НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»
УДК _____

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

зі спеціальності 123 "Комп'ютерна інженерія"

на тему: «Система агрегації та фільтрації контенту з використанням штучного інтелекту»

Виконав (-ла):
студент (-ка) ІІ курсу, групи ІО-22мп
Святощук Ростислав Олексійович _____

Керівник:
доц. каф. ОТ, к.т.н., с.н.с,
Долголенко Олександр Миколайович _____

Консультант з нормоконтролю:
проф. каф. ОТ, д.т.н., професор
Кулаков Юрій Олексійович _____

Рецензент:
доц. каф. РТС РТФ, к.т.н., доцент
Катін Павло Юрійович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський)

Спеціальність *123. Комп'ютерна інженерія*
(код і назва)

Освітньо-професійна програма *Комп'ютерні системи та мережі*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
«_____» _____ 2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Святощуку Ростиславу Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації Система агрегації та фільтрації контенту з використанням штучного інтелекту

Науковий керівник дисертації Долголенко Олександр Миколайович, доц. каф. ОТ, к.т.н., с.н.с.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «07» листопада 2023 р. № 5168-с

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження система агрегації та фільтрації контенту з використанням великих мовних моделей (LLM)

4. Предмет дослідження агрегація та фільтрація контенту з використанням великих мовних моделей (LLM)

5. Перелік завдань, які потрібно розробити: огляд предметної області, огляд наявних технологій вирішення проблеми, розробка вимог до системи агрегації контенту, огляд та вибір засобів для розробки, опис функціоналу розробленого програмного продукту

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1.	Долголенко О.М., доц. каф. ОТ		
2.	Долголенко О.М., доц. каф. ОТ		
3.	Долголенко О.М., доц. каф. ОТ		
4.	Долголенко О.М., доц. каф. ОТ		

7. Дата видачі завдання 01.09.2023

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1.	Затвердження теми роботи	01.09.23-07.09.23	
2.	Вивчення та аналіз завдання	08.09.23-21.09.23	
3.	Робота з джерелами	22.09.23-28.09.23	
4.	Розробка архітектури та загальної структури програми	29.09.23-05.10.23	
5.	Розробка структур окремих інтерфейсів програми	06.10.23-19.10.23	
6.	Програмна реалізація	20.10.23-09.11.23	
7.	Тестування	10.11.23-13.11.23	
8.	Розробка стартап-проєкту	14.11.23-18.11.23	
9.	Оформлення пояснювальної записки	19.11.23-25.12.23	

Студент Ростислав СВЯТОЩУК _____
(підпис)

Науковий керівник дисертації Олександр ДОЛГОЛЕНКО _____
(підпис)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Система агрегації та фільтрації контенту з використанням штучного інтелекту

студентом: Святощуком Ростиславом Олексійовичем

Обсяг роботи. Робота складається із вступу та п’яти розділів. Загальний обсяг роботи: 81 аркушів основного тексту, 18 ілюстрацій, 30 таблиць. При підготовці використовувалася література з 32 різними джерелами.

Актуальність теми. Інформаційне перевантаження негативно впливає на різні аспекти життя людини: продуктивність на роботі, якість та ефективність відпочинку. В результаті цього падає задоволеність від життя та збільшується ризик розвитку психічних захворювань. Для боротьби з інформаційним перевантаженням необхідні нові інструменти для автоматизованої обробки контенту.

Мета роботи – трансформація наявних наукових здобутків у готові для використання елементи системи фільтрації та агрегації контенту.

Для досягнення даної мети було поставлено і вирішено наступні завдання:

- дослідження та аналіз предметної області та існуючих рішень;
- розробка вимог до системи агрегації контенту;
- дослідження та вибір архітектури та програмних засобів, необхідних для побудови системи агрегації контенту;
- розробка та розгортання системи агрегації та фільтрації контенту з використанням розглянутих методів.

Об’єкт дослідження – процес агрегації та фільтрації контенту.

Предмет дослідження – агрегація та фільтрація контенту з використанням великих мовних моделей (LLM)

Наукова новизна. полягає у використанні великих мовних моделей для категоризації та фільтрації контенту за гнучкими правилами, визначеними користувачем та без попереднього навчання.

Практична цінність. Отримані результати можуть використовуватись для автоматизації обробки великих об’ємів контенту.

Особистий внесок. Проведено дослідження ринку, обрано оптимальні технології для створення системи, спроектовано та розроблено систему для агрегації контенту.

Ключові слова: система агрегації контенту, велика мовна модель, фільтрація контенту, LangChain, Django.

ABSTRACT

for a master's thesis

on the topic: AI-powered Content Aggregation and Filtering System

Author: Rostyslav Sviatoshchuk

Scope of the work. The paper consists of an introduction and five chapters. Total volume of work: 81 pages of the main text, 18 illustrations, 30 tables. The literature from 32 different sources was used in the preparation.

Relevance of the topic. Relevance of the topic. Information overload negatively impacts various aspects of human life: work productivity, quality, and effectiveness of rest. As a result, life satisfaction decreases and the risk of developing mental disorders increases. New tools for automated content processing are needed to combat information overload.

Purpose of the work. The aim of the work is to transform existing scientific achievements into ready-to-use elements of a content filtering and aggregation system.

To achieve this goal, the following tasks were set and solved:

- Research and analysis of the subject area and existing solutions;
- Development of requirements for the content aggregation system;
- Research and selection of architecture and software tools necessary for building the content aggregation system;
- Development and deployment of the content aggregation and filtering system using the reviewed methods.

The object of research is the process of content aggregation and filtration.

The subject of research is content aggregation and filtration using Large Language Models (LLMs).

Scientific novelty. the utilization of Large Language Models for content categorization and filtration based on flexible rules defined by the user without prior training.

Practical value. The obtained results can be utilized for automating the processing of large volumes of content.

Personal contribution. Market research has been conducted, optimal technologies for system creation have been selected, and a system for content aggregation has been designed and developed.

Keywords: content aggregation system, Large Language Model, content filtration, LangChain, Django.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ 4
ВСТУП 5
РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ 6
1.1. Інформаційне перевантаження 6
1.1.1. Причини та наслідки інформаційного перевантаження 6
1.1.2. Причини та наслідки інформаційного перевантаження 7
1.2. Сучасний стан споживання контенту 10
1.3. Огляд систем рекомендації контенту 12
1.3.1. Традиційні системи рекомендації контенту 12
1.3.2. Традиційні системи рекомендації контенту 15
1.4. Обробка природньої мови (NLP) 16
ВИСНОВКИ ДО РОЗДІЛУ 1 19
РОЗДІЛ 2. РОБОТА З ВЕЛИКИМИ МОВНИМИ МОДЕЛЯМИ 20
2.1. Загальна інформація 20
2.2. Налаштування великих мовних моделей 22
2.3. Використання LLM 24
2.4. Просунуті методи 26
2.4.1. Zero-shot/Few-shot Prompting 27
2.4.2. Chain-of-Thought 27
2.4.3. RAG 27
ВИСНОВКИ ДО РОЗДІЛУ 2 28
РОЗДІЛ 3. РОЗРОБКА ВИМОГ ТА ВИБІР ТЕХНОЛОГІЙ 29
3.1. Загальна інформація 29

3.2. Функціональні вимоги 29
3.2.1. Користувач 29
3.2.1. Інструменти 31
3.3. Нефункціональні вимоги 32
3.4. Вибір технологій 33
3.4.1. Backend 34
3.4.2. Frontend 37
3.4.3. База даних 38
3.4.4. Засоби штучного інтелекту 39
ВИСНОВКИ ДО РОЗДІЛУ 3 43
РОЗДІЛ 4. РОЗРОБКА РІШЕННЯ 44
4.1. Підготовка 44
4.2. Проєктування 46
4.3. Макетування інтерфейсу 48
4.4. Огляд моделей 50
4.5. Інструменти 54
4.5.1. Фільтрація 56
4.5.2. Моніторинг 57
4.5.3. Підсумовування 58
4.6. Отримання даних 59
4.7. Зовнішній вигляд 60
ВИСНОВКИ ДО РОЗДІЛУ 4 62
РОЗДІЛ 5. РОЗРОБКА СТАРТАП-ПРОЄКТУ 63
5.1. Інформаційна картка стартап-проєкту 63
5.2. Технологічний аудит ідеї стартап-проєкту 66

5.3. Аналіз ринкових можливостей запуску 68
5.4. Ринкова стратегія 74
5.5. Команда для розробки стартап-проєкту 75
ВИСНОВКИ ДО РОЗДІЛУ 5 77
ЗАГАЛЬНІ ВИСНОВКИ ДО РОБОТИ 78
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 79

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

LLM	Large Language Model
API	Application programming interface
ШІ	штучний інтелект
PE	Prompt engineering
ORM	Object-Relational Mapper
HTML	HyperText Markup Language
URL	Uniform Resource Locator
RAG	Retrieval-Augmented Generation

ВСТУП

Кількість інформації, яка споживається людиною щодня, неухильно зростає. Це призводить до високої конкуренції за увагу споживача між платформами, такими як соціальні мережі, новинні та тематичні сайти, та між авторами в межах цих платформ. Соціальні мережі та месенджери стали повсякденною частиною нашого життя. Розповсюдження інформаційних технологій має позитивний вплив на загальну ефективність, але, водночас, великі об’єми інформації можуть перешкоджати концентрації, викликати помилки, продукувати технострес та негативно впливати на життя [1], [2].

Основними стресовими факторами у використанні цифрових технологій є інформаційне перевантаження та ефект постійної присутності, спричинені великою кількістю інформації та постійним підключенням до мережі інтернет. Однак дослідження показали, що фільтрування, пріоритезація та подання інформації мають позитивний вплив на сприйняття інформаційного потоку і рекомендуються для автоматизації цих процесів обробки.

Однак, крім негативних наслідків, зростання доступної кількості інформації, а також розвиток обчислювальних спроможностей відкривають нові можливості для якісної та ефективної обробки інформації.

Автоматичне агрегування та фільтрація контенту, контрольованого користувачами, може суттєво позитивно впливати на здоров'я та ефективність обробки інформації.

Водночас, існує розрив між науковими розробками, які фокусуються на окремих методиках фільтрування та агрегації, та прикладним використанням цих розробок у програмному забезпеченні для кінцевих споживачів.

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Інформаційне перевантаження

1.1.1. Причини та наслідки інформаційного перевантаження

Інформаційне перевантаження [1] — стан, що може виникати при споживанні надмірних обсягів інформації людиною. Явище інформаційного перевантаження актуальне для різних галузей, пов’язаних з обробкою великих обсягів інформації: медицина, аналітика, освіта тощо. Однак, з розвитком систем миттєвої доставки повідомлень на побутовому рівні, проблема стає актуальною для всього суспільства.

Інформаційне перевантаження часто визначають як ситуацію, в якій кількість інформації починає негативно впливати на продуктивність або ефективність прийняття рішень, призводить до гірших результатів та економічних та соціальних втрат. Це поняття тісно пов'язане з техностресом — стресом, спричиненим використанням інформаційно-комунікаційних технологій. Інформаційне перевантаження є важливим аспектом техностресу, причому основним стресовим фактором є не тільки саме перевантаження, а й знаходженням в умовах постійної доступності завдяки мережі Інтернет.

У літературі про технострес виділяють п'ять основних стресових факторів:

- техноперевантаження, викликане необхідністю працювати довше і швидше;
- техновторгнення, викликане постійною доступністю, навіть під час відпочинку;
- техноскладність, викликана складністю цифрових інструментів та відчуттям відсутності відповідних навичок;
- технонебезпека, викликана ціною помилки;
- техноневизначеність, викликана постійними оновленнями ПЗ та зміною в технологіях.

Інформаційне перевантаження у цифровому середовищі може мати серйозні негативні наслідки, а саме:

- Зниження продуктивності: велика кількість інформації ускладнює концентрацію та пріоритезацію;
- Погіршення якості прийняття рішень: перевантаження даними може перешкоджати ефективній обробці та аналізу інформації;
- Підвищення рівня стресу та втоми: Надмірна кількість інформації може спричинити стрес і втому, викликати відчуття перевантаження і вигорання, а також стримувати креативність і новаторство;
- Розриви у комунікації: Велика кількість інформації може призвести до непорозумінь та помилок підчас спілкування, оскільки важливі повідомлення можуть загубитися або бути неправильно інтерпретовані.

Для зменшення негативного впливу інформаційного перевантаження важливо впроваджувати стратегії фільтрації та організації інформації, надавати технічну підтримку та проводити тренінги з використання ПЗ, а також сприяти здоровому балансу між роботою та відпочинком, особистим життям.

Спад значень після 2019 року спричинено поступовим послабленням обмежень, пов'язаних із COVID-19, і є поверненням до звичної норми. Водночас, значення між 2020 та 2021 роками мають незначні відхилення, адже повного зняття обмежень не відбулося.

1.1.2. Причини та наслідки інформаційного перевантаження

Не існує єдиної класифікації стратегій боротьби з інформаційним перевантаженням, однак деякі дослідники групують методи за рівнями[2]:

- рівень представлення інформації,
- персональний рівень,
- рівень завдань та процесів,
- рівень проєктування організаційних процесів,
- рівень інформаційних технологій.

Ці рівні тісно між собою пов’язані, і стосуються різних аспектів запобігання: від проведення навчання, зміни організаційних процесів до впровадження інформаційних технологій та змін в представлення інформації.

На рівні представлення інформації дослідники при створенні ПЗ та дашбордів рекомендують приділяти увагу кількості та якості інформації, візуальне представлення та можливостям з індивідуалізації.

На персональному рівні рекомендується покращувати рівень володіння інформаційно-комунікаційними технологіями, чітко розмежовувати роботу, приватне життя, перерви та вільний час.

На рівні організації процесів та завдань пропонується впроваджувати системне навчання новим навичкам, технічну підтримку, застосовувати системи фільтрації та пріоритизації інформації, стандартизувати процеси.

На рівні інформаційних технологій пропонується застосовувати:

- теґи, категорії — для покращення пошуку та роботи з інформацією;
- фільтрацію — для обмеження кількості інформації за певними критеріями;
- системи прийняття рішень;
- алгоритми екстракції головної інформації — для автоматизації перетворення інформації та пришвидшення її розуміння;
- автоматизацію завдань моніторингу — для автоматичного відслідковування певних подій.

Особливу увагу на багатьох рівнях приділяють саме фільтруванню.

Фільтрування є ефективною стратегію і допомагає зменшити перевантаження, дозволяючи користувачам зосереджуватися на корисній інформації, проте має й недоліки: може призвести до того, що користувачі можуть менше взаємодіяти з новою інформацією. Важливим питанням є й прозорість алгоритмів фільтрації та розуміння того хто або що контролює інформацію, яку бачить користувач.

Приклади фільтрації включають:

- ігнорування електронних листів та сповіщень у соціальних медіа від певних людей, джерел чи на визначену тематику;
- зменшення джерел інформації;
- перегляд тільки найновіших або найбільш актуальних повідомлень[1].

Для зменшення інформаційного перевантаження важливим є ефективне управління персональною інформацією[1]. Воно передбачає активне керування власним інформаційним простором з використанням вищезгаданих стратегій впорядкування інформації, а також традиційних прийомів управління часом, організації робочого простору, делегування завдань. Їх застосування допомагає збалансувати потоки інформації та сприяє більш продуктивній роботі.

Ще одним ефективним способом боротьби з інформаційним перевантаженням є призначення певного часу для обробки нової інформації. Метод передбачає застосування пріоритизацію та розподіл інформації для споживання в менш зайнятий період роботи. Згідно пріоритетів інформацію розподіляють та оброблять:

- негайно,
- тоді, коли з'являється вільний час,
- зберігають для майбутнього використання,
- повністю ігнорують.

Практичні способи організації цього процесу включають: розміщення матеріалів у папку «Прочитати пізніше», встановлення правила завжди чекати певний час перед тим, як діяти відповідно до нової інформації. Це сприяє кращому управлінню потоком інформації та зменшує тиск, спричинений необхідністю негайної реакції.

Такі методи як фільтрація, теґи, автоматичний моніторинг, виділення головного, зміна та індивідуалізація представлення, відкладення обробки в часі є перспективні для реалізації і безпосередньо пов’язані з обробкою природньої мови.

1.2. Сучасний стан споживання контенту

Інформація в межах одного ресурсу може розподілятися за джерелами (автор, організація, сторінка тощо), категоріями тощо. Водночас, більшість популярних сервісів мають feed — динамічну стрічка контенту (рис. 1.1).

Історично стрічка контенту була хронологічною, однак із збільшенням кількості інформації, важливу роль при користуванні цифровими інструментами та додатками почали відігравати рекомендаційні системи, які визначали більш релевантний чи важливий контент для користувача.

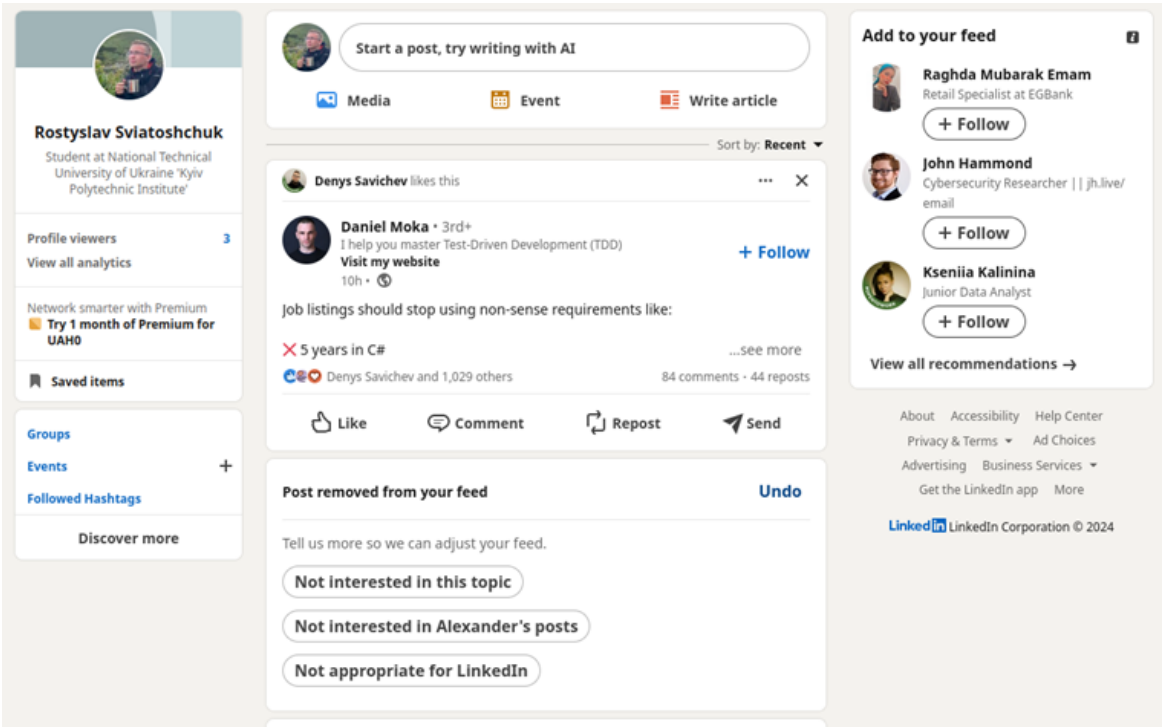

Рис. 1.1. Стрічка LinkedIn

Рекомендаційні системи стали потужним інструментом та основним функціоналом різних продуктових рішень, таких як Spotify, YouTube Music, Netflix тощо.

Також, технологія використовується в сфері e-commerce (рис. 1.2). Coursera та Udemy допомагають підбирати курси та навчальні матеріали на основі пройдених курсів, інтересів користувача тощо.

Рис. 1.2. Інтернет магазин

У вищезгаданих прикладах рекомендаційні системи є доданою вартістю, а основні послуги оплачуються за рахунок підписки прямо чи опосередковано (функціонал сприяє залученню користувачів до використання платних функцій тощо). Це взаємовигідні для користувача та власника продукту умови.

Водночас, такі сервіси як соціальні мережі мають іншу механіку роботи. Основна частина їх заробітку — реклама, тобто користувачі сплачують за користування часом та увагою. Це стає основною метрикою, яка впливає на заробіток власників, тому з’являються динамічні стрічки, вміст яких наповнюється не лише рекламою, а й контентом, який сприяє залученості користувачів. Такі стрічки експлуатують особливості роботи нашого мозку, який розвивався в умовах дикої природи та повинен був реагувати на всю доступну

інформацію для того, щоб вижити. Часто вони можуть сприяти популяризації більш сенсаційного або фейкового контенту, оскільки такі матеріали привертають більше користувачів, що може застосовуватись для маніпуляцій користувачами та зменшувати користь від витраченого на платформі часу.

Іншим критичним аспектом цих систем є їхня складність і непрозорість. Алгоритми, які керують рекомендаціями щодо контенту, часто базуються на складних моделях машинного навчання. Цей "чорний ящик" ускладнює розуміння того, як саме користувачам рекомендується той чи інший контент.

Сервіс-орієнтований підхід також часто призводить до відсутності контролю з боку користувача. Попри те, що ці системи розроблені з урахуванням уподобань користувачів, вони, як правило, не дозволяють їм розуміти або впливати на основні механізми, що лежать в основі цих рекомендацій.

І, на жаль, в більшості випадків немає жодних альтернатив відмовитись від використання стрічок або змінити їх поведінку, навіть оплативши відповідні послуги. Водночас, платформи оберігають своє монопольне становище, обмежуючи можливості до створення альтернативних клієнтських додатків [3]. Кількість користувачів та спільнот, що вже сформувалися та присутні на платформі, дозволяють компаніям тримати як творців, так і споживачів контенту в «заручниках».

Також, збільшення попиту на згенерований користувачами контент - дані для навчання великих мовних моделей, сприяє обмеженню сторонніх користувацьких додатків, оскільки API часто використовується для безкоштовного отримання цих даних. Водночас дані продаються [4]

1.3. Огляд систем рекомендації контенту

1.3.1. Традиційні системи рекомендації контенту

Рекомендаційні системи є корисною технологією, яка може зменшити проблему перевантаження інформацією. Вона дозволяє прогнозувати ступінь

важливості інформації, яка буде рекомендована користувачеві, в тому числі на основі його активності.

Традиційні рекомендаційні системи поділяють на три види [5]:

- content-based (на основі контенту),
- colaborative-filtering (на основі колаборативної фільтрації)
- гібридні.

1.3.1.1 Традиційні системи рекомендації контенту

Фільтрація на основі вмісту (content-based filtering) являє собою метод, який рекомендує елементи на основі їхньої схожості з тими, які користувач вподобав раніше [5].

Однак обмеженням систем такого виду є нездатність рекомендувати нові елементи, не пов'язані з минулим вибором користувача. Це обмежує різноманіття контенту, пропонованого користувачу.

Незважаючи на недолік, такий метод знайшов застосування в різних сферах, включаючи рекомендацію музики, фільмів, для задач електронної комерції. Системи такого типу реалізуються на основі алгоритмів інтелектуального аналізу тексту, семантичного аналізу, TF-IDF, нейронних мереж, наївний байєсів класифікатор, SVM.

1.3.1.2 Системи на основі колаборативної фільтрації

Колаборативна фільтрація [5] стала фундаментальною моделлю для побудови рекомендаційних систем. Вона створює базу даних уподобань користувачів на основі їхніх оцінок, і на основі цих даних прогнозує та рекомендує товари, які відповідають їхнім смакам.

Розрізнюють системи спільної фільтрації на основі пам'яті та фільтрації на основі моделей. Спільна фільтрація на основі пам'яті поділяється на спільну фільтрацію на основі користувачів та спільну фільтрацію на основі об’єктів.

У методах заснованих на моделях, модель навчається реконструювати значення взаємодій між користувачем та об'єктом на основі власного

представлення користувачів та об'єктів. Далі на основі цієї моделі можна робити нові рекомендації. Однак репрезентації користувачів та об'єктів, отримані моделлю, складно інтерпретувати людині.

У методах на основі пам'яті, не передбачається ніякої моделі. Алгоритми безпосередньо працюють з взаємодією користувача з об'єктом: наприклад, користувачі представляються через їхню взаємодію з об'єктами, а пошук найближчих сусідів у цих представленнях використовується для створення пропозицій

Фільтрація на основі користувачів базується на пошуку схожих користувачів з точки зору взаємодії з предметами. Як правило, кожен користувач взаємодіяв лише з кількома товарами, і це робить метод досить чутливим до будь-яких зафіксованих взаємодій. З іншого боку ми отримуємо більш персоналізовані результати, оскільки остаточна рекомендація ґрунтується лише на взаємодіях, зафіксованих для користувачів, схожих на нашого користувача, що нас цікавить.

Фільтрація на основі об’єктів, навпаки, ґрунтується на пошуку схожих елементів з точки зору взаємодій між користувачами. Як правило, багато користувачів взаємодіяли з об'єктом, тому пошук по сусідству набагато менш чутливий до поодиноких взаємодій. З іншого боку, в рекомендаціях враховуються взаємодії від усіх типів користувачів (в тому числі тих, що дуже відрізняються від нашого користувача), що робить метод менш персоналізованим. Таким чином, цей підхід є менш персоналізованим, але більш надійним.

Незважаючи на те, що колаборативна фільтрація використовується частіше, ніж фільтрація на основі контенту, вона стикається з проблемами масштабованості та розрідженості, які впливають на точність рекомендацій. Щоб подолати ці обмеження, дедалі частіше використовують гібридні системи, які поєднують спільну фільтрацію та фільтрацію на основі контенту.

1.3.1.3 Гібридні рекомендаційні системи

Колаборативна фільтрація стала фундаментальною моделлю для побудови рекомендаційних систем. Вона створює базу даних уподобань користувачів на основі їхніх оцінок, і на основі цих даних прогнозує та рекомендує товари, які відповідають їхнім смак. Гібридна модель рекомендацій виникла з метою усунення обмежень моделей фільтрації на основі контенту та спільної фільтрації, а також для підвищення ефективності рекомендацій. Класифікується на сім типів в залежності від методі ітеграції: Weighted, Switching, Cascade, Mixed, Feature Combination, Feature Augmentation, Meta-level Hybridization [5].

Наприклад, Mixed Hybridization об'єднує дані історії користувача, щоб рекомендувати елементи без потреби в оцінках користувачів, тоді як Feature Combination та Feature Augmentation інтегрують різні типи даних і моделі рекомендацій. Meta-Level використовує всю модель однієї системи рекомендацій як вхідні дані для іншої, стискаючи вподобання користувача для полегшення обробки.

Основна мета гібридної моделі полягає у вирішенні проблеми розрідженості - відсутності достатньої кількості даних про відгуки користувачів.

1.3.2. Традиційні системи рекомендації контенту

Згідно принципів роботи розглянутих вище можна побачити, що традиційні рекомендаційні системи під час своєї роботи спираються на дані та вподобання користувачів, однак працюються в доволі вузькій тематиці сервісу.

Стрімкий розвиток великих мовних моделей, що розпочався після виходу ChatGPT від компанії OpenAI, зробив можливим використовувати знання, отримані LLM в процесі навчання [6].

Ці переднавчені генеративні моделі відкривають нові можливості для рекомендаційних систем. Наприклад, вони допомагають зробити рекомендації більш персоналізованими та індивідуалізованими, інтерфейс взаємодії з системо більш природнім та ефективним. Основна суть генеративних моделей у тому, що вони вміють використовувати дані, на яких навчалися, для різних завдань.

Два основних способи застосування у рекомендаційних системах [6]:

- Моделі, що навчаються безпосередньо (Directly trained models) — де модель навчається на даних взаємодії. Це дозволяє використовувати менші та менш різноманті набори даних для навчання;
- Переднавчені моделі (pretrained models) — використовує готові переднавчені на дуже різноманітних даних моделі, часто в різних модальностях (текст, зображення, відео, аудіо).

Якщо перший спосіб потребує даних користувача та в результаті , то другий спосіб дозволяє досягати результатів використовуючи такі властивості великих мовних моделей як навчання в контексті, розуміння патернів, zero-shot/few-shot Prompting тощо.

1.4. Обробка природньої мови (NLP)

Велика кількість контенту, яку споживає людина представляє собою текст або може бути перетворена в текст (аудіо, частково відео). Вона слабко структурована, що ускладнює автоматизацію традиційними методами.

Обробка природної мови (NLP) є одним з напрямків у галузі штучного інтелекту, який охоплює ряд складних завдань: машинний переклад, системи відповідей на запитання, узагальнення текстів тощо. Робота в галузі NLP зосереджується на розробці та впровадженні моделей, систем і алгоритмів, які спрямовані на розв'язання практичних проблем розуміння людської мови.[7]

Обробку природної мови (NLP) можна поділити на дві основні підгалузі: фундаментальні та прикладні дослідження. У рамках першої категорії розглядаються загальні проблеми, що становлять основу для створення складних систем на базі людської мови. До таких завдань відносять моделювання мови, морфологічний аналіз, синтаксичну обробку або парсинг та семантичний аналіз. Окрім того, NLP охоплює прикладні аспекти, такі як автоматичне витягування релевантної інформації (наприклад, іменованих сутностей та їх взаємозв'язків) із

текстів, переклад текстів між мовами, узагальнення документів, автоматичне відповідання на запитання, класифікація та кластеризація документів.

Обробка природної мови (NLP) також традиційно поділяється на дві підгалузі: розуміння природної мови (NLU), яке фокусується на семантичному аналізі та визначенні призначеного значення тексту, та генерація природної мови (NLG), яка зосереджена на створенні тексту машиною. NLP є окремою дисципліною, але часто використовується спільно з технологією розпізнавання мовлення, яка прагне перетворювати усне мовлення на слова, перекладаючи звук у текст і навпаки.[7].

До основних прикладних завдань належать:

- Sentiment analysis: Класифікація емоційного наміру тексту, вираженого як позитивний, негативний або нейтральний.
- Toxicity classification: Відгалуження аналізу емоцій, яке виявляє ворожий намір, загрози, образи та ненависть.
- Machine translation: Автоматизований переклад тексту між різними мовами.
- Named entity recognition: Виділення елементів тексту, як-то імен, організацій, місць та кількостей.
- Spam detection: Класифікація електронних листів як спаму або неспаму.
- Grammatical error correction: Корекція тексту за граматичними правилами
- Topic modeling: Виявлення абстрактних тем у колекції документів
- Text generation: Створення тексту, подібного до людського письма, у різних жанрах та форматах
- Autocomplete: Передбачення наступного слова у реченні.
- Chatbots: Автоматизація однієї сторони розмови у діалозі.
- Information retrieval: Знаходження найбільш релевантних документів за запитом.
- Summarization: Скорочення тексту, щоб виділити найважливішу інформацію.

- Question answering: Відповіді на запитання, поставлені людиною, в природній мові.

Для наших задач будуть необхідні класифікація тексту (за довільними категоріями), виділення іменованих сутностей, теґування, topic modeling, summarization. Існує велика кількіть методів, що використовуються для вирішення цих задач, однак найбільш універсальним та інтуітивно зрозумілим для користувачів буде використання великих мовних моделей (LLM)..

ВИСНОВКИ ДО РОЗДІЛУ 1

В першому розділі було розглянуто проблематику інформаційного перевантаження, а саме причини його виникнення та наслідки впливу. Також був виконаний аналіз методів боротьби з інформаційним перевантаження, за результатами якого відзначено перспективні для реалізації в системі підходи.

Також було розглянуто рекомендаційні системи, як основний засіб управління контентом на сучасних платформах типу Facebook, Twitter. Виконано огляд різних типів рекомендаційних систем.

Оскільки більшість контенту, яку споживає людина, представлена в текстовому форматі, було проведено розглянуто основні завдання, які вирішуються в межах обробки природньої мови (NLP).

РОЗДІЛ 2
РОБОТА З ВЕЛИКИМИ МОВНИМИ МОДЕЛЯМИ

2.1. Загальна інформація

Велика мовна модель (Large Language Model, LLM) - це різновид систем штучного інтелекту, які використовують глибоке навчання та нейронні мережі для обробки та розуміння природної мови. Ці моделі є "великими" через їхній розмір і обсяги навчальних даних, яких вони потребують. Зазвичай вони занадто великі, щоб працювати на одній машині, і доступ до них здійснюється через веб-інтерфейс або API. [8]

LLM універсальні і можуть впоратися з різноманітними текстовими завданнями, на відміну від моделей, навчених для конкретних завдань. Вони навчаються з різних джерел даних, таких як книги, документи та веб-сторінки, що дозволяє їм розуміти і генерувати текст різними мовами, причому текст часто неможливо відрізнити від написаного людиною.

Великі мовні моделі стали особливо відомими після виходу моделі та продукту ChatGPT (рис. 2.1) від компанії OpenAI.

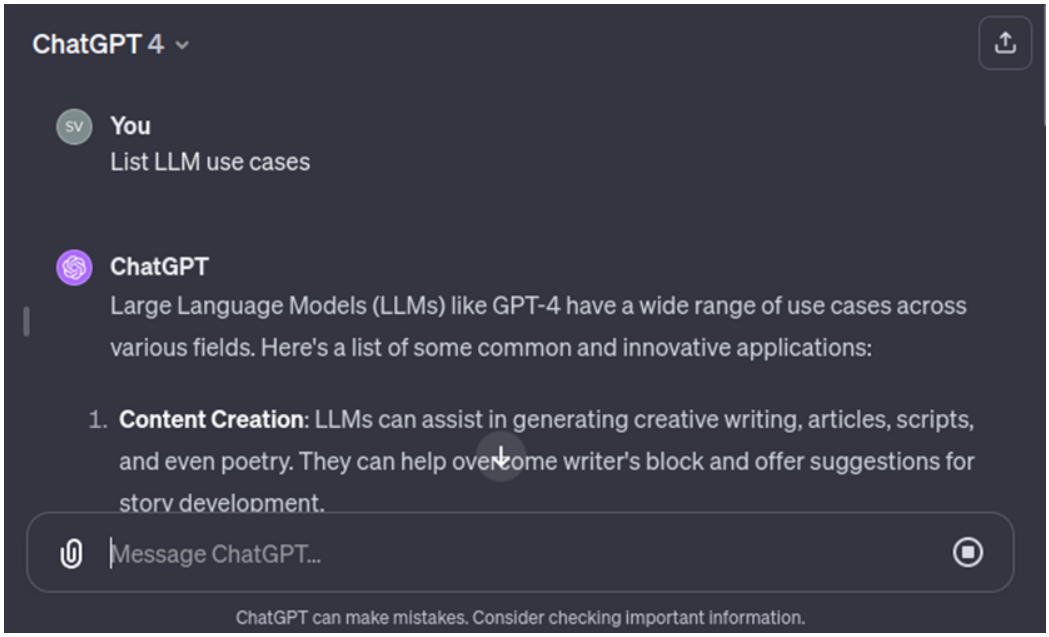

Рис. 2.1. Інтерфейс ChatGPT

Якість нової моделі дозволила користувачам виконувати нескладні побутові завдання: генерувати статті, відповідати на запитання і навіть писати програмний код.

Основою роботи сучасних LLM є трансформери [8], що складаються з двох частин (див. рис. 2.2): кодера, який обробляє вхідні дані, та декодера, який використовує цю інформацію для створення виходу. Моделі трансформерів спочатку навчаються на обширних текстових наборах даних, а потім проходять тонке налаштування для конкретних завдань, що дозволяє їм розширювати своє розуміння мови.

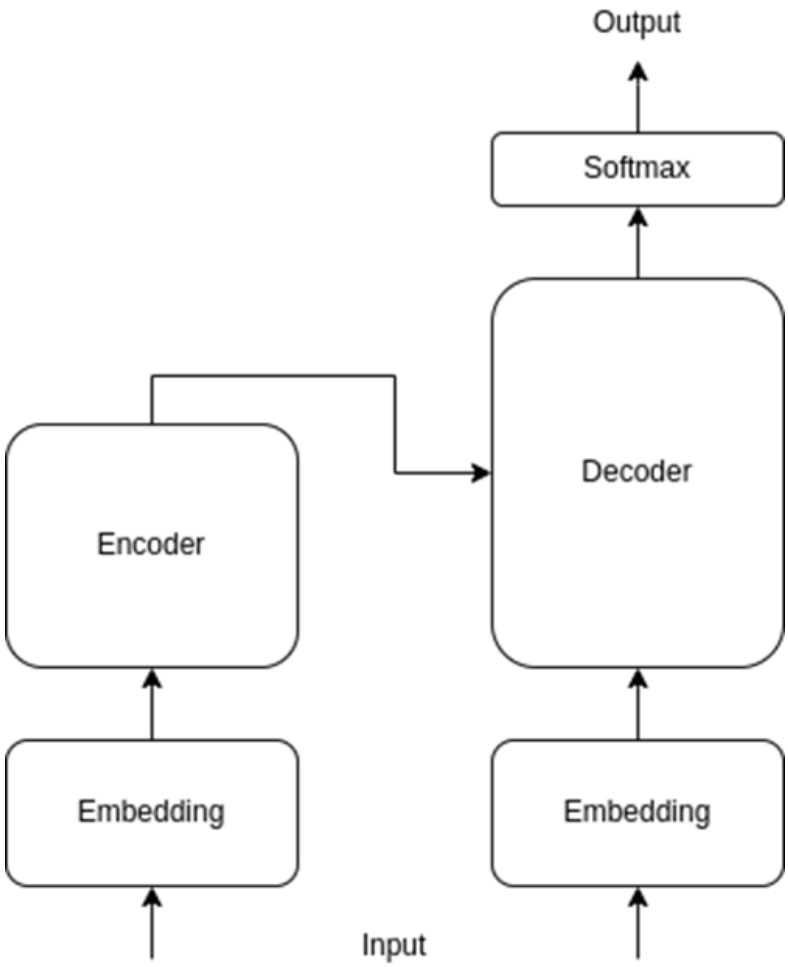

Рис. 2.2. Будова архітектури трансформера

Ключова особливість трансформерів - механізм уваги, що допомагає моделі зосереджуватись на важливих частинах тексту, розуміти контекст і

значення. Вони здатні паралельно обробляти великі обсяги тексту, що значно покращує ефективність навчання на великих масивах даних.

Навчальні дані для великих мовних моделей (LLM) включають широкий спектр текстових матеріалів, таких як книги та статті, що представляють собою різні форми письмової людської мови. У процесі навчання LLM вчаться передбачати наступне слово в послідовності тексту, що покращує їх здатність до генерації відповідей, які є релевантними в контексті запиту.

Використання великих мовних моделей (LLM) охоплює широкий спектр застосувань. Вони покращують пошукові системи, такі як Google та Bing, роблячи результати пошуку більш точними та відповідними. У сфері генерації контенту LLM використовуються для створення маркетингових матеріалів, статей та навіть для креативних завдань, як-от створення історій.

2.2. Налаштування великих мовних моделей

Взаємодія з великими мовними моделями часто відбувається з використанням API та, крім власне запиту, дозволяє керувати іншими параметрами, які впливають на відповідь моделі. Розуміння впливу значень цих параметрів на роботу моделі дозволяє підвищити надійність роботи та якість відповідей. В таблиці 2.1 представлені поширені параметри [9], з якими можна зіткнутися при використанні різних великих мовних моделей.

Таблиця 2.1

Параметри конфігурації великих мовних моделей

Параметр	Опис
Temperature	Контролює рівень випадковості в процесі генерації тексту. Вища температура означає більшу різноманітність і непередбачуваність, а нижча - більшу узгодженість і послідовність.

Таблиця 2.1(продовження)

Параметри конфігурації великих мовних моделей

Параметр	Опис
Top-P	Контролює кількість слів, які розглядаються як можливі кандидати на наступне слово в процесі генерації тексту. Вище значення — більше різноманітності та креативності, нижче — більше точності та надійності.
Max Length	Контролює довжину згенерованого тексту. Більша кількість токенів забезпечує більше деталей та інформації, тоді як менша кількість токенів відповідає більшій лаконічності та простоті.
Stop Sequences	Спеціальні рядки, які змушують модель припинити генерацію токенів. Застосовуються для контролю довжини та структури відповіді.
Frequency Penalty	Штраф застосовується до токенів на основі їхньої повторюваності у відповіді та інструкції. Чим частіше з'являється токен, тим вищий штраф, що зменшує ймовірність повторення. Цей механізм ефективний для мінімізації повторення слів у відповідях.
Presence Penalty	Контролює, наскільки згенерований текст відображає присутність певних слів або фраз у вихідному тексті. Накладається на повторні токени, незалежно від їхньої частоти. Всі повторювані лексеми отримують однаковий штраф, незалежно від того, чи з'являються вони двічі або десять разів. Вищі значення запобігають надмірному повторенню фраз у моделі. Більший штраф за присутність дозволяє генерувати різноманітний або творчий текст, тоді як менший штраф сприяє збереженню фокусу у відповідях моделі.

2.3. Використання LLM

Ефективність результатів, отриманих за допомогою великих мовних моделей, таких як GPT-3.5-turbo або GPT-4, значною мірою залежить від якості та деталізації наданих підказок. Добре продумані підказки повинні містити чіткі інструкції або запитання, а також відповідний контекст, вхідні дані або приклади. Структура запиту [10] може містити такі складові елементи:

- Системне повідомлення: повідомлення, в якому вказується роль або описується поведінка моделі, наприклад чатбот, асистент.
- Вказівка: описує конкретне завдання або вказівку, котру модель повинна виконати. Наприклад, яку дію виконати або яку інформацію надати.
- Контекст: додаткова інформація або відомості, які можуть допомогти моделі отримати більш точні та релевантні відповіді. Контекст відіграє вирішальну роль у формуванні розуміння моделлю запиту.
- Формат відповіді: вказує на бажаний тип або формат результату. Він допомагає моделі зрозуміти, як структурувати відповідь
- Приклад: застосовується для підвищення якості відповідей

Не всі елементи є обов’язковими. Структура запиту повинна бути адаптована для забезпечення оптимальної роботи моделі та отримання бажаного результату (рисунок 2.3).

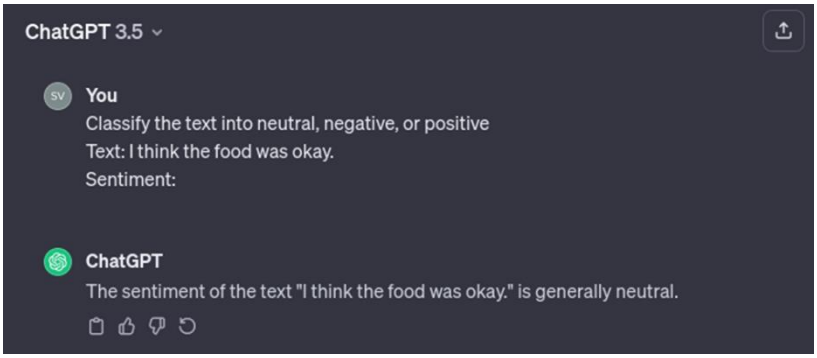

Рис. 2.3. Приклад запиту на класифікацію

В таблиці 2.2 зібрані загальні рекомендації від компанії OpenAI [11], які допомагають покращити якість відповідей.

Таблиця 2.2

Загальні рекомендації, при роботі з LLM

Стратегія	Опис	Тактики
Чіткі інструкції	Передбачає чітке формулювання потреб, уникнення двозначностей	▪ Додавання деталей до запиту ▪ Призначення певної ролі та вказування особливостей ба відповідей ▪ Використання роздільники для відокремлення частин запиту ▪ Надання покрокових інструкцій ▪ Вказування очікуваної довжини відповіді
Довідкова інформація	Надання довідкового матеріалу може допомогти моделі надавати більш точні та обґрунтовані відповіді.	▪ Надання вказівок відповідати на основі довідкової інформації ▪ Давати вказівки відповідати з посиланням на довідкову інформацію
Декомпозиція завдання	Розбиття комплексних завдань на складові для підвищення	▪ Попередньо класифікуйте завдання для застосування найбільш ефективних інструкцій

Таблиця 2.2 (продовження)

Загальні рекомендації, при роботі з LLM

Стратегія	Опис	Тактики
Декомпозиція завдання	точності та керованості.	▪ Підсумовувати та фільтрувати попередню комунікацію для ▪ Підсумовуйте великі документи частинами та рекурсивно
Час на роздуми	Уточнення та розлогі відповіді	▪ Давати вказівки використовувати внутрішній монолог та пояснення перед тим як надавати остаточну відповідь ▪ Надавати уточнювальні запити щодо попередніх відповідей моделі
Зовнішні інструменти	Використання додаткових інструментів для розширення можливостей моделі.	▪ Використовувати зовнішні джерела інформації (embedding-based search тощо) ▪ виконувати згенерований код для отримання точних результатів, використовувати зовнішні API ▪ надавати моделям доступ до виклику спеціальних функцій

2.4. Просунуті методи

Сучасні LLM, такі як GPT-3, орієнтовані на виконання інструкцій і навчені на великих обсягах даних; тому вони здатні виконувати деякі завдання одразу, "з нуля", але їм часто важко виконувати більш складні завдання.

2.4.1. Zero-shot/Few-shot Prompting

Для покращення їхньої ефективності ми можемо використати техніку, яка називається підказкою з кількома прикладами [12]. Вона полягає в тому, що ми даємо моделі кілька прикладів або демонстрацій прямо в самій підказці. Ці приклади виступають орієнтиром, допомагаючи моделі зрозуміти і краще виконати наступні подібні завдання. По суті, за допомогою цих початкових прикладів ми готуємо модель до того, щоб вона могла генерувати кращі відповіді для наступних завдань.

2.4.2. Chain-of-Thought

Підказка "Ланцюжок думок" [13] дає змогу здійснювати складні міркування за допомогою проміжних етапів. Ви можете комбінувати її з підказками з кількома пострілами, щоб досягти кращих результатів у більш складних завданнях, які вимагають міркувань перед відповіддю.

2.4.3. RAG

Для більш складних завдань, що потребують додаткових знань, можна побудувати систему на основі мовної моделі, яка звертається до зовнішніх джерел знань для виконання завдань [14]. Це забезпечує більшу фактичну узгодженість, підвищує надійність згенерованих відповідей і допомагає пом'якшити проблему "галюцинацій".

Перевагою RAG є те, що його можна легко оновлювати без перенавчання всієї моделі. Вона може адаптуватися до нової інформації, оскільки завжди шукає найновіші документи. Це дуже корисно, оскільки зазвичай мовні моделі мають фіксовані знання, які не змінюються. З RAG вони можуть використовувати найсвіжішу інформацію, що робить їхні відповіді точнішими та актуальнішими.

ВИСНОВКИ ДО РОЗДІЛУ 2

В другому розділі було розглянуто роботу з великими мовними моделями, їх внутрішній устрій та сучасний стан.

Також було розглянуто типові налаштування та основні методи роботи з великим мовними моделями, такими як Zero-shot/Few-shot Prompting, Chain-of-Thought, RAG.

Отримані дані знадобляться для побудови інструментів з використанням штучного інтелекту, які будуть зручними, зрозумілими для користувача, а також ефективними.

РОЗДІЛ 3
РОЗРОБКА ВИМОГ ТА ВИБІР ТЕХНОЛОГІЙ

Для спрощення перевірки життєздатності системи агрегації контенту доцільно розробити MVP з базовим інтерфейсом. Це дозволить перевірити зручність та в майбутньому презентувати ідею стартапу інвесторам.

Будь-яку розробку слід починати з виокреслення вимог. Це допоможе обрати технології для реалізації проєкту, а також пришвидшити розробку, уникнувши проблем.

Для цього вимоги було структуровано з використанням таблиць в якій кожній вимогі призначений ідентифікатор, короткий заголовок та детальний опис.

3.1. Загальна інформація

Для перевірки MVP нам буде достатньо спростити додаток, забезпечивши функціонал для одного користувача, що буде адміністратором та користувачем системи. Він зможе додавати джерела інформації та використовувати різний інструментарій, в тому числі застосовувати засоби штучного інтелекту.

В якості джерел інформації буде використано Telegram, а в майбутньому можна буде реалізувати підтримку додаткових джерел (напр. використавши публічні API чи технології web-scraping)

3.2. Функціональні вимоги

3.2.1. Користувач

Оскільки функціонал доступний неавторизованому користувачу обмежений, можна виділити одну функціональну вимогу — можливість увійти в систему (див. таблицю 3.1).

Користувач, який пройшов процес авторизації отримує доступ до основного функціоналу, що наведений в таблиці 3.2.

Таблиця 3.1

Можливості неавторизованого користувача

ID	Заголовок	Опис
FR1	Авторизація	Неавторизованому користувачу недоступний перегляд чи зміна жодних ресурсів. Він має виконати вхід заповнивши поля логіну та паролю.

Таблиця 3.2

Можливості авторизованого користувача

ID	Заголовок	Опис
FR2	Зміна даних аккаунту	Користувач має можливість акуталізувати свої дані акаунтунту: ▪ Ім’я ▪ Електронна пошта ▪ пароль
FR3	Додавання джерела контенту	Користувач має можливість додати нове джерело інформації вказавши посилання на RSS і/або інші необхідні дані для доступу до інформації. Кожному джерелу буде призначений ідентифікатор. Також користувач може вказати коротке ім’я та опис.
FR4	Видалення джерела контенту	Користувач має можливість видалити джерело контенту. Для видалення необхідно підтвердити дію.

Таблиця 3.2 (продовження)

Можливості авторизованого користувача

ID	Заголовок	Опис
FR5	Перегляд списку джерел контенту	Користувач має можливість ознайомитись зі списком підключених джерел контенту на відповідній сторінці.
FR6	Створення стрічки контенту	Користувач має можливість створити стрічку контенту з кількох джерел інформації. В подальшому він зможе застосовувати до стрічки різні інструменти.

3.2.1. Інструменти

Окрім базового функціоналу для роботи з джерелами та стрічка користувачу доступні інструменти, з якими можна ознайомитись в таблиці 3.3.

Таблиця 3.3

Можливості інструментів користувача

ID	Заголовок	Опис
FR7	Фільтрування контенту за ключовими словами	Користувач може встановити фільтр для стрічки контенту вказавши ключові слова. Контент, що міститиме визначені користувачем слова не показиватиметься в стрічці, однак, при потребі користувач може ознайомитись з прихованим контентом.
FR8	Моніторинг контенту за ключовими словами	Користувач може призначити ряд ключових слів, наявність яких в контенті може: ▪ показувати контент першим ▪ приховувати контент ▪ генерувати сповіщення

Таблиця 3.3 (продовження)

Можливості інструментів користувача

ID	Заголовок	Опис
FR9	Фільтрування контенту за текстовими правилами (LLM)	Користувач може встановити «розумний» фільтр, вказавши критерії фільтрування контенту вказавши довільну текстову підказку. Рішення про показ контенту буде приймати LLM на основі цих підказок.
FR10	Моніторинг контенту за текстовими правилами (LLM)	Користувач може встановити «розумний» моніторинг контенту, вказавши критерії за допомогою довільної текстової підказки. Рішення про показ контенту буде приймати LLM на основі цих підказок.
FR1	Саммаризація контенту (LLM)	Користувач може встановити функцію самаризації контенту, вказавши критерії та особливіості за допомогою довільної текстової підказки.
FR12	Формування звітів за певний період	Користувач може формувати звіти за певний період, враховуючи побажання користувача, що подані за допомогою довільної текстової підказки.

3.3. Нефункціональні вимоги

Нефункціональні вимоги визначають критерії оцінки загальних характеристик системи, а не окремих функцій та можливостей (див. табл. 3.4).

Таблиця 3.4

Нефункціональні вимоги

ID	Заголовок	Опис
NFR1	Безпека та конфіденційність	Сервіс має відповідати вимогам до безпеки: ▪ безпечне з'єднання (з використанням протоколу HTTPS) ▪ безпека (шифрування) чутливих даних в базі даних
NFR2	Інтерфейс	Інтерфейс має бути лаконічним та зручним, а також дотримано стандартні підходи до розробки
NFR3	Локалізація	Основна мова інтерфейсу - англійська
NFR4	Підтримка	Забезпечена підтримка останніх версій типових браузерів, а саме: ▪ Chromium (Google Chrome) ▪ Gecko (Mozilla Firefox)
NFR5	Швидкість	Реалізована перевірка та валідація даних, а також обробка виключних ситуацій.
NFR6	Надійність	Реалізована перевірка та валідація даних, а також обробка виключних ситуацій.

3.4. Вибір технологій

Вибір засобів та технологій розробки відповідальна та важлива справа, яка впливатиме на перебіг проєкту — швидкість впровадження нового функціоналу, швидкість усунення помилок та вартість розробки та підтримки проєкту.

Враховуючи контекст, а саме розробку MVP для подальшого залучення додаткових фінансових та людських ресурсів, основним критерієм вибору стане швидкість та простота розробки та інтеграції, наявні вміння розробника.

Під час проєктування та розробки проєкт зазвичай поділяють на клієнтську частину (frontend) і серверну частину (backend) [15]. Такий поділ зручний, оскільки дозволяє розподілити роботу між спеціалізованими розробниками. Серверну частину описують мовою програмування, фреймворками, бібліотеками та базою даних. До серверної частини також належить питання вибору веб-сервера, на якому працюватиме сервіс.

3.4.1. Backend

При виборі технологій для серверної частини не варто спочатку визначати мову програмування, а потім вибирати відповідний фреймворк.

Сучасна розробка веб-проєктів базується на використанні готових бібліотек та інструментів, що допомагає скоротити часові витрати і зберегти кошти для інших важливих напрямків, таких як взаємодія з користувачами та маркетинг [16]. Тому комплексний підхід є пріоритетним, особливо для стартапів або нових проєктів у великих компаніях.

Однак вибір, заснований на мові програмування, може бути виправданий, якщо в наявності є незадіяні розробники. Це дозволяє заощадити на пошуку, забезпеченні та управлінні новою командою.

Оскільки дипломний проєкт виконується індивідуально, без прив'язки до компанії та її працівників, виникає можливість самостійно вибирати технології.

3.4.1.1. Spring (Java)

Spring призначений для застосування на складних, великих та навантажених проєктах для автоматизації підприємств, державних установ. Він є частиною платформи Java EE [17]. Особливості:

- Надійність роботи: просунуті механізми для тестування, захищеність згідно усталених стандартів індустрії;

- Гнучкість налаштування: велика кількість функціоналу.

3.4.1.2. ASP.NET core (C#)

ASP.NET Core, який з'явився у 2016 році, об'єднав напрацювання ASP.NET MVC та Web API, що раніше були окремими модулями в його попередній версії [18].

Основні особливості цього фреймворку включають:

- Сумісність з продуктами екосистеми компанії Microsoft, водночас широка підтримка різних платформ: Windows, macOS, Linux та ін.;
- Модульність.

3.4.1.3. Laravel (PHP)

Laravel займає вагому нішу ринку, передусім середніх проєктів. Має велику кількість модулів та пакетів, що спрощує розробку за рахунок екосистеми [19].

Особливості:

- Притримується архутектури MVC;
- Active Record (Eloquent ORM);
- Надає гнучку вбудовану систему аутентифікації (доступ до ресурсів згідно ролей);
- Вбудована система логування, тестування (з підтривкою PHPUnit) обробка виключень та помилок;
- Широкий набір інтеграцій з різноманітними системами розсилання сповіщень, електронної пошти, тощо, модульність.

3.4.1.4. FastAPI (Python)

FastAPI – відносно новий веб-фреймворк, дуже мінімалістичний. Призначений для розробки API, не має вбудованого сервера для розробки, проте забезпечує необхідний функціонал для валідації [20]. Користувацький інтерфейс можна реалізувати окремо, з використанням фреймворків React, Angular тощо.

3.4.1.5. Django (Python)

Django має відкритий вихідний код і позиціонує себе як "веб-фреймворк для перфекціоністів із дедлайнами", оскільки включає безліч необхідних для швидкої розробки проекту. Він вже має набір соновних інструментів, які вже налаштовані за замовчуванням.

Також має широкий набір сторонніх модулів. Завдяки цьому його часто обирають для створення MVP та прототипів, щоб продукт якнайшвидше представити користувачам.

Основна сфера застосування цього фреймворку – невеликі та середні проекти з автоматизації підприємств. Однак він також підходить для великих проектів, зокрема на цьому фреймворку працюють такі відомі проєкти як YouTube, Dropbox, Spotify [21].

Серед особливостей:

- Архітектура MVT;
- Active Record (Вбудована ORM);
- Генерація HTML з використанням механізму шаблонів;
- Вбудовані аутентифікація, панель адміністратора, користувацькі сесії та інші необхідні модулі;
- Вбудований сервер для розробки;
- Якісна документація та розвинена спільнота розробників, велика кількість навчального матеріалу.

3.4.1.6. Flask (Python)

Flask притримується іншої концепції, що відрізняється від попереднього фреймворку: має лише базові інструменти розробки, що дозволяє гнучкіше структурувати проєкт та більш вільно підлаштовуватись під специфічні вимоги до продукту.

Підсистеми, що будуть необхідні в процесі розробки реалізуються самотужки або використовуються наявні модулі, розроблені сторонніми розробниками[22].

Серед особливостей:

- Базовий вбудовані модулі, що реалізують базові клієнтські сесії, базовий сервер для розробки, генератор HTML (на основі шаблонів) та налагоджувач. Підтримка юніт-тестування
- Докладна документація.

3.4.2. Frontend

При розробці клієнтської частини є два підходи: застосування вбудованих у фреймворк засобів для генерації HTML за допомогою шаблонів або використання окремих спеціалізованих фреймворків. Вибір окремих фреймворків сповільнить процес, оскільки проект невеликий і має простий інтерфейс, тому було вирішено використовувати вбудовані шаблони.

Крім власне самого рушія шаблонів для пришвидшення розробки можна використовувати бібліотеки стилів Bootstrap, Pure.CSS та ін.

Порівняння можливостей шаблонних рушіїв, вбудованих у фреймворки, наведено в таблиці 3.5.

Таблиця 3.5

Порівняльна таблиця рушіїв для шаблонів

Фреймворк	Spring	ASP.NET Core	Laravel	Django	Flask
Назва	FreeMarker	Razor Pages	Blade	Templates	Jinja2
Вбудований	Ні	Так	Так	Так	Так
Змінні	Так	Так	Так	Так	Так
Умовні конструкції	Так	Так	Так	Так	Так
Цикли	Так	Так	Так	Так	Так
Функції	Так	Так	Так	Так	Так
Наслідування	Так	Так	Так	Так	Так
Виключення	Так	Так	Так	Так	Так

Вони є у всіх розглянутих рішеннях, крім Spring Boot [23]. У цьому фреймворку можна додати сторонній рушій, встановивши відповідний пакет.

З таблиці можна зробити висновок, що розглянуті фреймворки мають однакові можливості та необхідний набір функцій.

3.4.3. База даних

База даних є однією з ключових складових серверної частини проєкту. За даними ресурсу Statistics & Data, найбільш популярними базами даних є Oracle (32,09%), MySQL (16,64%), Microsoft SQL Server (13,72%), PostgreSQL (4,94%) та MongoDB (4,48%) [24]. З них Oracle і Microsoft пропонують пропрієтарні рішення, призначені для великих і складних проектів, що не є доцільним для поточного проєкту. Порівняння баз даних представлено в таблиці 3.6.

Сучасні системи управління базами даних мають подібні функціональні набори, і для звичайного користувача різниця між ними майже не відчутна. Проте PostgreSQL пропонує ширший спектр типів даних і вважається більш надійною в індустрії. Документо-орієнтовані бази, як-от MongoDB, створені переважно для неструктурованої інформації. SQLite застосовується для невеликих проєктів.

Таблиця 3.6

Порівняльна таблиця баз даних

Характеристика	MongoDB	MySQL	PostgreSQL	SQLite
Тип	Документо-орієнтована	Реляційна	Реляційна	Реляційна
Структурованість	Ні	Так	Так	Так
Типи даних	Достатньо	Достатньо	Розширений набір	Обмежений набір
Підтримка Linux	Так	Так	Так	Так
Доступність	Open-source	Open-source	Open-source	Open-source

3.4.4. Засоби штучного інтелекту

Основою продукту стане AI-функціональність і безумовним лідером по роботі з штучним інтелектом є мова програмування Python. Водночас Python є багатопрофільною мовою, і широко використовується для розробки веб-додатків, автоматизації.

Водночас, використання іншої мови програмування для back-end ускладнює розробку — необхідно розгортати та підтримувати окремий сервіс для функціональності пов’язаної з ШІ.

3.4.4.1. Модель

Доступ до моделі можна отримати у стороннього провайдера скориставшись API або розгорнути одну з моделей з відкритим вихідним кодом на власному сервері.

Найбільш просунуті моделі від компаній OpenAI [25], Antropic [26] та Google [27] доступні платно з використанням API. В таблиці 3.7 наведена вартість використання комерційних моделей.

Водночас відкриті моделі активно розвиваються та деякі їх версії можна розгорнути навіть на ноутбуці. Найкращі параметри швидкодії забезпечує використання GPU, водночас для деяких популярних моделей існують оптимізовані для виконання на CPU версії.

Таблиця 3.7

Вартість використання хмарних моделей.

Модель	Розмір контексту	Вартість, 1 млн. токенів на вхід	Вартість, 1 млн. токенів на вихід	Компанія
gpt-4o	128K	$5.00	$15.00	OpenAI
gpt-4o-2024-05-13	128K	$5.00	$15.00	OpenAI
gpt-3.5-turbo-0125	16K	$0.50	$1.50	OpenAI

Таблиця 3.7 (Продовження)

Вартість використання хмарних моделей.

Модель	Розмір контексту	Вартість, 1 млн. токенів на вхід	Вартість, 1 млн. токенів на вихід	Компанія
gpt-3.5-turbo-instruct	4K	$1.50	$2.00	OpenAI
Haiku	200K	$0.25	$1.25	Anthropic
Sonnet	200K	$3	$15	Anthropic
Opus	200K	$15	$75	Anthropic
Gemini Flash	128K/1M	$0.35/$0.70	$1.05/$2.10	Google
Gemini 1.5 Pro	128K/1M	$3.50/$7.00	$10.50/$21.00	Google

Водночас відкриті моделі активно розвиваються та деякі їх версії можна розгорнути навіть на ноутбуці. Найкращі параметри швидкодії забезпечує використання GPU, водночас для деяких популярних моделей існують оптимізовані для виконання на CPU версії.

Для розгортання моделей на власному залізі існують різні способи та програмне забезпечення Ollama [28], llama.cpp [29], transformers.

Хоча існують певні загальні метрики для визначення можливостей великих мовних моделей, такі як MMLU, на практиці необхідно тестувати різні моделі, можливо навіть менші локальні моделі дозволять отримати прийнятні результати.

3.4.4.2. Бібліотеки

Оскільки велика мовна модель лише виступає лише в ролі генератора токенів, для більш складних дій необхідно використовувати надбудови.

Для реалізації AI-функціональності в нашому проєкті ми розглядаємо кілька бібліотек та інструментів, призначених для роботи з великими мовними моделями (LLM).

LangChain [30] та LLamaIndex [31] є двома потужними бібліотеками для роботи з LLM, які мають свої унікальні переваги та недоліки (див. таблицю 3.8).

Таблиця 3.8

Порівняльна таблиця бібліотек.

Характеристика	LangChain	LLamaIndex
Основна функція	Побудова ланцюгів обробки даних для LLM	Індексація та інтеграція даних з LLM
Підтримка моделей	Широка (GPT, BERT, інші)	Обмежена
Гнучкість	Висока	Помірна
Простота використання	Складна для новачків	Відносно проста
Підтримка форматів даних	Широка (текст, зображення, інші)	Обмежена, переважно текст
Документація	Добре документована	Добре документована

LangChain є бібліотекою, що дозволяє будувати потужні додатки на основі LLM шляхом об'єднання різних моделей і інструментів. Її головними перевагами є гнучкість у побудові ланцюгів обробки даних та підтримка інтеграції з багатьма моделями, що робить її ідеальною для реалізації складних AI-функцій. Основним недоліком LangChain є те, що вона може бути складною у налаштуванні для новачків.

У порівнянні з нею, LLamaIndex є інструментом для інтеграції та індексації даних у великих мовних моделях, з перевагами у підтримці різних форматів

даних та ефективному індексуванні, але може потребувати значних ресурсів для обробки великих обсягів даних.

З огляду на наші потреби, LangChain виглядає більш підходящим варіантом завдяки своїй гнучкості та широкій підтримці моделей.

ВИСНОВКИ ДО РОЗДІЛУ 3

В цьому розділі були розроблені функціональні та нефункціональні вимоги до розроблюваної системи. Функціональні вимоги визначають можливості користувача та інструментарію. Вимоги до розроблюваної системи були формалізовані та сформовано чіткий перелік з короткими заголовками та унікальними ідентифікаторами. Таке подання допомогло обрати технології розробки та дозволить ефективніше спланувати процес розробки продукту.

Було розглянуто доступні технології та обрано мову програмування Python у зв’язці з фреймворком Django та бібліотекою LangChain. Це дозволить значно прискорити розробку MVP, оскільки не потрібно створювати окремі сервіси для інтеграції бібліотек для роботи з LLM.

Для розробки користувацького інтерфейсу застосовуватиметься вбудований функціонал шаблонів фреймворку Django.

РОЗДІЛ 4
РОЗРОБКА РІШЕННЯ

4.1. Підготовка

Перш ніж розпочати розробку, необхідно виконати низку підготовчих заходів [32] і підготувати:

- структуру проєкту,
- систему контролю версій,
- віртуальне середовище,
- базу даних.

Добре структурований проєкт зменшує кількість помилок при внесенні змін і скорочує час на пошук потрібних файлів. Системи контролю версій дозволяють відстежувати стан проєкту після внесення змін і в разі помилок або змін у підході до вирішення завдання відновлювати попередню версію.

Для цього проєкту обрано git як стандартну систему контролю версій, що широко використовується як у комерційних, так і у open-source проєктах. Хоча розробка ведеться індивідуально, git буде використовуватися разом з хмарним сервісом GitHub для надійного зберігання та запобігання втраті даних у випадку раптового виходу з ладу комп'ютера. За наявності інтернет-з'єднання роботу над проєктом можна продовжити з будь-якого комп'ютера з встановленим git.

Віртуальне середовище дозволяє ізолювати проєкт і запобігти конфліктам між різними версіями бібліотек і модулів [32].

Щоб створити Django-проєкт, потрібно перейти у віртуальне середовище виконання та встановити пакет django разом з іншими необхідними бібліотеками. Після цього рекомендується зберегти залежності у спеціальному файлі, використовуючи команду pip freeze > requirements.txt. Цей файл міститиме всі необхідні для роботи сторонні модулі, і його потрібно оновлювати після прийняття рішення про використання нових залежностей [32].

Далі потрібно створити структуру проєкту за допомогою команди django-admin startproject aicontentaggregator, де aicontentaggregator – назва проєкту. Структуру проєкту після виконання всіх цих дій показано на рисунку 4.1.

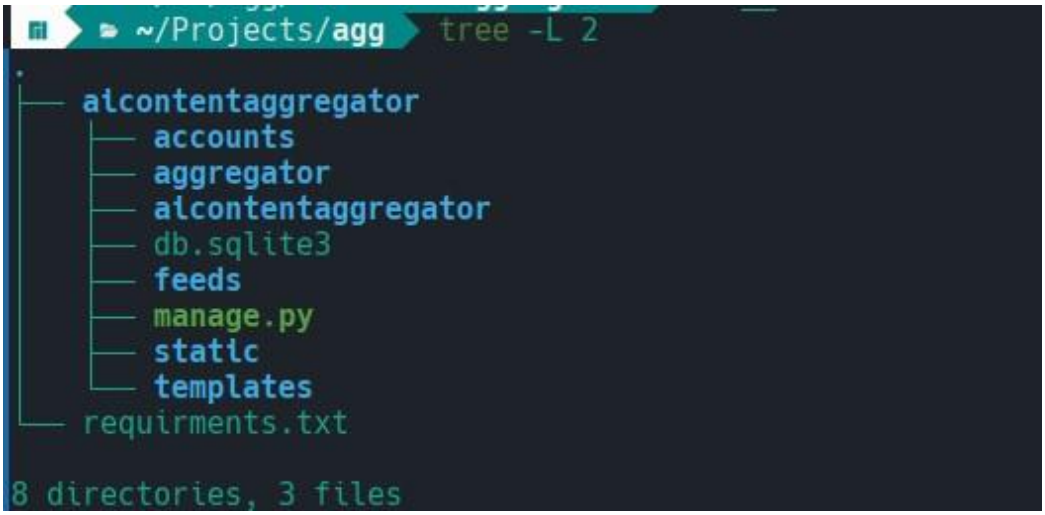

Рис. 4.1. Структура проєкту

Після створення Django-проєкту важливо забезпечити безпеку чутливих даних у файлі settings.py. Різні методи зберігання налаштувань можуть бути використані, але рекомендується використання змінних оточення [22]. Для цього необхідно імпортувати модуль os та замінити пряме представлення чутливих даних (рис. 4.2).

```
15
16
17  # Build paths inside the project like this: BASE_DIR / 'subdir'.
18  BASE_DIR = Path(__file__).resolve().parent.parent
19
20
21  # Quick-start development settings - unsuitable for production
22  # See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/
23
24  # SECURITY WARNING: keep the secret key used in production secret!
25  SECRET_KEY = str(os.getenv('DJANGO_SECRET_KEY'))
26
27  # SECURITY WARNING: don't run with debug turned on in production!
28  DEBUG = True
29
30  ALLOWED_HOSTS = []
```

Рис. 4.2. Файл setting.py

4.2. Проєктування

Однією з найкращих практик є поділ проєкту на логічні частини [32]. Проаналізувавши вимоги до продукту, можна розподілити функціонал на кілька Django-додатків: accounts, aggregator, feeds. Accounts включатиме все, що стосується сторінок авторизації, sources - все, що пов'язане з отриманням контенту, feeds - з формуванням стрічок та інструментами.

На наступному етапі потрібно визначити необхідні сторінки та шляхи. Це важливо для створення зрозумілої та впорядкованої системи, зручною як для користувачів, так і для розробників. Шляхи можна умовно розподілити на групи для спрощення навігації:

- для роботи з обліковим записом;
- для роботи з джерелами;
- для роботи з стрічками контенту;
- для налаштування інструментів стрічки

Детальніше про сторінки та шляхи в таблицях 4.1–4.4.

Таблиця 4.1

Шляхи для роботи з обліковим записом

Назва	Шлях	Опис
Sign In	/login	Сторінка входу в систему
Logout	/logout	Виконує вихід з облікового запису та перенаправляє на сторінку входу в систему.
Profile	/account	Особистий кабінет. Включає інформацію, пов'язану з обліковим записом користувача.
Edit Profile	/account/edit	Інтерфейс для зміни даних профілю

Таблиця 4.2

Шляхи для роботи з джерелами

Назва	Шлях	Опис
Sources	/sources	Містить список усіх доданих користувачем джерел
Create Source	/source/create	Сторінка створення нового джерела
Source Page	/source/<uuid>	Містить інформацію про джерело
Edit Source	/source/<uuid>/edit	Інтерфейс для редагування даних джерела — опису, посилання, заголовку

Таблиця 4.3

Шляхи для роботи з стрічками контенту

Назва	Шлях	Опис
Feeds	/feeds	Містить список усіх створених користувачем стрічок контенту
Create Feed	/feed/create	Сторінка створення нової стрічки конетнту
Feed Page	/feed/<uuid>	Містить інформацію про стрічку контенту та її вміст
Edit Feed	/feed/<uuid>/edit	Інтерфейс для редагування стрічки контенту

Таблиця 4.4

Шляхи для роботи з інструментами

Назва	Шлях	Опис
Налаштування фільтрації	/feed/<uuid>/filter	Містить список усіх створених користувачем інструментів

Таблиця 4.4 (продовження)

Шляхи для роботи з інструментами

Назва	Шлях	Опис
Налаштування моніторингу	/feed/<uuid>/monitor	Сторінка створення нової стрічки конетнту
Налаштування звітів	/feed/<uuid>/reports	Містить інформацію про стрічку контенту та її вміст

4.3. Макетування інтерфейсу

Для прискорення розробки та формування уявлення про вигляд продукту можна створювати макети сторінок. Це допомагає уникнути непорозумінь і полегшує розуміння задач, що виникають під час роботи, як у випадку роботи в команді, так і у випадку одного розробника.

На рисунку 4.3 зображені макети сторінок авторизації та реєстрації, які мають стандартний вигляд.

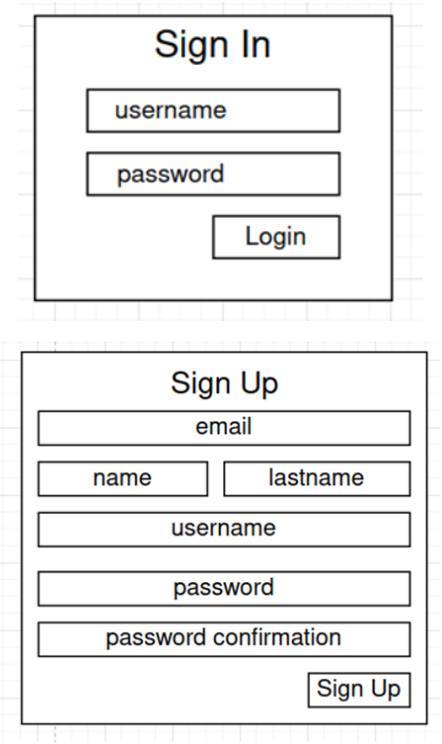

Рис. 4.3. Макет сторінок авторизації

Для відображення доступних джерел контенту можна використати спискове представлення (рис. 4.4). На цьому макеті також показано навігаційну панель з посиланнями на списки джерел, сторінки профілю та виходу з облікового запису. Поряд із заголовком знаходиться кнопка для створення нового джерела контенту.

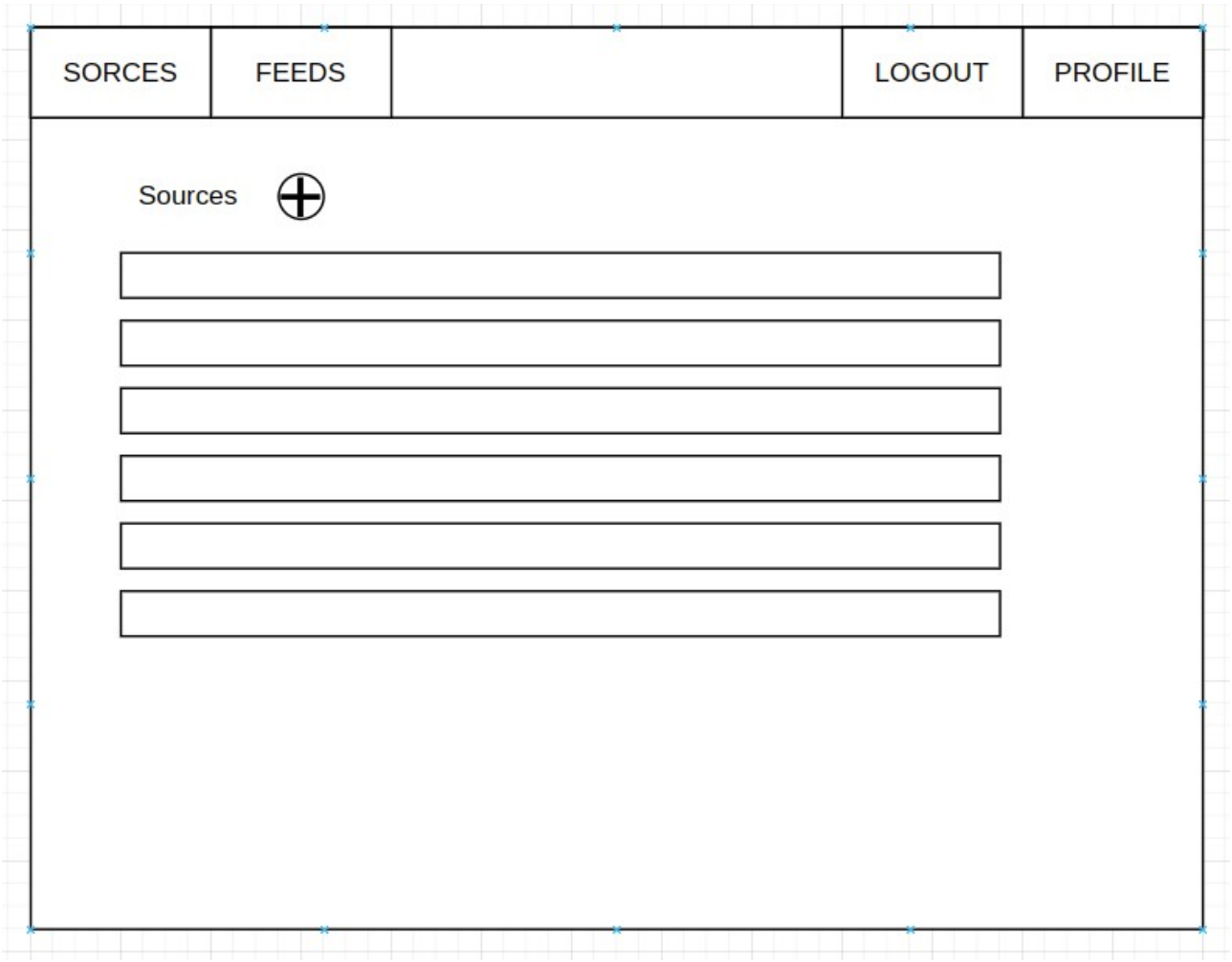

Рис. 4.4. Макет сторінок Sources/Feeds

У проєкті необхідно відображати такі дані, як джерела контенту, стрічки контенту та інструменти. Хоча існує кілька способів їх відображення, найбільш зручним є використання таблиць. Приклад такого інтерфейсу показано на рисунку 4.5. Для додавання нових даних до таблиць можна використовувати форми.

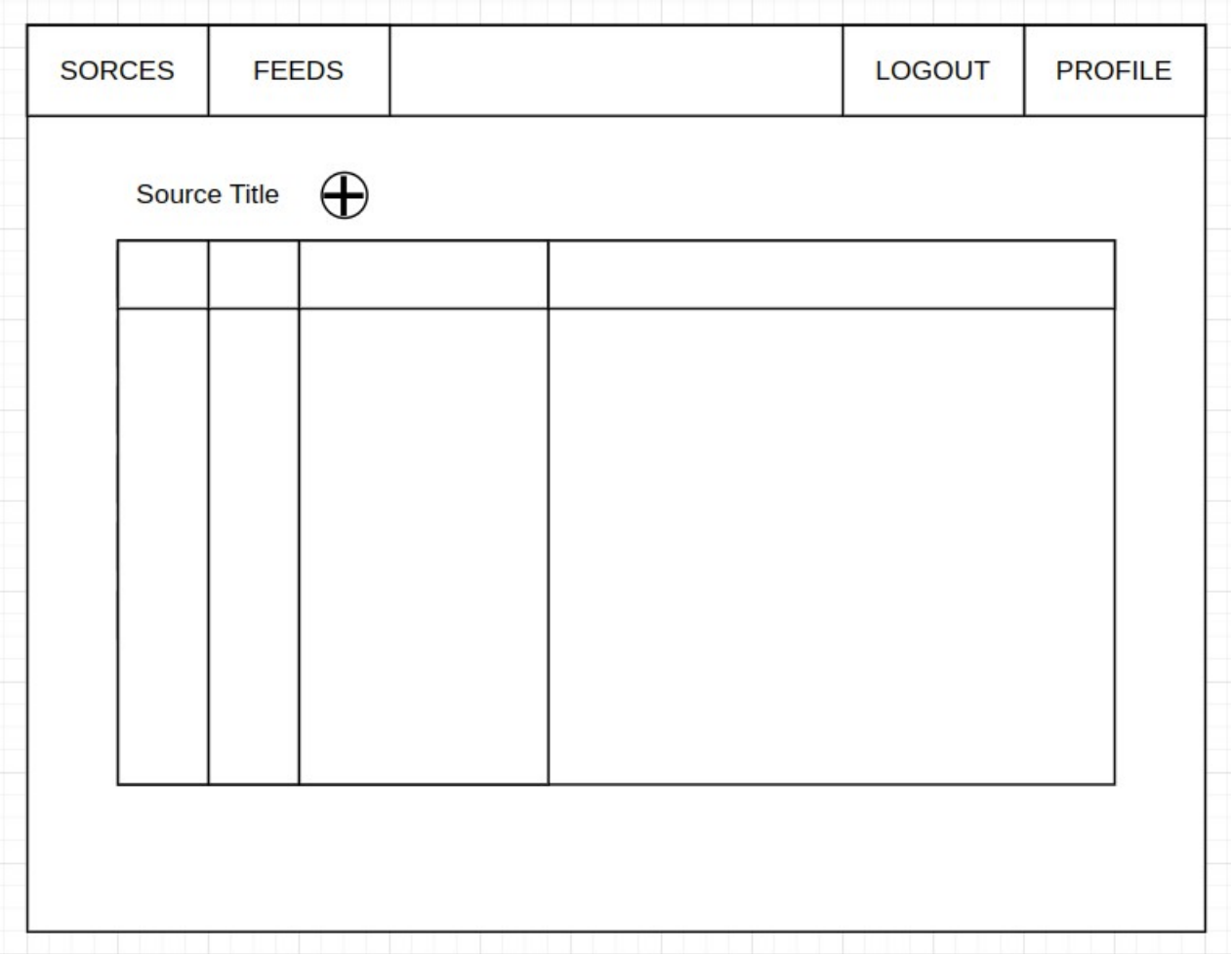

Рис. 4.5. Макет сторінок перегляду конкретного джерела чи стрічки

4.4. Огляд моделей

Під час розробки системи було реалізовано наступні моделі (детальніше у таблицях 4.5–4.6):

- Source
- Post
- Feed
- FeedSource
- FeedItem

Модель Source (Джерело контенту) зберігає всю необхідну інформацію для представлення окремого джерела. Опис властивостей та методів, які знаходяться в даній моделі, наведений в таблиці 4.5.

Таблиця 4.5

Модель Source

Властивість	Тип	Призначення
id	PrimaryKey	Ідентифікатор джерела, використовується для управління даними
title	CharField	Довільна назва джерела, яку вказує користувач, використовується для зручнішої ідентифікації
channel_id	CharField	Ідентифікатор каналу, з якого отримується контент
description	TextField	Використовується користувачем для опису джерела, додавання своїх нотаток, тощо
get_absolute_url	Метод	Метод, який використовується для отримання посилання на детальний перегляд інформації про джерело
__str__	Метод	Метод, який використовується для отримання текстового відображення моделі

Модель Post (Одиниця контенту) зберігає всю необхідну інформацію для представлення окремої публікації. Опис властивостей і методів, що містяться в цій моделі, наведено в таблиці 4.6.

Таблиця 4.6

Модель Post

Назва складової	Тип складової	Опис складової
content	TextField	Вміст одиниці контенту, отриманого з джерела
source	ForeignKey	Джерело, звідки було отримано одиницю контенту. Також застосовується метод видалення CASCADE
date	DateTimeField	Дата публікації одиниці контенту
post_id	IntegerField	Ідентифікатор публікації в межах джерела, для усунення дублювання

Модель Feed (Стрічка) містить усю необхідну інформацію для відображення окремої стрічки (таблиця 4.7).

Таблиця 4.7

Модель Feed

Властивість	Тип	Призначення
id	PrimaryKey	Ідентифікатор стрічки контенту для управління даними
title	CharField	Довільна назва стічки, яку вказує користувач, використовується для зручнішої ідентифікації

Таблиця 4.7 (продовження)

Модель Feed

Властивість	Тип	Призначення
description	TextField	Використовується користувачем для опису стрічки контенту, додавання своїх нотаток, тощо
get_absolute_url	Метод	Метод, який використовується для отримання посилання на детальний перегляд інформації про стрічку контенту
__str__	Метод	Метод, який використовується для отримання текстового відображення моделі

Модель FeedSource (Джерело стрічки) містить усю необхідну інформацію для відображення окремого джерела стрічки, а опис її властивостей і методів подано в таблиці 4.8. Основні складові цієї моделі включають feed та source. Поле feed є зовнішнім ключем (ForeignKey), яке посилається на стрічку, куди потраплятиме контент. Аналогічно, поле source також є зовнішнім ключем, що вказує на джерело отриманого контенту.

Модель FeedItem (Елемент стрічки) містить усю необхідну інформацію для відображення окремого елемента стрічки, а опис її властивостей і методів подано в таблиці 4.5.

Таблиця 4.8

Модель FeedSource

Назва складової	Тип складової	Опис складової
feed	ForeignKey	Стрічка, яка збирає контент.
source	ForeignKey	Джерело, звідки було отримано одиницю контенту.

Таблиця 4.9

Модель FeedItem

Назва складової	Тип складової	Опис складової
modified_content	TextField	Опціональний, видозмінений системою вміст контенту
original_post	ForeignKey	Посилання на оригінальну одиницю контенту

4.5. Інструменти

Для реалізації інструментів використовується бібліотека LangChain. LangChain — це комплексна бібліотека, розроблена для полегшення створення додатків, які використовують великі мовні моделі (LLMs). Основні компоненти бібліотеки включають ланцюги, ретривери, векторні сховища, інструменти та агенти.

Ланцюги (chains) дозволяють створювати послідовні виклики, що можуть включати мовні моделі, інші ланцюги або утиліти, що забезпечує можливість виконання складних процесів (рис. 4.6).

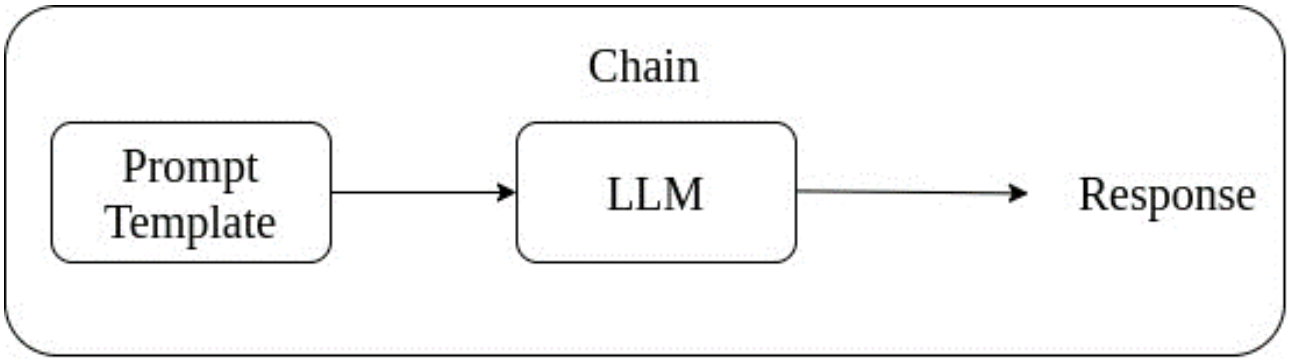

Рис. 4.6. Внутрішній устрій LangChain

Ретривери (retrievers) відповідають за пошук та повернення релевантних документів на основі запиту, дозволяючи ефективно працювати з великим обсягом неструктурованих даних.

Векторні сховища (vector stores) займаються збереженням та пошуком векторних представлень даних, що є корисним при виконанні пошуку за схожістю. Інструменти (tools) надають інтерфейси для взаємодії з зовнішнім світом, такими як бази даних або API (рис. 4.7), а набори інструментів (toolkits) об’єднують пов’язані інструменти для виконання специфічних завдань.

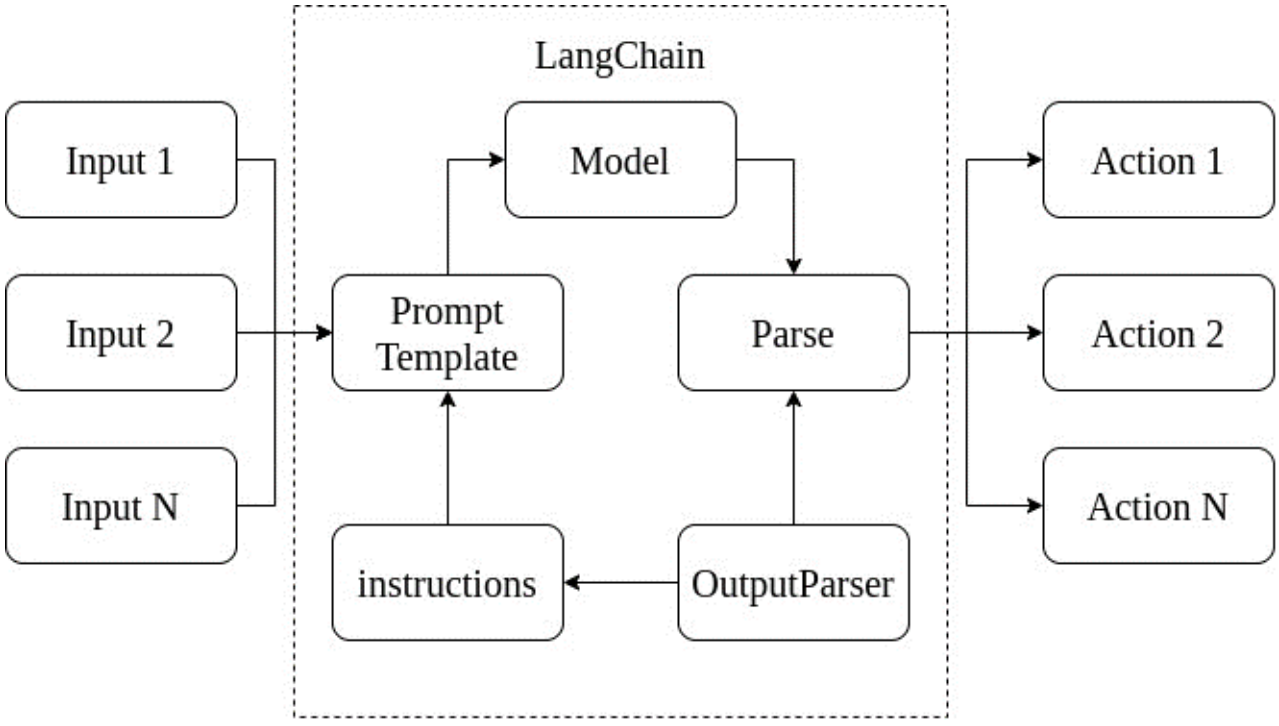

Рис. 4.7. LangChain, взаємодія

Агенти (agents) використовують мовну модель як рушій для прийняття рішень та виконання дій, дозволяючи будувати більш складні системи, що можуть автономно виконувати завдання. Важливою складовою LangChain є також система зворотних викликів (callbacks), що дозволяє підключатися до різних етапів роботи додатків для логування, моніторингу та інших цілей.

Бібліотеку можна використовувати з різними великими мовними моделями, в тому числі локальними (рис. 4.9).

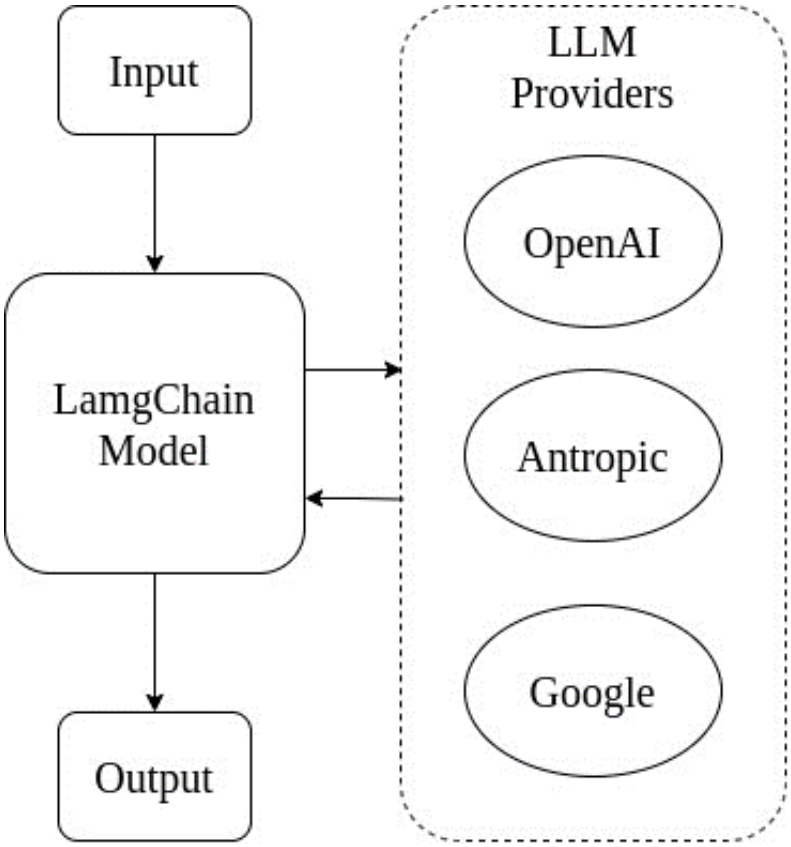

Рис. 4.8. Внутрішній устрій LangChain

4.5.1. Фільтрація

Функція filter_content призначена для фільтрації контенту на основі заданих користувачем критеріїв. Користувач надає пост та критерії, які функція використовує для прийняття рішення про відображення чи приховування контенту.

Функція ініціалізує модель LangChain, створює запит з критеріями та постом, і надсилає його до OpenAI. OpenAI аналізує пост і повертає відповідь, на основі якої функція вирішує, чи додати пост до відфільтрованого списку, який потім повертається користувачу. Деталі роботи функції можна ознайомитись на рисунку 4.9.

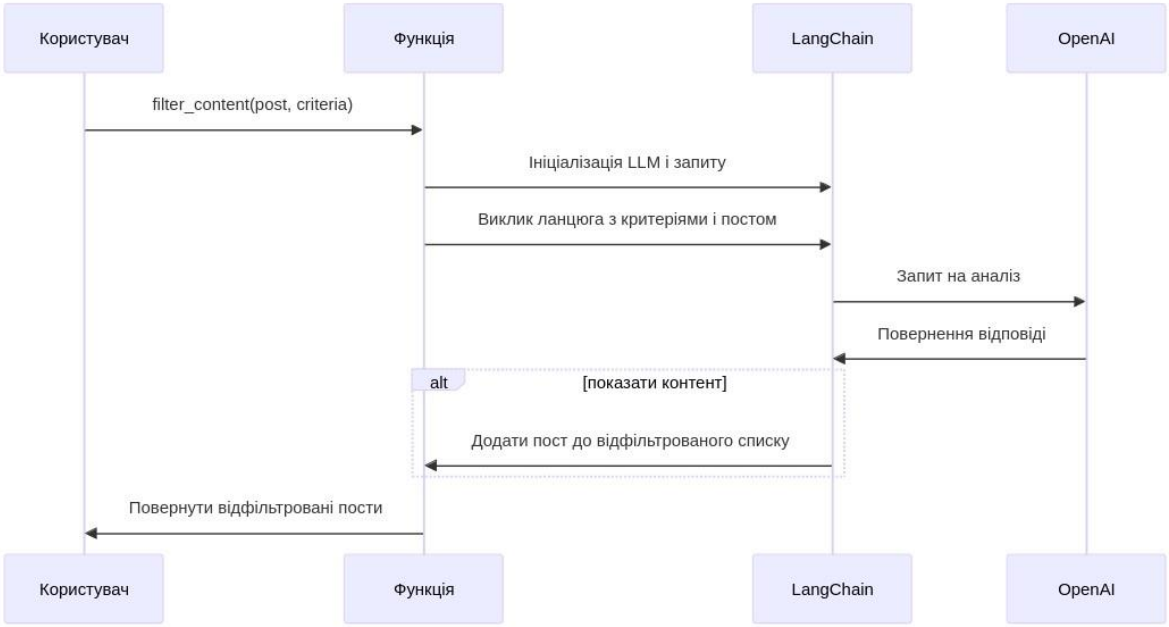

Рис. 4.9. Фільтрація контенту

4.5.2. Моніторинг

Функція monitor_content використовується для моніторингу контенту з метою генерації сповіщень на основі важливості постів, визначеної за критеріями користувача. Користувач надає пост та критерії важливості.

Функція ініціалізує модель LangChain, створює запит з критеріями та постом, і надсилає його до OpenAI. OpenAI аналізує пост і повертає відповідь, яка вказує, чи є пост важливим. Якщо пост важливий, він додається до списку

сповіщень, який потім повертається користувачу. Деталі роботи функції можна ознайомитись на рисунку 4.10.

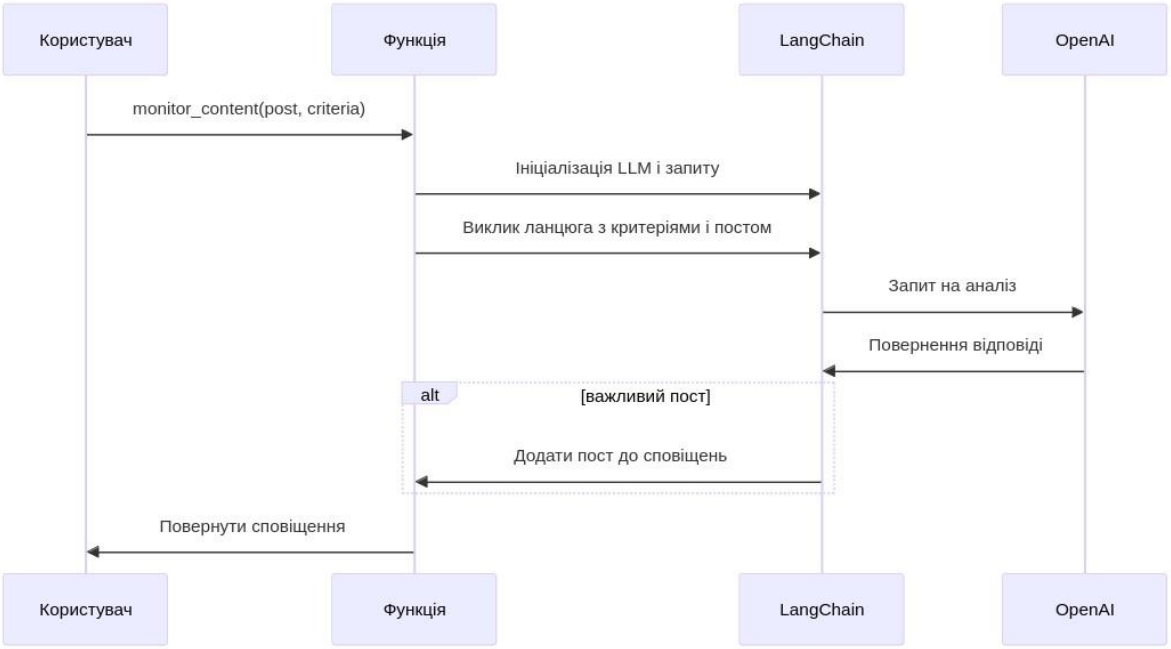

Рис. 4.10. Моніторинг контенту

4.5.3. Підсумовування

Функція summarize_content призначена для створення звіту, що підсумовує надані пости відповідно до опису користувача. Користувач надає опис та список постів для підсумовування.

Функція ініціалізує модель LangChain, створює запит з описом та постами, і надсилає його до OpenAI. OpenAI аналізує надану інформацію і генерує підсумок, який повертається функції, а потім звіт передається користувачу. Деталі роботи функції можна ознайомитись на рисунку 4.11.

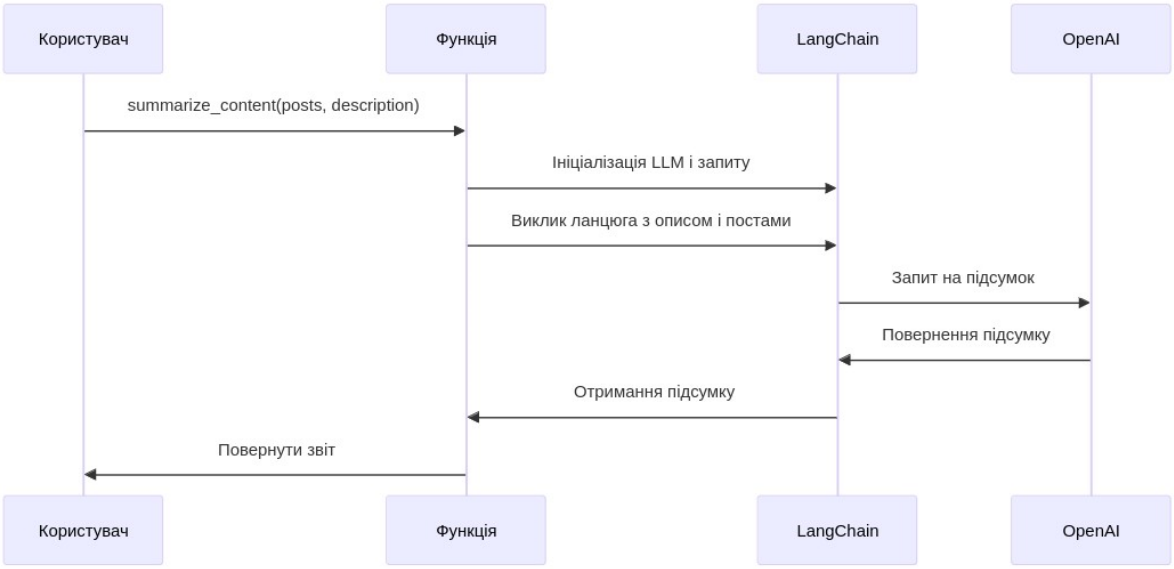

Рис. 4.11. Підсумовування контенту

З кодом функцій можна ознайомитися у відповідних додатках.

4.6. Отримання даних

Для отримання даних використовується бібліотека telegram2rss та Django-apscheduler.

Функція update_source відповідає за оновлення контенту для конкретного джерела. Вона використовує ID каналу Telegram та обмежує кількість сторінок для отримання нових повідомлень. Після цього отримані повідомлення перевіряються на наявність у базі даних. Якщо запис ще не збережений, він додається разом із необхідною інформацією, такою як зміст та дата публікації.

4.7. Зовнішній вигляд

Інтерфейс додатка дозволяє адміністратору додавати нові джерела інформації, редагувати їх або видаляти. На рисунку 4.12 представлена таблиця джерел контенту, де кожне джерело має назву, ідентифікатор каналу, опис, а також можливість редагувати або видалити запис. Користувач може легко керувати своїми джерелами контенту через цю таблицю, додаючи нові джерела через кнопку додавання, розташовану зверху таблиці.

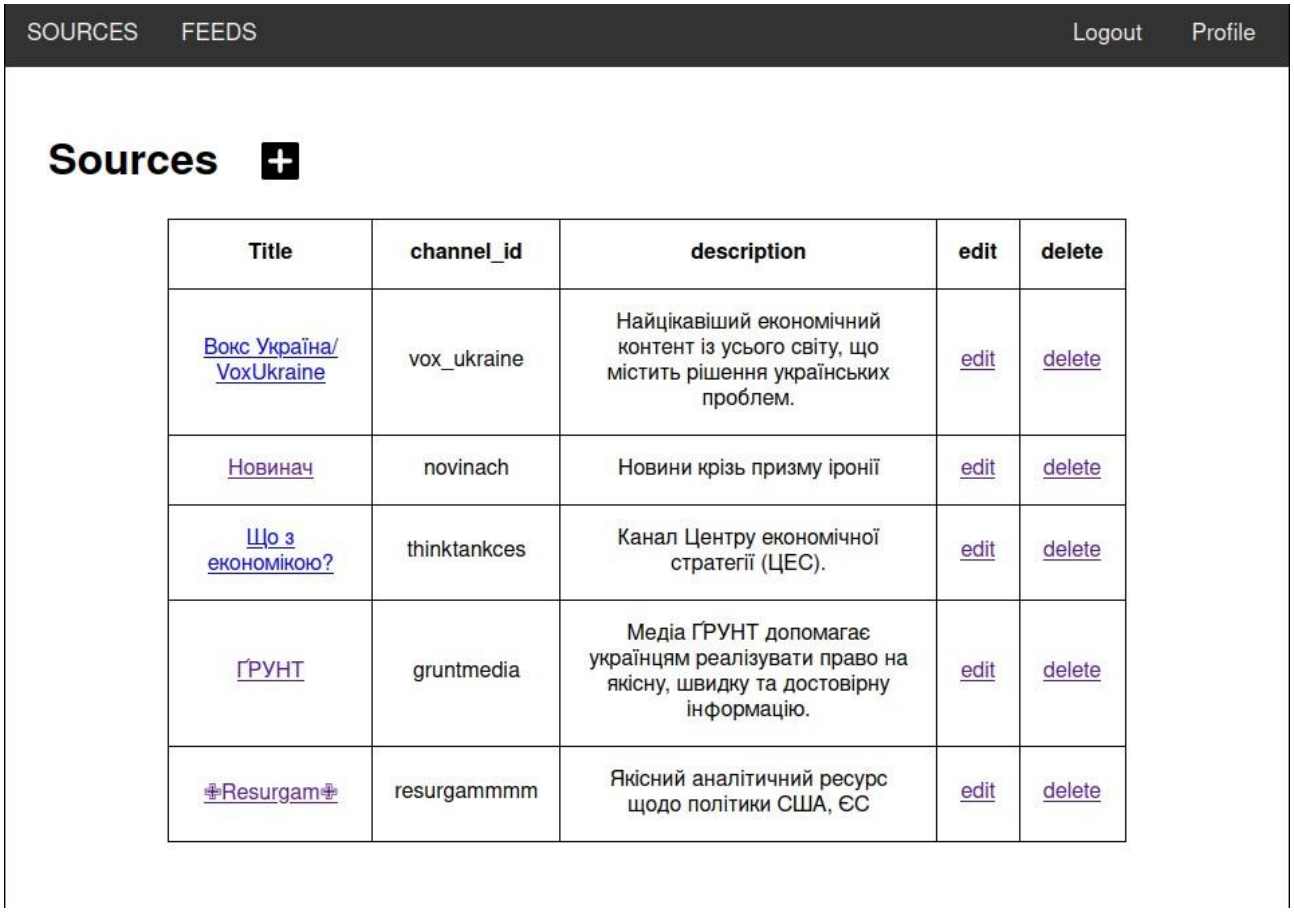

Рис. 4.12. Сторінка Sources

На рисунку 4.13 представлений вміст конкретного джерела інформації. Тут користувач може переглядати окремі пости з детальною інформацією про кожен з них, включаючи дату та час публікації. Є можливість переглядати зміст постів безпосередньо в таблиці.

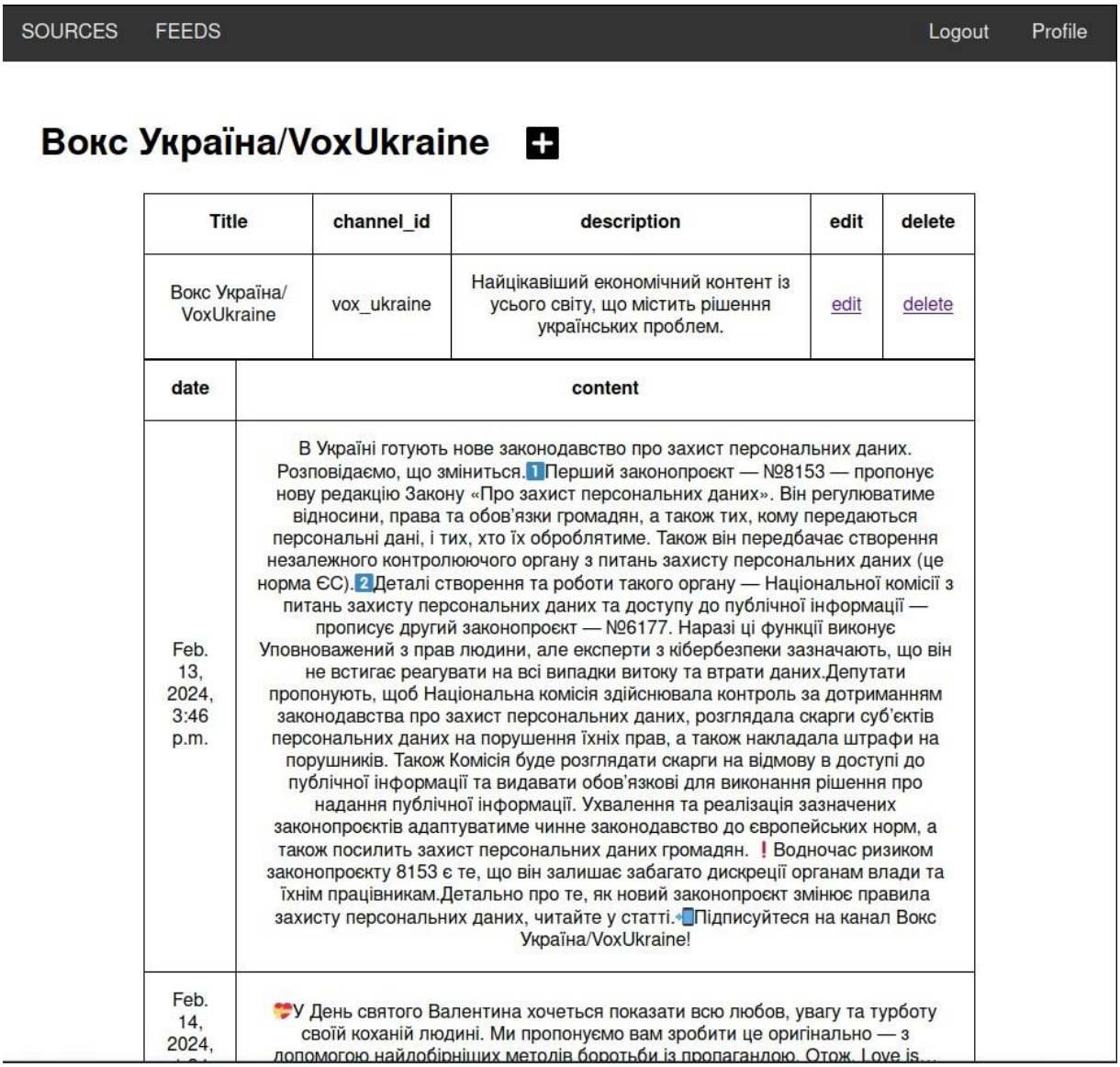

Рис. 4.13. Сторінка джерела

ВИСНОВКИ ДО РОЗДІЛУ 4

Було описано ключові етапи розробки проєкту, включаючи налаштування середовища розробки, створення проєкту, розробку структури веб-ресурсу, а також створення моделей та макетів інтерфейсу.

Під час підготовки були виконані наступні кроки: створено директорію проєкту, налаштовано систему контролю версій, створено віртуальне середовище виконання для Python. Також було створено Django-проєкт та налаштовано його конфігурацію для забезпечення надійного зберігання чутливої інформації.

На наступному етапі були описані сторінки та шляхи, необхідні для відображення сервісу та обробки дій користувачів.

Під час розробки системи було реалізовано моделі для зберігання даних про необхідні в рамках проєкту сутності: Source, Post, Feed, FeedSource, FeedItem. Описано призначення їх властивостей та методів.

Додатково були реалізовані інструменти для фільтрації та моніторингу контенту на основі ключових слів та текстових правил, що використовують можливості великих мовних моделей (LLM). Розроблений інтерфейс забезпечує зручний доступ до всіх функцій, необхідних для ефективного управління та аналізу інформаційних потоків.

РОЗДІЛ 5
РОЗРОБКА СТАРТАП-ПРОЄКТУ

5.1. Інформаційна картка стартап-проєкту

Загальні відомості про стартап проєкт описано в інформаційній картці проєкту (таблиця 5.1).

Таблиця 5.1

Інформаційна картка проєкту

Назва пункту	Опис пункту
1. Назва проєкту	Система агрегації та фільтрації контенту з використанням штучного інтелекту
2. Автори проєкту	Святощук Ростислав
3. Коротка анотація	Метою даного проєкту є розробка та реалізація системи для агрегації та фільтрації контенту, яка застосовуватиме сучасні здобутки в галузі штучного інтелекту.
4. Термін реалізації проєкту	4 місяці
5. Необхідні ресурси	Фінансові: ▪ заробітна плата для задіяних спеціалістів: менеджера, back-end розробника, front-end розробника і розробника DevOps; ▪ витрати на обладнання та матеріали, необхідні для роботи: робоче приміщення, ноутбуки, комунальні послуги, кава, снеки;

Таблиця 5.1 (продовження)

Інформаційна картка проєкту

Назва пункту	Опис пункту
5. Необхідні ресурси	▪ витрати, пов'язані з веденням бізнесу: податки, юридичні витрати, збори, тощо. Інтелектуальні: Для розробки стартапу використовуються технології з відкритим кодом, що розповсюджуються безкоштовно. Ліцензії не потрібні. Людські: Необхідна команда досвідчених спеціалістів для управління невеликою компанією, або хоча б командою розробників: ▪ менеджер з досвідом розвитку нових продуктів ▪ senior back-end розробник; ▪ senior front-end розробник; ▪ UI/UX designer ▪ DevOps; ▪ NLP/AI спеціаліст; ▪ згодом продакт-аналітик, бухгалтер, тестувальники

Таблиця 5.1 (продовження)

Інформаційна картка проєкту

Назва пункту	Опис пункту
6. Опис проблеми, яку вирішує проєкт	Проект дасть можливість ефективно управляти інформаційними потоками контенту з різних джерел, відслідковувати необхідні користувачу події та фільтрувати інформаційних шум, використовуючи AI. В епоху генеративного контенту це дозволить контролювати обсяги спожитої інформації та зробити їх більш ефективними.
7. Головні цілі та завдання проєкту	Ціль – розробка веб-додатку для агрегації та фільтрації контенту. Веб-додаток надаватиме можливість користувачам Інтернету додавати джерела інформації та автоматизувати різні аспекти роботи з інформацією. Завдання: ▪ створення веб-додатку з мінімалістичним та зручним інтерфейсом; ▪ розробка модулів отримання інформації з різних джерел; ▪ розробка інструментів роботи з контентом, з використанням LLM. ▪ розгортання веб-додатку;

Таблиця 5.1 (продовження)

Інформаційна картка проєкту

Назва пункту	Опис пункту
	▪ робота із залучення користувачів, збір відгуків, доопрацювання продукту.
8. Очікувані результати	Очікується, що розроблений веб-додаток стане популярним серед користувачів завдяки своїй функціональності та зручності у використанні. Зібрані відгуки допоможуть постійно вдосконалювати продукт, роблячи його ще більш ефективним та привабливим для аудиторії. Це дозволить в перспективі досягти окупності продукту

5.2. Технологічний аудит ідеї стартап-проєкту

Перед початком розробки стартап-проекту варто провести аудит технологій. Правильний вибір технологій для розробки дозволить реалізувати більше корисних для клієнта функцій, а також спростить і пришвидшить розробку.

Обраний технологічний стек добре підходить для розробки продукту, добре задокументований, підтримується і регулярно оновлюється.

Таблиця 5.2 демонструє основні завдання проекту та доступність технології, яку можна використати для її реалізації.

Таблиця 5.2

Технологічна здійсненність завдань проекту

Завдання проєкту	Технологія реалізації	Наявність технології	Доступність технології
Зберігання даних про облікові записи, користувачів, доступ до даних	Система управління базами даних PostgreSQL	Наявна	Розповсюджується безкоштовно
Серверна частина додатку, обробка запитів, реалізація бізнес-логіки	Мова програмування Python, веб-фреймворк Django	Наявна	Розповсюджується безкоштовно
Користувацький інтерфейс	Вбудований шаблонізатор	Наявна	Розповсюджується безкоштовно
Отримання інформації з різних джерел	RSS, API, web scraping	Наявна	Розповсюджується безкоштовно
Інструменти роботи з контентом з використанням LLM	GPT-3.5, GPT-4, LLAMA	Активно розвивається	Передові моделі платні, також наявні open source, які розповсюджується безкоштовно (в т.ч. для комерційного використання)

Завдяки таблиці вище добре видно, що всі завдання проекту можуть бути реалізовані за допомогою існуючих програмних технологій. Обрані технології є надійними, швидкими, легким в освоєнні. Їх використання може полегшити

працю розробників. Масштабованість обраних платформ дозволяє в майбутньому покращити проект і розширити функціонал.

Частина технологій, які пов’язані з функціонуванням веб-додатку розповсюджуються безкоштовно, що дозволить зменшити вартість розробки та підтримки продукту. В частині з основним функціоналом інструментів для здобуття функціональної переваги можна використати платні технології, допоки розвиток LLM з відкритим кодом не досягне необхідного рівня.

5.3. Аналіз ринкових можливостей запуску

Перед початком розробки та запуском в експлуатацію стартап-проекту варто провести аналіз ринкових можливостей. В аналіз ринкових можливостей входить пошук потенційних клієнтів, можливостей та загроз. Також потрібно оцінити конкурентоспроможність стартапу в порівнянні з головними гравцями на ринку. У таблиці 5.3 показані основні характеристики потенційного ринку.

Таблиця 5.3

Загальна характеристика потенційного ринку проєкту

№ п/п	Показники стану ринку	Характеристика
1.	Кількість головних гравців	4
2.	Обсяг продажів	Незначний
3.	Динаміка ринку (якісна оцінка)	Зростає
4.	Наявність обмежень для входу	Наявність початкового капіталу, необхідного для розробки та ведення активного маркетингу
5.	Специфічні вимоги до стандартизації та сертифікації	Формально відсутні

Для того, щоб визначити, чи користуватиметься проект попитом на ринку, необхідно здійснити пошук клієнтів, сформувати з них групи цільової аудиторії. Загальні характеристики потенційних клієнтів надані в таблиці 5.4.

Таблиця 5.4

Характеристика потенційних клієнтів стартап-проєкту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
Боротьба з інформаційним перевантаженням	Приватні користувачі різних вікових та демографічних категорій.	Відмінність в цілях використання	Простота налаштування та використання, великий вибір відтримуваних джерел
Ефективна обробка інформації	Приватні компанії, державні установи	Особливості взаємодії з державними органами та установами	Надійність роботи при великих обсягах інформації, функціональність

Відштовхуючись від основних завдань стартап-проекту, від характеристик ринкового середовища, а також від аналізу цільових груп потенційних клієнтів можна визначити основні фактори загроз (таблиця 5.5).

Таблиця 5.5

Фактори загроз

Фактор	Зміст загрози	Можлива реакція компанії
Конкуренція	Деякі організації вже мають схожі проекти, що можуть завадити виходу стартапу на ринок.	Запуск маркетингової програми, покращення якості за рахунок тіснішої співпраці з клієнтами, новий функціонал.
Відсутність інвестицій	Відсутність інвестицій не дасть змогу вкластися в розробку, розгортання та розвиток стартапу.	Укладання контракту на розробку з клієнтом або пошук додаткових джерел фінансування.
Ціна розробки та розгортання системи	Для створення системи необхідний певний обсяг фінансів, пропорційний до складності розробки та розгортання	Залучення додаткових коштів, або використання інших технологій з метою зменшення складності розробки і розгортання.

Окрім факторів загроз, аналіз ринкового середовища дає можливість оцінити фактори можливостей. Фактори можливостей – це потенційні сприятливі умови ринку.

Необхідно заздалегідь оцінити можливість виникнення сприятливої події та розробити стратегію дій компанії. Фактори можливостей, що мають велике значення для цього стартапу, показані в таблиці 5.6.

Таблиця 5.6

Фактори можливостей

Фактор	Зміст можливості	Можлива реакція компанії
Залучення інвестицій	Отримання додаткових коштів на розробку.	Виділення додаткових фінансів на пришвидшення розробки або покращення якості.
Підписання контракту із замовником	Тісна співпраця із замовником для додаткового фінансування і більшої відповідності системи його вимогам.	Призначення бізнес-аналітика для зв'язку з замовником, залучення додаткових коштів для розробки додаткового функціоналу.
Інтеграція в існуючу компанію	Інтеграція стартап-проекту в програмну систему великої компанії.	Залучення ресурсів компанії для розвитку стартапу в тісній інтеграції з існуючими системами.

Аналіз ринкового середовища показав, що на цьому ринку наявна певна конкуренція. Вибір систем, подібних тій, що розробляється в цьому стартап-проєкті, досить поширений.

На ринку є клієнти, які можливо, захочуть скористатися функціоналом такої системи. При виході на ринок слід враховувати як фактори можливостей, так і фактори загроз і мати відповідний план розвитку компанії.

Для того, щоб зацікавити клієнта, стартап-проект повинен бути конкурентоспроможним, тобто бути в очах користувача принаймні, не менш привабливим, ніж конкуренти.

Одним зі способів підвищення конкурентоспроможності є технології, використані для розробки. Іншими факторами конкурентоспроможності є доступні джерела інформації, можливість використовувати користувацькі розширення, зручність використання системи, привабливість інтерфейсу, орієнтація на потреби користувача. Фактори конкурентоспроможності показані в таблиці 5.7.

Таблиця 5.7

Обґрунтування факторів конкурентоспроможності

Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор значущим)
Різноманіття джерел інформації	Більше різноманіття джерел, доступних за замовчуванням спонукатиме користувачів переходити з інших продуктів.
Можливість використання користувацьких розширень	Дозволяє користувачам самостійно закривати свої потреби в додатковому функціоналі, що сильно підвищує конкурентоздатність.
Зручність використання системи	Зручність полегшує роботу користувача з системою.
Мінімалістичність інтерфейсу	Простота інтерфейсу також полегшує роботу користувача з системою. Зрозумілість інтерфейсу дозволяє користувачу одразу розуміти послідовність дій в роботі з системою, що підвищує ефективність роботи.
Орієнтація на потреби користувача	Дозволить краще їх задовольнити. Гнучким системам вдається впоратися з цим краще.

Таблиця 5.8 допоможе визначити сильніші та слабші сторони стартап-проекту в порівнянні з основними конкурентами на ринку

Таблиця 5.8

Порівняльний аналіз сильних та слабких сторін стартап-проекту

Критерій конкурентно-спроможності	Оцінка 1-30	Оцінка конкурентів в порівнянні зі стартап-проектом						
		-3	-2	-1	0	+1	+2	+3
Різноманіття джерел інформації	21		+					
Можливість використання користувацьких розширень	30	+						
Зручність використання системи	28					+		
Привабливість інтерфейсу	25						+	
Орієнтація на потреби користувача	29				+			

Для завершення аналізу ринкових можливостей можна підбити підсумки за допомогою SWOT-таблиці. SWOT-аналіз (Strengths, Weaknesses, Opportunities, Threats) — це спосіб аналізу середовища, що з’явився в 1963 році.

В ході SWOT-аналізу складається таблиця, що позначає внутрішні особливості проекту як сильні сторони (Strengths) та слабкі сторони (Weaknesses). Особливості зовнішнього середовища, що можуть допомогти в реалізації, називаються можливостями (Opportunities), а ті, що можуть завадити — загрозами (Threats).

Для проведення SWOT-аналізу важливо систематизувати виявлені сильні та слабкі сторони, можливості та загрози шляхом їх внесення в окрему таблицю. У цьому аналітичному інструменті таблиця SWOT використовується для структуризації внутрішніх і зовнішніх факторів, які можуть впливати на організацію чи проект.

SWOT-таблиця стартап-проекту зображена у таблиці 5.9.

Таблиця 5.9

SWOT-таблиця стартап-проекту

Сильні сторони: ▪ Різноманіття джерел інформації; ▪ Користувацькі розширення; ▪ Простота інтерфейсу; ▪ Орієнтація на потреби користувача.	Слабкі сторони: ▪ Необхідність знайти фінансування для запуску; ▪ Невідомість серед користувачів
Можливості: ▪ Залучення інвестицій; ▪ Підписання контракту із замовником; ▪ Інтеграція в існуючу компанію.	Загрози: ▪ Конкуренція з існуючими рішеннями.

5.4. Ринкова стратегія

Проведений аналіз ринку розкрив потреби користувачів, рівень конкурентів, потенційних клієнтів. Були визначені сильні та слабкі сторони стартап-проекту в порівнянні з конкурентами, можливості та загрози ринкового середовища. На основі аналізу ринку можна будувати ринкову стратегію компанії.

Потенційними клієнтами проекту є активні користувачі мережі інтернет, приватні підприємства чи державні установи, яким необхідні інструменти більш

ефективної роботи з інформаційними потоками. В ринкових умовах, що склалися, доцільним буде використання однієї з трьох ринкових стратегій.

Це самостійне створення системи та продаж доступу до неї, створення системи на замовлення якогось із потенційних бізнес-партнерів, або привернення уваги великої IT-компанії та створення системи під її керівництвом та із використанням її коштів.

Складність різних етапів реалізації цих стратегій гарно проілюстровано в таблиці 5.10.

Таблиця 5.10

Можливі ринкові стратегії стартап-проекту

Стратегія	Легкість виходу на ринок	Конкуренція	Фінансування	Можливості для розвитку
Самостійний вихід на ринок	Складно	Середня	За рахунок стартового капіталу	Високі
Розробка системи для замовника	Середня складність	Низька	За рахунок замовника	Низькі
Інтеграція з компанією для спільної розробки	Легко	Середня	За рахунок компанії	Середні

5.5. Команда для розробки стартап-проєкту

Для визначення найефективнішого складу команди для успішної реалізації проекту спершу важливо окреслити ключові етапи роботи та ретельно відібрати кваліфікованих фахівців для кожного з них. Основні етапи роботи, а також відповідні спеціалісти наведені в таблиці 5.11.

Таблиця 5.11

Етапи розробки проєкту та необхідні спеціалісти

Етап розробки	Спеціаліст
Дослідження ринкової ситуації та пошук інвесторів	Менеджер, бізнес-аналітик
Створення компанії та юридичне забезпечення	Менеджер
Узгодження технічного завдання	Менеджер, бізнес-аналітик
Створення, налаштування та підтримка середовища для розробки	DevOps
Розробка серверної частини веб-сайту	Розробник back end
Розробка AI інструментів	NLP/AI спеціаліст
Розробка клієнтської частини веб-сайту та інтерфейсу користувача	Розробник front end
Тестування	Розробник, тестувальник
Розгортання та налаштування системи на сервері	DevOps
Супровід в ході експлуатації	DevOps, розробник
Взаємодія з клієнтами та аналіз результатів	Менеджер

ВИСНОВКИ ДО РОЗДІЛУ 5

Було проведено ретельний технічний аудит проекту, що дозволило ідентифікувати його сильні та слабкі сторони, а також оцінити ринкову ситуацію. Дослідження ринкового середовища показало, що існують значні можливості виділитися на фоні конкурентів, використовуючи ШІ. Було зазначено, що на ринку є потенційні клієнти, які можуть зацікавитися функціоналом розглядуваної системи.

При виході на ринок важливо уважно оцінити як можливості, так і потенційні загрози, та розробити відповідний стратегічний план для подальшого розвитку компанії. Виділено ключові фактори успіху, що включають реалізацію можливостей та вирішення проблем.

Результати аналізу підтвердили реальну можливість успішного запуску стартапу на ринку. Для оптимізації реалізації проекту було розроблено чітку стратегію розвитку та сформовано ефективну і компетентну команду розробників.

ЗАГАЛЬНІ ВИСНОВКИ ДО РОБОТИ

У даній роботі детально розглянуто проблему інформаційного перевантаження та засоби її вирішення. Аналіз причин виникнення та наслідків впливу показав необхідність впровадження ефективних методів боротьби з інформаційним перевантаженням. Було оглянуто сучасні рекомендаційні системи на платформах, таких як Facebook і Twitter, та оглянуто основні завдання обробки природної мови (NLP).

Другий розділ зосереджений на великих мовних моделях (LLM). Розглянуто їхній внутрішній устрій, сучасні підходи до використання та налаштування, включаючи Zero-shot/Few-shot Prompting, Chain-of-Thought та RAG.

Третій розділ присвячено визначенню функціональних та нефункціональних вимог до продукту. Формалізація вимог дозволила чітко структурувати процес розробки та вибрати відповідні технології, зокрема Python, Django та LangChain для прискорення розробки MVP та інтеграції з великими мовними моделями.

Четвертий розділ описує ключові етапи розробки проєкту: налаштування середовища, створення структури веб-ресурсу, моделей та макетів інтерфейсу. Впроваджено інструменти для фільтрації та моніторингу контенту з використанням великих мовних моделей, що дозволяє ефективно управляти інформаційними потоками.

П'ятий розділ присвячено стартап проєкту та технічному аудиту проєкту та аналізу ринкової ситуації. Оцінено конкурентне середовище та ідентифіковано потенційних клієнтів. Аналіз підтвердив реальну можливість успішного запуску стартапу та створення ефективної команди розробників.

Загалом, виконана робота заклала міцну основу для реалізації інноваційного продукту, що сприятиме ефективному управлінню інформаційними потоками та задоволенню потреб користувачів в умовах сучасного інформаційного середовища.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. D. Bawden i L. Robinson Information Overload: An Overview / , Oxford: Oxford University Press, 2020.
2. M. Arnold, M. Goldschmitt, i T. Rigotti Dealing with information overload: a comprehensive review / M. Arnold, M. Goldschmitt, i T. Rigotti. - Front. Psychol, 2023
3. Meta cuts off third-party access to Facebook Groups, leaving developers and customers in disarray // TechCrunch [Електронний ресурс] – URL: https://techcrunch.com/2024/02/05/meta-cuts-off-third-party-access-to-facebook-groups-leaving-developers-and-customers-in-disarray/
4. W. Davis Reddit has a new AI training deal to sell user content [Електронний ресурс] // The Verge – URL: https://www.theverge.com/2024/2/17/24075670/reddit-ai-training-license-deal-user-content.
5. H. Ko, S. Lee, Y. Park, i A. Choi A Survey of Recommendation Systems: Recommendation Models, Techniques, and Application Fields / H. Ko, S. Lee, Y. Park, i A. Choi. – Electronics, 2023
6. W. Fan, Z. Zhao, J. Li, Y. Liu, X. Mei, Y. Wang, Z. Wen, F. Wang, X. Zhao, J. Tang, i Q. Li Recommender Systems in the Era of Large Language Models (LLMs) / . arXiv, 2023.
7. Natural Language Processing (NLP) - A Complete Guide [Електронний ресурс] // DeepLearning.ai – URL: https://www.deeplearning.ai/resources/natural-language-processing/
8. How LLM Works and How LLM is built Guide [Електронний ресурс] // Toloka.ai – URL: https://toloka.ai/blog/how-llm-works/.
9. LLM Settings [Електронний ресурс] // Nextra – URL: https://www.promptingguide.ai/introduction/settings.
10. Formalizing Prompts [Електронний ресурс] // LearnPrompting – URL: https://learnprompting.org/docs/basics/formalizing.

11. OpenAI Platform [Електронний ресурс] // OpenAI – URL: https://platform.openai.com.
12. M. Oleszak Zero-Shot and Few-Shot Learning with LLMs Guide [Електронний ресурс] // neptune.ai – URL: https://neptune.ai/blog/zero-shot-and-few-shot-learning-with-llms.
13. Chain-of-Thought Prompting: Helping LLMs Learn by Example [Електронний ресурс] // Deepgram. – URL: https://deepgram.com/learn/chain-of-thought-prompting-guide
14. What is RAG? - Retrieval-Augmented Generation Explained [Електронний ресурс] // AWS . – URL: https://aws.amazon.com/what-is/retrieval-augmented-generation/.
15. What Is the Difference Between Front-End and Back-End Development? [Електронний ресурс] // Concepta. – URL: https://www.conceptatech.com/blog/difference-front-end-back-end-development
16. Things to know about web frameworks [Електронний ресурс] // Softformance – URL: https://www.softformance.com/blog/things-to-know-about-web-frameworks/
17. Spring Boot [Електронний ресурс] – URL: https://spring.io/projects/spring-boot
18. ASP.NET – Pros and Cons [Електронний ресурс] // PopArt. – URL: https://www.popwebdesign.net/popart_blog/en/2020/03/asp-net-pros-and-cons-you-ought-to-know/
19. Laravel [Електронний ресурс] – URL: https://laravel.com/
20. Чому я обираю FastAPI [Електронний ресурс] // DOU. – URL: https://dou.ua/forums/topic/37547/
21. What is Django? Advantages and Disadvantages [Електронний ресурс] // Hackr.io. – URL: https://hackr.io/blog/what-is-django-advantages-and-disadvantages-of-using-django
22. Python Flask: pros and cons [Електронний ресурс] // Dev. – URL: https://dev.to/detimo/python-flask-pros-and-cons-1mlo

23. Comparison of web template engines [Електронний ресурс] // Wikiwand. – URL: https://www.wikiwand.com/en/Comparison_of_web_template_engines
24. Most Popular Databases [Електронний ресурс] // Statistics & Data. – URL: https://statisticsanddata.org/data/most-popular-databases-2006-2022/
25. OpenAI Pricing [Електронний ресурс] // OpenAI – URL: https://openai.com/api/pricing/
26. Gemini API Pricing [Електронний ресурс] // Google for Developers– URL: https://ai.google.dev/pricing
27. Claude [Електронний ресурс] // Anthropic – URL: https://www.anthropic.com/claude
28. Llama.cpp Tutorial: A Complete Guide to Efficient LLM Inference and Implementation [Електронний ресурс] // Llama.cpp – URL: https://www.datacamp.com/tutorial/llama-cpp-tutorial
29. Ollama [Електронний ресурс] // Ollama – URL: https://ollama.com.
30. LangChain [Електронний ресурс] // LangChain – URL: https://www.langchain.com/.
31. LlamaIndex [Електронний ресурс] // LlamaIndex – URL: https://docs.llamaindex.ai/en/stable/.
32. Daniel Roy Greenfeld. Two Scoops of Django 3.x: Best Practices for the Django Web Framework / Daniel Roy Greenfeld, Audrey Roy Greenfeld. – USA: Feldroy Press, 2022.

ДОДАТОК А
ЛІСТИНГ ФРАГМЕНТІВ КОДУ ПРОГРАМИ

requirments.txt

```
APScheduler==3.10.4
asgiref==3.8.1
beautifulsoup4==4.12.3
certifi==2024.6.2
charset-normalizer==3.3.2
Django==5.0.6
django-apscheduler==0.6.2
feedgen==1.0.0
idna==3.7
lxml==5.2.2
python-dateutil==2.9.0.post0
pytz==2024.1
requests==2.32.3
six==1.16.0
soupsieve==2.5
sqlparse==0.5.0
telegram2rss==0.1.1
tzlocal==5.2
urllib3==2.2.1
```

aicontentaggregator\settings.py

```
"""
Django settings for aicontentaggregator project.
Generated by 'django-admin startproject' using Django 5.0.6.
For more information on this file, see
https://docs.djangoproject.com/en/5.0/topics/settings/
For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.0/ref/settings/
"""
from pathlib import Path
import os
# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = str(os.getenv('DJANGO_SECRET_KEY'))
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
ALLOWED_HOSTS = []
```

```
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'accounts.apps.AccountsConfig',
    'aggregator.apps.AggregatorConfig',
    'feed.apps.FeedConfig',
    "django_apscheduler",
]
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
ROOT_URLCONF = 'aicontentaggregator.urls'
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
WSGI_APPLICATION = 'aicontentaggregator.wsgi.application'
# Database
# https://docs.djangoproject.com/en/5.0/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
# Password validation
# https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators
AUTH_PASSWORD_VALIDATORS = [
```

```
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
# Internationalization
# https://docs.djangoproject.com/en/5.0/topics/i18n/
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True

USE_TZ = True
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.0/howto/static-files/
STATIC_URL = 'static/'
# Default primary key field type
# https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
LOGGING = {
    "version": 1,
    "disable_existing_loggers": False,
    "handlers": {
        "console": {
            "class": "logging.StreamHandler",
        },
    },
    "root": {
        "handlers": ["console"],
        "level": "INFO",
    },
}
```

aicontentaggregator\urls.py

```
"""
URL configuration for aicontentaggregator project.
The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/5.0/topics/http/urls/
Examples:
Function views
```

```
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('accounts.urls')),
    path('', include('aggregator.urls')),
    path('', include('feed.urls')),
]
```

accounts\urls.py

```
from django.urls import path

from . import views

urlpatterns = [
    path('login/', views.log_in, name='login'),
    path('register/', views.register, name='register'),
    path('logout/', views.log_out, name='logout'),
    path('profile/', views.ProfileView.as_view(), name='profile'),
    path('/profile/edit', views.login, name='profile_edit'),
]
```

accounts\views.py

```
from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login, logout
from django.contrib import messages
from django.views import generic, View
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib.auth.models import User
from .decorators import logged_out
```

```
@logged_out
def log_in(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('/sources')
        else:
            messages.info(request, 'Incorrect login data')
    return render(request, 'login.html')

def log_out(request):
    logout(request)
    return redirect('login')

@logged_out
def register(request):
    form = RegistrationForm()
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, 'Registration successful!')
            return redirect('login')
    context = {'form': form}
    return render(request, 'register.html', context)

class ProfileView(LoginRequiredMixin, View):
    def get(self, request):
        context = {'user': request.user}
        return render(request, 'profile.html', context=context)
, views.login, name='profile_edit'),
]
```

accounts\urls.py

```
from django.urls import path

from . import views

urlpatterns = [
    path('login/', views.log_in, name='login'),
    path('register/', views.register, name='register'),
    path('logout/', views.log_out, name='logout'),
    path('profile/', views.ProfileView.as_view(), name='profile'),
    path('/profile/edit'
```

accounts\decorators.py

```
from django.shortcuts import redirect


def logged_out(view_func):
    def wrapper_func(request, *args, **kwargs):
        if request.user.is_authenticated:
            return redirect('trips')
        else:
            return view_func(request, *args, **kwargs)
    return wrapper_func
```

templates\login.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link rel="stylesheet" href="{% static 'css/login.css' %}">
        <title>Login Page</title>
    </head>
    <body>
    <div class="wrap-login">
        <section class="form">
            <h2>Log in to your Account</h2>
            <form class="loginbox" method="POST" action="" autocomplete="off">
                {% csrf_token %}
                <input placeholder="Username" name="username" type="text"
id="username" required>
                <input placeholder="Password" name="password" type="password"
id="password" required>
                <input type="submit" value="Login" id="submit">
            </form>
            {% for message in messages %}
                <div id="messages">{{message}}</div>
            {% endfor %}
            <p>Don't have an account? <a href="{% url 'register' %}">Sign
Up</a></p>
        </section>
    </div>
    </body>
</html>
```

templates\register.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link rel="stylesheet" href="{% static 'css/register.css' %}">
        <title>Registration Page</title>
    </head>
    <body>
    <div class="wrap-reg">
        <section class="form">
            <h2>Register your Account</h2>
            <form class="regbox" method="POST" action="">
                {% csrf_token %}
                {{ form.email }}
                <div id="name-surname">
                    {{ form.first_name }}
                    {{ form.last_name }}
                </div>
                {{ form.username }}
                {{ form.password1 }}
                {{ form.password2 }}
                <input type="submit" value="Register" id="submit-reg">
            </form>

            {% for field, error in form.errors.items %}
            <div>
                {{ error|striptags }}
            </div>
            {% endfor %}
            <p>Already signed up? <a href="{% url 'login' %}">Log In</a></p>
        </section>
    </div>
    <script>
        var form_fields = document.getElementsByTagName('input')
        form_fields[1].placeholder = 'Email Address..'
        form_fields[2].placeholder = 'Name..'
        form_fields[3].placeholder = 'Surname..'
        form_fields[4].placeholder = 'Username..'
        form_fields[5].placeholder = 'Password..'
        form_fields[6].placeholder = 'Repeat Password..'
    </script>
    </body>
</html>
```

aggregator\models.py

```
from django.db import models
from django.urls import reverse

class Source(models.Model):
    title = models.CharField(max_length=100)
    channel_id = models.CharField(max_length=100, blank=False, unique=True)
    description = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def get_absolute_url(self):
        return reverse("source_detail", kwargs={"pk": self.pk})

    def __str__(self):
        return f"{self.title} (@{self.channel_id})"

class Post(models.Model):
    content = models.TextField()
    source = models.ForeignKey(Source, on_delete=models.CASCADE,
related_name="posts")
    date = models.DateTimeField()
    post_id = models.IntegerField(unique=True, primary_key=True)

    is_processed = models.BooleanField(default=False)

    def __str__(self):
        return f"{self.source}/{self.post_id} {self.content}"
```

aggregator \views.py

```
from django.views.generic import ListView, DetailView, CreateView, UpdateView,
DeleteView
from django.contrib.auth.mixins import LoginRequiredMixin

from .models import Source


class SourceListView(LoginRequiredMixin, ListView):
    template_name = "sources.html"
    model = Source

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
```

```
        context["sources"] = Source.objects.filter().order_by("title")
        return context

class SourceDetailView(LoginRequiredMixin, DetailView):
    template_name = "source.html"
    model = Source

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["posts"] = self.object.posts.all().order_by("date")
        return context

class SourcesEditView(LoginRequiredMixin, UpdateView):
     template_name = "edit_source.html"
     model = Source
     fields = ["title", "channel_id", "description"]

class SourcesDeleteView(LoginRequiredMixin, DeleteView):
     template_name = "delete_source.html"
     model = Source
     success_url = "/sources/"

class SourcesCreateView(LoginRequiredMixin, CreateView):
     template_name = "create_source.html"
     model = Source
     fields = ["title", "channel_id", "description"]
```

aggregator\urls.py

```
from django.urls import path

from .views import SourceListView, SourceDetailView

urlpatterns = [
    path("", SourceListView.as_view(), name="sources"),
    path("sources/", SourceListView.as_view(), name="sources"),
    path("sources/<int:pk>/", SourceDetailView.as_view(), name="source_detail"),
    path("sources/<int:pk>/edit/", SourcesEditView.as_view(), name="edit_source"),
    path("sources/<int:pk>/delete/", SourcesDeleteView.as_view(),
name="delete_source"),
]
```

feeds\models.py

```
from django.views.generic import ListView, DetailView, CreateView, UpdateView,
DeleteView
from django.contrib.auth.mixins import LoginRequiredMixin

from .models import Feed, FeedItem, FeedSource


class FeedListView(LoginRequiredMixin, ListView):
    template_name = "feeds.html"
    model = Feed

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["feeds"] = Feed.objects.all().order_by("title")
        return context


class FeedDetailView(LoginRequiredMixin, DetailView):
    template_name = "feed.html"
    model = Feed

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["feed_items"] = self.object.feeditem_set.all()  # or
FeedItem.objects.filter(feed=self.object)
        return context


class FeedCreateView(LoginRequiredMixin, CreateView):
    template_name = "create_feed.html"
    model = Feed
    fields = ["title", "description"]


class FeedUpdateView(LoginRequiredMixin, UpdateView):
    template_name = "edit_feed.html"
    model = Feed
    fields = ["title", "description"]


class FeedDeleteView(LoginRequiredMixin, DeleteView):
    template_name = "delete_feed.html"
    model = Feed
    success_url = "/feeds/"
```

feeds \views.py

```
from django.views.generic import ListView, DetailView, CreateView, UpdateView,
DeleteView
from django.contrib.auth.mixins import LoginRequiredMixin

from .models import Feed


class FeedListView(LoginRequiredMixin, ListView):
    template_name = "sources.html"
    model = Feed

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["sources"] = Source.objects.filter().order_by("title")
        return context

class FeedDetailView(LoginRequiredMixin, DetailView):
    template_name = "source.html"
    model = Source

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["posts"] = self.object.posts.all().order_by("date")
        return context

class FeedEditView(LoginRequiredMixin, UpdateView):
     template_name = "edit_source.html"
     model = Feed
     fields = ["title", "channel_id", "description"]

class FeedDeleteView(LoginRequiredMixin, DeleteView):
     template_name = "delete_source.html"
     model = Feed
     success_url = "/sources/"

class FeedCreateView(LoginRequiredMixin, CreateView):
     template_name = "create_source.html"
     model = Feed
     fields = ["title", "channel_id", "description"]
```

feeds\urls.py

```
from django.urls import path

from .views import FeedListView, FeedDetailView

urlpatterns = [
    path("", SourceListView.as_view(), name="sources"),
    path("sources/", FeedListView.as_view(), name="sources"),
    path("sources/<int:pk>/", FeedDetailView.as_view(), name="source_detail"),
    path("sources/<int:pk>/edit/", FeedEditView.as_view(), name="edit_source"),
    path("sources/<int:pk>/delete/", FeedDeleteView.as_view(),
name="delete_source"),
]
```

feeds\ instruments.py

```
from langchain.llms import OpenAI
from langchain.prompts import ChatPromptTemplate
from langchain.chains import LLMChain

def filter_content(posts, user_criteria):
    # Initialize the LLM and create the prompt
    llm = OpenAI(model_name="gpt-4")
    prompt_template = ChatPromptTemplate.from_messages([
        ("system", "You are a content filtering assistant."),
        ("user", "Given the criteria: {criteria}, decide whether to show or hide
each post: {post}")
    ])
    chain = LLMChain(llm=llm, prompt=prompt_template)

    filtered_posts = []
    for post in posts:
        response = chain.invoke({"criteria": user_criteria, "post": post})
        if "show" in response:
            filtered_posts.append(post)

    return filtered_posts

def monitor_content(posts, user_criteria):
    llm = OpenAI(model_name="gpt-4")
    prompt_template = ChatPromptTemplate.from_messages([
        ("system", "You are a content monitoring assistant."),
        ("user", "Given the criteria: {criteria}, determine if this post is
important: {post}")
    ])
    chain = LLMChain(llm=llm, prompt=prompt_template)
```

```
    alerts = []
    for post in posts:
        response = chain.invoke({"criteria": user_criteria, "post": post})
        if "important" in response:
            alerts.append(post)

    return alerts


def summarize_content(posts, user_description):
    llm = OpenAI(model_name="gpt-4")
    prompt_template = ChatPromptTemplate.from_messages([
        ("system", "You are a content summarization assistant."),
        ("user", "Given the description: {description}, summarize the following
posts: {posts}")
    ])
    chain = LLMChain(llm=llm, prompt=prompt_template)

    summary = chain.invoke({"description": user_description, "posts": posts})
    return summary
```

templates\base.html

```
{% load static %}
<html>
    <head>
        {% block title %}<title>AIA</title>{% endblock %}
        <link rel="stylesheet" href="{% static 'css/base.css' %}">
    </head>
    <body>
        <header>
            <div class="navbar">
                <a class="navbar-elem" href="{% url 'sources' %}">SOURCES</a>
                <a class="navbar-elem" href="{% url 'sources' %}">FEEDS</a>

                <div class="navbar-elem account">
                    <a class="navbar-elem" href="">Logout</a>
                    <a class="navbar-elem" href="">Profile</a>
                </div>
            </div>
        </header>
        <div class="main">
            {% block content %}{% endblock %}
        </div>
        <footer>
        </footer>
    </body>
</html>
```

templates\source.html

```
{% extends "base.html" %}
{% block title %}<title>{{source.title}}</title>{% endblock %}
{% block content %}
<div class="create">
  <h1>{{source.title}}</h1>
  <div class="create-trip">
    <a href=""><img src="https://www.svgrepo.com/show/59773/plus-square-button.svg"
alt="Add source"></a>
  </div>
</div>
<table>
  <tr>
    <th>Title</th>
    <th>channel_id</th>
    <th>description</th>
    <th>edit</th>
    <th>delete</th>
  </tr>
  <tr>
    <td>{{source.title}}</td>
    <td>{{source.channel_id}}</td>
    <td>{{source.description}}</td>

    <td><a href="">edit</a></td>
    <td><a href="">delete</a></td>
  </tr>
</table>
{% if posts %}
<table>
  <tr>
    <th>date</th>
    <th>content</th>
    <!-- <th>description</th> -->
  </tr>
  {% for post in posts %}
  <tr>
    <td>{{post.date}}</td>
    <td>{{post.content}}</td>
    <!-- <td>{{source.description}}</td> -->
  </tr>
  {% endfor %}
</table>

{% else %}
  <p>There are no posts</p>
{% endif %}
{% endblock %}
```