

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Система розумного пошуку для сайту»

Виконав студент групи ПЗм-22-2
ПОНТКІВСЬКИЙ Владислав Ігорович

Керівник роботи:
МОРОЗОВ Андрій Васильович

Рецензент:
ЄФРЕМОВ Юрій Миколайович

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ПІОНТКІВСЬКИЙ Владислав Ігорович**

Керівник роботи: **МОРОЗОВ Андрій Васильович**

Тема роботи: **«Система розумного пошуку для сайту»,**

затверджена Наказом закладу вищої освіти від **«18» вересня 2023 р. № 479/с**

Вихідні дані для роботи: **система розумного пошуку для сайту з використанням традиційних алгоритмів пошуку та нейронних мереж, аналіз методів пошуку.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Морозов А.В.	22.09.2023р.	22.09.2023р.
2	Морозов А.В.	22.09.2023р.	22.09.2023р.
3	Морозов А.В.	22.09.2023р.	22.09.2023р.

РЕФЕРАТ

Випускна кваліфікаційна магістерська робота включає аналіз та вибір інструментарію для розробки системи розумного пошуку для сайту.

Мета роботи полягає в розробці системи розумного пошуку для сайту з використанням та поєднанням існуючих методів та алгоритмів розумного пошуку та використанням нейронних мереж для покращення релевантності пошукових результатів.

Пояснювальна записка складається зі вступу, трьох розділів, переліку умовних позначень та списку використаної літератури. Записка дипломної роботи містить 51 ілюстрацію, 5 таблиць та 31 посилання на джерела бібліографії. Повний обсяг роботи становить 66 сторінок, з них 54 сторінки основного тексту.

В першому розділі визначено поняття інформаційного пошуку, існуючі алгоритми та методи для пошуку та вирішення пов'язаних з цим проблем. Було описано та проведено аналіз існуючих систем інформаційного пошуку. Також було поставлено мету дослідження.

В другому розділі описано вимоги до сервісу та його складових, а також вимоги та розробка алгоритмів для реалізації системи розумного пошуку з використанням традиційних методів та алгоритмів та методів з використанням машинного навчання та нейронних мереж.

В третьому розділі було описано та протестовано роботу системи розумного пошуку для сайту та показано розроблений графічний інтерфейс.

КЛЮЧОВІ СЛОВА: ІНФОРМАЦІЙНИЙ ПОШУК, TF-IDF, BM25, BERT, VECTOR DATABASE, GO.

ABSTRACT

The graduation qualification master's thesis includes analysis and selection of tools for developing smart search system.

The purpose of the work is to develop a smart search system using and combining existing methods and algorithms of smart search and using neural networks to improve search results relevance.

The explanatory note consists of an introduction, three chapters, a list of conditional abbreviations, and a list of used literature. The thesis note contains 51 illustrations, 5 tables and 31 reference to bibliographic sources. The total volume of the work consists of 66 pages, of them 54 pages of the main text

The first section defines the concept of information search, existing algorithms and methods for searching and solving related problems. The existing information search systems were described and analyzed. The purpose of the research was also set.

The second section describes the requirements for the service and its components, as well as the requirements and development of algorithms for the implementation of a smart search system using traditional methods and algorithms and methods using machine learning and neural networks.

In the third chapter, the operation of the intelligent search system for the site was described and tested, and the developed graphical interface was shown.

KEYWORDS: INFORMATION SEARCH, TF-IDF, BM25, BERT, VECTOR DATABASE, GO.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1. ФОРМУЛЮВАННЯ ЗАВДАННЯ ТА ВИВЧЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ	9
1.1. Аналіз проблем та особливостей розумного пошуку	9
1.2. Аналіз існуючих алгоритмів розумного пошуку.	12
1.3. Аналіз існуючих систем	22
1.4. Постановка задачі дослідження.....	23
Висновки до 1 розділу.....	24
РОЗДІЛ 2. РОЗРОБКА СИСТЕМИ РОЗУМНОГО ПОШУКУ	25
2.1. Вимоги до системи та її складових.....	25
2.2. Проектування системи розумного пошуку	27
2.3. Реалізація системи розумного пошуку	33
Висновки до 2 розділу.....	50
РОЗДІЛ 3. ОПИС РОЗРОБЛЕНОЇ СИСТЕМИ РОЗУМНОГО ПОШУКУ ДЛЯ САЙТУ	51
3.1. Тестування та представлення роботи сервісу	51
3.2. Графічний інтерфейс.....	59
Висновки до 3 розділу.....	62
ВИСНОВКИ	63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – application programming interface

ІІ – інформаційний пошук

UML – Unified Modeling Language

BERT – Bidirectional Encoder Representations from Transformers

NLP – Natural Language Processing

TF-IDF – Term Frequency - Inverse Document Frequency

BM25 – Best Match 25

CPU – Central processing unit

ВСТУП

Актуальність теми: В сучасних інформаційних системах кількість інформації збільшується і користувачі стикаються з великим об'ємом даних. Для покращення користувацького досвіду з отримання та пошуку інформації були створені системи розумного пошуку. Системи розумного пошуку допомагають зробити пошук більш ефективним, зокрема за допомогою аналізу контексту та інтелектуальної фільтрації результатів.

Системи розумного пошуку спрямовані на поліпшення користувацького досвіду шляхом повернення точних та релевантних результатів, що відповідають індивідуальним потребам користувача. З огляду на зростання використання мобільних пристроїв системи розумного пошуку важливі для оптимізації пошукового досвіду та різних платформах.

Процес створення систем розумного пошуку є активним та важливим напрямом досліджень. Один з напрямків дослідження є впровадження методів машинного навчання в процес пошуку. Розробка вищеприписаної системи вимагає комплексного підходу та поєднання різних аспектів інформаційних технологій для досягнення максимальної ефективності та точності.

Таким чином, робота спрямована на розробку системи, які будуть не тільки полегшувати процес пошуку релевантних результатів, але й сприяти підвищенню їх якості. Використання розробленого сервісу допоможе створити зручний та ефективний інструмент, яким може бути використаний в будь-якій інформаційній системі.

Мета роботи полягає в розробці системи розумного пошуку для сайту.

З метою досягнення поставленої задачі були сформульовані наступні **завдання:**

- аналіз існуючих методів та алгоритмів для реалізації пошуку в інформаційних системах;
- порівняльний аналіз існуючих систем;

- визначення вимог до сервісу та відображення результатів перевірки;
- визначення вимог та розробка алгоритмів для реалізації мікросервісів системи розумного пошуку для сайту;
- опис налаштування і тестування сервісу.

Об'єктом дослідження виступає система розумного пошуку для сайту з використанням традиційних методів пошуку та використанням нейронних мереж.

Предметом дослідження є традиційні алгоритми та методи для пошуку інформації в інформаційних системах та методи і алгоритми машинного навчання для вирішення задач з пошуку.

Наукова новизна даної роботи полягає в розробці технології, яка поєднує різні методи та алгоритми для пошуку інформації, зокрема машинне навчання.

Практична цінність полягає в наданні користувачам зручного та ефективного інструменту для пошуку в інформаційних системах.

РОЗДІЛ 1. ФОРМУЛЮВАННЯ ЗАВДАННЯ ТА ВИВЧЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз проблем та особливостей розумного пошуку

Інформаційний пошук (ІП) (англ. Information retrieval) — наука про пошук неструктурованої документальної інформації. Особливо це відноситься до пошуку інформації в документах, пошук самих документів, добуття метаданих з документів, пошуку тексту, зображень, відео та звуку у локальних реляційних базах даних, у гіпертекстових базах даних таких, як Інтернет та локальні інтранет. Інформаційний пошук — велика міждисциплінарна область науки, яка стоїть на перетині когнітивної психології, інформатики, інформаційного дизайну, лінгвістики, семіотики, бібліотечної справи, та статистики. Вперше виділив як міждисциплінарну галузь відомий угорський дослідник Золтон Жулен у 1989 році [1].

Автоматичні системи інформаційного пошуку використовують для зменшення так званого «інформаційного перевантаження». Багато університетів та публічних бібліотек використовують системи ІП для полегшення доступу до книжок, журналів та інших документів. Найвідомішим прикладом систем ІП можна назвати пошукові системи в Інтернеті.

Об'єктом інформаційного пошуку є текстова інформація, зображення, аудіо, відео інформація.

Завданням інформаційного пошуку є знаходження відповідних (до пошукового запиту) інформаційних об'єктів, або документів серед доступного для пошуку матеріалу.

Завдання для інформаційного пошуку задається у вигляді інформаційного запиту (query), який може містити слова, фрази чи речення або комбінацію їх. Переважна більшість пошукових систем орієнтована на роботу з пошуковими

термінами — словами або словосполученнями, які пошукова система розпізнає як одне ціле [2].

Пошукова система переглядає всі доступні інформаційні одиниці (документи) зі збірки і відбирає документи відповідні до інформаційного запиту. Оскільки реальні пошукові системи знаходять не всі відповідні документи, говорять про точність пошукових систем (system accuracy). Результатом роботи пошукової системи є список відібраних документів (retrieved documents list), серед яких є відповідні до запиту документи (relevant documents). Для ідеальної пошукової системи список відібраних документів та відповідних документів повинні збігатися. В реальних пошукових системах в списках відібраних документів знаходяться і невідповідні до запиту документи. Тому говорять про ефективність пошукових систем.

Ефективність пошукових систем оцінюється двома параметрами: пошукова відповідність (precision) та пошукова якість (recall).

Пошукова відповідність визначає частку відповідних документів серед відібраних на запит. Пошукова відповідність визначає якість отриманого результату інформаційного пошуку. Пошукова якість визначає частку отриманих системою відповідних до запиту документів серед загального числа відповідних до запиту документів у збірці. Загальне число відповідних до запиту документів завжди є невідомим і може бути встановлене лише при повному перегляді збірки людиною.

Крім того роботу пошукових систем оцінюють швидкістю — часом, за який отримують список відповідних до запиту документів.

Розумний пошук представляє революційний підхід до пошуку інформації та технології пошуку. На відміну від традиційних пошукових систем на основі ключових слів, які в першу чергу покладаються на точну відповідність ключових слів, інтелектуальні пошукові системи розроблені для розуміння та інтерпретації запиту користувача більш схожим на людину та контекстним способом.

Основні характеристики розумного пошуку включають:

1. Обробка природної мови (NLP): Розумні пошукові системи використовують методи NLP, щоб зрозуміти семантику та наміри запиту користувача. Це означає, що вони можуть розпізнавати синоніми, мовні нюанси та навіть почуття. Наприклад, якщо користувач шукає «компактний автомобіль», розумна пошукова система може розпізнати, що «маленький автомобіль» або «міні-автомобіль» концептуально пов'язані, і включити їх у результати. [3]
2. Семантичне розуміння: Розумний пошук виходить за межі зіставлення ключових слів, розуміючи значення слів у контексті. Це дозволяє механізму визначати зв'язки між словами та надавати більш точні результати. Наприклад, у відповідь на запит про «канікули в Каліфорнії» розумний пошук може розглядати різні пов'язані поняття, як-от «пляжні напрямки», «туристичні пам'ятки» та «проживання».
3. Контекстуальна обізнаність: Розумні пошукові системи враховують контекст користувача, його місцезнаходження та попередню історію пошуку. Якщо користувач часто шукає теми, пов'язані з технологіями, система може віддавати пріоритет результатам, пов'язаним з технологіями, навіть якщо поточний запит явно не пов'язаний з технологіями.
4. Машинне навчання: Багато розумних пошукових систем включають алгоритми машинного навчання, щоб постійно покращувати результати пошуку. Ці алгоритми навчаються на основі взаємодії та відгуків користувачів, адаптуючись і стаючи точнішими з часом.
5. Персоналізація: Розумні пошукові системи часто пропонують персоналізовані результати. Вони враховують уподобання та поведінку кожного користувача, щоб адаптувати результати пошуку. Наприклад, під час пошуку «найкращі ресторани» результати можуть відрізнитися

залежно від дієтичних уподобань користувача або попередніх відгуків про ресторани.

Основна мета розумного пошуку – надати користувачам високорелевантні, контекстуально точні та персоналізовані результати, зменшуючи ймовірність нерелевантних або хибнопозитивних результатів. Ця технологія знайшла застосування в різних областях, зокрема в електронній комерції, охороні здоров'я, управлінні вмістом і системах рекомендацій, докорінно змінивши спосіб пошуку та відкриття інформації.

1.2. Аналіз існуючих алгоритмів розумного пошуку.

У сучасному світі, коли обсяг інформації в Інтернеті стрімко зростає, важливість систем розумного пошуку для веб-сайтів стає неоспоримою. Користувачі очікують від пошукових систем не лише швидкості та доступності, але й високої точності та релевантності результатів. Це вимагає вдосконалених алгоритмів, які здатні не лише ідентифікувати ключові слова, але й розуміти контекст та індивідуальні потреби користувачів.

У даному контексті, важливо провести аналіз існуючих алгоритмів розумного пошуку, оскільки це дозволяє визначити та розуміти найефективніші методи оптимізації пошукових результатів. Від традиційних підходів, таких як TF-IDF та BM25, до сучасних розробок, таких як BERT та Semantic Search, кожен алгоритм пропонує свій унікальний підхід до вирішення завдань розумного пошуку.

Буде розглянуто основні аспекти та переваги існуючих алгоритмів, визначаючи їхню актуальність та вплив на сучасний веб-пошук. Розгляд цих алгоритмів в контексті їхнього застосування допоможе нам краще зрозуміти еволюцію пошукових систем та тенденції їхнього розвитку в майбутньому.

Найпоширенішими в існуючих системах є традиційні підходи, такі як TF-IDF та BM25, за допомогою яких можна проводити пошук по тексту.

TF-IDF (Term Frequency-Inverse Document Frequency): TF-IDF вимірює важливість терміна в документі відносно набору документів (корпусу). Він призначає оцінку на основі частоти терміна в документі та його рідкості в корпусі. Високі оцінки надаються термінам, які часто зустрічаються в документі, але рідко в корпусі. Це підкреслює терміни, які відрізняються від документа. TF-IDF часто використовується для подання колекції документів у вигляді чисельних векторів, які відображають важливість використання кожного слова з певного набору слів (кількість слів набору для пошуку розмірності вектора) у кожному документі. Подібна модель називається векторною моделлю і дає можливість порівнювати тексти, порівнюючи їх представлений вектор у певній метриці (евклідова відстань, косинусна міра, манхеттенська відстань, відстань Чебішова та інші), означає кластерний аналіз. [4]

Як можна побачити на рисунку 1.1 TF-IDF це частота слова (Term frequency) помножена на обернену частоту документа.

$$\text{TF-IDF} = \text{TF} \cdot \text{IDF}$$

Рис 1.1 Формула TF-IDF

На рисунку 1.2 можна побачити формулу для розрахунку частоти слова. TF (term frequency — частота слова) — відношення числа входжень обраного слова до загальної кількості слів документа. Таким чином, оцінюється важливість слова в межах обраного документа. Термін був введений Карен Спарк Джонс. [5]

$$\text{TF} = \frac{n_i}{\sum_k n_k}$$

Рис 1.2 Формула Term frequency

IDF (inverse document frequency — обернена частота документа) — інверсія частоти, з якою слово зустрічається в документах колекції. Використання IDF

зменшує вагу широкоживаних слів. Формула розрахунку оберненої частоти документа показана на рисунку 1.3.

$$\text{IDF} = \log \frac{|D|}{|(d_i \supset t_i)|}$$

Рис 1.3 Формула Inverse document frequency

Загальна кількість документів ділиться на кількість документів, в яких зустрічається це слово. З результату береться логарифм. Вибір основи логарифма у формулі не має значення, адже зміна основи призведе до зміни ваги кожного слова на постійний множник, тобто вагове співвідношення залишиться незмінним. Як результат більшу вагу TF-IDF отримують слова з високою частотою появи в межах документа та низькою частотою вживання в інших документах колекції.

Для покращення пошуку на основі TF-IDF було розроблено новий алгоритм, який покращував результати пошуку і вирішував проблему з налаштування пошуку та впливом довжини документа на релевантність результатів.

Окарі BM25 (BM — аббревіатура від best matching) — це функція ранжирування, яка використовується пошуковими системами для оцінки релевантності документів певному пошуковому запиту. Він заснований на системі ймовірнісного пошуку, розроблений у 1970-х і 1980-х роках Стівеном Е. Робертсоном, Карен Спарк Джонс та іншими. [6]

Назва фактичної функції ранжирування – BM25. Повна назва, Окарі BM25, включає назву першої системи, яка її використовувала, яка була інформаційно-пошуковою системою Окарі, реалізованою в Лондонському міському університеті у 1980-х і 1990-х роках. BM25 та його новіші варіанти, напр. BM25F (версія BM25, яка може враховувати структуру документа та прив’язаний текст)

представляє TF-IDF-подібні функції пошуку, які використовуються для пошуку документів.

BM25 — це функція пошуку з набору слів, яка ранжує набір документів на основі термінів запиту, що з'являються в кожному документі, незалежно від їх близькості в документі. Це сімейство функцій із дещо різними компонентами та параметрами. На рисунку 1.4 наведено один із найвідоміших екземплярів функції.

$$\text{score}(D, Q) = \sum_{i=1}^n \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgl}}\right)}$$

Рис 1.4 Формула BM25 score

Формула вираховує оцінку (score) документу для даного запиту враховуючи довжину документу, середню довжину документу в корпусі та параметри з TF-IDF.

TF-IDF та BM25 використовують корпус документів та слів в документах для розрахунку оцінки. Поширеною проблемою для обох алгоритмів є часто вживані слова. Як приклад можна навести “a”, “the” та тому подібні. Для вирішення даної проблеми при розрахунку релевантності такі слова помічаються як стоп-слова та виключаються з розрахунку.

При підрахунку слів виникає проблема з визначенням чи слово вже було вжите в документі. Для вирішення цієї проблеми слова нормалізуються перед аналізом. Процес нормалізації називається стеммінгом.

Стемінг - це процес обрізання афіксів (префіксів, суфіксів) у словах з отриманням їх основної (базової) форми, яка називається "стемінг". Стемінг використання в обробці природної мови і є одним із методів текстової нормалізації. Мета стеммінгу - зведення слова до форми, яка відображає його базове значення або корінь. [7]

Процес стеммінгу може бути дещо грубим, і він може привести до того, що отримані стеми можуть бути не завжди лексично правильними словами. Проте це може бути корисним в аналізі тексту, особливо в завданнях, де важливо редукція слова до його основного значення без урахування морфологічних варіацій.

Існує декілька алгоритмів стеммінгу, кожен з яких має свої особливості та правила для отримання результату.

Алгоритм стеммінгу Портера є одним з найбільш широко використовуваних за рахунок своєї простоти та зрозумілості. Він є ефективним для англійської мови, але може бути занадто агресивним, призводячи до обрізання слів занадто сильно. [9]

Алгоритм Ловінга є більш швидким та менш агресивним алгоритмом в порівнянні з алгоритмом Портера. Він є ефективним для простих задач. До його недоліків відносять меншу універсальність в порівнянні з іншими алгоритмами, він не завжди враховує морфологічні правила за рахунок чого може занадто сильно обрізати слова. [8]

Snowball stemming це оновлена та вдосконалена версія алгоритму Портера. Може використовуватись як єдиним стемер для декілької мов, є більш точним для мов зі складною морфологією, побудований з використанням модульної структури. Оголошений як розширення алгоритму Портера, Snowball став більш універсальним та модульним інструментом для створення слів у різних мовах та для різних завдань.

Підсумовуючи вище сказане можна сказати, що при виборі алгоритму стеммінгу потрібно враховувати морфологічні властивості мови, швидкість роботи алгоритму для отримання найкращих результатів.

Використовуючи алгоритми стеммінгу для приведення слів до нормальної форми одне і те ж слово в різних формах буде враховано коректно при розрахунку TF-IDF та BM25.

TF-IDF та BM25 розраховані на пошук по слову і за рахунок цього контекст запиту не зберігається. Наприклад якщо шукати “star wars” буде здійснено пошук окремо для слів star та wars. Для вирішення даної проблеми використовуються N-грами.

N-грами представляють собою послідовність з N слів, яка повторюється при розбитті тексту на фрагменті фіксованої довжини (N). Їх використання може допомогти у врахуванні локальних контекстів та уточненні семантичного вмісту тексту.

Загальною метою використання N-gram є вдосконалення моделі за рахунок врахування локального семантичного контексту та вивчення синтаксичних зв’язків між словами.

При розрахунку TF-IDF та BM25 також варто приймати до уваги той факт, що запит може бути написаний з помилками. Процес стемінгу може допомогти лише в деяких випадках, тому використовуються інші рішення.

Найбільш поширеними є алгоритми на основі відстані редагування – Алгоритм Левенштейна [10]. В його основі лежить визначення відстані редагування між двома рядками та надає інформацію про кількість деяких операцій для перетворення одного рядка в інший. Маючи словник всіх слів можна побудувати BK-Tree з використання відстані Левенштейна для швидкого пошуку слів, які могли бути надруковані неправильно.

На рисунку 1.5. можна побачити приклад побудованого BK-Tree в якому кожна нода це слово і з’єднання між ними це відстань Левенштейна.

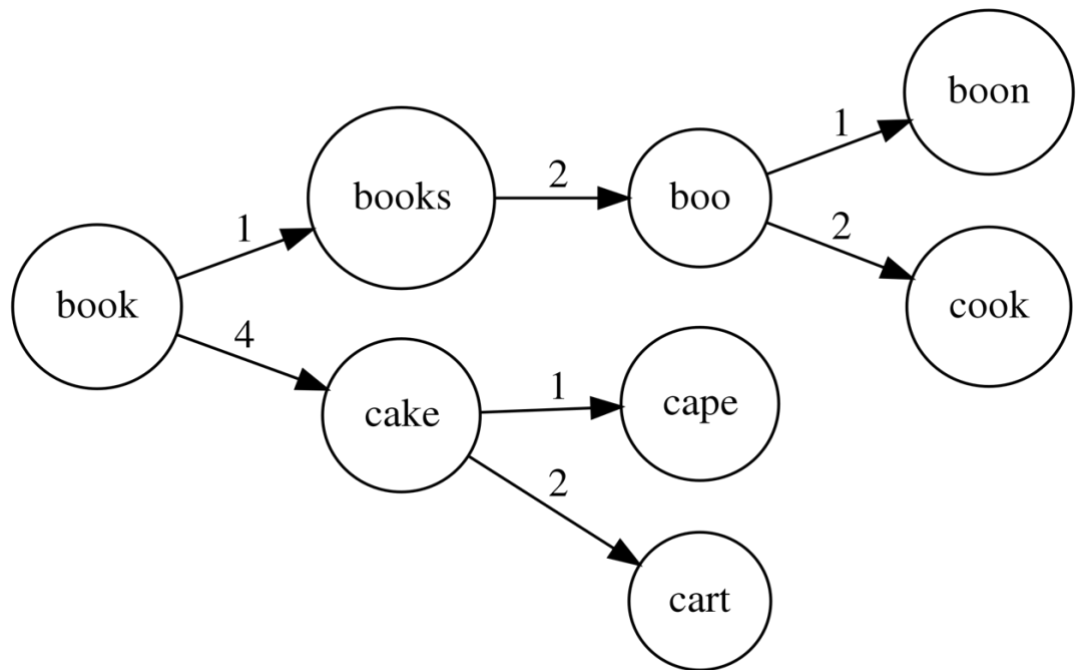


Рис 1.5 Приклад BK-Tree

Також для виправлення помилок використовується автомат Левенштейна (Levenshtein automaton). Автомат Левенштейна для рядка w і числа $n \in \mathbb{N}$ є автоматом зі скінченним станом, який може розпізнати набір усіх рядків, відстань Левенштейна від w яких не перевищує n . Тобто рядок $x \in \Sigma^*$ у формальній мові, розпізнаний автоматом Левенштейна, тоді і тільки тоді, коли x можна перетворити на w за допомогою щонайбільше n односимвольних вставок, видалень і замінів.

Автомати Левенштейна можна використовувати для виправлення орфографії шляхом пошуку слів у заданому словнику, близьких до слова з помилкою. У цьому додатку, як тільки слово ідентифікується як таке, що має орфографічну помилку, його автомат Левенштейна може бути побудований, а потім застосований до всіх слів у словнику, щоб визначити, які з них близькі до слова з помилкою. Якщо словник зберігається в стиснутому вигляді як триєд, час для цього алгоритму (після створення автомата) пропорційний кількості вузлів у триєді, значно швидше, ніж використання динамічного програмування для обчислення відстані Левенштейна окремо для кожного словникового слова. [11]

Також можна знайти слова в звичайній мові, а не в скінченному словнику, які близькі до даного цільового слова, обчислюючи автомат Левенштейна для слова, а потім використовуючи конструкцію декартового добутку, щоб поєднати його з автоматом для звичайна мова, надаючи автомат для мови перетину. В якості альтернативи, замість використання конструкції продукту, як автомат Левенштейна, так і автомат для заданої регулярної мови можна обійти одночасно за допомогою алгоритму зворотного відстеження (backtracking).

Автомати Левенштейна використовуються в Lucene для повнотекстового пошуку, який може повертати релевантні документи, навіть якщо запит написаний з помилкою. [12]

TF-IDF та BM25 можна використати як основу для створення векторної моделі. Векторна модель (англ. vector space model) — в інформаційному пошуку алгебраїчне представлення колекції документів векторами одного спільного для всієї колекції простору вікна.

Векторна модель є основою для вирішення багатьох завдань інформаційного пошуку, таких як: пошук документа за запитом, класифікація документів, кластеризація документів.

Маючи векторну модель при пошуку створюється вектор запиту і проводиться пошук найближчого вектору. Найбільш поширеними методами є косинус подібності та евклідова відстань.

Косинус подібність (англ. cosine similarity) — коефіцієнт подібності двох ненульових векторів у передгільбертовому просторі, який обчислюється як косинус кута між ними [13]. Для двох заданих векторів позначок, A та B , косинус подібності, $\cos(\theta)$, представлений за допомогою скалярного добутку та довжини, як показано на рисунку 1.6, де A_i та B_i — координати векторів A та B відповідно.

$$\text{подібність} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

Рис 1.6 Формула Cosine similarity

Евклідова відстань (Евклідова метрика) — формула традиційної відстані між двома точками для Евклідового простору. На рисунку 1.7 можна побачити формулу для розрахунку Евклідової відстані між точками p та q .

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

Рис 1.7 Формула для розрахунку Евклідової відстані

При створенні векторної моделі на основі TF-IDF та BM25 вектори, які будуть згенеровані матимуть велику розмірність за рахунок того, що вектор матиме в собі інформацію про кожне слово. В такому векторі буде багато нулів і вони називаються розрідженими.

Для вирішення проблеми розрідженості векторів створених на основі традиційних алгоритмів використовуються Векторні моделі представлення слів (Word Embeddings) і методи глибокого навчання. Прикладом векторних моделей представлення слів є трансформаторні моделі, як-от BERT / RoBERTa / DistilBERT тощо.

Такі моделі використовують нейронні мережі для приведення тесту до векторної форми з меншою розмірністю враховуючи контекст. Такий вид пошуку називається Семантичним пошуком (Semantic search).

Семантичний пошук — це технологія пошукової системи, яка інтерпретує значення слів і фраз. Результати семантичного пошуку повернуть вміст, який відповідає значенню запиту, на відміну від вмісту, який буквально відповідає словам у запиті. [14]

Семантичний пошук — це набір можливостей пошукової системи, який включає розуміння слів з наміру шукача та їх контексту пошуку.

Цей тип пошуку призначений для покращення якості результатів пошуку шляхом більш точної інтерпретації природної мови в контексті. Семантичний пошук досягає цього шляхом зіставлення мети пошуку із семантичним значенням за допомогою таких технологій, як машинне навчання та штучний інтелект.

Семантичний пошук базується на векторному пошуку, який дозволяє семантичному пошуку надавати та ранжувати вміст на основі релевантності контексту та релевантності наміру. Векторний пошук кодує деталі інформації, доступної для пошуку, у поля пов'язаних термінів або елементів або векторів, а потім порівнює вектори, щоб визначити, які з них найбільш схожі.

Семантичний пошук із підтримкою векторного пошуку дає результати, працюючи на обох кінцях конвеєра запиту одночасно: коли запит запускається, пошукова система перетворює запит на вбудовування, яке є числовим представленням даних і пов'язаних контекстів. Вони зберігаються у векторах. Алгоритм kNN, або алгоритм k-найближчого сусіда, потім зіставляє вектори існуючих документів (семантичний пошук стосується тексту) з векторами запиту. Потім семантичний пошук генерує результати та ранжує їх на основі концептуальної релевантності.

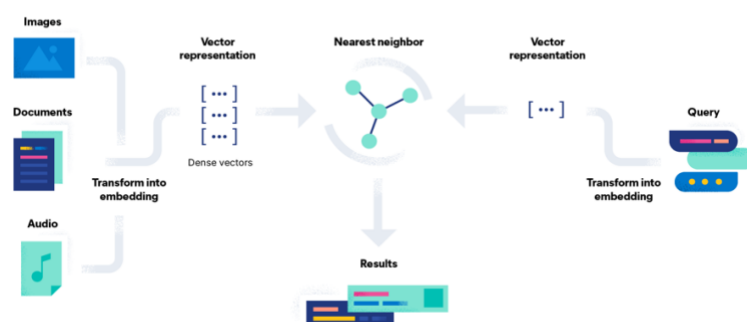


Рис 1.8 Приклад векторного пошуку

На рисунку 1.8 можна побачити приклад реалізації векторного пошуку з використанням моделей на основі нейронних мереж для створення векторного представлення для різних типів інформації. [15]

1.3. Аналіз існуючих систем

Пошук інформації є поширеною та дуже актуальною задачею, тому існує велика кількість систем, які допомагають користувачам з виконанням такого завдання. Кожна система має свої недоліки та переваги, проаналізувавши які буде можливість краще визначити аспекти, які потребують доопрацювання чи покращення.

Одним з найбільш популярних та вживаних сервісів для пошуку є Elasticsearch.

Elasticsearch — вільне програмне забезпечення, пошуковий сервер, розроблений на базі Lucene. Надає розподілений, багатоклієнтський повнотекстовий пошуковий рушій. Elasticsearch може використовуватись для індексування та пошуку будь-яких типів документів. Він надає масштабовний пошук, має пошук близький до реального часу і підтримку мультиарендності. Elasticsearch має можливість розподілення, індекси можуть бути розділені на сегменти, при чому кожен сегмент може мати нуль чи більше реплік. Кожен вузол містить один чи більше сегментів і діє як координатор делегування операцій на потрібний сегмент. Балансування та маршрутизація виконується автоматично. Одним з основних недоліків Elasticsearch є потреба в значних ресурсах (CPU, пам'ять, диск), особливо при обробці великих обсягів даних. Також Elasticsearch може займати місце значного обсягу на диску через індексацію та зберігання даних. [16]

Ще одним широко вживаним програмним забезпеченням для пошуку є Algolia.

Algolia — це хмарна платформа пошуку та виявлення, яка надає розробникам і компаніям інструменти для створення швидкого, релевантного та

настроюваного пошуку на своїх веб-сайтах і мобільних додатках. Algolia відома своєю швидкістю, масштабованістю та простотою інтеграції, що робить її популярним вибором для організацій, які прагнуть покращити свої функції пошуку. Algolia дозволяє проводити створення індексів та шукати по них в режимі реального часу, рангування результатів пошуку з використанням різних факторів, фільтрування результатів пошуку, використання різних мов, аналітику з використання системи. Основним недоліком даної системи є неможливість контролю над інфраструктурою, висока ціна за використання та неможливість впливу на процес індексації та внесення будь яких змін при потребі. [17]

1.4. Постановка задачі дослідження

Проаналізувавши існуючі методи та алгоритми пошуку інформації можна виявити основні аспекти які мають бути виправлені та покращенні в майбутній системі. Основним недоліком існуючих систем є недоступність пошуку з урахуванням контексту чи необхідність його додаткового налаштування.

Також важливими фактором для покращення користувацького досвіду, зокрема в інтеграції до існуючих систем, можливість імпортування та експортування інформації в системі, простота інтерфейсів для взаємодії з системою.

Важливою характеристикою системи є її швидкодія та раціональне використання наданих ресурсів. Майбутня система повинна бути оптимізованою, при цьому маючи потрібний для пошуку функціонал.

Також основним недоліком є вузька направленість та обмеженість певних програмних додатків, що відображається на універсальності та охопленні ринку у різних сферах.

Аналізуючи переваги та недоліки, можемо сформулювати перше уявлення про те, як має функціонувати система. Він має бути простим в користуванні та налаштуванні, бути розширюваним, мати простий та зручний програмний

інтерфейс та мати функціонал для пошуку використовуючи як традиційні методи, так і з використанням нейронних мереж для більшої релевантності та розуміння контексту.

Варто розділити створення системи на етапи, для спрощення проектування та подальшої реалізації.

На першому етапі варто спроектувати та розробити систему для пошуку з використанням традиційних методів пошуку, враховуючи необхідність розширення її в подальшому для підтримки пошуку з використанням нейронних мереж.

На другому етапі варто розробити підсистему для пошуку використовуючи нейронні мережі та векторний пошук і інтегрувати її в розроблену на першому етапі систему для розширення доступних можливостей пошуку.

На третьому етапі потрібно розробити інтерфейс та інтегрувати систему пошуку до нього.

Висновки до 1 розділу

В першому розділі було визначено поняття розумного пошуку, існуючі алгоритми для реалізації пошуку в інформаційних системах, різниця між алгоритмами та їх переваги і недоліки. Також було проведено аналіз методів вирішення поширених проблем при пошуку, таких як виправлення помилок при пошуці та приведення слів до базової форми для покращення індексації документів та подальшого пошуку по них.

Також було проаналізовано існуючі системи, визначено їх слабкі та сильні сторони та поставлена задача дослідження.

На основі проведених досліджень було підкреслено переваги та недоліки існуючих платформ, що допоможе нам при реалізації застосунку.

РОЗДІЛ 2. РОЗРОБКА СИСТЕМИ РОЗУМНОГО ПОШУКУ

2.1. Вимоги до системи та її складових

Вимоги до програмного забезпечення представляють собою набір критеріїв, які стосуються властивостей, якості та функціональності майбутнього або розробленого програмного продукту. Ці вимоги формуються під час аналізу та зафіксовані в специфікації вимог, діаграмах прецедентів та інших документах аналізу та розробки вимог. [18]

Функціональні програмні вимоги (функціональність) виконуються, завдяки чому необхідно функціонувати, щоб задовольнити бізнес-потреби користувачів. Іноді їх також вимогами поведінки. Вони включають в себе конкретні вимоги, які можна сформулювати через "повідомлення", наприклад, "Система повинна автоматично відправляти користувачеві підтвердження замовлення". Функціональні вимоги поради, що саме розробник має реалізувати. [19]

Додаткові функціональні вимоги, необхідні нефункціональні вимоги, які потрібні умови виконання функцій, таких як безпека інформації в базах даних, аутентифікація доступу до системи та інші. Ці вимоги не стосуються інших функцій, але виконуються умови, які повинні бути виконані в процесі їх реалізації. Вони описують принципи взаємодії з середовищем та іншими системами, а також результати показників часу роботи, захисту даних і якості з дотриманням вимог стандартів. [20]

Після вивчення термінології можемо перейти до визначення вимог до системи розумного пошуку для сайту. Система повинна працювати ефективно та швидко, без затримок та помилок.

Головна мета системи - забезпечити швидкий пошук за ключовими словами в тестовому контенті. Надзвичайно важливо, щоб система надавала результат за максимально короткий проміжок часу.

Багато користувачів можуть робити орфографічні або типографічні помилки при введенні свого запиту. Важливо, щоб система підтримувала можливість пошуку з можливістю автоматичної корекції для виправлення помилки і надання користувачам очікуваного результату.

Для того, щоб система надавала результати потрібно, щоб система мала можливість додавати інформацію в систему. Система має мати інтерфейс для завантаження даних в систему для подальшого пошуку по ній.

Один з важливих аспектів системи розумного пошуку є пошук з врахуванням контексту, зокрема врахування контекстної інформації для покращення розуміння запитань та надання більш точних результатів.

Система повинна мати можливість конфігурувати параметри пошуку та індексації. Це дасть можливість користувачав налаштувати систему під свої особисті потреби та покращити результати її роботи з урахуванням їх доменної області та контексту.

Також варто розглянути нефункціональні вимоги до системи, яка буде розроблена. Нефункціональні вимоги до системи розумного пошуку містять аспекти, які не стосуються конкретної функціональності системи, але забезпечують якість та умови її роботи. Ці вимоги допомагають налаштувати параметри ефективності, безпеки, надійності та інших аспектів.

Система повинна видавати швидкі та ефективні результати пошуку, щоб користувачі не чекали надто довго на відповідь.

Система має бути стійкою та надійною, забезпечуючи безперебійну роботу та уникнення відмов.

Також система має працювати стабільно при різних умовах та навантаженнях з можливістю автоматичного відновлення роботи після виникнення відмов чи збоїв.

Всі вказані вище нефункціональні вимоги грають важливу роль у забезпеченні високоякісного та ефективного функціонування системи розумного пошуку.

З врахування всіх вказаних вище вимог можна побудувати діаграму, яка буде показувати процес взаємодії з системою з точки зору кінцевого користувача та адміністратора. На рисунку 2.1 можна побачити, що адміністратор відповідальний за налаштування системи та додавання даних для пошуку в неї, а користувач взаємодії з системою лише при пошуку потрібної йому інформації.

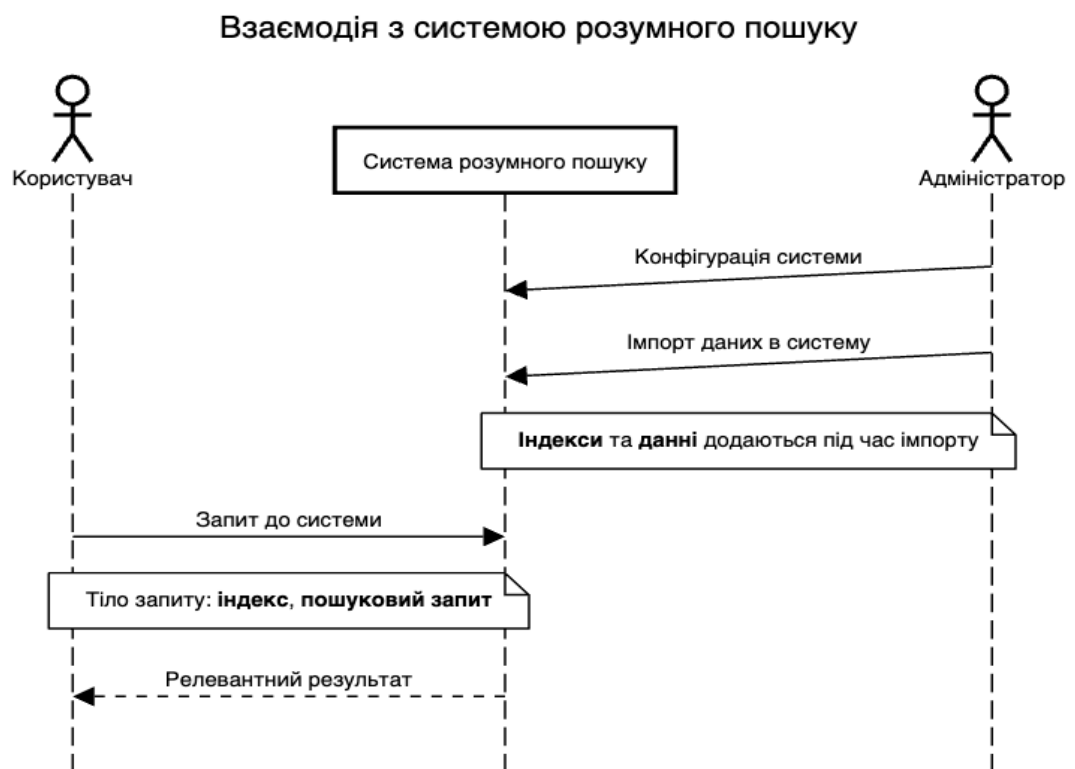


Рис 2.1 Діаграма взаємодії з системою розумного пошуку

2.2. Проектування системи розумного пошуку

Для задоволення поставлених вище вимог до системи потрібно розробити систему для розумного пошуку.

Система буде складатись з двох мікросервісів, які будуть пов'язані між собою та реалізовувати потрібний для коректного пошуку функціонал. Один з мікросервісів буде вхідною точкою в систему та буде реалізовувати традиційні

алгоритми пошуку з врахуванням можливості пошуку з помилками і виступатиме як проксі до другого мікросервісу. Другий мікросервіс буде відповідальним за розумний пошук, зокрема семантичний пошук з використанням нейромереж.

На рисунку 2.2 можна побачити приклад архітектури системи, яка описує основні частини системи та зв'язки між ними.

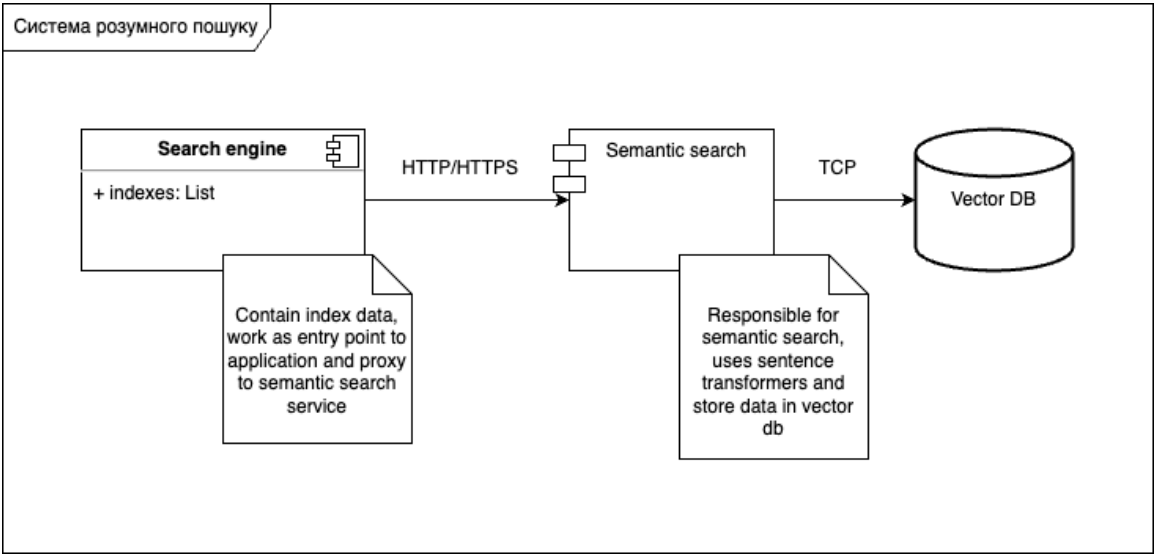
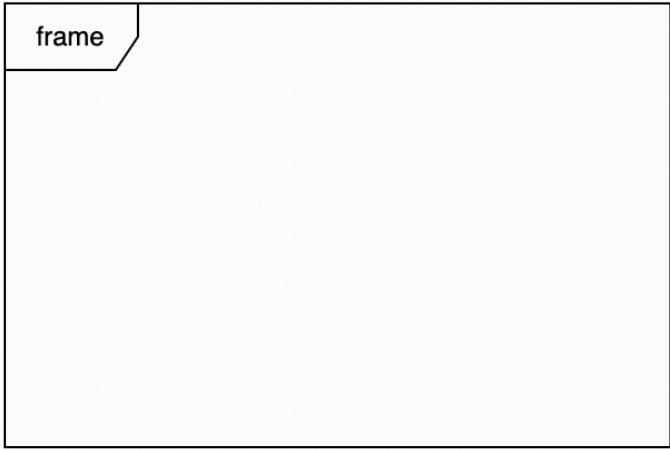
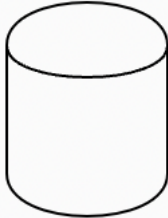
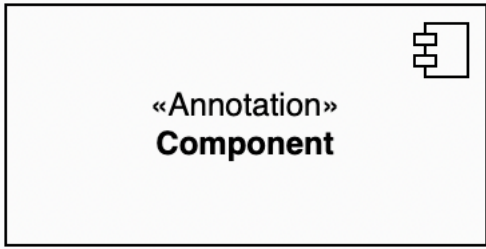


Рис 2.2 Діаграма архітектури системи розумного пошуку

В таблиці 2.1 можна побачити опис використаних для створення діаграми елементів.

Таблиця 2.1

Елемент	Позначення	Коментар
Компонент з атрибутами	Component + Attribute1: Type + Attribute2: Type	Компонент який має певні важливі атрибути

Рамка		Рамка в якій знаходиться весь контент діаграми
База даних		Позначає базу даних
Коментар		Коментар з додатковими поясненнями
Повідомлення, зв'язок між компонентами		Зв'язок між компонентами
Компонент		Компонент без атрибутів

Входною точкою в системі буде мікросервіс, який названий в діаграмі як Search Engine. Він має працювати як вхідна точка в систему маючи API для взаємодії з системою. Також цей мікросервіс матиме в собі функціонал для пошуку по тексту традиційними методами (TF-IDF, BM25) і зберігатиме ці дані

локально. Також цей сервіс виступатиме як проксі для мікросервісу позначеного в діаграмі як Semantic search.

На основі показаної вище архітектури створимо Data Flow діаграму. Данний вид діаграм описує процес поширення інформації по системі. На рисунку 2.3 можна побачити Data Flow Diagram [21] для системи розумного пошуку.

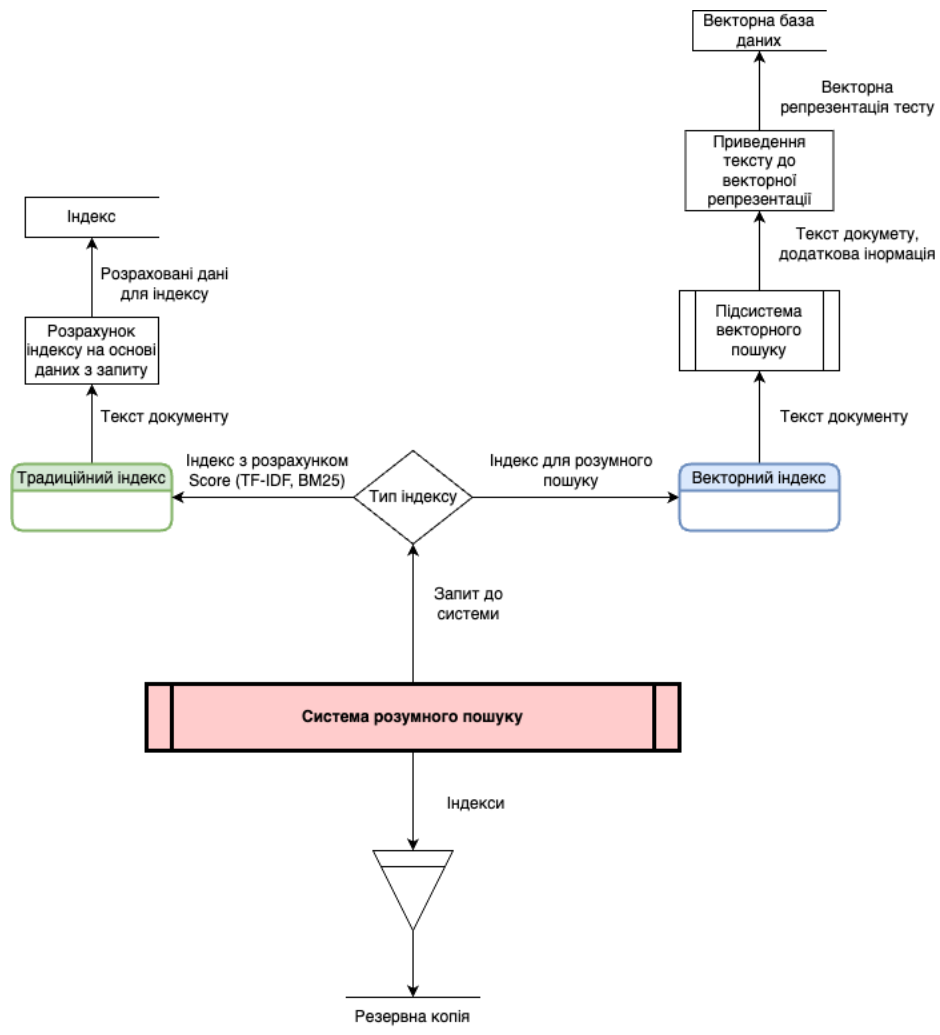
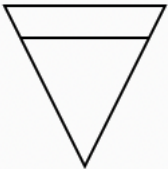
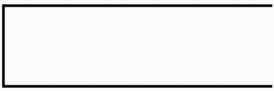
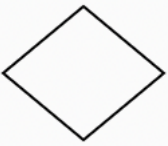
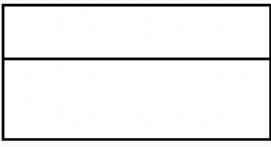
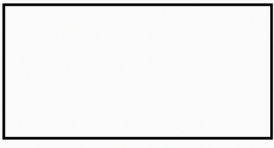


Рис 2.3 Data Flow Diagram для системи розумного пошуку

В таблиці 2.2 описано використані для Data Flow діаграми елементи з коротких описом їх призначення [22].

Таблиця 2.2

Елемент	Позначення	Коментар
Процес		Основний процес
Фінальний репорт, архів		Репорт чи архів.
Джерело даних		База даних чи будь яке інше джерело даних
Рішення		Показує вибір між
Сутність		Підсистема
Функція, сутність		Функція
Зв'язок між компонентами		Зв'язок між документами яких показує дані якими обмінюються компоненти

Як можна побачити на діаграмі показаній вище система розумного пошуку матиме два види індексів. Один з індексів буде відповідати за традиційні методи пошуку і триматиме інформацію в пам'яті додатку. Другий буде працювати як проксі до окремого мікросервісу і відповідатиме за розумний пошук. Дані для розумного пошуку зберігаються в векторній базі даних для швидшого пошуку.

На рисунку 2.4 можна побачити sequence діаграму для роботи системи розумного пошуку при запиті на пошук від кінцевого користувача.

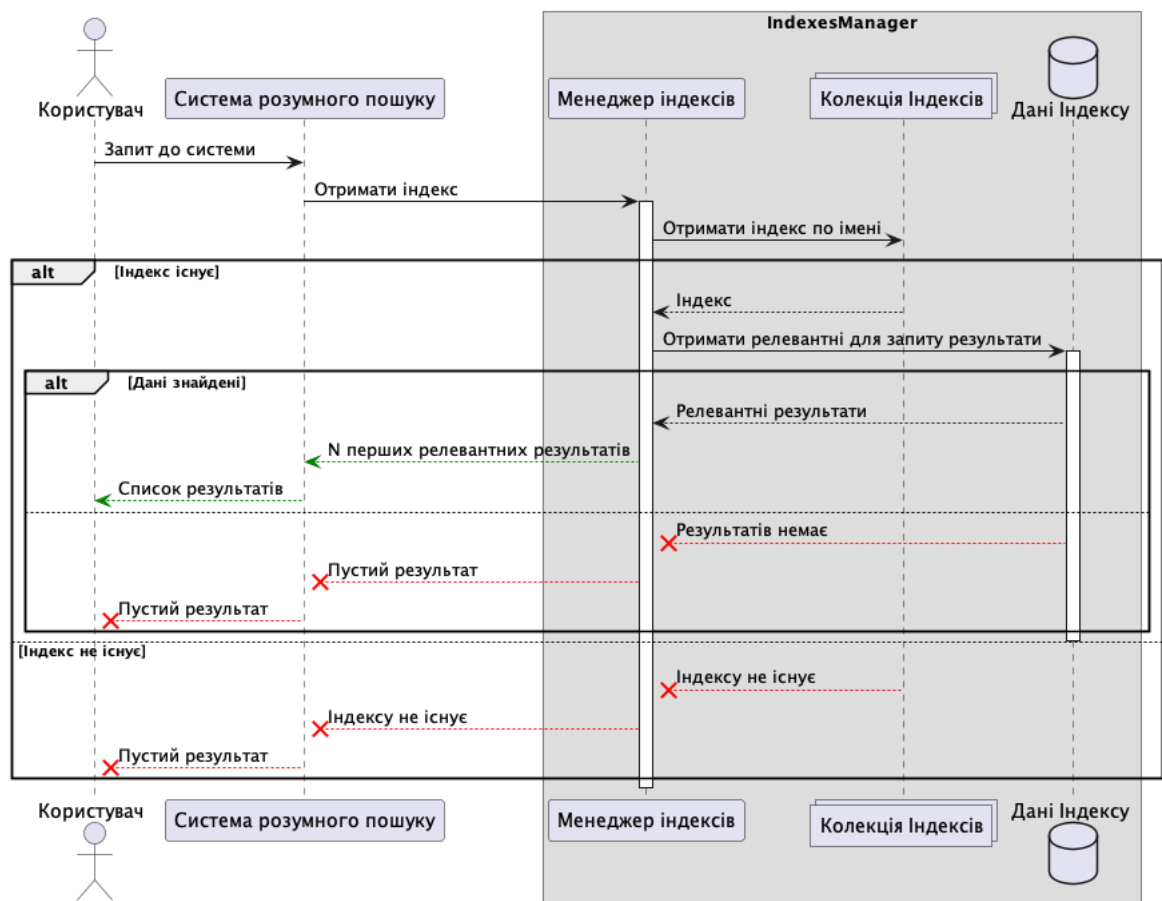


Рис 2.4 Sequence діаграма роботи системи розумного пошуку

Також варто показати створення індексів та додання інформації для пошуку.

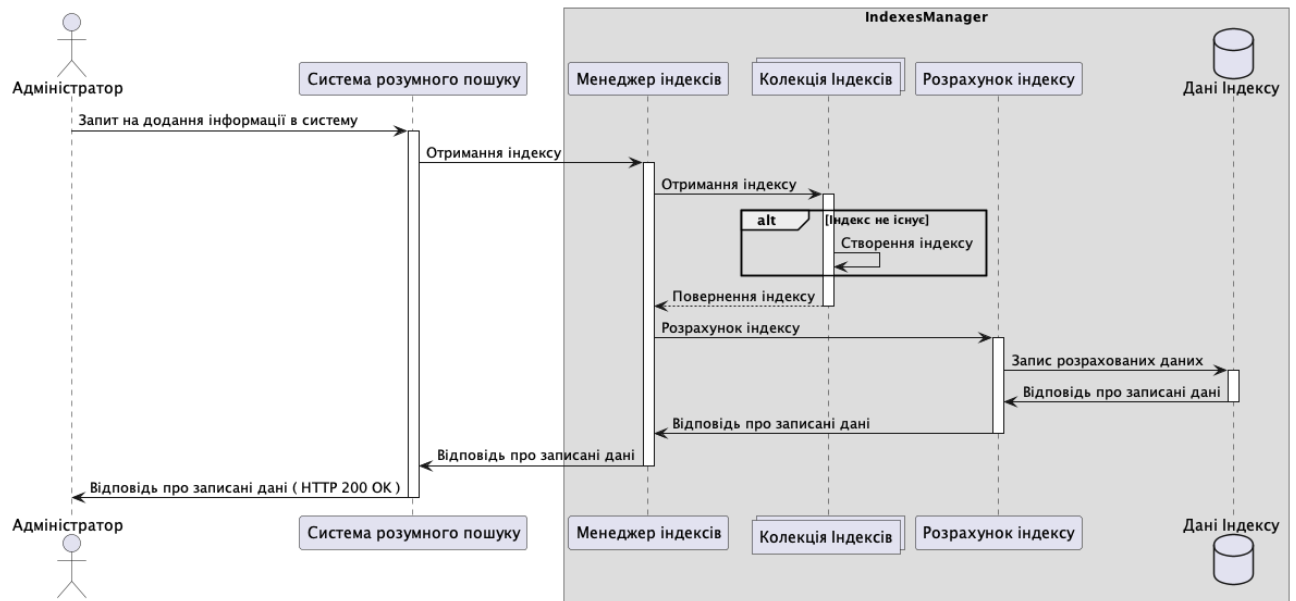


Рис 2.5 Sequence діаграма для створення та додавання даних в систему розумного пошуку

Діаграми зазначені вище описують загальну картину роботи системи без деталей про реалізацію окремих алгоритмів роботи.

В підсумку система реалізує визначені вище функціональні вимоги. Наступним кроком буде розробка системи на основі визначеної архітектури.

2.3. Реалізація системи розумного пошуку

На основі зазначених вище діаграм архітектури та поведінки системи можна почати розробку системи розумного пошуку. Як було вказано вище система складатиметься з двох мікросервісів і реалізує можливість додавати інформацію та здійснювати пошук по ній.

Логічним буде почати імплементацію з сервісу, який є вхідною точкою в систему та з яким буде інтегруватись мікросервіс для семантичного пошуку.

Для імплементації мікросервісу було вибрано мову програмування Go.

Go — компільована мова програмування із вбудованими засобами для паралельних обчислень і використання пакетів віддаленого керування. Цю мову програмування розробив Google. Офіційно мову представлено в листопаді 2009

року. Мова підтримується для операційних систем Linux, Android, Mac OS X та Windows. [23]

В таблиці 2.3 вказані основні характеристики мови Go, які мали вирішальну роль при виборі мови для реалізації.

Таблиця 2.3

Характеристика	Опис
Синтаксис	Синтаксис мови Go простий та лаконічний, що робить код легким для читання та розуміння.
Ефективність	Мова Go призначена для створення швидких та ефективних програм. Вона забезпечує високий рівень продуктивності завдяки своїй оптимізованій роботі з пам'яттю та ефективним механізмом конкурентної обробки.
Автоматичне управління пам'яттю	Go використовує автоматичне управління пам'яттю (Garbage Collection), що полегшує роботу програміста та сприяє уникненню витоку пам'яті.
Простота	Мова Go прагне до простоти та зосередженості на основних концепціях програмування, виключаючи складні або непотрібні особливості.
Інструменти розробки	Go постачається з різноманітними інструментами розробки, такими як

	компілятор, тестові інструменти, аналізатор коду та інші, що полегшують роботу програміста.
Крос-платформенність	Go є крос-платформенною мовою програмування, що означає, що програми, написані на Go, можуть запускатися на різних операційних системах без значних змін.
Модульність	Мова сприяє модульності та розбиттю програм на невеликі, незалежні компоненти.

Мова програмування Go була розроблена з орієнтацією на простоту, ефективність та швидкість розробки, і вона отримала популярність серед розробників завдяки своїм перевагам для написання сучасного програмного забезпечення.

Розглянемо структуру проекту та найкращі практики для проектів написаних з використанням Go.

В таблиці 2.4 можна побачити опис структури проекту та опис кожного з каталогів та його ролі в проекті. Дана структура проекту є загально прийнятою в спільноті розробників та є широко поширеною в існуючих проектах. Це базовий макет для проектів Go-додатків. Це не є офіційним стандартом, визначеним командою розробників Go; тим не менше, це звід шаблонів програмування в екосистемі Go, що склалися історично. Деякі з цих шаблонів більш популярні та відомі, ніж інші. Також в цьому макеті є кілька покращень, у тому числі кілька додаткових дерикторій, що використано в будь-якому достатньому реальному застосуванні [24].

Таблиця 2.4

Каталог	Опис
/pkg	Код бібліотеки, який можна використовувати зовнішніми програмами (наприклад, /pkg/engine). Інші проекти імпортують ці бібліотеки, очікуючи, що вони будуть працювати. Варто зауважити, що внутрішній каталог є кращим способом гарантувати, що ваші приватні пакети не будуть імпортовані, оскільки це забезпечується Go.
/internal	Приватний код додатків та бібліотек. Цей код, який ви не хочете, щоб інші імпортували у ваші програми або бібліотеки. Варто звернути увагу, що компонування цього шаблону забезпечується самим компілятором Go. Варто звернути увагу, що проект не обмежений директором внутрішнього верхнього рівня. Проект може мати більше одного внутрішнього каталогу на будь-якому рівні дерева проекту.
/cmd	Головні застосунки цього проекту. Ім'я каталогу для кожної додатки має збігатися з ім'ям виконуваного файлу, який ви хочете мати (наприклад,

	/cmd/myapp). Зазвичай є невелика функція main, яка імпортує та захищає код із каталогів /internal та /pkg і нічого більше.
--	---

З врахуванням зазначеної вище структури додатку було реалізовано мікросервіс для системи розумного пошуку. На рисунку 2.6 можна побачити фільтну структуру проекту, яка відповідає показаній вище структурі.

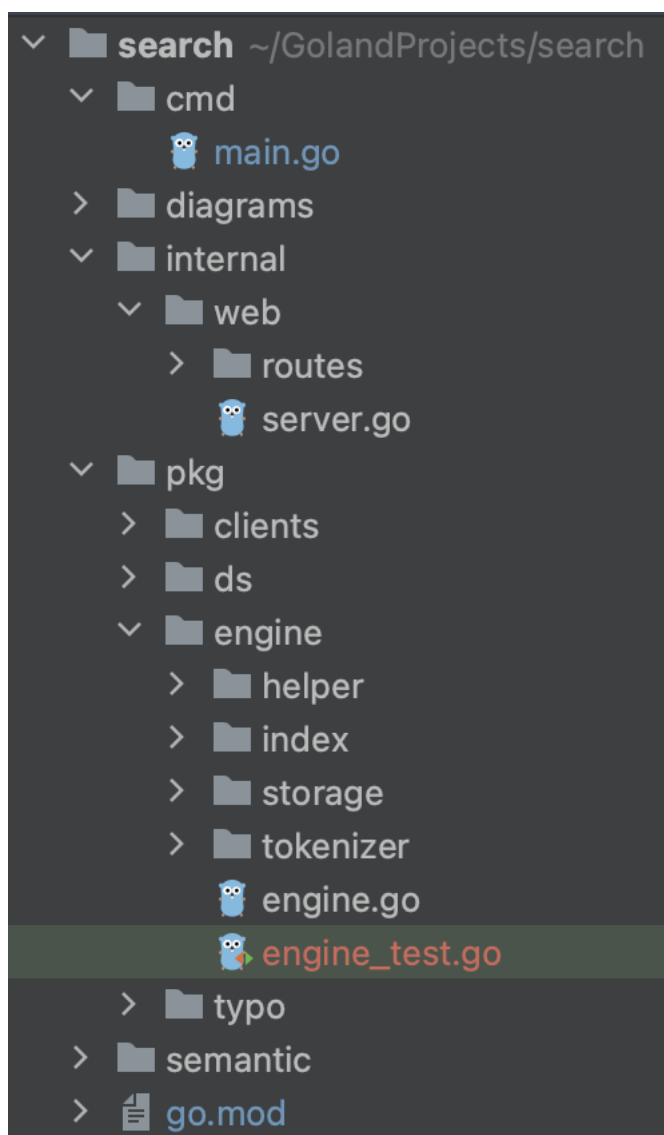


Рис 2.6 Фінальна структура проекту

Основна логіка пошуку реалізована в пакеті /pkg/engine. Вхідною точкою є структура Engine, яка зберігає в собі інформацію про індекси. На рисунку 2.7 можна побачити UML діаграму [25] для цієї структури.

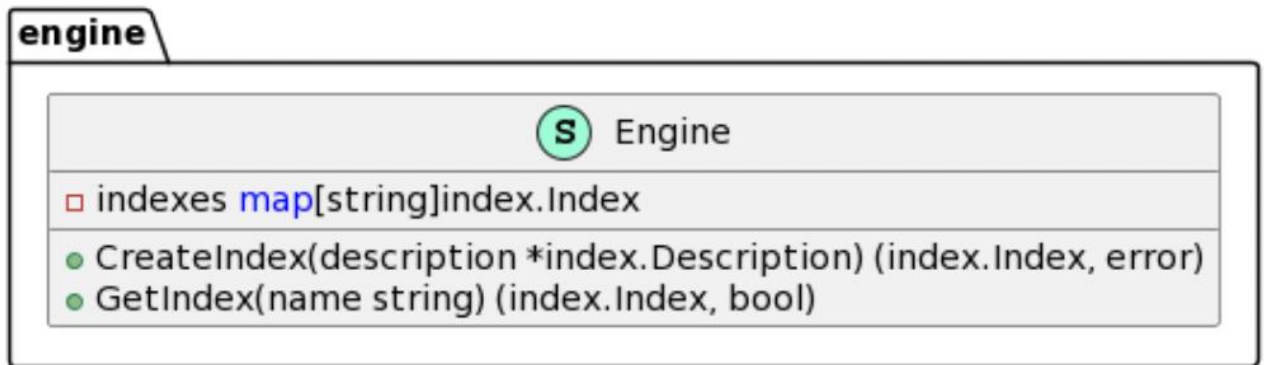


Рис 2.7 UML діаграма структури Engine

Як можна побачити вище на діаграмі структура Engine має інформацію про індекси. Як можна побачити на рисунку 2.8 index.Index це інтерфейс, який має пару реалізацій.

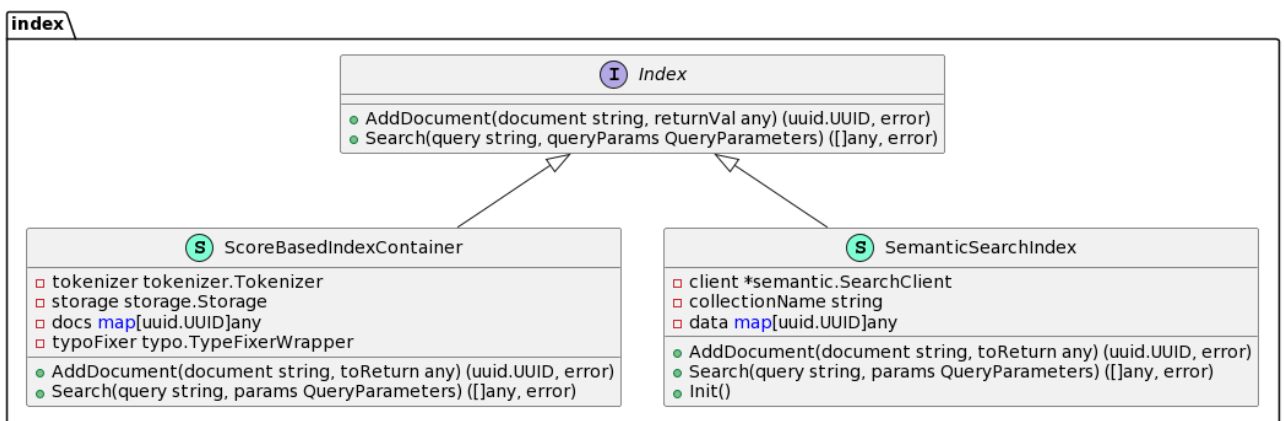


Рис 2.8 UML діаграма інтерфесу Index та його реалізацій

ScoreBasedIndexContainer – це реалізація інтерфейсу Index яка відповідає за розрахунок та зберігання інформації про документи і використовує традиційні алгоритми розрахунку релевантності документу для заданого пошукового запиту.

ScoreBasedIndexContainer має можливість додавання та пошуку по документах використовуючи традиційні алгоритми, такі як TF-IDF та BM25.

Також в разі помилки в запиті він має можливість пошуку з виправленням помилок. При вирішенні цієї проблеми було використано структуру даних BK-Tree з відстанню Левінштейна для визначення різниці між словами. Для додання контекстуальної інформації він використовує генерацію N-грам. Прикладом такого контексту може бути пошук по словосполученню Home alone. Без генерації N-грам система шукатиме релевантні документи по кожному слову окрему і результат пошуку може відрізнятись від очікуваного. При використанні цієї техніки система знатиме про це словосполучення та зможе коректно знайти релевантні документи.

Для створення N-грам було використано метод sliding window. Для довжини вікна використовувалось максимальна кількість слів в N-грамі. Кожне слово в вікні поєднувалось з наступним і таким чином створювались N-грами для заданого тексту. На рисунку 2.9 можна побачити ідею яка стоїть за методом sliding window.

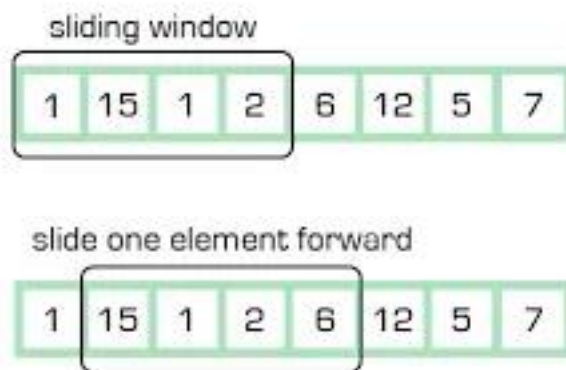


Рис 2.9 Sliding window метод

При пошуці по поширених словах результатів пошуку може бути багато і знаходження найбільш релевантних результатів може зменшити час відповіді системи. Так як система повинна повертати задану кількість найбільше релевантних результатів для їх визначення використалась структура даних Max Heap.

Структура даних купи — це тип двійкового дерева, який зазвичай використовується в інформатиці для різних цілей, включаючи сортування, пошук і впорядкування даних. У Мах-Неар ключ, присутній у кореневому вузлі, повинен бути більшим або дорівнювати ключам, присутнім у всіх його дочірніх вузлах. Також у Мах-Неар найбільший елемент є першим, який виривається з купи. На рисунку 2.10 можна побачити приклад цієї структури даних. Як видно на рисунку найбільший елемент знаходиться на вершині дерева, тому його отримання відбувається за константний час.

Завдяки використанню даної структури даних найбільш релевантні елементи отримуються швидше, ніж використовуючи сортування або прямолінійний пошук по всіх елементах.

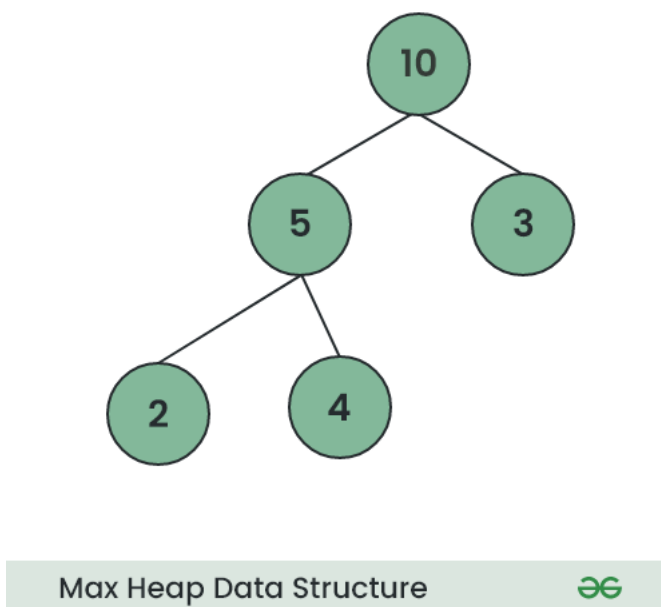


Рис 2.10 Приклад структури даних Мах Неар

Розрахунок потрібних для індексів даних здійснюється в реалізаціях інтерфейсу Storage. На рисунку 2.11 можна побачити UML діаграму для інтерфейсу Storage та його реалізацій. Як видно на діаграмі в інтерфейсу Storage є дві реалізації – для TF-IDF та BM25 відповідно.

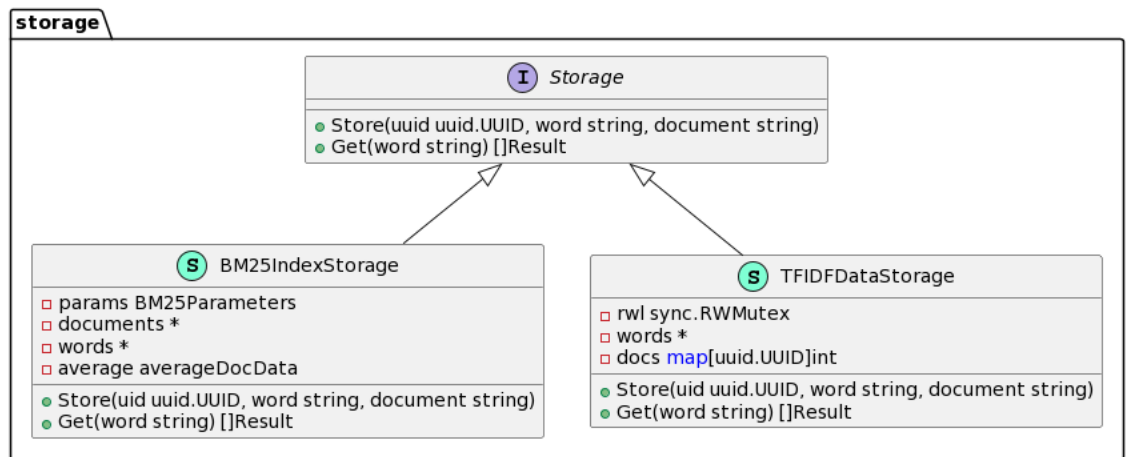


Рис 2.11 UML діаграму інтерфейсу Storage та його реалізацій

Враховуючи діаграми показані вище та загальну інформацію про індекс можна побудувати діаграму роботи даного типу індексу. На рисунку 2.12 можна побачити sequence діаграму роботи системи при додаванні елементу до індексу.

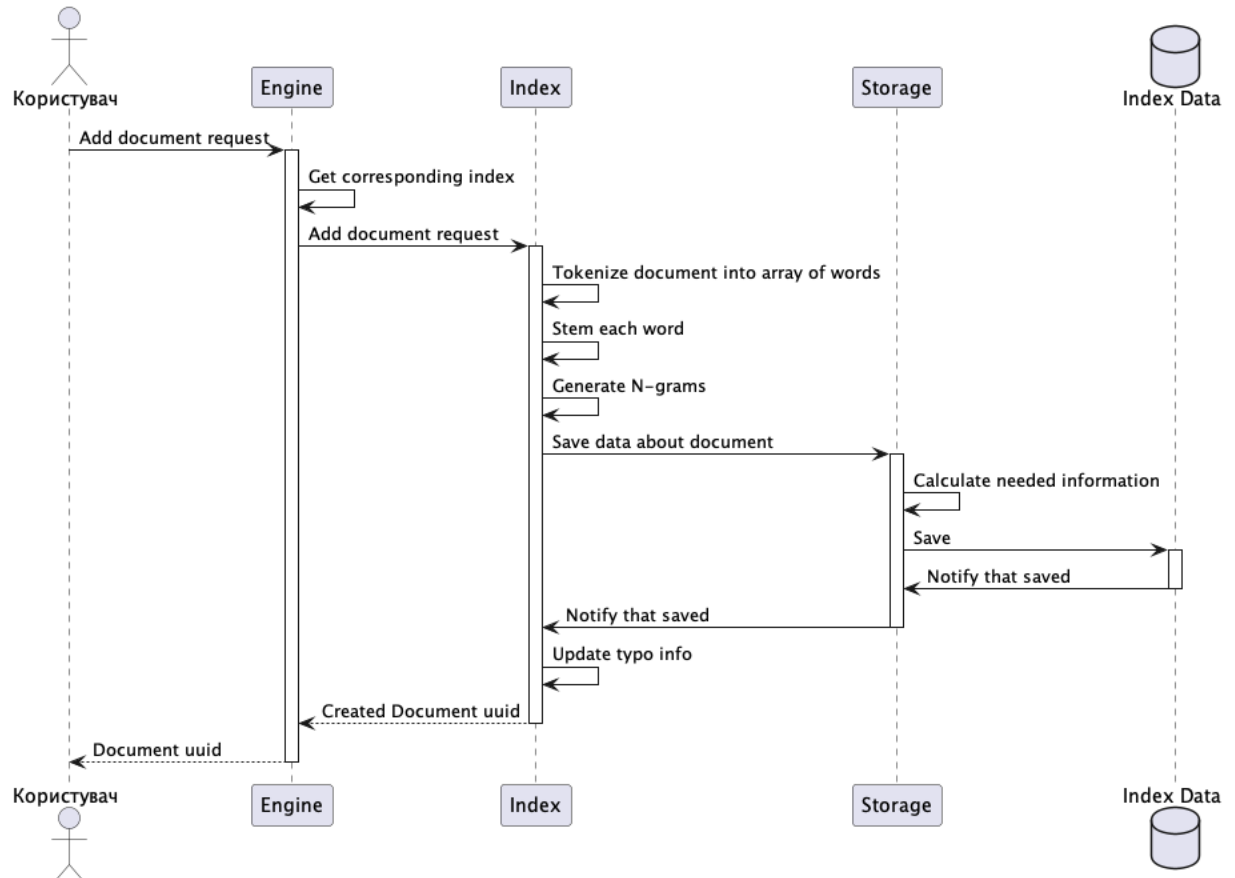


Рис 2.12 Sequence діаграма роботи ScoreBasedIndex при додаванні документу

Також важливо описати процес пошуку, так як ця функція є основною для системи і всі попередні діаграми були описом підготовки до процесу пошуку.

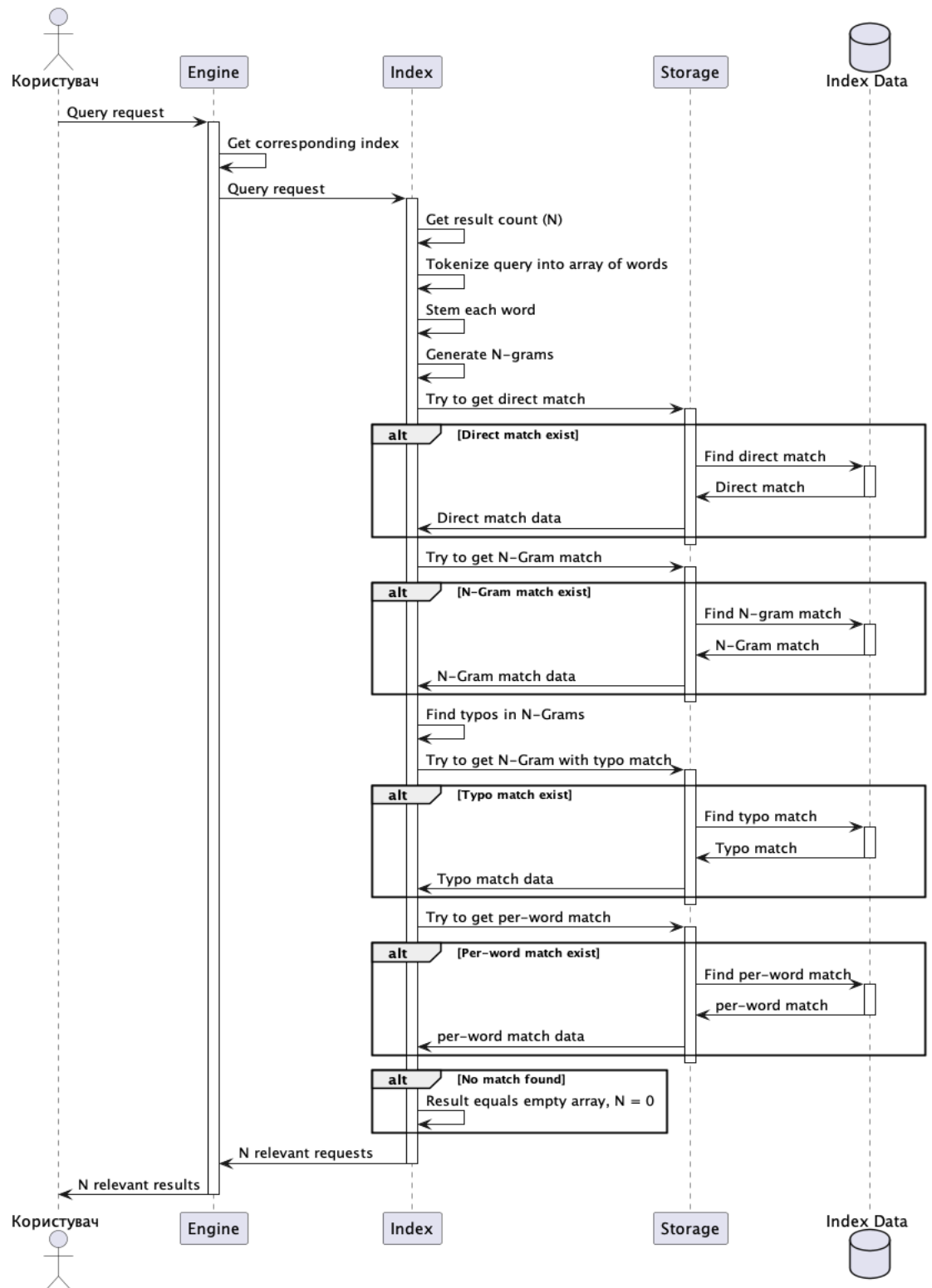


Рис 2.13 Sequence діаграму пошуку релевантних документів в ScoreBasedIndex

На рисунку 2.13 можна побачити sequence діаграму, яка описує процес пошуку. Як можна побачити з діаграми всі заздалегідь розраховані дані використовуються у процесі пошуку для пришвидшення цього процесу.

Також інтерфейс Index реалізує SemanticSearchIndex, який працює як проксі для мікросервісу відповідального за семантичний пошук. Реалізація має в собі REST клієнт який передає потрібні дані до сервісу, який повертає всю необхідну інформацію. На рисунку 2.14 можна побачити sequence діаграму роботи SemanticSearchIndex.

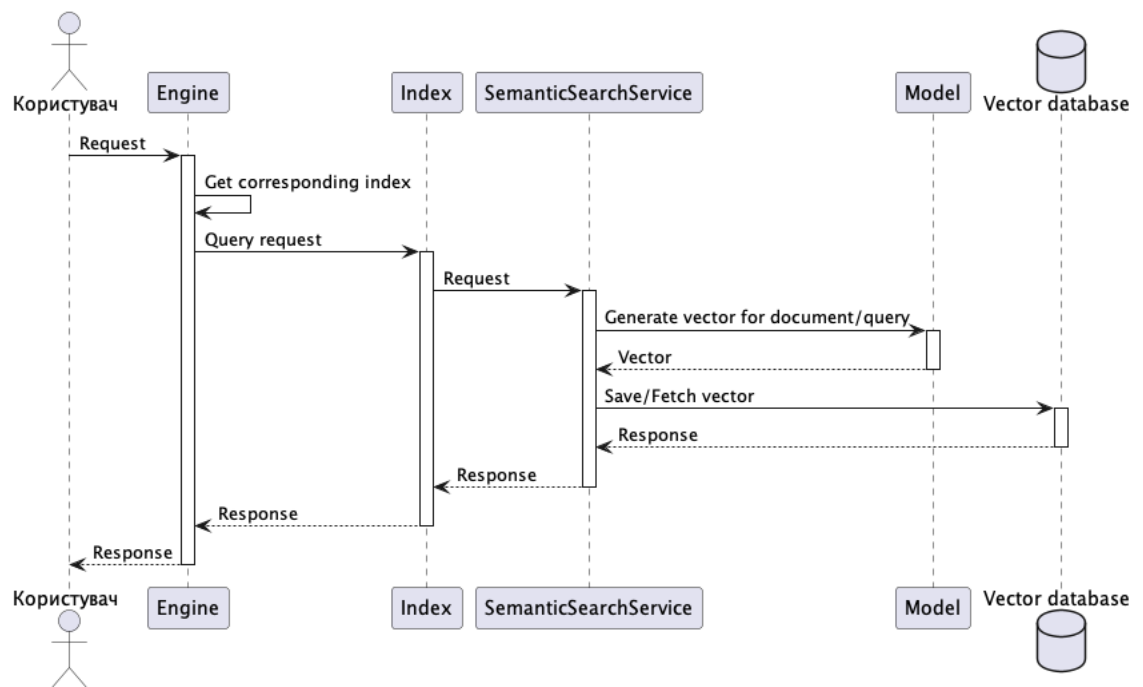


Рис 2.14 Sequence діаграми роботи SemanticSearchIndex

Як можна побачити на діаграмі цей сервіс виступає як проксі та передає всі необхідні дані іншому мікросервісу.

Перед розробкою сервісу для семантичного пошуку спочатку потрібно перевірити чи працюватиме запланована архітектура. Для перевірки гіпотези було розроблено Jupiter notebook з реалізацією Semantic search на основі заздалегідь заготовленого датасету.

На рисунку 2.15 можна побачити результат роботи функції `pandas.info()` для обраного для тестування датасету [26].

```

RangeIndex: 34886 entries, 0 to 34885
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Release Year          34886 non-null  int64
1   Title                 34886 non-null  object
2   Origin/Ethnicity      34886 non-null  object
3   Director              34886 non-null  object
4   Cast                 33464 non-null  object
5   Genre                 34886 non-null  object
6   Wiki Page            34886 non-null  object
7   Plot                 34886 non-null  object
dtypes: int64(1), object(7)

```

Рисунок 2.15 Результат pandas.info() для тестового датасету

В даному датасеті нас цікавлять колонки Title, Plot, WikiPage – Назва фільму, опис сюжету та сторінка вікіпедії відповідно.

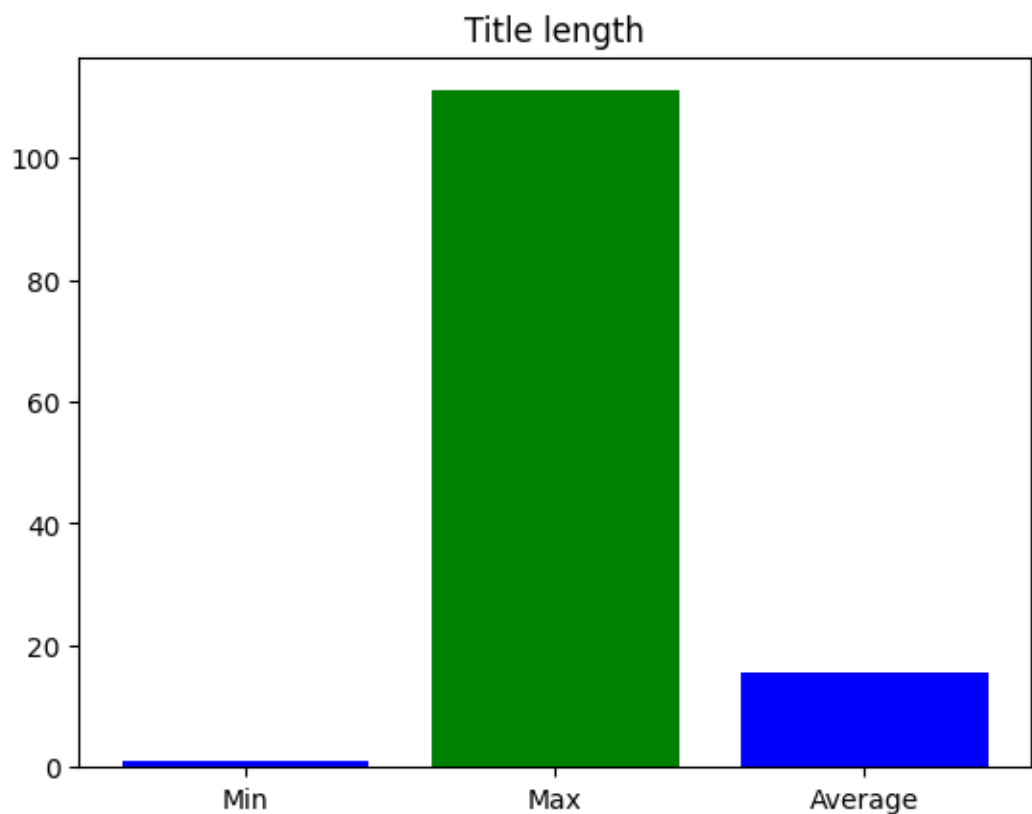


Рис 2.16 Bar Chart з інформацією про назви фільму

Як можна побачити на рисунку 2.16 середня довжина для назв фільмів з тестового датасету є невеликою, приблизно 20-25 символів.

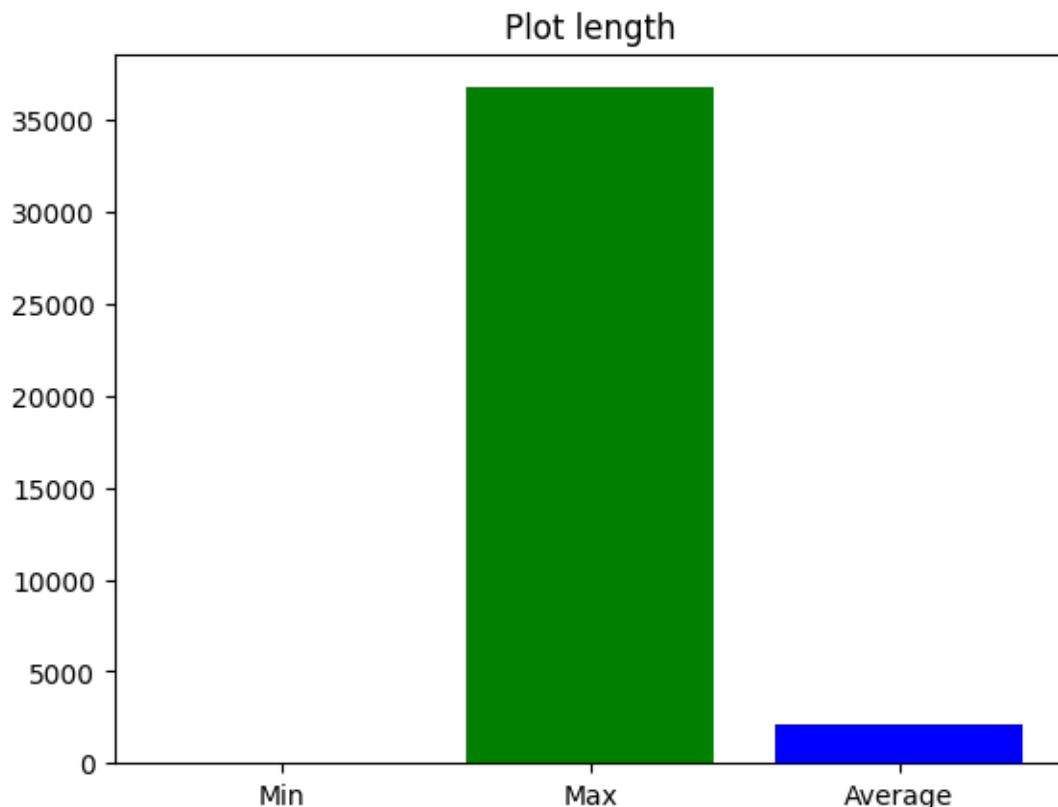


Рис 2.17 Bar Chart з інформацією про сюжети фільму

Як показано на рисунку 2.17 середня довжина опису сюжету фільму складає приблизно 2500-3000 слів.

Основою роботи мікросервісу для семантичного пошуку є перетворення тесту запису в векторну репрезентацію та подальший пошук по ній. Для досягнення цієї мети було використано моделі BERT (Bidirectional Encoder Representations from Transformers).

BERT (Bidirectional Encoder Representations from Transformers) є масштабним моделюванням мовлення, яке використовує трансформерну архітектуру для розуміння семантики тексту. В рамках BERT sentence transformer, використовується підхід, який дозволяє вбудовувати та отримувати векторні представлення для цілих речень або текстових фрагментів за допомогою попередньо навчених моделей BERT [27].

BERT sentence transformer використовує мережі глибокого навчання для створення контекстуалізованих векторних представлень речень, що полегшує роботу з текстовою інформацією та вирішує завдання, пов'язані з аналізом семантики тексту.

Таблиця 2.5

Характеристика	Опис
Bidirectional Context	BERT може "розуміти" слова в контексті всього речення, оскільки він використовує бідирекційні (з двох напрямків) трансформерні шари. Це дозволяє моделі ліпше розуміти семантичні залежності та взаємодії між словами.
Масштабованість та Попереднє Навчання	BERT використовує попереднє навчання на великих корпусах текстів, що дозволяє моделі засвоювати широкий спектр знань та контекстів.
Sentence Embeddings	Модель BERT sentence transformer генерує векторні представлення для цілих речень або текстових фрагментів, що можна використовувати для порівняння схожості текстів, кластеризації або подальшого аналізу.
Трансформерна Архітектура	Заснована на трансформерній архітектурі, що дозволяє ефективно моделювати довгострокові залежності в тексті та розуміти його контекст.

Попередньо Навчені Моделі	Існують попередньо навчені моделі BERT, які можна використовувати для вбудовування речень. Такі моделі можуть бути натреновані на великих наборах даних та потім використовуватися для різних завдань аналізу текстів.
Fine-tuning	В деяких випадках модель може бути додатково налаштована (fine-tuned) на конкретних завданнях для поліпшення її продуктивності в конкретному контексті.

В таблиці 2.5 мона побачити основні характеристики моделей BERT та їх опис. Описані вище характеристики стали ключовим фактором при виборі даного типу моделі для реалізації поставленої задачі.

Для реалізації поставленої задачі було використано попередньо навчену модель "all-MiniLM-L6-v2" [28].

```
df["vector"] = encoder.encode(df["Plot"]).tolist()
df.head(10)
```

	Title	Wiki Page	Plot	vector
0	Kansas Saloon Smashers	https://en.wikipedia.org/wiki/Kansas_Saloon_Sm...	A bartender is working at a saloon, serving dr...	[0.0013572077732533216, 0.02601543627679348, -...
1	Love by the Light of the Moon	https://en.wikipedia.org/wiki/Love_by_the_Ligh...	The moon, painted with a smiling face hangs ov...	[0.011549485847353935, 0.07491014152765274, 0....
2	The Martyred Presidents	https://en.wikipedia.org/wiki/The_Martyred_Pre...	The film, just over a minute long, is composed...	[-0.019372638314962387, 0.04275881126523018, -...
3	Terrible Teddy, the Grizzly King	https://en.wikipedia.org/wiki/Terrible_Teddy,...	Lasting just 61 seconds and consisting of two ...	[0.018185051158070564, 0.022814368829131126, 0...
4	Jack and the Beanstalk	https://en.wikipedia.org/wiki/Jack_and_the_Bea...	The earliest known adaptation of the classic f...	[-0.0362669862806797, -0.011878646910190582, 0...
5	Alice in Wonderland	https://en.wikipedia.org/wiki/Alice_in_Wonderl...	Alice follows a large white rabbit down a "Rab...	[0.0974636822938919, 0.005085118114948273, 0.0...
6	The Great Train Robbery	https://en.wikipedia.org/wiki/The_Great_Train_...	The film opens with two bandits breaking into ...	[-0.10890253633260727, 0.0368213951587677, -0....
7	The Suburbanite	https://en.wikipedia.org/wiki/The_Suburbanite	The film is about a family who move to the sub...	[-0.010947960428893566, -0.014319323934614658, ...
8	The Little Train Robbery	https://en.wikipedia.org/wiki/The_Little_Train...	The opening scene shows the interior of the ro...	[-0.1060173436999321, -0.021518142893910408, -...
9	The Night Before Christmas	https://en.wikipedia.org/wiki/The_Night_Before...	Scenes are introduced using lines of the poem....	[0.009964034892618656, 0.11225380003452301, -0...

Рис 2.18 Результат створення векторів на основі поля Plot з використанням моделі

На рисунку 2.18 можна побачити, що з використанням описаної вище моделі було створено векторні представлення для сюжетів фільмів.

Після того, як модель повертає векторне представлення речень вони зберігаються в векторну базу даних.

Система керування векторними базами даних (СКВБД, англ. vector database management system, VDBMS) або просто векторна база даних (англ. vector database) — це база даних, призначена для зберігання векторів, які зазвичай використовують із великими мовними моделями [29].

Для реалізації системи було обрано векторну базу даних Qdrant. Qdrant — пошукова система подібності векторів. Функції Qdrant, що розширюються, дозволяють використовувати всілякі нейронні мережі або семантичну відповідність, фасетний пошук та інші програми. Qdrant ідеально підходить для розгортання додатків на основі зіставлення вбудовувань, створених кодерами нейронної мережі [30].

На рисунку 2.19 показано код, який на основі створених вище векторів заповнює базу даних для подальшого пошуку.

```
collection_name = "films"
client.upload_records(
    collection_name=collection_name,
    records=[
        models.Record(
            id=idx, vector=doc["vector"], payload={"title": doc["Title"], "wiki": doc["Wiki Page"]}
        )
        for idx, doc in df.iterrows()
    ],
)
```

Рис 2.19 Код для заповнення векторної бази даних векторами створеними за допомогою моделі

На рисунку 2.20 можна побачити, що згенеровані моделлю вектори є схожими для фільмів, які мають схожий сюжет. Це досягається за рахунок того, що модель враховує контекст при генерації векторів.

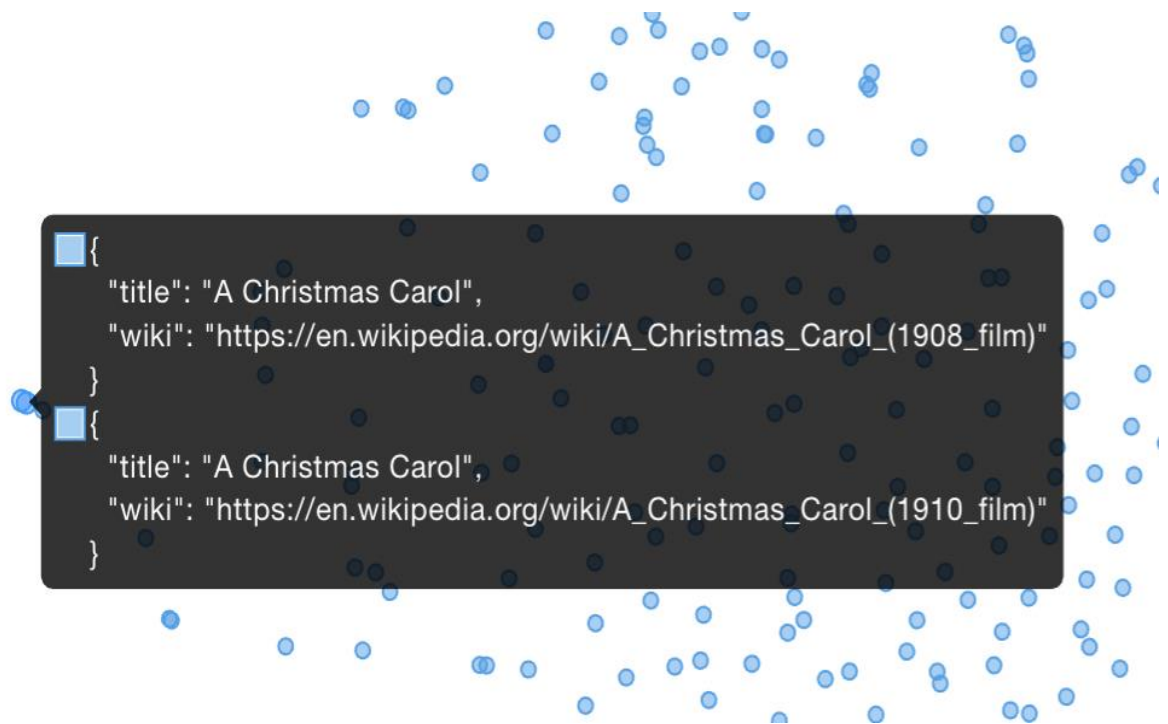


Рис 2.20 Візуалізація векторів, які зберігаються в векторній базі даних

Процес пошуку відбувається за схожим принципом. Тіло запиту передається моделі, яка створює векторну репрезентацію. Потім система робиться пошук схожих векторів в векторній базі даних і як результат повертається N найбільш релевантних результатів.

Приклад роботи системи, навченої з використанням зазначеного вище датасету можна побачити на рисунку 2.21.

```

Query: film about iraq war
Results:
{'title': 'A Soldier Speaks after Death', 'wiki': 'https://en.wikipedia.org/wiki/A_Soldier_Speaks_after_Death'}
{'title': 'Nefes: Vatan Sağolsun', 'wiki': 'https://en.wikipedia.org/wiki/Nefes:_Vatan_Sa%C4%9Folsun'}
{'title': 'The Heroes of Desert Storm', 'wiki': 'https://en.wikipedia.org/wiki/The_Heroes_of_Desert_Storm'}
{'title': 'Path to War', 'wiki': 'https://en.wikipedia.org/wiki/Path_to_War'}
{'title': 'In the Sands of Babylon', 'wiki': 'https://en.wikipedia.org/wiki/In_the_Sands_of_Babylon'}

```

Рис 2.21 Результат семантичного пошуку

На основі описаного вище Jupiter notebook було створено мікросервіс для семантичного пошуку. В ньому було використано модель та базу даних описані вище. Для створення мікросервісу було використано мікрофреймворк Flask.

Flask — мікрофреймворк для вебдодатків, створений з використанням Python. Його основа складає інструментарій Werkzeug та рушій шаблонів Jinja2. Поширюється відповідно до умов ліцензії BSD. Flask називається мікрофреймворком, оскільки він не вимагає спеціальних засобів чи бібліотек. У ньому відсутній рівень абстракції для роботи з базою даних, перевірки форм або інші компоненти, які надають широковживані функції за допомогою сторонніх бібліотек. Однак, Flask має підтримку розширень, які надають додаткові властивості таким чином, наче вони були доступні у Flask із самого початку. Існують розширення для встановлення об'єктно-реляційних зв'язків, перевірки форм, контролю процесу завантаження, підтримки різноманітних відкритих технологій аутентифікації та декількох поширених засобів для фреймворку. Розширення оновлюються частіше ніж базовий код. [31]

Висновки до 2 розділу

В даному розділі було поставлено вимоги та її складових де було описано функціональні та нефункціональні вимоги. Було визначено як саме система повинна працювати та інтегруватись з іншими системами.

Також було описано архітектуру системи розумного пошуку. Було розроблено діаграми, які описують архітектуру, зокрема діаграми розгортання, sequence діаграми з детальним описом роботи системи.

Було реалізовано систему розумного пошуку, зокрема два мікросервіси які відповідають за роботу системи. Спочатку система отримує дані, обробляє їх для пришвидшення подальшого пошуку — формування індексу. Реалізовано декілька методів обробки та пошуку по даних з використанням традиційних методів та з використанням нейронних мереж для покращення результатів.

РОЗДІЛ 3. ОПИС РОЗРОБЛЕНОЇ СИСТЕМИ РОЗУМНОГО ПОШУКУ ДЛЯ САЙТУ

3.1. Тестування та представлення роботи сервісу

В даному розділі буде показано результати роботи розробленого сервісу та протестовано його роботу, швидкість повернення результатів та їх релевантність.

Почати тестування варто зі створення індексу, так як самі індекси тримають в собі всю інформацію пов'язану з пошуком.

На рисунку 3.1 можна побачити запит на створення індексу з типом score та алгоритмом підрахунку релевантності BM25.

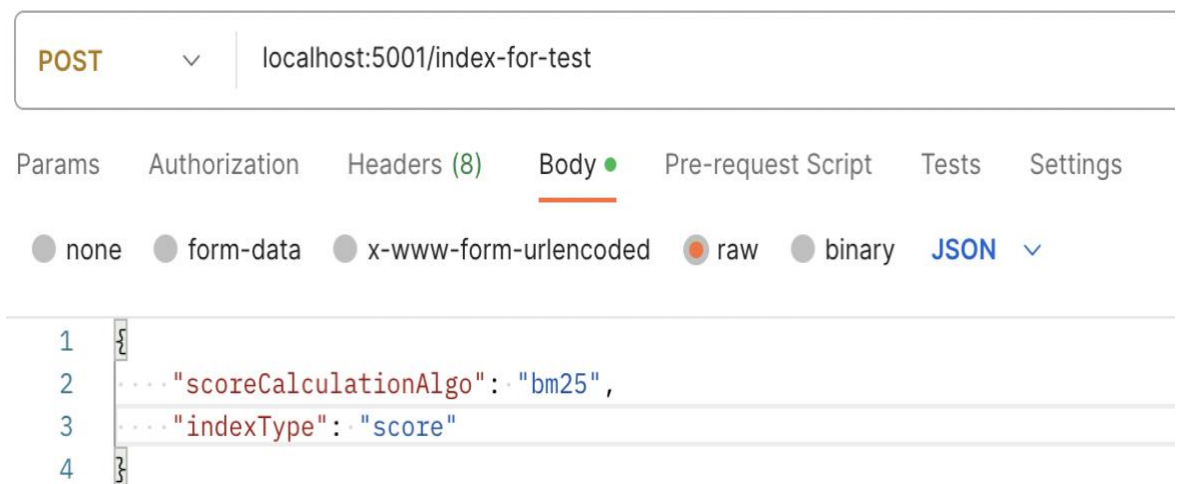


Рис 3.1 Запит на створення індексу

Якщо індексу не існує та конфігурація правильна буде створено новий індекс. Результат виклику API описаного вище рисунку можна побачити на рисунку 3.2.



Рис 3.2 Відповідь на запит на створення індексу

У випадку коли індекс вже існує система повинна коректно відпрацювати та повернути помилку замість переписування вже існуючої інформації. На рисунку 3.3 показано відповідь на API виклик описаний вище за умови, що індекс вже існує.

```

1  {
2    "error": "Index already exist"
3  }
```

Рис 3.3 Відповідь при спробі створити вже існуючий індекс

Коли індекс вже створений потрібно додати інформацію по якій планується провиритись пошук. Для цього є окремий ендпоінт, який потрібно викликати.

На рисунку 3.4 можна побачити приклад запиту, за допомогою якого документ додається до створеного завчасно індексу.

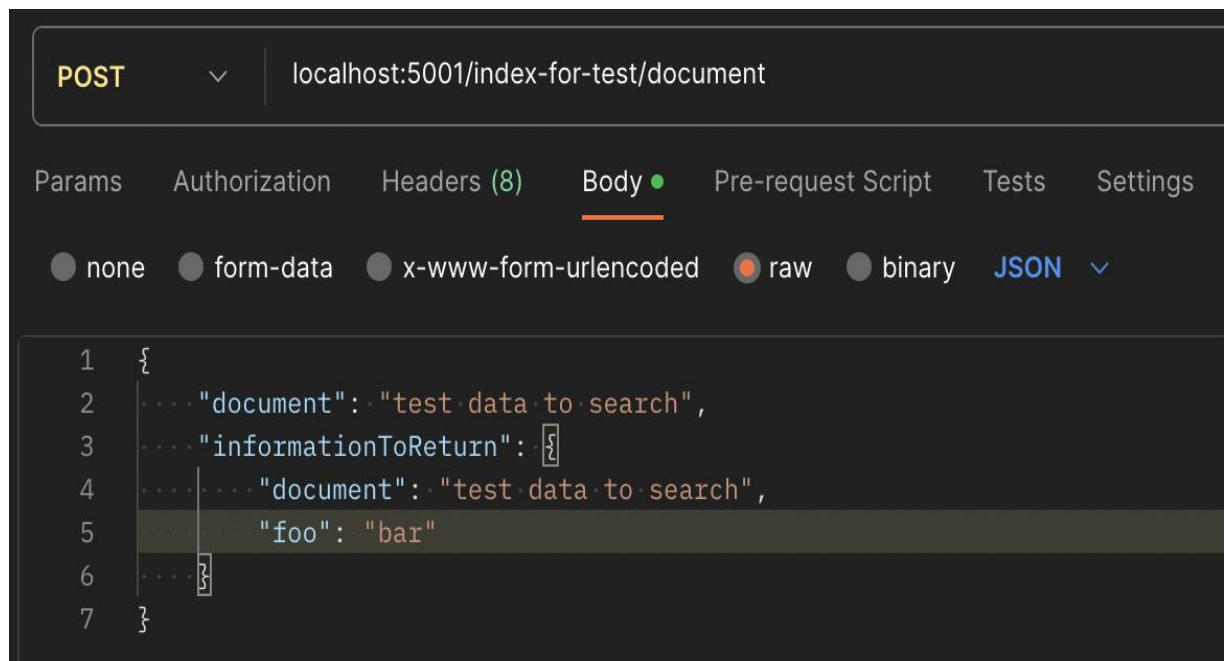


Рис 3.4 Запит на додання документу

Якщо документ був доданий і збережений буде повернуто відповідь з кодом HTTP 201 Created та UUID відповідного документу. Приклад такої відповіді можна побачити на рисунку 3.5.

```

{
  "request": {
    "document": "test data to search",
    "informationToReturn": {
      "document": "test data to search",
      "foo": "bar"
    }
  },
  "uuid": "67c478dc-a331-11ee-9b92-16e3375d0045"
}

```

Рис 3.5 Відповідь на запит на додання документу

Для коректного додання документу до індексу індекс повинен існувати та запит має бути коректним. Приклад коректного запиту був показаний вище. На рисунку 3.6 можна побачити відповідь системи при спробі додати документ до неіснуючого індексу.

```

{"error": "Index not exist"}

```

Рис 3.6 Відповідь при спробі додати документ до неіснуючого індексу

Коли індекс наповнений даними для пошуку можна починати тестування самого пошуку. На рисунку 3.7 можна побачити приклад запиту до систему для пошуку. Даний пошуковий запит повинен повернути доданий вище документ з додатковими даними зазначеними в тілі запиту при доданні документу.

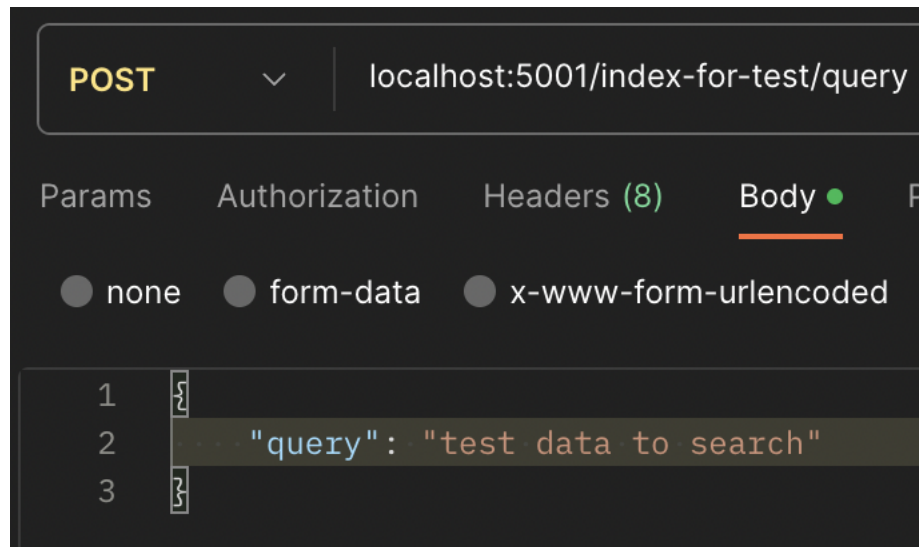


Рис 3.7 Пошуковий запит до системи розумного пошуку

Результат повернутий системою на запит описаний вище можна побачити на рисунку 3.8. Як можна побачити дані з тіла запиту на додання документу повертаються при пошуці.



Рис 3.8 Відповідь системи на пошуковий запит

Показаний вище результат є прикладом пошуку системи з поверненням Direct match. В цьому випадку система перевіряє чи існує документ який повністю відповідає запиту на пошук.

Випадок зазначений вище є одним з спеціальних кейсів. Більш поширеним є пошук по фразі чи слову які можуть знаходитись в документі. Приклад такого запиту можна побачити на рисунку 3.9.

```
{
  "query": "test data"
}
```

Рис 3.9 Приклад запису до системи для пошуку по окремій фразі

Відповідь на описаний вище запит можна побачити на рисунку 3.10. Як і очікувалось система має повернути тей ж документ, так як пошуковий запит є частиною документу.

```
{
  "payload": [
    {
      "document": "test data to search",
      "foo": "bar"
    }
  ],
  "query": "test data"
}
```

Рис 3.10 Відповідь системи на пошуковий запит

В процесі формування індексу система проводить процес стемінгу, тому якщо слово вжите в іншій формі документ також має бути знайдений.

```
{
  "document": "testing data is being stemmed",
  "informationToReturn": {
    "document": "testing data is being stemmed",
    "bar": "foo"
  }
}
```

Рис 3.11 Запит на додання документу зі словом test вжитим в іншій формі

Після того, як новий документ був доданий при спробі пошуку система має повернути обидва документи, так як вони мають в собі тіло запиту. Як можна побачити на рисунку 3.12 результат має обидва документи.

```
{
  "payload": [
    {
      "bar": "foo",
      "document": "testing data is being stemmed"
    },
    {
      "document": "test data to search",
      "foo": "bar"
    }
  ],
  "query": "test data"
}
```

Рис 3.12 Повернення коректний результатів завдяку процесу стемінгу

Також в функціональних вимогах було описано можливість системи знаходити результати для запитів, які містять помилки. На рисунку 3.13 можна побачити такий запит – замість слова test тут написано text. Система повинна повернути ті самі результати.

```
{
  "query": "text data"
}
```

Рис 3.13 Запит до системи з помилкою

На рисунку 3.14 можна побачити результат який повертає система при спробі запиту який містить помилки.

```

{
  "payload": [
    {
      "bar": "foo",
      "document": "testing data is being stemmed"
    },
    {
      "document": "test data to search",
      "foo": "bar"
    }
  ],
  "query": "text data"
}

```

Рис 3.14 Система повертає результати при помилках в запиті

Також система розумного пошуку має повертати найбільш релевантні результати. Для досягнення потрібних результатів в процесі створення індексу для кожного документу генеруються N-грами. За допомогою N-грам можна зберігати контекст і повертати результати, які очікує користувач.

Для того, щоб показати важливість N-грам для результатів пошуку було створено індекс, в який було додано 30 тисяч документів. Документи містять в собі інформацію про фільми. До цієї інформації відпоситься Назва, Опис сюжету та Постер до фільму. Ми хочемо знайти фільм Home alone. На рисунку 3.15 показано запит до системи, і ми очікуємо що буде повернуто результат, який буде пов'язаний з цим фільмом.

```

{
  "query": "Home alone",
  "limit": 3
}

```

Рис 3.15 Запит до системи з одним словом

За рахунок того, що фільмів яка мають слова Home та alone в назві багато повернені результати є релевантними, але система не знає про існування контексту в якому Home alone має сенс. На рисунку 3.16 можна побачити результат який повертає система при вимкненій генерації N-грам.

```
{
  "payload": [
    "All Alone",
    "I Think We're Alone Now",
    "A Man Alone"
  ],
  "query": "Home alone"
}
```

Рис 3.16 Результат повернутий системою без використання N-грам

Як можна побачити з показаного вище результату результати є релевантними і мають слова з запиту, але саме потрібний нам фільм не був повернений.

На рисунку 3.17 можна побачити, що при генерації N-грам система починає розуміти, що слова Home та alone можуть бути використані разом і за рахунок цього релевантність результату збільшується.

```
{
  "payload": [
    "A Girl Walks Home Alone at Night",
    "Home Alone 4",
    "Home Alone 3"
  ],
  "query": "Home alone"
}
```

Рис 3.17 Результат повернутий системою з використанням N-грам

Вказані вище результати використовували традиційні методи пошуку, описані вище.

Однією з функціональних вимог поставлених до системи була можливість пошуку з врахуванням контексту, а не тільки ключових слів та алгоритмів маніпуляції з ними.

На рисунку 3.18 можна побачити приклад запиту для семантичного пошуку. Навіть з врахуванням генерації N-грам система не зможе коректно зрозуміти та знайти релевантні результати.

```
{
  "query": "film about girl playing soccer",
  "limit": 3
}
```

Рис 3.18 Приклад пошукового запиту для семантичного пошуку

На рисунку 3.19 можна побачити результат повернутий системою з використанням мереж - семантичного пошуку.

```
{
  "title": "Bend It Like Beckham",
  "overview": "Jess Bhamra, the daughter of a strict Indian couple in London, is not permitted to play organized soccer, even though she is 18. When Jess is playing for fun one day, her impressive skills are seen by Jules Paxton, who then convinces Jess to play for her semi-pro team. Jess uses elaborate excuses to hide her matches from her family while also dealing with her romantic feelings for her coach, Joe."
}
```

Рис 3.19 Приклад результату повернутого системою з використанням семантичного пошуку

Як підсумок можна сказати, що функціональні вимоги до системи поставлені раніше були реалізовані, система працює стабільно та швидко.

3.2. Графічний інтерфейс

Для тестування системи було розроблено графічний інтерфес для кращої демонстрації результатів пошуку та більшої наглядності її результатів.

Попередньо було створено два індекси з документами які містять інформацію про фільми – зокрема Назву, Короткий опис фільму, Жанр та

Картинку для постеру. Один індекс був створений з використання BM25 алгоритму, інший з використанням семантичного пошуку описаного вище.

На рисунку 3.20 можна побачити пусту сторінку для пошуку. З лівої сторони знаходиться дропдаун для вибору кількості результатів пошуку, по середині пошукова строка і справа перемикач між індексами. Коли перемикач ввімкнений система використовує семантичний пошук для знаходження релевантних результатів.

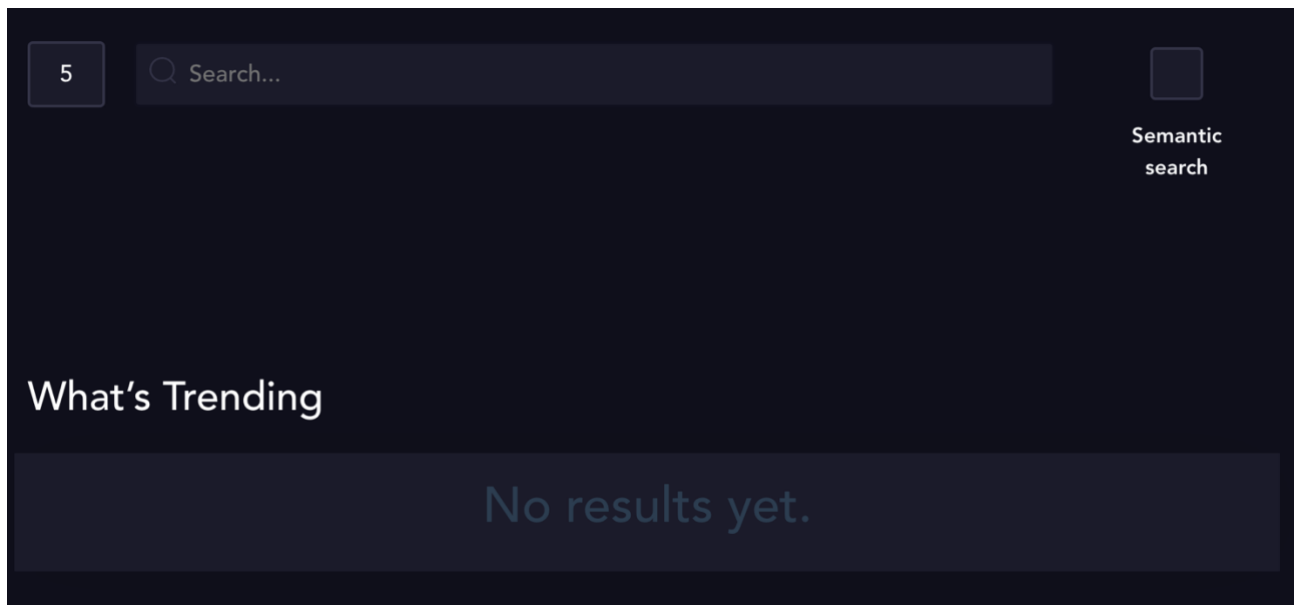


Рис 3.20 Пуста сторінка пошуку

На рисунку 3.21 показано виведення результатів пошуку використовуючи традиційний метод пошуку. Також відразу можна побачити як система повертає результати для запиту з помилками та з генерацією N-грам.

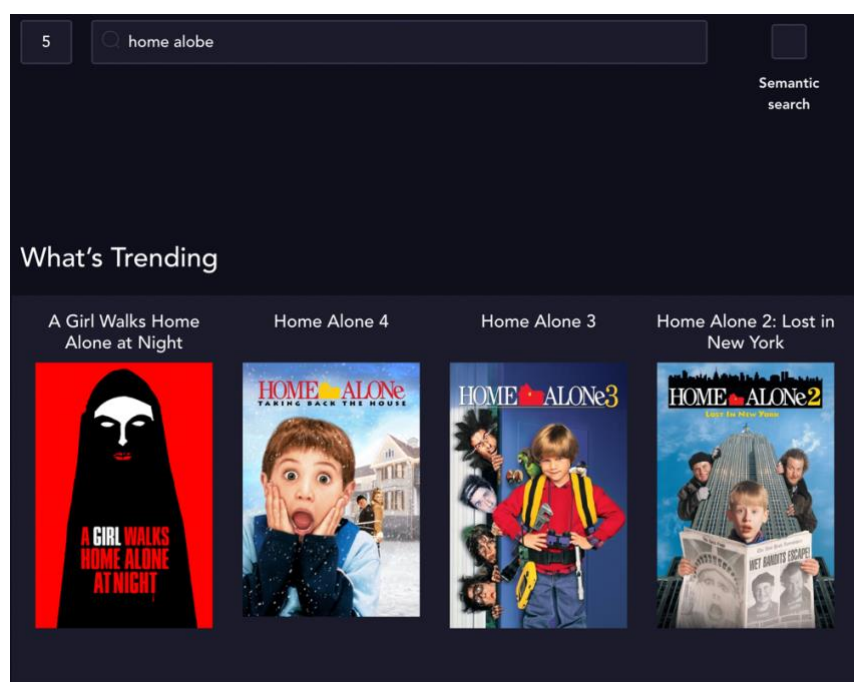


Рис 3.21 Відображення результатів пошуку

На рисунку 3.22 можна побачити результат роботи семантичного пошуку. Якщо подивитись на жанри фільмів повернутих системою можна побачити, що всі вони є драмами.

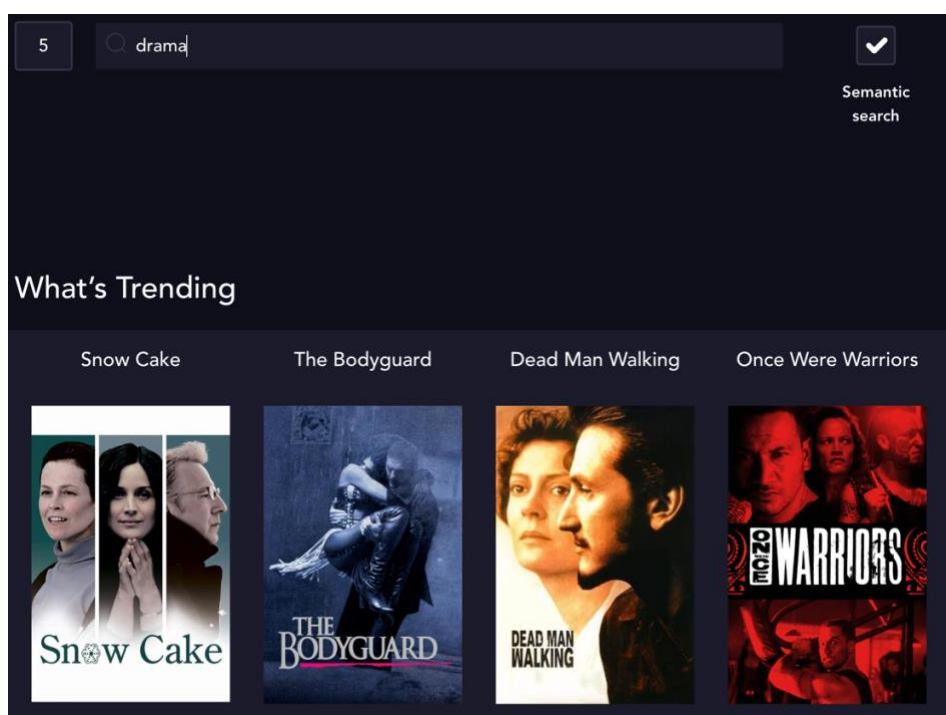


Рис 3.22 Відображення результатів пошуку з використанням семантичного пошуку

Висновки до 3 розділу

В даному розділі було показано та детально описано роботу системи. Було показано реалізацію кожної з визначених вище функціональної та нефункціональної вимог, протестовано систему та наглядно показано різницю між різними видами пошуку. Також було показано графічний інтерфейс розроблений для візуалізації та наглядності роботи системи розумного пошуку для сайту.

ВИСНОВКИ

В даній роботі було реалізовано програмний додаток, систему, яка є зручною для використання. Було проведено аналіз та порівняння різних типів пошуку – зокрема традиційних та з використанням нейронних мереж. Було підготовлено технічне завдання та вибрано коректні інструменти для реалізації системи.

Було реалізовано систему, яка є легкою в налаштуванні та підтримує різні типи пошуку. Було розроблено підсистему, яка за допомогою нейронних мереж покращує процес пошуку та робить його результати більш релевантними.

Після створення програми було проведено тестування. Воно полягало в перевірці якості програмного додатку та правильності його роботи.

Отримані результати відповідають поставленим до системи вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Schütze, Hinrich; Christopher D. Manning; Raghavan, Prabhakar (2008). Introduction to information retrieval. Cambridge, UK: Cambridge University Press. ISBN 0-521-86571-9. Архів оригіналу за 12 листопада 2018.
2. F. Crestani and G. Pasi. Soft Information Retrieval: Applications of Fuzzy Set Theory and Neural Networks. in «Neuro-fuzzy Techniques for Intelligent Information Systems», N.Kasabov and Robert Kozma Editors, Physica-Verlag, Springer-Verlag Group , 287—313, 1999.
3. What Is Natural Language Processing (NLP)? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.oracle.com/eg/artificial-intelligence/what-is-natural-language-processing/>.
4. TF-IDF — Term Frequency-Inverse Document Frequency [Електронний ресурс] — Режим доступу до ресурсу: <https://www.learn datasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/>.
5. Jones K. S. A statistical interpretation of term specificity and its application in retrieval // Journal of Documentation : журнал. — MCB University : MCB University Press, 2004. — Т. 60, № 5. — С. 493-502.
6. Understanding the BM25 Ranking Algorithm [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@evertongomede/understanding-the-bm25-ranking-algorithm-19f6d45c6ce>.
7. Introduction to Stemming [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/introduction-to-stemming/>.
8. Lovins, J. B. (1968); Development of a Stemming Algorithm, Mechanical Translation and Computational Linguistics, 11, 22—31
9. P. Willett. The Porter stemming algorithm: then and now. Program: Electronic Library and Information Systems. — 2006. — Vol. 40, iss. 3. — P. 219—223. — ISSN 0033-0337.

10. Navarro, Gonzalo (2001). "A guided tour to approximate string matching". ACM Computing Surveys. 33 (1): 31–88.
11. Schulz, Klaus U.; Mihov, Stoyan (2002). "Fast String Correction with Levenshtein-Automata". International Journal of Document Analysis and Recognition. 5 (1): 67–85.
12. Lucene's FuzzyQuery is 100 times faster in 4.0 [Електронний ресурс]. – 2011.
– Режим доступу до ресурсу: <https://blog.mikemccandless.com/2011/03/lucenes-fuzzyquery-is-100-times-faster.html>.
13. Singhal, Amit (2001). "Modern Information Retrieval: A Brief Overview". Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4): 35–43.
14. Bast, Hannah; Buchhold, Björn; Haussmann, Elmar (2016). "Semantic search on text and knowledge bases". Foundations and Trends in Information Retrieval. 10 (2–3): 119–271.
15. What is semantic search? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.elastic.co/what-is/semantic-search>.
16. Understanding Elasticsearch [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@eshwar.koride7/everything-about-elasticsearch-a8f36757457e>.
17. What is Algolia? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.algolia.com/doc/guides/getting-started/what-is-algolia/>.
18. Вимоги до програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: <https://studfile.net/preview/5130988/page:2/>.
19. Козак О.Л. Опорний конспект лекцій з курсу —Аналіз вимог до програмного забезпечення для студентів напрямку підготовки — Програмна інженерія [Електронний ресурс] / О.Л. Козак. – Тернопіль, 2011. – 56 с. – Режим доступу до ресурсу: http://dspace.wunu.edu.ua/retrieve/14135/FCIT_kKN_sPZS_dAVPZ_%20LE_C.pdf

20. Специфікація вимог до програмного забезпечення: Методичні вказівки до лабораторних робіт № 1, 2 з дисципліни «Аналіз вимог до програмного забезпечення» для студентів базового напрямку 6.050103 «Програмна інженерія» / Укл.: О.Є. Білас, Н. Б. Мельник – Львів: Видавництво Національного університету «Львівська політехніка», 2015. – 16 с.
21. Yourdon, Edward. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. New York: Yourdon Press, 1979. ISBN 978-0138544713. P. 473.
22. Data Flow Diagrams (DFD) Explained [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: <https://www.artofba.com/uk/post/data-flow-diagrams-dfd-explained-1>.
23. What is Go? Golang Programming Language Meaning Explained [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <https://www.freecodecamp.org/news/what-is-go-programming-language/>.
24. Standard Go Project Layout [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/golang-standards/project-layout>.
25. What is Unified Modeling Language (UML)? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/>.
26. Wikipedia Movie Plots [Електронний ресурс] – Режим доступу до ресурсу: <https://www.kaggle.com/datasets/jrobischoon/wikipedia-movie-plots>.
27. Semantic Search with S-BERT is all you need [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <https://medium.com/mlearning-ai/semantic-search-with-s-bert-is-all-you-need-951bc710e160>.
28. Pretrained Models [Електронний ресурс] – Режим доступу до ресурсу: https://sbert.net/docs/pretrained_models.html.
29. What is a vector database? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ibm.com/topics/vector-database>.
30. Qdrant documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://qdrant.tech/documentation/>.

31.Flask [Электронный ресурс] – Режим доступа до ресурсу:
<https://uk.wikipedia.org/wiki/Flask>.