Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ВИКОРИСТАННЯ МІРКОСЕРВІСНОЇ АРХІТЕКТУРИ ДЛЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

Текстова частина до курсової роботи

за спеціальністю «Комп’ютерні науки» 122

Керівник курсової роботи

ас. Бабич Т. А.

_____

(підпис)

Виконав студент 3 курсу

Куценко А.О.

«____»_____2023 р.

Київ 2023

ЗМІСТ

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ 2
Календарний план виконання курсової роботи 3
Вступ 4
1. Дослідження 6
1.1 Архітектури систем 6
1.1.1 Моноліт 7
1.1.2 Мікросервіси 8
1.1.3 Серверлесс 9
1.2 Рекомендаційні системи 10
1.2.1 Підходи вирішення задачі рекомендації 10
1.2.1.1 Колаборативна фільтрація 11
1.2.1.2 Фільтрація на основі вмісту 13
1.2.1.3 Гібридна фільтрація 14
1.2.1.4 Демографічна фільтрація 14
1.2.2 Проблематика рекомендаційних систем 15
1.2.3 Приклад використання алгоритмів рекомендаційних систем 15
1.3 Сучасні технології для побудови backend частини веб-додатків 17
1.3.1 Фреймворки для створення backend частини веб-додатку 17
1.3.2 СКБД 20
1.3.2.1 Реляційні СКБД 20
1.3.2.2 Нереляційні СКБД 21
1.3.3 Брокери повідомлень 22
2. Реалізація рекомендаційної системи за допомогою мікросервісної архітектури 24
2.1 Опис проєкту 24
2.2 Вибрані технології 25
2.3 Модель даних 26
2.4 Алгоритм рекомендацій 28
2.5 Архітектура 34
Висновок 37
Список використаної літератури 38

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики,

к.ф.-м.н., доц. Гороховський С.С

_____

(підпис)

«___» _____ 2023 р

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту 3-го курсу, факультету інформатики Куценку Андрію

Тема: Використання мікросервісної архітектури для рекомендаційних систем

Вихідні дані:

Зміст ТЧ до курсової роботи:

Зміст

Вступ

1. Дослідження

2. Реалізація рекомендаційної системи за допомогою мікросервісної архітектури

Висновок

Список використаної літератури

Дата видачі «____»_____ 2023 р. Керівник _____

(підпис)

Завдання отримала _____

(підпис)

Календарний план виконання курсової роботи

Тема: Використання мікросервісної архітектури для рекомендаційних систем:

№	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання індивідуального завдання	14.02.2023	
2.	Дослідження матеріалів для курсової роботи	06.03.2023	
3.	Реалізація системи рекомендацій	03.04.2023	
4.	Написання текстової частини	24.04.2023	
5.	Надання роботи керівнику на перевірку	10.05.2023	
6.	Виправлення на основі перевірки	11.05.2023	
7.	Кінцева перевірка роботи	12.05.2023	
8.	Захист курсової роботи	22.05.2023	

Куценко А.О. _____

Бабич Т.А. _____

«____» _____ _____р.

Вступ

У сучасному світі інформаційні технології є невід’ємною частиною нашого життя. Кожен день ми користуємося різноманітними додатками на смартфонах, комп’ютерах та інших електронних пристроях. Ці додатки роблять наше життя простішим та зручнішим, допомагаючи нам з різними задачами, від комунікації з друзями до роботи та навчання.

Розвиток айті додатків є актуальною темою в наш час, оскільки технологічний прогрес не стоїть на місці, а наші потреби та вимоги постійно зростають. Розробники повинні працювати над тим, щоб створювати продукти, які задовольняють потреби користувачів та відповідають сучасним вимогам.

Одним з найважливіших аспектів сучасних веб-застосунків є соціальна важливість продукту. Тобто наскільки корисним і ефективним є додаток для користувачів, наскільки він допомагає будувати соціальні взаємовідносини та по-можливості вирішувати потреби суспільства. Наприклад сервіси онлайн-знайомств допомагають спростити пошук друзів, відносин, тощо.

Невід'ємною частиною сучасних веб-додатків стала система рекомендацій. Рекомендації - це те, що персоналізує досвід користувача з продуктом, допомагає задовольнити його потреби та зацікавити в подальшому використанні сервісу. Вони не просто заохочують, а й фільтрують інформацію, яка не цікавить, або не подобається людині, таким чином приносячи позитивні емоції та бажання продовжити.

В сучасному світі для користувачів не стільки цікава візуальна частина сайту, або сервісу, а як користь, яку він принесе, або інформацію, яку в кінцевому результаті отримає особа. Тому наразі компанії вкладають чимало зусиль для розробки якісних рекомендаційних систем, які будуть максимально задовольняти бажання людей, та показувати контент, котрий співпадає з вподобаннями особистості.

Сучасні веб-додатки стають сильно ресурсно залежними зі зростанням користувачів та контенту, а й відповідно навантаження на рекомендаційні системи також збільшується за рахунок великих обчислень. Тому виробники продуктів поступово збільшують свої потужності і для нормального функціонування додатку - змінюють його архітектуру, і одна з таких - мікросервіси.

1. Дослідження

Ця частина присвячена дослідженню які бувають типи побудови архітектури сучасних веб-додатків, різновидів підходів впровадження рекомендаційних систем та сучасних технологій котрі використовуються для реалізації перерахованих речей.

1.1 Архітектури систем

Впровадження архітектурних рішень таких, які використовуються в сьогодення - є доволі явищем. Але ж в чому полягає причина їх створення?

Після популяризації соціальних мереж - кількість користувачів інтернету значно почала стрімко зростати. На квітень 2023-ого року існує приблизно 5.18 млрд користувачів інтернету з яких біля 4.8 млрд складають унікальні користувачі соціальних мереж, що дорівнює майже 60% населенню планети.[1]

В зв'язку зі зростанням кількості людей в інтернеті - з'явився попит на інші сервіси, які могли б бути оцифровані. Тому з кожним роком навантаження на різні платформи ставало все більше, кількість даних користувачів - зростала і вже неможливо було зберігати, або опрацьовувати інформацію тільки на одному комп'ютері(машині), тому розробниками впроваджувались різноманітні рішення, включно створення нових архітектур веб-додатків.

Основні чинники, які змінюються під час зміни архітектури:

1. Швидкість розробки
2. Властивість масштабуватися
3. Продуктивність
4. Обробка та збереження даних
5. Безпека
6. Вартість

Отже побудова архітектури додатку — це комплекс заходів, який спрямований на те, щоб чітко визначити, як буде побудована система.[2]

1.1.1 Моноліт

Монолітна архітектура - це модель в котрій рішення представлене в одному модулі, який працює самостійно та немає залежностей від інших додатків/модулів.

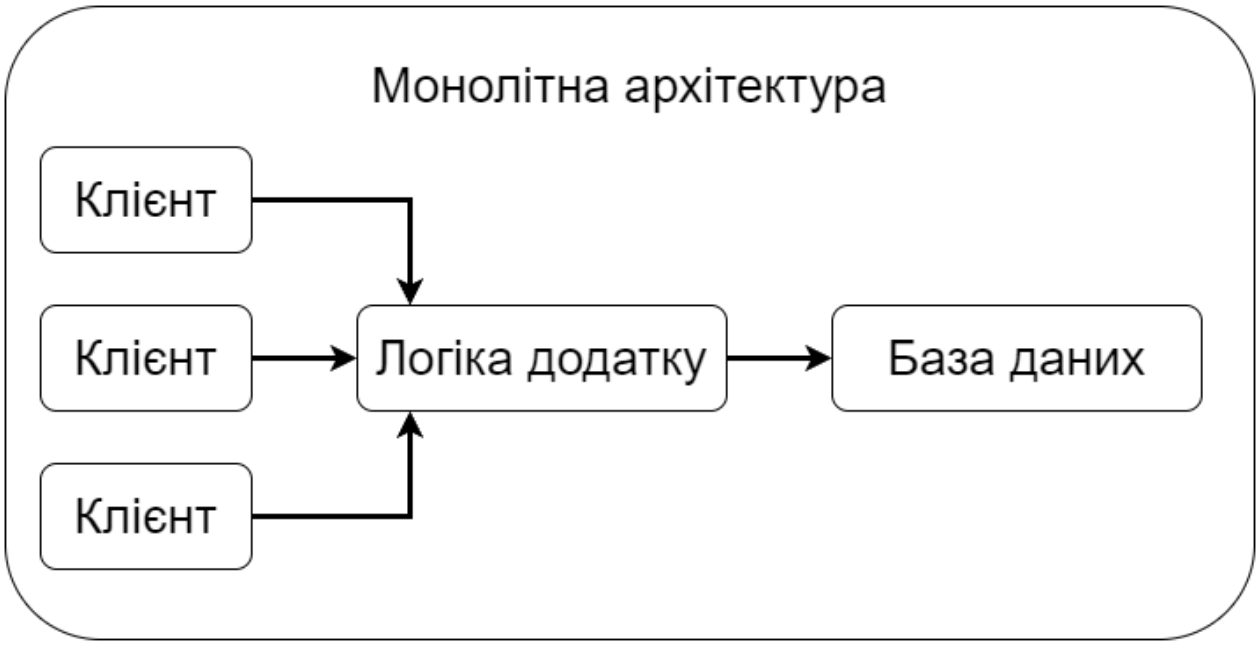

Рисунок 1.1 – Модель монолітної архітектури

Переваги:

1. Просте розгортання. Оскільки моноліти зазвичай мають тільки одну точку входу - це спрощує процес розгортання.
2. Швидка розробка. Оскільки моноліт являє собою єдину логіку - розробка його стає набагато швидшою.
3. Зручна відладка. Через відсутність додаткових залежностей від інших сервісів - відладка стає набагато швидшою та зручнішою.

Недоліки:

1. Складне масштабування. Складність полягає в тому, що якщо навантаження зростає лише на певний участок логіки - ми не можемо масштабувати його окремо і приходиться весь моноліт.
2. Надійність. Якщо хоча б один участок логіки виходить з ладу - виходить весь моноліт.
3. Зміна та оновлення технологій. Зміна, або оновлення технологій в моноліти - означає створення всього додатку заново.
4. Недостатня гнучкість. Зазвичай зміна однієї частини логіки призводить до зміни інших.

1.1.2 Мікросервіси

Мікросервісна архітектура - це рішення, яке базується на розподілі модулів на окремі системи, які спілкуються між собою за допомогою повідомлень.[2]

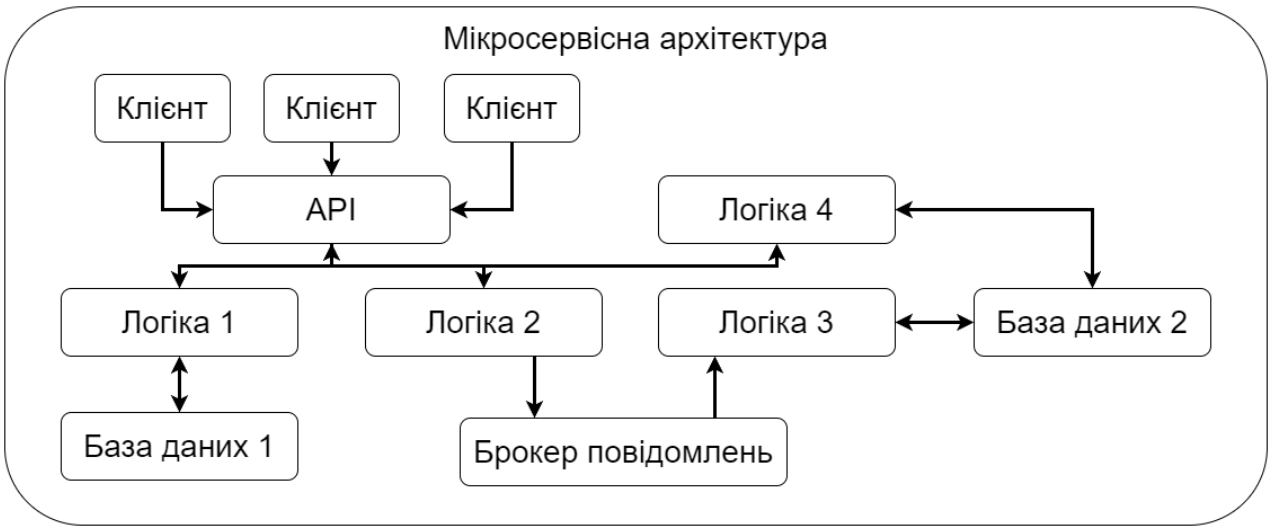

Рисунок 1.2 – Приклад мікросервісної архітектури

Переваги:

1. Гнучкість. Через те що, кожен модуль являє собою незалежний сервіс - зміна в ньому не впливатиме на іншу систему.
2. Масштабування. В разі зростання навантаження на певний модуль, то через його самостійність ми можемо масштабувати тільки його.

3. Гнучкість технологій. Кожен модуль може реалізовуватися та оновлюватися незалежно від технологій які використовуються в інших сервісах.
4. Надійність. Через масштабування сервісів - вихід одного з екземплярів зазвичай не виводить з ладу всю систему.

Недоліки:

1. Повільна та складна розробка. З'являється додаткова складність узгодження всіх сервісів та інформацією, яка між ними обмінюється.
2. Відладка. Збій в одному сервісі може залежати від іншого, тому інколи це ускладнює процес відладки.
3. Розгортання. Через складність системи - ускладнюється процес розгортання початкового проєкту, та подальше погодження з новими сервісами.

1.1.3 Серверлесс

Серверлесс не є зовсім є архітектурою як таковою, хоча його виділяють окремо. В першу чергу це про керування ресурсами та масштабування.

В основоположеннях серверлессу лежить мікросервісна архітектура, але завдяки хмарним технологіям на кожен сервіс ми можемо динамічно зменшувати, або збільшувати ресурси в залежності від його навантаження.

Переваги:

1. Гнучкість. Дублює мікросервіси
2. Надійність. Так само як у мікросервісах.
3. Легкість використання. Через використання хмарних технологій - менш проблематичне розгортання і масштабування.
4. Ціна. Через використання тільки тих потужностей, які потрібні - відсутня переплата.

Недоліки:

1. Відладка. Така ж проблема як у мікросервісах.
2. Залежність від хмари. Переніс додатку з однієї хмари на іншу - довгий та складний процес.

1.2 Рекомендаційні системи

Рекомендаційна система — підклас системи фільтрації інформації, яка будує рейтинговий перелік об'єктів, яким користувач може надати перевагу. [3]

Через стрімке поширення доступу до інтернету - кількість контенту (текстового, відео, аудіо, тощо) зростає і дуже тяжко самостійно виділити той, який може сподобатися. Тому на допомогу приходять рекомендаційні системи. Вони виступають фільтром, який поміж всієї інформації спробує виділити ту, яка може зацікавити.

Рекомендації можуть будуватися на різноманітній взаємодії користувача з контентом(уподобання, коментарі, час перебування на сторінці, тощо), на внутрішніх опитуваннях на платформах, на запитах в пошукових системах та на безлічі інших метрик.

Наразі система рекомендацій є невід'ємною частиною більшості веб-застосунків, котрі включають в собі соціальну мету. Дані системи допомагають утримати увагу користувача на платформі, тим самим приносячи користь розробникам.

1.2.1 Підходи вирішення задачі рекомендації

Через різноманітність задач - було винайдено багато рекомендаційних систем, але зазвичай виділяють ці як основні:

1. Колаборативна фільтрація - базується на знаходженні подібності між користувачами або товаром. Ідея полягає в тому, щоб рекомендувати товари, які сподобалися іншим користувачам зі схожими інтересами.
2. Фільтрація на основі вмісту - використовує інформацію про властивості товару (наприклад, жанр фільму, автор книги, тощо) і знаходить подібні товари для рекомендацій.
3. Гібридна фільтрація - комбінація спільної та гібридної фільтрації. Зазвичай використовуються ваги для кожного алгоритму, щоб дати більш точні рекомендації.
4. Демографічна фільтрація - використовує інформацію про вік, стать, географічну локацію та інші демографічні дані, щоб рекомендувати товари, які можуть бути цікавими для певної групи користувачів.

1.2.1.1 Колаборативна фільтрація

Ідеологія колаборативної фільтрації полягає в тому, що найкращі рекомендації для користувача будуть отримані від інших, хто має схожі смаки.

Зазвичай процес колаборативної фільтрації складається з таких кроків:

1. Користувач показує свої вподобання за допомогою взаємодії з контентом системи. В ітозі ці вподобання можна сприймати як цікавість до користувача до певної ділянки.
2. Далі система порівнює вподобання одного до уподобань інших користувачів та знаходить найбільш схожих.
3. Для схожих користувачів - система рекомендує контент який не був ще показаний одному, але іншими вже високо оцінений.

Існують такі типи колаборативної фільтрації:

1. Заснований на пам’яті.

   Цей підхід використовує дані про рейтинг користувача для розрахунку схожості між користувачами або предметами. Це був початковий підхід, що використовувався в багатьох торгових системах. Він ефективний і простий у реалізації. [4] Обчислюється як сукупність оцінок деяких схожих користувачів щодо елемента.

2. Заснований на сусідстві

   Алгоритм на основі сусідства обчислює схожість між двома користувачами або елементами та створює прогноз для користувача, беручи середньозважене значення всіх оцінок.

   Один з недоліків - продуктивність такого алгоритму є малою при розріджених даних.

3. Заснований на моделі

   Даний підхід надає рекомендації, вимірюючи параметри статистичних моделей для оцінок користувачів, побудованих за допомогою таких методів як, метод байєсівських мереж, кластеризації, латентно-семантичної моделі , такі як сингулярний розклад, імовірнісний латентно-семантичний аналіз, прихований розподіл Діріхле і марковський процес вирішення на основі моделей. Моделі розробляються з використанням інтелектуального аналізу даних, алгоритмів машинного навчання, щоб знайти закономірності на основі навчальних даних.[4]

4. Гібридний

   Це тип поєднує в собі заснований на сусідстві та на моделі, тим самим позбуваючись проблеми недостатньої кількості даних про користувача. Але він є складним в розробці та дорогий в обслуговуванні.

Проблеми:

1. Масштабованість. З збільшенням даних в системі - розрахунки стають занадто складними. Наприклад маючи мільйон користувачів O(M) і сто мільйонів постів O(N), то складність обрахунків становитиме O(MN).
2. Шахрайство. За допомогою спеціальної гарної оцінки певного контенту та поганої іншого - легко змінювати рекомендації інших користувачів.
3. Проблема запуску. Як і для більшості систем - недостатність даних робить складнішим успішне прогнозування.

1.2.1.2 Фільтрація на основі вмісту

Фільтрація на основі вмісту використовує властивості контенту, щоб рекомендувати інший контент, схожий на те, що подобається користувачу на основі минулих взаємодій з системою.[5]

Переваги:

1. Незалежність від інших. Даний алгоритм може надавати рекомендації які стосуються конкретного користувача незалежно від уподобань інших і це полегшує масштабування при великій кількості користувачів.
2. Ексклюзивний підхід. В даному випадку ми враховуємо індивідуальні інтереси користувача, тому можемо йому рекомендувати те, що цікавить малу кількість інших.

Недоліки:

1. Залежність від властивостей. Успішність даного алгоритму напряму залежить від наскільки добре описані властивості контенту.
2. Відсутність розширення інтересів. Через специфіку індивідуального підходу - матриця інтересів користувача буде змінюватися дуже повільно, бо не будуть враховуватися інтереси схожих користувачів.

1.2.1.3 Гібридна фільтрація

Гібридна система рекомендацій – це тип системи рекомендацій, який поєднує колаборативну фільтрацію та на основі вмісту.

Поєднання колаборативної фільтрації та фільтрації на основі вмісту може допомогти у подоланні недоліку, з якими ми стикаємося при їх окремому використанні, а також може бути більш ефективним у деяких випадках. Підходи до гібридної системи рекомендацій можна реалізувати різними способами, наприклад, використовуючи створення прогнозів окремо, а потім об’єднуючи прогнози.[6]

1.2.1.4 Демографічна фільтрація

Демографічна фільтрація зазвичай використовує дані атрибути для опису користувача:

1. Стать
2. Вік
3. Країна проживання
4. Доходи домашніх господарств
5. Рівень освіти
6. Статус і вид зайнятості
7. Родинні стосунки та батьківський статус
8. Тип громади
9. Тощо[7]

Такий тип фільтрації є дуже специфічним, оскільки вузьконаправлено визначає користувача, або контент. Через що в рекомендаційних системах внутрішнього контенту - використовують рідко, але він є популярним для

таргетування реклами, бо багато товарів є якраз специфічно розроблені для певного кластера населення.

1.2.2 Проблематика рекомендаційних систем

Основною проблемою рекомендаційних систем така, що ми не можемо точно визначити наперед уподобання людини і як вона вестиме себе в майбутньому. Ми здатні тільки прогнозувати, але це не гарантує сто відсоткове співпадіння з реальністю.

Іншою проблемою може стати - недостатність, або розрідженість даних про користувача, контент, тощо, що ускладнює побудову моделі об’єкту.

Також є проблема білих ворон, хоч вона і зустрічається рідко. Даним терміном описують користувачів чиї вподобання мають властивість швидко та кардинально змінюватися, тому знаходження потрібних рекомендацій стає майже неможливою.

1.2.3 Приклад використання алгоритмів рекомендаційних систем

Прикладом одного з останніх й самого успішного використання рекомендаційної системи - є соцмережа TikTok. Це соцмережа орієнтована на поширення швидких та коротких відео.

Її феномен складався в тому, що в неї був найбільший показник DAU (Daily Active Users) та підтримка високого рівня залучення користувачів (тобто оптимізація для утримання та часу, проведеного на платформі) є однією з найбільших.

Даний успіх був забезпечений за допомогою декількох факторів:

1. Система рекомендації.

   Вона складається з двох основних підходів:

   - за тегом вмісту. Тобто якщо користувач подивився і вподобав контент з певним вмістом 4 рази, то відповідно йому система пропонуватиме 5-ий раз подивитися контент, який містить той самий вміст;

   - за поведінкою схожості. Наприклад якщо одночасно користувач з іншими вподобали 4 поспіль схожих контенти з однаковим вмістом, і потім інші вподобали якийсь інший вміст, то і першому користувачу буде запропоновано контент з цим іншим вмістом.[8]

2. Аналіз нового контенту.

   Контент аналізується не тільки за описом та хештегами які вказали, автори, а й за об’єктами які є в кадрі, які були звуки під час відео, написи, тощо.[8]

3. Аналіз зворотнього зв'язку.

   Зв’язок поділяється на два типи:

   - явний: вподобайки, коментарі, підписки;

   - не явний: час відтворення, відсоток перегляду та кількість повторного перегляду.[8]

Отже в основі рекомендаційної системи лежить гібридна фільтрація, але особливість полягає ще в тому, що для цього було виділено багато ресурсів, щоб система могла обробляти одночасно багато даних, та в онлайні аналізувати поведінку набагато більшої вибірки користувачів, ніж в інших соцмережах, що потягло за собою більші витрати, але й при цьому прискорило зростання самої соцмережі.

1.3 Сучасні технології для побудови backend частини веб-додатків

З розвитком технологій для різних типів задач почали з'являтися різноманітні рішення, кожне з яких має свою специфіку та сферу застосування. Через що наразі є велика кількість різноманітних технічних рішень кожної проблеми.

Мікросервісна архітектура надає можливість застосовувати різноманітні технології, оскільки кожен сервіс займається окремою задачею, що дозволяє використати рішення, яке найкраще підійде для конкретного модулю.

1.3.1 Фреймворки для створення backend частини веб-додатку

Фреймворк - це сукупність готових інструментів, котрі полегшують процес розробки. Вони містять набір модулів та бібліотек, котрі вже мають готові рішення для певних задач та допомагають зменшити розмір коду та пришвидшити його написання.

Також фреймворки визначають підхід до розробки та стандарти практик, що дозволяє розробникам більш структурувати проєкт, що робить код більш прийнятним для інших та зменшує ризик виникнення помилки.

Фреймворки часто оновлюються, що несе за собою покращення та більшу стабільність тим самим покращується сам додаток.

Найпопулярніші фреймворки та сфери застосування:

1. ASP .NET Core

   Це міжплатформний фреймворк розроблений Microsoft. Початково був створений для хмарних обчислень.

   Переваги:

   - Висока продуктивність і швидкість роботи;
   - Висока масштабованість: ASP.NET дозволяє легко масштабувати

додатки відповідно до зростаючої навантаженості;

- Відкритий код: ASP.NET є відкритим джерелом, що дозволяє розробникам розширювати та змінювати функціональність фреймворку під свої потреби.

Мінуси:

- Має великий темп розвитку порівняно з іншими, через що потребує постійного удосконалення знань.

2. Django

Це веб-фреймворк Python з відкритим вихідним кодом, який дотримується архітектурного шаблону MVC, який розділяє логіку програми, представлення та дані на окремі компоненти.[9]

Плюси:

- Має вбудований адміністративний інтерфейс, що дозволяє легко керувати базою даних та адмініструвати веб-додаток;

- Забезпечує високий рівень безпеки завдяки вбудованим заходам захисту від атак XSS та SQL injection.

Мінуси:

- Зазвичай не підходить для складних проєктів;

- Має певні обмеження щодо конфігурування серверів та налаштування хостингу;

- Може бути недостатньо гнучким для великих проектів, які вимагають багато налаштувань та власних рішень.

3. ExpressJS

Це швидка, легка та гнучка структура веб-додатків на основі Node.js. Він надає відмінний набір функцій для веб додатків.[9]

Плюси:

- Легкий та простий у використанні;

- Підтримує роботу з відкритими стандартами та протоколами, такими як HTTP, WebSocket, SSL, тощо.

Мінуси:

- Не має вбудованої архітектури та структури, що може призвести до неоднакового стилю програмування та складнощів при співпраці більшої кількості розробників;

- Не має такої великої кількості готових компонентів та модулів, які є у конкурентів.

4. Ruby on Rails

Це фреймворк, побудований на мові програмування Ruby. Завдяки своїм потужним інструментам ця платформа дозволяє розробникам створювати сучасні та масштабовані веб-програми.

Він надає перевагу простоті використання та зручності, надаючи стандартні рішення для щоденних проблем. Таким чином розробники можуть зосередитися на створенні функцій, адаптованих до унікальних потреб їхніх програм.[9]

Плюси:

- Є дуже гнучким та забезпечує можливість створення додатків різного типу та розміру:

- Має вбудовану структуру додатку та кілька готових компонентів, що дозволяє розробляти додатки швидше та ефективніше.

Мінуси:

- Споживає багато ресурсів;

- Має складну архітектуру.

5. CakePHP

Це популярний фреймворк з відкритим кодом, яка широко використовується для створення надійних і масштабованих веб-додатків.

Він написаний на PHP і відповідає архітектурному шаблону MVC.[9]

Плюси:

- Відкритий код;
- Має вбудовані інструменти, які дозволяють швидко створити веб-додатки. Також має готові шаблони для створення інтерфейсу користувача.

Мінуси:

- Повільніший ніж інші фреймворки;
- Не детальна документація.

1.3.2 СКБД

Наразі жоден веб-додаток не може обійтися без бази даних, якщо там присутній зв’язок між користувачами, тому СКБД відіграють важливу роль у створенні додатку. Вони визначають як буде відбуватися зберігання даних та взаємодію з ними. Від них залежить побудова проєкту, його швидкість та багато інших факторів.

1.3.2.1 Реляційні СКБД

Реляційні СКБД - це система керування реляційними базами даних, тобто такими в котрих формально описані таблиці, та власне нормалізований зв'язок між ними.

Реляційні СКБД бувають:

1. Серверні.

   Вони працюють на централізованому сервері, тому різні додатки мають до них доступ. Прикладами є:

   - PostgreSQL

- MySQL
- Oracle Database
- Microsoft SQL server

2. Вбудовані.
   Дані СКБД вбудовуються безпосередньо в додаток, тому мають обмежену масштабованість. Прикладами є:
   - SQLite
   - H2
   - Firebird

3. Хмарні.
   Ці СКБД зберігаються в хмарному середовищі. Прикладами є:
   - Amazon RDS
   - Microsoft Azure SQL Database
   - Google Cloud SQL

1.3.2.2 Нераляційні СКБД

В нереляційних СКБД дані зберігаються у вигляді неструктурованих, або напівструктурованих даних, таких як документи, ключ-значення, графи тощо.

Нереляційні СКБД дозволяють більш гнучко працювати з даними ніж реляційні СКБД, особливо з даними великого об'єму або з даними, які мають змінну структуру. Вони також дозволяють швидше зберігати та отримувати дані завдяки своїй децентралізованій архітектурі та можливості розподілу навантаження на багато вузлів.

Основними прикладами є:

1. MongoDB

2. Cassandra
3. Redis
4. Couchbase
5. Amazon DynamoDB

1.3.3 Брокери повідомлень

Брокер повідомлень - це архітектурний шаблон для перевірки, перетворення та маршрутизації повідомлень. Він забезпечує зв’язок між програмами, мінімізуючи взаємну обізнаність, яку програми повинні мати одна про одну, щоб мати можливість обмінюватися повідомленнями.[10]

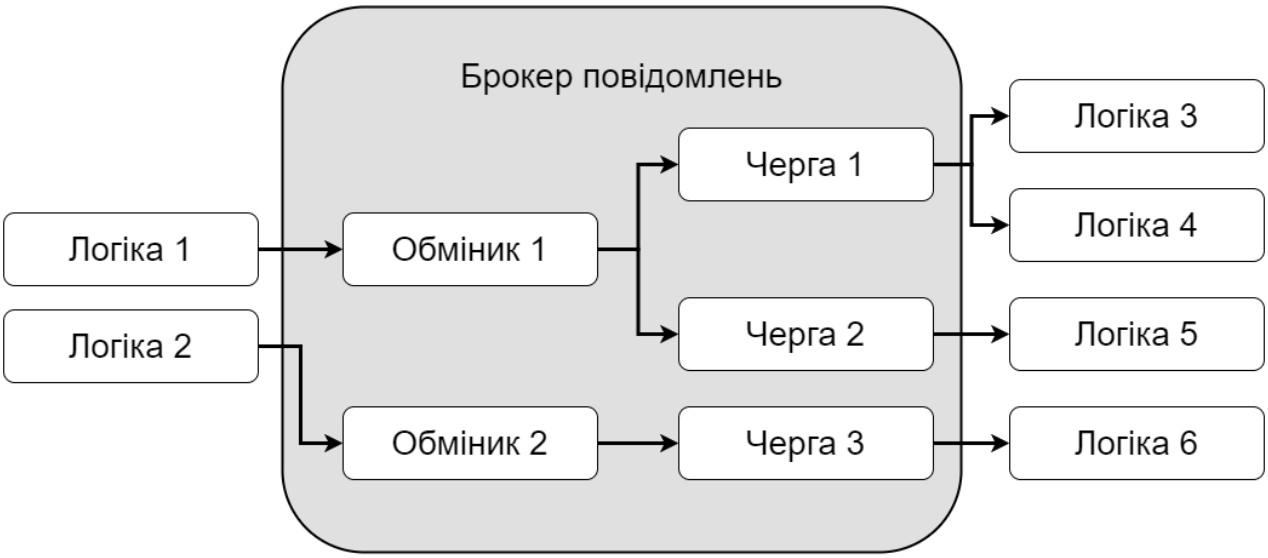

Рисунок 1.3 – Приклад моделі брокера повідомлень

Зазвичай брокери використовують в ситуаціях коли потрібен однобічний зв’язок, бо початковому модулю може вже буде байдуже, що далі буде відбуватися. Таким чином замість HTTP запитів між модулями - ми можемо використовувати брокер для мінімізації відповіді користувачу.

Також інколи брокери повідомлень використовують для логування.

Приклади брокерів повідомлень:

1. RabbitMQ
2. Apache Kafka
3. Apache ActiveMQ
4. Apache pulsar

2. Реалізація рекомендаційної системи за допомогою мікросервісної архітектури

Даний розділ присвячений практичному застосуванню отриманих знань з дослідження та реалізації рекомендаційної системи використовуючи міркросервісну архітектуру.

2.1 Опис проєкту

В сучасних реаліях розміри капіталістичних гігантів починають зменшуватися через спрощення процесу розробки, але за рахунок цього починають з’являтися все більше стартапів, які пропонують конкурентоспроможні рішення.

Однак дуже поширена проблема при створенні стартапу - пошук команди. У людини може бути чудова ідея, але вона буде спеціаліст тільки в конкретній сфері, але часто з'являється потреба ще й в інших спеціалістах для повної реалізації.

Наприклад ви інженер з електроніки і придумали як реалізувати гибкі сонячні батареї і хотіли б застосувати цю технологію на жалюзях і запустити даний стартап. Але у вас немає навичок з моделювання, виробництва з пластику, маркетингу та менеджменту. У вас можуть бути знайомі, котрі спеціалізуються в деяких з цих сфер, але не факт, що ви закриєте всі потреби.

Це проблема є дуже поширеною і даний проєкт, котрий описаний далі - орієнтований на її вирішення.

Отже уявімо платформу на якій є користувачі, котрі вказують свою спеціалізацію, свої уподобання, які сфери цікавлять, тощо. Є можливість створювати стартапи, в котрих описується яких спеціалістів вони потребують, додавати до яких сфер належить стартап, опис, тощо. Користувачі можуть

писати на своїх сторінках дописи з їх думками, або дослідженнями, а також створювати пости на сторінках стартапів, наприклад ділячись своїми успіхами. Користувачі можуть підписуватися один на одного, на стартапи, подавати заявки на долучання до команди стартапу, якщо там триває набір.

Але вся ця соцмережа буде не ефективна без рекомендаційної системи. Проблема полягає в тому, що в разі великої кількості користувачів та стартапів - буде дуже важко знайти проєкти, які можуть бути особисто цікавими, або знайти кандидатів до членів команди, яким буде цікавий напрям стартапу, або просто інших користувачів, для спілкування, або допомоги. А також рекомендаційна система відіграватиме ключову роль, щоб утримати користувача як можна довше в системі та визвати цікавість до системи в стрічці рекомендацій.

Отже задача в цьому розділі - побудувати рекомендаційну систему для даної соцмережі використовуючи мікросервісну архітектуру.

2.2 Вибрані технології

Основою даного проєкту буде ASP .Net фреймворк.

Даний фреймворк є самим найшвидшим в розвитку та має велику кількість бібліотек для спрощення написання коду. А також кожного року виходять нові версії, котрі оптимізовують та пришвидшують працю всієї системи. І при використанні ASP .Net - проєкт буде кросплатформенним, що допоможе швидко масштабувати та переносити додаток з платформи на платформу в разі потреби.

В подальшій реалізації буде використовуватися .Net 7.0, оскільки це остання стабільна версія, на момент написання, а також має чимало оновлень, враховуючи LINQ, де була пришвидшена його робота.

LINQ - внутрішня мова запитів для взаємодії з даними, а також використовується для праці з Entity Framework, котрий також буде використовується, тому будуть розглянуті два типу праці з базою даних:

- через Entity Framework та внутрішні інструменти;
- через прямі SQL запити.

Для реалізації зберігання даних та взаємодією мною буде використаний PostgreSQL. Одним з головних факторів його вибору є те, що він має відкритий код та є безкоштовним для використання, порівняно з іншими аналогами реляційних СКБД.

Також PostgreSQL частіше використовують для великих проєктів в котрих є багато зв’язків та інформації, через його швидкість, що може бути корисним в разі розширення соцмережі.

Одним з інструментів для побудови мікросервісної архітектури - буде брокер повідомлень RabbitMQ. Далі буде показано та написано яким чином він буде застосований.

2.3 Модель даних

Модель даних побудована так, що в основі лежать три таблиці:

- User - сам користувач та інформація про нього;
- Startup - стартап та інформація про нього;
- Post - пост з заголовком, текстом і датой, котрий може належати користувачеві, або стартапу.

Також додаткові таблиці для функціонування соцмережі, але для наших цілей - вони не важливі.

Основні таблиці з котрими ми будемо взаємодіяти:

- Tag - теги(хештеги, або просто слова, які можуть охарактеризувати об'єкт) для метрик;
- User_stat_Tag - матриця уподобань відносно користувача відносно тегів;
- User_standard_Tag - стандартні теги, котрі користувач вибрав сам;
- Post_stat_Tag - матриця статистики поста;
- Post_standard_Tag - теги, котрі вказано в пості;
- Startup_stat_Tag - матриця статистики стартапу;
- Startup_standatd_Tag - теги, котрі вказано в стартапі.

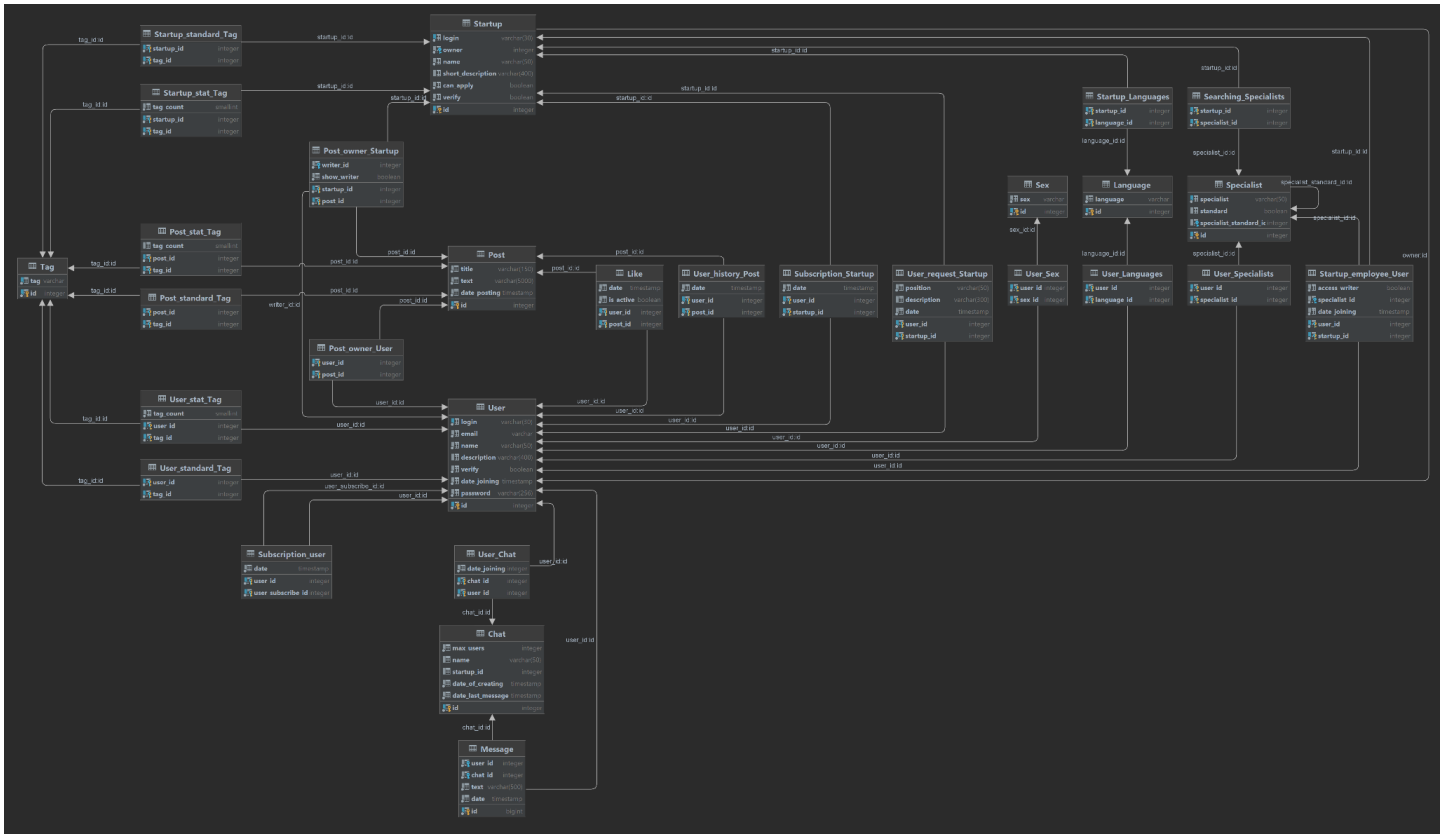

Рисунок 2.1 – Модель бази даних для проєкту

Таким чином за допомогою тегів ми будемо розмічати дані для опису об’єктів, а як саме це буде використано - буде описано в наступному розділі.

2.4 Алгоритм рекомендацій

Вся система рекомендацій даного проєту побудована використовуючи фільтрацію на основі вмісту. Тобто нам байдуже наскільки наші смаки співпадають зі смаками іншими, бо будемо орієнтуватися тільки на те, наскільки наші уподобання співпадають з матрицею інтересу об’єкта.

Отже ми маємо таблиці стандартних тегів для користувача, поста та стартапу, які не мають лічильника і є сталими(не змінюються системою рекомендацій, а тільки самими користувачами), це зроблено для того, щоб постійно зберігати дані про стандартні інтереси.

Також ми маємо додаткові таблиці для кожної сутності, де ми вже будемо вести відношення кожного об’єкту до тегів, але вже маючи лічильник, щоб можна було мати градацію в інтересах для більш кращої та точної рекомендації.

Серед додаткових параметрів у нас є:

- MinCountValue - мінімальне значення лічильника
- MaxCountValue - максимальне значення лічильника
- PercentOfStandart - відсоток важливості стандартних тегів для кінцевого значення лічильника тегу для посту або користувача
- PercentOfStandartStartup - відсоток важливості стандартних тегів для кінцевого значення лічильника тегу для стартапу

Значення лічильників інтересів будуть змінюватися в разі вподобання користувачем посту. Отже алгоритм такий:

1. Будується список всіх інтересів посту. Якщо PercentOfStandart = 0, то беруться теги тільки з статистики, якщо PercentOfStandart = 100 - тільки з стандартних, в іншому випадку - об'єднуються.

2. Подібний список будується для інтересів користувачів.
3. В разі присутності тега в списку поста та списку користувача, то лічильник тегу в статистиці збільшується на 1. Якщо цей тег не присутній у користувача - тег додається до статистики зі значенням MinCountValue. Якщо тег котрий присутній у списку користувача, але відсутній у поста, то значення зменшується на 1. В разі, якщо хоч один тег в новій статистиці користувача перевищує MaxCountValue - всі значення лічильників тегів зменшується на 1. Якщо у тега значення менше MinCountValue, то він видаляється з таблиці.
4. Обернено проводимо процедуру для матриці інтересів поста.

```
2 references
public Dictionary<int, int> CountStats(Dictionary<int, int> userTags, Dictionary<int, int> postTags, List<int> postStandartTags)
{
    var totalPost = new Dictionary<int, int>();

    foreach (var tag in postTags)
        totalPost.Add(tag.Key, (int)Math.Round((float)tag.Value / 100 * (100 - percent)));

    foreach (var tag in postStandartTags)
        if (totalPost.ContainsKey(tag))
            totalPost[tag] += ((int)Math.Round((float)maxValue / 100 * percent));
        else
            totalPost.Add(tag, (int)Math.Round((float)maxValue / 100 * percent));

    var answer = new Dictionary<int, int>();

    foreach (var tag in userTags)
        answer.Add(tag.Key, tag.Value - 1);

    foreach (var tag in totalPost)
        if (answer.ContainsKey(tag.Key))
            answer[tag.Key] += 2;
        else
            answer.Add(tag.Key, minValue + 1);

    foreach (var tag in answer)
        if (tag.Value > maxValue)
            foreach (var tag2 in answer)
                answer[tag2.Key]--;

    foreach (var tag in answer)
        if (tag.Value < minValue)
            answer.Remove(tag.Key);

    return answer;
}
```

Рисунок 2.2 – Процедура для підрахунку нових значень тегів користувача

У даного підходу плюс в тому, що він займає мало потужностей для оновлення інтересів, ми можемо регулювати параметри градації, та важливості тегів користувача, та вже присвоєних нами. Також при такому алгоритмі легко

можуть з'являтися нові інтереси у користувача та зникати старі, тим самим матриця не буде постійно зростати займаючи зайву пам’ять.

Статистика стартапу вираховується за допомогою середнього значення всіх його постів.

Отже тепер розглянемо, як працюють рекомендації спираючись на наші метрики.

Спочатку розглянемо алгоритм рекомендацій постів серед тих на кого користувач підписаний(інші користувачі та стартапи):

1. Для користувача створюємо нову статистику, де значення лічильникам стандартних тегів присвоюємо MaxCountValue та множимо на PercentOfStandart, а для наявної статистики значення лічильника тегів множимо на (100% - PercentOfStandart), після чого об’єднуємо ці статистики і в разі співпадіння тегів - значення додаються. Після чого ми отримуємо “справжню” матрицю інтересів користувача.
2. Знаходимо всі пости, які належать до користувачів та стартапів на які підписаний користувач, та ще не бачив.
3. Для кожного посту створюємо аналогічну “справжню” матрицю інтересів, як ми зробили для користувача. Після чого створюємо матрицю X, де перше значення рядку - це значення лічильника в матриці користувача, якщо тег відсутній, то присвоюється 0, а друге значення рядку - це значення лічильника такого ж тегу в матриці поста, а якщо відсутній - береться 0.

   Далі для нової отриманої матриці X, обраховуємо S(score), де n - кількість рядків в матриці, l - кількість вподобайок постам, які користувач поставив постам користувача, або стартапу, кому належить цей пост, а h - кількість

показів постів.

$$S = \left(1 - \frac{l}{h}\right) * \sum_{i=1}^{n} |Xi1 - Xi2|$$

4. Сортуємо від найменшого до найбільшого значення S і таким чином отримуємо список від найцікавіших для користувача постів - до найнецікавих.

```
2 references
public List<int> GetLimitRecomendation(int user_id, int limit)
{
    var result = new Dictionary<int, int>();

    var posts_id = _postRepository.GetUserPostsIDsSubscrription(user_id);

    var user_stat = GetUserStat(user_id);

    foreach (var post in posts_id)
    {
        var percentOfInteresting = _postRepository.GetPercentInterestingInOwning(post, user_id);
        var score = 0;
        if (percentOfInteresting > 0)
        {
            var post_stat = GetPostStat(post);
            score = GetScore(user_stat, post_stat, percentOfInteresting);
        }
        result.Add(post, score);
    }

    var sortedResult = result.OrderBy(post => post.Value).Take(limit).Select(post => post.Key).ToList();
    if (sortedResult == null)
        sortedResult = new List<int>();

    return sortedResult;
}
```

Рисунок 2.3 – Процедура для отримання вибірки найцікавіших постів

Дана формула враховує не тільки співпадіння інтересів з кожним постом, а ще й цікавість користувача, до того, кому належить цей пост, що часто є важливим фактором хотіння його побачити.

Інші не менш важливі рекомендації - це цікаві стартапи для користувача, де йому було б цікаво і він зміг би взяти участь. В базі даних (рис. 2.1) є

спеціальна сутність Specialist з атрибутом specialist_standard_id, котрий посилається на сутність Specialist, щоб ідентифікувати схожі спеціальності, наприклад “3D-designer” “3D дизайнер”. Також є сутність Language, щоб підібрати стартапи, де користувач зможе взаємодіяти з командою.

Отже для визначення найцікавіших стартапів для приймання участі робимо наступні кроки:

1. Для користувача створюємо нову статистику, де значення лічильникам стандартних тегів присвоюємо MaxCountValue та множимо на PercentOfStandart, а для наявної статистики значення лічильника тегів множимо на (100% - PercentOfStandart), після чого об’єднуємо ці статистики і в разі співпадіння тегів - значення додаються. Після чого ми отримуємо “справжню” матрицю інтересів користувача.
2. Для кожного стартапу створюємо нову статистику подібним чином, але тепер замість PercentOfStandart використовуємо PercentOfStandartStartup. Потім створюємо аналогічну матрицю X, як в минулому алгоритмі, але тепер дивимось тільки на ті теги, які є у користувача, а якщо у стартапа залишаються ще якісь - не звертаємо увагу.
   Далі для нової отриманої матриці X, обраховуємо S(score), де *n* - кількість рядків в матриці

$$S = \sum_{i=1}^{n} |Xi1 - Xi2|$$

3. Тепер сортуємо від найменшого до найбільшого значення S, отримуючи список від найбільш рекомендованого до найменш для користувача.

```
WITH user_stat_tags AS(
                        WITH user_tags AS (
                        SELECT tag_id, ROUND(tag_count * @notStandartValue) AS tag_count
                        FROM "User_stat_Tag"
                        WHERE user_id = @user_id
                    ),
                    user_standard_tags AS (
                            SELECT tag_id, @standartValue AS tag_count
                            FROM "User_standard_Tag"
                            WHERE user_id = @user_id
                        )
                        SELECT
                           COALESCE(user_tags.tag_id, user_standard_tags.tag_id) AS tag_id,
                           COALESCE(user_tags.tag_count, 0) + COALESCE(user_standard_tags.tag_count, 0) AS tag_count
                        FROM user_tags
                        FULL OUTER JOIN user_standard_tags ON user_tags.tag_id = user_standard_tags.tag_id
                    ),
                    startup_tags AS (
                        SELECT
                            s.id AS startup_id,
                            t.id AS tag_id,
                            COALESCE(st.tag_count, 0) AS tag_count
                        FROM "Startup" s
                        CROSS JOIN "Tag" t
                        LEFT JOIN "Startup_stat_Tag" st ON st.startup_id = s.id AND st.tag_id = t.id
                    ),
                    startup_standard_specialists AS (
                        SELECT startup_id, specialist_id
                        FROM "Searching_Specialists"
                    ),
                    startup_standard1_specialists AS (
                        SELECT startup_id AS startup_id, sp.specialist_standard_id AS specialist_id
                        FROM startup_standard_specialists ss
                        JOIN "Specialist" AS sp ON sp.id = ss.specialist_id
                        WHERE sp.specialist_standard_id IS NOT NULL
                    ),
                    startup_standard2_specialists AS (
                        SELECT startup_id AS startup_id, sp.id AS specialist_id
                        FROM startup_standard1_specialists ss
                        JOIN "Specialist" AS sp ON sp.specialist_standard_id = ss.specialist_id
                        WHERE sp.specialist_standard_id IS NOT NULL
                    ),
```

Рисунок 2.4 – SQL запит для отримання рекомендованих стартапів(продовження 2.5)

В даному випадку ми маємо PercentOfStandartStartup, щоб відокремити відсоток впливу в стартапах та постах, або користувачах. Це робиться для того, бо наприклад в підборі стартапів набагато важливіше значення стандартних тегів, а не статистика постів, а в підборі постів - навпаки значення статистики, а не стандартні.

Також по такій логіці була реалізована рекомендація користувачів які підійдуть для стартапу, якщо власник хоче знайти сам.

```
    startup_specialits AS (
        SELECT startup_id, specialist_id
        FROM startup_standard_specialists
        UNION
        (SELECT startup_id, specialist_id
        FROM startup_standard1_specialists
        UNION
        SELECT startup_id, specialist_id
        FROM startup_standard2_specialists)
    ),
    user_standard_specialists AS (
        SELECT user_id, specialist_id
        FROM "User_Specialists"
        WHERE user_id = @user_id
    ),
    user_standard1_specialists AS (
        SELECT user_id AS user_id, sp.specialist_standard_id AS specialist_id
        FROM user_standard_specialists ss
        JOIN "Specialist" AS sp ON sp.id = ss.specialist_id
        WHERE sp.specialist_standard_id IS NOT NULL
    ),
    user_standard2_specialists AS (
        SELECT user_id AS user_id, sp.id AS specialist_id
        FROM user_standard1_specialists ss
        JOIN "Specialist" AS sp ON sp.specialist_standard_id = ss.specialist_id
        WHERE sp.specialist_standard_id IS NOT NULL
    ),
    user_specialists AS (
        SELECT user_id, specialist_id
        FROM user_standard_specialists
        UNION
        (SELECT user_id, specialist_id
        FROM user_standard1_specialists
        UNION
        SELECT user_id, specialist_id
        FROM user_standard2_specialists)
    ),
    startup_language AS (
        SELECT startup_id
        FROM "Startup_Languages" ul
        JOIN "User_Languages" sl ON sl.language_id = ul.language_id
        WHERE user_id = @user_id
    )
SELECT
  st.startup_id
FROM startup_tags st
LEFT JOIN user_stat_tags ut ON ut.tag_id = st.tag_id
JOIN startup_language sl on sl.startup_id = st.startup_id
JOIN "Startup" sta ON st.startup_id = sta.id WHERE sta.can_apply = true AND st.startup_id IN (SELECT DISTINCT ss.startup_id
                                                                                             FROM user_specialists us
                                                                                             JOIN startup_specialits AS ss ON us.specialist_id = ss.specialist_id
    )
GROUP BY st.startup_id
ORDER BY SUM(ABS(st.tag_count - ut.tag_count))
LIMIT @limit
```

Рисунок 2.5 – Продовження SQL запиту для отримання рекомендованих стартапів

2.5 Архітектура

Архітектура в застосуванні рекомендаційних системах для додатків є дуже важливою. Оскільки це багатоскладова структура і часто стає ресурсоємною, то до цього питання потрібно ставитися відповідально під час побудови проєкту, щоб потім було менше проблем.

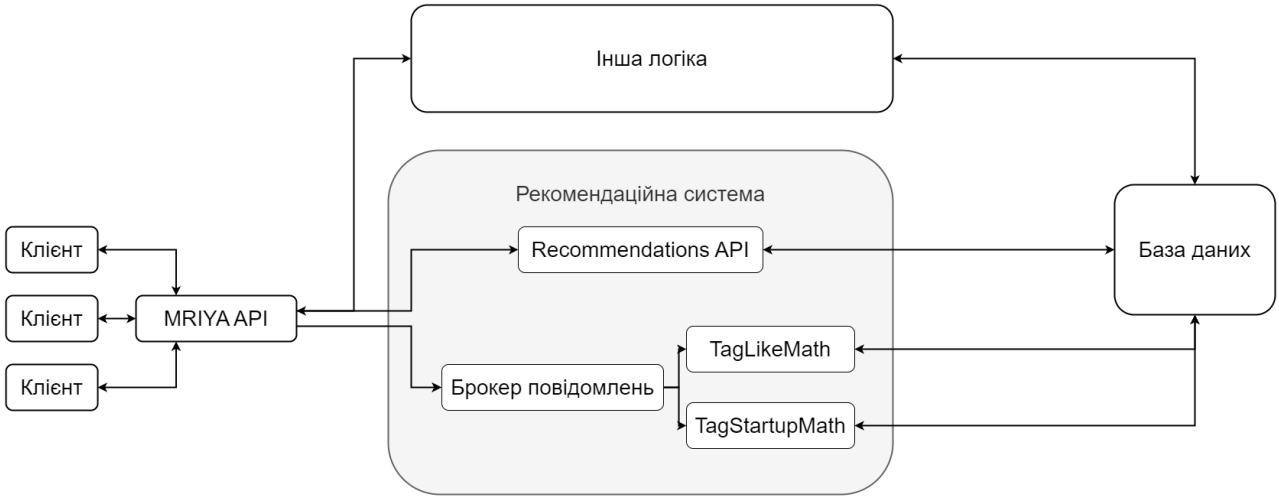

Рисунок 2.6 – Модель побудови архітектури рекомендаційної системи

Під час побудови рекомендаційної системи я поділив сервіси на дві групи:

1. Ті з якими користувач взаємодіє прямо зараз
2. Ті, на відповідь котрих користувачу байдуже і це не впливає на процес користування платформою під час взаємодії.

Через що було створений сервіс Recommendations API в котрий обробляє запити з основного API з яким взаємодіє користувач, та в відповідь надає рекомендовані пости, стартапи та користувачів для стартапів.

А оновлення статистики після вподобайки, або оновлення статистики стартапу - не є важливим для користувача, оскільки він не отримає від цього ніякої відповіді, тому це однобічні сервіси і їх процес реалізований через брокер повідомлень.

Тому в разі зростання користувачів і потреби в обрахунках рекомендацій - можна окремо масштабувати Recommendations API на більші потужності і не затримувати обробку запитів користувачів, що покращить досвід користування платформою.

Так само з TagLikeMath та TagStartupMath, але їх розгортання ще простіше, оскільки MRIYA API не потрібно знати інформацію про них, то відповідно в інших сервісах не потрібно інформувати про збільшення даних сервісів.

Висновок

Під час написання даної курсової роботи було проведено дослідження про використання різноманітних методів для рекомендаційних систем, сучасних типів архітектур систем та технологій, якими вони можуть бути реалізовані. Були розглянуті плюси та мінуси застосування даних підходів, що дало змогу оцінити та зрозуміти, що краще використовувати під час реалізації проєкту.

Кінцевим результатом курсової роботи стала реалізація власної рекомендаційної системи зі застосуванням мікросервісної архітектури для соціальної мережі. В ітозі був розроблений власний алгоритм оцінки для методу фільтрації на основі вмісту, розроблена архітектура для подальшого легкого масштабування та розгортання проєкту на серверах.

В подальшому я би хотів зробити алгоритм для аналізу текстового контенту, щоб автоматично додавати більше метрик постам, користувачам та стартапам, щоб стартова матриця інтересів не залежала тільки від тих тегів котрі вводить користувач, бо їх може не бути взагалі, що приповільнить просування. Також можна було б реалізувати систему, котра могла б визначати схожі теги та згруповувати їх.

Отже, завдяки цій роботі вдалося дослідити, на зараз, важливу тему рекомендаційних систем та застосувати дані знання на проєкті, який в подальшому може бути застосований в реальному веб-додатку.

Список використаної літератури

1. [Електронний ресурс] DIGITAL 2023 APRIL GLOBAL STATSHOT REPORT https://datareportal.com/reports/digital-2023-april-global-statshot
(Перевірка на доступ до ресурсу - 12.05.2023)

2. [Електронний ресурс] Як обрати архітектуру для веб додатку https://blog.ithillel.ua/articles/web-application-architecture
(Перевірка на доступ до ресурсу - 12.05.2023)

3. [Електронний ресурс] Рекомендаційна система https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D0%BE%D0%BC%D0%B5%D0%BD%D0%B4%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0
(Перевірка на доступ до ресурсу - 12.05.2023)

4. [Електронний ресурс] Колаборативна фільтрація https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0_%D1%84%D1%96%D0%BB%D1%8C%D1%82%D1%80%D0%B0%D1%86%D1%96%D1%8F
(Перевірка на доступ до ресурсу - 12.05.2023)

5. [Електронний ресурс] Content-based Filtering https://developers.google.com/machine-learning/recommendation/content-based/basics
(Перевірка на доступ до ресурсу - 12.05.2023)

6. [Електронний ресурс] A Guide to Building Hybrid Recommendation Systems for Beginners https://analyticsindiamag.com/a-guide-to-building-hybrid-recommendation-systems-for-beginners/
(Перевірка на доступ до ресурсу - 12.05.2023)

7. [Електронний ресурс] Demographic filtering
https://trymata.com/blog/learn/demographic-filtering/
(Перевірка на доступ до ресурсу - 12.05.2023)

8. [Електронний ресурс] Inside TikTok's "For You" Algorithm https://www.music-tomorrow.com/blog/a-complete-guide-tiktok-for-you-algorithm-for-musicians
(Перевірка на доступ до ресурсу - 12.05.2023)

9. [Електронний ресурс] The best 10 backend frameworks for your web application
https://blog.back4app.com/backend-frameworks/#The_best_10_backend_frameworks_for_your_web_application
(Перевірка на доступ до ресурсу - 12.05.2023)

10. [Електронний ресурс] Message broker
https://en.wikipedia.org/wiki/Message_broker
(Перевірка на доступ до ресурсу - 12.05.2023)