МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет імені Тараса Шевченка
Навчально-науковий інститут філології
Кафедра української мови та прикладної лінгвістики

# РЕКОМЕНДАЦІЙНА СИСТЕМА РЕСУРСІВ ІЗ ПРИКЛАДНОЇ ЛІНГВІСТИКИ

Кваліфікаційна робота магістра
за спеціальністю 035 «Філологія»,
спеціалізацією 035.10 «Прикладна лінгвістика»,
галузі знань 03 «гуманітарні науки»
ОП "Прикладна лінгвістика (редакторсько-перекладацька та експертна діяльність)"
**Анни ПОЛТЬЄВОЇ**

**Наукові керівники:**
доктор філософії, доц. Юлія ЦИГВІНЦЕВА,
Валентина РОБЕЙКО

**Рецензент:**
д.філол.н., проф. Євгенія КАРПІЛОВСЬКА

«Допущено до захисту»
Протокол № 10 засідання кафедри
української мови та прикладної лінгвістики
ННІФ від 04.06.2023
Завідувач кафедри_____**Сергій Різник**

Київ-2023

**ЗМІСТ**

## СПИСОК СКОРОЧЕНЬ

РС — рекомендаційна система

P2P — peer-to-peer (на рівних)

NLP — natural language processing, обробка природної мови

UMN — Університет Міннесоти (США)

MIT — Массачусетський технологічний інститут (США)

SVD — singular value decomposition, декомпозиція сингулярного значення

MF — матрична факторизація

SMC — Secure Multiparty Computation, безпечне багатостороннє обчислення

API — application programming interface, прикладний програмний інтерфейс

REST — Representational State Transfer, трансфер репрезентаційного стану

HTTP — Hyper Text Transfer Protocol, протокол трансферу гіпертексту

MVC — Model View Controller, модель, представлення, контролер

MVP — Minimum Viable Product, мінімальний життєспроможний продукт

CLI — Command-Line Interface, інтерфейс командного рядка

ЛСА — латентний семантичний аналіз

BERT — Bidirectional Encoder Representations from Transformers, двонаправлені представлення енкодера в трансформерах

# ВСТУП

Сучасна людина живе в умовах постійного інформаційного перевантаження, надлишку інформації, яку необхідно швидко шукати та якісно обробляти. Покращити взаємодію користувачів з інформацією допомагають рекомендаційні системи — актуальний напрям розвитку новітніх інформаційних технологій. Рекомендаційні системи застосовують у різних сферах інтернет-життя: електронній комерції, медіа, розвагах, соціальних мережах та ін.

Про актуальність аналізованого явища свідчить активний дослідницький діалог з цієї тематики. У сфері рекомендаційних систем існує значна кількість спеціалізованих конференцій та семінарів, які є ключовими форумами для обміну знаннями, презентації новітніх досліджень та обговорення перспективних напрямків розвитку цієї галузі. Один із найвідоміших заходів — ACM Recommender Systems (RecSys) [63] — щорічна конференція, заснована у 2007 році, яка є провідною платформою для академічних досліджень та практичних застосувань у сфері рекомендаційних систем.

Окрім спеціалізованих конференцій, рекомендаційні системи обговорюють на більш традиційних наукових заходах, які фокусують увагу на базах даних, інформаційних та адаптивних системах. У цьому контексті варто згадати Special Interest Group on Information Retrieval (SIGIR); User Modeling, Adaptation and Personalization (UMAP); а також Special Interest Group on Management Of Data (SIGMOD) [63]. Ці конференції залучають дослідників і практиків із різних галузей, що стимулює до інтеграції рекомендаційних систем у широкий контекст суміжних наукових дисциплін.

На українських теренах у цій сфері продуктивно працюють В. Щербань і Ю. Гайдейчук [10], котрі розробили систему вибору відеофільмів, що базується на рекомендаційних алгоритмах. Результати їхніх досліджень показують ефективність цієї системи в наданні користувачам персоналізованих рекомендацій щодо відеорозваг. О. Нікітін описав проєктування рекомендаційної системи для наукометричної бази даних Scopus [5], а

А. Ширалієв — рекомендаційні системи для інтернет-ресурсів, що використовують алгоритми машинного навчання [9]. В. Байдак і О. Мазурова [1] провели дослідження з розробки системи рекомендацій ігор, що базується на комбінованому підході. Вони поєднали методи колаборативної та контентної фільтрації для поліпшення точності та релевантності рекомендацій в галузі ігрового середовища. Метрики подібності та алгоритми класифікації для розробки ефективних методів колаборативної та контентної фільтрації досліджували також Є. Мелешко, С. Семенов і В. Хох [3]. Їхні розвідки висвітлюють важливі аспекти у галузі визначення схожості об'єктів та уточнення профілів користувачів для покращення точності рекомендаційних систем. С. Чалий, В. Лещинський та І. Лещинська [8] фокусувалися на вивченні темпоральних обмежень у рекомендаційних системах. Їхні дослідження стосуються аналізу впливу часових факторів на точність та актуальність рекомендаційних алгоритмів. В. Міхав, Є. Мелешко та їхні колеги [4] вдосконалили різноманітні підходи до рекомендаційних систем у peer-to-peer (P2P) середовищі. Вони вивчали алгоритми розподілу рекомендаційних завдань між вузлами, ураховуючи їхні можливості та навантаження. Окрім того, вони розвивали методи для оцінювання якості рекомендацій у P2P системах, використовуючи метрики, які враховують особливості децентралізованої структури мережі.

Завдяки рекомендаційним системам користувачам надаються персоналізовані рекомендації, що значно полегшує вибір серед великого обсягу доступної інформації. Особливо це важливо для сфер, у яких є багато інформації, але відсутні конкретні способи її пошуку та фільтрації. Однією з таких сфер є прикладна лінгвістика. Це наукова галузь, що швидко розвивається та охоплює велику кількість різноманітних ресурсів, різнорівневих і багатоспрямованих. Саме тому вибір найбільш корисних матеріалів із цього широкого спектра може бути складним. Традиційні джерела інформації, як-от рекомендації колег та посилання в книжках, мають свої недоліки, зокрема упередженість думок та застарілість матеріалу, що обмежує доступ дослідників

до актуальних джерел. Рекомендаційні системи ефективно фільтрують інформацію та пропонують результат, орієнтований на користувача. З огляду на це, **метою** магістерської роботи є розроблення ефективної рекомендаційної системи (РС), спрямованої на відбір найбільш релевантних ресурсів із прикладної лінгвістики.

**Об'єктом** дослідження є рекомендаційні системи, процеси та технології, пов'язані з розробленням РС, які базуються на використанні методів обробки природної мови. Ці процеси та технології охоплюють широкий спектр дій, включаючи збір, обробку, аналіз та інтерпретацію текстових даних з метою розуміння їх семантики та контексту.

**Предмет дослідження** — конкретний аспект створення системи рекомендацій, а саме рекомендаційної системи ресурсів із прикладної лінгвістики. Це означає, що дослідження спрямоване на розробку алгоритмів та методів, які забезпечать точні та релевантні рекомендації користувачам у сфері прикладної лінгвістики. Для досягнення мети роботи ми зосередимося на процесах та технологіях створення рекомендаційних систем, а також методах і підходах реалізації вебзастосунків для надання доступу до нашої рекомендаційної системи широкому загалу користувачів. Для цього необхідно виконати такі **завдання**:

1) схарактеризувати основні терміни та поняття, особливості досліджуваної сфери;

2) з'ясувати історію, принципи роботи, типи та проблеми рекомендаційних систем;

3) проаналізувати наявні ресурси, їх класифікацію та структурування;

4) визначити критерії та механізми, які використовують для забезпечення релевантності та якості рекомендаційної системи;

5) здійснити порівняльний аналіз рекомендаційних систем;

6) розробити адаптивний вебзастосунок, який буде пристосовуватися до потреб і вподобань кожного користувача, надавати рекомендації, ураховуючи його індивідуальні особливості та інтереси;

7) побудувати ефективну модель, яка зможе точно визначати відповідність між запитом користувача та доступними ресурсами;

8) забезпечити доступ тільки авторизованим користувачам з метою гарантування безпеки та конфіденційності.

**Теоретико-методологічна основа** роботи ґрунтується на інтеграції знань з інформаційних технологій, обробки природної мови та рекомендаційних систем. Застосування методів обробки природної мови (Natural Language Processing, NLP) дозволяє аналізувати текстові дані та розуміти зміст ресурсів із прикладної лінгвістики. Це сприяє створенню ефективної моделі, здатної точно встановлювати відповідність між запитом користувача та наявними ресурсами.

Наукові та професійні спільноти дослідників і розробників, які працюють у галузі прикладної лінгвістики, ще не мають такої рекомендаційної системи, яка б допомагала їм знаходити відповідні ресурси та матеріали для своїх досліджень і проєктів. Таким чином, **новизна** дослідження очевидна, а **практичне значення** роботи полягає в тому, що така система може полегшити процес пошуку, відбору та доступу до відповідних джерел і даних, які необхідні для проведення досліджень у галузі прикладної лінгвістики.

**Структура роботи.** Магістерська робота складається зі списку скорочень, вступу, трьох розділів, висновків до розділів, загальних висновків, списку використаних джерел із 87 позицій та додатків.

У вступі обґрунтовано актуальність теми, здійснено огляд найновішої літератури та прикладних розробок з цієї проблематики, сформульовано мету, завдання, об’єкт та предмет дослідження, окреслено теоретико-методологічну основу, новизну та практичне значення роботи, описано її структуру.

У першому розділів з’ясовано історію рекомендаційних систем, принципи їх роботи, функції та типи; розглянуто проблеми, пов'язані з рекомендаційними системами, та методи збереження конфіденційності; зроблено огляд та порівняльний аналіз схожих рекомендаційних систем.

У другому розділі детально описано побудову РС ресурсів із прикладної лінгвістики, алгоритм рекомендаційної системи, а також її клієнтську та серверну частини.

У третьому розділі проаналізовано принципи, методи та засоби побудови вебзастосунку для РС, а також представлені результати розробки вебзастосунку.

Висновки узагальнюють теоретичні положення та містять стислий підсумок результатів реалізації програмних продуктів.

Загальний обсяг роботи — 77 с., основного тексту — 63 с.

# РОЗДІЛ 1. ПОНЯТТЯ ТА СУТНІСТЬ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

У цьому розділі ми зосередимося на розгляді основних аспектів рекомендаційних систем (РС), а саме на їхній роботі, історії розвитку, функціональності, методах забезпечення конфіденційності тощо. Для цього необхідно визначитися з базовими термінами, що використовуються в контексті рекомендаційних систем. Серед них можна виділити такі поняття, як "рекомендаційна система", "елемент рекомендаційної системи", "модель" та "уподобання". Окрім того, важливим є уточнення їхніх тлумачень, що дасть можливість зрозуміти їхню сутність та роль у функціонуванні рекомендаційних систем.

Рекомендаційні системи (РС) — це "програмні інструменти та методи, що надають пропозиції щодо елементів, які можуть бути корисними для користувача. Пропозиції стосуються різних процесів прийняття рішень, наприклад, що купувати, яку музику слухати чи які онлайн-новини читати" (тут і далі переклад наш. – А. Полтьєва) [64, с. 1].

Одиниці товарів або послуг, що рекомендують РС називають просто "елемент" (або "компонент"). Говорячи більш формально, елемент — це "загальний термін, який використовується для позначення того, що система рекомендує користувачам" [64, с. 1]. РС розробляють для надання корисних і ефективних пропозицій для певного типу елемента – це можуть бути і товари, і послуги. До того ж системи можуть бути налаштовані як на конкретні типи компонентів (наприклад, Youtube/Spotify рекомендують тільки відео/аудіо відповідно), так і працювати більш узагальнено (приміром, Amazon/Rozetka рекомендують все, що в них продається: починаючи від книжок і закінчуючи кухонним кахлем). Елементи можуть бути схарактеризовані за своєю складністю, вартістю або корисністю. Оцінка елемента може бути позитивною, якщо він корисний для користувача, або негативною, якщо компонент не підходить і користувач прийняв неправильне рішення, вибираючи його [63].

Модель у цьому контексті розуміють як алгоритм машинного навчання, який вміє робити певне передбачення (у нашому випадку модель вміє передбачати вподобання користувача).

Уподобання – чітко висловлені думки щодо елементів або груп елементів [59].

### 1.1. Історія рекомендаційних систем

Рекомендаційні системи почали розвиватися після становлення інтернету та різного роду онлайн-сервісів для купівлі-продажу товарів та послуг (у тому числі сервісів, які постачають розважальний контент), зокрема таких відомих компаній, як eBay, Amazon, Spotify, Netflix, Youtube та багато інших.

Зрозуміло, що рекомендаційні системи базуються на практиці фільтрування інформації. Першою системою фільтрації інформації була Tapestry, яку розробили в 1992 році Д. Голдберг та ін. [22]. Вона ґрунтувалася на спільній фільтрації (collaborative filtering) за допомогою людської оцінки.

Згодом дослідники з Університету Міннесоти (UMN) та Массачусетського технологічного інституту (MIT) розробили службу рекомендацій новин під назвою GroupLens [55], ключовим компонентом якої була колаборативна модель фільтрації між користувачами (user-user collaborative filtering). Професор Дж. Рідл заснував дослідницьку лабораторію в UMN, також названу GroupLens, у якій започаткували дослідження систем рекомендацій. Для музики та відео подібні технології рекомендацій були застосовані системами Ringosystem [78] і Video Recommender [83] відповідно. Восени 1997 року GroupLens розробили і свій проєкт MovieLens [49] та навчила першу версію рекомендованої моделі з набором даних EveryMoviedatset. Після цього кілька наборів даних MovieLens постійно випускалися протягом 1998–2019 років і стали одними з найпопулярніших наборів даних для рекомендаційних досліджень.

Разом із появою електронної комерції галузь усвідомила економічну цінність рекомендацій. Net Perceptions, перша компанія, яка зосередилась на продажі системи маркетингових рекомендацій, була заснована в 1996 році.

Серед її клієнтів — Amazon, Best Buy, JCPenney тощо. Б. Шафер та ін. [16] пояснили, як системи рекомендацій допомагають сайтам електронної комерції збільшити продажі шляхом аналізу шести вебсайтів за трьома критеріями: інтерфейси, модель рекомендацій і дані користувачів.

З точки зору моделей рекомендацій, до 2005 року основними були різні технології колаборативної фільтрації, наприклад, колаборативна фільтрація між користувачами, спільна фільтрація за елементами (item-item collaborative filtering) і колаборативна фільтрація на основі декомпозиції сингулярного значення (SVD based collaborative filtering) [14].

Завдяки змаганням, влаштованим Нетфліксом у 2006–2009 рр., дослідники ретельно вивчали моделі матричної факторизації. Саме тоді й набула популярності ідея про те, що оцінка рекомендаційних систем повинна базуватися не на стандартних метриках, а на оцінці користувачів. Перл Пу та ін. [57] запропонували орієнтовану на користувача структуру оцінювання систем рекомендацій; Дж. Констан і Дж. Рідл [41] звернулися до еволюції досліджень системи рекомендацій від аналізу, зосередженого виключно на алгоритмах, до дослідження, зосередженого на досвіді користувача.

Перша конференція, присвячена рекомендаційним системам, — ACM Recommender Systems [40] — відбулася в Університеті Міннесоти у 2007 році. Наразі вона стала одним із найважливіших щорічних наукових заходів, який стосується вивчення рекомендаційних систем.

Учені почали використовувати й інші традиційні моделі, що зарекомендували себе як ефективні для вирішення проблем у сфері обробки природної мови. Зокрема, у 2007 році почали застосовувати логістичну регресію та її модифікації, що скоротило кількість помилок в оцінці рейтингу на 30%, а також метод факторизаційних машин (Factorization Machines) і його модифікації, наприклад, сферообізнані факторизаційні машини (Field-aware Factorization Machines), які враховують поля ознак під час моделювання ваги кожної пари ознак.

Починаючи з 2016 року, нейронні мережі посіли центральне місце в дослідженнях та розвитку систем рекомендацій. У промисловості було застосовано різні моделі для поліпшення рекомендаційного процесу. Зокрема, моделі Wide&Deep, DeepFM, YouTubeDNN і correct-sfx виявилися ефективними для покращення точності рекомендацій аудіо та відео. Для моделювання послідовної інформації та користувацьких інтересів були запропоновані моделі DIN і DIEN з використанням механізму уваги. К. Ванг та ін. [78] запропонували DCN і DCN V2 для автоматичного й ефективного вивчення прогностичної взаємодії ознак обмеженого ступеня. В академічному середовищі дослідники також запропонували важливі глибокі моделі рекомендацій, як-от FNN, PNN, NeuralCF, NFM, CVAE. Ж. Сун та ін. [87] створили контрольні показники, зокрема відтворювані та справедливі оцінні метрики для алгоритмів перших N рекомендацій на основі неявного зворотного зв’язку. Ксін Жао та ін. [79] запропонували уніфіковану структуру для розробки та відтворення рекомендаційних алгоритмів для дослідницьких цілей. Існують також деякі інші моделі рекомендацій з відкритим кодом, які значною мірою просунули прогрес у дослідженнях систем рекомендацій.

Останніми роками спостерігається зростання досліджень, спрямованих на розв’язання проблеми упередженості в системах рекомендацій шляхом аналізу причинно-наслідкових зв'язків, що були викликані рекомендаційними алгоритмами. Ці дослідження прагнуть виявити та розуміти головні причини упередженості, які можуть виникати в рекомендаційних системах. Шляхом вивчення цих причин та впливу наслідків на процес рекомендацій, дослідники сподіваються знайти шляхи усунення або зменшення упередженості, що приведе до більш релевантних і різносторонніх рекомендацій для користувачів. Це відкриває нові можливості для вдосконалення систем рекомендацій та забезпечення глибшого розуміння процесів, які лежать в основі рекомендаційних систем [19, 87]. Т. Шнабель та ін. [75] запропонували підхід до обробки упереджень відбору шляхом адаптації моделей та оцінок на основі причинного висновку. У 2018 році Х. Торстен викладав курс під назвою

"Контрфактне машинне навчання" [84]. Більшість змісту курсу базувалася на прикладах із систем пошуку інформації та систем рекомендацій.

Отже, зростання обсягу даних та доступ до різноманітної інформації сприяли розробці нових методів обробки даних, що дозволило створити більш персоналізовані і точні рекомендаційні системи. Починаючи з простих методів на основі колаборативної фільтрації, РС еволюціонували до складних алгоритмів, що використовують методи засновані на контенті, факторизаційних машинах, машинному та глибокому навчанні.

### 1.2. Принципи роботи рекомендаційних систем

У своїй найпростішій формі персоналізовані рекомендації в РС пропонуються як ранжовані списки елементів. Виконуючи це ранжування, рекомендаційні системи намагаються передбачити, які продукти чи послуги найбільше підходять користувачу, виходячи з його вподобань та обмежень. Щоб виконати таке обчислювальне завдання, РС збирають користувацькі вподобання, які або виражаються явно (експліцитний збір інформації), наприклад, як оцінки продуктів, або виводяться шляхом інтерпретації дій користувача (імпліцитний збір інформації). Наприклад, деякі системи можуть розглядати перехід на сторінку конкретного продукту як неявний знак переваги цього конкретного елемента або такого виду елементів [65].

Через свою специфіку рекомендаційні системи зберігають багато типів інформації, зокрема [61]:

1. Поведінкову інформацію — неявну інформацію, яку система рекомендацій може зібрати під час взаємодії користувача з ширшою системою. Наприклад, перегляд продукту у вебмагазині або частковий перегляд фільму на сайті відео.
2. Контекстну інформацію, тобто оточення, у якому робиться запит рекомендації. Типовими прикладами контекстної інформації є місцезнаходження, соціальна група, час, дата та мета.

3. Доменну інформацію (знання предметної області). Знання домену (сфери використання) визначають зв’язок між стереотипом користувача й елементами вмісту. Така інформація зазвичай статична, але може й змінюватися з часом.
4. Метадані елемента — описову інформацію про рекомендовані одиниці. Приклади метаданих: вид кухні для ресторанів, жанр для фільмів і виконавець для музики тощо.
5. Історію покупок або споживання — список елементів, які раніше було придбано чи спожито користувачем.
6. Відгуки про рекомендацію — інформацію про рекомендацію, надану користувачем. Зворотній зв’язок може бути виражений як позитивний, негативний або більш нюансований, аспектний (також із зазначенням причини).
7. Соціальну інформацію, що описує стосунки між різними користувачами. Багато сайтів дозволяють користувачам указувати списки друзів (або щось подібне), членство в спільноті тощо.
8. Атрибути користувача. Наприклад, демографічна інформація, дохід і сімейний стан.
9. Уподобання користувача, що виражаються скалярною мірою (оцінка елементів за шкалою від 1 до, приміром, 5 зірок), двійковим індикатором (збереження чи не збереження у списку улюблених товарів) або текстом (теги та коментарі).

Уся ця інформація, що зберігається в базах даних системи, використовується для передбачень. Для цього її переводять у матриці користувачів і матриці елементів, і за допомогою їхніх поєднань, модель знаходить залежності в уподобаннях користувачів. Існує два способи процесу передбачення за матрицею [77]:

1. Внутрішньоматричне передбачення стосується вироблення рекомендацій щодо тих елементів, які були оцінені хоча б одним користувачем системи. Тобто, у нас є елемент, уже оцінений певними користувачами, і система

робить передбачення, чи підійде цей елемент іншим користувачам. Рис. 1.1.(а) ілюструє внутрішньоматричне передбачення.

2. Позаматричне передбачення. Рисунок 1.1. (b) ілюструє позаматричне прогнозування, де елементи 4 і 5 ніколи не оцінювалися, що ще іноді називають "рекомендацією холодного старту". РС, яка не може працювати з позаматричним прогнозуванням, не може рекомендувати недавно опубліковані елементи своїм користувачам.

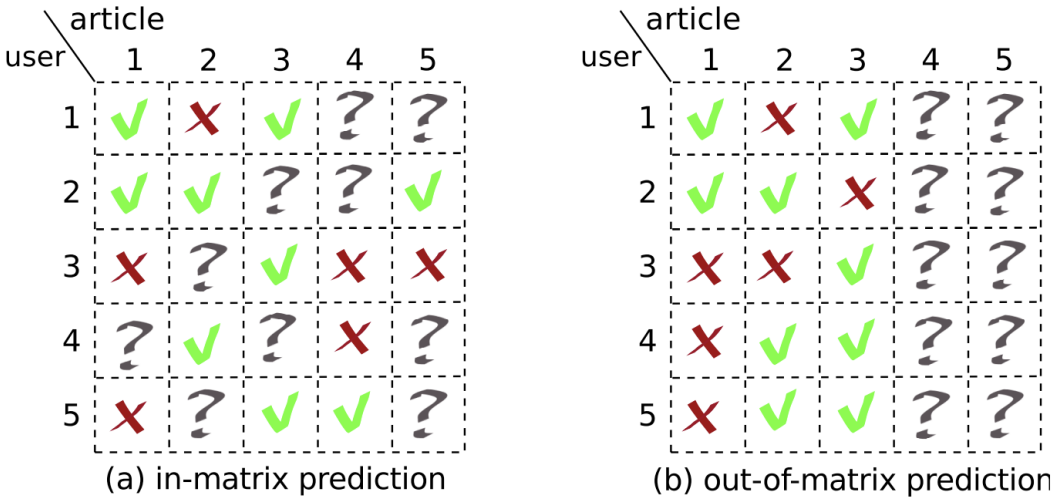


*Рис. 1.1.* ***Ілюстрація двох завдань для систем рекомендації наукових статей, де √ означає "подобається", × "не подобається", а ? "невідомий" [79].***

Формально задачу рекомендації можна представити у вигляді формули [9]:

$$D = \{(u, i, r_{ui})\}_{(u,i) \in R},$$

де $r_{ui} \in \mathbb{R}$ — кількісна величина вподобання об'єкта, $\boldsymbol{i} \in \boldsymbol{I}$ — об'єкт, обраний користувачем, $\boldsymbol{u} \in \boldsymbol{U}$, $\boldsymbol{U}$ — множина користувачів (users), $\boldsymbol{I}$ — множина об'єктів (items), $\mathbf{R} \subseteq \boldsymbol{U} \times \boldsymbol{I}$ — множина пар користувач-об'єкт (рис. 1), про які відома ступінь уподобання.

За відомою інформацією $D$ потрібно вміти будувати передбачення вподобання $\hat{r}_{ui} \approx r_{ui}$ для нових пар $\boldsymbol{u},\boldsymbol{i} \notin \mathbf{R}$.

### 1.3. Функції рекомендаційних систем

Рекомендаційні системи є важливим інструментом мережевого маркетингу, інтернет-комунікації та розваг, онлайнових засобів масової інформації та ін. РС набули такої популярності завдяки низці функцій [65], як-от:

- збільшення кількості проданих товарів. Ця концепція стосується не лише комерційних сервісів, а й некомерційних програм, які мають подібні цілі, навіть якщо користувач не здійснює жодних фінансових витрат при виборі продукту. Наприклад, контент-мережа має за мету збільшити кількість прочитаних новин на своєму вебсайті. Узагалі, з точки зору постачальника послуг, основним завданням впровадження систем рекомендацій є підвищення коефіцієнта конверсії, тобто збільшення кількості користувачів, які приймають рекомендації та споживають товари або послуги, порівняно з кількістю звичайних відвідувачів, які лише переглядають інформацію;
- урізноманітнення проданих товарів. Ще одна головна функція рекомендаційних систем — надати користувачеві можливість вибирати елементи, які може бути важко знайти без точної рекомендації. Наприклад, у таких магазинах, як Rozetka, постачальник послуг зацікавлений продати всі свої товари, а не лише найпопулярніших. Часом це важко зробити без РС, оскільки постачальник послуг не може дозволити собі ризик реклами товарів, які навряд чи задовольнять смак конкретного користувача. Таким чином, рекомендаційна система пропонує непопулярні товари окремим користувачам;
- підвищення задоволення користувачів. Інтеграція точних рекомендацій та зручного інтерфейсу є важливими факторами, що впливають на те, як користувачі сприймають системи. Таке позитивне сприйняття збільшує

активність використання системи та підвищує ймовірність успішного використання наданої рекомендації;

- підвищення лояльності користувача. Користувач повинен бути лояльним до вебсайту, який під час відвідування впізнає старого клієнта та ставиться до нього як до цінного відвідувача. Взаємодія користувача із сайтом протягом тривалого періоду сприяє вдосконаленню його матриці, що відображає систематичні уподобання користувача. Це дозволяє забезпечити більш точне налаштування рекомендаційних результатів відповідно до особистих уподобань користувача;
- покращення розуміння бажань користувача. Ще однією важливою функцією рекомендаційних систем є збір та опис уподобань користувача, які можуть бути накопичені явно або спрогнозовані самою рекомендаційно. системою. Отримані знання про користувача можуть бути використані постачальником послуг для реалізації різноманітних цілей, як-от удосконалення управління запасами та виробництвом товарів. Наприклад, у сфері туризму провайдери можуть використовувати ці дані для просування конкретного регіону серед нових сегментів споживачів.

### 1.4. Типи рекомендаційних систем

У науковій літературі рекомендаційні системи класифікують по-різному. Відмінності можна пояснити тим, що деякі дослідники виокремлюють конкретні типи систем, тоді як інші вважають їх модифікаціями загальніших типів. Зокрема, виділяють такі два основні типи: колаборативна фільтрація (collaborative filtering) та система на основі контенту (content-based), або фільтрації змісту.

Перший тип рекомендаційних систем відбирає результати на основі вподобань юзерів, які за певними параметрами схожі на даного користувача. Знайшовши невеликий набір схожих користувачів, РС будує матрицю користувачів та елемент (user-item matrix) [37].

Існує два способи використання цієї матриці в моделі: метод пам'яті (memory-based method) використовує всю матрицю для передбачення результатів, що математично можна виразити як [33]:

$$r_{us} = aggr_{u \in u'} r_{u's'},$$

де $r_{us}$ – передбачений рейтинг *s* для користувача, *u, u'* – можина користувачів, які оцінили елемент *s* та входять в одну групу з користувачем *u, aggr* – функція агрегації, у найпростішому варіанті – загальне середнє, і тоді вся формула буде виглядати таким чином [2]:

$$r_{us} = \frac{1}{N} \sum_{u \in u'} r_{u's'},$$

де *N* – потужність множини *u'*.

Цей тип є найстарішим, але досі залишається популярним.

Водночас метод моделі (model-based method) використовує цю інформацію для тренування моделі класифікатора [37].

Такі алгоритми шукають оцінку $\hat{r}$, як функцію $\hat{r}(\boldsymbol{u}, \boldsymbol{i}; \theta)$ з деякого сімейства функцій параметру $\theta \in \Theta$. Модель машинного навчання реалізується шляхом мінімізації регуляризованого емпіричного ризику [9]:

$$L(\theta) = \sum_{u,i \in R} l(\hat{r}(u, i; \theta), r_{ui}) + \lambda \Omega(\theta) \rightarrow min_{\theta \in \Theta},$$

де $l(\hat{r}, r_{ui})$ – функція втрат регресії, $\boldsymbol{\lambda}$ — сила регуляризації, $\Omega(\theta)$ — регуляризатор на множині параметрів $\boldsymbol{\theta}$.

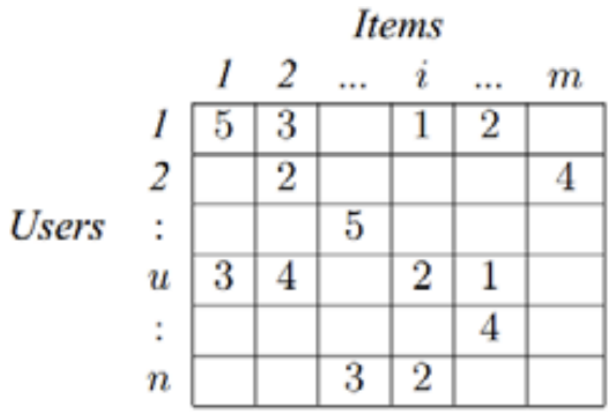


*Рис. 1.2. Приклад матриці користувачів та елементів* [37].

Багато з найбільш ефективних алгоритмів колаборативної фільтрації базуються на матричній факторизації (MF). Алгоритми MF використовуються для розкладання матриці взаємодії користувача з елементом на добуток двох прямокутних матриць нижчої розмірності [50]. Однак вони мають свої недоліки. Холодний старт є проблемою для нових користувачів, коли система не може визначити необхідні характеристики користувача. Іншою проблемою є проблема сірої вівці, яка полягає у складності надання рекомендацій для користувача, який відрізняється від усіх інших користувачів. Окрім того, такі системи вразливі до шилінгових атак, під час яких рейтинг певних елементів штучно завищують за допомогою проплачених оцінок [64]. Однією з модифікацій цього методу є нейронне колаборативне фільтрування [35].

Системи на основі контенту (content-based), або фільтрації змісту включають контент-аналізатор, який створює представлення контенту елементів, та профільний навчальний модуль, який обробляє минулі вподобання користувача. За допомогою фільтрувального компоненту на основі цих представлень і вподобань система генерує нові прогнози рекомендацій для користувача.

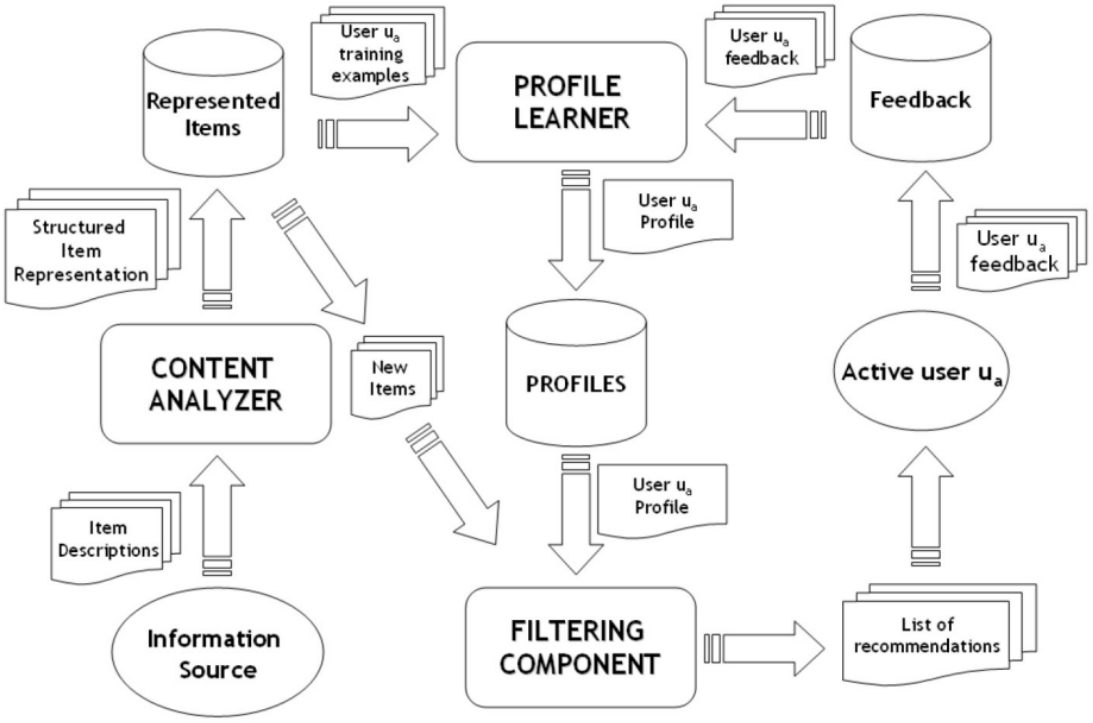


*Рис. 1.3. Алгоритм роботи рекомендаційної системи, заснованої на контенті*

Представлення елементу контенту в РС може бути реалізоване через мішок слів або векторне представлення. Системи рекомендацій, які базуються на вмісті, використовують різноманітні алгоритми машинного навчання, як-от наївний Байєс, опорні векторні машини, дерева рішень та k найближчих сусідів. Оскільки і мішок слів, і векторне представлення можуть мати велику кількість вимірів, для їх обробки часто використовують такі методи, як латентний розподіл Діріхле. До того ж обробка природної мови (NLP) може бути необхідною для використання семантичних і синтаксичних характеристик контенту [50].

У системах рекомендації аудіо- та відеоматеріалу на основі контенту прийнято використовувати аудіовізуальні властивості контенту для рекомендацій, однак для систем рекомендування книжок більшість досліджень використовують лише метадані книжки (наприклад, автора й жанр) і залишають поза увагою власне текст.

Цей метод дає змогу уникати деяких проблем колаборативної фільтрації, оскільки він не залежить від рейтингів. Однак точність таких систем страждає, якщо в елемента відсутні або неправильні метадані. Іншою проблемою є надспеціалізація, коли система видає надто вузьке коло результатів [37].

Серед інших типів вважаємо за потрібне зупинитися на таких:

Демографічні системи. Якщо детальна інформація про вподобання користувача недоступна, демографічна інформація може призвести до дещо персоналізованих рекомендацій. Grundy [26] є прикладом такої РС. Демографічна інформація може містити вік, стать, країну проживання, рівень освіти тощо. Демографічна інформація відповідає стереотипу, а елементи, пов'язані з цим стереотипом, рекомендуються користувачеві, тому персоналізація обмежена через узагальнення до стереотипу.

Системи на основі знань. Звертаючись до рекомендаційної системи, користувач вносить до неї свої інтереси. Потім вона видає певну кількість потенційних рекомендацій на основі вміщених у ній (експертних) знань. Можливо, користувач може залишити відгук і рекомендація буде уточнена. Після кількох ітерацій рекомендація адаптується до користувача. Entree [68] є прикладом такої системи, створеної, щоб допомогти відвідувачам знайти ресторан.

Контекстно-свідомі системи. Контекстна інформація, як-от місцезнаходження, динаміка груп, час, дата та мета, може бути використана для покращення рекомендацій. На відміну від уподобань користувача та доменних знань, які є відносно статичними, контекст є динамічним за своєю природою. Кожна рекомендація, навіть для того самого користувача, може мати унікальний контекст. Г. Адомавічус й А. Тужилін провели дослідження, присвячені аналізу контекстної інформації в РС. Вони виокремили три можливих способи, якими ця контекстна інформація може бути інтегрована в наявні рекомендаційні системи [30]:

(1) Використання попереднього фільтру, щоб видалити із системи елементи вмісту (і пов'язану з ними інформацію), які не відповідають контексту.

(2) Використання постфільтру, щоб видалити рекомендації, які не відповідають контексту.

(3) Додавання контексту до моделі системи рекомендацій і використання контекстної інформації під час процесу рекомендацій.

Ансамблі. Ансамблі поєднують кілька систем рекомендацій *одного типу* для підвищення ефективності. Ідея ансамблів полягає в тому, щоб дізнатися кілька думок, перш ніж прийняти рішення. А. Шклар та ін. [11] детально описав використання ансамблів для спільної фільтрації.

Гібриди. Гібридні системи рекомендацій [56], як і ансамблі, є комбінацією декількох систем рекомендацій. Однак у випадку гібридної системи використовуються різні типи рекомендаційних систем. Берк [67] подав докладний огляд різних методів гібридизації. Згідно з його висновками, при розгляді гібридизації не всі основні системи рекомендацій можна об'єднати безпосередньо.

Соціальні системи (social recommender systems / community-based recommender systems) [61]. Розвиток соціальних мереж зробив доступнішою соціальну інформацію про користувача (наприклад, списки друзів). Оскільки друзі зазвичай мають спільні інтереси, інформація, яку вони надають системі рекомендацій, найімовірніше, підійде користувачеві. До того ж докази свідчать про те, що люди схильні більше покладатися на рекомендації своїх друзів, ніж на рекомендації схожих, але анонімних людей [74]. Або соціальна інформація може бути використана для визначення спільнот схожих користувачів. До прикладу, І. Костас та ін. [40] використали соціальну інформацію в LastFM для покращення спільної фільтрації.

### 1.5. Проблеми рекомендаційних систем

Огляд проблем рекомендаційних систем є важливим етапом дослідження, оскільки РС здатні значно полегшити вибір користувачів та забезпечити зростання прибутків компаній, проте вони мають свої обмеження та недоліки, які можуть вплинути на якість рекомендацій та задоволення користувачів.

Існують різноманітні фактори, які впливають на ефективність рекомендаційних систем, зокрема [59]:

- Відсутність інформації про нових користувачів.

Оскільки більшості системам потрібна попередня інформація про користувача, рекомендація для юзера, що тільки зареєструвався буде випадковою й, імовірно, нерелевантною, що збільшує ризик втратити цього користувача. Деякі системи пропонують при реєстрації оцінити кілька рекомендацій, інші екстрактують демографічну інформацію або соціальні мережі цього користувача та його друзів/підписників [37].

- Питання збору даних.

Багато користувачів не знають про кількість і обсяг інформації, яку може зібрати постачальник послуг, і про те, що можна отримати з цієї інформації. Це може бути пов’язано з тим, що заяви про конфіденційність рідко читають, і люди звикли до користування Інтернетом. Зазвичай немає іншого способу відмовитися від такого збору даних, окрім як взагалі не використовувати систему. Оскільки практика збору не відповідає очікуванням користувачів, це занепокоєння стосується обсягу використання інформації.

- Питання збереження даних.

Інформацію в Інтернеті часто важко видалити, постачальник послуг може навіть навмисно перешкоджати видаленню даних. Це пояснюється тим, що інформація про користувачів має комерційну цінність як для конкурентної переваги завдяки аналізу, так і для продажу даних. Окрім того, інформація, яка очевидно стерта з одного місця, може все ще зберігатися в іншому місці системи, наприклад, у резервних копіях, щоб її могли знайти інші. Проблема збереження даних стосується передбачуваного терміну служби, оскільки інформація може бути доступною довше, ніж планувалося.

- Проблеми продажу даних.

Велика кількість інформації, яка зберігається в онлайн-системах, імовірно, буде мати цінність для третіх сторін і в деяких випадках може бути продана. Оцінки користувачів, уподобання та історії покупок потенційно цікаві

для маркетингових цілей. Продаж даних зазвичай суперечить очікуванням користувачів щодо конфіденційності. Незважаючи на те, що дані часто анонімізуються перед продажем для захисту конфіденційності користувачів, повторна ідентифікація є загрозою, яку часто не помічають або ігнорують. Наприклад, інформація, опублікована Netflix як частина призу їхніх рекомендованих систем, хоча й анонімна, дозволяла повторну ідентифікацію [13]. А. Нараянан і В. Шматіков пов'язали анонімні записи із загальнодоступними записами (такими як IMDb) на основі схожості рейтингів і часу оцінювання. Якщо два записи дають однакову оцінку фільму приблизно в один і той самий час, вони, імовірно, належать одній людині. Більша кількість схожих рейтингів фільмів (за рейтингом і за часом) підвищує достовірність зв'язку між записами. Це занепокоєння стосується в основному обсягу використання інформації.

- Використання інформації про користувачів в особистих цілях.

Постачальник послуг як організація має повний доступ до інформації, і його співробітники можуть цим скористатися. Це суперечить передбачуваній конфіденційності, яку постачальник послуг обіцяє своїм користувачам.

- Рекомендації, що розкривають інформацію.

Рекомендації за своєю суттю базуються на інформації, що міститься в системі рекомендацій. Наприклад, у спільній фільтрації цією інформацією є оцінки всіх користувачів, а в системах рекомендацій на основі знань — це експертні знання. Кожна рекомендація розкриває незначну частину інформації про особисті дані користувача. Незрозуміло, як велика кількість рекомендацій впливає на розкриття інформації. Це може бути використано для розкриття інформації про інших користувачів (порушення їх конфіденційності) або інформації про саму систему рекомендацій (потенційно призведе до зворотного проєктування системи). Н. Рамакрішнан та ін. [52] розглядали конфіденційність ексцентричних користувачів (користувачів із незвичними рейтингами) з точки зору графіка. Переглядаючи результати рекомендацій, ці користувачі піддаються більшому ризику, ніж середньостатистичні користувачі. Оскільки ексцентричні

користувачі не можуть сховатися в натовпі інших користувачів, коли їхні дані використовуються для надання рекомендацій, а інші дані не підходять. Рекомендації, виведені системою, ґрунтуються лише на кількох користувачах із сильною кореляцією між введенням ексцентричних користувачів і виходом рекомендацій. Це суперечить передбачуваній широті аудиторії.

- Використання спільного пристрою або акаунту.

Конфіденційність вдома може бути такою ж важливою, як і конфіденційність в інтернеті. Під час спільного використання пристрою, як-от приставки чи комп'ютера, або під час входу в онлайн-сервіс, контролювати конфіденційність щодо родини та друзів може бути важко. Наприклад, якщо дружина хоче приховати від чоловіка факт придбання йому подарунка і в неї немає приватного облікового запису, її чоловік може ненавмисно побачити покупку або отримати рекомендації на її основі. Хоча деякі служби дозволяють створювати окремі облікові записи, це не завжди можливо. Скажімо, цільова реклама працює з файлами cookie, які зберігаються у веббраузері.

- Перегляд приватної інформації сторонніми особами.

Користувачі можуть помилково вважати, що певна інформація зберігається лише для постачальника послуг або обмеженої аудиторії, хоча насправді це не так. Це може бути пов'язано з недоліками дизайну з боку постачальника послуг або з недостатнім розумінням чи увагою користувача до своєї конфіденційності. Коли третя особа переглядає таку приватну інформацію, виникає конфлікт щодо передбачуваної широти аудиторії. Розенблюм [23] показав, наприклад, що інформація в соціальних мережах є набагато доступнішою для широкої аудиторії, ніж це розуміють її власники.

- Неточність рекомендацій.

Ще одне серйозне обмеження багатьох алгоритмічних підходів до розробки рекомендаційних систем пов'язане з тим, що ці алгоритми були розроблені для збору всіх вхідних даних лише один раз. Потім вони завершують роботу, повертаючи свої рекомендації. У багатьох випадках ця модель неефективна, оскільки користувачі можуть не знати повністю про переваги,

доки вони певною мірою не взаємодіють із системою та приблизно не розуміють спектр альтернатив. Або хочуть переглянути кілька альтернативних варіантів, перш ніж переконатися, що деякі з рекомендацій можуть їм підійти. Також існує ймовірність того, що система може спочатку помилитися у своїх пропозиціях, і користувачеві необхідно буде надати додаткову інформацію, яка зможе розв'язати ці проблеми і допоможе зрештою отримати кращі рекомендації [64].

- Інтеграція довгострокових і короткострокових уподобань користувача в процесі побудови списку рекомендацій [64].

Системи рекомендацій можна розділити на два класи: ті, які створюють довгостроковий профіль, сформований шляхом агрегування всіх даних про зібрані системою транзакції користувача (наприклад, колаборaційна фільтрація), і ті, що більше зосереджені на охопленні ефемерних уподобань користувача. Очевидно, що обидва аспекти є важливими, і до уваги при розв'язанні проблеми інтеграції вподобань можна брати або точний запит користувача, або доступність елементів. Насправді розробникам потрібні нові дослідження, щоб побудувати гібридні моделі, які зможуть правильно вирішити, чи рухатися до уподобань умовного користувача, коли є достатньо доказів того, що короткострокові уподобання користувача відрізняються від довгострокових [62].

Серед лінгвістичних проблем рекомендаційних систем варто згадати роботу з багатьма мовами. З огляду на глобалізацію та багатомовний контекст сучасного світу, рекомендаційні системи повинні бути здатні працювати з різними мовними ресурсами та користувачами з усього світу. Це ставить перед розробниками виклик забезпечити адекватну обробку й аналіз текстової інформації в різних мовах, а також точність та релевантність рекомендацій незалежно від мовного контексту. Врахування особливостей кожної мови, таких як граматичні правила, синтаксичні особливості, семантичні нюанси та культурні відмінності, стає важливим аспектом розробки лінгвістично орієнтованих рекомендаційних систем.

Розв’язання вищеописаних проблем допоможе покращити якість рекомендацій та забезпечити збільшення задоволення користувачів від використання рекомендаційних систем.

### 1.6. Методи збереження конфіденційності в рекомендаційних системах

Оскільки робота РС залежить від уподобань користувачів і їхніх попередніх дій, багато користувачів можуть почуватися невпевнено через проблеми з конфіденційністю. До того ж в ансамблях і гібридних рекомендаційних системах, де застосовують дві або більше РС, може виникнути потреба використовувати їхні об’єднані дані для надання користувачам точніших рекомендацій. Цей тип обчислень, який спирається на дані більш ніж однієї сторони, називають багатостороннім обчисленням (multi-party computation) [34]. Щоб використовувати його, системи повинні мати можливість робити це безпечно, не дозволяючи іншим об’єктам читати та зберігати дані користувача. Саме тоді стають необхідними методи інтелектуального аналізу даних, що зберігають конфіденційність. Серед таких методів виокремлюють безпечне багатостороннє обчислення, рандомізацію та пертурбацію, k-Анонімність [27].

Для максимізації конфіденційності або мінімізації розкриття інформації найкращим рішенням є Secure Multiparty Computation (SMC, безпечне багатостороннє обчислення) — це метод переходу, який уперше був представлений у задачі Е. Яо про мільйонера [28]. SMC можна класифікувати як теоретико-інформаційний або обчислювальний метод [13]. В обчислювальній моделі передбачається, що супротивник обмежений поліноміальним часом. В інформаційно-теоретичній моделі супротивник вважається необмеженим. Розв’язують багато завдань для вирішення різних аспектів (наприклад, складності, змагальності, кількості корумпованих сторін) SMC. Загалом існує два типи супротивників, пов’язаних із визначеннями SMC: напівчесні та зловмисні [46]. Напівчесна змагальна модель часто призводить до більш

ефективних протоколів збереження конфіденційності, але зловмисна модель є менш обмежувальною і, отже, більш реалістичною.

Основна ідея пертурбаційного підходу полягає в тому, що дані змінюють (наприклад, додають шум, але зберігають базову статистику) перед тим, як їх оприлюднити та проаналізувати. Ключовим є те, що вихідний розподіл можна приблизно відновити зі спотворених даних.

Останнім часом сфера рандомізації зміщується в бік диференційованої конфіденційності [21], яка має на меті приховати зв'язок між інформацією про окремих користувачів у вхідних даних (інформація про користувача) і вихідних даних (рекомендація). Це досягається тим, що користувачі оприлюднених даних обчислювально не відрізняються від більшості інших користувачів у цьому наборі даних. Необхідний рівень шуму залежить від того, як часто будуть використовуватися дані, і зазвичай передбачає балансування між точністю вихідних даних і конфіденційністю вхідних даних. Така нерозрізненість також стосується колаборативних систем рекомендацій, де користувач не є в змозі ідентифікувати оцінки окремих користувачів у результатах, які він отримує. Оскільки кожна рекомендація розкриває трохи інформації про вхідні дані (навіть із шумом), то більша кількість рекомендацій передбачає більший обсяг доданого шуму, щоб забезпечити той самий рівень конфіденційності. Ф. МакШеррі й І. Миронов [29] запропонували алгоритми спільної фільтрації в системі диференціальної конфіденційності. Шум додається до коваріаційної матриці елементів (для подібності елементів). Оскільки коваріаційна матриця елемента менша за користувальницьку коваріаційну матрицю, потрібно додати менше шуму та зберегти більшу точність.

Недоліком цих методів є те, що їхню безпеку важко формально довести, як це роблять, наприклад, у класичній криптографії. Рівні шуму в методах диференціальної конфіденційності не повинні перевищувати початкові вихідні дані й таким чином повністю позбавляти користі результатів. Водночас необхідно додати достатньо шуму, щоб приховати дані користувача. У

поєднанні з кількома результатами обчислень і зовнішньою інформацією для захисту конфіденційності користувача потрібно ще більше шуму [61].

- k-Анонімність.

Цей метод був розроблений для запобігання атак із зовнішніх посилань [52]. Основна ідея полягає в тому, що набір даних є k-анонімним, якщо кожен запис з'являється принаймні k разів відповідно до попередньо визначеного набору атрибутів квазіідентифікатора. Основним підходом до досягнення k-анонімності є узагальнення та придушення інформації.

### 1.7. Порівняльний аналіз схожих рекомендаційних систем

На сьогодні розроблена велика кількість РС як загального, так і конкретного спрямування. Оскільки ми створюємо вузькоспеціалізовану систему рекомендування ресурсів із прикладної лінгвістики, то аналогами вважатимемо саме системи рекомендування книжок, адже більшість ресурсів міститимуть саме книжки та інші схожі матеріали (наприклад, інтернет-статті та посилання Github-інструменти).

Серед схожих систем нам вдалося знайти дві системи — Bookmix [18] і LiveLib [48], які мають власну спільноту й здійснюють колаборативну фільтрацію запитів, а також мають додатковий функціонал, зокрема зберігання в "улюблені", можливість запрошення друзів, створення тематичних груп та обговорень, а також публікацію рецензій. Однак, книжки там є тільки російською та англійською мовами. З-поміж англомовних систем близькою є Goodreads [36].

Х. Алхарті та Д. Інкпен [37] використали набір даних Litrec для розробки системи рекомендацій книг на основі контенту з урахуванням мовних особливостей книг. Набір даних Litrec має оцінки 1927 користувачів із 3710 літературних книг і містить повні тексти книг, розмічені за частиномовною належністю слів. Система, яку розробили Х. Алхарті та Д. Інкпен, базується на аналізі лексичних, символічних, синтаксичних і стилістичних особливостей текстів книг.

Для лінгвістичного аналізу автори використовували Linguistic Inquiry and Word Count (LIWC), який є популярним ресурсом, що фокусує увагу на граматичних, змістових і психологічних категоріях слів. Використовуючи словник LIWC 2015, дослідники обчислили 94 категорії, як-от: відсоток латинських слів, службових слів, слів афекту тощо.

Інші вимірювання тексту обчислюють за допомогою вбудованого тегера GutenTags, який використовує стилістичний лексикон для обчислення стилістичних аспектів, які зазвичай враховують під час аналізу англійської літератури. Система виділяє шість антиномій: розмовний — літературний, конкретний — абстрактний та суб'єктивний — об'єктивний. Окрім того, дослідники використовують розпізнавач імен LitNER з урахуванням вигаданих власних назв, щоб ідентифікувати кількість персонажів і кількість місць, згаданих у книзі. Нарешті, у системі представлено вимірювання читабельності тексту шляхом обчислення показника легкості читання за Флешем.

Н. Грінквіст та ін. [53] впровадили гібридну систему CBF/CF. Щоб зібрати більше інформації, вони об'єднали дані goodbooks-10k із даними оглядів Amazon. Для представлення книг дослідники використовували вектори TF-IDF описів книг, тегів і категорій. Автори зауважили, що використовували описи книг у своєму підході, заснованому на вмісті, але незрозуміло, як вони отримали описи, оскільки останніх немає в наборі даних goodbooks-10k.

У вільному доступі не вдалося знайти власне українських систем рекомендації книжок. Такі РС проаналізовано в магістерських роботах і статтях О. Когулько [2], О. Нікітіна [5], А. Ширалієва [9], В. Щербаня і Ю. Гайдейчука [10], однак самих систем у вільному доступі немає.

Ми прийняли рішення розробити рекомендаційну систему, яка ґрунтується на знаннях, з метою подолати проблему холодного старту, що є суттєвою перепоною для нових систем з обмеженим трафіком. Такий підхід також сприятиме розв'язанню проблеми конфіденційності, оскільки ми збиратимемо мінімальний обсяг інформації про користувачів. Шляхом побудови системи на основі знань, ми здатні створити ефективну модель рекомендацій,

яка не залежить від великої кількості персональних даних та може надавати релевантні рекомендації, навіть для нових користувачів із обмеженим взаємодією. Такий підхід має потенціал забезпечувати якісні рекомендації, зберігаючи приватність користувачів і зменшуючи ризики, пов'язані зі збором та обробкою особистих даних.

## Висновки до розділу 1

Рекомендаційні системи — це програми, які надають інформацію та рекомендації користувачам на основі їхніх уподобань і поведінки. Вони є важливими складниками сучасних технологій у сфері онлайн-покупок і розваг, що знаходять своє застосування в різних галузях, зокрема електронній комерції, музиці, книгах, відео, соціальних мережах тощо.

Основними підходами до збору інформації та надання рекомендацій є колаборативний підхід та визначення на основі контенту.

Математичні моделі та алгоритми відіграють важливу роль у рекомендаційних системах. Деякі з них включають логістичну регресію, факторизаційні машини, нейронні мережі тощо.

Аналіз наукової літератури з теми дозволив не тільки з'ясувати основні поняття, окреслити історію досліджуваного явища, визначити ключові функції та принципи роботи РС, а й закцентувати на деяких проблемах та обмеженнях, з якими стикаються розробники й користувачі рекомендаційних систем. Це, зокрема, проблеми з точністю, етичним збором, зберіганням і використанням даних, конфіденційністю користувачів. Збереження конфіденційності є однією з ключових проблем у рекомендаційних системах. Деякі з методів збереження конфіденційності охоплюють рандомізацію даних, безпечне багатостороннє обчислення та k-анонімність.

У процесі дослідження зроблено огляд аналогів нашої рекомендаційної системи, що уможливило глибше розуміння та чіткіше визначення способів реалізації подібних задач, а також дало змогу об'єктивно оцінити переваги нашої РС.

Було прийнято рішення створювати РС, засновану на знаннях, що допоможе уникнути проблеми холодного старту, яка є значною перешкодою для нової системи з малим трафіком. Такий підхід також дозволить уникнути проблем із конфіденційністю, оскільки про користувачів буде збиратися мінімальна кількість інформації.

# РОЗДІЛ 2. РЕАЛІЗАЦІЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

Розробка рекомендаційної системи є складним завданням, яке потребує знань із різних галузей, як-от статистика, машинне навчання та програмування. У цьому розділі будуть описані деталі реалізації системи рекомендації ресурсів із прикладної лінгвістики. Особлива увага буде приділена аналізу принципів, методів та засобів побудови системи, опису даних, які використовуються в системі, огляду алгоритмів та інструментів, які застосовуються для реалізації РС, а також опису архітектури системи та її компонентів. Усі ці деталі допоможуть зрозуміти, як рекомендаційна система працює та як її можна вдосконалити в майбутньому.

## 2.1. Алгоритм рекомендаційної системи

У процесі розробки вебзастосунку було реалізовано рекомендаційну систему, засновану на знаннях. Це означає, що наша система має збережену інформацію про певні ресурси й видає результати на запит користувача на основі цієї інформації. Детальніше про типи рекомендаційних систем див. підрозділ 1.4. У нашій системі знання зберігаються у вигляді ембедингів, які беруться з опису ресурсу. Опис ресурсу формується вручну користувачем вебзастосунку, а ембединги формуються пізніше на серверній частині, про що йшлося в попередньому підрозділі.

Ембедингами називають n-вимірні вектори слів (токенів), і вони є найкращим способом числового кодування текстових даних зі збереженням семантичної інформації [82]. Розмірність цих векторів може коливатися від кількох десятків до кількох сотень, залежно від конкретної реалізації, і це наразі вважається оптимальним для збереження семантичного змісту тексту.

Ідея ембедингів базується на теорії латентного семантичного аналізу (ЛСА) [47], яка була розроблена в 50-х роках минулого століття лінгвістами та філософами [43, 45]. ЛСА припускає, що контекст, у якому вживається слово, містить взаємозв'язки, що суттєво впливають на значення слова або набору слів. Слова з подібними контекстами (за дистрибуційною гіпотезою) вірогідно мають

подібне або ідентичне значення [45]. ЛСА вважає, що такі взаємозв'язки можуть бути представлені у вигляді матриці, яка містить відносну частоту вживання кожного слова в кожному контексті, і використовує алгоритми лінійної алгебри для виявлення схожості між словами [43].

Існує багато різних варіантів векторного представлення слів. Наприклад, один з таких варіантів полягає у створенні матриці, що складається з великої кількості текстів, де рядки представляють слова, а стовпці – документи, що містять ці слова. Цей підхід дає вектори розмірністю n, де n – кількість документів. Оскільки такі вектори є розсіяними, це може бути незручним для певних завдань та займати багато комп'ютерної пам'яті. Тому можна застосовувати математичні методи, зокрема сингулярний розклад матриці (SVD) або невід'ємну факторизацію матриць (NMF), для зменшення розмірності векторних просторів.

Є й інші способи створення векторних представлень слів, скажімо, терміно-термінова матриця. Також можна використовувати нейронні мережі, як-от word2vec, для створення щільних векторів у процесі роботи системи [6]. Вектори можна застосовувати для класифікації та кластеризації документів, пошуку інформації, знаходження синонімів та багатозначності, аналізу асоціацій слів у корпусі текстів, машинного навчання та видобування текстів. Також можна використовувати їх для пошуку найкращої подібності між групами термінів в семантичному плані, наприклад у текстах [43].

Існує різне програмне забезпечення для створення та використання ембедингів, таких як Word2vec Томаса Міколова, GloVe Стенфордського університету, GN-GloVe, AllenNLP, ELMo, BERT [66], fastText, Gensim, Indra та Deeplearning4j. Щоб зменшити розмірність векторних просторів слів та візуалізувати ембединги слів та кластерів, застосовують методи Principal Component Analysis (PCA) та T-Distributed Stochastic Neighbour Embedding (t-SNE) [84].

З поняттям ембедингу тісно пов’язане поняття тензору. Тензор — це багатовимірний масив або матриця числових значень, що використовується для

представлення даних, параметрів і результатів обчислень у різних алгоритмах машинного навчання. Ембединги часто представляють у вигляді тензорів в рамках машинного навчання за допомогою різних інструментів, таких як TensorFlow і PyTorch. Ембединги можна розглядати як особливий вид тензора, оптимізований для представлення категоріальних або дискретних даних у безперервному векторному просторі.

Під час розробки нашої рекомендаційної системи ми зіткнулися з багатьма викликами, пов'язаними з роботою з мовами. Оскільки наша система призначена для користувачів з усього світу, ми повинні бути готові обробляти описи ресурсів, які можуть бути написані будь-якою мовою. Це ставило перед нами завдання вибору ефективної мультилінгвальної моделі, яка здатна працювати з різними мовами.

У нашій роботі ми використали мультилінгвальні ембединги BERT (Bidirectional Encoder Representations from Transformers, двонапрямні представлення енкодеру в трансформерах), а конкретніше, ембединги, навчені на моделі paraphrase-multilingual-MiniLM-L12-v2 [66], що є загально доступною моделлю в Python-бібліотеці sentence_transformers від HuggingFace. Ця модель була вибрана, оскільки вона забезпечує якісну обробку текстів на понад 100 мовах світу, що робить її ідеальним кандидатом для нашої рекомендаційної системи. Застосування мультилінгвальної моделі BERT дозволяє здійснювати ефективне представлення текстових даних, ураховуючи семантичну схожість слів та контексту. Використання цієї моделі дозволило нам забезпечити високу якість рекомендацій незалежно від мови описуваного ресурсу, що є дуже важливим для задоволення потреб користувачів з різних країн та представників різноманітних культур.

Розглянемо, як реалізовано процеси роботи з ембедингами в нашому додатку.

Коли створюють новий ресурс і викликається EmbeddingsService.dump(@resource), опис ресурсу, що зберігається в атрибуті @resource.description перетворюється на масив векторів, які записують у файл

"embeddings.pkl" разом з ідентифікатором ресурсу для того, щоб спростити пошук необхідного ресурсу на етапі пошуку кандидатів на рекомендації.

```python
from sentence_transformers import SentenceTransformer
from pickle import dump as pickle_dump
from sys import argv

def main(id):
    description = open(f'./tmp/{id}.description.txt', 'r').read().strip()
    model = SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')
    description_embeddings = model.encode(description)

    with open('./db/embeddings.pkl', 'ab') as f:
        pickle_dump({id: description_embeddings}, f)

if __name__ == '__main__':
    main(argv[1])      You, last week • Add script for saving an embedding …
```

*Рис. 2.1. Код файлу embed.py, що відповідає за створення та збереження ембедингів ресурсу*

Цей файл було зашифровано в бінарному форматі з метою економії місця в системі зберігання даних. Бінарний формат є більш ефективним для зберігання даних порівнянно з текстовим форматом, оскільки кожен символ у текстовому форматі займає один байт, тоді як у бінарному форматі можуть бути використані маски та біти для зберігання більшої кількості інформації в меншому обсязі. Таким чином, застосування бінарного формату може допомогти зберегти простір на диску та зменшити час доступу до даних під час їх обробки. Проте збереження даних у файлі замість бази даних може мати свої недоліки. Зокрема, як ілюструє рис. 2.1, для внесення змін до одного з ембедингів потрібно перезаписати весь файл. Така операція має алгоритмічну складність O(n) як щодо часу, так і в пам'яті, що може створити проблеми в разі роботи з великими обсягами даних.

```python
from encode_query import load_embeddings
from sentence_transformers import SentenceTransformer
from sys import argv
import pickle

def main(model, id):
    embeddings = load_embeddings()
    for dict in embeddings:
        if id in dict.keys():
            dict[id] = model.encode(open(f'./tmp/{id}.description.txt', 'r').read())
            break
    with open('book_embeddings.pkl', 'wb') as f:
        pickle.dump(embeddings, f)


if __name__ == '__main__':
    model = SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')
    id = argv[1]
    main(model, id)      You, now • Uncommitted changes
```

*Рис. 2.2. Код файлу update.py, що відповідає за зміну та збереження ембедингів ресурсу.*

Щодо того, як обчислюються рекомендації, то код можна побачити в Додатку 3, оскільки він занадто великий, щоб подавати його в основному тексті роботи.

Наша система рекомендацій працює на основі запитів користувачів. Конкретніше це виглядає так: якщо користувач вводить запит "Я хочу почитати щось про статистичну лінгвістику", то цей запит перетворюється на вектор ембедингів, які, нарівні з усіма іншими ембедингами у файловому сховищі, нормалізуються за допомогою L2-норми [20]. Після цього косинусна подібність між вектором запиту та векторами всіх описів ресурсів обчислюється для знаходження k (у нашому випадку k=10) найбільш схожих описів, використовуючи алгоритм k найближчих сусідів [44]. Косинусна подібність між ембедингами дозволяє оцінити ступінь схожості між запитом та описами ресурсів.

Алгоритм k найближчих сусідів (k-Nearest Neighbors, або k-NN) є одним з найпростіших алгоритмів машинного навчання, що використовується для класифікації та регресії.

Основна ідея алгоритму полягає в тому, щоб із навчального набору даних знайти k точок, найближчих до нового зразка, і визначити клас чи значення цього зразка на основі більшості класів або середнього значення з найближчих точок.

Наприклад, якщо ми маємо навчальний набір даних, що містить інформацію про квіти (довжину і ширину пелюсток та чашолистків), і ми хочемо класифікувати нову квітку за її параметрами, ми можемо використати k-NN, щоб знайти k найближчих квіток до нашого нового зразка за його параметрами і визначити клас цієї квітки на основі більшості класів із найближчих точок.

Алгоритм k-NN може бути використаний як для класифікації, так і для регресії, залежно від того, який результат ми хочемо отримати. Скажімо, для класифікації можна використовувати більшість класів із k найближчих точок (що ми й робимо в рамах задачі пошуку рекомендацій), а для регресії можна використовувати середнє значення з k найближчих точок.

Однією з переваг алгоритму k-NN є його простота й незалежність від розподілу даних. Однак для ефективної роботи алгоритму потрібно мати достатньо великий навчальний набір даних, і підбирати правильне значення k для конкретної задачі.

Найважливішим елементом процесу підбору кандидатів у нашому коді є змінна top_k_indices, у яку помістили індекси 10 найбільш схожих ембедингів з бази даних (DB_embeddings). Цей список індексів отримується за допомогою функції tf.math.top_k для тензору scores, що є результатом обчислення косинусної подібності між вектором запиту та всіма векторами в базі даних.

Функція tf.math.top_k повертає значення та індекси 10 найкращих елементів вхідного тензору. У цьому випадку вона повертає кортеж, що містить два тензори: перший містить найкращі 10 оцінок, а другий — відповідні індекси цих оцінок. Змінна top_k_indices потім отримує індексовий тензор за допомогою синтаксису індексування "[1]", який потім перетворюється у масив NumPy за допомогою методу numpy().

Нарешті, top_k_indices стискається та перетворюється на список за допомогою методу tolist(). Отриманий список містить індекси 10 найбільш схожих ембедингів у базі даних до ембедингу запиту. Саме за цими індексами знаходять ідентифікатори ресурсів, найбільш схожих на запит. Можна помітити, що функція не повертає ці ідентифікатори, а виводить у консоль. Через особливість виклику Python-скрипту з Ruby-програми, Ruby отримає назад ідентифікатори, тільки якщо їх вивели в консоль. За цими ідентифікаторами програма знаходить необхідні ресурси (див. рис. 2.3 рядок 18).

### 2.2. Серверна частина рекомендаційної системи (бекенд)

З погляду серверної частини реалізація рекомендаційної системи не є важким завданням, однак потребує додавання кількох елементів системи й модифікації кількох наявних елементів.

```ruby
class EmbeddingsService
  def dump(resource)
    File.write("#{Rails.root}/tmp/#{resource.id}.description.txt", resource.description)
    `python3 ./app/embeddings/embed.py #{resource.id}`
    File.delete("#{Rails.root}/tmp/#{resource.id}.description.txt")
    puts "Dumped embeddings for resource ##{resource.id}"
  end

  def update_embedding(resource)
    File.write("#{Rails.root}/tmp/#{resource.id}.description.txt", resource.description)
    `python3 ./app/embeddings/update.py #{resource.id}`
    File.delete("#{Rails.root}/tmp/#{resource.id}.description.txt")
    puts "Updated embeddings for resource ##{resource.id}"
  end

  def fetch_candidates_for(query)
    output = `python3 ./app/embeddings/encode_query.py #{query.gsub(/[()]/, '')}`
    Resource.where(id: eval(output))
  end
end
```

*Рис. 2.3. Код класу EmbeddingsService*

Насамперед роботу рекомендаційної системи було інкапсульовано в окремий сервіс, названий EmbeddingsService, відповідно до стандартних шаблонів програмування, зокрема MVC, про який ішлося в минулому розділі. Розглянемо цей сервіс більш детально.

Як можна побачити на рис. 2.3, EmbeddingsService є доволі малим сервісом, що відповідає за три функції, пов'язані з ембедингами, про які йтиметься в наступному підрозділі. EmbeddingsService відповідає за створення і збереження ембедингів (метод dump), оновлення ембедингів (метод update_embedding), що необхідно у випадку, якщо опис ресурсу змінився, і пошук кандидатів до запиту користувача на основі ембедингів (метод fetch_candidates_for).

Під час роботи ми зіткнулися з проблемою, коли деякий код, пов'язаний з ембедингами, мав бути написаний на мові програмування Python без можливості прямого перекладу на мову програмування Ruby, оскільки скрипти використовували сторонні бібліотеки, які були реалізовані тільки на мові програмування Python, а реалізація їхньої функціональності власноруч неможлива через складність такої задачі. Існує кілька варіантів вирішення цієї проблеми, наприклад бібліотеки на Ruby, які дозволяють імпортувати сторонні бібліотеки на мові Python, однак такі рішення погано працюють із бібліотеками, які використовувалися в коді. Тому було обрано інше рішення — використано вбудовані можливості мови Ruby викликати shell-команди за допомогою зворотних лапок (``).

Команда оболонки (shell-команда) — це текстова команда, яка надходить до оболонки інтерфейсу командного рядка (CLI) для виконання певного завдання або набору завдань. В обчислювальній техніці оболонка — це програма, яка інтерпретує команди користувача, а потім виконує їх. Інтерфейси командного рядка — командний рядок Windows, термінал macOS і оболонка Unix/Linux — дозволяють користувачам взаємодіяти з оболонкою, вводячи команди та отримуючи вихідні дані. Команди оболонки можна використовувати для маніпулювання файлами, запуску програм, керування процесами,

налаштування параметрів системи та виконання багатьох інших завдань. Це дозволило нам викликати Python-скрипт прямо з нашого застосунку, що ілюструє рис. 2.3 у рядках 4, 11 і 17.

Однак передача параметрів, необхідних для роботи скрипту, також виявилася нетривіальною задачею, оскільки описи ресурсів, які потрібно було перетворити на ембелинги, могли містити символи кінця рядка (\n), що одразу викликало неправильне виконання програми. Для уникнення цієї проблеми було вирішено зберігати опис ресурсу в тимчасовому файлі, який скрипт без проблем міг знайти за допомогою ID ресурсу й прочитати, а після завершення виконання Python-програми, цей файл видалявся. Саме тому в рядках 3 і 10 можна побачити створення цих тимчасових файлів, а в рядках 5 і 12 — їхнє видалення.

Метод fetch_candidates_for також мав проблему з текстовою природою параметра query, що передається йому. Якщо цей параметр містив дужки, це викликало некоректне виконання програми, тому було вирішено позбуватися дужок, адже вони не несуть ніякої семантичної інформації, важливої для виконання поставленого перед нами завдання.

Створення EmbeddingsService потребувало модифікації коду контролера, який відповідає за обробку запитів і відправку результатів їхньої обробки (детальніше про роботу контролера див. пункт 3.1.1).

Зокрема, при створенні нового ресурсу цей сервіс повинен також створювати й зберігати його ембединги.

```
18  def create
19    @resource = Resource.create(resource_params.merge({user_id: params[:user_id]}))
20    if @resource.save
21      render json: { message: "Successfully created a resource ##{@resource.id}" },
22                   status: :created
23      embeddings_service.dump(@resource) unless @resource.description&.empty?
24    else
25      render json: { errors: @resource.errors.full_messages }, status: :unprocessable_entity
26    end
27  end
28
```

*Рис. 2.4. Код методу, що відповідає за створення нового ресурсу*

Як видно на рис. 2.4, ми створюємо ембединги для ресурсу, якщо ресурс було успішно збережено і тільки якщо ресурс має опис. Це відбувається вже після повернення користувачу відповіді на запит, оскільки створення ембедингів, як для методу контроллера, є доволі повільним процесом (до 20 секунд порівняно зі звичайною відповіддю контроллера у декілька мілісекунд), і ми не хочемо змушувати користувача стільки чекати.

Схожі зміни було додано до методу, що відповідає за зміну ресурсу, який вже був створений, що можна побачити на рис. 2.5.

```
29  def update
30    desciption_changed = @resource.description == resource_params[:description] ? false : true
31    if @resource.update(resource_params)
32      render json: { notice: 'Resource was successfully updated' }, status: :ok
33      embeddings_service.update_embedding(@resource) if desciption_changed      You, 23 hours ag
34    else
35      render json: { errors: @resource.errors.full_messages }, status: :unprocessable_entity
36    end
37  end
```

*Рис. 2.5. Код методу, що відповідає за зміну ресурсу, що вже існує*

Як бачимо, тут ми змінюємо ембединги тільки в тому випадку, якщо зміни в ресурсі було успішно збережено і тільки якщо ці зміни містили опис ресурсу, на якому базуються ембединги, адже викликати таку повільну операцію на описі, що не змінився, не має потреби, особливо враховуючи, що зміна ембедингів є ще повільнішою, ніж додавання нового ембедингу (алгоритмічна складність O(n) для зміни порівняно з O(1) для додавання), що пов’язано з особливістю додавання ембедингів, про що йтиметься в наступному підрозділі.

Наостанок розглянемо код методу, який власне повертає результати пошуку кандидатів на запит користувача (рис. 2.6).

```ruby
def recommend
  query = params[:query]
  result = embeddings_service.fetch_candidates_for(query)

  render json: {recommendations: result.joins(:user)
                                       .select('resources.*, users.username, users.email')}
end

```

*Рис. 2.6. Код методу, що відповідає за повернення результатів із пошуку кандидатів для рекомендації користувачу*

Як бачимо, код у контролері не має нічого специфічного, оскільки логіку пошуку кандидатів інкапсульовано в embeddings_service.fetch_candidates_for, що було розглянуто раніше. Однак варто зазначити, що fetch_candidates_for також є повільним методом, і в цьому випадку ми не можемо нічого вдіяти, окрім як попросити користувача почекати (див. рис. 2.6).

### 2.3. Клієнтська частина рекомендаційної системи (фронтенд)

Для реалізації клієнтської частини було створено окремий компонент, який інкапсулює реалізацію рекомендаційної системи, й окремий компонент, що інкапсулює реалізацію пошуку в базі даних ресурсів із прикладної лінгвістики, однак обидва ці компоненти подано на одній сторінці, оскільки в кінцевому результаті вони повертають набір ресурсів, якими цікавиться користувач, тож логічним видається розміщення цих компонентів на одній сторінці.

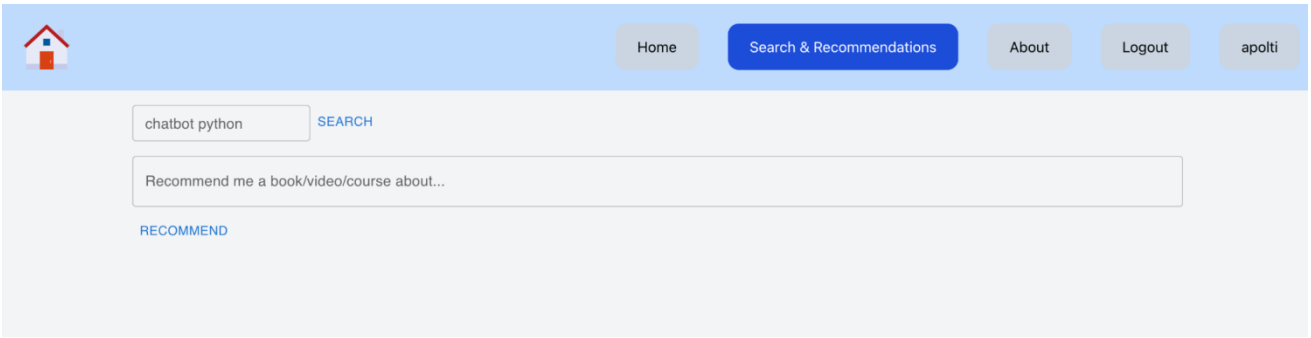


*Рис. 2.7. Загальний вигляд сторінки "Search & Recommend" (знайти і порекомендувати) як тільки користувач заходить на неї*

Однак між цими двома компонентами існує різниця в кількості інформації, яку вони повертають: рекомендаційна система повертає 10 результатів, а пошукова система — стільки, скільки знайшлося на запит користувача.

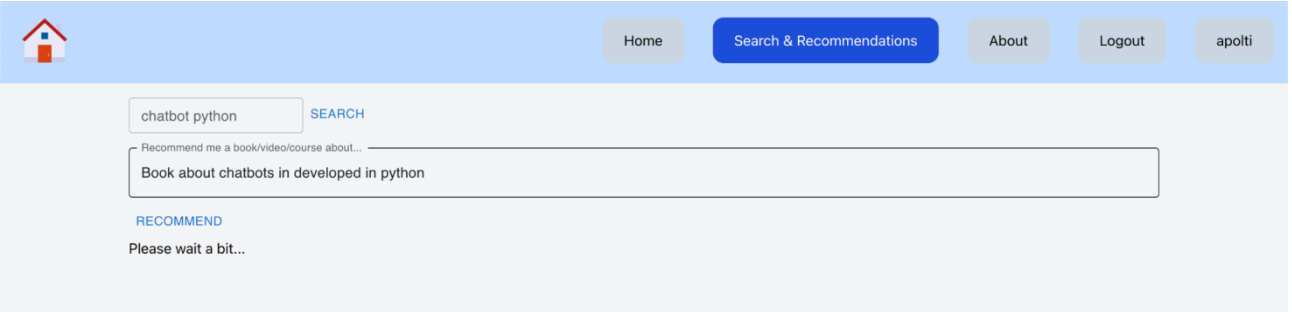


*Рис. 2.8. Режим очікування відповіді сервера при натисканні кнопки "Рекомендувати" на сторінці "Знайти і рекомендувати"*

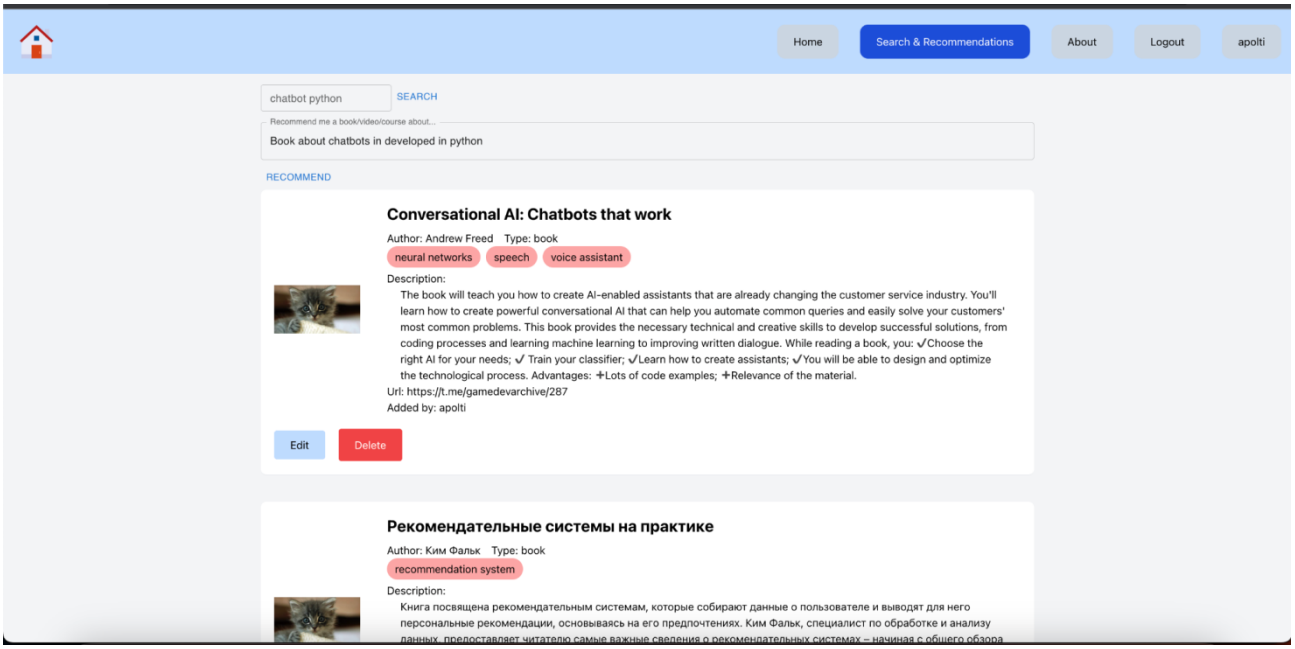


*Рис. 2.9. Сторінка після повернення результатів із серверної частини*

Ця сторінка не потребує реєстрації користувача, тож кожен охочий може зайти на неї та отримати результати на свій запит.

## Висновки до розділу 2

Загальною метою розділу було створення функціональної та ефективної рекомендаційної системи. Ми зробили огляд структури РС, а також представили алгоритм, який використовуємо в нашій рекомендаційній системі.

Описано клієнтську частину рекомендаційної системи, яка відповідає за взаємодію з користувачем. Визначено основні функціональні можливості клієнтської частини, до прикладу  пошук ресурсу, отримання рекомендацій, відображення результатів та забезпечення зручного інтерфейсу для взаємодії із системою.

Розглянуто серверну частину рекомендаційної системи. Проаналізовано архітектурні аспекти серверної частини,  пов'язані з обробкою ембедингів описів ресурсів, які необхідні для формування рекомендацій, а саме обробку, збереження і змінення даних описів ресурсів та обробку запитів.

Оглянуто алгоритм, який використовуємо в рекомендаційній системі. Він базується на використанні мультилінгвальної моделі BERT для перетворення запитів користувача у векторні представлення, відомі як ембединги. За допомогою цих ембедингів система обчислює косинусну подібність між вектором запиту та векторами описів доступних ресурсів. Для пошуку найбільш подібних до запиту користувача описів  використовується алгоритм пошуку k найближчих сусідів.

Застосування мультилінгвальної моделі BERT дозволяє здійснювати ефективне представлення текстових даних, ураховуючи семантичну схожість слів та контексту. Косинусна подібність між ембедингами дає змогу оцінити ступінь схожості між запитом та описами ресурсів. Алгоритм пошуку k найближчих сусідів забезпечує вибір найбільш схожих описів, які відповідають запиту користувача.

Цей алгоритм є важливим компонентом рекомендаційної системи, оскільки він допомагає забезпечити персоналізовані рекомендації, ураховуючи інтереси та потреби користувача.

# РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ДЛЯ СИСТЕМИ РЕКОМЕНДАЦІЇ РЕСУРСІВ ІЗ ПРИКЛАДНОЇ ЛІНГВІСТИКИ

Оскільки наша система рекомендації ресурсів повинна бути у відкритому доступі, щоб нею могли користуватися всі охочі прикладні лінгвісти, було вирішено розробити вебзастосунок, де б зберігалася вся інформація. Також у вільний доступ на Github викладено код реалізації. Ці рішення дозволять:

- залучити всіх охочих до розширення бази ресурсів,
- усім охочим долучитися до розробки цього проєкту,
- збільшувати спільноту прикладних лінгвістів.

Проте існують і певні потенційні ризики, пов’язані з цим підходом, а саме:

- Зловживання користувачами: користувачі можуть заповнювати базу даних системи непотрібною інформацією або спамом, що призводить до зниження якості рекомендацій і займає додатковий простір у базі даних.
- Обмежена якість ресурсів: якщо ресурси неналежно описані або містять недостатньо інформації, система може зазнавати складнощів у наданні відповідних рекомендацій.
- Ризик безпеки: розгортання системи рекомендацій у відкритому доступі завжди супроводжується ризиком хакерських атак. Зловмисники можуть аналізувати код програмного забезпечення та використовувати потенційні вразливості проти системи, що може призвести до витоку конфіденційної інформації користувачів.

Отже, при оприлюдненні рекомендаційних систем необхідно враховувати ці потенційні ризики й уживати заходів для гарантування безпеки та якості системи.

## 3.1. Аналіз принципів, методів та засобів побудови вебзастосунку

Для роботи було вирішено розділити вебінтерфейс та серверну частину. Це дасть змогу:

- мати більшу автономність,

- мати можливість розробляти їх окремо одне від одного, що допоможе на етапі спільної роботи декількох людей, якщо хтось захоче долучитися до створення цього продукту,
- розмістити вебінтерфейс і серверну частину на різних платформах, оскільки після закриття можливості безкоштовного місця на платформі Heroku, стало складно розміщувати моноліт (тобто, і вебінтерфейс, і сервер разом) на одній платформі безоплатно.

### 3.1.1 Поняття API, REST, MVC

Для того, щоб розробити якісний вебзастосунок, який відповідає сучасним стандартам і вимогам, нам довелося ознайомитися з найкращими практиками та архітектурними шаблонами веброзробки. Важливі для нашого дослідження її аспекти та поняття потрактуємо нижче.

Серверна частина розроблена у вигляді API. "Прикладний програмний інтерфейс (API) — це сукупність засобів та правил, що дозволяють стандартизовану взаємодію між окремими модулями програмного забезпечення або між програмним та апаратним забезпеченням" [12]. Тобто, це програма, яка створена для того, щоб взаємодіяти з іншими програмами, при цьому API не знає, як ці сторонні програми працюють. У веброзробці API – це, як правило, набір функцій коду (наприклад, методів, властивостей, подій і URL-адрес), які розробник може використовувати у своїх програмах для взаємодії з компонентами вебінтерфейсу користувача або іншим програмним/апаратним забезпеченням користувача [12]. У нашому випадку серверна частина (web API) отримує від вебінтерфейсу запити, взаємодіє з базою даних для отримання необхідної інформації і опрацьовує ембединги, які зберігають семантичну інформацію про описи ресурсу й використовуються для створення рекомендацій.

*Рис. 3.1. Схема взаємодії вебінтерфейсу, бази даних та API*

Перевагою API є те, що він дозволяє відкрити доступ до ресурсів багатьом програмам одночасно. Наприклад, якщо у компанії є сайт, який отримує дані із серверного API, і вона захотіла створити до нього мобільний застосунок, то потрібно створити лише інтерфейс для користувача, адже API все одно, хто є клієнтом: чи то вебінтерфейс, чи мобільний застосунок, чи навіть інший API. Як і кому відкривати доступ, вирішувати компанії. Підключення до API та створення застосунків, які отримують дані або функціональні можливості, надані API, можна виконати за допомогою розподіленої інтеграційної платформи, яка з'єднує все, у тому числі застарілі системи та Інтернет речей (IoT).

У контексті дослідження варто звернути увагу на окремий тип API, який називають RESTful. RESTful API — це прикладний програмний інтерфейс, що дотримується принципу стилю архітектури REST (Representational State Transfer) при роботі з HTTP-запитами (Hyper Text Transfer Protocol) [31]. REST також використовує поняття ресурсів, до яких можна отримати доступ за допомогою HTTP-запитів і виконати низку операцій, а саме CRUD-операцій, тобто create (створити), read (прочитати), update (змінити), delete (видалити). При цьому HTTP-хедери та найменування URL-запиту має свій формат за конвенцією, наприклад, щоб отримати (тобто прочитати) статтю під номером 3, нам потрібно відправити запит GET http://some-url.com/articles/1, щоб видалити її (якщо в користувача є таке право) — DELETE http://some-url.com/articles/1. Якщо ж ми хочемо отримати всі статті, запитом буде GET /articles. REST-архітектура дозволяє стандартизувати роботу з HTTP-запитами й уникнути URL на зразок http://some-url.com/usi-statti.

Наш API було створено за правилами версіювання API [17], а саме:

1. Зворотна сумісність.
2. Оновлення документації API, щоб відобразити нові версії.
3. Адаптація версії API до вимог бізнесу.
4. Питання безпеки API на першому плані.
5. Налаштування версій API відповідно до масштабу.

Також важливим архітектурним шаблоном, який ми використали для створення вебзастосунку є шаблон проєктування Model View Controller (модель, представлення, контролер, MVC), який застосовуємо у вибраному фреймворку, про який ітиметься далі. MVC є популярним способом організації коду. Його ідея полягає в тому, що кожен модуль нашого коду має свої обов'язки, і вони відрізняються від іншого модуля [51]. Частина коду зберігає дані нашої програми, частина робить гарний вивід інформації для користувача, а частина коду контролює роботу програми. MVC організовує основні функції коду у власні, акуратно організовані блоки. Завдяки цьому розділенню відповідальності обдумувати свою програму, переглядати її та додавати новий функціонал набагато легше та доступніше, бо зрозуміло, що і де повинно бути розміщене [78].

**Модель**: код моделі зазвичай відображає речі реального світу. Цей код може містити необроблені дані або визначати основні компоненти вашої програми. Наприклад, якби ви створювали програму зі списком завдань, які потрібно виконати, код моделі визначав би, що таке "завдання", а що таке "список", які атрибути вони повинні мати, оскільки це основні компоненти такої програми.

**Представлення**: код представлення складається з усіх функцій, які безпосередньо взаємодіють із користувачем. Це код, завдяки якому ваша програма виглядає привабливо, а також визначає, як ваш користувач бачить її та взаємодіє з нею. Сюди відносяться HTML-сторінки, які відображаються у користувача. На противагу їм, це можуть бути JSON-об’єкти, якщо серверна частина працює як API.

**Контролер**: код контролера діє як зв’язок між моделлю та представленням, отримуючи дані користувача та вирішуючи, що з ними робити. Це логічний центр програми, який пов’язує модель і представлення. Іноді можна зустріти метафору, що контролер є "мозком" програми, однак ми вважаємо, що це твердження вводить в оману, оскільки зазвичай прийнято контролер робити "худим", або "дурним", щоб він тільки мінімально обробляв

запит і віддавав результат (представлення), а от модель прийнято робити "товстою", тобто такою, що обробляє більшу частину логіки, пов’язану з ресурсом, який запитує користувач. Така стратегія називається "товста модель — худий контролер" [54].

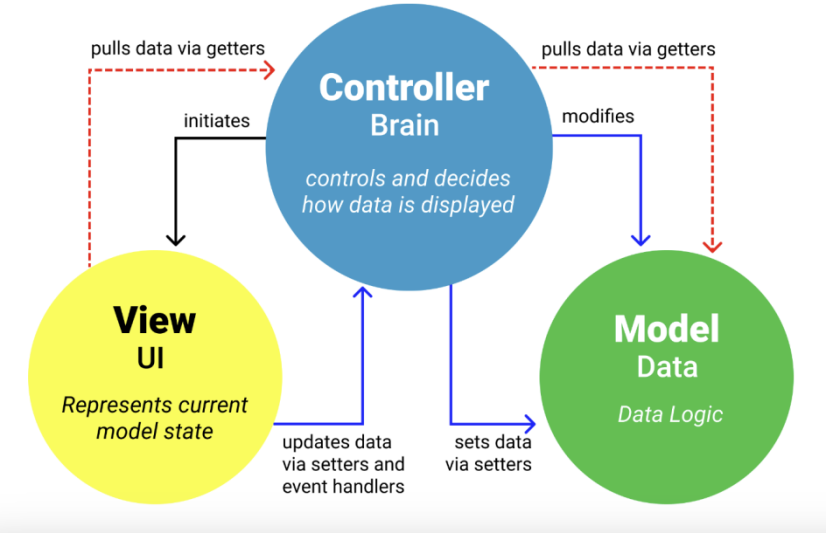


*Рис. 3.2. Високорівнева схема представлення шаблону MVC [38]*

### 3.1.2. Технологічні рішення реалізації вебзастосунку

Для реалізації серверної частини вебзастосунку у вигляді API було обрано мову програмування Ruby та фреймворк Ruby on Rails.

Ruby — це динамічна мова програмування з відкритим кодом, зосереджена на простоті та продуктивності [71], яка нагадує Python. Вона стала популярною завдяки двом своїм фреймворкам для веброзробки — Sinatra і Ruby on Rails, хоча ця мова має й інші бібліотеки для різних цілей, зокрема машинного навчання, однак серверна частина веброзробки є її основною нішею.

Ruby on Rails, своєю чергою, є фреймворком веброзробки, який застосовують у своїх додатках такі компанії, як GitHub, GitLab, Shopify, Twitch, AirBnB, Kickstarter, Hulu тощо [72].

Ми обрали ці технології, оскільки вони мають значні переваги:

- мова Ruby є високорівневою мовою програмування з простим, але багатим синтаксисом, що дозволяє абстрагуватися від деталей реалізації та сконцентруватися на власне розробці логіки програми;

- фреймворк Ruby on Rails дозволяє швидко налагодити роботу та розробити мінімальний життєспроможний продукт (Minimum Viable Product, MVP) [82] за лічені дні або іноді навіть години;
- Ruby on Rails також має великий набір дефолтних конвенцій, яких достатньо для розробки більшості застосунків, багато корисних інструментів і бібліотек, а також велику спільноту програмістів, готових допомогти з проблемами, що можуть виникнути під час розробки [62].

Однак ці технології мають і свої недоліки, зокрема Ruby є неефективною з точки зору швидкості та пам’яті, а фреймворк Ruby on Rails не відзначається гнучкістю, якщо з якихось причин проєкту потрібно вийти за межі чинних конвенцій. Але ці недоліки, загалом, проявляються у великих і складних проєктах і не повинні завадити розробці нашого невеликого вебзастосунку.

Для тестування серверної частини застосунку було використано фреймворк Rspec [70] і його бібліотеку для Ruby on Rails — rspec-rails [70], що здійснює юніт, інтеграційне та системне тестування за методологією "розробка, орієнтована на поведінку" (Behaviour Driven Development, BDD) [15]. Ця методологія передбачає, що ми орієнтуємося на те, як повинна поводитися програма, я не на те, який алгоритм вона повинна використовувати.

Як базу даних було обрано об’єктно-реляційну базу даних з відкритим кодом PostgreSQL через її гнучкість, широкий функціонал [60] та гарну інтеграцію з нашим бекенд-фреймворком за допомогою бібліотеки pg [58].

Для фронтенду було обрано Javascript фреймворк React [63], оскільки він є дуже популярним, полегшує створення інтерактивних інтерфейсів користувача.

Основною його особливістю є компоненти як інкапсульовані елементи сторінки зі своїм станом, що можуть змінюватися. Цей фреймворк є декларативним, тобто якщо якийсь компонент змінив стан, то React повторно відобразить не всю сторінку, а тільки цей компонент. Також React не робить припущень щодо решти стеку технологій, тобто його можна безболісно інтегрувати із серверною частиною на Ruby on Rails.

Для зберігання даних було використано інструмент Redux [65], тому що він є популярним рішенням для збереження інформації в браузері (що нам потрібно для зберігання інформації про аутентифікацію користувача). Для стилізації фронтенд-компонентів було обрано фреймворк MaterialUI, який хоч і має лише середню візуальну привабливість, проте має багато готових компонентів, що значно пришвидшує розробку.

Увесь проєкт було докеризовано за допомогою інструментів Docker [24] і docker-compose [25]. Docker є менеджером контейнерів, що, зі свого боку, є просто програмами, ізольованими від решти операційної системи. Це дає можливість встановлювати власні залежності просто в цей ізольований простір, що дозволяє впевнитися у тому, що програма завжди буде виконуватися на різних операційних системах і в різних середовищах буде працювати однаково. Це дасть змогу полегшити деплой і залучити нових розробників до процесу, адже їм не доведеться встановлювати всі залежності самостійно й розбиратися з помилками, які неминуче виникнуть під час цього, а тільки необхідно буде встановити Docker і docker-compose, які завантажать залежності всередині контейнера, де вже було перевірено, що це відбувається без проблем і програма працює правильно. Docker-compose є допоміжним інструментом роботи з кількома контейнерами, адже для кожного сервісу потрібно створювати окремий контейнер, як-от для серверної частини, для бази даних, для допоміжних сервісів (наприклад, pgAdmin, що є сервісом візуалізації бази даних, чи mailcatcher, який перехоплює електронні листи, надіслані нашою серверною частиною, що необхідно для коректної їх розробки).

Код проєкту було викладено у вільний доступ на Github за посиланням https://github.com/Poltieva/nlp-resources [59], що дозволить усім охочим долучитися до покращення продукту. Для цього було налагоджено засоби безпеки, зокрема неможливість перезаписувати код на головній гілці, а також додано інструменти безперервної інтеграції [81] у вигляді Github Actions [32] для оптимізації процесу розробки. Процес розробки відстежується за допомогою Kanban-дошки [80] на вебресурсі Trello [76].

### 3.2. Результати розробки вебзастосунку рекомендації ресурсів із прикладної лінгвістики

#### 3.2.1. Даталогічна модель даних

Основою будь-якого мобільного або вебзастосунку є його бізнес-логіка, яка зазвичай зберігається в базах даних.

```
sqlite> sqlite> pragma table_info('resources');
cid  name         type         notnull  dflt_value                        pk
---  -----------  -----------  -------  --------------------------------  --
0    id           INTEGER      1                                          1
1    name         varchar      1                                          0
2    description  TEXT         0        NULL                              0
3    url          varchar      0        NULL                              0
4    medium       INTEGER      1                                          0
5    created_at   datetime(6)  1                                          0
6    updated_at   datetime(6)  1                                          0
7    author       varchar      0        NULL                              0
8    keywords     TEXT         0        NULL                              0
9    image_url    TEXT         0        'https://placekitten.com/640/360'  0
```

*Рис. 3.3. Скриншот таблиці ресурсів*

У нашому випадку основним носієм бізнес-логіки є таблиця, у якій зберігається інформація про ресурси з прикладної лінгвістики (див. рис. 3.3 вище для детальної інформації). Вона має 10 стовпців, зокрема:

- ім’я для вказання назви ресурсу;
- опис для більш детальної інформації про ресурс. Він може бути порожнім, хоча цього не рекомендують;
- URL для зберігання посилання на ресурс. Хоч при розробці рекомендаційної системи ми орієнтувалися на праці з рекомендації книжок, та оскільки в нашій системі можуть бути й інші типи ресурсів, ми не надаємо колонки для зберігання тексту книжки, а тільки URL. До того ж зберігання всього тексту книжки порушує авторське право;
- вид ресурсу (medium), який може бути одним із заздалегідь визначених варіантів, а саме: книжка, відео, курс, стаття, подкаст, інше. Тому ця

колонка має цілочисельний тип, а список варіантів задається на рівні моделі (див. рис. 3.4). Вид спеціально створений так, аби користувачам вистачило категорій і не доводилося вигадувати своїх;

- автор ресурсу. Колонка може бути порожньою;
- ключові слова, що автоматично створюються на основі опису ресурсу. Хоча ця колонка має тип текст, насправді вона зберігає масив рядків, що в SQLite3 відповідає типу текст;
- URL для картинки, якщо користувач хоче додати її, якщо ні, то буде автоматично надано картинку з котиками, що є у вільному доступі на сайті https://placekitten.com розміром 640x360 (щоразу нову);
- стовпці для зберігання інформації про те, коли було створено й змінено рядок бази даних, щоб слідкувати за зміною об'єкта (також створюються автоматично).

```
1   # frozen_string_literal: true
2
3   class Resource < ApplicationRecord
4     enum medium: {
5       book: 0,
6       video: 1,
7       course: 2,
8       article: 3,
9       podcast: 4,
10      other: 5
11    }, _prefix: true
```

*Рис. 3.4. Скриншот коду моделі ресурсів з визначенням виду ресурсу*

Інформація про користувачів зберігається в окремій таблиці із зазначенням ніку на нашому сайті, електронної пошти, паролю, з можливістю додати посилання на картинку і створити короткий опис про себе, а також містить деякі допоміжні стовпці для аутентифікації користувача.

```
sqlite> pragma table_info('users');
cid  name                   type         notnull  dflt_value  pk
---  ---------------------  -----------  -------  ----------  --
0    id                     INTEGER      1                    1
1    username               varchar      1                    0
2    image                  varchar      0        NULL        0
3    bio                    TEXT         0        NULL        0
4    email                  varchar      1        ''          0
5    encrypted_password     varchar      1                    0
6    created_at             datetime(6)  1                    0
7    updated_at             datetime(6)  1                    0
8    reset_password_token   varchar      0                    0
9    reset_password_sent_at datetime(6)  0                    0
10   remember_created_at    datetime(6)  0                    0
```

*Рис. 3.5. Скриншот таблиці користувачів*

Оскільки було вирішено аутентифіковувати користувачів за допомогою JWT-токенів [39], то нам знадобилося кілька допоміжних таблиць для аутентифікації — oauth_applications зберігає інформацію про програми, яким було надано токени, oauth_access_tokens зберігає інформацію власне про випущені токени.

```
sqlite> sqlite> pragma table_info('oauth_access_tokens');
cid  name                  type         notnull  dflt_value  pk
---  --------------------  -----------  -------  ----------  --
0    id                    INTEGER      1                    1
1    resource_owner_id     INTEGER      0        NULL        0
2    application_id        INTEGER      1                    0
3    token                 varchar      1                    0
4    refresh_token         varchar      0        NULL        0
5    expires_in            INTEGER      0        NULL        0
6    revoked_at            datetime(6)  0        NULL        0
7    created_at            datetime(6)  1                    0
8    scopes                varchar      0        NULL        0
9    previous_refresh_token varchar     1        ''          0
```

*Рис. 3.6. Скриншот таблиці випущених токенів*

```
sqlite> sqlite> pragma table_info('oauth_applications');
cid  name          type          notnull  dflt_value  pk
---  ------------  ------------  -------  ----------  --
0    id            INTEGER       1                    1
1    name          varchar       1                    0
2    uid           varchar       1                    0
3    secret        varchar       1                    0
4    redirect_uri  TEXT          0                    0
5    scopes        varchar       1        ''          0
6    confidential  boolean       1        1           0
7    created_at    datetime(6)   1                    0
8    updated_at    datetime(6)   1                    0
```

*Рис. 3.7. Скриншот таблиці програм, що використовують токени*

### 3.2.2. Огляд вебзастосунку

Домашня сторінка сайту є списком із усіма ресурсами з прикладної лінгвістики. Вона відкрита для перегляду усіма користувачами, у тому числі незареєстрованими, однак для зареєстрованих користувачів відкритий функціонал додавання нових ресурсів, редагування і видалення вже наявних.

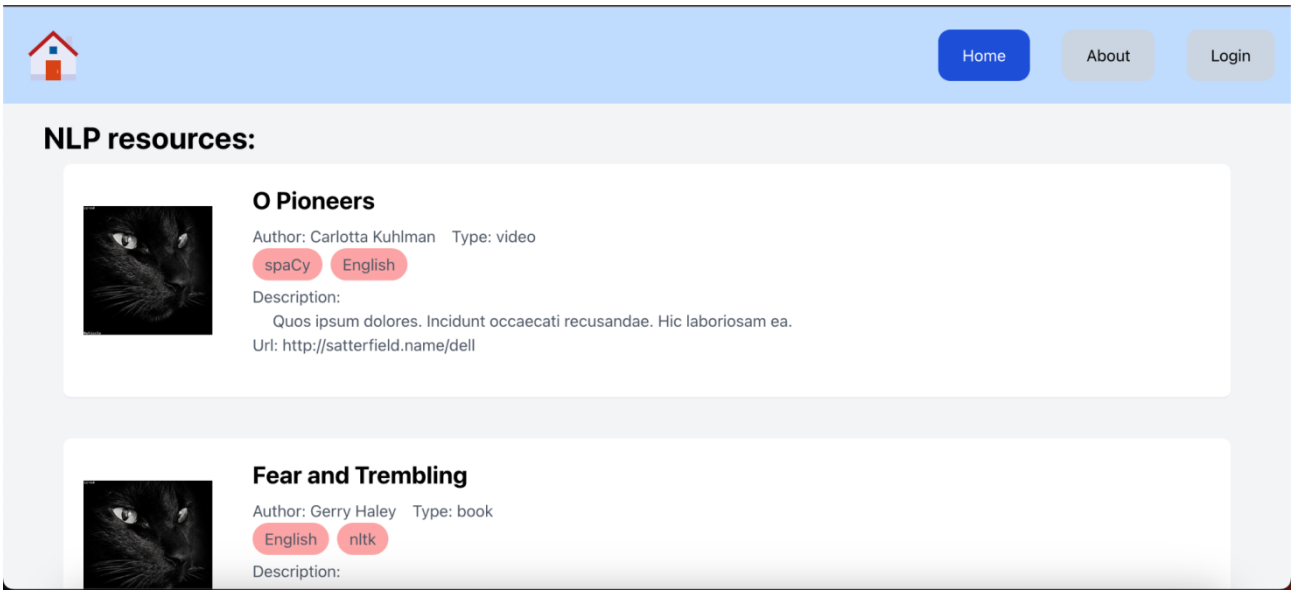


*Рис. 3.8. Скриншот домашньої сторінки сайту (без реєстрації)*

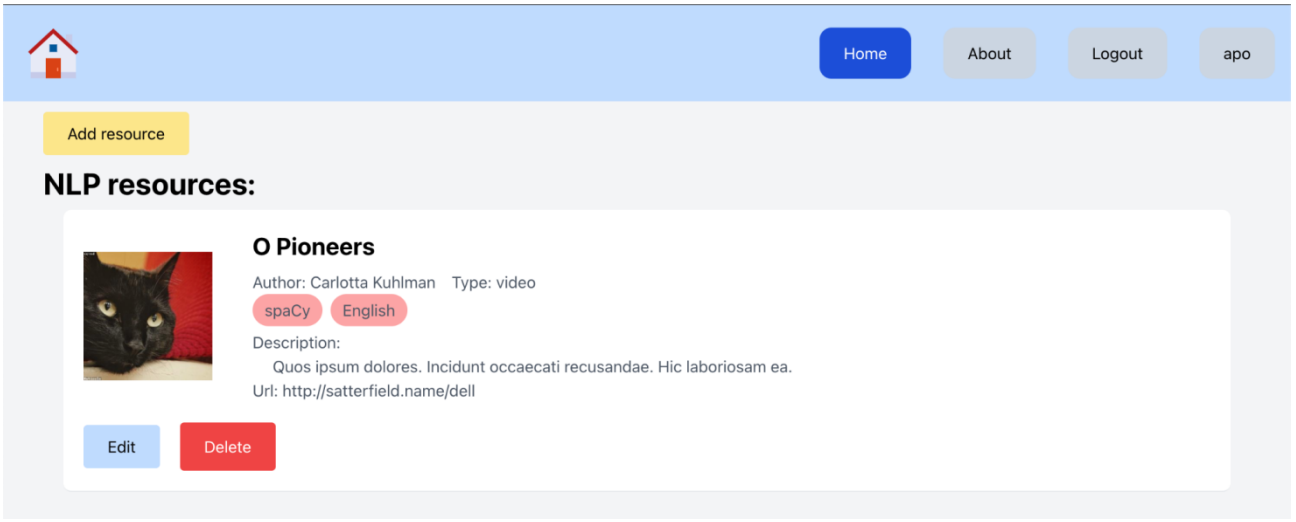


*Рис. 3.9. Скриншот домашньої сторінки сайту (з реєстрацією)*

Є також можливість редагувати профіль користувача та секція з питаннями-відповідями й стандартний функціонал реєстрації-входу-виходу.

При натисканні на кнопку "Створити ресурс" користувачу відкривається сторінка з формою.

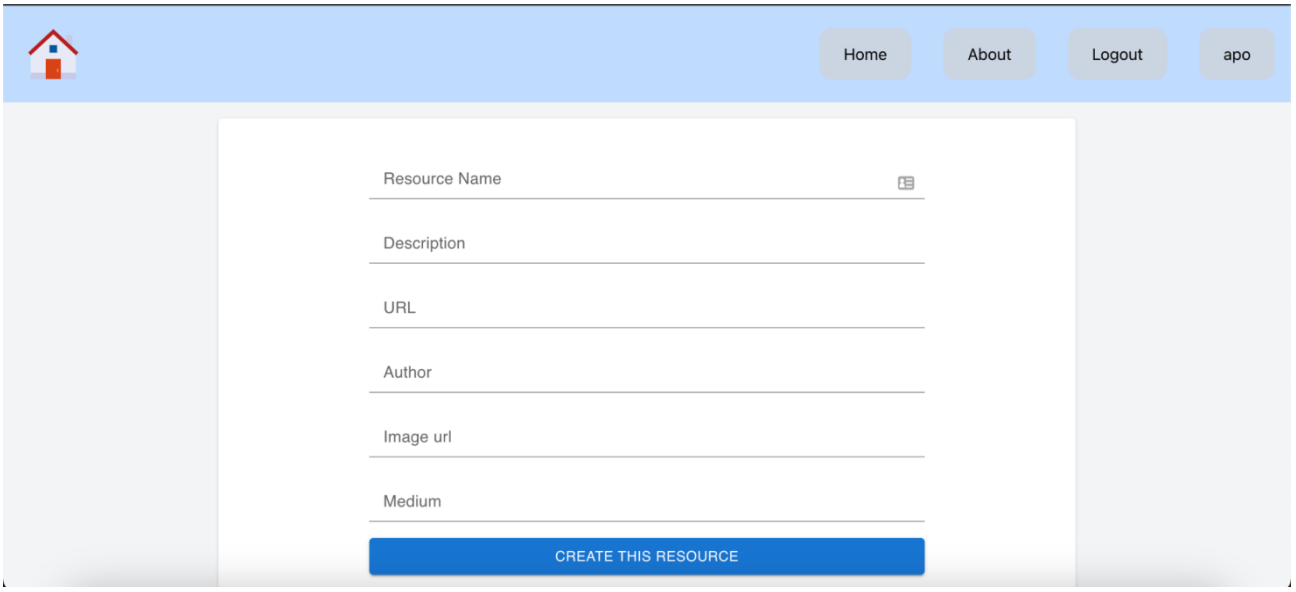


*Рис. 3.10. Скриншот сторінки створення нового ресурсу*

Якщо користувач хоче змінити ресурс, що вже існує, після натискання кнопки "Редагувати" в картці цього ресурсу йому відкриється схожа форма, але із заповненими полями.

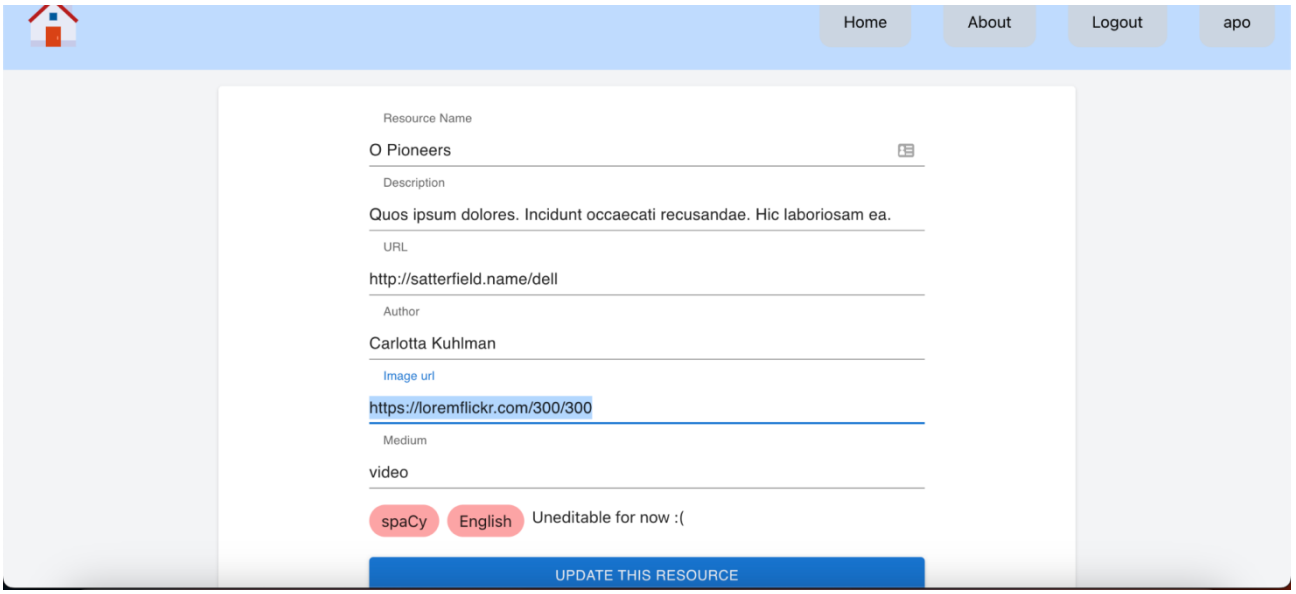


*Рис. 3.11. Скриншот сторінки редагування наявного ресурсу*

Доступні функціональні можливості вебзастосунку дозволяють користувачам швидко та зручно знаходити необхідні ресурси, а також збирати та обробляти дані для дальшого вдосконалення РС. З урахуванням отриманих результатів, можна зробити висновок про корисність розробленого вебзастосунку як інструменту для використання системи рекомендації ресурсів із прикладної лінгвістики.

Важливо зазначити, що застосунок залишається відкритим для подальшого розширення, оскільки деякі заплановані ідеї ще не були повністю реалізовані через обмежений часовий ресурс. Для подальшого розвитку застосунку перспективними є:

- тестування серверної та клієнтської частини. Створення бази автоматичних тестів допоможе виявити та виправити можливі помилки й проблеми;
- пошук ресурсів за тегами. Можливість пошуку ресурсів за використовуваними тегами дозволить користувачам швидко знаходити потрібні ресурси на основі їхніх інтересів та попередніх уподобань;
- використання ембедингів тегів для покращення процесу створення рекомендацій. Ембединги тегів можуть створювати компактні числові

представлення тегів, що полегшує роботу з ними та дозволяє точніше налаштовувати рекомендації відповідно до індивідуальних уподобань користувачів.

Ці функціональні можливості можуть сприяти подальшому розвитку та покращенню застосунку, роблячи його функціональнішим і зручнішим для користувачів.

## Висновки до розділу 3

Для того, аби успішно реалізувати рекомендаційну систему та надати користувачам вільний доступ до неї, ми розробили вебзастосунок. Було проведено аналіз принципів, методів та засобів побудови такого інструменту для системи рекомендації ресурсів із прикладної лінгвістики, зокрема поняття API, REST та MVC, що є основними принципами, за якими здійснювалася розробка нашого вебзастосунку.

Окрім того, було проаналізовано технологічні рішення, необхідні для реалізації вебзастосунку.

Застосунок поділено на серверну частину у вигляді API на фреймворці Ruby on Rails з тестуванням на основі Rspec і докеризацією та вебінтерфейс на ReactJS [65] + Redux [66] з використанням CSS фреймворку Bootstrap. Унаслідок цієї роботи було отримано інформаційну вебсистему, яка швидко працює при великій кількості інформації, проста в дизайні й зрозуміла для користувача. Система повністю адаптивна до всіх пристроїв та працює на всіх сучасних браузерах.

Створено вебінтерфейс застосунку і даталогічну модель даних, реалізовано аутентифікацію користувачів.

Даталогічну модель даних розроблено таким чином, що вона охоплює всі необхідні сутності, які дозволяють забезпечити ефективну та стабільну роботу застосунку в цілому і системи рекомендування зокрема, наведено схеми бази даних. Також проведений огляд вебзастосунку (у вигляді скриншотів з описами), де було продемонстровано його основні функції та можливості.

Результат огляду вебзастосунку дозволяє зробити висновок про його ефективність та зручність для користувачів. Функціональні можливості вебзастосунку забезпечують користувачам швидкий та зручний пошук необхідних ресурсів, а також відкривають можливість збирати та обробляти дані для подальшого вдосконалення рекомендаційних алгоритмів. Отже, розроблений вебзастосунок корисним інструментом для використання системи

рекомендації ресурсів із прикладної лінгвістики, однак у нього є потенціал для розширення та удосконалення.

# ВИСНОВКИ

Рекомендаційні системи є важливим складником сучасних технологій, які надають інформацію та рекомендації користувачам на основі їхніх уподобань та поведінки. Вони використовують два основні підходи для збору інформації та надання рекомендацій — колаборативний підхід і підхід на основі контенту.

Колаборативний підхід використовує інформацію про взаємодію користувачів із системою, зокрема їхні оцінки, і враховує схожість між користувачами або об'єктами для надання рекомендацій. Визначення на основі контенту використовує характеристики самого об'єкта або інформацію про нього, щоб зробити рекомендації. Наприклад, це може бути використання ключових слів, опису або категорій об'єктів.

Рекомендаційні системи можуть застосовувати різні математичні алгоритми для здійснення рекомендацій. Деякі з них охоплюють логістичну регресію, факторизаційні машини та нейронні мережі. Ці алгоритми дають системі можливість аналізувати інформацію про користувачів та об'єкти, знаходити кореляції та патерни, що допомагають зробити точні рекомендації на основі цих взаємозв'язків.

Незважаючи на їхню важливість і користь, РС стикаються з рядом проблем, які вимагають уваги та розв’язання. Серед цих проблем можна виділити точність рекомендацій, етичні аспекти збору, зберігання та використання даних, а також конфіденційність користувачів.

Одним з основних аспектів, який викликає занепокоєння користувачів, є збереження конфіденційності їхніх персональних даних. Безпека та конфіденційність інформації в рекомендаційних системах є важливою умовою, щоб користувачі могли вірити цим системам та спокійно працювати з ними. Відсутність довіри може призвести до відмови від використання системи або навіть до втрати користувачів. Для забезпечення конфіденційності інформації в рекомендаційних системах використовують такі рішення та підходи, як рандомізація даних, безпечне багатостороннє обчислення та k-анонімність.

У результаті роботи була створена інформаційна вебсистема, яка відрізняється високою продуктивністю, простим дизайном та зрозумілим інтерфейсом для користувачів. Цей вебзастосунок забезпечує швидкий доступ до рекомендованих ресурсів і відповідає потребам користувачів із прикладної лінгвістики. Для успішної реалізації застосунку було проаналізовано низку різних технологічних рішень. З-поміж них — Ruby on Rails, Rspec, Docker, ReactJS та Redux. Як основний фреймворк для розробки ми використовували Ruby on Rails , для написання тестів — Rspec, для контейнеризації додатку — Docker, а для реалізації користувацького інтерфейсу — ReactJS та Redux.

Розроблена РС складається з двох компонентів. Клієнтська частина системи відповідає за взаємодію з користувачем і має такі функціональні можливості, як пошук, рекомендації та зручний інтерфейс. Серверна частина системи зосереджена на обробці ембедингів описів ресурсів та обробці запитів. Використовуємо алгоритм на основі мультилінгвальної моделі BERT, який перетворює запити користувача у векторні представлення, а косинусна подібність та алгоритм пошуку k найближчих сусідів допомагають знайти найбільш подібні описи ресурсів. Цей алгоритм забезпечує персоналізовані рекомендації та покращує якість рекомендацій для користувачів.

Отже, реалізована система має важливе практичне значення для дослідників, які цікавляться прикладною лінгвістикою, оскільки наразі немає порталу, який би зберігав та рекомендував ресурси з прикладної лінгвістики.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Байдак В.Є., Мазурова О.О. Комбінований підхід до побудови рекомендаційної системи для онлайн-системи продажу електронних ігор. Інноваційні технології: матеріали наук.-техн. конф. студентів, аспірантів, докторантів та молодих учених / за заг. ред. П.В. Горінова, К.О. Бабікової, Л.М. Мельничук; ІНТЛ НАУ (м. Київ, 25-26 листоп. 2020 р.). Київ, 2020. URL: http://cnt.nau.edu.ua/sites/default/files/mat_20.pdf#page=106
2. Когулько О.С. Рекомендаційна інформаційна система на основі вподобань користувачів. URL: https://ela.kpi.ua› bitstream› Kohulko_magistr
3. Мелешко Є.В., Семенов С.Г., Хох В.Д. Дослідження методів побудови рекомендаційних систем в мережі Інтернет. *Системи управління, навігації та зв'язку.* 2018. Вип.1. С. 131-136. URL: https://repository.kpi.kharkov.ua/server/api/core/bitstreams/92a26146-6437-41de-886b-59b17f86ad66/content
4. Міхав В.В., Мелешко Є.В., Дрєєв О.М., Лавданський А.О. Модель рекомендаційної системи для ком'ютерних мереж типу peer to peer. *Системи управління, навігації та зв'язку.* Збірник наукових праць. Полтава: ПНТУ, 2023. Т. 1 (71). С. 112-117. URL: http://journals.nupp.edu.ua/sunz/article/view/2839
5. Нікітін О.М. Інформаційна технологія проектування рекомендаційної системи для наукометричної бази даних Scopus. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/86867/1/Nikitin_mag_rob.pdf;jsessionid=8BF24DDED9975D75856B7F99C592FF5C
6. Романюк А. Векторні представлення слів для української мови. Україна модерна. 2019. № 27. URL: https://uamoderna.com/images/archiv/27-2020/27_46_72%20Andriy%20ROMANYUK_compressed.pdf
7. Фальк К. Рекомендательные системы на практике / пер. с англ. Д. М. Павлова. Москва : ДМК Пресс, 2020. 448 с.: ил.
8. Чалий С., Лещинський В., Лещинська І. Доповнення вхідних даних рекомендаційної системи в ситуації циклічного холодного старту з

використанням темпоральних обмежень типу "NEXT". *Системи управління, навігації та зв'язку.* Збірник наукових праць. Полтава: ПНТУ, 2019. Т. 4 (56). С. 105-109. URL: http://journals.nupp.edu.ua/sunz/article/view/1653

9. Ширалієв А.Е. О. Рекомендаційні системи для інтернет-ресурсів, що використовують алгоритми машинного навчання. URL: http://mmsa.kpi.ua/sites/default/files/abstracts/2017_b_sa_smdm_shyraliiev_ae_uk_presentation.pdf
10. Щербань В.С., Гайдейчук Ю.А. Рекомендаційна система вибору відеофільмів. URL: http://inmad.vntu.edu.ua/portal/static/4D3532BF-0BD6-4F4F-AC63-8DBD3350C894.pdf
11. Alon Schclar Ensemble methods for improving the performance of neighborhood-based collaborative filtering. / Alon Schclar, Alexander Tsikinovsky, Lior Rokach, Amnon Meisels, and Liat Antwarg. / In Proceedings of the third ACM conference on Recommender systems, RecSys '09, pages 261–264, New York, NY, USA, 2009. ACM.
12. API / Mdn Web Docs Glossary. URL: https://developer.mozilla.org/en-US/docs/Glossary/API?retiredLocale=uk
13. Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the Netflix prize dataset. CoRR: Computing Research Repository, pages 1–24, 2006.
14. Badrul Sarwar, George Karypis, Joseph Konstan, and John T. Riedl Appli-cation of Dimensionality Reduction in Recommender System - A Case Study (2000).
15. Behavior-driven development / Wikipedia. URL: https://en.wikipedia.org/wiki/Behavior-driven_development
16. Ben Schafer, Joseph Konstan, and John Riedl. 1999. Recommender systems in e-commerce. In Proceedings of the 1st ACM conference on Electronic commerce.158–166.

17. Best Practices For Your API Versioning Strategy / akana. URL: https://www.akana.com/blog/api-versioning#best-practices
18. BookMix.ru. Клуб любителей книг. URL: https://bookmix.ru/users/recommendations.phtml
19. Bowen Yuan Improving Ad Click Prediction by Considering Non-displayed Events. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management. / Bowen Yuan, Jui-Yang Hsia, Meng-Yuan Yang, Hong Zhu, Chih-Yao Chang, Zhenhua Dong, and Chih-Jen Lin. 2019. p. 329–338.
20. Brownlee, Jason Gentle Introduction to Vector Norms in Machine Learning. URL: https://machinelearningmastery.com/vector-norms-machine-learning/
21. Cynthia Dwork. Differential privacy. In Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II, pages 1–12, 2006.
22. David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. 1992. Using collaborative filtering to weave an information tapestry. Communications ofThe ACM 35, 12 (1992), 61–70
23. David Rosenblum. What anyone can know: The privacy risks of social networking sites. IEEE Security & Privacy, 5(3):40–49, May 2007.
24. Docker. URL: https://www.docker.com/
25. Docker Compose. URL: https://docs.docker.com/compose/gettingstarted/
26. Elaine Rich. User modeling via stereotypes. Cognitive Science, 3(4): 329–354, 1979
27. Elnabarawy, Islam Survey of Privacy-Preserving Collaborative Filtering / Islam Elnabarawy, Student Member, IEEE, Wei Jiang, Member, IEEE, and Donald C. Wunsch II, Fellow, IEEE. URL: https://arxiv.org/pdf/2003.08343.pdf
28. Fabiana Lorenzi and Francesco Ricci. Case-based recommender systems: A unifying view. In Intelligent Techniques for Web Personalization, volume 3169

of Lecture Notes in Computer Science, pages 89–113. Springer Berlin / Heidelberg, 2005.

29. Frank McSherry and Ilya Mironov. Differentially private recommender systems: building privacy into the Netflix prize contenders. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 627–636, 2009.
30. Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, Recommender Systems Handbook, pages 217–253. Springer US, 2011.
31. Gillis, Alexander S. REST API (RESTful API). URL: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API
32. Github Actions / Ginhub. URL: https://github.com/features/actions
33. Gleb Beliakov, Tomasa Calvo and Simon James. Aggregation of preferences in recommender systems. School of Information Technology, Deakin University – 2008. P. 705-735.
34. Goldreich O. Secure multi-party computation. 1998. URL: https://www.wisdom.weizmann.ac.il/~oded/pp.html
35. Goodbooks-10k: a new dataset for book recommendations / FastML. URL:
36. Goodreads. URL: https://www.goodreads.com/
37. Haifa Alharthi and Diana Inkpen. 2019. Study of linguistic features incorporated in a literary book recommender system. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, pages 1027–1034. ACM.
38. Hernandez, Rafael D. The Model View Controller Pattern – MVC Architecture and Frameworks Explained. URL: https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/
39. Introduction to JSON Web Tokens. URL: https://jwt.io/introduction
40. Ioannis Konstas, Vassilios Stathopoulos, and Joemon M. Jose. On social networks and collaborative recommendation. In Proceedings of the 32nd

international ACM SIGIR conference on Research and development in information retrieval, SIGIR ’09, pages 195– 202, New York, NY, USA, 2009. ACM.

41. Joseph A. Konstan and John Riedl. 2012. Recommender systems: from algorithms to user experience. User Modeling and User-adapted Interaction 22, 1 (2012), 101–123
42. Joseph A. Konstan, John Riedl, and Barry Smyth. 2007. Proceedings of the 2007ACM conference on Recommender systems. In Proceedings of the 2007 ACM conference on Recommender systems
43. Jurafsky, Daniel, Martin, James Speech and Language Processing (3rd ed. draft) URL:: https://web.stanford.edu/~jurafsky/slp3/
44. K-nearest Neighbour / Scholarpedia. URL: http://www.scholarpedia.org/article/K-nearest_neighbor
45. Landauer, T. K., Foltz, P. W., Laham, D. (1998). Introduction to Latent Semantic Analysis. Discourse Processes, випуск 25, с. 259-284.
46. Leysia Palen and Paul Dourish. Unpacking "privacy" for a networked world. In CHI ’03: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 129–136, New York, NY, USA, 2003. ACM.
47. Latent Semantic Analysis / Wikipedia URL: https://en.wikipedia.org/wiki/Latent_semantic_analysis
48. LiveLib. URL: https://www.livelib.ru/books
49. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems Article No.: 19 pp 1–19 URL: https://doi.org/10.1145/2827872
50. Melania Berbatova Overview on NLP Techniques for Content-Based Recommender Systems for Books / Melania Stoyanova Berbatova Sofia University "St. Kliment Ohridski". URL: https://aclanthology.org/R19-2009.pdf
51. MVC: Model, View, Controller / Codecademy Team. URL: https://www.codecademy.com/article/mvc

52. Naren Ramakrishnan, Benjamin J. Keller, Batul J. Mirza, Ananth Y. Grama, and George Karypis. Privacy risks in recommender systems. IEEE Internet Computing, 5(6):54–62, November 2001.
53. Nicholas Greenquist, Doruk Kilitcioglu, and Anasse Bari. 2019. Gkb: A predictive analytics framework to generate online product recommendations. In 2019 IEEE 4th International Conference on Big Data Analytics (ICBDA), pages 414–419. IEEE.
54. Pardeep Kumar Working With Thin Controller And Fat Model Concept In Laravel. URL: https://www.codementor.io/@pardeepkumar905/working-with-thin-controller-and-fat-model-concept-in-laravel-nwl7ljsst
55. Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and JohnRiedl. 1994. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. Research Papers in Economics (1994)
56. P. C. Vaz, D. Martins de Matos, and B. Martins. Stylometric Relevance-feedback Towards a Hybrid Book Recommendation Algorithm. In Proceedings of the Fifth ACM Work- 165 shop on Research Advances in Large Digital Book Repositories and Complementary Media, BooksOnline '12, pages 13–16, New York, NY, USA, 2012a. ACM. ISBN 978-1- 4503-1714-6. doi: 10.1145/2390116.2390125.. URL: http://doi.acm.org/10.1145/2390116.2390125.
57. Pearl Pu, Li Chen, and Rong Hu. 2011. A user-centric evaluation framework for recommender systems. In Proceedings of the fifth ACM conference on Recommender systems, Vol. 612. p. 157–164.
58. PG gem. URL: https://rubygems.org/gems/pg
59. Poltieva, A. NLP Resources / Github. URL: https://github.com/Poltieva/nlp-resources
60. PostgreSQL. URL: https://www.postgresql.org/

61. Privacy in Recommender Systems. *Social Media Retrieval*, edited by Xian-Sheng Hua, et al., Springer London, 2012, pp. 263–281.,. URL: https://link.springer.com/chapter/10.1007/978-1-4471-4555-4_12
62. Rak, Victor Pros & Cons of Ruby on Rails You Should Know Before Choosing the Technology for Your Startup. URL: https://sloboda-studio.com/blog/pros-and-cons-of-ruby-on-rails/
63. ReactJS. URL: https://reactjs.org/
64. Recommender Systems Handbook / R. Francesco, R. Lior, S. Bracha, K. B. Paul. – Dordrecht: Springer, 2015. – 1009 p.
65. Redux. URL: https://redux-toolkit.js.org/
66. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. URL: https://arxiv.org/abs/1908.10084
67. Robin Burke. Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction, 12:331–370, November 2002.
68. Robin Burke. Knowledge-based recommender systems. In Encyclopedia of Library and Information Systems, volume 69, pages 180–200, 2000.
69. Rspec. URL: https://rspec.info/
70. Rspec Rails. URL: https://github.com/rspec/rspec-rails
71. Ruby. URL: https://www.ruby-lang.org/en/
72. Ruby on Rails. URL: https://rubyonrails.org/
73. Separation of concerns / Wikipedia. URL: https://en.wikipedia.org/wiki/Separation_of_concerns
74. Sinha, R.R., Swearingen, K.: Comparing recommendations made by online systems and friends. In: DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries (2001)
75. Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, andThorsten Joachims. 2016. Recommendations as treatments: Debiasing learning and evaluation. International conference on machine learning. PMLR, 1670–1679.
76. Trello. URL: https://trello.com/b/HZTp3gYy/development

77. Upendra Shardanand and Pattie Maes. 1995. Social information filtering: algo-rithms for automating "word of mouth". In Proceedings of the SIGCHI Conferenceon Human Factors in Computing Systems. 210–217.
78. Wang, C. and Blei., D. M. Collaborative topic modeling for recommending scientific articles. In Proc. 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '11, pages 448–456. ACM, 2011. ISBN 978-1-4503-0813-7. doi: 10.1145/ 2020408.2020480.. URL: http://doi.acm.org/10.1145/2020408.2020480.
79. Wayne Xin Zhao RecBole: Towards a Unified, Comprehensive and Efficient Framework for Recommendation Algorithms. arXiv: Information Retrieval (2020) / Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Kaiyuan Li, Yushuo Chen,Yujie Lu, Hui Wang, Changxin Tian, Xingyu Pan, Yingqian Min, Zhichao Feng,Xinyan Fan, Xu Chen, Pengfei Wang, Wendi Ji, Yaliang Li, Xiaoling Wang, and Ji-Rong Wen. 2020.
80. What Is a Kanban Board and How to Use It? Basics Explained. / kanbanize. URL: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban-board
81. What is Continuous Integration?/ AWS. URL: https://aws.amazon.com/devops/continuous-integration/
82. What is 'Minimum Viable Product' / The Economic Times. URL: https://economictimes.indiatimes.com/definition/minimum-viable-product
83. Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. 1995. Recommend-ing and evaluating choices in a virtual community of use. In Proceedings of theSIGCHI Conference on Human Factors in Computing Systems. 194–201
84. Word Embeddings [Електронний ресурс] / Wikipedia – Режим доступу: https://en.wikipedia.org/wiki/Word_embedding
85. Zhenhua Dong Counterfactual learning for recommender system. In Fourteenth ACM Conference on Recommender Systems. / Zhenhua Dong, Hong Zhu,

Pengxiang Cheng, Xinhua Feng, Guohao Cai, Xi-uqiang He, Jun Xu, and Jirong Wen. 2020. // p. 568–569.

86. Zhenhua Dong. Counterfactual Machine Learning. / 2018. URL: http://www.cs.cornell.edu/courses/cs7792/2018fa/

87. Zhu Sun Are We Evaluating Rigorously? Benchmarking Recommendation for Reproducible Evaluation and Fair Comparison. In Fourteenth ACM Conference on Recommender Systems. / Zhu Sun, Di Yu, Hui Fang, Jie Yang, Xinghua Qu, Jie Zhang, and Cong Geng. 2020. p. 23–32.

# ДОДАТКИ

Додаток 1

Посилання на вебзастосунок: https://nlp-resources.vercel.app/

Додаток 2

schema.rb

```
ActiveRecord::Schema[7.0].define(version:
2022_08_21_174818) do
  create_table "oauth_access_tokens", force: :cascade do
|t|
    t.integer "resource_owner_id"
    t.integer "application_id", null: false
    t.string "token", null: false
    t.string "refresh_token"
    t.integer "expires_in"
    t.datetime "revoked_at"
    t.datetime "created_at", null: false
    t.string "scopes"
     t.string "previous_refresh_token", default: "", null:
false
                t.index     ["application_id"],     name:
"index_oauth_access_tokens_on_application_id"
                 t.index     ["refresh_token"],     name:
"index_oauth_access_tokens_on_refresh_token",     unique:
true
               t.index     ["resource_owner_id"],     name:
"index_oauth_access_tokens_on_resource_owner_id"
```

```
                    t.index       ["token"],       name:
"index_oauth_access_tokens_on_token", unique: true
 end


create_table "oauth_applications", force: :cascade do |t|
 t.string "name", null: false
 t.string "uid", null: false
 t.string "secret", null: false
 t.text "redirect_uri"
 t.string "scopes", default: "", null: false
 t.boolean "confidential", default: true, null: false
 t.datetime "created_at", null: false
 t.datetime "updated_at", null: false
            t.index             ["uid"],             name:
"index_oauth_applications_on_uid", unique: true
end


create_table "resources", force: :cascade do |t|
 t.string "name", null: false
 t.text "description"
 t.string "url"
 t.integer "medium", null: false
 t.datetime "created_at", null: false
 t.datetime "updated_at", null: false
 t.string "author"
 t.text "keywords"
          t.text          "image_url",          default:
"https://placekitten.com/640/360"
```

```
end
create_table "users", force: :cascade do |t|
 t.string "username", null: false
 t.string "image"
 t.text "bio"
 t.string "email", default: "", null: false
 t.string "encrypted_password", null: false
 t.datetime "created_at", null: false
 t.datetime "updated_at", null: false
 t.string "reset_password_token"
 t.datetime "reset_password_sent_at"
 t.datetime "remember_created_at"
 t.index ["email"], name: "index_users_on_email", unique:
true
       t.index       ["reset_password_token"],       name:
"index_users_on_reset_password_token", unique: true
  t.index  ["username"],  name:  "index_users_on_username",
unique: true
end
add_foreign_key                    "oauth_access_tokens",
"oauth_applications", column: "application_id"
end
```

encode_query.py

```
from sentence_transformers import SentenceTransformer
from sys import argv, exit
import pickle
import tensorflow as tf

def encode_query(model, query):
    try:
        query_embedding = model.encode(query)
    except:
        print("Error encoding query")
        exit(1)
    return query_embedding

def load_embeddings():
    in_path = './db/embeddings.pkl'

    books_data = []
    with open(in_path, 'rb') as f:
        while True:
            try:
                book_embeddings = pickle.load(f)
                books_data.append(book_embeddings)
            except EOFError:
                break
    return books_data

def cosine_similarities(a, b):
    return tf.matmul(a, b, transpose_b = True)
```

```
def main(model, query, tags = []):
    DB_data = load_embeddings()
     DB_embeddings = tf.constant([list(d.values())[0] for
d in DB_data])
       DB_embeddings  =  tf.nn.l2_normalize(DB_embeddings,
axis=1)
    query_embedding = encode_query(model, query, tags)
    query_embedding =
tf.nn.l2_normalize(tf.expand_dims(tf.constant(query_embe
dding),0), axis=1)
          scores   =   cosine_similarities(query_embedding,
DB_embeddings)
              top_k_indices    =    tf.math.top_k(scores,
10)[1].numpy().squeeze().tolist()

     candidates  =  [list(DB_data[i].keys())[0]  for  i  in
top_k_indices]

    return candidates

if __name__ == '__main__':
                                          model              =
SentenceTransformer('paraphrase-multilingual-MiniLM-L12-
v2')
    query = argv[1]
    print(main(model, query))
```