервісів через оркестратор контейнерів це можна було зробити без серйозних змін у вже існуючому коді.

Система має бути надійною та відмовостійкою. Тобто у випадку вимкнення застосунку та його ввімкнення, не було загублено жодного поста, що надійшов під час вимкнення.

Система має бути безпечною та надійною. Для цього мають бути виконані наступні вимоги:

– HTTPS є обов’язковим для всіх зовнішніх запитів (OpenAI API, Telegram Bot API);

– токени Telegram і OpenAI зберігаються лише у засекречених секціях;

– персональні дані обмежуються Chat ID, списком тем і підписок; текст користувацьких чатів не зберігається.

3.2 Програмна реалізація застосунку

3.2.1 Реалізація бота на платформі Telegram

Підсистема користувацького інтерфейсу сервісу побудована навколо Telegram-бота, який виконує роль взаємодії з кінцевим користувачем. У структурі системи модуль Telegram-бота розгорнуто як фоновий сервіс ASP.NET Core на базі стандартного контракту IHostedService. Такий підхід дозволяє запускати компонент у тому ж процесі, що й решта мікросервісів, забезпечуючи спільний життєвий цикл, централізовані механізми конфігурації та логування.

У межах цього пункту наведено деталізований опис ключових програмних артефактів.

Компонент TgUiHostedService реалізує типовий шаблон сервісу платформи .NET 8 (IHostedService) і відповідає за інтеграцію Telegram Bot API з внутрішнім UI-шаром застосунку. Клас реєструється у контейнері залежностей як singleton і запускається разом із хостом, тому не потребує окремої точки входу.

У конструкторі через DI впроваджується:

– ITelegramBotClient – низькорівневий клієнт Bot API, конфігурований токеном доступу;

– ILogger<TgUiHostedService> – адаптер до для структурованого логування;

– HandlerService – фасад, що інкапсулює бізнес-логіку оброблення вхідних повідомлень;

– UserService – сервіс домену, відповідальні за синхронізацію профілю користувача.

Метод StartAsync ініціалізує CancellationTokenSource, формує екземпляр ReceiverOptions без фільтрації і викликає StartReceiving у клієнта Telegram. У такий спосіб створюється асинхронний цикл оброблення оновлень, який виконується у пулі потоків і не блокує життєвий цикл хоста.

Метод StopAsync коректно завершує роботу сервісу через скасування токена, забезпечуючи коректне завершення роботи програми без втрати подій.

Основна логіка розташована у приватному HandleUpdateAsync. Спершу з об’єкта Update дістається профіль користувача (RetrieveUserFromUpdate – розширювач, що дозволяє це виконати на основі різних типів оновлень). Далі UserService виконує операцію EnsureUserExistsAndIsUpdatedAsync, яка або створює новий запис у таблиці Users, або актуалізує атрибути наявного. Отриманий доменний об’єкт передається у HandlerService, де відбувається маршрутизація:

– UpdateType.Message спрямовується до HandleMessageAsync для розбору текстових і медіа-повідомлень;

– UpdateType.CallbackQuery – до HandleCallbackQueryAsync, що обробляє інтерактивні кнопки інтерфейсу.

Усі інші типи оновлень логуються як необроблені, що полегшує подальше розширення функціоналу. Обидва шляхи повертають Task, тому контекст виконання не блокується і може масштабуватися паралельно.

Метод HandleErrorAsync перехоплює винятки, що надходять із каналу отримання оновлень. Для класу ApiRequestException формується деталізований рядок з кодом та описом помилки Telegram API, а для будь-якої іншої виняткової ситуації виконується повне логування. Це спрощує пошук причин відмови й відповідає нефункціональним вимогам щодо спостережуваності.

Таким чином, TgUiHostedService забезпечує низькорівневу обв’язку між Telegram-інфраструктурою та доменними сервісами, ізолюючи мережеві аспекти від бізнес-логіки, підтримує асинхронну, масштабовну обробку подій і вписується у загальну мікросервісну структуру системи.

Компонент HandlerService виконує функцію прикладного контролера, що інкапсулює всю маршрутизацію вхідних подій Telegram-бота. Об’єкт створюється через механізм інверсії керування та одержує чітко визначений набір залежностей:

– ITelegramBotClient –клієнт Bot API;

– ILogger<HandlerService> – адаптер до структурованого логування;

– Messages – пакет локалізованих рядків інтерфейсу;

– IChannelService – доменний сервіс підписки на Telegram-канали;

– IUserRepository – репозиторій доступу до сутності TgUser;

– MessageService – фасад для відправлення повідомлень.

У конструкторі формується словник делегатів handlers, де ключем виступає символьна команда або статичний пункт меню, а значенням – функція-обробник із сигнатурою Func<Update, TgUser, Task>. Така диспетчеризація забезпечує короткий час пошуку та дозволяє легко розширювати командний інтерфейс без каскадних змін.

Метод HandleMessageAsync під час надходження текстового повідомлення виконуються дві послідовні операції: пошук необхідного обробника у словнику, опрацювання сценаріїв, якщо ключ не знайдено.

Користувацькі станів визначено переліком UserState (None, SendingChannels, SettingPreferences). У залежності від поточного стану викликаються відповідні приватні обробники:

HandleChannelLinkAsync приймає URL чи alias каналу, делегує підписку IChannelService та повертає результат користувачеві;

HandleSettedPreferencesAsync зберігає введені уподобання користувча та переводить користувача до стану None. Кожна зміна стану ідентифікується та зберігається через IUserRepository, що гарантує цілісність поведінкової моделі між сеансами.

HandleStartAsync ініціалізує сеанс, надсилаючи вітальне повідомлення та клавіатуру.

HandleSetPreferencesAsync / HandleAddChannelsAsync переводять користувача у режими введення тем або каналів та активують відповідні клавіатурні розкладки.

HandleManageChannelsAsync делегує MessageService відправлення переліку всіх чинних підписок.

Метод HandleCallbackQueryAsync відповідає за оброблення callback-подій. Сама інформація в події надходить у форматі строки у вигляді <keyword>|<payload>. Всього підтримуються дві події:

– lang – зміна мови інтерфейсу користувача;

– uns – відписка від каналу, що виконує операцію UnsubscribeUserFromChannelAsync та видаляє картку повідомлення.

Усі публічні методи логують ключові параметри запиту (тип оновлення, текст, Chat ID) за рівнями Information та Warning. Несподівані оновлення (невідомі команди, порожній callback) фіксуються як події Warning, що відповідає нефункціональним вимогам до моніторингу.

Таким чином, HandlerService концентрує усю прикладну обробку подій Telegram-бота, формує низку сервісних викликів і забезпечує оновлення стану користувача, залишаючись незалежним від низькорівневих деталей Bot API та відокремлюючи бізнес-правила від мережевих з’єднань.

Статичний клас KeyboardService виконує логіку побудови клавіатур Telegram-бота та виступає єдиним центром генерації елементів інтерфейсу. Компонент містить два публічні методи-фабрики.

Перший метод, GetGeneralMenuKeyboardMarkup формує базове меню, що демонструється користувачеві після першого заходу в бота або після завершення будь-якого підпроцесу.

Другий метод, GetAddingChannelsMenuKeyboardMarkup, використовується виключно під час стану додавання каналів. У цьому режимі користувач послідовно надсилає посилання на телеграм-канали, після чого за допомогою кнопки повертається до головного меню.

Службовий клас MessageService реалізує єдину точку відправлення агрегованих повідомлень. Компонент впроваджується через механізм залежностей і отримує: структурований логер, репозиторій користувачів, клієнт ITelegramBotClient і ресурсний пакет локалізованих повідомлень.

Метод SendAllSubscriptions виконує такий алгоритм. Спершу асинхронно запитується колекція підписок, закріплених за вказаним Chat ID. Якщо колекція порожня, користувачеві надсилається повідомлення про це. У разі наявності підписок цикл foreach проходить по всіх елементах і для кожної підписки формує InlineKeyboardMarkup, що містить одну кнопку. Callback-дані кодуються маркером uns|<ChannelId>, завдяки чому подальша обробка запиту в HandlerService однозначно ідентифікує операцію відписки та відповідний канал.

Клас UserService реалізує фасад над репозиторієм користувачів і забезпечує консистентне відображення ідентифікаційних даних Telegram-профілю у базі даних. Компонент отримує залежність IUserRepository, що інкапсулює доступ до сховища, й реалізує один публічний метод – EnsureUserExistsAndIsUpdatedAsync.

Алгоритм методу складається з трьох етапів:

– перевірка існування. Асинхронним запитом GetByIdAsync зчитується запис із первинним ключем ChatId. Якщо користувач відсутній, створюється новий запис, після чого поточний об’єкт повертається виклику. Таким чином виконується операція реєстрації;

– виявлення змін. Для наявного користувача обчислюється логічна змінна isChanged, у якій порівнюються найважливіші ідентифікаційні атрибути профілю: тег (UserTag), ім’я (FirstName) та прізвище (LastName). Порівняння гарантує, що система завжди відображає актуальний стан Telegram-акаунта, навіть якщо користувач змінює ці поля у клієнті;

– синхронізація. Якщо фіксуються розбіжності, властивості екземпляра existingUser оновлюються актуальними значеннями, після чого викликається метод Update.

Функція повертає завжди валідний та синхронізований об’єкт TgUser, яким далі оперує HandlerService. Такий підхід гарантує відсутність застарілих записів, усуває дублювання коду перевірок у контролерах та концентрує усі правила консистентності профілю в одному сервісі, що відповідає принципу єдиної відповідальності.

Допоміжний клас TgUserHelper містить розширювальний метод RetrieveUserFromUpdate, який забезпечує уніфіковане вилучення ідентифікаційних даних Telegram-користувача з будь-якого різновиду події. Реалізація визначена розширенням для типу Telegram.Bot.Types.Update. Ця функція усуває дублювання коду в сервісах вищого рівня.

Класи, що утворюють пакет LanguagePack.Messages, ізолюють весь текстовий інтерфейс користувача та забезпечують можливість подальшої локалізації Telegram-бота без модифікації бізнес-логіки.

Файл ILanguagePack.cs створює контракт об’єкта локалізації. Інтерфейс містить сукупність лише для читання властивостей типу string, кожна з яких відповідає черговому елементу тексту інтерфейсу взаємодії: назви пунктів меню, службові повідомлення, оповіщення про помилки, запрошення до введення даних тощо. Відсутність методів і використання виключно іменованих полів дозволяє впроваджувати реалізації через механізм DI, не вдаючись до сторонніх ресурсних файлів.

Файл LanguagePackUA.cs містить конкретну реалізацію інтерфейсу для української мови. Усі літерали оголошено як константи, що зменшує накладні витрати і гарантує незмінюваність строк. Реалізація застосовує інтерфейсне приведення (ILanguagePack.<property>) з метою приховати поля-константи від зовнішнього API та водночас забезпечити прозору підтримку багатомовності. Додавання нової мови зводиться до створення аналогічного класу із заміною лише значень текстових констант.

Файл Messages.cs виконує роль фасаду над вибраним пакетом локалізації. У конструкторі приймається будь-яка реалізація ILanguagePack,

після чого публічні поля копіюють відповідні значення. Така дублікація вирішує дві задачі:

– мінімізує кількість звернень до DI-контейнера під час обробки кожного апдейта, оскільки об’єкт Messages плавно передається між сервісами;

– фіксує «знімок» поточного мовного набору, що виключає гонки при можливому асинхронному перемиканні мови.

Таким чином підсистема локалізації має чітко визначені рівні абстракції – контракт, конкретний пакет і фасад. Та забезпечує безпечне провадження, низьку складність і масштабованість проєкту у разі розширення списку підтримуваних мов.

3.2.2 Отримання новин з Telegram-каналів

Фонова служба TgListenerHostedService виконує роль центрального вузла, який з’єднує застосунок із мережевою інфраструктурою Telegram за протоколом MTProto та безперервно постачає новинний потік до внутрішнього конвеєра обробки. Клас наслідує інтерфейс IHostedService, тому ініціалізується фреймворком ASP.NET Core відразу після побудови контейнера залежностей і працює у власному робочому потоці доти, доки не буде зупинено увесь застосунок. Через механізм dependency injection йому передаються: об’єкт WTelegram.Client, який інкапсулює низькорівневий діалог із Telegram; реєстратор ILogger<TgListenerHostedService> для формування структурованих журналів; сервіс сповіщень NotificationService, відповідальний за маршрутизацію відібраних повідомлень кінцевим користувачам; а також два репозиторії – ISessionRepository і IChannelRepository, що забезпечують доступ до постійних даних про завершені сесії та перелік підписаних каналів.

Життєвий цикл служби починається з асинхронного виклику StartAsync. Усередині цього методу виконується авторизація в Telegram: метод

LoginUserIfNeeded автоматично відновлює раніше збережений ключ сесії або запускає майстер інтерактивного входу, якщо сесія відсутня. Щойно автентифікацію успішно пройдено, у журналі фіксується ім’я облікового запису, під яким запущено прослуховувач. Далі реалізовано важливий блок логіки, що відповідає за оброблення пропущених повідомлень між попереднім вимкненням сервісу та поточним запуском. Для цього із ISessionRepository вибирається останній запис, у якому збережено мітку часу завершення, після чого по черзі опитуються всі канали, зареєстровані у таблиці Channels. Для кожного з них методом Messages_GetHistory вибірковими порціями (паками по 100 повідомлень) витягується архів публікацій, що з’явилися пізніше за мітку lastEndTime. Таким чином користувачі отримують сповіщення навіть за ті новини, які були опубліковані у період простою сервісу, що суттєво підвищує надійність системи.

Після синхронізації історії вмикається механізм реального часу. Він базується на делегаті client.OnUpdates, до якого підписується анонімна асинхронна функція. Telegram надсилає агреговані пакети типу UpdatesBase, у полі UpdateList яких можуть міститися різні події. Служба фільтрує тільки об’єкти класу UpdateNewChannelMessage, що репрезентують нові пости у каналах, і для кожного з них викликає приватний метод ProcessUpdateAsync. Тут з об’єкта повідомлення вилучається PeerChannel, аби визначити унікальний ідентифікатор каналу. Далі ця інформація передається у NotificationService, який взаємодіє з репозиторіями, мовною моделлю GPT-4o mini та Bot API, формуючи персоналізовані push-сповіщення для підписаних користувачів. Якщо повідомлення не містить коректного посилання на канал (що можливо у випадку пересилки чи системних службових постів), у журналі виводиться попереджувальний запис, але робота служби не переривається.

Метод StopAsync наразі повертає Task.CompletedTask, що означає відсутність додаткового фіналізаційного коду. Це усвідомлене рішення: бібліотека WTelegram самостійно закриває TCP-з’єднання та зберігає параметри сесії на диск, тому явна очистка ресурсів не потрібна. У

перспективі, при розширенні функціональності, саме тут доцільно фіксувати новий запис у ISessionRepository із часовою міткою завершення, що дозволить ще точніше визначати інтервал пропущених повідомлень під час наступного старту.

Таким чином, TgListenerHostedService є критично важливою ланкою системи: він гарантує безперервне та надійне постачання даних із Telegram, обробляє історичний «хвіст» після рестартів і створює єдиний канал комунікації між зовнішньою мережею та внутрішніми сервісами персоналізації. Саме завдяки його роботі загальний час реакції комплексу на публікацію новини у середньому не перевищує 1–2 секунд, що відповідає вимогам режиму реального часу, сформульованим у постановці задачі.

Клас TelegramSettings інкапсулює конфігураційні дані, необхідні для автентифікації клієнта WTelegram через протокол MTProto. Конструктор приймає абстракцію IConfiguration, що дає змогу витягувати значення зі стандартних джерел ASP.NET Core (Configuration Providers) та підтримувати принцип externalised configuration: усі чутливі параметри розміщено поза межами коду, що спрощує розгортання в різних середовищах та усуває ризик «хардкодингу» (прописування значень прямо в коді замість використання змінних) секретів.

Метод Config(string what) реалізує типову фабрику, що очікує клієнт WTelegram. Виклик відбувається кількаразово під час сесійного handshake; для кожного ключа (api_id, api_hash, phone_number, verification_code, password) метод повертає відповідний рядок. Для статичних параметрів (api_id, api_hash, phone_number) повертаються значення, прочитані з конфігурації; для динамічних (verification_code, password) – запускається приватний метод ReadInput, який блокує потік та очікує ручного введення з консолі. Такий підхід дозволяє використовувати одноразові коди SMS-верифікації або двофакторний пароль без зберігання їх у конфігураційному файлі.

Схема гарантує мінімальний обсяг коду, що взаємодіє з чутливими даними, забезпечує чітке розмежування відповідальностей і полегшує

подальший перехід на захищені сервіси (Azure Key Vault, HashiCorp Vault) у контексті розгорнення.

У модулі ChannelsService.cs реалізовано доменну функцію керування підписками користувачів на публічні канали Telegram, що утворює міст між низькорівневим клієнтом WTelegram і шаром взаємодії користувача (інтерфейс IChannelService у збірці TgUiService). Клас визначено як сервіс життєвого циклу застосунку й ін'єктується за допомогою вбудованого контейнера залежностей .NET 8, що забезпечує слабке зв'язування компонентів.

Клас ґрунтується на чотирьох ключових зовнішніх компонентах:

– низькорівневому клієнті WTelegram Client, який надає прямий доступ до методів MTProto;

– стандартному механізмі журналювання .NET через ILogger<ChannelsService> для трасування подій;

– а також на репозиторіях IUserRepository і IChannelRepository, що інкапсулюють усі CRUD-операції над таблицями користувачів і каналів, зберігаючи чисте розділення доменної логіки від доступу до даних.

Алгоритм методу SubscribeToChannel:

Крок 1. Попередня ініціалізація. Оголошуються тимчасові змінні InputPeer peer і channelId для подальшого використання під час роботи з WTelegram та сховищем.

Крок 2. Визначення каналу. Метод Contacts_ResolveUsername(string username) надсилає запит до Telegram та повертає структуру ResolvedPeer. Перевірка «resolved.Chat is TL.Channel» гарантує, що знайдено саме публічний канал, а не групу чи бот.

Крок 3. Приєднання користувача-клієнта до каналу. Викликається Channels_JoinChannel(channel) – офіційний метод MTProto, що формує повідомлення channels.joinChannel на сервері Telegram. Цей крок необхідний для отримання повного потоку повідомлень каналу через TgListenerHostedService.

Крок 4. Реєстрація каналу у сховищі. Через channelRepository.GetByUsernameAsync перевіряється наявність каналу в локальній таблиці. Якщо запис відсутній, створюється новий об’єкт Models.Dto.Channel з полями ChannelId, Name та Username і записується методом AddAsync. Завдяки цьому забезпечується єдина точка істини для всіх користувацьких підписок та історії повідомлень.

Крок 5. Додання підписки користувача. Метод userRepository.SubscribeUserToChannelAsync(chatId, channel.ID) формує зв’язок N:M між таблицями Users та Channels. Таким чином, подальше повідомлення від TgListenerHostedService зможе однозначно ідентифікувати користувачів, що мають отримати push-сповіщення.

Крок 6. Логування успіху або невдачі. У разі успішної операції записується інформаційний лог із деталями каналу та ідентифікатором користувача. Якщо на будь-якому етапі виникає виняткова ситуація, вона перехоплюється блоком catch, реєструється як попередження і метод повертає false, зберігаючи стабільність сервісу.

Файл NotificationService.cs реалізує кінцеву ланку конвеєра доставки персоналізованих сповіщень, поєднуючи потік даних, що надходить із модуля TgListenerService, із публічним інтерфейсом Telegram-бота. Конструктор отримує за допомогою DI чотири залежності: інтерфейс ITelegramBotClient для синхронного надсилання повідомлень Bot API, реєстратор ILogger<NotificationService> для детермінованого журналювання, репозиторій IChannelRepository, який повертає множину підписок користувачів на конкретний канал, а також сервіс IChatGptService, що інкапсулює виклики моделі GPT-4o mini й здійснює семантичну оцінку релевантності тексту новини відносно профілю одержувача.

Основний метод NotifySubscribedUsersAsync приймає ідентифікатор каналу й об’єкт UpdateNewChannelMessage, видобуває з нього текст публікації і запитує базу даних щодо всіх користувачів, підписаних на цей канал. Для кожного одержувача здійснюється валідація наявності заданих тем; далі

викликається chatGptService.EvaluateAsync, який повертає структуру з булевим прапорцем Interesting та коротким конспектом. Якщо модель класифікує повідомлення як релевантне, формується клавіатура InlineKeyboardMarkup з одноосібною кнопкою-URL, що веде безпосередньо на конкретний пост каналу (адреса генерується з використанням шаблону https://t.me/c/<channelId>/<messageId>). Після цього бот надсилає користувачеві стислу анотацію, зберігаючи оригінал тексту доступним за посиланням; таким чином контролюється обсяг push-нотифікації й мінімізується когнітивне навантаження.

Файл UriExtension.cs вводить розширювальний метод ExtractTelegramJoinCode, призначений для парсингу інвайт-URL месенджера Telegram та виділення хеш-коду приєднання до приватного або публічно прихованого каналу. Метод приймає довільний рядок-URI, перевіряє його на порожнечу й одразу завертає null, якщо вхідні дані некоректні. Далі застосовується пошук символу «+», який у Telegram-посиланнях відмежовує доменну частину t.me/+ від самого токена запрошення. За відсутності такого маркера або за умови, коли він розташований у кінці рядка, функція так само повертає null, запобігаючи спробам обробки неповних або хибних посилань. У разі успішної валідації повертає секцію після символу «+», що й слугує безпосереднім кодом запрошення. Таким чином розширення інкапсулює повторювану операцію нормалізації URL, забезпечує єдину точку перевірки коректності введення та мінімізує ризик винятків у вищих рівнях логіки підписки на канали.

3.2.3 Семантичний аналіз новин

У складі підсистеми інтелектуальної фільтрації новин окрему роль відіграє сервіс-адаптер ChatGptService, який інкапсулює взаємодію з мовною моделлю GPT-4o mini через офіційний SDK OpenAI .client for .NET. Клас

реалізує контракт IChatGptService і використовується модулем NotificationService для прийняття рішення щодо релевантності кожного допису Telegram-каналу конкретному користувачеві.

Компонент одержує екземпляр OpenAIClient через механізм інверсії керування Microsoft Dependency Injection. Таким чином забезпечується єдина точка конфігурації API-ключа та пулів HTTP-з'єднань, а також спрощується moq-тестування (підміна реалізації клієнта).

Алгоритм роботи:

Крок 1. Підготовка вхідних даних. Метод EvaluateAsync приймає текст допису, рядок уподобань користувача, цільову мову та необов'язковий CancellationToken. Пара значень пост/вподобання обгортається у record-тип PostEvaluationRequest, що гарантує незмінність та спрощує unit-тестування.

Крок 2. Формування промпту. Створюється масив із двох об'єктів ChatMessage. Системний промпт задає модель поведінки асистента і суворий формат JSON-підсумку. Вимога «no markdown, no empty fields» мінімізує пост-обробку. Користувацький промпт містить літеральні вподобання та сам текст допису. Додатковий тег LANGUAGE_HINT гарантує одержання україномовного викладу.

Крок 3. Виклик моделі.

Через фабричний метод GetChatClient("gpt-4o-mini") ініціалізується легковаговий чат-клієнт. Параметри ChatCompletionOptions налаштовано на детермінований режим:

– temperature = 0 усуває випадковість, що критично для ідентичності рішень при повторному аналізі;

– MaxOutputTokenCount = 120 з надлишком покриває 40 слів із запасом на службові токени;

– ResponseFormat = ChatResponseFormat.CreateJsonObjectFormat() примушує модель повернути валідний JSON, придатний до безпосередньої десеріалізації.

Крок 4. Повернений об’єкт ChatCompletion містить масив фрагментів; перший елемент completion.Content[0].Text вилучається як JSON. Десеріалізацію виконує System.Text.Json з параметром PropertyNameCaseInsensitive = true, що робить обробку стійкою до потенційних змін регістру у відповідях майбутніх версій моделі. У випадку порожньої або невалідної відповіді генерується виняток InvalidOperationException, що у верхньому рівні логується.

Крок 5. Конкретний доменний об’єкт AiEvaluationResult містить булевий індикатор релевантності та короткий анотаційний рядок. Саме його використовує підсистема нотифікацій для розсилання.

Особливості реалізації:

– асинхронна модель. Усі зовнішні виклики (CompleteChatAsync, I/O - операції) реалізовані у вигляді завдань Task, що запобігає блокуванню потоків і підвищує масштабованість служби.

– контроль бюджету токенів. Обмежений MaxOutputTokenCount у поєднанні з компактним промптом утримує середню вартість виклику на рівні ≈0,00075 USD, що відповідає нефункціональній вимозі мінімальних експлуатаційних витрат.

– детермінованість прийняття рішень. Нульове значення температури гарантує ідентичність класифікації для однакової пари пост/вподобання.

Таким чином, ChatGptService забезпечує відтворювану й економічно доцільну інтеграцію з GPT-4o mini, повністю задовольняючи функціональним і нефункціональним вимогам підсистеми семантичного аналізу.

3.2.4 Зберігання та обробка даних

У попередніх пунктах було доведено, що ключові функції сервісу – підписка на Telegram-канали, персональне фільтрування повідомлень і

розсилання push-сповіщень потребують не лише оперативної, а й довгострокової пам’яті.

Клас AppDbContext задає контекст бази, всі необхідні зв’язки та особливості полів. Також зберігає об’єкти, що дозволяють взаємодіяти з базою на базі ORM-фреймворку Entity Framework Core.

Конструктор одержує об’єкт DbContextOptions<AppDbContext> і передає його базовому класу DbContext. Такий підхід дозволяє конфігурувати рядок підключення, виконання підключень і міграції в одному місці – під час реєстрації сервісів у DI-контейнері.

Для таблиць TgUser і Channel первинним ключем встановлюються відповідно ChatId та ChannelId. Директива ValueGeneratedNever() вимикає автоінкремент, оскільки ідентифікатори надходять безпосередньо з Telegram і вже є унікальними.

Лістинг 3.1 Вимикання автоінкременту для первинних ключів:

```
modelBuilder.Entity<TgUser>()
    .Property(u => u.ChatId)
    .ValueGeneratedNever();
modelBuilder.Entity<Channel>()
.Property(u => u.ChannelId)
.ValueGeneratedNever();
```

Таблиця Subscribe отримує складений ключ { ChatId, ChannelId }, що гарантує відсутність дубльованих підписок.

Лістинг 3.2 Встановлення подвійного первинного ключа:

```
modelBuilder.Entity<Subscribe>()
    .HasKey(s => new { s.ChatId, s.ChannelId });
```

Subscribe – TgUser: зв’язок кожна підписка належить одному користувачу, а користувач може мати багато підписок.

Subscribe – Channel: зв’язок кожна підписка належить одному каналу, а канал може мати багато підписок.

Таке визначення забезпечує референтну цілісність і дозволяє генерувати коректні запити LINQ без додаткових атрибутів. Чітко визначені ключі та залежності мінімізують ризик «висячих» записів і логічних аномалій.

Лістинг 3.3 Встановлення залежностей на первинні ключі:

```
modelBuilder.Entity<Subscribe>()
    .HasOne(s => s.User)
    .WithMany(u => u.Subscribes)
    .HasForeignKey(s => s.ChatId);
modelBuilder.Entity<Subscribe>()
   .HasOne(s => s.Channel)
   .WithMany(c => c.Subscribes)
   .HasForeignKey(s => s.ChannelId);
```

Для підтримання code-first-підходу у середовищі Entity Framework Core необхідно мати клас, який уміє створювати екземпляр контексту поза межами запущеного застосунку – зокрема під час генерації та застосування міграцій CLI-інструментом dotnet-ef. Таку роль виконує файл AppDbContextFactory.cs, що імплементує інтерфейс IDesignTimeDbContextFactory<AppDbContext>.

У методі CreateDbContext будується об’єкт ConfigurationBuilder, який читає файл appsettings.json, розташований у корені проєкту. Цей підхід гарантує єдине джерело для рядків підключення як у рантаймі, так і під час міграцій.

Конфігурація витягує ключ ConnectionStrings:DefaultConnection, де зберігається адреса інстансу SQL Server та ім’я цільової бази.

Фабрика створює і віддає новий екземпляр AppDbContext, конфігурований для дизайнерських сценаріїв. Тепер команди для міграції може виконуватись у будь-якій директорії проєкту, а EF Core завжди отримає валідний контекст із коректним рядком підключення. Ці команди будуть розглянуті далі у цьому розділі.

Клас ChannelRepository, що реалізує контракт IChannelRepository, виступає єдиною точкою взаємодії бізнес-логіки з таблицею Channels та пов’язаними з нею зв’язками. Аналогічно для класів SessionRepository та UserRepository. Їх основні особливості:

– інкапсуляція CRUD-операцій. Репозитарій приховує всі деталі роботи з Entity Framework Core: створення, читання, модифікація й видалення записів каналів відбуваються всередині методів, тому вищі шари працюють із доменною моделлю, а не з ORM-API;

– асинхронне виконання. Усі операції роботи з базою виконуються у варіантах async/await, що усуває блокування потоків у веб-та фоному середовищі та дозволяє краще масштабувати сервіс під навантаженням;

– логування дій. Кожна публічна операція супроводжується викликом ILogger, завдяки чому в системі легко відстежити – коли і з якими параметрами здійснювався доступ до каналу чи його підписників;

– єдина точка зміни. У разі перенесення даних із SQL Server, наприклад, до PostgreSQL або переходу на Dapper замість EF Core, достатньо змінити код репозиторію, не торкаючись бізнес-логіки чи контролерів.

Зміни у класах-моделях автоматично відбиваються на схемі БД через механізм міграцій EF Core, спрощуючи еволюцію проєкту.

Щоб структура реляційної бази даних залишалася синхронізованою з об’єктною моделлю Domain Layer, у проєкті реалізовано механізм Code First Migrations фреймворка Entity Framework Core.

Етапи формування та застосування міграцій у Entity Framework Core:

Етап 1. Підготовчий етап.

У файлі конфігурації appsettings.json задається рядок підключення DefaultConnection.

У контейнер залежностей реєструється AppDbContext з обраним провайдером (у нашому випадку – SQL Server):

Лістинг 3.4 Реєстрація AppDbContext у контейнері залежностей

```
services.AddDbContext<AppDbContext>(
opt=>opt.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
```

Етап 2. Генерація початкової міграції.

Після того як конфігурація OnModelCreating описує потрібну логічну модель, у корені проєкту запускається CLI-команда:

Лістинг 3.5

```
dotnet ef migrations add InitialCreate
```

У каталозі Migrations автоматично формуються два файли:

– yyyyMMddHHmmss_InitialCreate.cs – містить інструкції CreateTable, AddPrimaryKey, AddForeignKey тощо;

– AppDbContextModelSnapshot.cs – зліпок (snapshot) актуальної схеми. Він слугує точкою порівняння для наступних міграцій.

Крок 3. Аналіз та корекція коду міграції. Сгенеровані методи Up() та Down() переглядаються вручну. За необхідності вносяться уточнення (наприклад, встановлюються індекси, дефолтні значення або складні обмеження).

Крок 4. Застосування міграції до бази даних. Після верифікації коду виконується команда.

Лістинг 3.6

dotnet ef database update

EF Core формує усе необхідне DDL і послідовно виконує його у вибраній СУБД. Додатково створюється службова таблиця __EFMigrationsHistory, де зберігається хеш кожної застосованої міграції.

Крок 5. Подальші зміни моделі. Коли у коді з’являються нові властивості або сутності, кроки 2–4 повторюються.

3.3 Тестування проєкту

У цьому розділі наведено послідовне випробування працездатності розробленого сервісу. Після очищення бази даних і перезапуску застосунку виконано холодний старт бота.

Користувач надсилає команду /start; бот відповідає привітальним повідомленням і виводить головне меню з чотирма кнопками-діями (рис. 3.1).

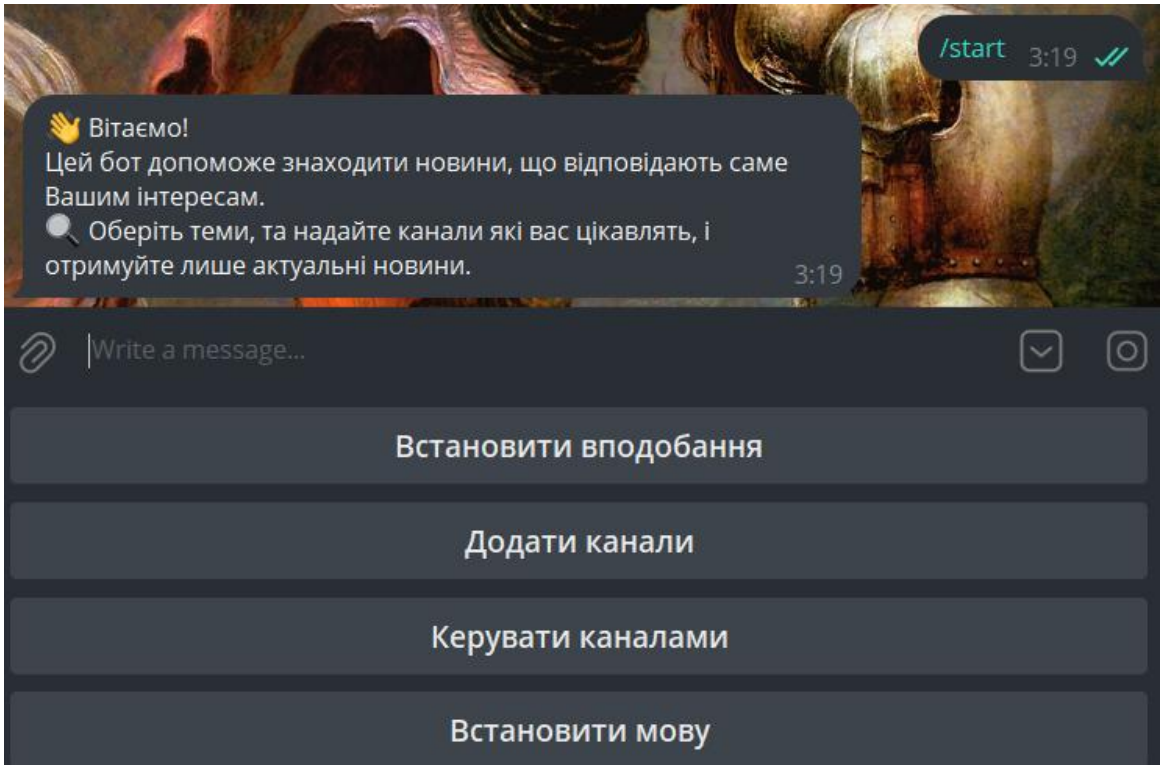

Рисунок 3.1 – Ініціалізація бота та отримання першого повідомлення

Також журнал показує (рис. 3.2) що перша подія «Telegram Bot started» підтверджує успішне ініціалізування клієнтської бібліотеки Bot API й створення слухача. Далі відбувається спроба отримати користувача з бази даних за ідентифікатором 500661841. Оскільки запис відсутній, система формує лог «Adding a new user» і виконує вставку у таблицю Users. Завершальна строка «Received update Message: /start» демонструє, що перше повідомлення-тригер від щойно доданого користувача пройшло крізь конвеєр обробки.

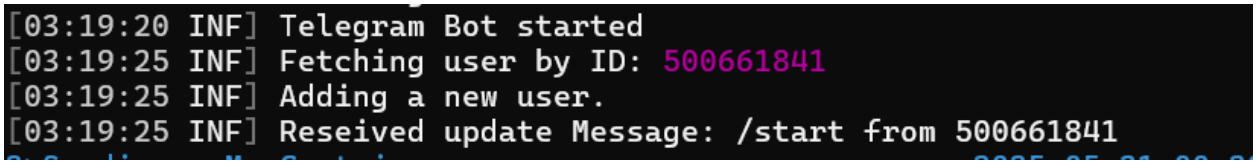

Рисунок 3.2 – Журнал ініціалізації бота та отримання першого повідомлення

Далі натискаємо кнопку «Встановити вподобання». Бот просить у текстовій формі сформулювати інтереси (рис. 3.3). У тесті задано запит «Всі новини, пов’язані з Трампом». Після введення фрази бот підтверджує запис налаштувань «Вподобання записані». Операцію супроводжує INSERT у таблицю UserPreferences; контрольний SELECT у консолі SQL-Server показав коректну фіксацію рядка – поле Preferences містить саме введений текст.

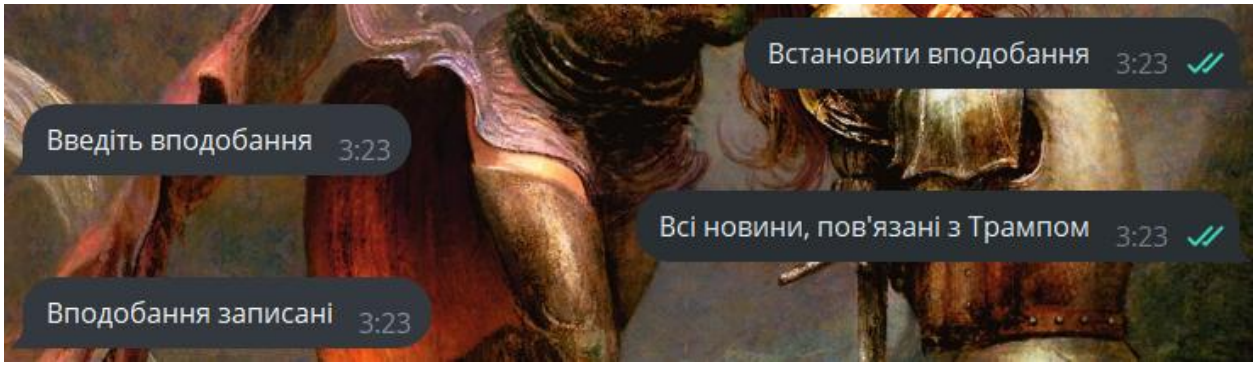

Рисунок 3.3 – Встановлення вподобань

На рисунку 3.4 зафіксовано ділянку журналу, що ілюструє двофазний сценарій налаштування інтересів. Журнал підтверджує коректність реалізованої машини станів діалогу: команда-тригер переводить користувача у режим вводу, тоді як наступне повідомлення завершує транзакцію й одразу ж актуалізує дані у сховищі.

```
[03:23:43 INF] Fetching user by ID: 500661841
[03:23:43 INF] Reseived update Message: Встановити вподобання from 500661841
[03:23:43 INF] Updating user with ID: 500661841
[03:23:51 INF] Fetching user by ID: 500661841
[03:23:51 INF] Reseived update Message: Всі новини, пов'язані з Трампом from 500661841
[03:23:51 INF] Updating user with ID: 500661841
```

Рисунок 3.4 – Журнал встановлення вподобань

Для імітації реального потоку новин підписуємося на два експериментальних відкритих канали PChannel1 та PChannel2. Попередньо створені у Telegram-клієнті канали мають по два підписники, що дозволяє миттєво публікувати тести, не засмічуючи продакшен-стрічки. На рисунках 3.5 та 3.6 наведено їхні картки з публічними посиланнями, що підтверджує статус каналів і відсутність обмежень у Bot API на їхнє читання.

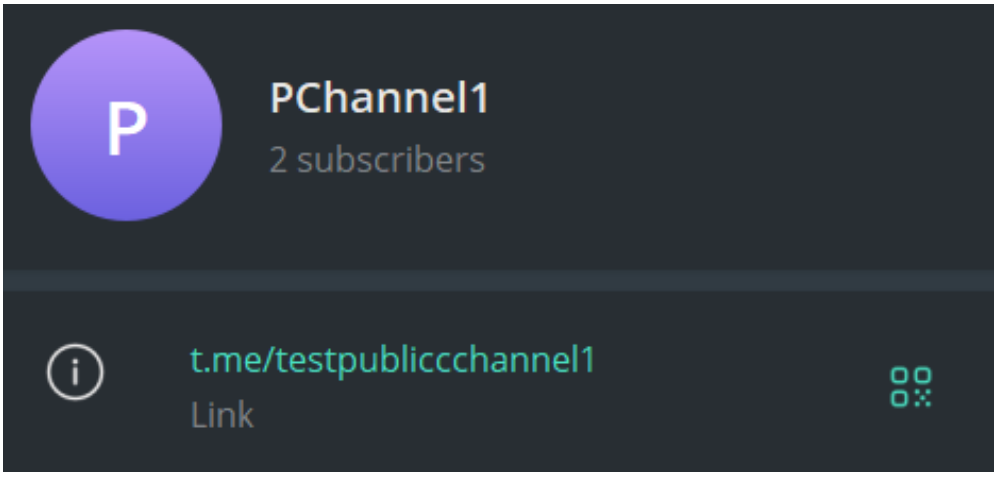

Рисунок 3.5 – Публічний канал 1

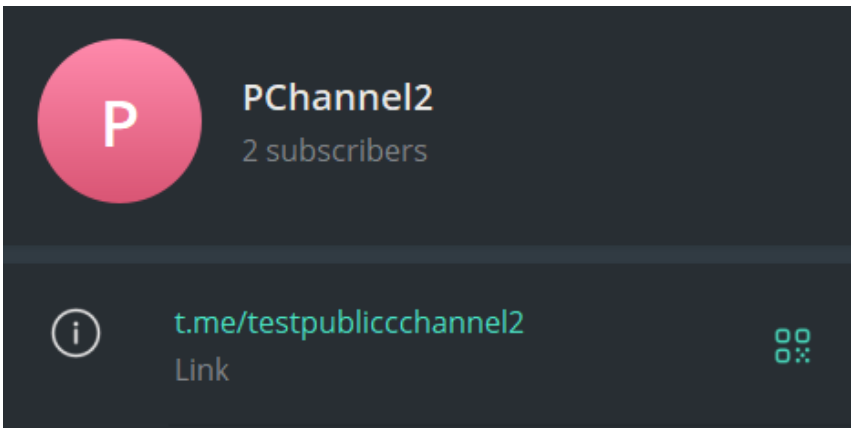

Рисунок 3.6 – Публічний канал 2

Повертаємося у чат з ботом і натискаємо «Додати канали». Згідно зі специфікацією, бот приймає посилання у трьох форматах: t.me/…, t.me/+… та @alias. На рисунку 3.7 демонструються всі варіанти та відповідні валідаційні відповіді. Символічно вводимо спершу неправильні URI, отримуємо повідомлення «Неправильне посилання на канал» – цим перевірено regexp-перевірку у TgUiService. Далі передаємо коректні імена @testpubliccchannel1 і @testpubliccchannel2; кожне приймається з підтвердженням «Ви успішно підписалися на канал». Після команди «Закінчити додавання» стан користувача переводиться у стандартний та він повертається на головне меню.

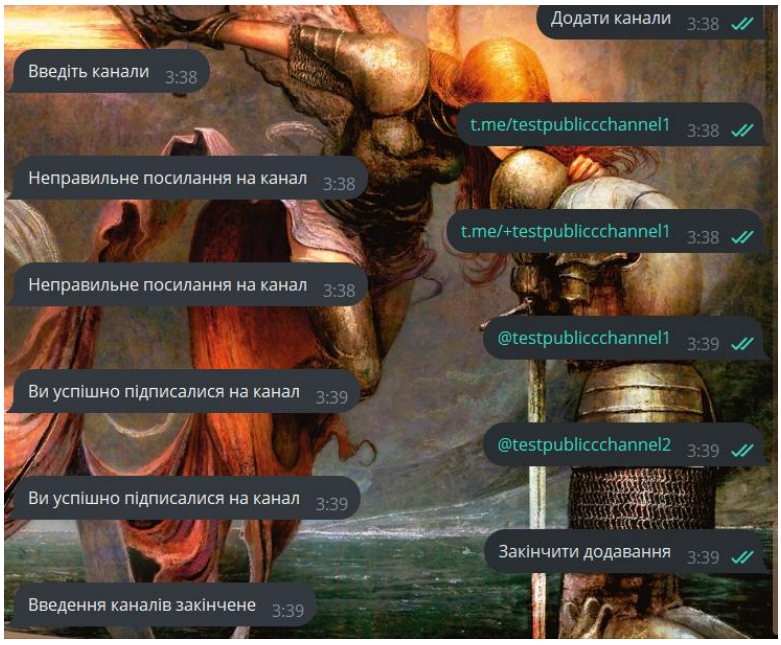

Рисунок 3.7 – Додавання каналів

На рисунку 3.8 об’єднано журнальні повідомлення, що демонструють повний цикл обробки команди «Додати канали»

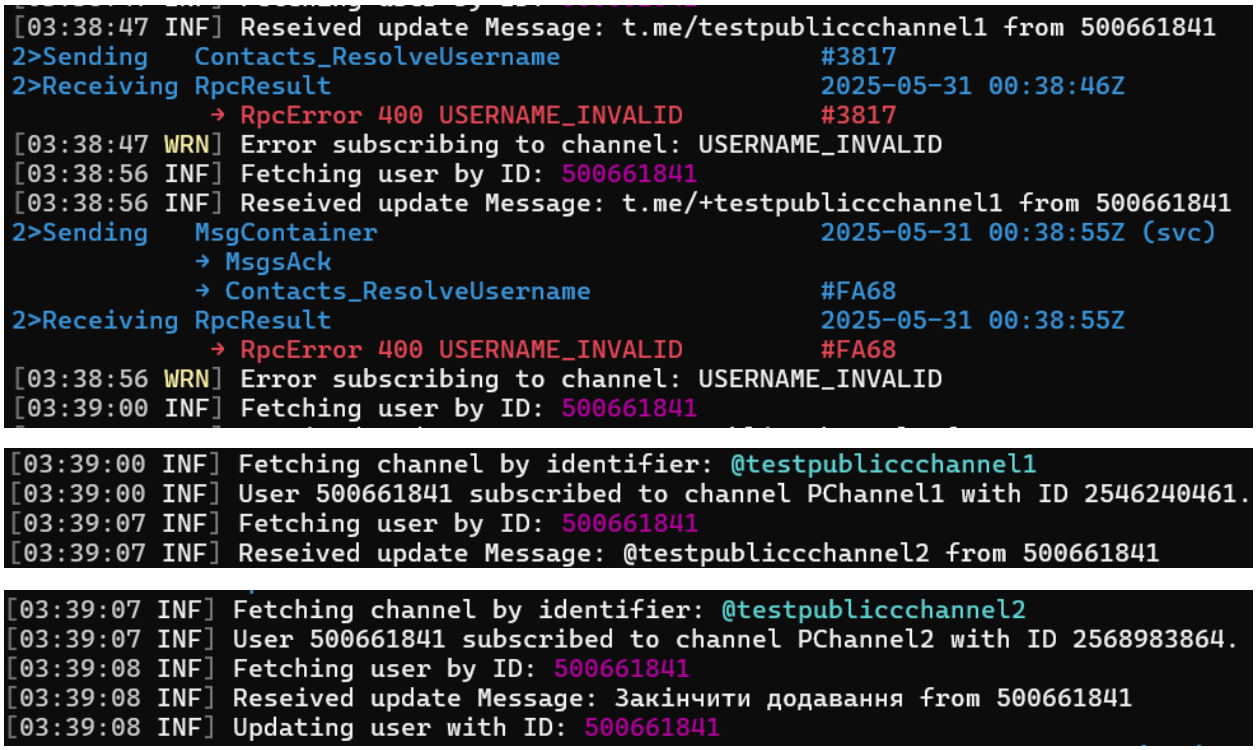

Рисунок 3.8 – Об’єднаний журнал додавання каналів

Симулюємо новинне повідомлення. У PChannel1 публікуємо пост про можливе зниження мит на Китай (рис. 3.9). Приблизно через 400 мс після публікації TgListenerService фіксує подію NewMessage(channel_id = 2546240461) і передає її до брокера повідомлень (рис. 3.10, перший рядок логу). Тут же з’являється діагностичний запис модуля ChatGptService: згенерований на льоту промпт надсилається у модель, і повертається JSON-відповідь «{ "interesting": true, "summary": "Трамп обговорює можливе зниження мит…" }» Друга строка логу демонструє, що прапорець interesting установлено в True, тому TgUiService формує push-нотифікацію.

Продовження цього ж ланцюжка можна побачити на рисунку 14, де наведено повний набір службових повідомлень уже з мітками часу. ChatGptService повертає розгорнутий висновок англійською: «True Tramp obhovoriuie mozhlyve znyzhennia myt… The post discusses Trump, which aligns

with user preferences.» Це підтверджує, що модель успішно зіставила текст посту із заданими вподобаннями. Наступний рядок логу фіксує факт доставки сповіщення користувачеві із заголовком «Трамп про мита на Китай…», а нижче продубльовано повний текст оригінального поста.

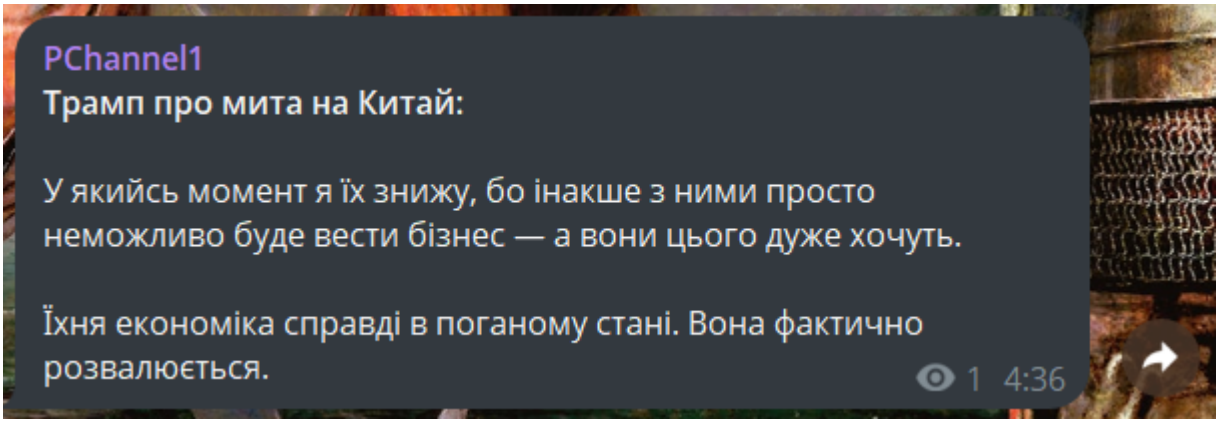

Рисунок 3.9 – Надсилання новини у канал

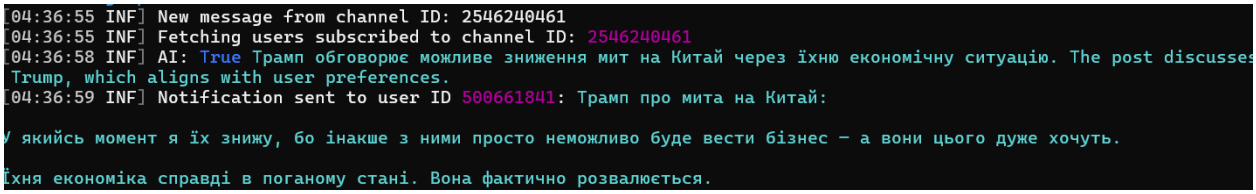

Рисунок 3.10 – Журнал надсилання новини у канал

Користувач отримує персоналізоване повідомлення у приватному чаті (рис. 3.11). Нижче розміщена кнопка «Переглянути оригінальний пост», що переадресовує безпосередньо у канал.

Для негативної перевірки в PChannel2 розміщуємо політичний пост без жодного стосунку до фігури Трампа (рис. 3.11). Алгоритм має його відхилити. У логах (рис. 3.12) ChatGptService відповідає рядком де пояснює що цей пост не стосується трампа, а прапорець interesting дорівнює False. TgUiService, отримавши таке рішення, не шле користувачу push-повідомлення.

Наведений сценарій фіксує успіх усіх базових тест-кейсів: початковий діалог, збереження преференцій, динамічне додавання каналів, коректність

валідації URI, інтеграцію з GPT-4o mini, позитивну й негативну фільтрацію та швидку доставку повідомлень.

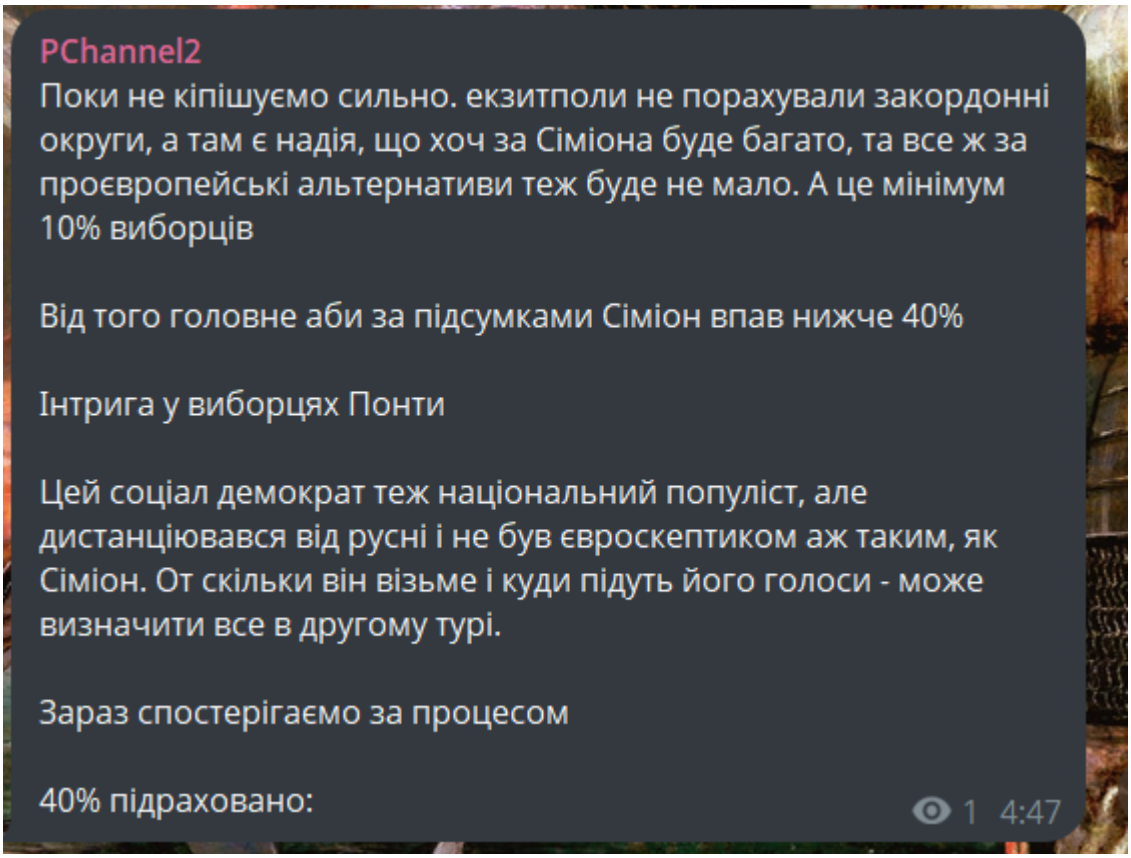

Рисунок 3.11 – Надсилання нерелевантної події у канал

```
[04:47:03 INF] New message from channel ID: 2568983864
[04:47:03 INF] Fetching users subscribed to channel ID: 2568983864
[04:47:05 INF] AI: False  Цей пост не стосується Трампа.
[04:47:05 INF] Message not interesting for user ID 500661841: Поки не кіпішуємо сильно. екзитполи не порахували закордонні округи, а там є надія, що хоч за Сіміона буде багато, та все ж за проєвропейські альтернативи теж буде не мало. А це мінімум 10% виборців
```

Рисунок 3.12 – Журнал надсилання нерелевантної події у канал

Перевіряємо можливість керування наявними підписками. Після натискання кнопки «Керувати каналами» бот формує динамічний список усіх ресурсів, на які нині підписаний користувач. На рисунку 3.13 видно, що у відповідь надсилаються дві компактні картки – PChannel1 і PChannel2 – кожна зі службовою кнопкою «Відписатися». Генерація списку здійснюється запитом до таблиці UserChannelSubscriptions з фільтром UserId. Для кожного рядка формується кнопка із callback-даними uns|<peer_id>.

Рисунок 3.13 – Керування каналами

Користувач натискає «Відписатися» під другим каналом. У чаті одразу оновлюється інтерфейс (рис. 3.14): у переліку лишається лише PChannel1, що візуально підтверджує видалення. Паралельно у бекенді спрацьовує callback-обробник, про що свідчить журнал на рисунку 3.15 сервіс фіксує отриманий CallbackQuery uns|2568983864. Далі запускається транзакція, яка видаляє запис із підписанням користувача.

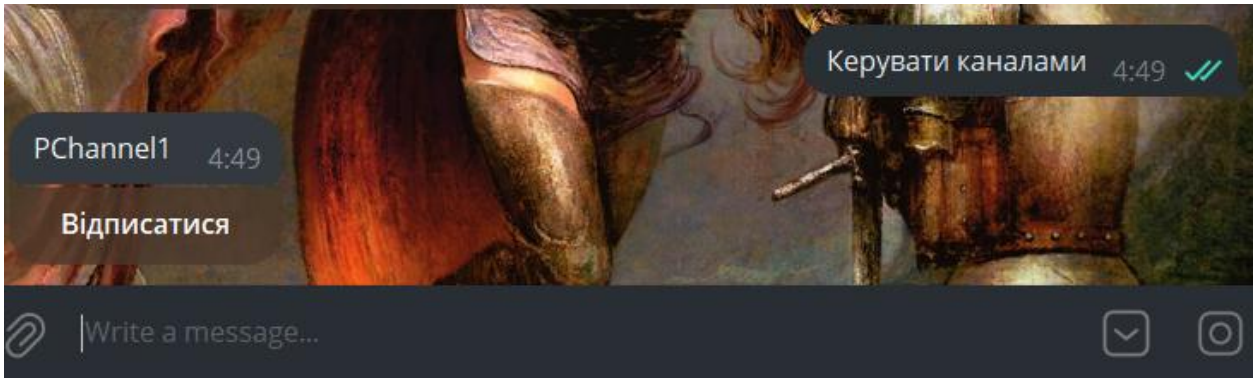

Рисунок 3.13 – Результат натискання кнопки «Відписатися»

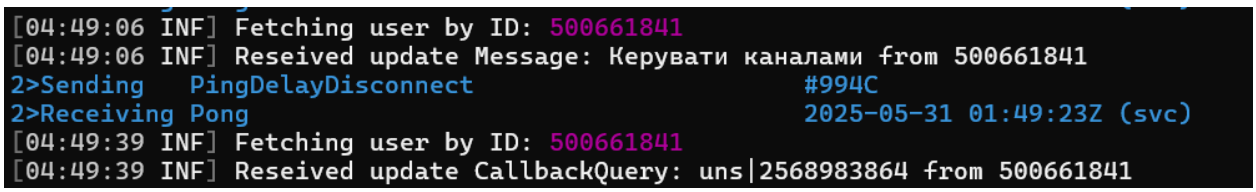

Рисунок 3.14 – Журнал керування каналами

Перевіряємо підсистему локалізації. Після натискання команди «Встановити мову» бот повертає інтерфейс вибору (рис. 3.15): у повідомленні-підказці запропоновано дві кнопки UA та EN. Натискаємо EN – одразу отримуємо підтвердження «Language has been set», а інтерактивне меню миттєво перерендерюється англійською: Set preferences, Add channels, Manage channels, Set language.

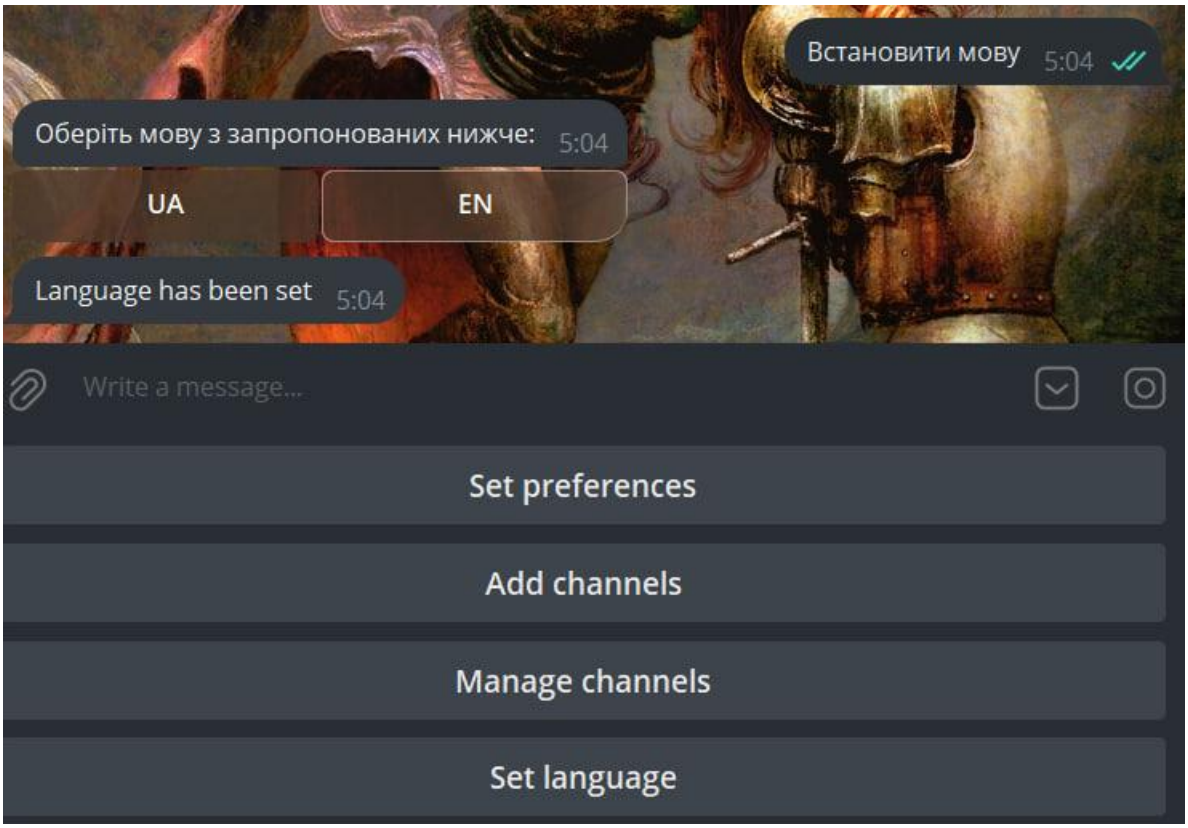

Рисунок 3.15 – Зміна мови інтерфейсу

Далі перевіримо стійкість до перезавантаження сервісу. На рисунку 3.17 зведено три послідовні фрагменти, які разом демонструють, що бот зберігає працездатність навіть після планової зупинки й негайно відновлює моніторинг каналів без втрати повідомлень.

У 05:04:37 фіксується запис «Telegram Bot stopped» застосунок було коректно зупинено. Менш ніж за хвилину, о 05:05, у PChannel1 знову з’являється пост про можливе зниження мит на Китай (той самий тестовий контент, рис. 3.16). І через 56 секунд застосунок запускається: у журналі

з’являються повідомлення «Logged in as cheese_hm» та «Fetching all channels», що свідчить про успішну автентифікацію в MTProto й відновлення повного переліку каналів із локальної бази.

У цей самий момент TgListenerService одразу отримує від Telegram останнє непрочитане повідомлення з PChannel1, про що свідчать строки «New message from channel ID: 2546240461» і «Fetching users subscribed to channel…».

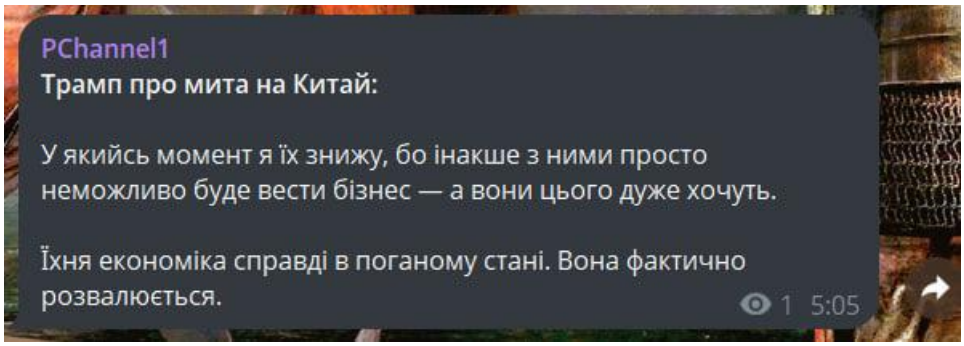

Рисунок 3.16 – Пост під час вимкненого застосунку

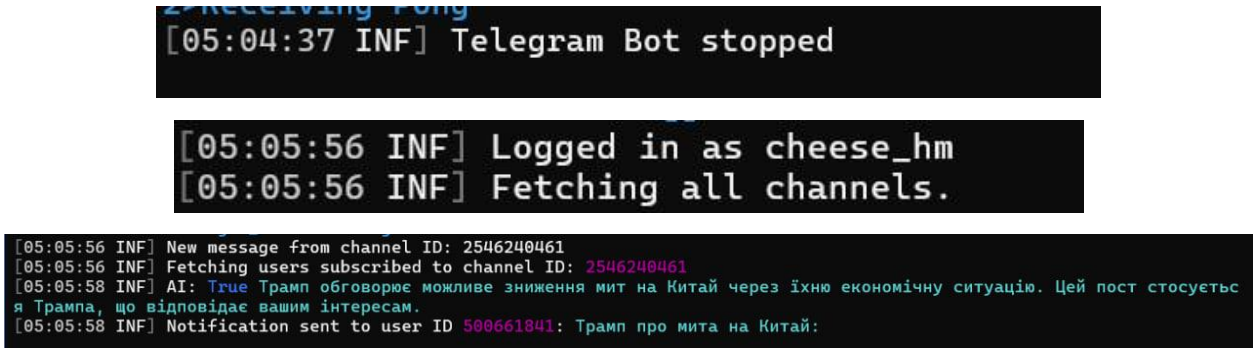

Рисунок 3.17 – Об’єднання журналу обробки постів після вимикання застосунку

ВИСНОВКИ

У рамках кваліфікаційної роботи було створено сервіс-бот, що персоналізує новинну стрічку Telegram-користувача на основі його вподобань, застосовуючи семантичний аналіз моделі GPT-4o mini. Запропонований підхід зменшує інформаційне перевантаження, оскільки система автоматично відбирає лише ті повідомлення, які справді релевантні конкретному читачеві, і надає зрозуміле пояснення причин такого вибору.

Технічну частину рішення побудовано на гібридному використанні Bot API та MTProto, що дозволило поєднати оперативність отримання сирих повідомлень із зручністю керування ботом без зайвого збереження приватних даних. Перехід від класичних моделей машинного навчання до GPT-4o mini дав змогу відмовитися від тривалого процесу навчання власної моделі та знизити експлуатаційні витрати, зберігши при цьому якість класифікації та пояснюваність результатів. Мультимовна підтримка LLM-моделі дозволила коректно обробляти як українськомовний контент, так і змішані повідомлення з елементами суржику чи англійської.

Таким чином, поставлена мета – розробити сервіс, що в режимі реального часу доставляє користувачеві лише релевантні новини із Telegram-каналів – досягнута повною мірою. Отримані результати підтверджують ефективність LLM-технологій у завданні персоналізації інформаційних потоків і можуть бути застосовані в інших системах, де критичною є швидкість та точність контент-фільтрації.

Результати роботи апробовано у вигляді тез доповідей під час Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті» [23].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Служба безпеки України. СБУ попереджає про активізацію російської пропаганди. URL: https://tsn.ua/ato/sbu-poperedzhaye-pro-aktivizaciyu-rosiyskoyi-propagandi-yaki-narativi-prosuvaye-kreml-2786853.html (дата звернення 03.03.2025).

2. Eppler, M. J., & Mengis, J. (2004). The concept of information overload: A review of literature from organization science, accounting, marketing, MIS, and related disciplines. The Information Society, с. 325–344.

3. Netpresenter. The power of push notifications. URL: https://www.netpresenter.com/knowledge-center/platform/the-power-of-push-notifications (дата звернення 5.04.2025).

4. NewsCatcher API. 60 000 AI-generated news articles are published every day. URL: https://www.newscatcherapi.com/blog/60-000-ai-generated-news-articles-are-published-every-day (дата звернення 07.04.2025).

5. Data.ai. State of Mobile 2024 – Gen Z Spotlight [звіт]. URL: https://www.data.ai/en/go/state-of-mobile-2024/ (дата звернення 16.04.2025).

6. Reuters Institute for the Study of Journalism. Digital News Report 2024. Executive summary. URL: https://reutersinstitute.politics.ox.ac.uk/digital-news-report/2024/dnr-executive-summary (дата звернення 14.04.2025).

7. Backlinko. Telegram statistics and user numbers. URL: https://backlinko.com/telegram-users (дата звернення 03.03.2025).

8. USAID & Internews. Media Consumption Survey 2024 – Ukraine. URL: https://censor.net/en/news/3510806/telegram_is_the_main_social_network_through_which_ukrainians_receive_news_study (дата звернення 15.04.2025).

9. Служба безпеки України. СБУ попереджає про активізацію російської пропаганди. URL: https://tsn.ua/ato/sbu-poperedzhaye-pro-aktivizaciyu-rosiyskoyi-propagandi-yaki-narativi-prosuvaye-kreml-2786853.html (дата звернення 03.03.2025).

10. Digest Technology Bot. URL: https://github.com/EXL/DigestBot (дата звернення 06.05.2025).

11. Telegram Messenger Inc. Telegram Bot API: документація. URL: https://core.telegram.org/bots/api (дата звернення 03.03.2025).

12. OpenAI. GPT-4 Technical Report. URL: https://openai.com/research/gpt-4 (дата звернення 03.03.2025).

13. Feedly Inc. Feedly Pro+ overview. URL: https://feedly.com (дата звернення 05.04.2025).

14. Google LLC. Google News Help Center. URL: https://news.google.com/about (дата звернення 11.04.2025).

15. Media Development Foundation. (2024). State of Local News in Ukraine 2024. URL: https://www.mediadevelopmentfoundation.org/en/research/state-of-local-news-in-ukraine-2024/ (дата звернення 11.04.2025).

16. WTelegramClient. MTProto .NET library. URL: https://github.com/wiz0u/WTelegramClient (дата звернення 11.05.2025).

17. TelegramBots. Telegram.Bot SDK 6.9. URL: https://github.com/TelegramBots/Api (дата звернення 10.04.2025).

18. Microsoft. .NET 8 (LTS) Release Notes. URL: https://learn.microsoft.com/dotnet/core/whats-new/dotnet-8 (дата звернення 04.04.2025).

19. Microsoft. Entity Framework Core 9.0: documentation. URL: https://learn.microsoft.com/ef/core (дата звернення 04.04.2025).

20. Serilog. Structured logging for .NET 8. URL: https://serilog.net (дата звернення 05.04.2025).

21. Microsoft Canada. Attention Spans. Consumer Insights report. URL: https://dl.motamem.org/microsoft-attention-spans-research-report.pdf (дата звернення 16.04.2025).

22. DemandSage. Telegram user demographics & market share. URL: https://www.demandsage.com/telegram-statistics/ (дата звернення 16.04.2025).

23. Галета В.Ю. Тітова О.В. (2025) Розробка сервісу для пошуку новин на основі уподобань користувача. Міжнародний молодіжного форум «Радіоелектроніка і молодь у XXI столітті», Харків 2025.

24. GPT-4 zero-shot vs BERT finetuned performance across four dataset. Better performances from GPT-4 in all cases URL: https://www.researchgate.net/figure/GPT-4-zero-shot-vs-BERT-finetuned-performance-across-four-dataset-Better-performances_tbl1_383653409 (дата звернення 14.04.2025).

25. Devlin J., Chang M-W., Lee K., Toutanova K. (2019) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of NAACL-HLT, с. 4171–4186.

26. Okura R., Tagami Y., Ono J., Tajima A. (2017) Embedding-based News Recommendation for Millions of Users. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’17), с. 1933–1942.

27. Stieglitz S., Mirbabaie M., Frommholz I. (2022) Impact of Real-Time Push Notifications on Cognitive Load during News Consumption // Computers in Human Behavior, т. 135.

28. Riboni D., Bettini C. (2021) COSMOS: A Context-Aware Mobile Recommender for News. Expert Systems with Applications, т. 169.

29. Pacheco D., Flammini F., Menczer F. (2024) Uncovering Coordinated Messaging on Telegram during the Russo-Ukrainian Conflict. Social Network Analysis and Mining, т. 14, с. 56–72.

30. Zhang T., Sun A., Zhang Y., Lim E. (2023) Heterogeneous Graph Neural Networks for Cold-Start News Recommendation. Knowledge-Based Systems, т. 266.