

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
МЕТОДУ РАНЖУВАННЯ ТОВАРІВ В ІНТЕРНЕТ-МАГАЗИНІ
RESEARCH AND SOFTWARE IMPLEMENTATION OF
THE METHOD FOR RANKING GOODS IN AN ONLINE STORE

Виконав(ла): студент(ка) 2 курсу, групи КНТ-133м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ЛИСЕНКО І.К.

(ПРИЗВИЩЕ та ініціали)

Керівник ГЛАДКОВА О.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЛИСЕНКО Ігоря Костянтиновича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методу ранжування товарів в інтернет-магазині. Research and Software Implementation of the Method for Ranking Goods in an Online Store

керівник проєкту (роботи) к.т.н., ГЛАДКОВА Ольга Миколаївна
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 07 ” жовтня 2024 року № 398

2. Строк подання студентом проєкту (роботи) 10 грудня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір і обґрунтування матеріалів і методів розробки. 3. Проектування архітектури системи. 4. Розробка програмної системи. 5. Тестування програмної системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ГЛАДКОВА О.М., доцент		
Нормоконтролер	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання “04” вересня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Ігор ЛИСЕНКО
(Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Ольга ГЛАДКОВА
(Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 107 с., 50 рис., 2 табл., 2 дод., 23 джерел.

ПЕРСОНАЛІЗОВАНІ РЕКОМЕНДАЦІЇ, МАШИННЕ НАВЧАННЯ, БАЙЄСІВСЬКЕ ПЕРСОНАЛІЗОВАНЕ РАНЖУВАННЯ, МАШИНИ ФАКТОРИЗАЦІЇ, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ.

Об'єкт дослідження – ранжування товарів у інтернет-магазині.

Предмет дослідження – методи та алгоритми ранжування товарів.

Мета роботи – розробка та впровадження алгоритму ранжування товарів для інтернет-магазину.

Матеріали, методи та технічні засоби: мова програмування JavaScript, об'єктно-орієнтоване програмування, фреймворк React.js, сховище даних Firebase, персональний комп'ютер з процесором M2 під управлінням MacOS.

Впроваджено гібридний алгоритм видачі персоналізованих рекомендацій, який використовує колаборативну та контентну фільтрації. Алгоритм здатний оцінювати послідовність дій користувача в інтернет-магазині, об'єктів взаємодії та подібні елементи до товарних позицій.

Галузь використання програмної розробки – електронна комерція.

ABSTRACT

Explanatory note to the master's thesis: 107 pages, 50 figures, 2 table, 2 appendices, 23 sources.

PERSONALIZED RECOMMENDATIONS, MACHINE LEARNING, BAYESIAN PERSONALIZED RANKING, FACTORIZATION MACHINES, INTELLIGENT DATA ANALYSIS.

The object of the study is the ranking of products in the online store.

The subject of the research is methods and algorithms for product ranking.

The purpose of the work is to develop and implement an algorithm for effective ranking goods for an online store.

Materials, methods and technical means: JavaScript programming language, object-oriented programming, React.js framework, Firebase data storage, personal computer with an M2 processor running MacOS.

A hybrid algorithm for issuing personalized recommendations has been developed, which uses collaborative and content filtering. The algorithm is capable of evaluating the sequence of user actions in the online store, interaction objects and similar elements to product items.

The field of use of software development is electronic commerce.

ЗМІСТ

Перелік скорочень та умовних позначок.....	7
Вступ.....	8
1 Аналіз предметної області.....	10
1.1 Актуальність персоналізації в інтернет-магазині.....	10
1.2 Формалізація математичної моделі рекомендаційної системи	13
1.3 Основні методи побудови персоналізованих рекомендацій	15
1.3.1 Метод колаборативної фільтрації	15
1.3.2 Метод контентної фільтрації	23
1.4 Огляд існуючих систем електронної комерції з персоналізованим ранжуванням товарів	26
1.4.1 Платформа Amazon.....	27
1.4.2 Інтернет-магазин Comfy	31
1.4.3 Інтернет-магазин Answer	32
1.5 Висновки за розділом 1	34
2 Вибір та обґрунтування матеріалів та методів розробки	35
2.1 Вибір методу персоналізації товарів в інтернет-магазині	35
2.2 Алгоритм роботи системи ранжування	40
2.3 Основні метрики оцінки системи ранжування	42
2.4 Висновки за розділом 2	44
3 Проектування архітектури системи	45
3.1 Моделювання бізнес-процесів в системі	45
3.2 Архітектура програмної системи	48
3.3 Основні вимоги до програмної системи	50
3.4 Висновки за розділом 3	50
4 Розробка програмної системи	51
4.1 Обґрунтування вибору засобів розробки.....	51
4.1.1 Мова програмування JavaScript.....	51
4.1.2 Фреймворк для розробки компонентів React.js.....	52

4.1.3 Сховище даних Firebase	53
4.2 Розробка основних компонентів системи.....	55
4.3 Висновки за розділом 4	69
5 Тестування програмної системи	70
5.1 Огляд функціональності програмної системи	70
5.2 Висновки за розділом 5	76
Висновки	77
Перелік джерел посилання	79
Додаток А Текст програми.....	82
Додаток Б Слайди презентації	101

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

B2A – Business-to-Administration;

B2B – Business-to-Business;

CTR – метрика, що показує частку числа дій до числа переглядів;

SKU – ідентифікатор товарної позиції;

IC – інформаційна система;

PC – рекомендаційна система;

СУБД – система управління базами даних.

ВСТУП

Сучасна постіндустріальна епоха відрізняється від своїх попередників всепроникними процесами глобалізації та цифровізації. Поява глобальних ринків не тільки підвищила рівень життя в усьому світі, але й спричинила безпрецедентний сплеск виробництва та споживання товарів і послуг. Швидкий розвиток інформаційних технологій ще більше прискорив процес глобалізації, дозволяючи підприємствам охоплювати клієнтів через географічні кордони. Поява таких онлайн-торгівельних платформ, як Amazon і eBay, у середині 90-х років революціонізувала роздрібну торгівлю, що призвело до створення електронних систем більшістю роздрібних компаній і виробників. Ці віртуальні платформи спростили процес купівлі, дозволяючи будь-кому придбати різноманітний асортимент продуктів лише за кілька кліків. Однак ця зміна загострила конкуренцію серед підприємств, спонукаючи їх зосередитися на дизайні вебсайтів і рекламних стратегіях, спрямованих на залучення потенційних відвідувачів і перетворення їх на клієнтів. Десятиліття досліджень у статистиці та машинному навчанні сприяли створенню структурованого інструменту, спрямованого на надання відповідної інформації користувачам, вимагаючи мінімальних вхідних даних. Цей інструмент зазвичай називають рекомендаційною системою або системою ранжування. Попит на системи рекомендацій є значним не лише в секторі роздрібною торгівлі, але вони також широко використовуються в потокових сервісах, таких як YouTube і соціальних мережах, таких як Instagram та Twitter.

Системи ранжування передбачають створення пропозицій щодо товарів, послуг або інших елементів для користувачів, використовуючи або переваги користувача, або переваги подібних користувачів щодо їхніх покупок. У зв'язку з широким розповсюдженням різних типів рекомендаційних систем товарів тема має значну актуальність. Створення спеціалізованої системи рекомендацій, призначеної для надання користувачам індивідуального та цікавого контенту, є важливим. Ця система генеруватиме

індивідуальні пропозиції шляхом аналізу вподобань користувачів з використанням алгоритмів ранжування, дозволяючи користувачам отримувати персоналізовані рекомендації, які відповідають їх інтересам.

Метою кваліфікаційної роботи магістра є розробка та впровадження ефективного алгоритму ранжування товарів для інтернет-магазину.

Об'єктом дослідження є системи ранжування товарів у інтернет-магазині.

Предметом дослідження є методи та алгоритми ранжування товарів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність персоналізації в інтернет-магазині

У сучасному світі спостерігається значне зростання кількості установ, які займаються збором та обробкою величезних обсягів даних. Ці дані можна описати як фундаментальні цінності або факти, систематично організовані в єдиній базі даних. Хоча багато людей можуть розглядати дані та інформацію як взаємозамінні терміни, насправді вони відрізняються за своєю суттю. Фактично інформація містить дані, які структуровані таким чином, щоб надати відповіді на запитання та вирішити проблеми. Інформаційна система, з іншого боку, відноситься до програмного забезпечення, яке допомагає впорядковувати та аналізувати дані. Торгівля є фундаментальним компонентом багатьох підприємств. У цьому контексті доцільно дослідити сферу електронної комерції, широко відомої як «електронна комерція», яка потребує впровадження інформаційних систем для забезпечення її успішної роботи. Інформаційні технології відіграють ключову роль у розвитку платформ електронної комерції, забезпечуючи безперебійний зв'язок між продавцями та клієнтами через мобільні та вебінтерфейси.

Крім того, ці платформи сприяють ефективному моніторингу та управлінню різними аспектами, такими як доставка, повернення та оплата придбаних товарів. Безсумнівно, світовий ринок електронної комерції характеризується надзвичайною динамічністю та послідовним зростанням. Прогнози та спостереження експертів підтверджують, що очікується, що обсяги електронної комерції перевищать традиційну роздрібну торгівлю як зараз, так і в майбутньому. Зауважимо, що динаміка показників електронної комерції в Україні, як показовий приклад, перевершує показники роздрібною торгівлі [1]. Таке стрімке зростання електронної комерції в усьому світі можна пояснити збільшенням кількості людей, для яких Інтернет став невід'ємною частиною повсякденного життя. У цьому ландшафті потреби людей, включаючи купівлю та продаж, поступово задовольняються. Варто

підкреслити, що поява електронної комерції створила нові можливості для компаній охопити та взаємодіяти з ширшою клієнтською базою, пропонуючи зручність і покращений досвід споживачів.

З безперервним прогресом інформаційних технологій і зростаючим визнанням онлайн-транзакцій значення та вплив електронної комерції на світовий ринок лише продовжуватимуть зростати [2].

Персоналізована система рекомендацій має важливе завдання вибору конкретних продуктів або предметів із заданого набору, які, швидше за все, зацікавлять користувача, виходячи з його вподобань та інтересів. Ці продукти можуть охоплювати широкий спектр категорій, включаючи предмети, які користувач має намір придбати, відео чи фільми, які вони хочуть переглянути, статті чи книги, які вони планують прочитати, музику чи подкасти, які їм сподобаються, і навіть вміст у соціальних мережах, який би викликав враження. Застосування систем рекомендацій також різниться в різних бізнес-моделях. Вони можуть слугувати основою продукту, як у випадку з TripAdvisor, щоб покращити загальну взаємодію з користувачем, як Netflix і Spotify, або навіть стимулювати додаткове просування продукту, як відомо в Amazon. В останні роки відбувся помітний сплеск інтересу до впливу систем рекомендацій на різні групи продуктів, що призвело до значної кількості наукових досліджень, присвячених дослідженню цієї тенденції. Ці аналізи дали інтригуючі висновки. Наприклад, ще в 2022 році команда YouTube досягла чудового збільшення рейтингу кліків для нішевих відео на 200% [3], впровадивши розширені алгоритми рекомендацій.

Подібним чином нещодавнє дослідження, проведене на краудсорсинговій платформі Amazon Mechanical Turk, продемонструвало, що запровадження системи рекомендацій призвело до помітного 20% зростання показників конверсії користувачів [4]. Крім того, згідно з комплексним дослідженням McKinsey, 35% усіх покупок на Amazon можна віднести до рекомендованих продуктів. Ці статистичні дані та дослідження проливають світло на суттєвий вплив, який системи рекомендацій можуть мати на

поведінку користувачів і ефективність бізнесу. Переваги систем рекомендацій для бізнесу абсолютно очевидні. Фактично, користувачі не тільки цінують персоналізовані рекомендації, але й очікують їх як стандартну функцію.

Згідно з недавнім дослідженням, проведеним компанією Smart Insights [5], 80% покупців заявили, що вони не будуть використовувати платформу, яка не має жодної форми персоналізації. Відсутність персоналізації перешкоджає потенційному прибутку, а також призводить до відтоку користувачів.

Ранжування товарів в інтернет-магазині передбачає створення ієрархічного розташування продуктів на основі певних впливових факторів. Як правило, процес сортування базується на цінових категоріях, де товари впорядковані в порядку зростання, починаючи з найнижчої ціни. Однак альтернативні методи сортування можуть бути реалізовані на основі різних параметрів. Наприклад, у сфері онлайн-продажу меблів предмети інтер'єру можна сортувати за розміром, а в сфері продажу електроніки – за класом енергоспоживання та схожими характеристиками. Ці додаткові параметри сортування пропонують відвідувачам додаткову зручність під час вибору продуктів, тим самим привертаючи більше уваги. Ранжування на основі ціни є найпростішим і найпоширенішим варіантом сортування. Коли використовується ця опція, на вебсайті створюється список продуктів, починаючи з мінімальної ціни вгору, хоча також можна вибрати порядок спадання. Це призводить до того, що найдорожчі продукти з'являються у верхній частині списку. На основі дослідження [6] визначено, що майже всі відвідувачі користуються цією функцією для створення списків у порядку зростання або спадання цін. Навіть покупці, які зайшли на сайт із певним продуктом, схильні дослідити схожі товари за різними цінами. Така поведінка є природною, оскільки загальновідомо, що така цікавість може стимулювати купівельну спроможність. Тому, якщо під час перегляду покупець натрапить на вигідний варіант, навряд чи упустить можливість зробити покупку. Для потенційних покупців, які все ще обмірковують свою покупку, послуга рейтингу продуктів виявляється неоціненною. Як правило, покупки

здійснюються на основі визначених фінансових обмежень, і навігація рейтинговим списком набагато простіша, ніж пошук у великому каталозі.

1.2 Формалізація математичної моделі рекомендаційної системи

Системи рекомендацій, як згадувалося раніше, мають широкий діапазон різновидів. Для більш чіткого розуміння проблеми, що розглядається, введемо кілька ключових понять. Позначимо множину всіх користувачів як U , а множину всіх товарів як I . Для певних пар користувачів (u) і елементів (i), де u належить до множини U , а i належить до множини I , задані оцінки r_{ui} . Ці рейтинги відображають рівень інтересу користувачів до відповідних продуктів. По суті, коли користувач (u) виявляє значний інтерес до продукту (i), присвоєний рейтинг (r) буде значно вищим за нуль. Щоб рекомендувати найбільш релевантні продукти для кожного користувача, потрібно створити колекцію товарів, позначену як $I_k(u)$, на основі попередньо визначених рейтингів r_{ui} . Мета полягає в тому, щоб вибрати продукти, які дають найбільшу суму r_{ui} , $i \in I(u)$ для певного користувача u . Процес створення рекомендацій можна описати таким чином: коли розглядається конкретна сутність під назвою «користувач-товар» $x_{ui} := (u, i)$, виникає необхідність передбачити значення цільової змінної $\hat{r}_{ui} \approx r_{ui}$, що вказує на рівень інтересу користувача до продукту. Потім ці прогнози використовуються для визначення k продуктів з найвищими оцінками $\hat{r}_{ui}$.

Список рекомендацій для користувача u визначено формулою (1.1):

$$I_k(u) = (\hat{r}_{ui_1}, \hat{r}_{ui_2}, \dots, \hat{r}_{ui_k}), \text{ де} \\ \hat{r}_{ui} = \text{Predict}(x_{ui}); \hat{r}_{ui_1} > \hat{r}_{ui_2} > \dots > \hat{r}_{ui_k} > \hat{r}_{ui_{k+1}} > \dots. \quad (1.1)$$

Існують також більш складні альтернативи, доступні для побудови рекомендацій, особливо при використанні операції перестановки на множині $I_k(u)$. Такий підхід дозволяє отримати більш швидкі та чіткі результати при

прогнозуванні рейтингів і полегшує диверсифікацію рекомендацій. Крім того, це дозволяє включати додаткові рекламні продукти в список рекомендацій. Однак у цьому дослідженні не розглядається аспект зміни порядку, оскільки він розроблений для задоволення конкретних бізнес-вимог, що робить його менш універсальним для узагальнення та дослідницьких цілей. Важливо вказати два ключових елементи в наведеному формулюванні: процес формулювання цільової змінної, а саме процентного рейтингу $\hat{r}_{ui}$, і алгоритм, прогнозування його значень, позначений як $Predict(x_{ui}) = \hat{r}_{ui} \approx \hat{r}_{ui}$.

Алгоритм визначення рейтингів потребує окремого вивчення, але, як правило, вони встановлюються на основі відгуків користувачів. У контексті рекомендації музики, подкастів, книг або відео, оцінки можна визначити за кількістю та тривалістю взаємодії з відповідним типом вмісту. У сфері соціальних мереж застосовуються додаткові інструменти, такі як вподобання, репости та огляди, які дозволяють користувачам взаємодіяти з вмістом у більш відчутний спосіб. Що стосується інтернет-магазинів, то інтерес часто вимірюється покупками і переглядами товару. Варто зазначити, що покупки мають більшу вагу, ніж прості перегляди. Фактори, які впливають на рейтинг зацікавленості, позначений як $\hat{r}_{ui}$, можна розділити на дві групи: явні та неявні. Явні фактори зазвичай охоплюють відгуки, які однозначно відображають настрій і інтерес користувача, такі як лайк чи репост у соціальній мережі, або акт здійснення покупки в інтернет-магазині. З іншого боку, неявні фактори вимагають певних припущень щодо поведінки користувача, щоб інтерпретувати відгук. Наприклад, час, витрачений на взаємодію з вмістом або перегляд продукту, не завжди може точно відображати справжній інтерес користувача. Можливо, користувач просто не в настрої для певного типу вмісту або що продукт випадково привернув його увагу, але не викликав справжнього інтересу. Однак, формулюючи припущення щодо моделей поведінки користувача (наприклад, припускаючи, що користувач постійно взаємодіє з вмістом, який вважає його цікавим, або дивиться лише на продукти, якими він справді зацікавлений), такий тип

зворотного зв'язку також можна використовувати. Це виявляється особливо цінним у випадках, коли інформація про явні фактори взаємодії обмежена.

Як правило, під час оцінки рейтингу зацікавленості $\hat{r}_{ui}$ за допомогою евристичних методів досліджується ряд факторів. Потім ці фактори включаються в остаточну оцінку шляхом присвоєння їм ваги та обчислення зваженої суми. Важливо відзначити, що неявні фактори отримують набагато менші ваги порівняно з явними. Цей підхід гарантує, що на формування остаточного рішення щодо рекомендацій продукту не впливають потенційні неточності. Завдяки присвоєнню менших ваг неявним факторам пом'якшуються потенційні упередження, спричинені припущеннями, що забезпечує більш збалансований і точний процес рекомендацій товарів.

1.3 Основні методи побудови персоналізованих рекомендацій

В цьому підпункті будуть розглянуті методи побудови персоналізованих рекомендацій: колаборативної фільтрації та контентної фільтрації.

1.3.1 Метод колаборативної фільтрації

Алгоритми колаборативної (спільної) фільтрації є найчастіше використовуваними в системах надання рекомендацій. Ці алгоритми належать до сімейства алгоритмів машинного навчання, які ґрунтуються на аналізі поведінки та переваг великої кількості користувачів щодо набору вибору або опцій, таких як продукти, які пропонує компанія, або вміст у додатку [8]. Взаємодія користувачів із цими опціями збирається та відображається в рейтинговому списку. Щоб вирішити будь-які відсутні оцінки та надати користувачам більш широкий спектр відповідних альтернатив, використовуються методи наближення. Один із поширених методів відомий як матрична факторізація, яка передбачає розбиття матриці «користувач-

продукт» на матрицю прихованих функцій користувача та альтернативних елементів. Перемноживши ці матриці, можна отримати наближення вихідної матриці. Серед відомих алгоритмів факторизації виділяються метод послідовних найменших квадратів і метод байєсівського персоналізованого ранжування. Останній метод фактично є модифікацією першого з додаванням того, що параметри двоїстої проблеми розглядаються як випадкові величини з певним розподілом. Спільна фільтрація відноситься до категорії алгоритмів, які виходять за рамки простого розгляду вмісту, рекомендованого користувачеві. Натомість вони враховують взаємодію між усіма користувачами та всіма елементами, такими як продукти чи послуги.

Для полегшення цього процесу вводиться рейтингова матриця:

- матрицю рейтингів $R = (r_{ij})_{i=\overline{1,N}, j=\overline{1,N}}$;
- матрицю ідентифікаторів товарів $I = \{i_1, \dots, i_M\}$;
- матрицю ідентифікаторів користувачів $U = \{u_1, \dots, u_N\}$.

Колаборативна фільтрація (КФ) – це техніка, метою якої є оцінка значення рейтингу $\hat{r}_{ij}$, коли немає прямої взаємодії між користувачем u_i та елементом i_j , шляхом використання подібності між u_i та іншими користувачами $u_k \ \forall k = \overline{1,N}, k \neq i$. Розглянемо сценарій за участю двох користувачів, які позначені u і v . У цьому контексті I_{uv} – представляє колекцію товарів або предметів i , де оцінки надані обома користувачами (1.2):

$$I_{uv} = \{i \in I \mid \exists r_{ui} \ \& \ \exists r_{vi}\}. \quad (1.2)$$

Міру подібності між двома користувачами, позначену як w_{vu} , можна визначити шляхом обчислення коефіцієнта кореляції Пірсона (1.3):

$$w_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \bar{r}_v)^2}}, \quad (1.3)$$

де $\bar{r}_u$ та $\bar{r}_v$ – усереднені рейтинги користувачів по множині I_{uv} .

Подібним чином, розглядаючи два товари, позначені як i та j , можливо ввести групу користувачів U_{ij} , для яких уже відомі рейтинги продуктів (1.4):

$$U_{uv} = \{u \in U \mid \exists r_{ui} \ \& \ \exists r_{uj}\} \quad (1.4)$$

Міру подібності між двома товарами, позначену як w_{vu} , можна визначити шляхом обчислення коефіцієнта кореляції Пірсона (1.5):

$$w_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_i)(r_{vj} - \bar{r}_j)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{uj} - \bar{r}_j)^2}}. \quad (1.5)$$

Подібність можна виміряти за допомогою інших засобів, ніж кореляція. Однією з альтернатив є обчислення скалярних добутків векторів рейтингу, призначених двом різним продуктам. Після визначення подібності між продуктами та користувачами можна розглянути різні підходи, щоб визначити, які продукти слід рекомендувати конкретному користувачеві, позначеному як користувач u . У підході на основі схожості користувачів (1.6) група користувачів, позначена як $U(u_0)$, визначена як подібна до u :

$$U(u_0) = \{v \in U \mid w_{u_0v} > \alpha\}. \quad (1.6)$$

Після цього етапу виконується розрахунок (1.7) для визначення частоти, з якою кожен продукт купується користувачами з ідентифікатором $U(u_0)$:

$$p_i = \frac{|\{u \in U(u_0) \mid \exists r_{ui}\}|}{|U(u_0)|}. \quad (1.7)$$

На наступному кроці система пропонує користувачеві вибрати k товарів, які мають найвищу цінність p_i . Однак у цього алгоритму є суттєвий недолік, який виникає, коли є користувачі з унікальними характеристиками та перевагами. У таких випадках стає складно визначити схожих користувачів, від яких можна отримати рекомендації. Отже, ці нетипові користувачі можуть не отримати жодних рекомендацій щодо продуктів.

В якості альтернативи можна прийняти підхід, який базується на подібності самих товарів. У цьому підході є колекція $I(u_0)$ продуктів, які схожі на ті, що зацікавили конкретного користувача системи (1.8):

$$I(u_0) = \{i \in I \mid \exists r_{u_0 i_0} w_{i_0 i} > \alpha\}. \quad (1.8)$$

Далі, визначається та оцінюється схожість між кожним продуктом із цього конкретного набору та користувачем (1.9):

$$p_i = \max_{i_0: \exists r_{u_0 i_0}} w_{i_0 i} \quad (1.9)$$

На наступному кроці система надає користувачеві рекомендацію k продуктів, які мають найвище значення p_i . Цей метод ефективно вирішує проблему з нетиповими користувачами, але має деякі недоліки. Одним із таких недоліків є вимога більшої кількості продуктів для кожного користувача, щоб отримати рекомендації високої якості. Ці алгоритми, відомі як засновані на пам'яті, спираються на підтримку матриці відомих рейтингів у пам'яті комп'ютера, що може стати громіздким, якщо матриця надто велика, що призводить до зниження продуктивності та обмежень масштабованості. Однак для невеликих систем цей підхід може виявитися задовільним і відносно простим рішенням. Важливо зазначити, що у вищезгаданих алгоритмах $\hat{r}_{u_0 i} = p_i$. Можливі інші варіанти, як-от обчислення $\hat{r}_{u_0 i}$ як зваженої суми різних значень p_i , зазвичай також використовують p_i як один із факторів в іншому

алгоритмі прогнозування рейтингу, наприклад $\hat{r}_{u_0 i_0} = \text{Predict}(x_{u_0 i_0}, p_i, \dots)$. Цей варіант часто використовується в сучасних і великомасштабних системах.

Сучасні підходи до колаборативної фільтрації покладаються на подібність оцінок оцінок. Однак існують альтернативні методи, які пропонують точнішу інтерпретацію вподобань користувача. Ці методи мають здатність розкривати шаблони в прихованих просторах, тобто за допомогою лише доступної матриці можна отримати векторне представлення як для користувача, так і для об'єкта. Це вводить концепцію факторизації матриці, яка, по суті, є процесом розкладання матриці на її складові частини.

В цих методах використовується наступні вхідні дані:

- матрицю рейтингів $R = (r_{ij})_{i=\overline{1,N}, j=\overline{1,N}}$;
- множина ідентифікаторів товарів $I = \{i_1, \dots, i_M\}$;
- множина ідентифікаторів користувачів $U = \{u_1, \dots, u_N\}$;
- розмірність вкладеного простору $l \in \mathbb{N}$.

Для досягнення результату необхідним стає пошук конкретних матриць. Ці матриці, позначені як $W = (w_{ij})_{i=\overline{1,N}, j=\overline{1,l}}$ і $V = (v_{ij})_{i=\overline{1,M}, j=\overline{1,l}}$ мають значний вплив на виконання вимоги $WV^T = R$. Однак важливо визнати, що ця проблема за своєю суттю має недоліки через існування кількох розв'язків і погану обумовленість розрідженої неквадратної матриці R . Таким чином, щоб вирішити цю проблему, рівність замінено умовою, яка має на меті мінімізувати відхилення між добутком WV^T і матрицею R . У цьому випадку відхилення вимірюється за допомогою, наприклад, евклідової відстані. Факторизаційні машини – це моделі, розроблені для вирішення цих проблем. Хоча ці алгоритми є відносно новими в галузі, вони здобули значну популярність завдяки своїм унікальним можливостям. Однією з ключових переваг є їхня здатність сприяти швидкому перенавчанню, що важливо в динамічному середовищі. Крім того, ці моделі використовують методи матричної факторизації, які можна виконувати одночасно на кількох машинах у багатопотоковий спосіб. Ця можливість паралельної обробки забезпечує

швидке навчання навіть при роботі з надзвичайно великими наборами даних. Варто зазначити, що ці алгоритми реалізовані на платформі Apache Spark, відомої своїми ефективними можливостями обробки великих даних [9].

Серед різноманітних алгоритмів матричної факторизації альтернативний виділяється алгоритм найменших квадратів, який можна представити формулою (1.10):

$$\min_{W,V} \sum_{(i,j): r_{ij} > 0} (r_{ij} - w_i v_j^T)^2. \quad (1.10)$$

Ітеративна оптимізація передбачає покроковий процес фіксації матриць W і V для підвищення продуктивності.

Цей підхід починається з фіксації однієї матриці, такої як V , а потім розв'язується задача опуклої оптимізації (1.11):

$$\min_V \sum_{(i,j): r_{ij} > 0} (r_{ij} - w_i v_j^T)^2. \quad (1.11)$$

Аналітичний розв'язок задачі вище може бути знайдено через ідентифікацію критичних точок (1.12):

$$w_i^* = \left(\sum_{(i,j): r_{ij} > 0} v_j^T v_j \right)^{-1} \sum_{(i,j): r_{ij} > 0} r_{ij} v_j^T. \quad (1.12)$$

Транспонування дозволяє визначити значення векторів v_j^* (1.13):

$$v_j^* = \left(\sum_{(i,j): r_{ij} > 0} u_i^T u_i \right)^{-1} \sum_{(i,j): r_{ij} > 0} r_{ij} u_i. \quad (1.13)$$

Вищенаведені кроки повторюються до досягнення умови сходження:

$$\Delta = \left\| w_{i_{p+1}}^* - w_{i_p}^* \right\| < \epsilon. \quad (1.14)$$

Метод ALS пропонує кілька переваг, однією з яких є можливість виконувати паралельні обчислення для операцій із зворотною матрицею та векторних скалярних добутків за допомогою інфраструктур, розроблених для обробки великих наборів даних, таких як Spark. Це дозволяє ефективно використовувати декілька машин у процесі обчислень.

Крім того, ще одним дуже ефективним алгоритмом для апроксимації рейтингу є система персоналізованого ранжування Байєса. У порівнянні з традиційними задачами матричної факторизації, цей алгоритм використовує байєсівський підхід для оцінки ймовірностей конкретних подій.

Основною метою персоналізованого ранжування Байєса є встановлення комплексного, лінійного та чітко визначеного зв'язку для кожного користувача та певного набору товарів. Це відношення, яке називають ранжування і позначають $\succ_u$, характеризується як антисиметричним, так і транзитивним. Щоб полегшити цей процес, вводяться дійсні матриці, включаючи $W = (w_{ij})_{i=\overline{1,N}, j=\overline{1,l}}$ та $V = (v_{ij})_{i=\overline{1,M}, j=\overline{1,l}}$, а також об'єднаний ваговий вектор $\theta = [W, V]$. Розглянемо модель, яка вирішує цю задачу (1.15):

$$\max_{\theta, u \in U} p(\theta \mid \succ_u). \quad (1.15)$$

Після застосування формули Байєса до задачі вище отримуємо (1.16):

$$p(\theta \mid \succ_u)p(\succ_u) = p(\succ_u \mid \theta)p(\theta) \Leftrightarrow p(\theta \mid \succ_u) \propto p(\succ_u \mid \theta)p(\theta) \quad (1.16)$$

Результат цього взаємозв'язку дозволяє оцінити ймовірність досягнення точного ранжування для конкретного вектора θ . Також, можна вибрати багатовимірний розподіл незалежних нормальних значень із середнім нульовим і постійним середнім квадратичним відхиленням λ .

В результаті, отримуємо наступну задачу максимізації (1.17):

$$\max_{\theta, u \in U} p(\succ_u \mid \theta) p(\theta), \theta \sim N(0, \lambda I). \quad (1.17)$$

Задачу вище можна вирішити, використовуючи метод, відомий як максимальна ймовірність. Щоб спростити аргумент, позначений виразом $\prod_{u \in U} p(\succ_u \mid \theta)$, можна звернутися до ранжування та включити обмеження, пов'язані з відношенням переваги, при призначенні ранжування товарам. Це обмеження $\forall j, k \in \succ_u \quad i_j \succ_u i_k \Leftrightarrow r_{ij} > r_{ik}$, яке означає наступне (1.18):

$$p(\succ_u \mid \theta) = \eta(r_{ij} - r_{ik}). \quad (1.18)$$

Алгоритми чисельної оптимізації значною мірою залежать від плавності функції, до якої вони застосовуються. Однак функція Хевісайда η не має такої властивості гладкості, що робить її непридатною для таких алгоритмів. Є спосіб апроксимації функції за допомогою сигмоїдної функції (1.19):

$$p(\succ_u \mid \theta) = \sigma(r_{ij} - r_{ik}) = \frac{1}{1 + e^{-(r_{ij} - r_{ik})}}; p(\theta) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}\theta^2}. \quad (1.19)$$

Відповідно до наведених вище рівнянь, застосовуючи логарифми та вилучаючи константи, можна підставити вирази вище у рівняння (1.20):

$$\max_{\theta} \sum_{i \in I(U), (j,k) \in \succ_u} \ln(\sigma(r_{ij} - r_{ik})) - \frac{1}{2\sigma^2} \|\theta\|^2. \quad (1.20)$$

Застосовуючи градієнтні методи, можна вирішити цю задачу. Один із підходів полягає у встановленні зв'язку від r_{ij} до 0 (1.21):

$$\frac{\partial L}{\partial \theta} = \sum_{i \in I(U), (j,k) \in \mathcal{R}_u} \frac{e^{-(r_{ij} - r_{ik})}}{1 + e^{-(r_{ij} - r_{ik})}} \frac{\delta}{\delta \theta} (r_{ij} - r_{ik}) - \sigma^2 \theta. \quad (1.21)$$

Якщо застосувати визначення θ , то можна виділити три випадки (1.22):

$$\frac{\delta}{\delta \theta} (r_{ij} - r_{ik}) = \begin{cases} (v_i - v_k), & \text{якщо } \theta = w_i \\ w_i, & \text{якщо } \theta = v_i \\ -w_i, & \text{якщо } \theta = v_k \end{cases} \quad (1.22)$$

Таким чином, була встановлена задача байєсівського персоналізованого ранжування. У дослідженні [14] надається ретельний аналіз алгоритму.

1.3.2 Метод контентної фільтрації

Фільтрація на основі вмісту спирається насамперед на інформацію про елементи всередині системи, а не на дані користувача. Цей підхід передбачає формування рекомендацій шляхом виявлення подібних елементів до тих, які користувач оцінював у минулому. Для цього створюються як профілі товарів (що представляють товари магазину), так і профілі користувачів. Використовуючи дані про елементи системи, стає можливим встановити релевантність певного елемента системи для конкретного користувача.

Щоб описати елементи системи та створити їхні профілі, система рекомендацій пов'язує кожен елемент із певним набором ключових слів, таких як назва продукту та відповідні категорії, до яких він належить.

Це дозволяє системі працювати ефективно та точно підбирати певні товари для користувачів.

Системи рекомендацій, які використовують фільтрацію на основі вмісту, покладаються на функцію $h(u, s)$, щоб зіставити користувача u з елементом системи s . Ця функція враховує як минулі оцінки користувача щодо інших елементів у системі $s' \in S$, так і схожість цих елементів з певним елементом s . Щоб надати рекомендації щодо нових товарів, система спочатку аналізує товари, оцінені користувачем раніше.

Потім вона визначає спільні риси серед цих елементів і використовує їх для пошуку подібних елементів, які користувач ще не оцінив. Це гарантує, що рекомендовані елементи є новими та адаптованими до вподобань користувача.

На рисунку 1.1 представлено алгоритм контентної фільтрації товарів.

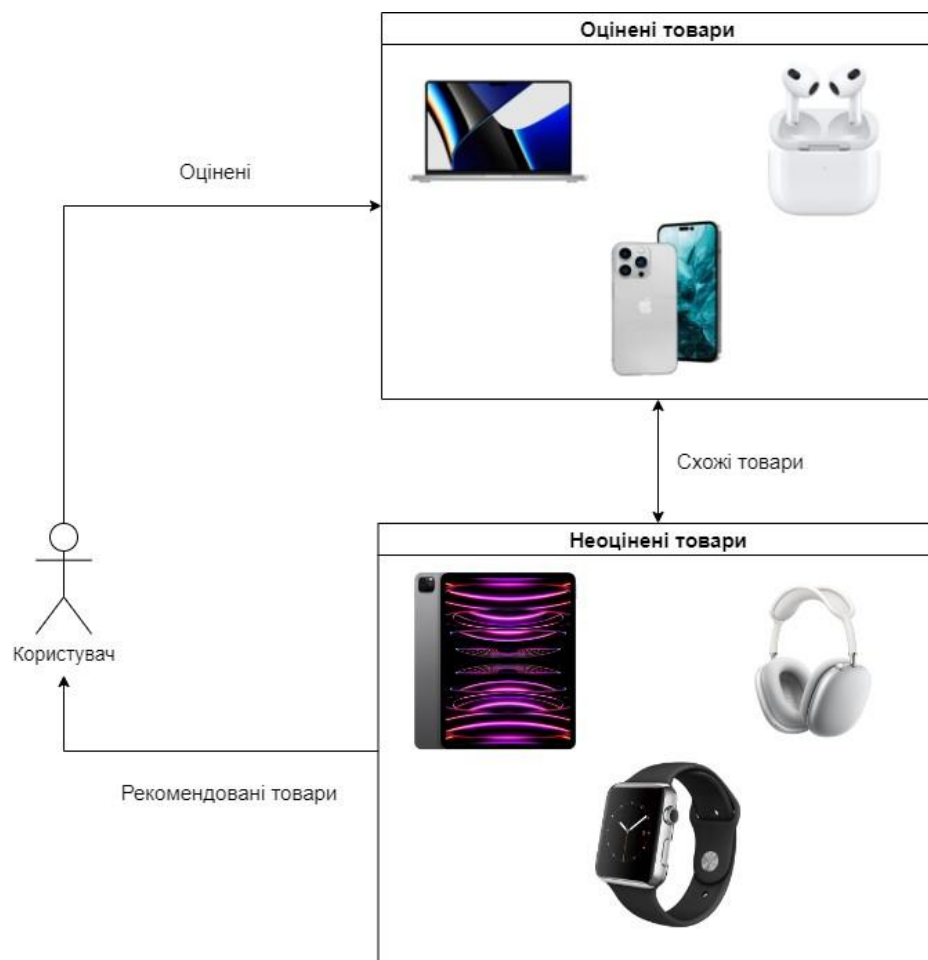


Рисунок 1.1 – Алгоритм контентної фільтрації

Концепцію можна спростити таким чином: якщо користувач, назовемо його u , позитивно оцінює товар під назвою « A », вказуючи на те, що він йому сподобався, і якщо інший товар під назвою « B » має схожі характеристики з товаром A , то можна припустити, що товар « B » матиме високу ймовірність сподобатися користувачу u . Це припущення формує основу профілю користувача, який складається з параметрів, призначених кожному елементу в межах набору під назвою S' . Цей набір охоплює всі товари, які були оцінені користувачем. Як правило, ці параметри представлені ключовими словами, яким присвоєно певні вагові коефіцієнти для кожного товару в системі.

Одним із поширених методів визначення цих коефіцієнтів є TF-IDF. Цей показник розраховується за наступною формулою (1.23):

$$TF_{ij} = \frac{f_{ij}}{\max_z f_{zj}}, \quad (1.23)$$

де f_{ij} представляє частоту певного ключового слова k_i , що зустрічається в товарі d_j . Одночасно ключове слово k_j міститься в n_i товарів в системі.

При використанні формули вище враховується виключно частота ключових слів. Такий підхід може призвести до сценарію, коли ключові слова, які найчастіше зустрічаються, матимуть найвищу вагу, що призведе до неточних прогнозів щодо уподобань конкретного користувача.

Щоб запобігти цьому, вводиться концепція $IDF(i_j)$. $IDF(i_j)$ являє собою зворотну частоту, з якою ключове слово з'являється в товарі магазину (1.24):

$$IDF(i_j) = \log \frac{N}{df(i_j)}, \quad (1.24)$$

де N представляє кількість елементів, які можуть бути рекомендовані.

Використовуючи формули 1.23 та 1.24, можливо обчислити коефіцієнт w_{ij} для ключового слова k_i в елементі d_j за допомогою рівняння (1.25):

$$w_{ij} = TF_{ij} * IDF(i_j). \quad (1.25)$$

Отже, профіль товару в інтернет-магазині можна задати таким чином:

$$Content(d_{ij}) = (w_{ij}, \dots, w_{kj}). \quad (1.26)$$

Таким чином, існує гіпотеза, що нові елементи також будуть добре оцінені користувачем. Набір оцінених користувачем елементів системи формує профіль користувача, а також ваговий вектор ключових слів (1.27):

$$(w_{ul}, \dots, w_{uk}), \quad (1.27)$$

де w_{ui} визначає важливість ключового слова k_i для користувача u .

Функцію зіставлення $h(u, s)$ можна обчислити як косинус кута, який утворюється між векторами $\vec{w}_s$ та $\vec{w}_u$ (1.28):

$$h(u, s) = \cos(\vec{w}_s, \vec{w}_u) = \frac{\vec{w}_u \vec{w}_s}{\|\vec{w}_u\| \|\vec{w}_s\|} = \frac{\sum_{i=1}^K w_{iu} w_{is}}{\sqrt{\sum_{i=1}^K w_{iu}^2} \sqrt{\sum_{i=1}^K w_{is}^2}}, \quad (1.28)$$

де K – загальна кількість ключових слів товарів в інтернет-магазині.

1.4 Огляд існуючих систем електронної комерції з персоналізованим ранжуванням товарів

Для проведення порівняльного аналізу систем електронної комерції з персоналізованим ранжування товарів було обрано наступні: глобальна маркетплейс система Amazon, а також інтернет-магазини Цитрус та Answear.

1.4.1 Платформа Amazon

Пошукова система Amazon, відома як A9, розроблена Amazon.com Inc та є незалежною пошуковою системою в Інтернеті. Нині Amazon постійно вдосконалює та покращує A9, щоб надати своїй величезній клієнтській базі персоналізований досвід. Кінцева мета полягає в тому, щоб клієнти знайшли саме те, що їм потрібно. У той час як кожен продавець на Amazon прагне досягти найвищої позиції на сторінці результатів пошуку, лише кільком вдається досягти бажаних перших місць. Алгоритм A9, який відповідає за ранжування продуктів, ретельно визначає, які елементи заслуговують на помітне розміщення у верхній частині сторінки. Враховуючи кількість математичних змінних, алгоритм A9 є дуже складним, і Amazon постійно інвестує в його вдосконалення, щоб досягти оптимальних результатів.

Систему пошуку товарів Amazon можна розбити на два основні етапи: зіставлення та ранжування. Ці етапи працюють разом, щоб визначити продукти, які тісно відповідають пошуковому запиту користувача, і згодом упорядкувати їх у порядку релевантності.

Процес роботи алгоритму відбувається наступним чином [12]:

- спочатку користувач вводить свій запит у рядок пошуку, вказуючи бажаний продукт або ключові слова;
- потім запит переходить до навігації через механізм відповідності, який ретельно витягує всі продукти, які містять слова або елементи з пошукового запиту, ця колекція продуктів відома як набором відповідності;
- набір відповідності потім проходить ряд етапів рейтингу, на цьому етапі застосовується функція машинного навчання, яка сортує та оцінює продукти на основі їх приналежності для запиту користувача;
- пошукова система відображає найбільш релевантні результати товарів для користувача, які точно відповідають його пошуковому запиту.

Процес створення переліку товарів за допомогою механізму відповідності передбачає три різні методи перераховані нижче[12, 14].

По-перше, основними кандидатами є товари, які безпосередньо відповідають пошуковому запиту, тобто вони містять точні слова чи фрази, які шукали, наприклад «бездротова ігрова миша». Однак покладатися виключно на лексичну відповідність може бути проблематично через відсутність урахування семантики. У той час як лексичний пошук зосереджений на пошуку точних збігів у базі даних, семантичний пошук виходить за рамки цього, аналізуючи значення слів і фраз.

По-друге, семантика відіграє вирішальну роль у вирішенні різноманітних проблем, пов'язаних із досвідом пошуку. Алгоритм Amazon A9 використовує кілька семантичних елементів. Це включає в себе гіперніми, які є словами, що охоплюють ширші категорії, як-от взуття, яке включає кросівки, черевики тощо. Додатково розглядаються слова-антоніми, які мають протилежне значення. Використовується також формування коріння, процес утворення кореня слів із відмінюваних форм.

Крім того, 10-15% запитів на Amazon написані з орфографічними помилками, тому алгоритм також враховує ці помилки при роботі з даними.

По-третьє, поведінкова аналітика клієнтів відносно списку товарів служить суттєвим індикатором того, чи відповідні продукти відповідають відповідним пошуковим запитам. Коли алгоритм визначає послідовну модель клієнтів, які шукають такі терміни, як «адаптер типу C», і спостерігає, що певний продукт генерує найбільшу кількість конверсій, це підтверджує успішний збіг. Оцінюючи купівельну поведінку, слід враховувати чотири ключові сигнали. По-перше, коли клієнти натискають продукт на основі свого пошукового запиту, це вказує на їхній інтерес. По-друге, додавання продукту в кошик означає сильніший намір зробити покупку. Нарешті, здійснення покупки є найпереконливішим сигналом наміру. З іншого боку, коли клієнти переглядають продукт, не взаємодіючи з ним, це вважається негативним сигналом. Однак, навіть маючи поведінкові дані, алгоритм стикається з

певними проблемами. Наприклад, продукти, які відображаються на першій сторінці, мають вищу ймовірність того, що вони будуть переглянуті, порівняно з продуктами на третій сторінці. Якщо продукт залишається невидимим для клієнтів, він не отримає кліків, що знижує конверсію товарів.

На етапі ранжирування алгоритм аналізує поведінкові дані за допомогою різних моделей для оптимізації взаємодії з клієнтами. Через те, що клієнти не завжди явно висловлюють бажаний продукт, алгоритм A9 навчений розшифровувати наміри на основі активності клієнта та аналізу аудиторії. Діяльність стосується того, як продукт буде використовуватися, тоді як аудиторія вказує на осіб, які ним будуть користуватися. Враховуючи обидва ці фактори, можливо значно підвищити шанси на те, що товари займуть високі позиції в релевантних пошукових запитах. Наприклад, якщо клієнт шукає "чоловічі кросівки", алгоритм визначить продукти, які відповідають заходам, пов'язаним з бігом, таким як CrossFit або біг на відкритому повітрі, а також задовольняють цільову аудиторію чоловіків і спортсменів.

Таким чином, включення ключових слів, орієнтованих на діяльність і орієнтованих на користувача, у списки товарів може значно підвищити видимість. Крім того, Amazon отримує інформацію про ці атрибути, аналізуючи відгуки користувачів, які зазвичай не індексуються.

Наприклад, якщо у списку продуктів відсутні такі ключові слова, як CrossFit, але в численних оглядах продукту згадується CrossFit, ймовірність отримання інформації про пошукові запити, пов'язані з CrossFit, зростає.

Щоб створити контекстно-залежну модель, алгоритм враховує як довгострокові, так і короткострокові інтереси. Довгострокові інтереси визначаються на основі ефективності історії покупок під час персоналізованого пошуку продукту, тоді як короткострокові інтереси отримують шляхом аналізу кліків у сеансах запитів на кількох сторінках результатів. Розглянемо приклад розкішної сумки високого класу. Незважаючи на отримання більшої кількості кліків через те, що клієнти

бажають переглянути більше зображень, ця сумка може мати нижчий коефіцієнт конверсії порівняно з дешевшими альтернативами.

Однак алгоритм розуміє контекстну інформацію про те, що розкішна сумка високого класу, як очікується, матиме нижчий коефіцієнт конверсії порівняно з іншими сумками. Отже, він бере до уваги поведінку користувачів і може не розглядати низький коефіцієнт конверсії як сильний негативний сигнал. Інший сценарій виникає, коли клієнт демонструє переваги бренду через свою історію пошуку чи покупок. У таких випадках алгоритм, швидше за все, відобразить додаткові результати від цього конкретного бренду.

На рисунках 1.2 – 1.3 представлений приклад формування рекомендацій на платформі Amazon за запитом “аксесуари для Iphone”.

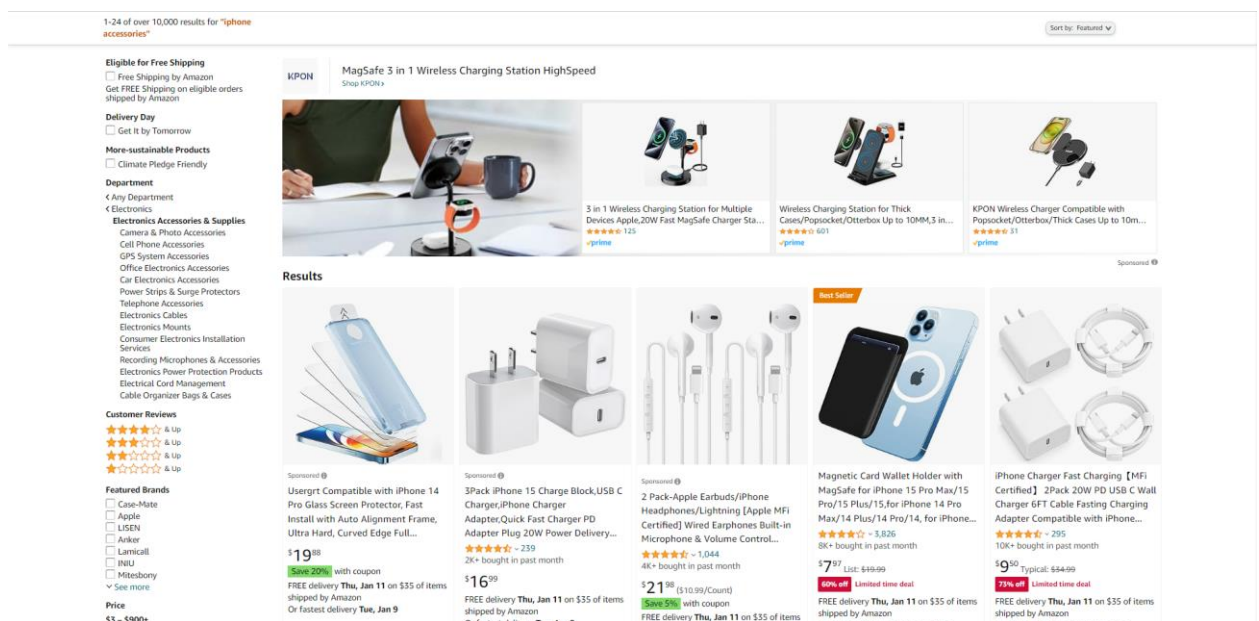


Рисунок 1.2 – Приклад формування рекомендацій на основі введеного пошукового запиту на платформі Amazon

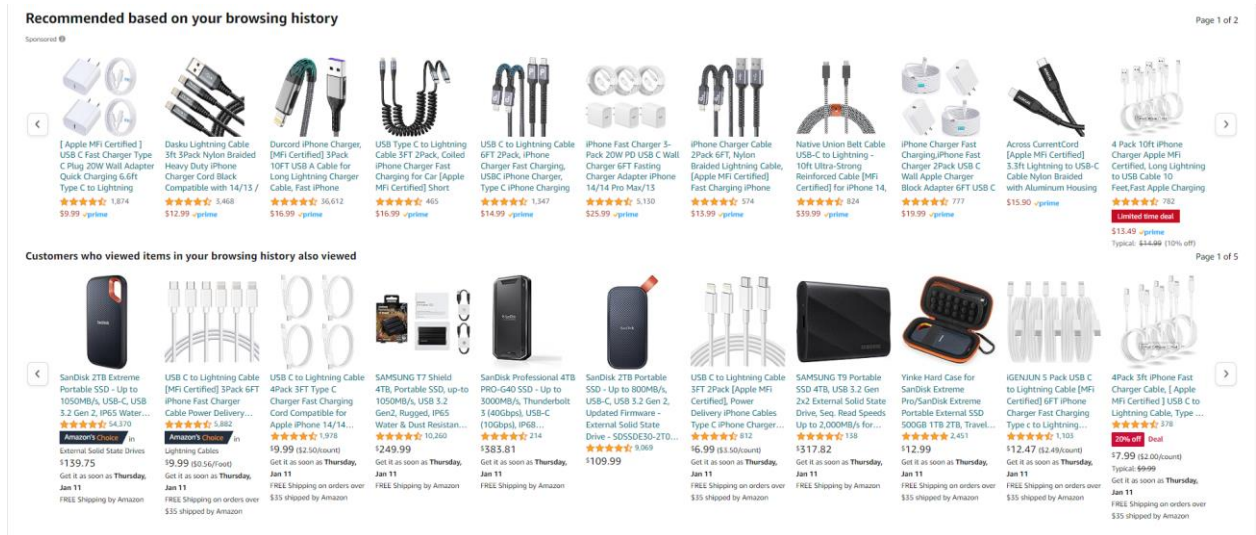


Рисунок 1.3 – Приклад формування рекомендацій на основі введеного пошукового запиту на платформі Amazon (продовження)

1.4.2 Інтернет-магазин Comfy

Інтернет-магазин "Comfy" – це сучасний онлайн-рітейлер, який пропонує широкий асортимент товарів для дому, електроніки, побутової техніки, а також товарів для відпочинку і хобі. Система рекомендацій в "Comfy" заснована на сучасних алгоритмах машинного навчання та аналізу даних. Вона враховує попередні покупки користувача, його перегляди та вподобання, а також товари, які часто купують разом. Ця система також аналізує загальні тенденції покупок та відгуки інших покупців, щоб надати найбільш точні та персоналізовані рекомендації. Також система рекомендацій враховує сезонність та актуальні тренди ринку, регулярно оновлюючи свої пропозиції залежно від пори року, святкових періодів або спеціальних акцій. Це означає, що під час осіннього сезону система може пропонувати товари для затишного дому, а в літній період – аксесуари для відпочинку на природі.

На рисунках 1.4 – 1.5 представлений пошук товарів за пошуковим запитом “ноутбук Apple” та надання рекомендацій на основі вподобань.

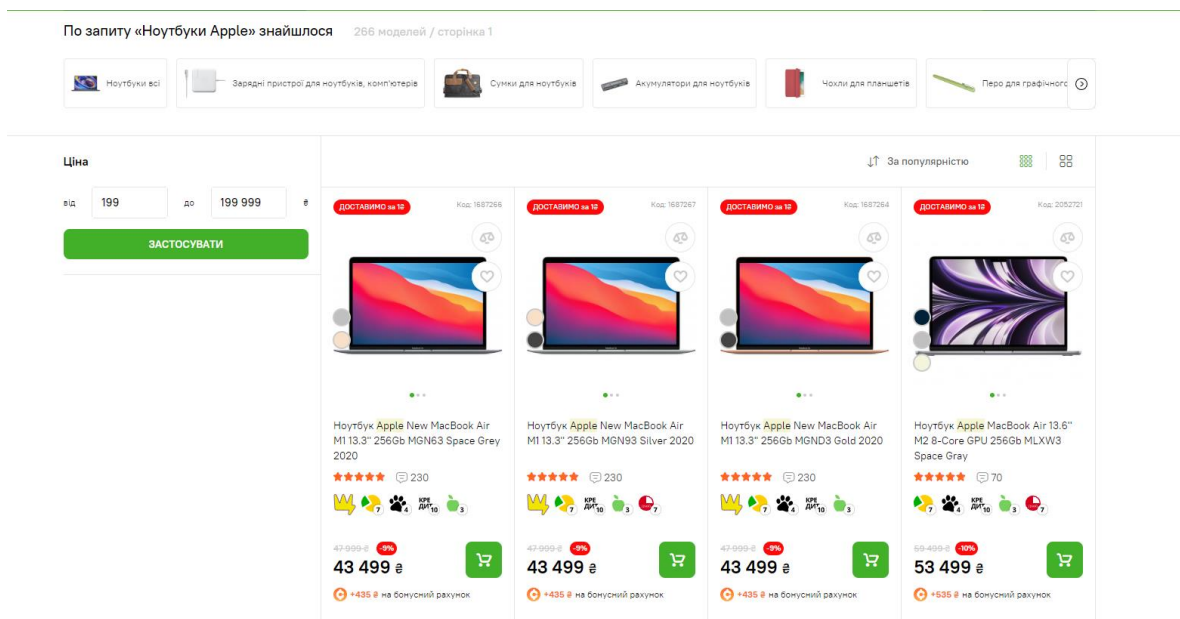


Рисунок 1.4 – Результат пошуку товарів за запитом “Ноутбуки Apple”

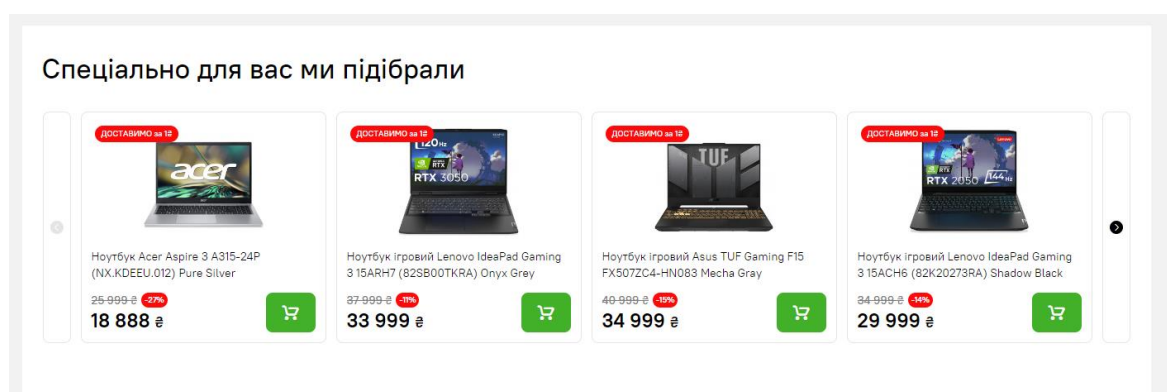


Рисунок 1.5 – Результат надання рекомендацій для користувача

На основі наведених персоналізованих рекомендацій на рисунку 1.5, можна припустити, що система використовує форму персоналізації, яка може включати методи контентної фільтрації для вибору товарів, які ймовірно відповідають інтересам користувача. Ноутбуки цієї секції вибрані на основі попередньої поведінки на сайті, а саме пошуку ноутбуків іншого виробника.

1.4.3 Інтернет-магазин Answer

Answer представляє собою високопрофільний онлайн-рітейлер, спеціалізований на комерційному розповсюдженні модного одягу та

аксесуарів. Ця електронна торгова платформа забезпечує широкий спектр товарів, орієнтований на різні демографічні групи, включаючи асортимент для чоловіків, жінок та дітей. Значним аспектом Answer є його інтеграція широкої палітри брендів, як відомих глобальних марок, так і нішевих лейблів, що дозволяє задовольнити різноманітні стилістичні та якісні запити споживачів.

Вебінтерфейс Answer розроблений із зосередженням на максимальній функціональності та зручності користувачів, пропонуючи продумані механізми фільтрації, які дозволяють ефективно навігувати через різноманітні категорії товарів. Система управління контентом сайту оптимізована для забезпечення швидкого доступу до новинок та ексклюзивних колекцій. Для забезпечення високої задоволеності клієнтів, Answer впроваджує комплексну стратегію клієнтського сервісу, включаючи ефективну логістику доставки, гнучкі опції оплати та зручну політику повернення. Акцент на динамічну взаємодію з клієнтами та постійне оновлення асортименту робить Answer привабливою точкою дестинації для покупок у цифровому просторі.

На рисунках 1.6 – 1.7 представлений результат пошуку товару за запитом “Спортивний костюм Under Armour” та відповідні рекомендації.

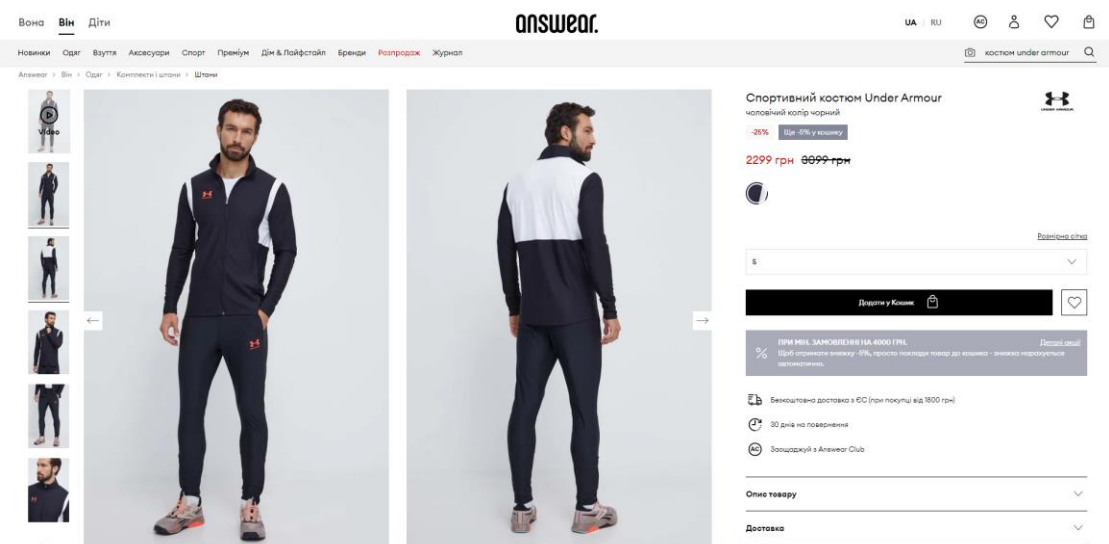


Рисунок 1.6 – Результат пошуку товару за запитом “Спортивний костюм Under Armour”

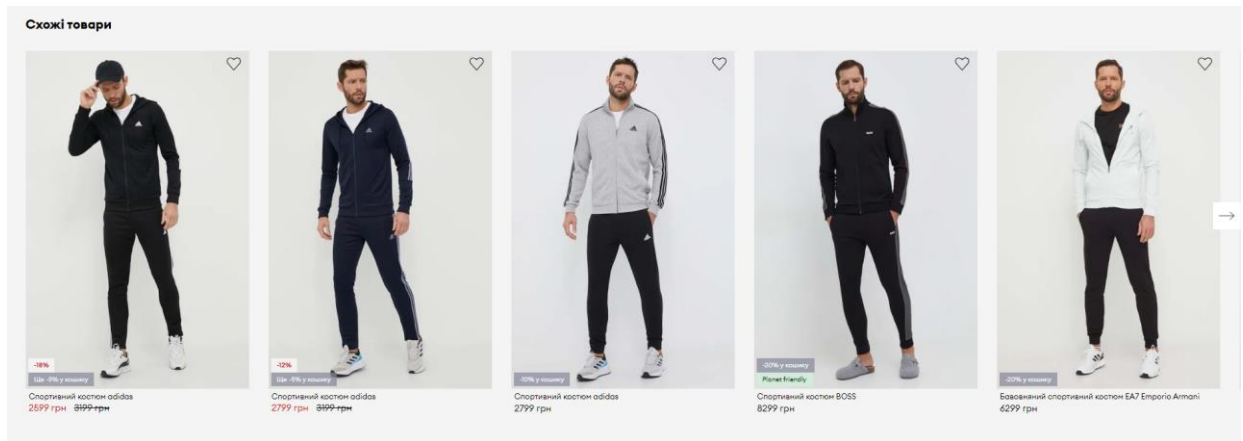


Рисунок 1.7 – Сформовані персоналізовані рекомендації

1.5 Висновки за розділом 1

В першому розділі проведено аналіз предметної області дослідження, а саме: визначено важливість персоналізації в інтернет-магазині, формалізовано математичну моделі рекомендаційної системи, систематизовано методи побудови персоналізованих рекомендацій, а також розглянуто існуючі системи електронної комерції з персоналізованими рекомендаціями. Класичні алгоритми можна розділити на дві основні групи: ті, що базуються на подібності, і ті, які використовують векторне представлення користувачів і об'єктів для виявлення шаблонів. Ці розширені алгоритми використовують повну історію взаємодії користувача та продукту для отримання стислих векторів, що представляють користувача та продукт. Одним із поширених типів алгоритмів, які обговорюються в цьому розділі, є матрична факторизація.

Алгоритми матричної факторизації, такі як алгоритм найменших квадратів і байєсівський персоналізований рейтинг, відіграють важливу роль у спільному створенні персоналізованих рекомендацій.

Вони використовують методи, щоб розбити вихідну матрицю взаємодії між користувачем і продуктом на підматриці нижчої розмірності.

2 ВИБІР ТА ОБГРУНТУВАННЯ МАТЕРІАЛІВ ТА МЕТОДІВ РОЗРОБКИ

2.1 Вибір методу персоналізації товарів в інтернет-магазині

У попередньому розділі було розглянуто два основні підходи до надання онлайн-рекомендацій, а саме колабораторну фільтрацію та фільтрацію на основі контенту. Ці методи відрізняються один від одного за стратегією та технікою. Кожен підхід має власний набір переваг і недоліків, залежно від конкретного контексту, в якому вони застосовуються. Проведемо детальніший аналіз методів ранжування, а також визначимо переваги гібридизації методів.

Колаборативна фільтрація – це метод системи рекомендацій, який покладається виключно на історичні взаємодії між користувачами та елементами для створення нових рекомендацій. Його мета – визначити вподобання подібних користувачів і класифікувати їх у кластери на основі їхніх уподобань. Завдяки цьому стає можливим рекомендувати елементи користувачам на основі переваг їхніх відповідних кластерів. Основний принцип спільної фільтрації полягає в тому, що попередні взаємодії користувача з елементами, оброблені системою, надають достатньо інформації для ідентифікації подібних користувачів або елементів. Це дозволяє системі робити прогнози та рекомендації на основі цих ідентифікованих закономірностей і розуміння. Спільна фільтрація, незважаючи на свої переваги, має певні недоліки, які необхідно враховувати. Одним із головних недоліків є проблема «холодного старту» для щойно доданих елементів. Це означає, що поки користувач не оцінить ці новинки, система не зможе генерувати для них рекомендації. Це створює проблему з точки зору надання точних і відповідних рекомендацій із самого початку. Крім того, розрідженість даних викликає ще одну проблему. Коли доступні дані обмежені, це може вплинути на якість рекомендацій на основі користувачів і ще більше посилити проблему холодного старту, про яку йшлося раніше.

Іншою проблемою, яка виникає при спільній фільтрації, є масштабування. Зі збільшенням набору даних зростає складність процесу рекомендацій [21].

Оскільки метод стикається з проблемами з точки зору розрідженості та масштабованості, впроваджується більш вдосконалений підхід, який називається матричною факторизацією. Ця техніка передбачає розкладання вихідної розрідженої матриці на матрицю меншої розмірності, яка містить приховані фактори або функції, таким чином зменшуючи розрідженість.

Фільтрація на основі вмісту працює, використовуючи концепцію, згідно з якою користувачі, як правило, віддають перевагу предметам, які мають подібні характеристики чи атрибути до тих, якими вони користувалися раніше. Для прикладу: якщо користувачу сподобався комедійний фільм із певним актором, цілком ймовірно, що він також оцінить інші комедійні фільми з тим же актором. Цей підхід ґрунтується на використанні деталей, що стосуються самих елементів, таких як їхні жанри, характеристики чи описи, щоб створити персоналізовані рекомендації для конкретного користувача. Важливо відзначити, що фільтрація на основі вмісту не потребує інформації про інших користувачів, їхні оцінки чи взаємодію з елементами. Додаткові помітні переваги фільтрації на основі вмісту включають її швидке виконання, що особливо важливо для комерційних вебсайтів. У цьому контексті важливо підтримувати швидкий час відгуку для вебсторінок, гарантуючи, що він не перевищує одну секунду. Ця особливість сприяє покращенню взаємодії з користувачем, мінімізуючи очікування та безперебійній навігацію вмістом.

Серед недоліків контентної фільтрації можна виділити те, що функціональність моделі обмежена пропозицією варіантів виключно на основі наявних інтересів користувача. Це означає, що модель не має можливості розширити коло інтересів користувача. Також, однією з проблем, з якою стикаються при реалізації такої моделі, є масштабованість. Кожного разу, коли представляється новий продукт, послуга або контент, він повинен пройти процес ідентифікації атрибутів і тегування. Цей процес може займати багато часу через великий обсяг елементів, яким потрібно призначити атрибути. Крім

того, підтримання узгодженості та точності в призначенні атрибутів має вирішальне значення, оскільки неправильні або непослідовні атрибути можуть поставити під загрозу ефективність рекомендацій товарів на основі вмісту.

Системи рекомендацій часто використовують комбінацію методів, відомих як гібридні підходи, для створення персоналізованих рекомендацій. Ці гібридні методи спрямовані на подолання обмежень окремих методів шляхом інтеграції кількох методів, таких як фільтрація на основі вмісту та спільна фільтрація. Існує кілька способів ефективного поєднання цих методів, включаючи включення функцій елементів у спільну фільтрацію, врахування характеристик користувача під час фільтрації вмісту або створення єдиної моделі на основі обох підходів. Розглянемо гібридний підхід, який передбачає включення функцій елементів у спільну фільтрацію. У таких системах основою є методи спільної фільтрації, але вони посилюються за рахунок включення інформації про елементи системи під час аналізу даних. Ця інформація використовується для визначення схожості між користувачами.

Основною перевагою цього методу формування рекомендацій є те, що він покращує релевантність рекомендацій, коли існує дефіцит пар користувачів, які спільно оцінили достатню кількість елементів системи. Крім того, цей підхід дозволяє користувачам отримувати рекомендації щодо елементів, схожих на інші елементи системи, а не виключно на основі вподобань подібних користувачів. Також, варто розглянути гібридний підхід, який поєднує використання функцій користувача та фільтрацію вмісту. Цей підхід передбачає зменшення розмірності профілів користувачів, які використовуються для фільтрації вмісту, що призводить до рекомендацій, які враховують не лише вміст, але й уподобання користувача.

Одним з ефективних методів реалізації цього гібридного підходу є поєднання фільтрації вмісту зі спільною фільтрацією, яка зараз є найпопулярнішим підходом у системах рекомендацій. Основна ідея цього методу полягає в тому, щоб використовувати характеристики як фільтрації вмісту, так і спільної фільтрації в одній системі. Для прикладу, одним із

варіантів реалізації є метод уніфікованих ймовірностей, який використовує принципи латентного семантичного аналізу (LSA) [21].

Гібридний метод пропонує часткове вирішення проблеми «холодного старту», з якою стикаються нові користувачі. Якщо в профілі користувача бракує достатньої інформації, система вирішує цю проблему, рекомендуючи елементи виключно на основі їх схожості один з одним, не враховуючи жодних інших даних користувача. Гібридний підхід має істотний недолік, яким є зниження точності рекомендацій, коли в профілі користувача недостатньо інформації. У таких випадках точність значною мірою залежить від даних, що стосуються об'єктів, які раніше взаємодіяли.

Крім того, ще одним недоліком гібридного методу фільтрації є його вимога до високої обчислювальної потужності для ефективної роботи, що пов'язано з необхідністю обробки величезної кількості даних [18, 21].

На основі наведеного в [16, 18, 21], сформуємо порівняльний аналіз розглянутих методів формування онлайн-рекомендацій в таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз методів формування персоналізованих рекомендацій

Критерій	Колаборативна фільтрація	Контентна фільтрація	Гібридна фільтрація
Основна ідея	Використовує дані про поведінку та вподобання інших користувачів для рекомендацій.	Використовує характеристики об'єктів (фільмів, книг товарів) для рекомендацій.	Комбінує обидва підходи для підвищення точності рекомендацій.
Методи	Базовані на пам'яті (user-item матриці) та алгоритми машинного навчання.	Ключові слова, метадані, характеристики об'єктів.	Мікс модель-базованих та базованих на пам'яті методів, об'єднання контентної та колаборативної інформації.

Продовження таблиці 2.1

Переваги	Ефективний для великих датасетів, добре виявляє складні шаблони. Не вимагає розуміння або аналізу самого контенту.	Не потребує даних про інших користувачів, ефективний для нових об'єктів. Може давати дуже специфічні рекомендації на основі характеристик об'єкта.	Може бути налаштована для використання в різних пропорціях колаборативної та контентної інформації. Комбінує переваги обох підходів, зменшуючи їх недоліки.
Недоліки	Проблема “холодного старту”. Якщо інформація про поведінку користувачів обмежена, результати можуть бути неточними. В великих датасетах може бути недостатньо взаємодій для точних рекомендацій.	Схильна до рекомендації об'єктів, схожих на ті, що вже оцінені. Може ігнорувати корисність об'єкта в різних контекстах. Якщо метадані неточні або неповні, результати будуть неточними.	Інтеграція різних систем і даних може бути технічно складною. Для ефективності часто потрібні обидва типи даних у великій кількості. Може виникнути ризик перенавчання на специфічних взаємодіях користувачів або об'єктів.
Примітки	Часто використовується в системах рекомендацій, як Netflix.	Важливо для рекомендацій, коли історія користувача недоступна.	Популярний підхід у сучасних системах через гнучкість та точність.

На основі наведених даних в таблиці 2.1, можна зробити наступні висновки щодо використання гібридного підходу в електронній комерції.

Гібридний підхід поєднує переваги колаборативної та контентної фільтрації. Він використовує дані про поведінку користувачів та характеристики об'єктів, що дозволяє створювати релевантні рекомендації.

Гібридна фільтрація знижує вплив індивідуальних недоліків кожного підходу. Наприклад, вона може усунути проблему "холодного старту" колаборативної фільтрації, використовуючи контентну інформацію для нових об'єктів або користувачів. Також, гібридні системи можуть бути налаштовані так, щоб збалансувати використання колаборативної та контентної інформації в залежності від специфіки контексту та потреб користувачів.

Гібридні системи можуть бути більш стійкими до перенавчання на особливих взаємодіях користувачів чи об'єктів. Вони забезпечують кращу персоналізацію, враховуючи унікальні вподобання та потреби кожного користувача. У складних системах, де потрібно враховувати велику кількість змінних та взаємодій, гібридний підхід часто виявляється більш ефективним.

Отже, можна зробити загальний висновок, що гібридний підхід забезпечує більш ефективну систему рекомендацій, що важливо для задоволення потреб користувачів і впорядкування великих обсягів інформації.

2.2 Алгоритм роботи системи ранжування

Для коректної роботи системи ранжування товарів, необхідно мати достатню кількість зареєстрованих користувачів, які оцінили принаймні два продукти в системі. Вхідні дані для цього методу складаються з характеристик продуктів і відповідних оцінок, наданих користувачами системи. Ці оцінки можуть бути явними, коли користувачі безпосередньо вводять свої оцінки в систему, або неявними, коли оцінки виводяться на основі таких факторів, як кількість переглядів або час, проведений на сторінці продукту. Самі рейтинги можуть бути представлені за допомогою різних числових або символічних рейтингів. Після завершення аналізу вхідних даних система генерує рекомендації на виході. Можна уявити дані у вигляді матриці, де кожен рядок відповідає користувачеві, а кожен стовпець – продукту.

Ця матриця призначена для збору оцінок користувачів для різних продуктів. Кожен рядок представляє вектор уподобань, призначених

користувачами, тоді як кожен стовпець представляє вектор оцінок, наданих користувачами для товарів (2.1):

$$R = (r_{u,i})_{u=1,i=1}^{U,I} \quad (2.1)$$

де R – представляє матрицю елементів, $r_{u,i}$ – є невідомими оцінками, $r'_{u,i}$ – є вже існуючими оцінками в матриці.

Мета полягає в тому, щоб спрогнозувати оцінки $r_{u,i}$, використовуючи наявні оцінки в матриці $r'_{u,i}$, а також визначити невідомі елементи, які матимуть найвищі значення у своїх відповідних рядках. Це дозволяє визначити, які товари, ймовірно, будуть найбільш прихильними для конкретних користувачів. Алгоритм прогнозування оцінок включає два етапи.

Перший – визначення подібності між даним користувачем та іншими користувачами в БД. Це робиться шляхом порівняння рейтингів і вподобань.

Другий. Використовуючи оцінки схожих користувачів, можна передбачити оцінку, яку вибраний користувач дасть конкретному продукту. Оцінкам користувачів, які найбільше схожі на вибраного користувача, надається найбільша вага в цьому процесі. Для початку розглядаємо лише елементи у векторах уподобань для кожного користувача та оцінки користувачів для кожного продукту, для якого відомі значення.

Визначення подібності між двома векторами дійсних чисел має вирішальне значення. Традиційний підхід передбачає обчислення коефіцієнта кореляції. Зокрема, при роботі з двома векторами, що представляють переваги користувача в системі, використовується коефіцієнт кореляції Пірсона (2.2):

$$w_{u,j} = \frac{\sum_a (r_{u,a} - \bar{r}_u)(r_{j,a} - \bar{r}_j)}{\sqrt{\sum_a (r_{u,a} - \bar{r}_u)^2} \sqrt{\sum_a (r_{j,a} - \bar{r}_j)^2}}, \quad (2.2)$$

де $\bar{r}_u$ – рейтинг, який виставлено користувачем u , $r_{u,j} \in [-1; 1]$.

2.3 Основні метрики оцінки системи ранжування

Показники класифікації та показники ранжування відіграють вирішальну роль в оцінці продуктивності систем рекомендацій. Ці показники допомагають оцінити, наскільки добре модель рекомендує відповідну підмножину товарів із усього набору доступних товарів на платформі.

У сфері рекомендаційних систем точність є загальновикористовуваним показником класифікації. Точність (precision) вимірює частку релевантних елементів, серед усіх елементів, які пропонуються для користувача (2.3):

$$Precision_u = \frac{TP_u}{TP_u + FP_u}, \quad (2.3)$$

Повнота, також відома як чутливість (recall), відноситься до відсотка релевантних суб'єктів, які визначені за результатами моделі (2.4):

$$Recall_u = \frac{TP_u}{TP_u + FN_u}, \quad (2.4)$$

де TP_u – представляє кількість релевантних елементів, які були ідентифіковані як релевантні моделлю, FP_u – кількість нерелевантних елементів, які були помилково позначені як релевантні моделлю, і FN_u – кількість релевантних елементів, які були позначені моделлю як нерелевантні.

У контексті надання рекомендацій вмісту точна модель спрямована на мінімізацію присутності нерелевантної інформації, тоді як чутлива модель гарантує, що релевантний вміст не буде пропущено. Під час оцінювання систем рекомендацій метрики ранжування використовуються для оцінки того, наскільки добре модель визначає пріоритетність найбільш відповідних елементів, враховуючи порядок наступних рекомендацій, що відрізняє їх від метрик класифікації. Розглянемо найпопулярніші метрики оцінювання.

Середньозважена точність (MAP, Mean Average Precision) (2.5 – 2.6):

$$MAP = \frac{1}{|U|} \sum_{u \in U} AP(u), \quad (2.5)$$

$$AP(u) = \sum_{k=1}^n Precision_u(k), \quad (2.6)$$

де $Precision_u(k)$ – точність кількох перших рекомендацій для кожного користувача та релевантність позицій у списку рекомендацій.

Мета оптимізації моделі рекомендацій, заснованої на MAP, полягає в тому, щоб навчити її створювати якомога більше релевантних позицій, незалежно від довжини списку рекомендацій. Іншими словами, акцент робиться на покращенні якості рекомендацій, а не на їх кількості.

Нормалізований дисконтований кумулятивний приріст (NDGG, Normalized Discounted Cumulative Gain) (2.7 – 2.9):

$$NDCG(k) = \frac{DCG(k)}{IDCG(k)}, \quad (2.7)$$

$$DCG(k) = \sum_{i=1}^k \frac{\hat{r}_i}{\log(i+1)}, \quad (2.8)$$

$$IDCG(k) = \sum_{i=1}^k \frac{r_i}{\log(i+1)}, \quad (2.9)$$

де $\hat{r}_i$ представляє рейтинги, присвоєні товарам на основі їх актуальності, визначеної моделлю. З іншого боку, r_i представляє справжні рейтинги цих продуктів, які розташовані в порядку спадання, відомого як ідеальний рейтинг. NDCG, на відміну від MAP (середньозважена точність),

накладає вищий штраф на модель за рекомендацію нерелевантних елементів на верхніх позиціях. Таким чином, оптимізація моделі відповідно до цього критерію гарантує, що найбільш релевантні елементи представлені як верхні елементи.

2.4 Висновки за розділом 2

В другому розділі проведено аналіз систем рекомендацій, включаючи необхідні дані для їх побудови та потенційні проблеми, які можуть виникнути під час їх розробки. Також, проведено аналіз переваг та недоліків цих методів. У результаті було встановлено, що поєднання існуючих методів є найбільш сприятливим підходом для вирішення таких проблем, як проблема «холодного запуску», розрідженість даних і відсутність новизни. Важливо підкреслити, що для кожного продукту має бути створена спеціальна система рекомендацій, адаптована до конкретних цілей. В результаті, було систематизовано обраний метод і конкретні дані, які збиратимуться через застосунок.

Крім того, проаналізовано основні показники, необхідні для оцінки продуктивності рекомендаційних систем, а також їх переваги та недоліки.

3 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

3.1 Моделювання бізнес-процесів в системі

На рисунку 3.1 представлена діаграма бізнес-процесів обробки даних системи електронної комерції з рекомендаційною системою (РС).

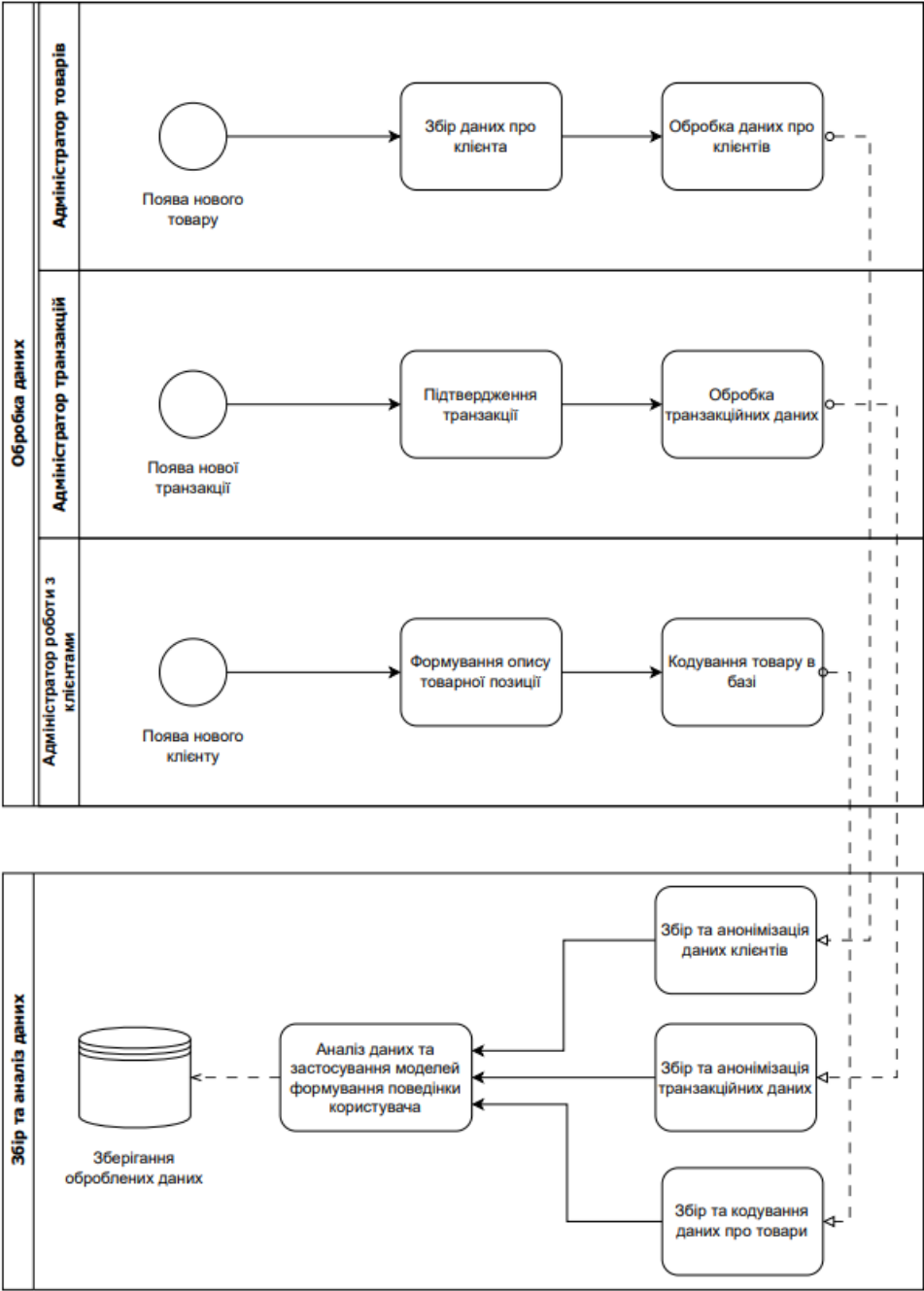


Рисунок 3.1 – Діаграма бізнес-процесів обробки даних системи з РС

З рисунку 3.1, де представлена діаграма бізнес-процесів обробки даних в системі електронної комерції з РС, можна виділити декілька етапів перерахованих нижче.

Починаючи з етапи обробку даних адміністратором товарів, схема демонструє стадію, на якій відбувається поява нового товару. Це вказує на момент, коли новий товар вводиться в систему, що може включати його реєстрацію, опис і присвоєння унікального ідентифікатора. Далі процес переходить до збору даних про клієнта, етапу, на якому інформація про клієнтів збирається для подальшої обробки. Обробка даних про клієнта може включати оновлення існуючих профілів та збір історії покупок для аналітики.

У центральній частині схеми представлений процес, пов'язаний з новою транзакцією, який описує дії, що виконуються при здійсненні нових продажів або замовлень. Підтвердження транзакції є критичною точкою в цьому процесі, оскільки це етап, на якому транзакція верифікується перед тим, як вона буде остаточно закріплена в системі. Обробка транзакційних даних вказує на аналіз та управління даними після того, як транзакція підтверджена.

Далі, представлено процес з участю адміністратора по роботі з клієнтами, що стосується появи нового клієнта. Цей етап включає введення інформації про нового клієнта в систему, його ідентифікацію та створення профілю. Потім інформація про товари підлягає формуванню опису товарної позиції, що може означати деталізацію характеристик товару, встановлення ціни та інші важливі параметри. Кодування товару свідчить про внесення цих даних у БД, що дозволяє системі ефективно зберігати та використовувати дані.

Далі, представлено процес, який стосується зберігання та обробки даних клієнтів і транзакцій. На схемі зображено сховище даних або базу даних, де відбувається зберігання оброблених даних. Це вказує на місце, де інформація зберігається після того, як пройшла через всі необхідні етапи обробки та анонімізації. Процес "Збір та анонімізація даних клієнтів" включає збір інформації про клієнтів, а потім її анонімізацію, щоб захистити особисті дані перед їх подальшим використанням або аналізом. Сусідній процес "Збір

та анонімізація транзакційних даних" є подібним до збору даних клієнтів, але зосереджений на транзакціях. Це означає, що дані про покупки, замовлення або інші транзакції також збираються та анонімізуються. Також, існує процес "Збір та кодування даних про товари", який відноситься до збору інформації про те, як товари представлені в системі. Це може включати деталі про те, як товари класифікуються та інші метадані, які допомагають в ідентифікації в БД.

Далі, на рисунку 3.2 представлений взаємозв'язок бізнес-процесів "Збір та аналіз даних" та "Формування рекомендацій товарів" в комерційній системі.

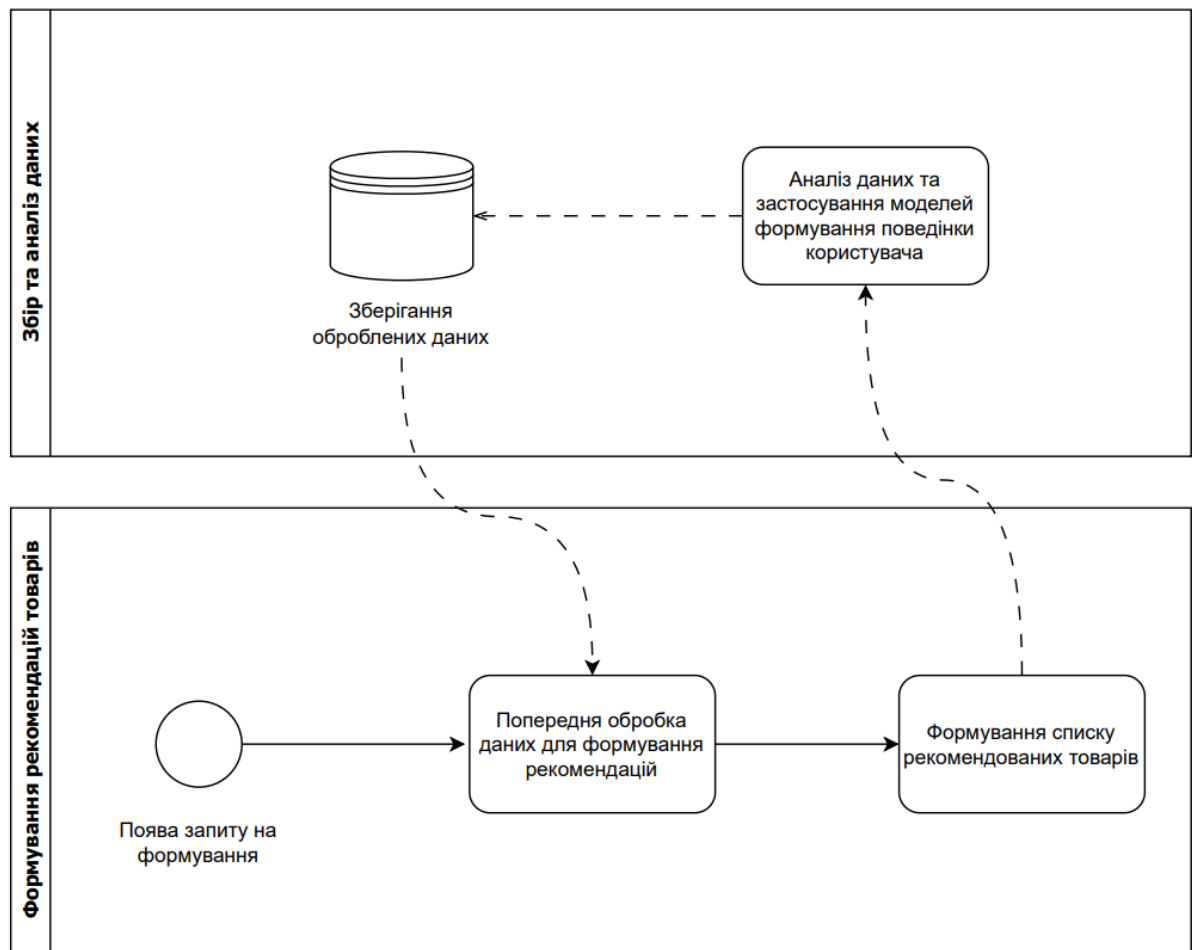


Рисунок 3.2 – Взаємозв'язок бізнес-процесів "Збір та аналіз даних" та "Формування рекомендацій товарів"

На рисунку 3.2 представлено процес, який починається з ініціації запиту. Після цього відбувається попередня обробка даних, мета якої –

підготовка до формування рекомендацій. У наступному етапі формується список рекомендованих товарів. Паралельно, в іншому потоці процесу, відбувається збір даних та їх аналіз, які після обробки зберігаються. Збережені дані використовуються для аналізу та застосування моделей, які впливають на формування поведінки користувача, що впливає на формування рекомендацій.

3.2 Архітектура програмної системи

Для розробки програмної системи з використанням рекомендаційної системи використовується компонентно-орієнтована архітектура. Згаданий підхід до проектування та розробки зосереджується на перетворенні елементів на окремі багаторазові компоненти, спрямовані на підвищення швидкості та ефективності розгортання та розробки програми. Різні типи компонентів служать різним цілям для розробників і компаній і можуть узгоджено інтегруватися в різні інтерфейси та модулі. Крім того, компанії мають можливість комерціалізувати ці компоненти через ринкові майданчики, дозволяючи іншим компаніям використовувати їх через API. Включення компонентної системи передбачає забезпечення того, щоб компоненти мали певні характеристики в системі, наприклад, були незалежними модулями, здатними обмінюватися даними через інтерфейси. Крім того, система повинна передбачати розбирання та повторне збирання компонентів, підкреслюючи можливість повторного використання, згуртованість і незалежність, що полегшує їх масштабованість в системі без необхідності значних налаштувань.

Спираючись на концепцію запитів AJAX, ці технології дозволяють здійснювати прямі виклики сервера з боку клієнта. Це полегшує плавне оновлення об'єктної моделі документа (DOM) і відображення вмісту без необхідності оновлення сторінки. Компоненти мають можливість запитувати власну бізнес-логіку та оновлювати свій інтерфейс незалежно, не заважаючи іншим компонентам або їхнім інтерфейсам користувача. Ця незалежність виявляється особливо корисною для завдань, які інакше можуть призвести до

непотрібних перезавантажень сторінки, спричиняючи зниження продуктивності. Кожен компонент має певні функції, які можна перевизначати або ізолювати відповідно до потреб програми. Наприклад, Facebook використовує компоненти для покращення роботи та ефективності своєї стрічки новин. React.js, зокрема, вирізняється ефективним керуванням компонентами. Він використовує віртуальний DOM і використовує метод «відмінювання» для ідентифікації змін елементів і рендеринг лише змінених елементів, уникаючи необхідності повторно рендерити весь компонент.

На рисунку 3.3 представлена візуалізація компонентів системи.

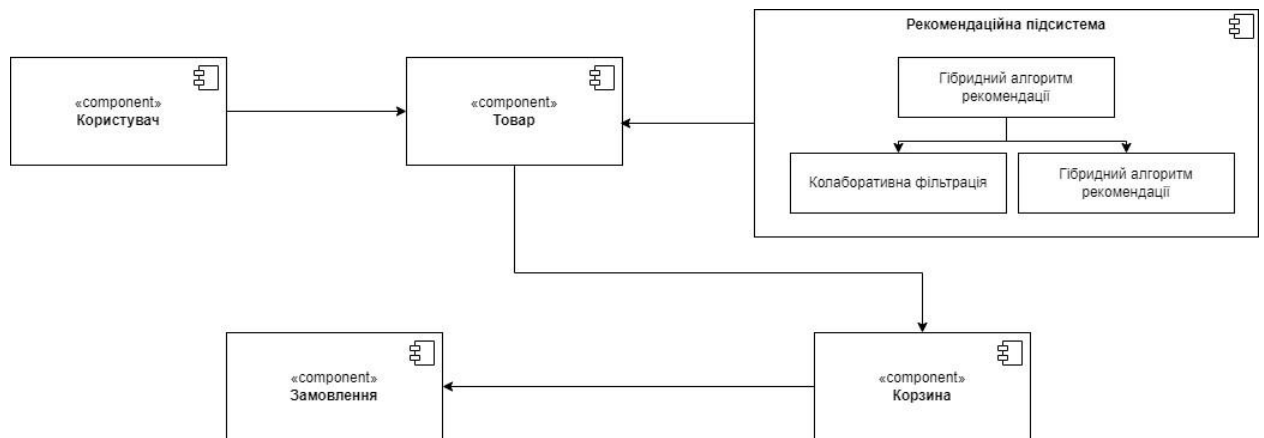


Рисунок 3.3 – Компонентна діаграма системи з РС

Отже, розробка на основі компонентів пропонує численні переваги, окрім простого розміщення багаторазово використовуваних фрагментів коду в програмних системах. Переваги використання компонентно-орієнтованої архітектури включають: незалежність і можливість повторного використання кожного компонента в архітектурі на основі компонентів, що полегшує декомпозицію моноліту інтерфейсу на менші, більш керовані компоненти. Це робить оновлення або модифікації безперешкодними, оскільки потрібно оновлювати лише відповідні компоненти, що спрощує процес модифікації коду. Крім того, багаторазове використання значно зменшує кількість коду, необхідного для бізнес-додатків у межах компонентної структури. Розробники можуть уникати повторюваного кодування та натомість зосереджуватися на

основних функціях, що дозволяє їм застосовувати ці компоненти до різноманітних програм, які задовольняють різноманітні потреби або працюють на різних платформах. Наприклад, розглянемо компонент, який відповідає за функцію автентифікації в програмі. Забезпечуючи зміну лише логіки авторизації в програмах, які використовують цей компонент, компонент залишається незмінним незалежно від ПЗ, в яке він інтегрований.

3.3 Основні вимоги до програмної системи

Нижче представлено основні вимоги користувача до програмної системи з персоналізованим ранжуванням товарів:

- отримання рекомендацій на основі оцінок інших користувачів;
- отримання рекомендацій на основі товарів, що сподобалися;
- функціонал автентифікації та безпеки користувача;
- легкість та зручність у використанні інтерфейсу системи;
- можливість оформлення замовлення в інтернет-магазині;
- надійність та стабільність роботи системи;
- пошук товарів з фільтрацією за різними параметрами;
- інтуїтивно зрозумілі фільтри для вдосконалення пошуку;
- простий та безпечний процес оформлення замовлення в магазині.

3.4 Висновки за розділом 3

Поведено проектування архітектури інформаційної системи з використанням персоналізованого ранжування. Для розробки обрано компонентно-орієнтовану архітектуру, переваги якої включають: незалежність і можливість повторного використання кожного компонента в архітектурі на основі компонентів, що полегшує декомпозицію моноліту інтерфейсу на менші, більш керовані компоненти. Визначено вимоги користувача до програмної системи з персоналізованими рекомендаціями.

4 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

4.1 Обґрунтування вибору засобів розробки

Для розробки програмної системи з використанням рекомендаційної системи використовуються наступні технології: мова програмування JavaScript, фреймворк для розробки компонентів React.js та сховище Firebase.

4.1.1 Мова програмування JavaScript

JavaScript, універсальна мова, яка постійно розвивається, охоплює кілька парадигм програмування. Він включає в себе динамічні характеристики, охоплює різні типи та оператори, а також пропонує низку стандартних вбудованих об'єктів і методів. Підтримуючи об'єктно-орієнтоване програмування через прототипи та класи об'єктів, JavaScript також охоплює принципи функціонального програмування. Використовуючи гнучкість функцій як першокласних об'єктів, JavaScript дозволяє створювати їх за допомогою виразів і забезпечує інтеграцію в інші частини кодової бази.

JavaScript зазвичай залежить від середовища виконання, такого як веббраузер, щоб забезпечити об'єкти та методи, які дозволяють сценаріям взаємодіяти з цим середовищем, як-от DOM вебсторінки. Ці середовища працюють в одному потоці. JavaScript також використовує середовище виконання для ІО сценаріїв, наприклад використання елементів HTML `<script>`. Це не властива функція мови, але вона поширена в більшості середовищ JavaScript. JavaScript послідовно обробляє повідомлення з черги. Для кожного повідомлення він викликає функцію, створюючи фрейм стека з аргументами функції та локальними змінними. Стек викликів розширюється або звужується відповідно до вимог функції. Коли функція завершується, а стек викликів порожній, JavaScript переходить до наступного повідомлення в черзі. Цей процес, відомий як цикл подій, дотримується моделі «виконання до завершення», де кожне повідомлення повністю обробляється перед переходом

до наступного. Однак модель паралелізму JavaScript не блокує: вона виконує введення/виведення програми через події та виклики, а не зупиняє виконання.

JavaScript також може використовуватися для реалізації складних алгоритмів фільтрації на стороні сервера (наприклад, через Node.js), обробляючи великі масиви даних для створення персоналізованих рекомендацій. На стороні фронтенду, JavaScript може бути використаний разом з фреймворками, такими як React, Angular чи Vue, для створення інтерактивних користувацьких інтерфейсів.

Це дозволяє ефективно представляти персоналізовані рекомендації користувачам, забезпечуючи високий рівень залученості та задоволення.

4.1.2 Фреймворк для розробки компонентів React.Js

React, також відомий як React.js або, є вільнодоступною інтерфейсною бібліотекою JavaScript, яка використовується для створення інтерфейсів користувача шляхом розгортання компонентів. Універсальність React очевидна, оскільки він полегшує створення односторінкових, мобільних, а також серверних додатків за допомогою фреймворків, таких як Next.js. Через те, що React зосереджується виключно на користувацькому інтерфейсі та рендерингу компонентів у DOM, додаткові бібліотеки часто інтегруються в програми React, щоб активувати такі функції, як маршрутизація та інші операції на стороні клієнта. Основний атрибут React полягає в його здатності оновлювати лише змінені сегменти сторінки, таким чином запобігаючи непотрібному повторному рендерингу незмінених елементів DOM.

Коли користувач хоче отримати доступ до вебсторінки, він вводить її URL-адресу у свій веббраузер, який потім надсилає запит на сервер для отримання вебсторінки. Згодом браузер відобразить отриману вебсторінку. Якщо користувач вирішує перейти на іншу сторінку на тому самому вебсайті, натиснувши посилання, на сервер надсилається новий запит для отримання нової сторінки. Цей процес зворотного зв'язку між вашим браузером

(клієнтом) і сервером повторюється для кожної нової сторінки чи ресурсу, до якого ви збираєтеся отримати доступ на вебсайті. Хоча цей звичайний метод завантаження ефективний, він може бути неефективним для вебсайтів, які сильно керуються даними, що призводить до поганої взаємодії з користувачем.

Крім того, у традиційній програмі JavaScript потрібна ручна маніпуляція DOM, щоб відобразити зміни під час оновлення даних, що потребує повного перезавантаження сторінки. Однак React використовує інший підхід, дозволяючи розробляти односторінкові програми (SPA).

На відміну від традиційних вебсайтів, односторінкові програми завантажують лише один документ HTML за початковим запитом, а потім використовують JavaScript для оновлення певних частин або вмісту сторінки, які потребують змін. Цей підхід, відомий як маршрутизація на стороні клієнта, позбавляє клієнта необхідності перезавантажувати всю вебсторінку для кожного нового запиту, що забезпечує покращену продуктивність і більш динамічну взаємодію з користувачем. Крім того, React використовує віртуальний DOM, який є представленням фактичного DOM. Щоразу, коли відбувається зміна стану даних, віртуальний DOM React оновлюється, щоб відобразити цю зміну. Згодом React порівнює віртуальний DOM із фактичним DOM, щоб визначити конкретні зміни. Потім він визначає найефективніший спосіб оновлення фактичного DOM без необхідності повторного відтворення.

4.1.3 Сховище даних Firebase

Firebase, платформа, розроблена Google, призначена для полегшення створення вебдодатків. Він надає низку інструментів і послуг для підтримки розробників у процесі створення першокласних додатків, підвищення якості додатків і розширення їх бази користувачів. Однією з його головних функцій є Realtime Database, яка функціонує як хмарна база даних NoSQL, що дозволяє зберігати дані в реальному часі та синхронізувати їх між користувачами. Крім того, він включає правила безпеки бази даних у реальному часі Firebase,

адаптовану мову правил для структурування даних і регулювання доступу до них. Крім того, під час автономних періодів пакети SDK бази даних у реальному часі використовують локальний кеш для керування та збереження змін, забезпечуючи безперебійну синхронізацію із сервером після повторного підключення. На рисунку 4.1 представлено порівняння підходів до розробки.

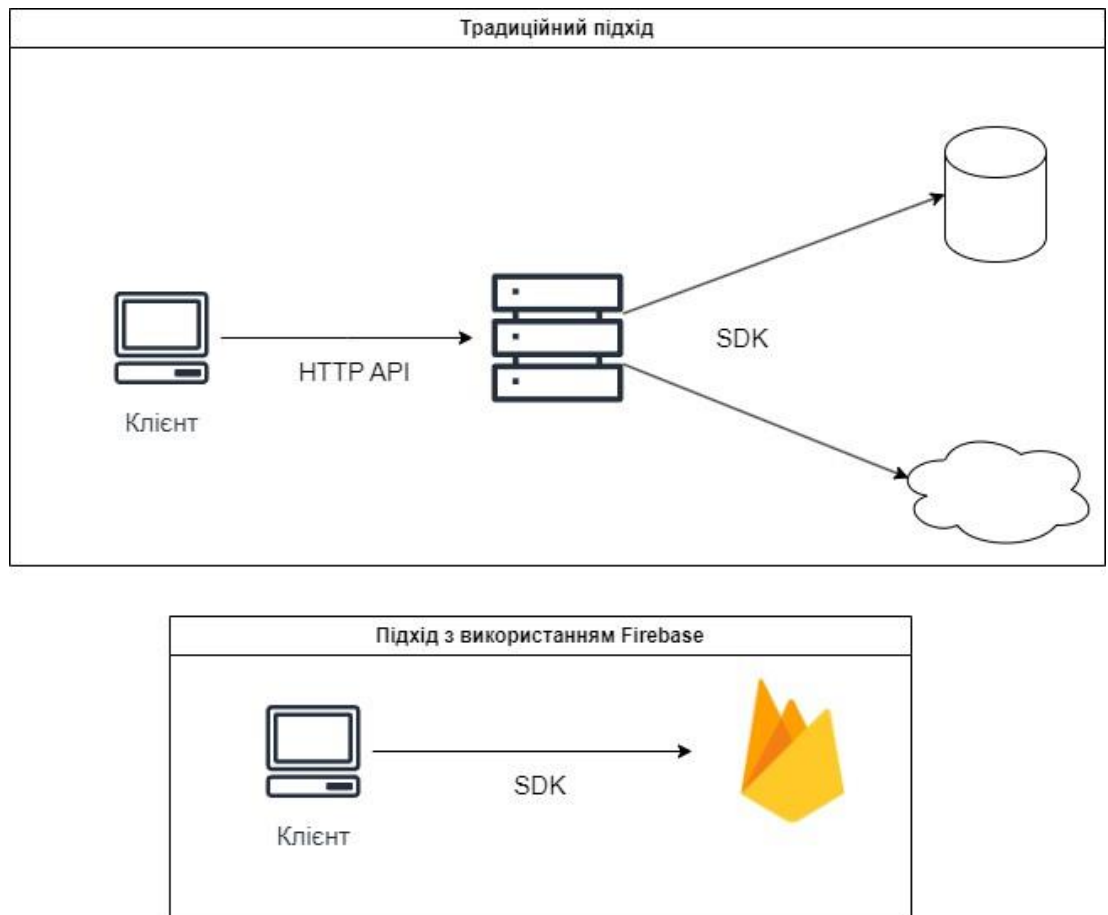


Рисунок 4.1 – Порівняння традиційного підходу до розробки вебзастосунків та підходу з використанням Firebase

Іншим невід’ємним компонентом є Firebase Storage – надійна, зручна служба зберігання об’єктів, призначена для розміщення створеного користувачами контенту, наприклад фотографій і відео. Ця послуга, підкріплена надійною інфраструктурою Google Cloud Storage, створена для забезпечення швидкого зберігання та отримання створеного користувачами вмісту безпосередньо з Firebase SDK. На додаток до цих основних компонентів, Firebase включає низку інших функцій, включаючи

автентифікацію, хостинг, хмарні функції та аналітику, таким чином представляючи себе як комплексну екосистему для розробки вебдодатків.

4.2 Розробка основних компонентів системи

Розглянемо детальніше реалізацію гібридного алгоритму фільтрації, який включає колаборативну на основі елементів та контентну фільтрацію.

На рисунках 4.2 – 4.4 представлена реалізація колаборативної фільтрації на основі елементів, де використовується підхід, описаний в підрозділі 2.2.

```
class ItemBasedRecommender {  
  // транспонує матрицю  
  transposeMatrix(matrix) {  
    return matrix[0].map((col, c) => matrix.map((row, r) => matrix[r][c]));  
  }  
  
  // видаляє зазначений індекс з даного масиву  
  removeIndex(arr, i) {  
    return arr.slice(0, i).concat(arr.slice(i + 1));  
  }  
  
  // повертає n найближчих сусідів  
  getNearestNeighbours(arr, n) {  
    arr.sort((a, b) => b.correlation - a.correlation);  
    return arr.filter((x) => x.correlation > 0).slice(0, n);  
  }  
  
  // повертає індекси масиву, що відповідають значенню  
  getMatchingIndices(arr, val) {  
    var indices = [];  
    for (let i = 0; i < arr.length; i++)  
      if (arr[i] == val)  
        indices.push(i);  
    return indices;  
  }  
}
```

Рисунок 4.2 – Реалізація компонентів підсистеми колаборативної фільтрації

```

// повертає набори елементів, які оцінили і користувач a, і b
getCommonSet(a, b) {
  let result = [];
  for (let i = 0; i < a.length; i++) {
    if (a[i] !== -1 && b[i] !== -1) result.push(i);
  }

  return result;
}

// середнє значення масиву
mean(arr) {
  let filteredArr = arr.filter((x) => x !== -1);
  return filteredArr.reduce((a, b) => a + b) / filteredArr.length;
}

// обчислює середні значення для кожного користувача
getUserAverages(matrix) {
  let result = [];
  for (var i = 0; i < matrix.length; i++)
    result.push(this.mean(matrix[i]));
  return result;
}

// обчислює sim(a, b) і повертає масив об'єктів
getSimilarity(item, items, userAverages) {
  let similarities = [];

  for (let i = 0; i < items.length; i++) {
    let commonSet = this.getCommonSet(item.ratings, items[i].ratings);

    let numerator = 0;
    let denominator1 = 0;
    let denominator2 = 0;
    for (let j = 0; j < commonSet.length; j++) {
      numerator += (item.ratings[commonSet[j]] - userAverages[commonSet[j]]) * (items[i].ratings[commonSet[j]] - userAverages[commonSet[j]]);
      denominator1 += Math.pow((item.ratings[commonSet[j]] - userAverages[commonSet[j]]), 2);
      denominator2 += Math.pow((items[i].ratings[commonSet[j]] - userAverages[commonSet[j]]), 2);
    }

    let denominator = Math.sqrt(denominator1) * Math.sqrt(denominator2);
    let result = denominator !== 0 ? numerator / denominator : 0;
    similarities.push({ item: items[i].item, ratings: items[i].ratings, correlation: result });
  }
}

```

Рисунок 4.3 – Реалізація компонентів підсистеми колаборативної фільтрації
(продовження)

```

// обчислює pred(a, p) і повертає масив з значеннями, заміненими передбачуваним значенням
getPrediction(item, items, neighbourhoodSize) {
  let prediction = [...item.ratings];
  let unratedItems = this.getMatchingIndicies(prediction, -1);

  for (let i = 0; i < unratedItems.length; i++) {
    let numerator = 0;
    let denominator = 0;
    let nearestNeighbours = this.getNearestNeighbours(this.getSimilarity(item, items, this.getUserAverages(items)), neighbourhoodSize);
    for (let j = 0; j < nearestNeighbours.length; j++) {
      numerator += nearestNeighbours[j].correlation * nearestNeighbours[j].ratings[unratedItems[i]];
      denominator += nearestNeighbours[j].correlation;
    }

    prediction[unratedItems[i]] = denominator !== 0 ? (numerator / denominator) : -1;
  }

  return prediction;
}

export default ItemBasedRecommender;

```

Рисунок 4.4 – Реалізація функції для формування передбачення рейтингів

В таблиці 4.1 представлено опис вищенаведених функцій.

Таблиця 4.1 – Опис розроблених функцій

Назва функції	Опис
<code>transposeMatrix(matrix)</code>	Транспонує матрицю (змінює рядки та стовпці місцями). Для кожного стовпця у вихідній матриці створює рядок у транспонованій матриці.
<code>removeIndex(arr, i)</code>	Видаляє елемент за зазначеним індексом <i>i</i> з масиву <i>arr</i> , створюючи новий масив із елементами до та після індексу <i>i</i> .
<code>getNearestNeighbours(arr, n)</code>	Отримує <i>n</i> найближчих сусідів із масиву <i>arr</i> . Спочатку сортує масив за спаданням кореляції, потім вибирає <i>n</i> перших із позитивною кореляцією.
<code>getCommonSet(a, b)</code>	Визначає спільний набір елементів для двох масивів <i>a</i> і <i>b</i> . Додає елемент до результату, якщо він присутній у обох масивах та не дорівнює -1.
<code>getMatchingIndicies(arr, val)</code>	Знаходить усі індекси в масиві <i>arr</i> , що відповідають заданому значенню <i>val</i> . Повертає масив цих індексів.
<code>mean(arr)</code>	Обчислює середнє значення елементів у масиві <i>arr</i> , ігноруючи елементи зі значенням -1. Використовується для подальших розрахунків.
<code>getUserAverages(matrix)</code>	Обчислює середнє значення для кожного користувача у матриці <i>matrix</i> , використовуючи функцію <code>mean</code> для кожного рядка.
<code>getSimilarity(item, items, userAverages)</code>	Розраховує схожість між двома товарами. Знаходить спільні рейтинги та обчислює кореляцію на основі середніх значень рейтингів.
<code>getPrediction(item, items)</code>	Розраховує передбачувані оцінки для неоцінених елементів у <i>item</i> через зважене середнє інших елементів в <i>items</i> , враховуючи кореляцію.

Отже, підсистема працює з масивами даних, представляючи користувацькі оцінки, і використовує математичні методи для аналізу та

прогнозування переваг користувачів. Спочатку вона транспонує матрицю оцінок для зручності обробки. Потім, за допомогою різних операцій, вона виключає непотрібні елементи, знаходить спільні характеристики між різними наборами даних та визначає рівень схожості або кореляції між різними елементами. Основними кроками є визначення схожості між користувачами або елементами на основі їх спільних рейтингів, використання цієї схожості для прогнозування невідомих рейтингів і вибір найбільш релевантних рекомендацій для кожного користувача.

Далі, на рисунках 4.5 – 4.9 представлений клас контентної фільтрації та реалізація основних методів для розрахунку подібності між товарами.

```
class ContentBasedRecommender {
  constructor(options = {}) {
    this.setOptions(options);

    this.data = {};
  }

  setOptions(options = {}) {
    // validation
    if ((options.maxVectorSize !== undefined) &&
      (!Number.isInteger(options.maxVectorSize) || options.maxVectorSize <= 0)) {
      throw new Error('Параметр maxVectorSize має бути цілим числом та бути більшим за 0');
    }

    if ((options.maxSimilarDocuments !== undefined) &&
      (!Number.isInteger(options.maxSimilarDocuments) || options.maxSimilarDocuments <= 0)) {
      throw new Error('Параметр maxSimilarDocuments має бути цілим числом та бути більшим за 0');
    }

    if ((options.minScore !== undefined) &&
      (!Number.isInteger(options.minScore) || options.minScore < 0 || options.minScore > 1)) {
      throw new Error('Параметр minScore має бути числом від 0 до 1');
    }

    this.options = Object.assign({}, defaultOptions, options);
  }

  train(documents) {
    this.validateDocuments(documents);

    if (this.options.debug) {
      console.log(`Total documents: ${documents.length}`);
    }

    const preprocessDocs = this._preprocessDocuments(documents, this.options);

    const docVectors = this._produceWordVectors(preprocessDocs, this.options);

    this.data = this._calculateSimilarities(docVectors, this.options);
  }
}
```

Рисунок 4.5 – Реалізація методів класу ContentBasedRecommender

```

validateDocuments(documents) {
  if (!isArray(documents)) {
    throw new Error('Документи мають являти собою масив об'єктів');
  }

  for (let i = 0; i < documents.length; i += 1) {
    const document = documents[i];

    if (!has(document, 'id') || !has(document, 'content')) {
      throw new Error('Документи повинні мати поля id та content');
    }

    if (has(document, 'tokens') || has(document, 'vector')) {
      throw new Error('Властивості "tokenів" та "вектора" зарезервовані та не можуть використовуватися як властивості документа');
    }
  }
}

getSimilarDocuments(id, start = 0, size = undefined) {
  let similarDocuments = this.data[id];

  if (similarDocuments === undefined) {
    return [];
  }

  const end = (size !== undefined) ? start + size : undefined;
  similarDocuments = similarDocuments.slice(start, end);

  return similarDocuments;
}

```

Рисунок 4.6 – Реалізація методів класу ContentBasedRecommender
(продовження)

```

export() {
  return {
    options: this.options,
    data: this.data,
  };
}

import(object) {
  const { options, data } = object;

  this.setOptions(options);
  this.data = data;
}

_preprocessDocuments(documents) {
  const processedDocuments = documents.map(item => {
    let tokens = this._getTokensFromString(item.content);
    return {
      id: item.id,
      tokens,
    };
  });

  return processedDocuments;
}

```

Рисунок 4.7 – Реалізація методів класу ContentBasedRecommender
(продовження)

```

_calculateSimilaritiesBetweenTwoVectors(documentVectors, targetDocumentVectors, options) {
  const data = {
    ...this.initializeDataHash(documentVectors),
    ...this.initializeDataHash(targetDocumentVectors)
  };

  for (let i = 0; i < documentVectors.length; i += 1) {
    if (options.debug) console.log(`Розрахунок оцінки подібності для документа: ${i}`);

    for (let j = 0; j < targetDocumentVectors.length; j += 1) {
      let documentVectorA = documentVectors[i];
      let targetDocumentVectorB = targetDocumentVectors[j];
      const idi = documentVectorA.id;
      const vi = documentVectorA.vector;
      const idj = targetDocumentVectorB.id;
      const vj = targetDocumentVectorB.vector;
      const similarity = vi.getCosineSimilarity(vj);

      if (similarity > options.minScore) {
        data[idi].push({
          id: targetDocumentVectorB.id,
          score: similarity
        });
        data[idj].push({
          id: documentVectorA.id,
          score: similarity
        });
      }
    }
  }

  this.orderDocuments(data, options);

  return data;
}

```

Рисунок 4.8 – Реалізація методу розрахунку подібності між двома векторами (товарами)

```

orderDocuments(data, options) {
  Object.keys(data)
    .forEach((id) => {
      data[id].sort((a, b) => b.score - a.score);

      if (data[id].length > options.maxSimilarDocuments) {
        data[id] = data[id].slice(0, options.maxSimilarDocuments);
      }
    });
}

export default ContentBasedRecommender;

```

Рисунок 4.9 – Реалізація методу для сортування товарів

Важливо зазначити, що в вищепредставлених методах використовується поняття документа, що ототожнюється з товарами в інтернет-магазині.

Нижче наведено опис вищепредставлених функцій.

Метод `setOptions` призначений для встановлення та валідації опцій класу. Він перевіряє, чи опції `maxVectorSize`, `maxSimilarDocuments` та `minScore` є валідними (наприклад, `maxVectorSize` та `maxSimilarDocuments` повинні бути цілими числами, більшими за 0; `minScore` - це число в діапазоні від 0 до 1). Якщо параметри невалідні, метод кидає виключення.

Після валідації об'єкт `options` зберігається як властивість `this.options`.

Метод `train` перевіряє, чи документи мають правильний формат за допомогою `validateDocuments`. Після цього він обробляє документи, перетворюючи їх в масив токенів з використанням `_preprocessDocuments`.

На основі цих токенізованих документів метод створює вектори слів за допомогою `_produceWordVectors`. Нарешті, він розраховує подібності між цими документами використовуючи `_calculateSimilarities`.

Метод `validateDocuments` перевіряє, чи поданий масив документів відповідає очікуваному формату. Кожен документ повинен мати поля `id` та `content`. Також перевіряється, що в документі немає полів `tokens` та `vector`, оскільки вони зарезервовані для внутрішнього використання.

Метод `getSimilarDocuments` повертає товари, подібні до заданого товару. Він приймає ідентифікатор товару та діапазон (початок та кількість), щоб визначити, які подібні товари повертати. Якщо для заданого ідентифікатора немає даних, повертається порожній масив.

Метод `_preprocessDocuments` використовується для обробки документів. Він перетворює вміст кожного документа на масив токенів, використовуючи `tokenizer` та стемінг, видаляє HTML теги та стоп-слова.

Метод `_produceWordVectors` є приватним та створює вектори слів для оброблених документів. Він використовує TF-IDF для визначення ваги кожного слова у документі, обмежуючи розмір вектора до `maxVectorSize`.

Методи `calculateSimilarities` та `calculateSimilaritiesBetweenTwoVectors` розраховують подібності між документами. `_calculateSimilarities` розраховує подібності всередині одного набору документів, а `_calculateSimilaritiesBetweenTwoVectors` – між двома наборами. Вони використовують косинусну подібність для визначення подібності товарів.

Далі, на рисунках 4.10 – 4.11 представлена реалізація компонента реєстрації та авторизації користувача в системі.

```
import { useReducer, useEffect } from 'react';

import { onAuthStateChanged, signInAnonymously } from 'firebase/auth';
import { doc, getDoc } from 'firebase/firestore';

import { auth } from 'db/config';
import { db } from 'db/config';

import AuthContext from './auth-context';

const initialState = {
  user: null,
  name: null,
  lastName: null,
  email: null,
  phoneNumber: null,
  addresses: [],
  isVerified: false,
  isAdmin: false,
  authIsReady: false,
};

const authReducer = (state, action) => {
  switch (action.type) {
    case 'AUTH_IS_READY': {
      return {
        user: action.payload.user,
        name: action.payload.name,
        lastName: action.payload.lastName,
        email: action.payload.email,
        phoneNumber: action.payload.phoneNumber || null,
        addresses: action.payload.addresses || [],
        isVerified: true,
        isAdmin: action.payload.isAdmin || null,
        authIsReady: true,
      };
    }

    case 'ANONYMOUS_AUTH_IS_READY': {
      return {
        ...initialState,
        user: action.payload.user,
        authIsReady: true,
      };
    }
  }
}
```

Рисунок 4.10 – Реалізація редуктора автентифікації (`authReducer`)

```

const AuthProvider = ({ children }) => {
  const [state, dispatch] = useReducer(authReducer, initialState);

  useEffect(() => {
    const unsub = onAuthStateChanged(auth, async (user) => {
      if (user) {
        const userRef = doc(db, 'users', user.uid);
        const userDoc = await getDoc(userRef);

        if (userDoc.exists()) {
          const userData = userDoc.data();
          dispatch({
            type: 'AUTH_IS_READY',
            payload: { user, ...userData },
          });
        } else {
          dispatch({
            type: 'ANONYMOUS_AUTH_IS_READY',
            payload: { user },
          });
        }
      } else {
        await signInAnonymously(auth);
      }
    });

    return () => unsub();
  }, []);

  console.log('auth-context', state);

  return (
    <AuthContext.Provider value={{ ...state, dispatch }}>
      {children}
    </AuthContext.Provider>
  );
};

export default AuthProvider;

```

Рисунок 4.11 – Реалізація провайдера автентифікації

Функція `authReducer` управляє станом аутентифікації, обробляючи різні типи дій: ініціалізація, анонімна автентифікація, вхід, вихід, оновлення користувача та адрес. Вона повертає оновлений стан в залежності від дії. `AuthProvider` – це компонент, який використовує `useEffect` для відстеження стану автентифікації користувача у `Firebase`. При зміні стану автентифікації, він викликає відповідні дії для оновлення локального стану.

Коли користувач автентифікований, додаток отримує дані користувача з Firestore та оновлює стан за допомогою `dispatch`. Якщо користувач не автентифікований, виконується анонімна автентифікація. `AuthProvider` використовує контекст для надання стану аутентифікації та функції `dispatch` дочірнім компонентам, дозволяючи їм взаємодіяти з даними автентифікації.

Далі, на рисунках 4.12 – 4.13 представлена реалізація редуктора та провайдера для управління компонентом товарів в інтернет-магазині.

```
import { useReducer, useEffect } from 'react';

import { useParams, useLocation, useNavigate } from 'react-router-dom';

import { collection, getDocs, query, where, orderBy } from 'firebase/firestore';
import { db } from 'db/config';

import ProductContext from './product-context';

const initialState = {
  productIsReady: false,
  selectedProduct: null,
  selectedVariant: null,
  selectedSkuId: '',
  selectedSize: '',
  singleSize: null,
};

const productReducer = (state, action) => {
  const { type, payload } = action;

  switch (type) {
    case 'CLEAR_PRODUCT': {
      return {
        ...initialState,
      };
    }

    case 'SET_PRODUCT': {
      return {
        ...state,
        productIsReady: true,
        selectedProduct: payload.product,
        selectedVariant: payload.variant,
      };
    }
  }
}
```

Рисунок 4.12 – Реалізація редуктора управління товарами (ProductReducer)

```
const ProductProvider = ({ children }) => {
  const { id: slugId } = useParams();
  const { pathname, state: slugState } = useLocation();
  const navigate = useNavigate();

  const [state, dispatch] = useReducer(productReducer, initialState);

  const getProduct = async () => {
    try {
      const slugArr = slugId.split('-');
      const selectedColor = slugArr.pop();
      const formattedSlug = slugArr.join('-');

      const productsRef = collection(db, 'products');
      const productQuery = query(
        productsRef,
        where('slug', '==', formattedSlug)
      );

      const productsSnapshot = await getDocs(productQuery);

      const productDoc = productsSnapshot.docs[0];

      if (productDoc) {
        const variantsRef = collection(productDoc.ref, 'variants');
        const variantCheckQuery = query(
          variantsRef,
          where('color', '==', selectedColor)
        );

        const variantCheckSnapshot = await getDocs(variantCheckQuery);

        if (variantCheckSnapshot.size === 0) {
          return { product: null, variant: null };
        }

        const productData = {
          productId: productDoc.id,
          ...productDoc.data(),
        };

        const skusRef = collection(productDoc.ref, 'skus');
        const skusQuery = query(skusRef, orderBy('order'));
        const skusSnapshot = await getDocs(skusQuery);
      }
    } catch (error) {
      console.error('Error fetching product data:', error);
    }
  };

  return (
    <ProductContext.Provider value={{ state, dispatch, getProduct }}>
      {children}
    </ProductContext.Provider>
  );
}
```

Рисунок 4.13 – Реалізація провайдера управління товарами (ProductProvider)

ProductReducer, який представлений на рисунку 4.12, управляє станом продуктів. Він обробляє різні дії, такі як вибір продукту, варіанту, розміру, а

також очищення інформації про вибраний продукт. Компонент `ProductProvider` використовує `useReducer` для управління внутрішнім станом і `useEffect` для відстеження змін у URL. Коли URL змінюється, викликається функція `getProduct`, яка здійснює запити до Firestore для отримання інформації про продукт і його варіанти, засновані на параметрах URL. Функція `getProduct` розділяє `slugId`, отриманий з URL, на складові для визначення вибраного кольору та продукту. Після цього вона робить запити до Firestore для отримання докладної інформації про продукт та його варіанти. У коді також є логіка для обробки варіантів продукту. Якщо виявляється, що для вибраного варіанту існує лише один розмір, стан відповідно оновлюється. Для управління навігацією використовуються `useParams`, `useLocation` та `useNavigate` з `react-router-dom`. Ці хуки дозволяють відстежувати стан URL і управляти навігацією в додатку. Провайдер контексту огортає дочірні компоненти, надаючи їм доступ до управління станом продуктів у додатку.

На рисунку 4.14 представлений вигляд колекції товарів в інтернет-магазині, яка зберігається в сховищі Firebase.

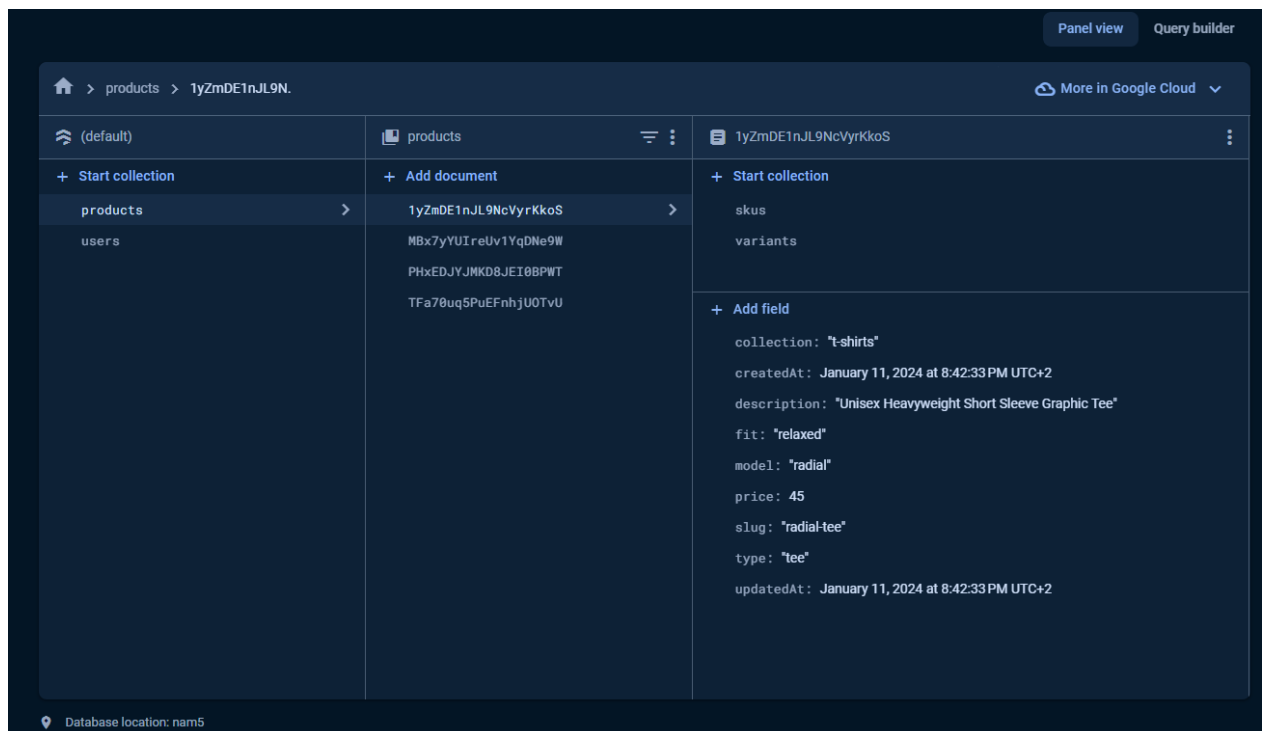


Рисунок 4.14 – Колекція товарів в сховищі Firebase

На рисунку 4.15 представлена реалізація хука React, який призначений для взаємодії з колекцією товарів у Firebase Firestore та обробки даних.

```
export const useCollection = () => {
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);
  const [hasMore, setHasMore] = useState(true);

  const productsRef = collection(db, 'products');

  const latestDoc = useRef();

  const getCollection = async ({
    collectionName = 'products',
    isNewQuery = true,
    sortBy = { field: 'createdAt', direction: 'asc' },
  }) => {
    setError(null);

    try {
      if (isNewQuery) {
        latestDoc.current = 0;
        setHasMore(true);
      }

      let productsQuery;

      let constraints = [orderBy(sortBy.field, sortBy.direction)];

      if (sortBy.field === 'createdAt') {
        constraints.unshift(orderBy('collection'));
      }

      if (sortBy.direction === 'desc' && !latestDoc.current) {
        constraints.push(limit(4));
      } else {
        constraints.push(
          startAfter(isNewQuery ? 0 : latestDoc.current),
          limit(4)
        );
      }

      if (collectionName === 'products') {
        productsQuery = query(productsRef, ...constraints);
      } else {
        productsQuery = query(
          productsRef,
          where('collection', '=', collectionName),
          ...constraints
        );
      }
    }
  }
}
```

Рисунок 4.15 – Реалізація хука для обробки ранжування колекції товарів

Хук `useCollection` використовує стани для управління завантаженням, помилками та перевіркою на наявність більше даних для завантаження. Функція `getCollection` всередині хука є асинхронною. Вона приймає об'єкт з

параметрами, такими як `collectionName`, `isNewQuery` та `sortBy`, для управління запитом до Firestore. Початково помилки обнуляються, а потім створюється запит до Firestore. Якщо це новий запит, він скидає останній документ для пагінації. Запити обмежуються чотирма елементами та сортуються за вказаними критеріями. Якщо обирається колекція продуктів, формується відповідний запит, інакше додається умова фільтрації за назвою колекції. Після отримання даних з Firestore, обробляється кожен документ продукту, створюючи детальний об'єкт продукту. Він також отримує дані про SKU та варіанти кожного продукту, формуючи відповідні об'єкти з детальною інформацією. Використовується допоміжна функція `formatDiscountNumber` для обчислення знижки на продукти. Повертає об'єкт з функцією `getCollection` та станами `isLoading`, `hasMore`, та `error`, які можуть використовуватися компонентами, що використовують цей хук, для управління рендерингом UI.

На рисунку 4.16 представлено сховище Firebase Storage, яке відповідає за збереження зображень товарів для їх відображення в інтернет-магазині.

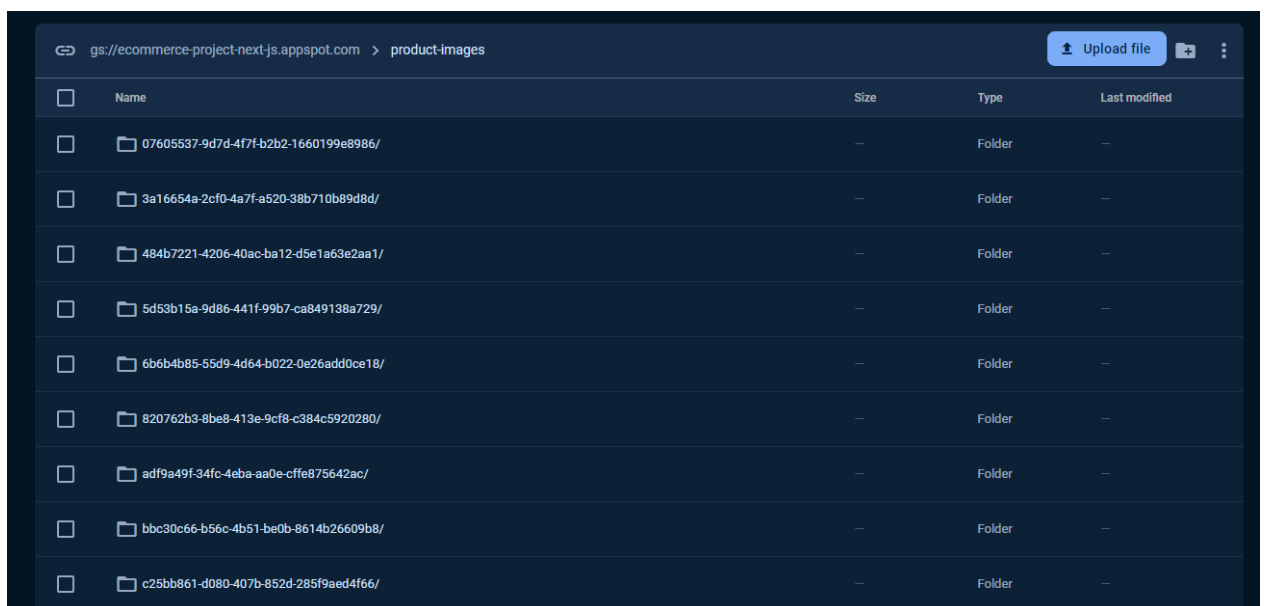


Рисунок 4.16 – Сховище зображень товарів з відповідними ідентифікаторами

4.3 Висновки за розділом 4

В четвертому розділі розглянуто основні етапи розробки програмної системи. Для розробки системи з використанням рекомендаційної системи використовуються наступні технології: мова програмування JavaScript, фреймворк для розробки компонентів React.js та сховище Firebase. Розглянуто лістинги реалізації компонентів програмної системи та алгоритмів фільтрації.

5 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

5.1 Огляд функціональності програмної системи

На рисунку 5.1 представлена головна сторінка, де реалізовано інтерактивний слайдер, який дозволяє переходити до секції товарів в магазині, а на рисунку 5.2 представлена секція з відображенням нових надходжень, Товари ранжуються відповідно даті створення за спаданням.

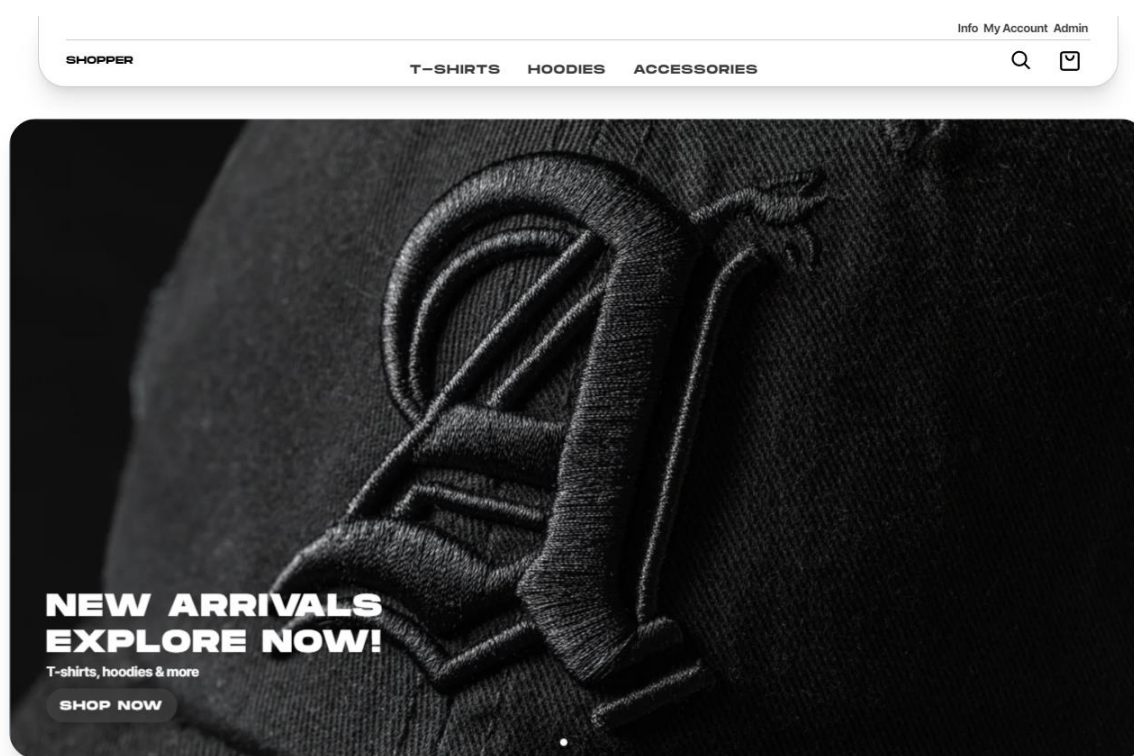


Рисунок 5.1 – Інтерактивний слайдер на головній сторінці

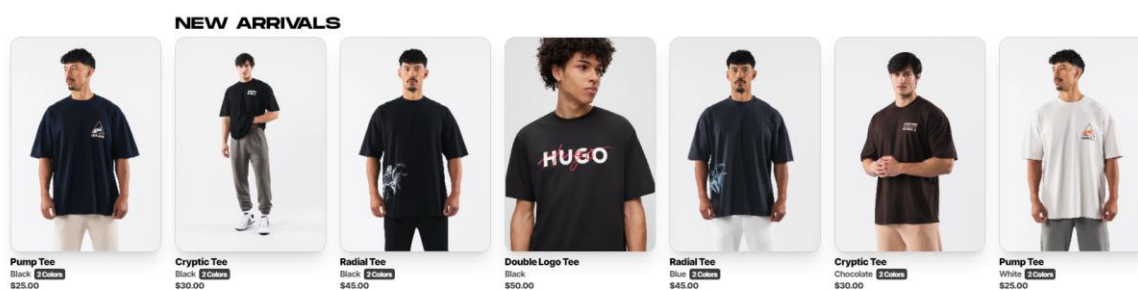


Рисунок 5.2 – Секція з відображенням нових надходжень товарів

На рисунку 5.3 представлена функціональність швидкого додавання вибраного товару в інтернет-магазині до корзини.

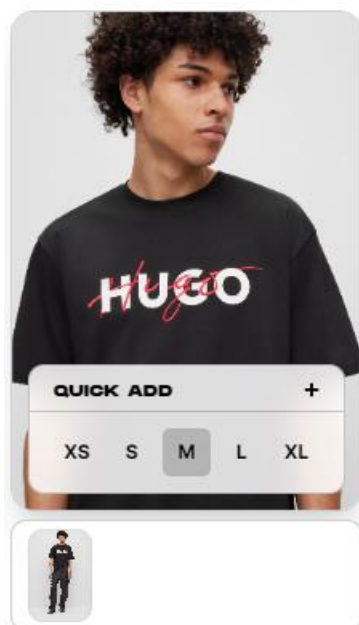


Рисунок 5.3 – Функціональність швидкого додавання товару до корзини

На рисунку 5.4 представлена модальна форма авторизації в системі.

A screenshot of a web application's login interface. At the top, there is a navigation bar with the text 'SHOPPER' on the left and 'Info Login' on the right. Below the navigation bar are three category links: 'T-SHIRTS', 'HOODIES', and 'ACCESSORIES'. The main content area is titled 'LOG INTO YOUR ACCOUNT'. It contains two input fields: 'Email:' with the placeholder text 'yourname@email.com' and 'Password:'. Below these fields is a large black button with the text 'LOGIN'. At the bottom of the form, there is a link that says 'New to Shopper? Create account'.

Рисунок 5.4 – Модальна форма авторизації в системі

На рисунку 5.5 представлена модальна форма реєстрації в системі.

CREATE ACCOUNT

Name:

Last Name:

Email:

Password:

CREATE ACCOUNT

Already have an account? [Login](#)

Рисунок 5.5 – Модальна форма реєстрації в системі

На рисунку 5.6 представлений вигляд особистого кабінету користувача

WELCOME BACK, ANDREW! [LOGOUT](#)

NO PLACED ORDERS YET

YOUR INFO

Andrew Miller
 andrewmiller@gmail.com
 No phone added yet

EDIT INFO

YOUR ADDRESSES

PRIMARY ADDRESS
Andrew Miller
Bankova street 50
Oregon, Michigan 61064

EDIT ADDRESSES

Рисунок 5.6 – Вигляд особистого кабінету користувача

На рисунках 5.7 – 5.8 зображено вигляд секції колекції товарів інтернет магазину, а також компонент налаштування фільтрації товарів за параметрами.

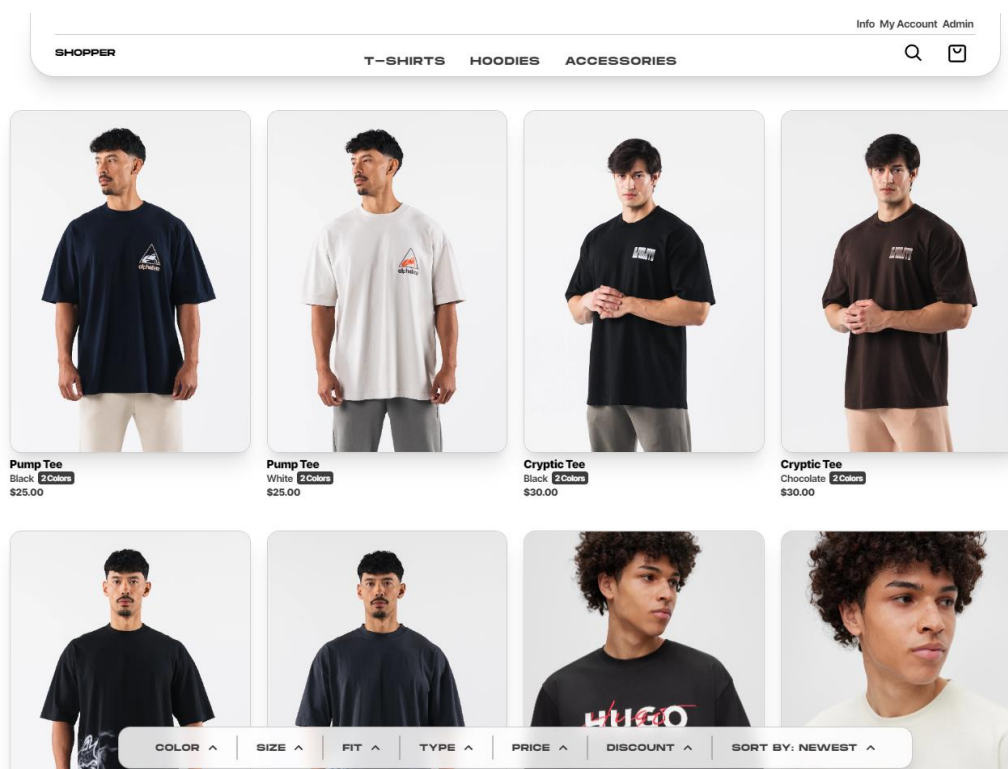


Рисунок 5.7 – Вигляд колекції товарів в інтернет-магазині (футболки)

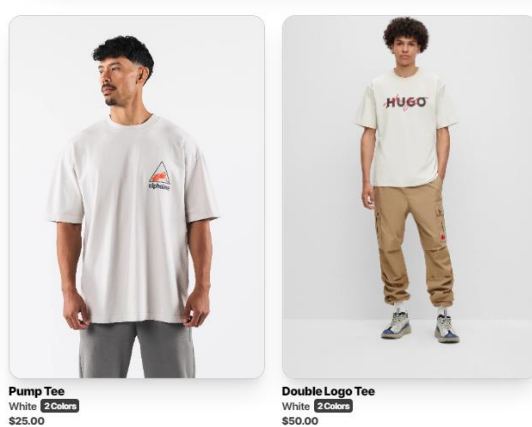


Рисунок 5.8 – Приклад фільтрації товарів за параметром кольору (білий)

На рисунках 5.9 – 5.10 зображено ранжування за спаданням ціни.

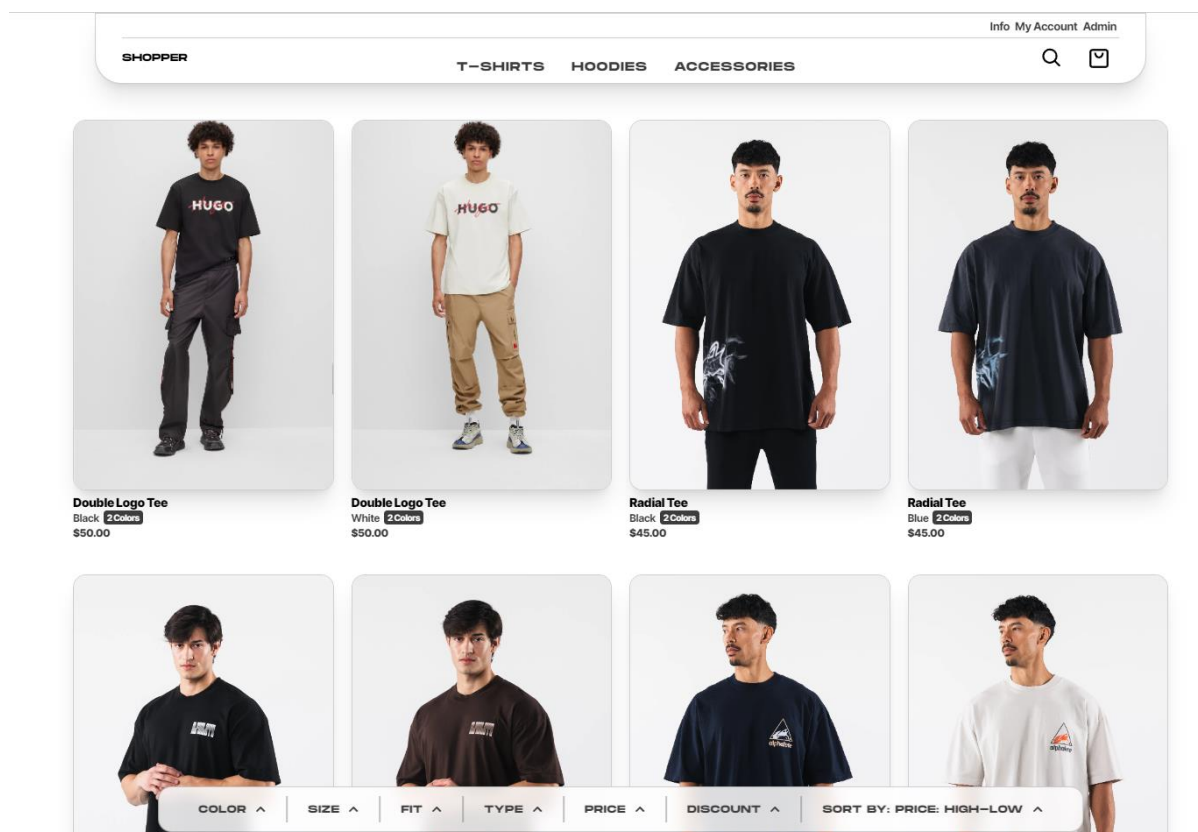


Рисунок 5.9 – Приклад ранжування за спаданням ціни

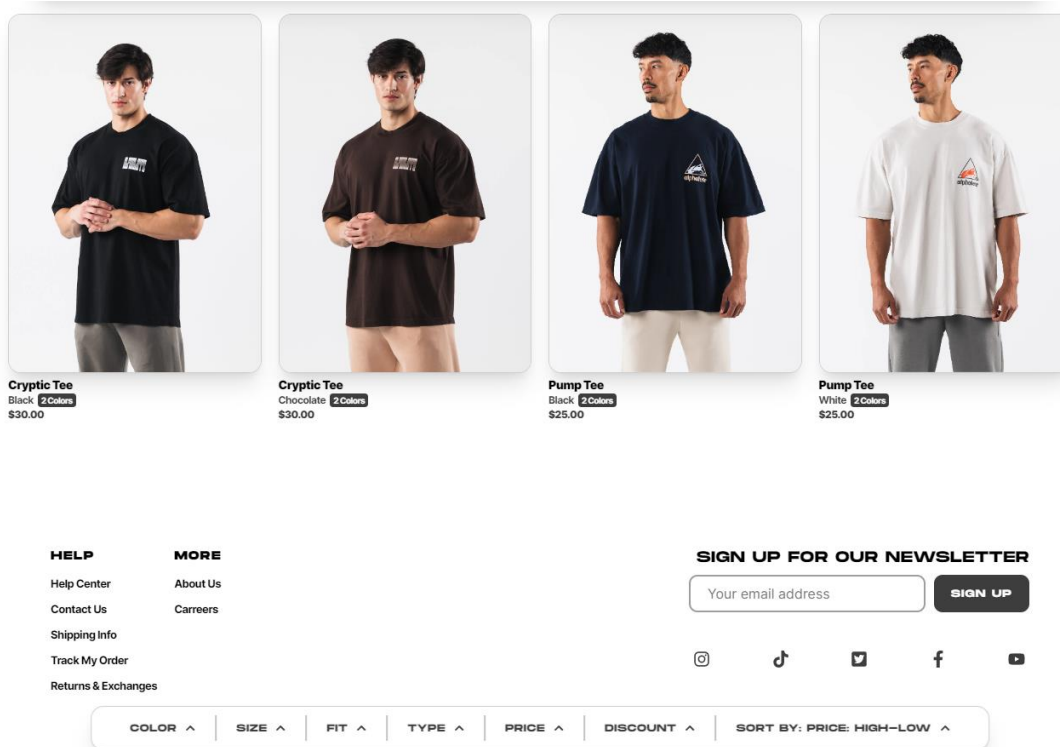


Рисунок 5.10 – Приклад ранжування за спаданням ціни (продовження)

На рисунку 5.11 представлено результат додавання товару до корзини.

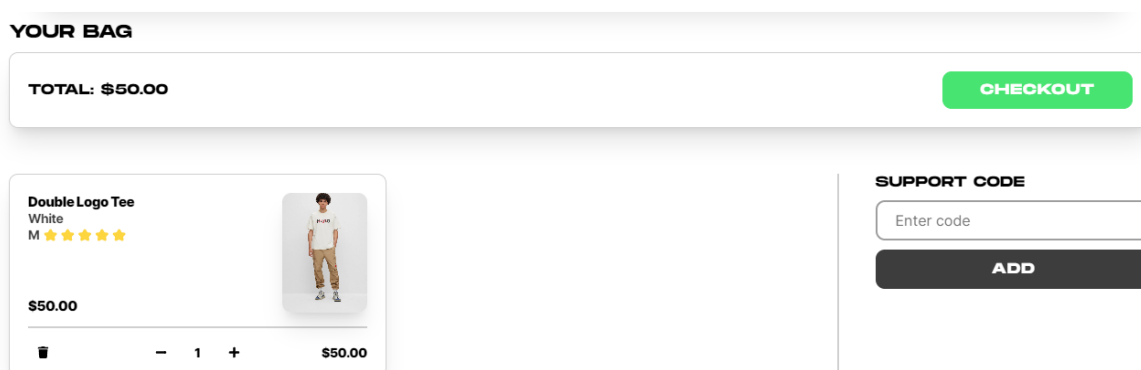


Рисунок 5.11 – Результат додавання товару до корзини (з оцінкою)

На рисунку 5.12 представлено результат надання рекомендацій користувачу, де використовується аналіз схожих характеристик (в даному випадку, основною характеристикою є одна колекція та схожі параметри товару), а також, на основі вподобань товарів (рейтингів) інших користувачів.

Рекомендовані товари завжди є унікальними та не мають дублювання.

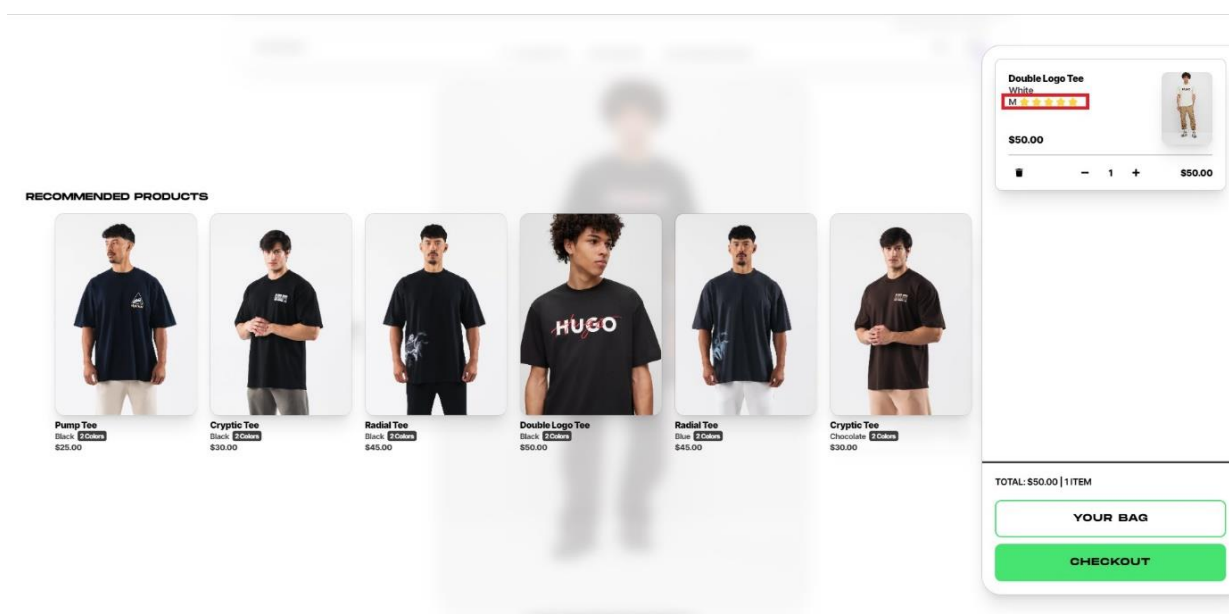


Рисунок 5.12 – Приклад ранжування товарів за коефіцієнтами подібності

5.2 Висновки за розділом 5

В п'ятому розділі протестовано розроблену програмну систему, яка має функціонал авторизації, реєстрації, додавання товарів до козиини, ранжування за фільтрами, а також отримання персоналізованих рекомендацій.

ВИСНОВКИ

На основі проведеного дослідження можна зробити наступні висновки.

В першому розділі проведено аналіз предметної області дослідження, а саме: визначено важливість персоналізації в інтернет-магазині, формалізовано математичну моделі рекомендаційної системи, систематизовано методи побудови персоналізованих рекомендацій, а також розглянуто існуючі системи електронної комерції з персоналізованими рекомендаціями. Класичні алгоритми можна розділити на дві основні групи: ті, що базуються на подібності, і ті, які використовують векторне представлення користувачів і об'єктів для виявлення шаблонів. Ці розширені алгоритми використовують повну історію взаємодії користувача та продукту для отримання стислих векторів, що представляють користувача та продукт. Одним із поширених типів алгоритмів, які обговорюються в цьому розділі, є матрична факторизація. Алгоритми матричної факторизації, такі як алгоритм найменших квадратів (ALS) і байєсівський персоналізований рейтинг (BPR), відіграють важливу роль у спільному створенні персоналізованих рекомендацій. Вони використовують методи, щоб розбити вихідну матрицю взаємодії між користувачем і продуктом на підматриці нижчої розмірності.

В другому розділі проведено аналіз систем рекомендацій, включаючи необхідні дані для їх побудови та потенційні проблеми, які можуть виникнути під час їх розробки. Також, проведено аналіз переваг та недоліків цих методів. У результаті було встановлено, що поєднання існуючих методів є найбільш сприятливим підходом для вирішення таких проблем, як проблема «холодного запуску», розрідженість даних і відсутність новизни. Важливо підкреслити, що для кожного продукту має бути створена спеціальна система рекомендацій, адаптована до конкретних цілей. В результаті, було систематизовано обраний метод і конкретні дані, які збиратимуться через застосунок. Крім того, проаналізовано основні показники, необхідні для оцінки продуктивності РС.

В третьому розділі проведено проектування архітектури інформаційної системи з використанням персоналізованого ранжування. Для розробки обрано компонентно-орієнтовану архітектуру, переваги якої включають: незалежність і можливість повторного використання кожного компонента в архітектурі на основі компонентів, що полегшує декомпозицію моноліту інтерфейсу на менші, більш керовані компоненти. Також, визначено основні вимоги користувача до системи з персоналізованими рекомендаціями.

В четвертому розділі розглянуто основні етапи розробки програмної системи. Для розробки системи з використанням рекомендаційної системи використовуються наступні технології: мова програмування JavaScript, фреймворк для розробки компонентів React.js та сховище Firebase. Розглянуто лістинги реалізації компонентів програмної системи та алгоритмів фільтрації.

В п'ятому розділі протестовано розроблену програмну систему, яка має функціонал авторизації, реєстрації, додавання товарів до кошика, ранжування за фільтрами, а також отримання персоналізованих рекомендацій.

Практична значимість полягає в тому, що розроблена система дозволяє користувачам переглядати товари за схожими характеристиками, враховуючи їхні вподобання та історію взаємодії. Це забезпечує зручний і швидкий доступ до відповідних товарів, покращуючи користувацький досвід. Для бізнесу система відкриває можливості підвищення продажів через покращення релевантності рекомендацій, зниження часу на пошук і створення лояльності.

Результати дослідження дали подальший розвиток рекомендаційним методам, зокрема алгоритмам матричної факторизації, таким як ALS і BPR. Розроблений метод поєднання різних підходів (контентна фільтрація та колаборативна фільтрація) дозволив вирішити проблеми "холодного старту", розрідженості даних та низької новизни рекомендацій. Вдосконалено алгоритми фільтрації, які враховують повну історію взаємодій користувача.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. How to Use Personalized Product Recommendations? [Electronic resource]. – Access mode: <https://abmatic.ai/personalized-recommendations>.
2. Personalized Product Recommendations. [Electronic resource]. – Access mode: <https://mailchimp.com/personalized-product-recommendations>.
3. Personalized recommender systems, ML and evaluation. [Electronic resource]. – Access mode: <https://www.btelligent.com/personalized-systems>.
4. An Ultimate Guide Into Personalized Recommendations & How To Use Them. [Electronic resource]. – Access mode: <https://www.banuba.com/blog/an-ultimate-guide-into-personalized-recommendations-how-to-use-them>.
5. How any business can benefit from personalization and recommendations. [Electronic resource]. – Access mode: <https://www.algolia.com/blog/product/how-any-business-can-benefit-from-personalization-and-recommendations>.
6. Personalized product recommendations and firm performance. [Electronic resource]. – Access mode: <https://pdf.sciencedirectassets.com/272634>.
7. What Is Collaborative Filtering: A Simple Introduction. [Electronic resource]. – Access mode: <https://builtin.com/science/collaborative-filtering>.
8. How Does Collaborative Filtering Work in Recommender Systems? [Electronic resource]. – Access mode: <https://www.turing.com/collaborative-filtering>.
9. Recommender Systems: Collaborative Filtering and Content-Based Filtering. [Electronic resource]. – Access mode: <https://ogre51.medium.com/recommender-systems-collaborative-filtering-and-content-based-filtering-20785d50d56f>.
10. Improved Collaborative Filtering Algorithm Based on Network Site Users. [Electronic resource]. – Access mode: <https://ieeexplore.ieee.org/document/9457342>.

11. Item-based Collaborative Filtering Algorithm Based on Group Weighted Rating. [Electronic resource]. – Access mode: <https://ieeexplore.ieee.org/document/9325767>.

12. What Content-Based Filtering is and Why You Should Use It. [Electronic resource]. – Access mode: <https://www.upwork.com/what-is-content-filtering>.

13. A Guide to Content-Based Filtering In Recommender Systems. [Electronic resource]. – Access mode: <https://www.turing.com/content-based-filtering>.

14. Recommendation Systems: Content-Based Filtering. [Electronic resource]. – Access mode: <https://medium.com/ml/content-based-filtering>.

15. Introduction to Content Based Recommendation. [Electronic resource]. – Access mode: <https://redfield.ai/content-based-recommendation>.

16. Collaborative Filtering VS Content-Based Filtering for Recommender Systems. [Electronic resource]. – Access mode: <https://analyticsindiamag.com/collaborative-filtering-vs-content-based-filtering-for-recommender-systems>.

17. Hybrid Recommender Systems: Combining Collaborative Filtering and Content-Based Filtering. [Electronic resource]. – Access mode: <https://aitechtrend.com/systems-combining-collaborative-filtering-and-content-based-filtering>.

18. Understanding the A10 Algorithm: Amazon Customer Experience as the Main Ranking Factor. [Electronic resource]. – Access mode: <https://www.sellerlabs.com/blog/understanding-amazon-a10-algorithm>.

19. Understanding Amazon's A9 Algorithm: How to Boost Your Rank. [Electronic resource]. – Access mode: <https://www.sellerapp.com/amazon-a9-algorithm>.

20. Amazon A9 Algorithm: SEO Secrets for Optimal Product Ranking. [Electronic resource]. – Access mode: <https://www.signalytics.ai/amazon-a9-algorithm-2>.

21. Understanding Component-Based Architecture. [Electronic resource]. – Access mode: <https://medium.com/component-based-architecture>.

22. What is Firebase? All the secrets unlocked. [Electronic resource]. – Access mode: <https://blog.back4app.com/firebase>.

23. React Architecture Pattern: Implementation + Best Practices. [Electronic resource]. – Access mode: <https://www.knowledgehut.com/react-architecture>.

ДОДАТОК А
Текст програми

A.1 Файл `item-based-recommender.js`

```

class ItemBasedRecommender {
  // транспонує матрицю
  transposeMatrix(matrix) {
    return matrix[0].map((col, c) => matrix.map((row, r) => matrix[r][c]));
  }

  // видаляє зазначений індекс з даного масиву
  removeIndex(arr, i) {
    return arr.slice(0, i).concat(arr.slice(i + 1));
  }

  // повертає n найближчих сусідів
  getNearestNeighbours(arr, n) {
    arr.sort((a, b) => b.correlation - a.correlation);
    return arr.filter((x) => x.correlation > 0).slice(0, n);
  }

  // повертає індекси масиву, що відповідають значенню
  getMatchingIndicies(arr, val) {
    var indicies = [];
    for (let i = 0; i < arr.length; i++)
      if (arr[i] == val)
        indicies.push(i);

    return indicies;
  }

  // повертає набори елементів, які оцінили і користувач a, і b
  getCommonSet(a, b) {
    let result = [];
    for (let i = 0; i < a.length; i++) {
      if (a[i] != -1 && b[i] != -1) result.push(i);
    }

    return result;
  }
}

```

```

// середнє значення масиву
mean(arr) {
  let filteredArr = arr.filter((x) => x !== -1);
  return filteredArr.reduce((a, b) => a + b) / filteredArr.length;
}

// обчислює середні значення для кожного користувача
getUserAverages(matrix) {
  let result = [];
  for (var i = 0; i < matrix.length; i++)
    result.push(this.mean(matrix[i]));
  return result;
}

// обчислює sim(a, b) і повертає масив об'єктів
getSimilarity(item, items, userAverages) {
  let similarities = [];

  for (let i = 0; i < items.length; i++) {
    let commonSet = this.getCommonSet(item.ratings, items[i].ratings);

    let numerator = 0;
    let denominator1 = 0;
    let denominator2 = 0;
    for (let j = 0; j < commonSet.length; j++) {
      numerator += (item.ratings[commonSet[j]] - userAverages[commonSet[j]]) *
        (items[i].ratings[commonSet[j]] - userAverages[commonSet[j]]);
      denominator1 += Math.pow((item.ratings[commonSet[j]] -
        userAverages[commonSet[j]]), 2);
      denominator2 += Math.pow((items[i].ratings[commonSet[j]] -
        userAverages[commonSet[j]]), 2);
    }

    let denominator = Math.sqrt(denominator1) * Math.sqrt(denominator2);
    let result = denominator !== 0 ? numerator / denominator : 0;
    similarities.push({ item: items[i].item, ratings: items[i].ratings, correlation:
      result });
  }
}

```

```

    return similarities;
}

// обчислює pred(a, p) і повертає масив з значеннями, заміненими
// передбачуваними значеннями
getPrediction(item, items, neighbourhoodSize) {
    let prediction = [...item.ratings];
    let unratedItems = this.getMatchingIndicies(prediction, -1);

    for (let i = 0; i < unratedItems.length; i++) {
        let numerator = 0;
        let denominator = 0;
        let nearestNeighbours = this.getNearestNeighbours(this.getSimilarity(item,
            items, this.getUserAverages(items)), neighbourhoodSize);
        for (let j = 0; j < nearestNeighbours.length; j++) {
            numerator += nearestNeighbours[j].correlation *
            nearestNeighbours[j].ratings[unratedItems[i]];
            denominator += nearestNeighbours[j].correlation;
        }

        prediction[unratedItems[i]] = denominator !== 0 ? (numerator / denominator) :
        -1;
    }

    return prediction;
}
}

export default ItemBasedRecommender;

```

A.2 Файл content-based-recommender.js

```

import { isNumber, isArray, has } from 'underscore';
import Vector from 'vector-object';
import striptags from 'striptags';
import { removeStopwords } from 'stopword';
import natural, { WordTokenizer } from 'natural';

```

```

const { TfIdf, PorterStemmer, NGrams } = natural;
const tokenizer = new WordTokenizer();

const defaultOptions = {
  maxVectorSize: 100,
  maxSimilarDocuments: Number.MAX_SAFE_INTEGER,
  minScore: 0,
  debug: false,
};

class ContentBasedRecommender {
  constructor(options = {}) {
    this.setOptions(options);

    this.data = {};
  }

  setOptions(options = {}) {
    // validation
    if ((options.maxVectorSize !== undefined) &&
      (!Number.isInteger(options.maxVectorSize) || options.maxVectorSize <= 0)) {
      throw new Error('Параметр maxVectorSize має бути цілим числом та бути
    більшим за 0');
    }

    if ((options.maxSimilarDocuments !== undefined) &&
      (!Number.isInteger(options.maxSimilarDocuments) || options.maxSimilarDocuments <= 0)) {
      throw new Error('Параметр maxSimilarDocuments має бути цілим числом та
    бути більшим за 0');
    }

    if ((options.minScore !== undefined) &&
      (!isNumber(options.minScore) || options.minScore < 0 || options.minScore > 1))
    {
      throw new Error('Параметр minScore має бути числом від 0 до 1');
    }

    this.options = Object.assign({}, defaultOptions, options);
  }
}

```

```

}

train(documents) {
  this.validateDocuments(documents);

  if (this.options.debug) {
    console.log(`Total documents: ${documents.length}`);
  }

  const preprocessDocs = this._preprocessDocuments(documents, this.options);

  const docVectors = this._produceWordVectors(preprocessDocs, this.options);

  this.data = this._calculateSimilarities(docVectors, this.options);
}

trainBidirectional(documents, targetDocuments) {
  this.validateDocuments(documents);
  this.validateDocuments(targetDocuments);

  if (this.options.debug) {
    console.log(`Total documents: ${documents.length}`);
  }

  const preprocessDocs = this._preprocessDocuments(documents, this.options);
  const preprocessTargetDocs = this._preprocessDocuments(targetDocuments,
this.options);

  const docVectors = this._produceWordVectors(preprocessDocs, this.options);
  const targetDocVectors = this._produceWordVectors(preprocessTargetDocs,
this.options);

  this.data = this._calculateSimilaritiesBetweenTwoVectors(docVectors,
targetDocVectors, this.options);
}

validateDocuments(documents) {
  if (!isArray(documents)) {
    throw new Error('Документи мають являти собою масив об'єктів');
  }
}

```

```

    }

    for (let i = 0; i < documents.length; i += 1) {
        const document = documents[i];

        if (!has(document, 'id') || !has(document, 'content')) {
            throw new Error('Документи повинні мати поля id і content');
        }

        if (has(document, 'tokens') || has(document, 'vector')) {
            throw new Error('Властивості "токенів" та "вектора" зарезервовані та не  
можуть використовуватися як властивості документа');
        }
    }
}

getSimilarDocuments(id, start = 0, size = undefined) {
    let similarDocuments = this.data[id];

    if (similarDocuments === undefined) {
        return [];
    }

    const end = (size !== undefined) ? start + size : undefined;
    similarDocuments = similarDocuments.slice(start, end);

    return similarDocuments;
}

export() {
    return {
        options: this.options,
        data: this.data,
    };
}

import(object) {
    const { options, data } = object;

```

```

    this.setOptions(options);
    this.data = data;
  }

  _preprocessDocuments(documents) {
    const processedDocuments = documents.map(item => {
      let tokens = this._getTokensFromString(item.content);
      return {
        id: item.id,
        tokens,
      };
    });

    return processedDocuments;
  }

  _getTokensFromString(string) {
    const tmpString = striptags(string, [], ' ')
      .toLowerCase();

    const tokens = tokenizer.tokenize(tmpString);

    const unigrams = removeStopwords(tokens)
      .map(unigram => PorterStemmer.stem(unigram));

    const bigrams = NGrams.bigrams(tokens)
      .filter(bigram =>
        (bigram.length === removeStopwords(bigram).length))
      .map(bigram =>
        bigram.map(token => PorterStemmer.stem(token))
          .join('_'));

    const trigrams = NGrams.trigrams(tokens)
      .filter(trigram =>
        (trigram.length === removeStopwords(trigram).length))
      .map(trigram =>
        trigram.map(token => PorterStemmer.stem(token))
          .join('_'));
  }

```

```

    return [].concat(unigrams, bigrams, trigrams);
}

_produceWordVectors(processedDocuments, options) {
    const tfidf = new TfIdf();

    processedDocuments.forEach((processedDocument) => {
        tfidf.addDocument(processedDocument.tokens);
    });

    const documentVectors = [];

    for (let i = 0; i < processedDocuments.length; i += 1) {
        if (options.debug) {
            console.log(`Створення вектора для документа: ${i}`);
        }

        const processedDocument = processedDocuments[i];
        const hash = {};

        const items = tfidf.listTerms(i);
        const maxSize = Math.min(options.maxVectorSize, items.length);
        for (let j = 0; j < maxSize; j += 1) {
            const item = items[j];
            hash[item.term] = item.tfidf;
        }

        const documentVector = {
            id: processedDocument.id,
            vector: new Vector(hash),
        };

        documentVectors.push(documentVector);
    }

    return documentVectors;
}

```

```

_calculateSimilaritiesBetweenTwoVectors(documentVectors,
targetDocumentVectors, options) {
  const data = {
    ...this.initializeDataHash(documentVectors),
    ...this.initializeDataHash(targetDocumentVectors)
  };

  for (let i = 0; i < documentVectors.length; i += 1) {
    if (options.debug) console.log(`Розрахунок оцінки подібності для документа:
${i}`);

    for (let j = 0; j < targetDocumentVectors.length; j += 1) {
      let documentVectorA = documentVectors[i];
      let targetDocumentVectorB = targetDocumentVectors[j];
      const idi = documentVectorA.id;
      const vi = documentVectorA.vector;
      const idj = targetDocumentVectorB.id;
      const vj = targetDocumentVectorB.vector;
      const similarity = vi.getCosineSimilarity(vj);

      if (similarity > options.minScore) {
        data[idi].push({
          id: targetDocumentVectorB.id,
          score: similarity
        });
        data[idj].push({
          id: documentVectorA.id,
          score: similarity
        });
      }
    }
  }

  this.orderDocuments(data, options);

  return data;
}

initializeDataHash(documentVectors) {

```

```

return documentVectors.reduce((acc, item) => {
  acc[item.id] = [];
  return acc;
}, {});
}

_calculateSimilarities(documentVectors, options) {
  const data = { ...this.initializeDataHash(documentVectors) };

  for (let i = 0; i < documentVectors.length; i += 1) {
    if (options.debug) console.log(`Calculating similarity score for document ${i}`);

    for (let j = 0; j < i; j += 1) {
      let documentVectorA = documentVectors[i];
      const idi = documentVectorA.id;
      const vi = documentVectorA.vector;
      let documentVectorB = documentVectors[j];
      const idj = documentVectorB.id;
      const vj = documentVectorB.vector;
      const similarity = vi.getCosineSimilarity(vj);

      if (similarity > options.minScore) {
        data[idi].push({
          id: documentVectorB.id,
          score: similarity
        });

        data[idj].push({
          id: documentVectorA.id,
          score: similarity
        });
      }
    }
  }

  this.orderDocuments(data, options);

  return data;
}

```

```

orderDocuments(data, options) {
  Object.keys(data)
    .forEach((id) => {
      data[id].sort((a, b) => b.score - a.score);

      if (data[id].length > options.maxSimilarDocuments) {
        data[id] = data[id].slice(0, options.maxSimilarDocuments);
      }
    });
}
}

```

```
export default ContentBasedRecommender;
```

A.3 Файл useAuth.jsx

```

import { useState } from 'react';

import {
  EmailAuthProvider,
  linkWithCredential,
  signInWithEmailAndPassword,
  signOut,
} from 'firebase/auth';
import { doc, getDoc, deleteDoc, setDoc } from 'firebase/firestore';

import { auth } from 'db/config';
import { db } from 'db/config';

import { useAuthContext } from './useAuthContext';
import { useCartContext } from './useCartContext';

import { handleError } from 'helpers/error/handleError';

export const useAuth = () => {
  const { user, dispatch: dispatchAuthAction } = useAuthContext();
  const { dispatch: dispatchCartAction } = useCartContext();

```

```

const [error, setError] = useState(null);
const [isLoading, setIsLoading] = useState(false);
const [defaultValue, setDefaultValue] = useState(false);

const signUp = async ({ name, lastName, email, password }) => {
  setError(null);
  setIsLoading(true);
  setDefaultValue({ name, lastName, email });

  try {
    const credential = EmailAuthProvider.credential(email, password);

    const userCredential = await linkWithCredential(
      auth.currentUser,
      credential
    );

    // if (!userCredential) {
    //   throw new Error('No se pudo crear la cuenta');
    // }

    const user = userCredential.user;

    const userData = {
      name,
      lastName,
      email,
      phoneNumber: null,
      addresses: [],
      isVerified: true,
    };

    await setDoc(doc(db, 'users', user.uid), userData);

    dispatchAuthAction({ type: 'LOGIN', payload: { user, ...userData } });
  } catch (err) {
    console.error(err);
    setError(handleError(err));
    setIsLoading(false);
  }
}

```

```

    }
  };

const login = async ({ email, password }) => {
  setError(null);
  setIsLoading(true);
  setDefaultValue({ email });

  try {
    dispatchCartAction({ type: 'IS_LOGIN' });
    const anonymousUser = user;

    const anonymousCartRef = doc(db, 'carts', anonymousUser.uid);
    const anonymousCartDoc = await getDoc(anonymousCartRef);

    await signInWithEmailAndPassword(auth, email, password);

    // if (!userCredential) {
    //   throw Error('Error');
    // }

    if (anonymousCartDoc.exists()) {
      deleteDoc(doc(db, 'carts', anonymousUser.uid));
    }
  } catch (err) {
    console.error(err);
    setError(handleError(err));
    dispatchCartAction({ type: 'IS_NOT_LOGIN' });
    setIsLoading(false);
  }
};

const logout = async () => {
  setError(null);
  setIsLoading(true);
  try {
    await signOut(auth);
    dispatchCartAction({ type: 'DELETE_CART' });
    dispatchAuthAction({ type: 'LOGOUT' });
  }
};

```

```

    } catch (err) {
      console.error(err);
      setError(handleError(err));
      setIsLoading(false);
    }
  };

  return { signUp, login, logout, isLoading, error, defaultValue };
};

```

A.4 Файл useCollection.jsx

```

import { useState, useRef } from 'react';
import { v4 as uuid } from 'uuid';

import {
  collection,
  getDocs,
  query,
  where,
  orderBy,
  startAfter,
  limit,
} from 'firebase/firestore';

import { db } from 'db/config';

import { formatDiscountNumber } from 'helpers/format';

export const useCollection = () => {
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);
  const [hasMore, setHasMore] = useState(true);

  const productsRef = collection(db, 'products');

  const latestDoc = useRef();

```

```

const getCollection = async ({
  collectionName = 'products',
  isNewQuery = true,
  sortBy = { field: 'createdAt', direction: 'asc' },
}) => {
  setError(null);

  try {
    if (isNewQuery) {
      latestDoc.current = 0;
      setHasMore(true);
    }

    let productsQuery;

    let constraints = [orderBy(sortBy.field, sortBy.direction)];

    if (sortBy.field === 'createdAt') {
      constraints.unshift(orderBy('collection'));
    }

    if (sortBy.direction === 'desc' && !latestDoc.current) {
      constraints.push(limit(4));
    } else {
      constraints.push(
        startAfter(isNewQuery ? 0 : latestDoc.current),
        limit(4)
      );
    }

    if (collectionName === 'products') {
      productsQuery = query(productsRef, ...constraints);
    } else {
      productsQuery = query(
        productsRef,
        where('collection', '==', collectionName),
        ...constraints
      );
    }
  }
}

```

```

const productsSnapshot = await getDocs(productsQuery);

if (productsSnapshot.size === 0) {
  setHasMore(false);
  setIsLoading(false);
  return [];
}

setIsLoading(true);

latestDoc.current =
  productsSnapshot.docs[productsSnapshot.docs.length - 1];

const productsPromises = productsSnapshot.docs.map(async (productDoc) => {
  const productData = {
    productId: productDoc.id,
    ...productDoc.data(),
  };

  const skusRef = collection(productDoc.ref, 'skus');

  const skusQuery = query(skusRef, orderBy('order'));

  const skusSnapshot = await getDocs(skusQuery);

  const skus = [];

  skusSnapshot.forEach((skuDoc) =>
    skus.push({
      skuId: skuDoc.id,
      ...skuDoc.data(),
    })
  );

  const variantsRef = collection(productDoc.ref, 'variants');

  const variantsSnapshot = await getDocs(variantsRef);

```

```

const productVariants = [];

variantsSnapshot.forEach((variantDoc) => {
  let variantSkus = skus
    .filter((sku) => sku.variantId === variantDoc.id)
    .map((sku) => ({
      size: sku.size,
      skuId: sku.skuId,
      quantity: sku.quantity,
    }));

  let availableQuantity = variantSkus.reduce((result, obj) => {
    if (!obj.size) {
      result['singleSize'] = obj.quantity;
    } else {
      result[obj.size] = obj.quantity;
    }
    return result;
  }, {});

  const sizes = Object.keys(availableQuantity);

  const isSoldOut = variantSkus.every((sku) => sku.quantity === 0);

  const { price: actualPrice, ...restProductData } = productData;
  const {
    variantPrice: currentPrice,
    images: variantImages,
    ...restVariantData
  } = variantDoc.data();

  const formattedVariantImages = variantImages.map((image) => ({
    ...image,
    url: `${restProductData.slug}-${restVariantData.color}`,
  }));

  productVariants.push({
    variantId: variantDoc.id,
    price: currentPrice,
  });
});

```

```

    actualPrice,
    ...restProductData,
    ...restVariantData,
    slides: formattedVariantImages,
    numberOfVariants: variantsSnapshot.size,
    availableQuantity,
    sizes,
    skus: variantSkus,
    discount: formatDiscountNumber({
      currentPrice,
      actualPrice,
    }),
    isSoldOut,
  });
});

```

```

const formattedProductVariants = productVariants.map((variant) => ({
  ...variant,
  id: uuid(),
  allVariants: productVariants,
}));

```

```

  return formattedProductVariants;
});

```

```

const products = await Promise.all(productsPromises);

```

```

  setIsLoading(false);
  return [].concat(...products);
} catch (err) {
  console.error(err);
  setError(err);
  setIsLoading(false);
}
};

```

```

return { getCollection, isLoading, hasMore, error };
};

```

ДОДАТОК Б
Слайди презентації

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Кафедра програмних засобів

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему:

“Дослідження та програмна реалізація методу ранжування товарів в інтернет-магазині”

Виконав: студент гр. КНТ 133-м
Ігор ЛИСЕНКО

Керівник: доцент, к.т.н. Ольга ГЛАДКОВА

Запоріжжя - 2024

1

Рисунок Б. 1 – Слайд 1

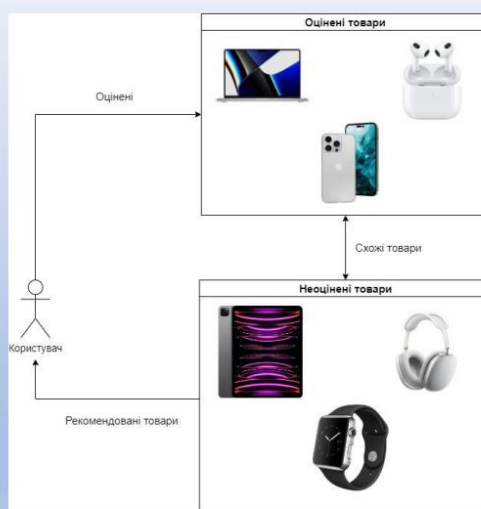
ФОРМАЛІЗАЦІЯ ПОСТАНОВКИ ЗАДАЧІ

- **Актуальність теми:** Сучасна постіндустріальна епоха відрізняється від своїх попередників всепроникними процесами глобалізації та цифровізації. Поява глобальних ринків не тільки підвищила рівень життя в усьому світі, але й спричинила безпрецедентний сплеск виробництва та споживання товарів і послуг. Системи ранжування передбачають створення пропозицій щодо товарів, послуг або інших елементів для користувачів, використовуючи або переваги користувача, або переваги подібних користувачів щодо їхніх покупок. У зв'язку з широким розповсюдженням різних типів рекомендаційних систем товарів тема має значну актуальність.
- **Об'єкт дослідження:** системи ранжування товарів в інтернет-магазині.
- **Предмет дослідження:** методи та алгоритми ранжування товарів.
- **Мета дослідження:** розробка та впровадження алгоритму ранжування товарів для інтернет-магазинів.

2

Рисунок Б. 2 – Слайд 2

АЛГОРИТМ КОНТЕНТНОЇ ФІЛЬТРАЦІЇ



3

Рисунок Б. 3 – Слайд 3

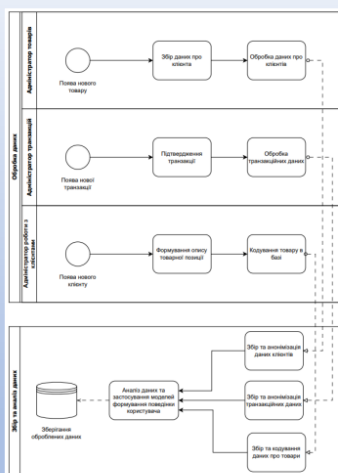
ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ ФОРМУВАННЯ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ

Критерій	Колаборативна фільтрація	Контентна фільтрація	Гібридна фільтрація
Основна ідея	Використовує дані про поведінку та вподобання інших користувачів для рекомендацій.	Використовує характеристики об'єктів (фільмів, книг товарів) для рекомендацій.	Комбінує обидва підходи для підвищення точності рекомендацій.
Переваги	Ефективний для великих датасетів, добре виявляє складні шаблони. Не вимагає розуміння або аналізу самого контенту.	Не потребує даних про інших користувачів, ефективний для нових об'єктів. Може давати дуже специфічні рекомендації на основі характеристик об'єкта.	Може бути налаштована для використання в різних пропорціях колаборативної та контентної інформації. Комбінує переваги обох підходів, зменшуючи їх недоліки.
Недоліки	Проблема "холодного старту". Якщо інформація про поведінку користувачів обмежена, результати можуть бути неточними.	Схильна до рекомендації об'єктів, схожих на ті, що вже оцінені. Може ігнорувати корисність об'єкта в різних контекстах.	Інтеграція різних систем і даних може бути технічно складною. Для ефективності часто потрібні обидва типи даних у великій кількості.

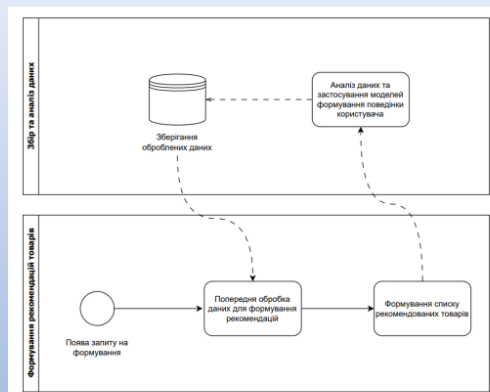
4

Рисунок Б. 4 – Слайд 4

МОДЕЛЮВАННЯ БІЗНЕС-ПРОЦЕСІВ В СИСТЕМІ



Діаграма бізнес-процесів обробки даних системи з РС

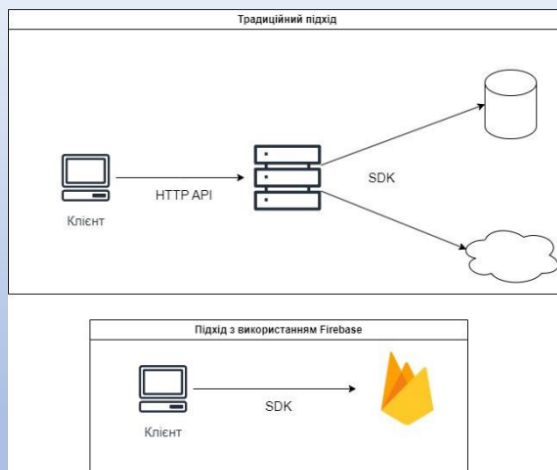


Взаємозв'язок бізнес-процесів "Збір та аналіз даних" та "Формування рекомендацій товарів"

5

Рисунок Б. 5 – Слайд 5

ПІДХІД РОЗРОБКИ ДОДАТКІВ З ВИКОРИСТАННЯМ FIREBASE

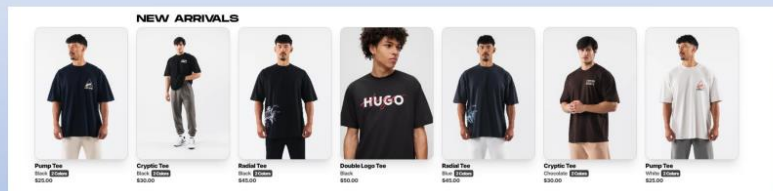


Порівняння традиційного підходу до розробки веб-додатків та підходу з використанням Firebase

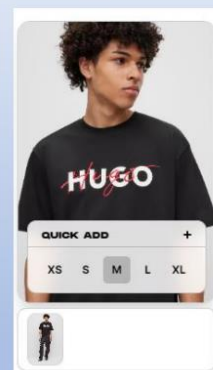
6

Рисунок Б. 6 – Слайд 6

ТЕСТУВАННЯ ФУНКЦІОНАЛЬНОСТІ ПРОГРАМНОЇ СИСТЕМИ



Слайдер для перегляду актуальних товарів

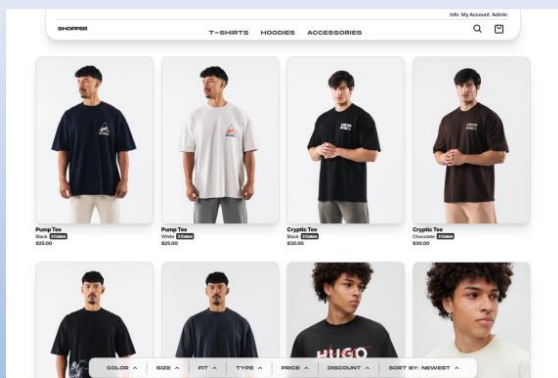


Функціональність швидкого додавання до кошика

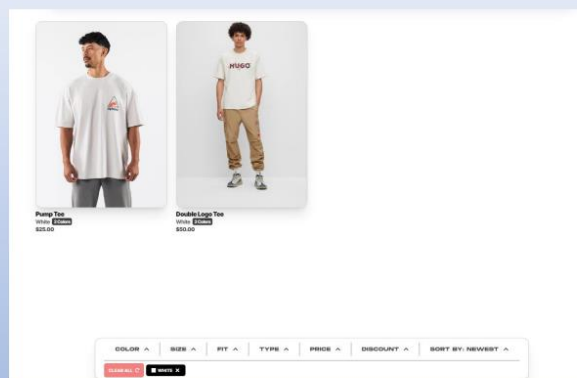
9

Рисунок Б. 9 – Слайд 9

ТЕСТУВАННЯ ФУНКЦІОНАЛЬНОСТІ ПРОГРАМНОЇ СИСТЕМИ



Сторінка колекції товарів

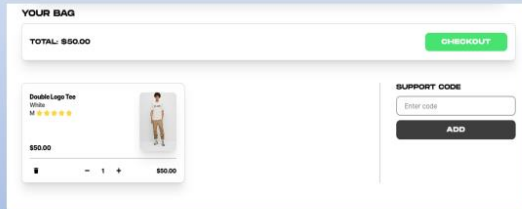


Ранжування товарів за фільтрацією кольорів

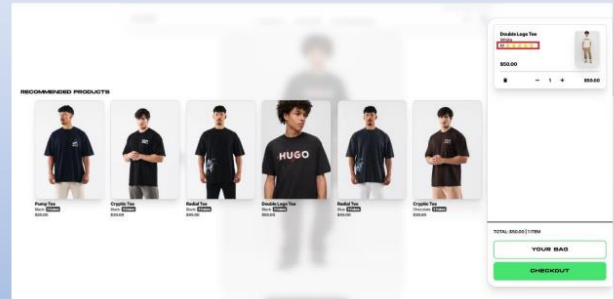
10

Рисунок Б. 10 – Слайд 10

ТЕСТУВАННЯ ФУНКЦІОНАЛЬНОСТІ ПРОГРАМНОЇ СИСТЕМИ



Додавання товару до корзини (з оцінкою)



Ранжування товарів за коефіцієнтами подібності

11

Рисунок Б. 11 – Слайд 11

ВИСНОВКИ

- В ході роботи проведено аналіз предметної області, а саме: визначено важливість персоналізації в інтернет-магазині, формалізовано математичну модель рекомендаційної системи, існуючі системи електронної комерції з персоналізованими рекомендаціями, а також сформовано ряд вимог до системи ранжування товарів.
- Проведено проектування архітектури інформаційної системи з використанням персоналізованого ранжування. Для розробки обрано компонентно-орієнтовану архітектуру, переваги якої включають: незалежність і можливість повторного використання кожного компонента в архітектурі на основі компонентів, що полегшує декомпозицію моноліту інтерфейсу на менші, більш керовані компоненти. Визначено вимоги користувача до системи з РС.
- Для розробки системи з використанням рекомендаційної системи використовуються наступні технології: мова програмування JavaScript, фреймворк для розробки компонентів React.js та сховище даних Firebase.
- В результаті, отримав подальший розвиток гібридний алгоритм формування персоналізованих рекомендацій, який використовує колаборативну та контентну фільтрації. Розроблений алгоритм здатний оцінювати послідовність дій користувача в інтернет-магазині, об'єктів взаємодії та подібні елементи до товарних позицій, в яких зацікавлений користувач.
- Практична значимість полягає в тому, що розроблена система дозволяє користувачам переглядати товари за схожими характеристиками, враховуючи їхні вподобання та історію взаємодії.

12

Рисунок Б. 12 – Слайд 12