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

Використання Elasticsearch для реалізації повнотекстового пошуку у комерційних системах

Текстова частина до курсової роботи
за спеціальністю «Комп'ютерні науки» - 122

Керівник курсової роботи
асистент
Яремко С. А.
_____
(підпис)
"____" _____ 2022 р.
Виконала студентка МП КН-1:
Яськова Д. В.
"____" _____ 2022 р.

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

ЗАТВЕРДЖУЮ

асистент

Яремко С. А.

_____

(підпис)

„ ____”_____2021р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студентці Яськовій Дарині Вадимівні
факультету магістерської програми
ТЕМА: **Використання Elasticsearch для реалізації повнотекстового пошуку у комерційних системах**

Зміст ТЧ до курсової роботи:

Анотація
Вступ
Розділ 1. Використання пошукових двигунів у застосунках
Розділ 2. Дослідження та аналіз предметної області
Розділ 3. Розробка застосунку
Висновок
Список використаної літератури

Дата видачі „ ___” _____ 2021 р.

Керівник _____
(підпис)

Завдання отримав _____
(підпис)

Календарний план виконання роботи

№	Назва етапу	Термін виконання	Примітка
1.	Отримання завдання на курсову роботу	08.10.2021	
2.	Огляд технічної літератури за темою роботи	10.11.2021	
3.	Вибір предметної області для застосунку	31.01.2021	
4.	Проведення дослідження	05.02.2021	
5.	Написання застосунку	01.03.2021	
7.	Перегляд застосунку із викладачем	10.05.2022	
8.	Коригування застосунку	15.05.2022	
9.	Обговорення змісту текстової частини	15.05.2022	
9.	Написання текстової частини	18.05.2022	
10.	Захист роботи	15.06.2022	

Зміст

Анотація

У цій роботі детально розглядається розробка веб-застосунку за допомогою API ElasticSearch для реалізації швидкого пошуку даних в тексті. Основною метою цієї роботи було:

- порівняти пошукові двигуни із базами даних;
- познайомитись із функціональністю пошукового двигуна ElasticSearch;
- використати функціональність ElasticSearch у власному веб-застосунку для збереження та швидкого пошуку веб-сторінок.

Вступ

Актуальність теми

Сучасний світ обростає неймовірною кількістю інформації. Ще два десятки років тому комп’ютери містили всього 512 МБ пам’яті. Ємності таких жорстких дисків було достатньо для довготривалого збереження різноманітних файлів. На сьогодні такої кількості пам’яті вистачить для встановлення декількох програм або збереження декількох документів.

З часом світ заполонила велика кількість даних, яку людський мозок не в змозі обробити без допомоги машин. Крім великого обсягу статей, фото та відео, необхідно звернути увагу і на якість усіх матеріалів, створених людьми. Простіше кажучи, відфільтрувати інформацію для отримання корисних даних із доступного набору. При цьому, відбір даних не повинен займати велику кількість часу. Інакше, фільтрація за допомогою комп’ютера втрачає будь-який сенс.

Пошуковий двигун ElasticSearch є зручним рішенням для реалізації повнотекстового пошуку. Він забезпечує масштабований пошук у реальному часі та може бути застосований до будь-якого типу документів.

Мета дослідження

Дослідження пошукового двигуна ElasticSearch при побудові додатку для збереження веб-сторінок.

Постановка задачі

1. Ознайомлення із загальними концепціями пошукових двигунів.
2. Ознайомлення із пошуковим двигуном ElasticSearch.
3. Проектування та створення застосунку з використанням ElasticSearch.

Розділ 1. Використання пошукових двигунів у застосунках

1.1 Короткі відомості про пошукові двигуни

Пошукові двигуни являють собою програмне забезпечення, створене для легкого та ефективного пошуку інформації. Для отримання результату із множини усіх можливих даних користувачеві необхідно ввести ключове слово чи фразу. Пошукова система обробляє запит, отриманий нею, і повертає список файлів із інформацією, яка найбільше відповідає вимогам [1].

Процес обробки даних пошуковим двигуном є складним. Тому алгоритм для знаходження бажаного тексту можна розбити на три стадії. Спочатку пошуковому двигуну необхідно провести процес дослідження усієї доступної йому інформації. Далі отримані дані необхідно опрацювати, відсортувавши їх таким чином, щоб вони могли бути ефективно оцінені і представлені під час відправки запиту користувачем. Після того, як усю отриману інформацію оброблено, пошуковий двигун повинен дати їй оцінку та презентувати для користувача результати, які відповідали б вказаним ключовим фразам під час фільтрування даних.

Зазвичай три згадані вище стадії називають скануванням, індексуванням та ранжуванням. Розглянемо детальніше ці три стадії для отримання чіткого розуміння про алгоритми роботи пошукових двигунів.

У своїй роботі пошукові двигуни застосовують програмне забезпечення, яке називають сканером, для отримання бажаного результату. Сканери проходять по масиву усіх можливих даних та збирають в одному місці всю доступну інформацію під час пошуку. Саме тому даний процес був названий скануванням. Веб-сканери також іноді називають павуками пошукових систем

через те, що вони проходять усю мережу, як павутину і знаходять потрібну ціль.

Цей процес є складним. Сканери знаходять веб-сервери (також відомі, як просто сервери), на яких розміщуються веб-сайти. Після знаходження доступних веб-сайтів на сервері павуки починають сканувати веб-сайти щодо отримання даних, вказаних у запиті користувача.

Припустимо, що список серверів було створено і встановлено, яка кількість веб-сайтів утримується на кожному веб-сервері. Кожен веб-сайт містить певну скінченну кількість сторінок, на якій знаходиться: текст, картинки, аудіозаписи, відеозаписи. Важливо з’ясувати, яка кількість даних розміщується на кожній сторінці.

Також сканери звертають увагу на посилання, які вбудовані у текст поточного веб-сайту, який вони обробляють. Сканери переходять за посиланням. При цьому не важливо, чи посилання є внутрішнім – на сторінки цього ж веб-сайту, або зовнішнім – на сторінки інших сайтів, які також зберігають певну корисну інформацію. Таким чином, павук може обробити ще більше корисних даних, збираючи посилання на всі можливі сайти та розглядаючи їх, як потенційні результати для заданого користувачем пошукового запиту.

Другою стадією при виконанні фільтрування даних пошуковим двигуном є індексування інформації. Усю сукупність байтів, знайдених сканером, необхідно організувати, відсортувати та зберігати у певному місці для того, щоб пізніше вона була опрацьована алгоритмами для представлення користувачеві пошуковим двигуном. Дана процедура називається індексуванням. Уся інформація, яка зазначена на сторінці, насправді не

зберігається у базі пошукового двигуна. Замість цього вказуються лише суттєві дані, які необхідні алгоритму для оцінки доцільності сторінки для того, щоб скласти рейтинг усіх отриманих даних.

Останньою важливою частиною при пошуку даних є власне запит від користувача на отримання певної інформації. Коли пошуковий двигун отримує такий запит, то починає перевіряти індекс щодо вмісту доцільної інформації в ньому, а потім сортується. Таке сортування результатів, знайдених пошуковою системою, називається рейтингом [2].

Проте пошукові двигуни використовуються не тільки для індексування веб-контенту. Область використання може варіювати від файлів одного персонального комп'ютера до логів великих розподілених обчислювальних систем. В таких випадках зазвичай пропустається стадія сканування, оскільки необхідні для індексування документи направляються зовнішніми сервісами до пошукового двигуна.

1.2 Порівняння пошукових двигунів із базами даних

У великій кількості пошукових двигунів наявна функціональність для збереження та маніпулювання даними. Так само для більшості баз даних реалізований текстовий пошук. Проте можливість баз даних знаходити доречну для користувача інформацію не є оптимально реалізованою.

Системи, створені для повнотекстового пошуку, вважаються зручнішими для фільтрації великої кількості тексту за вказаним ключовим словом чи словами. Вони створені таким чином, що здатні забезпечити можливість розширеного пошуку тексту та пристосовані до відбору даних для

створення рейтингу знайденої інформації. Пошукові двигуни мають можливість відсортувати документи по доречності відповідно до запиту користувача з урахуванням помилок при написанні ключових слів та нечітких запитів. Результати, які повертають пошукові системи відповідно до нечітких запитів, можуть відповідати лише частково фразі, вказаній користувачем для фільтрації інформації.

Також в пошукові двигуни вбудовано алгоритми, які здатні зрозуміти наміри користувача при введенні запиту та запропонувати свій варіант ключової фрази при допущенні помилок. Це дає можливість пошуковим системам бути більш гнучкими та зручними у користуванні.

І хоча реляційні бази даних налаштовані на роботу з таблицями у вигляді рядків та стовпців та підтримують гнучкий пошук, сортування даних по доречності не буде виконано так само якісно, як за допомогою систем повнотекстового пошуку. Отримання результатів щодо пошукових запитів буде повільним і забезпечить неякісну роботу для користувачів [3].

1.3 Короткий опис ElasticSearch

ElasticSearch є сучасним програмним забезпеченням для пошуку та аналітики даних. Він являє собою абстрактний шар, створений над бібліотекою Apache Lucene. ElasticSearch з’явився завдяки розробнику Шей Бенону у 2010 році. Це програмне забезпечення є рішенням з відкритим вихідним кодом, створеним мовою програмування Java. Також цей пошуковий двигун являє собою NoSQL базу даних. Однак, на відміну від більшості

NoSQL баз даних, особливістю ElasticSearch є здатність виконувати оптимальний пошук даних [4].

Для комунікації з ElasticSearch використовується Representational State Transfer (REST API). За допомогою базового HTTP інтерфейсу можливо легко працювати із JSON-запитами.

Найменша одиниця в ElasticSearch – це документ, а найбільша – індекс. Кожен документ містить набір ключів (назви полів чи властивостей), які відповідають певним значенням (рядкам, числам, логічним виразам, датам, масивами значень, географічним розташуванням або іншим типам даних).

Документ індексується і зберігається в базі даних. Після цієї процедури ElasticSearch сервер визначає відповідальний набір індексів за щойно доданий документ. Кожен пошуковий двигун містить фіксований набір Lucene індексів [5].

Для взаємодії з документами використовується Document API. Як і інші інтерфейси, цей використовує протокол HTTP. Наприклад, для додавання нового документу використовується метод POST /<index>/_doc/, де <index> – це назва необхідного індексу, а для видалення документу із вказаного індексу достатньо зробити запит DELETE /<index>/_doc/<_id>.

Маппінгом (mapping) називають процес, який визначає, як документ та його поля зберігаються в індексі. Кожний документ складається із полів різних типів. Для визначення даних можна використовувати динамічний та явний маппінг. Кожен із методів забезпечує різні переваги. Наприклад, явний маппінг полів допомагає відкинути поля, які не потрібно використовувати та більш точно маніпулювати типами даних. При цьому користувач має можливість дозволити ElasticSearch додати нові поля пізніше динамічно і їхні

типи будуть встановлені за замовченням. Це може бути корисно, якщо ми наперед не знаємо загальної схеми документів.

При створенні нового екземпляру ElasticSearch кожен користувач працює з нодами (nodes). Колекція об’єднаних нод називається кластером. Кластер може складатися з однієї або більше нод. Кожна нода в кластері може обробляти HTTP і транспортний трафік за замовчуванням. Для комунікації між нодами використовується виключно транспортний шар. HTTP шар використовується REST клієнтами. Кожна нода знає про всі інші ноди в кластері, тому може перенапрявляти запити клієнтів на відповідну ноду.

Розділ 2. Дослідження та аналіз предметної області

Головною ідеєю застосунку є створення додатку для збереження веб-контенту. Користувач за допомогою розширення має змогу зберігати необхідний контент, а пізніше переглядати його у додатку і виконувати повнотекстовий пошук.

2.1 Second-brain концепція

Щодня люди споживають неймовірну кількість інформації. Вони дізнаються новини від рідних та друзів, отримують знання під час дослідження завдань, які отримали на роботі, перечитують матеріали для виконання лабораторних робіт в університеті, переглядають фото чи відео після закінчення продуктивного дня увечері, знову спілкуються з близькими перед сном. Усі перелічені вище заходи призводять до того, що людський мозок не запам’ятовує дані, які отримав тим чи іншим чином. Окрім цього, мозок влаштований таким чином, що інформація, яка не використовується, з часом стирається з пам’яті, так як є непотрібною.

У такому випадку з’являється бажання придбати пам’ять, як це можливо зробити у хмарних сховищах. Звичайно, у реальному житті такої можливості не існує, проте було створено підхід для розвантаження людського мозку і отримання здатності запам’ятати більше. Tiago Forte описав такий підхід у вигляді концепції Second-brain у статті ‘Building a Second Brain: An Overview’.

Автор вказує, що Second Brain являє собою методологію для збереження і регулярного нагадування про ідеї та думки, що з’явилися під час отримання

життєвого досвіду. Така ідея допомагає розширити людську пам’ять та інтелект з використанням сучасних технологічних засобів та мереж.

Автор вважає, що методологія Second Brain не лише допомагає зберегти ідеї, але і втілює їх в реальне життя. Вона забезпечує доступний шлях до “другого мозку” шляхом створення зовнішнього цифрового репозиторію для речей, які були вивчені раніше, а також для ресурсів, з яких була отримана ця інформація.

За допомогою розвантаження мислення та пам’яті у “другий мозок” справжній біологічний мозок людини стає вільним для уяви, створення нових ідей, а також має можливість існувати тут і зараз. За допомогою Second Brain можна споживати велику кількість інформації, при цьому не відслідковувати всі дрібні деталі.

Саме тому цю концепцію було покладено в основу застосунку. Пошукові двигуни є найкращим програмним способом зберігати і впорядковувати великі обсяги інформації. Вони надають можливість легкого пошуку по цій інформації та слугують своєрідним “другим мозком”. Проте використання пошукового двигуну напряму для кінцевих користувачів є зовсім незручним, бо вони надають лише програмний інтерфейс. Отже, у наступних розділах описано проектування та розробку графічного застосунку-імплементації ідеї “другого мозку”, що у своєму ядрі використовує пошуковий двигун ElasticSearch [6].

2.2 Вимоги до застосунку

Наш застосунок має лише один тип користувачів - кінцеві користувачі сервісу. Виділимо функціональні вимоги для них.

Користувач повинен мати можливість:

- Створювати/редагувати директорії, організовуючи їх у дерево.
- Зберігати веб-сторінку або її частину, вказавши при цьому назву сторінки та місце у дереві директорій.
- Редагувати в додатку збережену раніше сторінку. Редагування відбувається за допомогою WYSIWYG-редактора.
- Виконувати повнотекстовий пошук по всіх сторінках.
- Отримувати підказки запиту при пошуку.

2.3 Огляд альтернативних рішень

Можливою альтернативою додатку, що розробляється у цій курсовій роботі, є Notion Web Clipper [7]. Він допомагає зберігати веб-сторінки, конвертуючи їх у свій формат. Надалі вся взаємодія із цим контентом відбувається у додатку Notion. При цьому у додатку можна виконати пошук серед збережених раніше веб-сторінок.

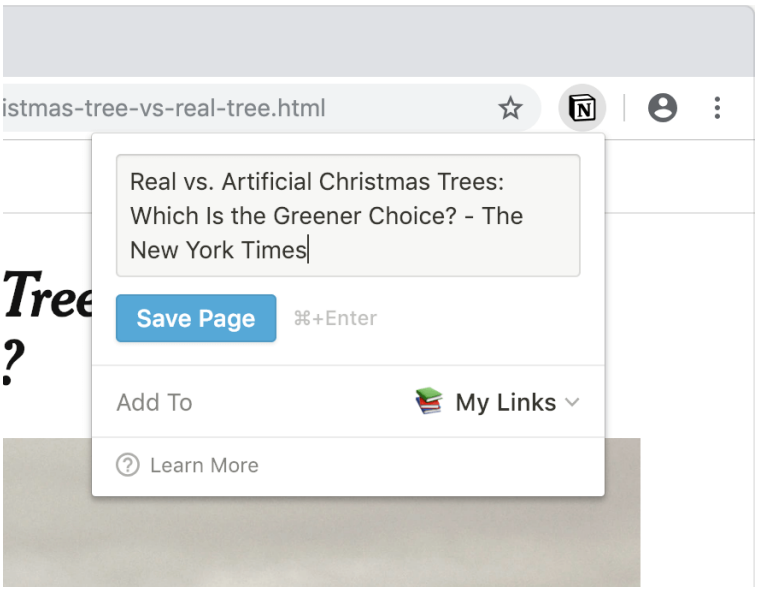


Рисунок 2.1 Інтерфейс розширення Notion Web Clipper [7]

Проте головною метою додатку Notion є організація персонального та корпоративного простору. Він надає можливості збереження, поширення інформації та можливості спільного редагування. Проте недоліком цього додатку є те, що конвертація веб-контенту у формат Notion відбувається дуже погано, часто втрачаються важливі деталі або навпаки непотрібна інформація потрапляє до вихідної сторінки. Notion Web Clipper не дає власноруч обирати контент для подальшої конвертації, а опрацьовує сторінки цілком. Таким чином якість збереження цих сторінок не дає можливості повноцінно користуватись Notion як реалізацією ідеї Second brain.

При цьому, веб-додаток, створений під час написання курсової роботи має на меті реалізувати саме Second Brain концепцію та звільнити користувача від значної кількості інформації, непотрібної для запам’ятовування.

Розділ 3. Розробка застосунку

3.1 Архітектура застосунку

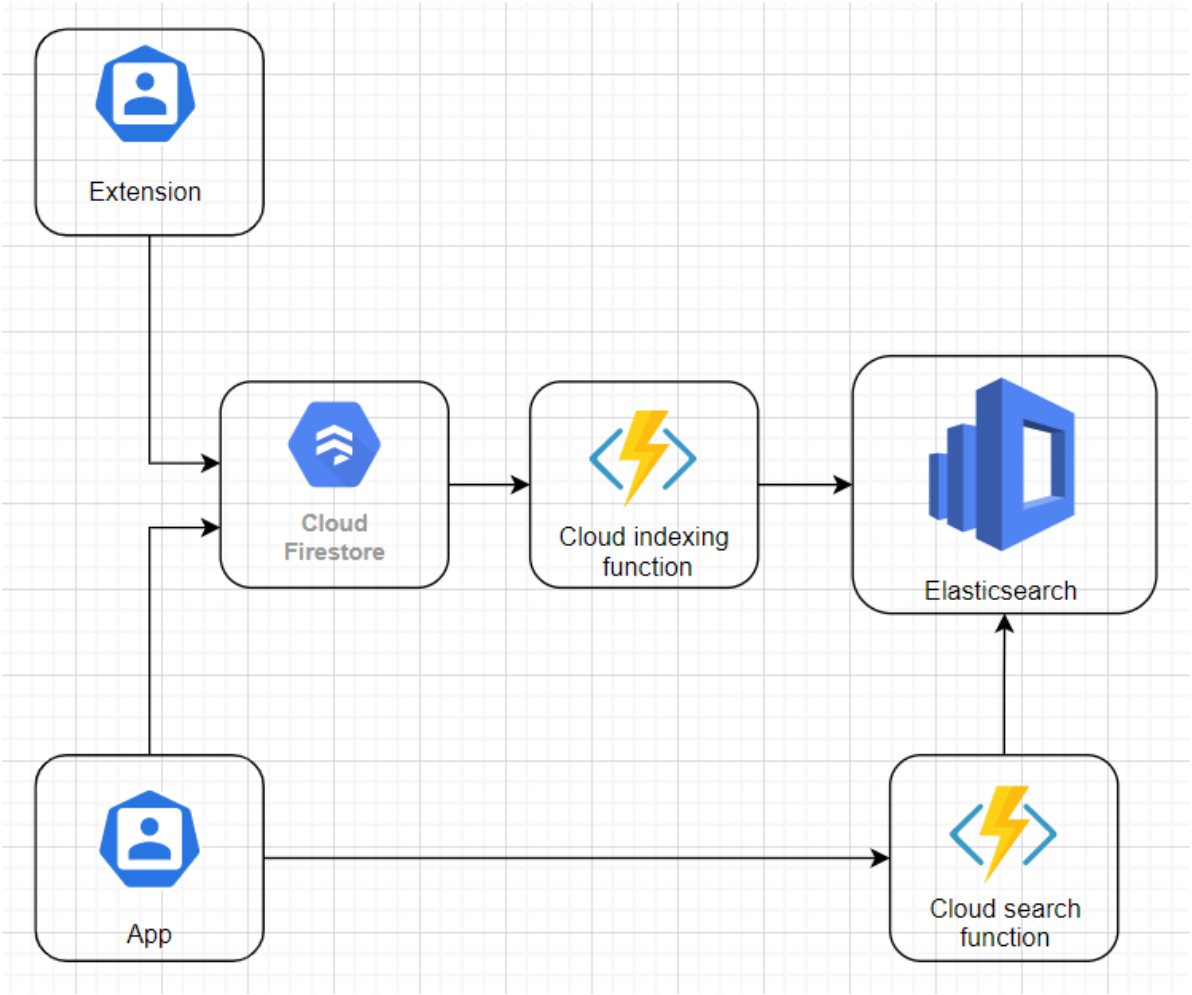


Рисунок 3.1 Архітектура застосунку

Архітектуру застосунку можна поділити на такі області:

- клієнтські застосунки (App, Extension),
- хмарні функції (Cloud indexing function, Cloud search function),
- сховища даних (Cloud Firestore, ElasticSearch).

При збереженні сторінок через розширення застосунку спочатку інформація зберігається у Firestore. На додавання нових сторінок підписується хмарна функція Cloud indexing function, котра надалі додає сторінки до індексу ElasticSearch.

При пошуку сторінок через інтерфейс застосунку йде запит на хмарну функцію Cloud search function, котра у свою чергу робить запит до ElasticSearch і повертає результат клієнту.

3.2 Опис компонентів

Застосунок складається із двох компонентів - розширення для браузеру та основного клієнта. За допомогою розширення користувач може обирати потрібні блоки інформації та додавати їх до свого аккаунту. Також користувач може сформувати структуру директорій і надалі згрупувати свої збереження за різними категоріями. За допомогою основного клієнта користувач може переглядати структуру директорій, додані сторінки та робити повнотекстовий пошук.

Для збереження даних було використано нереляційну базу даних Firestore, а для індексації та подальшого пошуку сервіс ElasticSearch. Використання бази даних було доречним рішенням для збереження даних користувача, котрі не мають індексуватися. Наприклад, інформація про користувача, дата збереження посилань, дерево директорій і так далі. Окрім цього Firestore має зручні налаштування безпеки та контролю доступу, а отже не було потреби налаштовувати їх вручну.

3.3 Опис технологій

3.3.1 Огляд React

React — це JavaScript-бібліотека для роботи з користувацькими інтерфейсами, котра була створена у 2013 році. Ця бібліотека використовується у застосунку для побудови UI застосунків, зокрема й для браузерних розширень. Вона вирішує проблеми із частковими оновленням DOM елементів та дублікацією коду.

3.3.2 Огляд Antd

Ant design – це бібліотека із готовими UI компонентами для створення зручних застосунків. Використовувалась у проекті для розробки багатофункціональних користувацьких інтерфейсів.

3.3.3 Огляд Firebase

Firebase – це платформа, розроблена Google для створення мобільних та веб застосунків.

У цьому застосунку використовувались наступні сервіси:

- Realtime Database

Сервіс, що дозволяє синхронізувати дані застосунків між клієнтами та зберегти їх у хмарі. Дані зберігаються у JSON форматі та оновлюються у режимі реального часу.

- Cloud Functions;

Хмарні функції, котрі надають можливість виконувати backend код у відповідь на триггер події або HTTP запит.

- Authentication.

Сервіс, що надає готове рішення для менеджменту аутентифікації у застосунку.

3.4 Проблеми збереження веб-контенту

Оскільки основним функціоналом застосунку є збереження інформації з веб-сторінок, велику частину уваги було приділено вирішенню саме цієї проблеми.

Першою спробою було збереження цілої сторінки у форматі raw HTML. Проте виникли наступні проблеми:

- Збереження статичних файлів. Оскільки веб-сторінки дуже часто посилаються на зовнішні статичні файли CSS, JS, картинки виникала проблема ідентичного відтворення такої сторінки у застосунку.
- Не всі веб-сервіси дозволяють вкладати свої сторінки у iFrame на інших сайтах.

- Безпека. Оскільки першою задумкою було вкладати сторінки у iFrame, треба було робити додаткові налаштування безпеки.

Тому було вирішено відмовитись від ідеї вкладання цілої веб-сторінки в наш застосунок, а натомість використати мову розмітки Markdown. Вона не містить скриптів та стилів, а отже ми можемо форматувати контент власноруч.

Проте тоді виникає проблема, що при конвертуванні сторінки в Markdown, залишається багато “сміття” - хедеру сторінки, навігації, посилань, реклами, що не мають відношення до бажаного контенту.

Для вирішення цієї проблеми користувачу було надано можливість власноруч обирати необхідну область сторінки для збереження. Потім цей елемент конвертується в Markdown, при цьому залишається лише необхідна інформація. Для конвертації використовується відома бібліотека turndown.

Також було вирішено надати користувачу можливість правити контент: додавати нотатки, виділяти, видаляти текст. Для цього доречно використати готову бібліотеку WYSIWYG-редактора. Оскільки HTML надає більше можливостей форматування тексту, то остаточним форматом збереження обрано саме його. Бібліотека showdown перетворює Markdown в остаточний HTML.

3.5 Налаштування ElasticSearch

3.5.1 Індексація

Як вже було наведено, остаточний формат збереження контенту - HTML. ElasticSearch не має типу за замовчування для HTML, проте дозволяє налаштовувати кастомні аналізатори тексту.

Отже, при створенні індексу оголосимо кастомний аналізатор htmlStripAnalyzer, що використовуватиме стандартний токенізатор. Додамо також фільтр, що приведе весь текст до нижнього регістру і фільтр, що видалить символи, що відносяться до мови розмітки HTML, а не до основної інформації. Таким чином, пошук не буде здійснюватись за тегами та атрибутами, яких не бачить користувач.

Отже, у результаті отримуємо:

```
"analysis": {
  "analyzer": {
    "htmlStripAnalyzer": {
      "type": "custom",
      "tokenizer": "standard",
      "filter": ["lowercase"],
      "char_filter": [
        "html_strip"
      ]
    }
  }
}
```

Тепер, використовуючи цей аналізатор й інші типи за замовчуванням оголосимо розмітку індексу. Маємо наступні поля:

- uid - ідентифікатор користувача, необхідний для фільтрування. Має тип keyword.
- title - заголовок сторінки. Тип text - для повнотекстового пошуку.

- content - html контент сторінки. Тип text - для повнотекстового пошуку.

```
"mappings": {
  "properties": {
    "uid": {
      "type": "keyword"
    },
    "title": {
      "type": "text"
    },
    "content": {
      "type": "text",
      "analyzer": "htmlStripAnalyzer"
    }
  }
}
```

Увесь запит створення індексу можна знайти в додатку (Додаток А).

Індексація, як вже було наведено, відбувається в функції Cloud indexing function за допомогою бібліотеки elasticsearch. Код індексації виглядає тривіально:

```
await elasticClient.index({
  index: 'sb-test',
  id: pageId,
  body: {
    uid: userId,
    title: name,
    content: content,
  }
});
```

3.5.2 Пошук

Пошук відбувається, як вже було наведено, в функції Cloud search function. Код пошуку виглядає наступним чином:

```
const result = await elasticClient.search({
  index: 'sb-test',
  body: {
    "query": {
      "bool": {
        "should": [
          { "match": { "content": queryText } },
          { "match": { "title": queryText } }
        ],
        "filter": [
          { "term": { "uid": uid } }
        ],
        "minimum_should_match": 1
      }
    },
```

За допомогою ключових слів filter та term відбувається фільтрація документів за ідентифікатором конкретного користувача. За допомогою ключових слів should та match відбувається повнотекстовий пошук по вмісту та заголовку сторінки. Кожному документу призначається score - міра релевантності цього документа запиту користувача. Далі документи сортуються по мірі зменшення релевантності.

Ми використали слово should, щоб об’єднати пошук за контентом та пошук за заголовком. Тобто хоча б одне з цих полів має відповідати запиту користувача.

3.5.3 Highlighting

Highlighting відбувається, як вже було наведено, в функції Cloud search function. Код пошуку виглядає наступним чином:

```
"highlight": {
  "fields": {
    "content": {},
    "title": {}
  },
  "no_match_size": 100
}
```

У полі fields вказуються поля, котрі ми бажаємо виділити, тобто контент та назву сторінки. У полі no_match_size вказується кількість символів, котрі ми хочемо повернути з початку поля, якщо не було знайдено жодного фрагменту для виділення.

У результаті пошуку слова Ban отримаємо наступну відповідь, де всі знайдені слова виділяються тегами :

```
"highlight" : {
  "title" : [
    "Ray <em>Ban</em> Hexagonal, gafas metálicas impregnadas del
      espíritu de los 60"
  ],
  "content" : [
    "<p>Las <strong>Ray <em>Ban</em> Hexagonal</strong> son el
      resultado de superponer un círculo y un cuadrado, este",
    "Las Ray-<em>Ban</em> RB3548N se caracterizan por sus lentes
      aplanadas mientras las Hexagonal RB3548 tienen las",
    """tienda/gafas-de-sol/ray-ban/" rel="noopener noreferrer"
      target="_blank">distribuidores autorizados Ray <em>Ban</em
      >""",
    "Compra ahora tus Ray <em>Ban</em> Hexagonales y no pagues
      gastos de envío.</p>"
  ]
}
```

3.5.4 Autocomplete

Для налаштування autocomplete необхідно додати у розмітку індексу додаткове поле suggest із спеціальним типом completion, котрий оптимізовано по швидкості для виконання autocomplete запитів[8]:

```
PUT sb-test/_mapping
{
  "properties": {
    "suggest": {
      "type": "completion"
    }
  }
}
```

Також при індексуванні нових документів у коді необхідно заповнити нове поле всіма рядками, які будуть автодоповнюватись, коли користувач починає щось писати. В нашому випадку це лише title сторінки. Тобто ми хочемо, щоб коли користувач починав вводити заголовок, ми пропонувати йому повний заголовок сторінки, щоб перейти на неї:

```
await elasticClient.index({
  index: 'sb-test',
  id: pageId,
  body: {
    uid: userId,
    title: name,
    content: content,
    suggest: {
      input: [name]
    }
  }
});
```

Сам autocomplete відбувається в функції Cloud search function:

```
result = await elasticClient.search({
  index: 'sb-test',
  body: {
    "_source": false,
    "query": {
      "term": {
        "uid": uid
      }
    },
    "suggest": {
      "title-suggest": {
        "regex": `.*${queryText.toLowerCase()}.*`,
        "completion": {
          "field": "suggest"
        }
      }
    }
  }
});
```

За допомогою query відбувається фільтрація документів за ідентифікатором конкретного користувача. За допомогою поля suggest відбувається пошук підходящих документів базуючись на регулярному виразі, який має на меті автодоповнення пошукового запиту навіть якщо він не є початком назви будь-якої сторінки.

Висновки

У цій курсовій роботі було детально досліджено використання пошукового двигуна ElasticSearch при фільтруванні інформації. У цій пошуковій системі реалізована можливість сортування інформації за її доречністю, зважаючи на пошуковий запит користувача. ElasticSearch має можливість взаємодіяти із нечіткими запитами та знаходити результати пошуку за частиною ключової фрази, яка була задана користувачем. Він забезпечує масштабований пошук у реальному часі та може бути застосований до будь-якого типу документів.

Для пошуку ключової інформації серед численних даних можна також використовувати бази даних. Проте сортування даних по доречності не буде виконано так само якісно, як це можливо зробити за допомогою пошукового двигуна ElasticSearch. Результати фільтрування даних за допомогою баз даних отримуються доволі повільно, що забезпечує неякісну роботу для користувачів при виконанні пошуку.

Як приклад доречного використання пошукового двигуна було розглянуто концепцію Second brain та розроблено застосунок-імплементацію цієї концепції. На прикладі застосунку показано основні налаштуванн ElasticSearch та підходи до роботи з ним.

В результаті дослідження можна зробити висновок, що пошуковий двигун ElasticSearch є зручною технологією для виконання різноманітних технічних задач у зв’язку з легкістю його інтеграції в будь-який додаток будь-якою мовою програмування завдяки відкритому Rest API та можливістю продуктивного виконання повнотекстового пошуку серед великої кількості інформації.

Список використаної літератури

1. What is a search engine [Електронний ресурс] – Режим доступу до ресурсу: https://www.bdc.ca/en/articles-tools/entrepreneur-toolkit/templates-business-guides/glossary/search-engine
2. What is a search engine and how do they work? | nibusinessinfo.co.uk [Електронний ресурс] – Режим доступу до ресурсу: https://www.nibusinessinfo.co.uk/content/what-search-engine-and-how-do-they-work
3. Full-Text Search Engines vs. Relational Databases | Lucidworks – Режим доступу до ресурсу: https://lucidworks.com/post/full-text-search-engines-vs-dbms/
4. What is Elasticsearch? | Elastic [Електронний ресурс] – Режим доступу до ресурсу: https://www.elastic.co/what-is/elasticsearch
5. Vidhya, R., and G. Vadivu. "Research document search using Elastic search." Indian Journal of Science and Technology 9.37 (2016)
6. Building a Second Brain: An Overview - Forte Labs [Електронний ресурс] – Режим доступу до ресурсу: https://fortelabs.co/blog/basboverview/
7. Notion Web Clipper for Chrome, Safari, Firefox, and mobile [Електронний ресурс] – Режим доступу до ресурсу: https://www.notion.so/web-clipper
8. Suggesters | Elasticsearch Guide [8.2] | Elastic [Електронний ресурс] – Режим доступу до ресурсу: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-suggesters.html

ДОДАТКИ

Додаток А

(Обов’язковий)

Імплементація функції Cloud indexing function

```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
const { initClient } = require('./elasticClient');

try {
  admin.initializeApp();
} catch (e) {}

const elasticClient = initClient();
const PATH = '/users/{userId}/pages/{pageId}';

exports.trigger = functions.database.ref(PATH).onWrite(async (change, context) => {
  const newValue = change.after.val();

  const { pageId, userId } = context.params;

  if (!newValue) {
    await elasticClient.delete({ index: 'sb-test', id: pageId });
    return;
  }

  const { content, name } = newValue;

  await elasticClient.index({
    index: 'sb-test',
    id: pageId,
    body: {
      uid: userId,
      title: name,
      content: content,
      suggest: {
        input: [name]
      }
    }
  });
});
```

Додаток Б

(Обов’язковий)

Імплементація функції Cloud search function

```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
const { initClient } = require('./elasticClient');

try {
  admin.initializeApp();
} catch (e) { }

const elasticClient = initClient();

exports.handler = functions.https.onCall(async (data, context) => {
  const queryText = data.query;
  const uid = context.auth.uid;

  let result;

  if (data.complete) {
    result = await elasticClient.search({
      index: 'sb-test',
      body: {
        "_source": false,
        "query": {
          "term": {
            "uid": uid
          }
        },
        "suggest": {
          "title-suggest": {
            "regex": `.*${queryText.toLowerCase()}.*`,
            "completion": {
              "field": "suggest"
            }
          }
        }
      }
    });
```

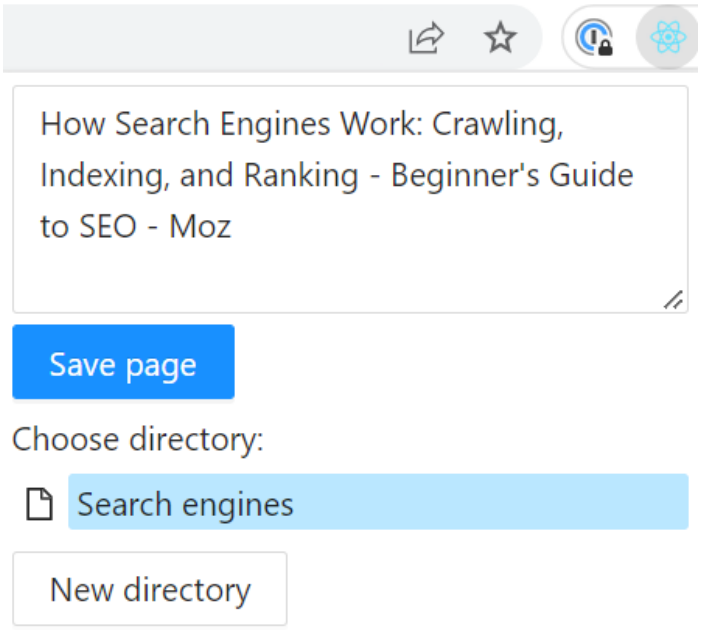
```
  } else {
    result = await elasticClient.search({
      index: 'sb-test',
      body: {
        "query": {
          "bool": {
            "should": [
              { "match": { "content": queryText } },
              { "match": { "title": queryText } }
            ],
            "filter": [
              { "term": { "uid": uid } }
            ],
            "minimum_should_match": 1
          }
        },
        "_source": false,
        "highlight": {
          "fields": {
            "content": {},
            "title": {}
          },
          "no_match_size": 100
        }
      }
    });
  }

  return { result: result.body, uid, queryText }
});
```

Додаток В

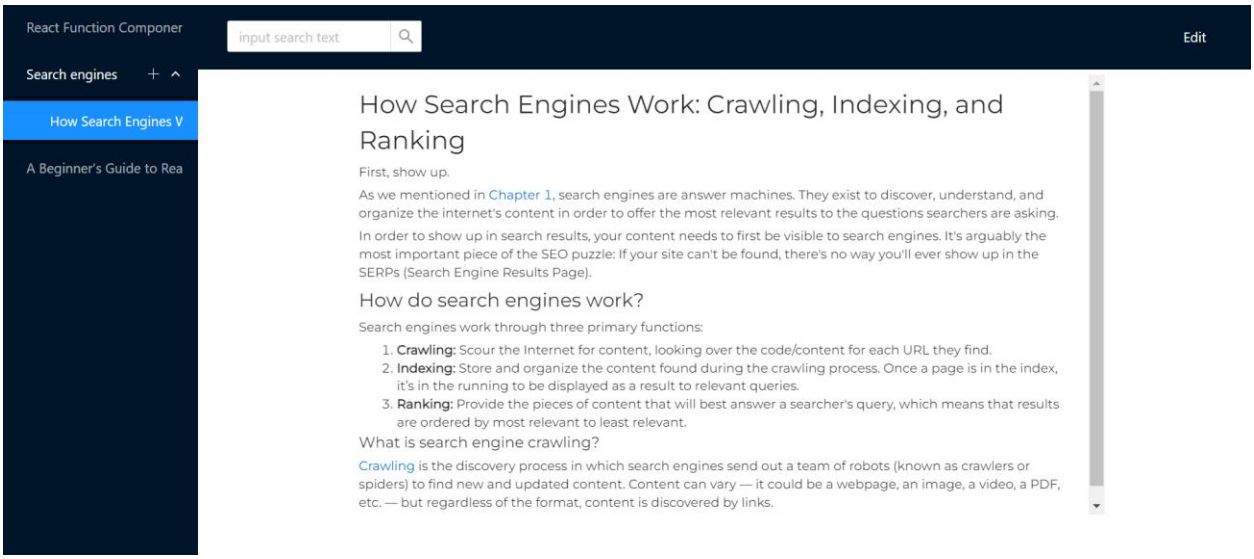
(Обов’язковий)

Скріншоти інтерфейсу застосунку

Розширення для збереження веб-контенту



Сторінка перегляду збереженого контенту



Сторінка редагування збереженого контенту