Issue 14, 2023

https://doi.org/10.23939/sisn2023.14.142

УДК 004.9

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ІНТЕЛЕКТУАЛЬНОГО ПОШУКУ КОНТЕНТУ В СИСТЕМАХ Е-КОМЕРЦІЇ

Ілля Балуш[1], Вікторія Висоцька[1, 2], Марина Шевченко[3], Оксана Бродяк[4]

[1] Національний університет "Львівська політехніка", кафедра інформаційних систем та мереж,
вул. С. Бандери, 12, Львів, Україна
[2] Університет Оснабрюка, Інститут комп'ютерних наук,
вул. Фрідріха Янсена, 1, Оснабрюк, Німеччина
[3] Університет Оснабрюка, кафедра міжнародної економічної політики,
вул. Роланда, 8, Оснабрюк, Німеччина
[4] Національний університет "Львівська політехніка", кафедра вищої математики,
вул. С. Бандери, 12, Львів, Україна
E-mail: illia.balush.kn.2017@lpnu.ua, ORCID: 0000-0002-0498-4719,
E-mail: Victoria.A.Vysotska@lpnu.ua, ORCID: 0000-0001-6417-3689,
E-mail: mshevchenko@uni-osnabrueck.de, ORCID: 0000-0003-2165-9907
E-mail: oksana.y.brodiak@lpnu.ua, ORCID: 0000-0002-9886-3589

© Балуш І. І., Висоцька В. А., Шевченко М. М., Бродяк О. Я., 2023

Описано розроблення технології інтелектуального пошуку контенту із реалізацією модуля систем е-комерції для формування списку рекомендацій постійному користувачу. Інтелектуальний пошук контенту ґрунтується на методах лінгвістичного аналізу, сучасних алгоритмах розбору і знаходження слів, рекомендаціях на основі вподобань користувачів. Основними складовими такого пошуку є парсинг текстових ланцюжків, виокремлення ключових слів, перевірка правопису, розпізнавання загальних скорочень та акронімів, семантичний аналіз тексту, пошук за релевантністю з використанням синонімів, фільтрів та сортування. Розроблено вебдодаток на базі Java і Elasticsearch з імплементацією рекомендаційної системи на основі алгоритму колаборативної фільтрації. Мета – розроблення технології інтелектуального пошуку товарів із формуванням списку рекомендацій для користувача. Об'єктом дослідження є процеси інтелектуального пошуку з можливістю генерування рекомендацій для користувачів у сфері будь-якої е-комерції без прив'язки до категоризації товару/послуг тощо. Предметом дослідження є методи та засоби інтелектуального пошуку рекомендаційних систем на основі алгоритму Collaborative Filtering для формування рекомендацій щодо товарів для користувачів, що орієнтується на загальні збіги вибору подібних користувачів. Під час експериментальної апробації розробленої системи здійснено низку пошукових запитів із NLP-алгоритмом і без, результати яких продемонстрували покращення роботи системи в межах 15–95 % залежно від ключового слова та наявності/відсутності помилок у словах пошуку. Також виконано порівняння швидкості виконання запитів з уже наявними системами. Кількість даних у сховищі може відрізнятися (похибка під час порівняння 60–70 мс). Наприклад, запит, який складається з одного або двох слів, опрацьовується значно швидше – на 20–70 мс порівняно з аналогами в таких межах. Але якщо слів три і більше, результати приблизно подібні – на 9–20 мс швидше.

Ключові слова: пошукова система; Elasticsearch; рекомендаційні системи; колаборативна фільтрація; вебдодаток; пошук; нечіткий пошук.

Вступ. Постановка проблеми

Сьогодні в інтернеті зберігається неймовірно велика кількість інформації. Крім того, з кожним днем її кількість тільки зростає. Це стосується як інтернет-ресурсів, що містять розважальний контент (Netflix, Megogo), так і інтернет-магазинів (Amazon, Rozetka тощо). Усі ці системи об'єднує

одне – щоб користувачі були задоволеними роботою цих ресурсів, вони повинні мати можливість швидко, просто і максимально ефективно шукати потрібну їм інформацію.

Крім того, одна з основ, яка допомагає утримати клієнтів на вебресурсі, – рекомендації. 60 % клієнтів переважно повертаються в магазини з рекомендаціями, а 75 % представників цифрового покоління, які виросли в епоху соціальних мереж, вважають, що рекомендаційні системи є невід'ємною частиною будь-якого магазину або розважальної платформи. Тому сьогодні швидкий і ефективний пошук інформації з можливістю рекомендацій є однією з основних потреб людей, а також бізнесу. Вибрана тема надзвичайно актуальна саме тепер, адже сьогодні попит на товари, які пропонує бізнес, особливо онлайн, зростає з небувалою швидкістю. Це стосується відео- та аудіопродукції, товарів, книг тощо. Особливо це актуально для українських покупців і користувачів в умовах війни. Адже, за статистикою, до війни приблизно 25 % українців регулярно купували товари через інтернет, 33 % українців хоча б раз на рік купували певний товар онлайн. Нині ці цифри істотно зросли не лише через знищення деяких фізичних магазинів на певних територіях, окупацію інших територій та масове вимушене переселення населення, що призвело до порушення логістики, поповнення складів новими товарами та взагалі ведення бізнесу. З іншого боку, періодичні та нещодавно систематичні блекаути, багатогодинні повітряні тривоги порушили ритм життя пересічних українців – і тих, хто надає торговельні послуги, і їхніх клієнтів. Тому ситуація змусила е-бізнес швидко адаптуватися та розвиватися у цих важких умовах, що допомагає підтримувати економічну стабільність України та української гривні.

Аналіз останніх досліджень та публікацій

Термін "електронний бізнес" ввела IT-корпорація IBM наприкінці 1990-х років і, як правило, він стосувався комп'ютеризованих процесів для автоматизації тогочасної торгівлі. Сьогодні обсяг терміна "електронний бізнес" залежить від того, наскільки поширилось поняття "бізнес". Згідно з відомою електронною енциклопедією з маркетингу Ryte Wiki, термін е-бізнес часто використовують як загальний термін для всіх електронно керованих бізнес-процесів компанії (https://en.ryte.com/wiki/E-Business). Ці процеси здебільшого автоматизовані, для цього використовують інтернет та інші цифрові інформаційні технології. Як навчальний предмет електронний бізнес охоплює розроблення та виробництво продукту, а також фінансування та маркетинг. Електронна комерція та електронні закупівлі – дві складові електронного бізнесу. Електронне навчання в найширшому значенні також можна класифікувати як електронний бізнес. Gabler Wirtschaftslexikon (Німецька енциклопедія) містить визначення, основане на п'яти стовпах та сферах е-бізнесу: "інформація, комунікація та транзакції передаються або опрацьовуються через цифрові мережі між економічними партнерами-учасниками" зокрема через п'ять сфер, таких як е-закупівлі, е-комерція (е-магазин, е-банк, дистанційна освіта тощо), е-ринок, е-спільнота та е-компанія. Основні цілі е-бізнесу – значною мірою реалізація автоматизованих процесів у торгівлі. Згідно із визначенням IBM (https://www.ibm.com/topics/intelligent-search), інтелектуальний пошук (англ. ***Intelligent search***) ґрунтується на технології штучного інтелекту та усуває накопичення даних, а також допомагає співробітникам/клієнтам швидко, легко й оперативно знаходити потрібну актуальну інформацію. Постійні та потенційні користувачі можуть використовувати інтелектуальний пошук для отримання інформації з будь-якого місця (всередині чи за межами компанії) і в наборах даних незалежно від формату: великі дані в базах даних, системи керування документами, цифровий контент, вебсторінки, на папері тощо. Інтелектуальний пошук і корпоративний пошук є синонімами пошуку природною мовою, пошуку штучного інтелекту або пошуку на основі штучного інтелекту та когнітивного пошуку. Тобто це різновид (підклас) інформаційного пошуку як техніки пошуку (***search engineering***). Почнемо з основних моментів для побудови пошукової системи. Будь-яка пошукова система (***search***

engine) складається зі сховища, де зберігаються усі необхідні дані, та програми, яка, використовуючи певні алгоритми, аналізує вхідний рядок, звертається до сховища і повертає релевантний результат. Складність полягає у тому, щоб правильно зберігати дані й коректно налаштувати алгоритми опрацювання пошукового рядка. Крім того, оскільки це також має бути і рекомендаційна система, необхідно правильно підібрати рекомендаційний алгоритм, котрий міститиме достатню кількість даних для коректних обчислень. Найбільше впливає на це серверна частина програми [1]. Постараємось побудувати стандартну на наш час модель пошукової системи. Для спрощення припустимо, що ми приймаємо лише текстові запити – набір слів. Бек-енд повертає результати, найвідповідніші вхідним словам. Серед вхідних слів немає логіки сполучення чи заперечення. Коли бек-енд отримує рядок пошукового запиту, він спочатку ділить його на певні частини або, по-іншому, – токени. Для кожного токена серверна частина робить запит до сховища. У сховищі зберігаються документи. Після виклику сховища бекенд вже має інформацію про те, в яких документах і скільки пошукових слів міститься. Потім їх об'єднують в остаточний список, упорядкований за релевантністю (найрелевантніші зверху), який повертається до користувача. Логічно, що для швидкого пошуку в сховищі необхідно правильно зберігати дані.

Припустимо, треба визначити, які статті новин The Washington Post містять слова "навколишнє середовище" та "здоров'я" (з самого початку). Один із підходів полягає у тому, щоб прочитати весь текст від початку, занотовуючи кожну статтю, що містить згадані слова. Зазвичай цей прийом розглядається як прямий огляд тексту, що займає значну частину часу.

Щоб уникнути такого лінійного сканування, одним з найпопулярніших способів для кожного запиту є індексація документів заздалегідь. За умови належного індексування на місці можна виконати таке завдання приблизно за кілька хвилин або секунд на сучасних машинах. Одним з індексів, який широко використовують для індексації великої колекції даних, є інвертовані індекси. Усі популярні пошукові системи, такі як Elasticsearch, Lucene, Solr, використовують інвертовані індекси, щоб забезпечити надзвичайно швидкий пошук у системі.

Інвертований індекс надає відповідність між термінами та документами, у яких трапляється цей термін. Тому нам не потрібно сканувати всю колекцію тексту, щоб отримати інформацію, що врешті-решт зменшує час пошуку. Багато інших функціональних можливостей, таких як частковий пошук, корекція орфографії, реалізують за допомогою цієї системи інвертованого індексу, щоб забезпечити набагато більше функціональних можливостей.

Порівняємо традиційну базу даних з інвертованим індексом [2]. Припустимо, що наша система містить чотири документи: Doc1 (Welcome to the Hotel California Such a lovely place), Doc2 (She's buying a stairway to Heaven), Doc3 (Hey Jude, don't make it bad), Doc4 (Take me to the heaven, welcome). У традиційній базі даних SQL дані виглядатимуть приблизно так, як у табл. 1.

Таблиця 1

Репрезентація зберігання даних у SQL

№	Контент
1	Welcome to the Hotel California Such a lovely place
2	She's buying a stairway to Heaven
3	Hey Jude, don't make it bad
4	Take me to the heaven, welcome

Отримати дані або інформацію на основі стовпця “Контент” важко та складно. Ефективність у традиційних базах даних SQL досягається за допомогою запитів за первинним ключем або побудови ефективних “індексів” для обходу цих таблиць баз даних. Завдяки використанню зворотних індексів отримання інформації з величезної кількості документів дуже легке та ефективне порівняно з традиційними базами даних. У зворотних індексах відображення відбувається від “Термінів” до “Документів” (табл. 2).

Таблиця 2

Зберігання даних у інвертованому індексі

Термін	Номер документа
uying	2
California	1
heaven	2, 4
Welcome	1, 4
Jude	3
stairway	2
bad	3

Табл. 2 показує, як працює простий перевернутий індекс. І це демонструє силу перевернутих індексів під час пошуку термінів. Наприклад, якщо ми шукаємо словосполучення “welcome heaven”, у нас немає точного відповідника в базі даних, але за допомогою перевернутого індексу ми можемо побачити, що користувач шукає Doc4 або Doc2 або Doc1 (Doc4 має найвищий рейтинг відтоді, як міститься у переліку документів з термінами “welcome” та “heaven”).

Двома основними компонентами перевернутого індексу є Словник та Списки проведення. Для кожного терміна в текстовій колекції існує список проведення, який містить інформацію про наявність терміна у цій колекції. Словник працює як структура даних пошуку поверх списків проведення. Враховуючи інвертований індекс та запит, найперше завдання – визначити, чи кожен термін запиту існує у словниковому запасі. Як і в прикладі, The Washinton Post, нам спочатку потрібно визначити, чи слово “середовище” насправді є в нашому словниковому запасі, тобто перевернутому індексі, і якщо так, то визначити відповідні проведення. Ця операція пошуку використовує класичну структуру даних, яка називається словником. Він має два широкі розділи рішень: хешування та дерева пошуку.

Фактичні дані індексу зберігаються у списку проведення. Доступ до нього здійснюється через словник пошукової системи. Кожен термін має власний список проводки. Оскільки фактичний розмір списку проведення занадто великий, його краще зберігати на диску, щоб зменшити вартість. Звичайно, реалізація дискових систем набагато складніша, ніж зберігання всього цього в оперативній пам’яті. Лише під час обробки запиту список проведення терміну запиту завантажується в пам’ять, як вимагають процедури обробки запитів. Немає фіксованого формату розміщення списків та індексу. Існує багато різних версій такого індексу, таких як частотний індекс, позиційний індекс, незалежний від схеми тощо. Популярним способом поділу документа на терміни є так званий ngram [3]. *N*-грами – це всі поєднання сусідніх слів або букв довжиною *n*, які можна знайти у вихідному тексті. Наприклад, якщо врахувати слово “fox”, усі 2-грами (або “біграми”) – це “fo” і “ox”. Також можна порахувати межу слова – це розширить список 2-грамів до “f”, “fo”, “ox” та “x”, а можна зробити те саме на рівні слова [4]. Як приклад, “hello world” текст містить такі біграми на рівні слова: “hello”, “world”, “hello world”. Основною функцією *n*-грамів є те, що вони фіксують мовну структуру

зі статистичного погляду, наприклад, яка буква чи слово, ймовірно, є наступною. Що довший *n*-грам (що вищий *n*), то більше контексту доведеться опрацювати. Оптимальна довжина справді залежить від програми, у разі невідповідності не вдасться здобути бажаного результату. В кінці отримуємо стовпець "Документи/Списки проводки", який допомагає визначити місцезнаходження терміна в нашій колекції. Наприклад, визначити, в якому документі вжито конкретний термін, або на якій позиції в документі використано термін тощо. Іншу інформацію, таку як частота терміна в документі, позиція терміна в документі тощо, також можна зберегти у стовпці списків проводки з ідентифікатором документа. Саме так ми зможемо правильно зберегти наші дані у сховищі.

Кожен користувач схильний робити помилки під час пошуку. Тому виникає проблема: як визначити, чи збережений термін у сховищі збігається із пошуковим терміном користувача? Тут допоможе відстань Левенштейна [5] – метрика, основна мета якої – виміряти різницю букв між двома послідовностями. Інакше кажучи, відстань Левенштейна – це мінімальна кількість замін символів або редагувань (тобто вставок, видалень чи підстановок), необхідних для того, щоб змінити одне слово на інше. Цей алгоритм названо на честь Володимира Левенштейна, який вираховував цю відстань у 1965 р. Відстань Левенштейна [6] також має іншу назву – відстань редагування, хоча ця назва може охоплювати більшу сім'ю метрик відстані. Це тісно пов'язано з попарним вирівнюванням рядків. Математично відстань Левенштейна [7] між двома рядками *a*, *b* (довжини | a | та | b | відповідно) задається левами, *b* (| a |, | b |):

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases} \quad (1)$$

де 1 (*ai* ≠ *bi*) – функція індикатора, дорівнює 0, коли *ai* ≠ *bi*, й 1 в іншому випадку, a leva, b (*i*, *j*) – відстань між першими *i* символами *a* та першими *j* символами *b*. Перший елемент у мінімумі відповідає видаленню (від *a* до *b*), другий – вставці, а третій – збігу або невідповідності, залежно від того, чи відповідні символи однакові. Відстань Левенштейна [8] між "FLOMAX" і "VOLMAX" дорівнює 3, оскільки три заміни у якомусь із цих слів змінюють одне слово на інше, і неможливо виконати ці зміни за допомогою менше ніж трьох редагувань. Відстань Левенштейна між "GILY" та "GEELY" дорівнює 2 (рис. 1, *а*).

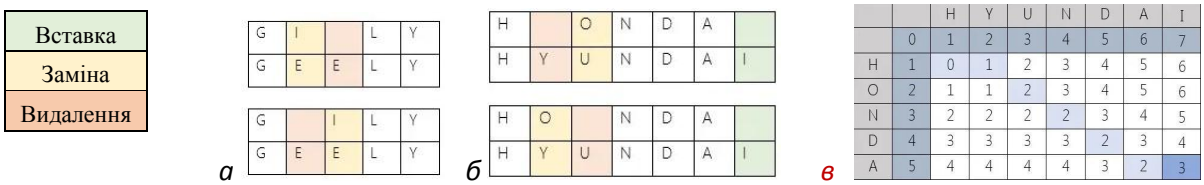


Рис. 1. Відстань між: а – "GILY" і "GEELY"; б – "HONDA" і "HYUNDAI"; в – "HONDA" і "HYUNDAI"

Відстань Левенштейна між "HONDA" та "HYUNDAI" становить 3 (рис. 1, *б*).

Алгоритм Левенштейна обчислює найменшу кількість операцій редагування, необхідних для модифікації одного рядка для отримання іншого рядка. Найпоширеніший спосіб обчислення – це підхід динамічного програмування:

- Ініціалізують матрицю, вимірюючи у (*m*, *n*) клітинку відстань Левенштейна між префіксом *m*-символів одного з *n*-префіксом іншого слова.
- Матрицю можна заповнити з верхнього лівого до нижнього правого кута.
- Кожен стрибок по горизонталі чи вертикалі відповідає вставці або видаленню.
- Зазвичай вартість встановлюється такою, що дорівнює 1, для кожної з операцій.
- Перехід по діагоналі може коштувати або один, якщо два символи в рядку та стовпці не збігаються, або 0, якщо збігаються. Кожна клітина завжди мінімізує витрати на місцевому рівні.
- Число в нижньому правому куті – це відстань Левенштейна між обома словами.

Приклад, що містить порівняння “HONDA” та “HYUNDAI”, подано на рис. 1, *в*.

За часткового збігу рядків мета пошуку – знайти збіги коротких рядків у великих текстах. Наприклад, такі короткі рядки можуть міститися у словниках, де, зазвичай, один з рядків є коротким, а інший може бути довільно довгим. Цей алгоритм має безліч застосувань. Наприклад, можна виконувати перевірку правопису, застосовувати у системах корекції оптичного розпізнавання символів та програмному забезпеченні для перекладу на природні мови.

Відстань Левенштейна також можна використати для того, щоб обчислити відстань між двома довшими рядками. Але, враховуючи вартість цього обчислення, можна допустити, що це недоцільно, адже вартість пропорційна до добутку двох довжин рядків. Тому варто використовувати цей алгоритм для нечіткого пошуку рядків, щоб допомогти покращити швидкість порівняння. Отже, ці підходи дають змогу побудувати систему, здатну шукати релевантні результати у сховищі. Розглянемо основні моменти пов’язані з вибором рекомендаційного алгоритму.

Система рекомендацій [9] – програма машинного навчання, яка надає користувачам рекомендації щодо їх вподобань із урахуванням їхніх історичних уподобань. Далі її можна визначити як систему, яка видає індивідуальні рекомендації як вихідні дані або наслідком має персоналізоване скерування користувача до цікавих об’єктів у більшому просторі можливих варіантів. Приклади:

· Пропонувати читачам статті новин, зважаючи на їхні інтереси.

· Надавати клієнтам пропозиції щодо того, що вони можуть придбати, беручи до уваги історію їхніх покупок або пошуків.

Виділяють три основні типи рекомендованих систем (табл. 3) [10–12].

Таблиця 3

Порівняльна таблиця рекомендаційних алгоритмів

	Переваги	Недоліки	Застосування
Колаборативна фільтрація	Враховуються оцінки і рейтинги користувачів, незалежно від тематики сервісу	Низька продуктивність для перших користувачів. Необхідна велика кількість інформації про оцінки користувачів	Інформаційні портали. Невеликі інтернет-магазини
Фільтрація на основі вмісту	Миттєво спрацьовує, навіть для перших користувачів. Коректно працює, навіть якщо даних мало	Зав’язана на контент сервісу. Не спирається на побажання користувачів	Блоги. Музичні або кіно-платформи
Гібридна фільтрація	Висока продуктивність. Відсутність проблем попередніх підходів	Важко підтримувати. Складний процес розроблення	Великі інтернет-магазини. Складні системи з великою кількістю користувачів

Розглядаючи табл. 3, можна зробити висновок, що для реалізації нашої системи [13] стане у нагоді алгоритм колаборативної фільтрації, адже плюси цього методу ідеально підходять для пошукової системи, а мінуси надто незначні, щоб не вибрати цей метод. Також порівняймо алгоритми нечіткого пошуку. Числа, які можна побачити на рис. 2, – це точність збігів слів між першою і другою колонкою, від 0 до 1. Порівнюючи усі результати, можна зробити висновок, що для нашої системи ідеально підходить алгоритм Левенштейна, адже в такому випадку точність пошукових результатів буде найвищою. Така точність досягається завдяки тому, що алгоритм Левенштейна порівнює слова, а не словосполучення, враховуючи довжину зміни слів або кількість процесів видалення, вставки, заміни букв у слові.

Word A	Word B	Cosine	Levenshtein	Trigram	Jaro-Winkler
Twitter	twitter	0.925	1.0	0.667	1.0
chien	niche	1.0	0.25	0.0	0.6
twitter v1	Twitter v2	0.866	0.5	0.6	0.96
ShazamIphone	ShazamAndroid	0.667	0.167	0.235	0.866
Famous Instagram SW	Famous Instagram	0.951	0.333	0.824	0.968
Int Facebook	CI Facebook	0.889	0.333	0.583	0.914
Int Facebook	Instagram Int	0.525	0.1	0.05	0.638

Рис. 2. Порівняння точності обчислень методів нечіткого пошуку

Отже, ми здійснили аналітичний огляд основних, популярних методів і алгоритмів для вирішення проблем побудови пошуково-рекомендаційної системи, а також виділили основні переваги і недоліки різних типів рекомендаційних систем. Порівняли точність та доречність використання алгоритмів нечіткого пошуку. Основною проблемою залишається те, що системи, популярні серед користувачів, такі як Netflix, Amazon, Rozetka, не публікують деталі реалізації свого пошукового двигуна. Крім того, із наведеного вище зрозуміло, що існує доволі багато варіантів вирішення однієї проблеми, зі своїми недоліками і перевагами. Оскільки тема цієї роботи – сфера e-commerce, цілком зрозуміло, що необхідно будувати власну систему, з коректно підібраними алгоритмами саме для цієї сфери, яка б задовольняла вимоги бізнесу.

Формулювання мети та постановка задачі

Мета роботи – створення технології інтелектуального пошуку товарів з формуванням списку рекомендацій для користувача. Для цього необхідно визначити, як необхідно зберігати та аналізувати інформацію про товари, які алгоритми рекомендацій можна застосувати у таких системах, як забезпечити зручне користування системою. Об’єкт дослідження – процеси інтелектуального пошуку з можливістю генерування рекомендацій для користувачів у сфері е-комерції без прив’язки до категоризації товару/послуг тощо. Предметом дослідження є методи та засоби інтелектуального пошуку рекомендаційних систем на основі алгоритму Collaborative Filtering для формування рекомендацій товарів для користувачів, що орієнтується на загальні збіги вибору подібних користувачів. Технологія також має ґрунтуватися на алгоритмі розбору й аналізу слів для пошуку, що забезпечує точний результат інтелектуального пошуку, з допуском введення помилки у пошукових словах. Необхідно додати можливість автозавершення слів під час пошуку в цій системі.

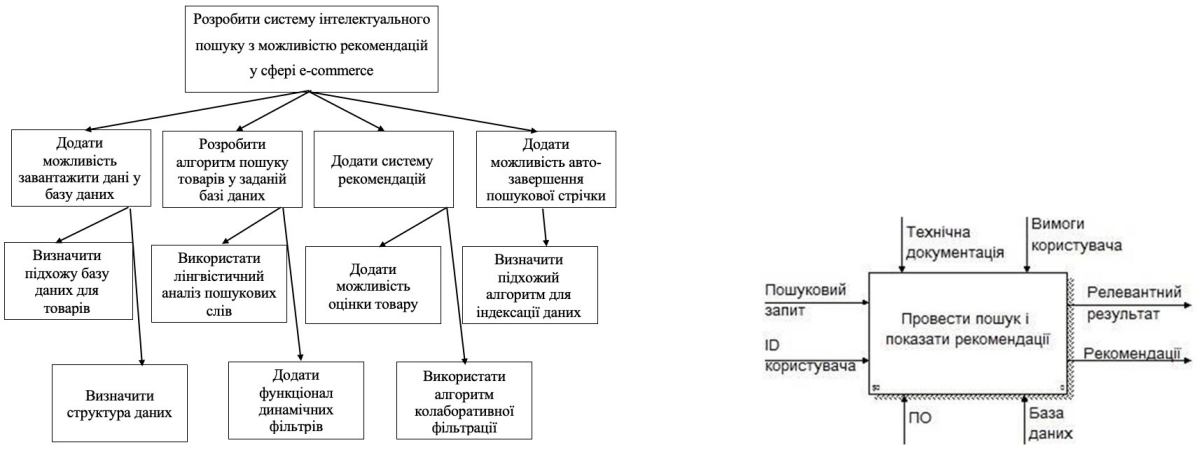


Рис. 3. Дерево цілей та IDEF0 першого рівня

Основна мета цієї системи – надати можливість користувачеві зручно і швидко знаходити необхідний товар. Це охоплює декілька аспектів [14], таких як можливість автозавершення

пошукових слів, допуск помилок у словах під час пошуку тощо. Крім того, система повинна вміти генерувати рекомендації для користувача, враховуючи покупки схожих користувачів.

Оскільки основна мета нашої система – надати можливість пошуку і рекомендацій для користувача, то основним механізмом буде здійснення пошуку і формування рекомендацій для користувача (рис. 3). Основні вхідні параметри цього процесу – ID користувача і власне пошуковий запит. Механізмами, які забезпечують коректну роботу цього процесу, будуть база даних або сховище, програмні бібліотеки. Керуючими механізмами будуть технічна документація, а також зібрані вимоги користувачів. Результатом цього процесу буде релевантний результат, згідно із запитом, а також рекомендації для користувача.

Виклад основного матеріалу

Спробуємо декомпонувати описану вище модель. Спершу виділимо декілька основних, менших блоків (рис. 7). Під час початкового звернення до програми необхідно ініціалізувати вхідні параметри. Тому на вході в перший процес потрібні ID користувача і пошуковий запит. Після ініціалізації можемо переходити до наступних процесів. Для успішного пошуку, передусім, необхідно проаналізувати вхідний текст з погляду програми [16]. Тому для процесу “Провести передпошукову обробку тексту” (рис. 7) вхідним параметром буде пошуковий рядок. Використовуючи лінгвістичний аналіз, ми зможемо опрацювати рядок до необхідного вигляду. В цьому нам допоможуть деякі готові програмні модулі, які можна підключити. Відтак, використовуючи оброблений рядок, знайдемо необхідні документи у сховищі. Тому вхідним параметром процесу “Провести пошук” є оброблений пошуковий рядок, а механізмами, які допоможуть, – база даних і програмні бібліотеки. Тепер перейдемо до іншого процесу “Провести аналіз рекомендацій” (рис. 4). На вході до процесу є тільки ID користувача. Оскільки дані про покупки користувачів зберігаються у сховищі, механізмом, котрий допоможе цьому процесу, є база даних. Використовуючи алгоритм колаборативної фільтрації і знаючи ID користувача, ми зможемо проаналізувати дані й надати рекомендації для цього користувача. Після цього рекомендації та результати пошуку переходять до наступного, фінального, процесу “Сформувати коректну відповідь”. Оскільки на фінальну відповідь можуть впливати певні вимоги користувача, до того ж користувачеві потрібно показати тільки певну частину інформації, нам необхідно сформувати відповідь, що міститиме тільки найкращі рекомендації, а також релевантний результат.

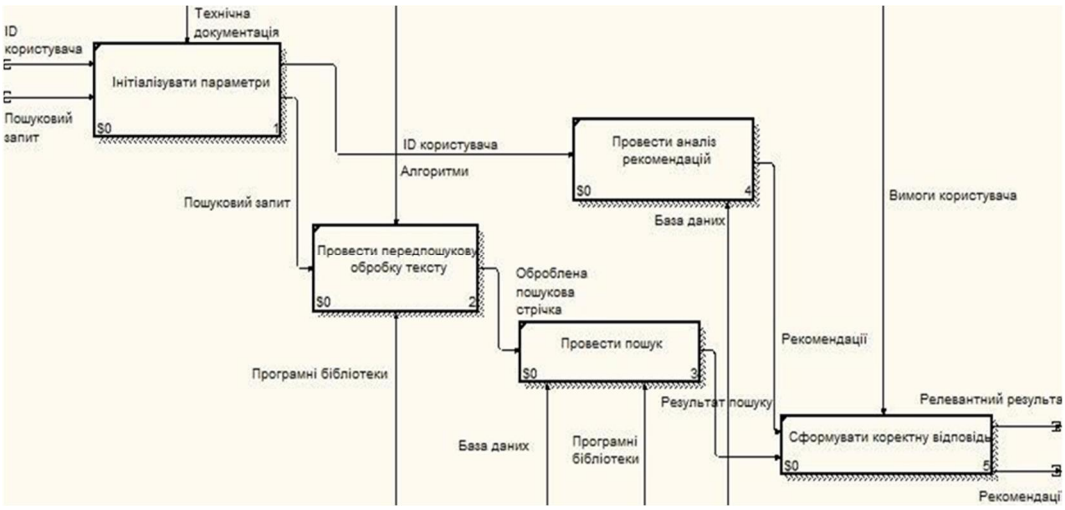


Рис. 4. IDEF0 другого рівня

Спробуємо декомпонувати процес передпошукового оброблення тексту (рис. 5). Спочатку, щоб виокремити якусь логічну одиницю в тексті, поділимо пошуковий запит на токени. Токен можна

розглядати як одне слово. Далі всі наші операції та процеси використовуватимуть і модифікуватимуть щойно сформовані токени. Для єдиного формату літер переведемо токени в один регістр (рис. 5). Стоп-слова – це слова будь-якою мовою, які не додають особливого значення реченню. Їх можна сміливо ігнорувати, це не зашкодить змісту речення. Тому третій процес вилучить усі токени, які є стоп-словами. Для успішного виконання цього процесу необхідно надати словник стоп-слів. Після цього відфільтровані токени йдуть у фінальний процес – стемінг токенів. Стемінг-алгоритм – це процес лінгвістичної нормалізації, під час якої варіантні форми слова зводяться до загальної форми. Тобто залишається переважно основна, корінна частина слова без суфіксів і закінчень. Після цього процесу наші токени, або опрацьований текст, готові для подальшого пошуку.

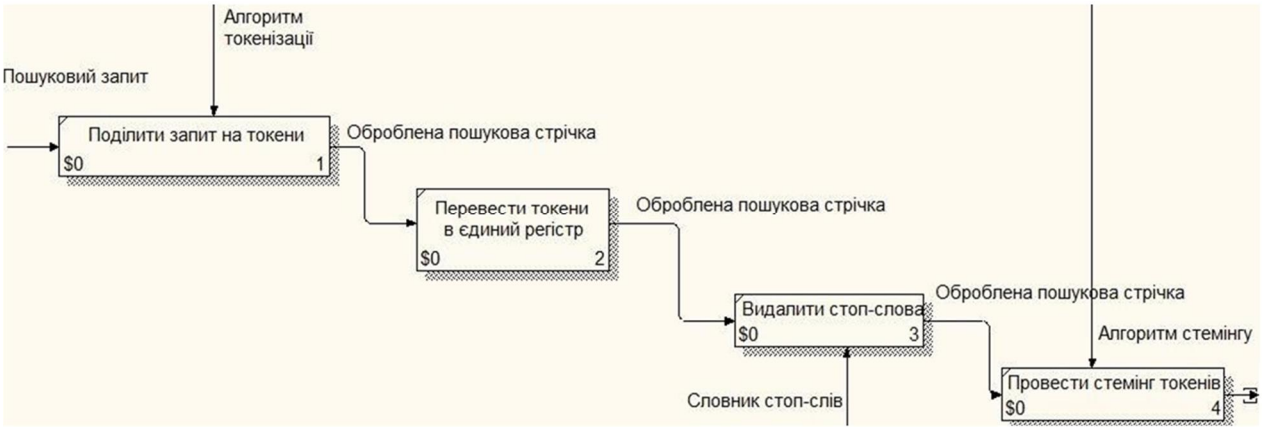


Рис. 5. IDEF0 третього рівня

Також розширимо процес аналізу рекомендацій (рис. 6). Під час першого процесу необхідно видобути дані про покупки користувачів. У цьому нам допоможе сховище або база даних. Маючи дані про покупки покупців, переходимо до наступного процесу. Тут необхідно застосувати алгоритм колаборативної фільтрації, який зможе надати передбачення щодо товарів, котрих користувач ще не придбав, на основі вибору товарів схожих користувачів. Дані, які будуть на виході виконання цього процесу, необхідно проаналізувати і вибрати найкращі рекомендації. Щоб вибрати найкращі рекомендації, необхідно керуватися вимогами користувачів. Після цього процесу можемо передати рекомендації до наступного процесу. Основою для побудови ієрархії процесів є ієрархія DFD. DFD [17] – спеціальна нотація, мета якої є моделювання інформаційних систем з погляду зберігання, оброблення і передавання даних. У нашій системі основною зовнішньою сутністю є користувач (рис. 7). В потоці даних він може передати свій ID, а також запит. Основним процесом або системою є Пошукова система. На виході система віддає релевантний результат і рекомендації для користувача. Перейдемо до DFD діаграми другого рівня. Частково вона схожа на IDEF0, тому деякі процеси можемо пропустити. Варто підкреслити, що на цій діаграмі в нас є три сховища даних, а саме: сховище інформації про покупки клієнтів, словник стоп-слів, необхідний для лінгвістичного аналізу, а також сховище інформації про товари підприємства.

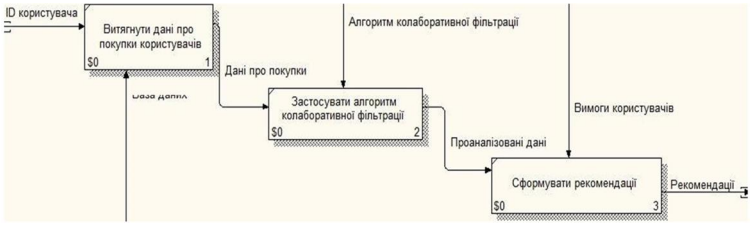


Рис. 6. IDEF0 третього рівня

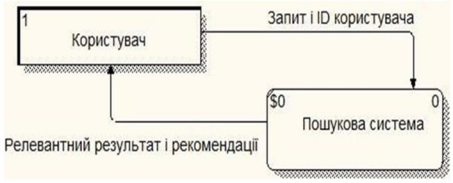


Рис. 7. DFD першого рівня

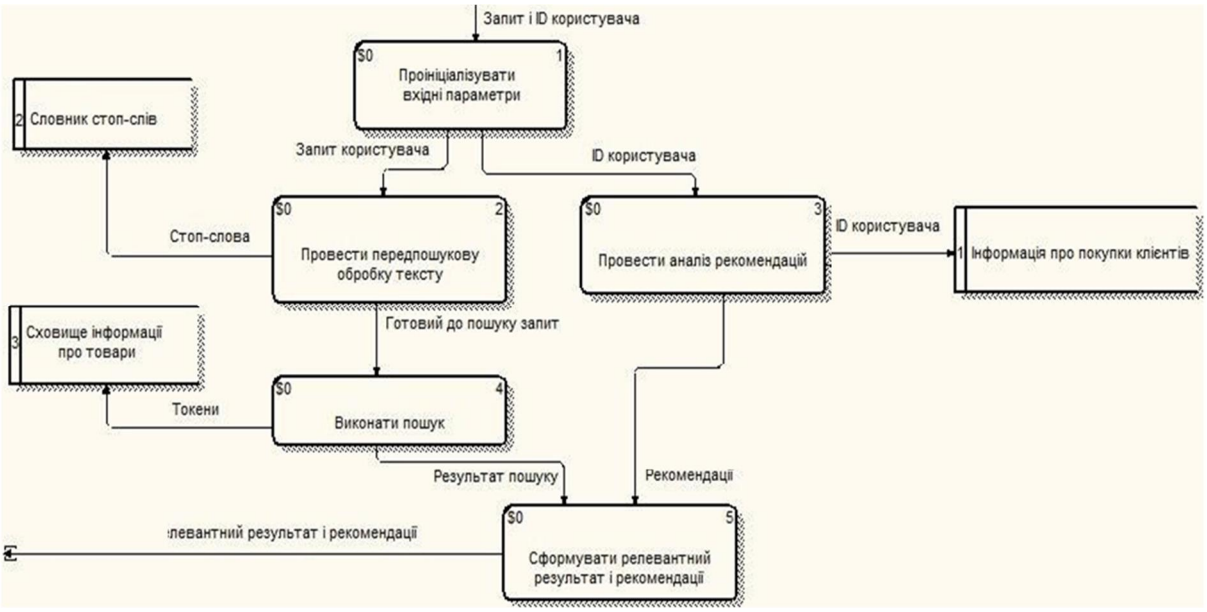


Рис. 8. DFD другого рівня

Декомпонуючи процес передпошукового оброблення тексту, отримуємо DFD діаграму (рис. 9). Одразу можна виділити, що на цьому рівні для процесу видалення стоп-слів ми використовуємо сховище даних – Словник стоп-слів. Загалом, процес передпошукової обробки складається з чотирьох основних процесів, котрі зв'язані між собою, а саме: поділ пошукового рядка на токени або слова, модифікація токенів у єдиний формат, фільтрація надлишкових слів, а також стемінг слів або виділення лише корінної частини слова. Після проходження цих процесів ми зможемо отримати повністю готовий до пошуку рядок або групу слів, які можемо надсилати далі. Всередині цих процесів будуть використовуватися розглянуті вище алгоритми. Важливо, щоб ці процеси були виконані в тій послідовності, у якій вони зараз, адже це прямо впливає на коректну роботу алгоритмів. Так само можна зробити з процесом аналізування рекомендацій. Сховищем даним тут є інформація про покупки клієнтів, зокрема і користувача, котрий робить запит. Надалі, використовуючи рекомендаційні алгоритми, зможемо припустити, що певний товар буде доречним для нашого користувача. Після цього можна сформувати рекомендації і передати їх до наступного процесу.

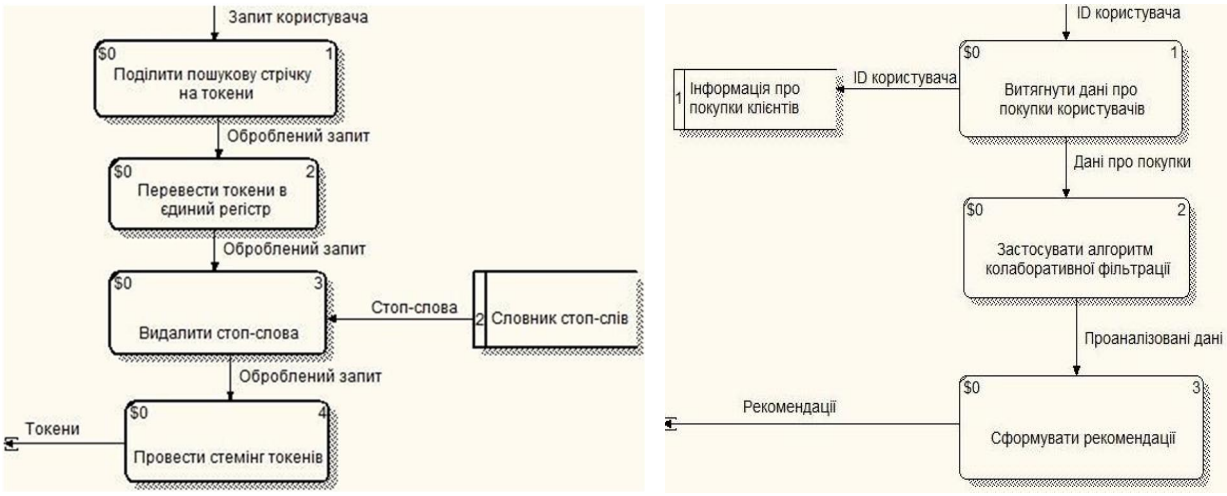


Рис. 9. DFD третього рівня

Розпочнемо зі сховища даних, де зберігатиметься інформація про всі товари. Цим сховищем вибрано Elasticsearch. Elasticsearch [18] – це високомасштабований механізм повнотекстового пошуку та аналітики з відкритим кодом. Для зберігання даних про покупки користувачів, а також оцінювання продукції вибрано MySQL [19] із урахуванням її доступності, надійності, можливості швидкого запуску.

Основною мовою програмування вибрано Java [20] із урахуванням відповідних переваг: ціна (безкоштовно); продуктивність (завдяки компілятору HotSpot JIT код виконується дуже швидко); ефективність (широкий спектр бібліотек, що містять оптимізований код, який легко писати); портативність (програми, написані на Java, можуть працювати практично на будь-якому пристрої); підтримка мови (компанія-розробник продовжує покращувати можливості мови, виправляючи помилки, підвищуючи продуктивність, а також спрощуючи написання коду).

Перейдемо до вибору алгоритмів. Основним алгоритмом для рекомендацій вибрано алгоритм Slope One. Алгоритм Slope One [22] – це система спільної фільтрації на основі оцінки елементів. Це означає, що він повністю оснований на рейтингу позицій користувача. Обчислюючи подібність між об’єктами, ми знаємо лише історію оцінок і рейтингів, а не сам вміст. Потім ця подібність використовується для прогнозування рейтингу потенційних користувачів для пар користувач-елемент, яких немає у наборі даних. Спочатку користувачі оцінюють різні елементи в системі. Далі алгоритм обчислює подібність. Після цього система робить прогнози щодо рейтингу товарів, які користувач ще не оцінив. Спільна фільтрація або CF [23, 24] – це спеціальна техніка або алгоритм, що переважно використовується системами, які повинні щось рекомендувати. У загальному розумінні спільну фільтрацію можна описати як процес фільтрації інформації із застосуванням методів, які використовують співпрацю між різними джерелами даних. Зазвичай метод спільної фільтрації використовують у разі дуже великого набору даних.

Для системи вибрано Java версії 8. Можна виділити такі переваги цієї версії [25]: краще виведення типу; лямбда і функціональні інтерфейси; інтерфейси за замовчуванням і статичні методи; відмінність між методами за замовчуванням і абстрактними методами в тому, що абстрактні методи повинні бути реалізовані, а методів за замовчуванням немає; повторювані анотації.

Maven – це інструмент управління збіркою, він визначає, як ваші файли .java збираються до .class, упаковуються у файли .jar (або .war або .ear) (попередньо/після), обробляються інструментами, керуючи вашим CLASSPATH, та виконуються всі інші різновиди завдань, необхідних для побудови вашого проєкту. Це схоже на Apache Ant або Gradle або Makefiles в C / C ++, але він намагається бути повністю автономним у ньому, тому що не потрібні ніякі додаткові інструменти чи сценарії, ураховуючи інші загальні завдання, такі як завантаження та встановлення необхідних бібліотек тощо.

Liquibase [26] – це рішення для управління змінами у схемі бази даних з відкритим кодом, яке дає змогу легко керувати редакціями змін у вашій базі даних. Liquibase являє собою систему управління версіями бази даних, переважно це стосується структури і меншою мірою вмісту бази. Опис бази, з одного боку, доволі абстрактний і дає змогу використовувати на нижньому рівні різні СУБД, а з іншого боку – завжди можна перейти на SQL-діалект конкретної СУБД, що доволі гнучко. Liquibase є усталеним проєктом з відкритим вихідним кодом, активно використовується за межами Java середовища і не потребує глибоких знань Java для роботи. Як опис структури бази і змін бази історично використовувався XML формат, проте тепер паралельно підтримується YAML і JSON. Liquibase використовує changeSets для представлення однієї зміни у вашій базі даних.

ChangeSet використовується для групування баз даних разом, і є одиницею змін, яку Liquibase виконує у базі даних. ChangeLog – це список змін, створених кількома наборами змін

Набір змін визначається трьома елементами: “ID” та “Author” та шлях до імені файла ChangeLog. Liquibase дає змогу виконувати такі завдання: підтримка відкату та автоматичне оновлення; вихід майбутнього відкату; контексти ChangeSet: ChangeSet можуть бути призначені “контекстами”, в яких вони запускатимуться; передумови ChangeLog та ChangeSet: передумови можуть бути додані до changeLog або окремі changeSets для перевірки стану бази даних перед спробою їх виконання; ChangeSet контрольні суми: Коли changeSet виконується, Liquibase зберігає контрольну суму і може не виконати або змінити виконання, якщо виявить зміну між початковим визначенням changeSet, коли він був запущений, і поточним визначенням; підтримка різниць.

Spring – найпопулярніша програма розроблення додатків для корпоративної Java. Переваги Git: безкоштовний і open-source; невеликий і швидкий; резервне копіювання; просте розгалуження.

Назва програми: “Search Engine”. Програма може бути запущена практично на будь-якій платформі: Linux, Windows, MacOS, тощо. Здебільшого програма написана на мові Java із використанням фреймворку Spring. Використовуються дві бази даних: MySQL (для збереження даних про покупки та рекомендації); Elasticsearch (для збереження пошукових документів та індексації даних). Для спрощення керування схемами бази даних використовується фреймворк Liquibase, котрий дає змогу легко змінювати структуру бази даних. Для керування залежністю між Java-бібліотеками використовують технологію Maven. Основні функції програми – це інтелектуальний пошук у документах, завантажених у сховище, а також можливість рекомендацій для користувачів. Крім того, для заповнення сховища даними створено модуль завантаження та індексації даних у сховище. Для симуляції покупки товарів створено спеціальну вхідну точку в програмі. Програма складається з трьох основних модулів: Data-loader, Engine-API, Engine-Server. На рис. 10 відображено основну взаємодію між модулями.

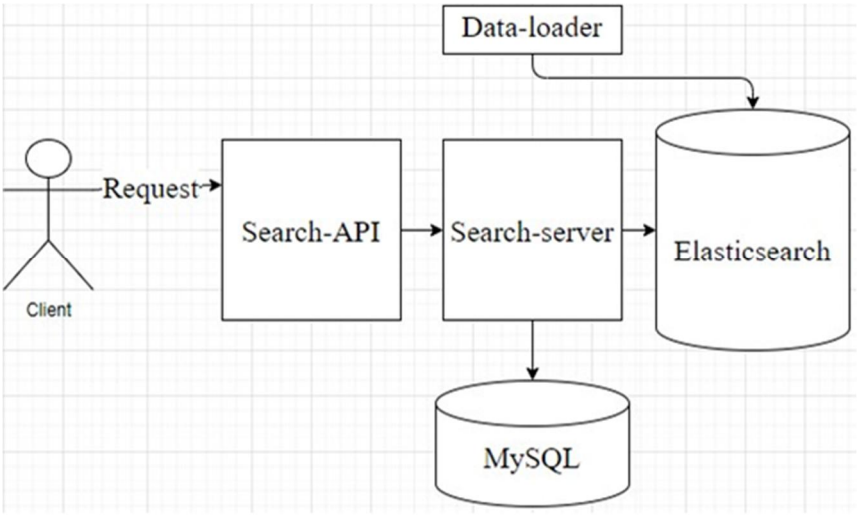


Рис. 10. Взаємодія між програмними модулями

Data-loader – модуль, необхідний для завантаження даних у сховище, містить у собі структуру документа. Документ може складатися з різних полів. Тому необхідно створити загальну структуру, враховуючи поля усіх документів з їхніми типами. Приклад такої структури подано на рис. 15. Крім того, необхідно вказати формат, у якому зберігатимуться дані у сховищі, тобто набір фільтрів, через

які проходитиме завантажуваний текст і перетворюватиметься на токени. Це необхідно для того, щоб у майбутньому можна було знайти шуканий документ за допомогою інвертованого індексу. Крім того в документі на рис. 11, *а* можна вказати бажані налаштування Elasticsearch індексу і кластера індексів.

Search-API – вхідний модуль, який приймає усі вхідні запити. В ньому вказано який тип має кожен вхідний запит, разом з усіма вхідними параметрами.

Рис. 11. Структура документів (а), приклад Pipeline (б), файл з агрегаціями (в), файл з визначеними варіантами сортування (г) та приклад з структурою таблиць (д)

Search-server – основний модуль, де виконується уся основна логіка програми. Для того, щоб було зручно керувати опрацюванням кожного запиту, введено таке поняття, як Pipeline. Pipeline – це певна послідовність виконання програмного коду. Це поняття високого рівня коду, тому може використовуватися універсально для кожного типу запитів. Pipeline складається зі стадій, або Stage-s. Кожна стадія, або Stage, у певний спосіб обробляє вхідні параметри і передає оброблені параметри далі на наступну стадію. Для зручності опису цих Pipeline, а також їхніх етапів створено спеціальний структурованих формат даних. Приклад такого Pipeline можна побачити на рис. 11, *б*. Насамперед у полі “name” задають ім’я цього Pipeline, далі – основні етапи виконання цього Pipeline. В проєкті містяться три основних Pipeline: /search (процес виконання інтелектуального пошуку); /search/recommendation (процес виконання алгоритму рекомендацій); /search/suggestion (процес генерування автопошукових слів). Крім того, для зручності пошуку використовують фільтри, які ґрунтуються на такому понятті, як агрегація. Для зручного використання агрегацій створено спеціальний документ зі строго визначеною структурою, в якому міститься опис основних агрегацій, а також їхніх можливостей. На рис. 11, *в* наведено приклад такого документа. В цьому прикладі показано чотири фільтри: Shop (за виробником), Discount (зі знижками), Price (за вибраною ціною) та Rate (за рейтингом).

Для сортування знайдених результатів за певними критеріями використовують поняття Sort. Для зручного і швидкого використання нових варіантів сортування сформовано документ, в якому можна описати бажане сортування, вказавши необхідне поле. На рис. 11, *г* наведено приклад такого

файлу, із двома варіантами сортування: Default (сортує поле "_score" за зниженням) та Cheap (сортує поле "product_price" за зростанням).

Для визначення структури бази даних, як згадувалося раніше, використовується технологія Liquibase. Для цієї технології необхідно створити відповідний файл з назвою "liquibase-changeslog.yml". Приклад такого файлу подано на рис. 11, *д*. Він визначає основні таблиці у базі даних, а також поля і їх тип. Нижче подано основні таблиці, які використовуються у програмі (табл. 4, 5).

Таблиця 4

Опис таблиць у базі даних

Таблиця	Поля	Опис
purchases	id – int; itemId – varchar(50); user – varchar(50); date – datetime; keywords – varchar(50)	Дані про покупки товарів
rates	id – int; itemId – varchar(50); keywords – varchar(50); user – varchar(50) ;date – datetime; rate - int	Оцінки товарів
recommendations	id – int; itemId – varchar(50); user – varchar(50)	Список рекомендацій

Вихідний код міститься на сервісі GitHub. Після завантаження вихідного коду з кореневої точки коду необхідно викликати команду: mvn clean install. Після цього будуть сформовані .jar файли кожного файлу. Для запуску програми необхідно перейти в кореневу директорію engine-api і виконати команду java-jar engine-api.jar. Програма містить шість основних точок входу після запуску сервера (табл. 5). В табл. 5 також наведено вихідні дані та їх формат для кожної вхідної точки.

Таблиця 5

Опис вхідних та вихідних точок

Вхідна точка /search	Параметри	Опис
query – пошукове словосполучення category – категорія товару count – бажана кількість повернутих результатів sort – послідовність сортування		Основна вхідна точка для пошуку товарів
/suggestion	query – частина слова або словосполучення count – бажана кількість повернутих результатів	Точка автозавершення слова або словосполучення, якщо в Query є незакінчені
/recommendation	userId – ідентифікатор користувача	Видає список рекомендацій для користувача
/inner/recommendation	userId – ідентифікатор користувача	Запускає алгоритм колаборативної фільтрації
/shop	itemId – ідентифікатор продукту userId – ідентифікатор користувача category – категорія товару	Записує покупку товару з вибраним ідентифікатором для вказаного користувача
/rate	itemId – ідентифікатор продукту userId – ідентифікатор користувача category – категорія товару rate – оцінка товару	Вхідна точка для оцінки товару з вибраним ідентифікатором для вказаного користувача

Продовження табл. 5

Вхідна точка /search	Формат
items – масив товарів такої структури: id – ідентифікатор товару source – виробник товару sourceUrl – посилання на виробника query – ключові слова товару productImage – фото товару productTitle – заголовок товару productDescription – опис товару productPrice – ціна продукту productPriceWithSale – ціна продукту зі знижкою onSale – вказує, чи є знижка на товар	
/suggestion	suggestion – автозавершене слово
/recommendation	topItems – масив товарів, які входять в топ-10 продажу recommendations – масив товарів, які рекомендує система для вказаного користувача bucketModels – масив доступних фільтрів, з такою структурою: name – варіант вибору в фільтрі count – кількість товарів із таким фільтром
/inner/recommendation	–
/shop	–
/rate	–

Повна назва програми: “Search-engine”. Скорочена назва програми: “SrchNgn”. Основна мова програмування: Java. Середовище розроблення: IntelliJ IDEA Community edition. Основне призначення програми: надати користувачу можливість інтелектуального пошуку і сфері e-commerce, а також рекомендації. Програма може застосовуватись для інтернет-магазинів малого і середнього бізнесу. Система здатна знаходити релевантний товар згідно із пошуковим запитом. Крім того, якщо пошуковий рядок неповний або містить помилки, система здатна знайти найрелевантніший продукт. На рис. 12 подано приклад пошуку в системі, де пошуковий рядок містить два слова, два з яких неповні. Система здатна видавати топ-10 товарів, які купують інші користувачі, а також рекомендувати товари, які можуть сподобатися користувачеві, орієнтуючись на вподобання і оцінки схожих користувачів. Для реалізації нечіткого пошуку використано алгоритм Левенштейна, який підраховує кількість замін букв у слові й на основі цього обчислює шукану довжину слова. Щоб надавати рекомендації щодо товарів, використано алгоритм колаборативної фільтрації, який не залежить від конкретної тематики сайту. Для індексації даних використано NLP-алгоритми [27–33]. Наприклад, edge-gram [27], який розділяє одне слово на вибрану кількість частин. Середній час виконання пошукового запиту становить 10–250 мс. Початковий запит може тривати до 350 мс. Це пов’язано з тим, що система ще не оптимізувала критичний шлях виконання програми. Після оптимізації час виконання зменшується до 100 мс. На рис. 13 відображено час виконання запиту.

Рис. 12. Пошуковий функціонал та рекомендації

Програма може працювати безперебійно. Для оновлення даних необхідно індексувати дані новим індексом і після цього перезапустити програми, вказавши новий індекс з даними. Для запуску програми необхідно виконати такі умови: встановити: JDK 8 і прописати необхідні шляхи до JDK у

змінних середовища Windows; Maven, вказати необхідні шляхи до Maven у змінних середовища Windows; Elasticsearch 7.12 та MySQL 15.1.

Рис. 13. Час виконання запиту

Після завантаження вихідного коду з Git репозиторію необхідно в кореневій директорії завантаженого коду виконати команду **mvn clean install**. Ця команда згенерує .jar файли для кожного із модулів. Після цього потрібно завантажити дані, перейшовши для цього в папку з модулем data-loader і далі в папку target. Там міститься згенерований .jar файл. Для його запуску треба виконати команду **java -jar data-loader.jar.** Після завантаження даних потрібно створити необхідну структуру бази даних в MySQL. Для цього треба перейти в корінну директорію проєкту і виконати команду **mvn liquibase:update -Denv=dev.** Після створення схеми бази даних необхідно перейти в папку з модулем searchapi, потім в папку target, відтак виконати команду **java -jar search-api.jar.** Після цього матимемо змогу виконувати пошукові запити за адресою:

http://localhost:8080/search?query=

Для коректної роботи програми необхідно, щоб бази даних MySQL і Elasticsearch були запущені й містили потрібні таблиці та індекси. Спочатку перевіримо наявність даних у сховищі. Для цього використаємо такий інструмент, як Kibana, який зазвичай надається у комплекті з Elasticsearch. На рис. 14 зображено усі наявні у сховищі індекси. Нас цікавить індекс **ecommerce**. Як бачимо, нижчого від такого індексу немає. Для того, щоб створити індекс ecommerce, необхідно запустити модуль dataloader і вказати йому шлях до даних. Файл із даними подано на рис. 15. Модуль data-loader, знайшовши файл з даними, почне зчитувати його і трансформує всі дані файлу у Java об'єкти. Надалі, збираючи Java об'єкти у невеликі групи, по 10000 об'єктів, почне надсилати їх у щойно створений Elasticsearch індекс. Групуючи об'єкти у невеликі групи, можна досягти швидкого і безперебійного завантаження даних.

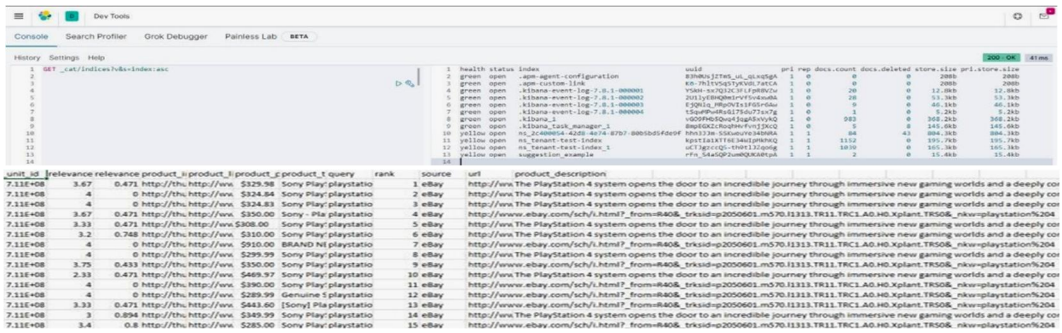


Рис. 14. Індекси у сховищі та файл з даними

Цей файл містить усю необхідну інформацію про товари. Для того, щоб оновити її, потрібно підготувати файл такої самої структури, видалити попередній індекс і запустити data-loader модуль. Завантажимо дані у сховище. Використовуючи IntelliJ IDEA, необхідно створити наступну точку

входу в програму і запустити її. Після виконання завантаження даних у сховище побачимо повідомлення, відображені на рис. 16.

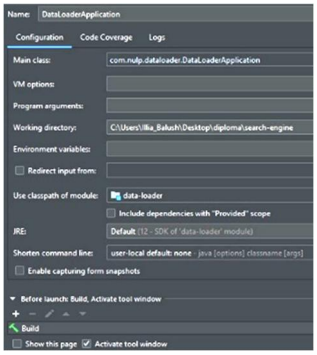


Рис. 15. Налаштування data-loader

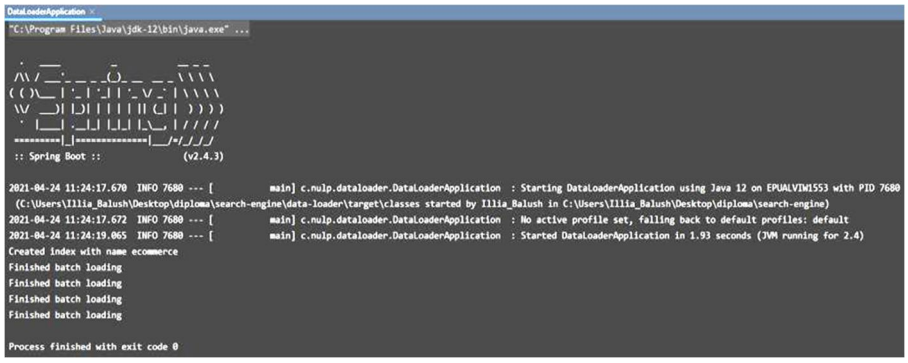


Рис. 16. Повідомлення після завантаження даних

Тепер перевіримо наявність даних у нашому сховищі. Для цього знову ж таки скористаємося Kibana. Виконаємо пошуковий запит за індексом з ім'ям ecommerce. Підготувавши дані, за якими відбуватиметься пошук, створимо таблиці у MySQL для моніторингу покупок і рекомендацій для користувачів. Використовуючи інструмент MySQL Workbench, бачимо (рис. 18), що ніякої бази даних у нас поки що немає. Наше завдання – створити відповідну базу даних, а також таблиці, необхідні для нормального функціонування програми. Виконавши команду **mvn liquibase:update-Denv=dev** у модуль engine-server, ми отримаємо повідомлення (рис. 18).

Рис. 17. Дані про товари у сховищі

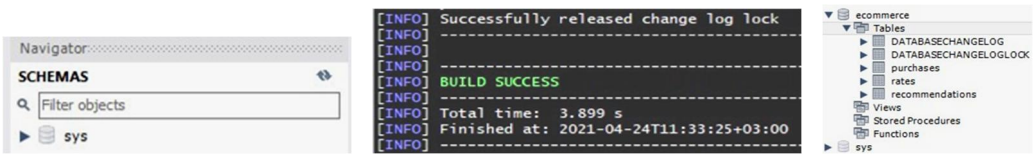


Рис. 18. MySQL, повідомлення про успішне створення БД та БД ecommerce

Ця команда створить всі необхідні компоненти, використовуючи технологію Liquibase. Вказавши в конфігурному файлі потрібні налаштування, ми зможемо однією командою створювати, повертати, зберігати базу даних і її вміст. І перевіримо знову базу даних. На рис. 18 бачимо нову базу даних ecommerce, а також таблиці: purchases, rates, recommendations. Ці таблиці, а також їхні налаштування ми вказували у файлі changeLog.yaml. Тепер, коли виконано всі необхідні умови для успішного запуску програми, спробуємо зробити це. Використовуючи IntelliJ IDEA, створимо вхідну

точку, як зображено на рис. 19. Після успішного запуску ми повинні побачити повідомлення, наведені на рис. 19. Виконаємо декілька пошукових запитів. Використаємо програму Postman для надсилання пошукових запитів. Спробуємо подивитись, які товари запропонує програма, шукаючи словосполучення “wi r” (рис. 19). На рис. 20 бачимо кілька товарів, які знайшла програма. Топ товарів, які задовольняють пошуковий запит: Wireless mouse – FORD SHELBY – безпровідна мишка; Wireless mouse – HP – безпровідна мишка; WII(R) – Sony – ігрова приставка; WII(R) – Nintendo – ігрова приставка; wii microphone – HP – мікрофон; wii microphone – Sony – мікрофон.

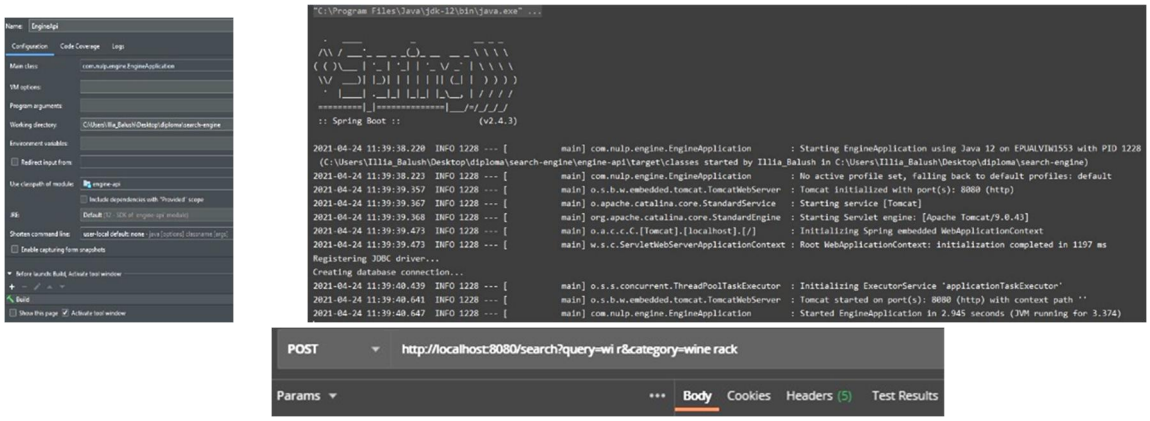


Рис. 19. Налаштування engine-api*, повідомлення після запуску, пошук за словосполученням “wi r”*

Рис. 20. Результат пошуку за словосполученням “wi r”

Всі описані вище продукти містять частину пошукової фрази, хоч пошукове словосполучення складається із двох незакінчених слів. Спробуємо знайти “Skinny Jeans”, але допустивши певні помилки в пошуковому запиті. Замість “Skinny Jeans” будемо шукати *“Skn Jen”* (рис. 20). На рис. 21 наведено результат пошуку. Програма знайшла товари, навіть попри нечіткий запит. В цьому допоміг алгоритм нечіткого пошуку Левенштейна. У найвищого результату найбільша оцінка схожості з пошуковим запитом згідно із алгоритмом Левенштейна. Крім того, на рис. 21 подано результат за релевантністю. Спробуємо посортувати ці результати за зниженням ціни.

Рис. 21. Результат пошуку за словосполученням “Skn Jen”

Ціна першого товару 6135.0 (рис. 22), а другого після нього – 4260.99 (посортовано за зниженням ціни). Таке сортування буде корисним для користувачів, які шукають найрелевантніший і дешевший варіант товару. Це збільшує шанс того, що користувач придбає щось на ресурсі, а також робить зручнішим користування сайтом. Перевіримо, як працює автозавершення пошукових слів. Для цього припустимо, що користувач написав у пошуковому рядку слово “rck”.

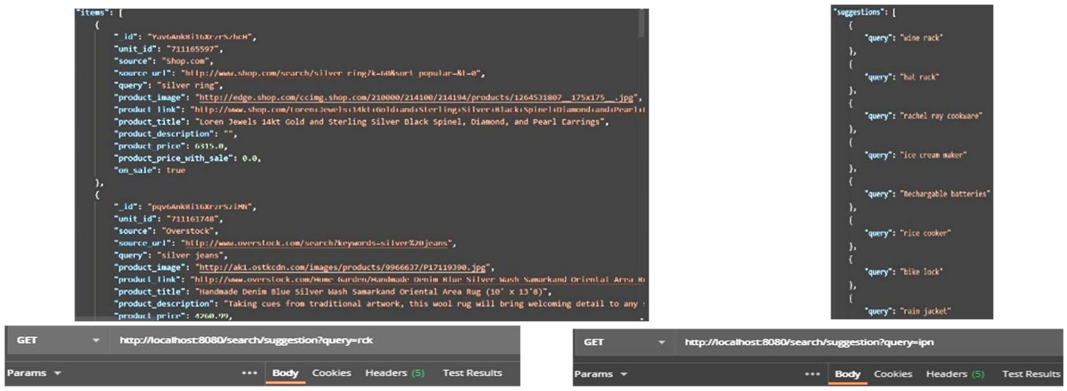
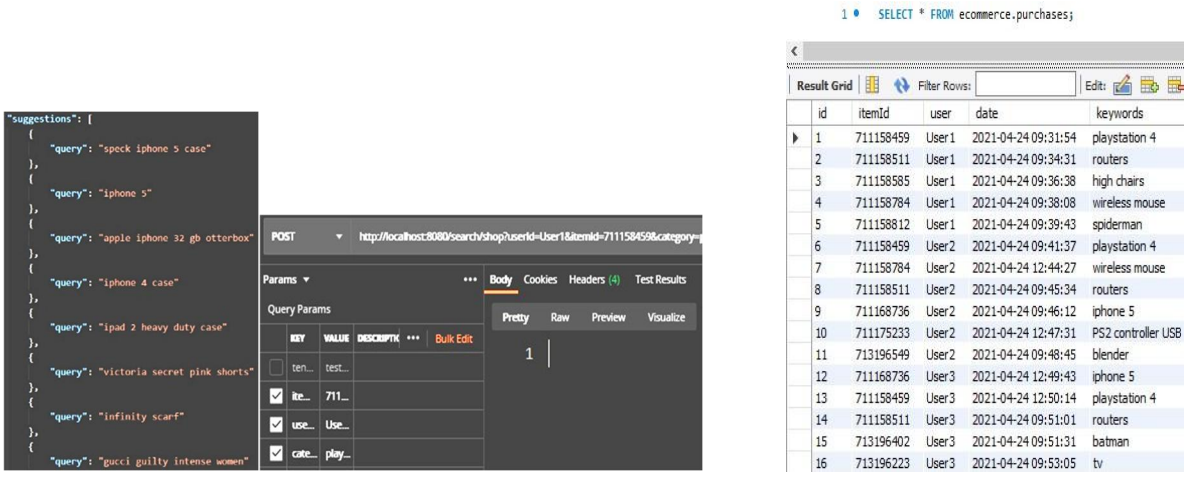


Рис. 22. Результат сортування, автозавершення слова “rck” та слова “Ipn”

Запропоновані програмою варіанти завершення слова “rck” => “wine rack”, “hat rack”, “Rachel ray cookware”, “ice cream maker” тощо (рис. 22). Послідовність цих результатів також визначено завдяки оцінюванню подібності до пошукової фрази, використовуючи алгоритми нечіткого пошуку. Спробуємо цей функціонал для слова “ipn”, маючи на увазі слово “iphone”. Згідно із рис. 23, *а* програма запропонувала такі варіанти завершення цього слова: “speck iphone 5 case”, “iphone 5”, “apple iphone 32 gb” тощо, що близько до шуканого. Найрелевантніші результати містяться вище. Що нижче запропонований товар, то менша його оцінка згідно із алгоритмом нечіткого пошуку. Ці варіанти автозавершення корисні, якщо користувач запам’ятав неповну назву товару або коли йому зручніше лише натиснути на запропонований варіант, замість того, щоб вводити повну назву.



а *б*

Рис. 23. Завершення слова “ipn” та приклад запиту для покупки товару

Тепер заповнимо таблицю покупок, використовуючи вхідну точку /search/shop. Нехай у нас є три користувачі User1, User2, User3 (рис. 24).

User1	Sony PlayStation 4 (PS4) (Latest Model)- 500 GB Jet Black Console (711158459) Porter-Cable 3-1/4 HP Five-Speed Router 7518 - Power Tools Routers (711158511) Cosco Slim Fold High Chair, Kontiki, Free Shipping, New (711158585) 2.4GHz Red Wireless Optical Mouse Mice (711158784) Anti-Venom IN HAND Marvel Legends (711158812)
User2	Sony PlayStation 4 (PS4) (Latest Model)-500 GB Jet Black Console (711158459) 2.4GHz Red Wireless Optical Mouse Mice (711158784) Porter-Cable 3-1/4 HP Five-Speed Router 7518 - Power Tools Routers (711158511) iPhone 5 Ease Fit Arm Band (711168736) Nintendo Wii U Pro Controller (711175233) Oster 14 Speed Blender (713196549)
User3	iPhone 5 Ease Fit Arm Band (711168736) Sony PlayStation 4 (PS4) (Latest Model)- 500 GB Jet Black Console (711158459) Porter-Cable 3-1/4 HP Five-Speed Router 7518 - Power Tools Routers (711158511) Batman Joker Graffiti (713196402) Samsung 60in 1080p 120hz smart led tv (713196223)

а

User	ps 4	routers	High chair	Wireless mouse	spiderman	Iphone 5	PS2	blender	batman	tv
1	5	4	3	4	3	?	?	?	?	?
2	4	4	?	4	?	3	5	4	?	?
3	2	3	?	?	?	5	?	?	4	3

б

Рис. 24. Списки покупок користувачів та оцінки користувачів

На рис. 23, *б* подано приклад запиту для покупки товару. Виконаємо запит до бази даних і перевіримо, чи всі покупки збережені. Для цього скористаємося MySQL і виконаємо запит прямо до бази даних, а саме в таблицю purchases. Тепер уявимо ситуацію, коли покупці починають оцінювати товар. Система передбачає випадок, коли покупець не може оцінити товар, якщо в системі немає запису про те, що він купував цей товар. Для оцінки товару будемо використовувати вхідну точку /search/rate. На рис. 24, *б* зручно переглянути оцінки всіх користувачів. На рис. 25 подано приклад запиту для користувача, щоб оцінити товар. Після оцінки товарів переглянемо дані про рейтинги у базі даних. Крім того, після кожної оцінки товару поле “rate” змінюватиметься залежно від середнього арифметичного всіх оцінок. На рис. 26 наведено дані товару до оцінок користувачів та після оцінювання. В таблиці (рис. 24, *б*) деякі клітинки містять “?”. Це означає, що ці покупці не мали змоги придбати вказаний товар. Тому цей товар може бути потенційно рекомендований користувачеві, якщо рейтинг товару перевищує 4. Запустимо рекомендаційний алгоритм і переглянемо результати рекомендацій (рис. 27).

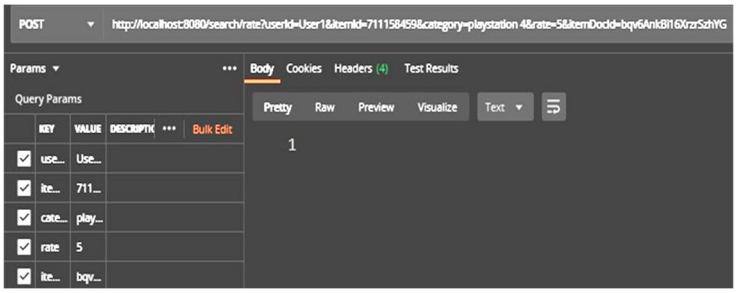
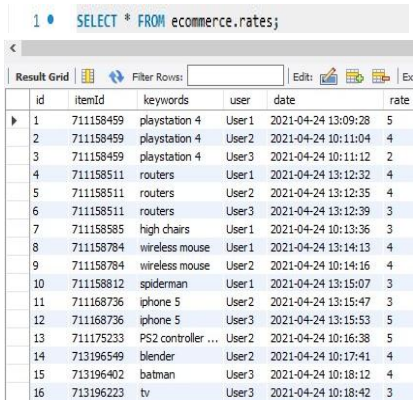


Рис. 25. Приклад запиту для оцінки товару



id	itemId	keywords	user	date	rate
1	711158459	playstation 4	User1	2021-04-24 13:09:28	5
2	711158459	playstation 4	User2	2021-04-24 10:11:04	4
3	711158459	playstation 4	User3	2021-04-24 10:11:12	2
4	711158511	routers	User1	2021-04-24 13:12:32	4
5	711158511	routers	User2	2021-04-24 13:12:35	4
6	711158511	routers	User3	2021-04-24 13:12:39	3
7	711158585	high chairs	User1	2021-04-24 10:13:36	3
8	711158784	wireless mouse	User1	2021-04-24 13:14:13	4
9	711158784	wireless mouse	User2	2021-04-24 10:14:16	4
10	711158812	spiderman	User1	2021-04-24 13:15:07	3
11	711168736	iphone 5	User2	2021-04-24 13:15:47	3
12	711168736	iphone 5	User3	2021-04-24 13:15:53	5
13	711175233	PS2 controller ...	User2	2021-04-24 10:16:38	5
14	713196549	blender	User2	2021-04-24 10:17:41	4
15	713196402	batman	User3	2021-04-24 10:18:12	4
16	713196223	tv	User3	2021-04-24 10:18:42	3

Рис. 26. Дані про оцінки користувачів

На рис. 28 бачимо, що після виконання запиту рекомендацій у таблиці recommendations з’явилися нові записи про рекомендації, а саме: користувачам User2 і User1 рекомендувати товар з ідентифікатором 713196223, а користувачеві User3 – товар з ідентифікатором 711175233. На рис. 29–30 подані товари, які рекомендуватимуть користувачам. Аналізуючи оцінки користувача User3, можна помітити тенденцію до нижчих оцінок порівняно з користувачами User1 і User2. Тому логічно припустити: якщо користувач User3 поставить оцінку 3 певному товару, то в користувачів User1 та User2 цей товар може отримати оцінку 4 або навіть 5. Саме так з товаром tv. Виконаємо наступний запит для користувача User1 (рис. 29).

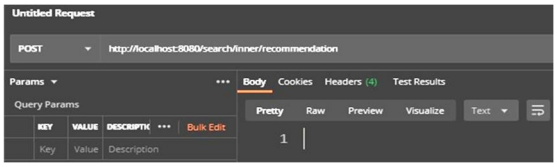


Рис. 27. Вхідна точка запуску рекомендацій

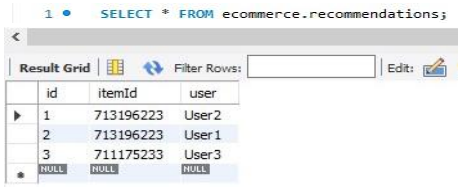
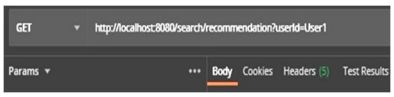


Рис. 28. Записи про рекомендації у баз даних

Рис. 29. Дані про товар до та після оцінок користувачів



User	ps 4	routers	High chair	Wireless mouse	spiderman	Iphone 5	PS2	blender	batman	tv
1	5	4	3	4	3	?	?	?	?	+
2	4	4	?	4	?	3	5	4	?	+
3	2	3	?	?	?	5	+	?	4	3

Рис. 30. Запит на рекомендації для користувача User1 та рекомендації для користувачів

Цей запит поверне нам топ-5 товарів за кількістю покупок серед всіх користувачів, товари, рекомендовані для вказаного користувача, а також деякі фільтри для зручності пошуку. На рис. 31–33 наведено результат виконання цього запиту. На рис. 31 також можна побачити, що найпопулярнішими товарами є playstation 4, а також wireless mouse. Це товари, які згідно із табл. 8 купили усі користувачі. На рис. 32 подано список доступних фільтрів, як можна використовувати, щоб швидше знаходити бажаний товар. А це збільшує можливість того, що користувач придбає певний товар на сайті.

Рис. 31. Топ-5 товарів серед користувачів

Рис. 32. Рекомендаційні товари для користувача User1

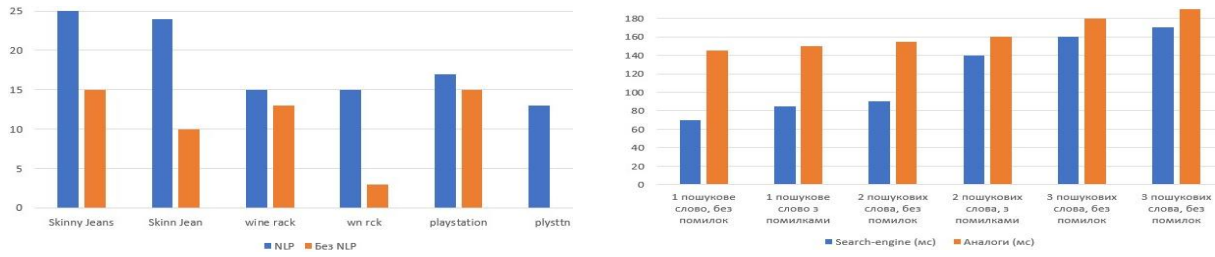
Рис. 33. Фільтри

Спробуємо використати один із фільтрів. Для цього виконаємо запит на рис. 34, використавши фільтр Shop зі значенням Walmart, тобто усі знайдені товари повинні бути з магазину Walmart. На рис. 35 видно, що товар з unit_id 711174392 виданий магазином Walmart. Наступний товар з unit_id 711175164 також із магазину Walmart. Це доводить справність роботи фільтрів. Важливо, що використання фільтрів не збільшує час пошуку товарів, а навпаки, зменшує його, адже багато товарів ми одразу відкидаємо, оскільки вони не задовольняють умови фільтрів.

Рис. 34. Запит з використанням фільтра

Рис. 35. Результат виконання запиту з фільтром

Здійснено також кілька пошукових запитів з NLP-алгоритмом і без. На рис. 36, *а* можна побачити кількість збігів у разі пошуку з NLP-алгоритмом, а також без нього.



а *б*

Рис. 36. Порівняння пошукових результатів з/без NLP (а) та швидкості виконання запитів з аналогами (б)

Здійснено також порівняння швидкості виконання запитів з уже наявними системами. Оскільки кількість даних у сховищі різниться, 60–70 мс може бути похибкою порівняння. З рис. 36, *б* видно, що запит, який складається з одного або двох слів, буде опрацьований значно швидше порівняно з аналогами. Але для трьох і більше слів результати доволі схожі.

Висновки

Розроблено технологію інтелектуального пошуку контенту для e-commerce з можливістю надання рекомендацій для користувачів. Реалізовано можливість пошуку серед товарів, надання рекомендацій товарів для користувачів на основі оцінок схожих користувачів, можливість авто-завершення пошукових слів, пошук із незавершеним словом. Для зручності пошуку додано можливість фільтрування пошукових результатів за певними критеріями. Також у програмі використано методи побудови графіків для відстеження товарів, яких продають найбільше, оцінки товарів від користувачів, регіонів, у яких певні товари найпопулярніші. Економічна доцільність створення цього програмного засобу полягає у тому, що він дає змогу малим і середнім бізнесам збільшити обсяги продажу товарів, зробивши їх доступними у мережі інтернет, а також полегшивши пошук серед всіх цих товарів і надаючи рекомендації, що, своєю чергою, збільшує кількість продажів. Розроблено модуль пошуку товарів у сфері e-commerce з можливістю надання рекомендацій для користувача. Для цього виконано низку досліджень. Визначено, як саме необхідно зберігати та аналізувати інформацію про товари. Враховано різні типи сховищ, їхні переваги та недоліки. Показано, як сховище може індексувати дані. Доведено, що для цієї системи необхідно

використовувати рекомендаційний алгоритм колаборативної фільтрації, адже вона враховує оцінки і рейтинги користувачів, а також не залежить від тематики сервісу. Система точніше надавала рекомендації порівняно з наявними сервісами. Порівняно основні алгоритми нечіткого пошуку і також вказано у вигляді таблиці їхню точність. Із урахуванням цих досліджень вибрано алгоритм Левенштейна, адже він порівнює слова, враховуючи кількість видалених, вставлених, замінених букв у слові. Створено дерево цілей, що відображає основні цілі й особливості, які необхідно врахувати під час розроблення цієї системи. Розроблено модуль інтелектуального пошуку з можливістю автозавершення пошукових слів, використанням фільтрів, для швидкого пошуку, сортуванням результатів, а також використанням NLP-алгоритмів. Побудовано пошукову систему практично для будь-якого типу бізнесу, популярного у користувачів, що містить багато інформації або товарів. Також наведено контрольний приклад, котрий підтверджує працездатність програми та основного функціоналу, розглянутого вище. Вищеописане доводить, що результати, досягнуті під час виконання роботи, відповідають поставленій меті та підтверджують розв'язання поставленої задачі. Під час експериментальної апробації розробленої системи здійснено низку пошукових запитів із NLP-алгоритмом і без, результати яких продемонстрували покращення роботи системи в межах 15–95 % залежно від ключового слова та наявності/відсутності помилок у словах пошуку. Також здійснено порівняння швидкості виконання запитів з уже наявними системами. Так, кількість даних у сховищі може відрізнятися (похибка під час порівняння 60–70 мс). Наприклад, відповідь на запит, який складається з одного або двох слів, буде надана значно швидше – на 20–70 мс порівняно з аналогами в межах. Але для трьох і більше слів результати приблизно подібні – на 9–20 мс швидше.

References

1. Eileen Pangu. The Search Backend (2020). URL: https://eileen-code4fun.medium.com/system-design-interview-mini-google-search-6fd319cd66ca.

2. Traditional Database (Forward Indexes) vs Search Engines (Inverted Index) (2020). URL: https://dev.to/search?q=%28Forward%20Indexes%29%20vs%20Search%20Engines. https://dev.to/im_bhatman/introduction-to-inverted-indexes-l04

3. Alex Franz, Thorsten Brants (2006). All Our N-gram are Belong to You, August.

4. David Guthrie (2017). A Closer Look at Skip-gram. NLP Research group, Department of Computer Science.

5. Levenshtein V. I. (1986). Binary codes with correction of dropouts, insertions and substitutions, 845–848.

6. Navarro Gonzalo (2006). A guided tour to approximate string matching. Department of Computer Science, University of Chile, 31–88. DOI: 10.1145/375360.375365

7. The Levenshtein Algorithm. Example (2021). URL: https://www.cuel.ogic.com/blog/the-levenshtein-algorithm.

8. Levenshtein distance (2011). URL: https://habr.com/ru/post/114997/.

9. Ghazanfar, Mustansar Ali; Prügel-Bennett, Adam; Szedmak, Sandor (2012). KernelMapping Recommender system algorithms. *School of Electronics and Computer Science*, 81–104. DOI: 10.1016/j.ins.2012.04.012

10. Ricci F., Rokach L., Shapira B. (2011). Introduction to Recommender Systems Handbook, Recommender Systems Handbook, Springer. Faculty of Computer Science, 32–35. DOI: 10.1007/978-0-387-85820-3_1

11. Linden, Gregory D., Brent Russell Smith, Nida K. Zada (2017). Automated detection and exposure of behavior-based relationships between browsable items, 40–42.

12. Terveen, Loren Hill (2011). Beyond Recommender Systems: Helping People Help Each Other items. University of Minnesota Twin Cities, February.

13. Chaitanya Belhekar. Collaborative Filtering Recommender System (2020). URL: https://medium.com/@chaitanyarb619/recommendationsystems-a-walk-trough-33587fecc195.

14. Introducing intelligence in relevance search (2020). URL: https://powerapps.microsoft.com/.

15. Robert P. Hanrahan. The IDEF Family of Methods (2020). URL: http://www.sba.oakland.edu/faculty/mathieson/mis524/resources/readings/idef/idef.html.

16. How search algorithms work. (2022). URL: https://www.google.com/intl/uk/search/howsearchworks/algorithms/.

17. DFD (2020). URL: https://habr.com/ru/company/trinion/blog/340064.

18. What is an Elasticsearch index? (2022). URL: https://www.elastic.co/what-is/elasticsearch.

19. 5 reasons to choose mysql. (2017). URL: https://dataconomy.com/2017/04/5-reasons-challenges-mysql/.
20. Java (2023). URL: https://techterms.com/definition/java.
21. Why would you choose Java programming language over others? (2008). URL: https:// stackoverflow.com/questions/209555/why-would-you-choose-the-javaprogramming-language-over-others.
22. A Collaborative Filtering Recommendation System in Java (2022). URL: https://www.baeldung.com/java-collaborative-filtering-recommendations.
23. Shi, Yue; Larson, Martha; Hanjalic, Alan (2014). Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys,* 47. DOI: 10.1145/2556270
24. Figueroa, Alejandro; Atkinson, John (2012). Contextual Language Models for Ranking Answers to Natural Language Definition Questions. Department of Computer Sciences, Universidad de Concepcion, Concepcion, Chile, November, 528–548. DOI: 10.1111/j.1467-8640.2012.00426.x
25. Features in the Java language 8 (2014). URL: https://javarush.com/groups/posts/1037-osobennosti-java-8--maksimaljhnoe-rukovodstvo-chastjh-1.
26. Liquibase allows you to perform the following (2022). URL: https://medium.com/search?q=Liquibase.
27. N Gram (2020). URL: https://stackoverflow.com/questions/18193253/what-exactly-is-an-n-gram.
28. Raza S., Ding C. (2022). News recommender system: a review of recent progress, challenges, and opportunities. *Artif. Intell. Rev.,* 55, 749–800. DOI: 10.1007/s10462-021-10043-x.
29. Zuo Y., Zeng J., Gong M., Jiao L. (2016). Tag-aware recommender systems based on deep neural networks. *Neurocomputing,* 204, 51–60. DOI: 10.1016/j.neucom.2015.10.134.
30. Lytvyn V., et. al. (2019). Design of a recommendation system based on Collaborative Filtering and machine learning considering personal needs of the user. *Eastern-European Journal of Enterprise Technologies,* 4(2), 6–28. DOI: 10.15587/1729-4061.2019.175507.
31. Balush I., Vysotska V., Albota S. (2021). Recommendation System Development Based on Intelligent Search, NLP and Machine Learning Methods. *CEUR Workshop Proceedings,* 584–617.
32. Demchuk A., Lytvyn V., Vysotska V., Dilai M. (2020). Methods and Means of Web Content Personalization for Commercial Information Products Distribution. *Lecture Notes in Computational Intelligence and Decision Making,* Vol. 1020. Springer, Cham. DOI: 10.1007/978-3-030-26474-1_24.
33. Tulashvili Y., Turbal Y., Abd Alkaleg D., Pasichnyk V., Kunanets N. (2020). The Optimal Tour Problem in Smart Tourism Recommender Systems. In *IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT),* Vol. 2, 246–250. DOI: 10.1109/CSIT49958.2020.9322043

INFORMATION TECHNOLOGY INTELLIGENT SEARCH OF CONTENT IN E-COMMERCE SYSTEMS

Illia Balush[1], Victoria Vysotska[1, 2], Maryna Shevchenko[3], Oksana Brodyak[4]

[1] Lviv Polytechnic National University, Information Systems and Networks Chair,
12, S. Bandery str., Lviv, Ukraine
[2] Osnabrück University, Institute of Computer Science, Friedrich-Janssen str., 1, Osnabrück, Germany
[3] Osnabrück University, International Economic Policy Chair, 8, Roland str., Osnabrück, Germany
[4] Lviv Polytechnic National University, Mathematics Department ,
12, S. Bandery str., Lviv, Ukraine
E-mail: illia.balush.kn.2017@lpnu.ua, ORCID: 0000-0002-0498-4719,
E-mail: Victoria.A.Vysotska@lpnu.ua, ORCID: 0000-0001-6417-3689,
E-mail: mshevchenko@uni-osnabrueck.de, ORCID: 0000-0003-2165-9907
E-mail: oksana.y.brodiak@lpnu.ua, ORCID: 0000-0002-9886-3589

© Balush I., Vysotska V., Shevchenko M., Brodyak O., 2023

The article describes the process of developing intelligent search technology for content for the implementation of the module of e-commerce systems for forming a list of recommendations for regular users. Intelligent search of content is based on methods of linguistic analysis, modern algorithms for parsing and finding words, and recommendations based on user preferences. The main components of

such a search are the parsing of text strings, the selection of keywords, the spelling check, the recognition of common abbreviations and acronyms, the semantic analysis of the text, the search by relevance with the extraction of synonyms, filters and sorting. A web application based on Java and Elasticsearch was developed with the implementation of a recommender system based on a collaborative filtering algorithm. The purpose of the work is to develop the technology of intelligent product search with the formation of a list of recommendations for the user. The object of the research is the processes of intelligent search with the possibility of generating recommendations for users in the field of any e-commerce without reference to the categorization of goods/services, etc. The subject of research is the methods and means of intelligent search of recommender systems based on the Collaborative Filtering algorithm for the formation of product recommendations for users, which is oriented on general coincidences of the choices of similar users. During the experimental testing of the developed system, a number of search queries were conducted with and without the NLP algorithm, the results of which demonstrated an improvement in system performance within the range of 15–95 % depending on the keyword and the presence/absence of errors in the search words. A comparison of the speed of execution of requests with already existing systems was also carried out. Yes, the amount of data in the storage may differ (error when comparing 60–70 ms). For example, a query that consists of 1 or 2 words will be found much faster by 20–70 ms compared to its counterparts. But for 3 and more, results are about the same – 9–20 ms faster.

Key words: search engine; elastic search; referral systems; collaborative filtering; web application; search; fuzzy search.