Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ДОСЛІДЖЕННЯ МЕТОДІВ РОЗРОБКИ ОНЛАЙН-МАГАЗИНІВ ТА РЕКОМЕНДАЦІЙНИХ СИСТЕМ

Текстова частина до кваліфікаційної роботи за спеціальністю „Комп'ютерні науки"

Керівник кваліфікаційної роботи
Ас. Курочкін А.В.
_____*(підпис)*
"__"_____2023 р.
Виконав студент
Кириченко Є.Б.
"__"_____2023 р.

Київ 2023

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри

Гороховський С.С.

(підпис) _____

«___»_____ 2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломну роботу

Студента Кириченка Євгенія Борисовича факультету інформатики 4 курсу

ТЕМА: ДОСЛІДЖЕННЯ МЕТОДІВ РОЗРОБКИ ОНЛАЙН-МАГАЗИНІВ ТА РЕКОМЕНДАЦІЙНИХ СИСТЕМ

Вихідні дані: Сервіс для перегляду аніме-товарів та оформлення замовлення, система рекомендацій для користувача.

Індивідуальне завдання:

Вступ

1.Аналіз предметної області. Постановка завдання дипломної роботи

2.Теоритичні відомості про розробку веб-застосунків

3.Реалізація програмного продукту

4. Теоретичні відомості про розробку рекомендаційних систем

5. Реалізація рекомендаційної системи

Висновки

Список використаної літератури

Додатки

Дата видачі «___» _____ 2023р.

Керівник _____Завдання отримав_____

(підпис) (підпис)

Календарний план виконання роботи:

Номер	Назва етапу	Термін виконання	Примітка
1.	Отримання теми кваліфікаційної роботи	12.12.2022	
2.	Огляд літератури за темою роботи	12.01.2023	
3.	Аналіз актуальності питання	19.01.2023	
4.	Огляд аналогів магазинів та рекомендаційних систем	27.01.2023	
5.	Постановка завдання	01.02.2023	
6.	Аналіз технологій розробки	11.02.2023	
7.	Аналіз технічого завдання	21.02.2023	
8.	Вибір технологій розробки	25.02.2023	
9.	Створення моделі даних	01.03.2023	
10.	Створення схем БД	07.03.2023	
11.	Розробка серверної частини	17.03.2023	
12.	Розробка клієнтської частини	07.04.2023	
13.	Поєдання клієнта та сервера	17.04.2023	

14.	Огляд рекомендаційних систем	01.05.2023	
15.	Вибір системи для застосунку	07.05.2023	
16.	Збирання даних для системи	08.05.2023	
17.	Реалізація системи	18.05.2023	
18.	Перегляд змісту роботи керівником	19.05.2023	
19.	Внесення змін та правок відповідно до зауважень керівника	21.05.2023	

Студент Кириченко Є.Б.

Керівник Курочкін А.В.

«___»_____

ЗМІСТ

АНОТАЦІЯ

В роботі досліджуються онлайн-магазини аніме-товарів та рекомендаційні системи різного типу. Розглянуто актуальність поставленої задачі. Розглянуто плюси і мінуси різних веб-застосунків та поставлено задачу розробити власний веб-застосунок для продажу аніме-товарів та рекомендаційної системи для нього. Для цього розглянуті переваги та недоліки різних систем. В теоретичній частині про веб-додатки досліджуються різні технології та принципи розробки застосунків, а в теоретичній частині про рекомендаційні системи розглянуто різні їх види та варіанти реалізації. В практичній частині в розділі про веб-додаток описано його реалізацію, а в практичній частині до рекомендаційних систем відповідно реалізацію власної системи та її взаємодію з додатком.

ВСТУП

Зараз майже у всіх людей на планеті є інтернет, і всі люди здійснюють покупки. Щоб зекономити час на пошуки товарів, можна замовити їх не виходячи з дому - в інтернет магазині. Так можна швидко підібрати товар з потрібними характеристиками і адекватною ціною. Ціль онлайн магазину - заробіток шляхом задоволення потреб клієнтів. Кожен з нас має потреби, і кожен з нас - клієнт. Відповідно до цього створення онлайн-магазинів несе суспільну користь.

В сучасному світі на просторах інтернету існує безліч сайтів, веб-додатків, онлайн магазинів. Розробляючи свій додаток, варто подумати про те, як привернути увагу користувачів. Крім безперебійної роботи та хорошого інтерфейсу потрібна взаємодія з користувачем. Одним з видів такої комунікації є система рекомендацій. Клієнт отримує нові пропозиції залежно від його вподобань, досвіду, статусу. Персоналізовані рекомендації можуть втримати клієнта на сайті. Крім того, хороші рекомендації значно збільшують продажі та змушують клієнта задуматись над покупкою товару, про який він раніше і не думав.

Метою роботи є створення веб-додатку для продажу аніме - товарів, який має приємний інтерфейс та поліпшення взаємодії користувача з додатком і підвищення продажів шляхом створення персоналізованої рекомендаційної системи.

Завданням є розробка додатку і системи рекомендацій для нього.

Об’єкт дослідження - додатки та рекомендаційні системи, що існують.

Інтернет магазин буде розроблений з використанням MERN stack - сервер Express, код бекенду буде написаний з використанням Node.js, документоорієнтована база даних MongoDB та React на фронтенді для розробки інтерфейсу. Рекомендаційна система буде розроблена на python з використанням бібліотек numpy, sklearn.

Структура курсової роботи

Робота складається з п’яти розділів.

Перший розділ

- вивчення актуальності розробки аніме-магазинів та рекомендаційних систем, аналіз аналогів та постановка завдання.

Другий розділ

- архітектури веб додатків та технології для їх розробки.
- порівняння різних підходів та технологічних рішень.

Третій розділ

- формування технічного завдання для додатку
- огляд технологій розробки
- план виконання завдань
- опис реалізації додатку: архітектура фронтенду та бекенду, класи, база даних.

Четвертий розділ

- огляд та пояснення роботи різних видів рекомендаційних систем
- визначення переваг та недоліків різних видів та підвидів систем

П’ятий розділ

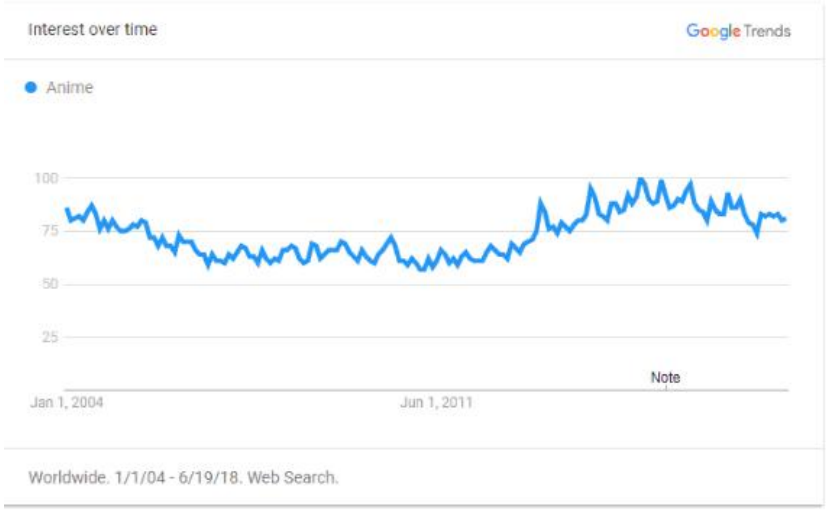
- обгрунтування вибору системи
- реалізація системи
- демонстрація роботи системи

РОЗДІЛ 1: ВИВЧЕННЯ НІШИ

1.1 Аналіз актуальності питання

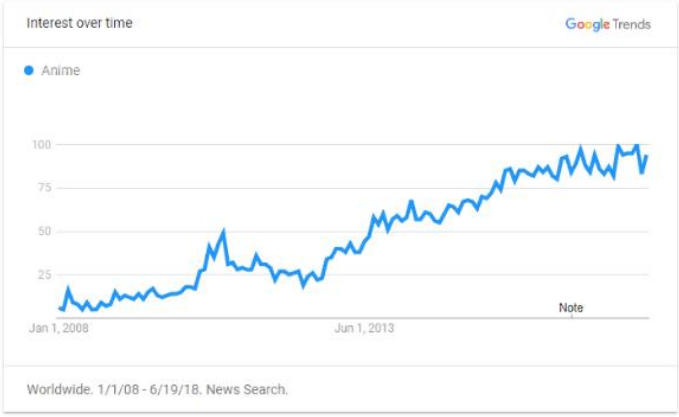
Ні для кого не секрет, що аніме шалено популярне у всьому світі. В гуглі кожної секунди користувачі здійснюють 40 000 запитів “Anime”, що означає більше трильйону запитів аніме на рік.

Anime (search term) via Google

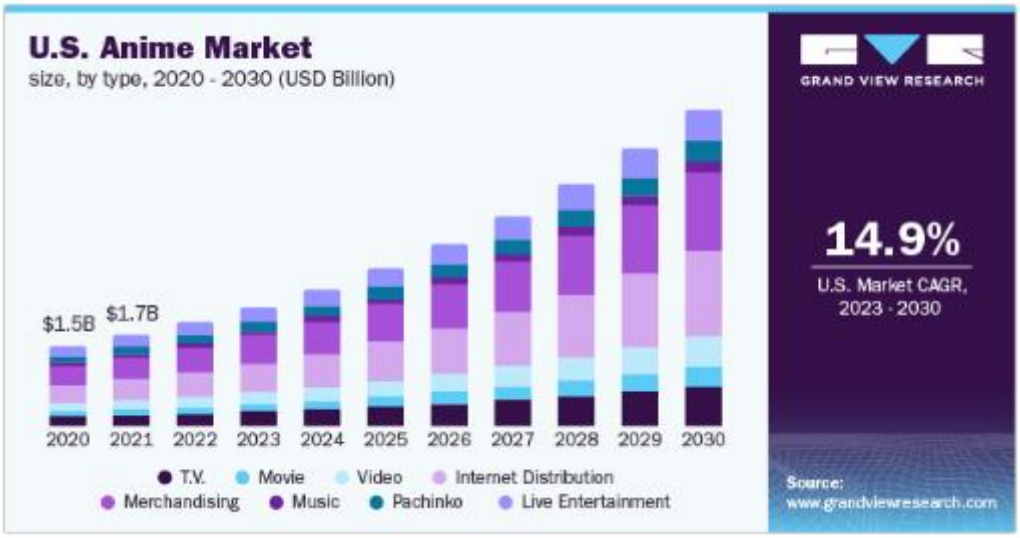


Новини пов’язані з аніме взагалі набирають шаленого росту. Зріст іде роками, не просідаючи.

. Anime (Topic + search term) via Google News

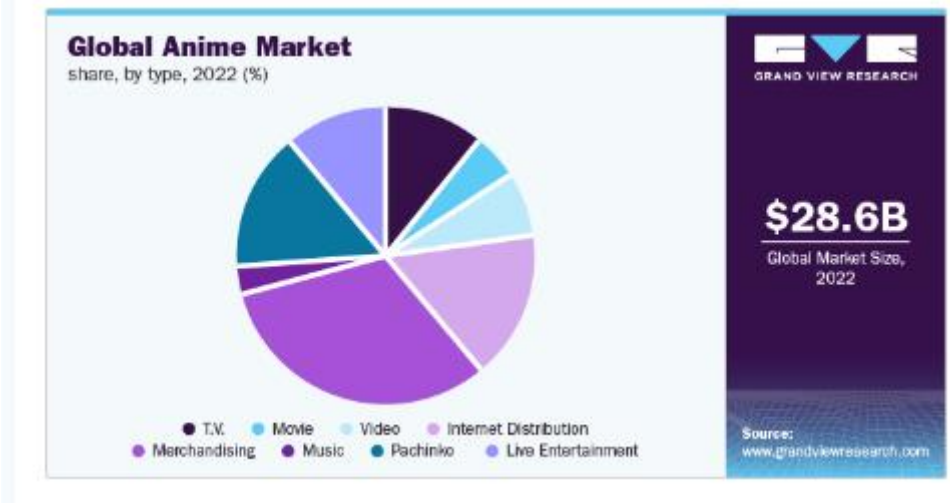


[1] Розмір глобального ринку аніме у 2022 році - 28.61 мільярдів. Очікується щорічний ріст на 9.8% з 2023 до 2030



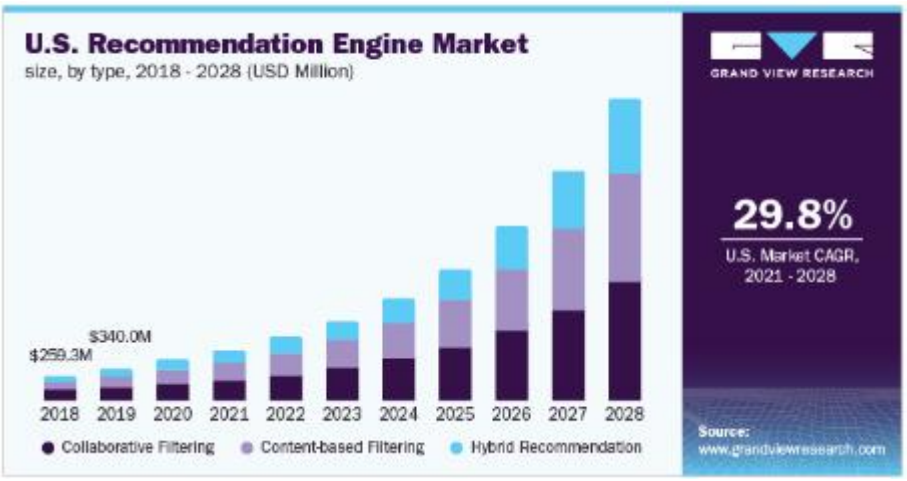
року.

На графіку видно, що за прогнозами мерчендайзинг зросте так само, як і всі інші аспекти ринку аніме. Більше того, у 2022 році сегмент мерчендайзингу склав найвищу частку ринку - 31 %. Експерти очікують, що даний сегмент буде домінувати найближчим часом



Отже, ми розібрали актуальність побудови магазину товарів, пов’язаних з аніме. Тож перейдімо до рекомендаційних систем.

[2] Розмір ринку рекомендаційних систем за 2020 рік оцінювався у 1.77 мільярдів доларів. Експерти прогнозували щорічний зріст цієї цифри на неймовірні 33% з 2021 до 2028 року. Росте попит на створення персонального досвіду для користувача.



За даними Adobe Research 2021, приблизно 34% користувачів куплятимуть більше товарів, якщо отримуватимуть рекомендації на основі попередніх покупок.

Сегмент collaborative filtering приніс найбільше доходу серед рекомендаційних систем у 2020 році - 40%, очікується що даний тип систем збереже лідерство принаймні до 2028. Очікується, що доходи від гібридних систем підвищуватимуться на 34,4% до 2028.

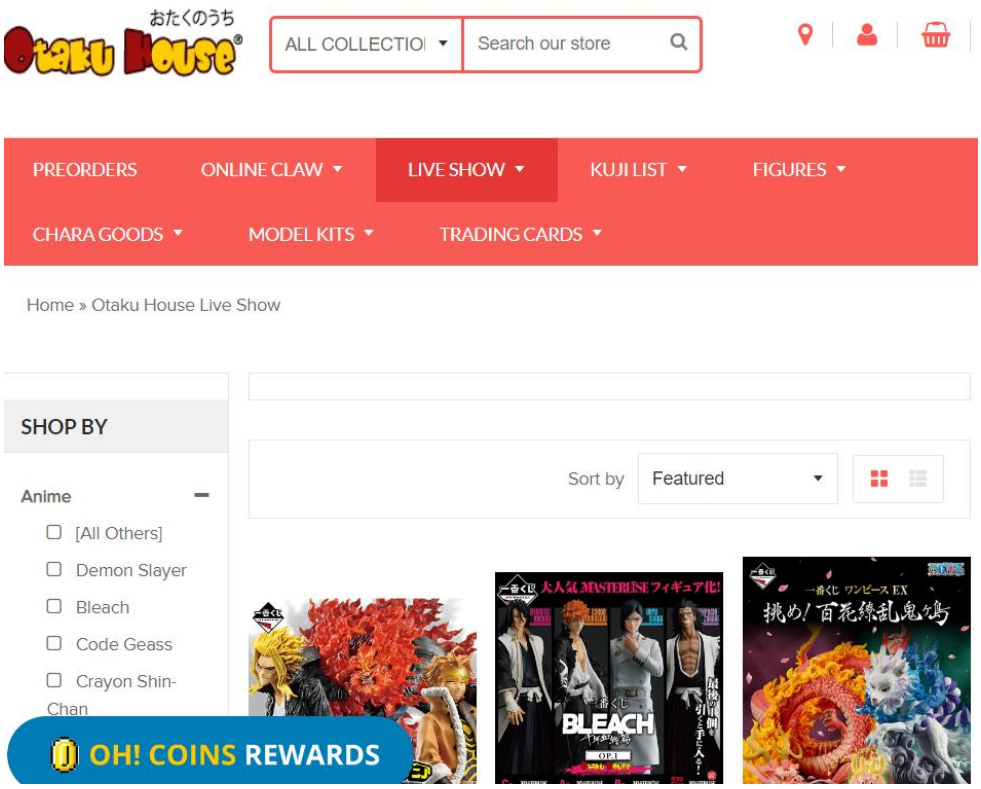
Отже, актуальність написання роботи розглянуто.

1.2 Аналіз рішень на ринку

Почнемо з магазинів аніме товарів.

Розглянемо один з перших, який випадає при prompt’і “buy anime merch”.

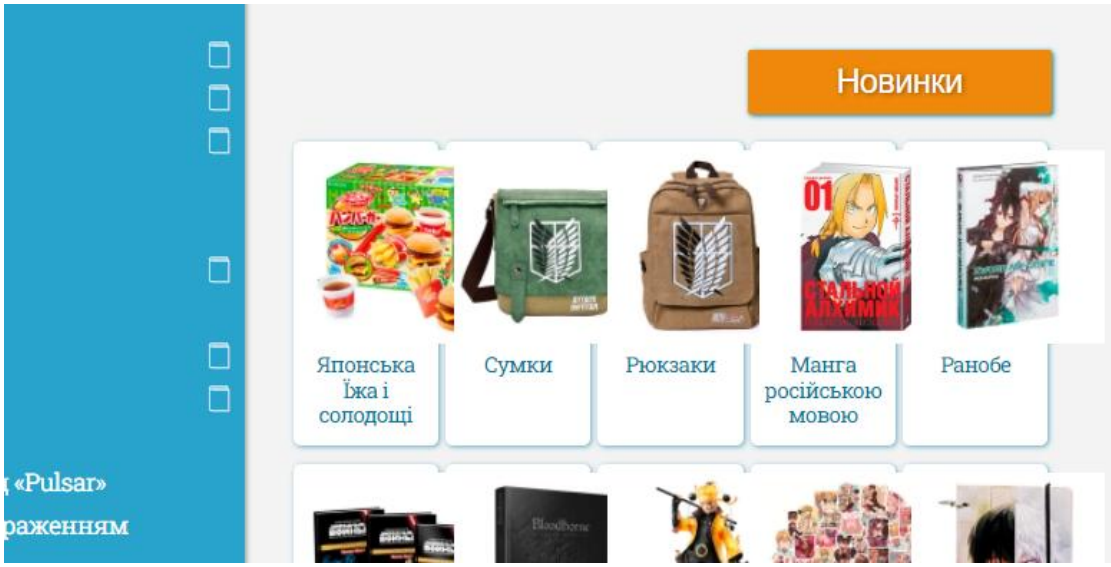
- Otaku House



На сайті є багато різних категорій - речі, постери, пазли ітд , фільтрів - за аніме, найкращі продажі, за датою додавання, алфавітний порядок. Є кабінет користувача, кошик. Є можливість оформити замовлення навіть без реєстрації. Основні важливі компоненти для магазину реалізовані.

Тепер розглянемо український аналог. Prompt - “магазин аніме”

- pulsar ua



Один з перших сайтів не має хорошої адаптивної верстки. Є багато різних категорій. Можна зареєструватись і мати особистий кабінет. Але замовлення можна оформити і без реєстрації. З подібними українськими аналогами можна і треба змагатись.

Перейдімо до розгляду аналогів додатків з рекомендаційними системами, на які і зробимо акцент в аналізі рішень.

- Netflix

80 % часу переглядів на Нетфлікс досягається завдяки роботі рекомендаційної системи .

Нетфлікс має складну рекомендаційну систему, яка складається з різних блоків, що в свою чергу використовують різні алгоритми. Так, у сервіса є настпуні підсистеми [3]:

-Personalised Video Ranking (PVR)

Розраховує які фільми більше підійдуть користувачеві. В алгоритм включаються особливості користувача, популярність відео та різні сторонні характеристики.

-Top-N Video Ranker

Робить те саме що й PVR, але з урахуванням лише верхньої частини оцінок.

- Trending Now Ranker

Знаходить найпопулярніші відео зараз. Алгоритм враховує особливості

дня/сезону та актуальні події у світі.

- Continue Watching Ranker

Пропонує користувачу додивитись відео, які він почав. Алгоритм вираховує вірогідність того, що користувач продовжить дивитись відео. Дана підсистема працює на основі таких даних як хвилина зупинки відео, час, що минув з моменту перегляду, пристрій перегляду.

- Video-Video Similarity Ranker

Знаходить фільми подібні до тих, які користувач переглядав. Алгоритм враховує подібність між рисами відео для формування релевантних рекомендацій.

Система вважається прикладом гібридної. Так, Personalised Video Ranking - це collaborative filtering, тобто працює на основі рейтингів усіх користувачів. На основі вподобань користувача, знаходиться подібність між ним, та іншими. Потім на основі цієї подібності система передбачає як користувач оцінить те чи інше відео. Video-Video Similarity Ranker в свою чергу є item-item content-based filtering системою, тобто фільм порівнюється з іншими на основі їх рис та характеристик. Таким чином знаходяться подібні.

Багатошарова система нетфлікс має реалізовані різні типи рекомендаційних систем для своїх підсистем, отримуючи вигоду з кожної.

Висновок

Отже, в даному розділі було розглянуто актуальність питання, а саме:

- пояснено перспективність створення онлайн-магазину для продажу товарів пов’язаних з аніме за рахунок надання оцінки експертів щодо поточних трендів та прогнозів на майбутнє.

- пояснено корисність рекомендаційних систем у сучасному світі в контексті створення конкурентних додатків, розглянуто поширеність різних систем та їх роль у бізнес процесах

Також, в розділі були розглянуті

- аналоги магазинів аніме-товарів: закордонний та український. Було визначено конкурентну спроможність на українському ринку

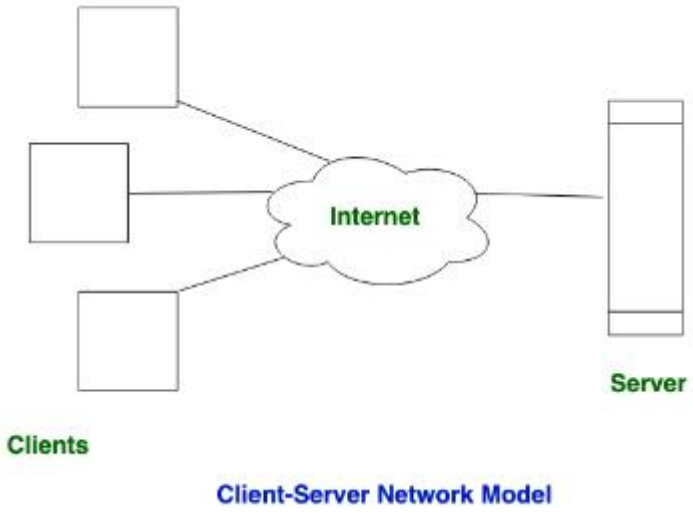
- аналог рекомендаційної системи у вигляді багатоскладової гібридної системи нетфлікс.

РОЗДІЛ 2: Аналіз теорії для розробки веб застосунків

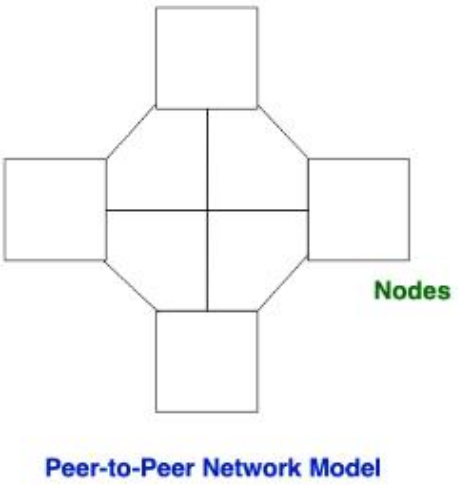
2.1 Peer-to-peer та Client-server

Розглянемо два різні архітектурні підходи для створення веб-додатків: Peer-to-peer та Client-server

В клієнт-серверній архітектурі клієнт і сервер чітко розділені: клієнт здійснює запит, сервер оброблює.



Peer-to-peer - однорангова мережа, тобто не передбачає поділ на клієнтів та сервери.



[4] Кожна нода є і клієнтом, і сервером водночас, відповідно може як робити запити, так і відповідати на них. Кожна нода має власні дані, коли в

клієнт-серверній архітектурі даними оперують централізовані сервери. Проте, Peer-to-peer підходить лише для маленьких мереж, тоді як клієнт серверна архітектура для будь-яких.

В моєму застосунку звісно буде використана клієнт-серверна архітектура.

2.2 Клієнт-серверна архітектура

Компоненти веб-додатків

Типово веб -додатки складаються з трьох компонентів [5]:

- Веб Браузер, фронтенд компонент, який відповідає за взаємодію з користувачем, а саме: контролює логіку відображення сторінок, отримує дані введені користувачем.

- Веб Сервер, бекенд компонент, який керує бізнес логікою і обробляє запити від користувача.

- Сервер Баз Даних, сховище для даних застосунку, керує операціями пов’язаними з даними.

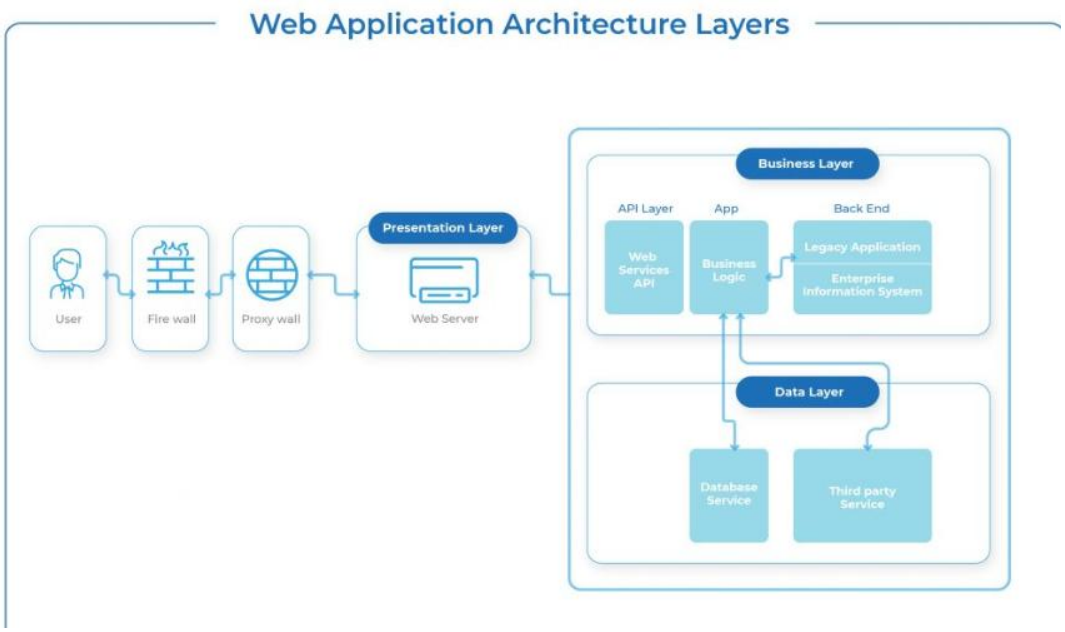
Трирівнева архітектура

[5] Двошарова архітектура, яка використовувалась раніше, мала 2 компоненти - інтерфейс/клієнт та бекенд, який зазвичай був сервером бази даних. Бізнес логіка в такій архітектурі була частиною клієнта або серверу баз даних. З дворівневою архітектурою важче вносити зміни, та не можна незалежно розробляти бізнес логіку. Також зі зростанням кількості користувачів знижується продуктивність застосунку.

Тришарова архітектура - спосіб відокремити компоненти, щоб кожен виконував свою функцію, і внесення змін в один компонент не тягнуло за собою зміни в інших. Таким чином можна легко міняти та тестувати код. Трирівнева архітектура включає в себе:

- Рівень клієнта

- Рівень застосунку (Бізнес логіка)
- Рівень даних



Запити йдуть на сервер, який керує бізнес логікою та робить запити в базу даних, після чого повертає клієнту відповідь. Компоненти в такій архітектурі можна легко замінювати.

Взаємодія клієнта з сервером

Клієнт та сервер взаємодіють між собою за допомогою HTTP протоколу. Клієнт відправляє запит на сервер, там він обробляється.

HTTP запити:

Method	Description
GET	Request for resource from server
POST	Submit data to the server
HEAD	Same as GET but does not return the body
PUT	The data within the request must be stored at the URL supplied, replacing any existing data.
DELETE	Delete a resource
OPTIONS	Return the HTTP methods supported by the server
CONNECT	Client requests the HTTP proxy to forward a TCP connection to some destination. Used to create a TCP/IP tunnel for secure connections using HTTP proxies.

(запити та їх функції)

На сервері

Сервер обробляє запити від клієнта, і звертається до бази даних.

CRUD	HTTP
CREATE	POST/PUT
READ	GET
UPDATE	PUT/POST/PATCH
DELETE	DELETE

Залежно від того які запити приходить на сервер, виконуються різні CRUD операції.

CRUD операції відповідають різним SQL запитам.

NAME	DESCRIPTION	SQL EQUIVALENT
Create	Adds one or more new entries	Insert
Read	Retrieves entries that match certain criteria (if there are any)	Select
Update	Changes specific fields in existing entries	Update
Delete	Entirely removes one or more existing entries	Delete

SOAP API vs REST API

API - інтерфейс за яким один додаток може доступатись до функцій іншого. У випадку клієнт-серверної взаємодії - отримувати інформацію від сервера.

[6] REST API використовує HTTP протокол і працює на основі URI (Uniform Resource Identifier), URL є типом URI. REST API використовує JSON формат для передачі даних. REST API легко створювати, масштабувати.

SOAP (Simple Object Access Protocol) сам по собі є протоколом. Він може надати вищий рівень безпеки, надійнішу комунікацію, забезпечити відповідність ACID (Atomicity, Consistency, Isolation, Durability). SOAP використовує XML для передачі даних, а запити до API - це HTTP POST запит.

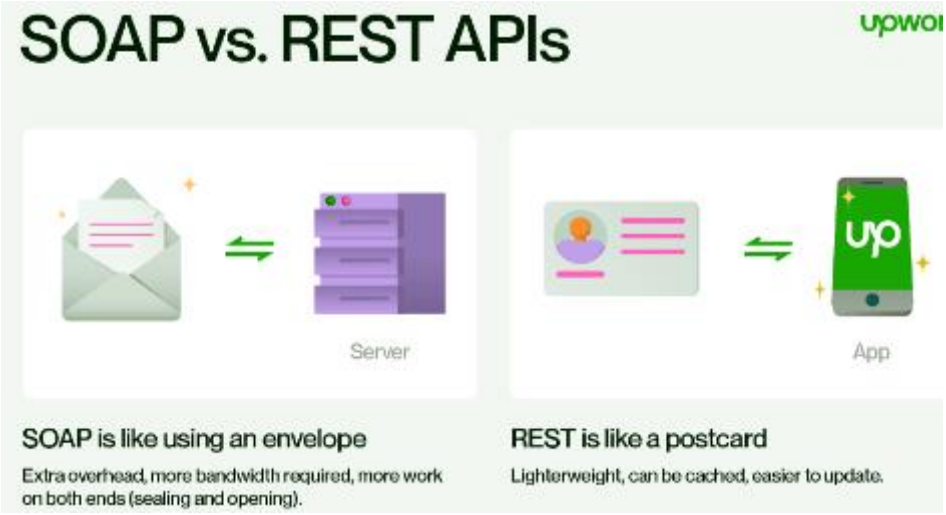
Тіло запиту має чітку структуру і складається з Envelope, Header, Body та Fault, де Envelope визначає API, до якого здійснюється запит, а body містить параметри запиту.

```
1  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soa
2    <soapenv:Header/>
3    <soapenv:Body>
4       <sch:UserDetailsRequest>
5          <sch:name>John</sch:name>
6       </sch:UserDetailsRequest>
7    </soapenv:Body>
8  </soapenv:Envelope>
```

Відповідь має такий же формат

```
1  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap
2    <soapenv:Header/>
3    <soapenv:Body>
4       <ns2:UserDetailsResponse xmlns:ns2="http://www.soapexample.
5          <ns2:User>
6             <ns2:name>John</ns2:name>
7             <ns2:age>5</ns2:age>
8             <ns2:address>Greenville</ns2:address>
9          </ns2:User
10      </ns2:UserDetailsResponse>
11   </soapenv:Body>
12 </soapenv:Envelope>
```

До REST API можна звертатись за допомогою усіх HTTP методів. Тут API, до якого йдуть запити визначається за допомогою URL, а тіло запиту передається у форматі Json.



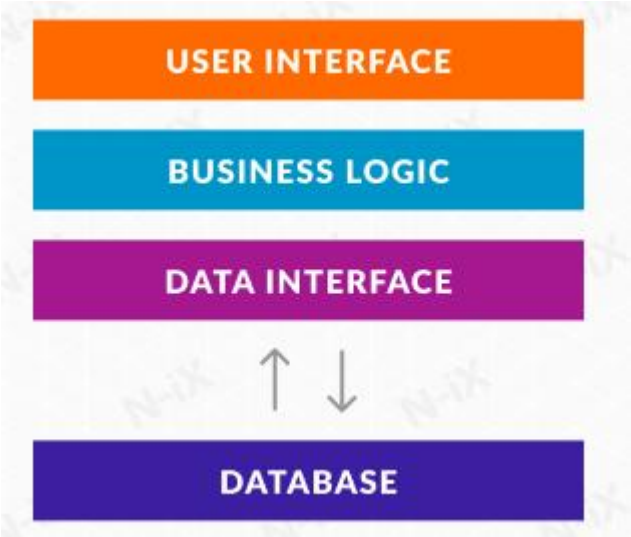
Загалом, для веб-додатків використовують REST API, так як

- ця комунікація не потребує багато ресурсів, і може використовувати різні формати, зокрема простий JSON.
- не треба зовнішні бібліотеки на стороні клієнта для звернення до API
- REST API швидше і простіше розробляти

У зв’язку з вищесказаним у своєму додатку я реалізовуватиму REST API.

2.3 Моноліт та мікросервіси

Є дві основні архітектури веб-додатків: моноліт та мікросервіси. [7] Моноліт традиційний метод побудови застосунків, він передбачає побудову однієї неподільної системи, яка складається з клієнта, сервера та бази. Вся бізнес логіка знаходиться в одному місці. При внесенні змін в будь-яку частину коду, розробник має оперувати з однією і тою ж базою коду.



Ось так виглядає традиційна монолітна архітектура.

Вона має свої переваги:

- Менше cross-cutting concerns, тобто легше логувати дії, виявляти та оброблювати помилки, оцінювати продуктивність додатку, менше проблем з його безпекою.

- Легше дебажити та тестувати

- Неподільну одиницю легко деплоїти

- Традиційна архітектура знайома всім розробникам, відповідно їх не треба донавчати, вони вже мають потрібні знання для розробки.

Та недоліки:

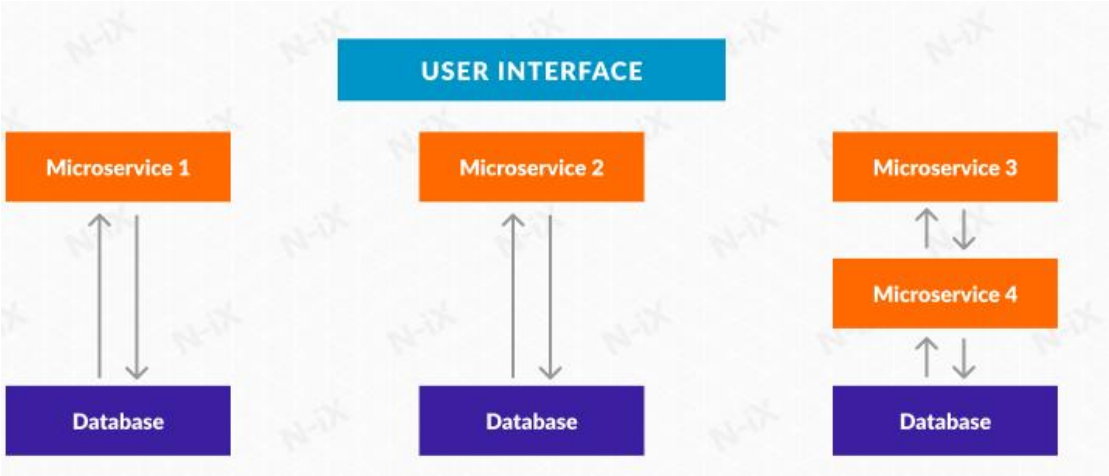
- Зі збільшенням коду в додатку, його все важче розуміти, вносити правки у код стає задачею, яка забирає немало часу.

- Висока зв’язність компонентів, тобто вносячи зміни в один, можуть знадобитися зміни в іншому.

- Не можна масштабувати компоненти незалежно одна від одної, так як додаток неподільний.

Мікросервісна архітектура передбачає поділ додатку на декілька незалежних між собою одиниць, які можуть спілкуватись між собою за допомогою API. Кожна одиниця, тобто сервіс, має свою логіку та базу даних. Кожен сервіс має свій функціонал, його можна оновлювати та масштабувати

незалежно від інших.



До переваг мікросервісної архітектури можна віднести:

- Незалежність компонентів дозволяє лекго вносити зміни в будь-який мікросервіс.

- Вихід з ладу одного мікросервісу не означатиме вихід з ладу всіх мікросервісів.

- Легко розуміти маленькі компоненти та вносити в них правки

- Можна масштабувати компоненти незалежно один від одного.

- Для різних мікросервісів можна використати різні технології, які краще підходять для виконання завдання.

До недоліків можна віднести:

- Потрібно встановлювати зв’язки між мікросервісами, які деплояться окремо.

- Cross-cutting concerns

- Важко тестувати та дебажити код.

Зрештою, кожна архітектура має свої переваги та недоліки.

Монолітну архітектуру доречно використати, якщо

- Додаток простий, отже не має сенсу розбивати його на менші компоненти

- Потрібно швидко розробити і запустити додаток

- Код є модульним, його досить просто розуміти

- Маленька команда розробників

Мікросервісну архітектуру доречно використати, якщо

- планується масштабування в майбутньому

- планується розширення можливостей додатку, поява нових функцій

- важливо, щоб одні компоненти працювали навіть тоді, коли інші вийшли з ладу

- планується використати різні технології для різних сервісів

- є велика команда розробників

Для онлайн-магазину аніме товарів я використаю монолітну архітектуру. Додаток матиме чітко визначену структуру контролерів, які відповідають за бізнес-логіку на сервері, відповідно вносити зміни в код не стане складною задачею. Також, враховуючи, що додаток розробляється однією людиною, мною, і я маю повністю розуміти роботу всіх компонент, не має сенсу розбивати додаток на мікросервіси, витрачаючи потім час на встановлення зв’язку між ними.

Проте, для рекомендаційної системи доцільно буде використати технології відмінні від тих, що будуть використані для написання основного функціоналу інтернет-магазину . Тому рекомендації будуть винесені в окремий мікросервіс. Сервер онлайн магазину буде звертатись до серверу сервісу рекомендацій для їх отримання. Це також дасть змогу легко замінити рекомендаційну систему в майбутньому.

Висновок

Отже, в розділі було розглянуто

- різні архітектурні підходи до створення веб-додатків Peer-to-peer та Client-server. Було визначено, що використовуватиметься клієнт-серверна архітектура.

- клієнт-серверну архітектуру, взаємодію клієнта та сервера

- SOAP та REST API, обрано REST архітектуру для реалізації API

- монолітну та мікросервісну архітектуру. Було визначено, що онлайн-магазин буде монолітним, проте система рекомендацій буде винесена в окремий сервіс.

РОЗДІЛ 3: Реалізація магазину для продажу аніме товарів

3.1 Аналіз ТЗ

Технічне завдання в розділі реалізації веб-додатку - написання магазину з приємним інтерфейсом та наступним функціоналом :

- Перегляд продукції
- Можливість додати товар до кошика
- Оформлення замовлення
- Написання відгуків та виставлення оцінок для товарів
- Перегляд схожих товарів
- Перегляд рекомендованих товарів

Крім того адміністраторам мають бути доступні такі дії як :

- Додавання товарів
- Додавання категорій товарів
- Видалення виконаних замовлень
- Зміна статусу замовлення (очікування, прийнято, доставляється).

Користувачам також треба надати змогу реєструватися в додатку.

Вимоги до даних

- Користувач - має пошту, пароль. Може оформлювати замовлення. Може мати одну з двох ролей: адміністратор чи звичайний користувач
- Категорія - має назву, керується тільки адміністраторами
- Продукт - додається адміністратором в базу. Користувач додає продукт в кошик. Користувач створює відгуки для продукту. Замовлення має список продуктів.
- Замовлення - має містити список товарів з кількістю та статус.
- Відгуки - містять інформації про користувача, який його створив, та продукт, відгук на який написано.

3.2 Вибір засобів розробки

Для розробки backend частини, яка включає в себе сервер, було обрано Node js [8]- середовище, що побудоване на Javascript рушієві Chrome V8. Рушій V8 перетворює джс код в швидкий машинний код.

[9] Node js використовує неблокувальну модель вводу-виводу на основі подій. Для кожного нового користувача не потрібно запускати новий потік, замість цього Node js має цикл подій, який дозволяє реалізовувати неблокувальні операції. Також варто зазначити, що з Node js можна використовувати безліч npm бібліотек, які розробляються людьми у всьому світі для рішення величезної кількості задач.

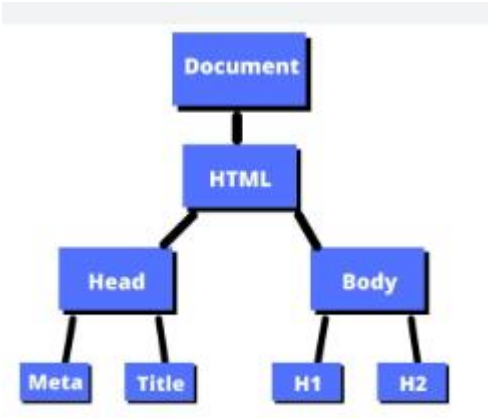
Для створення серверу буде використаний Express js - фреймворк для Node js. Дана технологія робить створення додатків швидшим за рахунок інструментів для маршрутизації, обробки запитів, роботи зі статичними файлами, шаблонізаторами, мідлвейрами.

Для розробки користувацького інтерфейсу та клієнтської частини буде використана JS бібліотека React. До переваг React входять [10]:

- Використання JSX - JavaScript XML - дозволяє писати HTML прямо з JavaScript коду. Це пришвидшує розробку і може полегшує читання коду. Крім того, JSX швидший за звичайний JavaScript код завдяки внутрішнім оптимізаціям.

- Virtual DOM. При завантаженні HTML сторінки браузер зчитує її та будує DOM дерево об’єктів. JavaScript код взаємодіє з DOM деревом, маніпулюючи його елементами, щоб динамічно змінювати контент. Проте робота з DOM деревом ресурсозатратний процес. Тому в React і створили Virtual DOM - більш легковажна абстракція DOM дерева. [11] Коли стан компонентів змінюється, створюється другий Virtual DOM. Далі дерева порівнюються між собою, виявляються зміни, які і вносяться в DOM. Операції маніпуляції з

елементами Virtual DOM значно швидші, включаючи і операцію порівняння дерев.



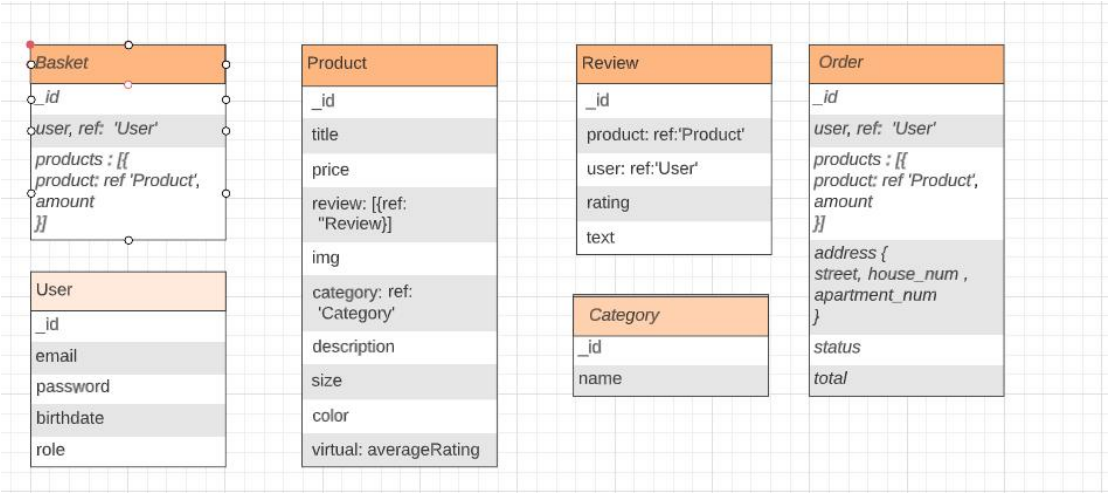
- Використання композиції компонентів для побудови більших, можливість перевикористання компонентів.

Буде використана база даних MongoDB.

3.3 Реалізація магазину аніме-товарів

3.3.1 Модель даних

Перед тим як починати створення бази даних, потрібно визначити модель, яка задовольнятиме вимогам додатку. Згідно з ними, у нас точно мають бути такі сутності, як: продукт, відгук, категорія, замовлення, користувач та кошик. Тому модель даних виглядатиме ось так:

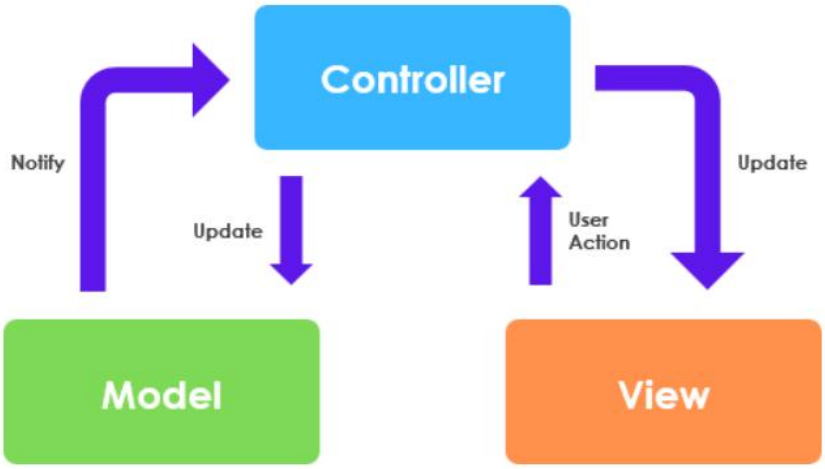


Так, як

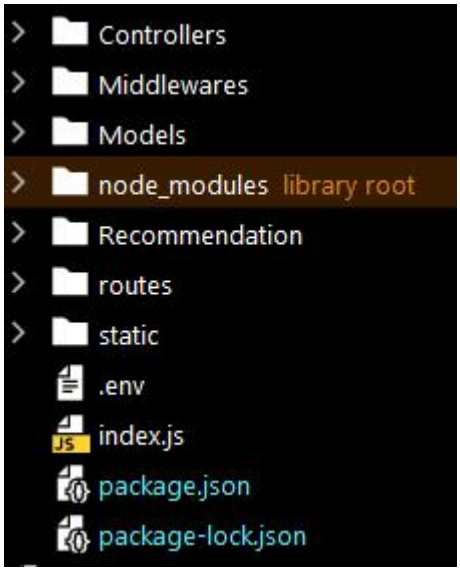
MongoDB є нереляційною базою даних, зв’язків у ній ми не створюємо. Проте, за допомогою ref забезпечується, що значення поля є id об’єкта моделі на яку посилаються. В кошику і замовленні є ref користувача і ref продукту, у продукту ref категорії, у відгуку ref користувача та продукту, відгук на який написано. У продукту є також віртуальне поле averageRating. Поля не буде в базі даних, але воно буде додаватись до об’єкту при його отриманні, після чого в коді йому буде присвоєне відповідне значення.

3.3.2 Сервер

Додаток реалізований за принципами клієнт-серверної архітектури та з використанням MVC (Model-View_Controller) паттерну, де Model - моделі Mongo, які відповідають за зв’язок з базою даних MongoDB, View - інтерфейс на React, Controller - сервер, на якому представлена логіка програми, оброблює взаємодію користувача з View (приймає запити), викликає відповідні методи моделі для роботи з базою даних.



На стороні сервера є наступна структура папок:



В файлі індекс відбувається під’єднання до бази даних, а також запускається сервер.

```
mongoose.connect(DB_URL, () => {
    console.log("connect")
})

app.use(cors())
app.use(express.json())
app.use(express.static(path.resolve(__dirname, 'static')))
app.use(fileUpload( options: {}))
app.use(router)


YevheniiKyr
app.listen(PORT, callback: ()=>console.log("server is started"))
```

В папці routes є файл індекс, який відповідає за роутинг у застосунку та окремо визначені роутери для кожної сутності.

```
router.use('/user', userRouter)
router.use('/product', productRouter)
router.use('/category', categoryRouter)
router.use('/basket', basketRouter)
router.use('/order', orderRouter)
router.use('/review', reviewRouter)
router.use('/auth', authRouter)
router.use('/payment', paymentRouter)


module.exports = router
```

```
const router = require('express').Router()
const productController = require("../Controllers/productController");
const checkRole = require("../Middlewares/checkRoleMiddleware")

router.get( path: '/' ,  productController.getAll)
router.get( path: '/:id', productController.getByID)
router.put( path: '/:id', checkRole( role: 'ADMIN'), productController.update)
router.delete( path: '/:id', checkRole( role: 'ADMIN'), productController.delete)
router.post( path: '/',  checkRole( role: 'ADMIN'),  productController.create)

module.exports = router
```

Роутери визначають яка функція контролера відпрацює залежно від того який запит на який маршрут надходить. Також в роутері використовуються middleware для перевірки ролі, авторизованості ітд.

Контроллери в свою чергу обробляють запити: дістають потрібну інформацію, викликають методи моделі для маніпуляції з даними та формують відповідь для відправлення на клієнт.

```
async delete(req, res) {
    try {
        const {id} = req.params
        if (!id) {
            res.status(400).json({message: 'no id'});
        }
        const deletedProduct = await Product.findByIdAndDelete(id);

        if (deletedProduct == null) {
            res.json({message: "Cant find with this id"})
        } else res.json(deletedProduct);
    } catch (e) {
        res.status(500).json(e)
    }

}
```

Для полегшення роботи з MongoDB використовується Mongoose ODM.

Це довзоляє нам створювати схему для бази даних, яка одночасно буде моделлю для звернень до бази.

```
const Order = new mongoose.Schema(
    definition: {
        user: {type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true},
        products: [
            {
                product: {type: mongoose.Schema.Types.ObjectId, ref: 'Product'},
                amount: {type: Number, required: true}
            }],

        address: {
            street: {type: String},
            house_num: {type: String},
            apartment_num: {type: String, required: false}
        },
        status: {type: String, default: "pending"},
        total: {type: Number, default: 0}
    },
    options: {timestamps: true}
);

module.exports = mongoose.model( name: "Order", Order)
```

Так як MongoDB документоорієнтована база, поле може бути масивом або мати вкладені атрибути. Для того, щоб посилатись на зовнішній об’єкт задається ref.

```
product: {type: mongoose.Schema.Types.ObjectId, ref: 'Product'
```

В застосунку є моделі: замовлення, категорія, продукт, відгук, кошик, користувач. Є контролери для взаємодії з кожною моделлю. Є роутери для співставлення маршрутів та функцій контролера.

Авторизація реалізована за допомогою jwt token [12]:

1) Після авторизації на сервері генерується токен, який вілправляється на клієнт.

2) Токен зберігається в кукі / локальному сховищі.

3) При здійснені запитів у headers додається токен.

4) На сервері токен перевіряється на валідність.

Переваги токенів:

- не треба зберігати на сервері

- надійно зашифровані

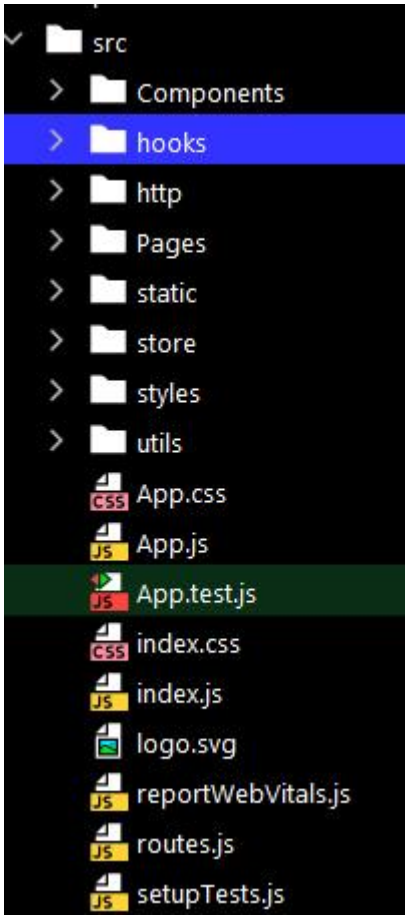
- роблять процес авторизації швидшим порівняно з сесіями, адже не потрібно робити пошук в базі даних

- дозволяють робити авторизацію через сторонні застосунки

3.3.3 Client

Клієнтська частина зроблена з використанням бібліотеки React js і відповідає за взаємодію користувача з застосунком.

Структура папок на клієнті



Основний компонент App.js - містить в собі NavBar, який буде доступним на всіх сторінках та AppRouter для здійснення маршрутизації. AppRouter визначає які маршрути яким користувачам доступні, та які компоненти за яким маршрутом відмальовуються.

```
<Routes>

    {user?.isAuth && user?.user?.role === 'ADMIN' && adminRoutes.map(({path : string , Component}
        <Route key={path} path={path}  element={<Component/>} exact/>
    )}
    {user?.isAuth && authRoutes.map(({path : string , Component}) =>
        <Route key={path} path={path}  element={<Component/>} exact/>
    )}
    {publicRoutes.map(({path : string , Component : FunctionComponent<object> }) =>
        <Route key={path} path={path}  element={<Component/>} exact/>
    )}
    <Route path="*" element={<Navigate to={SHOP_ROUTE} replace />} />


</Routes>
```

Сторінки, що відмальовуються знаходяться в папці Pages і є функціональними компонентами, що складаються з власного коду та компонентів з папки Components. React дозволяє робити їх композиції. Для полегшення процесу верстки використаний react-bootstrap. В сторінках та компонентах використовуються різноманітні хуки:

- щоб керувати станом компонентів (useState)
- для підвантаження даних коли це потрібно (useEffect)
- для навігації useNavigate

Крім того для керування станом об’єктів використаний state manager mobx. З його допомогою створюються різноманітні Stores, за якими слідкує mobx, і які дозволяють отримувати дані з нього у тій точці програми, де це потрібно. В порівнянні з традиційним підходом нам більше не треба передавати дані через пропси на декілька рівнів вверх чи вниз.

```
export default class UserStore {

    1 usage  YevheniiKyr
    constructor() {
        this._isAuth = false
        this._user = null
        makeAutoObservable(this)
    }

    4 usages  YevheniiKyr
    setIsAuth(bool){
        this._isAuth = bool
    }
    4 usages  YevheniiKyr
    setUser(user) {
        this._user = user
    }


    5+ usages  YevheniiKyr
    get isAuth() {
        return this._isAuth
    }
    5+ usages  YevheniiKyr
    get user() {
        return this._user
    }


}
```

makeAutoObserveble робить так, що компоненти в яких задіяний даний store будуть перемальовуватись при зміні стану об’єктів всередині store. Для відслідковування змін компоненти загортають в observer.

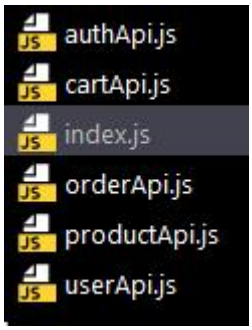
```
const MainPage = observer( baseComponent: () => {
```

Щоб мати можливість дістати Store у будь-якому файлі, потрібно загорнути App в Context.Provider, і передати йому екземпляри наших Store об’єктів.

```
<React.StrictMode>
    <Context.Provider value={{
        user: new UserStore(),
        product: new ProductStore(),
        basket: new BasketStore(),
        reviewsContext: new ReviewStore(),
        optionsStore: new OptionsStore()
    }} >
    <App />
    </Context.Provider>
</React.StrictMode>
```

При взаємодії користувача з клієнтською частиною веб-додатку є

необхідність відправляти запити на сервер. Для цього реалізовані відповідні запити в файлах папки http




```
export const fetchBasketById = async (basket_id) => {
    const {data} = await $authHost.get( url: 'basket/' + basket_id)

    console.log("BASKET DATA BY ID" + data)
    return data
}
```

Для виконання запитів використаний axios, налаштований для відправки запитів на сервер

```
const $host = axios.create({
    baseURL: process.env.REACT_APP_API_URL
})

5+ usages  YevheniiKyr
const $authHost = axios.create({
    baseURL: process.env.REACT_APP_API_URL
})

1 usage  YevheniiKyr
const authInterceptor = config => {
    config.headers.authorization = `Bearer ${localStorage.getItem( key: 'token')}
    return config
}

$authHost.interceptors.request.use(authInterceptor)
```

Висновок

Отже, в даному розділі було визначено технічне завдання для виконання, в якому визначено основні вимоги до додатку. Після цього розглянуті технології для побудови застосунку і аргументовано їх швидкодію та перевагу. Було визначено використати Node js для серверної частини, React для клієнтської, MongoDB для рівня даних

Далі розглянули MVC патерн, використаний для реалізації додатку. Була описана реалізація серверної частини, включаючи опис моделей, контролерів, процесів роутингу, взаємодії компонент та структури роботи сервера загалом. Розглянуто клієнтську частину, включаючи сторінки, компоненти, роутинг, взаємодію клієнта та сервера.

РОЗДІЛ 4: Аналіз теорії для розробки рекомендаційних систем

4.1 Збір та обробка інформації

Для персоналізації рекомендацій необхідно збирати інформацію про користувача. Першим кроком потрібно визначити, які дані збирати. Наприклад, історія пошуку користувача на сайті, переглянуті товари, куплені чи додані в кошик, рецензії на товари. Крім того можна збирати дані про стать, вік, національність користувача.

Після того, як визначено, які ж дані збирати, треба вирішити як це зробити. Найочевиднішим видається збирати дані прямо з бази даних. Також можна логувати дії користувачів при взаємодії з додатком, брати дані зі сторонніх сервісів.

Дані, що збираються, можна оброблювати перед збереженням до бази даних. Тоді значення там вже будуть готові до використання в системі рекомендацій. Це може забезпечити більшу швидкодію. Можна оброблювати дані і після збереження при їх отриманні. Цей підхід може дати гнучкість. Наприклад, при зміні даних, ми будемо оброблювати актуальні дані. Також можна легко змінити обробку даних не змінюючи її структури.

Тож дані потребують обробки. Наприклад: видалення дублікатів, порожніх чи неправильних з боку семантики значень, зрештою приведення даних до універсального вигляду. Порожні значення можуть заповнюватись медіаною, середнім значення , за допомогою методів інтерполяції і так далі , залежно від вимог системи. Важливим етапом при обробці даних є нормалізація - збалансування впливу різних ознак на роботу моделі, наприклад шляхом переведення значень усіх ознак у діапазон від 0 до 100.

Важливо відібрати ознаки, важливі для роботи конкретної моделі, і відкинути неістотні. Для цього можна використовувати методи кореляції,

частотні аналізи. Даний крок дозволить зменшити кількість вхідних даних, сконцентруватись на дійсно важливих аспектах і зробити більш ефективні алгоритми.

Для забезпечення надійної роботи над даними для рекомендаційної моделі потрібно також виявляти аномалії - значення ознак, що сильно відхиляються від середніх, не входять у встановлені інтервали, або набір значень, що є несумісним, виходить за межі норми. Після виявлення аномальних значень їх можна не обробляти далі, тобто позбутись від них, або скорегувати їх.

Дані можна зберегти у файлах чи базі даних, локально або у хмарі. Важливо щоб сховище для зберігання даних було надійним для мінімізації можливості витоку даних.

Перед використанням збережених даних в моделі їх потрібно провалідувати.

4.2 Вибір алгоритмів та моделей

Після обробки дані готові для використання. На цьому етапі потрібно обрати модель та алгоритм для генерації рекомендацій. Розглянемо які види рекомендаційних систем існують :

- Collaborative filter
- Content-based filter
- Гібридні моделі

Метрики подібності [13]

Перед розглядом різних видів рекомендаційних систем потрібно розібратись з розрахунком подібності. Адже рекомендаційні системи так чи інакше використовують різні метрики для надання релевантних рекомендацій. Для визначення подібності в використовують різні методи. Наприклад:

- Евклідова відстань

Вимірюється евклідова відстань між точками на n-розмірній площині. Менша відстань означає більшу подібність. Евклідова відстань в даному контексті - корінь суми квадратів різниць між значеннями у двох векторах.

$$d_{Eucl}(p,q) = \\ = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+...+(q_n-p_n)^2} = \\ = \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}$$

Евклідову відстань використовують коли важливою є різниця в магнітуді значень.

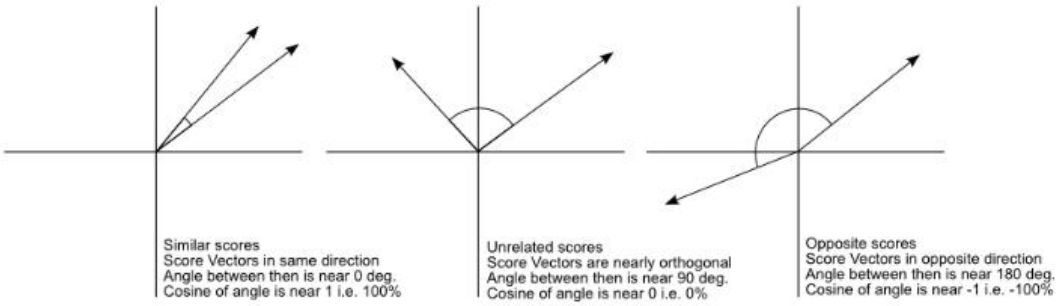
- кореляція Пірсона

Вимірюється лінійна залежність між векторами взаємодій.

$$r = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

Значення кореляції може набувати значення від -1 до 1, де -1 вважається абсолютною негативною кореляцією, а 1 відповідно позитивною. Таким чином, ми можемо робити передбачення за допомогою користувачів з протилежними смаками: якщо кореляція -1 і користувачу А не сподобався об’єкт, можна зробити припущення, що він сподобається користувачу Б. Якщо ж кореляція 0, то користувачі між собою абсолютно не пов’зані. Кореляція Пірсона використовується коли магнітуда значень не є занадто важливою.

- Косинусна подібність



Вимірюється косинус кута між векторами. Чим менший кут між векторами, тим більша подібність між користувачами/об’єктами.

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\cdot\|\vec{b}\|}$$

За такою формулою розраховується косинус кута між векторами.

Та інші методи.

4.2.1 Collaborative filtering

Підхід використовує подібність між вподобаннями користувачів. Collaborative filter не потребує властивостей об’єктів, в даній системі потрібні взаємодії інших користувачів з об’єктами, взаємодія поточного користувача з об’єктами, і визначення подібності цих взаємодій. Collaborative filter можна розділити на дві великі категорії: Memory-based collaborative filtering та Model-based collaborative filtering

4.2.1.1 Memory-based collaborative filtering

За основу беруть матрицю взаємодій користувачів з об’єктами. В цьому типі систем не використовуються моделі.

- user-based collaborative filtering [14]

На основі оцінок, історії пошуків, покупок користувача, він

співставляється з іншими користувачами, які надавали схожі оцінки та відгуки або мають схожу історію. Якщо подібним користувачам об'єкт сподобався, система робить висновок, що поточному користувачу він сподобається теж. Щоб здійснити user-based filtering потрібно створити матрицю рейтингів, в якій будуть продукти і оцінки користувачів, рядки в матриці - вектори оцінок користувачів продуктам. Після цього за допомогою метрик подібності знайти схожих користувачів. Ми отримаємо вектор подібності користувача до інших. Для створення вектору оцінок можна використовувати схожих користувачів : якщо продукт не оцінений можна встановити середнє значення оцінок схожих користувачів. Деякі користувачі ставлять загалом нижчі бали за інших, і це треба враховувати при створенні вектору взаємодій. Потрібно нормалізувати значення, наприклад привівши їх в діапазон 1-5.

- item-based collaborative filtering

Цей підхід працює на основі подібності між об'єктами. Якщо два об'єкти мають схожі взаємодії з користувачами, і користувач зацікавлений в одному з них, то можна зробити висновок, що він зацікавлений і в другому. Щоб здійснити item-based filtering потрібно створити вектори взаємодій для кожного об'єкта - тобто вектор, який відображає схожість об'єкта з усіма іншими. З матриці оцінок користувачів об'єктам можна розрахувати схожість між об'єктами.

Розберемо, як можуть працювати memory -based системи [13]

- Mean, тобто середнє значення.

Найпростіший метод, адже для передбачення беруть середнє значення оцінок усіх користувачів, які оцінили об'єкт. В даному методі всі користувачі розглядаються як однакові, тобто подібність усіх визначена як однакова.

- Weighted Mean, середнє зважене

В цьому методі враховується вага кожного користувача, тобто чим більша схожість, тим більший його вплив на передбачення.

$$W = \frac{\sum_{i=1}^{n} w_i X_i}{\sum_{i=1}^{n} w_i}$$

W = weighted average

n = number of terms to be averaged

w_i = weights applied to x values

X_i = data values to be averaged

Середнє зважене враховує вагу кожного користувача при множенні ваги на оцінку. Знаходиться скалярний добуток векторів схожості та векторів оцінок, після чого він ділиться на суму усіх елементів ваги для знаходження зваженого середнього.

При знаходженні Weighted Mean не враховуються користувачі, які не оцінили об’єкт. Підхід більш ефективний за Mean адже враховує подібність між користувачами.

- User demographics

При створенні user-based системи можуть бути створені фільтри гендеру, статі, віку ітд. Тоді при передбаченні можуть враховуватись тільки люди з тієї ж групи.

User-based collaborative filtering працює на вподобаннях користувачів та подібності між користувачами. Даний підхід може враховувати такі дані як вік, стать ітд. Чим більше схожих користувачів, тим кращі рекомендації будуть підібрані.

В item-based collaborative filtering в свою чергу потрібні менші ресурси для обчислення, так як об’єкт має невеликий набір визначених властивостей та ознак, готових до використання. Обчислення подібності між користувачами є більш витратним, адже користувачі можуть мати великі обсяги вподобань, та більше параметрів для визначення подібності. Якщо є багато об’єктів без взаємодії з користувачами, то використання Item-based дає змогу краще враховувати їх при підборі рекомендацій, тоді як user-based просто проігнорує об’єкти, які мають мало взаємодій з користувачами або не мають їх взагалі.

4.2.1.2 Model-based collaborative filtering

За допомогою матриці взаємодій, використовуючи алгоритми машинного навчання, ми намагаємось передбачити як користувач оцінить об’єкт.

Розглянемо моделі машинного навчання, які використовуються для подібних передбачень

Алгоритми кластеризації

Кластеризація [15] - дані розподіляються на декілька груп, таким чином що в одній групі найбільш подібні об’єкти, а в різних групах об’єкти відповідно менш схожі. Отож ціллю кластеризації є виділення груп об’єктів залежно від їх подібності та неподібності.

Серед алгоритмів кластеризації виділяють наступні [15] :

- Density-Based Methods

Кластери визначаються на основі щільності даних, тобто області з високою щільністю об’єднуються у кластери. Таким чином розподіли можуть бути довільної форми, якщо можна з’єднати щільні області.

В дану категорію можна віднести алгоритм DBSCAN [16] - Density-based spatial clustering of applications with noise. Він працює, використовуючи параметри eps - радіус навколо точки, та minPts - кількість точок, які мають знаходитись в радіусі eps, щоб точка могла бути визнаною ядром. Якщо навколо менше minPts точок то точка переноситься в список можливих Noise - точки, які не мають сусідів у радіусі eps, і не можуть бути віднесені до кластера. Якщо точка все ж визнана ядром, потрібно обійти її сусідів у радусі eps, і приєднати до кластеру. Якщо у сусіда виявляється теж minPts точок у радіусі eps, то сусід не створює новий кластер, а доєднується до існуючого. З цим сусідом треба провести аналогічну процедуру обходу. Кінцевий результат алгоритму - множина кластерів та точок визнаних noise. Точки шуму не мають достатньо сусідів, щоб створити власний кластер, і знаходяться надто далеко

від інших, щоб бути віднесеними до того, що вже існує.

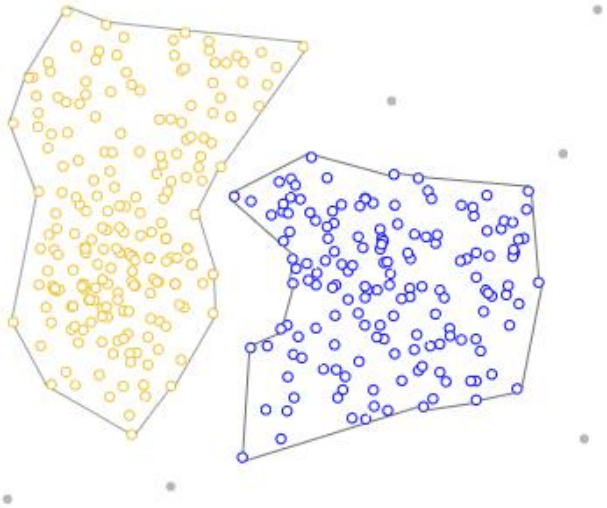


Figure 2: Example of density-based clustering.

- **Hierarchical Based Methods**

[17] Існує два види алгоритмів цього класу. В першому, який називають Agglomerative, дані групуються в дерево кластерів. На початку кожна точка сприймається як окремий кластер. Знаходимо 2 найближчі кластери та з'єднуємо, і так доки не все не буде об'єднано в один кластер. На виході маємо дерево кластерів. В другому, Divisive, на початку всі точки - один кластер, і на кожній ітерації ми розділяємо його на менші за подібністю.

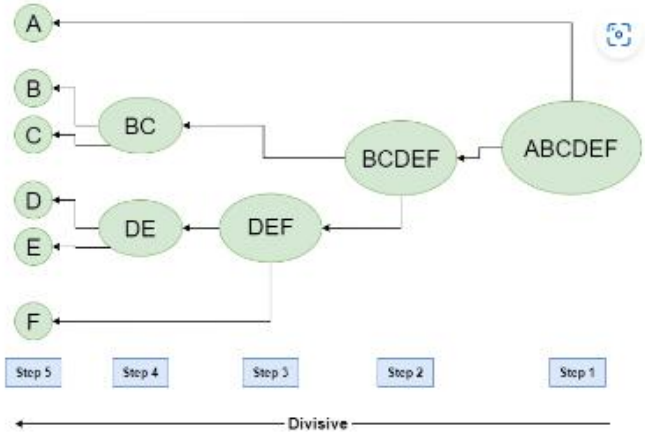
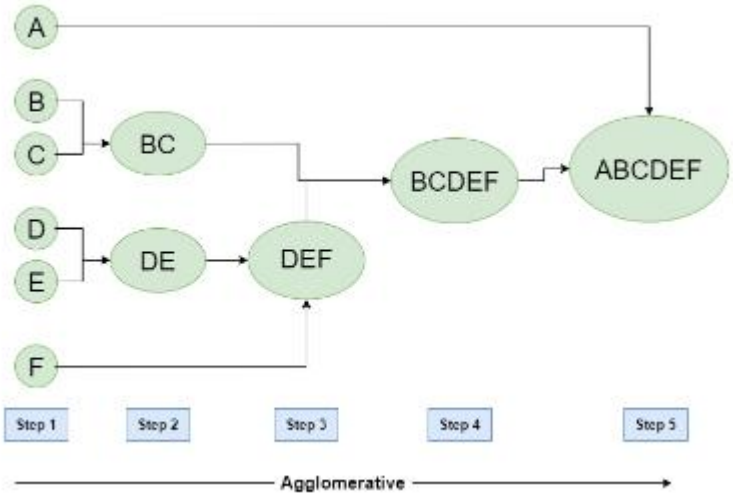


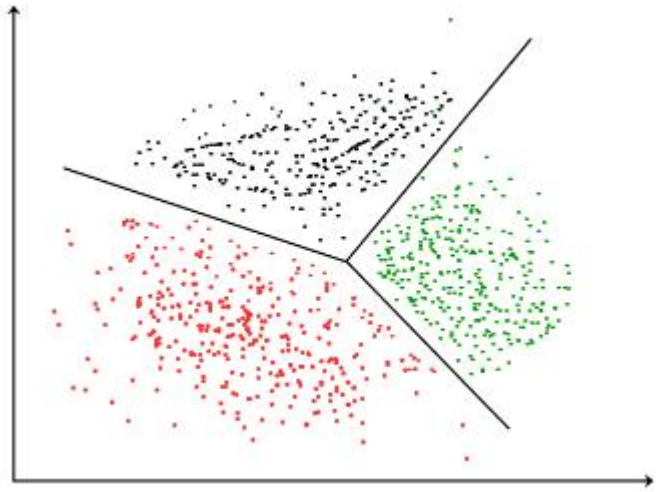
Figure – Divisive Hierarchical clustering



- **Partitioning methods**

Метод для поділу об’єктів на кластери, в якому кожний поділ створює новий кластер. В цю категорію можна занести метод K-середніх , CLARANS та інші.

Метод K-середніх [18] означає, що ми хочемо розподілити дані по k кластерам. Спершу визначають кількість кластерів k, після цього обираються k точок, які будуть вважатись центрами кластерів. Далі кожна точка відноситься до кластера, відстань до якого найменша. Тоді розраховуються нові центри кластерів - середнє арифметичне ознак усіх точок. Кожна точка знову відноситься до найближчого центру, а ті перераховуються, цей процес відбувається ітеративно, поки при кожній ітерації в кластер не будуть потрапляти одні і ті ж точки.



-**Grid-based Methods**

Простір даних складається з певної кількості комірок, які утворюють сіткоподібну структуру. Операції кластеризації виконуються швидко і не залежать від кількості об’єктів. Наприклад, STING (Statistical Information Grid Clustering Algorithm), Wave Cluster, CLIQUE.

STING [19]

- В даному алгоритмі використовується багатовимірна структура яка ділить простір на скінченну кількість клітинок, і увага приділяється не точкам, а простору навколо них, клітинкам. Простір розділений на прямокутні комірки, яких в свою чергу декілька шарів.

Отримані кластери можна використовувати для надання рекомендацій. Наприклад, для передбачення можна брати лише користувачів з того ж кластеру, або надавати їм більшу вагу при обрахунку рекомендацій.

Matrix Factorization based algorithms

Матриця взаємодій між користувачами і об’єктами може бути факторизована на дві менші матриці, після чого можна обрахувати вектори взаємодій user-user або item-item шляхом множення відповідних векторів отриманих менших матриць. В класичному розумінні матрична факторизація - це отримання з матриці A розміром m*n матриць X розміром m*r і Y r*n таких, що A = XY. Це дає змогу не зберігати матрицю m*n, а зберігати дві матриці m*r і n*r, що на великих m і n відчутно зменшує використання пам’яті. У випадку з таблицею взаємодій користувачів з об’єктами матрична факторизація може давати нам прогноз для незаповнених полів.

Отже, завдання - дві новоутворені матриці в добутку мають давати матрицю максимально подібну до матриці взаємодій користувачів та об’єктів. [14] Одна з матриць - пов’язана з характеристиками користувачів, а інша - з характеристиками об’єктів . Таким чином після проведення матричної факторизації ми навчимось робити прогноз про оцінку об’єкта користувачем.

У користувачів та об’єктів може бути дуже багато характеристик, тому для початку треба зменшити їх розмірність. Для проведення матричної факторизації є декілька методів. Серед них:

- SVD: Singular value decomposition

Фактично метод є розбиттям матриці на дві матриці і 1 вектор

$$A = UWV^T$$

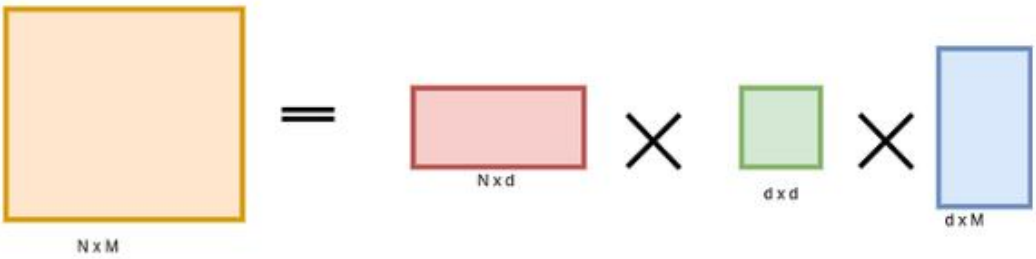
U - item матриця розміру m*k, left singular vector

V t - k*n matrix, right singular vector

W - k*k діагональна матриця

Фактично матриця W потрібна для того щоб множення U та V t в результаті дало матрицю A.

SVD декомпозиція :



Де

N - кількість об’єктів items

M - Кількість користувачів

D - розмір вектору характеристик

[20] Щоб знайти SVD спершу треба обрахувати single values, які будуть відображені в матриці W. Singular values є показниками важливості кожного singular vector у відображенні оригінальної матриці A. Чим більше значення Singular value, тим важливіший відповідний singular vector. Singular values обраховують за допомогою знаходження айгенвекторів, або власних векторів, таблиці A*A t. Власний вектор - ненульовий вектор, який при множенні на матрицю A дає вектор, паралельний вхідному. Після знаходження власних чисел і відповідно матриці W, переходимо на знаходження правого singular vector. В кінці обчислюємо лівий singular vector. SVD не показує себе добре на

розріджених матрицях - тобто матрицях з багатьма нулями, незаповненими значеннями. Тому існують варіації алгоритму SVD SSVD і randomized SVD (rSVD), які краще підходять для розріджених даних.

- **NMF (Non-Negative Matrix Factorization)**

[14] Метод називається так, тому що всі значення будуть матриці будуть позитивні. Рейтинг - позитивний, а якщо відсутній, то 0.

Даний метод передбачає розбиття оригінальної матриці A на 2 матриці, таким чином, що

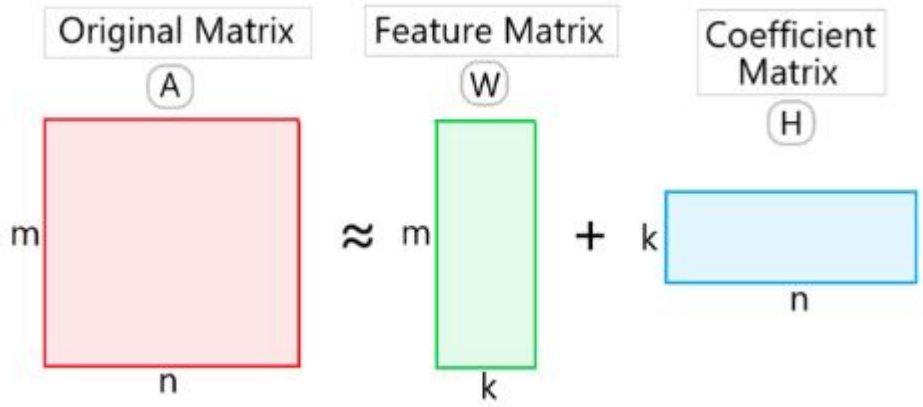
A = W*H

Де

A - матриця n*m , n - кількість користувачів, m - кількість об’єктів.

W - матриця характеристик n*d, d - розмір вектора характеристик.

H - матриця коефіцієнтів d*m, де елементи - міри ваги характеристик B.



(ілюстрація)

Завдання NMF - зменшення розмірності векторів характеристик та виділення істотних і важливих. Коли ми ставимо меншу розмірність d - ціль NMF не змінюється (знайти дві матриці, які в результаті дають оригінальну). Тому NMF ефективний алгоритм, який дає змогу після факторизації отримати матриці зі значно меншою розмірністю.

Як працює NMF

1) Приймає дані з певними атрибутами, після чого створює новий набір характеристик, де кожна - лінійна комбінація початкових.

2) Робить матричну факторизацію - отримуємо дві матриці, описані вище.

NMF працює краще з розрідженими даними.

- WMF (Weighted matrix factorization)

В SVD та NMF всі взаємодії між користувачами і об’єктами розглядаються як однакові за важливістю. Тоді як у WMF у кожної взаємодії є своя вага (weight). Вагу можна регулювати, щоб різні взаємодії по-різному впливали на роботу рекомендаційного алгоритму. Наприклад , одну вагу отримають об’єкти які мають оцінку, а іншу відповідно ті, які її не мають.

Функції втрати

Кожен з алгоритмів матричної факторизації має свою функцію втрат.

Так, для SVD це

$$Loss= \Sigma_i\Sigma_j(A_{ij} - U_iV_j^T)^2$$

Де Aij це оцінка в початковій таблиці, а u, v - значення, які згенерував алгоритм.

Для NMF функція така ж, але з обмеженням : NMF входить до категорії Only MF - тобто враховує лише взаємодії, які відбулись

Функцію можна записати як

$$Loss= \Sigma_i\Sigma_j(A_{ij} - U_iV_j^T)^2 \quad \text{for } r(i,j)=1 \text{ or } (i,j) \in (obs/rated)$$

.

Для WMF функція витрат буде наступною:

$$Loss=W_0\ \Sigma_i\Sigma_j(A_{ij} - U_iV_j^T)^2 \quad \text{for } r(i,j)=1 \text{ or } (i,j) \in (rated) + W_1\ \ \Sigma_i\Sigma_j(U_iV_j^T)^2 \quad \text{for } r(i,j)=0 \text{ or } (i,j) \notin (rated)$$

Тут вага взаємодії різниться залежно від того, чи взаємодія відбулась, а саме w0, якщо відбулась, w1 інакше.

Наглядна ілюстрація

Отже, ми розглянули функції витрат. Дані алгоритми факторизації мінімізують відповідні функції витрат.

4.2.1.3 Memory based vs Model based

Як одні, так і інші системи мають свої переваги та недоліки.

[22] До переваг memory-based можна віднести:

- Прості для розуміння, легко та швидко імплементуються
- Ефективні для відносно невеликих наборів даних

До недоліків:

- Якщо матриці сильно розріджені, рекомендації можуть бути неточними
- Відчутне зниження продуктивності при збільшенні даних

До переваг model-based можна віднести:

- Ефективно оперує великими наборами даних
- Працює з розрідженими даними краще за memory-based системи, відповідно може надавати більш релевантні рекомендації
- Вміє долати проблему “Холодного старту” для нових користувачів чи об’єктів
- model-based система вивчає зв’язки між характеристиками об’єктів та оцінками. Тоді як memory-based опирається на метрики подібності, model -based знаходить неочевидні зв’язки між даними, які використовує для надання рекомендацій.

До недоліків:

- Не така проста для розуміння. Потребує знань алгоритмів машинного

навчання, вміння будувати та тренувати модель.

- Потребує більше обчислювальних ресурсів
- При поганому підборі навчальних даних якість системи знижується

Memory-based можна швидко зрозуміти та імплементувати, проте у них можуть виникати проблеми з масштабуванням та релевантністю рекомендацій. Model-based долає ці проблеми, проте потребує обізнаності в темі машинного навчання та більших ресурсів для проведення обчислень.

4.2.2 Content-based filtering

[14] Content-based filtering використовує аналіз характеристик об’єктів для надання рекомендацій. Якщо користувачу сподобався один об’єкт, то може сподобатись подібний. На відміну від item-based collaborative filtering Content-based filtering не вимагає історії взаємодії різних користувачів з об’єктами і не визначає схожість об’єктів на основі цих взаємодій. Для роботи Content-based filtering повинен мати дані лише про вподобання користувача, рекомендації якому надаються. Хоча, варто зазначити, що для хорошої роботи моделі все ж можуть бути корисні певні дані про взаємодію користувачів з об’єктами. Наприклад, може знадобитись показник популярності об’єкту чи відслідковування зміни вподобань користувачів.

Перед тим як знаходити схожість між об’єктами, потрібно визначитись із засобами формування векторів. Уявімо, що у нас є текстові описи об’єктів. Тобто завдання - сформувати з цих описів вектори, які будуть вміщувати в собі інформацію про те, як часто зустрічається кожне слово. Такі завдання виконують векторизатори, серед яких найбільш популярними є CountVectorizer і TF-IDFVectorizer.

CountVectorizer[13] - простий векторизатор. Для початку підраховується кількість слів які зустрічаються у всіх документах. Популярні слова, артиклі можна не включати в множину, бо це не має сенсу. Отже, розмір словника N - тоді кожин опис об’єкта можна представити у вигляді n-розмірного вектора.

Де значення кожного виміру - кількість разів, які слово зустрічається в даному описі. Для деяких рішень CountVectorizer занадто примітивний, тому що не враховує вагу кожного слова. Для інших рішень - вага не потрібна, в цих випадках і потрібно обирати даний алгоритм векторизації.

TF-IDFVectorizer - CountVectorizer, в якому кожне слово має свою вагу. Для її обрахування використовується формула:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

tf_{ij} = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

З формули випливає, що вага буде більше, якщо слово зустрічається в меншій кількості документів, і більше в даному документі.

Після отримання векторів відбувається розрахунок подібності.

Зрештою, об’єкти з найбільшим показником подібності рекомендуються користувачеві.

4.2.3 Content based vs Collaborative

До переваг Content-based filtering систем можна віднести наступні особливості [23, 24]:

- не вимагає великої кількості даних про взаємодію багатьох користувачів з об’єктами, адже використовує властивості об’єктів та вподобання конкретного користувача для знаходження рекомендацій. Відповідно, рекомендаційна система легко масштабується.

- користувачі отримують персоналізовані рекомендації на основі власних вподобань і аналізу властивостей об’єктів навіть якщо рекомендовані об’єкти не є популярними.

- використовує властивості об’єктів доступні та зрозумілі користувачеві,

відповідно рівень довіри до системи може зростати.

До недоліків можна віднести:

- Характеристики, за якими буде визначатися подібність прямо задаються людиною, яка створює систему. Тому якість моделі повністю залежить від знань предметної області розробника.

- Рекомендації створюються на основі поточних вподобань користувача, відповідно об’єкти іншої специфіки, які могли би сподобатись користувачу, не будуть рекомендовані.

- важко визначати нові рекомендації для малоактивних користувачів.

До переваг Collaborative систем можна віднести:

- Враховує смаки та особливості користувача за допомогою схожих на нього. Таким чином можемо отримати більш орієнтовані на користувача рекомендації

- Може генерувати нові рекомендації, які зовсім не схожі на об’єкти, які користувач вподобав у минулому. В рекомендаціях будуть об’єкти популярні серед схожих користувачів

- Розріджені матриці можуть заповнюватися за допомогою схожих користувачів

До недоліків можна віднести:

- Популярні об’єкти можуть рекомендуватися частіше (bias problem)

- З великою кількістю користувачів та об’єктів обрахунки будуть важчими порівняно з тими, які відбуваються в content-based системах

Порівнюючи ці види систем, потрібно розуміти, що Memory based та Model based Collaborative досить сильно відрізняються, що ми й розібрали у відповідному розділі.

Розглянемо порівняльний аналіз використань різних систем на практиці [25, с. 151].

Feature/challenge	*Content-based RS*	*Collaborative filtering RS*	
		Memory-based	*Model-based*
Complexity (Shi et al., 2014)	Higher	Lower (best)	Higher
Prediction quality (Sarwar et al., 1998)	Considerable	Lower	Higher (best)
Contextualisation (Adomavicius and Tuzhilin, 2015)	Higher (best)	Lower	Considerable
Engineering effort (Balabanovic and Shoham, 1997)	Considerable (best)	Higher	Higher
Over-specialisation (Adomavicius and Tuzhilin, 2005)	Lower	Considerable	Higher (best)
Scalability (Isinkaye et al., 2015)	Higher (best)	Considerable	Higher
User independence (Lops et al., 2011)	Higher (best)	Lower	Lower
Concept drift (Gama et al., 2014)	Lower (best)	Higher	Considerable
Serendipity (Ziegler et al., 2005)	Higher	Lower (best)	Lower (best)
Data sparsity (Isinkaye et al., 2015)	Lower (best)	Higher	Higher
Cold start user (Adomavicius and Tuzhilin, 2005)	Higher	Higher	Higher
Cold start item (Adomavicius and Tuzhilin, 2005	Low (best)	Higher	Higher
Cost (Burke, 2002)	High	Low (best)	High
Flexibility (Konstan and Riedl, 2012)	Considerable	Higher	Lower
Over fitting (Popescul et al., 2001)	Higher	Higher	Lower (best)
Gray sheep (Chauhan, 2016; Ghazanfar and Prugel-Bennett, 2011)	Considerable (best)	Higher	Considerable (best)
Shilling attack (Burke et al., 2005)	Lower (best)	Higher	Considerable
Prediction speed (Shani and Gunawardana, 2011)	Lower	Considerable	Higher (best)

Можна побачити наступне:

- content based системи масштабувати легше ніж будь-яку з collaborative.

- content-based система менш залежна від користувачів

- проблеми “холодного старту” для знаходження схожих об’єктів в content-based системах немає, адже не враховуються користувачі.

- model-based collaborative менше страждає проблемою overfitting, це коли рекомендаційна система стає занадто орієнтованою на тренувальних/початкових даних і в результаті гірше працює з новими.

- вартість memory-based системи менша, ніж content-based

- швидкість здійснення передбачень у порядку спадання: model-based, memory based, content based.

4.2.4 Гібридні системи

Гібридні системи комбінують різні content-based та collaborative моделі

для створення передбачень. Гібридні моделі можуть бути різні[13]:

- Окремо реалізовують content-based та collaborative підсистеми для надання рекомендацій

- Імплементують content-based техніки в collaborative, і навпаки

В розділі з аналогами рекомендаційних систем була детально розглянута гібридна багатоскладова система Netflix.

Подібні системи дозволяють користуватись перевагами обох систем, в деяких аспектах нівелюючи недоліки.

Висновок

Отже, в даному розділі спершу було розглянуто дані, які можуть знадобитися для написання рекомендаційних систем. Наприклад, оцінки користувачів, історія пошуку, переглянуті товари, куплені чи додані в кошик. Після чого визначені методи збору, такі як логування, зберігання в бд. Визначені необхідні кроки при підготовці даних, такі як нормалізація, виявлення аномалій, видалення дублікатів.

Для знаходження схожих об’єктів було описано основні популярні метрики подібності: косинусна подібність, евклідова відстань, кореляція Пірсона.

Детально розглянуті види рекомендаційних систем, а саме : collaborative, content-based, hybrid. Здійснено порівняльний аналіз collaborative та content-based систем, в якому визначено переваги та недоліки кожної. Визначено підтипи collaborative систем: memory-based та model-based, здійснено порівняльний аналіз між ними, в якому визначено переваги та недоліки кожного підходу. Визначено підтипи memory-based систем: user-based та item-based, пояснено їх роботу та принципову різницю між ними. Визначено такі підходи до model-based систем, як кластеризація та матрична факторизація. Розглянуто різні види кластеризацій та факторизацій відповідно.

Визначено якими можуть бути гібридні системи та як працювати.

Загалом, описані різні види систем, їх переваги та недоліки.

Розділ 5. Реалізація рекомендаційної системи

5.1 Технології

Для реалізації рекомендаційної системи було обрано мову Python, так як вона має потужні бібліотеки для роботи з даними і написання рекомендаційних систем та алгоритмів машинного навчання. Зокрема будуть використані бібліотеки pandas, numpy, sklearn, surprise

5.2 Опис системи

В зв'язку з тим, що гібридна система може використовувати переваги і content-based І collaborative систем, буде реалізована гібридна система з декількома складовими різних типів:

- Гібридна система для пошуку схожих за описом товарів, які сподобаються користувачеві
- Колаборативна система для надання рекомендацій
- Система для пошуку схожих товарів за характеристиками товарів, включаючи ціну

5.3 Реалізація системи

Collaborative система для рекомендацій

Варто відзначити, що як мінімум перший час, система буде працювати з невеликим набором даних. Виходячи з цього немає сенсу використовувати model-based техніки одразу. Memory-based техніки будуть ефективними. Крім того, матриці оцінок будуть визначені так, що не будуть сильно розрідженими. Щоб перейти від memory-based техніки до model-based достатньо змінити

невелику частину коду. Тому в майбутньому, при необхідності, ми зможемо перейти на model-based підсистему

Отже, я обрав систему, яка працює на основі матриці рейтингів, і використовує weighted mean алгоритм та косинусну подібність. Для надання k рекомендацій конкретному користувачеві на основі його подібності з іншими розраховуються всі оцінки, які він міг би поставити, після чого знаходяться k продуктів з найвищими потенційними оцінками.

Для передбачення використовується weighted mean (середнє-зважене) - memory-based техніка.

В нашому випадку використовується user based варіант memory based підходу, тобто оцінки передбачаються на основі подібності між користувачами.

Ми знаходимо матрицю подібності між користувачами за допомогою метрики косинусної подібності між векторами оцінок.

```
r_matrix = pd.read_csv('rates.csv')
r_matrix_dummy = r_matrix.copy().fillna(0)
cosine_sim = cosine_similarity(r_matrix_dummy, r_matrix_dummy)
```

За допомогою бібліотеки pandas файл rates.csv зчитується в матрицю. Після цього для знаходження подібності відсутні значення заповнюємо нулями, і знаходимо матрицю подібності.

Визначаємо функцію для передбачення оцінки

```
def cf_user_wmean(user_id, prod_id):

    sim_scores = cosine_sim.iloc[int(user_id)]
    m_ratings = r_matrix.iloc[:, prod_id]

    idx = m_ratings[m_ratings.isnull()].index
    m_ratings = m_ratings.dropna()
    sim_scores = sim_scores.drop(idx)

    wmean_rating = np.dot(sim_scores, m_ratings) / sim_scores.sum()

    return wmean_rating
```

Беремо вектор подібності користувача, для якого здійснюються рекомендації, з усіма іншими.

```
sim_scores = cosine_sim.iloc[user_id]
```

Отримуємо оцінки усіх користувачів, поставлені на продукт

```
m_ratings = r_matrix.iloc[:, prod_id]
```

При обрахунку середнього зваженого користувачі, які не оцінили продукт не враховуються.

```
idx = m_ratings[m_ratings.isnull()].index
m_ratings = m_ratings.dropna()
sim_scores = sim_scores.drop(idx)
```

Знаходимо добуток векторів подібності та оцінок, і ділимо його на суму елементів вектора подібності. Таким чином для кожної оцінки буде своя вага залежно від показника подібності користувачів.

```
wmean_rating = np.dot(sim_scores, m_ratings) / sim_scores.sum()
```

Тепер розглянемо функцію надання рекомендацій

```
def get_top_k_rec(_id, k):
    rates = []
    for i in range(0, len(r_matrix.iloc[0])):
        rates.append(cf_user_wmean(_id, i))
    sorted_indices = np.argsort(rates)[::-1]
    sorted_indices = sorted_indices[:k]
    sorted_pairs = [(i, rates[i]) for i in sorted_indices]
    return sorted_pairs
```

Для кожного продукту ми знаходимо передбачувану оцінку. Після цього сортуємо їх і повертаємо k відсортовані пари елементів (індекс, оцінка) з найвищими рейтингами.

Гібридна система для пошуку схожих за описом товарів, які сподобаються користувачеві

Для початку розберімося чому ця підсистема може вважатись гібридною

і як вона працює. Гібридна тому що за основу взята модель content-based системи, а в допоміжних цілях використовується collaborative filtering.

Алгоритм роботи підсистеми:

1) Знаходимо k схожих за описом продуктів (content-based)

2) За допомогою подібності між користувачами, розрахованої на основі їх оцінок, здійснюємо передбачення, як користувач оцінить продукти отримані в першому пункті (collaborative). З k продуктів обираємо n з найвищими потенційними оцінками (n<k).

Для реалізації першого пункту, а саме знаходження схожих за описом товарів потрібно перевести текстові описи у вектори. Для цього доцільно використати tf-idf векторизатор, в якому слова мають вагу на відміну від count vectorizer. Тобто, чим рідше слово зустрічається в усіх документах, і чим частіше в нашому, тим більше воно впливає на обрахунок подібності.

```
def find_products_similarity(products):
    # Assuming the `products` variable is a list of dictionaries with a `description` key
    descriptions = [product['description'] for product in products]
    tfidf = TfidfVectorizer(stop_words='english')
    tfidf_matrix = tfidf.fit_transform(descriptions)
    cosine_sim_desc = linear_kernel(tfidf_matrix, tfidf_matrix)
    return cosine_sim_desc
```

Дана функція дістає описи продуктів, після чого за допомогою метрики косинусної подібності обраховує подібність описів.

```
cosine_sim2 = linear_kernel(tfidf_matrix, tfidf_matrix)
```

Після цього збираємо k найбільш подібних описів.

```
def content_recommender(prod_id, products, k):
    products_sim = find_products_similarity(products)
    idx = prod_id
    print(products_sim)
    sim_scores = list(enumerate(products_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:k + 1]
    return sim_scores
```

Далі за допомогою описаного в попередній підсистемі weighted mean

обраховуємо як користувач оцінить ці продукти, і обираємо n, з найкращими передбачуваними оцінками.

```
def get_top_k_rec_from_score_tuples(k, scores):
    rates = []
    user_id = 0
    for i in range(0, len(scores)):
        rates.append([scores[i][0], cf_user_wmean(user_id, scores[i][0])])
    sorted_rates = sorted(rates, key=lambda x: x[1], reverse=True)
    sorted_rates = sorted_rates[:k + 1]
    return sorted_rates
```

Функція приймає на вхід кількість елементів, яку треба вибрати, та масив елементів (індекс, подібність) . Далі в масив rates збираються пари (індекс, оцінка на основі колаборативної фільтрації). Зрештою rates сортується за оцінками, та знаходиться n продуктів з найвищими потенційними рейтингами.

Система для пошуку схожих товарів за характеристиками

Враховуються такі характеристики як: категорія, опис, ціна. В даному випадку для векторизації текстових даних доцільно використати countVectorizer, який не враховує вагу слів. Якщо є багато товарів однієї категорії, це не означає що категорія стає менш важливим показником. До того ж для балансу між довгим описом і категорією може знадобитись дублювати назву категорії декілька разів, адже категорія є важливішою за якесь спільне слово у описі.

Для використання векторизатора треба кожному продукту поставити у відповідність стрічку, яка міститиме його характеристики.

```
def format_for_vectorization(products):
    strings = []
    for prod in products:
        words = prod['description'].split()
        # Count the number of words
        word_count = len(words)
        strings.append((prod['category'] + ' ') * word_count + prod['description'])
    return strings
```

Стрічка складається з опису товару та назви категорії, яка повторюється

n разів, де n - кількість слів у описі. Таким чином для категорії штучно задана велика вага, і категорія відіграватиме важливу роль при знаходженні подібності.

```
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(strings_from_products)
cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

Знаходимо матрицю подібності між векторами [категорія опис колір] A. Не має сенсу додавати ціну для строкового порівняння. Замість цього утворимо матрицю коефіцієнтів B, яка буде показувати подібність цін.

```
def get_price_coeff_matrix(price_list):
    mean = np.mean(price_list)
    n = len(price_list)
    diff_matrix = [[0] * n for _ in range(n)]

    for i in range(n):
        for j in range(n):
            m = np.max([1 - abs(price_list[i] - price_list[j]) / (mean * 2), 0])
            diff_matrix[i][j] = m
    print(diff_matrix)
    return diff_matrix
```

Тут аргументом є список цін продуктів

```
price_list = [product['price'] for product in products]
price_diff_coeff = get_price_coeff_matrix(price_list)
```

Елемент в матриці подібності цін розраховується за підібраною формулою. Різниця цін двох товарів ділиться на середнє значення ціни усіх товарів помножене на 2. Так ми отримуємо щось накшталт коефіцієнту різниці цін. Віднімаємо його від одиниці і отримуємо коефіцієнт подібності. Згідно з формулою він може бути менше одиниці, але це не підходить для матриці подібності цін, тому якщо значення менше нуля, ставимо 0.

Щоб отримати фінальну матрицю подібності характеристик товарів домножимо кожен елемент A(I,j) на відповідний елемент B(I,j).

```
cosine_sim = np.multiply(cosine_sim2, price_diff_coeff)
```

Функція рекомендації працює наступним чином:

1) Дістаємо з матриці подібності вектор подібності продукту з усіма іншими. Кожен елемент вектора - індекс продукту в матриці та показник подібності.

2) Сортуємо за показником подібності

3) Повертаємо k елементів (індекс, подібність) з найвищим показником

```
def content_recommender(product_id, products, k):
    idx = product_id
    products_sim = find_products_similarity(products)
    print("PRSIM")
    print(products_sim)

    sim_scores = list(enumerate(products_sim[idx]))

    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:1+k]

    return sim_scores
```

5.4 Взаємодія рекомендаційної системи та онлайн-магазину

У відповідних розділах описані реалізація онлайн-магазину та рекомендаційної системи. Потрібно розібратись як вони поєднані між собою.

За допомогою Flask був створений сервер на Python, який надає API для взаємодії з рекомендаційною системою. Flask сервер обробляє наступні маршрути :

1)

```
@app.route('/collaborative_recommendations')
```

Для отримання рекомендацій від collaborative підсистеми.

За маршрутом відпрацьовує відповідна функція

```
def get_k_rec():
    user_idx = int(request.args.get('user_idx'))
    sorted_pairs = userBasedColaborative.get_top_k_rec(user_idx, 3)
    sorted_pairs_as_dicts = [{'id': int(pair[0]), 'score': float(pair[1])} for pair in sorted_pairs]

    return jsonify(sorted_pairs_as_dicts)
```

Можна побачити, що з параметрів запиту дістається індекс користувача, для якого потрібно сформувати рекомендації

Сервер повертає відповідь у вигляді пар (індекс, подібність) у форматі json.

2)

```
@app.route('/similar_description')
```

Для знаходження товарів зі схожим описом за допомогою гібридної системи.

```
def get_k_similar_by_desc():
    prod_idx = int(request.args.get('prod_idx'))
    products = json.loads(request.args.get('products'))
    user_idx = int(request.args.get('user_idx'))

    scores = tfIdf.content_recommender(prod_idx, products, 5)
    sorted_pairs = tfIdf.get_top_k_rec_from_score_tuples(3, scores, user_idx)

    sorted_pairs_as_dicts = [{'id': int(pair[0]), 'score': float(pair[1])} for pair in sorted_pairs]

    return jsonify(sorted_pairs_as_dicts)
```

Можна побачити, що крім індексу користувача в параметрах передається індекс продукту та всі продукти для знаходження подібних при content-based фільтрації .

3)

```
@app.route('/similar_features')
```

Для content-based системи на основі характеристик продукту.

```
def get_k_similar_by_all():
    prod_idx = int(request.args.get('prod_idx'))
    products = json.loads(request.args.get('products'))

    sorted_pairs = countVectorizerRecommender.content_recommender(prod_idx, products, 3)

    sorted_pairs_as_dicts = [{'id': int(pair[0]), 'score': float(pair[1])} for pair in sorted_pairs]

    return jsonify(sorted_pairs_as_dicts)
```

Тут достатньо індексу продукту та власне продуктів.

Від клієнта йдуть запити на Express сервер онлайн-магазину, і звідти вже йдуть запити на Flask сервер для отримання відповідних видів рекомендацій.

Наприклад,

```
export const fetchRecommendations = async (user_id) => {

    const {data} = await $authHost.get( url: 'collaborative_rec/',  config: {
        params: {
            user_id: user_id
        }
    })
    console.log("REVIEWS " + data)
    return data
}
```

Запит на сервер

```
router.get( path: '/collaborative_rec', recController.getCollaborative)
```

Створено recRouter на сервері для обробки запитів, пов’язаних з рекомендаціями, та recController для виконання логіки запитів.

```
const {data} = await axios.get( url: 'http://localhost:5000/collaborative_recommendations',  config: {
        params: {
            user_idx: user_idx
        }
    }
)
```

Метод getCollaborative recController контролера, який містить запит на відповідний endpoint Flask сервера.

Користувачі містять унікальну стрічку id, а в запит передається саме індекс.

```
const users = await User.find()
const user_idx = users.findIndex(user => (user._id.toString().trim()) === user_id.trim() )
```

Для отримання індексу користувача в матриці оцінок, знаходиться його індекс серед усіх користувачів бази.

У відповідь від сервера приходить пара (id,score), де id - індекс продукту в базі даних.

```
const idx = data.map(data => data.id)
const products = await Product.find()
const products_from_indexes = idx.map(id => products[id])
```

Для відправлення рекомендацій на клієнт збираються індекси продуктів,

після чого обираються відповідні об’єкти з бази даних.

```
res.json(products_from_indexes)
```

Висновок

В даному розділі були обрані технології для розробки рекомендаційних систем, а саме бібліотеки для Python, які спрощують реалізацію рекомендаційних систем та мають потужний інструментарій для впровадження алгоритмів машинного навчання: pandas, numpy, sklearn, surprise.

Було обрано для реалізації гібридну комплексну систему, визначено технічне завдання для неї, включаючи підсистеми, які увійдуть до неї:

- Гібридна система для пошуку схожих за описом товарів
- Collaborative система для надання рекомендацій
- Content-based система для пошуку схожих товарів за характеристиками

Описана реалізація системи, для кожної підсистеми пояснено вибір рекомендаційної моделі та алгоритмів, потрібних для її реалізації.

В розділі описано Flask сервер та API, яке він надає для взаємодії з рекомендаційною системою, наведено приклад здійснення запиту для отримання рекомендацій.

Висновок

За мету ставилось дослідити методи розробки онлайн-магазинів та рекомендаційних систем. Завданням було написання онлайн-магазину для продажу товарів, пов’язаних з аніме та розробка рекомендаційної системи до нього.

В першому розділі розглянуто актуальність онлайн-магазинів аніме товарів і систем рекомендацій у контексті сьогодення. Було визначено важливість рекомендаційних систем для конкурентноспроможності на ринку.

В другому розділі описано теорію, необхідну для реалізації онлайн-магазину, включаючи різні архітектури та підходи до розробки веб-додатків. Обрано клієнт-серверну архітектуру та пояснено взаємодію між клієнтом та сервером в мережі. Крім того проведено порівняння монолітної та мікросервісної архітектури, визначено переваги та недоліки кожної. Вирішено розробляти основну складову онлайн-магазину як моноліт, проте рекомендаційну система винести в окремий мікросервіс.

В третьому розділі спершу аргументовано вибір технологій для написання додатку: React, Node, Mongo. Описано переваги обраних технологій. Після цього описана модель даних, реалізація клієнтської та серверної частини, взаємодія між ними.

В четвертому розділі спершу розглянуто збір та обробку даних для подальшого використання в системах рекомендацій. Далі описано collaborative, content-based та гібридні системи. Розглянуті та описані підтипи кожної з систем. Проведено порівняльний аналіз для розуміння яку систему в яких випадках доречно використовувати.

В останньому п’ятому розділі пояснюється вибір технологій для створення рекомендаційної системи. Описано завдання, які мають бути виконані та пояснено структуру системи. Після цього описано реалізацію, пояснено вибір моделей для кожної з підсистем.

На виході отримали онлайн-магазин для продажу аніме-товарів з багатоскладовою гібридною рекомендаційною системою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Anime market size, share & growth analysis report, 2030. *Market Research Reports & Consulting | Grand View Research, Inc.* URL: https://www.grandviewresearch.com/industry-analysis/anime-market (date of access: 20.05.2023).

2. Recommendation engine market report, 2021-2028. *Market Research Reports & Consulting | Grand View Research, Inc.* URL: https://www.grandviewresearch.com/industry-analysis/recommendation-engine-market-report (date of access: 20.05.2023).

3. Chong D. Deep dive into netflix's recommender system. *Medium*. URL: https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48 (date of access: 20.05.2023).

4. Difference between Client-Server and Peer-to-Peer Network - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/difference-between-client-server-and-peer-to-peer-network/ (date of access: 20.05.2023).

5. Web application architecture: the latest guide 2022. *ClickIT*. URL: https://www.clickittech.com/devops/web-application-architecture/ (date of access: 20.05.2023).

6. SOAP vs. REST: A Look at Two Different API Styles | Upwork

7. Beznos M. Microservices vs monolithic architecture: how each of them can impact your business?. *Software Development Company - N-iX*. URL: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/ (date of access: 20.05.2023).

8. Node.js. *Node.js*. URL: https://nodejs.org/uk (date of access: 20.05.2023).

9. Vashchenko M. Чем на самом деле является node.js?. *Хабр*. URL: https://habr.com/ru/articles/420123/ (дата звернення: 20.05.2023).

10. ReactJS Tutorials - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/reactjs-tutorials/ (date of access: 20.05.2023).

11. ReactJS Virtual DOM - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/reactjs-virtual-dom/ (date of access: 20.05.2023).

12. Cookie vs Token authentication. *Engineering Education (EngEd) Program | Section*. URL: https://www.section.io/engineering-education/cookie-vs-token-authentication/ (date of access: 20.05.2023).

13. Banik R. Hands-On Recommendation Systems with Python: Start building powerful and personalized, recommendation engines with Python. Packt Publishing, 2018. 146 p.

14. Roy A. Introduction to recommender systems- 1: content-based filtering and collaborative filtering. *Medium*. URL: https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421 (date of access: 20.05.2023).

15. Clustering in machine learning - geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/clustering-in-machine-learning/ (date of access: 20.05.2023).

16. Садовников П. Интересные алгоритмы кластеризации, часть вторая: dBSCAN. *Хабр*. URL: https://habr.com/ru/articles/322034/ (дата звернення: 20.05.2023).

17. Hierarchical clustering in data mining - geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/hierarchical-clustering-in-data-mining/ (date of access: 20.05.2023).

18. Swarndeep Saket J, Dr. Sharnil Pandya. An overview of partitioning algorithms in clustering techniques. International Journal of Electrical and Computer Engineering

19. Difference between STING and OPTICS - GeeksforGeeks. *GeeksforGeeks*.

URL: https://www.geeksforgeeks.org/difference-between-sting-and-optics/ (date of access: 20.05.2023).

20. Singular value decomposition (SVD) - geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/singular-value-decomposition-svd/ (date of access: 20.05.2023).

21. Non-Negative matrix factorization - geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/non-negative-matrix-factorization/ (date of access: 20.05.2023).

22. P. H. Aditya, Indra Budi, Qorib Munajat. A comparative analysis of memory-based and model-based collaborative filtering on the implementation of recommender system for E-commerce in Indonesia: A case study PT X.

23. Content-based filtering advantages & disadvantages | machine learning | google for developers. *Google for Developers*. URL: https://developers.google.com/machine-learning/recommendation/content-based/summary (date of access: 20.05.2023).

24. What is a content-based recommendation system in machine learning?| analytics steps. *Analytics Steps - A leading source of Technical & Financial content*. URL: https://www.analyticssteps.com/blogs/what-content-based-recommendation-system-machine-learning (date of access: 20.05.2023).

25. Taneja A., Arora A. Recommendation research trends: review, approaches and open issues. *International journal of web engineering and technology*. 2018. Vol. 13, no. 2. P. 123. URL: https://doi.org/10.1504/ijwet.2018.092831 (date of access: 20.05.2023).