ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
(повне найменування вищого навчального закладу)
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
(повне найменування інституту, назва факультету (відділення))
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до кваліфікаційної роботи
магістра
(освітній рівень)

на тему: «Розробка веб-додатку для ведення блогу з адаптивним дизайном»

Виконав: студент 6 курсу, групи 6ПР2
спеціальності
121 - «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

Плутенко Данило Валерійович
(прізвище та ініціали)
Керівник к.т.н., доцент Козуб Н. О.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)

Хмельницький - 2025

Херсонський національний технічний університет
(повне найменування вищого навчального закладу)

Факультет, відділення Інформаційних технологій та дизайну
Кафедра Програмних засобів і технологій
Освітній рівень Магістр
Спеціальність 121 – Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувачка кафедри

програмних засобів і технологій

к.т.н., доц. О.Є.Огнєва

“ 15 ” Вересня 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Плутенко Данило Валерійович
(прізвище, ім’я, по батькові)

Тема роботи «Розробка веб-додатку для ведення блогу з адаптивним дизайном»

керівник роботи к.т.н. доцент Козуб Н. О.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від
2. Строк подання студентом роботи
3. Вихідні дані до роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та	Підпис, дата

	посада консультанта	завдання видав	завдання прийняв

7. Дата видачі завдання _____09.09.2025_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання роботи	Термін виконання етапів роботи	Примітки
1.			
2.			
3.			
4.			
5.			
6.			
7.			
8.			
9.			
10.			

Студент _____ Д.В.Плутенко
(підпис) (прізвище та ініціали)

Керівник роботи _____ Н.О.Козуб
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота магістра: 90 сторінок, 16 рисунків, 2 додатків, 29 джерел.

Мета роботи полягає у підвищенні ефективності взаємодії користувача з контентом на веб-сайті для ведення блогу шляхом розробки та реалізації системи, що поєднує адаптивний дизайн інтерфейсу з інтелектуальним модулем персоналізації контенту, що базується на алгоритмах машинного навчання (ML).

Об'єкт дослідження – процес взаємодії користувача з контентом на сучасних блог-платформах та веб-ресурсах.

Предмет дослідження – методи та алгоритми персоналізації контенту, зокрема метод фільтрації на основі вмісту (Content-Based Filtering), а також технології реалізації адаптивного веб-дизайну.

Методи дослідження – аналіз та узагальнення науково-технічної інформації, порівняльний аналіз, методи системного проєктування, методи математичного моделювання (векторні моделі представлення тексту, метрики подібності), програмна реалізація та експериментальне тестування. У роботі проведено комплексний аналіз сучасних контент-платформ, що дозволило виявити ключові тенденції розвитку веб-ресурсів та обґрунтувати актуальність впровадження систем персоналізації. Досліджено основні класи алгоритмів рекомендаційних систем. На основі аналізу для практичної реалізації обрано метод фільтрації на основі вмісту, що дозволяє надавати релевантні рекомендації без необхідності збору великих обсягів даних про поведінку користувачів.

Результат роботи: розроблено програмний продукт – функціональний веб-сайт для ведення блогу, архітектура якого включає інтелектуальний модуль на основі машинного навчання, що реалізує персоналізовані рекомендації. Програмна реалізація поєднує сучасні технології адаптивного веб-дизайну та серверний модуль, що реалізує алгоритм рекомендацій на основі моделі TF-IDF та косинусної подібності.

Новизна роботи: полягає в адаптації та застосуванні методу машинного навчання (фільтрації на основі вмісту) для створення інтегрованого інтелектуального модуля в архітектурі сучасної блог-платформи, що вирішує проблему "холодного старту" та забезпечує персоналізацію контенту на початкових етапах функціонування ресурсу.

Ключові слова: веб-сайт, блог-платформа, адаптивний дизайн, responsive design, персоналізація контенту, рекомендаційна система, система фільтрації інформації, залучення користувачів, user engagement, фільтрація на основі вмісту, Content-Based Filtering, машинне навчання, обробка природної мови, NLP, TF-IDF, векторне представлення тексту, косинусна подібність, проблема холодного старту.

АНОТАЦІЯ

Кваліфікаційна робота магістра складається зі вступу, трьох розділів, загальних висновків, переліку посилань та додатків.

Перший розділ «Аналіз предметної області та методів персоналізації контенту» містить комплексне дослідження сучасних контент-орієнтованих веб-платформ та методів штучного інтелекту, що застосовуються для їх роботи. Проведено детальний огляд та порівняльний аналіз провідних ресурсів з метою виявлення ключових тенденцій в адаптації контенту. У розділі обґрунтовано роль рекомендаційних систем, проведено класифікацію основних алгоритмів машинного навчання (колаборативна фільтрація, фільтрація на основі вмісту, гібридні підходи) та виконано поглиблене дослідження методу фільтрації на основі вмісту.

Другий розділ «Проєктування веб-додатку з інтелектуальним модулем на основі ML» присвячено розробці архітектурних та проектних рішень для системи «Re:Action». У розділі представлено загальну клієнт-серверну архітектуру, розроблено детальний алгоритм роботи модуля машинного навчання з відповідною математичною моделлю на основі TF-IDF, спроєктовано логічну ER-діаграму бази даних. Наприкінці розділу наведено обґрунтування вибору технологічного стеку, що включає Python, Flask та бібліотеку Scikit-learn для реалізації наукомісткого ядра системи.

Третій розділ «Реалізація веб-додатку для ведення блогу «Re:Action»» описує процес програмної розробки та тестування продукту. Розділ охоплює налаштування середовища, реалізацію моделей даних, розробку модуля автентифікації та управління профілями. Детально описано реалізацію основного функціоналу блогу. Окрему увагу приділено розробці та інтеграції модуля машинного навчання для персоналізованих рекомендацій, включаючи механізм зважування сигналів (лайків та переглядів). Наприкінці розділу представлено результати функціонального тестування та проаналізовано проблеми, що виникли під час розробки.

ABSTRACT

The Master's qualification thesis consists of an introduction, three chapters, general conclusions, a list of references, and appendices.

The first chapter, "Analysis of the Subject Area and Content Personalization Methods," contains a comprehensive study of modern content-oriented web platforms and the Artificial Intelligence methods applied to their operation. A detailed review and comparative analysis of leading resources were conducted to identify key trends in content adaptation. The chapter substantiates the role of recommender systems, provides a classification of the main Machine Learning algorithms (collaborative filtering, content-based filtering, hybrid approaches), and offers an in-depth study of the content-based filtering method.

The second chapter, "Design of a Web Application with an ML-based Intelligent Module," is dedicated to the development of architectural and design solutions for the "Re:Action" system. This chapter presents the general client-server architecture, develops a detailed algorithm for the Machine Learning module with its corresponding mathematical model based on TF-IDF, and designs a logical ER diagram for the database. The end of the chapter provides a justification for the choice of the technology stack, including Python, Flask, and the Scikit-learn library for implementing the science-intensive core of the system.

The third chapter, "Implementation of the 'Re:Action' Blogging Web Application," describes the software development and testing process of the product. The chapter covers the environment setup, data model implementation, and the development of the authentication and user profile management module. It details the implementation of the core blog functionality. Special attention is given to the development and integration of the Machine Learning module for personalized recommendations, including a signal weighting mechanism (likes and views). The end of the chapter presents the results of functional testing and analyzes the challenges encountered during development.

Зміст

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface

CRUD – Create, Read, Update, Delete

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

IDE – Integrated Development Environment

JSON – JavaScript Object Notation

ML – Machine Learning

NLP – Natural Language Processing

ORM – Object-Relational Mapping

REST – Representational State Transfer

SQL – Structured Query Language

СУБД – Система управління базами даних

TF-IDF – Term Frequency-Inverse Document Frequency

UI – User Interface

UX – User Experience

WYSIWYG – What You See Is What You Get

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується інтеграцією методів штучного інтелекту (ШІ) у веб-додатки для створення глибоко персоналізованого користувацького досвіду. Успішність таких платформ, зокрема блогів, залежить не лише від якості контенту, але й від здатності системи за допомогою алгоритмів машинного навчання (ML) адаптуватися до індивідуальних інтересів користувача. Поняття «адаптивності» в цьому контексті виходить за межі технічної адаптації інтерфейсу (адаптивний дизайн) і включає в себе інтелектуальну адаптацію контенту, що є ключовим завданням сучасних веб-систем.

Актуальність теми. В умовах експоненційного зростання обсягів інформації виникає проблема інформаційного перевантаження, що ускладнює для користувачів пошук релевантного контенту та призводить до зниження їхньої залученості. Традиційні методи навігації виявляються неефективними, що робить задачу інтелектуальної фільтрації та персоналізованої подачі інформації за допомогою алгоритмів штучного інтелекту надзвичайно актуальною. Розробка веб-сайту, що поєднує сучасні підходи до адаптивного дизайну з інтелектуальними алгоритмами рекомендації контенту, є вирішенням цієї проблеми, спрямованим на значне покращення користувацького досвіду та підвищення конкурентоспроможності веб-ресурсу.

Об’єкт дослідження. Процес взаємодії користувача з контентом на сучасних блог-платформах та веб-ресурсах.

Предмет дослідження. Методи машинного навчання персоналізації контенту для підвищення залученості користувачів, зокрема метод фільтрації на основі вмісту (Content-Based Filtering), а також технології реалізації адаптивного веб-дизайну.

Методи дослідження. Для досягнення поставленої мети в роботі використано комплекс наукових методів: аналіз та узагальнення науково-технічної інформації для вивчення існуючих підходів; порівняльний аналіз для оцінки переваг та недоліків різних платформ та алгоритмів; методи

системного аналізу та проєктування для розробки архітектури системи; методи математичного моделювання, зокрема векторні моделі представлення тексту, для побудови рекомендаційного модуля; програмна реалізація та експериментальне тестування для перевірки працездатності розробленого рішення.

Мета і задачі дослідження. Метою кваліфікаційної роботи є підвищення ефективності взаємодії користувача з контентом на веб-сайті для ведення блогу шляхом розробки та реалізації системи подвійної адаптації: адаптивного дизайну інтерфейсу та інтелектуальної персоналізації контенту.

Основним завданням для досягнення поставленої мети є вирішення наступних задач:

- провести аналіз сучасних контент-платформ та дослідити існуючі алгоритми рекомендаційних систем;
- обґрунтувати вибір методу фільтрації на основі вмісту як найбільш доцільного для реалізації в рамках проєкту;
- розробити загальну архітектуру програмної системи та математичну модель модуля машинного навчання для рекомендацій;
- здійснити програмну реалізацію веб-сайту з адаптивним дизайном та інтегрованим модулем персоналізації;
- провести експериментальну перевірку працездатності розробленої системи.

Наукова новизна одержаних результатів полягає в адаптації та застосуванні методу машинного навчання (фільтрації на основі вмісту) для створення автономного інтелектуального модуля в архітектурі сучасної блог-платформи, що дозволяє надавати персоналізовані рекомендації без необхідності збору великих обсягів даних про поведінку користувачів на початковому етапі функціонування ресурсу.

Практичне значення одержаних результатів. Практична цінність роботи полягає у розробці програмного продукту — функціонального веб-сайту для ведення блогу, який демонструє ефективне поєднання адаптивного дизайну та системи персоналізації контенту. Розроблений рекомендаційний

модуль може бути інтегрований в інші контент-орієнтовані веб-проєкти. Результати дослідження можуть слугувати методичною основою для розробників при створенні подібних інтелектуальних веб-систем.

Структура: робота складається зі вступу, трьох розділів, висновків, списку використаних джерел, додатків. Кваліфікаційна робота магістра складається з 90 сторінок, 16 рисунків, 2 додатків, 29 джерел.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ПЕРСОНАЛІЗАЦІЇ КОНТЕНТУ

1.1. Огляд сучасних блог-платформ та веб-ресурсів

Еволюція всесвітньої мережі трансформувала веб-ресурси зі статичних сховищ інформації в динамічні, інтерактивні екосистеми, орієнтовані на користувача. На сучасному етапі розвитку Інтернету, що характеризується домінуванням так званої «економіки уваги», ключовим фактором конкурентоспроможності будь-якого контент-орієнтованого проєкту є його здатність ефективно керувати увагою користувача. Щодня створюються мільйони одиниць контенту — статті, відео, аудіозаписи, — що призводить до феномену інформаційного перевантаження (information overload), коли обсяг доступної інформації значно перевищує когнітивні можливості людини для її обробки.

В цих умовах традиційні моделі подання контенту, такі як зворотна хронологічна стрічка, втрачають свою ефективність. Вони не враховують індивідуальних інтересів, вподобань та контексту споживання інформації кожного окремого користувача. Як наслідок, значна частина релевантного для користувача контенту залишається непоміченою, що призводить до зниження залученості (user engagement) та відтоку аудиторії.

Вирішенням цієї проблеми стало впровадження систем адаптації, які можна класифікувати за двома основними напрямками:

- Адаптація на рівні інтерфейсу — забезпечення коректного та зручного відображення контенту на будь-якому типі пристроїв (настільні комп'ютери, планшети, смартфони). Ця задача вирішується за допомогою методологій адаптивного та відгукливого веб-дизайну (adaptive & responsive web design), що на сьогодні є галузевим стандартом.
- Адаптація на рівні контенту — інтелектуальна фільтрація та ранжування інформації з метою надання кожному користувачу персоналізованого набору матеріалів, що з найбільшою ймовірністю відповідатимуть його інтересам. Цей напрямок базується на використанні

рекомендаційних систем (recommender systems), що є однією з найважливіших прикладних галузей машинного навчання (machine learning) [23].

Метою даного підрозділу є проведення комплексного аналізу функціональних та архітектурних рішень, що застосовуються на провідних світових веб-платформах для реалізації контентної адаптації. Аналіз буде сфокусований на виявленні ключових механізмів персоналізації, їхніх переваг та недоліків. Для забезпечення системності огляду, розглянуті платформи будуть згруповані за їхнім основним призначенням та моделлю роботи з контентом.

Першою групою для аналізу є платформи, що безпосередньо орієнтовані на створення та споживання текстового та змішаного контенту, оскільки вони є найбільш релевантними до теми даної роботи.

Одним із найбільш репрезентативних прикладів сучасної контент-платформи, що успішно поєднує мінімалістичний адаптивний дизайн з потужною системою персоналізації, є Medium.com[1]. Запущена у 2012 році, ця платформа позиціонується не як інструмент для створення окремих блогів, а як єдина видавнича екосистема, де головним пріоритетом є якість контенту та комфорт читання. На відміну від децентралізованих CMS на кшталт WordPress, Medium є централізованою платформою, де дистрибуція контенту майже повністю контролюється внутрішніми алгоритмами.

Ключовим елементом, що забезпечує високу залученість користувачів, є персоналізована головна стрічка "For You". Вона є динамічним потоком статей, який індивідуально формується для кожного користувача. Фундаментом цієї системи є безперервний збір та аналіз великої кількості сигналів про взаємодію користувача з контентом.

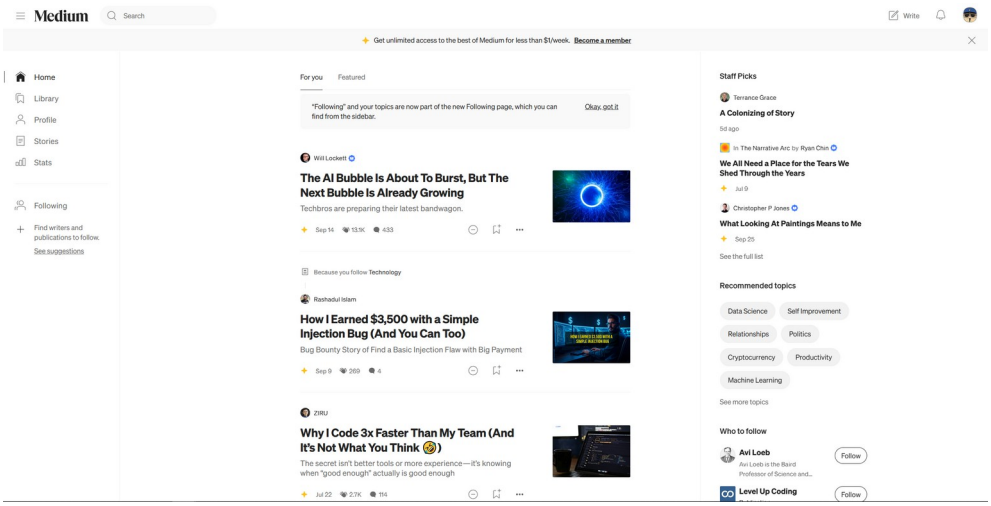


Рис. 1.1 Приклад головної сторінки платформи Medium з персоналізованою стрічкою рекомендацій

Сигнали, що аналізуються алгоритмами платформи, можна умовно розділити на дві великі категорії. Перша категорія — це явні сигнали (explicit signals), що представляють собою свідомі дії користувача, які прямо вказують на його інтереси. До них належать підписка на конкретних авторів або тематичні видання (publications), а також вибір цікавих тем (тегів) під час реєстрації. На додачу до цих прямих вказівок, система надає значної ваги неявним сигналам (implicit signals) — поведінковим метрикам, що опосередковано відображають рівень зацікавленості. Ключовими серед них є загальний час читання статті (reading time), глибина прокрутки сторінки, а також використання фірмової функції "Claps". Ця функція, на відміну від бінарного "лайка", дозволяє оцінити статтю за шкалою від 1 до 50, надаючи алгоритму більш гранульовані та точні дані про ступінь вподобання.

Для обробки цих масивів даних Medium впроваджує складну гібридну модель рекомендацій[28]. Основним її компонентом є фільтрація на основі вмісту (Content-Based Filtering). У рамках цього підходу застосовуються методи обробки природної мови (Natural Language Processing, NLP) для семантичного аналізу кожної нової статті. На основі цього аналізу для публікації створюється числовий вектор, що відображає її тематичну суть. Коли користувач взаємодіє зі статтею на певну тему, система шукає в базі даних інші статті з тематично близькими векторами. Цей контент-

орієнтований аналіз потужно доповнюється колаборативною фільтрацією (Collaborative Filtering). Даний підхід базується на аналізі поведінки великих груп користувачів, знаходячи "цифрових двійників" — користувачів зі схожими смаками. Після цього система може рекомендувати статті, які сподобались цим "двійникам", але які початковий користувач ще не бачив[29].

Проте екосистема Medium не є виключно автоматизованою. Важливу роль в ній відіграє людський фактор у вигляді тематичних видань (publications) та кураторства. Видання функціонують як онлайн-журнали, що дозволяє групувати контент за певною тематикою та підтримувати редакційні стандарти. Водночас команда внутрішніх кураторів платформи вручну відбирає найкращі матеріали ("Staff Picks"), забезпечуючи їм додаткове просування. Це дозволяє підтримувати загальний високий стандарт якості та допомагає талановитим авторам знаходити свою аудиторію.

Таким чином, Medium є прикладом успішної реалізації централізованої контент-платформи, де адаптивність проявляється не лише у відгукливому дизайні, але й у глибокій персоналізації контенту. Система ефективно вирішує проблему інформаційного перевантаження для читача та проблему видимості (discoverability) для автора, що є характерним для сучасних контент-екосистем.

На противагу складній, централізованій екосистемі Medium, платформа Blogger, що належить компанії Google, є представником класичної моделі децентралізованого блогінгу[2]. Запущена у 1999 році, Blogger була однією з перших систем, що зробила ведення особистого онлайн-щоденника доступним для масової аудиторії. Її архітектура та функціональність відображають філософію раннього вебу, де кожен блог є автономним сайтом, а механізми дистрибуції та персоналізації контенту відіграють другорядну роль.

Основною парадигмою Blogger є надання користувачу простого та безкоштовного інструменту для створення та адміністрування власного,

ізольованого блогу. На відміну від Medium, тут відсутня єдина стрічка рекомендацій або глобальна система пошуку контенту всередині платформи. Кожен блог існує як окремий веб-сайт зі своєю унікальною адресою, а його просування та залучення аудиторії повністю покладається на зусилля автора.

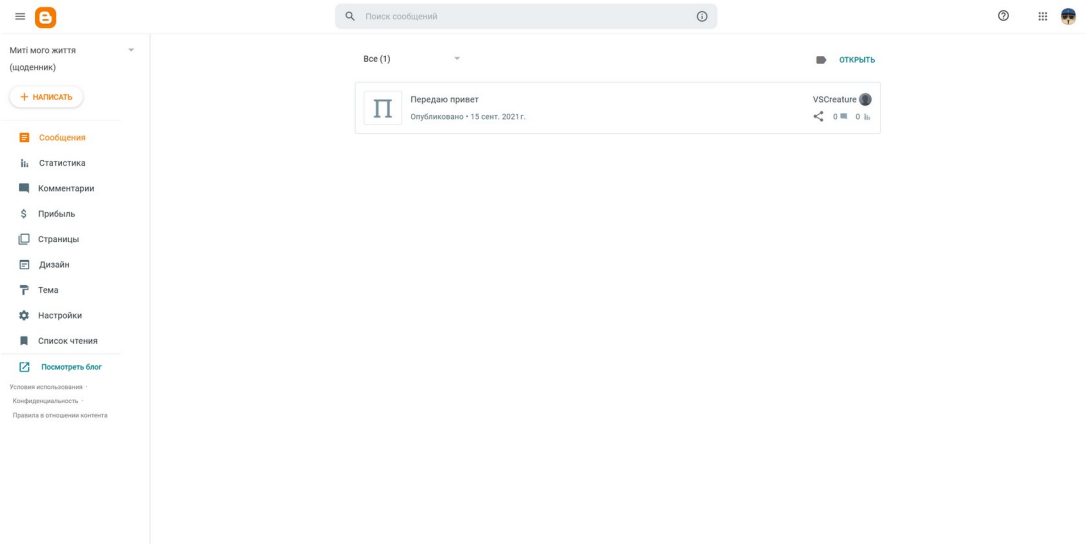


Рис. 1.2 Інтерфейс головної сторінки типового блогу на платформі Blogger

У контексті адаптивного дизайну платформа пропонує набір готових шаблонів, більшість з яких забезпечують коректне відображення на мобільних пристроях. Користувачам доступний візуальний редактор для зміни базових елементів дизайну, а також можливість редагувати HTML/CSS код шаблону, що надає певну гнучкість. Проте, порівняно з більш сучасними системами, ці можливості кастомізації є доволі обмеженими.

Ключовою відмінністю Blogger від сучасних платформ є майже повна відсутність вбудованої персоналізації контенту. Система не аналізує поведінку відвідувачів для надання їм індивідуальних рекомендацій. Контент у блозі зазвичай організовано у зворотному хронологічному порядку, а єдиними інструментами для навігації є архів за датами, категорії та пошук по сайту, які налаштовуються автором вручну. Платформа не пропонує механізмів для рекомендації інших статей, які могли б зацікавити читача на основі його поточної активності.

Оскільки Blogger не має внутрішніх інструментів для просування контенту, автори змушені покладатися на зовнішні канали, в першу чергу на пошукові системи. Глибока інтеграція з екосистемою Google є однією з небагатьох переваг платформи, що дозволяє легко підключити сервіси Google Analytics для відстеження трафіку та Google AdSense для монетизації. Саме через AdSense може реалізовуватися певний рівень опосередкованої персоналізації: рекламні оголошення, що показуються відвідувачам, є персоналізованими на основі їхньої глобальної активності в мережі. Однак цей механізм не впливає на подання основного контенту блогу.

Таким чином, Blogger є яскравим прикладом платформи "першого покоління", що успішно вирішує базову задачу публікації контенту, але ігнорує проблему інформаційного перевантаження та необхідність інтелектуальної дистрибуції. Аналіз Blogger важливий для розуміння еволюційного шляху, який пройшли контент-платформи від простих інструментів до складних систем, орієнтованих на утримання уваги користувача. Цей контраст підкреслює, що в сучасних умовах наявність лише адаптивного дизайну без адаптації контенту є недостатньою для побудови конкурентоспроможного веб-ресурсу.

Платформа Tumblr[3], заснована у 2007 році, є унікальним гібридом, що поєднує елементи традиційного блогінгу та соціальної мережі. На відміну від Blogger, де кожен сайт є ізольованою одиницею, або Medium, що фокусується на довгих текстах, Tumblr створений для швидкого обміну різноманітним контентом: зображеннями, GIF-анімаціями, цитатами, посиланнями, аудіо та короткими текстовими записами. Цей формат отримав назву "мікроблогінг", і саме соціальна взаємодія між користувачами є ядром функціонування платформи та основою для її механізмів персоналізації.

Центральним елементом інтерфейсу Tumblr є єдина, нескінченна стрічка контенту, що агрегує дописи з блогів, на які підписаний користувач. Ця стрічка є персоналізованою за замовчуванням, оскільки її наповнення визначається вибором самого користувача. Проте, на відміну від простої

хронологічної стрічки, алгоритми Tumblr інтегрують у неї додатковий рекомендований контент, щоб стимулювати відкриття нових авторів та тем.

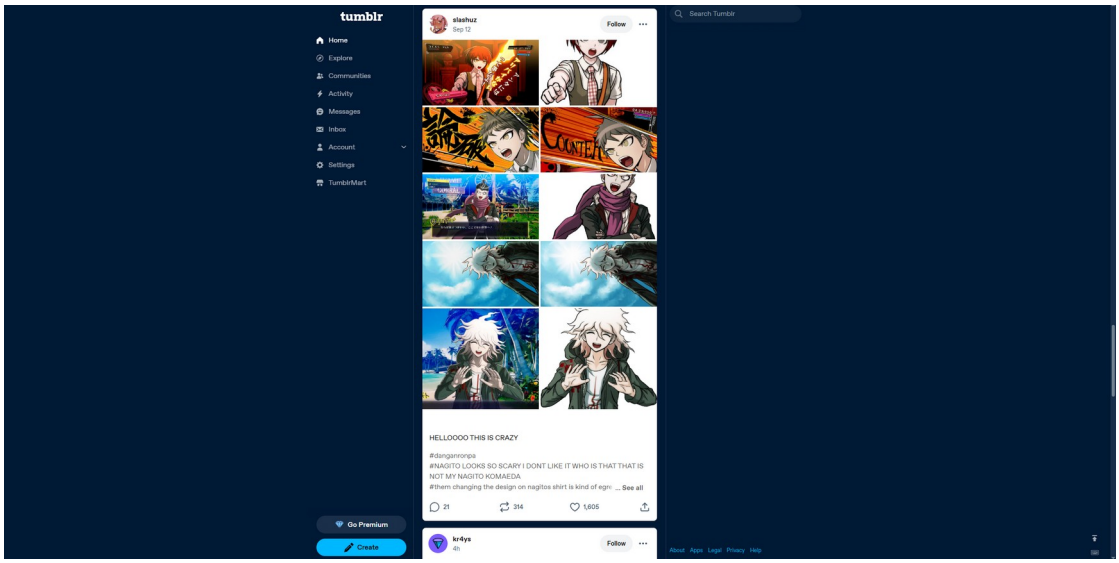


Рис. 1.3 Приклад стрічки контенту на платформі Tumblr

Механізм персоналізації на Tumblr тісно пов'язаний з його ключовими соціальними функціями: "лайком" (like) та "реблогом" (reblog). Коли користувач робить реблог, він фактично копіює чужий допис у свій власний блог, часто додаючи власні коментарі чи теги. Ця дія є значно сильнішим сигналом про зацікавленість, ніж простий лайк, і саме вона лежить в основі вірального поширення контенту на платформі.

Рекомендаційна система Tumblr аналізує ці соціальні сигнали для персоналізації контенту. Якщо користувач часто лайкає або ребложить дописи з певними тегами (наприклад, #art або #anime), алгоритм починає показувати йому в стрічці популярні дописи з цими ж тегами з блогів, на які він ще не підписаний. Крім того, система використовує елементи колаборативної фільтрації: вона аналізує активність інших користувачів зі схожими смаками і може рекомендувати блоги або окремі дописи, які популярні серед цієї групи.

Важливу роль в організації контенту відіграють теги. Користувачі активно додають їх до своїх дописів, що дозволяє не лише класифікувати контент, але й створювати тематичні потоки. Кожен тег має власну сторінку,

де можна переглянути найновіші та найпопулярніші дописи з ним, що є ще одним інструментом для пошуку цікавого контенту поза межами власної стрічки.

Таким чином, Tumblr демонструє модель, де персоналізація контенту базується не стільки на глибокому семантичному аналізі самого контенту (як у Medium), скільки на аналізі соціального графу та поведінкових патернів користувачів. Соціальні дії — підписки, лайки, а особливо реблоги — є основними даними для роботи рекомендаційних алгоритмів. Цей підхід є надзвичайно ефективним для платформ з візуально-орієнтованим та швидко споживаним контентом, де соціальне підтвердження (популярність допису) є головним критерієм його цінності.

Якщо попередньо розглянуті платформи демонстрували різні ступені інтеграції персоналізації, то відеохостинг YouTube[4], що належить компанії Google, є прикладом екосистеми, існування та домінування якої практично повністю побудоване на ефективності її рекомендаційних алгоритмів. З огляду на колосальні обсяги контенту, що завантажується на платформу (понад 500 годин відео щохвилини), ручний пошук та навігація стають неефективними. YouTube здійснив парадигмальний зсув від моделі "пошуку" контенту до моделі його "відкриття" (discovery), де система проактивно пропонує користувачу релевантні відео, часто ще до того, як він сформулював свій запит.

Центральним елементом цієї екосистеми є рекомендаційна система, що базується на сучасних архітектурах штучного інтелекту, зокрема на моделях глибокого навчання (deep learning). Її головна мета — не просто максимізувати кількість кліків, а максимізувати довгострокове задоволення користувача (user satisfaction), ключовою проксі-метрикою для якого є загальний час перегляду (watch time) та тривалість сесії (session duration).

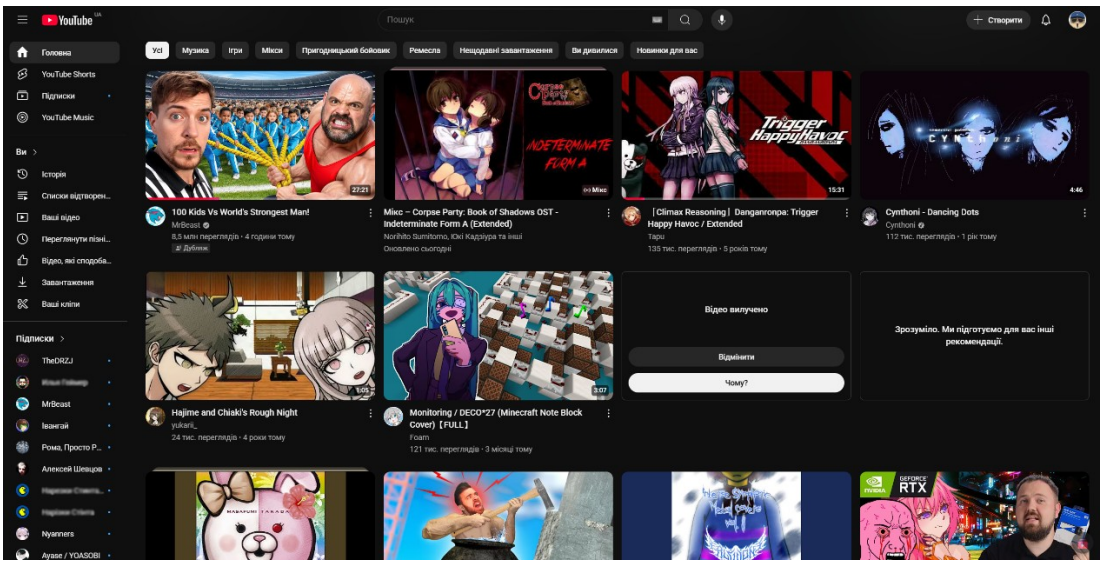


Рис. 1.4 Головна сторінка YouTube, що повністю складається з персоналізованих відео-рекомендацій

Архітектура рекомендаційної системи YouTube є двоступеневою. На першому етапі, що називається генерацією кандидатів (candidate generation), система з мільярдів доступних відео відбирає декілька сотень, які є найбільш релевантними для даного користувача. Цей відбір базується на широкому аналізі історії активності користувача, включаючи його перегляди, пошукові запити та демографічні дані. На цьому етапі активно застосовуються методи колаборативної фільтрації, що дозволяють знаходити відео, популярні серед користувачів зі схожими поведінковими патернами.

На другому, більш тонкому етапі, відбувається ранжування (ranking). Відібрані кілька сотень відео-кандидатів пропускаються через значно складнішу нейронну мережу. Ця мережа для кожного відео прораховує прогнозну оцінку — ймовірність того, що користувач не просто натисне на нього, але й буде дивитися його протягом тривалого часу. Для цього аналізуються сотні сигналів: від персональних вподобань користувача до контекстуальних факторів, таких як час доби чи тип пристрою. Відео з найвищою прогнозною оцінкою займають перші позиції на головній сторінці або у списку "Наступне".

Джерелом даних для цих складних моделей є безперервний потік сигналів від користувачів. Система аналізує не лише явні дії, такі як

вподобання (likes), дизлайки, підписки або натискання кнопки "Не цікавить", але й приділяє значно більшу вагу неявним поведінковим метрикам. До них належать вже згаданий час перегляду, відсоток переглянутого відео, швидкість кліків на мініатюру (Click-Through Rate, CTR), а також кількість поширень відео. Кожна дія користувача миттєво враховується і використовується для оновлення його профілю інтересів, що робить рекомендації динамічними та чутливими до змін у смаках.

Персоналізований контент представлений на всіх ключових поверхнях взаємодії з платформою: на головній сторінці, у бічній панелі "Наступне" (Up Next), що є критично важливою для утримання користувача в рамках однієї сесії, в результатах пошуку, які також ранжуються персонально, та у сповіщеннях.

Таким чином, YouTube демонструє найвищий рівень розвитку алгоритмічної персоналізації. Це замкнена екосистема, де поведінка користувачів генерує дані, які навчають моделі, що, в свою чергу, формують контентне середовище, яке стимулює подальшу поведінку. Аналіз YouTube є важливим, оскільки він показує потенціал та складність сучасних рекомендаційних систем, що базуються на глибокому навчанні. Водночас він піднімає питання про етичні аспекти такої глибокої персоналізації, зокрема проблему "фільтраційних бульбашок" (filter bubbles), коли користувач бачить лише той контент, що підтверджує його існуючі погляди[25].

Платформа Reddit[5], що позиціонує себе як "головна сторінка Інтернету", представляє унікальну модель організації та подання контенту, яка базується на децентралізованій структурі спільнот. На відміну від YouTube чи Medium, де контент пропонується користувачу переважно централізованим алгоритмом, на Reddit персоналізація є результатом свідомих дій самого користувача. Це робить платформу яскравим прикладом системи, керованої користувачем (user-driven personalization), доповненої елементами демократичного ранжування.

Вся архітектура Reddit побудована навколо "сабреддітів" (subreddits) — тисяч автономних тематичних форумів, присвячених практично будь-якій

темі, від глобальних новин (r/news) та науки (r/science) до надзвичайно вузьких ніш. Кожен сабреддіт має власні правила, модераторів та спільноту користувачів. Саме цей механізм підписки на сабреддіти є ядром персоналізації.

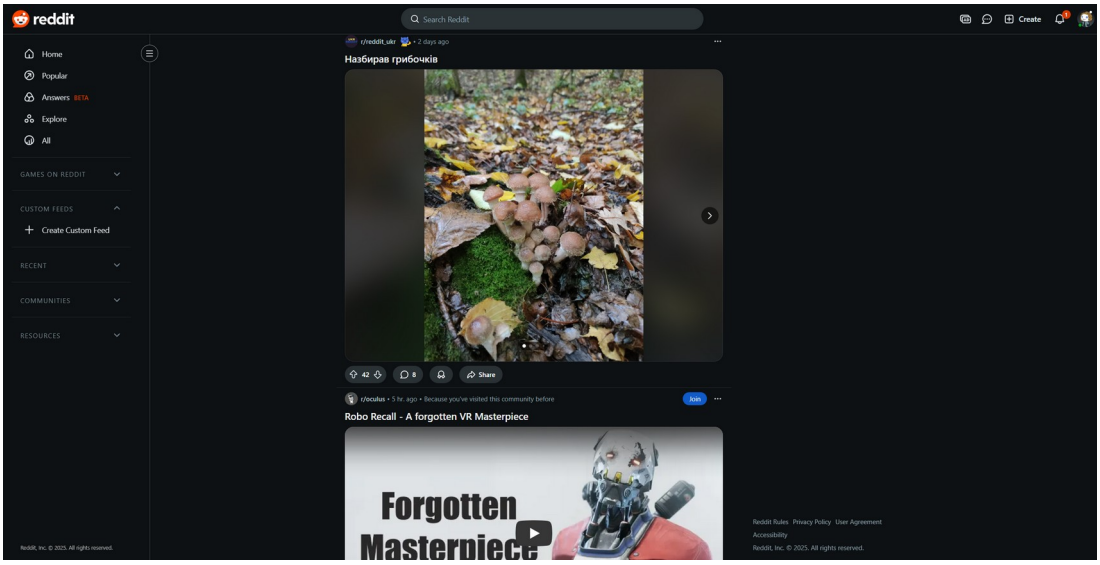


Рис. 1.5 Приклад головної стрічки Reddit, сформованої на основі підписок користувача на сабреддіти

Головна сторінка зареєстрованого користувача (Home) є агрегованою стрічкою, що складається з найпопулярніших на даний момент дописів з усіх сабреддітів, на які він підписаний. Таким чином, користувач сам виступає архітектором власного інформаційного простору, явно вказуючи системі, контент з яких тематичних спільнот він бажає отримувати. Це фундаментально відрізняється від "чорної скриньки" алгоритмів YouTube, де логіка рекомендацій не завжди є прозорою.

Ранжування дописів у межах стрічки та всередині кожного сабреддіту відбувається за допомогою демократичного механізму голосування. Користувачі можуть голосувати "за" (upvote) або "проти" (downvote) кожного допису та коментаря. Сума голосів формує рейтинг допису, а алгоритми ранжування (наприклад, "Hot", "Best", "Top") використовують цей рейтинг, а також час публікації та швидкість набору голосів, щоб визначити, які дописи

показувати на перших позиціях. Цей підхід дозволяє спільноті самостійно фільтрувати контент, виводячи нагору найцікавіші та найякісніші матеріали.

Хоча основна модель персоналізації є керованою користувачем, Reddit також інтегрує елементи алгоритмічних рекомендацій для стимулювання росту та залучення. Система періодично пропонує користувачу нові сабреддіти, які можуть його зацікавити. Ці рекомендації базуються на аналізі його поточних підписок та перетині аудиторії з іншими спільнотами. Наприклад, якщо значна частина користувачів, підписаних на r/python, також підписана на r/MachineLearning, система з високою ймовірністю порекомендує цей сабреддіт новому користувачеві, що цікавиться програмуванням на Python.

Таким чином, Reddit демонструє альтернативну та вельми ефективну модель контентної адаптації. Вона покладається на інтелект спільноти та свідомий вибір користувача, а не на складні поведінкові моделі. Цей підхід має важливу перевагу: він значно зменшує ризик утворення "фільтраційних бульбашок", оскільки користувач завжди має повний контроль над джерелами своєї інформації і може легко додавати або видаляти їх. Водночас, ефективність такої моделі залежить від активності самого користувача у пошуку та виборі цікавих йому спільнот.

Платформа TikTok[6], що належить компанії ByteDance, являє собою феномен у світі соціальних медіа, що здійснив революцію в споживанні короткого відеоконтенту. Успіх TikTok практично повністю завдячує його рекомендаційному алгоритму, який лежить в основі головної стрічки "Для вас" (For You Page, FYP). Цей алгоритм є прикладом екстремальної та гіпер-ефективної персоналізації, здатної з надзвичайною точністю та швидкістю визначати інтереси користувача та адаптувати контентне середовище під нього.

На відміну від багатьох інших платформ, де початкова стрічка формується на основі підписок, TikTok одразу занурює нового користувача в алгоритмічно сформовану стрічку "Для вас". Це дозволяє системі миттєво почати збір даних про поведінку користувача, не чекаючи, поки він сам

почне формувати свій соціальний граф. Архітектура платформи оптимізована для безперервного та швидкого споживання контенту, що, в свою чергу, генерує величезні обсяги даних для навчання моделей машинного навчання.

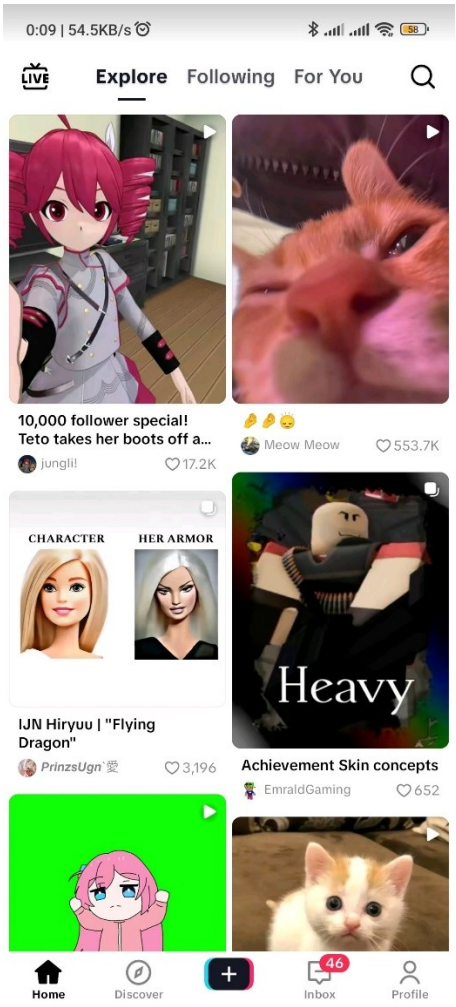


Рис. 1.6 – Приклад інтерфейсу стрічки "Для вас" на платформі TikTok

Рекомендаційна система TikTok є надзвичайно чутливою до найменших поведінкових сигналів. Якщо на YouTube ключовою метрикою є загальний час перегляду, то для коротких 15-60 секундних відео TikTok аналізує значно більш гранульовані показники. Алгоритм враховує не лише явні дії, такі як лайки, коментарі, поширення та підписки, але й приділяє особливу увагу неявним сигналам, що свідчать про рівень залученості. До найважливіших з них належать:

- Відсоток повторних переглядів (rewatch rate): Якщо користувач переглядає відео декілька разів, це є надзвичайно сильним позитивним сигналом.
- Відсоток повного перегляду (completion rate): Чи додивився користувач відео до кінця.
- Швидкість свайпу: Як швидко користувач прогортає відео. Миттєвий свайп є сильним негативним сигналом.
- Взаємодія з елементами відео: Система також аналізує інформацію, що міститься в самому відео, таку як використаний аудіо-трек, ефекти, хештеги та текст опису. Якщо користувач позитивно реагує на кілька відео з певним музичним треком, алгоритм почне пропонувати йому більше контенту з цією ж музикою.

Особливістю алгоритму є його здатність до швидкого навчання та диверсифікації контенту. Система не замикає користувача в рамках однієї теми, а постійно "тестує" його реакцію на новий, потенційно цікавий контент з суміжних областей. Якщо користувач, що цікавиться кулінарією, позитивно відреагує на кілька відео про подорожі, алгоритм поступово почне інтегрувати цю нову тему в його стрічку. Це дозволяє уникнути швидкого "вигорання" та постійно підтримувати інтерес.

Таким чином, TikTok демонструє, як аналіз мікро-сигналів у поєднанні з величезними обсягами даних та швидким циклом навчання дозволяє досягти безпрецедентного рівня персоналізації. Модель TikTok є настільки ефективною, що вона змусила інші платформи, зокрема Instagram (з функцією Reels) та YouTube (з функцією Shorts), копіювати її основні принципи. Аналіз цієї платформи є важливим для розуміння сучасних тенденцій у споживанні контенту, де швидкість, динаміка та гіпер-персоналізація відіграють вирішальну роль.

Платформа Pinterest[7] займає унікальну нішу в цифровому ландшафті, позиціонуючи себе не як соціальна мережа, а як візуальна пошукова система для ідей (visual discovery engine). Її основна мета — допомагати користувачам знаходити натхнення для різноманітних аспектів життя: від

рецептів та ідей для ремонту до моди та подорожей. Ця орієнтація на майбутні плани та проекти, а не на минулі події (як у традиційних соцмережах), визначає специфіку її рекомендаційних алгоритмів.

В основі платформи лежить концепція "пінів" (pins) — візуальних закладок (зображень, інфографіки, коротких відео), які користувачі можуть зберігати на власні тематичні "дошки" (boards). Саме акт збереження піна на дошку є ключовим сигналом для системи, що свідчить про довгострокові інтереси та наміри користувача. Цей сигнал є значно потужнішим, ніж тимчасовий "лайк", оскільки він відображає бажання повернутися до ідеї в майбутньому.

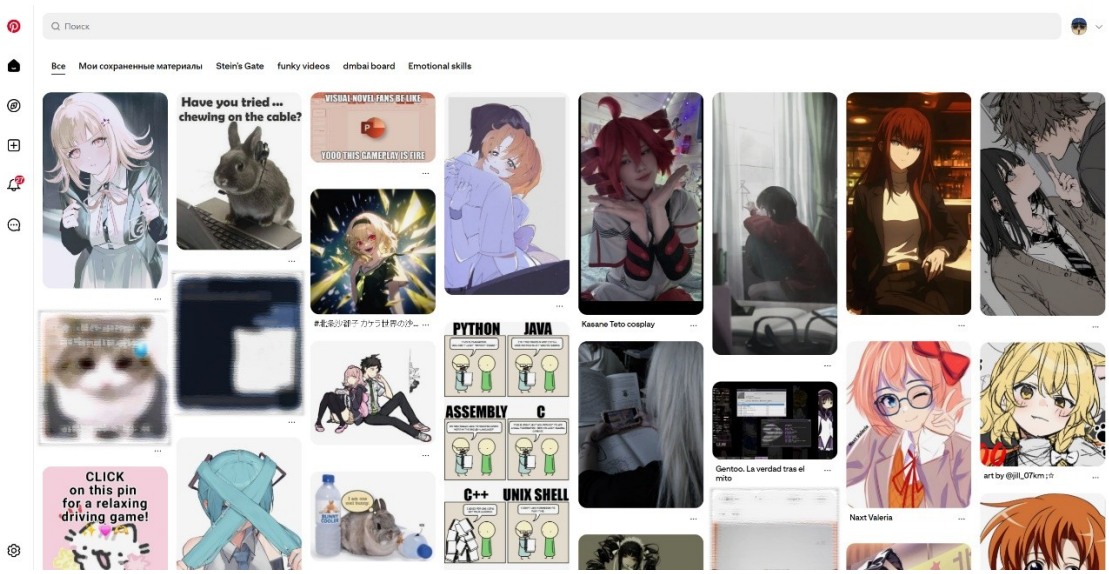


Рис. 1.7 Головна сторінка Pinterest, що є персоналізованою сіткою візуальних рекомендацій ("пінів")

Рекомендаційна система Pinterest є багатокомпонентною і використовує як аналіз поведінки користувачів, так і глибокий аналіз самих зображень. Головна стрічка користувача є нескінченною сіткою пінів, сформованою на основі його попередньої активності. Ключовими елементами, що живлять цю систему, є:

- Граф інтересів (Interest Graph): Pinterest створює детальний профіль інтересів для кожного користувача. Цей профіль складається з тем, дошок та конкретних пінів, з якими він взаємодіяв. Наприклад, якщо користувач

створює дошку "Ідеї для вітальні" і зберігає туди піни з сірими диванами та дерев'яними столами, система робить висновок про його інтерес до "скандинавського стилю в інтер'єрі".

- Технології комп'ютерного зору (Computer Vision): Це одна з найсильніших сторін Pinterest. Платформа використовує нейронні мережі для аналізу кожного завантаженого зображення. Система здатна розпізнавати об'єкти на фото (наприклад, "стілець", "лампа", "ваза"), визначати стиль ("мінімалізм", "лофт"), кольорову гаму та навіть загальну естетику. Це дозволяє їй знаходити візуально схожі піни. Функція "Visual Search" (візуальний пошук) дозволяє користувачу виділити частину будь-якого зображення і знайти інші піни з цим же або схожим об'єктом.
- Колаборативна фільтрація: Як і багато інших платформ, Pinterest аналізує, які піни зберігають разом різні користувачі. Якщо багато людей, що зберегли пін А, також зберігають пін Б, то система з високою ймовірністю порекомендує пін Б новому користувачеві, який щойно зберіг пін А.

Цікавою особливістю Pinterest є його здатність передбачати тренди. Аналізуючи мільярди пошукових запитів та збережених пінів, платформа може з високою точністю прогнозувати, які теми, стилі та продукти стануть популярними в майбутньому, що робить її потужним інструментом для маркетологів та бізнесу.

Таким чином, Pinterest демонструє, як персоналізація може бути побудована навколо візуальних даних та довгострокових намірів користувачів. На відміну від систем, що оптимізують миттєве залучення (як TikTok), Pinterest фокусується на наданні корисної та релевантної інформації для майбутніх проектів. Його аналіз є важливим, оскільки показує ефективність застосування технологій комп'ютерного зору для створення глибоко персоналізованого візуального досвіду.

Музичний стрімінговий сервіс Spotify[8] є яскравим прикладом того, як глибока персоналізація може стати не просто додатковою функцією, а ядром продукту та його головною конкурентною перевагою. В умовах, коли більшість стрімінгових сервісів пропонують доступ до практично ідентичних

музичних каталогів, що налічують десятки мільйонів треків, саме здатність допомогти користувачу зорієнтуватися в цьому океані музики та відкрити для себе нових виконавців стає вирішальним фактором. Spotify досяг цього завдяки розробці однієї з найдосконаліших рекомендаційних систем у галузі.

Центральним елементом персоналізації на Spotify є не стільки головна сторінка, скільки автоматично генеровані персоналізовані плейлисти. Найвідомішими з них є "Discover Weekly" (Щотижневий мікс відкриттів), "Release Radar" (Радар новинок) та щоденні мікси ("Daily Mixes"). Ці плейлисти є кінцевим продуктом складної системи, що обробляє величезні масиви даних.

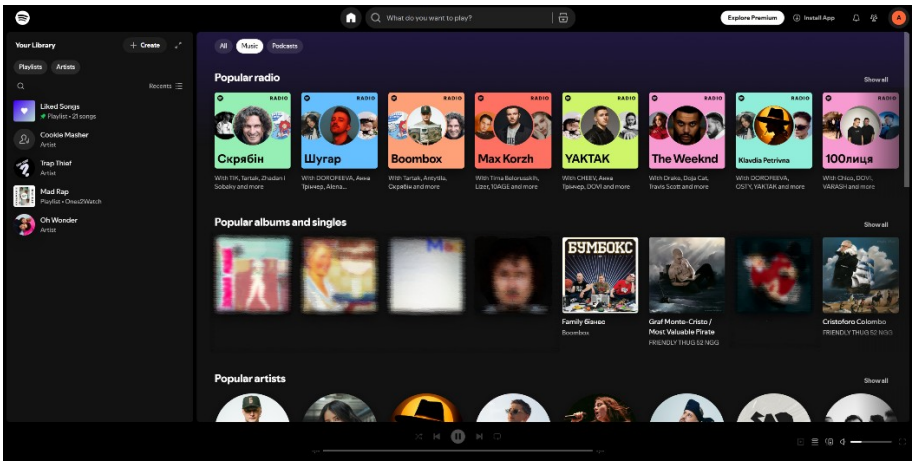


Рис. 1.8 Інтерфейс Spotify, що містить персоналізовані плейлисти

В основі рекомендаційної системи Spotify лежить гібридна модель, що поєднує три фундаментальні підходи:

Перший і найважливіший підхід — це колаборативна фільтрація (Collaborative Filtering). Ця модель працює за принципом "люди, схожі на вас, також слухають...". Система аналізує мільярди плейлистів, створених користувачами по всьому світу. Вона знаходить користувачів зі схожими музичними смаками, аналізуючи перетини в треках, які вони слухають, додають до плейлистів та "лайкають". Потім система знаходить треки, які популярні серед цієї групи "смакових двійників", але яких ще немає у вашій

бібліотеці, і пропонує їх вам. Саме цей механізм є основою для плейлиста "Discover Weekly"[8].

Другий підхід — це аналіз аудіосигналу на основі обробки природної мови та глибокого навчання. Spotify не просто розглядає треки як абстрактні ідентифікатори, а аналізує їхній внутрішній зміст. Кожен трек пропускається через нейронні мережі, які аналізують його аудіо-характеристики: темп, тональність, енергійність, танцювальність, настрій (мажорний/мінорний), наявність вокалу тощо. Це дозволяє системі створювати детальний "акустичний профіль" для кожного треку. Цей аналіз дає змогу знаходити музично схожі композиції, навіть якщо їх ніколи не слухали разом одні й ті ж люди. Цей метод є ключовим для функції "Spotify Radio", яка створює нескінченний потік музики на основі одного обраного треку чи виконавця.

Третій підхід — це аналіз текстових даних (Natural Language Processing, NLP). Система сканує інтернет (музичні блоги, новинні сайти, рецензії), щоб зрозуміти, про яких виконавців та які треки говорять, якими епітетами їх описують, до яких жанрів відносять. Це дозволяє групувати виконавців за стилістичною близькістю та відстежувати нові тренди, що є особливо важливим для плейлиста "Release Radar", який пропонує новинки від виконавців, схожих на тих, яких ви слухаєте.

Таким чином, Spotify демонструє винятково ефективну синергію трьох потужних підходів до персоналізації. Поєднуючи аналіз поведінки спільноти (колаборативна фільтрація), глибокий аналіз самого контенту (аудіо-профілі) та аналіз зовнішнього контексту (веб-дані), сервіс створює глибоко персоналізований та динамічний музичний досвід. Цей приклад показує, що найпотужніші рекомендаційні системи часто є гібридними, використовуючи сильні сторони різних моделей для досягнення максимальної точності та релевантності.

Переходячи від аналізу контент-платформ до сфери електронної комерції, важливо розглянути, як принципи персоналізації застосовуються для рекомендації фізичних та цифрових товарів. Глобальний маркетплейс AliExpress[9], що належить Alibaba Group, є яскравим прикладом платформи,

де рекомендаційні системи є критично важливим інструментом для покращення користувацького досвіду та стимулювання продажів. З огляду на асортимент, що налічує сотні мільйонів товарів від тисяч продавців, ефективна навігація та пошук релевантних продуктів без інтелектуальної допомоги були б неможливими.

Рекомендаційна система AliExpress, як і в багатьох інших великих e-commerce проєктах, є багатошаровою та використовує різноманітні дані для формування персоналізованих пропозицій, які відображаються на всіх ключових етапах взаємодії користувача з сайтом.

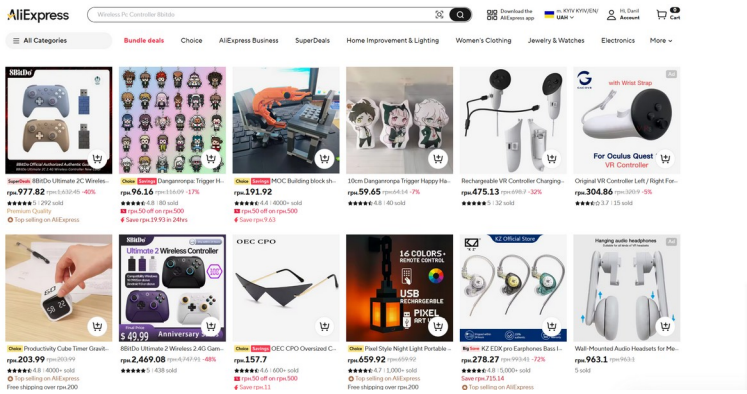


Рис. 1.9 Головна сторінка AliExpress персоналізованими товарними рекомендаціями

Основним завданням системи є прогнозування ймовірності покупки товару конкретним користувачем. Для вирішення цього завдання використовуються моделі машинного навчання, що навчаються на величезних масивах даних про поведінку користувачів. Ці дані можна класифікувати за кількома типами. По-перше, це історія взаємодії користувача: попередні покупки, переглянуті товари, товари, додані до кошика або списку бажань, а також пошукові запити. Ці дані формують довгостроковий профіль інтересів користувача.

По-друге, система активно аналізує поточну сесію користувача. Товари, які користувач переглядає "тут і зараз", є надзвичайно сильним сигналом про його короткострокові наміри. На основі цих даних формуються динамічні рекомендаційні блоки, такі як "Схожі товари" (Similar items) на

сторінці продукту. Для визначення "схожості" використовуються як аналіз текстових атрибутів товару (назва, опис, категорія), так і технології комп'ютерного зору для аналізу зображень продукту.

Третім і, мабуть, найпотужнішим компонентом є колаборативна фільтрація, яка працює на основі аналізу поведінки мільйонів користувачів. Класичним прикладом її реалізації є блоки "Ті, хто переглядав цей товар, також цікавилися" (Customers who viewed this item also viewed). Система знаходить групи користувачів зі схожими патернами переглядів та покупок і використовує ці знання для перехресних рекомендацій. Цей підхід дозволяє користувачам відкривати для себе товари, які вони могли б не знайти за допомогою прямого пошуку.

Персоналізовані рекомендації інтегровані в усі ключові точки взаємодії з платформою. Головна сторінка для кожного зареєстрованого користувача є унікальною і містить добірки товарів на основі його інтересів. Рекомендації також з'являються на сторінках категорій, у результатах пошуку (де персоналізовані товари можуть отримувати вищі позиції), на сторінці товару та навіть у кошику. Крім того, платформа активно використовує персоналізацію в маркетингових комунікаціях, надсилаючи електронні листи та push-сповіщення з добірками товарів, які з високою ймовірністю зацікавлять користувача.

Таким чином, AliExpress демонструє, як методи персоналізації, що застосовуються для контенту, успішно адаптуються для сфери електронної комерції. Ключовою відмінністю є кінцева мета: якщо для контент-платформ головною метрикою є залученість та час, проведений на сайті, то для e-commerce — це конверсія в покупку. Аналіз подібних платформ є важливим, оскільки він показує універсальність підходів, заснованих на машинному навчанні, та їхню здатність вирішувати бізнес-задачі в різних предметних областях.

Проведений комплексний аналіз провідних світових веб-платформ з різних сегментів — від текстового блогінгу та соціальних медіа до стрімінгових сервісів та електронної комерції — дозволяє ідентифікувати

низку фундаментальних тенденцій, що визначають сучасний стан розвитку веб-ресурсів.

По-перше, адаптивний дизайн став безумовним гігієнічним стандартом галузі. Усі без винятку успішні платформи забезпечують безшовний та комфортний досвід взаємодії незалежно від типу пристрою користувача. Це свідчить про те, що адаптація на рівні інтерфейсу більше не є конкурентною перевагою, а є базовою вимогою для будь-якого сучасного веб-проєкту.

По-друге, ключовим полем для конкуренції та головним фактором утримання аудиторії стала адаптація на рівні контенту, реалізована через механізми персоналізації. Аналіз показав, що не існує єдиного універсального підходу до її впровадження. Натомість можна виділити цілий спектр моделей, ефективність яких залежить від типу контенту та бізнес-цілей платформи:

- Системи, керовані користувачем (на прикладі Reddit[5]), де персоналізація є результатом свідомого вибору користувача, що забезпечує високий рівень прозорості та контролю.
- Системи, засновані на соціальному графі (на прикладі Tumblr[3]), де рекомендації формуються на основі дій спільноти (лайків, поширень).
- Системи, засновані на аналізі контенту (Content-Based), де використовуються технології NLP (Medium)[29] або комп'ютерного зору (Pinterest)[7] для пошуку тематично або візуально схожих об'єктів.
- Системи, засновані на колаборативній фільтрації, що аналізують поведінкові патерни мільйонів користувачів для виявлення неочевидних зв'язків (Spotify, AliExpress) [8][9].
- Системи на основі глибокого навчання (YouTube, TikTok)[4][6][25], що є найскладнішими, але й найефективнішими, оскільки здатні аналізувати сотні гранульованих сигналів у реальному часі для створення гіпер-персоналізованого досвіду.

По-третє, очевидною є тенденція до гібридизації моделей. Найбільш досконалі системи (як у Spotify)[23] одночасно використовують декілька

підходів, комбінуючи їхні сильні сторони для досягнення максимальної точності рекомендацій.

Таким чином, для розробки конкурентоспроможного веб-сайту для ведення блогу в сучасних умовах необхідно вийти за рамки простої реалізації адаптивного дизайну. Проєкт має включати в себе вбудований механізм персоналізації контенту. Враховуючи обмеження дипломної роботи (відсутність великих обсягів даних про поведінку користувачів на старті), найбільш раціональним та реалістичним для впровадження є підхід, заснований на аналізі вмісту (Content-Based Filtering). Цей метод не вимагає даних про інших користувачів і може надавати якісні рекомендації на основі аналізу семантичної схожості самих текстів статей. Подальше дослідження в рамках даної роботи буде сфокусовано саме на аналізі, проєктуванні та реалізації такого механізму.

1.2. Проблема залучення користувачів та роль рекомендаційних систем

Як було зазначено у попередньому підрозділі, сучасне інформаційне середовище характеризується надлишковою пропозицією контенту. Цей феномен створює фундаментальну проблему для власників веб-ресурсів — проблему залучення та утримання користувачів (user engagement & retention). Залучення користувачів є комплексним поняттям, що описує якість користувацького досвіду, яка характеризується ступенем концентрації уваги та поглинання в процес взаємодії з системою. Високий рівень залучення безпосередньо корелює з ключовими показниками ефективності веб-ресурсу: тривалістю сесії, глибиною переглядів (кількість сторінок за сесію), частотою повторних візитів та, зрештою, лояльністю аудиторії.

Традиційні методи навігації, такі як статичні меню, ієрархічні категорії та хронологічні стрічки, виявляються неефективними в умовах великих обсягів контенту. Вони перекладають всю когнітивну роботу з пошуку релевантної інформації на самого користувача. Це, в свою чергу, призводить

до виникнення психологічного ефекту, відомого як "парадокс вибору" (paradox of choice)[26]. Цей ефект, описаний психологом Баррі Шварцем, полягає в тому, що надмірна кількість варіантів не полегшує, а ускладнює процес прийняття рішень, викликаючи у людини відчуття невпевненості, тривоги та, зрештою, незадоволення зробленим вибором або повну відмову від нього. У контексті блог-платформи з сотнями чи тисячами статей "парадокс вибору" проявляється в тому, що користувач, не знайшовши швидко щось цікаве, з високою ймовірністю покине сайт, навіть якщо на ньому є десятки релевантних для нього матеріалів.

Стратегічним рішенням цієї проблеми стало впровадження інтелектуальних рекомендаційних систем, що є практичним застосуванням технологій ШІ. Рекомендаційна система — це клас систем фільтрації інформації, головною метою яких є прогнозування уподобань користувача щодо певних об'єктів (статей, відео, товарів) та надання йому персоналізованого списку цих об'єктів. Замість того, щоб змушувати користувача "витягувати" інформацію з системи, рекомендаційна система проактивно "проштовхує" найбільш релевантний контент до користувача, виступаючи в ролі персонального гіда по інформаційному простору.

Роль рекомендаційних систем у підвищенні залученості є багатогранною. З точки зору користувацького досвіду, вони виконують декілька критично важливих функцій. По-перше, вони значно знижують когнітивне навантаження та спрощують процес навігації. По-друге, вони підвищують цінність контенту, пропонуючи матеріали, що точно відповідають поточним та довгостроковим інтересам користувача. По-третє, вони сприяють "випадковим відкриттям" (serendipity), пропонуючи контент з суміжних, потенційно цікавих для користувача областей, про які він міг і не здогадуватися.

З точки зору бізнес-цілей платформи, роль рекомендаційних систем є ще більш вагомою. Вони безпосередньо впливають на ключові метрики ефективності. Пропонуючи користувачу безперервний потік цікавого контенту, вони збільшують тривалість його перебування на сайті.

Детальніше, вплив рекомендаційних систем на бізнес-метрики можна структурувати за кількома напрямками. Перший — це збільшення споживання контенту. Ефективна система рекомендацій створює для користувача "шлях найменшого опору" до релевантного контенту, що призводить до збільшення кількості переглянутих сторінок за сесію. Для e-commerce платформ аналогічний ефект виражається у збільшенні кількості переглянутих товарів та середнього чеку. Другий напрямок — це підвищення конверсії. Рекомендації можуть бути націлені не лише на утримання уваги, але й на стимулювання цільових дій: підписки на розсилку, реєстрації, переходу на платний тариф або покупки товару. Персоналізовані пропозиції, що враховують попередню поведінку, мають значно вищий коефіцієнт конверсії порівняно з загальними, нетаргетованими закликами до дії.

Третій, менш очевидний, але стратегічно важливий аспект — це підвищення різноманітності споживання (consumption diversity). Платформи, що покладаються лише на показники популярності ("тренди"), ризикують потрапити у пастку "ефекту Матвія", коли популярний контент стає ще популярнішим, повністю затьмарюючи менш відомі, але потенційно якісні матеріали. Це призводить до гомогенізації користувацького досвіду та швидкого "вигорання" аудиторії. Рекомендаційні системи, навпаки, здатні диверсифікувати споживання, пропонуючи користувачам нішевий контент з "довгого хвоста"[27]. Це не лише покращує досвід для користувачів з унікальними смаками, але й створює більш здорову та стійку екосистему для авторів контенту, даючи шанс бути поміченими навіть невеликим блогерам.

Це дозволяє задіяти весь масив контенту, а не лише його найпопулярнішу частину, що значно підвищує загальну цінність платформи.

Таким чином, у сучасній веб-архітектурі рекомендаційні системи перестали бути опціональним доповненням і перетворилися на один з центральних компонентів, що визначають успішність контент-орієнтованих проєктів. Вони є найефективнішим інструментом для боротьби з інформаційним перевантаженням та "парадоксом вибору", що дозволяє одночасно покращити користувацький досвід та досягти стратегічних бізнес-

цілей. Подальший аналіз буде присвячено класифікації та дослідженню конкретних алгоритмів, що лежать в основі таких систем[23].

1.3. Класифікація та аналіз алгоритмів рекомендаційних систем

Рекомендаційні системи, незважаючи на різноманітність їхніх реалізацій, базуються на обмеженій кількості фундаментальних алгоритмічних підходів. Розуміння принципів роботи, переваг та недоліків кожного з цих підходів є необхідною умовою для проєктування ефективної системи, що відповідає специфіці конкретної предметної області. У науковій літературі прийнято виділяти три основні класи алгоритмів: колаборативна фільтрація, фільтрація на основі вмісту та гібридні підходи.

1.3.1. Колаборативна фільтрація (Collaborative Filtering, CF)

Колаборативна фільтрація є одним із найперших та найпоширеніших підходів до побудови рекомендаційних систем. Основна ідея, що лежить в її основі, полягає у використанні "мудрості натовпу": система автоматично знаходить групи користувачів зі схожими смаками та генерує рекомендації на основі їхніх уподобань. Іншими словами, якщо користувачу А та користувачу Б сподобався однаковий набір статей, то стаття, яка сподобалась користувачу А, з високою ймовірністю буде рекомендована користувачу Б.

Для роботи алгоритмів CF необхідні дані про взаємодію користувачів з об'єктами, які зазвичай представляються у вигляді розрідженої матриці "користувач-об'єкт" (user-item interaction matrix). Значеннями в цій матриці можуть бути явні оцінки (наприклад, рейтинг від 1 до 5 зірок) або неявні сигнали (факт перегляду, покупки, прослуховування).

Існує два основних підкласи колаборативної фільтрації:

- Підходи, засновані на пам'яті (Memory-Based CF): Цей підхід безпосередньо працює з усією матрицею взаємодій. Він, у свою чергу, поділяється на User-Based CF, де система шукає "сусідів" (схожих

користувачів) і рекомендує об'єкти, які сподобались цим сусідам, та Item-Based CF, де система шукає схожі об'єкти на основі того, що їх однаково оцінювали одні й ті ж користувачі. Item-Based підхід часто є більш стабільним та масштабованим.

- Підходи, засновані на моделі (Model-Based CF): Замість того, щоб працювати з усією матрицею, цей підхід використовує методи машинного навчання для побудови компактної моделі, що апроксимує дані. Найпопулярнішим методом є матрична факторизація (Matrix Factorization, наприклад, SVD), яка намагається "розкласти" вихідну матрицю на дві матриці меншого розміру, що представляють приховані (латентні) фактори для користувачів та об'єктів. Ці латентні фактори можуть інтерпретуватися як приховані характеристики (наприклад, для фільмів це можуть бути жанри, режисери, актори, які система вивчила автоматично).

Переваги CF: здатність генерувати несподівані та нові рекомендації (serendipity), оскільки алгоритму не потрібні знання про сам об'єкт, а лише дані про взаємодію з ним.

Недоліки CF: гостро стоїть проблема "холодного старту" (cold start) — система не може генерувати рекомендації для нових користувачів або нових об'єктів, по яких ще немає даних. Також проблемою є розрідженість даних (data sparsity) та низька масштабованість для Memory-Based підходів[23].

1.3.2. Фільтрація на основі вмісту (Content-Based Filtering, CBF)

Фільтрація на основі вмісту є альтернативним підходом, що намагається вирішити проблему "холодного старту" для нових об'єктів. Основна ідея полягає в тому, щоб рекомендувати користувачу об'єкти, схожі на ті, що йому сподобались у минулому. Схожість при цьому визначається на основі аналізу характеристик (атрибутів) самих об'єктів.

Процес роботи CBF-системи зазвичай складається з трьох етапів:

1. Видобування характеристик (Feature Extraction): Для кожного об'єкта в системі створюється профіль, що складається з його ключових

атрибутів. Для текстової статті це можуть бути ключові слова, для фільму — жанр, актори, режисер, для товару — категорія, бренд, колір.

2. Побудова профілю користувача: На основі об'єктів, з якими користувач позитивно взаємодіяв, система будує узагальнений профіль його інтересів. Цей профіль також представляється у вигляді набору характеристик та їхньої ваги.

3. Генерація рекомендацій: Система порівнює профіль користувача з профілями всіх об'єктів і рекомендує ті, які є найбільш схожими.

Для реалізації цього підходу для текстових даних часто використовуються методи обробки природної мови, зокрема, модель TF-IDF (Term Frequency-Inverse Document Frequency) для векторизації текстів та косинусна подібність (cosine similarity) для визначення схожості між векторами.

Переваги CBF: вирішує проблему "холодного старту" для нових об'єктів, рекомендації є прозорими та легко інтерпретуються, не потребує даних про інших користувачів.

Недоліки CBF: обмежена здатність до відкриття нового (serendipity) — система рекомендує лише те, що схоже на вже відоме, і не може вийти за рамки існуючого профілю користувача. Якість рекомендацій сильно залежить від повноти та якості характеристик об'єктів[24].

1.3.3. Гібридні підходи (Hybrid Approaches)

Оскільки як колаборативна фільтрація, так і фільтрація на основі вмісту мають свої сильні та слабкі сторони, найсучасніші та найефективніші рекомендаційні системи використовують гібридні підходи, що комбінують елементи обох моделей. Мета гібридизації — використати переваги одного підходу для компенсації недоліків іншого.

Існує багато стратегій гібридизації, серед яких найпоширенішими є:

- Зважування (Weighted): Результати обох систем (CF та CBF) обчислюються незалежно, а потім комбінуються за допомогою зваженої суми для отримання фінального рейтингу.
- Перемикання (Switching): Система використовує один метод (наприклад, CF), але якщо він не може згенерувати рекомендацію (наприклад, через "холодний старт"), вона перемикається на інший (CBF).
- Каскадна модель (Cascade): Один метод використовується для грубої фільтрації та створення списку кандидатів, а другий — для більш точного ранжування цього списку. Наприклад, CBF може відібрати 100 тематично схожих статей, а CF — відранжувати їх на основі популярності серед схожих користувачів.
- Комбінування характеристик (Feature Combination): Вихідні дані однієї моделі використовуються як вхідні характеристики для іншої.

Гібридні системи зазвичай демонструють значно вищу точність та стійкість порівняно з "чистими" реалізаціями, але їхня розробка та налаштування є значно складнішими[23].

Таблиця 1.1 Порівняльна таблиця алгоритмів рекомендаційних систем

Характеристика	Колаборативна фільтрація (CF)	Фільтрація на основі вмісту (CBF)	Гібридні підходи
Основний принцип	Рекомендувати те, що сподобалось схожим користувачам ("мудрість натовпу").	Рекомендувати об'єкти, схожі на ті, що сподобались користувачу раніше.	Комбінування переваг CF та CBF для компенсації недоліків.
Необхідні дані	Матриця взаємодій "користувач-об'єкт" (оцінки, перегляди, покупки).	Характеристики (атрибути) об'єктів (текст, жанр, теги).	Дані, необхідні для обох підходів.

Переваги	Здатність до відкриття нового (serendipity), не потребує аналізу контенту.	Вирішує "холодний старт" для об'єктів, прозорість рекомендацій.	Вища точність, стійкість до "холодного старту" та розрідженості даних.
Недоліки	"Холодний старт" для користувачів/об'єктів, розрідженість даних.	Обмежена новизна рекомендацій, залежність від якості характеристик.	Висока складність розробки та налаштування.
Приклад	"Користувачі, що купили цей товар, також купили..." (Amazon, AliExpress).	Рекомендація статей на ту ж тему, що й прочитана (новинні сайти).	Персоналізовані плейлисти (Spotify), стрічка новин (Facebook).

1.4. Дослідження методів фільтрації на основі вмісту (Content-Based Filtering)

Як було обґрунтовано у висновках до попередніх підрозділів, для розробки рекомендаційного модуля в рамках даної дипломної роботи було обрано підхід фільтрації на основі вмісту (Content-Based Filtering, CBF). Такий вибір зумовлений його ключовою перевагою — здатністю функціонувати без великих обсягів даних про поведінку користувачів, що є критично важливим для новоствореного веб-ресурсу.

Загальний процес роботи CBF-системи можна декомпозувати на кілька послідовних етапів. Першою і найважливішою задачею є представлення контенту у числовому форматі. Оскільки моделі штучного інтелекту не можуть працювати з неструктурованим текстом напряму, кожну статтю необхідно перетворити на структурований числовий вектор. Цей процес називається векторизацією. Найбільш поширеною моделлю для цього є "Мішок слів" (Bag-of-Words), яка представляє документ як набір слів без урахування їхнього порядку. Для кількісної оцінки важливості кожного слова використовується статистична міра TF-IDF[24][29] (Term Frequency – Inverse Document Frequency). Її головна ідея полягає в тому, що вага слова є тим вищою, чим частіше воно зустрічається в конкретній статті, і чим рідше воно

зустрічається в усіх інших статтях на сайті. Такий підхід дозволяє автоматично виділити специфічні терміни, що найкраще характеризують тему статті, та знизити вагу загальновживаних слів. Після розрахунку ваги TF-IDF для всіх значущих слів, кожна стаття представляється у вигляді числового вектора, що є її "цифровим відбитком" у багатовимірному семантичному просторі.

Наступним логічним кроком є профілювання користувача. Після того, як усі статті представлені у векторному форматі, система створює профіль інтересів для кожного користувача. Цей профіль будується на основі характеристик тих статей, з якими користувач позитивно взаємодіяв (наприклад, прочитав чи вподобав). Поширеним методом є обчислення усередненого вектора інтересів, де система бере числові вектори всіх статей, які сподобались користувачу, і знаходить їхнє середнє арифметичне. Результатом є новий, узагальнений вектор, що відображає центральні інтереси користувача. Наприклад, якщо користувач читав переважно статті про Python та машинне навчання, у його профільному векторі найбільшу вагу матимуть терміни, пов'язані саме з цими темами.

Фінальний етап — це генерація рекомендацій шляхом розрахунку подібності. Система порівнює профільний вектор користувача з векторами всіх статей, які він ще не бачив, і ранжує їх за ступенем схожості. Найбільш популярною метрикою для цього є косинусна подібність[24] (cosine similarity). Суть цього методу полягає у вимірюванні кута між двома векторами: якщо вони спрямовані в один бік (статті дуже схожі за тематикою), значення подібності буде близьким до одиниці; якщо ж вони вказують у різних напрямках, значення буде близьким до нуля. Перевага цього методу в тому, що він враховує саме тематичну спрямованість, а не довжину статті. Для формування списку рекомендацій система обчислює косинусну подібність між вектором користувача та векторами всіх непрочитаних статей, сортує їх за отриманим значенням і пропонує користувачу N найкращих результатів.

Таким чином, підхід фільтрації на основі вмісту надає повний, логічно обґрунтований інструментарій для створення ефективної рекомендаційної системи для текстового контенту, що є ідеальним рішенням для початкового етапу розробки блог-платформи.

1.5. Висновки до розділу 1

У першому розділі даної кваліфікаційної роботи було проведено комплексне теоретичне та аналітичне дослідження предметної області, пов'язаної з розробкою сучасних контент-орієнтованих веб-платформ.

Проведений детальний огляд широкого спектру провідних світових ресурсів, від спеціалізованих блог-платформ до глобальних медіа-екосистем, дозволив виявити дві ключові тенденції. По-перше, адаптивний дизайн утвердився як фундаментальний стандарт галузі, що забезпечує доступність контенту на будь-якому пристрої. По-друге, головним фактором конкурентної боротьби за увагу користувача стала інтелектуальна адаптація самого контенту, реалізована через системи персоналізації. Було встановлено, що успішні платформи активно використовують широкий спектр алгоритмів: від керованих вибором користувача (Reddit) до складних моделей глибокого навчання, що аналізують поведінкові сигнали в реальному часі (YouTube, TikTok).

У ході дослідження було обґрунтовано теоретичну та практичну важливість рекомендаційних систем як основного інструменту для вирішення проблеми інформаційного перевантаження та підвищення залученості аудиторії. На основі системного аналізу було проведено класифікацію основних алгоритмічних підходів, виявлено їхні сильні та слабкі сторони. Зокрема, було детально розглянуто принципи роботи колаборативної фільтрації, що базується на "мудрості натовпу", та фільтрації на основі вмісту, що фокусується на аналізі характеристик самих об'єктів.

За результатами порівняльного аналізу було зроблено висновок, що для розробки нового веб-ресурсу в умовах відсутності великих масивів даних про

поведінку користувачів, найбільш доцільним та реалістичним для впровадження є підхід фільтрації на основі вмісту (Content-Based Filtering). Детальне дослідження цього методу показало його спроможність надавати якісні рекомендації, спираючись виключно на семантичний аналіз текстового контенту статей. Концептуальна модель, що включає векторизацію текстів за допомогою TF-IDF та розрахунок схожості через косинусну подібність[24], є математично обґрунтованою та повною основою для побудови ефективного рекомендаційного модуля. Таким чином, результати, отримані в даному розділі, формують міцний теоретичний фундамент для подальшого проєктування та програмної реалізації веб-сайту з інтегрованою системою персоналізації.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ З ІНТЕЛЕКТУАЛЬНИМ МОДУЛЕМ НА ОСНОВІ ML

2.1. Розробка загальної архітектури системи

Архітектура програмної системи визначає її основні компоненти, їхні функції та взаємозв'язки. Правильно спроєктована архітектура є запорукою надійності, масштабованості та можливості подальшого розвитку продукту. Для розробки веб-сайту для ведення блогу з інтегрованим модулем персоналізації було обрано класичну клієнт-серверну архітектуру (Client-Server Architecture) з чітким розділенням відповідальності між клієнтською та серверною частинами.

В основі системи лежить модель взаємодії, де клієнтська частина (веб-браузер користувача) відповідає за відображення інтерфейсу та взаємодію з користувачем, а серверна частина — за обробку даних, реалізацію бізнес-логіки та роботу інтелектуального модуля. Обмін даними між клієнтом та сервером здійснюється за допомогою програмного інтерфейсу додатків (API), реалізованого на основі архітектурного стилю REST[10] (Representational State Transfer).

Загальна архітектура системи представлена на рисунку 2.1.

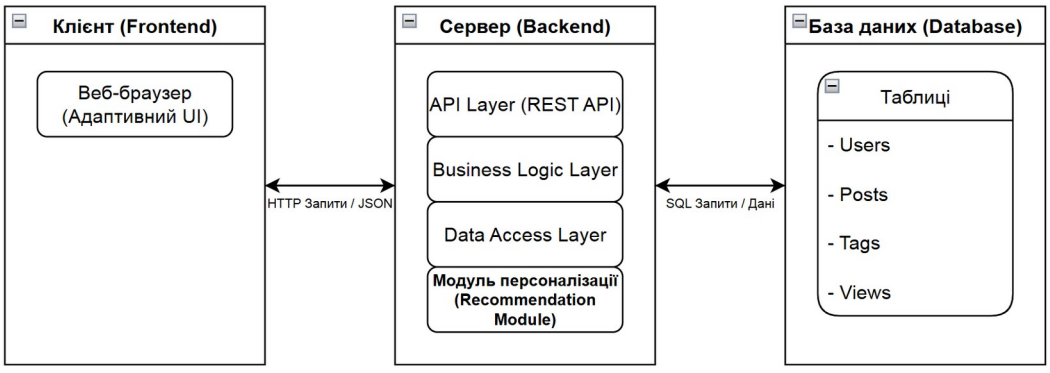


Рис. 2.1 Загальна архітектура системи

Розглянемо детальніше кожен з компонентів представленої архітектури.

2.1.1. Клієнтська частина (Frontend)

Клієнтська частина є односторінковим застосунком (Single Page Application, SPA), що виконується у веб-браузері користувача. Вона відповідає за весь візуальний аспект системи та взаємодію з користувачем. Її основними функціями є:

- Рендеринг інтерфейсу користувача: Відображення сторінок сайту (головної, сторінки статті, профілю користувача тощо). Саме на цьому рівні реалізуються принципи адаптивного дизайну, що забезпечують коректне відображення інтерфейсу на пристроях з різною роздільною здатністю екрана.
- Обробка дій користувача: Реєстрація кліків, введення тексту в форми (реєстрація, коментарі), надсилання запитів на сервер.
- Взаємодія з API: Формування та надсилання HTTP-запитів до серверної частини для отримання даних (наприклад, списку статей, тексту конкретної статті, персоналізованих рекомендацій) та відправки даних (створення нового коментаря, вподобання статті).
- Управління станом: Збереження та оновлення даних на стороні клієнта для забезпечення швидкої та плавної роботи інтерфейсу.

2.1.2. Серверна частина (Backend)

Серверна частина є ядром системи, де реалізована вся основна логіка. Для даного проєкту обрано монолітну архітектуру бекенду, оскільки вона є простішою в розробці та розгортанні порівняно з мікросервісною, і при цьому повністю задовольняє вимоги проєкту. Серверна частина складається з кількох логічних шарів:

- Шар API (API Layer): Відповідає за прийом HTTP-запитів від клієнта, їх валідацію, автентифікацію та авторизацію користувачів. Цей шар є "вхідними воротами" до всієї логіки системи.

- Шар бізнес-логіки (Business Logic Layer): Містить основну логіку додатку: управління користувачами, створення та редагування статей, робота з коментарями, обробка вподобань.
- Шар доступу до даних (Data Access Layer): Забезпечує взаємодію з базою даних, абстрагуючи логіку роботи з SQL-запитами.
- Модуль машинного навчання (ML Module): Це наукомістке ядро системи, що реалізує методи штучного інтелекту для персоналізації контенту, виділене в окремий логічний модуль. Він не пов'язаний безпосередньо з обробкою запитів, а надає сервіс для генерації рекомендацій. Його робота складається з двох процесів:
  - Пакетна обробка (batch processing): Періодично, за розкладом, цей процес аналізує всі статті в базі даних, розраховує для них TF-IDF вектори та зберігає їх. Це дозволяє уникнути ресурсоємних обчислень під час кожного запиту.
  - Обробка запиту в реальному часі: При отриманні запиту на рекомендацію для конкретного користувача, модуль отримує з бази даних інформацію про прочитані ним статті, на льоту будує його профіль інтересів, порівнює його з попередньо розрахованими векторами статей і повертає відсортований список ідентифікаторів рекомендованих статей.

2.1.3. База даних (Database)

Для зберігання всієї інформації системи використовується реляційна система управління базами даних (СУБД). Вона відповідає за надійне зберігання та швидкий доступ до даних. Основними сутностями, що зберігаються в базі, є: користувачі, статті, теги (категорії), коментарі, а також дані про взаємодію користувачів зі статтями (перегляди, вподобання).

Обрана архітектура є гнучкою та масштабованою. Чітке розділення на клієнтську та серверну частини дозволяє розвивати їх незалежно одна від одної. Виділення модуля персоналізації в окремий логічний компонент дає

змогу в майбутньому легко замінити або вдосконалити алгоритм рекомендацій, не зачіпаючи основну бізнес-логіку додатку.

2.2. Математична модель та алгоритм роботи модуля машинного навчання

На основі обраного в першому розділі методу фільтрації на основі вмісту (Content-Based Filtering), у даному підрозділі розроблено математичну модель та детальний покроковий алгоритм для функціонування сервісу персоналізованих рекомендацій. Цей алгоритм є ядром інтелектуального модуля на основі ШІ та призначений для вирішення задачі ранжування статей на основі їхньої семантичної подібності до інтересів користувача.

Треба зробити формальну постановку задачі. Нехай $D=\{d_1,d_2,\ldots,d_n\}$ — це множина всіх документів (статей) у системі, а $U=\{u_1,u_2,\ldots,u_m\}$ — множина всіх користувачів. Для кожного користувача u_j існує підмножина прочитаних (або позитивно оцінених) ним статей $D_j \subset D$.

Завдання рекомендаційного сервісу полягає в тому, щоб для заданого користувача u_j та множини непрочитаних ним статей $D^{'}=D \vee D_j$ обчислити функцію ранжування $Rank(d_i,d_j)$, яка кожній статті $d_i \in D^{'}$ ставить у відповідність числову оцінку релевантності. На основі цієї оцінки формується фінальний відсортований список рекомендацій.

В основі математичної моделі лежить векторно-просторова модель (Vector Space Model), де кожен текстовий документ представляється у вигляді вектора в багатовимірному просторі термінів[24].

Спочатку на основі всієї колекції документів D формується загальний словник $V=\{t_1,t_2,\ldots,t_k\}$, що є множиною всіх унікальних значущих термінів (слів) у корпусі. Розмірність словника k визначає розмірність векторного простору.

Кожен документ d_i з множини D представляється у вигляді k - вимірного вектора $v_i=\{w_{i1},w_{i2},\ldots,w_{ik}\}$, де кожна компонента w_{ij} є вагою терміну t_i в документі d_i. Вага розраховується за формулою TF-IDF[24][29]:

$$w_{ij} = TF(t_j, d_i) \times IDF(t_j, D)$$

де $TF(t_j, d_i)$ — це частота терміну t_j в документі d_i, а $IDF(t_j, D)$ — обернена частота документа для терміну t_j у всій колекції D.

- $TF(t_j, d_i) = f(t_j, d_i) / ¿ d_i \vee ¿$, де $f(t_j, d_i)$ — кількість входжень терміну t_j в документ d_i, а $¿ d_i \vee ¿$ — загальна кількість термінів у документі.
- $IDF(t_j, D) = \log ¿$, де $¿ D \vee ¿$ — загальна кількість документів, а знаменник — кількість документів, що містять термін t_j.

Профіль інтересів користувача u_j також представляється у вигляді вектора p_j в тому ж k-вимірному просторі. Цей вектор обчислюється як центроїд (усереднений вектор) TF-IDF векторів усіх статей D_j, з якими користувач позитивно взаємодіяв:

$$p_j = \left(\frac{1}{|D_j|} \right) * \sum v_i$$

для всіх v_i, що відповідають статтям $d_i \in D_j$.

Алгоритм роботи сервісу рекомендацій можна розділити на два етапи: попередню обробку, яка виконується періодично, та обробку запиту на рекомендацію в реальному часі.

Перший етап — це попередня обробка (Offline-етап). Він виконується за розкладом (наприклад, раз на добу) або при додаванні нових статей до системи. Мета цього етапу — виконати всі ресурсоємні обчислення заздалегідь.

- Крок 1 (Очищення та токенізація): Взяти всі текстові документи D з бази даних. Для кожного документа виконати попередню обробку тексту: переведення в нижній регістр, видалення стоп-слів (артиклів, прийменників), пунктуації та, за потреби, стеммінг або лематизацію.
- Крок 2 (Побудова словника): На основі очищених текстів сформувати загальний словник термінів V.
- Крок 3 (Розрахунок TF-IDF векторів): Для кожної статті d_i в системі обчислити її TF-IDF вектор v_i відповідно до формули вище.

- Крок 4 (Збереження векторів): Зберегти розраховані вектори в базу даних або окреме сховище, асоціюючи кожен вектор з ідентифікатором відповідної статті.

Другим етапом є обробка запиту на рекомендацію (Online-етап). Він виконується миттєво, коли користувач заходить на сторінку, де потрібно показати рекомендації.

- Крок 1 (Ідентифікація користувача): Отримати ідентифікатор поточного користувача u_j.
- Крок 2 (Отримання історії взаємодій): Зробити запит до бази даних та отримати список ідентифікаторів статей D_j, з якими користувач позитивно взаємодіяв.
- Крок 3 (Побудова профілю користувача): Завантажити попередньо розраховані TF-IDF вектори для всіх статей зі списку D_j. Обчислити профільний вектор користувача D_j відповідно до другої формули вище.
- Крок 4 (Визначення пулу кандидатів): Отримати з бази даних список всіх статей D. Сформувати множину статей-кандидатів для рекомендації $D^{'}$, виключивши з D ті статті, які користувач вже бачив ($D^{'} = D \vee D_j$).
- Крок 5 (Розрахунок подібності та ранжування): Для кожної статті-кандидата $d_i \in D^{'}$ завантажити її TF-IDF вектор v_i. Обчислити косинусну подібність між профільним вектором користувача p_j та вектором статті v_i:

$$Score(d_i) = \frac{p_j \cdot v_i}{¿ \vee p_j \vee ¿ * ¿ \vee v_i \vee ¿}$$

- Крок 6 (Формування списку рекомендацій): Відсортувати статті-кандидати $D^{'}$ у спадаючому порядку за їхніми оцінками $Score$. Повернути перші N статей з відсортованого списку як фінальну рекомендацію.

Така двохетапна архітектура дозволяє забезпечити високу швидкість відповіді сервісу, оскільки всі обчислювально складні операції (обробка текстів та розрахунок TF-IDF) виносяться в офлайн-етап, тоді як онлайн-етап виконує лише швидкі векторні операції[11].

2.3. Проєктування структури бази даних

Надійне зберігання даних та забезпечення швидкого доступу до них є фундаментальною вимогою для будь-якої динамічної веб-системи. Для даного проєкту було обрано реляційну модель даних, оскільки вона забезпечує цілісність, структурованість та гнучкість[12] у роботі з пов'язаними сутностями. Проєктування бази даних включає визначення основних сутностей, їхніх атрибутів та зв'язків між ними. Нижче наведено опис логічної схеми бази даних, що складається з ключових таблиць, необхідних для функціонування блог-платформи та її рекомендаційного модуля.

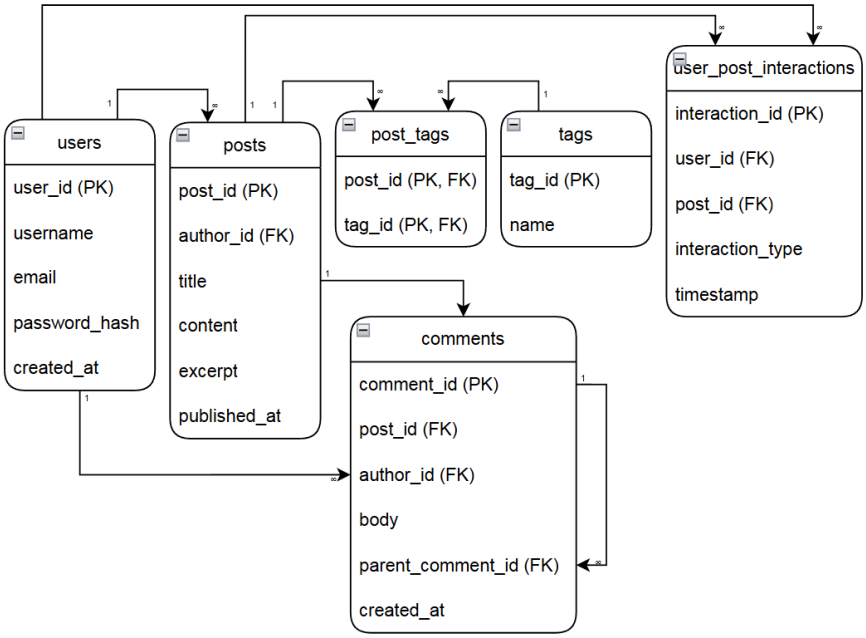


Рис. 2.2 ER-діаграма бази даних

Схема бази даних включає наступні основні сутності:

- Таблиця users

Призначена для зберігання інформації про зареєстрованих користувачів. Є основою для реалізації функціоналу автентифікації та авторизації.

o user_id: INT, PRIMARY KEY, AUTO_INCREMENT – унікальний ідентифікатор користувача.

- username: VARCHAR(50), UNIQUE, NOT NULL – логін користувача.
- email: VARCHAR(255), UNIQUE, NOT NULL – адреса електронної пошти.
- password_hash: VARCHAR(255), NOT NULL – хеш пароля для безпечного зберігання.
- created_at: TIMESTAMP – дата та час реєстрації.

- Таблиця posts

Зберігає безпосередньо контент блогу. Кожен запис у цій таблиці представляє одну статтю.

- post_id: INT, PRIMARY KEY, AUTO_INCREMENT – унікальний ідентифікатор статті.
- author_id: INT, FOREIGN KEY (references users.user_id) – ідентифікатор автора статті (зв'язок "один-до-багатьох" з таблицею users).
- title: VARCHAR(255), NOT NULL – заголовок статті.
- content: TEXT, NOT NULL – повний текст статті.
- excerpt: VARCHAR(500) – короткий опис або анотація.
- published_at: TIMESTAMP – дата та час публікації.

- Таблиця tags

Є довідником для категоризації контенту.

- tag_id: INT, PRIMARY KEY, AUTO_INCREMENT – унікальний ідентифікатор тегу.
- name: VARCHAR(50), UNIQUE, NOT NULL – назва тегу (наприклад, "Технології").

- Таблиця post_tags

Проміжна таблиця для реалізації зв'язку "багато-до-багатьох" між статтями та тегами.

- post_id: INT, FOREIGN KEY (references posts.post_id) – ідентифікатор статті.
- tag_id: INT, FOREIGN KEY (references tags.tag_id) – ідентифікатор тегу.

- PRIMARY KEY (post_id, tag_id) – складений первинний ключ для забезпечення унікальності пар.

- Таблиця user_post_interactions

Ключова таблиця для функціонування рекомендаційної системи. Фіксує дані про взаємодію користувачів з контентом.

- interaction_id: INT, PRIMARY KEY, AUTO_INCREMENT – унікальний ідентифікатор взаємодії.
- user_id: INT, FOREIGN KEY (references users.user_id) – ідентифікатор користувача.
- post_id: INT, FOREIGN KEY (references posts.post_id) – ідентифікатор статті.
- interaction_type: VARCHAR(20) – тип взаємодії ('view', 'like').
- timestamp: TIMESTAMP – дата та час взаємодії.

- Таблиця comments

Призначена для зберігання коментарів до статей.

- comment_id: INT, PRIMARY KEY, AUTO_INCREMENT – унікальний ідентифікатор коментаря.
- post_id: INT, FOREIGN KEY (references posts.post_id) – ідентифікатор статті, до якої належить коментар.
- author_id: INT, FOREIGN KEY (references users.user_id) – ідентифікатор автора коментаря.
- body: TEXT, NOT NULL – текст коментаря.
- parent_comment_id: INT, FOREIGN KEY (references comments.comment_id), NULL – ідентифікатор батьківського коментаря для реалізації ієрархічної структури.
- created_at: TIMESTAMP – дата та час створення.

Описана структура є нормальною, забезпечує мінімальну надлишковість даних та є достатньою для реалізації всього запланованого функціоналу. Для підвищення продуктивності запитів необхідно буде додати

індекси на поля, що використовуються як зовнішні ключі, та на ті, що часто фігурують в умовах пошуку.

2.4. Обґрунтування вибору програмних засобів та технологій

Вибір технологічного стеку є одним із найбільш стратегічних рішень на етапі проєктування, оскільки він безпосередньо визначає архітектурну стійкість, швидкість розробки, а також подальшу масштабованість та супровід програмного продукту. Для реалізації веб-сайту для ведення блогу з інтегрованим модулем персоналізації необхідно було обрати набір інструментів, що здатен задовольнити дві ключові вимоги: забезпечити потужну підтримку наукомістких обчислень для модуля машинного навчання та гарантувати високу швидкість розробки інтерфейсу.

Ядром системи, її серверною частиною (Backend), було обрано мову програмування Python[16]. Цей вибір є практично безальтернативним для проєктів, що включають елементи Data Science, оскільки Python володіє найпотужнішою та найрозвиненішою у світі екосистемою для наукових обчислень. Для створення самого веб-додатку було використано мікро-фреймворк Flask[10][22]. Його філософія мінімалізму та гнучкості дозволила уникнути зайвої складності, характерної для монолітних фреймворків, і сфокусуватися на створенні чистого REST API. Для забезпечення повної функціональності бекенду було інтегровано низку важливих розширень. Flask-SQLAlchemy[12] було обрано як ORM (Object-Relational Mapper), що дозволяє керувати базою даних за допомогою Python-класів, значно підвищуючи безпеку та швидкість роботи з даними. Система автентифікації була реалізована за допомогою двох стандартних розширень: Flask-Login[17] для управління сесіями користувачів, захисту сторінок та реалізації функціоналу входу/виходу, та Flask-Bcrypt, який використовує надійний алгоритм bcrypt для безпечного хешування та зберігання паролів, що є критичним аспектом інформаційної безпеки.

Ключову роль в інтелектуальному модулі відіграють бібліотеки для машинного навчання. Scikit-learn[11] було обрано як стандарт галузі, що надає вже оптимізовані та готові реалізації алгоритму TF-IDF та функції косинусної подібності, дозволяючи зосередитися на логіці рекомендацій, а не на низькорівневих обчисленнях. Хоча NumPy та Pandas не використовувались безпосередньо для кожного запиту, їхня наявність у віртуальному середовищі є обов'язковою для будь-яких операцій з векторними та табличними даними, що підкреслює наукову спрямованість проєкту. Додатково для роботи з контентом було інтегровано бібліотеку Markdown[18], яка на сервері перетворює розмітку тексту на безпечний HTML-код, що є необхідним кроком для коректного відображення форматованих статей.

Клієнтська частина (Frontend) була побудована на класичному, перевіреному часом стеку HTML5, CSS3 та JavaScript (ES6+)[19][20][21]. Цей вибір забезпечив максимальну прозорість коду та уникнення залежності від складних JS-фреймворків. Для ефективної та швидкої реалізації адаптивного дизайну було обрано CSS-фреймворк Bootstrap[13]. Його вбудована система сіток та велика бібліотека готових UI-компонентів дозволили створити інтерфейс, що автоматично адаптується до будь-якої роздільної здатності екрана. Більше того, вбудована підтримка темних тем у Bootstrap 5 стала основою для реалізації функції перемикання теми на стороні клієнта. Для створення контенту було інтегровано SimpleMDE — легкий WYSIWYG-редактор, що працює з Markdown[14]. Його вибір був зумовлений відсутністю вимог до API-ключів та гнучкістю, що дозволило легко налаштувати його для коректної роботи з динамічною темною темою.

Для зберігання даних було обрано SQLite[12]. Ця вбудована СУБД є ідеальним рішенням для етапу розробки, оскільки вона не вимагає налаштування окремого серверного процесу і зберігає всю базу в одному файлі, що значно прискорює процес прототипування.

Таблиця 2.1 Загальна таблиця обраних технологій

Компонент	Технологія	Обґрунтування вибору
Backend	Python + Flask	Простота, гнучкість, потужна екосистема для ML
Автентифікація	Flask-Login, Flask-Bcrypt	Стандартні, надійні та перевірені рішення для Flask.
Machine Learning	Scikit-learn	Готові реалізації алгоритмів TF-IDF та косинусної подібності.
Обробка контенту	Markdown, SimpleMDE	Зручний формат розмітки та легкий WYSIWYG-редактор без API-ключів.
Frontend	HTML, CSS, JavaScript	Простота, повний контроль, відсутність зайвої складності фреймворків
Адаптивний дизайн	Bootstrap	Надійна система сіток, готові компоненти, вбудована підтримка тем.
База даних	SQLite + Flask-SQLAlchemy	Нульова конфігурація для розробки, зручна робота через ORM.

Таким чином, обраний технологічний стек є комплексним та збалансованим рішенням, що поєднує потужність наукових бібліотек Python для реалізації інтелектуального модуля з простотою та надійністю перевірених часом інструментів для веб-розробки, що дозволило ефективно вирішити всі поставлені в рамках проєкту задачі.

2.5. Висновки до розділу 2

У другому розділі кваліфікаційної роботи було виконано етап комплексного проєктування веб-сайту для ведення блогу з інтегрованим модулем персоналізації контенту. Результатом цього етапу є повний набір проектних рішень, що є теоретичним та інженерним фундаментом для подальшої програмної реалізації системи.

Було розроблено загальну архітектуру програмного продукту, засновану на класичній клієнт-серверній моделі з використанням REST API для взаємодії. Дана архітектура передбачає чітке розмежування відповідальності між клієнтською частиною (Frontend) та серверною частиною (Backend), що реалізує всю бізнес-логіку. Для забезпечення гнучкості та модульності, модуль персоналізації було спроєктовано як автономний логічний компонент.

Центральним елементом проєктної роботи стала розробка детальної математичної моделі та покрокового алгоритму функціонування сервісу рекомендацій. Модель, що базується на векторному представленні текстових документів за допомогою метрики TF-IDF та розрахунку подібності через косинусну міру, є формалізованою та готовою до імплементації. Розроблений двохетапний алгоритм забезпечує високу продуктивність системи в реальному часі.

Також було спроєктовано логічну структуру бази даних. Розроблена ER-діаграма включає всі необхідні сутності для зберігання контенту, даних про користувачів та їхні взаємодії, включаючи механізми для реалізації системи лайків та коментарів, а також передбачає правила каскадного видалення для забезпечення цілісності даних.

На завершення було проведено детальний аналіз та обґрунтування вибору технологічного стеку. Обраний набір технологій є комплексним рішенням: для серверної частини це Python з фреймворком Flask та його розширеннями (Flask-SQLAlchemy, Flask-Login, Flask-Bcrypt); для реалізації ML-модуля — бібліотека Scikit-learn; для клієнтської частини — класичний стек HTML/CSS/JS з фреймворком Bootstrap та WYSIWYG-редактором SimpleMDE; для бази даних — SQLite. Цей стек є збалансованим, сучасним та оптимальним рішенням з огляду на вимоги проєкту. Таким чином, у другому розділі було створено повну технічну та проектну документацію, що дозволяє перейти до етапу безпосередньої програмної реалізації системи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВЕБ-САЙТУ ДЛЯ ВЕДЕННЯ БЛОГУ «RE:ACTION»

3.1. Створення проекту та налаштування робочого середовища

Вибір назви для програмного продукту, «Re:Action», не є випадковим. Він відображає центральну концепцію, закладену в архітектуру веб-сайту: створення динамічної системи, що реагує на дії користувача. Назва є грою слів, що поєднує "Reaction" (реакція) — ключові дії користувача, такі як перегляди та вподобання, які є даними для аналізу, — та "Re: Act" (повторна дія), що символізує дію системи у відповідь, а саме надання персоналізованого контенту.

Для реалізації програмного коду було обрано інтегроване середовище розробки (IDE) Visual Studio Code (VSC). Це сучасний, легкий та багатофункціональний редактор коду, який має низку переваг для даного проєкту. По-перше, він надає відмінну підтримку мови Python, включаючи підсвітку синтаксису, автодоповнення коду (IntelliSense), інструменти для відлагодження та інтеграцію з лінтерами. По-друге, VSC має вбудований термінал, що дозволяє виконувати всі необхідні команди (активація віртуального середовища, встановлення залежностей, запуск сервера) в єдиному вікні, не перемикаючись між різними програмами. По-третє, завдяки величезній екосистемі розширень, середовище легко налаштовується для роботи з усіма технологіями, що використовуються в проєкті, включаючи HTML, CSS, JavaScript та SQLite.

Перехід від етапу проєктування до безпосередньої програмної реалізації розпочався з формування надійної та масштабованої структури проєкту. Фундаментом для розробки став патерн "фабрика додатку" (application factory), який є стандартом для сучасних Flask-проєктів[10][22]. Цей підхід передбачає створення екземпляра додатку всередині спеціальної функції, що дозволяє уникнути глобальних змінних та циклічних імпортів, а також спрощує конфігурацію та тестування. Відповідно до цього патерну, було створено основний програмний пакет blog, який інкапсулює всю логіку

додатку. Всередині цього пакету було створено логічно розділені модулі у вигляді окремих Python-файлів: models.py для визначення моделей даних, auth.py для логіки автентифікації, posts.py для управління контентом, users.py для профілів користувачів та main.py для основних маршрутів. Така модульна структура забезпечує високий рівень організованості коду та спрощує його подальшу підтримку.

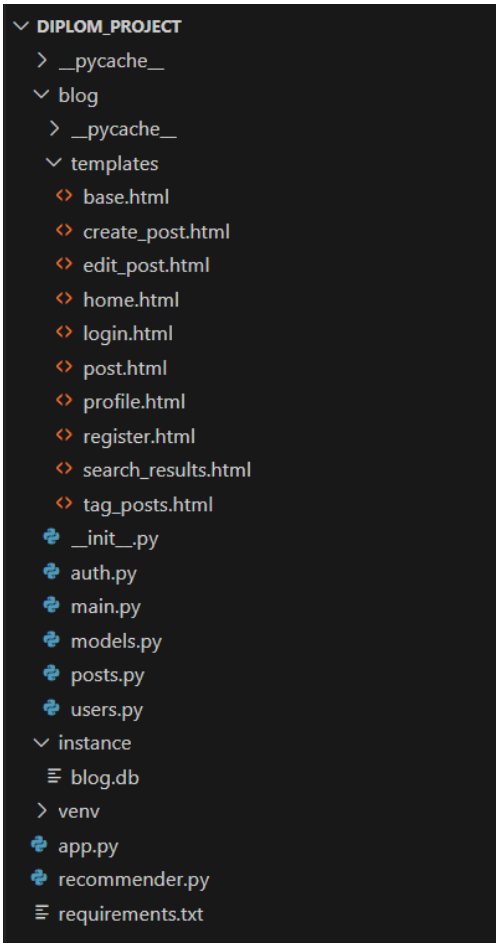


Рис. 3.1 Структура папок та файлів проєкту в середовищі розробки Visual Studio Code

Ключовим кроком у підготовці робочого середовища стало створення віртуального оточення за допомогою стандартного модуля Python venv[16]. Використання віртуального середовища є критично важливою практикою, що дозволяє створити ізольовану "пісочницю" для кожного окремого проєкту. Всі залежності — бібліотеки та фреймворки — встановлюються всередину цієї ізольованої директорії, а не в глобальне середовище

операційної системи. Це гарантує, що версії бібліотек, необхідні для даного проєкту (наприклад, Flask 3.0), не будуть конфліктувати з іншими версіями, що можуть знадобитися для інших проєктів. Процес створення та активації віртуального середовища виконувався за допомогою стандартних команд python -m venv venv та .\venv\Scripts\activate у вбудованому терміналі середовища розробки.

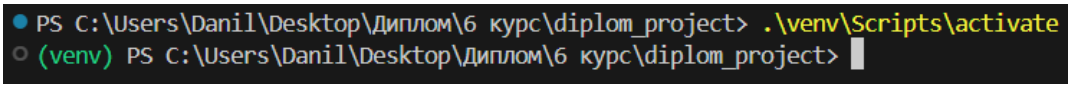


Рис. 3.2 Процес активації віртуального середовища venv у терміналі PowerShell

Для забезпечення відтворюваності робочого середовища та спрощення процесу розгортання проєкту було створено файл requirements.txt. Цей файл є декларативним списком всіх зовнішніх бібліотек, від яких залежить проєкт, із зазначенням їхніх точних версій. Він був згенерований автоматично за допомогою команди pip freeze > requirements.txt, виконаної в активованому віртуальному середовищі. Наявність цього файлу дозволяє будь-якому розробнику налаштувати ідентичне робоче оточення однією командою pip install -r requirements.txt, що унеможливлює виникнення помилок, пов'язаних з несумісністю версій бібліотек.

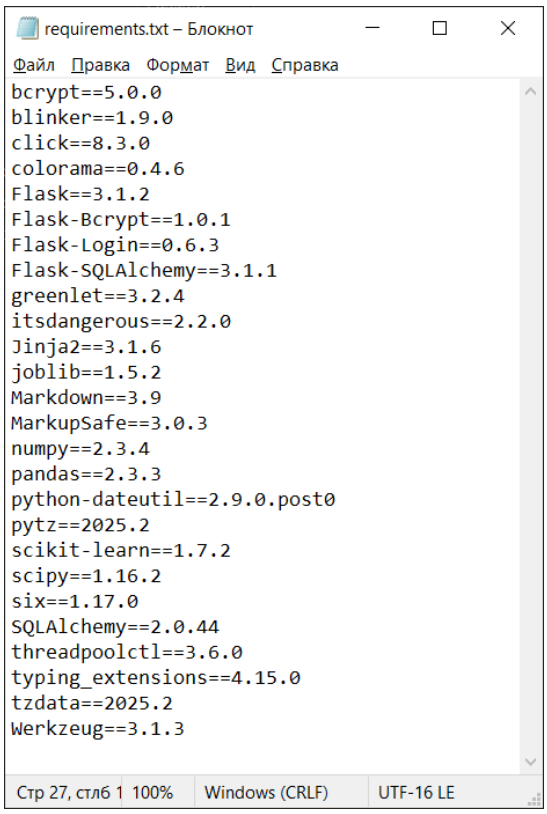


Рис. 3.3 Фрагмент згенерованого файлу requirements.txt з переліком залежностей проєкту

Завершальним етапом налаштування робочого середовища стала програмна реалізація патерну "фабрика додатку" у файлі blog/__init__.py. Цей файл виконує роль "складального цеху" для всього проєкту: він імпортує необхідні компоненти, ініціалізує розширення Flask та реєструє модулі (Blueprints), що містять логіку обробки запитів.

Центральним елементом файлу є функція create_app(), яка інкапсулює весь процес створення та конфігурації екземпляра додатку. На початку цієї функції створюється екземпляр класу Flask, після чого відбувається його конфігурація. Зокрема, встановлюється SECRET_KEY, що є критично важливим для захисту сесій користувачів та підпису криптографічних токенів, а також SQLALCHEMY_DATABASE_URI, що вказує додатку шлях до файлу бази даних SQLite.

Далі відбувається ініціалізація всіх розширень Flask. Для кожного розширення, такого як SQLAlchemy для роботи з базою даних, Bcrypt для хешування паролів та LoginManager для управління автентифікацією,

створюється глобальний екземпляр, який потім прив'язується до конкретного екземпляра додатку за допомогою методу .init_app(app). Такий підхід дозволяє уникнути жорсткої прив'язки розширень до глобального об'єкта app, що є ключовою перевагою патерну "фабрики". Також у цьому файлі визначається функція load_user, декорована @login_manager.user_loader, яка є обов'язковою для Flask-Login і відповідає за завантаження об'єкта користувача з бази даних за його ID, що зберігається в сесії.

```python
def create_app():
    app = Flask(__name__, instance_relative_config=True)

    @app.template_filter('markdown')
    def markdown_filter(s):
        return markdown.markdown(s)

    app.config['SECRET_KEY'] = 'your_very_secret_key_123'
    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///blog.db'

    db.init_app(app)
    bcrypt.init_app(app)
    login_manager.init_app(app)

    from .auth import auth as auth_blueprint
    app.register_blueprint(auth_blueprint)

    from .main import main as main_blueprint
    app.register_blueprint(main_blueprint)

    from .posts import posts as posts_blueprint
    app.register_blueprint(posts_blueprint)

    from .users import users as users_blueprint
    app.register_blueprint(users_blueprint)

    return app
```

Рис. 3.4 Фрагмент коду "фабрики додатку" з файлу blog/__init__.py, що демонструє ініціалізацію розширень

Після ініціалізації всіх сервісів відбувається реєстрація модулів Blueprint. Кожен логічний блок нашого додатку (auth, main, posts, users) попередньо визначається як об'єкт Blueprint у відповідному файлі. У функції create_app() ці "креслення" імпортуються та реєструються в основному додатку за допомогою методу app.register_blueprint(). Цей механізм дозволяє ефективно структурувати великий додаток, розбиваючи його на незалежні компоненти з власними маршрутами, шаблонами та логікою.

Нарешті, головний файл проєкту, app.py, що знаходиться в кореневій директорії, було значно спрощено. Його єдина відповідальність тепер — імпортувати функцію-фабрику create_app з пакету blog та запустити створений нею додаток. Такий підхід повністю відокремлює конфігурацію та ініціалізацію від безпосереднього запуску, що є найкращою практикою в розробці на Flask. У цьому ж файлі було реалізовано логіку для автоматичного створення всіх таблиць бази даних при першому запуску додатку, що спрощує процес початкового розгортання.

Таким чином, налаштоване робоче середовище та структура проєкту повністю відповідають сучасним стандартам професійної веб-розробки. Вони забезпечують високий рівень модульності, спрощують подальшу розробку та тестування, а також закладають надійний фундамент для всієї подальшої програмної реалізації функціоналу веб-сайту «Re:Action».

3.2. Реалізація моделей даних та конфігурація бази даних

Основою будь-якого динамічного веб-додатку є його модель даних, яка визначає структуру, атрибути та взаємозв'язки ключових сутностей. У даному проєкті для взаємодії з базою даних було використано розширення Flask-SQLAlchemy, що реалізує патерн ORM (Object-Relational Mapping)[12]. Цей підхід дозволяє представити таблиці реляційної бази даних у вигляді Python-класів, а операції з даними (створення, читання, оновлення, видалення) виконувати за допомогою об'єктно-орієнтованих методів, що значно спрощує розробку та підвищує надійність коду.

Вся логіка визначення моделей даних була інкапсульована в окремому модулі blog/models.py. Кожен клас у цьому файлі, що успадковується від базового класу db.Model, автоматично відображається у відповідну таблицю в базі даних SQLite. Цей підхід, відомий як "Code First", дозволяє керувати структурою бази даних безпосередньо з Python-коду, що є надзвичайно зручним на етапі розробки та подальшої підтримки проєкту.

Центральною моделлю системи є клас User, що відповідає за зберігання даних про користувачів. Окрім стандартних атрибутів, таких як унікальний ідентифікатор, ім'я користувача, email та хеш пароля, цей клас було розширено для інтеграції з іншими модулями системи. По-перше, він успадковується від UserMixin — спеціального "домішкового" класу з бібліотеки Flask-Login[17], який додає стандартні методи, необхідні для роботи системи автентифікації (наприклад, is_authenticated, is_active). По-друге, в моделі реалізовано метод avatar(), який на основі хешу MD5 від email-адреси користувача генерує посилання на його зображення через зовнішній сервіс Gravatar. Це дозволило реалізувати функціонал аватарів без необхідності створювати власну складну систему завантаження та зберігання файлів.

```python
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin, current_user
import datetime
from hashlib import md5

db = SQLAlchemy()

post_tags = db.Table('post_tags',
    db.Column('post_id', db.Integer, db.ForeignKey('post.id'), primary_key=True),
    db.Column('tag_id', db.Integer, db.ForeignKey('tag.id'), primary_key=True)
)

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    email = db.Column(db.String(255), unique=True, nullable=False)
    password_hash = db.Column(db.String(255), nullable=False)
    posts = db.relationship('Post', backref='author', lazy=True)
    def avatar(self, size):
        digest = md5(self.email.lower().encode('utf-8')).hexdigest()
        return f'https://www.gravatar.com/avatar/{digest}?d=identicon&s={size}'
    def __repr__(self):
        return f'<User {self.username}>'
```

Рис. 3.5 Фрагмент коду з файлу blog/models.py, що демонструє визначення моделі User

Основною контентною сутністю системи є модель Post, що програмно представляє статтю в блозі. Цей клас містить атрибути, що відповідають основним полям статті: title для заголовка, content для основного тексту, excerpt для короткої анотації та published_at для фіксації дати публікації.

Зв'язок з автором реалізовано через поле author_id, яке є зовнішнім ключем (ForeignKey) до таблиці users. Для зручності доступу до об'єкта автора з об'єкта поста, у зв'язку db.relationship в моделі User було використано параметр backref='author'. Це дозволяє звертатися до автора статті за допомогою простого виразу post.author, що робить код у шаблонах та логіці більш читабельним.

Для організації та категоризації контенту було створено модель Tag, яка є простим довідником назв тегів, та проміжну асоціативну таблицю post_tags. Ця таблиця, визначена за допомогою конструкції db.Table, не є повноцінною моделлю-класом, оскільки не містить власних даних, окрім двох зовнішніх ключів — post_id та tag_id. Такий підхід є стандартним для реалізації зв'язку "багато-до-багатьох" в SQLAlchemy. У самій моделі Post цей зв'язок визначається через db.relationship з параметром secondary=post_tags. Ця конфігурація дозволяє SQLAlchemy автоматично керувати записами в проміжній таблиці. Наприклад, при додаванні об'єкта Tag до колекції post.tags SQLAlchemy самостійно створює відповідний запис у таблиці post_tags, що значно спрощує логіку програми.

Для функціонування системи рекомендацій та соціальних механізмів було реалізовано три додаткові моделі: Comment, Like та UserPostInteraction. Всі вони спроєктовані як "дочірні" сутності по відношенню до моделей User та Post і містять відповідні зовнішні ключі для встановлення зв'язків. Модель Comment додатково містить рекурсивний зв'язок parent_comment_id, що дозволяє реалізувати ієрархічну структуру відповідей на коментарі. Модель UserPostInteraction відіграє ключову роль у зборі даних для рекомендаційної системи, фіксуючи кожен факт перегляду або вподобання статті конкретним користувачем.

Особливу увагу при реалізації моделей було приділено забезпеченню цілісності даних при видаленні. Як було з'ясовано під час тестування, спроба видалити "батьківський" об'єкт (наприклад, Post), на який посилаються "дочірні" об'єкти (Comment, Like), призводить до помилки IntegrityError через обмеження зовнішніх ключів. Для вирішення цієї проблеми у визначенні

зв'язків на стороні моделі Post було використано параметр cascade="all, delete-orphan".

```python
class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(255), nullable=False)
    content = db.Column(db.Text, nullable=False)
    excerpt = db.Column(db.String(500), nullable=True)
    published_at = db.Column(db.DateTime, nullable=False, default=datetime.datetime.utcnow)
    author_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    tags = db.relationship('Tag', secondary=post_tags, backref='posts', lazy=True)
    comments = db.relationship('Comment', backref='post', lazy=True, cascade="all, delete-orphan")
    interactions = db.relationship('UserPostInteraction', backref='post', lazy=True, cascade="all, delete-orphan")
    likes = db.relationship('Like', backref='post', lazy='dynamic', cascade="all, delete-orphan")
    def is_liked_by(self, user):
        if not user.is_authenticated:
            return False
        return self.likes.filter_by(user_id=user.id).count() > 0
    def __repr__(self):
        return f'<Post "{self.title}">'
```

Рис 3.6 Приклад реалізації каскадного видалення у моделі Post

Ця опція надає інструкцію для SQLAlchemy: при видаленні будь-якого екземпляра Post, необхідно автоматично знайти та видалити всі пов'язані з ним записи в таблицях comments, likes та user_post_interactions. Аналогічний механізм каскадного видалення було налаштовано і для проміжної таблиці post_tags. Такий підхід гарантує, що в базі даних не залишиться "осиротілих" записів, які посилаються на неіснуючі об'єкти, та забезпечує коректну роботу функції видалення контенту без порушення цілісності даних.

Конфігурація підключення до бази даних була визначена у "фабриці додатку" create_app(), що знаходиться в модулі blog/__init__.py. Рядок конфігурації app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///blog.db' вказує SQLAlchemy використовувати драйвер SQLite та зберігати всю базу даних у файлі blog.db. Завдяки параметру instance_relative_config=True при створенні Flask-додатку, цей файл автоматично розміщується у спеціальній директорії instance, що відокремлює дані від кодової бази проєкту.

Процес створення бази даних та всіх визначених у моделях таблиць було автоматизовано. У головному файлі запуску app.py було додано блок коду, який перед першим запуском веб-сервера виконує команду db.create_all() в контексті додатку. Ця команда аналізує всі класи, що успадковуються від db.Model, та генерує відповідні SQL-команди CREATE

TABLE для створення повної структури бази даних. Такий підхід значно спрощує процес початкового налаштування та розгортання системи, оскільки не вимагає ручного виконання SQL-скриптів.

3.3. Розробка модуля автентифікації та управління профілями користувачів

Надійна система автентифікації є фундаментальним компонентом будь-якого веб-додатку, що передбачає взаємодію з зареєстрованими користувачами. Вона відповідає за перевірку особистості користувача, управління доступом до захищених ресурсів та підтримку стану сесії. У проєкті «Re:Action» вся логіка, пов'язана з автентифікацією (реєстрація, вхід, вихід) та управлінням профілями, була винесена в окремі, логічно ізольовані модулі blog/auth.py та blog/users.py відповідно, що відповідає принципам модульної архітектури.

Для реалізації цього функціоналу було використано два спеціалізовані розширення Flask: Flask-Login та Flask-Bcrypt[17]. Flask-Login надає повний інструментарій для управління сесіями користувачів. Після успішної перевірки облікових даних, розширення створює для користувача захищену сесію, зберігаючи його унікальний ідентифікатор у cookies браузера. При кожному наступному запиті Flask-Login автоматично завантажує об'єкт користувача з бази даних, роблячи його доступним через глобальний об'єкт current_user. Це дозволяє легко перевіряти статус автентифікації користувача (current_user.is_authenticated) в будь-якій частині додатку. Також Flask-Login надає зручний декоратор @login_required[17], який дозволяє захистити доступ до певних сторінок, автоматично перенаправляючи неавторизованих користувачів на сторінку входу.

Критично важливим аспектом є безпечне зберігання паролів. Зберігати паролі у відкритому вигляді є неприпустимим з точки зору інформаційної безпеки. Для вирішення цієї задачі було інтегровано розширення Flask-Bcrypt, яке реалізує надійний криптографічний алгоритм хешування

bcrypt[22]. При реєстрації нового користувача його пароль не зберігається в базі даних. Замість цього, він пропускається через функцію bcrypt.generate_password_hash(), яка перетворює його на хеш — довгий рядок символів фіксованої довжини, відновити з якого початковий пароль є обчислювально неможливою задачею. Саме цей хеш зберігається в полі password_hash таблиці users.

```python
@auth.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('main.home'))
    if request.method == 'POST':
        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')
        hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')
        new_user = User(username=username, email=email, password_hash=hashed_password)
        db.session.add(new_user)
        db.session.commit()
        login_user(new_user)
        flash(f"Акаунт для {username} успішно створено!", "success")
        return redirect(url_for('main.home'))
    return render_template('register.html')
```

Рис 3.7 Фрагмент коду з файлу blog/auth.py, що демонструє процес хешування пароля при реєстрації

Процес входу користувача на сайт, реалізований у функції login(), є зворотним до процесу реєстрації. Коли користувач вводить своє ім'я та пароль у форму входу, система спочатку знаходить у базі даних користувача з відповідним username. Якщо такий користувач існує, система переходить до перевірки пароля. Для цього використовується функція bcrypt.check_password_hash(). Ця функція приймає два аргументи: хеш, що зберігається в базі даних (user.password_hash), та пароль, щойно введений користувачем. Вона виконує ту ж саму операцію хешування над введеним паролем і порівнює отриманий результат зі збереженим хешем. Тільки якщо результати збігаються, перевірка вважається успішною. Такий підхід гарантує, що система ніколи не працює з паролем у відкритому вигляді після його введення користувачем.

У разі успішної автентифікації викликається функція login_user(user) з бібліотеки Flask-Login, яка створює для користувача захищену сесію. Якщо ж користувача з таким іменем не знайдено або перевірка пароля не вдалася, генерується flash-повідомлення про помилку, і користувач залишається на сторінці входу.

```
25  @auth.route('/login', methods=['GET', 'POST'])
26  def login():
27      if current_user.is_authenticated:
28          return redirect(url_for('main.home'))
29      if request.method == 'POST':
30          username = request.form.get('username')
31          password = request.form.get('password')
32          user = User.query.filter_by(username=username).first()
33          if user and bcrypt.check_password_hash(user.password_hash, password):
34              login_user(user)
35              flash("Ви успішно увійшли на сайт.", "success")
36              return redirect(url_for('main.home'))
37          else:
38              flash("Неправильне ім'я користувача або пароль.", "danger")
39      return render_template('login.html')
```

Рис 3.8 Фрагмент коду функції login(), що демонструє процес перевірки хешу пароля

Для розширення соціального функціоналу та покращення навігації було реалізовано сторінку профілю користувача, логіка якої винесена в окремий модуль blog/users.py. Кожна сторінка профілю має унікальну URL-адресу формату /user/<username>. При переході за такою адресою, система знаходить у базі даних користувача за його іменем. Якщо користувач знайдений, на сторінку передається об'єкт user, який містить всю необхідну інформацію для відображення: ім'я користувача, дату реєстрації, а також посилання на аватар, що генерується методом user.avatar().

Ключовим елементом сторінки профілю є список всіх статей, автором яких є даний користувач. Цей функціонал реалізовано за допомогою зв'язку db.relationship, визначеного в моделі User. Завдяки параметру backref='author', SQLAlchemy автоматично створює "зворотний" зв'язок, що дозволяє легко отримати доступ до всіх пов'язаних об'єктів Post через атрибут user.posts. У HTML-шаблоні profile.html використовується цикл для ітерації по цій

колекції та відображення списку статей у вигляді карток, що забезпечує консистентний вигляд з головною сторінкою сайту. Імена авторів у всіх частинах сайту (на головній сторінці, на сторінці статті) є гіперпосиланнями, що ведуть на відповідну сторінку профілю, створюючи таким чином зручну систему навігації між контентом та його авторами.

3.4. Реалізація основного функціоналу блогу: управління контентом

Центральним функціоналом будь-якої блог-платформи є можливість створення та управління контентом. У проєкті «Re:Action» цей функціонал реалізовано у вигляді повного CRUD-циклу (Create, Read, Update, Delete) для сутності "стаття" (Post). Вся логіка, пов'язана з управлінням статтями, коментарями, тегами та лайками, була інкапсульована в окремому модулі blog/posts.py, що відповідає принципам модульної архітектури.

Створення та редагування статей. Процес створення нової статті доступний лише авторизованим користувачам, що забезпечується за допомогою декоратора @login_required для відповідних маршрутів. Для зручності авторів, замість стандартного HTML-елемента <textarea>, було інтегровано легкий WYSIWYG (What You See Is What You Get) редактор SimpleMDE[14]. Цей JavaScript-редактор, що базується на синтаксисі Markdown, надає користувачам інтуїтивно зрозумілу панель інструментів для форматування тексту: створення заголовків, виділення жирним шрифтом чи курсивом, додавання списків, посилань та цитат.

При відправці форми, текст зі статті, що містить Markdown-розмітку, зберігається в базу даних у чистому вигляді. Для коректного відображення цього контенту на сторінках сайту було створено спеціальний фільтр для шаблонізатора Jinja2. Цей фільтр використовує Python-бібліотеку Markdown[18] для перетворення збереженої розмітки на HTML-код безпосередньо перед рендерингом сторінки. Такий підхід є більш безпечним та гнучким, ніж зберігання готового HTML в базі даних.

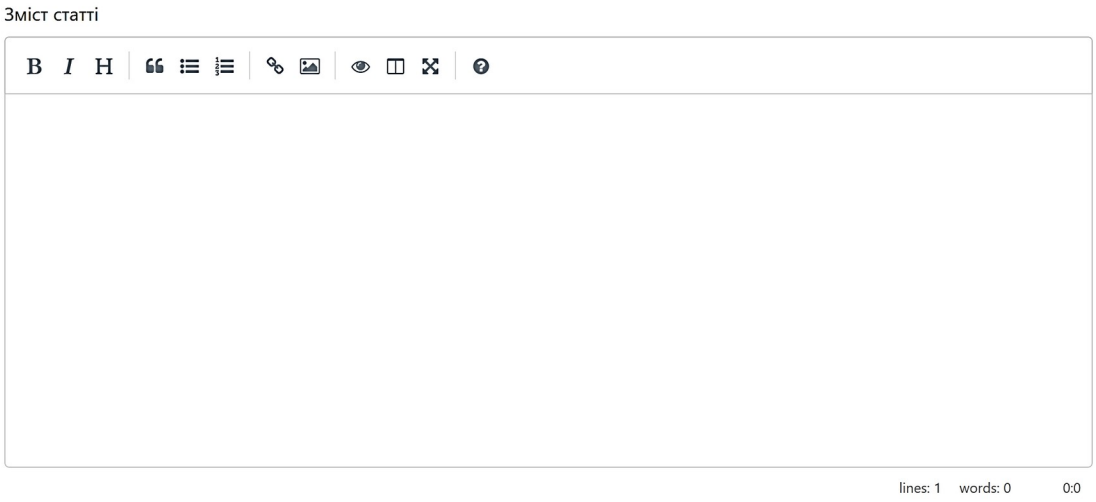


Рис 3.9 Інтерфейс створення нової статті з інтегрованим WYSIWYG-редактором SimpleMDE

Функціонал редагування статті, реалізований у функції edit_post(), використовує той же самий шаблон та редактор. Перед відображенням форми система завантажує з бази даних існуючий текст статті та передає його в шаблон, заповнюючи відповідні поля. Важливим аспектом безпеки є перевірка прав доступу: редагувати або видаляти статтю може виключно її автор. Ця перевірка реалізована шляхом порівняння current_user (об'єкт поточного авторизованого користувача) з post.author (об'єкт автора статті). У випадку спроби несанкціонованого доступу, система генерує помилку 403 (Forbidden).

Відображення статей. Читання статей реалізовано за допомогою двох основних маршрутів. Головна сторінка (/), логіка якої знаходиться в модулі blog/main.py, відображає список всіх статей у зворотному хронологічному порядку. Для підвищення продуктивності та зручності навігації при великій кількості контенту було реалізовано механізм пагінації, що розбиває список статей на сторінки. Окрема сторінка для кожної статті (/post/<int:post_id>) реалізована за допомогою динамічної маршрутизації, де унікальний ідентифікатор статті передається як параметр URL.

Для покращення навігації та структурування контенту було реалізовано систему категоризації статей за допомогою тегів. При створенні або

редагуванні статті автор може вказати список релевантних тегів. Для забезпечення зручного інтерфейсу введення було інтегровано JavaScript-бібліотеку Tagify[15]. Вона перетворює стандартне поле вводу на інтерактивний елемент, де кожен тег відображається у вигляді окремої "бульбашки". Ключовою особливістю реалізації є функція автодоповнення: коли користувач починає вводити назву тегу, система пропонує йому варіанти зі списку всіх тегів, що вже існують у базі даних. Це не лише прискорює процес, але й сприяє консистентності, запобігаючи створенню дублікатів (наприклад, "python" та "Python"). На серверній стороні логіка обробки тегів автоматично знаходить існуючі теги в базі або створює нові, якщо їх ще не існує, після чого встановлює зв'язок "багато-до-багатьох" зі статтею. Кожен тег на сторінці є гіперпосиланням, що веде на окрему сторінку, де зібрані всі статті, відмічені цим тегом.

Соціальна взаємодія на платформі реалізована через систему коментарів та лайків. Будь-який авторизований користувач може залишити коментар до статті. Для зручності, редагування та видалення коментарів реалізовано за допомогою модальних вікон Bootstrap[13], що дозволяє виконувати ці дії без перезавантаження сторінки. Система прав доступу гарантує, що редагувати або видаляти коментар може виключно його автор.

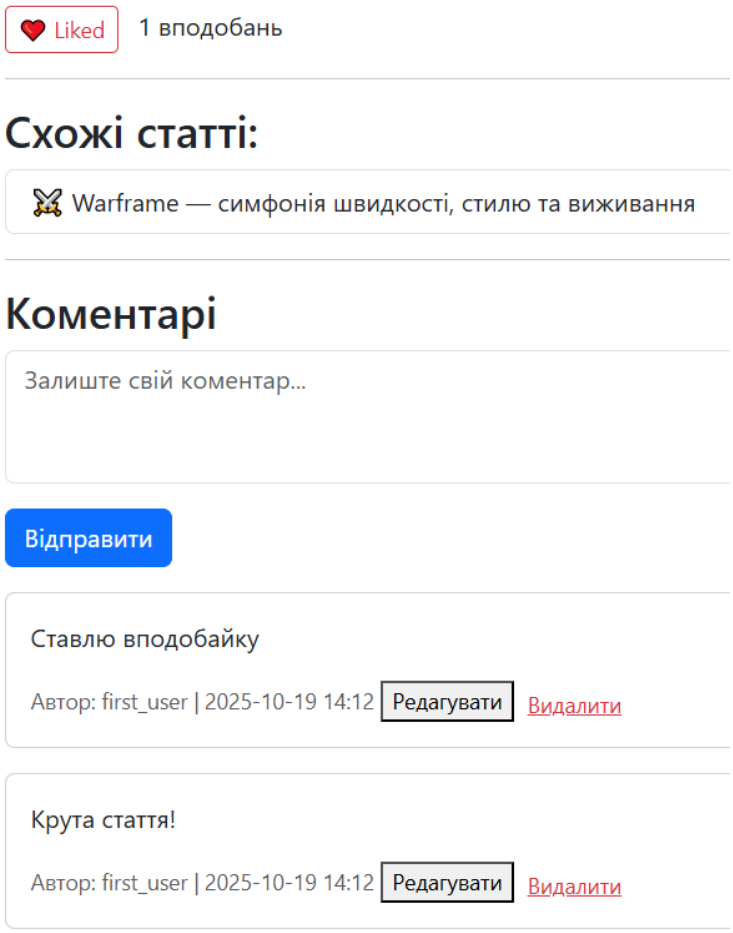


Рис 3.10 Реалізація системи коментарів та лайків на сторінці статті

Функціонал лайків реалізовано як простий перемикач: перше натискання на кнопку "Лайк" створює запис у таблиці likes, пов'язуючи користувача та статтю; повторне натискання видаляє цей запис. Кількість лайків для кожної статті обчислюється динамічно за допомогою запиту post.likes.count(). Важливо зазначити, що кожна дія (перегляд та лайк) не лише відображається в інтерфейсі, але й фіксується в таблиці user_post_interactions. Ця таблиця є джерелом даних для модуля персоналізації: лайк вважається значно сильнішим сигналом про зацікавленість, ніж простий перегляд, і отримує більшу вагу при побудові профілю інтересів користувача, що буде детальніше розглянуто в наступному підрозділі. Таким чином, реалізований функціонал управління контентом не лише забезпечує базові потреби блог-платформи, але й тісно інтегрований з інтелектуальним ядром системи.

3.5. Розробка та інтеграція модуля машинного навчання для персоналізованих рекомендацій

Інтелектуальне ядро веб-сайту «Re:Action», що реалізує методи штучного інтелекту для вирішення задачі персоналізації, було розроблено у вигляді автономного програмного модуля recommender.py. Його головне завдання — аналіз текстового контенту та поведінки користувачів для генерації двох типів персоналізованих рекомендацій: списку статей, схожих на поточну, та загальної стрічки рекомендованих статей для головної сторінки. Розробка модуля базувалася на класичному, але ефективному методі машинного навчання — фільтрації на основі вмісту (Content-Based Filtering) з використанням бібліотеки scikit-learn[11].

Програмна реалізація алгоритму, описаного в розділі 2.2, інкапсульована у двох основних функціях. Перша функція, get_recommendations(), відповідає за пошук статей, семантично близьких до тієї, яку користувач переглядає в даний момент. При її виклику, вона приймає на вхід об'єкт поточної статті та список всіх статей з бази даних. Спочатку відбувається етап векторизації: тексти всіх статей (заголовок + контент) перетворюються на числові TF-IDF вектори за допомогою класу TfidfVectorizer з scikit-learn[11][24]. Далі, за допомогою функції cosine_similarity, обчислюється матриця косинусної подібності[11][24] між всіма векторами. З цієї матриці вибирається рядок, що відповідає поточній статті, і на його основі формується відсортований список найбільш схожих статей, які і відображаються користувачу.

```python
def get_user_recommendations(user_history_posts, all_posts, num_recommendations=10):
    if not user_history_posts:
        return []

    post_texts = [p.title + " " + p.content for p in all_posts]
    tfidf_vectorizer = TfidfVectorizer()
    tfidf_matrix = tfidf_vectorizer.fit_transform(post_texts)

    post_id_to_index = {post.id: i for i, post in enumerate(all_posts)}

    history_indices = [post_id_to_index[p.id] for p in user_history_posts if p.id in post_id_to_index]
    if not history_indices:
        return []

    history_vectors = tfidf_matrix[history_indices]

    user_profile_vector = np.asarray(history_vectors.mean(axis=0))

    similarity_scores = cosine_similarity(user_profile_vector, tfidf_matrix)

    scores = list(enumerate(similarity_scores[0]))

    sorted_scores = sorted(scores, key=lambda item: item[1], reverse=True)
```

Рис 3.11 Фрагмент коду з файлу recommender.py, що демонструє процес векторизації та розрахунку подібності

Друга, більш складна функція, get_user_recommendations(), реалізує логіку для формування персоналізованої стрічки "Рекомендовані" на головній сторінці. На відміну від першої функції, вона аналізує не одну статтю, а весь профіль інтересів користувача. Для побудови цього профілю система спочатку збирає з бази даних історію всіх взаємодій поточного користувача — переглядів (view) та лайків (like).

Ключовою особливістю реалізації є зважування сигналів. Було прийнято рішення, що лайк є значно сильнішим індикатором зацікавленості, ніж простий перегляд. Це реалізовано програмно: при формуванні історії для аналізу, ідентифікатор статті, яку користувач лайкнув, додається до списку з більшою вагою (у даній реалізації — тричі: один раз за факт взаємодії і двічі додатково за лайк). Далі, на основі TF-IDF векторів статей з цієї "зваженої" історії, обчислюється усереднений вектор, який і є числовим представленням профілю інтересів користувача. На фінальному етапі цей профільний вектор порівнюється з векторами всіх статей на сайті, і користувачу пропонуються ті, що мають найвищу косинусну подібність і які він ще не бачив.

Інтеграція цього модуля в основний додаток відбувається на рівні виклику відповідних функцій у контролерах (post() та home()). Важливо зазначити, що поточна реалізація виконує всі обчислення "на льоту" при

кожному запиті. Для проєкту даного масштабу такий підхід є прийнятним. Однак, для систем з великою кількістю контенту та користувачів, необхідно було б впроваджувати механізми кешування або виносити ресурсоємні обчислення TF-IDF векторів у фонові задачі, як це було передбачено на етапі проєктування.

Варто зазначити, що обрана реалізація методу фільтрації на основі вмісту, незважаючи на свою ефективність для вирішення проблеми "холодного старту", має певні іманентні обмеження. По-перше, якість рекомендацій безпосередньо залежить від якості та обсягу текстового контенту. Система може знаходити лише семантично схожі статті, але не здатна до "випадкових відкриттів" (serendipity), тобто не може рекомендувати статті з абсолютно нових для користувача тематик, які могли б його зацікавити. По-друге, поточна реалізація не враховує складніші семантичні зв'язки між словами (наприклад, синоніми або контекст), оскільки модель "мішок слів" розглядає кожне слово ізольовано.

Для подальшого вдосконалення рекомендаційного модуля можна розглянути декілька напрямків. Перший — це впровадження більш просунутих технік обробки природної мови (NLP), таких як стеммінг, лематизація або використання n-грам, що дозволить краще розуміти семантику тексту. Другий, більш складний напрямок — це перехід до гібридної моделі, яка б поєднувала поточний Content-Based підхід з елементами колаборативної фільтрації. Такий підхід дозволив би аналізувати поведінку схожих користувачів і генерувати більш різноманітні та несподівані рекомендації, що значно збагатило б користувацький досвід. Проте, для проєкту на даному етапі розробки, реалізований модуль є оптимальним та збалансованим рішенням, що повністю виконує поставлені задачі.

3.6. Реалізація користувацького інтерфейсу та адаптивного дизайну

Користувацький інтерфейс (UI) є обличчям будь-якого веб-додатку та безпосередньо впливає на досвід взаємодії користувача (UX). У проєкті «Re:Action» розробці інтерфейсу приділялася значна увага з метою створення чистого, інтуїтивно зрозумілого та адаптивного дизайну. Вся логіка представлення була реалізована за допомогою системи шаблонів Jinja2, інтегрованої у фреймворк Flask.

Основою для всіх сторінок сайту став базовий шаблон base.html. Цей підхід, відомий як успадкування шаблонів (template inheritance), дозволив уникнути дублювання коду та забезпечити консистентний вигляд всього додатку. У файлі base.html було визначено загальну HTML-структуру документа (<html>, <head>, <body>), підключено всі необхідні CSS-стилі та JavaScript-бібліотеки, а також реалізовано наскрізні елементи інтерфейсу, такі як навігаційна панель (navbar) та футер (за потреби). Центральним елементом цього шаблону є конструкція {% block content %}{% endblock %}, яка визначає "контейнер", куди всі дочірні шаблони (наприклад, home.html чи post.html) вставляють свій унікальний контент.

Для швидкої та ефективної розробки адаптивного дизайну було використано CSS-фреймворк Bootstrap 5[13]. Вибір цього фреймворку зумовлений його надійністю, чудовою документацією та, що найважливіше, потужною системою сіток (grid system), заснованою на Flexbox. Ця система дозволяє легко створювати гнучкі макети, які автоматично перебудовуються залежно від ширини екрана пристрою. Наприклад, на головній сторінці сайту було використано стандартні класи Bootstrap для карток (.card), контейнерів (.container) та кнопок (.btn), що одразу забезпечило охайний та адаптивний вигляд без необхідності написання великої кількості власного CSS-коду.

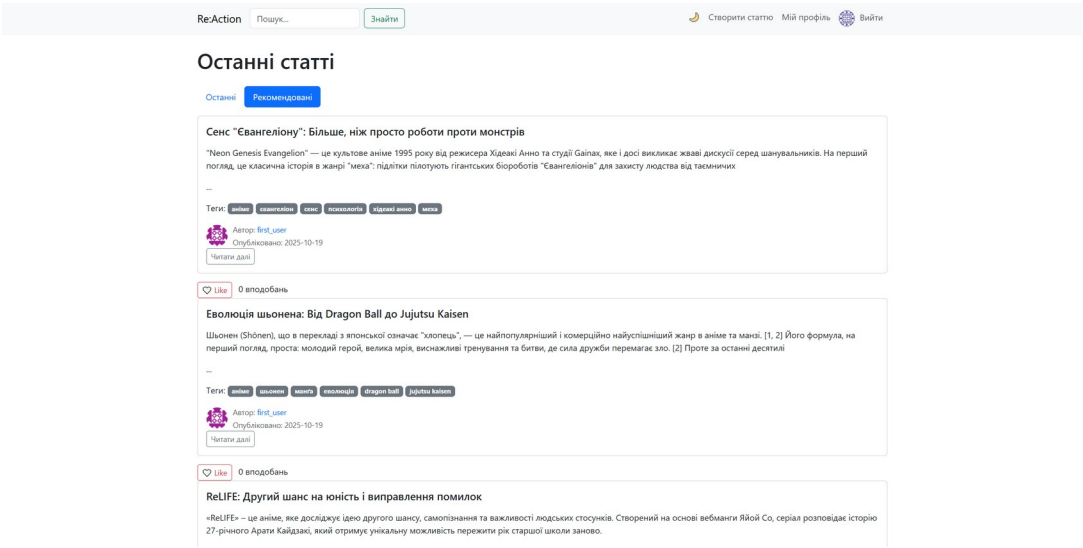


Рис 3.12 Вигляд головної сторінки на десктопному екрані

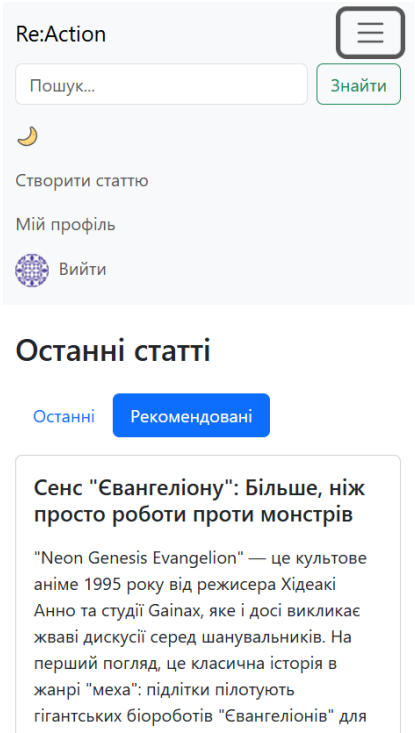


Рис 3.13 Вигляд головної сторінки на мобільному екрані

Окрім базової адаптивності, забезпеченої фреймворком Bootstrap, було реалізовано додатковий функціонал для покращення користувацького досвіду, а саме — можливість динамічного перемикання між світлою та темною темами інтерфейсу. Ця функція є особливо актуальною для контент-орієнтованих сайтів, оскільки дозволяє користувачам налаштувати вигляд сторінки для комфортного читання в різних умовах освітлення.

Реалізація цього функціоналу базується на вбудованій підтримці колірних режимів у Bootstrap 5. Фреймворк автоматично змінює свою палітру кольорів (фон, текст, кнопки, рамки), якщо до головного тегу <html> додати атрибут data-bs-theme="dark"[13]. Таким чином, задача зводиться до програмного управління цим атрибутом за допомогою JavaScript.

Для цього в базовий шаблон base.html було додано спеціальний скрипт. При завантаженні сторінки цей скрипт спочатку перевіряє localStorage браузера — невелике сховище даних на стороні клієнта. Він шукає, чи був раніше збережений вибір теми користувачем. Якщо так, він одразу застосовує збережену тему, що дозволяє уникнути "миготіння" світлої теми при завантаженні. Якщо збереженого вибору немає, за замовчуванням застосовується світла тема.

У навігаційній панелі було розміщено кнопку-перемикач. Скрипт "слухає" кліки по цій кнопці. При кожному кліку він визначає поточну тему, змінює її на протилежну (з light на dark або навпаки), застосовує новий атрибут до тегу <html> та, що важливо, зберігає новий вибір у localStorage, щоб "запам'ятати" його для майбутніх візитів користувача.

Особливу увагу було приділено інтеграції цієї функції зі сторонніми JavaScript-бібліотеками, такими як WYSIWYG-редактор SimpleMDE. Оскільки ці бібліотеки створюють власні елементи інтерфейсу, вони не реагують на зміну атрибуту data-bs-theme автоматично. Для вирішення цієї проблеми було розроблено додаткову логіку в JavaScript, яка при перемиканні теми динамічно додає до елементів редактора спеціальні CSS-класи (.editor-dark), що активують відповідні стилі для темного режиму, забезпечуючи таким чином візуальну консистентність всього інтерфейсу.

3.7. Тестування працездатності системи

Після завершення основного етапу розробки було проведено функціональне тестування для перевірки працездатності всіх ключових модулів веб-сайту «Re:Action». Тестування проводилося вручну за

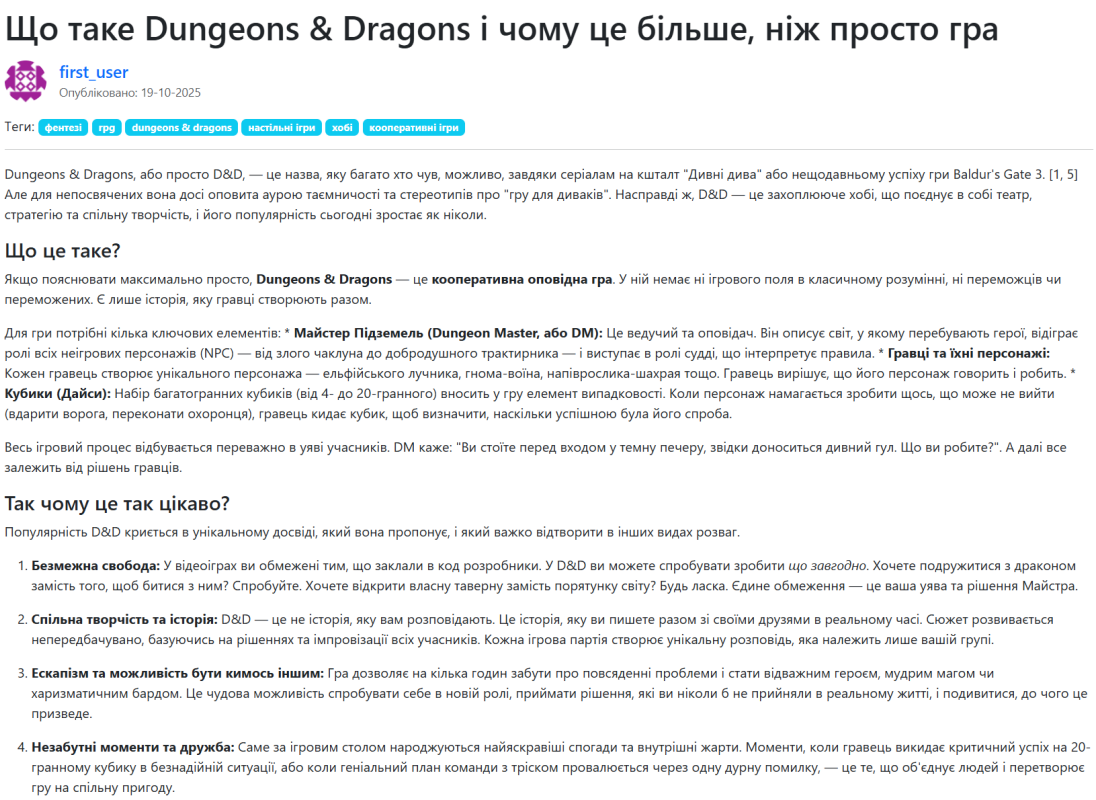
заздалегідь визначеними сценаріями (тест-кейсами), що охоплюють основні шляхи взаємодії користувача з системою. Метою тестування було підтвердження відповідності реалізованого функціоналу поставленим вимогам та виявлення потенційних помилок.

Сценарій 1: Реєстрація та автентифікація користувача.

- Кроки: 1. Перехід на сторінку /register; 2. Заповнення форми реєстрації валідними даними; 3. Натискання кнопки "Зареєструватися"; 4. Перевірка автоматичного входу та перенаправлення на головну сторінку; 5. Вихід з системи через відповідне посилання в навігації; 6. Перехід на сторінку /login; 7. Введення облікових даних щойно створеного користувача; 8. Перевірка успішного входу; 9. Спроба входу з невірним паролем.
- Очікуваний результат: Успішна реєстрація, автоматичний вхід, відображення flash-повідомлень про успішні дії. При спробі входу з невірним паролем система має показати повідомлення про помилку.
- Фактичний результат: Тестування пройдено успішно. Всі кроки виконано відповідно до очікуваного результату, система коректно обробляє як успішні сценарії, так і помилкові.

Сценарій 2: Управління контентом (CRUD для постів).

- Кроки: 1. Вхід в систему; 2. Перехід на сторінку створення статті (/post/new); 3. Заповнення форми, включаючи форматування тексту в редакторі SimpleMDE та додавання тегів; 4. Публікація статті; 5. Перевірка відображення нової статті на головній сторінці; 6. Перехід на сторінку статті, перевірка коректного відображення форматованого тексту та тегів; 7. Перехід на сторінку редагування; 8. Зміна заголовка та набору тегів, збереження змін; 9. Перевірка оновлення даних на сторінці статті; 10. Видалення статті; 11. Перевірка відсутності статті на головній сторінці.
- Очікуваний результат: Система дозволяє створювати, читати, оновлювати та видаляти статті. Форматування тексту та теги зберігаються та відображаються коректно.
- Фактичний результат: Тестування пройдено успішно. Весь CRUD-цикл працює без помилок.



Що таке Dungeons & Dragons і чому це більше, ніж просто гра

first_user
Опубліковано: 19-10-2025

Теги: фентезі | rpg | dungeons & dragons | настільні ігри | хобі | кооперативні ігри

Dungeons & Dragons, або просто D&D, — це назва, яку багато хто чув, можливо, завдяки серіалам на кшталт "Дивні дива" або нещодавньому успіху гри Baldur's Gate 3. [1, 5] Але для непосвячених вона досі оповита аурою таємничості та стереотипів про "гру для диваків". Насправді ж, D&D — це захоплююче хобі, що поєднує в собі театр, стратегію та спільну творчість, і його популярність сьогодні зростає як ніколи.

Що це таке?

Якщо пояснювати максимально просто, **Dungeons & Dragons** — це **кооперативна оповідна гра**. У ній немає ні ігрового поля в класичному розумінні, ні переможців чи переможених. Є лише історія, яку гравці створюють разом.

Для гри потрібні кілька ключових елементів: * **Майстер Підземель (Dungeon Master, або DM):** Це ведучий та оповідач. Він описує світ, у якому перебувають герої, відіграє ролі всіх неігрових персонажів (NPC) — від злого чаклуна до добродушного трактирника — і виступає в ролі судді, що інтерпретує правила. * **Гравці та їхні персонажі:** Кожен гравець створює унікального персонажа — ельфійського лучника, гнома-воїна, напівролика-шахрая тощо. Гравець вирішує, що його персонаж говорить і робить. * **Кубики (Дайси):** Набір багатогранних кубиків (від 4- до 20-гранного) вносить у гру елемент випадковості. Коли персонаж намагається зробити щось, що може не вийти (вдарити ворога, переконати охоронця), гравець кидає кубик, щоб визначити, наскільки успішною була його спроба.

Весь ігровий процес відбувається переважно в уяві учасників. DM каже: "Ви стоїте перед входом у темну печеру, звідки доноситься дивний гул. Що ви робите?". А далі все залежить від рішень гравців.

Так чому це так цікаво?

Популярність D&D криється в унікальному досвіді, який вона пропонує, і який важко відтворити в інших видах розваг.

1. **Безмежна свобода:** У відеоіграх ви обмежені тим, що заклали в код розробники. У D&D ви можете спробувати зробити *що завгодно*. Хочете подружитися з драконом замість того, щоб битися з ним? Спробуйте. Хочете відкрити власну таверну замість порятунку світу? Будь ласка. Єдине обмеження — це ваша уява та рішення Майстра.
2. **Спільна творчість та історія:** D&D — це не історія, яку вам розповідають. Це історія, яку ви пишете разом зі своїми друзями в реальному часі. Сюжет розвивається непередбачувано, базуючись на рішеннях та імпровізації всіх учасників. Кожна ігрова партія створює унікальну розповідь, яка належить лише вашій групі.
3. **Ескапізм та можливість бути кимось іншим:** Гра дозволяє на кілька годин забути про повсякденні проблеми і стати відважним героєм, мудрим магом чи харизматичним бардом. Це чудова можливість спробувати себе в новій ролі, приймати рішення, які ви ніколи б не прийняли в реальному житті, і подивитися, до чого це призведе.
4. **Незабутні моменти та дружба:** Саме за ігровим столом народжуються найяскравіші спогади та внутрішні жарти. Моменти, коли гравець викидає критичний успіх на 20-гранному кубику в безнадійній ситуації, або коли геніальний план команди з тріском провалюється через одну дурну помилку, — це те, що об'єднує людей і перетворює гру на спільну пригоду.

Рис 3.14 Результат тестування відображення форматованого тексту та тегів на сторінці статті

Сценарій 3: Тестування соціального функціоналу (коментарі та лайки).

- Кроки: 1. Вхід в систему; 2. Перехід на сторінку статті; 3. Додавання коментаря; 4. Перевірка відображення коментаря; 5. Редагування власного коментаря через модальне вікно; 6. Видалення власного коментаря; 7. Встановлення та зняття лайка; 8. Перевірка оновлення лічильника лайків та вигляду кнопки; 9. Вхід під іншим користувачем та перевірка відсутності кнопок редагування/видалення для чужих коментарів.
- Очікуваний результат: Користувачі можуть додавати коментарі, ставити лайки та керувати лише власним контентом.
- Фактичний результат: Тестування пройдено успішно. Система коректно розмежовує права доступу.

Сценарій 4: Тестування модуля рекомендацій та пошуку.

- Кроки: 1. Вхід в систему; 2. Послідовний перегляд кількох статей на схожу тематику (наприклад, про Python); 3. Перехід на головну сторінку та

активація стрічки "Рекомендовані"; 4. Аналіз запропонованих статей; 5. Встановлення лайка статті на іншу тематику (наприклад, про дизайн); 6. Повторна перевірка стрічки "Рекомендовані"; 7. Використання форми пошуку для пошуку за ключовим словом, тегом та іменем автора.

- Очікуваний результат: Стрічка "Рекомендовані" відображає статті, релевантні історії переглядів, при цьому лайки мають сильніший вплив на результат. Пошук коректно знаходить та відображає відповідні статті.
- Фактичний результат: Тестування підтвердило працездатність обох модулів. Рекомендаційна система адекватно реагує на дії користувача, а пошук знаходить релевантні результати за різними критеріями.

Загалом, проведене тестування підтвердило, що всі основні функціональні модулі веб-сайту «Re:Action» працюють коректно та відповідно до поставлених вимог.

3.8. Аналіз проблем, що виникли під час розробки, та відомі обмеження

У процесі розробки та тестування веб-сайту «Re:Action» було виявлено та вирішено низку технічних проблем, а також визначено певні обмеження поточної реалізації, що є природною частиною ітеративного процесу створення програмного продукту.

Однією з ключових проблем, що потребувала детального аналізу, стало забезпечення цілісності даних при видаленні об'єктів. Початкова реалізація призводила до помилки IntegrityError при спробі видалити статтю, до якої були прив'язані коментарі або записи про взаємодію. Це було пов'язано з обмеженнями зовнішніх ключів у реляційній базі даних. Проблема була вирішена на рівні моделей даних шляхом конфігурації каскадного видалення (опція cascade="all, delete-orphan" в SQLAlchemy). Це рішення дозволило автоматизувати процес видалення всіх "дочірніх" записів при видаленні "батьківського", що забезпечило стабільну та передбачувану роботу функціоналу.

Інша значна проблема виникла при інтеграції сторонніх JavaScript-бібліотек з динамічною системою перемикання тем. Зокрема, WYSIWYG-редактор SimpleMDE та бібліотека для роботи з тегами Tagify не підтримували зміну колірної схеми "з коробки". Це призводило до візуальних дефектів, коли при увімкненні темної теми елементи редакторів залишалися світлими, що погіршувало користувацький досвід. Проблема була вирішена шляхом розробки кастомного JavaScript-коду, який "слухає" подію зміни теми та динамічно додає до елементів редакторів спеціальні CSS-класи. Для цих класів, у свою чергу, були написані додаткові CSS-правила, що перевизначають стандартні стилі бібліотек для відповідності темній палітрі Bootstrap.

Незважаючи на успішну інтеграцію темної теми для більшості елементів, деякі компоненти сторонніх бібліотек виявилися особливо стійкими до перепризначення стилів. Як видно на рисунку 3.15, випадаючий список автодоповнення тегів у бібліотеці Tagify зберіг світлий фон навіть у темному режимі, що створює візуальний дисонанс. Це є прикладом відомого обмеження, коли глибока кастомізація стороннього компонента вимагає значно більших зусиль, ніж його базова інтеграція.

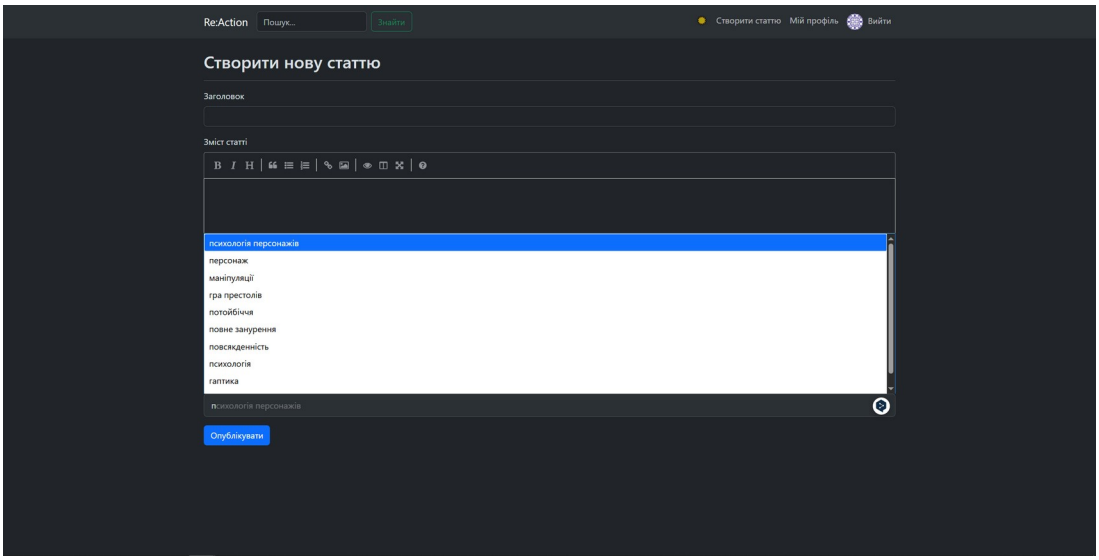


Рис 3.15 Приклад візуального артефакту: світлий випадаючий список Tagify у темному режимі інтерфейсу

Також варто відзначити відомі обмеження поточної реалізації в інших аспектах. По-перше, рекомендаційний модуль, хоч і є функціональним, виконує всі обчислення "на льоту" при кожному запиті. Для сайту з великим обсягом контенту це може призвести до зниження продуктивності. У майбутньому цей модуль потребуватиме оптимізації шляхом кешування результатів або винесення ресурсоємних обчислень TF-IDF векторів у фонові задачі. По-друге, реалізована система пошуку є базовою і не використовує сучасні підходи на основі ШІ, такі як семантичний пошук, виконуючи його за простим входженням підрядка, не враховуючи морфологію слів чи їхню релевантність. Для більш просунутого пошуку необхідно було б інтегрувати спеціалізовані пошукові рушії, такі як Elasticsearch або Whoosh. Однак, для цілей даного дипломного проєкту, поточна реалізація є достатньою та повністю демонструє основні принципи роботи системи.

3.9. Висновки до розділу 3

У третьому розділі кваліфікаційної роботи було виконано етап практичної програмної реалізації веб-сайту для ведення блогу «Re:Action» та проведено тестування його основного функціоналу. Цей етап базувався на проектних рішеннях та архітектурі, розроблених у попередньому розділі, і його результатом є повноцінний, функціональний програмний продукт.

Процес розробки розпочався з налаштування робочого середовища та створення модульної структури проєкту за патерном "фабрика додатку" у фреймворку Flask. Було реалізовано повний набір моделей даних за допомогою Flask-SQLAlchemy, що точно відображають спроєктовану ER-діаграму, та налаштовано механізми каскадного видалення для забезпечення цілісності даних. На основі цих моделей було розроблено ключові функціональні модулі системи.

Було успішно реалізовано надійний модуль автентифікації користувачів, що включає функціонал реєстрації, входу, виходу та управління сесіями за допомогою розширень Flask-Login та Flask-Bcrypt.

Також було створено сторінки профілів користувачів з відображенням їхніх статей та інтеграцією аватарів через сервіс Gravatar. Основний функціонал блогу було реалізовано у вигляді повного CRUD-циклу для статей, доповненого зручним WYSIWYG-редактором SimpleMDE для форматування контенту. Для покращення навігації та соціальної взаємодії було впроваджено системи тегів, коментарів та лайків.

Ключовим етапом розробки стала програмна реалізація та інтеграція інтелектуального модуля персоналізованих рекомендацій. Було створено автономний модуль recommender.py, що інкапсулює логіку Content-Based Filtering. Цей модуль використовує бібліотеку scikit-learn для векторизації текстового контенту статей за допомогою моделі TF-IDF та розрахунку їхньої семантичної подібності. Було реалізовано два типи рекомендацій: пошук схожих статей на сторінці поточного поста та формування персоналізованої стрічки "Рекомендовані" для головної сторінки. Важливою особливістю реалізації є механізм зважування сигналів, де "лайк" отримує більшу вагу, ніж простий перегляд, що дозволяє системі точніше визначати справжні інтереси користувача.

Паралельно було розроблено користувацький інтерфейс, що базується на CSS-фреймворку Bootstrap 5. Було забезпечено повну адаптивність дизайну, що гарантує коректне відображення сайту на пристроях з різною роздільною здатністю екрана. Окрім цього, було успішно інтегровано функціонал динамічного перемикання між світлою та темною темами, що значно покращує користувацький досвід.

Проведене наприкінці розробки функціональне тестування за ключовими сценаріями підтвердило працездатність всіх реалізованих модулів. Система коректно виконує всі заявлені функції: від реєстрації користувачів та управління контентом до генерації персоналізованих рекомендацій. Також у ході розробки було проаналізовано та вирішено низку технічних проблем, пов'язаних із цілісністю даних та інтеграцією сторонніх бібліотек.

Таким чином, у третьому розділі було продемонстровано повний цикл розробки програмного продукту — від налаштування середовища до реалізації складної бізнес-логіки, інтеграції модуля машинного навчання та фінального тестування. Результатом є готовий до використання веб-сайт, що повністю відповідає меті та задачам, поставленим у даній кваліфікаційній роботі.

ЗАГАЛЬНІ ВИСНОВКИ

У даній кваліфікаційній роботі було вирішено актуальну науково-практичну задачу підвищення ефективності взаємодії користувача з контентом на веб-сайті для ведення блогу шляхом розробки та реалізації системи, ключовим елементом якої є інтелектуальний модуль на основі машинного навчання для персоналізації контенту.

Проведено глибокий аналіз предметної області, в ході якого було досліджено широкий спектр сучасних контент-платформ. Це дозволило виявити ключові тенденції галузі, зокрема критичну важливість систем персоналізації для боротьби з інформаційним перевантаженням. Було проведено системний аналіз основних алгоритмів рекомендаційних систем, на основі якого для практичної реалізації було обґрунтовано вибір методу фільтрації на основі вмісту (Content-Based Filtering) як найбільш оптимального для вирішення проблеми "холодного старту".

Розроблено комплексну проектну документацію для веб-сайту «Re:Action». Спроєктовано загальну архітектуру системи, засновану на клієнт-серверній моделі з використанням патерну "фабрика додатку". Створено детальну математичну модель та покроковий алгоритм роботи модуля персоналізації, що базується на векторній моделі TF-IDF та метриці косинусної подібності. Також було спроєктовано нормалізовану схему бази даних та обґрунтовано вибір технологічного стеку, що поєднує Python/Flask для серверної частини[10][22] та Scikit-learn для реалізації ML-модуля[11].

Здійснено повний цикл програмної реалізації спроєктованої системи. Було створено функціональний веб-сайт, що включає надійний модуль автентифікації, повний CRUD-функціонал для управління статтями, систему тегів, коментарів та лайків. Ключовим результатом є успішна розробка та інтеграція модуля машинного навчання, який аналізує історію взаємодій користувача (перегляди та лайки) для формування релевантної стрічки контенту. Реалізовано повністю адаптивний користувацький інтерфейс з функцією перемикання між світлою та темною темами.

Проведено функціональне тестування розробленого програмного продукту, яке підтвердило його працездатність та відповідність поставленим вимогам. Система коректно виконує всі заявлені функції, а модуль рекомендацій адекватно реагує на дії користувача.

Таким чином, мета кваліфікаційної роботи була повністю досягнута. Створений програмний продукт «Re:Action» є практичним підтвердженням ефективності поєднання сучасних підходів до адаптивного дизайну та методів машинного навчання для створення інтелектуальних контент-орієнтованих веб-систем. Результати роботи мають практичну цінність і можуть бути використані як основа для розробки подібних проєктів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Medium [Електронний ресурс]. – Режим доступу: https://medium.com/.
2. Blogger [Електронний ресурс] : Google. – Режим доступу: https://www.blogger.com/.
3. Tumblr [Електронний ресурс] : Automattic. – Режим доступу: https://www.tumblr.com/.
4. YouTube [Електронний ресурс] : Google. – Режим доступу: https://www.youtube.com/.
5. Reddit [Електронний ресурс] : Reddit Inc. – Режим доступу: https://www.reddit.com/.
6. TikTok [Електронний ресурс] : ByteDance. – Режим доступу: https://www.tiktok.com/.
7. Pinterest [Електронний ресурс] : Pinterest Inc. – Режим доступу: https://www.pinterest.com/.
8. Spotify [Електронний ресурс] : Spotify AB. – Режим доступу: https://www.spotify.com/.
9. AliExpress [Електронний ресурс] : Alibaba Group. – Режим доступу: https://www.aliexpress.com/.
10. Flask Documentation [Електронний ресурс] : Pallets Projects. – Режим доступу: https://flask.palletsprojects.com/.
11. Scikit-learn: Machine Learning in Python [Електронний ресурс] : Scikit-learn developers. – Режим доступу: https://scikit-learn.org/stable/documentation.html.
12. SQLAlchemy Documentation [Електронний ресурс] : SQLAlchemy authors. – Режим доступу: https://www.sqlalchemy.org/.
13. Bootstrap Documentation [Електронний ресурс] : The Bootstrap Authors. – Режим доступу: https://getbootstrap.com/.
14. SimpleMDE Markdown Editor [Електронний ресурс] : Next Step Webs Inc. – Режим доступу: https://simplemde.com/.
15. Tagify - Tags input component [Електронний ресурс] : Yair Even Or. – Режим доступу: https://yaireo.github.io/tagify/.

16. Python 3.11 Documentation [Електронний ресурс] : Python Software Foundation. – Режим доступу: https://docs.python.org/3/.
17. Flask-Login Documentation [Електронний ресурс] : Max Halford. – Режим доступу: https://flask-login.readthedocs.io/.
18. Gruber J. Markdown: Syntax [Електронний ресурс] / J. Gruber. – Режим доступу: https://daringfireball.net/projects/markdown/syntax.
19. HTML: HyperText Markup Language [Електронний ресурс] : MDN Web Docs. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTML.
20. CSS: Cascading Style Sheets [Електронний ресурс] : MDN Web Docs. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS.
21. JavaScript [Електронний ресурс] : MDN Web Docs. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
22. Grinberg M. Flask Web Development / M. Grinberg. – 2nd Edition. – O'Reilly Media, 2018.
23. Aggarwal C. C. Recommender Systems: The Textbook / C. C. Aggarwal. – Springer, 2016.
24. Manning C. D. Introduction to Information Retrieval / C. D. Manning, P. Raghavan, H. Schütze. – Cambridge University Press, 2008.
25. Deep Neural Networks for YouTube Recommendations / P. Covington, J. Adams, E. Sargin // Proceedings of the 10th ACM Conference on Recommender Systems. – 2016.
26. Schwartz B. The Paradox of Choice: Why More Is Less / B. Schwartz. – Ecco, 2004.
27. Anderson C. The Long Tail: Why the Future of Business Is Selling Less of More / C. Anderson. – Hyperion, 2006.
28. System recommendations 101 [Електронний ресурс] : Medium Engineering Blog. – Режим доступу: https://medium.com/engineering/system-recommendations-101-9214340026e2.

29. TF-IDF for Document Ranking [Електронний ресурс] : Towards Data Science. – Режим доступу: https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089.

ДОДАТКИ

ДОДАТОК А

Файл requirements.txt

```
bcrypt==5.0.0
blinker==1.9.0
click==8.3.0
colorama==0.4.6
Flask==3.1.2
Flask-Bcrypt==1.0.1
Flask-Login==0.6.3
Flask-SQLAlchemy==3.1.1
greenlet==3.2.4
itsdangerous==2.2.0
Jinja2==3.1.6
joblib==1.5.2
Markdown==3.9
MarkupSafe==3.0.3
numpy==2.3.4
pandas==2.3.3
python-dateutil==2.9.0.post0
pytz==2025.2
scikit-learn==1.7.2
scipy==1.16.2
six==1.17.0
SQLAlchemy==2.0.44
threadpoolctl==3.6.0
typing_extensions==4.15.0
tzdata==2025.2
Werkzeug==3.1.3
```

Файл app.py

```
from blog import create_app, db

app = create_app()

if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

Файл recommender.py

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

def get_recommendations(current_post, all_posts, num_recommendations=3):
```

```
    post_texts = []
    index_to_post_id = {}
    current_post_index = -1

    for i, post in enumerate(all_posts):
        post_texts.append(post.title + " " + post.content)
        index_to_post_id[i] = post.id
        if post.id == current_post.id:
            current_post_index = i

    if current_post_index == -1:
        return []

    tfidf_vectorizer = TfidfVectorizer()

    tfidf_matrix = tfidf_vectorizer.fit_transform(post_texts)

    cosine_sim_matrix = cosine_similarity(tfidf_matrix, tfidf_matrix)

    similarity_scores = list(enumerate(cosine_sim_matrix[current_post_index]))

    sorted_scores = sorted(similarity_scores, key=lambda item: item[1], reverse=True)

    recommended_post_indices = []
    for index, score in sorted_scores:
        if index == current_post_index:
            continue
        recommended_post_indices.append(index)
        if len(recommended_post_indices) == num_recommendations:
            break

    recommended_posts = []
    for index in recommended_post_indices:
        post_id = index_to_post_id[index]
        rec_post = next((p for p in all_posts if p.id == post_id), None)
        if rec_post:
            recommended_posts.append(rec_post)

    return recommended_posts

def get_user_recommendations(user_history_posts, all_posts, num_recommendations=10):
    if not user_history_posts:
        return []

    post_texts = [p.title + " " + p.content for p in all_posts]
    tfidf_vectorizer = TfidfVectorizer()
```

```
        tfidf_matrix = tfidf_vectorizer.fit_transform(post_texts)

        post_id_to_index = {post.id: i for i, post in enumerate(all_posts)}

            history_indices = [post_id_to_index[p.id] for p in user_history_posts if p.id in
post_id_to_index]
        if not history_indices:
            return []

        history_vectors = tfidf_matrix[history_indices]

        user_profile_vector = np.asarray(history_vectors.mean(axis=0))

        similarity_scores = cosine_similarity(user_profile_vector, tfidf_matrix)

        scores = list(enumerate(similarity_scores[0]))

        sorted_scores = sorted(scores, key=lambda item: item[1], reverse=True)

        recommended_posts = []
        history_post_ids = {p.id for p in user_history_posts}

        for index, score in sorted_scores:
            post_id = all_posts[index].id
            if post_id not in history_post_ids:
                recommended_posts.append(all_posts[index])

            if len(recommended_posts) >= num_recommendations:
                break

        return recommended_posts
```

Файл users.py

```
from flask import Blueprint, render_template
from .models import User

users = Blueprint('users', __name__)

@users.route('/user/<string:username>')
def profile(username):
    user = User.query.filter_by(username=username).first_or_404()
    return render_template('profile.html', user=user)
```

Файл posts.py

```
from flask import Blueprint, render_template, request, flash, redirect, url_for, abort
from flask_login import login_required, current_user
from .models import db, Post, Tag, Comment, UserPostInteraction, Like
```

```
from recommender import get_recommendations
import json

posts = Blueprint('posts', __name__)

@posts.route('/post/new', methods=['GET', 'POST'])
@login_required
def create_post():
    if request.method == 'POST':
        title = request.form.get('title')
        content = request.form.get('content')

        if not title or not content:
            flash("Заголовок та зміст статті не можуть бути порожніми.", "danger")
            all_tags = Tag.query.all()
            existing_tags = [tag.name for tag in all_tags]
                return  render_template('create_post.html',  existing_tags=existing_tags,  title=title,
content=content)

        new_post = Post(title=title, content=content, author=current_user)

        tags_json = request.form.get('tags', '[]')
        tag_names = []
        try:
            tags_list = json.loads(tags_json)
            tag_names = [tag['value'].strip().lower() for tag in tags_list if tag.get('value')]
        except json.JSONDecodeError:
            pass

        if tag_names:
            for name in tag_names:
                if not name: continue
                tag = Tag.query.filter_by(name=name).first()
                if not tag:
                    tag = Tag(name=name)
                    db.session.add(tag)
                new_post.tags.append(tag)

        db.session.add(new_post)
        db.session.commit()
        flash("Вашу статтю успішно опубліковано!", "success")
        return redirect(url_for('main.home'))

    all_tags = Tag.query.all()
    existing_tags = [tag.name for tag in all_tags]
    return render_template('create_post.html', existing_tags=existing_tags)
```

```
@posts.route('/post/<int:post_id>')
def post(post_id):
    post_to_show = Post.query.get_or_404(post_id)

    if current_user.is_authenticated:
        interaction_exists = UserPostInteraction.query.filter_by(
            user_id=current_user.id,
            post_id=post_to_show.id,
            interaction_type='view'
        ).first()

        if not interaction_exists:
            new_view = UserPostInteraction(user_id=current_user.id, post_id=post_to_show.id,
interaction_type='view')
            db.session.add(new_view)
            db.session.commit()

    from recommender import get_recommendations
    all_posts = Post.query.all()
    recommendations = get_recommendations(post_to_show, all_posts)

                    return render_template('post.html', post=post_to_show,
recommendations=recommendations)

@posts.route('/post/<int:post_id>/edit', methods=['GET', 'POST'])
@login_required
def edit_post(post_id):
    post_to_edit = Post.query.get_or_404(post_id)
    if post_to_edit.author != current_user:
        abort(403)

    if request.method == 'POST':
        title = request.form.get('title')
        content = request.form.get('content')

        if not title or not content:
            flash("Заголовок та зміст статті не можуть бути порожніми.", "danger")
            all_tags = Tag.query.all()
            existing_tags = [tag.name for tag in all_tags]
                        return render_template('edit_post.html', post=post_to_edit,
existing_tags=existing_tags)

        post_to_edit.title = title
        post_to_edit.content = content
```

```
            post_to_edit.tags.clear()
            tags_json = request.form.get('tags', '[]')
            tag_names = []
            try:
                tags_list = json.loads(tags_json)
                tag_names = [tag['value'].strip().lower() for tag in tags_list if tag.get('value')]
            except json.JSONDecodeError:
                pass

            if tag_names:
                for name in tag_names:
                    if not name: continue
                    tag = Tag.query.filter_by(name=name).first()
                    if not tag:
                        tag = Tag(name=name)
                        db.session.add(tag)
                    post_to_edit.tags.append(tag)

            db.session.commit()
            flash("Вашу статтю успішно оновлено!", "success")
            return redirect(url_for('posts.post', post_id=post_to_edit.id))

        all_tags = Tag.query.all()
        existing_tags = [tag.name for tag in all_tags]
        return render_template('edit_post.html', post=post_to_edit, existing_tags=existing_tags)

    @posts.route('/post/<int:post_id>/delete', methods=['POST'])
    @login_required
    def delete_post(post_id):
        post_to_delete = Post.query.get_or_404(post_id)
        if post_to_delete.author != current_user:
            abort(403)
        db.session.delete(post_to_delete)
        db.session.commit()
        flash("Вашу статтю було видалено.", "success")
        return redirect(url_for('main.home'))

    @posts.route('/tag/<string:tag_name>')
    def posts_by_tag(tag_name):
        tag = Tag.query.filter_by(name=tag_name).first_or_404()
                                                                posts_list                              =
Post.query.filter(Post.tags.any(name=tag.name)).order_by(Post.published_at.desc()).all()
        return render_template('tag_posts.html', posts=posts_list, tag_name=tag.name)

    @posts.route('/post/<int:post_id>/comment', methods=['POST'])
    @login_required
```

```
def add_comment(post_id):
    post = Post.query.get_or_404(post_id)
    comment_body = request.form.get('comment_body')
    if comment_body:
        new_comment = Comment(body=comment_body, author=current_user, post=post)
        db.session.add(new_comment)
        db.session.commit()
        flash('Ваш коментар додано.', 'success')
    return redirect(url_for('posts.post', post_id=post_id))

@posts.route('/comment/<int:comment_id>/edit', methods=['POST'])
@login_required
def edit_comment(comment_id):
    comment = Comment.query.get_or_404(comment_id)
    if comment.author != current_user:
        abort(403)

    new_body = request.form.get('comment_body')
    if new_body:
        comment.body = new_body
        db.session.commit()
        flash('Ваш коментар оновлено.', 'success')

    return redirect(url_for('posts.post', post_id=comment.post_id))

@posts.route('/comment/<int:comment_id>/delete', methods=['POST'])
@login_required
def delete_comment(comment_id):
    comment = Comment.query.get_or_404(comment_id)
    if comment.author != current_user:
        abort(403)

    post_id = comment.post_id
    db.session.delete(comment)
    db.session.commit()
    flash('Ваш коментар видалено.', 'success')

    return redirect(url_for('posts.post', post_id=post_id))

@posts.route('/like/<int:post_id>', methods=['POST'])
@login_required
def like_action(post_id):
    post = Post.query.get_or_404(post_id)
    like = Like.query.filter_by(user_id=current_user.id, post_id=post.id).first()

    if like:
```

```
            db.session.delete(like)
            db.session.commit()
        else:
            new_like = Like(user_id=current_user.id, post_id=post.id)
            db.session.add(new_like)
            db.session.commit()

            interaction_exists = UserPostInteraction.query.filter_by(
                user_id=current_user.id,
                post_id=post.id,
                interaction_type='like'
            ).first()
            if not interaction_exists:
                    new_interaction = UserPostInteraction(user_id=current_user.id, post_id=post.id,
interaction_type='like')
                db.session.add(new_interaction)
                db.session.commit()

        return redirect(request.referrer or url_for('main.home'))
```

Файл models.py

```
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin, current_user
import datetime
from hashlib import md5

db = SQLAlchemy()

post_tags = db.Table('post_tags',
    db.Column('post_id', db.Integer, db.ForeignKey('post.id'), primary_key=True),
    db.Column('tag_id', db.Integer, db.ForeignKey('tag.id'), primary_key=True)
)

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    email = db.Column(db.String(255), unique=True, nullable=False)
    password_hash = db.Column(db.String(255), nullable=False)
    posts = db.relationship('Post', backref='author', lazy=True)
    def avatar(self, size):
        digest = md5(self.email.lower().encode('utf-8')).hexdigest()
        return f'https://www.gravatar.com/avatar/{digest}?d=identicon&s={size}'
    def __repr__(self):
        return f'<User {self.username}>'

class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
```

```
        title = db.Column(db.String(255), nullable=False)
        content = db.Column(db.Text, nullable=False)
        excerpt = db.Column(db.String(500), nullable=True)
                published_at = db.Column(db.DateTime, nullable=False, default=datetime.datetime.utcnow)
        author_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
        tags = db.relationship('Tag', secondary=post_tags, backref='posts', lazy=True)
         comments = db.relationship('Comment', backref='post', lazy=True, cascade="all, delete-orphan")
            interactions = db.relationship('UserPostInteraction', backref='post', lazy=True, cascade="all, delete-orphan")
        likes = db.relationship('Like', backref='post', lazy='dynamic', cascade="all, delete-orphan")
        def is_liked_by(self, user):
            if not user.is_authenticated:
                return False
            return self.likes.filter_by(user_id=user.id).count() > 0
        def __repr__(self):
            return f'<Post "{self.title}">'

    class Tag(db.Model):
        id = db.Column(db.Integer, primary_key=True)
        name = db.Column(db.String(50), unique=True, nullable=False)
        def __repr__(self):
            return f'<Tag {self.name}>'

    class Comment(db.Model):
        id = db.Column(db.Integer, primary_key=True)
        body = db.Column(db.Text, nullable=False)
        created_at = db.Column(db.DateTime, nullable=False, default=datetime.datetime.utcnow)
        author_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
        post_id = db.Column(db.Integer, db.ForeignKey('post.id'), nullable=False)

        author = db.relationship('User')

        def __repr__(self):
            return f'<Comment {self.id}>'

    class UserPostInteraction(db.Model):
        id = db.Column(db.Integer, primary_key=True)
        user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
        post_id = db.Column(db.Integer, db.ForeignKey('post.id'), nullable=False)
        interaction_type = db.Column(db.String(20), nullable=False)
        timestamp = db.Column(db.DateTime, nullable=False, default=datetime.datetime.utcnow)

        user = db.relationship('User')
```

```
    def __repr__(self):
        return f'<Interaction user:{self.user_id} post:{self.post_id}>'

class Like(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    post_id = db.Column(db.Integer, db.ForeignKey('post.id'), nullable=False)
```

Файл main.py

```
from flask import Blueprint, flash, redirect, render_template, request, url_for
from flask_login import current_user
from .models import Post
from recommender import get_user_recommendations
from .models import Post, User, Tag, UserPostInteraction
from sqlalchemy import or_ # Імпортуй or_ для складних запитів

main = Blueprint('main', __name__)

@main.route('/')
def home():
    page = request.args.get('page', 1, type=int)
    feed_type = request.args.get('feed', 'latest')

    pagination = None

    if feed_type == 'recommended' and current_user.is_authenticated:
        interactions = UserPostInteraction.query.filter_by(user_id=current_user.id).all()

        if not interactions:
            posts = []
              flash("Ваша історія порожня. Почитайте або вподобайте кілька статей, щоб ми могли запропонувати вам рекомендації!", "info")
        else:
            weighted_history_ids = []
            for interaction in interactions:
                weighted_history_ids.append(interaction.post_id)
                if interaction.interaction_type == 'like':
                    weighted_history_ids.extend([interaction.post_id] * 2)

            user_history = Post.query.filter(Post.id.in_(weighted_history_ids)).all()
            all_posts = Post.query.all()

            posts = get_user_recommendations(user_history, all_posts)

    else:
        feed_type = 'latest'
```

```
        pagination = Post.query.order_by(Post.published_at.desc()).paginate(page=page, per_page=5, error_out=False)
        posts = pagination.items

        return render_template('home.html', posts=posts, feed_type=feed_type, pagination=pagination)

@main.route('/search')
def search():
    query = request.args.get('q', '')

    if not query:
        return redirect(url_for('main.home'))

    results = Post.query.join(User).join(Post.tags).filter(
        or_(
            Post.title.contains(query),
            Post.content.contains(query),
            User.username.contains(query),
            Tag.name.contains(query)
        )
    ).distinct().all()

    return render_template('search_results.html', posts=results, query=query)
```

Файл auth.py

```
from flask import Blueprint, render_template, request, flash, redirect, url_for
from flask_login import login_user, logout_user, current_user
from .models import db, User
from . import bcrypt

auth = Blueprint('auth', __name__)

@auth.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('main.home'))
    if request.method == 'POST':
        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')
        hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')
        new_user = User(username=username, email=email, password_hash=hashed_password)
        db.session.add(new_user)
        db.session.commit()
        login_user(new_user)
        flash(f"Акаунт для {username} успішно створено!", "success")
```

```
            return redirect(url_for('main.home'))
        return render_template('register.html')

    @auth.route('/login', methods=['GET', 'POST'])
    def login():
        if current_user.is_authenticated:
            return redirect(url_for('main.home'))
        if request.method == 'POST':
            username = request.form.get('username')
            password = request.form.get('password')
            user = User.query.filter_by(username=username).first()
            if user and bcrypt.check_password_hash(user.password_hash, password):
                login_user(user)
                flash("Ви успішно увійшли на сайт.", "success")
                return redirect(url_for('main.home'))
            else:
                flash("Неправильне ім'я користувача або пароль.", "danger")
        return render_template('login.html')

    @auth.route('/logout')
    def logout():
        logout_user()
        flash("Ви вийшли зі свого акаунту.", "info")
        return redirect(url_for('main.home'))
```

Файл __init__.py

```
    from flask import Flask
    import markdown
    from flask_sqlalchemy import SQLAlchemy
    from flask_bcrypt import Bcrypt
    from flask_login import LoginManager
    from .models import db, User

    bcrypt = Bcrypt()
    login_manager = LoginManager()
    login_manager.login_view = 'auth.login'
    login_manager.login_message = "Будь ласка, увійдіть, щоб отримати доступ до цієї сторінки."
    login_manager.login_message_category = "warning"

    @login_manager.user_loader
    def load_user(user_id):
        return User.query.get(int(user_id))

    def create_app():
        app = Flask(__name__, instance_relative_config=True)
```

```
    @app.template_filter('markdown')
    def markdown_filter(s):
        return markdown.markdown(s)

    app.config['SECRET_KEY'] = 'your_very_secret_key_123'
    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///blog.db'

    db.init_app(app)
    bcrypt.init_app(app)
    login_manager.init_app(app)

    from .auth import auth as auth_blueprint
    app.register_blueprint(auth_blueprint)

    from .main import main as main_blueprint
    app.register_blueprint(main_blueprint)

    from .posts import posts as posts_blueprint
    app.register_blueprint(posts_blueprint)

    from .users import users as users_blueprint
    app.register_blueprint(users_blueprint)

    return app
```

Файл tag_posts.html

```
{% extends "base.html" %}

{% block content %}
<h1 class="mb-4">Статті за тегом: <span class="badge bg-info">{{ tag_name }}</span></h1>

{% if posts %}
{% for post in posts %}
<div class="card mb-3">
    <div class="card-body">
        <h5 class="card-title">{{ post.title }}</h5>
        <p class="card-text">{{ post.content[:200] }}...</p>
        {% if post.tags %}
        <p>
            Теги:
            {% for tag in post.tags %}
            <a href="{{ url_for('posts.posts_by_tag', tag_name=tag.name) }}"
                class="badge bg-secondary text-decoration-none">{{ tag.name }}</a>
            {% endfor %}
        </p>
        {% endif %}
```

```
            <p class="card-text"><small class="text-muted">Автор: {{ post.author.username }} | Опубліковано: {{
                post.published_at.strftime('%Y-%m-%d') }}</small></p>
            <a href="{{ url_for('posts.post', post_id=post.id) }}" class="btn btn-sm btn-outline-secondary">Читати далі</a>
        </div>
    </div>
    {% endfor %}
    {% else %}
    <p>Не знайдено жодної статті з цим тегом.</p>
    {% endif %}
    {% endblock %}
```

Файл search_results.html

```
{% extends "base.html" %}

{% block content %}
<h1 class="mb-4">Результати пошуку за запитом: "{{ query }}"</h1>

{% if posts %}
{% for post in posts %}
<div class="card mb-3">
    <div class="card-body">
        <h5 class="card-title">{{ post.title }}</h5>
        <p class="card-text">{{ post.content[:200] }}...</p>
                                <p class="card-text"><small class="text-muted">Автор: {{ post.author.username }}</small></p>
            <a href="{{ url_for('posts.post', post_id=post.id) }}" class="btn btn-sm btn-outline-secondary">Читати далі</a>
        </div>
    </div>
    {% endfor %}
    {% else %}
    <p>За вашим запитом нічого не знайдено.</p>
    {% endif %}
    {% endblock %}
```

Файл register.html

```
{% extends "base.html" %}

{% block content %}
<h2>Реєстрація нового користувача</h2>
<hr>
<form method="POST" action="{{ url_for('auth.register') }}">
    <div class="mb-3">
        <label for="username" class="form-label">Ім'я користувача</label>
        <input type="text" class="form-control" id="username" name="username" required>
```

```
        </div>
        <div class="mb-3">
          <label for="email" class="form-label">Email адреса</label>
          <input type="email" class="form-control" id="email" name="email" required>
        </div>
        <div class="mb-3">
          <label for="password" class="form-label">Пароль</label>
              <input type="password" class="form-control" id="password" name="password"
required>
        </div>
        <button type="submit" class="btn btn-primary">Зареєструватися</button>
      </form>
      {% endblock %}
```

Файл profile.html

```
      {% extends "base.html" %}

      {% block content %}
      <div class="row">
        <div class="col-md-4">
          <img src="{{ user.avatar(200) }}" class="img-fluid rounded-circle mb-3" alt="Avatar">
          <h3>{{ user.username }}</h3>
          <p class="text-muted">Дата реєстрації: {{ user.id }}</p> <!-- Поки що просто ID для
прикладу -->
        </div>
        <div class="col-md-8">
          <h4>Статті автора:</h4>
          <hr>
          {% for post in user.posts | sort(attribute='published_at', reverse=True) %}
          <div class="card mb-3">
            <div class="card-body">
              <h5 class="card-title">{{ post.title }}</h5>
               <a href="{{ url_for('posts.post', post_id=post.id) }}" class="btn btn-sm btn-outline-
secondary">Читати
                  далі</a>
            </div>
          </div>
          {% else %}
          <p>Цей автор ще не опублікував жодної статті.</p>
          {% endfor %}
        </div>
      </div>
      {% endblock %}
```

Файл post.html

```
      {% extends "base.html" %}
```

```
{% block content %}
<article>
  <h1 class="mb-3">{{ post.title }}</h1>
  <div class="d-flex align-items-center mb-3">
    <img src="{{ post.author.avatar(50) }}" class="rounded-circle"
      style="width: 50px; height: 50px; margin-right: 15px;">
    <div>
      <h5 class="mb-0">
            <a href="{{ url_for('users.profile', username=post.author.username) }}"
class="text-decoration-none">{{
          post.author.username }}</a>
      </h5>
      <small class="text-muted">
        Опубліковано: {{ post.published_at.strftime('%d-%m-%Y') }}
      </small>
    </div>
  </div>
  {% if post.tags %}
  <div class="mb-3">
    Теги:
    {% for tag in post.tags %}
     <a href="{{ url_for('posts.posts_by_tag', tag_name=tag.name) }}" class="badge bg-info
text-decoration-none">{{
      tag.name }}</a>
    {% endfor %}
  </div>
  {% endif %}
  <hr>
  <div>
    {{ post.content | markdown | safe }}
  </div>
</article>
<hr class="my-4">
{% if current_user.is_authenticated and current_user.id == post.author.id %}
<div class="mb-3">
   <a href="{{ url_for('posts.edit_post', post_id=post.id) }}" class="btn btn-secondary btn-
sm">Редагувати</a>
    <form action="{{ url_for('posts.delete_post', post_id=post.id) }}" method="POST"
class="d-inline">
    <button type="submit" class="btn btn-danger btn-sm"
          onclick="return confirm('Ви впевнені, що хочете видалити цю
статтю?');">Видалити</button>
  </form>
</div>
<div class="mt-3">
```

```
            <form action="{{ url_for('posts.like_action', post_id=post.id) }}" method="POST" class="d-
inline">
                <button type="submit" class="btn btn-outline-danger btn-sm">
                    {% if post.is_liked_by(current_user) %}
                        Liked
                    {% else %}
                        Like
                    {% endif %}
                </button>
            </form>
            <span class="ms-2">{{ post.likes.count() }} вподобань</span>
        </div>
        {% endif %}
        <hr>

        {% if recommendations %}
        <h3>Схожі статті:</h3>
        <div class="list-group">
            {% for rec_post in recommendations %}
             <a href="{{ url_for('posts.post', post_id=rec_post.id) }}" class="list-group-item list-group-
item-action">
                {{ rec_post.title }}
            </a>
            {% endfor %}
        </div>
        {% endif %}
        <hr>
        <h3>Коментарі</h3>
        {% if current_user.is_authenticated %}
        <form method="POST" action="{{ url_for('posts.add_comment', post_id=post.id) }}">
            <div class="mb-3">
                                <textarea      name="comment_body"      class="form-control"      rows="3"
placeholder="Залиште свій коментар..."
                    required></textarea>
            </div>
            <button type="submit" class="btn btn-primary">Відправити</button>
        </form>
        {% else %}
        <p><a href="{{ url_for('auth.login') }}">Увійдіть</a>, щоб залишити коментар.</p>
        {% endif %}

        {% for comment in post.comments | sort(attribute='created_at', reverse=True) %}
        <div class="card my-3">
            <div class="card-body">
                <p>{{ comment.body }}</p>
                <small class="text-muted">
```

```
                Автор: {{ comment.author.username }} | {{ comment.created_at.strftime('%Y-%m-
%d %H:%M') }}
              {% if current_user == comment.author %}
               <button type="button" class-="btn btn-link btn-sm text-secondary p-0 ms-2" data-bs-
toggle="modal"
                 data-bs-target="#editCommentModal-{{ comment.id }}">
                 Редагувати
              </button>

                 <form action="{{ url_for('posts.delete_comment', comment_id=comment.id) }}"
method="POST" class="d-inline">
                 <button type-="submit" class="btn btn-link btn-sm text-danger p-0 ms-1"
                   onclick="return confirm('Ви впевнені?');">Видалити</button>
              </form>

              <div class="modal fade" id="editCommentModal-{{ comment.id }}" tabindex="-1">
                 <div class="modal-dialog">
                   <div class="modal-content">
                                <form method="POST" action="{{ url_for('posts.edit_comment',
comment_id=comment.id) }}">
                     <div class="modal-header">
                       <h5 class="modal-title">Редагувати коментар</h5>
                                                  <button type="button" class="btn-close"
data-bs-dismiss="modal"></button>
                     </div>
                     <div class="modal-body">
                       <textarea name="comment_body" class="form-control" rows="3"
                         required>{{ comment.body }}</textarea>
                     </div>
                     <div class="modal-footer">
                                  <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Закрити</button>
                                   <button type="submit" class="btn btn-primary">Зберегти
зміни</button>
                     </div>
                   </form>
                 </div>
               </div>
              </div>
              {% endif %}
            </small>
          </div>
        </div>
        {% endfor %}
        {% endblock %}
```

Файл login.html

```
{% extends "base.html" %}

{% block content %}
<h2>Вхід на сайт</h2>
<hr>
<form method="POST" action="{{ url_for('auth.login') }}">
  <div class="mb-3">
    <label for="username" class="form-label">Ім'я користувача</label>
    <input type="text" class="form-control" id="username" name="username" required>
  </div>
  <div class="mb-3">
    <label for="password" class="form-label">Пароль</label>
      <input type="password" class="form-control" id="password" name="password"
required>
  </div>
  <button type="submit" class="btn btn-success">Увійти</button>
</form>
<p class="mt-3">
  Ще не маєте акаунту? <a href="{{ url_for('auth.register') }}">Зареєструватися</a>
</p>
{% endblock %}
```

Файл home.html

```
{% extends "base.html" %}

{% block content %}
<h1 class="mb-4">Останні статті</h1>

<div class="nav nav-pills mb-3">
  <a class="nav-link {% if feed_type == 'latest' %}active{% endif %}"
    href="{{ url_for('main.home', feed='latest') }}">Останні</a>
  {% if current_user.is_authenticated %}
  <a class="nav-link {% if feed_type == 'recommended' %}active{% endif %}"
    href="{{ url_for('main.home', feed='recommended') }}">Рекомендовані</a>
  {% endif %}
</div>

{% if posts %}
{% for post in posts %}
<div class="card mb-3">
  <div class="card-body">
    <h5 class="card-title">{{ post.title }}</h5>
    <p class="card-text">{{ post.content[:300] | markdown | safe }}...</p>
    {% if post.tags %}
    <p>
      Теги:
      {% for tag in post.tags %}
```

```
                <a href="{{ url_for('posts.posts_by_tag', tag_name=tag.name) }}"
                   class="badge bg-secondary text-decoration-none">{{ tag.name }}</a>
                {% endfor %}
            </p>
            {% endif %}
            <div class="d-flex align-items-center">
                <img src="{{ post.author.avatar(40) }}" class="rounded-circle"
                   style="width: 40px; height: 40px; margin-right: 10px;">
                <div>
                    <small class="text-muted">
                        Автор: <a href="{{ url_for('users.profile', username=post.author.username) }}"
                           class="text-decoration-none">{{ post.author.username }}</a>
                        <br>
                        Опубліковано: {{ post.published_at.strftime('%Y-%m-%d') }}
                    </small>
                </div>
            </div>
              <a href="{{ url_for('posts.post', post_id=post.id) }}" class="btn btn-sm btn-outline-
secondary">Читати далі</a>
        </div>
    </div>
    <div class="mt-3">
        <form action="{{ url_for('posts.like_action', post_id=post.id) }}" method="POST" class="d-
inline">
            <button type="submit" class="btn btn-outline-danger btn-sm">
                {% if post.is_liked_by(current_user) %}
                    Liked
                {% else %}
                    Like
                {% endif %}
            </button>
        </form>
        <span class="ms-2">{{ post.likes.count() }} вподобань</span>
    </div>
    {% endfor %}
    {% else %}
    <p>Ще немає жодної статті. Станьте першим!</p>
    {% endif %}
    {% if pagination %}
    <nav>
        <ul class="pagination justify-content-center">
            <li class="page-item {% if not pagination.has_prev %}disabled{% endif %}">
                <a class="page-link"
                                        href="{{    url_for('main.home',    page=pagination.prev_num,
feed=feed_type) }}">Попередня</a>
            </li>
```

```
                {% for page_num in pagination.iter_pages(left_edge=1, right_edge=1, left_current=2,
right_current=2) %}
                {% if page_num %}
                <li class="page-item {% if page_num == pagination.page %}active{% endif %}">
                   <a class="page-link" href="{{ url_for('main.home', page=page_num, feed=feed_type)
}}">{{ page_num }}</a>
                </li>
                {% else %}
                <li class="page-item disabled"><span class="page-link">...</span></li>
                {% endif %}
                {% endfor %}

                <li class="page-item {% if not pagination.has_next %}disabled{% endif %}">
                   <a class="page-link"
                                          href="{{    url_for('main.home',    page=pagination.next_num,
feed=feed_type) }}">Наступна</a>
                </li>
             </ul>
          </nav>
          {% endif %}
          {% endblock %}
```

Файл edit_post.html

```
          {% extends "base.html" %}

          {% block content %}
          <h2>Редагувати статтю</h2>
          <hr>
          <form method="POST">
             <div class="mb-3">
                <label for="title" class="form-label">Заголовок</label>
                      <input    type="text"    class="form-control"    id="title"    name="title"    required
value="{{ post.title }}">
             </div>
             <div class="mb-3">
                <label for="content" class="form-label">Зміст статті</label>
                               <textarea      class="form-control"      id="content"      name="content"
rows="10">{{ post.content }}</textarea>
             </div>
             <div class="mb-3">
                <label for="tags" class="form-label">Теги (через кому)</label>
                 <input type="text" class="form-control" id="tags" name="tags" placeholder="Додайте
теги...">
             </div>
             <button type="submit" class="btn btn-primary">Зберегти зміни</button>
          </form>
```

```
<script>
    var simplemde = new SimpleMDE({ element: document.getElementById("content"), spellChecker: false });

    var input = document.querySelector('input[name=tags]');
    var whitelist = {{ existing_tags | tojson | safe }};
    var tagify = new Tagify(input, {
      whitelist: whitelist,
      dropdown: { enabled: 0, closeOnSelect: false }
    });
    {% if post and post.tags %}
    var existingPostTags = {{ post.tags | map(attribute = 'name') | list | tojson | safe }};
    if (existingPostTags && existingPostTags.length > 0) {
      tagify.addTags(existingPostTags);
    }
    {% endif %}

    function applySimpleMDEDarkTheme() {
      const isDark = document.documentElement.getAttribute('data-bs-theme') === 'dark';
      const editorToolbar = document.querySelector('.editor-toolbar');
      if (editorToolbar) {
        const editorWrapper = editorToolbar.parentNode;
        if (isDark) editorWrapper.classList.add('editor-dark');
        else editorWrapper.classList.remove('editor-dark');
      }
    }
    setTimeout(applySimpleMDEDarkTheme, 100);
    const themeSwitcher = document.getElementById('theme-switcher');
    if (themeSwitcher) {
      themeSwitcher.addEventListener('click', () => setTimeout(applySimpleMDEDarkTheme, 50));
    }
</script>
{% endblock %}
```

Файл create_post.html

```
{% extends "base.html" %}

{% block content %}
<h2>Створити нову статтю</h2>
<hr>
<form method="POST">
  <div class="mb-3">
    <label for="title" class="form-label">Заголовок</label>
    <input type="text" class="form-control" id="title" name="title" required value="{{ title or '' }}">
```

```
            </div>
            <div class="mb-3">
                <label for="content" class="form-label">Зміст статті</label>
                 <textarea class="form-control" id="content" name="content" rows="10">{{ content or '' }}</textarea>
            </div>
            <div class="mb-3">
                <label for="tags" class="form-label">Теги</label>
                 <input type="text" class="form-control" id="tags" name="tags" placeholder="Додайте теги...">
            </div>
            <button type="submit" class="btn btn-primary">Опублікувати</button>
        </form>

        <script>
            var simplemde = new SimpleMDE({ element: document.getElementById("content"), spellChecker: false });

            var input = document.querySelector('input[name=tags]');
            var whitelist = {{ existing_tags | tojson | safe }};
            var tagify = new Tagify(input, {
                whitelist: whitelist,
                dropdown: { enabled: 0, closeOnSelect: false }
            });
            {% if post and post.tags %}
            var existingPostTags = {{ post.tags | map(attribute = 'name') | list | tojson | safe }};
            if (existingPostTags && existingPostTags.length > 0) {
                tagify.addTags(existingPostTags);
            }
            {% endif %}

            function applySimpleMDEDarkTheme() {
                const isDark = document.documentElement.getAttribute('data-bs-theme') === 'dark';
                const editorToolbar = document.querySelector('.editor-toolbar');
                if (editorToolbar) {
                    const editorWrapper = editorToolbar.parentNode;
                    if (isDark) editorWrapper.classList.add('editor-dark');
                    else editorWrapper.classList.remove('editor-dark');
                }
            }
            setTimeout(applySimpleMDEDarkTheme, 100);
            const themeSwitcher = document.getElementById('theme-switcher');
            if (themeSwitcher) {
                 themeSwitcher.addEventListener('click', () => setTimeout(applySimpleMDEDarkTheme, 50));
            }
```

```
</script>
{% endblock %}
```

Файл base.html

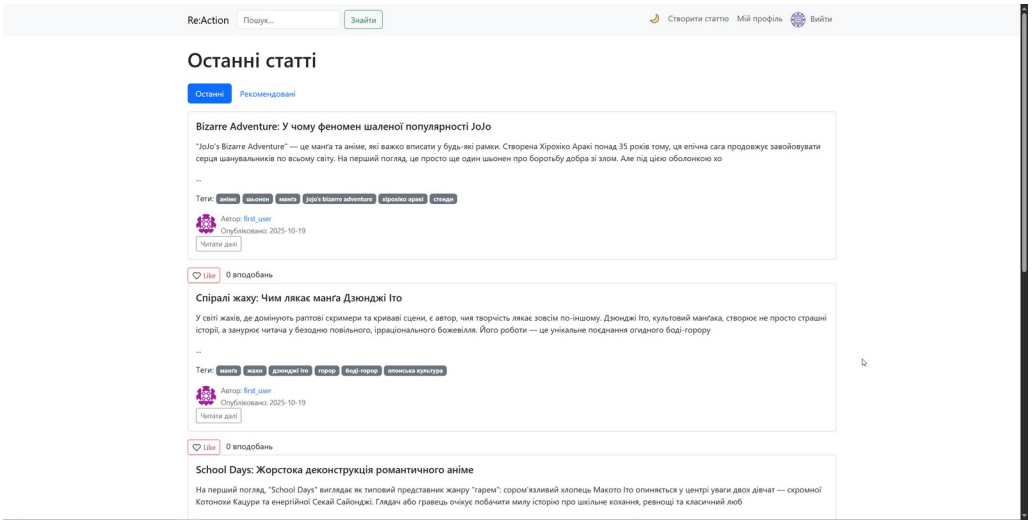
```
<!doctype html>
<html lang="uk">

<head>
   <style>
      body,
      .card,
      .navbar,
      .modal-content {
         transition: background-color 0.3s ease, color 0.3s ease;
      }

      .editor-dark .CodeMirror {
         background-color: #212529;
         color: #dee2e6;
      }

      .editor-dark .editor-toolbar>a {
         color: #dee2e6 !important;
      }

      .editor-dark .editor-toolbar>a.active,
      .editor-dark .editor-toolbar>a:hover {
         background: #495057;
         border-color: #495057;
      }

      html[data-bs-theme="dark"] .tagify {
         --tag-bg: #0d6efd;
         --tag-text-color: white;
         --input-color: #dee2e6;
         --placeholder-color: #adb5bd;
         background: #2b3035;
         border-color: #495057;
      }

      html[data-bs-theme="dark"] .tagify__dropdown {
         background: #2b3035 !important;
         border-color: #495057 !important;
      }

      html[data-bs-theme="dark"] .tagify__dropdown__item {
         color: #000000 !important;
      }

      html[data-bs-theme="dark"] .tagify__dropdown__item--active {
         background: #0d6efd !important;
         color: white !important;

      }
   </style>
   <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <title>Мій блог</title>
        <link     href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
   <link rel="stylesheet" href="https://cdn.jsdelivr.net/simplemde/latest/simplemde.min.css">
   <script src="https://cdn.jsdelivr.net/simplemde/latest/simplemde.min.js"></script>
   <script src="https://cdn.jsdelivr.net/npm/marked/marked.min.js"></script>
   <script src="https://unpkg.com/@yaireo/tagify"></script>
          <link      href="https://unpkg.com/@yaireo/tagify/dist/tagify.css"      rel="stylesheet"
type="text/css" />

</head>
```

```
        <body>
            <nav class="navbar navbar-expand-lg bg-body-tertiary">
                <div class="container">
                    <a class="navbar-brand" href="{{ url_for('main.home') }}">Re:Action</a>

                        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarNav"
                        aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
                        <span class="navbar-toggler-icon"></span>
                    </button>

                    <div class="collapse navbar-collapse" id="navbarNav">
                        <form class="d-flex me-auto my-2 my-lg-0" action="{{ url_for('main.search') }}"
method="GET">
                                        <input class="form-control me-2" type="search" name="q"
placeholder="Пошук..." aria-label="Search">
                        <button class="btn btn-outline-success" type="submit">Знайти</button>
                    </form>

                    <div class="navbar-nav ms-auto">
                        <a href="#" class="nav-link" id="theme-switcher">Тема</a>

                        {% if current_user.is_authenticated %}
                                <a class="nav-link" href="{{ url_for('posts.create_post') }}">Створити
статтю</a>
                                            <a class="nav-link" href="{{ url_for('users.profile',
username=current_user.username) }}">Мій
                            профіль</a>
                        <a class="nav-link" href="{{ url_for('auth.logout') }}">
                            <img src="{{ current_user.avatar(30) }}" class="rounded-circle"
                                style="width: 30px; height: 30px; margin-right: 5px;">
                            Вийти
                        </a>
                        {% else %}
                        <a class="nav-link" href="{{ url_for('auth.login') }}">Увійти</a>
                        <a class="nav-link" href="{{ url_for('auth.register') }}">Реєстрація</a>
                        {% endif %}
                    </div>
                </div>
            </div>
        </nav>

        <main class="container mt-4">
            {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
            {% for category, message in messages %}
            <div class="alert alert-{{ category }} alert-dismissible fade show" role="alert">
                {{ message }}
                        <button type="button" class="btn-close" data-bs-dismiss="alert" aria-
label="Close"></button>
            </div>
            {% endfor %}
            {% endif %}
            {% endwith %}

            {% block content %}{% endblock %}
        </main>

                                                                                                <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
        <script>
            (function () {
                'use strict';

                const themeSwitcher = document.getElementById('theme-switcher');
                const htmlTag = document.documentElement;

                function applyTheme(theme) {
```

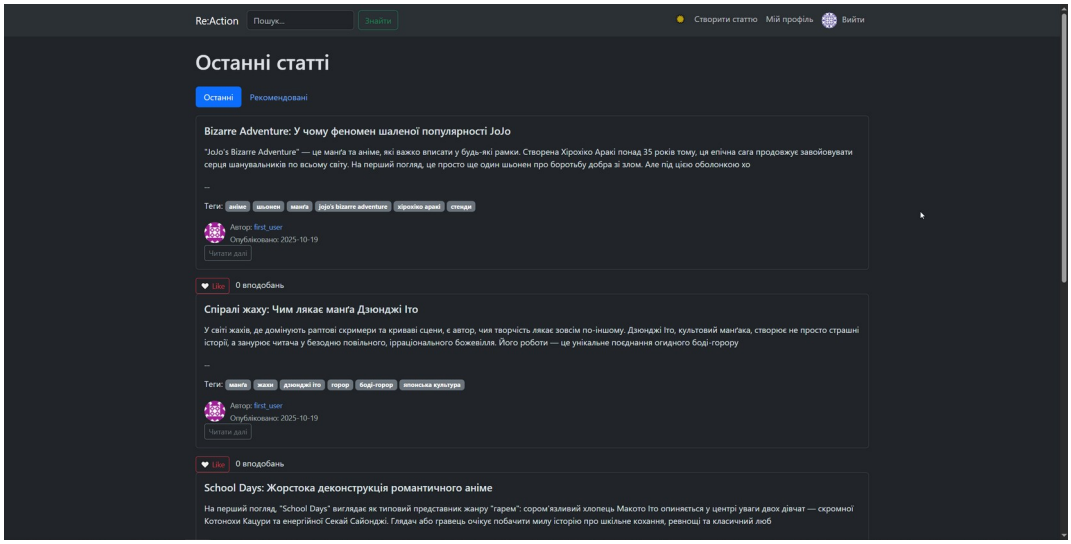
```
                htmlTag.setAttribute('data-bs-theme', theme);
                if (theme === 'dark') {
                    themeSwitcher.innerHTML = '    ';
                } else {
                    themeSwitcher.innerHTML = '    ';
                }
            }

            function toggleTheme() {
                 const currentTheme = htmlTag.getAttribute('data-bs-theme') === 'dark' ? 'dark' :
'light';
                const newTheme = currentTheme === 'dark' ? 'light' : 'dark';

                localStorage.setItem('theme', newTheme);

                applyTheme(newTheme);
            }

            themeSwitcher.addEventListener('click', function (e) {
                e.preventDefault();
                toggleTheme();
            });

            const savedTheme = localStorage.getItem('theme') || 'light';
            applyTheme(savedTheme);

        })();
    </script>
</body>

</html>
```

ДОДАТОК Б

Головна сторінка веб-сайту «Re:Action»

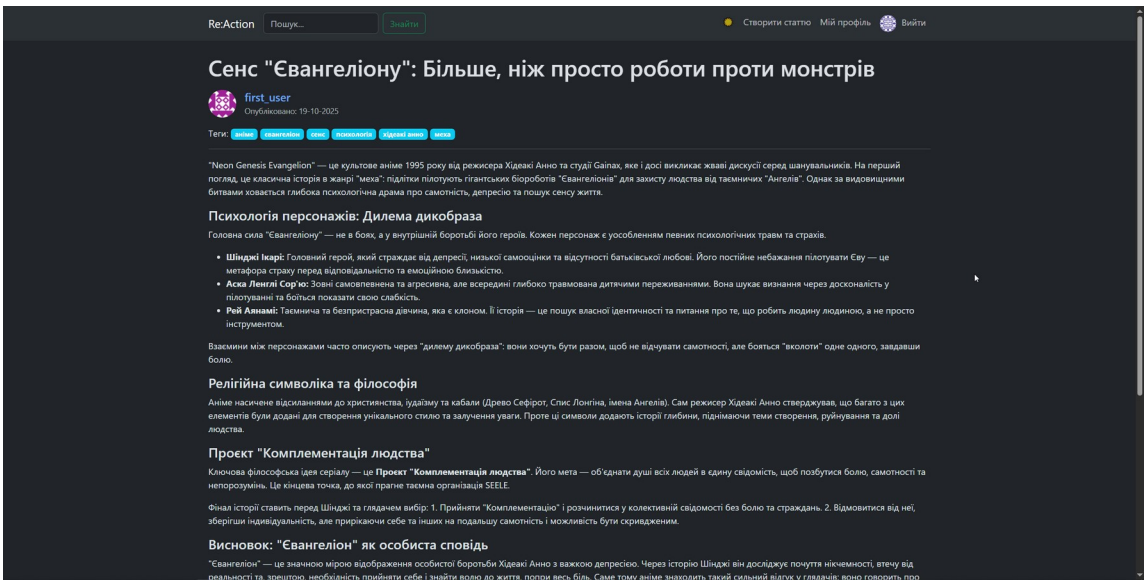


Головна сторінка веб-сайту «Re:Action» але темна тема

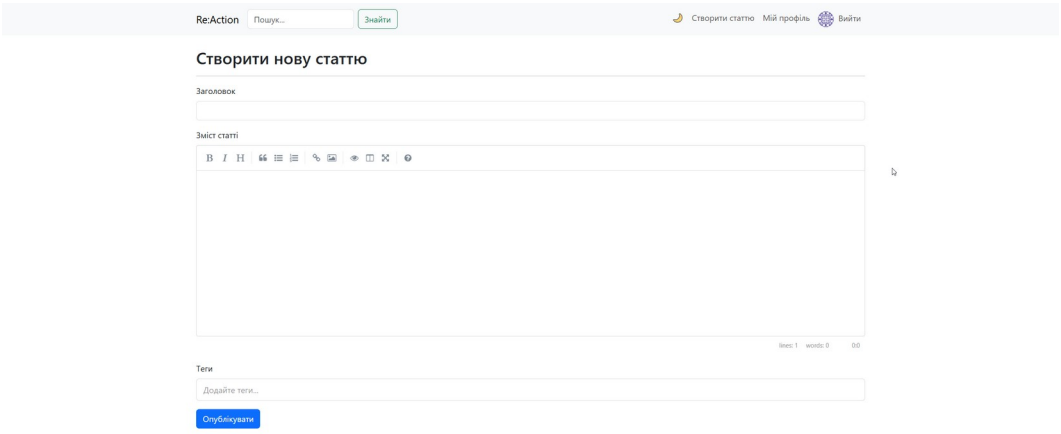


Приклад статті на веб-сайті «Re:Action»

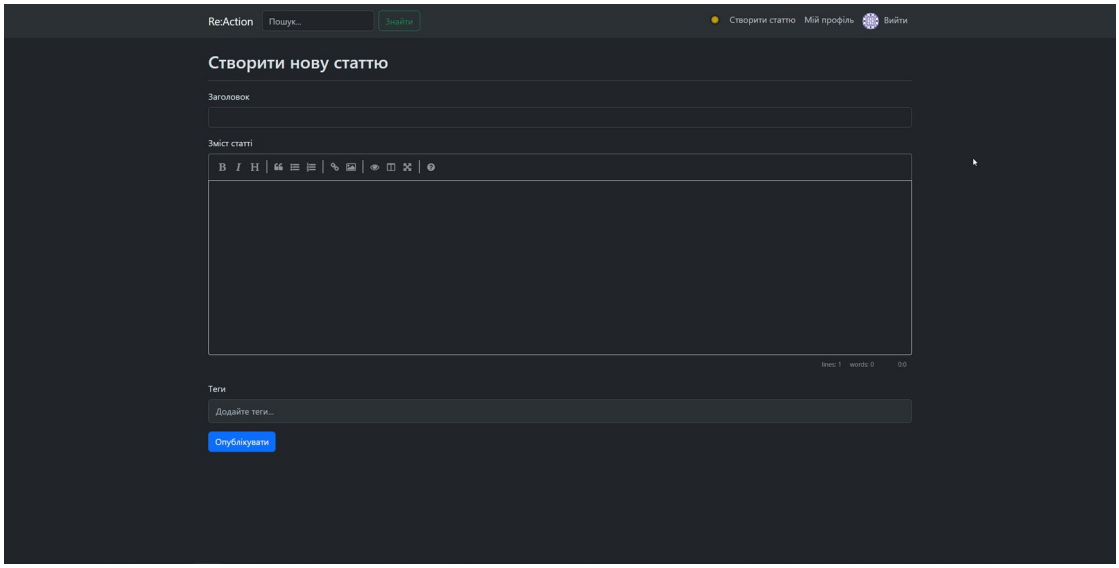
Приклад статті на веб-сайті «Re:Action» але темна тема



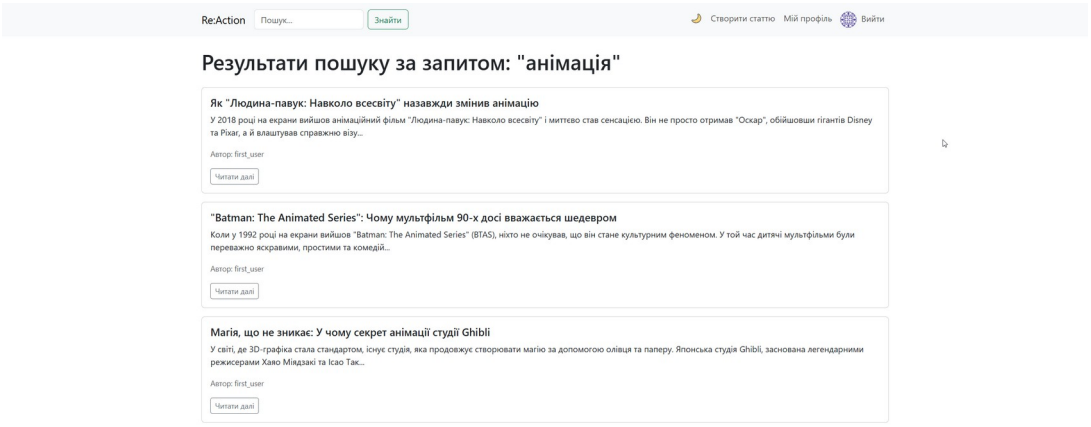
Сторінка створення статтей на веб-сайті «Re:Action»



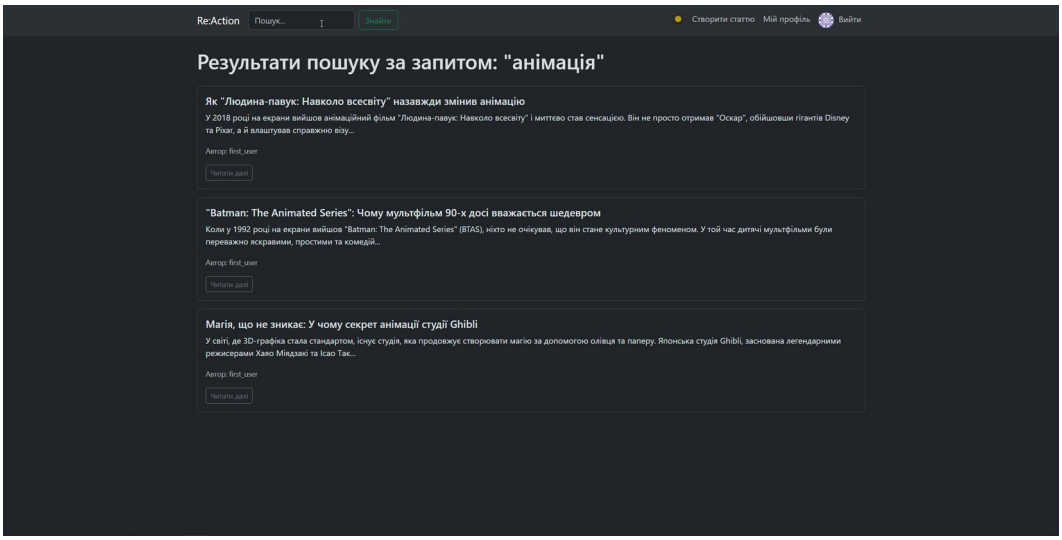
Сторінка створення статтей на веб-сайті «Re:Action» але темна тема



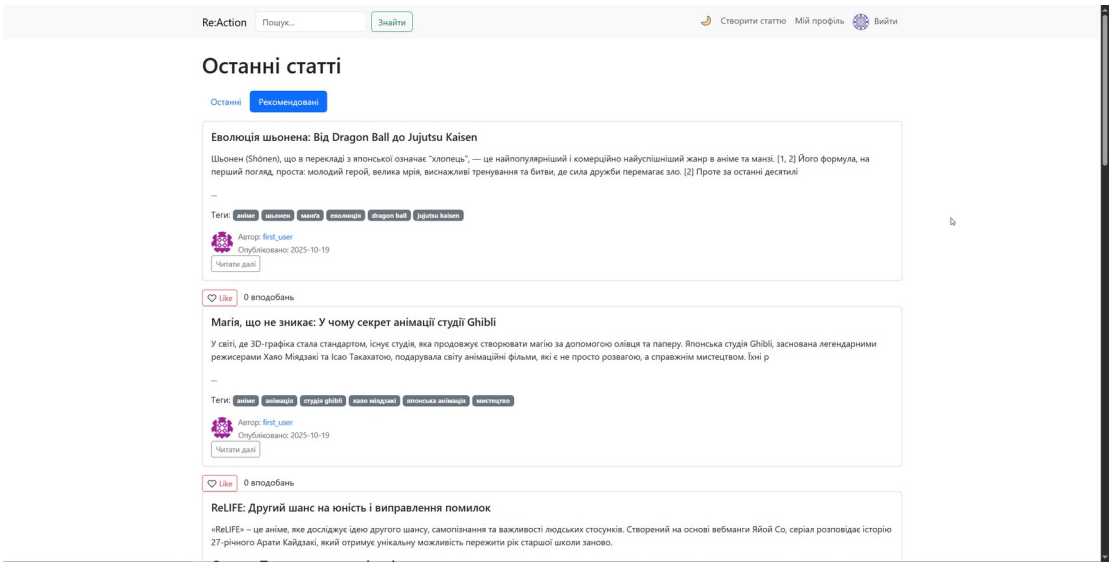
Приклад пошуку статті за запитом на веб-сайті «Re:Action»



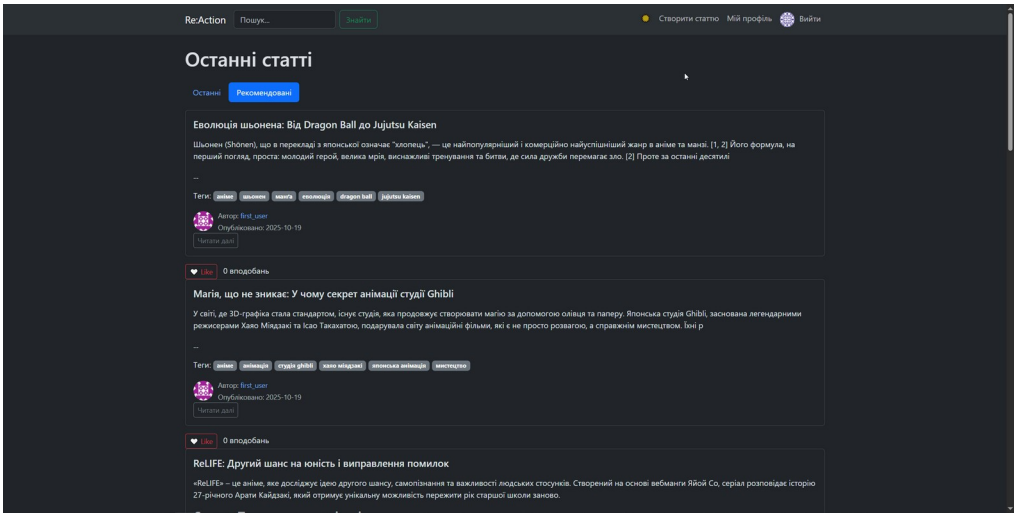
Приклад пошуку статті за запитом на веб-сайті «Re:Action» але темна тема



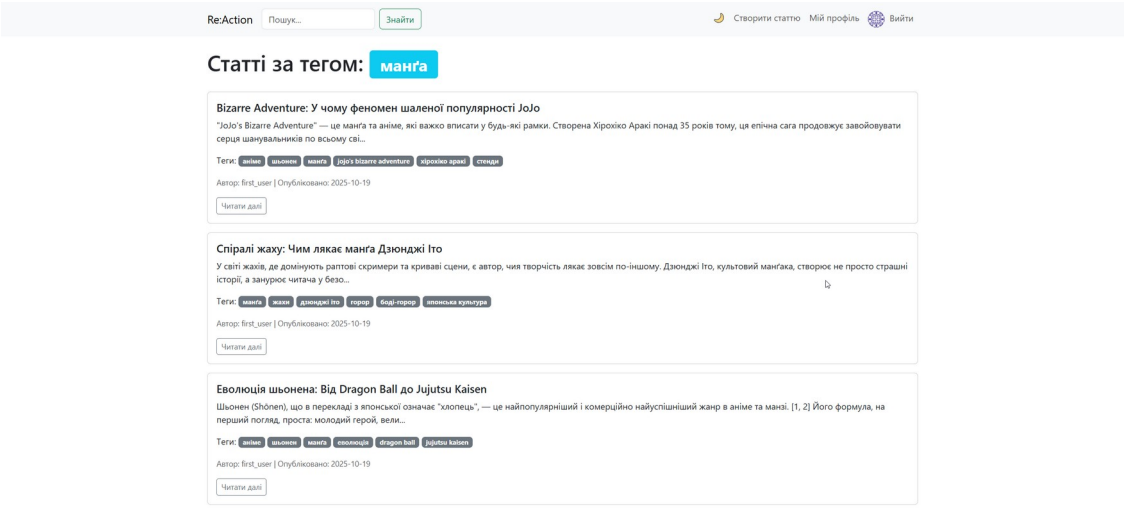
Приклад рекомендацій користувача на веб-сайті «Re:Action»



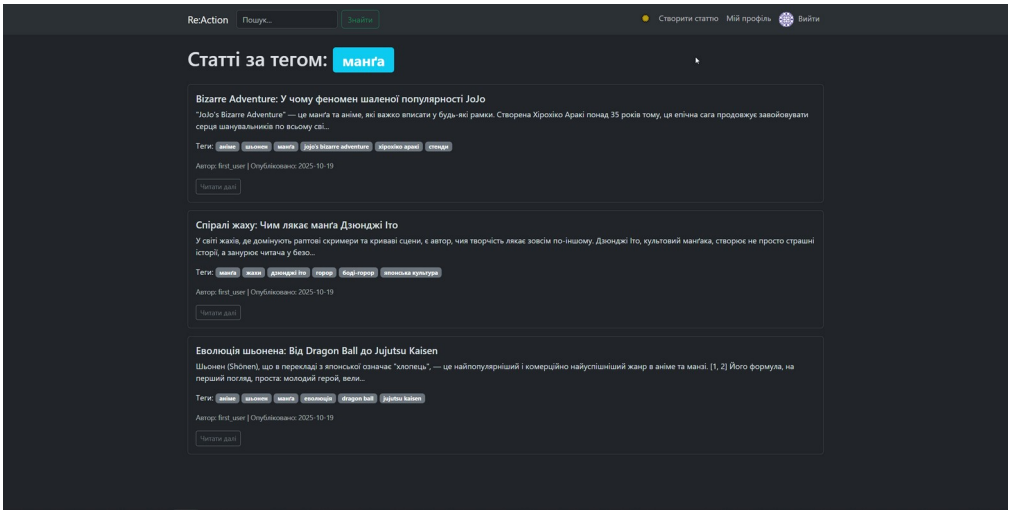
Приклад рекомендацій користувача на веб-сайті «Re:Action» але темна тема



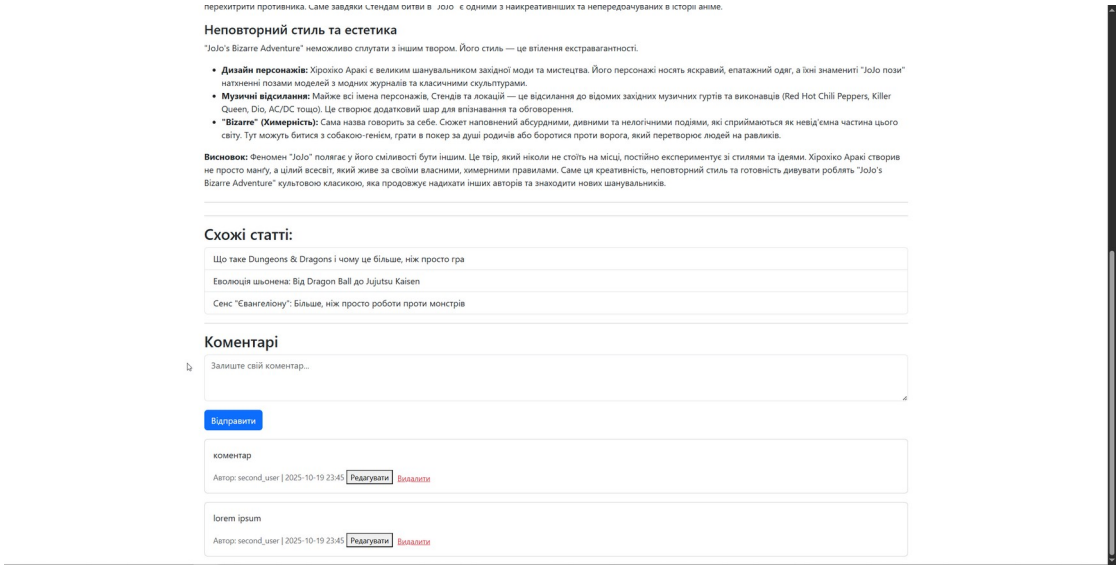
Приклад пошуку статті за тегом на прикладі веб сайті «Re:Action»



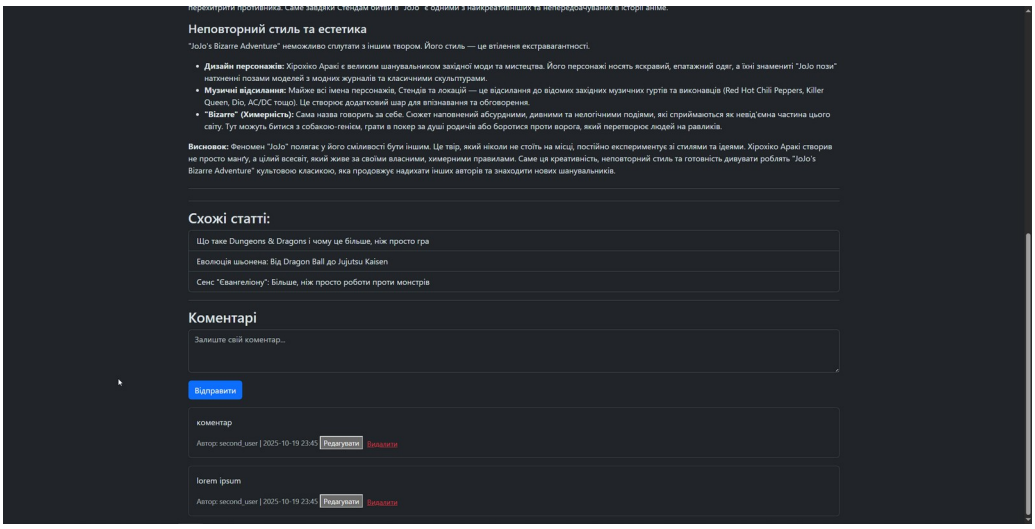
Приклад пошуку статті за тегом на прикладі веб сайті «Re:Action» але темна тема



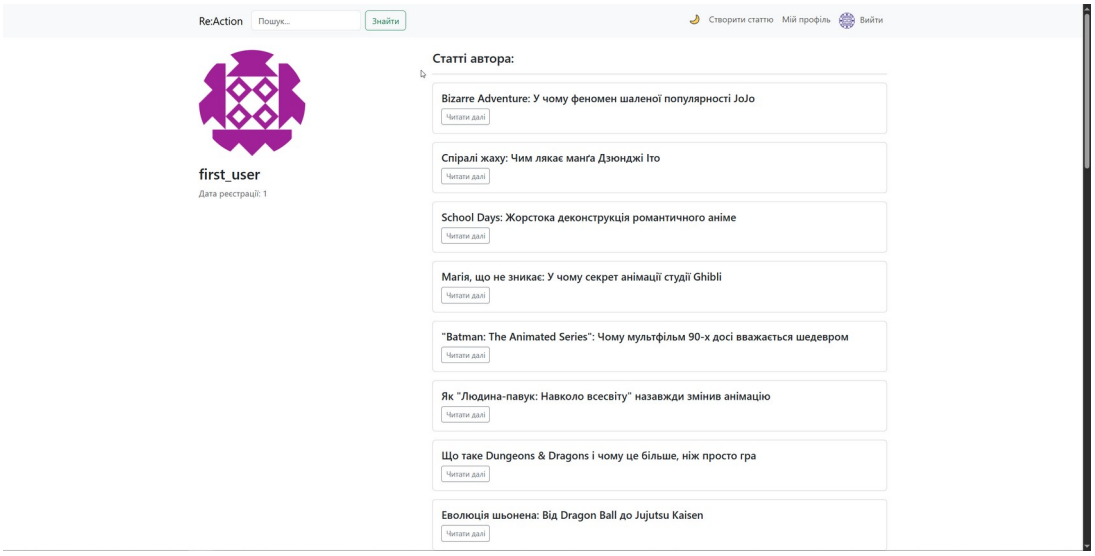
Блоки коментарів та схожих статтей на веб-сайті «Re:Action»



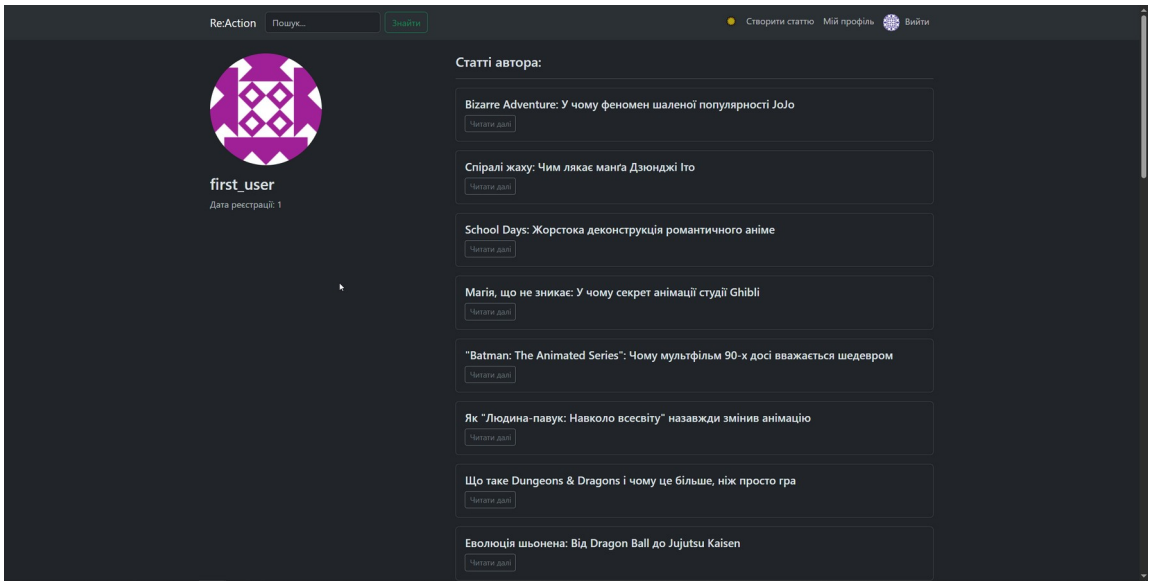
Блоки коментарів та схожих статтей на веб-сайті «Re:Action» але темна тема



Профіль користувача на веб-сайті «Re:Action»



Профіль користувача на веб-сайті «Re:Action» але темна тема



ДОДАТОК В

ТЕЗИ

Плутенко Д.В., студент кафедри програмних засобів і технологій, Херсонський національний технічний університет, м.Хмельницький, Україна,
Козуб Н.О., к.т.н., доцент кафедри програмних засобів і технологій, Херсонський національний технічний університет, м.Хмельницький, Україна.

РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ ВЕДЕННЯ БЛОГУ З АДАПТИВНИМ ДИЗАЙНОМ

Сучасний етап розвитку всесвітньої мережі характеризується переходом від моделі «пошуку інформації» до моделі її «відкриття» (discovery). В умовах експоненційного зростання обсягів цифрового контенту користувачі стикаються з проблемою інформаційного перевантаження та «парадоксом вибору». Традиційні методи організації контенту, такі як зворотна хронологічна стрічка або статичні рубрикатори, втрачають свою ефективність, оскільки не враховують індивідуальних інтересів та контексту споживання інформації. Конкурентоспроможність сучасного веб-ресурсу визначається двома факторами: адаптивністю інтерфейсу (технічна доступність на будь-якому пристрої) та адаптивністю контенту (семантична відповідність інтересам користувача). Якщо адаптивний дизайн (Responsive Web Design) вже став галузевим стандартом, то впровадження персоналізації, заснованої на методах штучного інтелекту, залишається складним завданням, особливо для нових проектів.

Для реалізації системи персоналізації було проаналізовано два основні класи алгоритмів: колаборативна фільтрація (Collaborative Filtering, CF) та фільтрація на основі вмісту (Content-Based Filtering, CBF). Колаборативна фільтрація, що базується на аналізі поведінки схожих користувачів («мудрість натовпу»), є ефективною на великих платформах

(YouTube, Spotify), проте має суттєвий недолік — проблему «холодного старту» (Cold Start Problem). Для новоствореного блогу, який ще не має накопиченої історії взаємодій великої кількості користувачів, цей метод є непридатним.

З огляду на це, у роботі обґрунтовано вибір методу фільтрації на основі вмісту (CBF). Цей підхід дозволяє будувати рекомендації, аналізуючи семантичні характеристики самих статей та зіставляючи їх з профілем інтересів конкретного користувача. CBF забезпечує роботу системи з першого дня її запуску та надає прозорі рекомендації, що базуються на фактичному змісті прочитаних матеріалів.

В рамках дослідження розроблено веб-додаток «Re:Action». Архітектура системи побудована за класичною клієнт-серверною моделлю з використанням патерну «фабрика додатку» (Application Factory), що забезпечує модульність та масштабованість. Серверна частина реалізована мовою програмування Python з використанням мікро-фреймворку Flask. Вибір Python зумовлений наявністю потужних бібліотек для наукових обчислень (NumPy, Scikit-learn), необхідних для реалізації ML-модуля. Зберігання даних організовано за допомогою реляційної СУБД (SQLite/PostgreSQL) з використанням ORM SQLAlchemy для об’єктно-орієнтованої роботи з даними.

Для реалізації системи було обрано архітектуру, яка забезпечує модульність, надійність та масштабованість. Загальна структура розробленого програмного комплексу представлена на рисунку 1.

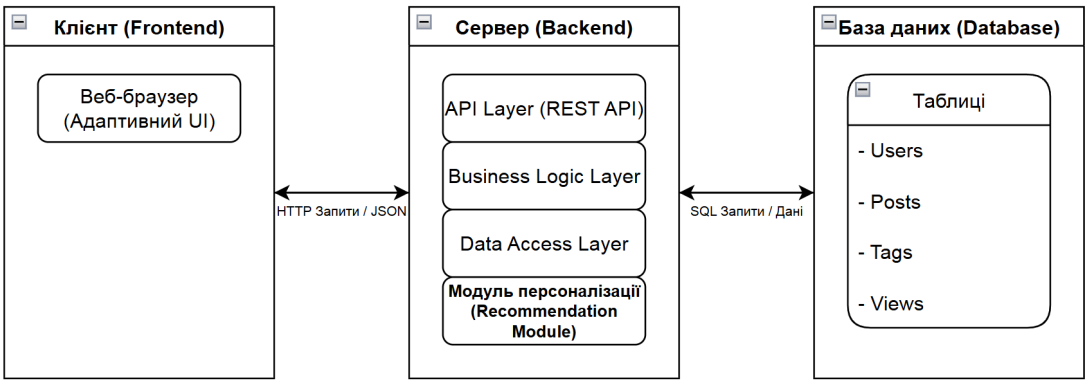


Рисунок 1 – Загальна архітектура системи

Як видно з рисунка 1, архітектура спроєктованої системи базується на клієнт-серверній моделі, що забезпечує чітке розмежування відповідальності компонентів.

Клієнтська частина (Frontend) відповідає за взаємодію з користувачем та адаптивне відображення інтерфейсу на різних пристроях. Вона формує запити до сервера та візуалізує отримані дані. Серверна частина (Backend), реалізована на базі мікро-фреймворку Flask, виступає ядром системи. Вона обробляє HTTP-запити, керує сесіями користувачів, виконує бізнес-логіку та здійснює операції з базою даних. Особливістю архітектури є виділений Модуль машинного навчання (ML Module). Він функціонує як окремий логічний компонент, що виконує ресурсоємні операції векторизації контенту та розрахунку подібності, взаємодіючи з базою даних для отримання історії переглядів, але не блокуючи основний потік обробки запитів користувача. Така структура гарантує високу продуктивність та гнучкість веб-додатку.

Ядром системи є алгоритм ранжування контенту. Процес обробки включає наступні етапи:

1. Попередня обробка тексту (NLP): Очищення контенту статей від стоп-слів та знаків пунктуації.
2. Векторизація: Використання моделі TF-IDF (Term Frequency – Inverse Document Frequency). Ця метрика дозволяє оцінити важливість слова в

контексті документа відносно всієї колекції статей. У результаті кожна стаття представляється як вектор у багатовимірному просторі.

3. Формування профілю користувача: У роботі запропоновано та реалізовано механізм зважених сигналів. Система розрізняє типи взаємодії: пасивний перегляд (view) та активне вподобання (like). При розрахунку усередненого вектора інтересів користувача, вектори статей, які отримали «лайк», враховуються з підвищеним коефіцієнтом (вагою), що дозволяє точніше визначити справжні уподобання.
4. Розрахунок подібності: Для визначення релевантності нових статей використовується косинусна подібність (Cosine Similarity) — метрика, що вимірює косинус кута між вектором профілю користувача та векторами потенційних рекомендацій.

Клієнтська частина розроблена з використанням технологій HTML5, CSS3 та JavaScript. Для забезпечення адаптивності використано фреймворк Bootstrap 5, який гарантує коректне відображення контенту на пристроях будь-якого типу (десктоп, планшет, смартфон) завдяки гнучкій системі сіток (Grid System).
Особливістю UI-реалізації є впровадження динамічної темної теми. Розроблений механізм використовує локальне сховище браузера (localStorage) для збереження вибору користувача та JavaScript-скрипти для адаптації стилів сторонніх компонентів (наприклад, WYSIWYG-редактора SimpleMDE та бібліотеки тегів Tagify), які не підтримують зміну тем «з коробки».

У результаті виконання роботи створено повнофункціональний веб-додаток «Re:Action», який демонструє ефективну синергію веб-інженерії та Data Science.
Практичне тестування підтвердило, що використання методу фільтрації на основі вмісту дозволяє вирішити проблему «холодного старту» та забезпечити користувача релевантним контентом одразу після початку

взаємодії з системою. Реалізований механізм зважування сигналів («лайк» вагоміший за «перегляд») підвищує точність рекомендацій порівняно з базовими алгоритмами. Розроблена система є масштабованою і може слугувати основою для створення спеціалізованих контент-платформ або корпоративних баз знань.

Список використаних джерел:

1. Aggarwal, C. C. (2016). Recommender Systems: The Textbook. Springer.
2. Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.
3. Scikit-learn: Machine Learning in Python https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html
4. TF-IDF for Document Ranking https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089
5. Flask Documentation https://flask.palletsprojects.com/
6. Bootstrap Documentation https://getbootstrap.com/docs/5.3/getting-started/introduction/